TU Clausthal
Clausthal University of Technology

The Multi-Agent Programming Contest 2012 Edition
Evaluation and Team Descriptions

Michael Köster, Federico Schlesinger, Jürgen Dix

IfI Technical Report Series IfI-13-01

IfI

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Federico Schlesinger
Contact: federico.schlesinger@tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Contents

Part I
Overview

The Multi-Agent Programming Contest 2012 Edition

Evaluation and Team Descriptions

Michael Köster, Federico Schlesinger, Jürgen Dix

Department of Informatics, Clausthal University of Technology,
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany

Abstract

The Multi-Agent Programming Contest, MAPC, is an annual, community-serving competition that attracts groups from all over the world. Its aim is to facilitate advances in programming multiagent systems (MAS) by (1) developing benchmark problems, (2) enabling head-to-head comparison of MAS's and (3) supporting educational efforts in the design and implementation of MAS's. We report about its eighth edition and give a detailed overview of the participants strategies and the overall contest.

1 Introduction

This paper serves as an introduction to the subsequent papers in this proceedings volume, each of which describes a team that participated in this years edition. We give a comprehensive overview of the Multi-Agent Programming Contest[1] 2012, an annual international event that has started in 2005 as an attempt to stimulate research in the field of programming multi-agent system by 1) identifying key problems, 2) collecting suitable benchmarks, and 3) gathering test cases which require and enforce coordinated action that can serve as milestones for testing multi-agent programming languages, platforms and tools. In 2012 the competition was organised and held for the eighth time.

Research communities in general benefit from competitions that attempt to evaluate different aspects of the systems under consideration and furthermore allow for comparing state of the art systems, act as a driver and catalyst for developments and pose challenging research problems.

In this paper we (1) briefly introduce the Contest and its infrastructure, (2) elaborate on the 2012 scenario and its differences with the 2011 edition, (3)

[1] `http://multiagentcontest.org`

introduce the seven teams that took part in the tournament, and (4) present results and findings acquired before, during and after the tournament.

More detailed information about the strategies of the teams are to be found in the remaining six papers in this volume.

1.1 Related Work

The Multi-Agent Programming Contest has generated quite a few publications over the years [9, 10, 11, 3, 4, 1, 8]. For a detailed account on the history of the contest as well as the underlying simulation platform, we refer to [1, 8, 5, 6]. A quick non-technical overview appears in [2].

Similar contests, competitions and challenges have taken place in the past few years. Among them we mention *Google's AI challenge*[2], the *AI-MAS Winter Olympics*[3], the *Starcraft AI Competition*[4], the *Mario AI Championship*[5], the *ORTS competition*[6], and the *Planning Competition*[7]. Every such competition rests in its own research niche. Originally, our Contest has been designed for problem solving approaches that are based on formal approaches and computational logics. But this is not a requirement to enter the competition.

1.2 The contest from 2005–2012

From 2005 to 2007 we used a classical gold miners scenario [10] and introduced the *MASSim* platform: A platform for executing the Contest tournaments.

From 2008 to 2010 we developed the cows and cowboys scenario which has been designed to enforce cooperative behavior among agents [4]. The topology of the environment was represented by a grid that contained, besides various obstacles, a population of simulated cows. The goal was to arrange agents in a manner that scared cows into special areas, called corrals, in order to get points. While still maintaining the core tasks of environment exploration and path planning, we also made the use of cooperative strategies an obligation.

The agents on Mars scenario, used during the 2012 edition and discussed in this paper, was firstly introduced in 2011 [5]. In short, we have generalized the environment topology to a weighted graph. Agents were expected to cooperatively establish a graph covering while standing their ground in an adversarial setting and reaching achievements.

[2] http://aichallenge.org/
[3] http://www.aiolympics.ro/
[4] http://eis.ucsc.edu/StarCraftAICompetition
[5] http://www.marioai.org/
[6] http://skatgame.net/mburo/orts/
[7] http://ipc.icaps-conference.org/

2 MAPC 2012: Agents on Mars

In this section we give a detailed overview of the 2012 agents on Mars scenario and point out differences to the scenario from 2011.

2.1 The Scenario

It is now a tradition to accompany the technical description of each scenario with a motivating little story:

> *In the year 2033 mankind finally populates Mars. While in the beginning the settlers received food and water from transport ships sent from earth shortly afterwards – because of the outer space pirates – sending these ships became too dangerous and expensive. Also, there were rumors going around that somebody actually found water on Mars below the surface. Soon the settlers started to develop autonomous intelligent agents, so-called All Terrain Planetary Vehicles (ATPV), to search for water wells. The World Emperor – enervated by the pirates – decided to strengthen the search for water wells by paying money for certain achievements. Sadly, this resulted in sabotage among the different groups of settlers.*
>
> *Now, the task of your agents is to find the best water wells and occupy the best zones of Mars. Sometimes they have to sabotage their rivals to achieve their goal (while the opponents will most probably do the same) or to defend themselves. Of course the agents' vehicle pool contains specific vehicles. Some of them have special sensors, some are faster and some have sabotage devices on board.*
>
> *Last but not least, your team also contains special experts, e.g. the repairer agents, that are capable of fixing agents that are disabled. In general, each agent has special expert knowledge and is thus the only one being able to perform a certain action. So your agents have to find ways to cooperate and coordinate among them.*

The environment's topology is constituted by a weighted graph. Each vertex has a unique identifier and a number that indicates its value. Each edge has a number that represents the costs of moving from one of its vertices to the other. These vertex-values are crucial for calculating the values of zones. A zone is a subgraph that is covered by a team of agents according to a coloring algorithm that is based on a domination principle.

Several agents can stand on a single vertex. If a set of agents dominates such a vertex, the vertex gets the color of the dominating team. A previously uncolored vertex that has a majority of neighbors (at least 2) with a specific color, inherits this color as well. Finally, if the overall graph contains a colored subgraph that constitutes a frontier or border, all the nodes that are inside this border are colored as well. This means that agents can color or cover a subgraph that has more vertices than the overall number of agents. Figure 1 shows a screenshot of a relatively small map, depicting, amongst other things, the graph coloring.

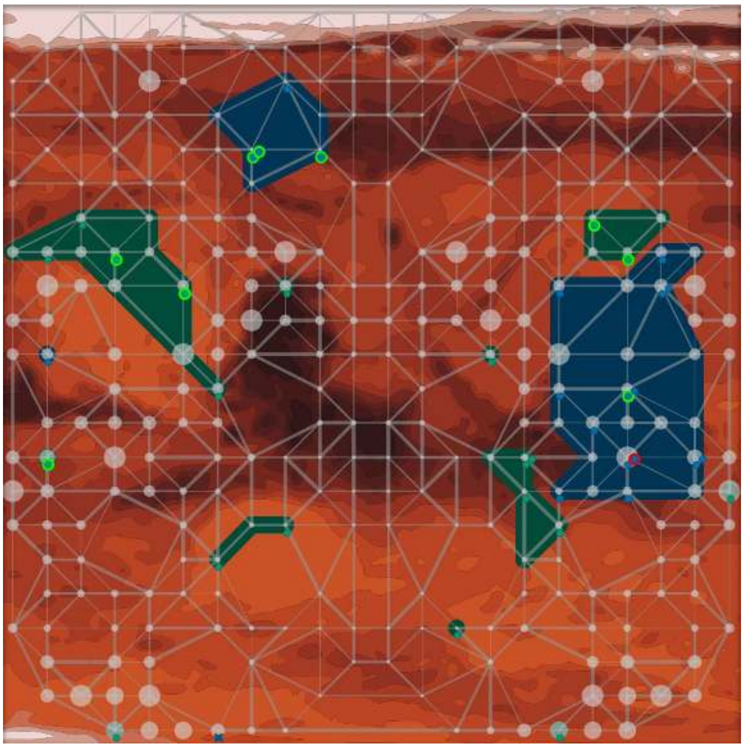


Figure 1: A screenshot of the agents on Mars scenario.

Before elaborating on the agent roles we have to specify the effectoric capabilities of the agents. Each agent, or vehicle, has a state that is defined by its position on the map, its current energy available for executing actions and its current health. On top of that, each team has a budget for equipping the vehicles during the simulation. These actions[8] are defined by the scenario:

- `skip` is the noop-action, which does not change the state of the environment,
- `recharge` increases the current energy of a vehicle by a fixed factor and can be performed at any time without costs,

[8] Of course, all the actions that cost energy will fail if the vehicle under consideration does not have enough energy.

- `attack` decreases the health of an opponent, standing on the same vertex, if successfully executed and decreases the current energy of the attacker,
- `parry` parries an attack and decreases the energy of the defending agent,
- `goto` moves the vehicle to a neighboring vertex while decreasing its energy by the weight of the traversed edge,
- `probe` yields the exact value of the vertex the vehicle is standing on and decreases the probing vehicle's energy,
- `survey` yields the exact weights of visible edges while decreasing the energy,
- `inspect` costs energy and yields the internals of all visible opponents,
- `buy` equips the vehicle with new components, which increase its performance, and cost money, and
- `repair` repairs a teammate, which again costs energy.

We have defined five different roles. Each team consists of four vehicles for each role, that is a total of twenty vehicles per team. This number increased from the 2011 edition, where teams were composed by 2 vehicles for each role, totaling 10 vehicles. Each role defines the vehicle's internals and its capabilities. The roles differ with respect to energy, health, strength and visibility range. The effectoric capabilities are as follows:

- **explorer** can skip, move to a vertex, probe a vertex, survey visible edges, buy equipment and recharge its energy,
- **repairer** can skip, move to a vertex, parry an attack, survey visible edges, buy equipment, repair a teammate and recharge its energy,
- **saboteur** can skip, move to a vertex, parry an attack, survey visible edges, buy equipment, attack an opponent and recharge its energy,
- **sentinel** can skip, move to a vertex, parry an attack, survey visible edges, buy equipment and recharge its energy,
- **inspector** can skip, move to a vertex, inspect visible opponents, survey visible edges, buy equipment and recharge its energy.

Achievements are tasks that, if fulfilled, contribute to the teams' budgets. We have defined a set of achievements that includes having zones with fixed values, inspecting a specific number of vehicles, probing a number of vertices, surveying a fixed number of edges and successfully performing and parrying a number of attacks.

In each step, each vehicle is provided with its currently available percepts:

- the state of the simulation, i.e. the current step,
- the state of the team, i.e. the current scores and money,
- the state of itself, i.e. its internals,
- all visible vertices, i.e. identifier and team,
- all visible edges, i.e. their vertices' identifiers,
- all visible vehicles, i.e. their identifier, vertices and team,
- probed vertices, i.e. their identifier and values,
- surveyed edges, i.e. their vertices' identifiers and weights, and
- inspected vehicles, i.e. their identifiers, vertices, teams and internals.

After sending percepts, the server grants some time for deliberation. After that the new state is computed. The simulation state transition is as follows:

1. collect all actions from the agents,
2. let each action fail with a specific probability,
3. execute all remaining `attack` and `parry` actions,
4. determine disabled agents,
5. execute all remaining actions,
6. prepare percepts,
7. deliver the percepts.

The introduction of the agents on Mars scenario was also accompanied by the release of an environment interface that has been developed to be compatible with the *environment interface standard* [7]. This standard allows Java based problem solving approaches to make use of a jar-file provided by the organizers that facilitated connecting to and communicating with the *MASSim* server. This is done my mapping the whole communication to Java-method invocations and callbacks.

2.2 Changes and Modifications to the Scenario from 2011

As already mentioned, we increased the number of agents to 20 and provided them with more energy. This results in less recharging and gives them more freedom: in 2011, recharge was by far the most used action.

The visualization was improved a lot (zones as well as high-valued vertices are highlighted, costs of the edges are depicted by their thickness. The last action from an agent at each vertex is illustrated: (1) green circle: successful sense action (probe, survey, inspect), (2) red circle: last action failed, (3) yellow star: successful attack, (4) indigo star: successful parry, (5) pink star: successful repair, and (6) crossed out: disabled.

Agents are now getting feedback as to why their actions failed (if they did). The (automatic) generation of maps has been improved (a map contains now several centers).

3 The Tournament

During past editions of the Contest, stability (i.e., the capacity to send actions to the *MASSim* server in time) was a big problem for some teams. It also affected the overall quality of the Contest and the possibility to draw conclusions about the strategies by looking at the results. To address this, we decided for the 2012 edition to implement a *qualification round*, in which teams were required to show that they were able to maintain good stability (i.e. timeout-rates below 5%) during a round of test matches. Only then they were allowed to take part in the tournament.

3.1 Participants and Results

Nine teams from all around the world registered for the Contest. Seven of them were able to pass the qualification round and took part in the tournament (see Table 1). Full introductions of the teams can be found in [12] and in the papers included in this volume.

Team AiWYX was a single-developer team from Sun Yat-Sen Univerity, China. The agents were developed in C++, using no agent-specific technologies. The approach used is centralized, where one agent gets all the percepts from the other agents and makes the decisions for the whole team.

Team PGIM comes from the Islamic Azad University of Malayer, Iran. The 3 developers used agent-specific technologies for developing their team: Prometheus, JACK. Nevertheless the team organization is not distributed, and agents broadcast their percepts.

Team LTI-USP from University of Sao Paulo, Brazil had three developers. Agents were implemented using Jason, CArtAgO and Moise. There is one agent that determines the best strategy, but each agent has its own thread,

Team	Affiliation	Platform/Language
AiWYX	Sun Yat-Sen University, China	C++
PGIM	Islamic Azad University of Malayer, Iran	Prometheus, JACK
LTI-USP	University of Sao Paulo, Brazil	Jason, CArtAgO, Moise
SMADAS-UFSC	Federal University of Santa Catarina, Brazil	Jason
Python-DTU	Technical University of Denmark	Python
Streett	- , USA	Java
TUB	TU Berlin, Germany	JIAC

Table 1: Participants of the 2012 edition.

with its own beliefs, desires and intentions. Agents broadcast new percepts, but communication load decreases over time.

Team SMADAS-UFSC is from Federal University of Santa Catarina, Brazil. It had six team members. The language of choice for agent development was Jason. Besides normal agent-communication provided by Jason, agents shared a common data-structure (blackboard) for storing the graph topology.

Team Python-DTU from the Technical University of Denmark is a regular contender of the Multi-Agent Programming Contest. For this edition it registered 6 members. As team's name suggest, Python was the language of choice. The agents follow a decentralized approach, where coordination is achieved through distributed algorithms, e.g. for auction-based agreement.

Team Streett was composed by a single independent developer from the USA. Agents were developed in Java, based on the sample agents provided with the *MASSim* platform. Agents shared only vital information and coordination was achieved by sharing location data.

Team TUB, TU Berlin, Germany, is another regular contender of the Multi-Agent Programming Contest, that presented for this edition as a single-developer team. The agents are developed in the JIAC platform (which won the contest several times in previous years).

The tournament took place from 10th to 12th September 2012. Each day each team played against two other teams so that in the end all teams played against all others. We started the tournament each morning at 10 am and finished at around 3 pm. A match between two teams consisted of 3 simulations only differing in the size of the graph. For a win the team got 3 points and for a draw 1 point. The results of this year's Contest are shown in Table 2.

Two teams, *SMADAS-UFSC* and *Python-DTU*, stood out from the rest and the tournament winner was decided by the match that confronted them, during the second day of the competition. *SMADAS-UFSC* won two of three simulations of that match and was crowned champion, leaving *Python-DTU* as runner-up for the second consecutive year. Both teams won all the matches they played against the rest of the teams without losing any simulations. The

Pos.	Team	Score			Difference	Points
1	SMADAS-UFSC	2778057	:	1043023	1735034	51
2	Python-DTU	2738397	:	1095251	1643146	48
3	TUB	2090849	:	1600914	489935	30
4	LTI-USP	1627177	:	1845601	-218424	27
5	AiWYX	2301358	:	1526768	774590	24
6	PGIM	1130432	:	2047735	-917303	9
7	Streett	192694	:	3699672	-3506978	0

Table 2: Results.

mid-table teams *TUB*, *LTI-USP* and *AiWYX* where relatively close while playing against each other. They could not catch up with the first two teams but clearly differentiated from the last two.

Thanks to the qualification round (as well as the optional test matches offered before it), there were no stability issues during the Contest. This was a great improvement compared to previous editions. Although some of the teams experimented a few crashes from time to time, the promptness of the developers to restart their agents ensured that the results of the simulation were not affected by these isolated events.

3.2 Overview of the Teams' Strategies

In this section we collect a few facts about the participating teams. For more detailed information we refer to the articles in these proceedings.

SMADAS: The winner of this years contest, from Brazil, used Jason, a dedicated MAS programming language. For some algorithms, Java was used to implement them, rather than Jason. The development needed 500 person hours distributed among 6 people. They used 7900 lines of code, 2400 of which were written in Java. Communication with the server was done through the EISMASSIM interface.

The system is decentralized. Agents were executed on the same machine to use shared memory (blackboard programming). But updating the blackboard was computationally difficult and thus could only be done every 3 steps.

The strategy was first to explore the map, find the best potential zones (high values) and then to conquer and defend them. An interesting idea was to make the opponents spend their money using a special agent: Hulk. If the team detects that there is no particular buying strategy, then the Hulk agent changes its behaviour.

They claim that the good performance is based on the various strategies that make the team very flexible against different opponents. Defending of the zones can still be improved.

Python-DTU: The danish team ended as runner-up for the second time in a row. The team did not use a dedicated platform or MAS programming language. They choose Python for efficiency and to have complete control over all features in the implementation. However, the team used the organizational model of *Moise*.

The solution they implemented is decentralized and heavily based on communications between the agents and on an auction-based agreement algorithm. They invested 300 person hours distributed among 6 people. 1500 lines of codes were written.

The strategy is based on dividing the game in three phases: randomly trying for achievements in the first phase, taking control of high valued areas and sending out explorers in the second phase, and trying to expand in the third phase.

The team claims that their buying algorithm has been detected in the qualification phase and a clever counter strategy was developed by another team that eventually led to the defeat.

TUB: The german team TUB, winner of several contests in the past, entered the contest for the 4th time (but with different team members). They use a centralized approach where agents share all their perceptions and intentions. It required 640 person hours (and 8000 lines of code)

First the agents probe and survey the whole graph. Explorers, attackers, repairers and inspectors only contribute to the zoning algorithm, if they have done their dedicated tasks. The team tries to find a balance between zoning and achievements points.

The team claims that they did not foresee very aggressive playing methods and that this led to several lost games.

LTI-USP: The motivation of the second brazilian team, (one professor and 2 students without previous experience in this scenario, was to test the JaCaMo framework (CArtAgO, Jason and Moise). They used a centralized approach for coordinating the agents and communication via speech-acts. 300 person-hours were invested and 3000 lines of code (a third in AgentSpeak, the rest in Java) were written.

The strategy was not to divide the game into phases but the agents into three subgroups: two for occupying zones and one for sabotaging the enemy. Communication with the server was through the EISMASSIM interface. The repairer agents stay where they are and wait until damaged agents come and see them. The sentinels always parry when an opponent saboteur is there and the own saboteurs always attack opponents in the same vertex.

No defense strategy has been implemented and the team claims that this was responsible for not doing better in the contest (zones were instable).

AiWYX: The chinese team consisted of just one person, a bachelor of science. He has a background in knowledge representation, game theories and distributed algorithms and used just plain C++. He invested ca. 250 person-hours and wrote 10000 lines of code. No agent programming technology was used at all, the system was centralized, all agents share their knowledge to build the map.

The strategy is to first go for areas where nobody else is and trying to expand them. If enemies attack, the agents draw back and look for better zones rather than attacking the enemies. Agents can dynamically change their behaviour at run-time. A big problem was that the agents did not attack the enemy team and that attacks from the enemy were not parried in a suitable way which resulted in instability of the zones.

PGIM: The iranian team consisted of one scientist and three students. They invested 8000 person-hours in total, using 7000 lines of code, to develop a decentralized system. After careful evaluation they chose Prometheus and Jack. Due to licensing problems, they could not use Jack and had to redo all in Java. Due to some misunderstanding of the scenario, they chose to first attack and destroy the opponents repairer agent, then to attack other agents and only in the third place to consider building zones.

Instability of the zones and not being able to conquer zones of some value were the main drawbacks.

Streett: This team consisted of an american student who, unfortunately, did not provide us with any information about his team.

4 Interesting Simulations

In this section we analyse three of the most interesting games using our newly developed statistics module. This involves analysing the following charts: (1) summed-up scores, (2) zone scores and achievement scores, (3) zone stabilities.

The summed-up score consists of the achievement-score plus the zone-score. Note that the achievement score decreases, when the buy action is executed.

summed-up scores: This chart depicts the summed-up score of each team in each step of the current simulation.

zone scores and achievement scores: This chart combines the charts for the step-score (zone-scores + achievement-scores) and the achievement-scores. The zone-score derives from the number and value of the currently dominated nodes, while the achievement score sums up (across all categories) all the achievements so far.

zone stabilities: This chart depicts the zone stabilities of each team in each step of the current simulation. The zone stability increases for one team, if the team can hold all conquered nodes over a longer period of time. If nodes are lost, the value decreases. The exact computation is as follows: For each node that is dominated by a team in a certain step the counter is increased by one. If the team does not dominate the node anymore the counter is reset. The overall zone stability is then the sum of all node counter values.

4.1 SMADAS-UFSC vs. Python-DTU – Simulation 1

The first simulation of the match between SMADAS-UFSC and Python-DTU was a close victory for the winners of the contest, by 127.546 to 121.312. The complete visualization of the simulation can be downloaded from our webpage [9]. Both teams started even, with a very small edge to Python-DTU in the first few steps. Then, SMADAS-UFSC took over from step 35 until step 259. Python-DTU managed to recover the lead at that point for around 50 steps but with no considerable difference. Finally, SMADAS-UFSC took over again from step 309 until the end of the simulation, with a tendency to further increase the score difference. Figure 2, which shows the summed scores at each step, presents this visually.

Figure 3 shows the *step-score* at each step (i.e., the value of the zone plus the unused achievement points at each step). To better display how the score is composed, also the unused achievement points at each step are displayed in the figure. Changes in step-score suggest that both teams attempted to conquer differentiated overlapping zones, as both teams maintained their zone value always above a relatively high minimum, but at several points in the graph the increase in the score for one team is correlated with a decrease in the opponent's score.

4.1.1 Achievements and Buying Strategy

Also from Figure 3 it becomes clear that the difference in achievement points is much more significant than the difference in the total score. Even though Python-DTU had more valuable zones during most steps of the simulation, SMADAS-UFSC earned more points per step because of achievement points.

[9] `http://www.multiagentcontest.org/downloads?func=fileinfo&id=1133`

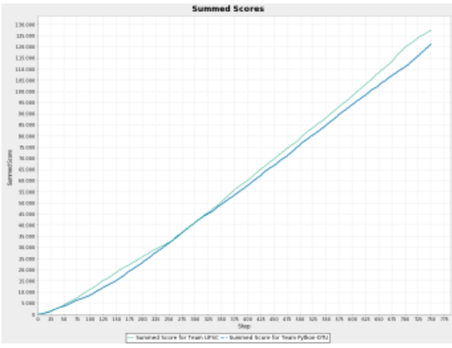


Figure 2: SMADAS-UFSC vs. Python-DTU (Sim 1): Summed scores.

Figure 3: SMADAS-UFSC vs. Python-DTU (Sim 1): Step-scores and Achievement points.

The buying strategy proved to be crucial: the clever strategy implemented by SMADAS-UFSC, which consisted in buying improvements for only one of their saboteurs in an attempt to drive the other teams to spend more achievement points in more agents, worked perfectly in this case. Both teams earned the same number of achievements points: 68. But Python-DTU spent 48 of those points improving the saboteurs, whereas SMADAS-UFSC only used 16 for improving one of theirs. This meant a difference that at the end of the match was of 32 extra points per step for SMADAS-UFSC with little variations after step 350, which was not easily compensated by the zone-score. A point to remark here is that doubling the number of agents per team with regards to the previous edition of the Multi-Agent Programming Contest increased the efficacy of this strategy.

It is worth noticing that, while SMADAS-UFSC attempted to start their buying strategy as early as possible (and also to earn as many achievements as early as possible), Python-DTU's approach was to compensate for the aggressive buying strategy by delaying the first round of buys until step 150. Half of the 16 achievement points spent by SMADAS-UFSC were spent before step 10. Their strategy also attempted to detect whether the other time was buying improvements to limit their own buys, and that explains the later buys at step 175.

Nevertheless, even when in general the buying strategy played in favor of UFSC-SMADAS, there seems to be a correlation between the first bulk of buys for Python-DTU at step 150 and an increase in their step scores. On the other hand, at that point of the simulation both teams were still scattered on the map and had not yet committed to defend a certain area.

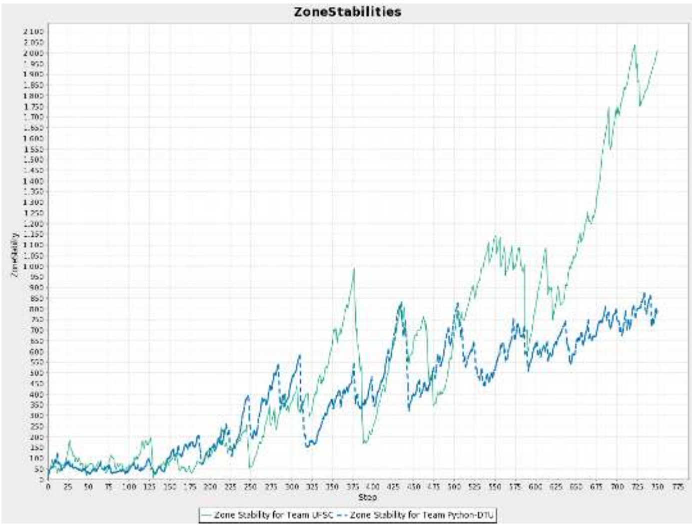


Figure 4: SMADAS-UFSC vs. Python-DTU (Sim 1): Zones' Stability.

4.1.2 Zone Stability

The *zone-stability*[10] graph in Figure 4 reaffirms the idea of overlapping but differentiated zones. Both teams' zone-stability have a clear tendency towards increasing, which means that a number of nodes remain unchallenged. At the same time, none of the zone-stability lines is smooth, which means that several nodes were being lost and recovered during simulation.

Two examples of area domination, one for each team, are presented in Figures 5 and 6. In Figure 5, at step 338 the value of the zone for Python-DTU was 223 and 140 for SMADAS-UFSC. In Figure 6, at step 417 those were respectively 160 and 219.

4.1.3 Actions per Role

SMADAS-UFSC. SMADAS-UFSC's *Explorers* used the `recharge` action the most, 55 percent of the times, followed by the `goto` action (35 percent). The `probe` action was used 303 times (10 percent), 302 of which were successful even though the map had only 300 vertices. The `survey` action was only

[10]The *zone- stability* is a measure that increases when a team keeps dominance of a node, without taking into account the values of the nodes. It was designed for post-match analysis only, as it is not used for computing the scores.

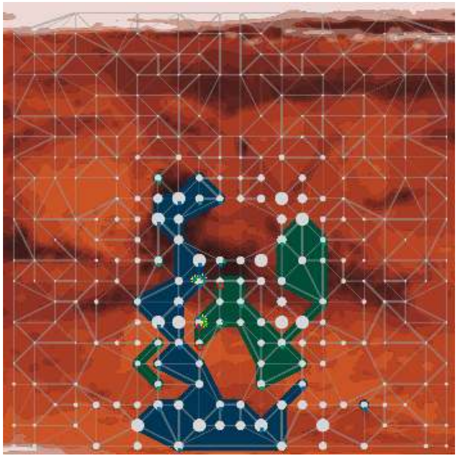


Figure 5: SMADAS-UFSC vs. Python-DTU (Sim 1): Simulation after 338 steps.

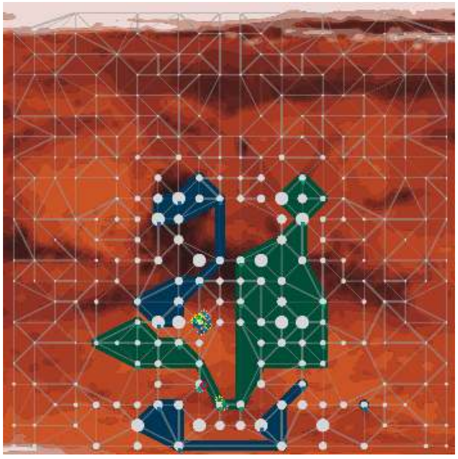


Figure 6: SMADAS-UFSC vs. Python-DTU (Sim 1): Simulation after 417 steps.

used 16 times (less than 1 percent). The *Sentinels* executed the `recharge` action half of the times, followed by the `goto` action (38 percent). They also used the `parry` action 10 percent of the times and the `survey` action only 2 percent. The *Saboteurs* were quite aggressive, using the `attack` action in 51 percent of all cases (85 percent of the attacks were successful). The `recharge` action was used 32 percent of the times, And the `goto` action in only 16 percent of the cases, meaning they were somehow static. The `survey` action was also only used in less than 1 percent of the times (18) and the `buy` action, as mentioned before, was used 8 times. The *Repairers* executed `goto`, `recharge` and `repair` close to a third of the times each (39 percent, 30 percent, and 28 percent respectively). They also chose the `survey` action and the `parry` action around 1 percent of the times each. Finally, the *Inspectors* used mainly the `recharge` action (58 percent) followed by the `goto` action (38 percent). The `survey` action was used only 63 times (2 percent) and the `inspect` action even less, 33 times (1 percent).

Python-DTU. The *Explorers* from Python-DTU used the `recharge` action extensively, 75 percent of the times. The `goto` action, in contrast, was used 15 percent of the times. The `probe` action was used on 305 occasions (10 percent), of which 300 were successful (the number of vertices on the map). The `survey` action was used only in two occasions. The *Sentinels* also used the `recharge` action 75 percent the times. It was followed by the `parry` action, 13 percent of the times, although less than half of the parries were successful.

They used the `goto` action even less than the Explorers, only 8 percent of the times. They also used the `survey` action 5 percent of the times. The *Saboteurs* used the `attack` action 38 percent of the times (76 percent of the attacks were successful). The `recharge` and `goto` actions were used 30 percent of the times each. The `buy` action was used 24 times. They used the `survey` action only once. The *Repairers* executed the `goto` action 35 percent of the times and the `repair` action 34 percent. The third choice was the `recharge` action, 26 percent of the times. They opted for the `parry` action 83 times (3 percent, less than half of the parries were successful) and for the `survey` action 36 times (1 percent). Finally, the *Inspectors* used the `recharge` action the most (67 percent). They used the `inspect` action much more than they rivals (24 percent) and the `goto` action much less (9 percent). They only surveyed in 4 occasions.

4.2 SMADAS-UFSC vs. Python-DTU – Simulation 2

The second simulation of the match between the winners and runner-ups of the contest was won by the latter, by an even closer score of 120.450 to 115.076. Thus Python-DTU maintained the lead during the whole simulation, although SMADAS-UFSC reduced that difference to just 2.474 points at step 578. This is shown in Figure 7. The complete visualization of the simulation can be downloaded at our webpage [11].

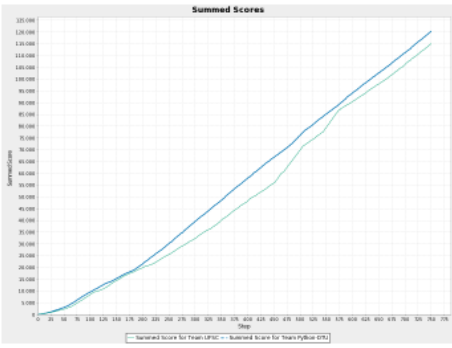


Figure 7: SMADAS-UFSC vs. Python-DTU (Sim 2): Summed scores.

Figure 8: SMADAS-UFSC vs. Python-DTU (Sim 2): Step-scores and Achievement points.

[11] `http://www.multiagentcontest.org/downloads?func=fileinfo&id=1120`

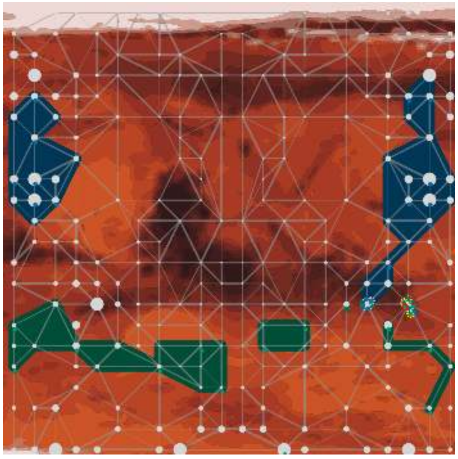


Figure 9: SMADAS-UFSC vs. Python-DTU (Sim 2): Simulation after 362 steps.

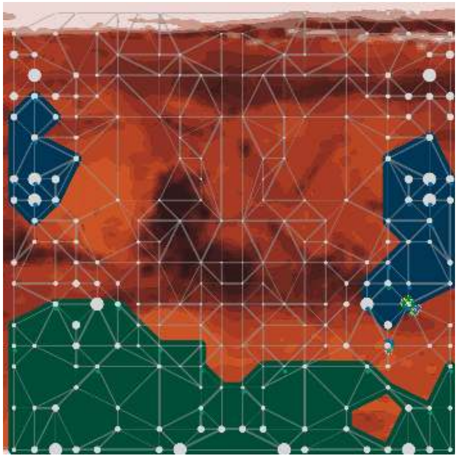


Figure 10: SMADAS-UFSC vs. Python-DTU (Sim 2): Simulation after 481 steps.

4.2.1 Zone Scores and Stability

Figure 8 presents the Step-scores and achievement points at each step of simulation 2. In spite of the two high peaks in the score for SMADAS-UFSC, the advantage for Python-DTU was clear during most of the simulation.

The map in this simulation has different characteristics compared to the first simulation: The most valuable nodes were scattered towards the outer edges of the graph. A clear pattern of which zones each team would attempt to dominate and keep, did not emerge until around step 250. Two different moments during the simulation are presented in Figure 9, at step 362, where the value of the zone for Python-DTU was 176 and 64 for SMADAS-UFSC; and in Figure 10, at step 481, where the values were 172 and 243 respectively. Both figures exemplify what happened during the game, once the teams settled for a region of the map: Python-DTU conquered two zones far away from each other, and although those zones were not very big, they were very stable: In fact, one of the two remained practically unchanged during most of the simulation.

SMADAS-UFSC, on the other hand, managed to build the biggest and most valuable zone by isolating the bottom of the map. However, this was an unstable zone that they were not able to keep for a very long time. Furthermore, SMADAS-UFSC's agents were not standing on the most valuable nodes of that zone, so whenever the zone collapsed, those nodes were lost and thus the zone-score decreased significantly.

Figure 11 shows this difference with respect to zone-stability for each team.

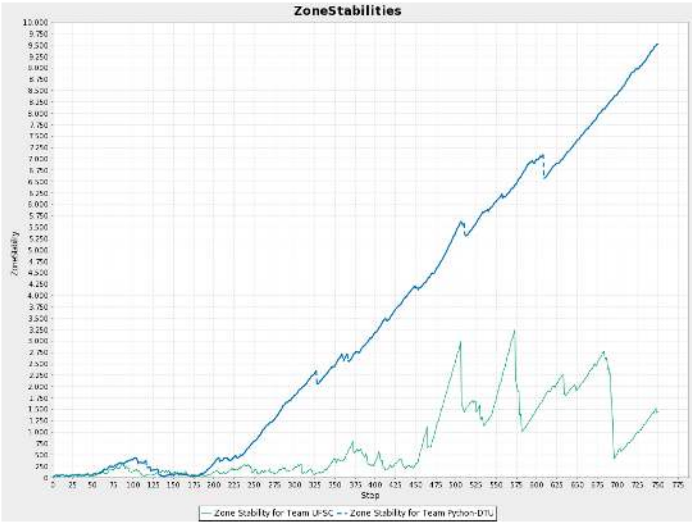


Figure 11: SMADAS-UFSC vs. Python-DTU (Sim 2): Zones' Stability.

As zone-stability takes into account the number of nodes in the zones, the two peaks in the zone-score of SMADAS-UFSC are also slightly reflected in the zone-stability graph. Nevertheless, zone-stability for Python-DTU is still much higher.

4.2.2 Achievements and buying strategy

During the second simulation, the buying strategy applied was the same as during the first one. This time, SMADAS-UFSC earned 68 achievement points and spent 14, whereas Python-DTU earned 66 and used 40. Nonetheless, as it can be seen in Figure 8, during this simulation the difference in achievement-points was not enough to compensate the difference in the zone-scores.

4.2.3 Actions per Role

SMADAS-UFSC. The *Explorers* of team SMADAS-UFSC used the `recharge` action in 61 percent of all cases, followed by `goto` (31 percent) and `probe` (8 percent). The `survey` action was only executed 10 times and the `buy` action was not used at all. Also, the *Sentinels* spend a lot of their time for recharging, i.e., the `recharge` action was used in 60 percent of all cases. Additionally, the main actions for this role were the `goto` action (31 percent) and the `parry` action (7 percent / 5 percent successful). Although the intended main purpose of the sentinel was to be used for surveying the edges the `survey` action was just used in 2 percent of the cases. Probably, because of the high visibility range of this role together with the information of the other roles these few executions were still enough. Finally, this type of agent did not `buy` anything. The behaviour of the *Saboteurs* was implemented in the following way. The `attack` command was executed 1302 times, i.e., in 43 percent of all cases, and was almost always successful (1123 times or 37 percent). The `recharge` action (37 percent) and the `goto` action (19 percent) were the second and third most used actions. The `survey` (25 times) and `buy` (7 times) action were only used sometimes, however the `buy` action was only used by this particular role. The main purpose of the *Repairers* was to go to some agents and repair them, therefore the `goto` (37 percent), the `recharge` (34 percent), and the `repair` (26 percent) action were used most often. The `survey` action was executed 42 times and the `parry` action 37 times (out of that 21 were successful). This is a huge difference to the Python-DTU Repairer that parried just one attack. Lastly, the *Inspectors* used mainly the `recharge` (72 percent) and `goto` action (25 percent). The `survey` action was used 53 times and `inspect` 20 times.

Python-DTU. The *Explorers* of team Python-DTU however used the `recharge` action extensively (more than 75 percent of all cases), followed by the `goto` action (14 percent) and `probe` action (8 percent). The `survey` and `buy` action were never used. The *Sentinels* executed the `recharge` action quite often (62 percent), followed by the `parry` (18 percent in total, but only 6 percent successful) and the `goto` action (12 percent). The `survey` action (7 percent) was only used seldom. The `buy` action was not used at all. The *Saboteurs* used the `attack` action in 39 percent of the cases. 33 percent were successful. A little bit less was the `recharge` action executed (33 percent in total / 30 percent successful). The `goto` action was applied in 27 out of hundred times. Additionally, this agent was the only one using the `buy` action. The action was used exactly 20 times, i.e., in 0.67 percent of the cases. Finally, the agent did not use the `survey` action once. The *Repairers* executed `goto` in 38 of the cases, followed by the `repair` (28 percent) and `recharge` action (33 percent in total / 31 percent successful). The `survey` action was used 17 times, the `parry` action just three times (out of that only one was successful) and the `buy` action was never executed. Finally, the *Inspectors* used mainly the `recharge` action (83 percent), followed by `inspect` (11 percent) and `goto` (5 percent). The `survey` action was executed 5 times and `buy` was never used.

4.3 PGIM vs. AiWYX – Simulation 1

The team *AiWYX* clearly won all simulations against *PGIM*. While the first simulation ended 81562 to 212016, the second resulted in 68748 to 107600 and the last in 75846 to 112466. The final position of *AiWYX* was 5 and *PGIM* got the 6th place.

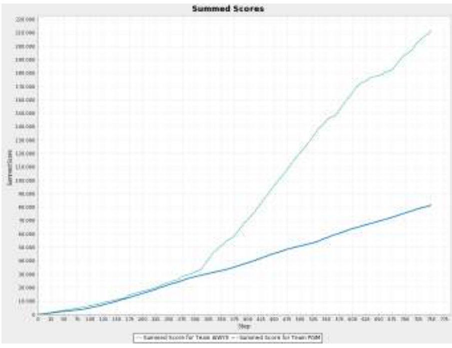


Figure 12: PGIM vs. AiWYX (Sim 1): Summed scores.

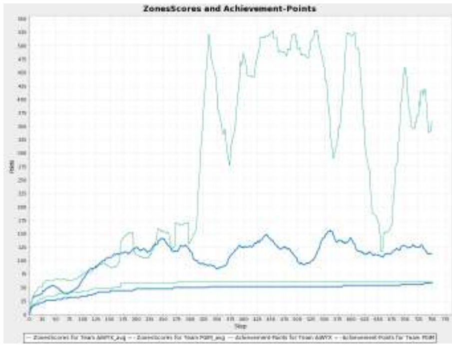


Figure 13: PGIM vs. AiWYX (Sim 1): Step-scores and Achievement points.

During the beginning of the match both teams were at the same level. At step 170 *AiWYX* conquered an area of more than 640 nodes but was not able to keep it for a longer period (cf. Figure 14). At step 312 *AiWYX* finally stabilized its zone(s) (cf. Figure 16 and 15). The team *PGIM*, however, was not able to conquer zones larger as 160 nodes and got therefore only the achievement for holding 80 nodes at the same time.

AiWYX used a novel strategy (not seen in the competition so far) for building zones: Instead of trying to conquer a small zone, probing the nodes in order to increase the value of the zone and finally defending, the team was positioning itself around an opponent's zone and thereby isolating the opponents zone from the rest of the graph. Figure 14 shows such a zone. At step 312 *AiWYX* finally stabilized its zone(s) (cf. Figure 15 and 16). As one can see this resulted in very large zones, basically containing all nodes the opponents did not conquer.

Nevertheless due to the lack of probing all conquered nodes the team *AiWYX* did not score all possible points but only a small subset. Additionally, the strategy was highly depending on the size of the map and more effective on larger maps. That is probably the reason why the team *AiWYX* scored the most points per simulation but did not reach a better place in the competition.

The complete visualization of the simulation can be downloaded from our

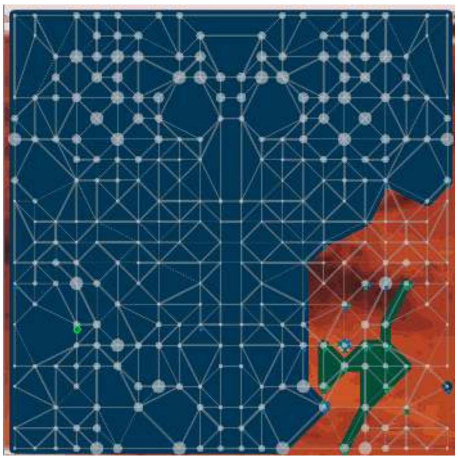


Figure 14: Simulation after 170 steps.

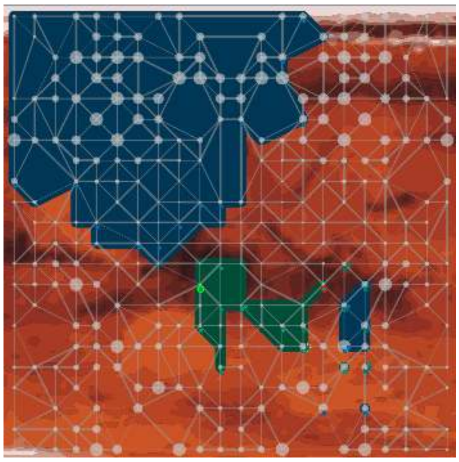


Figure 15: Simulation after 312 steps.

webpage [12]. In the following, we will discuss this simulation in more detail.

4.3.1 Scores

The evolution of the zone scores and achievement points are depicted in Figure 13. While the development of the achievement points is similar (both teams did not invest the points for agent improvements), the flows of the zone scores are different. From step 0 to 300 it was a head to head competition but after step 312 *AiWYX* was able to occupy a large zone and *PGIM* was not able to increase its zone score anymore.

4.3.2 Zone Stability

The zone stability of team *PGIM* was low, i.e., under 500 points per step. In contrast, the zone stability of *AiWYX* was quite good and was almost always higher than that for *PGIM*. This is one reason why the team *AiWYX* won the match.

4.3.3 Achievements

The team *AiWYX* conquered a zone with an impressive value of 640 points, attacked 640 times the opponents successfully, probed 160 nodes, and surveyed 640 edges. Additionally, It inspected 20 times an opponent. An interesting fact is that the agents did not try to parry an attack.

[12] http://www.multiagentcontest.org/downloads?func=fileinfo&id=1148

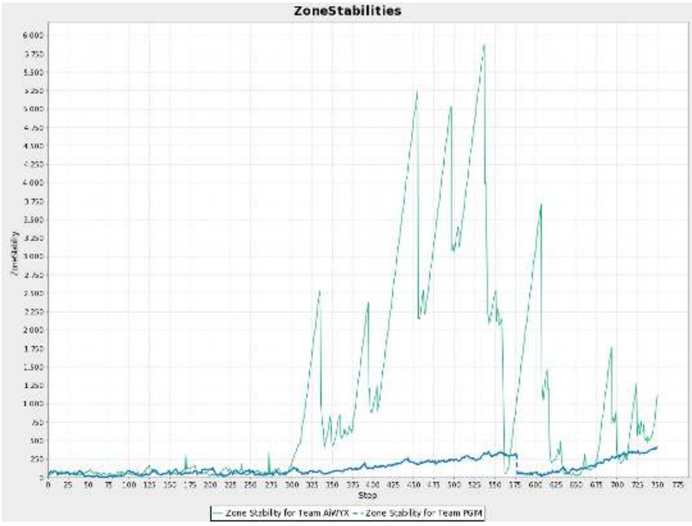


Figure 16: PGIM vs. AiWYX (Sim 1): Zones' Stability

The team *PGIM* made the following highest achievements: It conquered an area of 80 nodes, attacked 320 successfully, probed 80 nodes and surveyed 640 edges. It inspected 10 times an opponent and parried 40 times attacks successfully.

4.3.4 Actions per Role

AiWYX. The *Explorers* of team *AiWYX* used the `recharge` action extensively (more than 50 percent of all cases), followed by the `goto` action (35 percent) and `probe` action (10 percent). The `survey` action was just used in just 1.7 percent. The *Sentinels* executed the `recharge` action quite often (53 percent), followed by the `goto` action (32 percent) and the `survey` action (4 percent). The *Saboteurs* used the `goto` action in 42 percent of the cases, followed by the `attack` (35 percent) and `recharge` action (22 percent). The *Repairers* executed `goto` in 54 the cases, followed by the `repair` (26 percent) and `recharge` action (18 percent). Finally, the *Inspectors* used mainly the `goto` (41 percent) and `recharge` action (56 percent). The `inspect` was just used 18 times (0.6 percent). `survey` was executed in 1.73 percent of the cases.

PGIM. The *Explorers* of team *PGIM* however used the `goto` action in 56 percent of all cases. 19 percent of the time they executed the `skip` action which does not have an effect. It would be more efficient to use the `recharge` action instead. This action was used in 11 percent of the cases. Finally, `probe` and `survey` were executed 8 and 5 percent of the times. The behaviour of the *Sentinels* was not optimal. The `skip` action was the most often used action (49 percent) followed by a `goto` command (37 percent). `parry` (2 percent), `survey` (4 percent), and `recharge` (8 percent) were just used seldom. Also the behaviour of the *Saboteurs* was not implemented in a good way. The `skip` action was used 1304 times, i.e., 43 percent of all cases although a `recharge` (13 percent) would be more efficient. The `goto` action was executed in 27 percent of all cases, followed by `survey` (3 percent) and `attack` (14 percent). For the *Repairers* the `goto` action was the main action (48 percent). This was followed by the `repair` (18 percent) and `recharge` action (21 percent). The `skip` action was executed 296 times, that corresponds to 10 percent. `survey` was used 84 times, i.e., 2,8 percent. The *Inspectors* used mainly the `goto` action (55 percent), followed by `skip` (26 percent) while `recharge` (14 percent) would be the better option. `survey` was used in 4 percent of the cases and `inspect` just 21 times (0,7 percent).

4.4 TUB vs. LTI-USP – Simulation 1

The team *TUB* won all simulations against *LTI-USP*. While the first simulation was a head to head finish (it ended 75083 to 77757), the second resulted in 66660 to 101310 and the last in 85187 to 165577. The final position of *TUB* was 3 and *LTI-USP* got the 4th place.

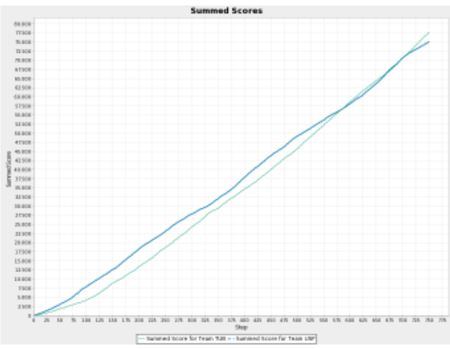


Figure 17: *TUB* vs. *LTI-USP* (Sim 1): Summed scores.

Figure 18: Step-scores and Achievement points.

During the beginning of the match *LTI-USP* was performing a little bit better than *TUB*. But at step 582 (Figure 19) *TUB* started to get more points than *LTI-USP*. Although *LTI-USP* was able to catch up (Figure 20) and temporally outrun the opponent (Figure 21) in step 706 (Figure 22)*TUB* took over again

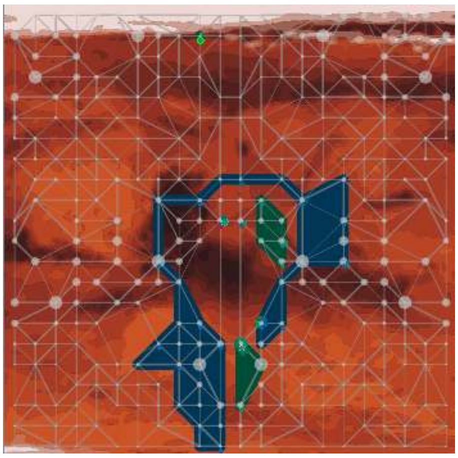


Figure 19: Simulation after 582 steps.

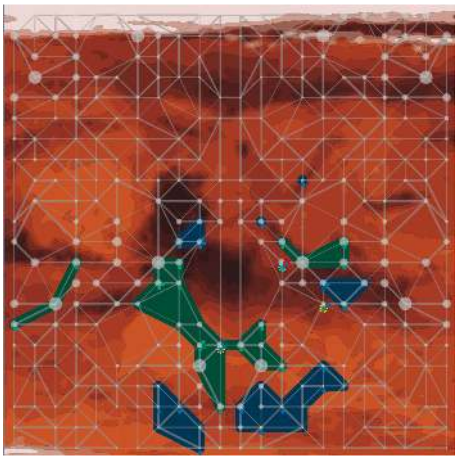


Figure 20: Simulation after 668 steps.

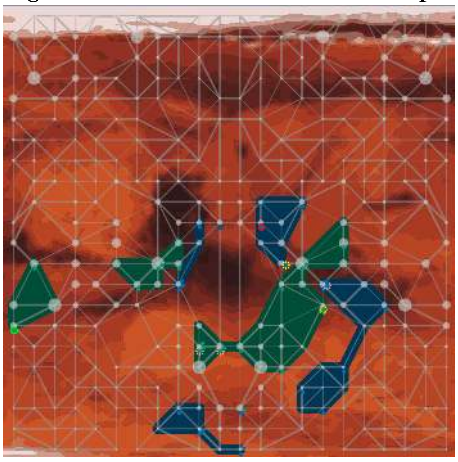


Figure 21: Simulation after 669 steps.

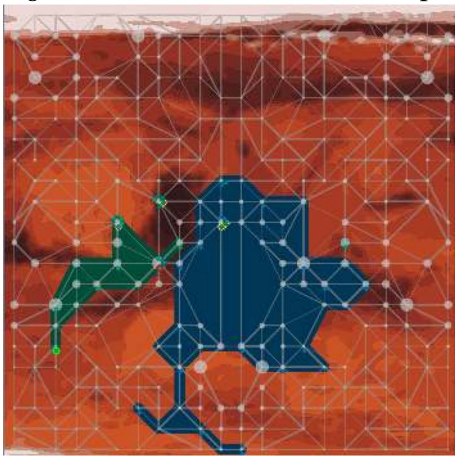


Figure 22: Simulation after 706 steps.

and won in the end being just some few points in front. The reason for this was that *TUB* had a almost stable zone that was bigger than the zone of the team *LTI-USP*.

The complete visualization of the simulation can be downloaded from our webpage [13]. In the following, we will discuss this simulation in more detail.

[13] `http://www.multiagentcontest.org/downloads/func-startdown/1146/`

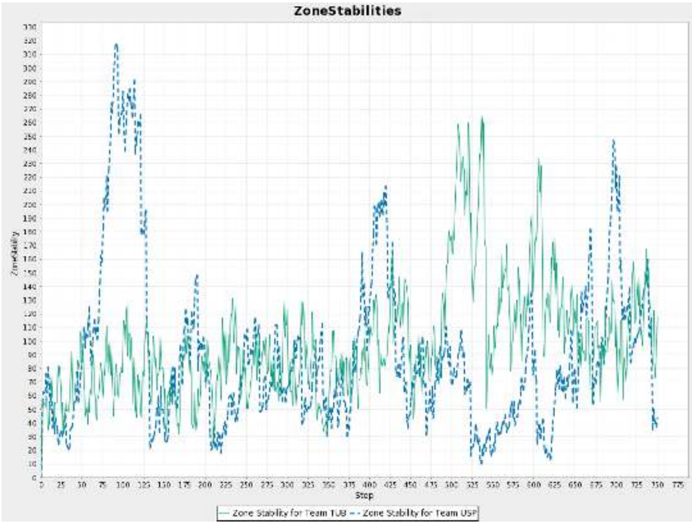


Figure 23: Zones' Stability

4.4.1 Scores

The evolution of the zone scores and achievement points are depicted in Figure 18. Both teams were performing quite similar, however TUB got more achievement points. Also, both teams invested some of the achievement points again for improvements for their agents.

4.4.2 Zone Stability

The zone stability of both teams was quite low, i.e., under 400 points per step.

4.4.3 Achievements

Team *TUB* got all achievements earlier than *LTI-USP* and additionally achieved more. In total *TUB* attacked 640 times, surveyed 640 times, proved 160 nodes and conquered once an area of 160 nodes. Furthermore it inspected 20 agents. However, the team *TUB* did not get any achievement points for parrying.

The team *LTI-USP* attacked only 320 times, surveyed 640 times, proved as well 160 nodes, parried 160 attacks and inspected also 20 agents.

4.4.4 Actions per Role

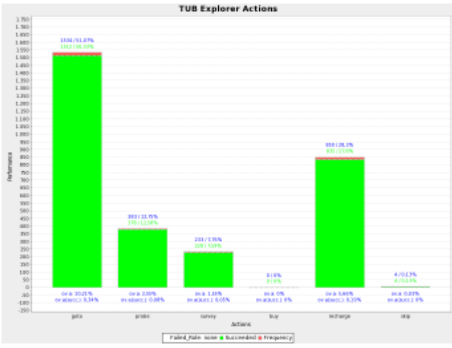


Figure 24: *TUB* Explorer Actions

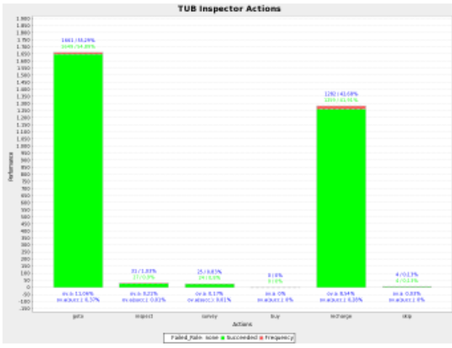


Figure 25: *TUB* Inspector Actions

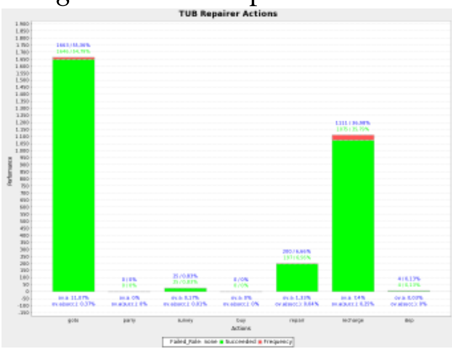


Figure 26: *TUB* Repairer Actions

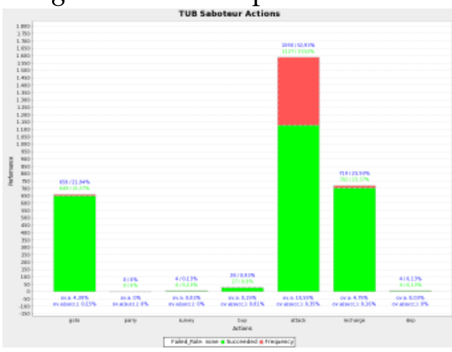


Figure 27: *TUB* Saboteur Actions

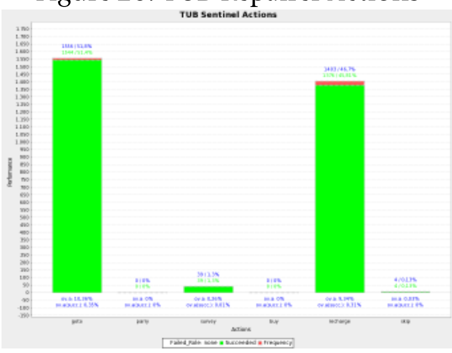


Figure 28: *TUB* Sentinel Actions

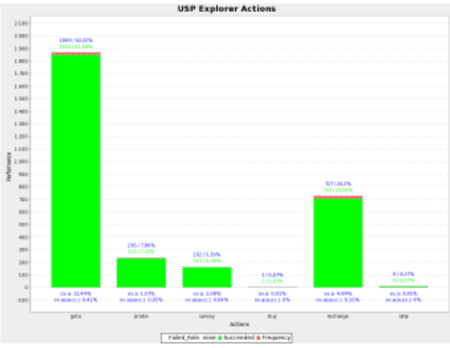


Figure 29: *LTI-USP* Explorer Actions

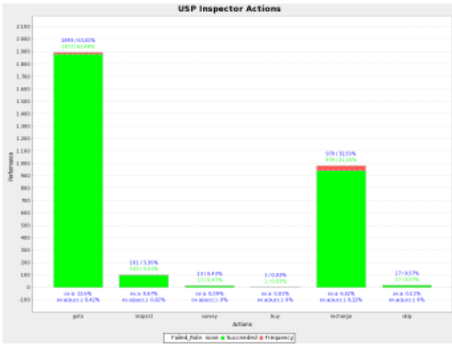


Figure 30: *LTI-USP* Inspector Actions

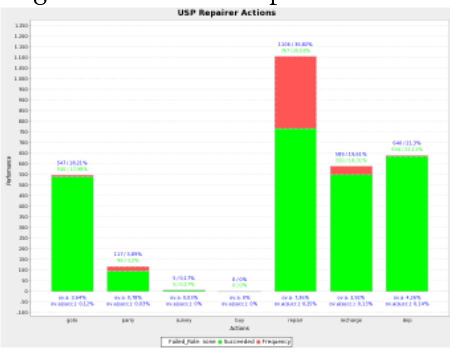


Figure 31: *LTI-USP* Repairer Actions

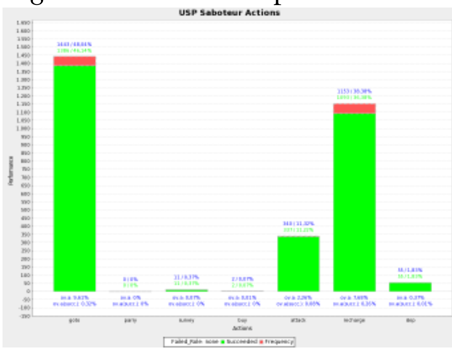


Figure 32: *LTI-USP* Saboteur Actions

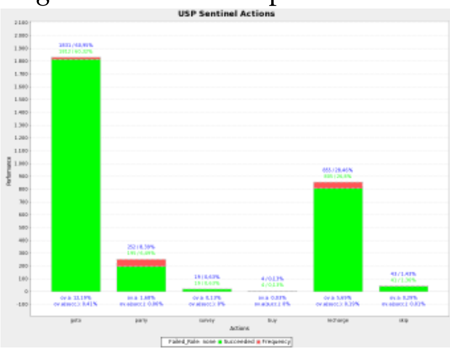


Figure 33: *LTI-USP* Sentinel Actions

4.5 Streett vs. TUB – Simulation 2

Streett was the worst-performing team during the 2012 contest and the only one that was unable to win any simulations. This simulation, that *TUB* won 120.565 to 7.174 can be used to explain some of the reasons for *Streett*'s bad performance.

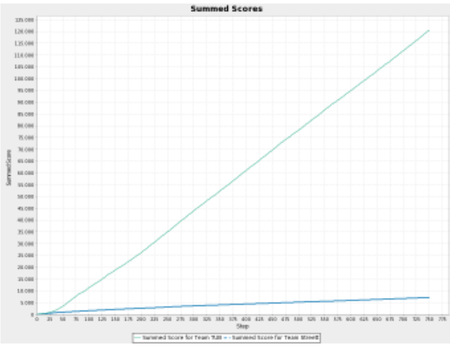


Figure 34: *Streett* vs. *TUB* (Sim 1): Summed scores.

Figure 35: Step-scores and Achievement points.

The most notable issue regarding *Streett*'s bad performance had to do with disabled agents, for which the team had many flaws. Firstly, disabled agents weren't moving at all and only used the recharge action, presumably waiting for repairs. Secondly, only one of the four repairers was executing the repair action, while the other three only recharged and moved around, possibly attempting to contribute in zone making. Finally, the only active repairer was buggy: it only repaired another agent successfully three times at the beginning of the simulation; afterwards, it remained static in a node along with two disabled teammates but always attempted to repair another agent that was in a different node, and thus failed. The result was that *Streett* only had a few agents active throughout the game, and were only those that remained away from the enemy's zone.

The situation was more or less stable relatively soon: at step 60 (Figure 36), half of the agents from *Streett* were disabled, while the repaired shared a node with two of them. *TUB* had built a valuable zone and used their agents to defend it until the end of the game. By step 200 (Figure 37) *Streett* had lost 13 agents. Figure 38 shows the moment when another agent from *Streett* moves close to *TUB*'s area and gets attacked and disabled. The 14 disabled agents remained in the exact same positions until the end of the game (Figure 39).

The complete visualization of the simulation can be downloaded from our webpage [14].

[14] `http://www.multiagentcontest.org/downloads/func-startdown/1095/`

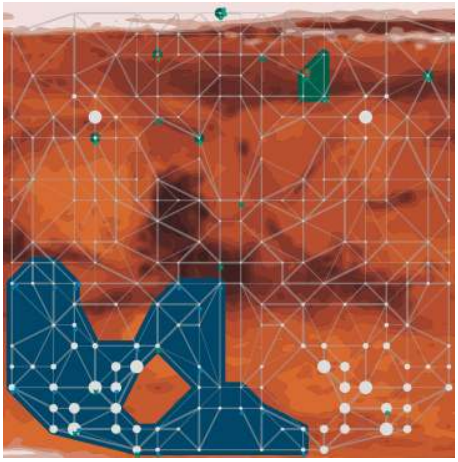


Figure 36: Simulation after 60 steps.

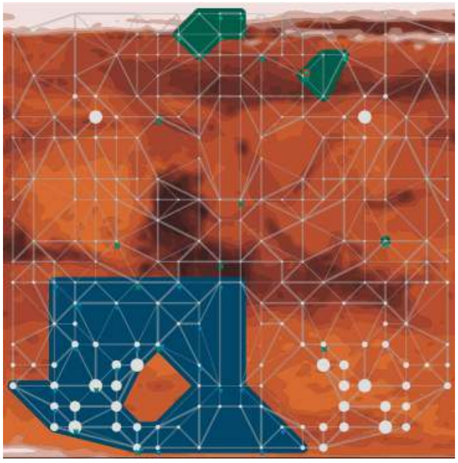


Figure 37: Simulation after 200 steps.

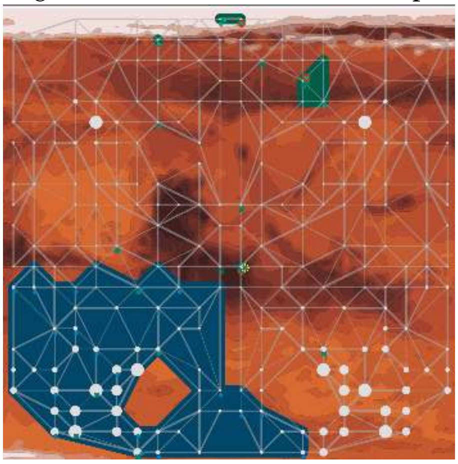


Figure 38: Simulation after 390 steps.

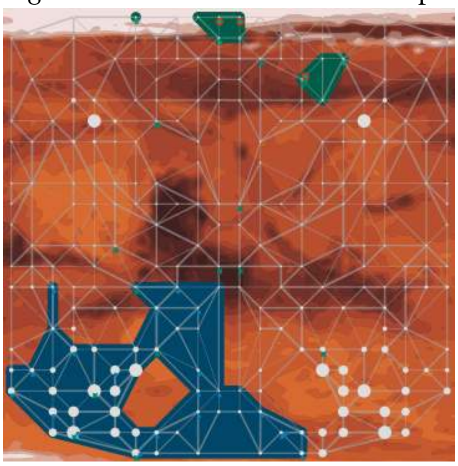


Figure 39: Simulation after 750 steps.

4.5.1 Scores

The difference in the score was almost exclusively because of the zone's score. Both teams stabilized in differentiated parts of the map, but while *TUB* used all the agents to build a big, valuable zone, *Streett* only built two very small, almost worthless zones involving very few agents, the rest of them being disabled and never repaired.

Figure 40: Zones' Stability

4.5.2 Zone Stability

TUB zone was big and stable as can be seen in the Zone Stability graph: although the value is fluctuating, the trend is always increasing. *Streett* also had an increasing trend in it's zone stability graph, although with much lower values.

4.5.3 Achievements

In the beginning of the simulation both teams where more or less even in terms of achievements earned. Nevertheless, *TUB* kept earning new achievements throughout the game, whereas *Streett* lost effectivity soon (step 32). None of the teams earned parry achievements, and both earned only few attack achievements: only `attack5` for *Streett* and `attacked20` for *TUB*, the latter due to disabled *Streett*'s agents not being repaired.

Both teams spent achievement points in improvements: *TUB* ended the match with 2 achievement points after having spent 40, and *Streett* spent all the 28 achievement points earned during the simulation.

4.5.4 Actions per Role

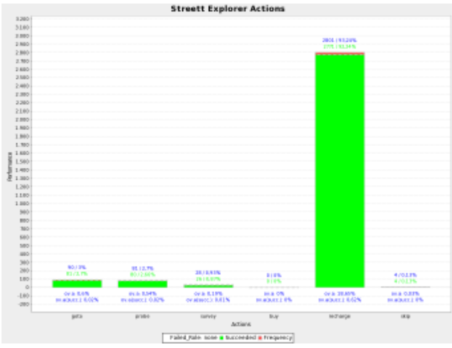


Figure 41: *Streett* Explorer Actions

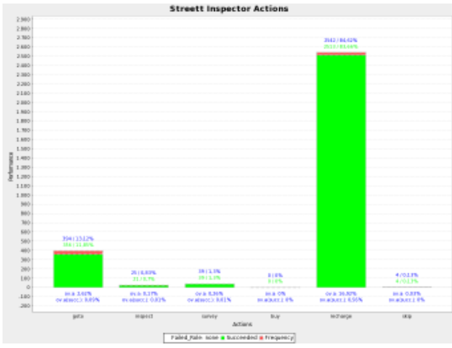


Figure 42: *Streett* Inspector Actions

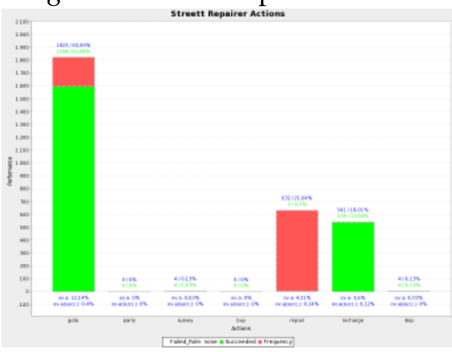


Figure 43: *Streett* Repairer Actions

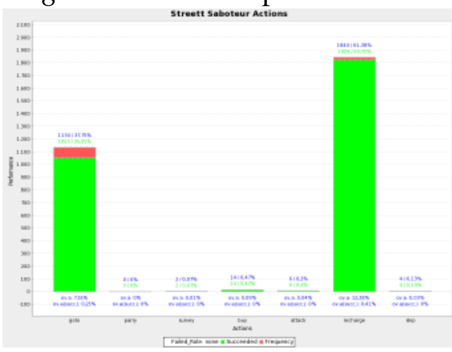


Figure 44: *Streett* Saboteur Actions

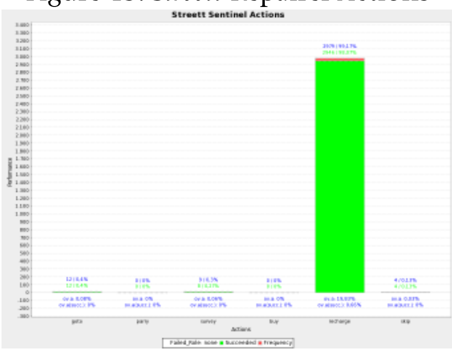


Figure 45: *Streett* Sentinel Actions

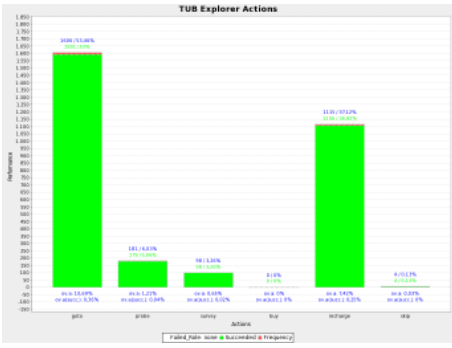


Figure 46: *TUB* Explorer Actions

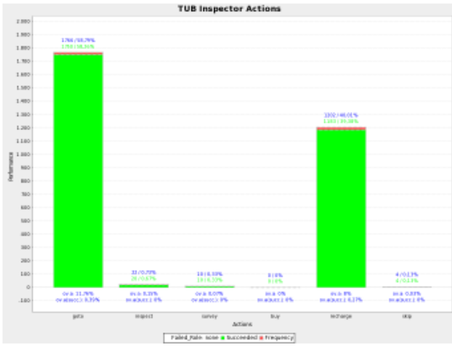


Figure 47: *TUB* Inspector Actions

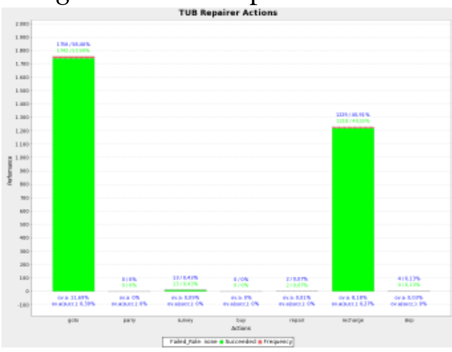


Figure 48: *TUB* Repairer Actions

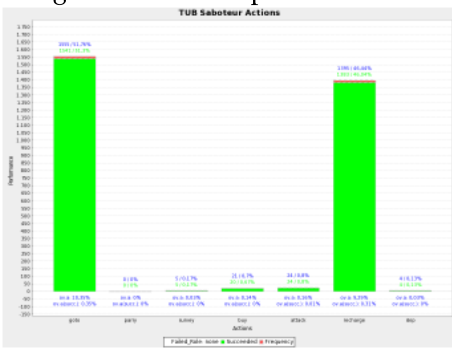


Figure 49: *TUB* Saboteur Actions

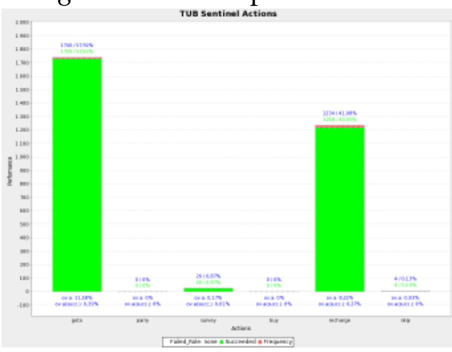


Figure 50: *TUB* Sentinel Actions

5 Summary, Conclusion and Future of the Contest

This paper provides an overview of the most recent edition of the Multi-Agent Programming Contest. We have introduced the Contest in general, and we elaborated on the current scenario in a more detailed way. We have also introduced the teams that took part and evaluated their performance. We compared three of the more interesting matches using our new visualisation and statistics modules.

This is our third newly designed scenario that we will also use, with some modifications and lessons learned from the 2012 edition, for the Contest in 2013. It is time to lean back and consider what we have achieved so far. *What conclusions (if any) can we draw from the "Agents on Mars" scenario? Can we observe some trends in the quality of the teams? What is the impact on the ProMAS community?* While these are critical and difficult questions that might be answered differently by different people, we collect a few observations that we consider relevant.

- Both times a dedicated Multi-Agent Programming Language/Platform won, but runner-up was Python-DTU, which did not use a dedicated platform, but was inspired by MAS technology.

  Nevertheless, other examples (e.g., the teams ranked 5–7 in this years edition) show that ad hoc implementations seem to perform worse than MAS inspired systems.

- The introduction of a qualification round increased the stability of the teams and therefore the whole contest a lot. This feature will be kept.

- Teams performing for the second time usually perform better. But the winners were both first time participants.

- The contest helped a lot to find bugs in the used platforms. This is an observation we made throughout the history of the contest. So it seems the scenario is demanding and most features of the used platform/language are indeed used (so that potential bugs surface). One team participated exactly because of this reason (testing their platform).

- We usually end up with as few as 7 to 9 teams that seriously want to participate. We believe this number could be much higher and does not really show a great impact on our community. On the other hand we have quite a variation: it is not always the same participants. Over the last 3 years, we had 20 different teams participating.

- The overall performance of the teams improved a lot with each new contest, although we increased the complexity considerably (size of the map, number of agents, difficulty of the task).

- Compared with the *cows and cowboys* scenario, we see much more co-operation among the agents, more dynamic behaviour, and a lot more interaction with the opposing team. In addition, the data to be handled (observing the environment, messages between the agents) has also increased a lot. While we have not yet excluded centralized approaches, the sheer amount of data makes it difficult for the systems to provide each agent with the central memory of the whole system.

  Also, in the current scenario, the computational costs of Dijkstra's algorithm is high so that it is not feasible for all agents to execute it at the same time.

- In the current scenario, there are indications that buying health and strength is much more important than investing the money for other reasons. Thus it may pay off to find a more balanced scenario that allows for more diverse strategies of the teams. This point makes us reconsider the precise values of the different parameter we have in our scenario.

The amount of work that went into implementing a team varied from one person with 250 person-hours to 6 people with 800 person hours and from 1500 to 10000 lines of code (the latter because no dedicated technology was used, interestingly, that was done by one single person).

It would be interesting to assess if it would be beneficial to steer the Contest into a more specialized direction in order to strengthen its niche in the research ecology. This includes but is not limited to focusing on the planning aspect of the competition, leaving behind path planning as the main facet of agent deliberation.

We could also focus on using a massive number of agents: lots of agents with different roles and thus different capabilities. This would allow us to take into account the scalability of agent-oriented programming platforms.

Additionally it would be worthwhile to focus on agent communication and to evaluate that aspect of the tournament by routing agent-messages through the *MASSim* server for proper evaluation.

Last but not least, the most important part of the contest are the contestants: We hope to attract more teams in the future — the contest is an excellent opportunity for a student project on Bachelor or Master level.

References

[1] T. Behrens, M. Dastani, J. Dix, M. Köster, and P. Novák, editors. *Special Issue about Multi-Agent-Contest*, volume 59 of *Annals of Mathematics and Artificial Intelligence*. Springer, Netherlands, 2010.

[2] Tristan Behrens, Mehdi Dastani, Jürgen Dix, Jomi Hübner, Michael Köster, Peter Novák, and Federico Schlesinger. The multi-agent programming contest. *AI Magazine*, to appear, 2013.

[3] Tristan Behrens, Mehdi Dastani, Jürgen Dix, and Peter Novák. Agent contest competition - 4th edition. In *Proceedings of Sixth international Workshop on Programming Multi-Agent Systems, ProMAS'08*, volume 5442 of *LNAI*. Springer, 2008.

[4] Tristan Behrens, Mehdi Dastani, Jürgen Dix, and Peter Novák. Agent contest competition: 4th edition. In Koen V. Hindriks, Alexander Pokahr, and Sebastian Sardiña, editors, *Programming Multi-Agent Systems, 6th International Workshop (ProMAS 2008)*, volume 5442 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2009.

[5] Tristan Behrens, Jürgen Dix, Jomi Hübner, Michael Köster, and Federico Schlesinger. MAPC 2011 Documentation. Technical Report IfI-12-01, Clausthal University of Technology, December 2012.

[6] Tristan Behrens, Jürgen Dix, Jomi Hübner, Michael Köster, and Federico Schlesinger. MAPC 2011 Evaluation and Team Descriptions. Technical Report IfI-12-02, Clausthal University of Technology, December 2012.

[7] Tristan Behrens, Koen Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61:3–38, 2011.

[8] Tristan Behrens, Michael Köster, Federico Schlesinger, Jürgen Dix, and Jomi Hübner. The Multi-agent Programming Contest 2011: A Résumé. In Louise Dennis, Olivier Boissier, and Rafael Bordini, editors, *Programming Multi-Agent Systems*, volume 7217 of *Lecture Notes in Computer Science*, pages 155–172. Springer Berlin / Heidelberg, 2012.

[9] Mehdi Dastani, Jürgen Dix, and Peter Novák. The first contest on multi-agent systems based on computational logic. In Francesca Toni and Paolo Torroni, editors, *Computational Logic in Multi-Agent Systems, 6th International Workshop, CLIMA VI*, volume 3900 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2005.

[10] Mehdi Dastani, Jürgen Dix, and Peter Novák. The second contest on multi-agent systems based on computational logic. In Katsumi Inoue, Ken Satoh, and Francesca Toni, editors, *Computational Logic in Multi-Agent Systems, 7th International Workshop, CLIMA VII*, volume 4371 of *Lecture Notes on Computer Science*, pages 266–283. Springer, 2006.

[11] Mehdi Dastani, Jürgen Dix, and Peter Novák. Agent contest competition - 3rd edition. In M. Dastani, A. Ricci, A. El Fallah Seghrouchni,

and M. Winikoff, editors, *Proceedings of ProMAS '07, Revised Selected and Invited Papers*, number 4908 in Lecture Notes in Artificial Intelligence, Honululu, US, 2008. Springer.

[12] Michael Köster, Federico Schlesinger, and Jürgen Dix. MAPC 2012 Evaluation and Team Descriptions. Technical Report IfI-13-01, Clausthal University of Technology, jan 2013.

Part II
Team Descriptions

6 SMADAS-UFSC

Team SMADAS-UFSC is from Federal University of Santa Catarina, Brazil. It had six team members. The language of choice for agent development was Jason. Besides normal agent-communication provided by Jason, agents shared a common data-structure (blackboard) for storing the graph topology.

SMADAS: a Cooperative Team for the Multi-Agent Programming Contest using Jason

Maicon Rafael Zatelli, Daniela Maria Uez, José Rodrigo Neri,
Tiago Luiz Schmitz, Jéssica Pauli de Castro Bonson, and Jomi Fred Hübner

Department of Automation and Systems Engineering
Federal University of Santa Catarina
CP 476, 88040-900 Florianópolis - SC - Brasil
{xsplyter,dani.uez,jrf.neri,tiagolschmitz,jpbonson}@gmail.com,
jomi@das.ufsc.br

Abstract. In this paper we describe the SMADAS system used for the Multi-Agent Programming Contest in 2012. This contest offers an useful context to evaluate tools, techniques, and languages for programming MAS. It is also a good opportunity to learn agent programming and test new features we are developing in our projects. Throughout the paper we highlight the main strategies of our team and comment on the advantages and disadvantages of our system as well as some improvements that still could be done. One important result from this experience regards the agent programming language we used, it provides suitable abstractions for the development of complex system and shows an increment in its maturity since no bugs was discovered this year.

1 Introduction

The empirical evaluation of proposals in the context of Multi-Agent Systems (MAS) is a quite complex task and the Multi-Agent Programming Contest [1, 3][1] offers an useful context for doing this evaluation. In particular, the latest Mars scenario has emphasised solutions based on cooperation, coordination, and decentralisation which are important topics for our research. This contest is thus selected as the environment to evaluate the proposals being developed by the authors in their master and Phd thesis. Among the authors, we have one PhD student, three master students, and one undergraduate student. The main approach is (*i*) to develop a *base* MAS for the contest, then (*ii*) the master and PhD students will change the base system using their corresponding proposals, and finally (*iii*) each proposal can be evaluated and compared against the base system. In this paper we report the development and the main features of this base team, called SMADAS (the acronym of our research group). Another objective for attending the contest is to improve the experience in developing MAS. Since most of the authors are just beginning on the domain, the concrete experience is important for their overall learning and maturity in critical analysis.

[1] `http://multiagentcontest.org`

2 System Analysis and Design

For the analysis of our systems, we adopted a prototype driven approach instead of a well known software engineering methodology because the problem seemed quite simple to solve and we had no experience with them. Thus we decided that it was better to use our time developing the system than learning a methodology.

Based on the agent contest scenario description, we divided the overall problem in sub-problems, each one analysed in detail: exploration, exploitation, attack and defense, buy, repair, and inspection. A team member was engaged with programming each strategy discussed on biweekly meetings. Forty five versions of the system were produced in this phase. These versions were tested and compared with the best teams from the last contest [6, 8, 7, 2] and also against our own versions of the system in order to select the most efficient one. In these preliminary tests, we identified some good strategies for the final implementation. To develop the SMADAS system, we spent about 500 hours, most of them testing the strategies.

The system has 20 agents of five types: repairer, saboteur, explorer, sentinel, and inspector. We considered two main distinct phases: exploration, in which the explorers identify all vertices and nodes in the map and find the best zones, and exploitation, where all agents try to conquest and defend these zones. During the match, if an agent senses a nearby enemy it calls a saboteur to attack it, and also if the agent is damaged it tries to find a repairer to be fixed.

Our agents are able to decide their own actions, however this autonomy produces some conflicting situations like two agents deciding to exploit different zones. These situations are solved using a centralized approach, which consists of a specific agent been responsible for the group decision. For example, one of the explorers defines the zones to exploit and one of the repairers defines the reparation order. Some conflicting situations are simply prevented by using a predefined priority order among the agents, where agents with higher priorities acts before agents with less priority.

The coordination among the agents is based on two communication mechanisms: blackboard and message exchanging. The blackboard is used to provide a global graph view to the agents, since some important information about the graph structure is synchronized in it. We decided to use a blackboard because the agents need an overall view of the scenario to be able to define the system exploitation strategy. The message exchanging is used to share information about the inspected enemies, the ally agent actions and damages, and about the map zones. The communication protocol used when a damaged agent needs to be repaired is shown in Fig. 1. It consists of the agent asking a repairer that contacts the other repairers to find out which one is the closest to the damaged agent. Then the other repairers inform their positions and the closest one is selected to repair the damaged agent. Thus, the selected repairer will send to the damaged agent the meeting path.

The SMADAS system is a truly MAS because the agents are autonomous, reactive, and proactive. They have autonomy to decide how and when to execute most of their actions, except the few conflicting situations explained before.

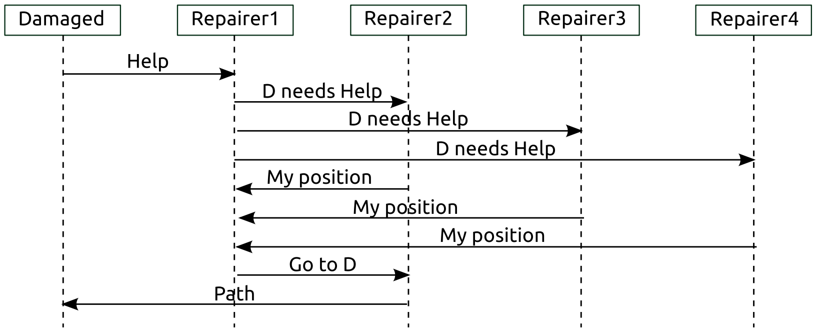


Fig. 1: A communication protocol used to define which repairer will repair a damaged agent. The damaged agent asks the `repairer1` for help, the `repairer1` then contacts the others repairers to find which one is the closest to the damaged agent. All repairer send their position and the `repairer1` elects the closest one. The selected repairer then sends the meet point to the damaged agent.

However, the agents also perform some actions in reaction to environment events, like the start of the step or a received message. Other reactive actions occurs when a saboteur attacks an enemy agent that is in the same vertex or when an agent runs away or defends itself from an enemy saboteur on the same vertex. Furthermore, the agents have a proactive behaviour, that shows up when they try to find a better vertex that improves the team score, contact the repairer when they are damaged, or look for enemies to attack.

3 Strategies

In our strategy both individual and group behavior are important. While the individual behavior is important when the agents are isolated in the map, the group behaviour is responsible for preventing redundant actions and for producing a coherent and cooperative global result. The agents are proactive in order to get achievement points and obtaining a good score. They also use their beliefs and the exchanged information to decide their next action.

As commented in the previous section, we consider two main strategies: exploration and exploitation. In the exploration phase the agents just explore the map and try to get as most achievement points as possible. After step 15, our agents go to a good zone to conquer it.

Since achievement points are important and they accumulate in each one of the 750 steps, it is desirable to obtain them as soon as possible. However, some achievements are more complicated to conquer after some time, hence they can be ignored. For example, it does not make sense to survey all edges in the graph, considering it takes a long time to be performed. Instead of it, our agents stay in a vertex getting more score by exploiting water wells. For the same reason we are not interested on inspecting all opponent agents, thus our inspectors only inspect them when they are near.

After the exploration phase, the exploitation phase starts. One of our explorers reasons about which are the two best zones in the map to be exploited. Exploiting two zones is advantageous since the map is symmetric and it is particularly important against teams that keep only one zone. In order to do that, we used a modified version of the BFS algorithm, that is run for all vertex, summing their values until some depth. The vertex with the highest sum represents where the best zone is (zone 1). After it, the algorithm tries to find the second best vertex to set the second best zone (zone 2), which may have some intersection with the first one. This algorithm is not optimal because its result is always a circular shape, when the ideal choice often has a free shape.

When the good zones are defined, an explorer organises the agents in two groups, one for zone 1 and another for zone 2. Each group has 10 members, with two agents of each type. The agents are then informed about the central vertex of its zone and how far they can go from it. The central vertex of an area is the one discovered in the exploration phase with the best sum. The distance they can go from it defines the border of the corresponding zone. After it, the agents are positioned in their zones. The non-saboteur agents take positions in vertices that have two neighbour vertices belonging to our team, but without anyone there. The saboteur agents scout their zones and attack opponents inside it, they also attack near enemy zones. We assume that if the enemy zone is not near, the opponent probably has a small zone and we do not need to attack them.

Table 1 shows the strategies and plans for each type of agent. There are plans with more steps (buy, repair, probe) and plans where the agents simply react (attack, parry, inspect, recharge, survey). We noticed that usually long-term plans are not a good idea, because the environment changes quickly. The strategies are explained in more details below.

- Buy: we concluded that it is be better to do not buy many things. We noticed it through tests between our MAS with a buying strategy where the agents buy more things against one where the agents just buy few things, and the second strategy won all matches in all simulations. Firstly the buying strategy consisted of only buying upgrades for the saboteurs: buy sabotage devices to have a strength equal to the highest enemy saboteur health value, and buy shields to have health one time greater than the highest enemy saboteur strength value. We did a second version of this strategy where just one saboteur (`Hulk`) buys upgrades, this had the benefit of decreasing our expenses while also making agent teams with a similar strategy waste money. Another improvement of the buying strategy was the addition of an agent named `Coach`, which received information about our enemies upgrades from the inspectors and used them to notice whether the enemy team is buying or not, if they were not buying anything this agent informs the agent `Hulk` to stop buying upgrades in the matches against this team and then save achievement points.

- Attack: the saboteurs always attack the opponent saboteurs first, and then the repairers. However, in the initial steps, attacking the explorers would be a good second option too, since it would be harder for the opponent team to explore the map. In order to prevent redundant attacks, there is a hierarchy defining which saboteur attacks first.
- Repair: the repair strategy consists of finding the closest available repairer to help a disabled agent, after it the repairer and the damaged agent move close to each other. If there are no available repairers the disabled agent moves to the closest repairer. If there is another closest disabled agent to repair or another repairer, they cancel the process and start it again with the closest agent.
- Parry: if there is an opponent saboteur in the same vertex that our agents, the formula $1/N$ defines the parrying probability, where N is the number of ally agents in the same vertex. This way we can prevent all agents from parrying the same saboteur. Our agents do not parry if there are more or the same number of ally saboteurs and opponent saboteurs, since the opponent probably will attack our saboteurs first. If an agent chooses not to parry, then it leaves the vertex.
- Probe: the explorers always probe the closest unprobed vertex and they repeat it until all vertices are probed. To avoid explorers probing the same vertex, there is a hierarchy which defines the explorers who act first.
- Inspect: the inspectors always inspects near enemies, the aim of inspection is to identify enemy saboteurs and to check if the opponent is using a buying strategy.
- Recharge: the agents always check if they have enough energy before doing an action, if they do not have or it is less than 2 points, then they recharge. They also recharge when they do not have any action to do.
- Survey: the agents only survey if there is an unsurveyed near edge. The sentinels are the main agents responsible for doing survey, but other agents do it too if they do not have anything to do in the step.

Action	Repairer	Saboteur	Explorer	Sentinel	Inspector
buy		x(Hulk)			
attack		x			
repair	x				
parry	x			x	
probe			x		
inspect					x
recharge	x	x	x	x	x
goto	x	x	x	x	x
survey	x	x	x	x	x

Table 1: Implemented strategies by agent type.

Finally, there are strategies to expand the team zone and to stop expanding. The goal of the first one is to conquer more vertices in the same zone: when an agent is participating in a zone occupation and it can go to another vertex without breaking the zone, it will do it. The second strategy stops the agents from expanding when they have a high score and to wait for the opponents reaction.

4 Software Architecture

This section describes the technologies and frameworks that we used to develop our agents and how they are integrated. We used the EISMASSim framework [4] to communicate with the contest server, since the competition is built on Java MASSim platform and Java EISMASSim framework is distributed with the competition files. The programming language used to develop our agents is Jason (version 1.3.8) [5]. Its concept of BDI agents provided useful resources to build our agents, like plans and intentions, which allowed us to implement the strategies and to provide our agents with long-term goals. Another advantage of Jason is its interpreter that allow us to call Java methods, which simplifies the implementation of some algorithms and enables them to run faster. These methods are integrated with our Jason agents using internal actions. More specifically we implemented two algorithms as Java methods: Dijkstra algorithm to find the best path between vertices and Breadth-First Search algorithm to locate the best area in the graph.

A blackboard was used to share and build knowledge about the environment in the form of a graph. The process to update information in the graph has a high computational cost, lasting more than one step. Therefore, to avoid losing steps, the graph is updated and shared every three steps. The agent interaction is divided in two modes: agent-to-environment and agent-to-agent. In the first mode, in each step the EISMASSim framework receives an XML text from the server with the agents percepts, these percepts are then translated into Jason environment perceptions for our agents. This translation however does not happen when our team conquers the full map and the quantity of perception is so huge that the agents are not able to process them on time. In this case, perception is disabled and a default action (e.g. recharge) is sent back to the server. The actions of the agents are translated into text and sent to the server by EISMASSim. The Fig. 2 exemplifies how actions and percepts are exchanged. The agent-to-agent interaction uses Jason speech act based communication.

5 Results

We have tried to develop a system as complete as possible and we created several strategies for each system feature, like exploring, exploiting, buying, repairing, and attacking. Hence we developed many versions of the system, we exhaustively tested each one against the others to select the more efficient. We also tested our system during the contest test phase against the teams provided by the contest

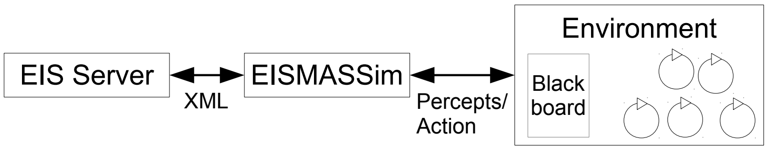
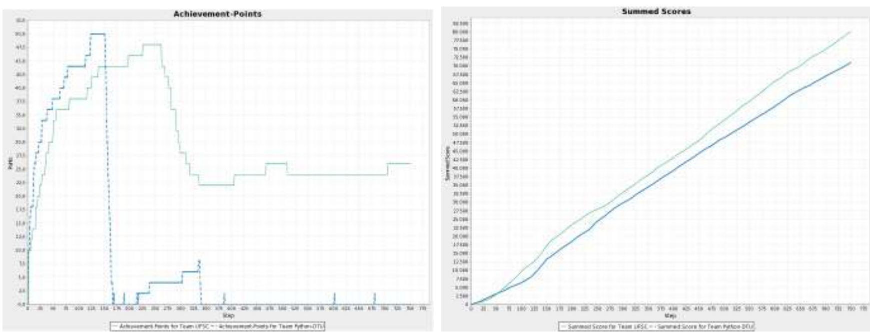


Fig. 2: Communication architecture.

organisation. This approach was our main advantage in the contest and one of the reasons we played eighteen matches against six different opponents and won seventeen. However our system has a worse performance when it confronts a passive system because it is not so offensive. If our agents are in a good map zone they do not bother about the opponent: they assume that the opponent is not in a good area. Also, our agents have no focus on defending a conquered zone and this explains the match we lost against Python-DTU during the contest.

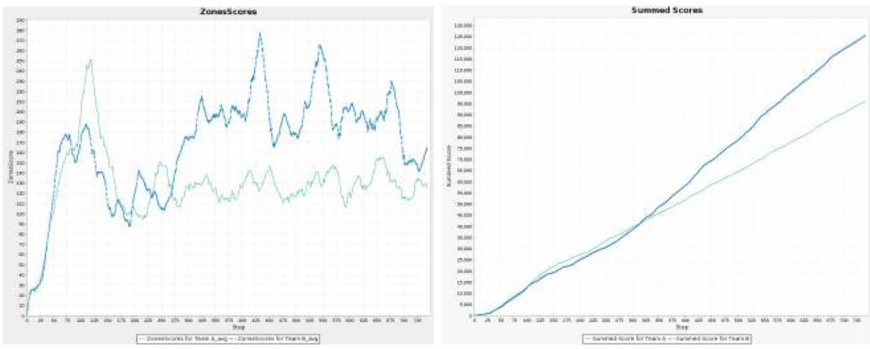
Two main strategies were responsible for the good performance of our system: the buying and exploitation strategies. The buying strategy was decisive because it forced our opponents to reinforce their agents spending a lot of their money. In a match against Python-DTU during the tests phase, for example, we conquered a small area but we won because we had more money. Fig. 3a shows the achievement points from this match. In the step 175 the Python-DTU (in blue) spent most of their money strengthening their agents and SMADAS (in green) spent only a part of its money. In the last 400 steps, from the step 350 to the 750, we had about 23 achievement points in each step, summing 9200 achievement points. In the end, this difference allowed us to win this match, as shown in Fig. 3b.



(a) (b)

Fig. 3: From the step 350, SMADAS-UFSC (in green) has more achievement points than Python-DTU (in blue) (a). This difference has decided the match for SMADAS-UFSC system (b).

Our exploitation strategy chooses two good zones in the map. It was efficient because usually the opponents are concerned about finding and conquering just one good zone. Thus while part of our agents are under attack in one of these zones, the other part are scoring in another zone. This strategy earns less points in each step, because our agents are divided in two smaller zones, but it has better results against an offensive opponent. Fig. 4 shows a comparison from our system performance using these two exploiting strategies. The system in green tries to conquer one single zone and the blue system looks for two zones. The blue system has fewer points at the beginning because it gets two smaller zones. However after some steps where the green system loses many points disputing a single zone, the blue system has one fixed zone scoring without any attack. This strategy was decisive in the match against the AiWYX system.



(a) (b)

Fig. 4: The green system tries to conquer one single zone and the blue system looks for two zones. The blue system finishes the match with a highest score because it keeps scoring in a zone without disputing it with opponents.

6 Conclusion

Participating in the contest was a worthy experience for all the team, we learned a lot about MAS developing and about the tools and languages we used. The contest result, where our team got the first place, is due both to the dedication on developing the strategies described in this papers and to the tools we used. For instance, the Jason programming language supports agent programming with abstract concepts like plans, beliefs, and goals which are suitable for the problem and very expressive. Different from previous participations in the contest where several bugs in Jason were discovered and fixed [9], we did not identify any bug in Jason this year, which shows the maturity of this language. Although we can evaluate the used tools positively in general, some features are still missing.

For example, it was very difficult to change, refactor, and debug the agents code since we have 5504 lines of Jason code and 20 agent instances running concurrently. The tools provided by Jason for debugging, like the sniffer and the mind inspector, are too specific and focused on the details. It is a hard task to identify a bug by looking at thousand of mind samples or message traces. High level abstractions and tools are required to help the debugging of complex MAS.

There is still a room for improvements in our system both in the strategies and the tools. Some of the improvements will be investigated in the authors' master and PhD thesis where proposals will be compared against the version of the system described in this paper. One particular drawback of the system is to be focused only on the agent aspect, all the code is "agent programming". More global aspects should be considered, for instance by organisation and interaction programming as first class abstractions. For that, new models and tools need to be developed.

For the current scenario of the contest, we would propose two improvements. (i) Inform opponent's score. It would allow participants to design strategies based on the current match result, rising more confrontations. (ii) Leave the graph less connected to increase the use of edges.

References

1. Tristan Behrens, Mehdi Dastani, Jürgen Dix, Michael Köster, and Peter Novák. The multi-agent programming contest from 20052010. *Annals of Mathematics and Artificial Intelligence*, 59:277–311, 2010.
2. Tristan Behrens, Michael Köster, Federico Schlesinger, Jürgen Dix, and Jomi Hübner. The multi-agent programming contest 2011: A résumé. In Louise Dennis, Olivier Boissier, and Rafael Bordini, editors, *Programming Multi-Agent Systems*, volume 7217 of *LNCS*, pages 155–172. Springer, 2012.
3. Tristan Behrens, Michael Köster, Federico Schlesinger, Jrgen Dix, and Jomi Hübner. The multi-agent programming contest 2011: A résumé. In *Programming Multi-Agent Systems*, volume 7217 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2012.
4. Tristan M. Behrens, Koen V. Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, April 2011.
5. Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
6. Dominic Carr, Sean Russell, Balazs Pete, G. O'Hare, and Rem Collier. Bogtrotters in space. In Louise Dennis, Olivier Boissier, and Rafael Bordini, editors, *Programming Multi-Agent Systems*, volume 7217 of *LNCS*, pages 197–207. Springer, 2012.
7. Marc Dekker, Pieter Hameete, Michiel Hegemans, Sebastiaan Leysen, Joris van den Oever, Jeff Smits, and Koen Hindriks. Hactarv2: An agent team strategy based on implicit coordination. In Louise Dennis, Olivier Boissier, and Rafael Bordini, editors, *Programming Multi-Agent Systems*, volume 7217 of *LNCS*, pages 173–184. Springer, 2012.
8. Mikko Ettienne, Steen Vester, and Jrgen Villadsen. Implementing a multi-agent system in python with an auction-based agreement approach. In Louise Dennis,

Olivier Boissier, and Rafael Bordini, editors, *Programming Multi-Agent Systems*, volume 7217 of *LNCS*, pages 185–196. Springer, 2012.

9. Jomi Fred Hübner and Rafael Heitor Bordini. Using agent- and organisation-oriented programming to develop a team of agents for a competitive game. *Annals of Mathematics and Artificial Intelligence*, 59(3-4):351–372, 2010.

Short Answers

A Introduction

1. What was the motivation to participate in the contest?

A: Evaluate the result of our master and PhD thesis.

2. What is the (brief) history of the team? (MAS course project, thesis evaluation, ...)

A: Our team was formed by members from the Multi-Agent Systems research group (called SMADAS) at Federal University of Santa Catarina (UFSC).

3. What is the name of your team?

A: Our team's name is SMADAS-UFSC.

4. How many developers and designers did you have? At what level of education are your team members?

A: Our team has six developers and everyone was involved with the system design. We have one PhD, one PhD student, three masters students and one undergraduate student.

5. From which field of research do you come from? Which work is related?

A: All team members work with Multi-Agent Systems and Artificial Intelligence.

B System Analysis and Design

1. Did you use a Multi-Agent programming languages? Please justify your answer.

A: We used the Jason language because all members are familiar with it.

2. If some Multi-Agent system methodology such as Prometheus, O-MaSE, or Tropos was used, how did you use it? If you did not, please justify.

A: We did not use any software engineering methodology because the problem seemed quite simple to solve and we had no experience with such methodologies.

3. Is the solution based on the centralisation of coordination/information on a specific agent? Conversely if you plan a decentralised solution, which strategy do you plan to use?

A: The system information is decentralised: each agent has all available information about the enemies and the graph. The coordination is centralised in a few cases, to solve some conflicting situations, like defining which agent should be repaired first or what is the best zone to exploit.

4. What is the communication strategy and how complex is it?

A: The agents use two mechanisms for communication: a blackboard and message exchanging. Some communication protocols are composed by one single message sent by an agents to others (e.g., when an enemy is inspected or when an agent report its action). Other protocols use more messages, for example when a damaged agent request a repair, nine messages are sent among the damaged agent and the repairers.

5. How are the following agent features considered/implemented: *autonomy*, *proactiveness*, *reactiveness*?

A: The agents are autonomous, reactive, and proactive. They have autonomy to decide how and when to execute their actions, they react to environment events and new messages, and are proactive while looking for a better vertex.

6. Is the team a truly **multi-**agent system or rather a centralised system in disguise?

A: The tasks of the team are decentralised among the agents which need to coordinate themselves to produce a coherent global behaviour.

7. How much time (person hours) have you invested (approximately) for implementing your team?

A: We expended about 500 hours developing the system.

8. Did you discuss the design and strategies of your agent team with other developers? To which extent did you test your agents playing with other teams?

A: We did not discuss the design or strategy with other teams before the contest.

C Software Architecture

1. Which programming language did you use to implement the Multi-Agent system?

A: The language used for programming our agents is Jason 1.3.8 [5].

2. How have you mapped the designed architecture (both Multi-Agent and individual agent architectures) to programming codes, i.e., how did you implement specific agent-oriented concepts and designed artifacts using the programming language?

A: The BDI concepts provided by the Jason language are the building blocks to develop our strategies.

3. Which development platforms and tools are used? How much time did you invest in learning those?

A: We used Eclipse platform with Jason 1.3.8 plug-in. These tools were known by all team members then we spend just few hours learning new features.

4. Which runtime platforms and tools (e.g. Jade, AgentScape, simply Java, . . .) are used? How much time did you invest in learning those?

A: We used EISMASSim framework to communicate with the environment and spent about 50 hours to learn it. For communication among the agents, we used Jason centralised infrastructure.

5. What features were missing in your language choice that would have facilitated your development task?

A: The Jason language has almost all features we needed to program our agents. However, for some algorithms, we preferred Java because it is faster.

6. Which algorithms are used/implemented?

A: We used two traditional algorithms for graphs: Dijkstra and breadth-first search.

7. How did you distribute the agents on several machines? And if you did not please justify why.

A: The agents were conceived to execute in the same machine to simplify blackboard programming, which uses shared memory. Future versions of the system will use distributed blackboards.

8. To which extent is the reasoning of your agents synchronized with the receive-percepts/send-action cycle?

A: The synchronisation with the environment is given by the reasoning cycle of Jason, where the first step includes the perception and the last the action.

9. What part of the development was most difficult/complex? What kind of problems have you found and how are they solved?

A: A blackboard has been used to share and build the knowledge about the environment. The process to update information in the graph has a high computational cost, lasting more than one step. Therefore, to avoid losing steps, the graph is updated and shared every three steps.

10. How many lines of code did you write for your software?

A: We have 7885 lines of code, 5504 written in Jason and 2381 written in Java.

D Strategies, Details and Statistics

1. What is the main strategy of your team?

A: We conceived our system strategy in two main phases: exploration, in which the explorers identify all vertices and nodes in the map and the best zones, and exploitation, where all agents try to conquest and defend these zones.

2. How does the overall team work together? (coordination, information sharing, ...)

A: Our agents exchange information to coordinate their activities.

3. How do your agents analyse the topology of the map? And how do they exploit their findings?

A: Some important information about the graph structure is shared and synchronized in the blackboard and it is used by the agents to move through the map. Despite this, agents do not use any information about topology to make the decisions.

4. How do your agents communicate with the server?

A: We use the EISMASSim framework to communicate with the server. External actions and usual perception are used by the agents to interact with the EISMASSim.

5. How do you implement the roles of the agents? Which strategies do the different roles implement?

A: The implemented strategies for each agent type is shown in Table 1.

6. How do you find good zones? How do you estimate the value of zones?

A: The system uses a modified version of the BFS algorithm to find the best zones in the map. It is run for all vertices, summing their values until some depth. The vertex with the highest sum represents where the best zone is (zone 1). After it, the algorithm tries to find the second best vertex to set the second best zone (zone 2).

7. How do you conquer zones? How do you defend zones if attacked? Do you attack zones?

A: With the zones defined, each agent is informed about the central vertex of its zone and how far they can travel inside it. The distance they can travel is the shortest path, in number of edges, between the central vertex and the target vertex. To defend these zones, the saboteurs attack all opponents inside the zone or in nearby vertices. The other agents stay in a vertex that has two neighbour vertices that belongs to our system. It is assumed that if the enemy zone is not near, the opponent likely has a small zone and then our agents do not try to attack it.

8. Can your agents change their behaviour during runtime? If so, what triggers the changes?

A: If the opponent does not have any buying strategy, the `Hulk` agent changes its behaviour and it stops buying upgrades. Besides it, in the start of the match the saboteurs attack the enemies, but after some steps they change their behaviour to attack the enemies.

9. What algorithm(s) do you use for agent path planning?

A: We used Dijkstra to path planning.

10. How do you make use of the buying-mechanism?

A: It was defined the minimum that the agents have to buy in order to make the enemy expend its money. In particular, we have *one* agent (named `Hulk`) that focus on buying and inducing all the opponents to also buy and spend their money.

11. How important are achievements for your overall strategy?

A: The achievement points are quite important since they accumulate each step. It is desirable to get the maximum of achievement points as soon as possible, but some achievements are hard to get. For example, our system does not surveys all edges and they do not inspect all opponents because it takes a long time and it is better to keep the agents in the best vertices, getting water wells score.

12. Do your agents have an explicit mental state?

A: The agents have their beliefs and use them to reason about their next action.

13. How do your agents communicate? And what do they communicate?

A: Our agents communicate indirectly by using the blackboard and directly by message exchanging.

14. How do you organize your agents? Do you use e.g. hierarchies? Is your organization implicit or explicit?

A: There is an explicit pre-defined hierarchy to prevent redundant actions: agents with higher priority decide before the others.

15. Is most of you agents' behavior emergent on an individual or team level?

A: In our strategy both individual and group behaviour are important. The individual behaviour is important when the agents are isolated in the map trying to get achievement points. The group behaviour is responsible for preventing redundant actions and conquering zones, for example.

16. If your agents perform some planning, how many steps do they plan ahead?

A: We do not use planning, all plans are previously programmed based on the strategies.

17. If you have a perceive-think-act cycle, how is it synchronized with the server?

A: We use the EISMAssim framework [4] to synchronize the agent actions to the server.

E Conclusion

1. What have you learned from the participation in the contest?

A: We learned a lot about MAS developing and about the tools and languages we used.

2. Which are the strong and weak points of the team?

A: Our strongest point is that we created several strategies for each system feature and tested them against each other to select the more efficient ones. Our weakness is that our system is not so offensive. Another problem is that our agents does not focus on defending their own zone.

3. How suitable was the chosen programming language, methodology, tools, and algorithms?

A: The Jason programming language was quite mature and suitable for the agent programming. However, we still need tools for programming and debugging at a higher level of abstraction.

4. What can be improved in the contest for next year?

A: We can improve our system both in the strategies and the tools. Our system is focused only on the agent aspect and more global aspects should be considered.

5. Why did your team perform as it did? Why did the other teams perform better/worse than yours?

A: Our system performed well because we focused on extensively testing all strategies.

6. Which other research fields might be interested in the Multi-Agent Programming Contest?

A: We think that some parts of the problem can be solved by optimisation techniques, which we plan to use in future versions of the systems.

7. How can the current scenario be optimized? How would those optimizations pay off?

A: We propose two improvements. (i) Inform opponent's score. It would allow participants to design strategies based on the current match result, rising more confrontations. (ii) Leave the graph less connected to increase the use of edges.

7 Python-DTU

Team Python-DTU from the Technical University of Denmark is a regular contender of the Multi-Agent Programming Contest. For this edition it registered 6 members. As team's name suggest, Python was the language of choice. The agents follow a decentralized approach, where coordination is achieved through distributed algorithms, e.g. for auction-based agreement.

Reimplementing a Multi-Agent System in Python

Jørgen Villadsen*, Andreas Schmidt Jensen, Mikko Berggren Ettienne, Steen Vester, Kenneth Balsiger Andersen, and Andreas Frøsig

Department of Informatics and Mathematical Modelling
Technical University of Denmark
Richard Petersens Plads, Building 321, DK-2800 Kongens Lyngby, Denmark

Abstract. We provide a brief description of our Python-DTU system, including the overall design, the tools and the algorithms that we used in the Multi-Agent Programming Contest 2012, where the scenario was called Agents on Mars like in 2011. Our solution is an improvement of our Python-DTU system from last year. Our team ended in second place after winning at least one match against every opponent and we only lost to the winner of the tournament. We briefly describe our experiments with the Moise organizational model. Finally we propose a few areas of improvement, both with regards to our system and to the contest.

1 Introduction

This paper documents our work with the Python-DTU team which participated in the Multi-Agent Programming Contest 2012 [7]. We also participated in the contest in 2009 and 2010 as the Jason-DTU team [4, 5], where we used the Jason platform [3], but this year we use just the programming language Python as we did in 2011 [6]. See `http://www.imm.dtu.dk/~jv/MAS` for an overview of our activities.

The scenario is based on the scenario from 2011 and has only been changed in a few ways. The most interesting change is the increase in number of agents from 10 to 20 agents per team.

Our focus for the 2012 version of the contest has been on reimplementing the system from 2011. Given that the scenario is very similar to that last year, we decided to look into ways of improving our system. We have been exploring the possibility of implementing an organization for the system using the Moise organizational model [1] as part of a two-student bachelor project.

The paper is organized as follows. In section 2 we discuss some of the ideas we have pursued. In section 3 we describe some of the facilities we have added in the improved system. Section 4 describes in detail our strategies and how the agents commit to goals. Finally, we conclude our work by discussing possible improvements of our system and the contest in section 5.

* Corresponding author: `jv@imm.dtu.dk`

2 System Analysis and Design

We chose to implement the system using Python as it is very fast and convenient to implement experimental systems in this language. Other useful features of Python are support of multiple programming paradigms, compact code and dynamic typing. We did not use any multi-agent programming languages because we wanted to have complete control of everything in the implementation. Last year we used Python 2 and we decided to upgrade to Python 3.

In order to make sure that our changes during the implementation phase improved our system, all new algorithms and architecture changes were tested against the older versions by comparing the data collected from the new statistics to see if the change made any differences.

2.1 Testing Moise

This year we wanted to try to implement some kind of organization for our system, so we made a substantial test implementation as part of a two-student bachelor project using the Moise organizational model [1]. We chose Moise because we have previous experience using it in combination with the Jason platform [3].

The Moise organizational model [1] is a formalism for organizational multi-agent systems where an organization is divided into three dimensions: structural, functional and deontic specification. The structural specification uses the concepts of *roles*, role *relations* and *groups* to build the individual, social and collective structural levels of an organization. Here, the roles an agent can enact are defined, and it is furthermore defined how roles are linked, e.g. by allowing agents enacting different roles to communicate. The collective level is specified using the notion of groups, in which it is determined which roles are allowed to be enacted and what links exists between agents both within internally in the group and with external agents. The functional specification specifies missions and plans using a so-called *social scheme* which is a goal decomposition tree that has as root the goal of that scheme. The responsibilities for each subgoal in a scheme are distributed in missions, which means that an agent choosing to commit to a mission effectively chooses to commit to the goals of that mission. The subgoals are created using the operators sequence, indicating that a goal is fulfilled when the sequence of subgoals are fulfilled, choice, in which a goal is fulfilled when a single subgoal is achieved, and parallelism, where all subgoals must be fulfilled, but no specific order is required. The deontic specification is the relation between the structural and functional specifications: it specifies on the individual level the permissions and obligations of a role on a mission. It makes it possible to specify that an agent enacting a certain role is obligated (or permitted) to commit to certain missions, and is therefore obligated (or permitted) to commit to the goals of that mission.

We follow the approach of S-Moise$^+$, which is an open-source implementation of an organizational middleware that follows the Moise-model [2]. Among other things it consists of a special agent, the *organizational manager*, which maintains consistency in the organization, i.e. by making sure that a single agent cannot enact two incompatible roles at the same time. This is done by letting the agents communicate with the manager when they want to join a group, enact a role or commit to a mission. If any such event is a violation of the organizational specification, the organizational manager will not allow it.

The plan trees and social schemes of Moise have a large potential, due to the fact that they will make sure that the right amount of agents will work together toward the best goal. We have chosen to only plan for a single subgoal for each agent, because of the very dynamic nature and the size of the map and number of agents. This makes the plans sufficiently small for the agents to coordinate themselves using direct communication, which makes the plan trees unnecessary.

It might be possible to split the agents into smaller groups to perform more coordinated plans, like finding the opponent's zones etc., but we did not have the time to try to implement groups. In the end we decided not to use Moise as we found that the benefits did not outweigh the needed effort to get the computation under the time limit, due to the quite large communication overhead of the organizational manager.

2.2 Agent behaviour

Our resulting system is a decentralized solution with a focus on time performance. The communication between the agents relies on shared data structures as this is a very fast way to communicate for the agents. The `Runner` class which coordinates communication is described in more detail in section 3.3.

Instead of letting the agents find goals based on their own knowledge alone they use the distributed knowledge of the entire team. This does add some communication which in some cases is unnecessary but in most cases the extra knowledge will produce better goals for the agents.

In each step each agent will find its preferred goals autonomously and assign each of them a *benefit* based on its own desires (i.e. the type of agent), how many steps are needed to reach the location and so on. In order to make sure that multiple agents will not commit to the same goal they communicate in order to find the most suitable agent for each goal. This is done using our auction-based agreement algorithm which will be discussed in more detail in section 4.3.

The agents in this contest are situated in an inaccessible environment which means that the world state can change without the agents noticing from step to step, e.g. if the opponent's agents move outside our agents' visibility range. Hence our agents should be very reactive to observable changes in the environment.

The agents are only proactive in a few situations. The most important one being the communication between a disabled agent and a repairer. They use their shared knowledge in order to decide which of the agents should take the last step and who should stay, so that they eventually are standing on the same vertex instead of simply switching positions. This is implemented by considering the current energy for each agent.

Some of our agents also attempt to be proactive by for example parrying if an opponent saboteur is on the same vertex. Furthermore, repairers will repair wounded agents since they are likely to be attacked again.

2.3 Random generation of the map

Last year all maps had one high-valued area, indicated by numbers on the vertices, as seen in figure 1. For this setting we developed an algorithm which places the agents in defensive positions inside the area in order to defend it. For more information we refer to the paper about our system from 2011 [6].

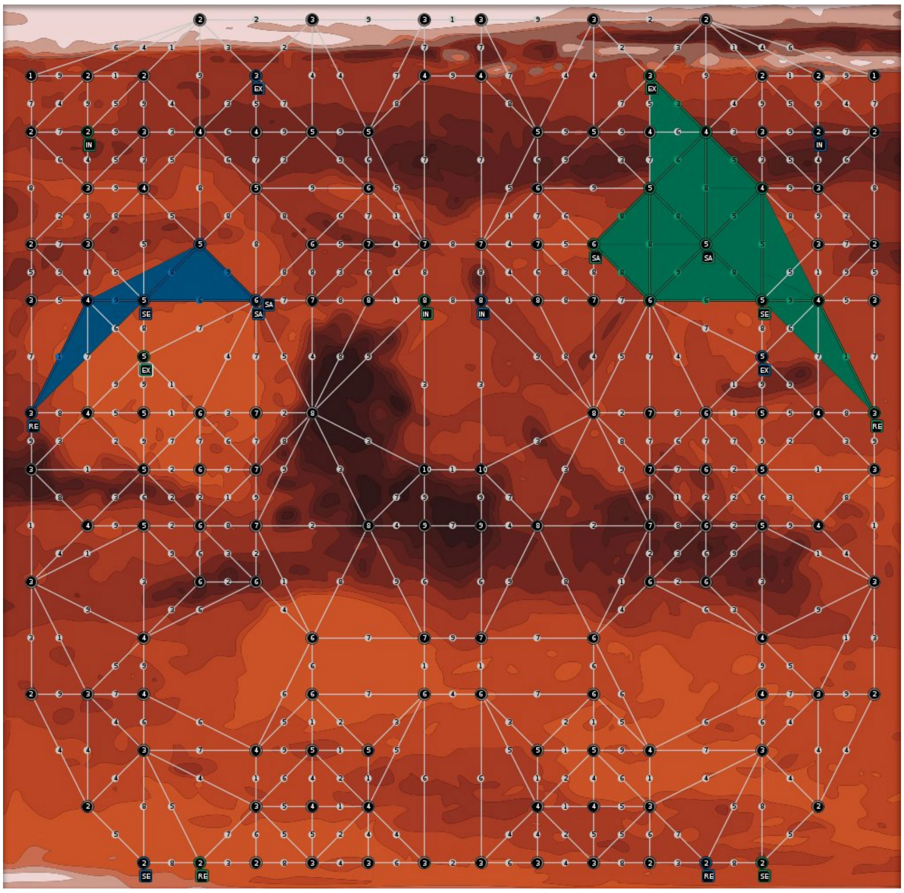


Fig. 1. An example of a map in the MAPC 2011.

This year the map generation algorithm has been updated to create more than one high-valued area. An example of this can be seen on figure 2, where the size of a vertex represents its value. In some cases this lead to situations where our agents would protect a single good area even though it would be better to make smaller groups and have control over several areas. Therefore our previous solution would only be effective in special cases, so we have implemented a new algorithm which takes multiple areas into consideration. The new solution is actually much simpler and it works well for both maps with multiple areas and maps with a single, high-valued area. In section 4.2 we describe the main properties of this algorithm.

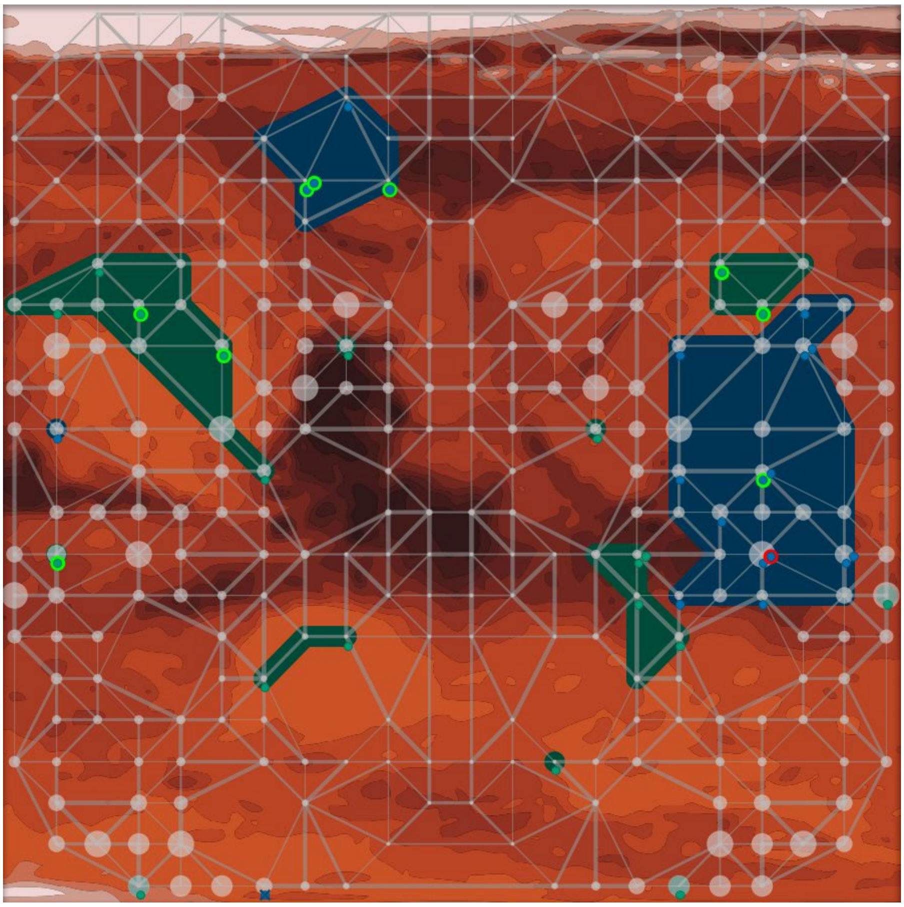


Fig. 2. An example of a map in the MAPC 2012.

3 Software Architecture

The software architecture, including the auction-based agreement approach, is thoroughly described in the paper about our system from 2011 [6] and will only be described briefly here. The rest of this section will describe a few minor facilities added this year.

5

3.1 Considerations

The competition is built on the Java MASSim-platform and EISMASSim framework which makes it easy to implement a system quickly without spending time on server communication and protocols. However, we did not utilize this framework but chose to implement our system in Python exclusively to have better control and complete knowledge about the implementation. Another solution based on EISMASSim, ActiveMQ and the Java implementation of Python, called Jython, was implemented as well. This solution was discarded due to performance issues. We also considered using a multi-agent framework such as Jason, but due to prior experiences, we thought that the benefits where outweighed by the increased complexity and thus chose to implement our own framework. We chose Python as we think it is in many ways superior with respect to development speed and succinctness compared to Java, C#, C++ and other languages that we have experience with. Furthermore Python supports multiple programming paradigms, including the functional, which has quite effective for this setting.

Last year we used a decentralized solution where the agents shared their percepts through a shared data structures but each kept their own copy of the graph representing the environment. The increase in the number of agents and the size of the maps for this year's competition, forced us to rethink and reimplement the percept sharing. To efficiently handle the increased amount of information, all agents share a single instance of the graph. To avoid deadlocks, percepts that lead to updates in this graph are handled with synchronized queues which allow safe exchange of data between multiple threads.

3.2 Testing using flags

A lot of testing was required for verifying that our system was improved compared to our previous system, so we needed an easy way to select which algorithms to use. In order to be able to run several instances of the program, we decided to create program arguments, or flags, for the system. In the beginning we had a configuration file in which we set flags. This was not a very practical way to do it as we had to have multiple configuration files in order to run more instances of the program. These flags make it possible to specify which algorithms the system should use. The help page for our multi-agent system where the different flags are described is shown below:

```
$ python ./bagent.py -h
usage: bagent.py [-h] [-b] [-d] [-a] [-w] [-l] [-v {0,1,2}] {a,b,Python-DTU}

positional arguments:
  {a,b,Python-DTU}      agent name prefix

optional arguments:
  -h, --help            show this help message and exit
  -b, --buy             make the agents shop for upgrades
  -d, --dummy           dummy agents
```

6

```
-a, --attack          do attack
-w, --weak_opp        attack EXP and INS in the start of the simulation
-l, --load_pickle     load vertices from pickled data
-v {0,1,2}, --verbosity {0,1,2}
```

The flags are used to start multiple instances of the system using different strategies. For example we can test whether it is better to use our buying strategy by starting the server and then start two instances of the system where the flag `-b` was passed to one of them. This was used to test whether it was beneficial to use our heuristics, but as we found that this was not the case we have removed them from the system.

3.3 Code structure and files

We briefly describe the main classes and files:

global_vars.py: We have all our global variables in this file. They are mainly used to make the implementation more dynamic and easier to maintain.

comm.py: This is the file where we have implemented the `Agent` class and the procedures used to communicate with the server. The `Communicator` class is implemented as processes such that all the agents can send and receive messages at the same time. The logic of the agents are implemented in the util.py and algorithms.py files.

bagent.py: This is where the main program is started and where the flags are parsed. It is also in this file that our `Runner` class is implemented. The `Runner` class starts and lets the agents do their calculations in a sequential fashion.

algorithms.py: Most interesting of our algorithms are implemented in this file, including:
- The *greedy zone control* which will be discussed in section 4.2.
- The *get goals algorithm* called by each agent. This algorithm is discussed in more detail in the paper about our system from 2011 [6].
- The *best-first search* used by each agent in order to find specialized goals according to their type.

util.py: We have implemented our graph representation of the map in this file. The file also includes a timer which was used to find bottlenecks in our code.

4 Strategies, Details and Statistics

In the competition each step of each achievement is exponentially harder to reach than the previous, thus our agents need a way to change their goals as the simulation progresses. We describe our strategy for getting achievements in section 4.1 and our zone control strategy in section 4.2. We describe how the agents decide what to do in section 4.3 and finally how communication works in section 4.4.

4.1 Getting achievements

In the beginning every agent will work towards achieving as many type specific goals as possible in a more or less disorganized fashion, e.g. the inspector will inspect every opponent it sees.

We do this to achieve as many achievements as possible as fast as possible. We tried implementing different heuristics to improve the first part of the strategy. We considered the following heuristics:

Survey heuristic: The agents always survey the vertex with the most outgoing edges if the steps needed to reach the vertex are the same (figure 3). The idea is to get survey achievements faster, but it turned out that even though we got the first few achievements faster, the last ones were achieved a lot later using this heuristic, so we did not use it.

Probe heuristic: The agents probe the vertex with the highest valued neighbours (figure 4). This worked very well in the scenario from 2011, but in the 2012 scenario it can be more beneficial to first find a lot of potentially high valued areas which can be probed later. This can be achieved using a random walk, which will reduce the time in each area increasing the chance that the agents might find more areas in less steps. We chose not to use the probe heuristic since a random walk was more successful.

Attack vulnerable opponents: This heuristic is only applied in the first 80 steps (a simulation has 750 steps). We prefer to attack agents that cannot parry, as this will get us more successful attacks. Furthermore, as added value this will also lead to fewer successful parries for the opponent. This turned out to give us a slight advantage in the beginning of the simulation, so we chose to use it.

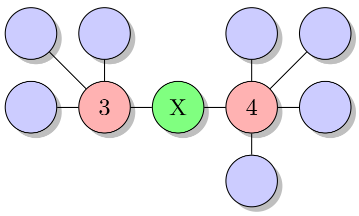


Fig. 3. Illustration of the heuristic values our agents would get trying to survey, standing on the green vertex. The vertex to the left has a heuristic value of 3 because it has three outgoing edges, whereas the one on the right has a slightly better heuristic value of 4.

After a certain number of steps the agents will proceed to the zone control part of our strategy. The sentinel is the only agent surveying after step 30. The explorers keep probing until step 150 and will probe in our target area for the next 50 steps to make sure we control as many vertices as possible. Afterwards

8

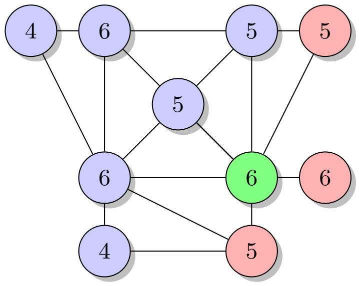


Fig. 4. Illustration of the heuristic values our agents would get trying to probe, standing on the green vertex where the blue ones are owned with the given value. The heuristic value of the red vertices are calculated by taking the mean of the known neighbouring vertices.

they will follow the zone control strategy. All other agents begin zone control after step 150.

4.2 Zone control

The zone control part of our strategy uses a very simple, but surprisingly effective, greedy algorithm. The algorithm works by first choosing the node with the highest value, and then by choosing a potential neighbour node. The potential value of choosing that node is then calculated as the value of the node plus the sum of all the neighbours which, according to the graph coloring algorithm [7], will be owned if the potential node is chosen. For each agent, the algorithm will choose the best node according to some parts of the graph coloring algorithm. If a vertex has not been probed the algorithm will use the value 1. This way we take some of the area coloring algorithm from the contest into consideration and as it is an inaccessible environment this is the best we could achieve.

This algorithm will to some extent choose the optimal area or several areas which are still fairly easy to maintain, even though our choices are limited by our (partial) knowledge of the map and the missing parts of the area coloring algorithm.

During the zone control part every type of agent has a specific job.

- *Repairers and saboteurs* do not directly participate in the zone control, instead they are trying to defend and maintain the zone.
- *Inspectors* keep inspecting from their given expand node, because the opponents might have bought something which we need to make a counter move against.
- *Explorers* will probe unprobed vertices within the target zone. When all vertices are probed they are assigned a vertex by the zone control strategy.
- The *sentinels* will stay on a vertex assigned by the zone control strategy and will parry if some of the opponent's saboteurs move to the sentinels position.

The last important change in the state of mind of the agents is that after step 150 the saboteurs start buying. They buy exactly enough extra health so that they will not get disabled by a single attack from an opponent saboteur that has not upgraded his strength. Furthermore we buy enough strength to disable any opponent saboteur in a single attack by buying strength for all our saboteurs every time we inspect the opponent saboteurs and find that it has more health than all other inspected saboteurs. This buying strategy is chosen in hope of dominating the map which will make it possible to gain control of the zone we want. The advantage is that we only try to out-buy in one specific field, thus we are unlikely to use all our achievement points. As this is a quite aggressive buying strategy we had to wait to step 150 to have enough achievement points to execute it.

4.3 Making decisions

The agents need a consistent way of figuring out what to do. We do this by letting every agent find the nearest goals according to their type. They do this by using a modified best-first search (BFS) which returns a set of goals. To make sure that every agent always has at least one goal the BFS returns as many goals as we have agents. This is a very agent-centered procedure meaning the agents simply commit to the goal with the highest benefit, instead of coordinating any bigger schemes. However, since the goals are more or less dependent on each other there is some implicit coordination. For example the repairers will often follow the saboteurs as these search for opponents and thus more often will share a vertex with an opponent saboteur and get disabled.

To decide which goal to pursue the agents use an auction algorithm. Every agent can bid on the goals they want to commit to and will eventually be assigned the one they are best suited for. This results in a good solution, which however might not be optimal. For further details we refer to the paper about our system from 2011 [6].

Even though our planner calculates a few turns ahead the agents recalculate every turn. We do this to adapt to newly discovered obstacles and facts, such as an opponent saboteur or the fact that the agent has been disabled. The agents will not end up walking back and forth as their previous goal will now be one step closer, thus the benefit of the goal has increased. If another goal becomes more valuable it means that it is a better goal than the one the agent was pursuing, thus changing the commitment makes sense, so we do not lose anything on recalculating each turn.

4.4 Communication

Communication and sharing of information is extremely important in any multi-agent system. In our system every percept received by the agents are stored in a shared data structure so that all agents have access to the complete distributed knowledge of the team at all times.

Actual communication in our system only happens when the agents are deciding what to do. When they are figuring out what to do the auction-based agreement algorithm is used on conflicting goals and thus two agents will never pursue the same goal.

5 Conclusion

In the process of reimplementing and improving the Python-DTU multi-agent system we have analysed the changes to the competition and used our findings to design and implement better algorithms for the increasingly complex tasks. We have considered imposing an explicit organization upon the agents, and for this purpuse we experimented with the Moise organizational model. While it had some advantages, such as the being able to ensure that the right amount of agents work together toward a certain goal using by use of roles and plan trees, we decided not to use Moise in the final version of our system, as its benefits did not outweigh the communication overhead caused by the organizational manager in the organizational middleware, S-moise^{+}.

All improvements to the algorithms are quite simple, but are nevertheless effective at reaching their goals. The simplicity and specialized approach is probably one of our strengths, as it makes it easy to implement special cases when certain improvements of the algorithms were necessary. Having aggressive saboteurs was also an advantage as it lead to the opponents being disabled often, which in turn gave us a larger zone score. Our greatest weakness was that our uncompromising attempt to have the strongest saboteurs could be countered by buying enough health on a single saboteur to make us use most of our achievement points for improving all of our saboteurs. This could lead to a large difference in step score gained from achievement points each step.

The many advanced programming constructs in Python, e.g. lambda functions, list comprehensions and filters made it possible to implement algorithms very efficiently.

One thing we have noticed during the competition is that it does not seem to pay off to buy anything other than health and strength. This meant that a lot of teams had more or less the same strategies. We think it could be interesting if many kinds of strategies could be sufficiently effective so that we might see the teams following different strategies. One idea could be to introduce ranged attacks which could be achievable through upgrades and should be limited by visibility range. This could allow for some other strategies, since the agents need to figure out where to hit the opponent a few steps in the future and how to avoid getting hit themselves. Furthermore, the teams will need to use their inspectors even more to find out whether or not to avoid possible ranged attacks from the opponent.

References

1. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Bossier. A Model for the Structural, Functional, and Deontic Specication of Organizations in Multiagent Systems. In Guilherme Bittencourt, and Geber Ramalho (Eds.): SBIA '02, LNCS 2507, 118-128, Springer 2002.
2. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Bossier. S-moise$^+$: A Middleware for Developing Organised Multi-Agent Systems. In Bossier et. al. (Eds.): COIN 2005, LNIA 3913, 64-78, Springer 2006.
3. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. Programming Multi-Agent Systems in AgentSpeak Using *Jason*. John Wiley & Sons, 2007.
4. Niklas Skamriis Boss, Andreas Schmidt Jensen, and Jørgen Villadsen. Building Multi-Agent Systems Using *Jason*. Annals of Mathematics and Artificial Intelligence, 59: 373-388, Springer 2010.
5. Steen Vester, Niklas Skamriis Boss, Andreas Schmidt Jensen, and Jørgen Villadsen. Improving Multi-Agent Systems Using *Jason*. Annals of Mathematics and Artificial Intelligence, 61: 297-307, Springer 2011.
6. Mikko Berggren Ettienne, Steen Vester, and Jørgen Villadsen. Implementing a Multi-Agent System in Python with an Auction-Based Agreement Approach. In Louise A. Dennis, Olivier Boissier, and Rafael H. Bordini (Eds.): ProMAS 2011, LNCS 7217, 185-196, Springer 2012.
7. Tristan Behrens, Michael Köster, Federico Schlesinger, Jürgen Dix, and Jomi Hübner. Multi-Agent Programming Contest — Scenario Description — 2012 Edition. Available online: `http://www.multiagentcontest.org/`, 2012.

Short Answers

A Introduction

1. What was the motivation to participate in the contest?

A: We find the contest very interesting for both research and teaching, cf. `http://www.imm.dtu.dk/~jv/MAS` for an overview of our activities.

2. What is the (brief) history of the team? (MAS course project, thesis evaluation, ...)

A: The team consists of both researchers/students from previous years and students taking a variant of the course 02295 Advanced Topics in Computer Science.

3. What is the name of your team?

A: The name of our team is Python-DTU where DTU is the short for the Technical University of Denmark. We started as the Jason-DTU team a few years ago.

4. How many developers and designers did you have? At what level of education are your team members?

A: We are 6 computer scientists: associate professor Jørgen Villadsen (PhD), Andreas Schmidt Jensen (PhD student), Mikko Berggren Ettienne (MSc student), Steen Vester (MSc student), Kenneth Balsiger Andersen (BSc student) and Andreas Frøsig (BSc student)

5. From which field of research do you come from? Which work is related?

A: Our field of research is AI with an emphasis on algorithms and logic (our section is called Algolog).

B System Analysis and Design

1. Did you use multi-agent programming languages? Please justify your answer.

A: No, we used plain Python.

2. If some multi-agent system methodology such as Prometheus, O-MaSE, or Tropos was used, how did you use it? If you did not, please justify.

A: We did not use any specific multi-agent system methodology, since we wanted to take a more direct and simple approach.

3. Is the solution based on the centralisation of coordination/information on a specific agent? Conversely if you plan a decentralised solution, which strategy do you plan to use?

A: We use an agent-centered system in which each agent first calculates their own goals, then they bid on the different goals and finally they use an auction algorithm to figure out which goals to pursue.

4. What is the communication strategy and how complex is it?

A: The communication strategy is very simple. The agents only talk to each other at the auction, and they share data using global data structures.

5. How are the following agent features considered/implemented: *autonomy*, *proactiveness*, *reactiveness*?

13

A: We have implemented the features as follows:
  - We have some degree of autonomy as each agent finds his own goals, but not complete autonomy as the goals are distributed using an auction algorithm.
  - The agents are proactive in certain situations. For instance, some of our agents will parry if a saboteur is located at the same node as the agent, and we also repair wounded agents since they will probably be attacked again later.
  - The agents are situated in an inaccessible environment, so we have chosen that our agents recalculate their goals at each turn and take new percepts into consideration, thus we have a high degree of reactiveness.

6. Is the team a truly **multi**-agent system or rather a centralised system in disguise?

A: The agents have direct access to a common dataset, so one could argue that it is a centralised system in disguise, but actually the agents themselves are communicating in order to figure out what to do. Furthermore, since all the agents find goals and decide which to pursue, the team is a true multi-agent system.

7. How much time (person hours) have you invested (approximately) for implementing your team?

A: We initially expected that we would have to invest approximately 200 man hours, but when the tournament started we had invested approximately 300 man hours.

8. Did you discuss the design and strategies of your agent team with other developers? To which extent did you test your agents playing with other teams?

A: We did not discuss strategies with other teams, but we used the test matches as an attempt to lure any good strategies.

C Software Architecture

1. Which programming language did you use to implement the multi-agent system?

A: Python 3.

2. How have you mapped the designed architecture (both multi-agent and individual agent architectures) to programming codes, i.e., how did you implement specific agent-oriented concepts and designed artifacts using the programming language?

A: Since we used plain Python there was no obvious relation between agent architecture in the design phase and the programming code. We used classes to represent agents.

3. Which development platforms and tools are used? How much time did you invest in learning those?

A: We have used *Mercurial* to version control our code and *gedit* and *vim* to implement the code. We have not used any time learning this as we had already used it before.

4. Which runtime platforms and tools (e.g. Jade, AgentScape, simply Java, ...) are used? How much time did you invest in learning those?

A: We have only used Python with a few special packages for argument parsing etc.

5. What features were missing in your language choice that would have facilitated your development task?

A: Some sort of precompiling for debugging purposes could have speeded up the implementation.

6. Which algorithms are used/implemented?

A: We have implemented the following algorithms:
   - Auction house
   - Best-first search
   - Greedy zone control area algorithm

7. How did you distribute the agents on several machines? And if you did not please justify why.

A: We used processes to allow the implementation to use all cores of the machine, which boosted the performance. We did not need several machines, since this was adequate for fulfilling the time limit criteria.

8. To which extent is the reasoning of your agents synchronized with the receive-percepts/send-action cycle?

A: The agents synchronize every time they have handled all percepts and as they start the auction house algorithm.

9. What part of the development was most difficult/complex? What kind of problems have you found and how are they solved?

A: Developing heuristics for different tasks proved to be quite hard. We used a lot of time on testing different implementations against each other to solve this.

10. How many lines of code did you write for your software?

A: We have written 1438 lines of code in Python including blank lines. This count includes various test functionalities.

D Strategies, Details and Statistics

1. What is the main strategy of your team?

A: The main strategy of our team is split into three parts. First we want to get as many random achievements as possible, then at step 30 we partially tries to take control of a high valued area, while explorers keep on probing more or less randomly. Finally, as we reach step 200 every agent is focused on expanding.

2. How does the overall team work together? (coordination, information sharing, ...)

A: All percepts are shared in a global datastructure and the goals are found individually by each agent, and if they conflict with any of the other agents goals then the auction algorithm is used.

3. How do your agents analyze the topology of the map? And how do they exploit their findings?

A: We simply save the map as a graph of vertices with a set of neighbouring vertices. The percepts are stored in a shared data structure including opponents specifications, types etc. Every time an agent perceives something from the environment, it will update the shared data structures.

4. How do your agents communicate with the server?

A: We communicate through a socket, where we handle strings of XML received from and sent to the server.

5. How do you implement the roles of the agents? Which strategies do the different roles implement?

A: Each agent has a tag telling what type it is, and the goals they create depend on this tag.

6. How do you find good zones? How do you estimate the value of zones?

A: Using a greedy algorithm considering common neighbours combined with the graph coloring algorithm used by the server.

7. How do you conquer zones? How do you defend zones if attacked? Do you attack zones?

A: We choose a zone to conquer judged by the sum of the values of the vertices in the zone. We have a greedy algorithm that takes most of the vertices we are going to own into account. Our saboteurs defend a zone by attacking opponents near it. If no opponents are near, the saboteurs will move towards a location where we last spotted an enemy in hope of finding some opponents and attack their zone.

8. Can your agents change their behavior during runtime? If so, what triggers the changes?

A: They change as described in our strategy which is a hard coded number of steps.

9. What algorithm(s) do you use for agent path planning?

A: We use a best-first search algorithm to decide which goals to commit to.

10. How do you make use of the buying-mechanism?

A: We make sure that our Saboteurs are strong enough to disable opponent saboteurs in a single attack by buying as much strength as they have health. Besides this we buy a single health point in case the opponents do not buy strength so that they cannot disable us in a single attack.

11. How important are achievements for your overall strategy?

A: Our entire strategy is build up around getting as many achievements as possible, thus it is extremely important.

12. Do your agents have an explicit mental state?

A: They are very active until they start controlling a zone at which point they go into a defensive state of mind.

13. How do your agents communicate? And what do they communicate?

A: Using a shared data structure in which they all percepts are shared with every agent.

14. How do you organize your agents? Do you use e.g. hierarchies? Is your organization implicit or explicit?

A: We do not organize them explicitly, but as saboteurs often gets disabled the repairers will follow and repair them. In the zone control part the agents will often organize themselves, but only as this gives more zone points.

15. Is most of you agents' behavior emergent on an individual or team level?

A: The agents decide autonomously which goals they want to pursue, and use an auction-based agreement algorithm to solve conflicting goals between agents on the team level.

16. If your agents perform some planning, how many steps do they plan ahead?

A: We only plan as far as the 4 nearest goals for each agent and we recalculate at each turn so we only plan a few steps ahead.

17. If you have a perceive-think-act cycle, how is it synchronized with the server?

A: For each turn we have a busy-wait until we have received the percepts and we send as soon as we have an action.

E Conclusion

1. What have you learned from the participation in the contest?

A: We have learned to implement a multi-agent system and all that it includes. We have been experimenting with a lot of algorithms and found that the simple ones are often better as they are fast which are useful in a fast changing world. We also learned that small tweaks and fine tuning in a multi-agent system really changes the behaviour and outcome a lot.

2. Which are the strong and weak points of the team?

A: Out team are very good at getting a lot of achievements very fast. We also defend our area pretty good. Our downside is that we almost only consider what we can see, this means that in some situations the opponent could have a large area and we would just let them. We ended up loosing because the opponents had a strategy to use as little achievement points as possible in order to make us use them all. In the long run this made us lose even though we had a better area. We never back down from a saboteur which in many cases would be good, as their saboteur would be occupied and we would not get disabled thus the repairer does not have to walk a long distance.

3. How suitable was the chosen programming language, methodology, tools, and algorithms?

A: It was a pretty good programming language though you had to get used to indent-based programming and no precompiling. It could have been interesting to try a multi-agent programming language to really model a system. In the end it seemed like our algorithms were quite effective even though they were very simple.

4. What can be improved in the contest for next year?

A: We could implement some of the missing strategies discussed above, for instance avoiding saboteurs or implementing a better attack strategy, e.g. prioritizing opponents based on current health (by inspection). We could also look into our buy strategies and adding an upper limit if needed.

5. Why did your team perform as it did? Why did the other teams perform better/worse than yours?

A: Our strategy worked pretty well which we anticipated as we played quite defensive. We lost because one of the opponents figured out our buying algorithm in the test-phase and implemented a clever counter strategy.

17

6. Which other research fields might be interested in the Multi-Agent Programming Contest?

A: Algorithms, logic, game theory and AI.

7. How can the current scenario be optimized? How would those optimization pay off?

A: It does not seem like it pays off to buy anything but health and strength. A reasonable change could be making visibility range more useful by using it as a parameter in the actions. For example this could mean that the saboteurs would get ranged attacks or inspectors could inspect all opponents inside the visibility range. This would 'force' a lot of new strategies to come into play, e.g. avoiding opponents artillery strikes etc.

8 TUB

Team TUB, TU Berlin, Germany, is another regular contender of the Multi-Agent Programming Contest, that presented for this edition as a single-developer team. The agents are developed in the JIAC platform (which won the contest several times in previous years).

Multi-Agent Programming Contest 2012 TUB Team Description

Axel Heßler, Thomas Konnerth, Pawel Napierala, Benjamin Wiemann

Technische Universität Berlin, Germany

Abstract. We describe our contribution to the Multi-Agent Programming Contest 2012, which has been developed by students and researchers of the DAI-Labor at TU Berlin, Germany, using the JIAC V agent framework and the agile JIAC methodology.

Introduction

Our team is called "TUB" and has participated consistently in the Multi-Agent Programming Contest [1–3] since 2007. Since our first participation, we consider the contest a very good opportunity to evaluate our platform and tools. The current team has been developed in the course "Multi Agent Contest"[1] by the following students: Pawel Napierala and Benjamin Wiemann, supervised by the following agent researchers: Thomas Konnerth and Axel Heßler (main contact). We have invested 640 hours approximately to create the contest version of our system and we are still not convinced that this version is competitive, although we have invested twice the time of last year's contribution.

System Analysis and Design

The methodology, which we have used during the course, borrows from the JIAC methodology, and can be described as bottom-up and agile methodology: we start with domain analysis, which is to build a first *ontology*: find the concepts of the domain, their structure and relationships with each other: agents, own team, opponents, nodes, edges, visited, probed, surveyed, weight.

As a second step the methodology says: make a *role model* and a *user interface (UI) prototype*. A role is specified by a number of capabilities or behaviours and the relationships with other roles. Identifying the roles was an easy task because they are easily collected from the scenario document. We then assigned simple and basic capabilities to the roles. As many of them were identical in each role, we created the generalised role of the *Mars invader*, which is a collection of the capabilities that all roles share, such as surveying, charging and moving. All other roles inherit from the invader role and add special capabilities such as probing, inspecting, and so on.

[1] Project 0435 L 774 at TU Berlin, Germany

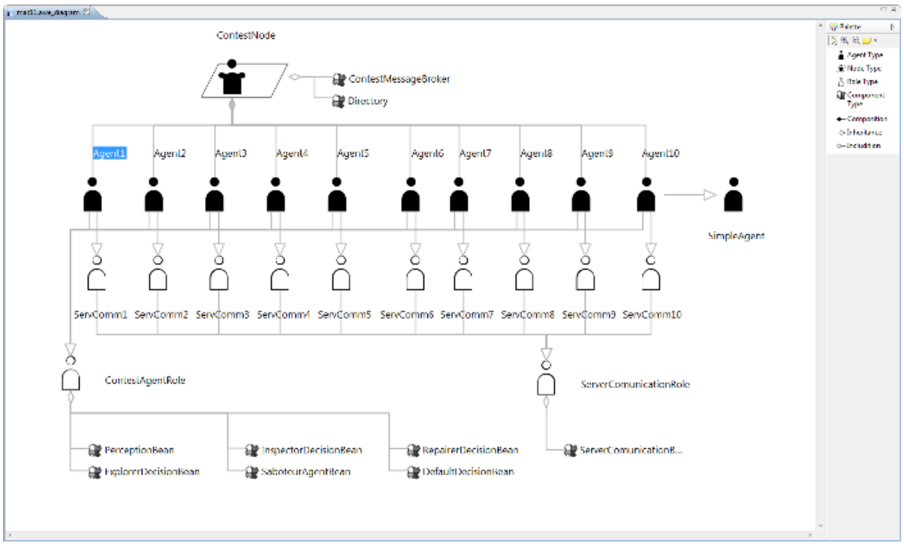


Fig. 1. TUB role model.

The role model was subject to many iterations. In Figure 1, an intermediate version of the role model is shown that is very close to the final role model. In principle, every contest agent in this role model could take every role (the *ContestAgentRole*), but during this contest the roles are static properties given by the contest server to every agent in the team. Common capabilities (*goto, survey, buy, recharge*) are implemented to the *DefaultDecisionBean* component. Special capabilities (*probe, inspect, attack, repair*) are implemented in the corresponding role specific component (e.g. *ExplorerDecisionBean* or *SaboteurDecisionBean*). Every agent instance has a specialization of the *ServerCommunicationBean* component with the credentials for authentication. Finally, every agent is instantiated once on the *ContestNode*, which provides the infrastructure for acquaintance and inter-agent communication. The role model has been generated with the help of the *AgentWorldEditor (AWE)*, which is part of the JIAC V tool suite *Toolipse*. The AWE generates configurations for all agents and agent nodes that are used by the JIAC V runtime at startup.

The UI prototype is a simple visualisation of the world graph. The problem here is that we could not find a solution to draw the graph in a repetable way during preparation. As a workaround we have patched the contest server to send the coordinates that project the graph to a grid as used by the monitor tool. The next step is *implementing* the simple and basic behaviours and then *evaluating* their function. After several iterations, when the basic actions can be reliably achieved by the agent, more complex capabilities are added, such as finding the most promising node to occupy or calculate the shortest or fastest path to an arbitrary node, and so on. The system can be distributed over several machines if available, without changing any line of code, even at runtime. This is one of the features of the JIAC agent framework [5] that is usually used for MAS administration and self-administration [6, 7]. However, we could not use this feature during the contest due to a lack of available hardware.

The agent system that runs our bots is mostly decentralised. As we use a component framework to build our agents, the functionalities for the roles are implemented within a dedicated component for each role. However, in order to simplify configuration, we decided to equip all of our agents with all components. The agents then decide based on the first message from the server, which role they take and keep that role for the remainder of the match. This way, it was very easy for us, to expand the team for the 2012 contest. All we had to do was to add a number of additional agents, and they took their roles automatically.

During the match, the basic cycle of our agents was triggered by the perceptions from the server. Whenever an agent receives a new perception, it starts the decision making cycle. In this cycle, the current state is evaluated and the agent decides what to do, based on its role. This decision is then forwarded to all other team members. Afterwards the agent waits for some time, in order to receive the decisions from the other team members. Depending on circumstances, this may lead to a reevaluation of the decision. Afterwards, the final decision is send to the server.

The only centralized or hierarchical part of the team organization is the zoning calculation. While this calculation can be performed by every agent, we have instead decided to let only one agent calculate the Zoning and propagate the results to all agents that participate in the zoning. This agent is selected among and by interested agents that want to know where to position in the zone, using a simple voting protocol. The result is then calculated by the selected agent and shared with the other interested agents.

Regarding the communication strategy of our team, we follow our 2007 − 2009 successful approach (e.g. in [4]) to distribute all perceptions and intentions among all other agents, where we could reach an appreciable enhancement of the team performance. In theory this approach should not scale very well as the number of perceptions and intentions sent around is $2n * (n - 1)$ per cycle. However, the JIAC V framework contains a messaging middleware that is capable of processing multicast messages for groups of agents. With this approach, each agent only needs to send one message that is then forwarded to all agents within the group. Thus the framework can handle the message very easily.

When it comes to coordination aspects we distinguish between explicit and implicit coordination. Implicit coordination can be achieved when the agents share their intentions. This notion of intention is often misunderstood when discussing the approach in the agent community. The intention in our case more often reflects a perform or achievement goal than the action that the agent has decided to execute. Taking the intentions of other agents into account, the actual agent can adopt the intention when it has a better utility or even dismiss its own decisions in case other agents will perform better. We have yet built only a few explicit coordination strategies into our agents, e.g. the collaboration between inspector and saboteur, or the unhealthy agent requesting the nearest repairer.

We have implemented general agent attributes such as autonomy, proactiveness and reactiveness as follows: JIAC V agents have their own thread of control and decide and act autonomously. We see the agents with low health level proactively seeking the repairer's help using a simple request, whereas probing or surveying has been implemented as a simple reactive behaviour: if the node is unprobed then probe.

Finally, our team was tested during the training matches that were organized by before the tournament in order to ensure that the agents run stable and can send their moves to the server within the allocated time.

Software Architecture

We have used the JIAC V agent framework to implement the contest MAS of our TUB team. For our agent researchers the contest is always an excellent reliability benchmarking of the framework, and also a test case for teaching principles of agent programming. We used a set of dedicated JIAC V plugins for the Eclipse IDE to create basic project structures and configurations. Then we added a number of components that were already available form last years contest, such as server communication and zone calculation functionalities. Finally, the biggest part of the work was invested in implementing and tuning the algorithms that control the actual actions of the agents. This was mostly done in Java, because the decisions and calculations are time critical, and we wanted to avoid the overhead from interpreting our declarative agent language.

As far as algorithms are concerned we experimented with Bellman-Ford and Dijkstra path finding algorithms for the movement calculations. However, the final team used a simple A-star algorithm, as other approaches proved to be to costly. They may become useful again, if we delegate the path finding to a dedicated agent that is not part of the team in future contests. Furthermore, the algorithm described in the contest scenario was used for the calculation of the zone scores.

Strategies, Details and Statistics

Every agents maintains its own world model (see Figure 2). Once the perception arrives, unknown vertices are added to the graph, which represents the physical world where the agents act in. Already known vertices are updated with the values from the perception. The perception is also shared with all other agents so that they can update their world model with information that is not visible to them.

The world model also contains a number of agent lists, i.e. team bots, enemy bots and special lists for interesting bots such as enemy's saboteurs and enemy's repairers (to either destroy or avoid them depending on the role of our agent).

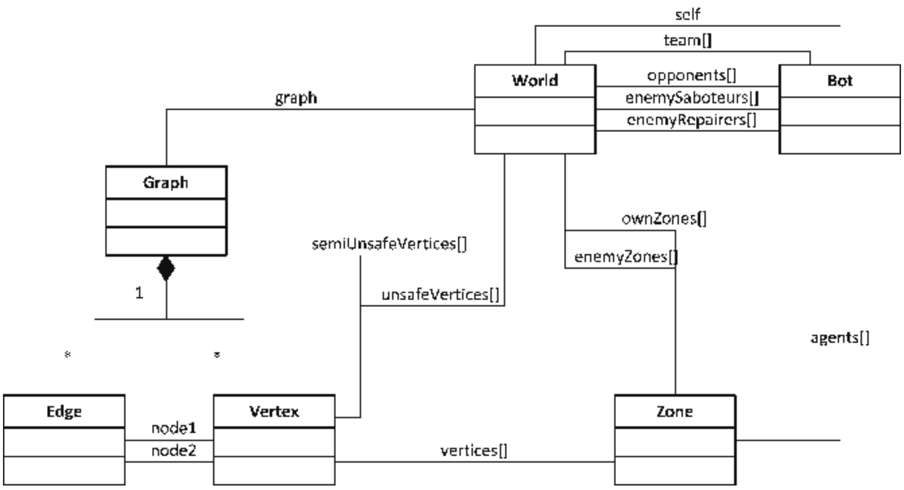


Fig. 2. Domain model.

Furthermore, the world model is updated by a number of Zones that support the decision process. The ownZones attribute is a list of zones that are held by the own team, enemyZones are the enemy's holdings as far as the own team can see them. A safe and a semisafe area are also calculated to give the bots a map of potential target vertices that are not reachable by enemy's saboteurs. The own saboteurs are more interested in the dangerous vertices because there are the first class targets to be destroyed: enemy's saboteurs.

The main strategy of our team is twofold: First, individual agents follows a simple, straightforward achievement collection strategy based on their roles. The behavior is as follows:

- **Explorer:** Explores the whole graph and if a node is not yet probed and the agent is situated on the unprobed node then it will try to probe until the probe action has been successfully achieved. If no probing is necessary, but there are unsurveyed edges connected to the current node, the agent will survey. Otherwise it will move to unprobed nodes.

- **Repairer:** If any agent is damaged and requires repairs, the repairer will move to that agent and repair it. Other repairers are repaired with a higher priority. If no repairs are necessary, the Repairer agents will participate in zoning.

- **Saboteur:** If any agents of the opposing team are detected, the Saboteur will try to catch them and attack them if possible. If no opposing agents are detected, or all agents are disabled, the Saboteur will participate in zoning. Furthermore, if enough achievement points are available, the saboteurs will buy increases for their attack-power and health attributes.

– **Sentinel:** Our strategy does not contain any special tasks for Sentinels. Therefore the Sentinels do always participate in the zoning.

– **Inspector:** The inspectors try to find agents of the opposing team and inspect them. This is mainly done to get the initial achievement points for the first ten inspects. When all opposing agents have been inspected, the Inspectors will stop inspecting and participate in zoning.

The second part of the strategy is an algorithm that we have called "zoning", i.e. two or more agents try to create and extend a zone in order to achieve the maximum zone score gain. The basic algorithm is rather simple. First of all, we determine which agents participate in the zoning. Then the current zone score is calculated. This calculation happens under the premise that all non-zoning moves are executed successfully but no other agent of our team or of the opposing team moves. In the next step, we calculate the possible permutations for this turn, i.e. what possible moves our zoning agents can make. As we only consider agents that participate in zoning and each agent can only make one step, the number of permutations is not extremely large and thus computable. For each such permutation, we calculate the zone score that results after the zoning agents have moved. Finally, we select the permutation with the best overall score. Thus our agents perform a local optimization for their zone score each turn.

While the initial implementation complexity and the computational cost of this zoning algorithm were both acceptable, the performance of our zoning algorithm within the contest was not very good. Obviously our agents miss the opportunity to identify *frontiers* that award a high number of points, unless these frontiers are already close to their current positions. This is likely the most important point for improvements on our strategy in future contests.

However, we should also mention that for the actual selection of both, targets for repairers and saboteurs, and for the zoning, we used greedy approaches. I.e. our agents simply take the closest target that maximizes utility. Unfortunately we could not find the time for the development of more elaborate strategies that achieve a global optimum – be it the discovery of a good border, or the detection of a high priority target for the saboteurs.

Conclusion

In summary, we are pleased withe the overall design and stability of our team. The agents worked flawlessly, did not break down, and submitted their actions in time to the server. However, the performance in terms of achieved scored is not what we had hoped for. Our strategy is probably too simple, and we need to improve the strategy for further contests. The most obvious points for this are the detection of globally optimal frontiers that our agents should occupy and a general improvement of situational awareness for all agents.

However, even though we think that our own performance in the contests could be improved, we wish to thank the organizers for the opportunity to test our framework and our agents. We think that the contests is a valuable addition to the multi agent community and hope that it will continue to be so for many years to come.

References

1. Tristan M. Behrens, Mehdi Dastani, Jürgen Dix, Michael Köster, and Peter Novák. The multi-agent programming contest from 2005-2010 - from gold collecting to herding cows. *Ann. Math. Artif. Intell.*, 59(3-4):277–311, 2010.
2. Tristan M. Behrens, Jürgen Dix, Jomi Hübner, and Michael Köster. Editorial. *Ann. Math. Artif. Intell.*, 61(4):257–260, 2011.
3. Tristan M. Behrens, Michael Köster, Federico Schlesinger, Jürgen Dix, and Jomi Fred Hübner. The multi-agent programming contest 2011: A résumé. In Louise A. Dennis, Olivier Boissier, and Rafael H. Bordini, editors, *ProMAS*, volume 7217 of *Lecture Notes in Computer Science*, pages 155–172. Springer, 2011.
4. Axel Hessler, Tobias Küster, Oliver Niemann, Aldin Sljivar, and Amir Matallaoui. Cows and Fences: JIAC V - AC09 Team Description. In Jürgen Dix, Michael Fisher, and Peter Novák, editors, *Proceedings of the 10th International Workshop on Computational Logic in Multi-Agent Systems 2009*, volume IfI-09-08 of *IfI Technical Report Series*. Clausthal University of Technology, 2009.
5. Benjamin Hirsch, Thomas Konnerth, and Axel Heßler. Merging agents and services — the JIAC agent platform. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 159–185. Springer, 2009.
6. Silvan Kaiser, Michael Burkhardt, and Jakob Tonn. Drag-and-drop migration: An example of mapping user actions to agent infrastructures. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen, editors, *The First International Workshop on Infrastructure and Tools for Multiagent Systems*, May 2010.
7. Alexander Thiele, Silvan Kaiser, Thomas Konnerth, and Benjamin Hirsch. MAMS service framework. In *SOCASE '09: Proceedings of The Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE) Workshop*, Lecture Notes in Computer Science (LNCS). Springer, 2009.

Short Answers

A Introduction

1. What was the motivation to participate in the contest?
A: Our motivation was to employ and evaluate the JIAC V framework. Furthermore we wanted to teach agent oriented principles to the students.
2. What is the (brief) history of the team? (MAS course project, thesis evaluation, ...)
A: The team was developed in a project course for bachelor students.
3. What is the name of your team?
A: The name of the team is "TUB".
4. How many developers and designers did you have? At what level of education are your team members?
A: We had 2 bachelor students working on the team who were supervised by to agent researchers.
5. From which field of research do you come from? Which work is related?
A: We come from the field of Agent oriented technology.

B System Analysis and Design

1. Did you use multi-agent programming languages? Please justify your answer.
A: We did not use the multi-agent programming language JADL that comes with the JIAC V framework, as we did not have enough time to train the bachelor students in this language. Furthermore, we have made the experience, that most work on the contest requires work on the algorithms, rather that work on "agent" problems such as coordination.
2. If some multi-agent system methodology such as Prometheus, O-MaSE, or Tropos was used, how did you use it? If you did not, please justify.
A: We used the JIAC methodology.
3. Is the solution based on the centralisation of coordination/information on a specific agent? Conversely if you plan a decentralised solution, which strategy do you plan to use?
A: The solution is based on sharing all knowledge between all agents, thus allowing them to come to identical solutions about the best course of action while still having decentralized decisions. However, for the zoning calculation, we use a centralized approach, as these calculations are rather expensive to compute.
4. What is the communication strategy and how complex is it?
A: The communication works in two steps. In the first step, all agents share their perceptions with all other agents via multicast messages. In the second step, once an agent has committed to a course of action, it informs all other agents about his actions. If actions collide (i.e. two agents try to probe the same node), one of the agents (usually the one that was slower to publish its intention) selects another action. Thus for each cycle we have one Multicast message (n-1 individual messages) and one normal message per agent, resulting in 2n*(n-1) messages for n agents.

5. How are the following agent features considered/implemented: *autonomy*, *proactiveness*, *reactiveness*?
A: Each agent decides autonomously about its course of action. It reacts to the actions of other agents and corrects its decisions if collisions occur.
6. Is the team a truly **multi**-agent system or rather a centralised system in disguise?
A: As the agents make their decisions autonomously and do not rely on a central instance for coordination, we regard it to be a true decentralized system.
7. How much time (person hours) have you invested (approximately) for implementing your team?
A: We invested approximately 640 hours of work.
8. Did you discuss the design and strategies of your agent team with other developers? To which extent did you test your agents playing with other teams?
A: We tested our team in the training matches that were organized before the actual tournament started.

C Software Architecture

1. Which programming language did you use to implement the multi-agent system?
A: Our agents were implemented with the JIAC V framework which is Java based.
2. How have you mapped the designed architecture (both multi-agent and individual agent architectures) to programming codes, i.e., how did you implement specific agent-oriented concepts and designed artifacts using the programming language?
A: For the communication and coordination of the agents, we used the appropriate JIAC V concepts. The individual functionalities for the roles of the agents were implemented in dedicated components for each role.
3. Which development platforms and tools are used? How much time did you invest in learning those?
A: Most of the work was done in Java with help of the Eclipse IDE. The Java implementations relied on the JIAC V framework. It took the bachelor students approximately two to three weeks to become familiar with this framework (they were already familiar with Java and Eclipse).
4. Which runtime platforms and tools (e.g. Jade, AgentScape, simply Java, ...) are used? How much time did you invest in learning those?
A: As a runtime platform we used JIAC V.
5. What features were missing in your language choice that would have facilitated your development task?
A: No features were missing.
6. Which algorithms are used/implemented?
A: We tried the Bellman-Ford and Dijkstra algorithms for path finding. The final solution however was based on the A-star algorithm, as the other algorithms proved to be to costly to be calculated by all agents.

7. How did you distribute the agents on several machines? And if you did not please justify why.

A: We did not distributed our agents across different machines, as our one server was more than capable to handle ten agents.

8. To which extent is the reasoning of your agents synchronized with the receive-percepts/send-action cycle?

A: The decision making was triggered by the receiving of perceptions and was finished before the timeout for each cycle.

9. What part of the development was most difficult/complex? What kind of problems have you found and how are they solved?

A: The most complex problem of the contest for us was the balancing of repair- and attack-actions. Furthermore the zoning algorithm for calculating the optimal placement of the agents proved to be rather complex when opposing agents were involved.

10. How many lines of code did you write for your software?

A: We did write approximately 8000 lines of code including comments.

D Strategies, Details and Statistics

1. What is the main strategy of your team?

A: Our agents try to optimize achievement and zoning points.

2. How does the overall team work together? (coordination, information sharing, ...)

A: The agents share their perceptions and intentions.

3. How do your agents analyze the topology of the map? And how do they exploit their findings?

A: The agents try to probe and survey all nodes and edges of the graph. The results are propagated to all agents.

4. How do your agents communicate with the server?

A: We have implemented our own connection to the server and our own parser for the perceptions.

5. How do you implement the roles of the agents? Which strategies do the different roles implement?

A: Each role is implemented in a dedicated component for the agents which is later configured into an agent. The explorers try to explore the whole graph as fast as possible. The repairers try to keep all teammates alive. The attackers try to disable the closest opposing agents. The inspectors make one pass to inspect all opposing agents in order to get the achievement points. All agents that have no role specific tasks left try to build a maximal zone.

6. How do you find good zones? How do you estimate the value of zones?

A: For our zoning algorithm, all agents that want to participate in a zone communicate this. Then the resulting zones for all possible moves of these agents are calculated and the best zone is selected, resulting in the agents to execute the appropriate moves. The zone score is calculated based on the known values of the nodes. Unknown nodes are valued with one point.

7. How do you conquer zones? How do you defend zones if attacked? Do you attack zones?

A: We do not explicitly attack or defend zones. Our attackers simply attack the closest opposing agents.

8. Can your agents change their behavior during runtime? If so, what triggers the changes?

A: Explorers contribute to zoning when they have finished the exploration. Repairers contribute to zoning if no teammate needs repairs. Attackers contribute to zoning if all opponents are disabled. Inspectors contribute to zoning of all opponents have been inspected. Repairers and Attackers may return to their default behavior if it is applicable again.

9. What algorithm(s) do you use for agent path planning?

A: The final team uses the A-star algorithm.

10. How do you make use of the buying-mechanism?

A: The attackers buy attack-power and health.

11. How important are achievements for your overall strategy?

A: We try to maximize the earned achievement points.

12. Do your agents have an explicit mental state?

A: Each agent has a central fact based (based on Linda like tuple space). The content of this fact base constitutes the mental state of the agent.

13. How do your agents communicate? And what do they communicate?

A: The agents communicate via messages that are equivalent to inform speechacts. They communicate their perceptions and intentions.

14. How do you organize your agents? Do you use e.g. hierarchies? Is your organization implicit or explicit?

A: The organization of our agents is decentralized and role-based.

15. Is most of you agents' behavior emergent on an individual or team level?

A: Individual behavior of the agents is programmed. Team based behavior is emergent.

16. If your agents perform some planning, how many steps do they plan ahead?

A: Our agents do not plan ahead.

17. If you have a perceive-think-act cycle, how is it synchronized with the server?

A: The cycle waits until the perceptions from the server arrive. The the agents try to calculate their actions as fast as possible and send them to the server as soon as all decissions are finished.

E Conclusion

1. What have you learned from the participation in the contest?

A: We underestimated the potential of aggressive attackers. Furthermore, algorithms with a high computational cost like the Dijkstra algorithm are applicable, but are too costly if all agents calculate them at the same time. This could be delegated to a specialized agent in future contests — the so called path finder.

2. Which are the strong and weak points of the team?

A: Our zoning algorithm worked but was only doing local optimization. This lead to a bad overall positioning for our team.

3. How suitable was the chosen programming language, methodology, tools, and algorithms?

A: The development of our team worked fine.

4. What can be improved in the contest for next year?

A: The organization of the contest was very good. The scenario was also good.

5. Why did your team perform as it did? Why did the other teams perform better/worse than yours?

A: Although we implemented two different teams during development for testing purposes, we underestimated the effectiveness of aggressive play. During the training matches we tried to improve our attackers, but were unable to make them truly competitive with the winning team.

6. Which other research fields might be interested in the Multi-Agent Programming Contest?

A: Game Theory, general Software Engineering, Coordination and Team organization, Self Organization

7. How can the current scenario be optimized? How would those optimization pay off?

A: The balance of achievement points and aggressive play styles can be modified in order to give the contest a different character. This is however not so much of an optimization. It rather is a way to keep the contest interesting.

9 LTI-USP

Team LTI-USP from University of Sao Paulo, Brazil had three developers. Agents were implemented using Jason, CArtAgO and Moise. There is one agent that determines the best strategy, but each agent has its own thread, with its own beliefs, desires and intentions. Agents broadcast new percepts, but communication load decreases over time.

LTI-USP Team : A JaCaMo based MAS for the MAPC 2012

Mariana Ramos Franco, Luciano Menasce Rosset, Jaime Simão Sichman

Laboratório de Técnicas Inteligentes (LTI)
Escola Politécnica (EP)
Universidade de São Paulo (USP)
{mafranko, luciano.rosset}@usp.br, jaime.sichman@poli.usp.br

Abstract. This paper describes the architecture and core ideas of the multi-agent system created by the LTI-USP team which participated in the 2012 edition of the Multi-Agent Programming Contest (MAPC 2012). This is the second year of the Agents on Mars scenario, in which the competitors must design a team of agents to find and occupy the best zones of a weighted graph. The team was developed using the *JaCaMo*[1] multi-agent framework and the main strategy was to divide the agents into three subgroups: two in charge of occupying the best zones in the map, and the other one in charge of sabotaging the opponents.

Keywords multi-agent system, multi-agent programming, JaCaMo, Jason, Cartago, Moise

1 Introduction

The Multi-Agent Programming Contest (MAPC) is held every year in an attempt to stimulate research in the field of programming Multi-Agent System (MAS) [2]. This is the second year of the Agent on Mars scenario, in which the competitors must design a team of 20 agents to explore and occupy the best zones of Mars, represented by a graph with valued vertices and weighted edges.

The LTI-USP, located at the University of São Paulo is one of the most relevant research groups in multi-agent systems in Brazil. The group participated in the 2010 edition of the MAPC [3] and the previous Cows and Cowboys scenario was used in the last two years of the Multi-Agent course held at the Department of Computer Engineering and Digital Systems of the University of São Paulo.

For this year's contest the LTI-USP team was formed by Mariana Ramos Franco (M.Sc. Student) and Luciano Menasce Rosset (Undergraduate Student), supervised by Prof. Jaime Simão Sichman (Professor). The M.Sc. student fully developed the multi-agent system, while the undergraduate student helped with the tests and gave some suggestions during the discussions about the adopted strategy.

The main motivation to participate in the contest was to test and to analyze the *JaCaMo*[1] framework, in order to identify the weak and strong aspects of the platform, and its performance limitations.

JaCaMo [1] is a platform for multi-agent programming which supports all levels of abstractions – agent, environment, and organisation – that are required for developing sophisticated multi-agent systems, by combining three separate technologies: *Jason*[2] [4], for programming autonomous agents; *CArtAgO*[3] [5], for programming environment artifacts; and *Moise*[4] [6], for programming multi-agent organisations.

2 System Analysis and Design

For the development of this project, as the main developer of the team had not a lot of previous experience with any multi-agent methodologies, we preferred to follow an iterative approach, which consisted in a cyclic process of prototyping, testing, analyzing, and refining. In the testing phase, we run our team against a previous version, and against the test teams provided in the contest software package. Next, after fixing the observed implementation issues and performance problems, we analyzed how effective the current strategy was and collected new ideas to improve it.

The adopted solution is based on the centralization of coordination, that is, one agent is responsible for determining which the best zone in the map is, and then conduct the other agents to occupy this zone. The choice of centralized coordination was made to allow the rapid development of the team, since our principal motivation was to focus on the *JaCaMo* platform performance issues and not on the coordination aspects.

In our team, each agent has its own view of the world, and they communicate with each other for the following purposes: (i) informing the others agents about the structure of the map; (ii) informing about the agent's or the opponent's position, role and status; (iii) asking for a repair; (iv) asking an agent to go to a determined vertex.

The agents' communication occurs via the speech acts provided by *Jason* and, to reduce the communication overhead, agents broadcast to all others only the new percepts, i.e., only percepts received from the contest server which produces an update on the agent's world model are broadcasted. For this reason, there is a strong exchange of information between the agents in the beginning of the match due to the broadcast of new percepts, specially those related to the map, such as vertices and edges. However, the communication overhead decreases as the agents' world model starts to be more complete.

The agent architecture is based on the BDI model. Each agent has its own beliefs, desires, intentions and control thread. The agents are autonomous to

[1] Available at http://jacamo.sourceforge.net/.
[2] Available at http://jason.sourceforge.net/.
[3] Available at http://cartago.sourceforge.net/.
[4] Available at http://moise.sourceforge.net/.

decide by themselves the next action to be performed, but in cooperation with each other, particularly with the coordinator agent. The agents are proactive in the sense that they pursue their selected intentions over time.

At each step, the agent decides which new plan will be executed to achieve a determined goal given only the state of the environment and the results of previous steps. There are no plans that last for more than one step and the plan's priority is determined by the order in which the plans were declared, i.e., the executed plan will be the first one to have its conditions satisfied. Some high priority plans can be considered reactive, such as the one which tells the agent to perform a recharge when running low on energy.

Approximately 300 man-hours were invested in the team development and, before the tournament, we participated in some test matches set by the organizers to ensure the stability of our team. Only during the competition did we discuss the design and strategies with the other participants.

3 Software Architecture

The prime requirement for this project was to create a MAS based on the *JaCaMo* multi-agent framework, making use of the *Moise* organisational artifacts. The architecture of the LTI-USP team is shown in Figure 1.

The agents are developed using the *Jason* MAS platform, which is a *Java*-based interpreter for an extended version of the *AgentSpeak* programming language for BDI agents [7]. Each agent is composed of plans, a belief base and its own world model. The plans are specified in *AgentSpeak* and the agent decides which plan will be executed according to its beliefs and the local view of the world.

The world model consists of a graph developed in *Java*, using simple data structures and classes. It captures every detail received from the MASSim contest server, such as: explored vertices and edges, opponents' position, disabled teammates, etc. At each step, the agent's world model is updated with the percepts received from the MASSim server, and with the information received from the other agents. The agent can access or change the state of its world model through the developed *Jason Internal Actions*. Some examples of internal actions are: `jia.closer_repairer(Pos)`, which returns to the agent the position of the closest repairer; and `jia.move_to_target(Pos,Target,NextPos)`, which tells the agent what the next movement to be performed is to achieve a desired position in the graph.

Some of the percepts received from the MASSim server are also stored in the agent's belief base, such as the agent's role, energy, position and team's money; allowing the agent to have a direct access to these information without a call for a *Jason Internal Action*. Percepts about vertices, edges and other agents were not stored in the belief base so as not to compromise the agent's performance, as it could be very expensive to update and to access the belief base with so much information. Moreover, since we wanted to update a belief when a new instance was inserted (instead of adding a second one), we decided to use the

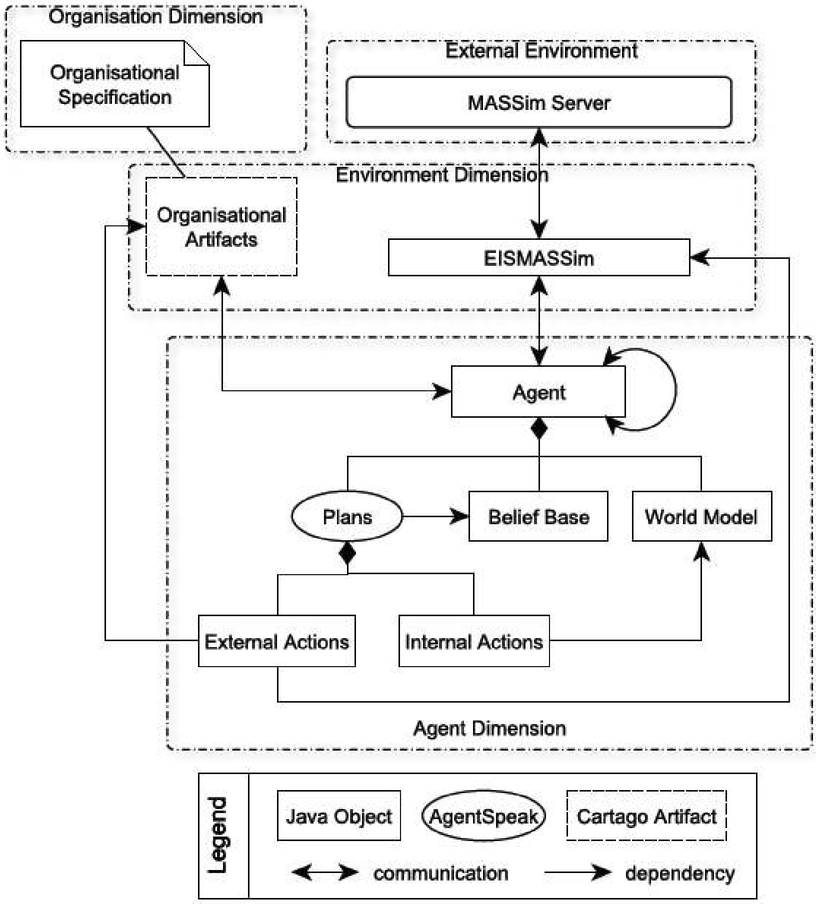


Fig. 1. LTI-USP Team Architecture

`IndexedBB` class provided in the *Jason* package, a customized version of the `DefaultBeliefBase` in which some beliefs are unique and indexed for faster access.

Agents communicate with the MASSim server through the EISMASSim environment-interface included in the contest software-package. EISMASSim is based on EIS[5] [8], which is a proposed standard for agent-environment interaction. It automatically establishes and maintains authenticated connections to the server and abstracts the communication between the MASSim server and the agents to simple Java-method-calls and call-backs. In order to use this interface, we extended the *JaCaMo* default agent architecture to perceive and to act not only on the *CArtAgO* artifacts, but also on the EIS environment as well.

[5] Available at http://sourceforge.net/projects/apleis/.

CArtAgO is a framework for environment programming based on the A&A meta-model [9]. In *CArtAgO*, the environment can be designed as a dynamic set of computational entities called artifacts, collected into workspaces, possibly distributed among various nodes of a network [1]. Each artifact represents a resource or a tool that agents can instantiate, share, use, and perceive at runtime. For this project, we did not create any new artifact; we only made use of the organisational artifacts provided in *Moise*.

Moise[6,10] is an organisational model for MAS based on three complementary dimensions: *structural*, *functional* and *normative*. The model enables a MAS designer to explicitly specify its organisational constraints, and it can be also used by the agents to reason about their organisation.

The *Moise structural specification* defines the roles played by the agents and the groups they take part in. As shown in Figure 2, we defined seven roles and four groups of agents for our team. Despite the five roles specified in the contest scenario (explorer, inspector, repairer, saboteur and sentinel), we created two other roles: `coordinator` and `martian`. The `coordinator` leads the other agents to occupy the best zones of Mars, and he does not communicate with the MASSim server. `Martian` is the default role adopted by the other agents in the beginning of the application, while they do not receive from the server the information about which role to play.

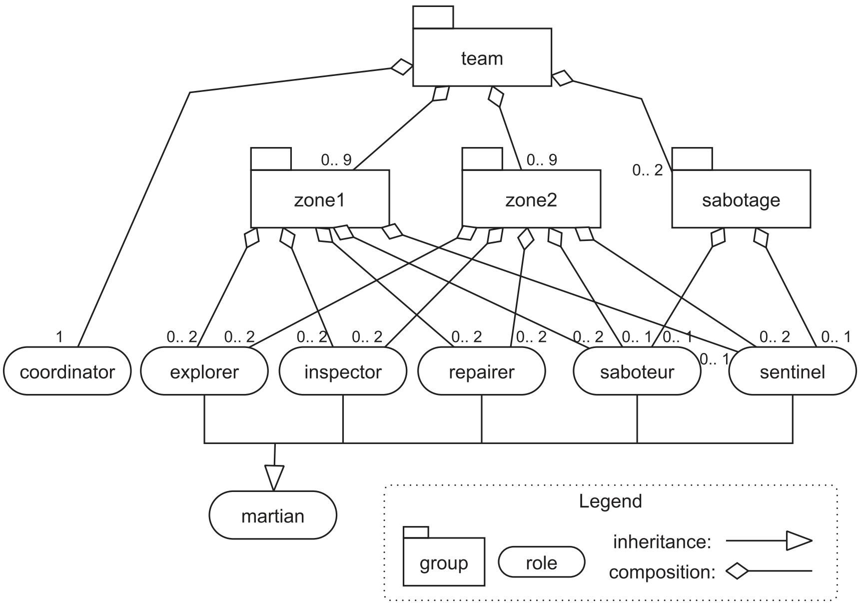


Fig. 2. *Moise structural specification* of the LTI-USP team.

The agents are divided into three subgroups: `zone1`, `zone2` and `sabotage`. The two first subgroups are responsible for finding and occupying the best zones in the map, while the sabotage subgroup must attack the opponent's best zone. Each subgroup has a global goal associated to it.

The *Moise functional specification* is composed of a set of schemes. Each scheme decomposes global goals into simpler goals and distributes them by assigning missions to the agents. It also specifies how these mission are related to each other, i.e., if they should be achieved concurrently or in a certain sequence. We have four schemes for our team, in which the global goals associated to them are: `coordinate`, `occupyZone1`, `occupyZone2` and `sabotage`. In Figure 3, these global goals are represented as the root of the trees that represent the schemes, and the leafs are goals which can be achieved by the agents. The label which appears just above a goal represents the mission that the agent must be committed to in order to achieve the related goal. The missions are described in the next section.

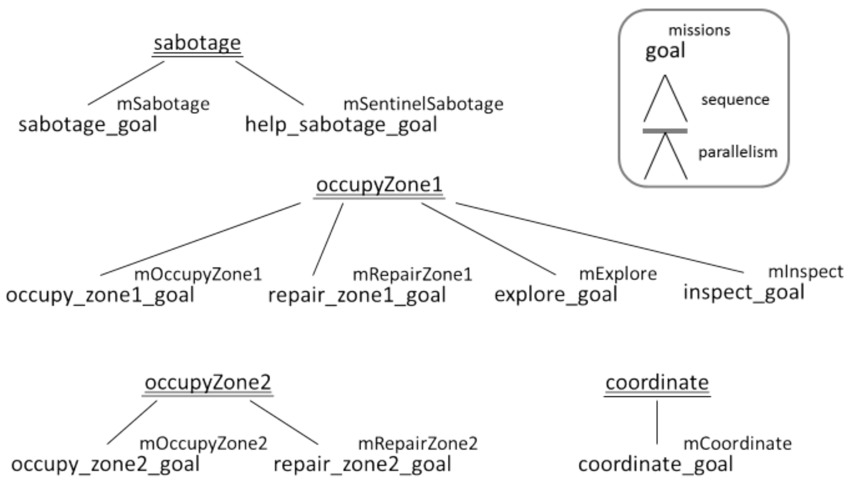


Fig. 3. *Moise functional specification* of the LTI-USP team.

The *Moise normative specification* links the *structural* and *functional* specifications by defining which role has the obligation or permission to commit to each mission. The normative specification for the LTI-USP team is shown in Table 1.

When the team starts, the `coordinator` agent creates the organisational artifacts and adopts the `coordinator` role, while the other 20 agents connect to the MASSim server and wait for the beginning of the simulation to known what role to play. Despite the fact that the agent's role is defined by their identification/credentials, we assumed in our team that the agent will only be aware of its role during the competition.

Table 1. *Moise normative specification* of the LTI-USP team.

Role	Mission	Deontic Relation
explorer	mExplore, mOccupyZone1	permission
explorer	mOccupyZone2	obligation
repairer	mRepairZone1, mRepairZone2	obligation
saboteur	mSabotage, mOccupyZone1, mOccupyZone2	obligation
sentinel	mSentinelSabotage, mOccupyZone1, mOccupyZone2	obligation
inspector	mInspect, mOccupyZone1	permission
inspector	mOccupyZone2	obligation
coordinator	mCoordinate	obligation

Once defined its role, the agent communicates with the `coordinator`, who tells him which group to join and the missions to commit. We decided to make the `coordinator` responsible for distributing the groups and missions among the other agents, because by doing so we thus eliminate the performance issues caused by two or more agents trying to adopt the same role in a group, or trying to commit to the same mission. For example, in the beginning of the simulation, as all agents perceive their roles at almost the same time, it is possible that all four saboteurs try to join the `sabotage` group but, as shown in Figure 2, only one saboteur is allowed in this group. In this case, three saboteurs will fail to join the `sabotage` group and will have to try to join another one. In the tests before the competition, we noticed that the organisational actions - such as `adoptRole` and `commitMission` - are very costly, and the number of retries performed by all the agents could be very high, causing some agents to loose some steps in the beginning of the simulation. Even eliminating this "concurrency" problem, we could observe during the competition that some agents still lost some steps until finally succeeding to commit to a mission on the organisation.

Our team consists of, approximately, 2000 lines of code in Java and 1200 lines in *AgentSpeak*, and the development was all carried on using the Eclipse IDE with the *Jason* plugin. The main developer was already familiar with both the development and the runtime platforms, i.e. ,the Eclipse IDE and the *JaCaMo* framework.

The agents were not distributed across several machines due to time constraints, but is our intention to work in the future on a distributed team, since this is supported by *JaCaMo*.

4 Strategies, Details and Statistics

The team strategy is a combination of the organisational strategy, the role dependent strategies and the coordination strategy.

4.1 Organisational strategy

As previously explained, one of the team's main strategy was to divide the agents into three subgroups: two of them in charge of occupying the best zones in the map, and the other one in charge of sabotaging the opponents. Below we describe the different missions related to each group.

- `occupyZone1`: Agents in this group have to occupy the best zone in the graph following the directions provided by the coordinator agent. In addition, one of the explorers works to probe the vertices of the graph to find the best zones to be occupied. In the exploration, he fixes the priority to the vertices belonging to the team's zone. Furthermore, one inspector has the mission of identifying the role of each agent in the opponent team. After the team has knowledge of all the opponents' role, the inspector joins the rest of the team for the mission of occupying the best zone in the graph.
- `occupyZone2`: All the agents on this group have the exclusive mission of occupying the second best zone in the graph, or to help the `zone1` group to form a larger area. Whether this group must join the other group or not is determined by the coordinator.
- `sabotage`: This group is formed by one saboteur and one sentinel. The saboteur's mission is to attack the opponent who occupies a good vertex; and the sentinel helps with the sabotage by moving inside the opponent's zone.

4.2 Role-dependent strategies

The explorers probe every vertex and survey all edges on its path, while inspectors can perform an inspect action whenever an opponent is in a neighbor vertex.

The priorities to run away, parry or attack, when an opponent is on the same vertex, are set to each agent's role. The saboteurs should always attack any opponent agent in the same vertex. It should first target the saboteurs, then repairers, and finally, the other opponents. The sentinels should always parry in the presence of an opponents saboteur. The repairers will decide between running away and parrying, in the presence of an opponent saboteur, depending, respectively, on if there is another teammate in the same vertex or not. Inspectors and explorers should always try to run away if an opponent saboteur is in the same vertex.

Repairing a disabled or damaged agent may break the structure of the area occupied. Having that in mind the repairers should stay put on their own vertices and wait for damaged and disabled agents to come for repairs. The disabled or damaged agent locates the closest repairer and heads to it, but if this repairer already has three or more agents to be repaired, the damaged agent will proceed to the second closest, and so on.

At each step, the team's score is computed by summing up the values of the zones and the current money. Thus the money obtained by the team has a big impact on its score. For this reason, we decided to limit the buy action, allowing the agents to purchase extension packs (such as `battery`, `shield` or `sabotageDevice`) only when a defined amount of money is reached. Furthermore, there is a specific buying strategy for each role. For example, the saboteurs can buy `sabotageDevices`, while the other agents cannot buy it.

4.3 Coordination strategy

The coordinator builds its local view of the world through the percepts broadcasted by the other agents. Whenever the world model is updated, it computes which the two best zones in the graph are. The best zone is obtained by calculating for each vertex the sum of its value with the value of all its direct and second degree neighbors. The vertex with the greatest sum of values is the center of the best zone. Zones with the sum of values below 10 are not considered in the calculation. The same computation is performed again to determine if there is a second best zone, but this time removing the vertices belonging to the first best zone from the analysis.

If two best zones are found, the coordinator agent will designate one first best zone for `zone1` group, and the second best zone for the `zone2` group. Otherwise, the same zone will be assigned for the two groups.

Given that the coordinator has assigned a zone for a group, all agents of the group are asked to occupy an empty vertex of the target zone. When all the agents are in the best zone, the coordinator starts to look to the neighbor vertices of the team's zone in which an agent can move, trying to increase the size of this zone.

5 Conclusion

Participating in the MAPC was a great opportunity to improve our knowledge of several multi-agent technologies by implementing a robust MAS through the *JaCaMo* framework. During the development, we had to deal with at least three different MAS technologies: *Jason*, *CArtAgO*, and *Moise*.

The team was built focusing to test the integration of these different MAS technologies, and not so much on the development of a better and decentralized strategy. Despite that, we believe that the team performed fairly well, finishing the competition in the fourth place.

Our greatest obstacle in the development of the team was to deal with the performance issues related to the use of the organisational artifacts. In a time limited context, as faced in this competition, the performance of a platform plays an important role, and we believe that these performance requirements may be a problem to the adoption of the *JaCaMo* in more real scenarios. Consequently, as future work we intend to perform a complete evaluation of the *JaCaMo* performance.

Besides these performance issues, the *JaCaMo* framework proved to be a very complete platform for the development of sophisticated multi-agent systems, by providing all the necessary features that we needed to developed our team.

Regarding possible extensions to the scenario, one idea is to change the score computation to consider only the sum of the zones values. In this way, the buying strategy will not impact directly the team score and it will be fairer to compare different strategies.

References

1. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming (2011)
2. Behrens, T., Köster, M., Schlesinger, F., Dix, J., Hübner, J.: The Multi-agent Programming Contest 2011: A Résumé. In Dennis, L., Boissier, O., Bordini, R., eds.: Programming Multi-Agent Systems. Volume 7217 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2012) 155–172
3. Gouveia, G., Pereira, R., Sichman, J.: The USP Farmers herding team. Annals of Mathematics and Artificial Intelligence **61** (2011) 369–383 10.1007/s10472-011-9238-x.
4. Bordini, R., Hübner, J., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. (2007)
5. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. Autonomous Agents and Multi-Agent Systems **23**(2) (June 2010) 158–192
6. Hübner, J.F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents. Autonomous Agents and Multi-Agent Systems **20**(3) (April 2009) 369–400
7. Rao, A.S.: Agentspeak(l): Bdi agents speak out in a logical computable language. In: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away: agents breaking away. MAAMAW '96, Secaucus, NJ, USA, Springer-Verlag New York, Inc. (1996) 42–55
8. Behrens, T.M., Dix, J., Hindriks, K.V.: The Environment Interface Standard for Agent-Oriented Programming - Platform Integration Guide and Interface Implementation Guide. Department of Informatics, Clausthal University of Technology, Technical Report **IfI-09-10** (2009)
9. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the a&a meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems **17**(3) (December 2008) 432–456
10. Hübner, J., Sichman, J., Boissier, O.: Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. International Journal of Agent-Oriented Software Engineering (2007) 1–27

Short Answers

A Introduction

1. What was the motivation to participate in the contest?

A: The main motivation to participate in the contest was to test and to analyze the JaCaMo framework, in order to identify the weak and strong aspects of the platform, and its performance limitations.

2. What is the (brief) history of the team? (MAS course project, thesis evaluation, ...)

A: The group participated in the 2010 edition of the MAPC and the previous Cows and Cowboys scenario was used in the last two years of the Multi-Agent course held at the Department of Computer Engineering and Digital Systems of the University of São Paulo.

3. What is the name of your team?

A: LTI-USP.

4. How many developers and designers did you have? At what level of education are your team members?

A: The LTI-USP team was formed by Mariana Ramos Franco (M.Sc. Student) and Luciano Menasce Rosset (Undergraduate Student), supervised by Jaime Simão Sichman (Professor).

5. From which field of research do you come from? Which work is related?

A: The LTI-USP, located at the University of São Paulo is one of the most relevant research groups in multi-agent systems in Brazil. In cooperation with other research groups in DAS/UFSC (Brazil) and ISCOD/LSTI/ENSMSE (France), our group is one of the responsibles for the development and maintenance of the Moise organisational model.

B System Analysis and Design

1. Did you use multi-agent programming languages? Please justify your answer.

A: We developed our team using the JaCaMo framework. JaCaMo is a platform for multi-agent programming which supports all levels of abstractions - agent, environment, and organisation - that are required for developing sophisticated multi-agent systems, by combining three separate technologies: Jason, for programming autonomous agents; CArtAgO, for programming environment artifacts; and Moise, for programming multi-agent organisations.

2. If some multi-agent system methodology such as Prometheus, O-MaSE, or Tropos was used, how did you use it? If you did not, please justify.

A: For the development of this project, as no member of the team had previous experience with any multi-agent methodologies, we preferred to follow an iterative approach, which consisted in a cyclic process of prototyping, testing, analyzing, and refining.

3. Is the solution based on the centralisation of coordination/information on a specific agent? Conversely if you plan a decentralised solution, which strategy do you plan to use?

A: The adopted solution is based on the centralization of coordination, that is, one agent is responsible for determining which the best zone in the map is, and then conduct the other agents to occupy this zone.

4. What is the communication strategy and how complex is it?

A: In our team, each agent has its own view of the world, and they communicate with each other for the following purposes: (i) informing the others agents about the structure of the map; (ii) informing about the agent's or the opponent's position, role and status; (iii) asking for repair; (iv) asking an agent to go to a determined vertex.
The agents' communication occurs via the speech acts provided by Jason and, to reduce the communication overhead, an agent broadcasts to all the other agents only the new percepts, i.e., only percepts received from the contest server which produces an update on the agent's world model are broadcasted. For this reason, there is a strong exchange of information between the agents in the beginning of the match due to the broadcast of new percepts, specially those related to the map, such as vertices and edges. However, the communication overhead decreases as the agents' world model starts to be more complete.

5. How are the following agent features considered/implemented: *autonomy*, *proactiveness*, *reactiveness*?

A: The agents are autonomous to decide by themselves the next action to be performed, but in cooperation with each other, particularly with the coordinator agent. The agents are proactive in the sense that they pursue their selected intention over time.
At each step, the agent decides which plan will be executed given only the state of the environment and the results of previous steps. The plan's priority is determined by the order in which the plans were declared, and the executed plan will be the first one to have its conditions satisfied. Some high priority plans can be considered reactive, such as the one which tells the agent to perform a recharge when running low on energy.

6. Is the team a truly **multi**-agent system or rather a centralised system in disguise?

A: Our system is a true multi-agent system. Each agent has its own beliefs, desires, intentions and control thread. Each agent decides by itself its next action.

7. How much time (person hours) have you invested (approximately) for implementing your team?

A: Approximately 300 man-hours were invested in the team development.

8. Did you discuss the design and strategies of your agent team with other developers? To which extent did you test your agents playing with other teams?

A: Only during the competition did we discuss the designs and strategies with the other participants.

C Software Architecture

1. Which programming language did you use to implement the multi-agent system?

A: Java and AgentSpeak.

2. How have you mapped the designed architecture (both multi-agent and individual agent architectures) to programming codes, i.e., how did you implement specific agent-oriented concepts and designed artifacts using the programming language?

A: The agents are developed using the Jason MAS platform, which is a Java-based interpreter for an extended version of the AgentSpeak programming language for BDI agents. Each agent is composed of plans, a belief base and its own world model. The plans are specified in AgentSpeak and the agent decides which plan will be executed according to its beliefs and the local view of the world. The world model consists of a graph developed in Java, using simple data structures and classes.

3. Which development platforms and tools are used? How much time did you invest in learning those?

A: All our code was written using the Eclipse IDE with the Jason plugin. All members were familiar with Eclipse.

4. Which runtime platforms and tools (e.g. Jade, AgentScape, simply Java, ...) are used? How much time did you invest in learning those?

A: We have used the JaCaMo platform to run our team. The main developer was already familiar with JaCaMo.

5. What features were missing in your language choice that would have facilitated your development task?

A: The JaCaMo framework provided all the necessary features that we needed to developed our team.

6. Which algorithms are used/implemented?

A: We used the breadth-first search algorithm to find the minimum path between two vertices in the graph.

7. How did you distribute the agents on several machines? And if you did not please justify why.

A: We did not distribute the agents in several machines due to time constraints, but is our intention to work after the tournament on a distributed team, since the JaCaMo framework facilitates this.

8. To which extent is the reasoning of your agents synchronized with the receive-percepts/send-action cycle?

A: At each step, the agent decides which action will be executed given only the state of the environment and the results of previous steps. So the agent reasoning is completely synchronized with the receive-percepts/send-action cycle.

9. What part of the development was most difficult/complex? What kind of problems have you found and how are they solved?

A: Our greatest obstacle in the development of the team was to deal with the performance issues related to the concurrent use of the organisational artifacts, and to solve it we decided to make the coordinator responsible for distributing the groups and missions among the other agents.

10. How many lines of code did you write for your software?

A: Approximately 2000 lines in Java and 1200 lines in AgentSpeak.

D Strategies, Details and Statistics

1. What is the main strategy of your team?

A: The main strategy was to divide the agents into three subgroups: two in charge of occupying the best zones in the map, and the other one in charge of sabotaging the opponents.

2. How does the overall team work together? (coordination, information sharing, ...)

A: One agent is responsible for determining which the best zone in the map is, and then conduct the other agents to occupy this zone. Each agent has its own world model, and only percepts received from the contest server which produces an update on the agent's world model are broadcasted.

3. How do your agents analyze the topology of the map? And how do they exploit their findings?

A: The explorers probe all unknown vertex and the results of map analysis are exploited to find the best zones to be occupied.

4. How do your agents communicate with the server?

A: Using the EISMASSim interface.

5. How do you implement the roles of the agents? Which strategies do the different roles implement?

A: The explorers probe and survey every vertex in their path, while inspectors can perform an inspect action whenever an opponent is in a neighbor vertex. The priorities to run away, parry or attack, when an opponent is on the same vertex, are set to each agent's role. The saboteurs should always attack any opponent agent in the same vertex. It should first target the saboteurs, then repairers, and finally, the other opponents. The sentinels should always parry in the presence of an opponents saboteur. The repairers will decide between running away and parrying, in the presence of an opponent saboteur, depending, respectively, on if there is another teammate in the same vertex or not. Inspectors and explorers should always try to run away if an opponent saboteur is in the same vertex.

Repairing a disabled or damaged agent may break the structure of the area occupied. Having that in mind the repairers should stay put on their own vertices and wait for damaged and disabled agents to come for repairs. The disabled or damaged agent locates the closest repairer and heads to it, but if this repairer already has three or more agents to be repaired, the damaged agent will proceed to the second closest, and so on.

6. How do you find good zones? How do you estimate the value of zones?

A: The best zone is obtained by calculating for each vertex the sum of its value with the value of all its direct and second degree neighbors. The vertex with the greatest sum of values is the center of the best zone. Zones with the sum of values below 10 are not considered in the calculation.

7. How do you conquer zones? How do you defend zones if attacked? Do you attack zones?

A: Given that the coordinator has assigned a zone for a group, all agents of the group are asked to occupy an empty vertex of the target zone. When all the agents are in the best zone, the coordinator starts to look to the neighbor vertices of the team's zone in which an agent can move, trying to increase the size of this zone.
We have not implemented a defense strategy, and because that the team did not perform well, trying to keep the conquest zone, when attacked.
We have developed a sabotage group to attack the opponents'zone.

8. Can your agents change their behavior during runtime? If so, what triggers the changes?

A: Yes. In the beginning, one inspector has the mission of identifying the role of each agent in the opponent team. After the team has knowledge of all the opponents' role, the inspector joins the rest of the team for the mission of occupying the best zone in the graph.

9. What algorithm(s) do you use for agent path planning?

A: Breadth-first search algorithm.

10. How do you make use of the buying-mechanism?

A: We decided to limit the buy action, allowing the agents to purchase extension packs (such as battery, shield or sabotageDevice) only when a defined amount of money is reached. Furthermore, there is a specific buying strategy for each role. For example, the saboteurs can buy sabotageDevices, while the other agents cannot buy it.

11. How important are achievements for your overall strategy?

A: The achievements were very important in the team score, because of that we limited the buy action.

12. Do your agents have an explicit mental state?

A: The agent's mental state consists of internal beliefs, desires, intentions, and plans.

13. How do your agents communicate? And what do they communicate?

A: The agents' communication occurs via the speech acts provided by Jason. They communicate with each other for the following purposes: (i) informing the others agents about the structure of the map; (ii) informing about the agent's or the opponent's position, role and status; (iii) asking for a repair; (iv) asking an agent to go to a determined vertex.

14. How do you organize your agents? Do you use e.g. hierarchies? Is your organization implicit or explicit?

A: We used the Moise model to explicitly specify the organizational constraints of our team. We organized our agents in three groups: two in charge of occupying the best zones in themap, and the other one in charge of sabotaging the opponents.

15. Is most of you agents' behavior emergent on an individual or team level?
A: Each agent acts individually and they are autonomous to decide by themselves the next action to be performed, but in cooperation with each other, particularly with the coordinator agent.
16. If your agents perform some planning, how many steps do they plan ahead?
A: Our agents do not plan ahead. Plans are recalculated in each step.
17. If you have a perceive-think-act cycle, how is it synchronized with the server?
A: After send an action, the agent stay in wait until receive new percepts from the server to start a new perceive-think-act cycle.

E Conclusion

1. What have you learned from the participation in the contest?
A: Participating in the MAPC was a great opportunity to improve our knowledge of several multi-agent technologies by implementing a robust MAS through the JaCaMo framework.
2. Which are the strong and weak points of the team?
A: We believe that the use of at least three different MAS technologies (Jason, CArtAgO, and Moise) is the strong point of our team. The use of a centralized coordination and the weak strategy in keeping the team's zones were the negative points.
3. How suitable was the chosen programming language, methodology, tools, and algorithms?
A: Besides the performance issues, the JaCaMo framework proved to be a very complete platform for the development of sophisticated multi-agent systems, by providing all the necessary features that we needed to developed our team.
4. What can be improved in the contest for next year?
A: Besides the test matches, the organization could leave a server running set up with a dummy team, so that the participants could test the connection and communication with the server at any time.
5. Why did your team perform as it did? Why did the other teams perform better/worse than yours?
A: The team was built focusing to test the integration of these different MAS technologies, and not so much on the development of a better and decentralized strategy.
6. Which other research fields might be interested in the Multi-Agent Programming Contest?
A: Algorithms, Game development, Game theory, AI, Robotics.
7. How can the current scenario be optimized? How would those optimization pay off?
A: Regarding possible extensions to the scenario, one idea is to change the score computation to consider only the sum of the zones values. In this way, the buying strategy will not impact directly the team score and it will be fairer to compare different strategies.

10 AiWYX

Team AiWYX was a single-developer team from Sun Yat-Sen Univerity, China. The agents were developed in C++, using no agent-specific technologies. The approach used is centralized, where one agent gets all the percepts from the other agents and makes the decisions for the whole team.

Conquering Large Zones by Exploiting Task Allocation and Graph-Theoretical Algorithms

Chengqian Li

Dept. of Computer Science,
Sun Yat-sen University
Guangzhou 510006, China
lichengq@mail2.sysu.edu.cn

Abstract. The Multi-Agent Programming Contest is to stimulate research in the area of multi-agent systems. In 2012, for the first time, a team from Sun Yat-sen University, Guangzhou, China, participated in the contest. The team is called AiWYX, and consists of a single member, who has just finished his undergraduate study. The system mainly exploits three strategies: strengthening action preconditions, task allocation optimization, and surrounding larger zones with shorter boundaries. With these strategies, our team is able to conquer large zones as early as possible, optimize collaboration, and ensure efficiency. The system was implemented in C++, and in this paper, we will introduce the design and architecture of AiWYX, and discuss the algorithms and implementations for these strategies.

Keywords: multi-agent system, distributing algorithm, task allocation optimization

1 Introduction

The Multi-Agent Programming Contest (MAPC) [1, 2] is held annually, in order for researchers to deepen the understanding about the cooperations and competitions among rational agents and also develop some powerful strategies in such environments. This year, for the first time, a team from Sun Yat-sen University, Guangzhou, China, participated in the contest. The team, called AiWYX, reached the fifth place in the contest. It consists of only one member: the author of this paper. I have just obtained my Bachelor degree and am now a PhD candidate. I am a member of the knowledge representation and reasoning group led by Professor Yongmei Liu. My motivation in participating in this contest was to gain experiences in designing multi-agent systems in order to facilitate my future research in this area. These years I am actively involved in the ACM International Collegiate Programming Contest (ICPC, see `http://icpc.baylor.edu`). Before this competition I had completed an undergraduate honors thesis on Squirrel World, which was proposed by Hector Levesque as an adaptation of the Monty Karel robot world written by Joseph Bergin and colleagues in Python (see `http://csis.pace.edu/~bergin/MontyKarel`). In Squirrel World, squirrels need to move around on a two-dimensional grid and gather acorns. Squirrels have both effectors (to do things in the world) and sensors (to gather information). Everything is known to the squirrels at the outset except for the locations of the acorns and some wall obstacles. The first squirrel or the first team of squirrels who gathers a certain number of acorns wins the game. I have adopted some of the strategies I developed for Squirrel World in the MAPC competition.

2 System Analysis and Design

I took part in the contest using the language C++, without using any multi-agent programming languages. There are two reasons for this. Firstly, my background is ACM/ICPC, so I am proficient in this language which is well-known for its efficiency and I did not program in Java which is not so efficient. Secondly, I did not have enough time to adapt myself to multi-agent programming languages.

We have exploited decentralization in implementing various strategies, however, the current implementation is restricted because we only deal with common knowledge [6]. When any agent's knowledge state is updated, other agents' knowledge state will be updated in precisely the same way, because of the assumption of common knowledge. Furthermore, we assume that communications between agents are perfect in this implementation. As to how to implement such strategies on a computer, we apply for a piece of main memory from the operating system, which stores the common knowledge. Hence, each agent has the same authority to access this memory space in order to communicate with other agents.

While such a team of agents is running in the competition, all agents have the goal that their team should reach a score higher than that of their rival. In any state of the world, any agent knows exactly what she should do next to achieve this goal and will start a new task immediately after completing one. In fact an agent can attain her goal by herself or through collaboration with others. Given a task, when there is only one agent intending to accomplish it, she will act by herself. However, if there are more, all such agents will collaborate to accomplish their task, that is, the task will be allocated to the agents in an optimal way. Moreover, the agents here are aggressive, that is, they keep exploring new areas of the world, never passively waiting for changes of the environment. Finally, in any state of the world, any agent is able to perform some action to achieve the goal, never lost in a dead-end.

To design and implement my system, I had spent about 250 hours. During this period, I did not discuss the design and strategies of my agent team with others, and I did not test my agents playing with other teams. I once tested my program by myself on a single computer, that is, I started a competition between two multi-agent teams, both of which were equipped with my own program. Both of them randomly selected a strategy at the beginning, thus they usually exploit different strategies, which helps me evaluate my team.

3 Software Architecture

I used C++ as the programming language, because it is so efficient and various mature data structures and algorithms are easy to code in C++. Each of the agents runs a separate program which is designed at four different levels, from the decision level to the physical level, as is described in Fig.1. Level 1 is the *decision*

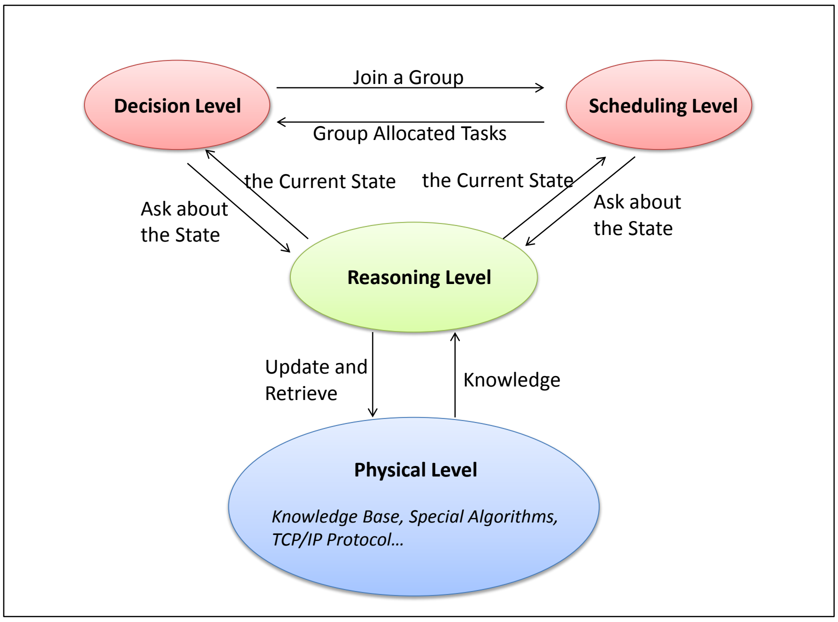


Fig. 1. Agent Model Diagram

level, which generates an action, or applies for joining a group, according to the current state. Such a group are to accomplish a task which cannot be handled by a single agent. For example, conquering a zone is a task, which cannot be accomplished by a single agent and need a group of them. If an action is generated, the agent will herself perform it, otherwise she will join a group for coordinating the task. If more than one agent applies for the same task, the first who applies will become a manager responsible for coordinating this group of agents in an optimal way. This manager agent produces the coordination in its program architecture at Level 2 (*scheduling level*), so Level 2 is responsible for scheduling and allocating tasks to each of them. Level 3 is responsible for manipulating and visiting the knowledge base (KB). When a percept is received by an agent, Level 3 will automatically update the knowledge base. On the other hand when being asked about the current state, it will retrieve specific information from the KB, so we call it *reasoning level*. Level 4 *(physical level)* contains various physical implementations, including KB, network communication (TCP/IP), and special algorithms such as string processing, Dijkstra algorithm [4], breadth-first search algorithm [3], minimum cost flow algorithm [8] and Hungarian algorithm [5, 7].

To develop my system, I used Gedit Text Editor in Linux system, together with the g++ compiler. With the flexible C++ programming language, I was able to implement all the features of my system quite efficiently, so no features are lost in my implementation. Although I did not distribute the agents on different machines when I participated in the contest, I am actually able to do so with minor adjustments, that is, to modify the number of user names and passwords in the initialization file. When agents are on the same machine, they communicate with each other by sharing main memory, otherwise they do so via the TCP/IP protocol. In the receive-percept period, if an agent receives a new percept, she will immediately perform reasoning to figure out her current state and update her knowledge base. In the meantime any other agent will neither think nor perform actions, until this update is completed. In the send-action period, each agent reasons on her knowledge base to figure out her state, then reacts according to our previously computed classification, before the action is sent to the server. Furthermore a multi-thread TCP/IP sender will send the action to the server. Note that our program is so efficient that any agent is always able to send her action to the server before the next percept arrives. The most difficult part of the whole development process was the optimization of team strategies. That is, how to classify all the possible states and how to compute the optimal action wrt each specific class. Roughly I solved these problems after a series of observations, experiments, and comparisons. In classification, I considered roles, injury, emergency, etc, and in the end, there were nearly 100 specific classes. For example, suppose there is an agent who knows that her role is a repairer and that she is neither injured nor in emergency, e.g., her energy value is too low. And if there is an injured teammate in her location, she will retrieve all these pieces of information from her knowledge base and consider all these factors to compute which specific state she is in. And she will finally generate a reaction to repair the injured. To design agents who react responsively and effectively, I classified all possible states, and for each class, compute the optimal response beforehand. In total, I wrote 10,000 lines of C++ code for my system.

4 Strategies, Details and Statistics

The main strategy of my agent system is that the whole team survey the edges and probe the nodes of the whole map to search for available areas, and then they try to occupy areas with higher values. If any minor event occurs, such as encountering enemies or getting injured, the agent will abort her task. No matter whether they are exploring a map or trying to occupy some area, the agents will cooperate in an optimal manner, avoiding redundant work, so that they are able to accomplish the task with the lowest cost.

4.1 Task Allocation

Given a set of tasks $w[1, \ldots, n]$ and the same number of agents $a[1, \ldots, n]$, an arrangement can be denoted as a matrix $Ar_{n\times n}$, where $Ar_{i,j} = 1$ if task w_i is allocated to agent a_j, otherwise, $Ar_{i,j} = 0$. Here our strategy is that each of the agents is allocated exactly one task, so in each of the rows and columns of Ar, there is exactly one '1'. We use matrix $C_{n\times n}$ to denote the costs (the number of steps or energy value an agent needs to accomplish a task), where $C_{i,j}$ denotes the cost needed for agent a_i to complete task w_j. Considering all possible arrangements, we hope to find a minimal value v such that each agent accomplishes her allocated task with costs no more than v. Let S be the set of possible arrangements such that the maximum cost is minimal, and let T be the elements in S such that the total cost is minimal. Algorithm 1, as shown in the following, returns one element in T. It involves two procedures, `Maximum_matching(Agents,Tasks,Edges)` based on Hungarian algorithm [5, 7], and `Min_cost_flow(source, sink, Agents, Tasks, Edges, Cost)` which is just the one in [8]. Table 1 shows the test results of Algorithm 1. Each row shows a specific type of 10 experiments, where the first three columns show the number of agents, tasks and edges respectively. The fourth here shows the average number of edges whose value is not greater than v . The last column shows the average running time.

We allocate each agent a unique task so that repetitive work is avoided so that we are able to minimize the total cost. As mentioned earlier, when any agent receives a new percept, any other agent will not perform any actions until this percept is passed to all of them. This ensures that all agents share a synchronized knowledge base based on the presumption of common knowledge. Each time an agent arrives at an unexplored location, she surveys this location, obtaining all adjacent nodes and the costs of respective edges. In this way, all locations explored form a connected component and the agents know all information about this subgraph, including the shortest path between any two nodes in this component. Their strategy now is to move to those nodes on the boundary, survey them and then continue this process again and again. This will accelerate the process of searching for more valuable areas. To avoid the case that two agents move to the same location to survey, and to minimize the total cost, we use Algorithm 1 to inform each agent where they should go. To communicate with the server, we use a multi-threaded TCP/IP protocol.

I have designed a particular strategy for each of the five roles in the game. When an agent realizes that she is acting in a certain role, say, repairer, she will follow the respective strategy. Only explorers will accept the mission of exploring the map and probing the value of the newly encountered node. After finishing exploring, they will join a group to conquer a large zone. Here sentinels will join a group to survey all the edges and after that, will join another group to conquer a large zone just as what the explorers do. If some enemies are found and the team does not know their roles and specific states, inspectors will join a group to inspect those enemies, to collect such information. Otherwise, they will join a group to survey all the edges and then join another group to conquer a large

```
input  : Agents, Tasks, Cost
output: arrangements
// binary search
low ← min_{x∈Agents,y∈Tasks} Cost(x,y) − 1
high  ← max_{x∈agents,y∈Tasks} Cost(x,y)
while low+1 ≤ high do
    mid ← ⌊(low+high)/2⌋
    Edges ← {(x,y)|x ∈ Agents, y ∈ Tasks, Cost(x,y) ≤ mid}
    if  Maximum_matching(Agents,Tasks,Edges) == |Tasks| then
    // Hungarian algorithm
        high ← mid
    else
        low ← mid
    end
end
Edge ← {(x,y)|x ∈ Agents, y ∈ Tasks, Cost(x,y) ≤ high}
Edges ← Edges ∪ {(source,x)|x ∈ Agents} ∪ {(x,sink)|x ∈ Tasks}
Cost(source,x) ← 0// for all x  ∈ Agents
Cost(x,sink) ← 0// for all x  ∈ Tasks
Min_cost_flow(source, sink, Agents, Tasks, Edges, Cost)
for (x,y) ∈ Edges do
    if flow(x,y)==1 then
        // flow is defined in Min_cost_flow
        arrangements(x) ← y
    end
end
return  arrangements
```

Algorithm 1: Min_max_cost_tast_allocation(agents,works,cost)

Table 1. Experimental results for Algorithm 1 (value of edge < 10000)

Agents	Tasks	Edges	Remaining edges	Time
20	300	3000	62	0.0039s
20	1000	10000	65	0.0088s
100	1000	50000	485	0.0571s
200	1000	100000	1243	0.1634s
500	1000	250000	3641	0.8317s
1000	1000	500000	8153	4.5360s

zone just as what sentinels do. If some injured teammates are found, repairers will run to them and repair them, otherwise, they will join a group to survey the edges and then another group to conquer a large zone in the same way. If some enemies are discovered, saboteurs will go to front line and fight with those enemies, otherwise, they just do what sentinels do in the same occasion.

4.2 Expanding Zones

Expanding a boundary node B, means adding all adjacent nodes of B, which are not occupied by the enemies, into the current zone. The agents first choose each node which is not occupied by enemies as a point zone and then repeat the following: find the boundary node P such that after expanding P the boundary increases the least (possibly by a negative number), and then, expand it. During the expanding process, we maintain the best zone found in the past, with the highest value. We say a zone B_1 is superior to another one B_2 if B_1 is more valuable than B_2. In details, we have the following Algorithm 2. The complexity of this algorithm is $O(N^2M)$, where N is the number of nodes and M is the number of edges in the graph. This is because the zone will only be expanded at most N times and at each expanding, at most M edges will be traversed. Table 2 shows the test results of Algorithm 2, where the first two columns show the number of nodes and edges respectively, and the third column shows the number of enemies, that is, the number of nodes occupied by enemies. The last two columns show the average running time for centralized and distributed algorithms respectively. Notice that in each type of experiments, the sum of the running time over all the machines for the distributed algorithm, is several times greater than the running time of the centralized algorithm, because in the centralized algorithm, we apply the hashing technique to examine whether a zone had already been computed before.

Note that Algorithm 2 can be made distributed, in that the expanding procedure can simultaneously begin at any number of nodes on the map. In particular, if we have as many machines as the nodes, we allocate each machine a unique node and instruct it to run a separate expanding procedure with that node.

4.3 Strategy Details

Formally below is the evaluation function for estimating the value of a zone:

$$\texttt{value}_{\texttt{Zone}} = \sum_{\texttt{i}\in\texttt{Zone}} \texttt{value}_{\texttt{i}}. \tag{1}$$

Our agents will calculate the most promising zone with Algorithm 2 and then move to the boundary of that zone and conquer it. Among them, the saboteurs always attack the nearest agent of the rival, so that this group of agents always attack the nearest area occupied by the enemies. If they are attacked by the enemies, they will recompute a new area not occupied by the enemies, and then move there. All agents are equipped with exactly the same program,

```
input  : Nodes, Edges, value, Enemy_Nodes
output: Best_Zone
for x ∈ Nodes do
  | Neighbor_x ← {y|(x,y) ∈ Edges}
end
Can_Not_Expand ← {x|x ∈ Enemy_Nodes or Enemy_Nodes ∩ Neighbor_x ≠ ∅}
// cannot be Expanded if an enemy is at or right beside
for i = 0 to p_2 − 1 do
  | // p_2 is a prime number, assumed 1000007
  | Hash_Zones_i ← ∅
end
for start_node ∈ Nodes do
  | Bound ← {start_node}
  | Zone ← Bound
  | while ∃x.x ∈ Bound ∧ (Neighbor_x − Zone − Enemy_Nodes ≠ ∅) do
  |   | // there exists a non-enemy point right outside the boundary
  |   | if Bound ⊆ Can_Not_Expand then // no point can be Expanded
  |   |   | S ← {x| min_{x∈Bound} |Eat_Nodesx|}
  |   |   | // the set of points needing the least agents if eating
  |   |   | T ← {x| max_{x∈S} Σ_{y∈Neighbor_x−Zone−Enemy_Nodes} value_y}
  |   |   | // set of nodes in S maximizing total cost
  |   |   | Zone ← Zone ∪ {x|x ∈ Neighbor_y − Zone − Enemy_Nodes ∧ min_{y∈T} y}
  |   |   | // Select any point in T, expand it
  |   | else
  |   |   | S ← {x| min_{x∈Bound−Can_Not_Expand} |Expand_Nodes_x|}
  |   |   | T ← {x| max_{x∈S} Σ_{y∈Neighbor_x−Zone} value_y}
  |   |   | Zone ← Zone ∪ {x|x ∈ Neighbor_y − Zone ∧ min_{y∈T} y}
  |   |   | // Select any point in T, expand it
  |   | end
  |   | Bound ← {x ∈ Zone|Neighbor_x ⊄ Zone}
  |   | if Σ_{x∈Best_Zone} value_x < Σ_{x∈Zone} value_x then
  |   |   | Best_Zone ← Zone
  |   | end
  |   | hash ← (Σ_{x_i∈Zone,0≤i<|Zone|} x_i × p_1^i) mod p_2
  |   | // p_1 and p_2 are prime numbers, p_1 is 1007 and p_2 is 1000007
  |   | if Zone ∈ Hash_Zones_hash then
  |   |   | break
  |   | end
  |   | Hash_Zones_hash ← Hash_Zones_hash ∪ Zone
  | end
end
return Best_Zone
```

Algorithm 2: Expand(Nodes, Edges, value, Enemy_Nodes)

Table 2. Experimental results for Algorithm 2

Nodes	Edges	Enemies	Time	Time (distributed)
100	300	20	0.0207s	0.002s
200	600	20	0.2910s	0.006s
300	900	20	1.2581s	0.013s
400	1200	20	3.5527s	0.023s
500	1500	20	7.4012s	0.034s
1000	3000	20	> 30s	0.128s

however, at each step during the contest, the strategy can be changed with a relatively small probability. Intuitively given an area, the safest strategy is to fully cover its boundary, that is, each boundary node is occupied by an agent. However, we sometimes take some risk, hoping to occupy more with the same number of agents. One possible risky strategy is that there is at least an agent at any two adjacent boundary nodes. At the start of the contest, we exploit such risky strategy to conquer an area. If this area is often disturbed by enemies, we will recompute a new area with the aforementioned safe strategy and then move there. To summarize, two factors can trigger strategy changes: (1) whether a conquered area is often disturbed; (2) a relatively small probability. During the procedure of path finding, we exploit Dijkstra Algorithm and Breadth-First Search Algorithm, and we also use Algorithm 1 to prevent any two agents from exploring the same location. During the contest, there is a certain strategy that only saboteurs will buy sabotage device and shield, and the strength value will always be equal to the health value or one unit more. However, according to empirical results, it is best not to buy any facilities. Considering that this does not cause big problems, at the start we randomly make a choice between these strategies. In the contest, we value achievements, from which we are able to obtain some scores at each step, so we try to acquire achievements swiftly, never spending them.

As mentioned earlier, all agents in our team are rational and good team players, that is, each will always try to complete the mission of the group. Moreover, recall that all communications are perfect and all agents will not perform any actions when a certain percept is being passed in the group. In our team all the agents are armed with exactly the same program so that they have equal status. When a list of agents are applying for the same mission, one of them will become a temporary project manager, which is responsible for allocating the mission in an optimal way. Later this project manager will become an ordinary agent and each agent will accomplish her allocated mission separately. Hence we organize our agents explicitly and no hierarchy is exploited. When an agent encounters something emergent, she immediately interrupts her allocated mission and tell all others in the group. The group will possibly relax the team mission so that they are able to accomplish it without this agent. Agents are able to perform planning in path finding and they need complete knowledge about the (local) initial state. Here we do not call a planner, but exploit Dijkstra Algorithm to obtain a shortest path from the source to the destination. To synchronize with the server, the agents use multi-thread TCP/IP listeners to listen to the message from the server, and decide what actions to perform accordingly. Furthermore, a multi-thread TCP/IP sender will send the action to the server. Note that our program is so efficient that any agent is always able to send her action to the server before the next percept arrives.

5 Conclusion

The participation of this contest has greatly improved my knowledge of multi-agent systems and stimulated my interest in conducting research in this area. I have learnt some important strategies to improve the performance of my agent team. Firstly, agents should be trained beforehand to strengthen the preconditions of their actions in order to reduce the search space. For example, the agents would realize that any node should not be surveyed repeatedly so they strengthen the precondition of the survey action. Secondly, the agents should record some optimal solutions in some cases, then with the learned experiences, they will be able to make best responses in similar cases. For instance, if a saboteur encounters an enemy for the first time, she deliberates over the optimal strategy, attacks that enemy, and learns that experience. Then if similar cases happen next time, she will simply behave according to this experience, without deliberation. Thirdly, the agents should keep a balance between maximizing their worst outcome and minimizing the best outcome of their enemies in the meantime.

One strong point of our team is that we use Algorithm 1 for task allocation to avoid repetitive work, hence decreasing cost of the team. Also, Algorithm 2 ensures that our agents are able to search for a large area, and then occupy it. Another is that our team is efficient in that it only takes the team about 0.2 second to make all decisions, on the 300-edge and 800-node map, in a perfect network. This enables us to develop more complex strategies in future contests. The weaknesses of our team are that we do not have a good strategy for disturbing the opponents and we are not able to defend our own area effectively. Because there is a great number of agents and the map is complex, our programs have to run with great efficiency. Hence we choose C++, which is known for its efficiency and flexibility, supporting various data structures and algorithms. Next year we are going to exploit effective strategies to attack enemies' zone and protect our own zone. The performance this year is not so satisfactory and there are many reasons: this was the first time for us to participate, the team consisted of only one member, I have just finished my undergraduate study with little research experience, and I had not enough time to implement all the ideas. For the next year, some changes we would think beneficial include: (1) servers should never send repetitive static information so as to relieve the pressure of network communication; (2) a percept should contain no information about the teammates because the agents should communicate with each other to broadcast such information.

Acknowledgements

I thank Professor Yongmei Liu for introducing me to the Multi-Agent Programming Contest. I am deeply grateful to Yi Fan for his generous and valuable help with the writing of this paper. This project was supported by the Natural Science Foundation of China under Grant No. 61073053.

References

1. Behrens, T., Dastani, M., Dix, J., Köster, M., Novák, P.: Special issue about Multi-Agent-Contest I. In: Annals of Mathematics and Artificial Intelligence. vol. 59. Springer, Netherlands (2010)
2. Behrens, T., Dix, J., Köster, M., Hübner, J.: Special issue about Multi-Agent-Contest II. In: Annals of Mathematics and Artificial Intelligence. vol. 61. Springer, Netherlands (2011)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Section 22.2: Breadth-first search. In: Introduction to Algorithms. pp. 531–539. MIT Press and McGraw-Hill (2001)
4. Dijkstra, E.W.: A note on two problems in connexion with graphs. In: Numerische Mathematik. vol. 1, pp. 260–271. Springer (1959)
5. Edmonds, J.: Maximum matching and a polyhedron with $0,1$ vertices. J. of Res. the Nat. Bureau of Standards 69 B, 125–130 (1965)
6. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. The MIT Press, Cambridge, Massachusetts (1995)
7. Kuhn, H.W., Yaw, B.: The Hungarian method for the assignment problem. Naval Res. Logist. Quart pp. 83–97 (1955)
8. Orlin, J.B.: A faster strongly polynomial minimum cost flow algorithm. Operations Research 41(2), 338–350 (1993)

Short Answers

A Introduction

1. What was the motivation to participate in the contest?
A: Our motivation was to gain deeper understanding about Multi-agent Systems.
2. What is the (brief) history of the team? (MAS course project, thesis evaluation, ...)
A: To test some strategies and then formalize them to gain a general method.
3. What is the name of your team?
A: My team name is AiWYX.
4. How many developers and designers did you have? At what level of education are your team members?
A: Only one. Bachelor of Software Engineering.
5. From which field of research do you come from? Which work is related?
A: Knowledge representation. Game theories and distributing algorithms.

B System Analysis and Design

1. Did you use multi-agent programming languages? Please justify your answer.
A: No, I didn't. Because of efficiency.
2. If some multi-agent system methodology such as Prometheus, O-MaSE, or Tropos was used, how did you use it? If you did not, please justify.
A: No, I am so proficient in this language and it is also famous because of efficiency.
3. Is the solution based on the centralisation of coordination/information on a specific agent? Conversely if you plan a decentralised solution, which strategy do you plan to use?
A: I have exploited decentralization in implementing various strategies, however, the implementation now is so restricted because we only deal with common knowledge.
4. What is the communication strategy and how complex is it?
A: All agents share exactly the same knowledge at any time. No agents will perform any action when a new percept is propagating. We apply for a piece of main memory from the operating system to store their knowledge.
5. How are the following agent features considered/implemented: *autonomy*, *proactiveness*, *reactiveness*?
A: In any state of the world, any agent is able to perform some actions to approach the goal, never lost in a dead-end. The agents here keep exploring new area of the world, never passively waiting for changes of the environments. In any state of the world, any agent knows exactly what it should do next to achieve this goal and it will invoke a new task after completing one.
6. Is the team a truly **multi**-agent system or rather a centralised system in disguise?

A: I have exploited decentralization in implementing various strategies, however, the implementation now is so restricted because we only deal with common knowledge.
7. How much time (person hours) have you invested (approximately) for implementing your team?
A: About 250 hours.
8. Did you discuss the design and strategies of your agent team with other developers? To which extent did you test your agents playing with other teams?
A: I am the only one in the team, however, my supervisor does provide some key tips. I once tested my program on a single computer, that is, I started a competition between two multi-agent teams, both of which were equipped with my own program.

C Software Architecture

1. Which programming language did you use to implement the multi-agent system?
A: C++
2. How have you mapped the designed architecture (both multi-agent and individual agent architectures) to programming codes, i.e., how did you implement specific agent-oriented concepts and designed artifacts using the programming language?
A: I mapped the designed architecture to several programming levels. At the first level is the agent directly thinks on the overall strategy. The second level is team work level that arrange the mission to every agent which join the mission. The third level is for reasoning with the knowledge base. The fourth level is responsible for various network communications and concrete algorithms.
3. Which development platforms and tools are used? How much time did you invest in learning those?
A: Just Gedit Text Editor. I didn't invest any time in learning that.
4. Which runtime platforms and tools (e.g. Jade, AgentScape, simply Java, ...) are used? How much time did you invest in learning those?
A: None. I didn't invest any time in learning that.
5. What features were missing in your language choice that would have facilitated your development task?
A: I have implemented all proposed features efficiently, due to the flexibility of this language.
6. Which algorithms are used/implemented?
A: Breadth-first search, Dijkstra algorithm, minimum cost flow algorithm and Hungarian algorithm.
7. How did you distribute the agents on several machines? And if you did not please justify why.
A: I am able to distribute the agents by changing initialization files but I did not do so in the contest.

8. To which extent is the reasoning of your agents synchronized with the receive-percepts/send-action cycle?

A: If an agent receives a new percept, any other agent will perform no actions until this percept is passed to all agents in this group.

9. What part of the development was most difficult/complex? What kind of problems have you found and how are they solved?

A: The most significant difficulty is how to make the best team strategies. By strengthening action preconditions and training the agents so that they know what should be best responses.

10. How many lines of code did you write for your software?

A: About 10,000 lines.

D Strategies, Details and Statistics

1. What is the main strategy of your team?

A: The whole team explores the whole map for available areas, and then they tried to occupy areas with higher values. If any minor events occur, such as encountering enemies or some teammates injured, they will abort this task to deal with that. In both processes, the agents cooperate in a optimal manner, so that they are able to accomplish the task with lower cost.

2. How does the overall team work together? (coordination, information sharing, ...)

A: We allocate each agent a unique task. When any agent receives a new percept, any other agent will not perform any actions until this percept is passed to all of them. This ensures that all agents share a synchronized knowledge base.

3. How do your agents analyze the topology of the map? And how do they exploit their findings?

A: Each time an agent arrives at an unexplored location, it collects all information about edges and nodes. Their strategy now is to move to those nodes on the boundary, survey them and then continue this process again and again.

4. How do your agents communicate with the server?

A: We exploit multi-threaded TCP/IP protocol.

5. How do you implement the roles of the agents? Which strategies do the different roles implement?

A: We have designed a particular strategy for each of the five roles in the game. When an agent realizes that it is acting a certain role, say, repairer, it will follow the respective strategy. Only explorers accept the mission of exploring the map and probe the value of the newly encountered node. Only sentinels and inspectors explorers will occupy the zones. Repairers will run to the injured and repair their teammates while saboteurs will go to front line and fight with enemies.

6. How do you find good zones? How do you estimate the value of zones?

A: First I will choose each node as a point zone with no enemies standing at and then repeat the following: find the boundary node P s.t. after expanding P

the boundary increases the least (possibly by a negative number), and then, expand it. During the expanding process, we maintain the optimal zone ever found. By the following:

$$\mathtt{value}_{\mathtt{Zone}} = \frac{\sum_{\mathtt{i} \in \mathtt{Zone}} \mathtt{value}_{\mathtt{i}}}{|\mathtt{agent\_number\_to\_occupy}|}.$$

7. How do you conquer zones? How do you defend zones if attacked? Do you attack zones?

A: Our agents will compute the most promising zone with Algorithm 2 and then move to the boundary and conquer it. If some enemies attack them, they will recompute a new area not occupied by the enemies, and then move there. Among them the saboteurs will always attack the nearest area occupied by the rival.

8. Can your agents change their behavior during runtime? If so, what triggers the changes?

A: Yes. We have set a random number to change their behaviors with a relatively small probability at each step. Furthermore if their zone is often disturbed, they will change their original strategy into a more safer one.

9. What algorithm(s) do you use for agent path planning?

A: Dijkstra algorithm,Breadth-First Search Algorithm and algorithm 1.

10. How do you make use of the buying-mechanism?

A: During the contest, there is a certain strategy that only saboteurs will buy sabotage device and shield and the strength value will always be equal to the health value or one unit more.

11. How important are achievements for your overall strategy?

A: In the contest, we value achievements for it is used for obtaining scores.

12. Do your agents have an explicit mental state?

A: No.

13. How do your agents communicate? And what do they communicate?

A: My agents communicate by sharing memory. They communicate about the information of map, enemies, status of friend agents, arrangement of team work mission.

14. How do you organize your agents? Do you use e.g. hierarchies? Is your organization implicit or explicit?

A: They are all equipped with the same program, share the same knowledge base at any time and act themselves. They are all at the same status. Every time a group of agents are applying for the same mission, one of them will be a part-time manager. Hence we organize our agents explicitly and no hierarchy is exploited.

15. Is most of you agents' behavior emergent on an individual or team level?

A: When an agent is in emergent situation, it will interrupt its allocated task to deal with it.

16. If your agents perform some planning, how many steps do they plan ahead?

A: It will only perform planning in path finding where the number of steps is equal to the number of nodes in a path.

17. If you have a perceive-think-act cycle, how is it synchronized with the server?

A: To synchronize with the server, multi-thread TCP/IP listeners listen to the message from the server, and the respective agent will decide which action to perform. Furthermore a multi-thread TCP/IP sender will send the action to the server. Note that our program is so efficient that any agent is always able to send its action to the server before the next percept arrives.

E Conclusion

1. What have you learned from the participation in the contest?

A: We strengthen the preconditions in order to restrict the search space. Moreover it should record some important information in previous procedure. Another issue is to keep the balance between maximizing our worst outcome and minimizing the best outcome of the enemies in the meantime. We ought to consider the response of our enemies when striving for our ideal outcome.

2. Which are the strong and weak points of the team?

A: One of our strong points is that we use Algorithm 1 for scheduling to avoid redundant work, decreasing cost of the team. Besides Algorithm 2 ensures that our agents search for a large and unexplored area and then occupy it. Weaknesses are that we are lack of a good disturbing strategy and not able to defend our own area effectively.

3. How suitable was the chosen programming language, methodology, tools, and algorithms?

A: I choose C++ to ensure efficiency and support various algorithms.

4. What can be improved in the contest for next year?

A: Add the strategy to attack enemies' zones. Try to protect my zone.

5. Why did your team perform as it did? Why did the other teams perform better/worse than yours?

A: My team just contain one person. And it's my first time participated in this contest. And I have just got a bachelor degree. And I don't have enough time to implement all my thought. Their teams disturb my zone, so my zone is not stable and I don't disturb enemy's zone.

6. Which other research fields might be interested in the Multi-Agent Programming Contest?

A: Distributed algorithms, Game theory.

7. How can the current scenario be optimized? How would those optimization pay off?

A: (1) servers should never send repetitive static information; (2) a percept should contain no information about the teammates. That will relieve the pressure of network communications.

11 PGIM

Team PGIM comes from the Islamic Azad University of Malayer, Iran. The 3 developers used agent-specific technologies for developing their team: Prometheus, JACK. Nevertheless the team organization is not distributed, and agents broadcast their percepts.

PGIM did not provide an extensive team description to include in this document.

12 Streett

Team Streett was composed by a single independent developer from the USA. Agents were developed in Java, based on the sample agents provided with the *MASSim* platform. Agents shared only vital information and coordination was achieved by sharing location data.

Streett did not provide an extensive team description to include in this document.

Part III
All Results in Great Detail

13 AiWYX vs. PGIM – Simulation 1

13.1 Scores, Zone Stability and Achievements

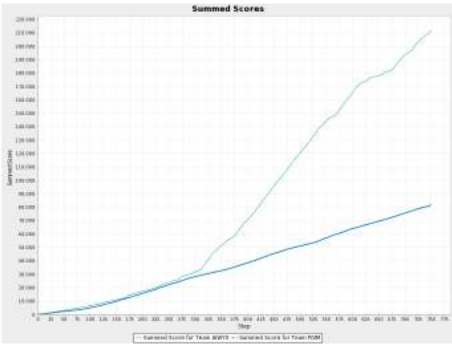


Figure 51: Summed scores.

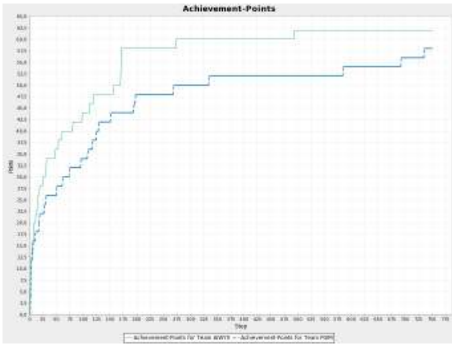


Figure 52: Achievement points.

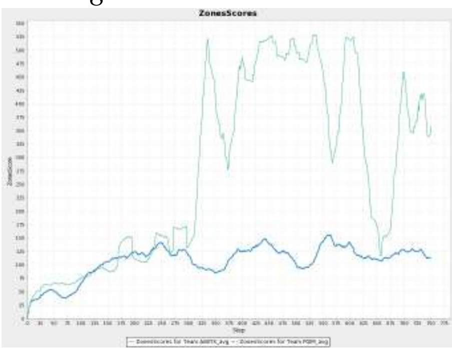


Figure 53: Zones scores.

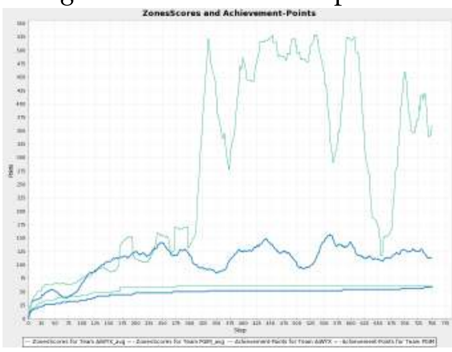


Figure 54: Zones scores and achievement points.

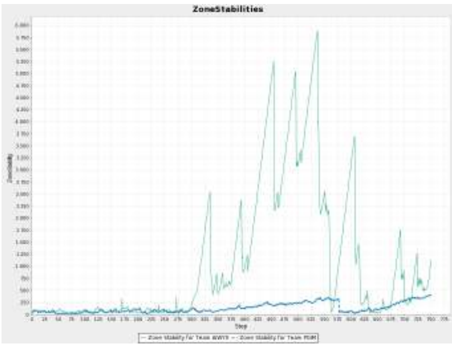


Figure 55: Zone Stabilities.

Step	AiWYX	PGIM
1	surveyed10, area20, surveyed40, area10, surveyed20	surveyed10, area10
2	surveyed80	area20, surveyed40, surveyed20
3		surveyed80
4	surveyed160, proved5	
5		proved5
6		surveyed160
7	proved10	
8	inspected5	
10		proved10
11	surveyed320	
14	proved20	
15	attacked5	
17		proved20
18	inspected10	surveyed320
24	attacked10	
27		inspected5
30	proved40	attacked5
31	attacked20	
47	surveyed640	
50		attacked10
53	attacked40	
60	proved80	
61		proved40
74		attacked20
80	inspected20	
95		area40
98	attacked80	
108		parried5
112	proved160	
117		inspected10
119	area40	
124		attacked40
128		area80
151		parried10
156	attacked160	
170	area160, area320, area80, area640	
194		attacked80
197		proved80
268		parried20
273	attacked320	
334		attacked160
493	attacked640	
584		attacked320
692		surveyed640
735		parried40

Figure 56: Achievements.

13.2 Stability

Reason	AiWYX	%	PGIM	%
failed away	1	0,01	8	0,05
failed parried	55	0,37		
failed wrong param			27	0,18
failed random	159	1,06	127	0,85
failed resources			1	0,01
failed attacked	62	0,41	48	0,32

Figure 57: Failed actions.

13.3 Achievements

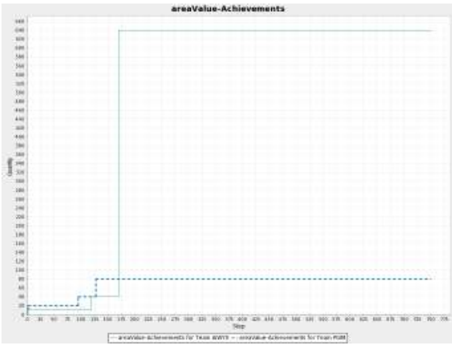


Figure 58: areaValueAchievements.

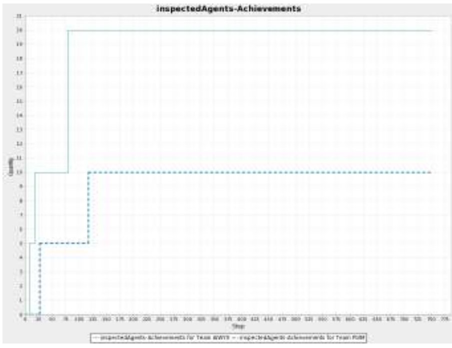


Figure 59: inspectedAgentsAchievements.

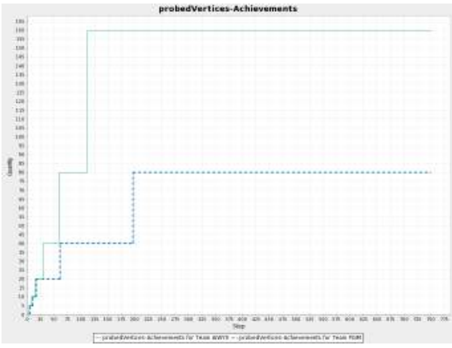


Figure 60: probedVerticesAchievements.

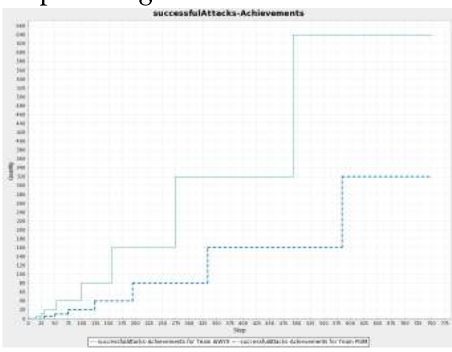


Figure 61: successfulAttacksAchievements.

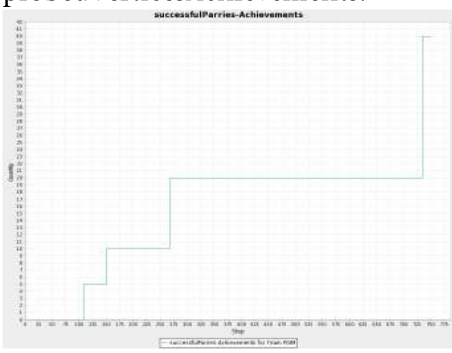


Figure 62: successfulParriesAchievements.

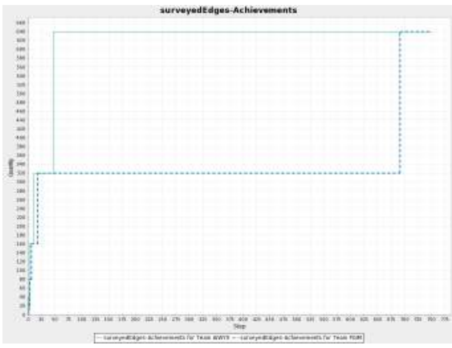


Figure 63: surveyedEdgesAchievements.

13.4 Actions per Role

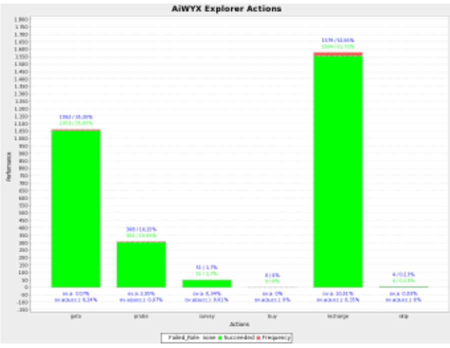


Figure 64: AiWYX vs. PGIM – Simulation 1 - AiWYX Explorer Actions.

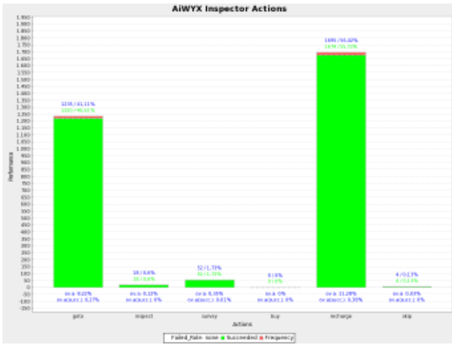


Figure 65: AiWYX vs. PGIM – Simulation 1 - AiWYX Inspector Actions.

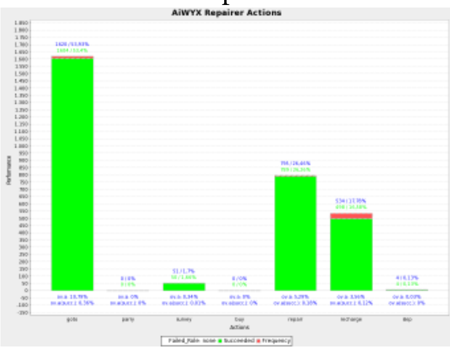


Figure 66: AiWYX vs. PGIM – Simulation 1 - AiWYX Repairer Actions.

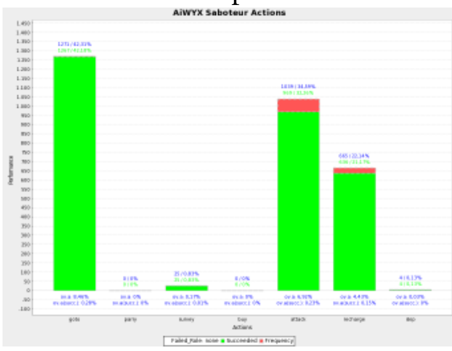


Figure 67: AiWYX vs. PGIM – Simulation 1 - AiWYX Saboteur Actions.

Figure 68: AiWYX vs. PGIM – Simulation 1 - AiWYX Sentinel Actions.

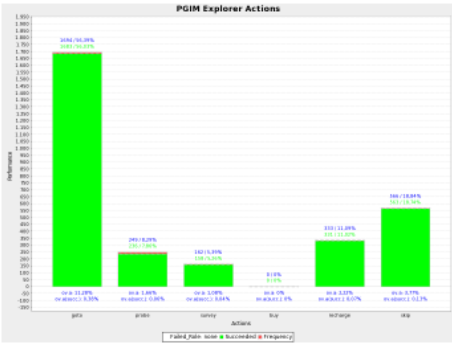


Figure 69: AiWYX vs. PGIM – Simulation 1 - PGIM Explorer Actions.

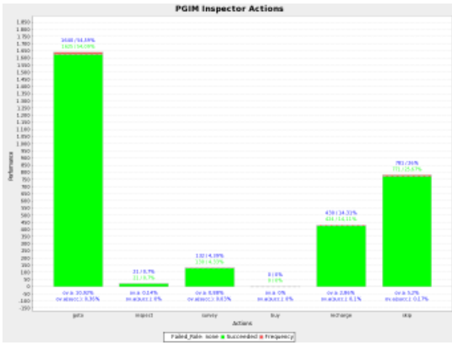


Figure 70: AiWYX vs. PGIM – Simulation 1 - PGIM Inspector Actions.

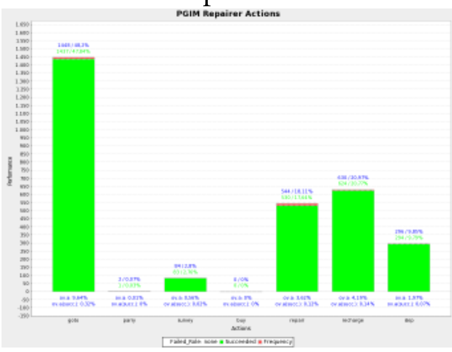


Figure 71: AiWYX vs. PGIM – Simulation 1 - PGIM Repairer Actions.

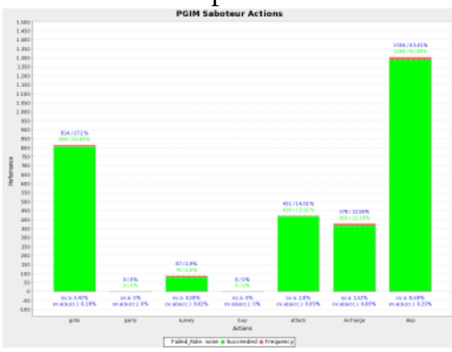


Figure 72: AiWYX vs. PGIM – Simulation 1 - PGIM Saboteur Actions.

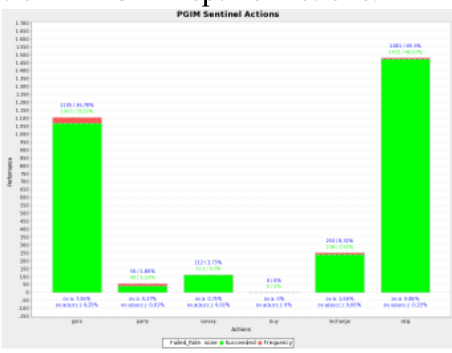


Figure 73: AiWYX vs. PGIM – Simulation 1 - PGIM Sentinel Actions.

14 AiWYX vs. PGIM – Simulation 2

14.1 Scores, Zone Stability and Achievements

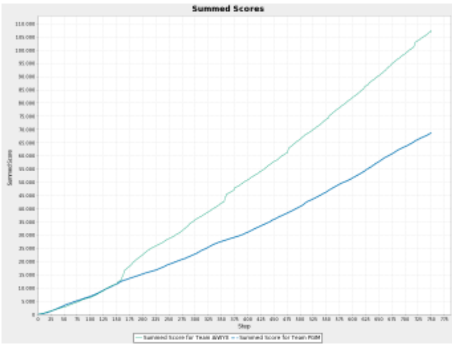


Figure 74: Summed scores.

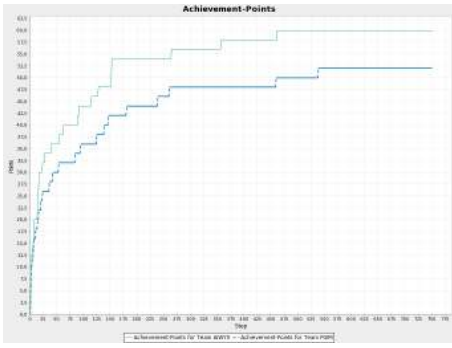


Figure 75: Achievement points.

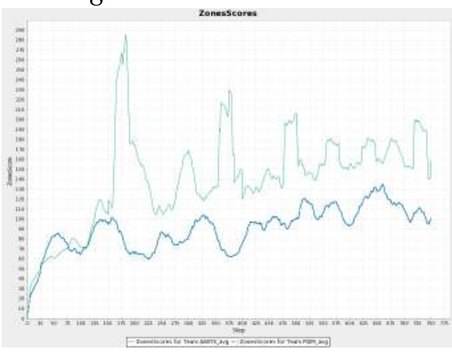


Figure 76: Zones scores.

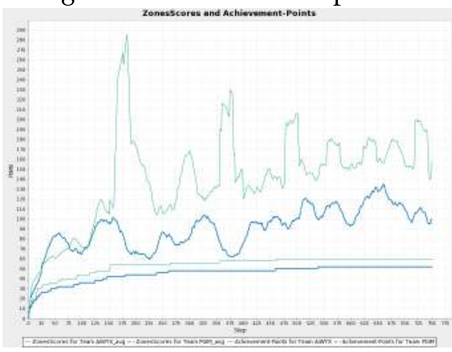


Figure 77: Zones scores and achievement points.

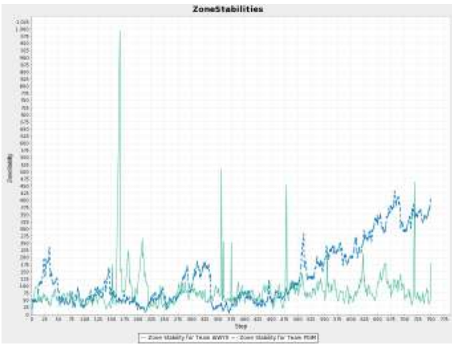


Figure 78: Zone Stabilities.

Step	AiWYX	PGIM
1	surveyed10, surveyed40, surveyed20	surveyed10, surveyed40, surveyed20
2	surveyed80, area10	surveyed80, area10
3	proved5	
4	surveyed160	proved5
6	area20	surveyed160
7	proved10, inspected5	proved10
10		area20
14	inspected10, proved20	proved20
15	attacked5	
16	area40	surveyed320
17	surveyed320	
20		area40
23	attacked10	attacked5
27	proved40	
36		attacked10
39	attacked20	
42		proved40
53		attacked20
55	proved80	
62	attacked40	
84		attacked40
89	surveyed640	
92	attacked80	
94		parried5
114	proved160	
124		area80
127	area80	
138		attacked80
146		parried10
151	attacked160	
152	area160, area320	
180		proved80
238		attacked160
260		parried20
264	attacked320	
356	area640	
459		attacked320
461	attacked640	
538		parried40

Figure 79: Achievements.

14.2 Stability

Reason	AiWYX	%	PGIM	%
failed away			12	0,08
failed parried	77	0,51		
failed wrong param			48	0,32
failed random	150	1	134	0,89
failed resources			1	0,01
failed attacked	83	0,55	80	0,53

Figure 80: Failed actions.

14.3 Achievements

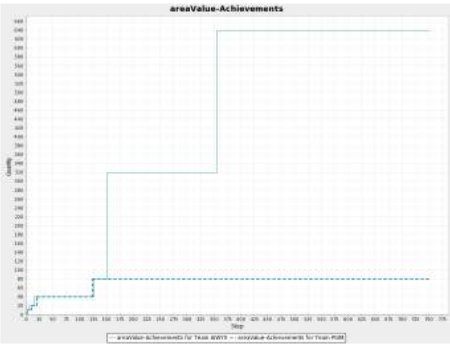


Figure 81: areaValueAchievements.

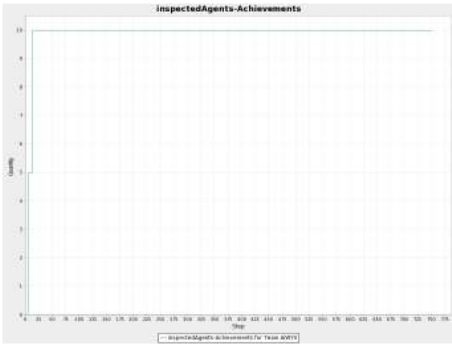


Figure 82: inspectedAgentsAchievements.

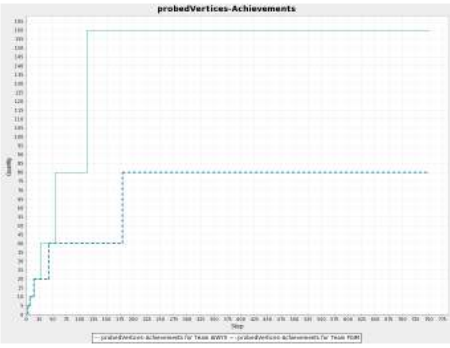


Figure 83: probedVerticesAchievements.

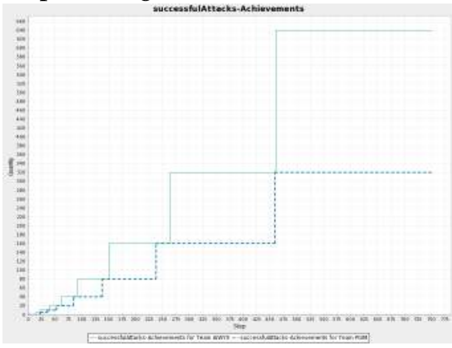


Figure 84: successfulAttacksAchievements.

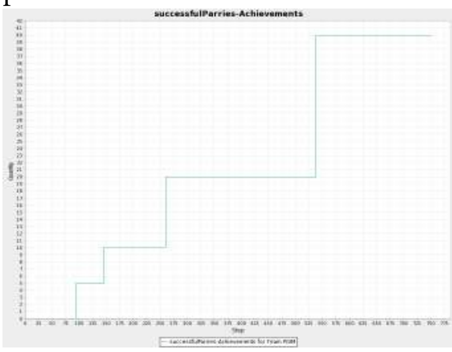


Figure 85: successfulParriesAchievements.

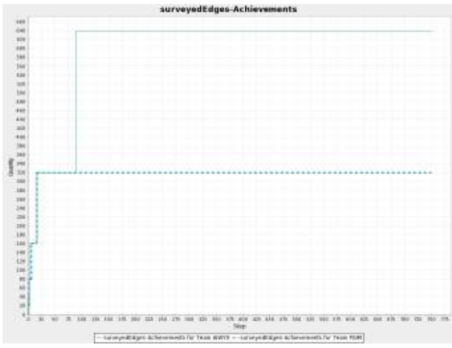


Figure 86: surveyedEdgesAchievements.

14.4 Actions per Role

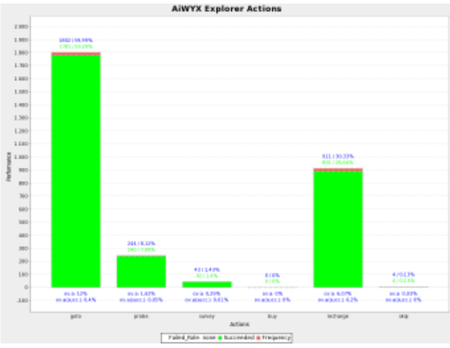


Figure 87: AiWYX vs. PGIM – Simulation 2 - AiWYX Explorer Actions.

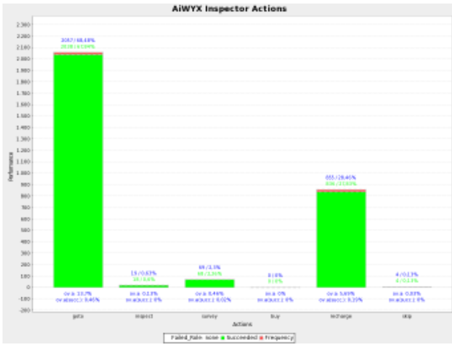


Figure 88: AiWYX vs. PGIM – Simulation 2 - AiWYX Inspector Actions.

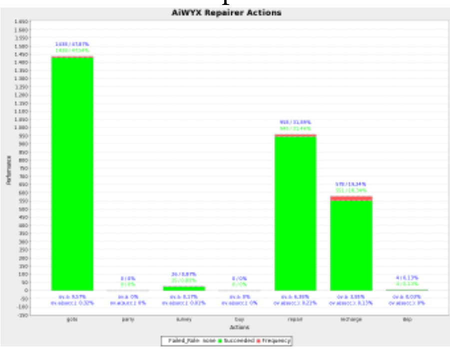


Figure 89: AiWYX vs. PGIM – Simulation 2 - AiWYX Repairer Actions.

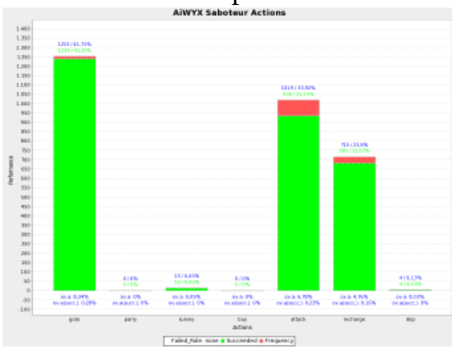


Figure 90: AiWYX vs. PGIM – Simulation 2 - AiWYX Saboteur Actions.

Figure 91: AiWYX vs. PGIM – Simulation 2 - AiWYX Sentinel Actions.

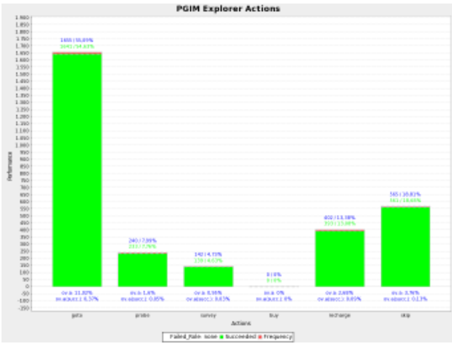


Figure 92: AiWYX vs. PGIM – Simulation 2 - PGIM Explorer Actions.

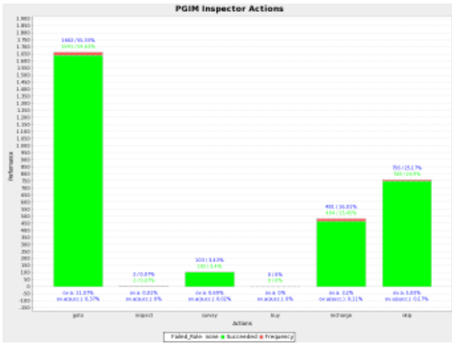


Figure 93: AiWYX vs. PGIM – Simulation 2 - PGIM Inspector Actions.

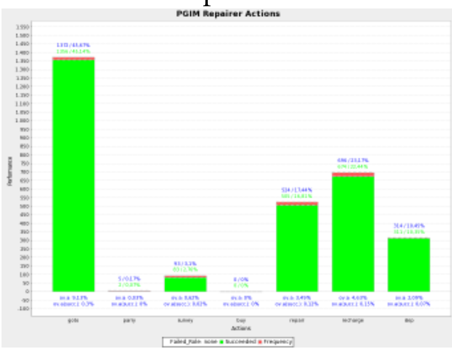


Figure 94: AiWYX vs. PGIM – Simulation 2 - PGIM Repairer Actions.

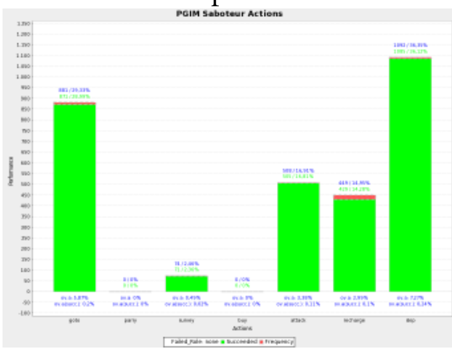


Figure 95: AiWYX vs. PGIM – Simulation 2 - PGIM Saboteur Actions.

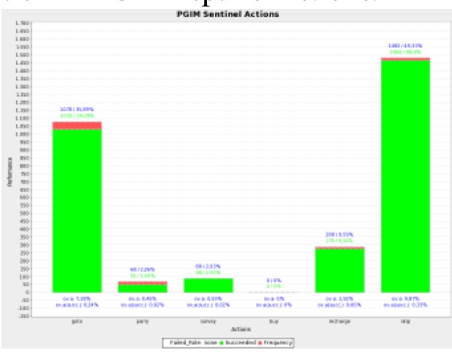


Figure 96: AiWYX vs. PGIM – Simulation 2 - PGIM Sentinel Actions.

15 AiWYX vs. PGIM – Simulation 3

15.1 Scores, Zone Stability and Achievements

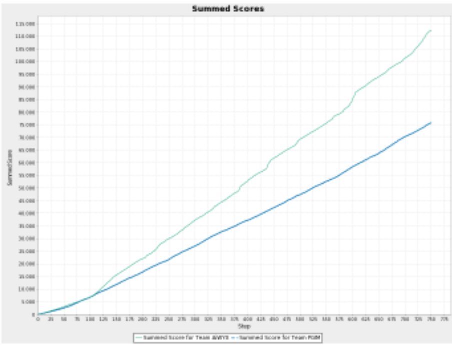


Figure 97: Summed scores.

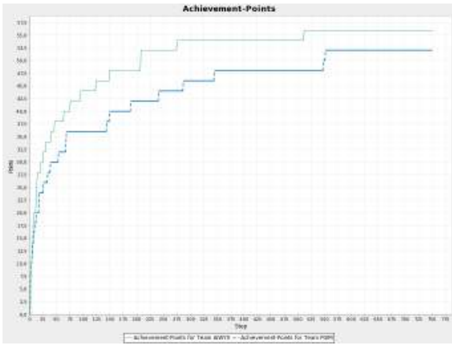


Figure 98: Achievement points.

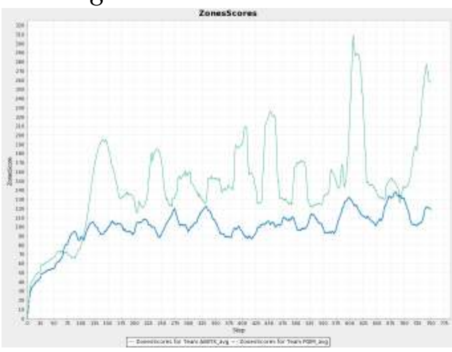


Figure 99: Zones scores.

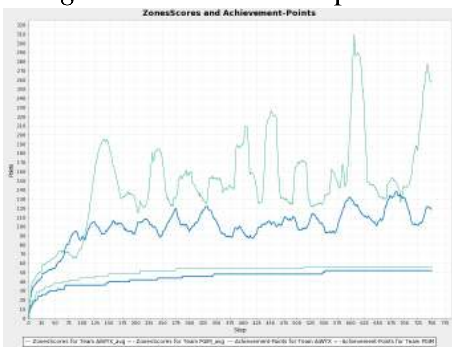


Figure 100: Zones scores and achievement points.

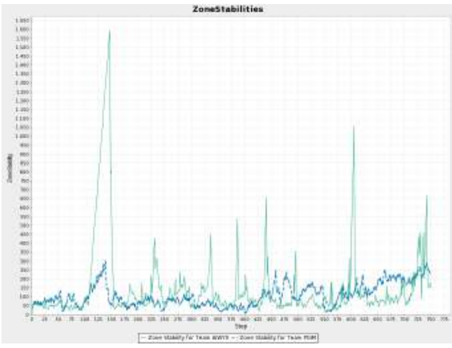


Figure 101: Zone Stabilities.

Step	AiWYX	PGIM
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
2	surveyed80	surveyed80
4	surveyed160, proved5	proved5
5		surveyed160
6	proved10, inspected5	
7		proved10
8	area20	
9		area20
11	attacked5	
12	proved20	attacked5
13	inspected10	
14	attacked10	
17		proved20, attacked10
20	surveyed320	
25	attacked20	area40
30	proved40	
33		attacked20
38		surveyed320
40	area40	
46	attacked40	
53		proved40
63	proved80	
67		attacked40
68		area80
75	attacked80	
94	area80	
123	proved160	
143		parried5
148	attacked160	attacked80
188		parried10
207	area160, area320	
240		proved80
274	attacked320	
286		attacked160
344		parried20
511	attacked640	
547		attacked320
551		parried40

Figure 102: Achievements.

15.2 Stability

Reason	AiWYX	%	PGIM	%
failed away			12	0,08
failed parried	95	0,63		
failed random	147	0,98	176	1,17
failed wrong param			55	0,37
failed resources			1	0,01
failed attacked	72	0,48	71	0,47

Figure 103: Failed actions.

15.3 Achievements

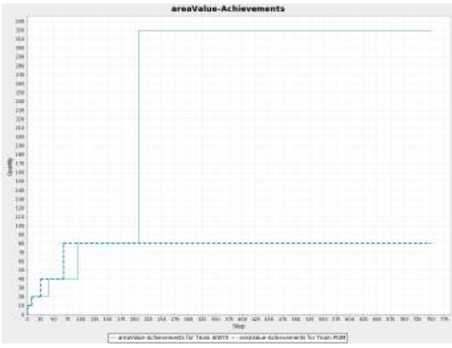


Figure 104: areaValueAchievements.

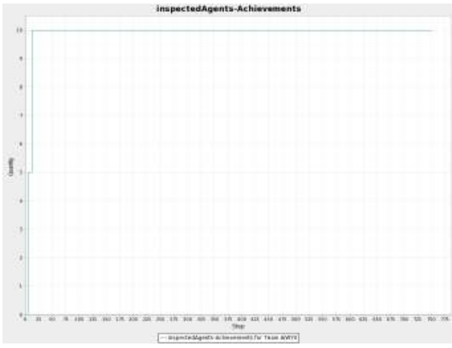


Figure 105: inspectedAgentsAchievements.

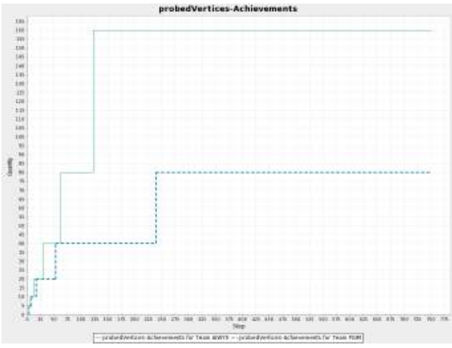


Figure 106: probedVerticesAchievements.

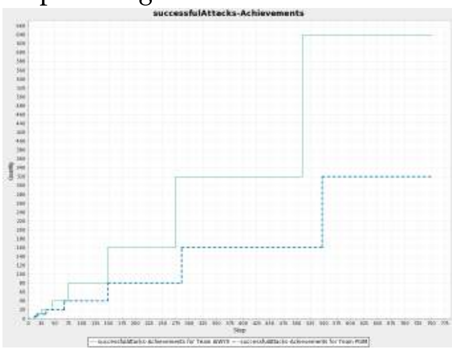


Figure 107: successfulAttacksAchievements.

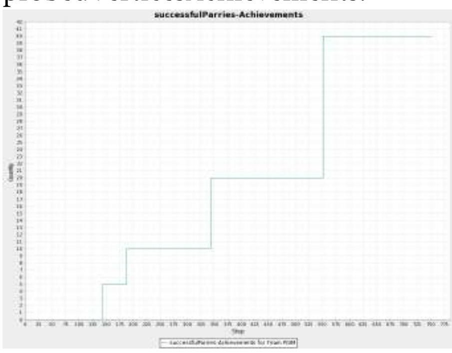


Figure 108: successfulParriesAchievements.

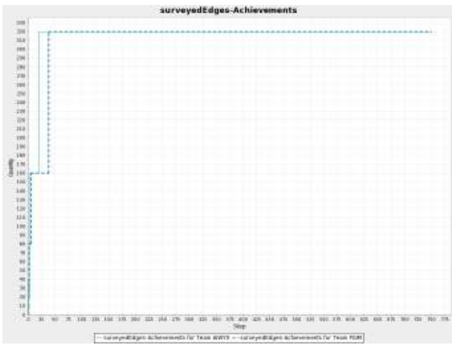


Figure 109: surveyedEdgesAchievements.

15.4 Actions per Role

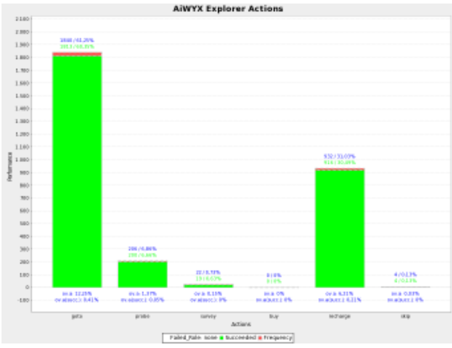


Figure 110: AiWYX vs. PGIM – Simulation 3 - AiWYX Explorer Actions.

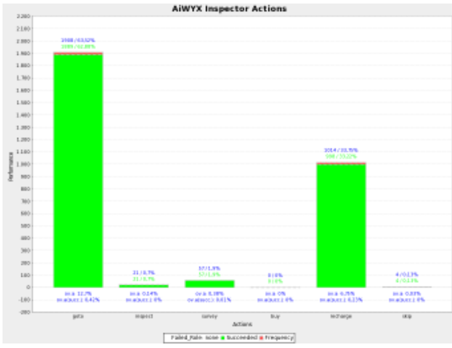


Figure 111: AiWYX vs. PGIM – Simulation 3 - AiWYX Inspector Actions.

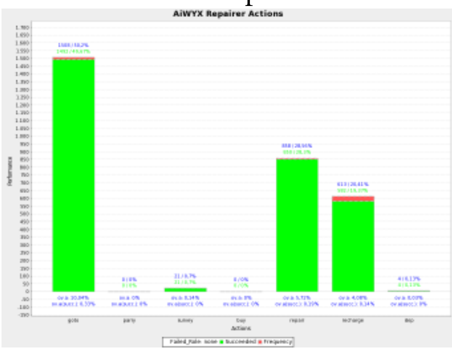


Figure 112: AiWYX vs. PGIM – Simulation 3 - AiWYX Repairer Actions.

Figure 113: AiWYX vs. PGIM – Simulation 3 - AiWYX Saboteur Actions.

Figure 114: AiWYX vs. PGIM – Simulation 3 - AiWYX Sentinel Actions.

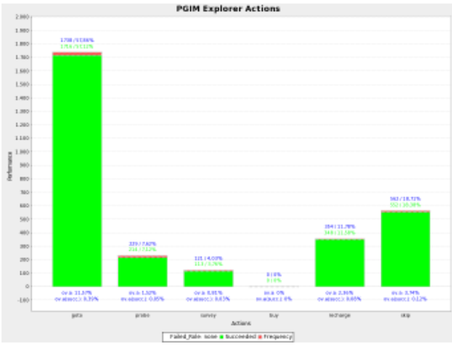


Figure 115: AiWYX vs. PGIM – Simulation 3 - PGIM Explorer Actions.

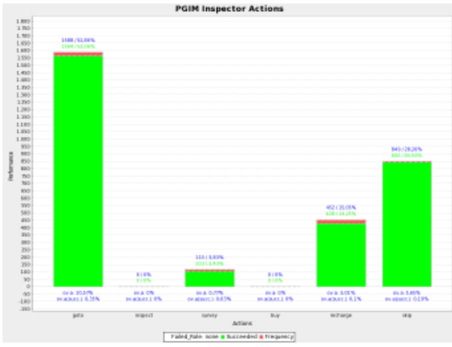


Figure 116: AiWYX vs. PGIM – Simulation 3 - PGIM Inspector Actions.

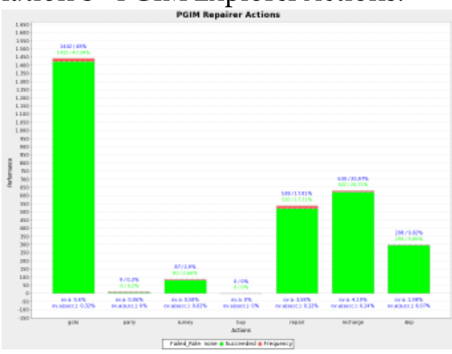


Figure 117: AiWYX vs. PGIM – Simulation 3 - PGIM Repairer Actions.

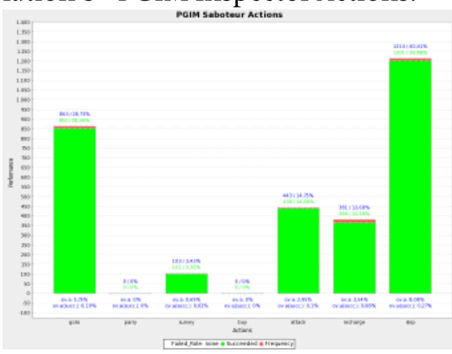


Figure 118: AiWYX vs. PGIM – Simulation 3 - PGIM Saboteur Actions.

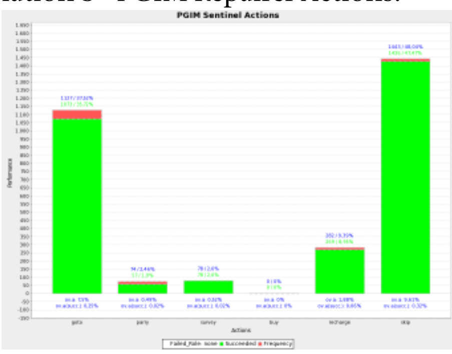


Figure 119: AiWYX vs. PGIM – Simulation 3 - PGIM Sentinel Actions.

16 AiWYX vs. Python-DTU – Simulation 1

16.1 Scores, Zone Stability and Achievements

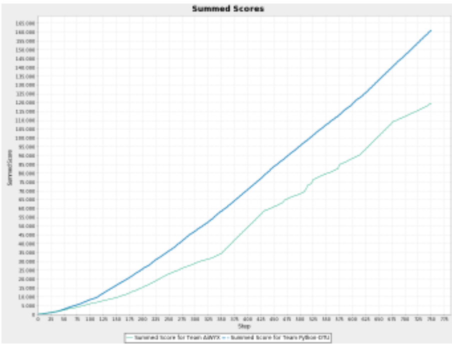


Figure 120: Summed scores.

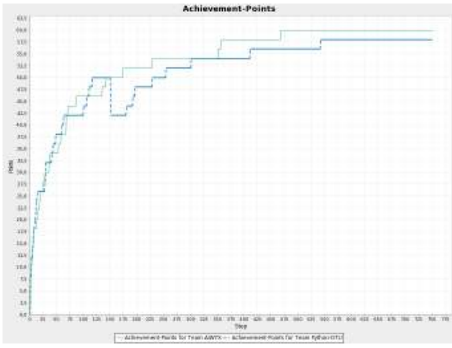


Figure 121: Achievement points.

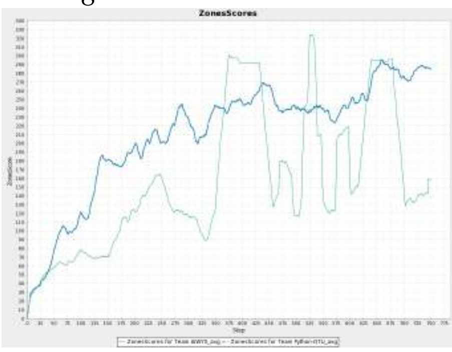


Figure 122: Zones scores.

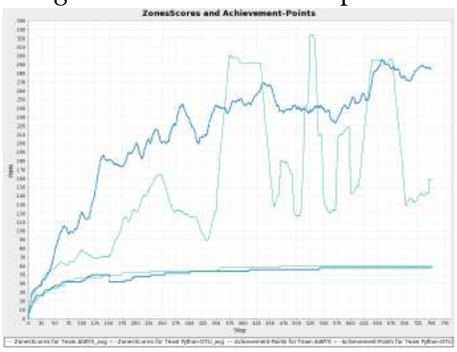


Figure 123: Zones scores and achievement points.

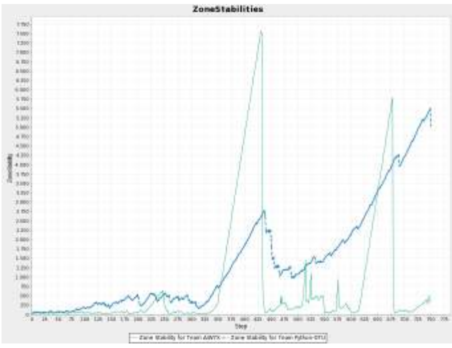


Figure 124: Zone Stabilities.

Step	AiWYX	Python-DTU
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
2	surveyed80	
3		surveyed80, proved5
4	proved5	
5	surveyed160	proved10
6		inspected5
7	proved10, attacked5	attacked5
9		surveyed160
11		inspected10
12		proved20
13	inspected5	
14		attacked10
15	surveyed320	
18	proved20	
19	area20	
25	attacked10	
28	inspected10	area20
29		proved40, surveyed320
36	proved40	
37	attacked20	
41		attacked20
43		area40
48		area80
53	area40	
58	attacked40	
60		proved80
63		attacked40
68	proved80, inspected20	
71	surveyed640	
86	attacked80	
100		attacked80
106		surveyed640
110		proved160
116		area160
135	attacked160	
142	proved160	
174	area80	
180		attacked160
192		parried5
196		parried10
228	attacked320	parried20
254		inspected20
301		attacked320
351	area160	
357	attacked640	
411		parried40
468	area320	
542		attacked640

Figure 125: Achievements.

16.2 Stability

Reason	AiWYX	%	Python-DTU	%
failed away	1	0,01		
failed parried	110	0,73		
failed random	152	1,01	146	0,97
failed attacked	72	0,48	150	1

Figure 126: Failed actions.

16.3 Achievements

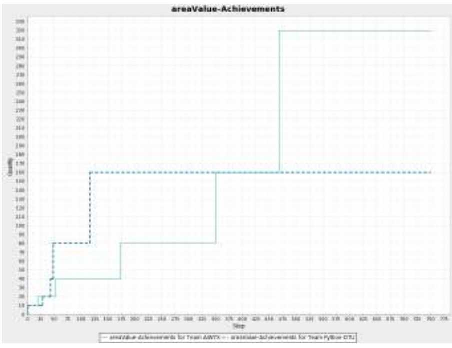


Figure 127: areaValueAchievements.

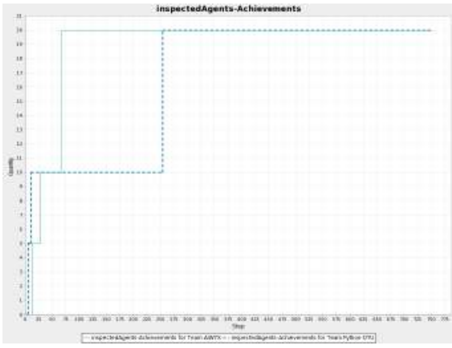


Figure 128: inspectedAgentsAchievements.

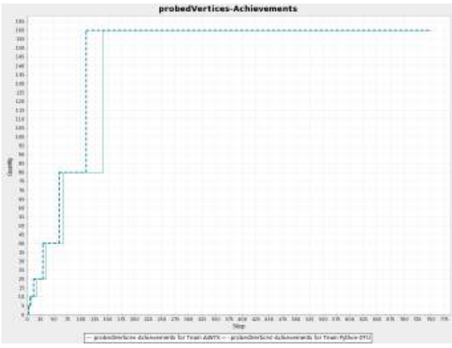


Figure 129: probedVerticesAchievements.

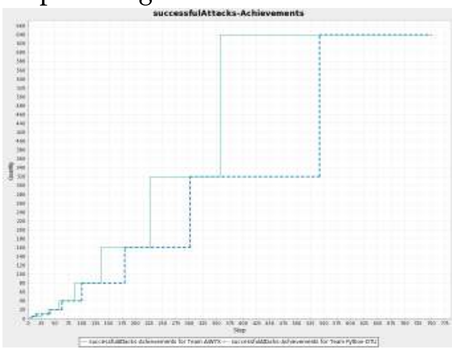


Figure 130: successfulAttacksAchievements.

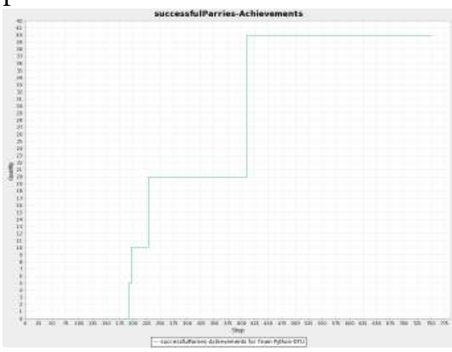


Figure 131: successfulParriesAchievements.

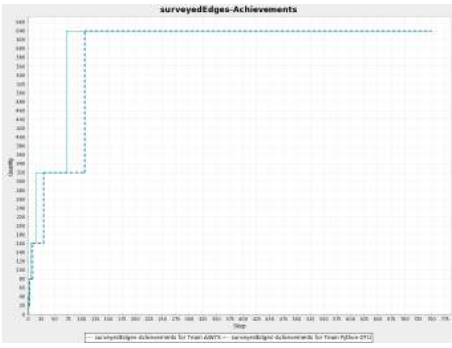


Figure 132: surveyedEdgesAchievements.

16.4 Actions per Role

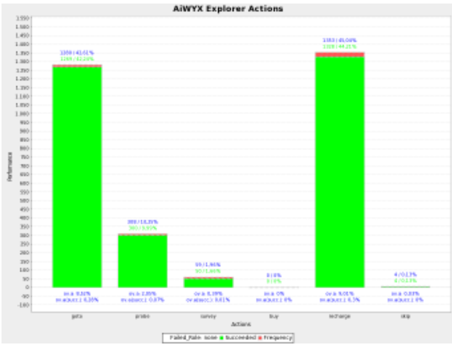


Figure 133: AiWYX vs. Python-DTU – Simulation 1 - AiWYX Explorer Actions.

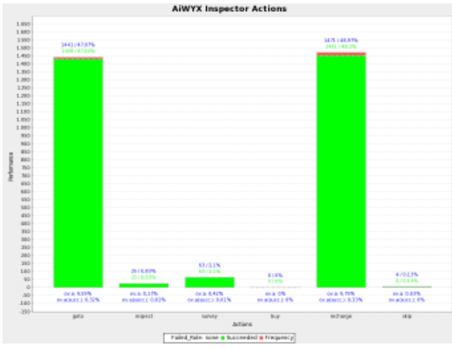


Figure 134: AiWYX vs. Python-DTU – Simulation 1 - AiWYX Inspector Actions.

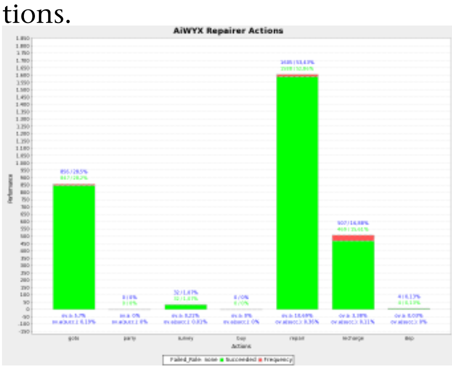


Figure 135: AiWYX vs. Python-DTU – Simulation 1 - AiWYX Repairer Actions.

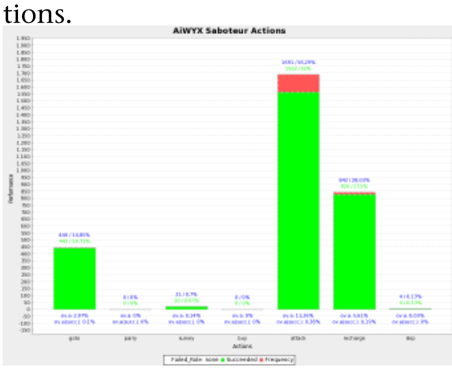


Figure 136: AiWYX vs. Python-DTU – Simulation 1 - AiWYX Saboteur Actions.

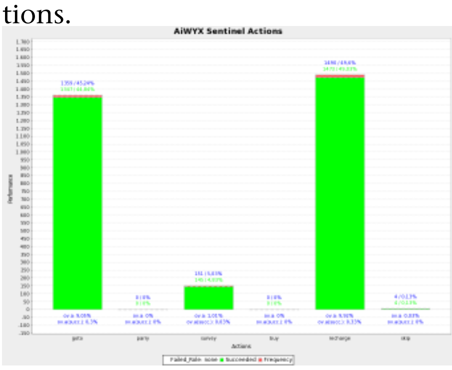


Figure 137: AiWYX vs. Python-DTU – Simulation 1 - AiWYX Sentinel Actions.

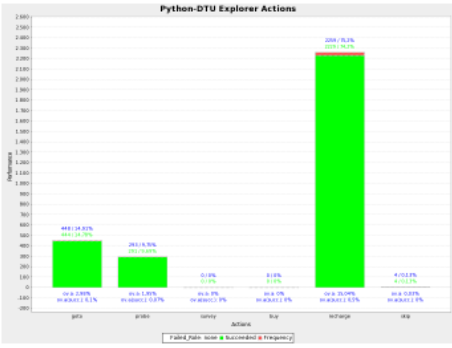


Figure 138: AiWYX vs. Python-DTU – Simulation 1 - Python-DTU Explorer Actions.

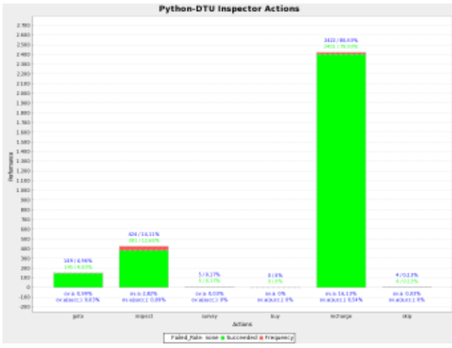


Figure 139: AiWYX vs. Python-DTU – Simulation 1 - Python-DTU Inspector Actions.

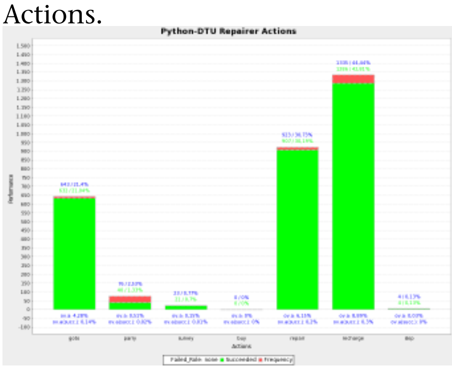


Figure 140: AiWYX vs. Python-DTU – Simulation 1 - Python-DTU Repairer Actions.

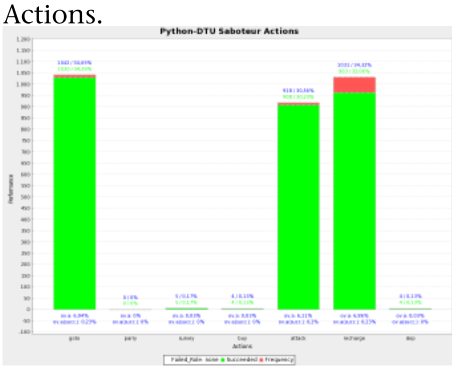


Figure 141: AiWYX vs. Python-DTU – Simulation 1 - Python-DTU Saboteur Actions.

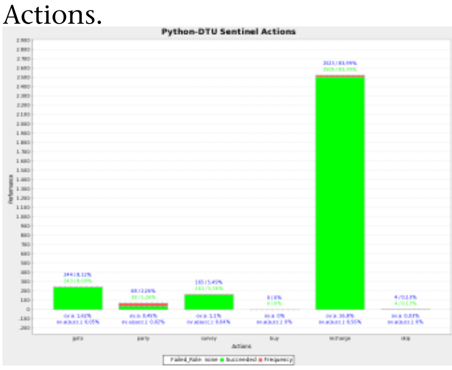


Figure 142: AiWYX vs. Python-DTU – Simulation 1 - Python-DTU Sentinel Actions.

17 AiWYX vs. Python-DTU – Simulation 2

17.1 Scores, Zone Stability and Achievements

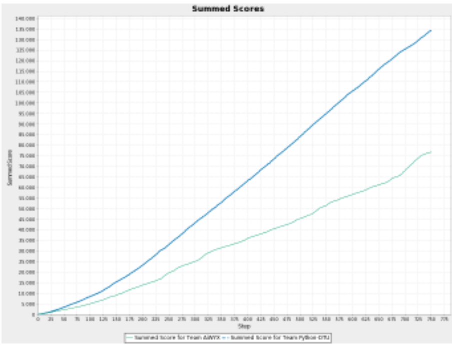


Figure 143: Summed scores.

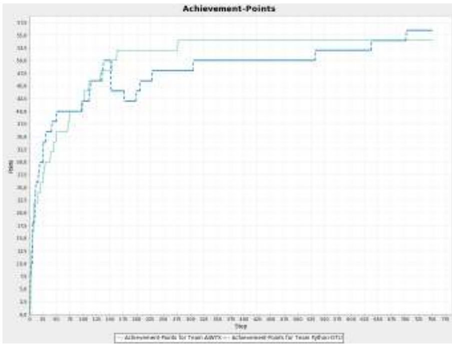


Figure 144: Achievement points.

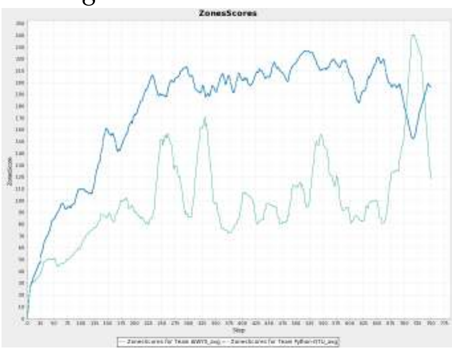


Figure 145: Zones scores.

Figure 146: Zones scores and achievement points.

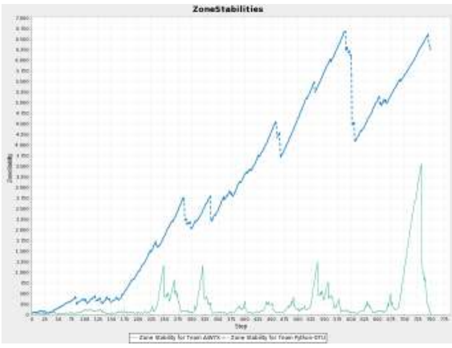


Figure 147: Zone Stabilities.

Step	AiWYX	Python-DTU
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
2	surveyed80	
3	inspected5	proved5
4	proved5	
5	surveyed160	proved10, surveyed80, attacked5, inspected5
7	area20, proved10	
8	attacked5	
9		proved20
10		inspected10, attacked10
11		surveyed160
15	surveyed320	
16		area20
18		attacked20
20	inspected10	
24	proved20	
25		proved40, surveyed320
28	attacked10	
30		area40
38	attacked20	
41		attacked40
44	proved40	
49	attacked40	
50		proved80
71	attacked80	
74	proved80	
96	surveyed640	
98		proved160
101	attacked160	
110	area40	
111		surveyed640
113		inspected20
132	proved160	
135		attacked80
137		area80
153	attacked320	
162	area80	
198		attacked160
206		parried5
228		area160
276	attacked640	
305		attacked320
532		attacked640
637		parried10
702		parried20

Figure 148: Achievements.

17.2 Stability

Reason	AiWYX	%	Python-DTU	%
failed away			2	0,01
failed parried	30	0,2		
failed random	166	1,11	142	0,95
failed	2	0,01		
failed attacked	63	0,42	165	1,1
noAction	2	0,01		

Figure 149: Failed actions.

17.3 Achievements

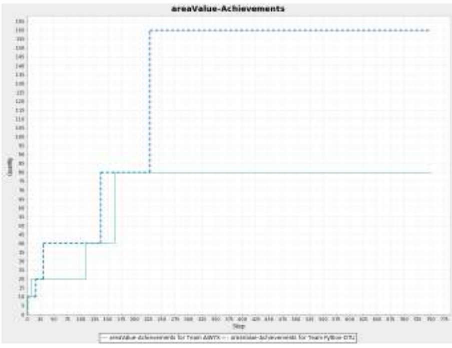


Figure 150: areaValueAchievements.

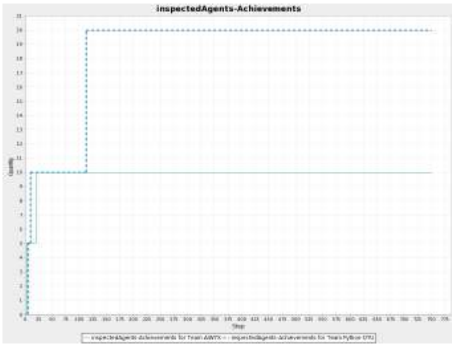


Figure 151: inspectedAgentsAchievements.

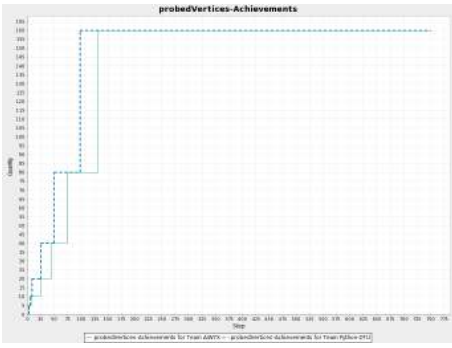


Figure 152: probedVerticesAchievements.

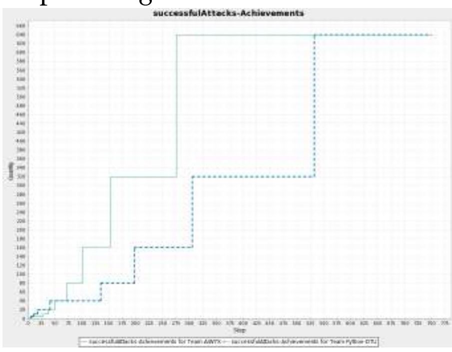


Figure 153: successfulAttacksAchievements.

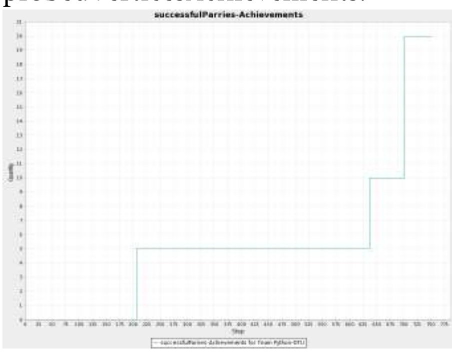


Figure 154: successfulParriesAchievements.

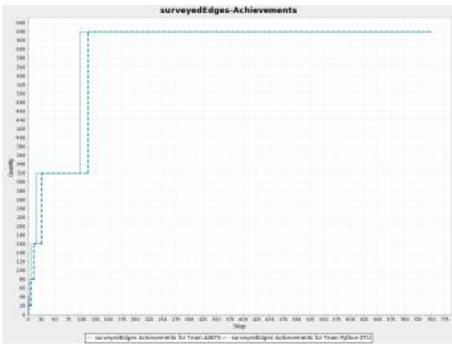


Figure 155: surveyedEdgesAchievements.

17.4 Actions per Role

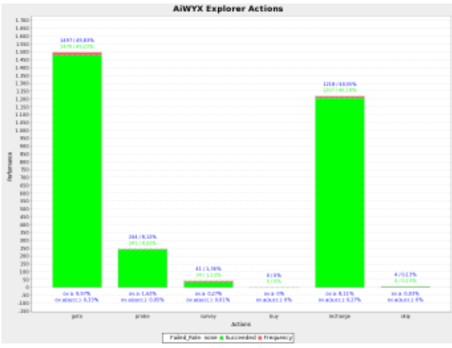


Figure 156: AiWYX vs. Python-DTU – Simulation 2 - AiWYX Explorer Actions.

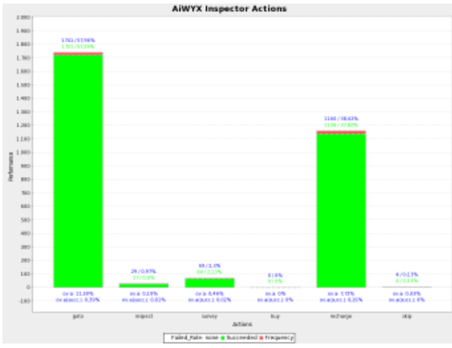


Figure 157: AiWYX vs. Python-DTU – Simulation 2 - AiWYX Inspector Actions.

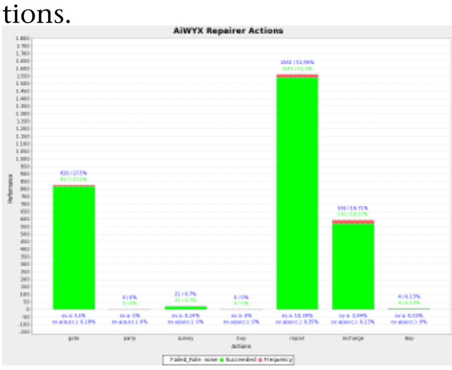


Figure 158: AiWYX vs. Python-DTU – Simulation 2 - AiWYX Repairer Actions.

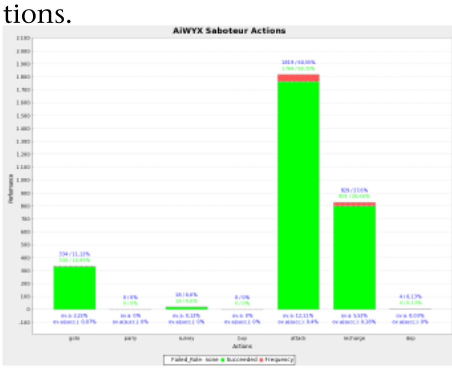


Figure 159: AiWYX vs. Python-DTU – Simulation 2 - AiWYX Saboteur Actions.

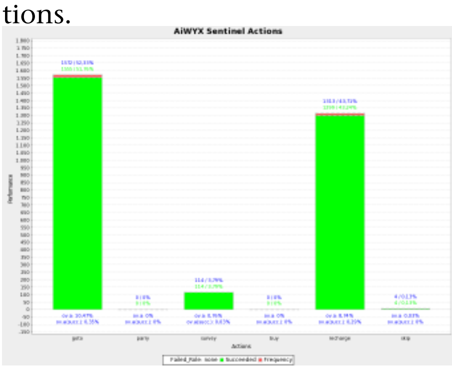


Figure 160: AiWYX vs. Python-DTU – Simulation 2 - AiWYX Sentinel Actions.

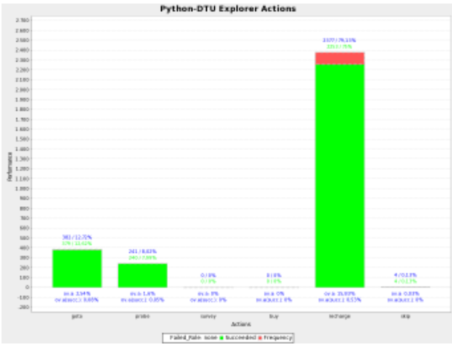


Figure 161: AiWYX vs. Python-DTU – Simulation 2 - Python-DTU Explorer Actions.

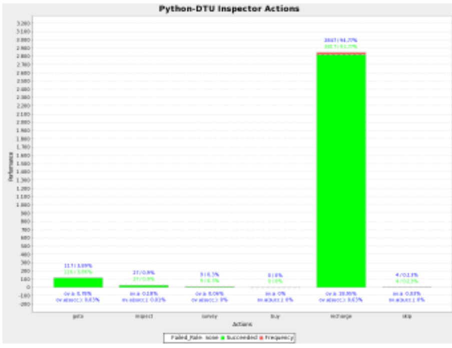


Figure 162: AiWYX vs. Python-DTU – Simulation 2 - Python-DTU Inspector Actions.

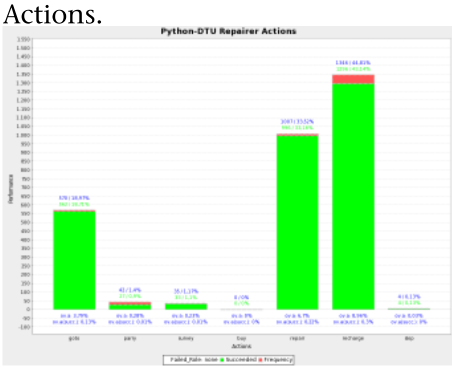


Figure 163: AiWYX vs. Python-DTU – Simulation 2 - Python-DTU Repairer Actions.

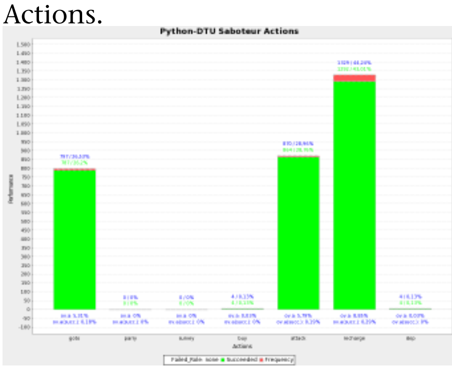


Figure 164: AiWYX vs. Python-DTU – Simulation 2 - Python-DTU Saboteur Actions.

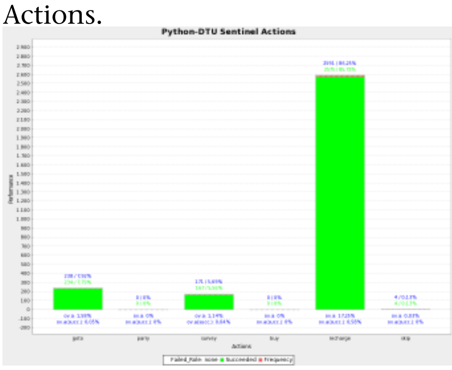


Figure 165: AiWYX vs. Python-DTU – Simulation 2 - Python-DTU Sentinel Actions.

18 AiWYX vs. Python-DTU – Simulation 3

18.1 Scores, Zone Stability and Achievements

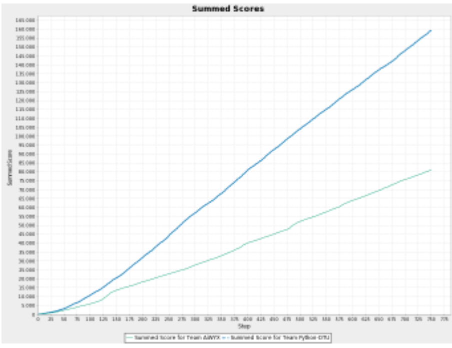


Figure 166: Summed scores.

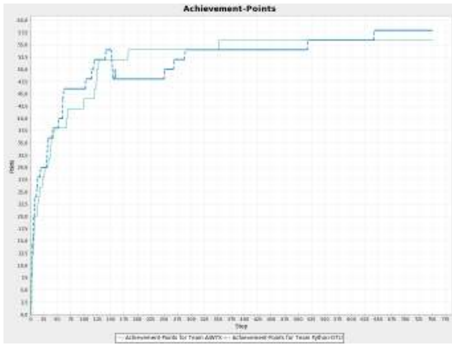


Figure 167: Achievement points.

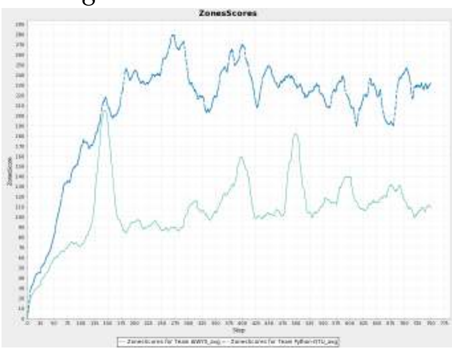


Figure 168: Zones scores.

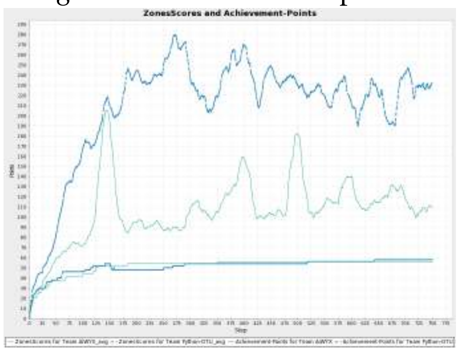


Figure 169: Zones scores and achievement points.

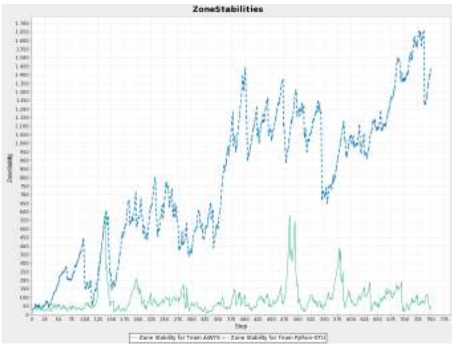


Figure 170: Zone Stabilities.

Step	AiWYX	Python-DTU
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
2	surveyed80	
3	proved5	surveyed80, proved5, inspected5
4		attacked5
5	surveyed160	area20, proved10
6	proved10, inspected5	
7		inspected10
8	attacked5	attacked10
11		proved20
12	attacked10	
13		surveyed160
14	inspected10	
17	proved20	
18		attacked20
22	attacked20	
26	area20	
30		inspected20
31		proved40
32		attacked40
33	surveyed320	
37	proved40	
38	attacked40	
41		surveyed320
43	area40	
52		area40
60		attacked80, area80
61		proved80
67	proved80	
69	attacked80	
99	inspected20	
103		parried5
114		proved160
118		area160
119	attacked160	
123	area160, area80	
127	proved160	
140		attacked160
157		parried10
182	attacked320	
250		parried20
267		attacked320
288		parried40
351	attacked640	
517		attacked640
642		parried80

Figure 171: Achievements.

18.2 Stability

Reason	AiWYX	%	Python-DTU	%
failed away	2	0,01		
failed parried	124	0,83		
failed random	147	0,98	132	0,88
failed attacked	104	0,69	161	1,07

Figure 172: Failed actions.

18.3 Achievements

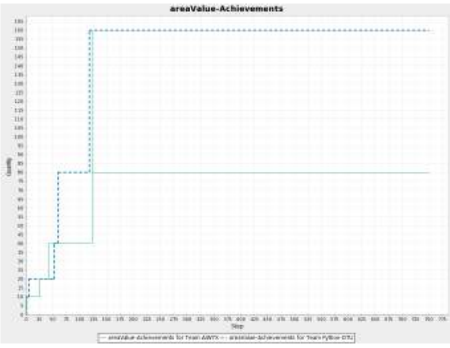


Figure 173: areaValueAchievements.

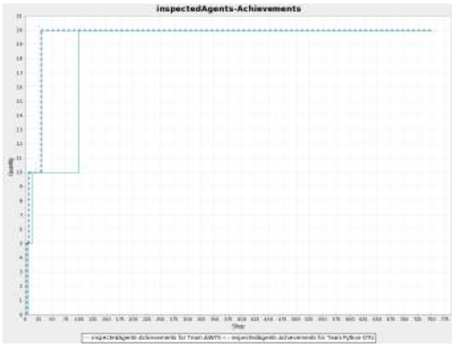


Figure 174: inspectedAgentsAchievements.

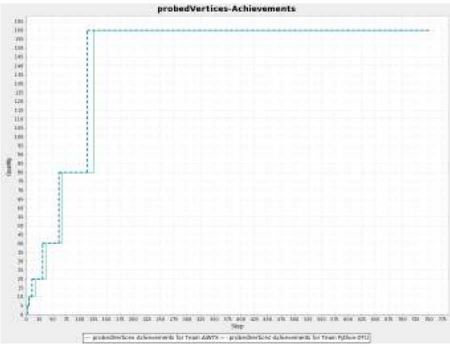


Figure 175: probedVerticesAchievements.

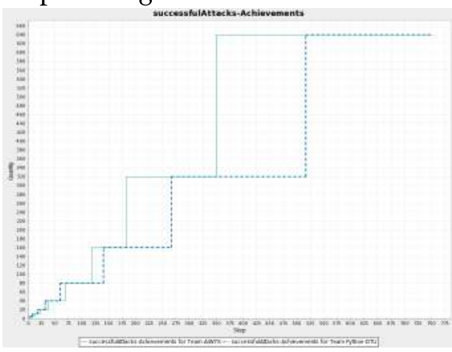


Figure 176: successfulAttacksAchievements.

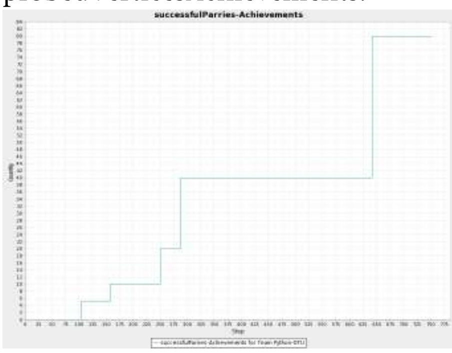


Figure 177: successfulParriesAchievements.

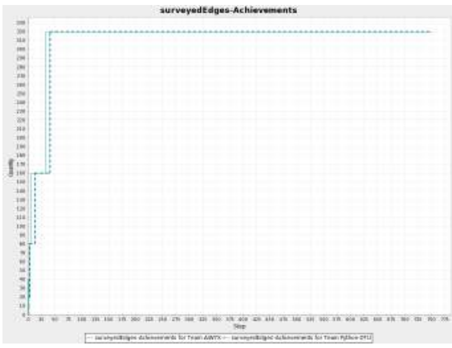


Figure 178: surveyedEdgesAchievements.

18.4 Actions per Role

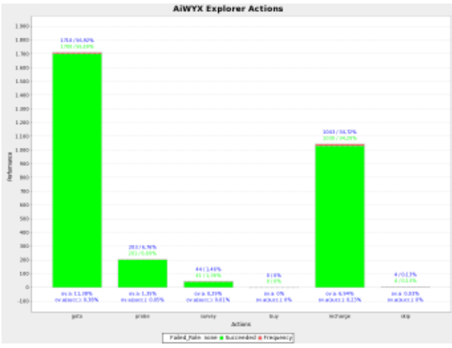


Figure 179: AiWYX vs. Python-DTU – Simulation 3 - AiWYX Explorer Actions.

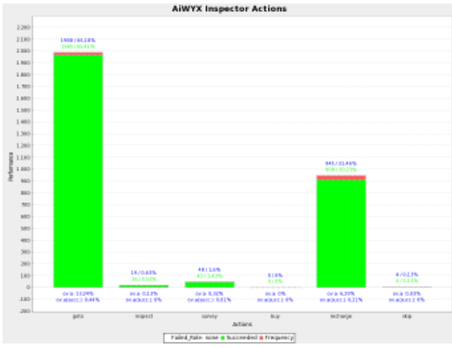


Figure 180: AiWYX vs. Python-DTU – Simulation 3 - AiWYX Inspector Actions.

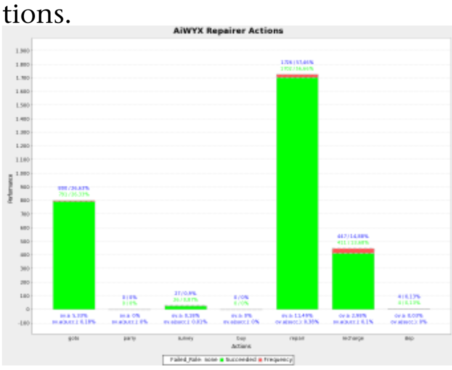


Figure 181: AiWYX vs. Python-DTU – Simulation 3 - AiWYX Repairer Actions.

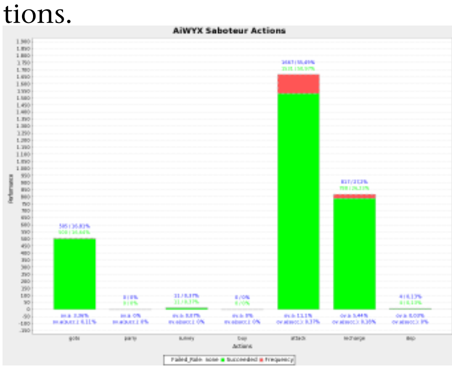


Figure 182: AiWYX vs. Python-DTU – Simulation 3 - AiWYX Saboteur Actions.

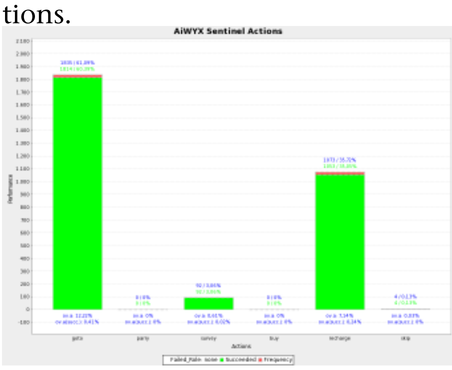


Figure 183: AiWYX vs. Python-DTU – Simulation 3 - AiWYX Sentinel Actions.

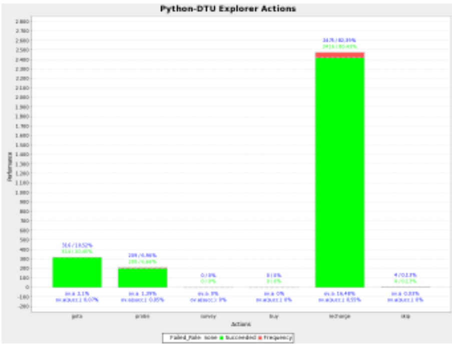


Figure 184: AiWYX vs. Python-DTU – Simulation 3 - Python-DTU Explorer Actions.

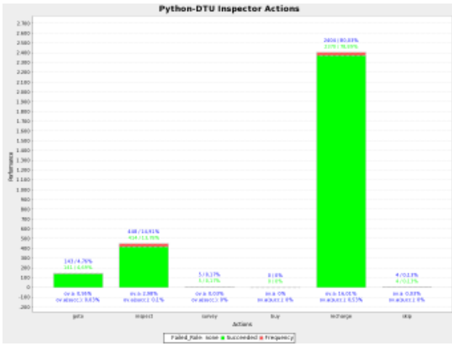


Figure 185: AiWYX vs. Python-DTU – Simulation 3 - Python-DTU Inspector Actions.

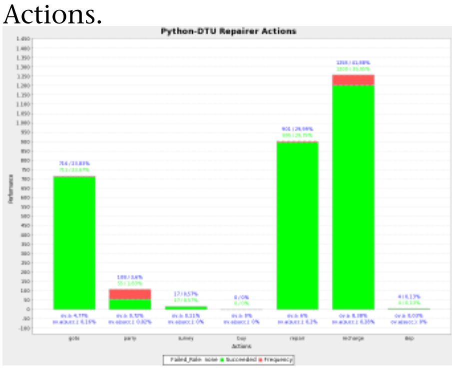


Figure 186: AiWYX vs. Python-DTU – Simulation 3 - Python-DTU Repairer Actions.

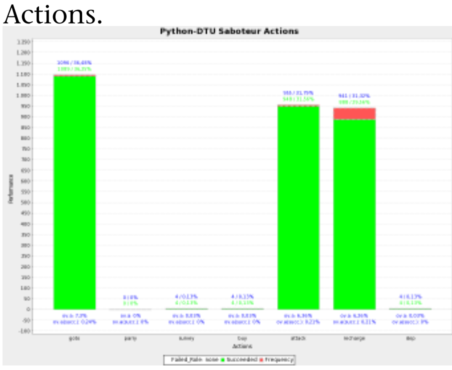


Figure 187: AiWYX vs. Python-DTU – Simulation 3 - Python-DTU Saboteur Actions.

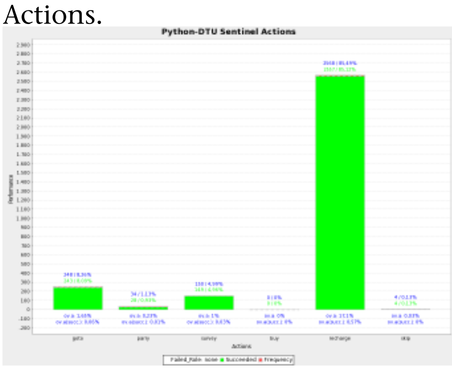


Figure 188: AiWYX vs. Python-DTU – Simulation 3 - Python-DTU Sentinel Actions.

19 AiWYX vs. Streett – Simulation 1

19.1 Scores, Zone Stability and Achievements

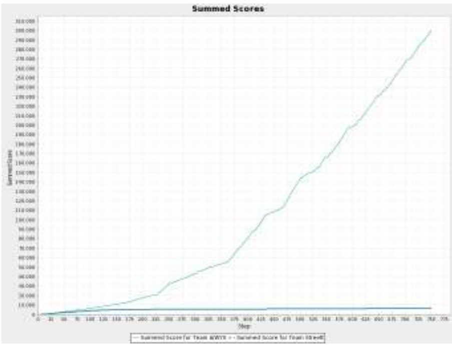


Figure 189: Summed scores.

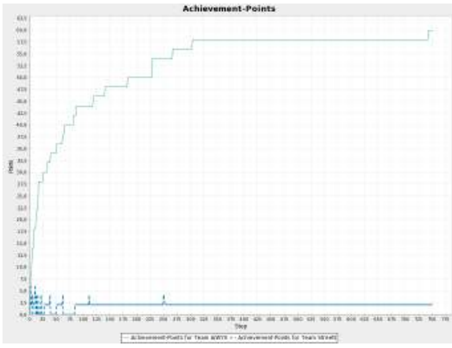


Figure 190: Achievement points.

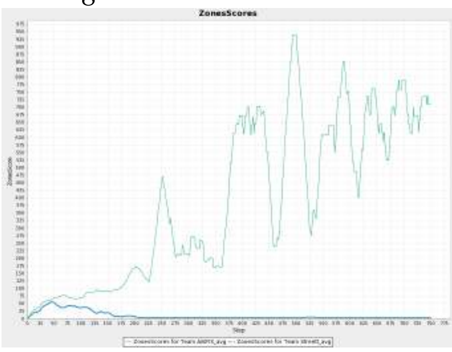


Figure 191: Zones scores.

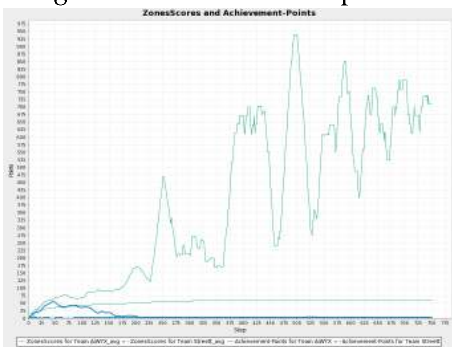


Figure 192: Zones scores and achievement points.

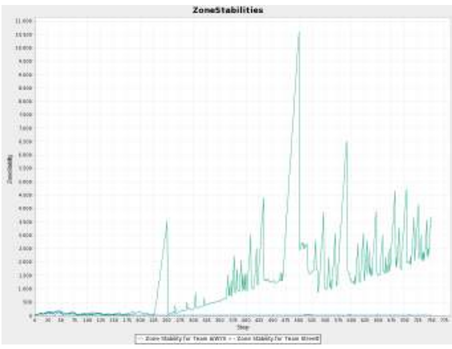


Figure 193: Zone Stabilities.

Step	AiWYX	Streett
1	surveyed10, surveyed80, surveyed40, surveyed20	surveyed40, surveyed10, surveyed20
2		surveyed80
3	surveyed160	
4	proved5, inspected5	proved5
6		proved10
7	proved10	
8	attacked5	
9		area10, attacked5
11	area10	
12	surveyed320	area20, proved20
15	inspected10	surveyed160, inspected5
16	area20, proved20	
20		attacked10
21		area40
24	area40	
27		proved40
32	proved40	
37		inspected10
38	attacked10	
49		attacked20
50	attacked20	
61	proved80	proved80
64	surveyed640	
81	inspected20	
84		surveyed320
86	attacked40	
110		attacked40
118	proved160	
140	attacked80	
182	area80	
228	area160, area320	
249		proved160
266	area640	
303	attacked160	
743	attacked320	

Figure 194: Achievements.

19.2 Stability

Reason	AiWYX	%	Streett	%
failed away			1	0,01
failed random	153	1,02	155	1,03
failed resources			302	2,01
failed attacked	24	0,16	56	0,37
failed status			4	0,03

Figure 195: Failed actions.

19.3 Achievements

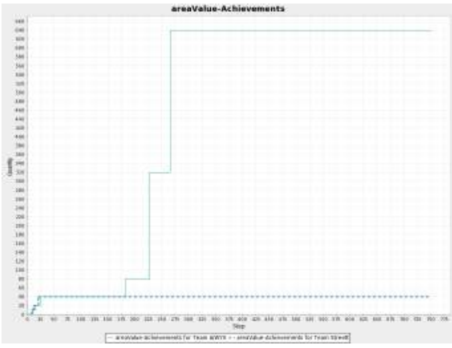


Figure 196: areaValueAchievements.

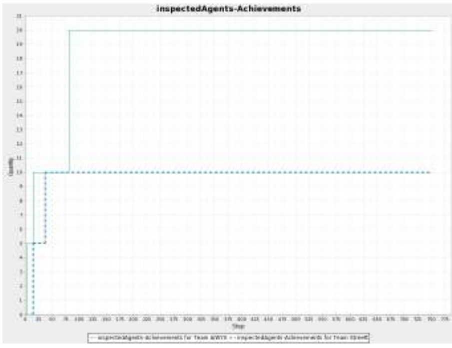


Figure 197: inspectedAgentsAchievements.

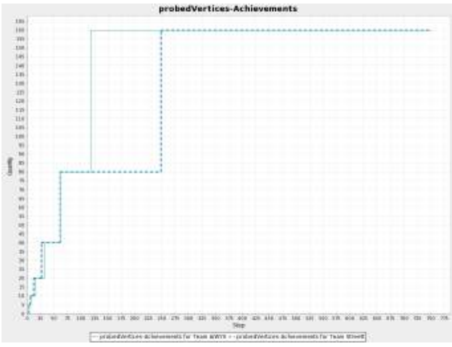


Figure 198: probedVerticesAchievements.

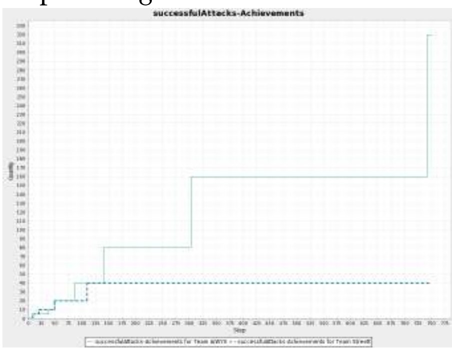


Figure 199: successfulAttacksAchievements.

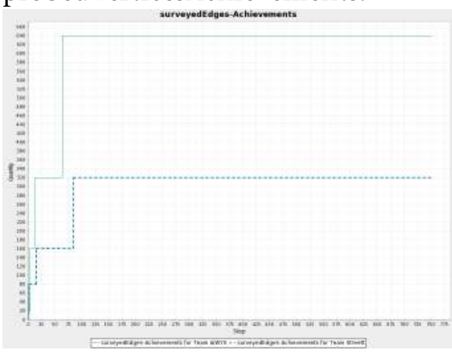


Figure 200: surveyedEdgesAchievements.

19.4 Actions per Role

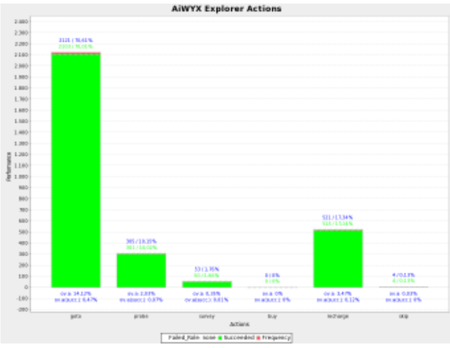


Figure 201: AiWYX vs. Streett – Simulation 1 - AiWYX Explorer Actions.

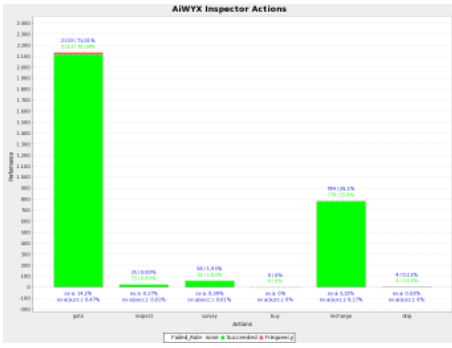


Figure 202: AiWYX vs. Streett – Simulation 1 - AiWYX Inspector Actions.

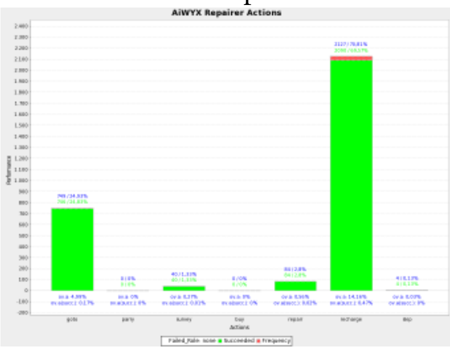


Figure 203: AiWYX vs. Streett – Simulation 1 - AiWYX Repairer Actions.

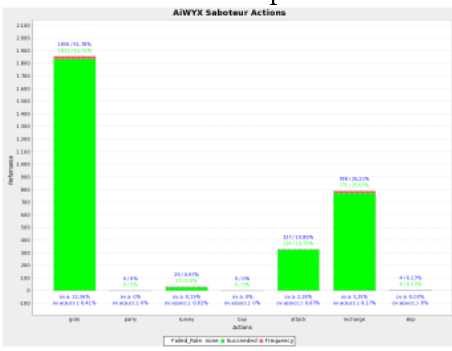


Figure 204: AiWYX vs. Streett – Simulation 1 - AiWYX Saboteur Actions.

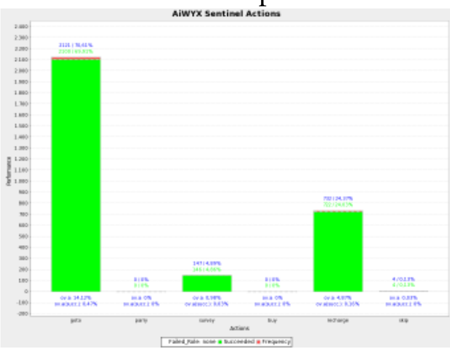


Figure 205: AiWYX vs. Streett – Simulation 1 - AiWYX Sentinel Actions.

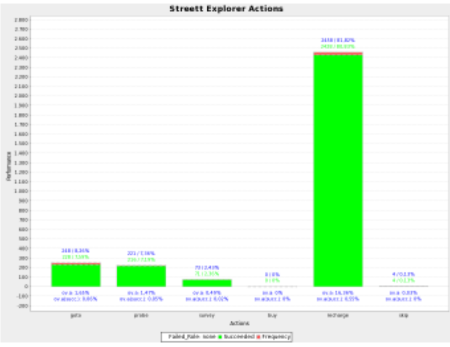


Figure 206: AiWYX vs. Streett – Simulation 1 - Streett Explorer Actions.

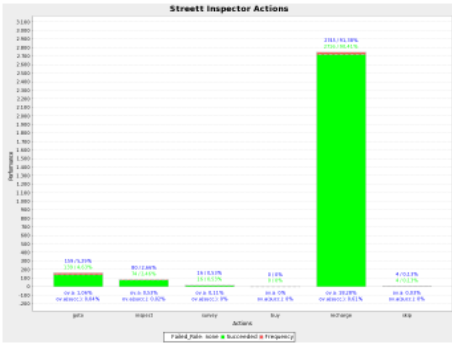


Figure 207: AiWYX vs. Streett – Simulation 1 - Streett Inspector Actions.

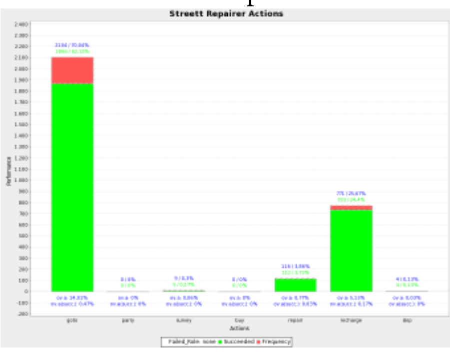


Figure 208: AiWYX vs. Streett – Simulation 1 - Streett Repairer Actions.

Figure 209: AiWYX vs. Streett – Simulation 1 - Streett Saboteur Actions.

Figure 210: AiWYX vs. Streett – Simulation 1 - Streett Sentinel Actions.

20 AiWYX vs. Streett – Simulation 2

20.1 Scores, Zone Stability and Achievements

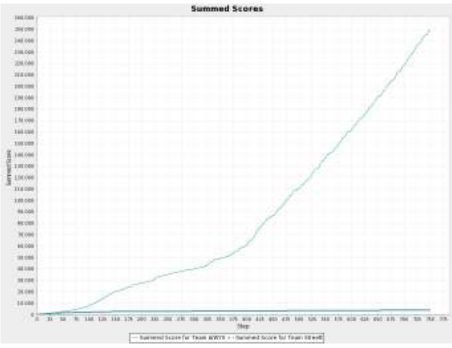


Figure 211: Summed scores.

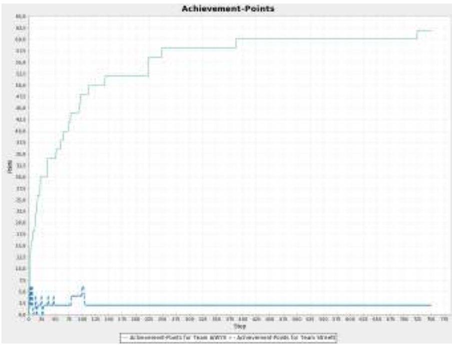


Figure 212: Achievement points.

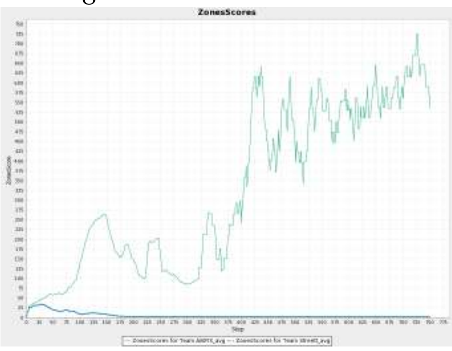


Figure 213: Zones scores.

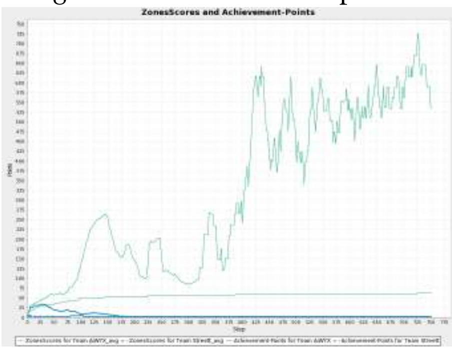


Figure 214: Zones scores and achievement points.

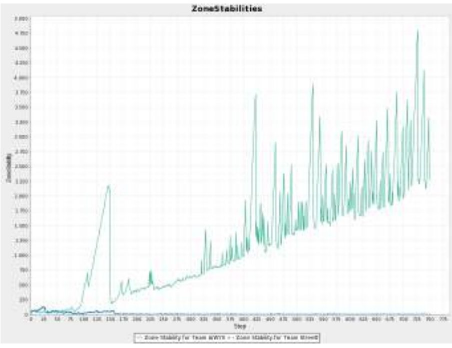


Figure 215: Zone Stabilities.

Step	AiWYX	Streett
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed20
2	surveyed80, inspected5	surveyed40, surveyed80, inspected5
3	proved5	
4	surveyed160	area10, proved5
6		area20, proved10
7	proved10	
11	attacked5	surveyed160
12	surveyed320	
13		attacked5
15	proved20	
16	attacked10	proved20
21	inspected10, area20	
23		inspected10
27		attacked10
35	proved40, attacked20	
36		proved40
46		attacked20
51	inspected20	
60	proved80	
65	attacked40	
74	surveyed640	
78	area40	
79		surveyed320
94	area80	
96	area160	
99		proved80
112	proved160	
142	attacked80	
223	area320, area640	
247	attacked160	
386	attacked320	
724	attacked640	

Figure 216: Achievements.

20.2 Stability

Reason	AiWYX	%	Streett	%
failed away			5	0,03
failed random	131	0,87	152	1,01
failed resources			317	2,11
failed attacked	6	0,04	82	0,55

Figure 217: Failed actions.

20.3 Achievements

Figure 218: areaValueAchievements.

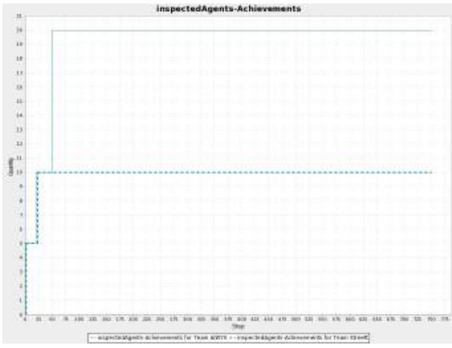


Figure 219: inspectedAgentsAchievements.

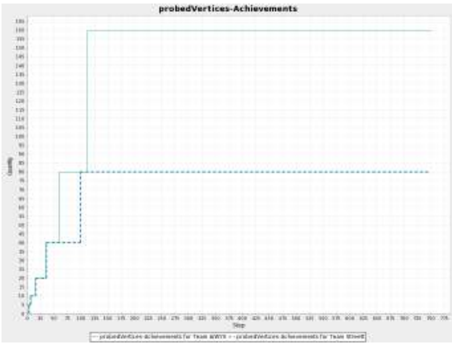


Figure 220: probedVerticesAchievements.

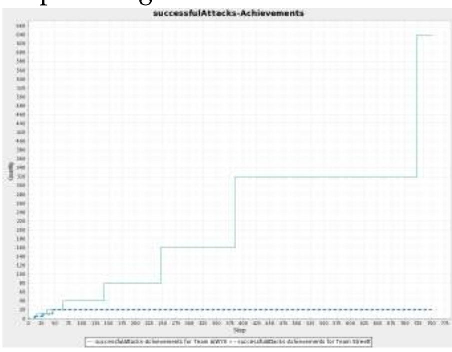


Figure 221: successfulAttacksAchievements.

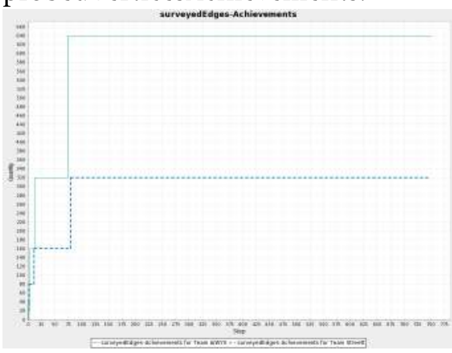


Figure 222: surveyedEdgesAchievements.

20.4 Actions per Role

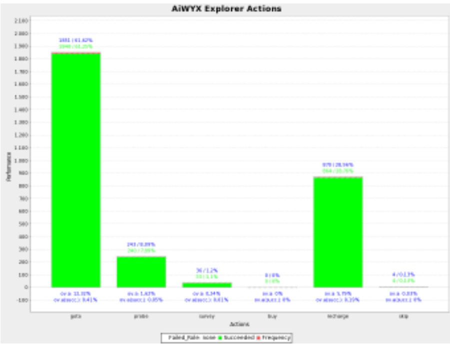


Figure 223: AiWYX vs. Streett – Simulation 2 - AiWYX Explorer Actions.

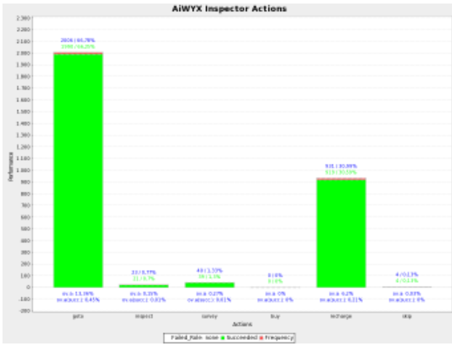


Figure 224: AiWYX vs. Streett – Simulation 2 - AiWYX Inspector Actions.

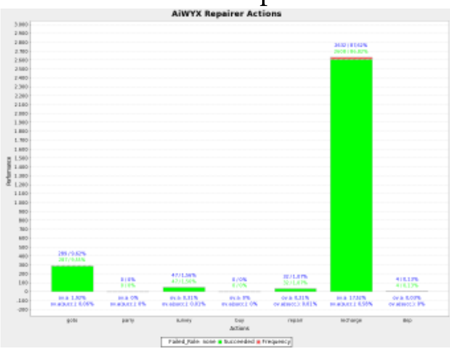


Figure 225: AiWYX vs. Streett – Simulation 2 - AiWYX Repairer Actions.

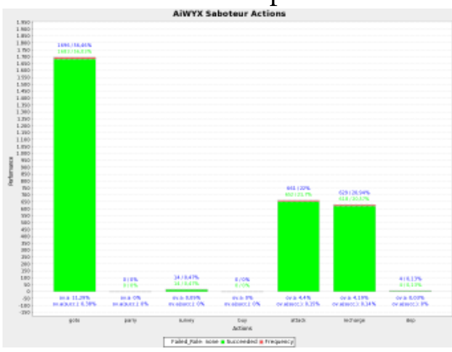


Figure 226: AiWYX vs. Streett – Simulation 2 - AiWYX Saboteur Actions.

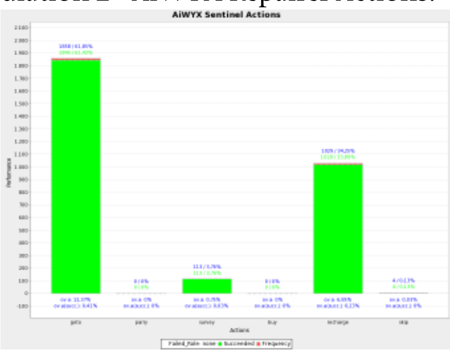


Figure 227: AiWYX vs. Streett – Simulation 2 - AiWYX Sentinel Actions.

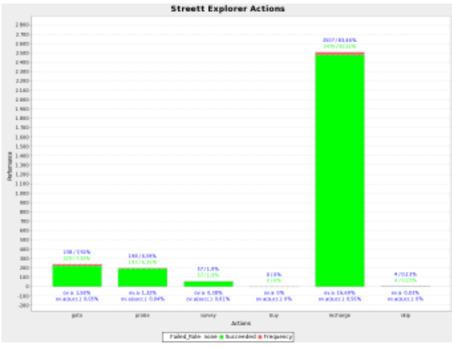


Figure 228: AiWYX vs. Streett – Simulation 2 - Streett Explorer Actions.

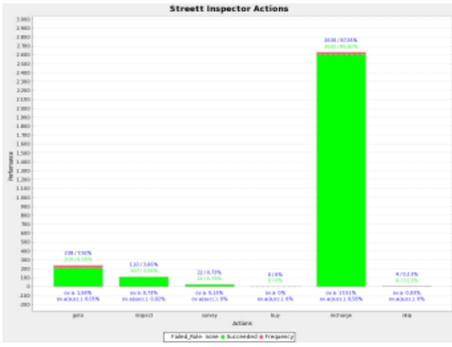


Figure 229: AiWYX vs. Streett – Simulation 2 - Streett Inspector Actions.

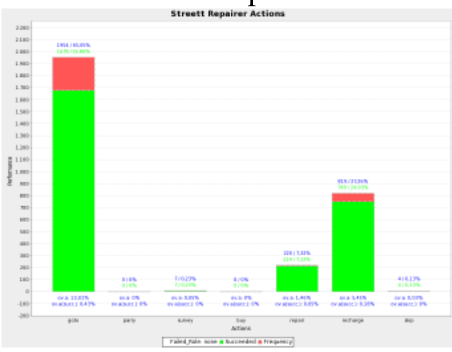


Figure 230: AiWYX vs. Streett – Simulation 2 - Streett Repairer Actions.

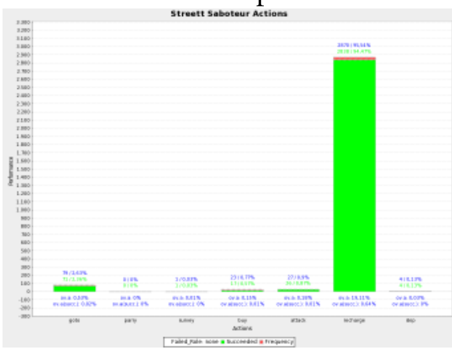


Figure 231: AiWYX vs. Streett – Simulation 2 - Streett Saboteur Actions.

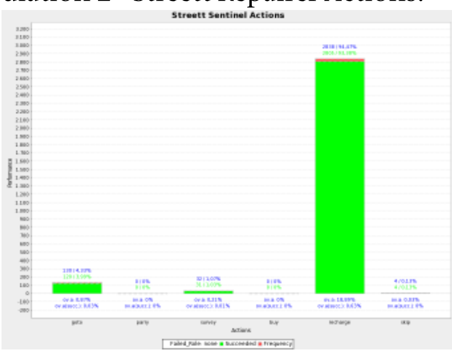


Figure 232: AiWYX vs. Streett – Simulation 2 - Streett Sentinel Actions.

21 AiWYX vs. Streett – Simulation 3

21.1 Scores, Zone Stability and Achievements

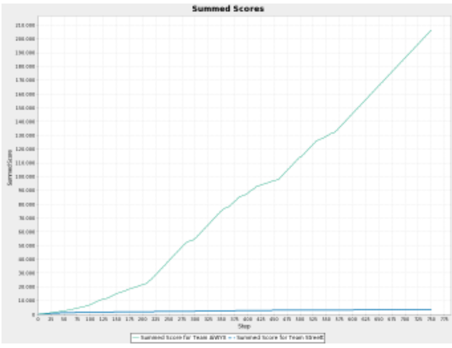


Figure 233: Summed scores.

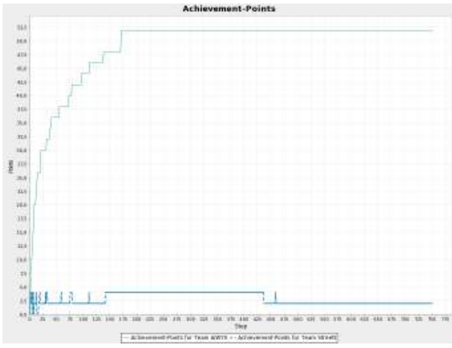


Figure 234: Achievement points.

Figure 235: Zones scores.

Figure 236: Zones scores and achievement points.

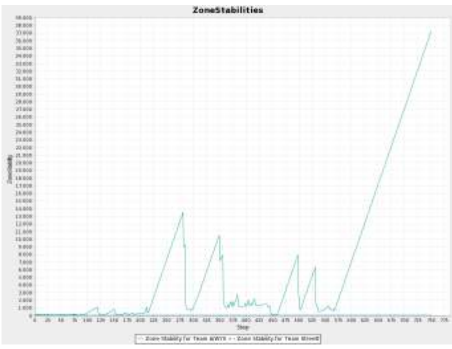


Figure 237: Zone Stabilities.

Step	AiWYX	Streett
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed20
2	surveyed80	surveyed40, surveyed80
3		area10
4	proved5	proved5
5	surveyed160	
6	inspected5	area20, proved10
7	proved10, attacked5	
8		inspected5
11	area20	
12		proved20
13	inspected10	
14	proved20	
16		surveyed160
18		attacked5
20	surveyed320, attacked10	
29		proved40
31	proved40	inspected10
32		attacked10
37	attacked20	
40	inspected20	
55	proved80	
58		attacked20
72	area40	
75		proved80
78	attacked40	
96	area80	
110		surveyed320
111	proved160	
137	attacked80	
141		inspected20
170	area160, area320	
458		attacked40

Figure 238: Achievements.

21.2 Stability

Reason	AiWYX	%	Streett	%
failed away			6	0,04
failed random	143	0,95	139	0,93
failed resources			276	1,84
failed attacked	5	0,03	51	0,34

Figure 239: Failed actions.

21.3 Achievements

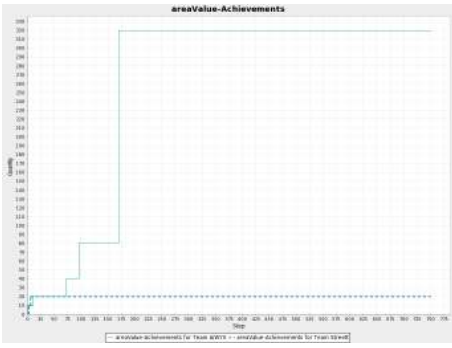


Figure 240: areaValueAchievements.

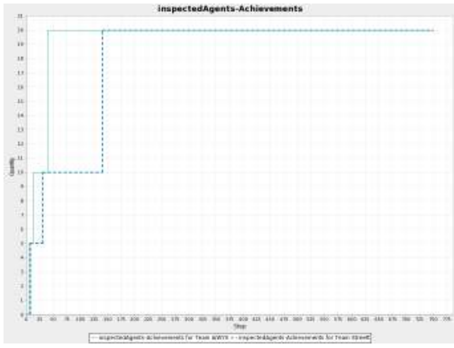


Figure 241: inspectedAgentsAchievements.

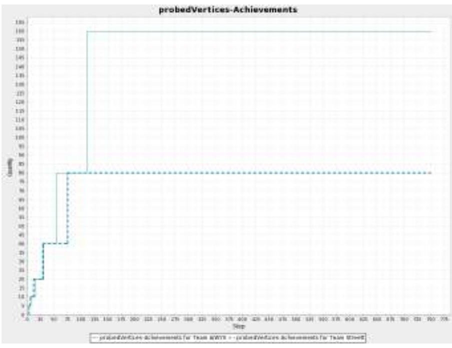


Figure 242: probedVerticesAchievements.

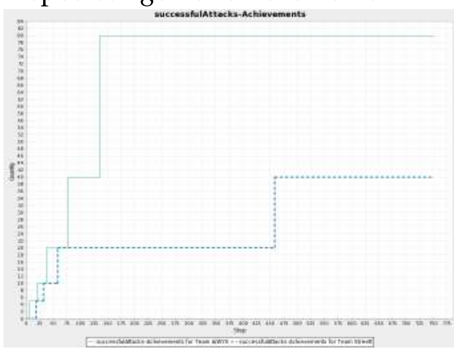


Figure 243: successfulAttacksAchievements.

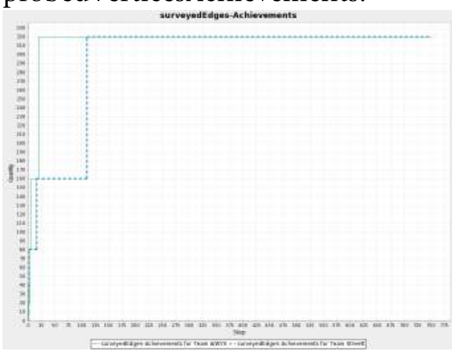


Figure 244: surveyedEdgesAchievements.

21.4 Actions per Role

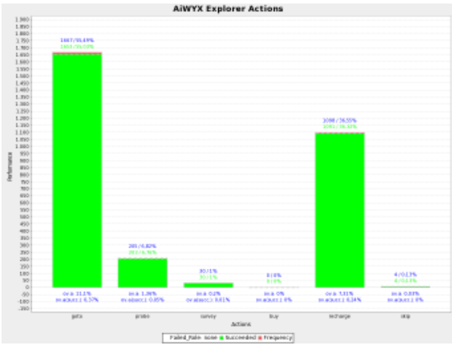


Figure 245: AiWYX vs. Streett – Simulation 3 - AiWYX Explorer Actions.

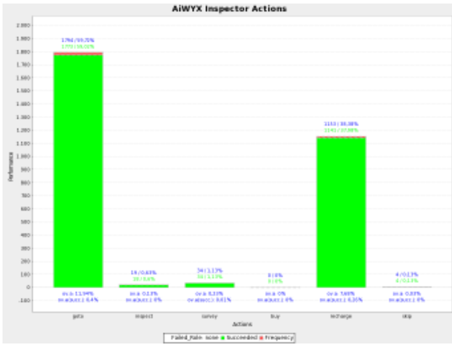


Figure 246: AiWYX vs. Streett – Simulation 3 - AiWYX Inspector Actions.

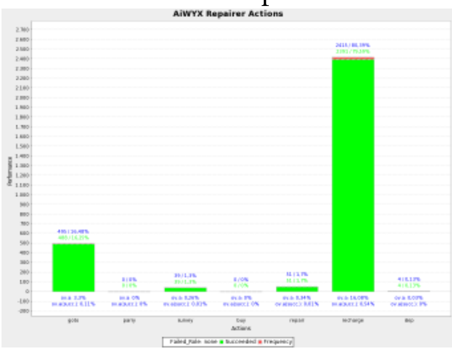


Figure 247: AiWYX vs. Streett – Simulation 3 - AiWYX Repairer Actions.

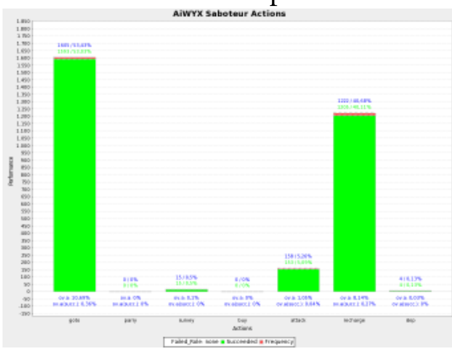


Figure 248: AiWYX vs. Streett – Simulation 3 - AiWYX Saboteur Actions.

Figure 249: AiWYX vs. Streett – Simulation 3 - AiWYX Sentinel Actions.

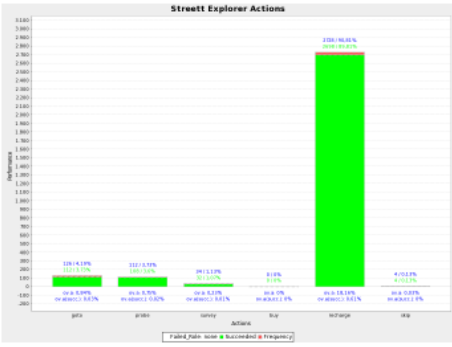


Figure 250: AiWYX vs. Streett – Simulation 3 - Streett Explorer Actions.

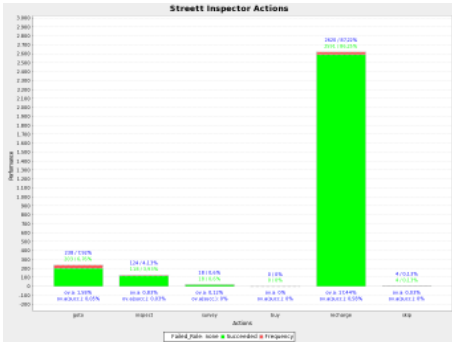


Figure 251: AiWYX vs. Streett – Simulation 3 - Streett Inspector Actions.

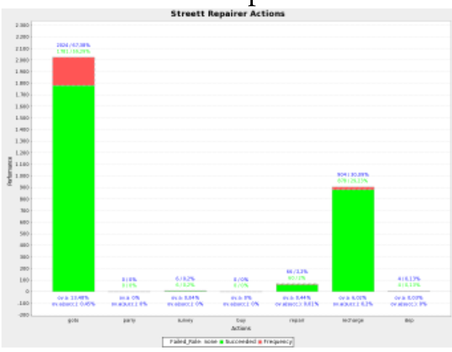


Figure 252: AiWYX vs. Streett – Simulation 3 - Streett Repairer Actions.

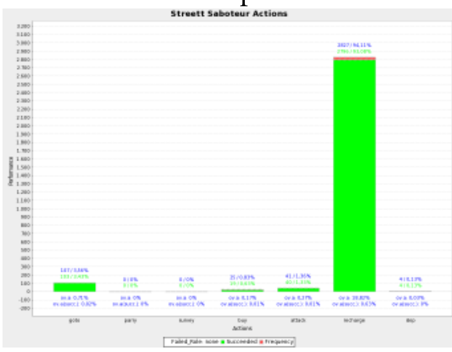


Figure 253: AiWYX vs. Streett – Simulation 3 - Streett Saboteur Actions.

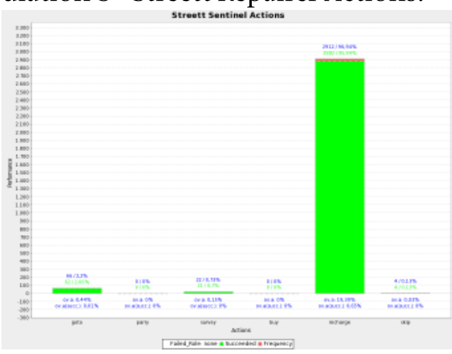


Figure 254: AiWYX vs. Streett – Simulation 3 - Streett Sentinel Actions.

22 AiWYX vs. TUB – Simulation 1

22.1 Scores, Zone Stability and Achievements

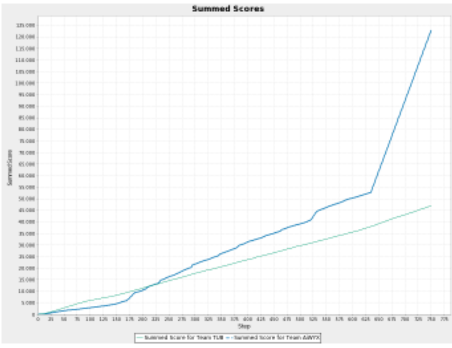


Figure 255: Summed scores.

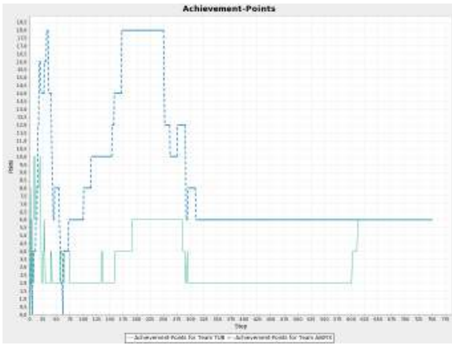


Figure 256: Achievement points.

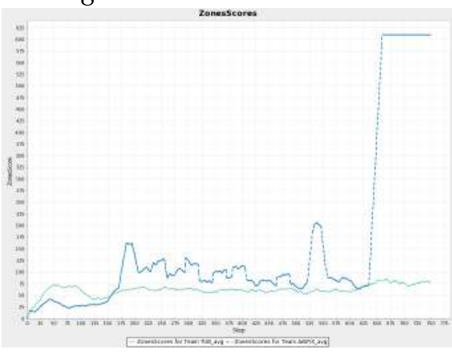


Figure 257: Zones scores.

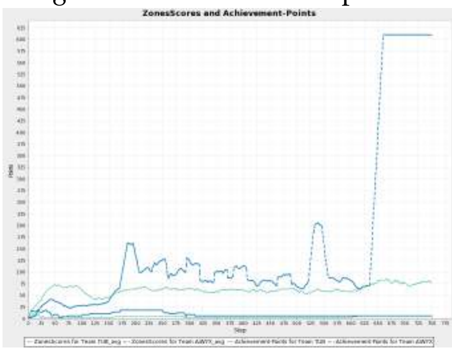


Figure 258: Zones scores and achievement points.

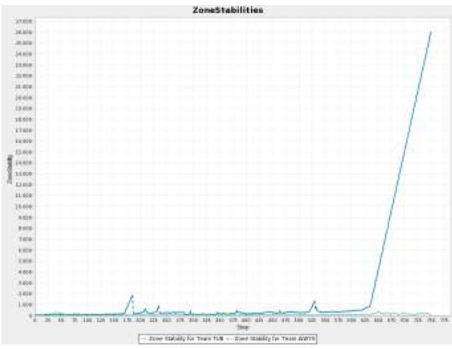


Figure 259: Zone Stabilities.

Step	TUB	AiWYX
1	surveyed10, surveyed80, surveyed40, surveyed20	surveyed10, surveyed40, surveyed20
2		surveyed80
3		proved5
5	attacked5, proved5	surveyed160, attacked5
7	area10	proved10
8	area20, inspected5	
9	attacked10	
11	proved10	inspected5
12	area40	
14		inspected10, attacked10
15		proved20
17		surveyed320, area10
18	attacked20	
25	inspected10	
26	proved20	
27	surveyed160, area80	attacked20
31		proved40
39	attacked40	
45		area20
46		attacked40
57	proved40	inspected20
62		surveyed640, proved80
65	attacked80	
72		attacked80
100		attacked160
114		proved160
134	attacked160	
153		area40
157		attacked320
159	surveyed320	
171		area160, area80
190	proved80	
275		attacked640
293	attacked320	
294		area320
601	proved160	
611	attacked640	

Figure 260: Achievements.

22.2 Stability

Reason	TUB	%	AiWYX	%
failed away	1	0,01		
failed random	142	0,95	152	1,01
failed resources			1	0,01
failed attacked	11	0,07	38	0,25

Figure 261: Failed actions.

22.3 Achievements

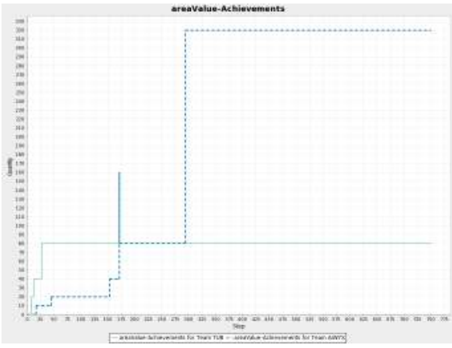


Figure 262: areaValueAchievements.

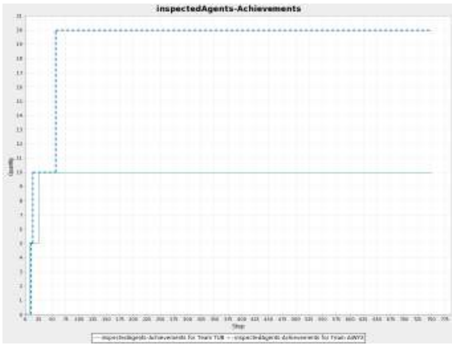


Figure 263: inspectedAgentsAchievements.

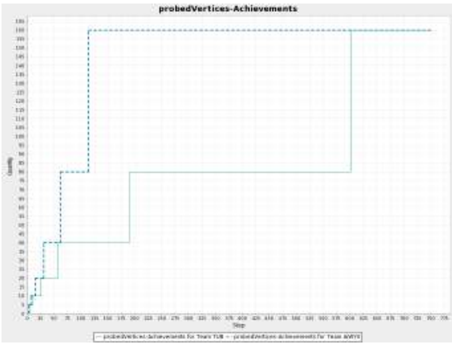


Figure 264: probedVerticesAchievements.

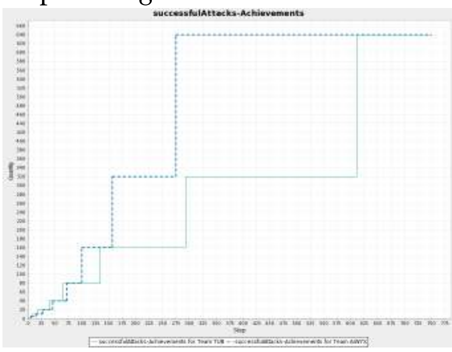


Figure 265: successfulAttacksAchievements.

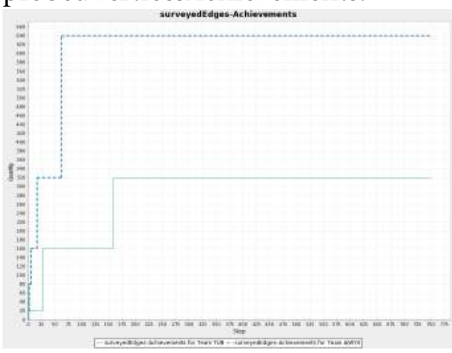


Figure 266: surveyedEdgesAchievements.

22.4 Actions per Role

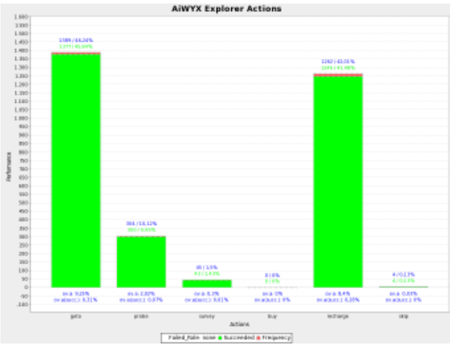


Figure 267: AiWYX vs. TUB – Simulation 1 - AiWYX Explorer Actions.

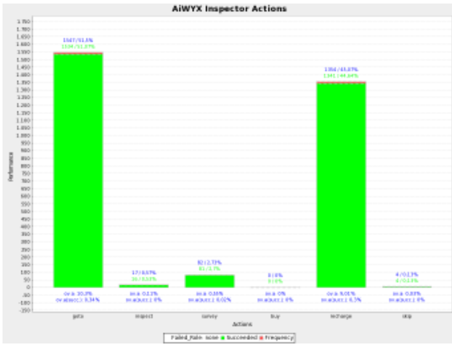


Figure 268: AiWYX vs. TUB – Simulation 1 - AiWYX Inspector Actions.

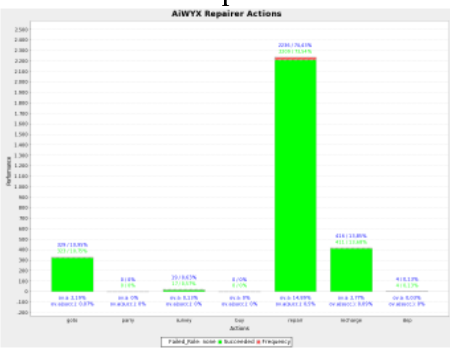


Figure 269: AiWYX vs. TUB – Simulation 1 - AiWYX Repairer Actions.

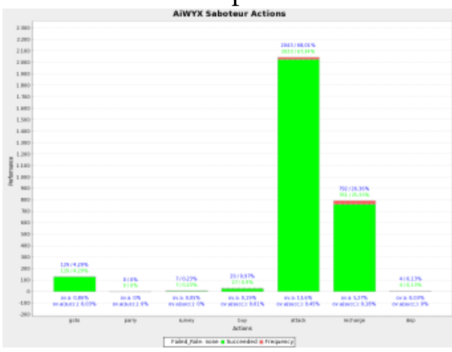


Figure 270: AiWYX vs. TUB – Simulation 1 - AiWYX Saboteur Actions.

Figure 271: AiWYX vs. TUB – Simulation 1 - AiWYX Sentinel Actions.

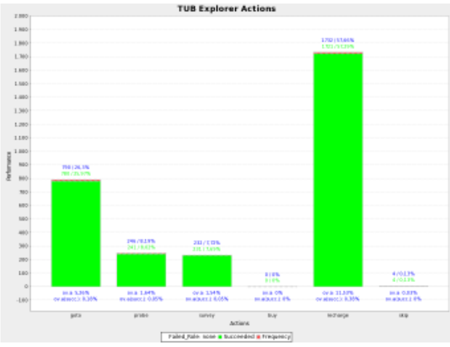


Figure 272: AiWYX vs. TUB – Simulation 1 - TUB Explorer Actions.

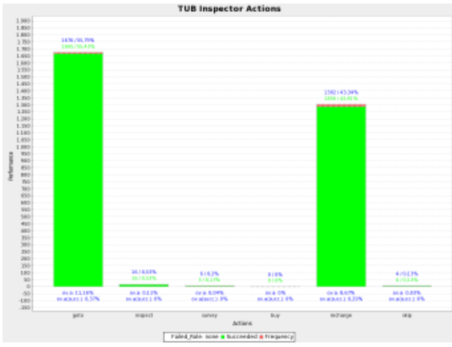


Figure 273: AiWYX vs. TUB – Simulation 1 - TUB Inspector Actions.

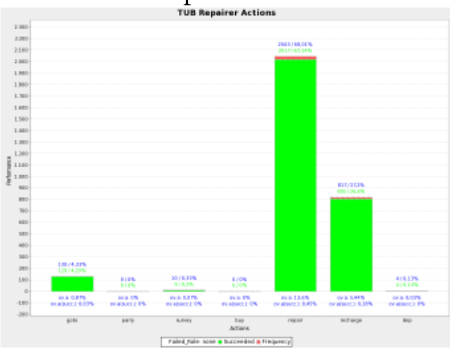


Figure 274: AiWYX vs. TUB – Simulation 1 - TUB Repairer Actions.

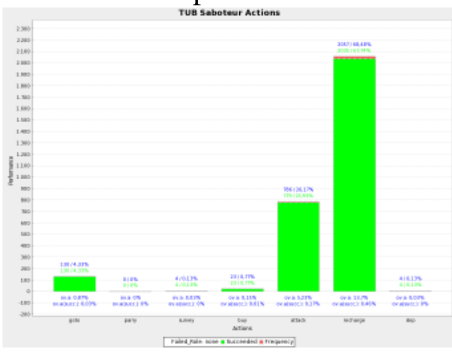


Figure 275: AiWYX vs. TUB – Simulation 1 - TUB Saboteur Actions.

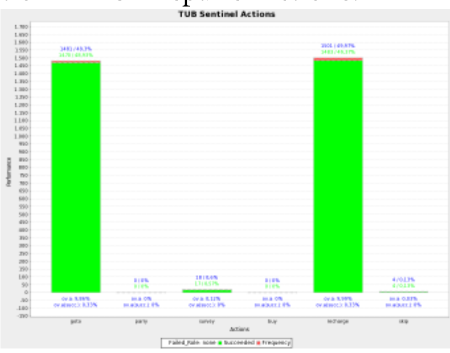


Figure 276: AiWYX vs. TUB – Simulation 1 - TUB Sentinel Actions.

23 AiWYX vs. TUB – Simulation 2

23.1 Scores, Zone Stability and Achievements

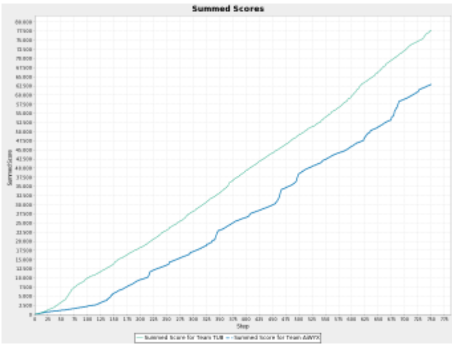


Figure 277: Summed scores.

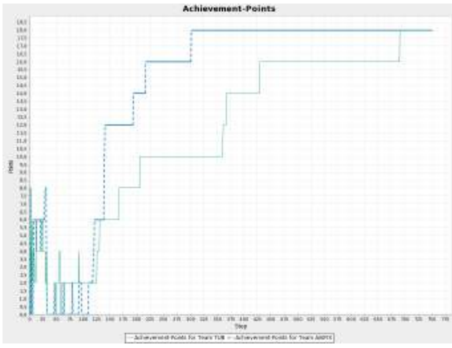


Figure 278: Achievement points.

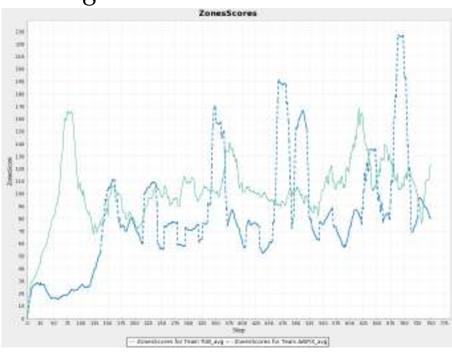


Figure 279: Zones scores.

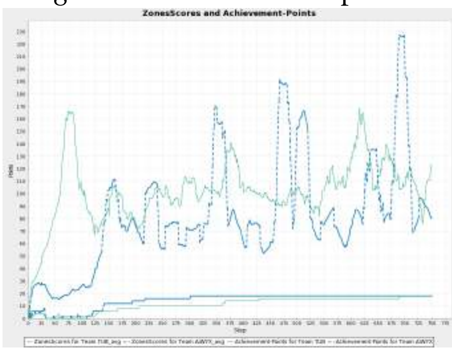


Figure 280: Zones scores and achievement points.

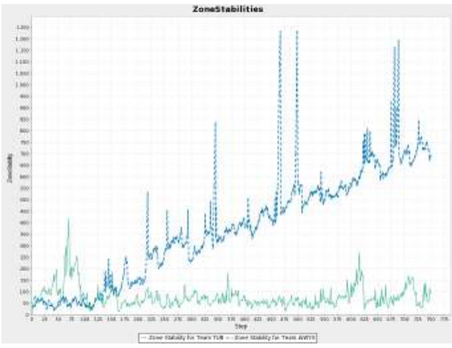


Figure 281: Zone Stabilities.

Step	TUB	AiWYX
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
3	surveyed80	surveyed80, proved5
4	area20, proved5	inspected5
6		surveyed160
7	inspected5	area20, proved10
8	attacked5	
11		attacked5
12	area40	inspected10
13		proved20
14	proved10	
21	attacked10	
22		surveyed320
23	inspected10	
25	proved20	
28		proved40
29	area80	
30		attacked10
31	surveyed160	
46		attacked20
50	attacked20	
55	proved40	
60	area160	
63		proved80
79		inspected20
91	attacked40	attacked40
109		attacked80
118		area40
120		proved160
125	attacked80	
130	proved80	
138		area160, area80
140		attacked160
166	surveyed320	
193		attacked320
205	attacked160	
216		area320
301		attacked640
360	inspected20	
366	attacked320	
429	proved160	
690	attacked640	

Figure 282: Achievements.

23.2 Stability

Reason	TUB	%	AiWYX	%
failed away	2	0,01		
failed wrong param	238	1,59		
failed random	126	0,84	144	0,96
failed resources	1	0,01	2	0,01
failed	124	0,83		
failed attacked	15	0,1	18	0,12
noAction	125	0,83		

Figure 283: Failed actions.

23.3 Achievements

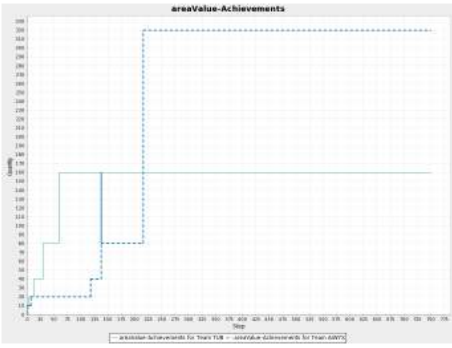


Figure 284: areaValueAchievements.

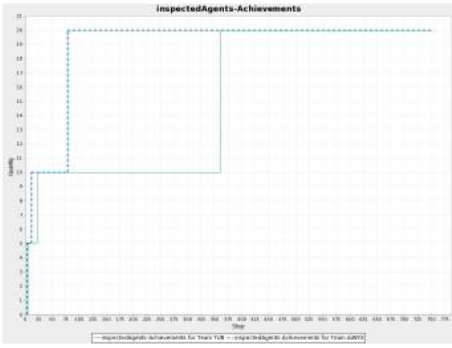


Figure 285: inspectedAgentsAchievements.

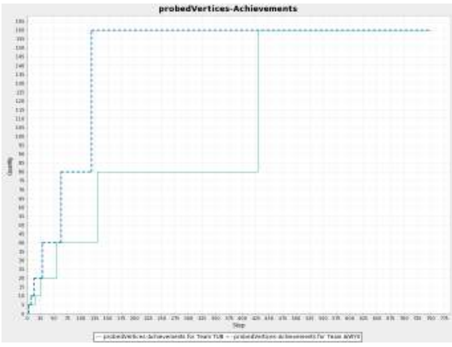


Figure 286: probedVerticesAchievements.

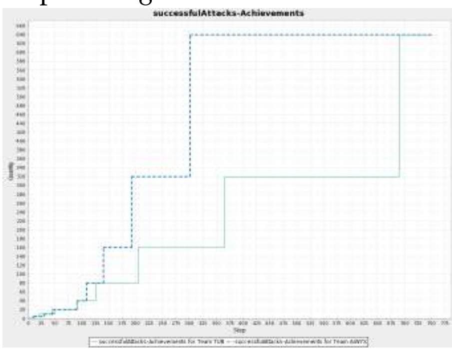


Figure 287: successfulAttacksAchievements.

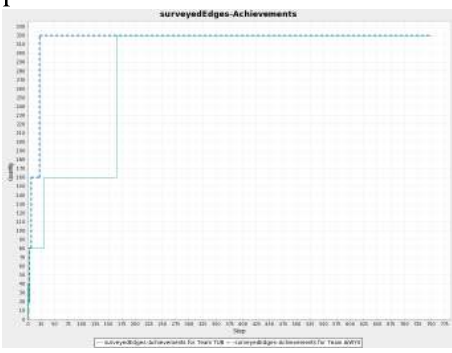


Figure 288: surveyedEdgesAchievements.

23.4 Actions per Role

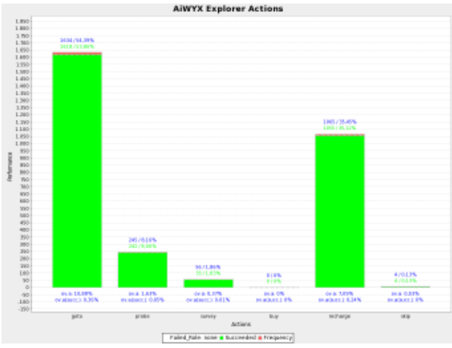


Figure 289: AiWYX vs. TUB – Simulation 2 - AiWYX Explorer Actions.

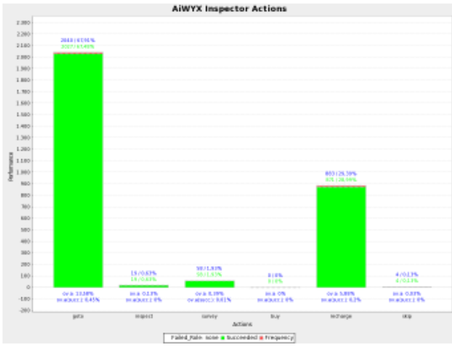


Figure 290: AiWYX vs. TUB – Simulation 2 - AiWYX Inspector Actions.

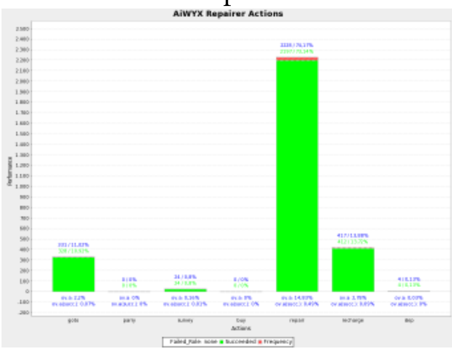


Figure 291: AiWYX vs. TUB – Simulation 2 - AiWYX Repairer Actions.

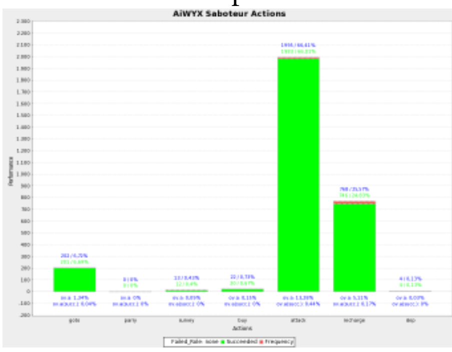


Figure 292: AiWYX vs. TUB – Simulation 2 - AiWYX Saboteur Actions.

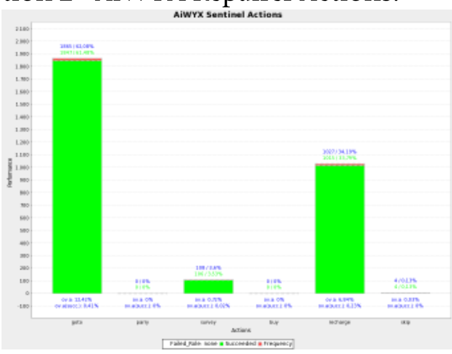


Figure 293: AiWYX vs. TUB – Simulation 2 - AiWYX Sentinel Actions.

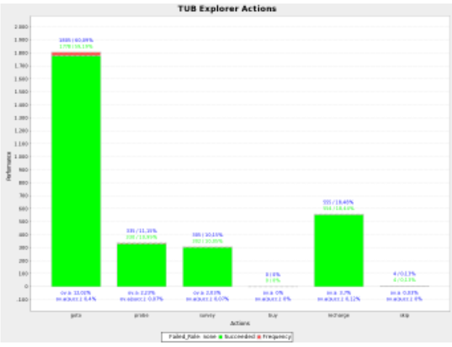


Figure 294: AiWYX vs. TUB – Simulation 2 - TUB Explorer Actions.

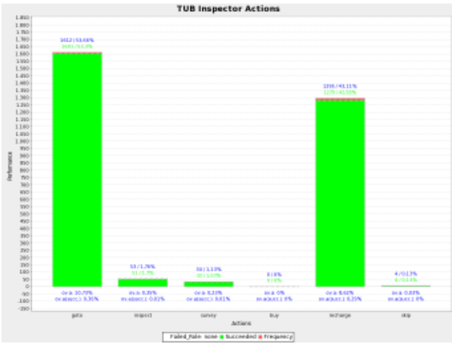


Figure 295: AiWYX vs. TUB – Simulation 2 - TUB Inspector Actions.

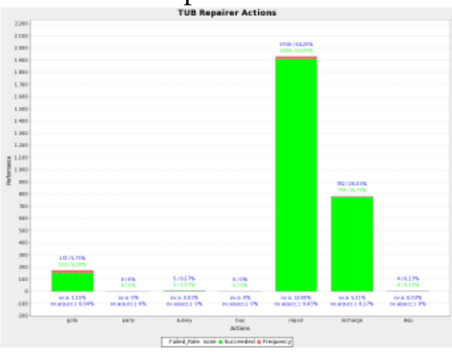


Figure 296: AiWYX vs. TUB – Simulation 2 - TUB Repairer Actions.

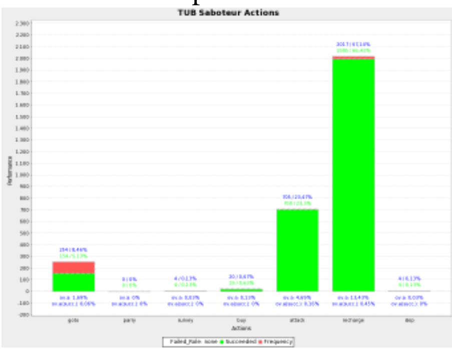


Figure 297: AiWYX vs. TUB – Simulation 2 - TUB Saboteur Actions.

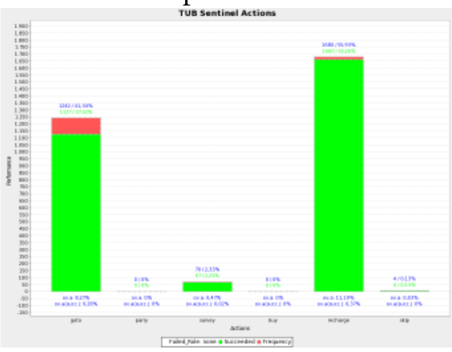


Figure 298: AiWYX vs. TUB – Simulation 2 - TUB Sentinel Actions.

24 AiWYX vs. TUB – Simulation 3

24.1 Scores, Zone Stability and Achievements

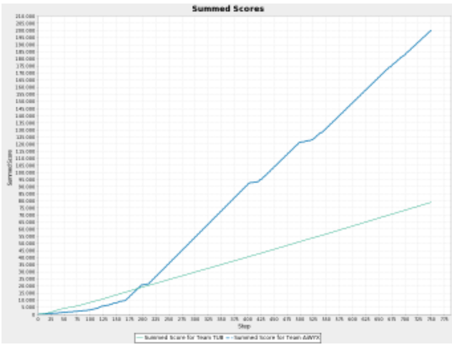


Figure 299: Summed scores.

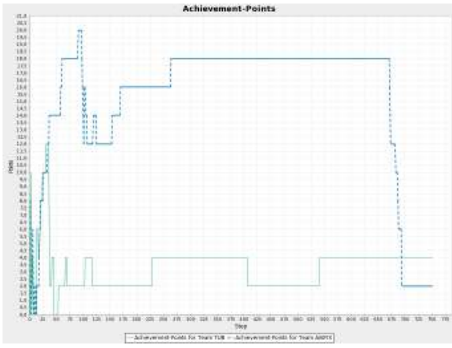


Figure 300: Achievement points.

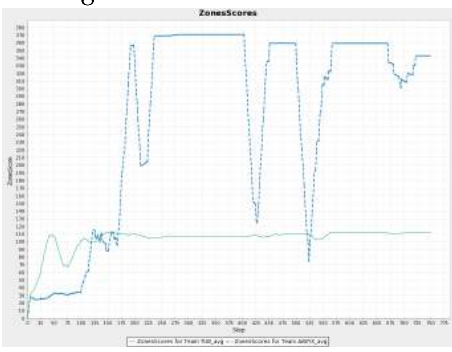


Figure 301: Zones scores.

Figure 302: Zones scores and achievement points.

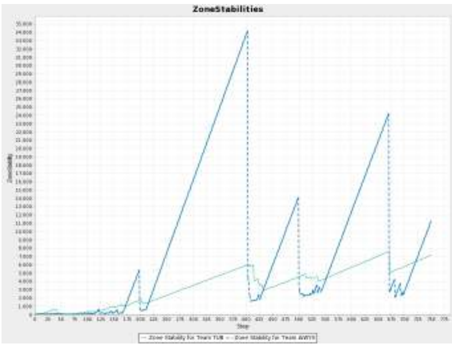


Figure 303: Zone Stabilities.

Step	TUB	AiWYX
1	surveyed40, surveyed10, surveyed20, area10, surveyed80	surveyed10, surveyed40, area10, surveyed20
2	area20	surveyed80
4	proved5, inspected5	proved5, inspected5
5		area20, surveyed160
7		proved10
10		attacked5
12	area40, proved10, attacked5	
13		inspected10
15	inspected10	
17		proved20
18	attacked10	attacked10
19		surveyed320
20	area80	
23	proved20	
24	surveyed160	
25		attacked20
29	attacked20	
33		proved40
36		attacked40
41	attacked40	
53	proved40	
57		attacked80
60		proved80
66	attacked80	
90		attacked160
101		area40, area160, area80
103	attacked160	
118		proved160
153		attacked320
169		area320
228	attacked320	
263		attacked640
540	attacked640	

Figure 304: Achievements.

24.2 Stability

Reason	TUB	%	AiWYX	%
failed away	2	0,01		
failed random	164	1,09	145	0,97
failed resources	3	0,02	4	0,03
failed attacked	11	0,07	74	0,49

Figure 305: Failed actions.

24.3 Achievements

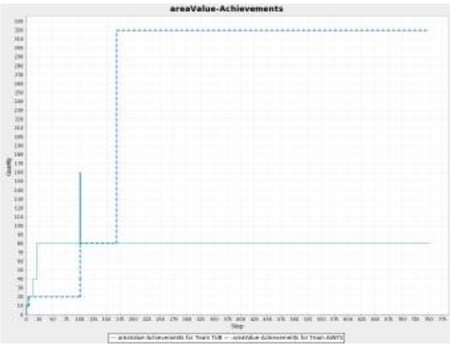


Figure 306: areaValueAchievements.

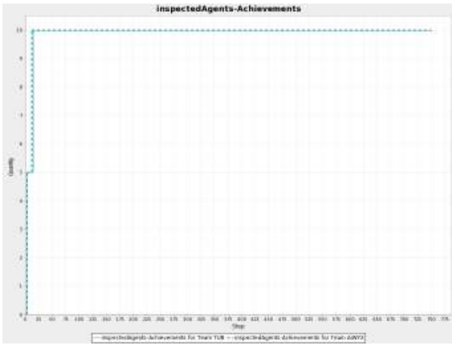


Figure 307: inspectedAgentsAchievements.

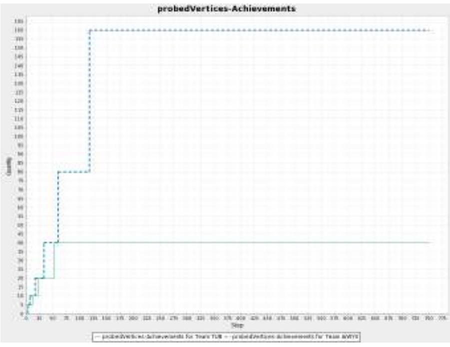


Figure 308: probedVerticesAchievements.

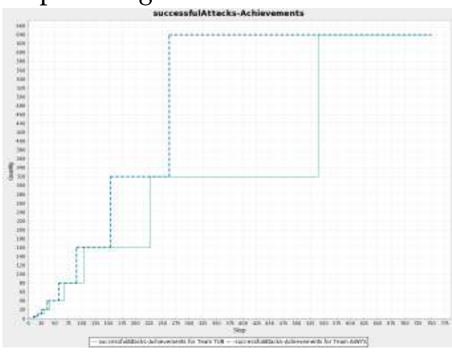


Figure 309: successfulAttacksAchievements.

Figure 310: surveyedEdgesAchievements.

24.4 Actions per Role

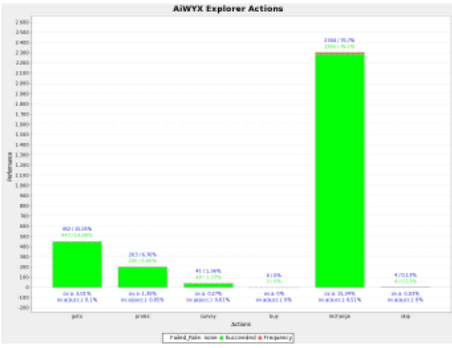


Figure 311: AiWYX vs. TUB – Simulation 3 - AiWYX Explorer Actions.

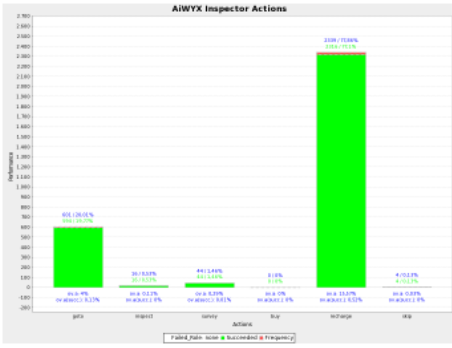


Figure 312: AiWYX vs. TUB – Simulation 3 - AiWYX Inspector Actions.

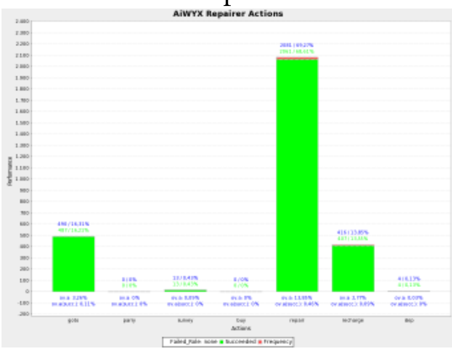


Figure 313: AiWYX vs. TUB – Simulation 3 - AiWYX Repairer Actions.

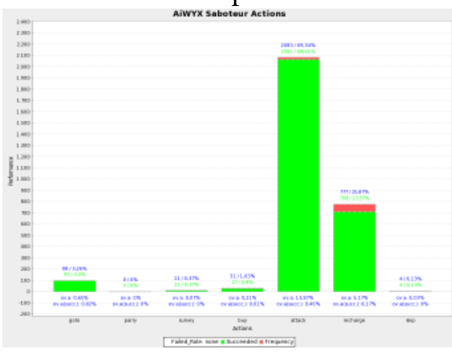


Figure 314: AiWYX vs. TUB – Simulation 3 - AiWYX Saboteur Actions.

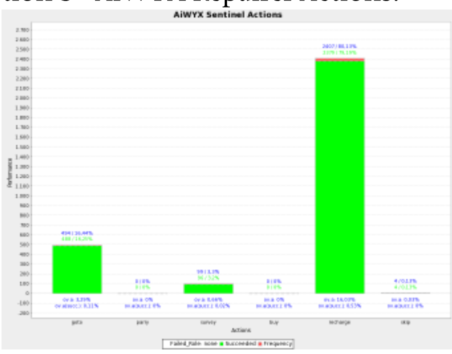


Figure 315: AiWYX vs. TUB – Simulation 3 - AiWYX Sentinel Actions.

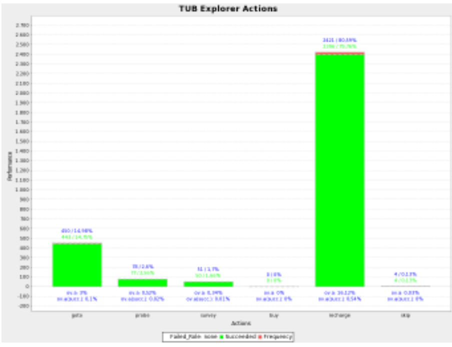


Figure 316: AiWYX vs. TUB – Simulation 3 - TUB Explorer Actions.

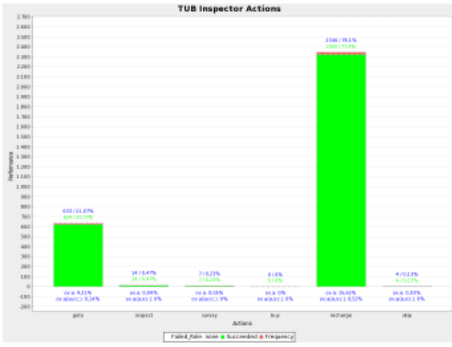


Figure 317: AiWYX vs. TUB – Simulation 3 - TUB Inspector Actions.

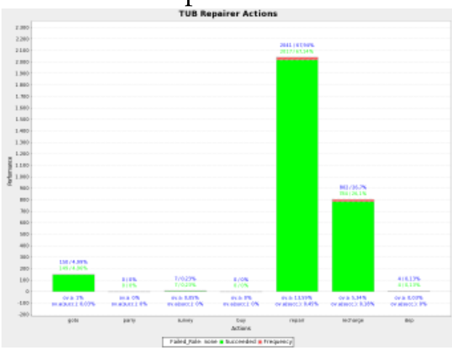


Figure 318: AiWYX vs. TUB – Simulation 3 - TUB Repairer Actions.

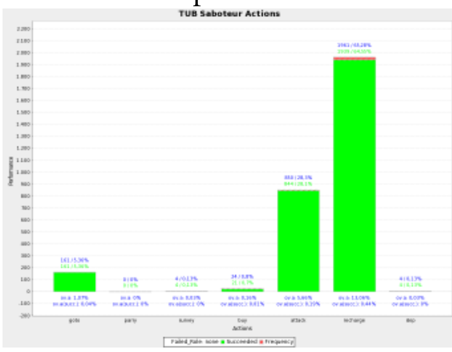


Figure 319: AiWYX vs. TUB – Simulation 3 - TUB Saboteur Actions.

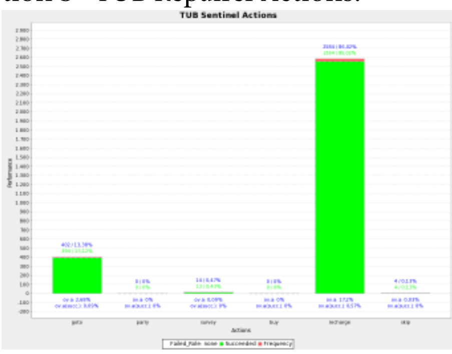


Figure 320: AiWYX vs. TUB – Simulation 3 - TUB Sentinel Actions.

25 AiWYX vs. UFSC – Simulation 1

25.1 Scores, Zone Stability and Achievements

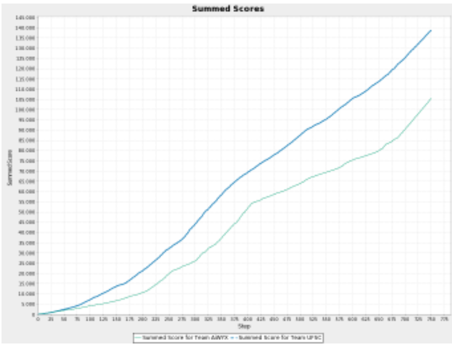


Figure 321: Summed scores.

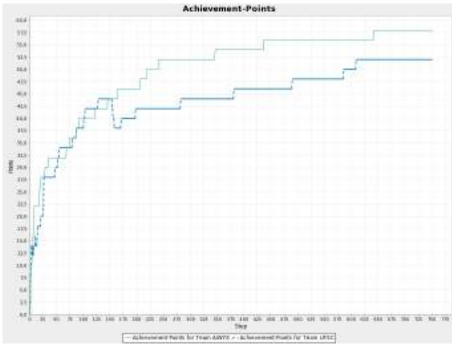


Figure 322: Achievement points.

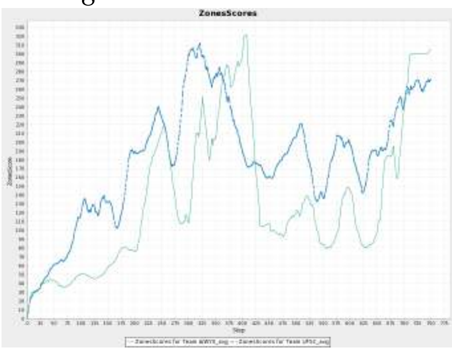


Figure 323: Zones scores.

Figure 324: Zones scores and achievement points.

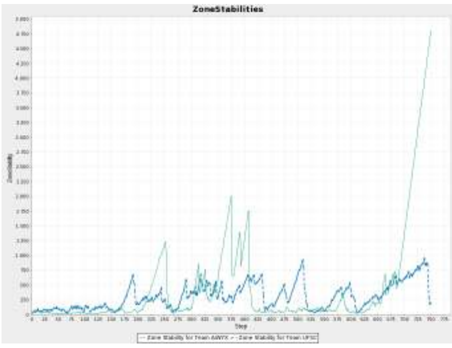


Figure 325: Zone Stabilities.

Step	AiWYX	UFSC
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
2	surveyed80	attacked5
3		surveyed80, proved5
4	proved5	
5	area20, surveyed160	proved10
7	proved10	surveyed160
8	attacked5, inspected5	
9		inspected5
10		area20
14		proved20, attacked10
17	inspected10	
18	surveyed320	
20	proved20	surveyed320
25		proved40
26		inspected10, attacked20, parried5
28	attacked10	
35	proved40	
47		area40
52		proved80
55		attacked40
68	attacked20	
74	proved80	
79		attacked80
86	inspected20	area80
91	surveyed640	
100		parried10
103		proved160
122	attacked40	
127		parried20
145	proved160	
164	area40	
170		attacked160
197		surveyed640
206	area80	
218	attacked80	
240	area160	
281		area160
345	attacked160	
380		attacked320
436	attacked320	
489		parried40
585		inspected20
608		attacked640
641	attacked640	

Figure 326: Achievements.

25.2 Stability

Reason	AiWYX	%	UFSC	%
failed away	4	0,03	1	0,01
failed parried	63	0,42		
failed random	151	1,01	162	1,08
failed resources			3	0,02
failed			192	1,28
failed attacked	118	0,79	40	0,27
noAction			194	1,29

Figure 327: Failed actions.

25.3 Achievements

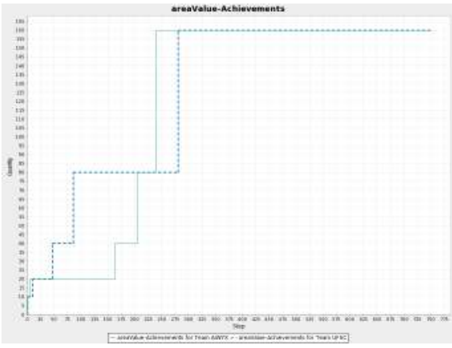


Figure 328: areaValueAchievements.

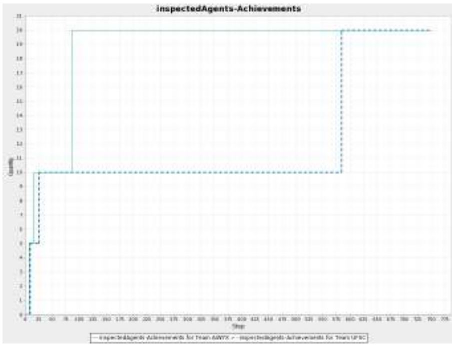


Figure 329: inspectedAgentsAchievements.

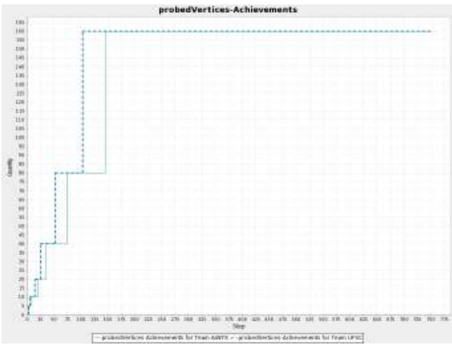


Figure 330: probedVerticesAchievements.

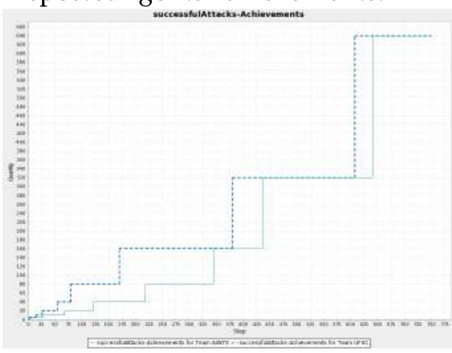


Figure 331: successfulAttacksAchievements.

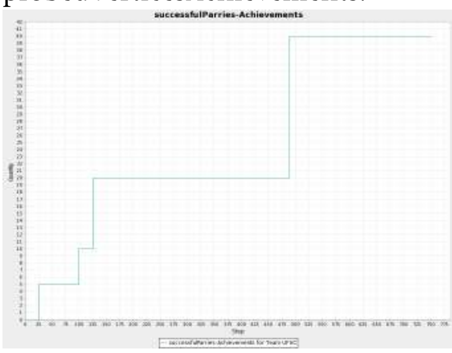


Figure 332: successfulParriesAchievements.

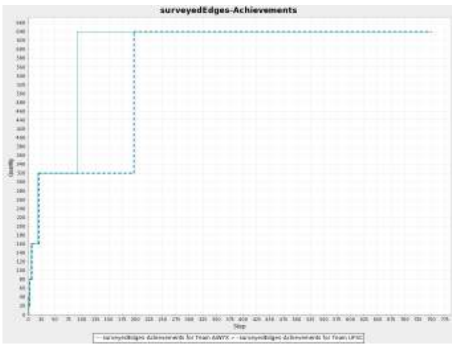


Figure 333: surveyedEdgesAchievements.

25.4 Actions per Role

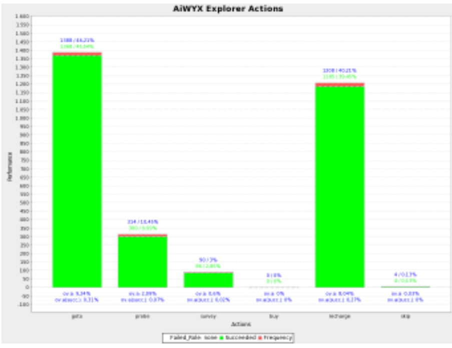


Figure 334: AiWYX vs. UFSC – Simulation 1 - AiWYX Explorer Actions.

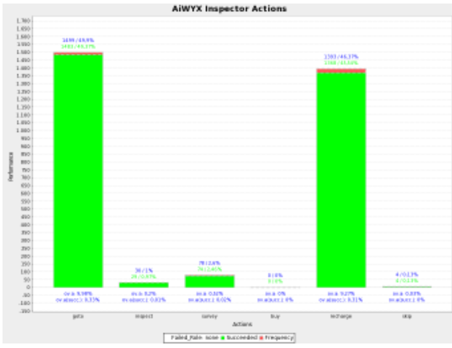


Figure 335: AiWYX vs. UFSC – Simulation 1 - AiWYX Inspector Actions.

Figure 336: AiWYX vs. UFSC – Simulation 1 - AiWYX Repairer Actions.

Figure 337: AiWYX vs. UFSC – Simulation 1 - AiWYX Saboteur Actions.

Figure 338: AiWYX vs. UFSC – Simulation 1 - AiWYX Sentinel Actions.

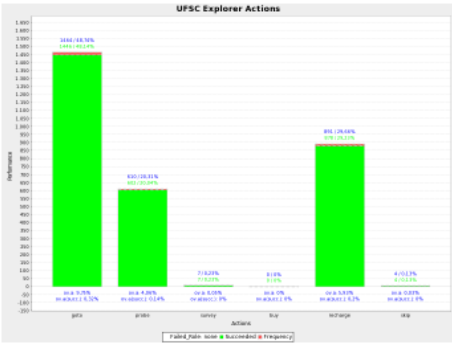


Figure 339: AiWYX vs. UFSC – Simulation 1 - UFSC Explorer Actions.

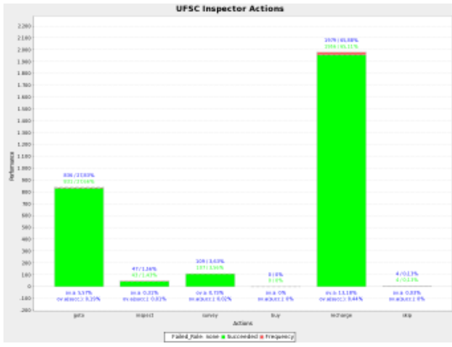


Figure 340: AiWYX vs. UFSC – Simulation 1 - UFSC Inspector Actions.

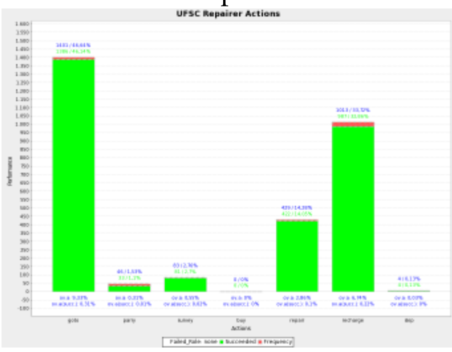


Figure 341: AiWYX vs. UFSC – Simulation 1 - UFSC Repairer Actions.

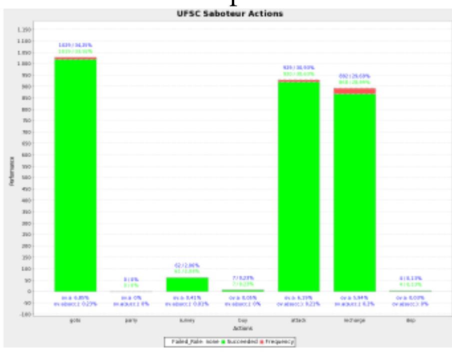


Figure 342: AiWYX vs. UFSC – Simulation 1 - UFSC Saboteur Actions.

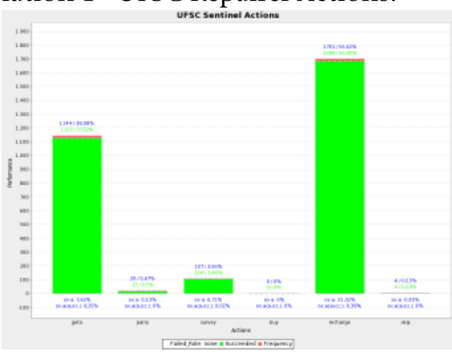


Figure 343: AiWYX vs. UFSC – Simulation 1 - UFSC Sentinel Actions.

26 AiWYX vs. UFSC – Simulation 2

26.1 Scores, Zone Stability and Achievements

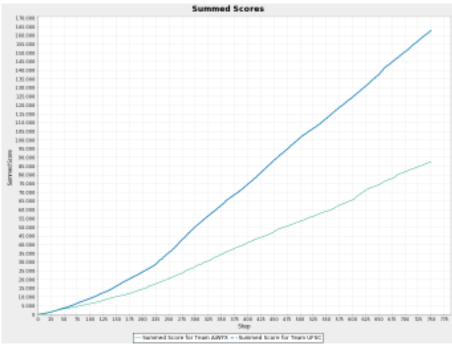


Figure 344: Summed scores.

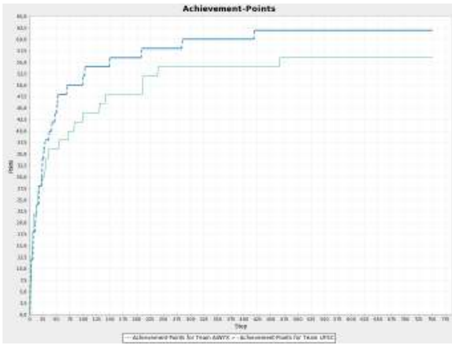


Figure 345: Achievement points.

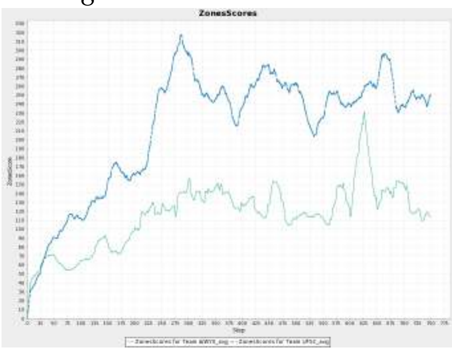


Figure 346: Zones scores.

Figure 347: Zones scores and achievement points.

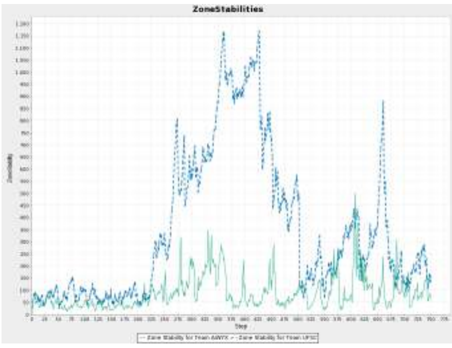


Figure 348: Zone Stabilities.

Step	AiWYX	UFSC
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, area10, surveyed20
2	area20, surveyed80	surveyed40
3		surveyed80, proved5
4	proved5	
5	surveyed160, inspected5	
6		proved10, surveyed160, attacked5
7	proved10, attacked5	
10		attacked10
11		proved20
13	proved20	inspected5
15	attacked10	
16	inspected10	
17		area20, attacked20
22		surveyed320
23	surveyed320	inspected10, proved40
26		parried5
28	area40	area40
30	attacked20	
34	proved40	
36		attacked40
41		parried10
47		proved80
51		parried20
52		attacked80
54	attacked40	
69		area80
72	proved80	
83	attacked80	
99	surveyed640	attacked160
103		proved160
130	proved160	
141	attacked160	
149		area160
208		attacked320
211	area160, area80	
239	attacked320	
284		parried40
418		attacked640
466	attacked640	

Figure 349: Achievements.

26.2 Stability

Reason	AiWYX	%	UFSC	%
failed away			4	0,03
failed parried	98	0,65		
failed random	166	1,11	147	0,98
failed resources			14	0,09
failed			16	0,11
failed attacked	131	0,87	73	0,49
noAction			16	0,11

Figure 350: Failed actions.

26.3 Achievements

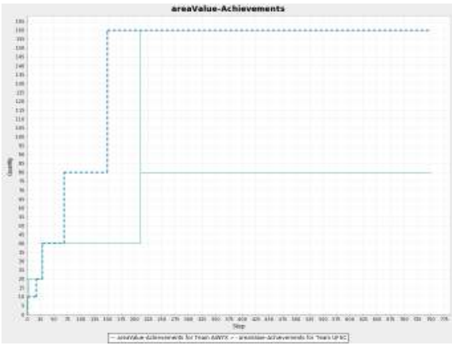


Figure 351: areaValueAchievements.

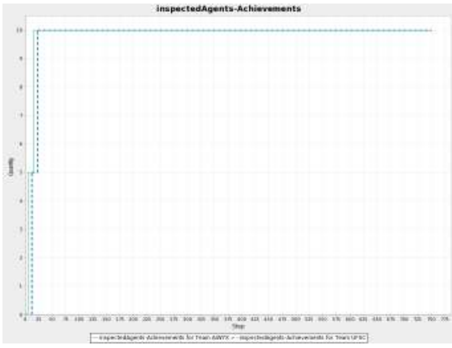


Figure 352: inspectedAgentsAchievements.

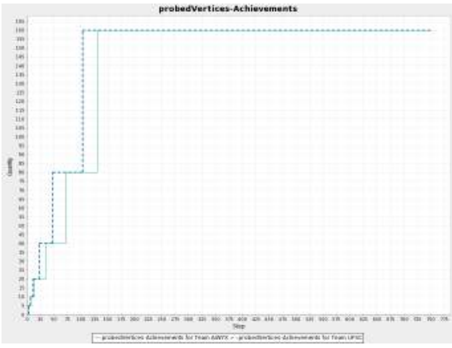


Figure 353: probedVerticesAchievements.

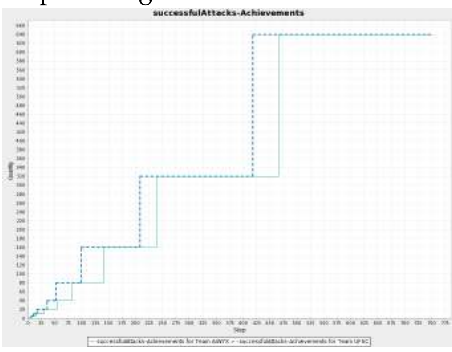


Figure 354: successfulAttacksAchievements.

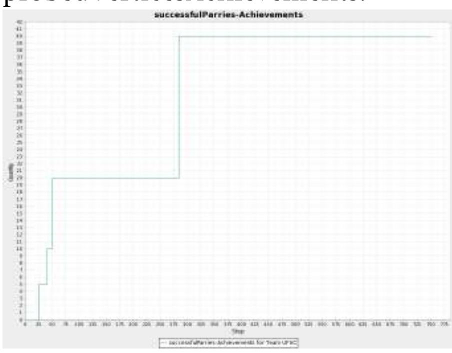


Figure 355: successfulParriesAchievements.

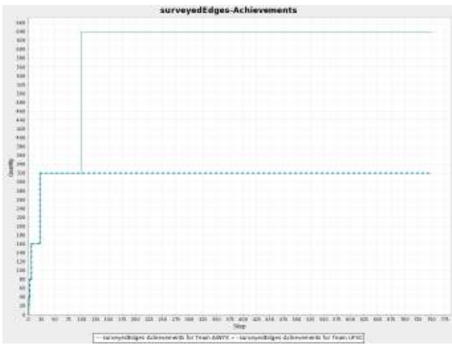


Figure 356: surveyedEdgesAchievements.

26.4 Actions per Role

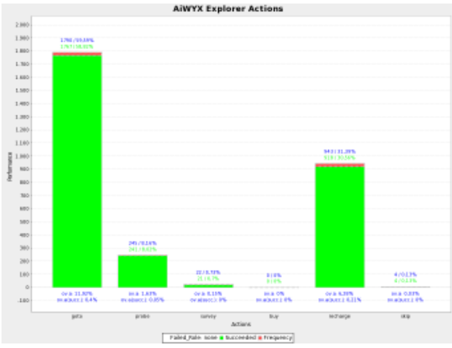


Figure 357: AiWYX vs. UFSC – Simulation 2 - AiWYX Explorer Actions.

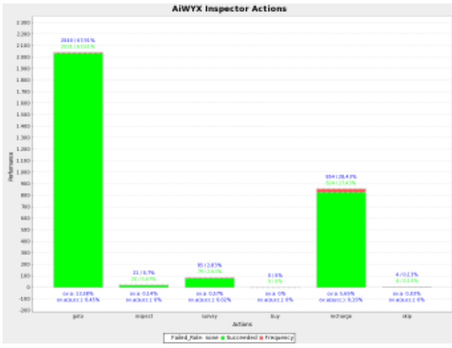


Figure 358: AiWYX vs. UFSC – Simulation 2 - AiWYX Inspector Actions.

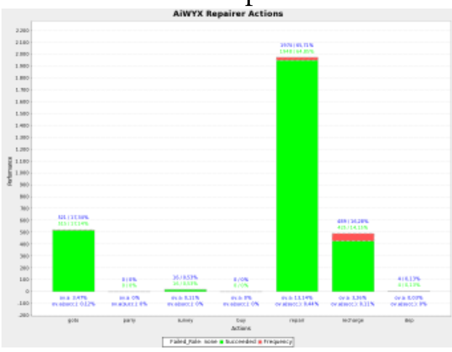


Figure 359: AiWYX vs. UFSC – Simulation 2 - AiWYX Repairer Actions.

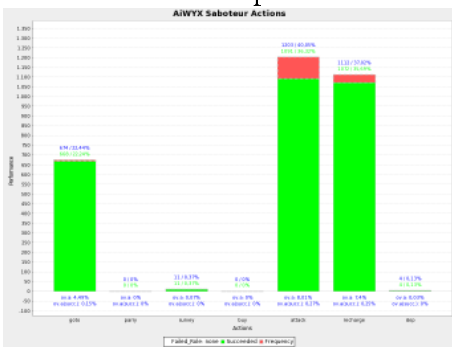


Figure 360: AiWYX vs. UFSC – Simulation 2 - AiWYX Saboteur Actions.

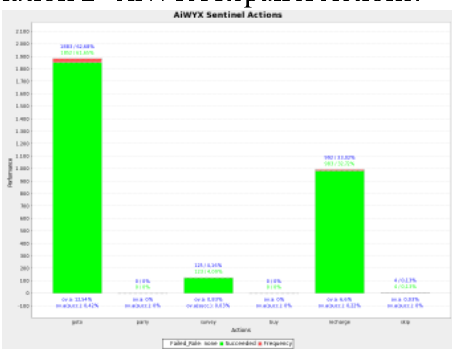


Figure 361: AiWYX vs. UFSC – Simulation 2 - AiWYX Sentinel Actions.

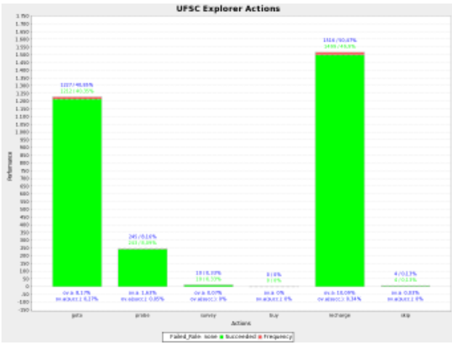


Figure 362: AiWYX vs. UFSC – Simulation 2 - UFSC Explorer Actions.

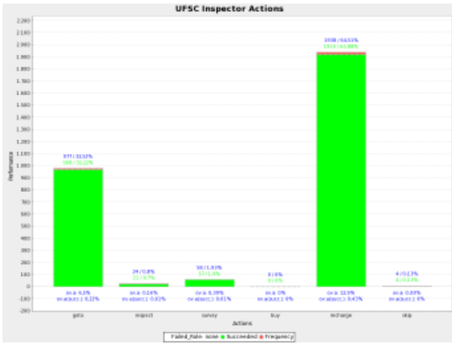


Figure 363: AiWYX vs. UFSC – Simulation 2 - UFSC Inspector Actions.

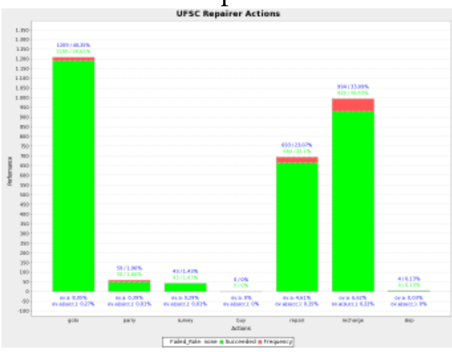


Figure 364: AiWYX vs. UFSC – Simulation 2 - UFSC Repairer Actions.

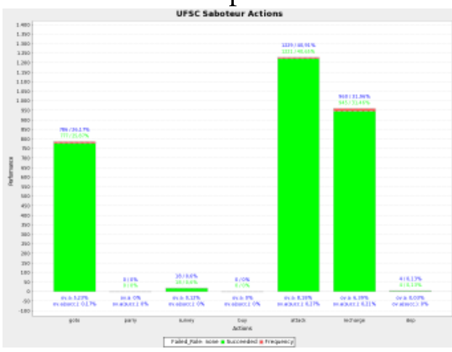


Figure 365: AiWYX vs. UFSC – Simulation 2 - UFSC Saboteur Actions.

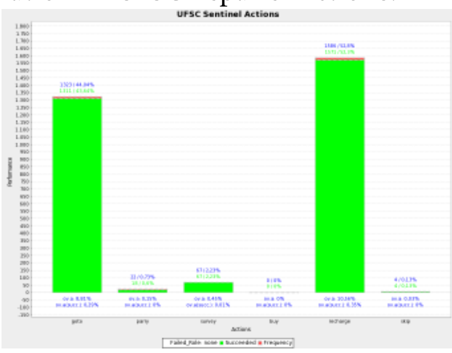


Figure 366: AiWYX vs. UFSC – Simulation 2 - UFSC Sentinel Actions.

27 AiWYX vs. UFSC – Simulation 3

27.1 Scores, Zone Stability and Achievements

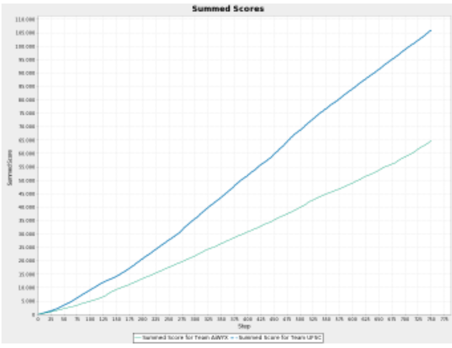


Figure 367: Summed scores.

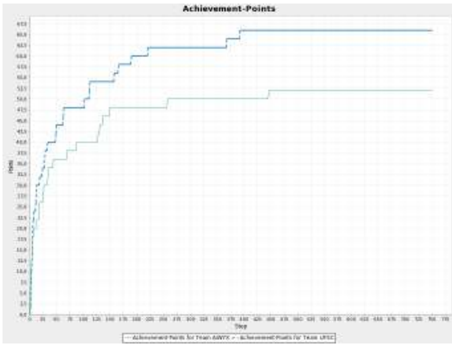


Figure 368: Achievement points.

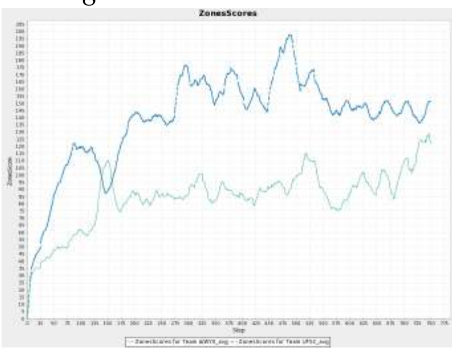


Figure 369: Zones scores.

Figure 370: Zones scores and achievement points.

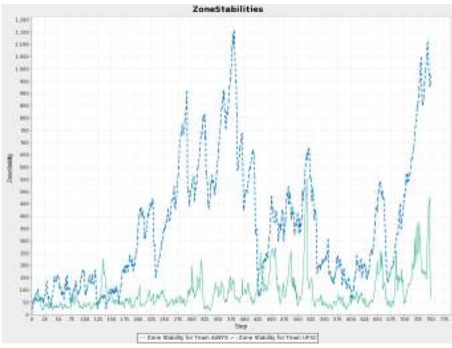


Figure 371: Zone Stabilities.

Step	AiWYX	UFSC
1	surveyed10, surveyed40, surveyed20, inspected5	surveyed10
2	surveyed80	surveyed40, surveyed20, inspected5
3	area10	proved5
4	proved5	inspected10, surveyed80
5	inspected10, surveyed160	area20, proved10, area10
6		attacked5
7	proved10	
8		surveyed160
11		proved20
12		parried5, attacked10
13	attacked5	
17	proved20, surveyed320	
18		attacked20
23		proved40
24	attacked10	
26	area20	
28		attacked40, parried10
33	attacked20	area40
34	proved40	
43	attacked40	
48		parried20
49		proved80
62		attacked80
63		area80
69	proved80	
86	attacked80	
102		parried40
111		proved160
112		attacked160
127	area40	
130	area80	
136	proved160	
149	attacked160	
157		parried80
165		surveyed320
189		inspected20
220		attacked320
256	attacked320	
367		area160
392		attacked640
445	attacked640	

Figure 372: Achievements.

27.2 Stability

Reason	AiWYX	%	UFSC	%
failed away			2	0,01
failed parried	135	0,9		
failed random	145	0,97	165	1,1
failed resources			15	0,1
failed	1	0,01	14	0,09
failed attacked	127	0,85	53	0,35
noAction	1	0,01	14	0,09

Figure 373: Failed actions.

27.3 Achievements

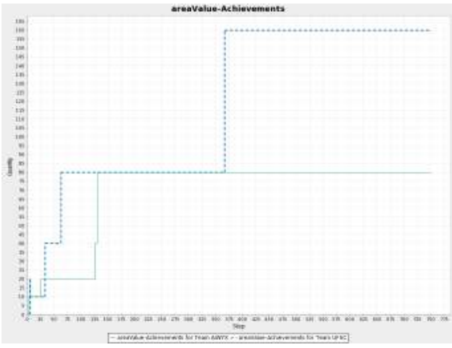


Figure 374: areaValueAchievements.

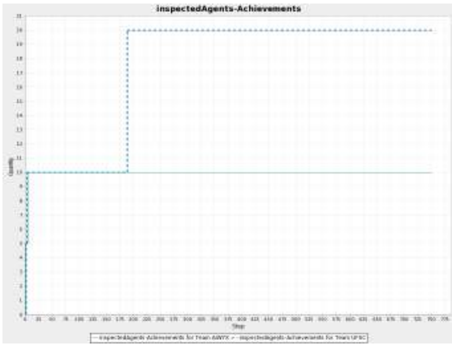


Figure 375: inspectedAgentsAchievements.

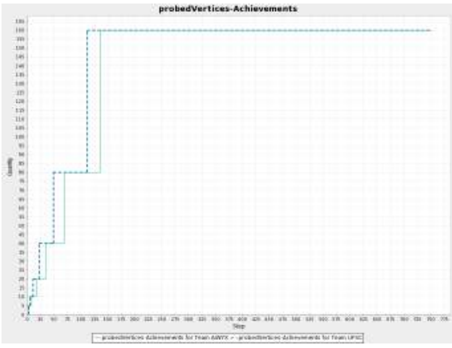


Figure 376: probedVerticesAchievements.

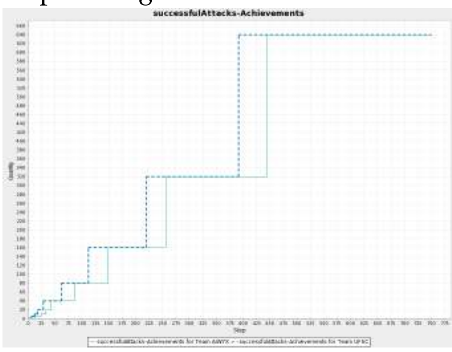


Figure 377: successfulAttacksAchievements.

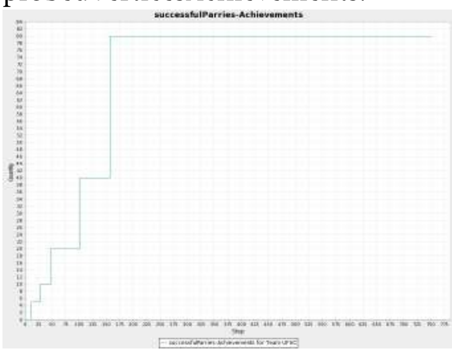


Figure 378: successfulParriesAchievements.

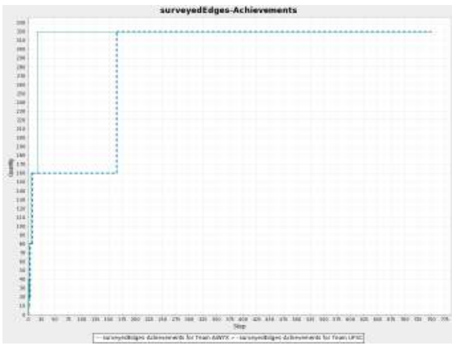


Figure 379: surveyedEdgesAchievements.

27.4 Actions per Role

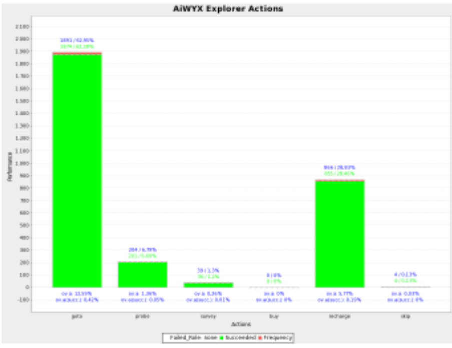


Figure 380: AiWYX vs. UFSC – Simulation 3 - AiWYX Explorer Actions.

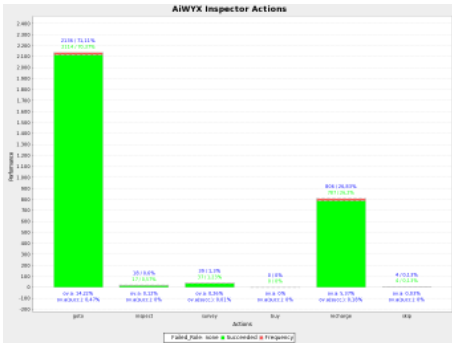


Figure 381: AiWYX vs. UFSC – Simulation 3 - AiWYX Inspector Actions.

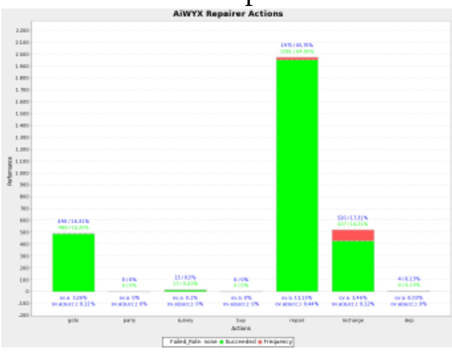


Figure 382: AiWYX vs. UFSC – Simulation 3 - AiWYX Repairer Actions.

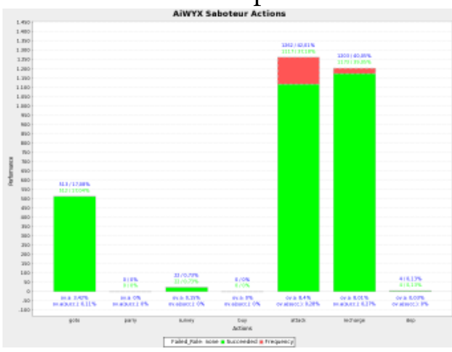


Figure 383: AiWYX vs. UFSC – Simulation 3 - AiWYX Saboteur Actions.

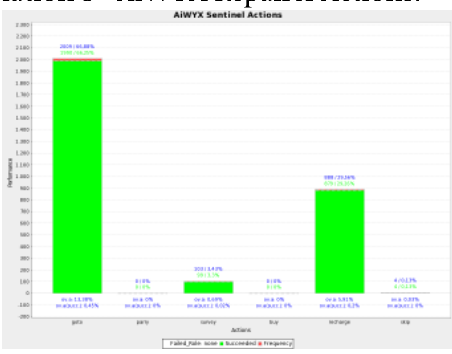


Figure 384: AiWYX vs. UFSC – Simulation 3 - AiWYX Sentinel Actions.

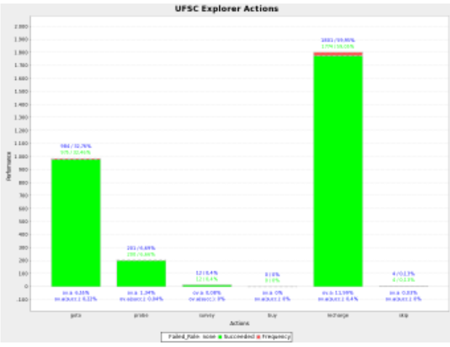


Figure 385: AiWYX vs. UFSC – Simulation 3 - UFSC Explorer Actions.

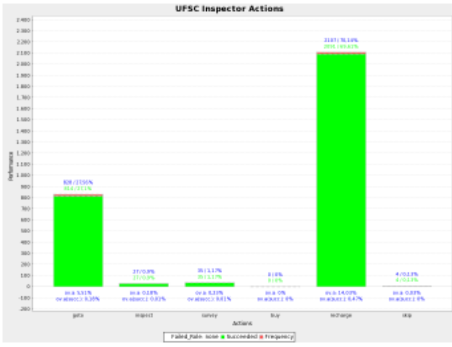


Figure 386: AiWYX vs. UFSC – Simulation 3 - UFSC Inspector Actions.

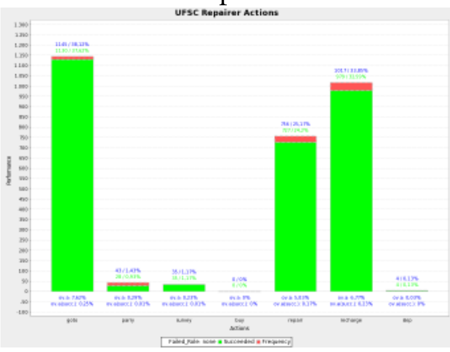


Figure 387: AiWYX vs. UFSC – Simulation 3 - UFSC Repairer Actions.

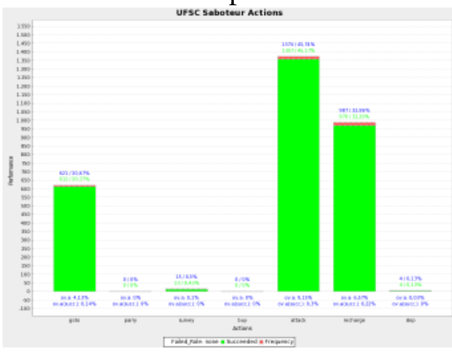


Figure 388: AiWYX vs. UFSC – Simulation 3 - UFSC Saboteur Actions.

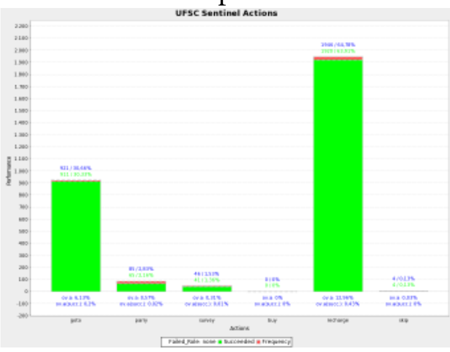


Figure 389: AiWYX vs. UFSC – Simulation 3 - UFSC Sentinel Actions.

28 AiWYX vs. USP – Simulation 1

28.1 Scores, Zone Stability and Achievements

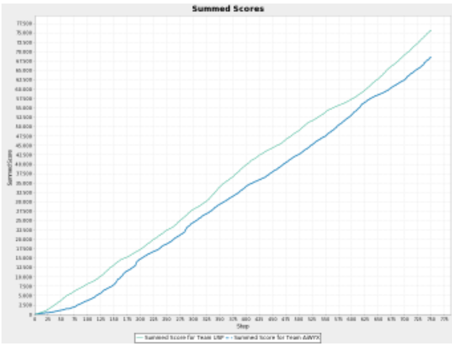


Figure 390: Summed scores.

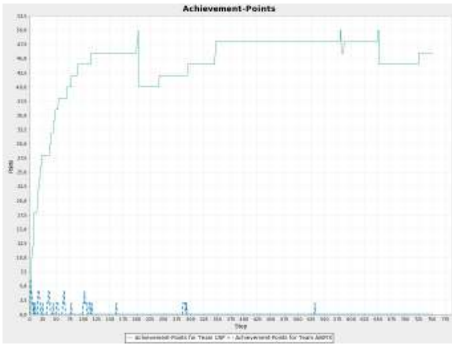


Figure 391: Achievement points.

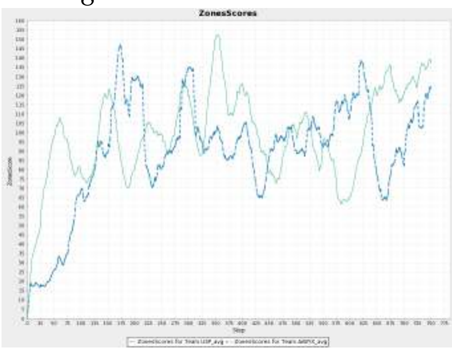


Figure 392: Zones scores.

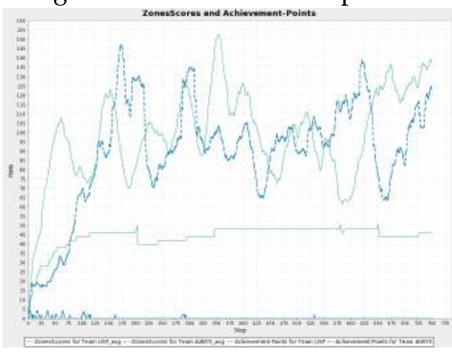


Figure 393: Zones scores and achievement points.

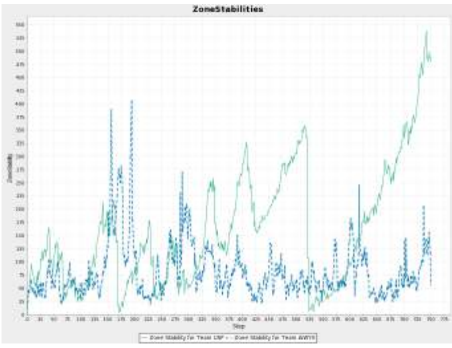


Figure 394: Zone Stabilities.

Step	USP	AiWYX
1		surveyed10, surveyed40, surveyed20
2	surveyed10, surveyed40, surveyed20	surveyed80, area10
3	surveyed80, area10	inspected5
4		proved5
5	proved5	surveyed160
7	area20	proved10
8	proved10, attacked5	
10		inspected10
14	surveyed160	
15	inspected5	proved20, attacked5
16		surveyed320
17	proved20	
19	area40	
22	attacked10	
23		attacked10
32		proved40
34		inspected20
36		area20
37	area80	
40	proved40	
43		attacked20
44	parried5	
47	attacked20	
49		surveyed640
53	parried10	
61		proved80
63		attacked40
70	inspected10	
77	parried20	area40
89	surveyed320	
99		attacked80
102		area80
110		proved160
114	attacked40	
115		area160
161		attacked160
199	attacked80	
202	parried40	
241	inspected20	
285		area320
292		attacked320
294	proved80	
345	attacked160	
346	parried80	
531		attacked640
579	parried160	
586	attacked320	
649	proved160	
725	surveyed640	

Figure 395: Achievements.

28.2 Stability

Reason	USP	%	AiWYX	%
failed away	311	2,07		
failed parried			376	2,51
failed random	162	1,08	151	1,01
failed	368	2,45		
failed resources	61	0,41	13	0,09
failed attacked	182	1,21	95	0,63
noAction	372	2,48		

Figure 396: Failed actions.

28.3 Achievements

Figure 397: areaValueAchievements.

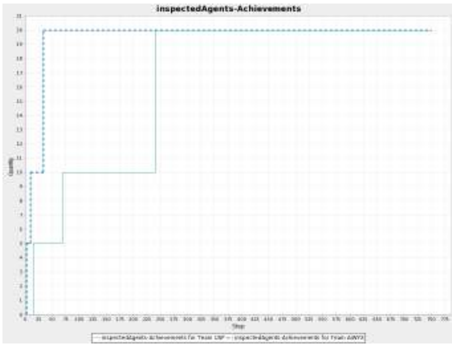


Figure 398: inspectedAgentsAchievements.

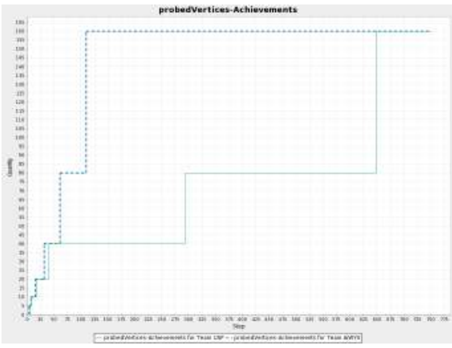


Figure 399: probedVerticesAchievements.

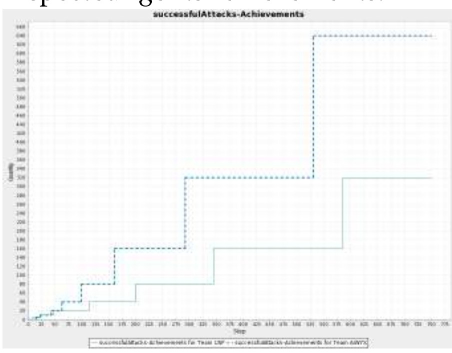


Figure 400: successfulAttacksAchievements.

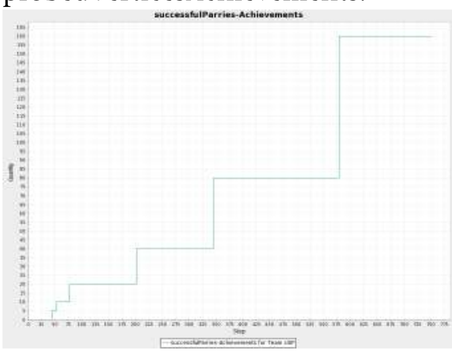


Figure 401: successfulParriesAchievements.

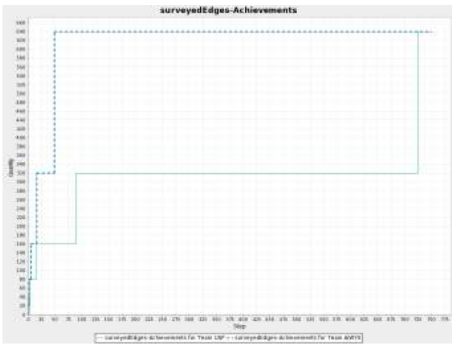


Figure 402: surveyedEdgesAchievements.

28.4 Actions per Role

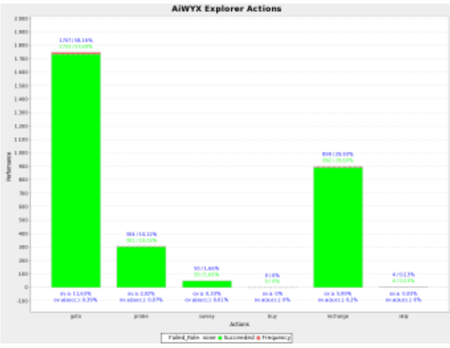


Figure 403: AiWYX vs. USP – Simulation 1 - AiWYX Explorer Actions.

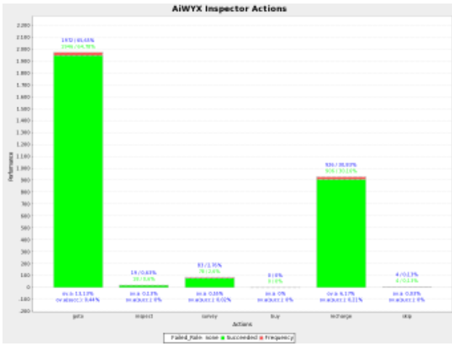


Figure 404: AiWYX vs. USP – Simulation 1 - AiWYX Inspector Actions.

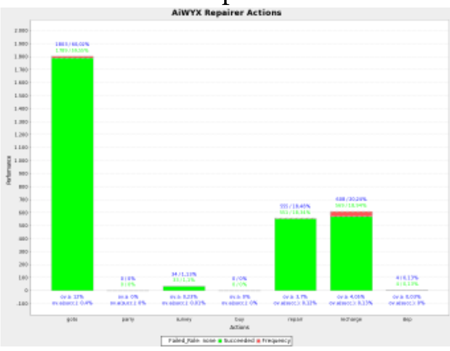


Figure 405: AiWYX vs. USP – Simulation 1 - AiWYX Repairer Actions.

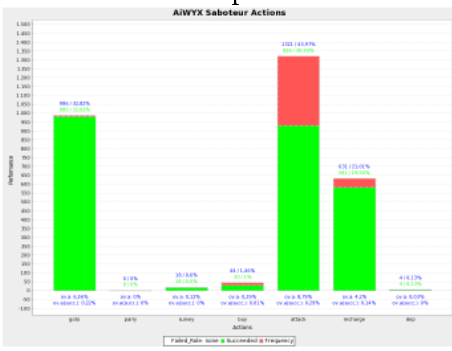


Figure 406: AiWYX vs. USP – Simulation 1 - AiWYX Saboteur Actions.

Figure 407: AiWYX vs. USP – Simulation 1 - AiWYX Sentinel Actions.

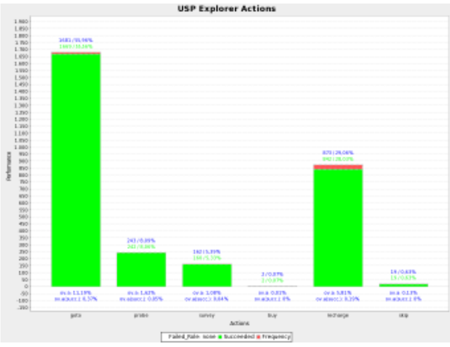


Figure 408: AiWYX vs. USP – Simulation 1 - USP Explorer Actions.

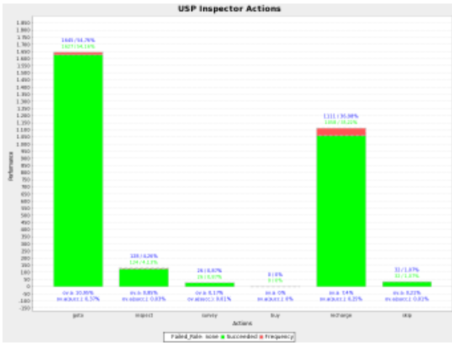


Figure 409: AiWYX vs. USP – Simulation 1 - USP Inspector Actions.

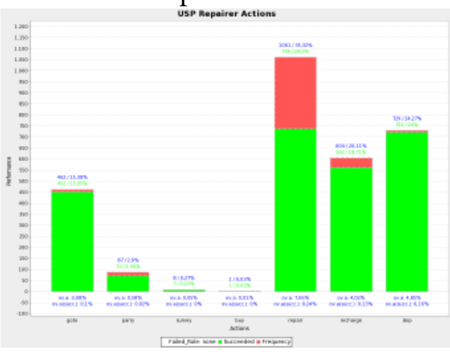


Figure 410: AiWYX vs. USP – Simulation 1 - USP Repairer Actions.

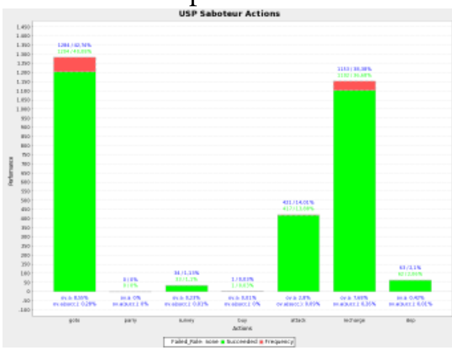


Figure 411: AiWYX vs. USP – Simulation 1 - USP Saboteur Actions.

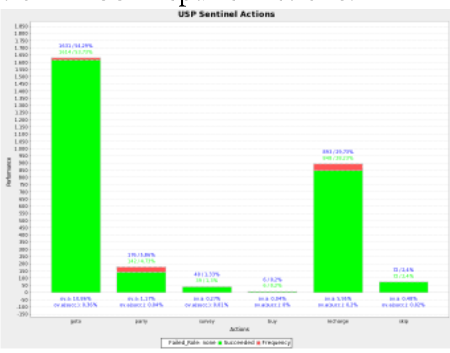


Figure 412: AiWYX vs. USP – Simulation 1 - USP Sentinel Actions.

29 AiWYX vs. USP – Simulation 2

29.1 Scores, Zone Stability and Achievements

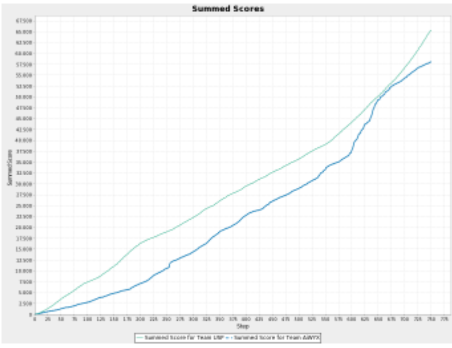


Figure 413: Summed scores.

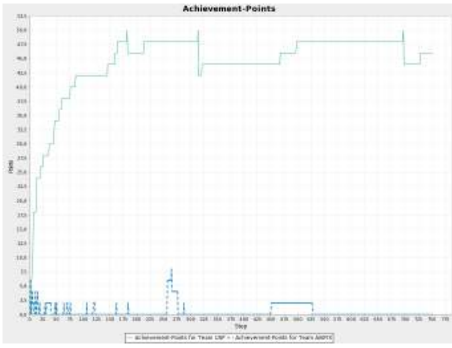


Figure 414: Achievement points.

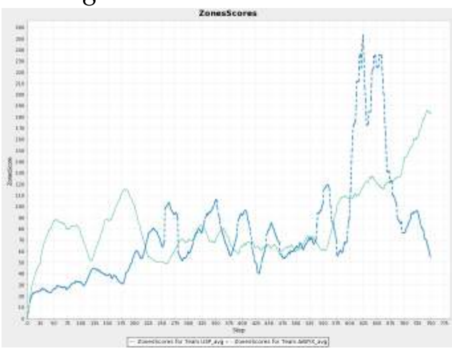


Figure 415: Zones scores.

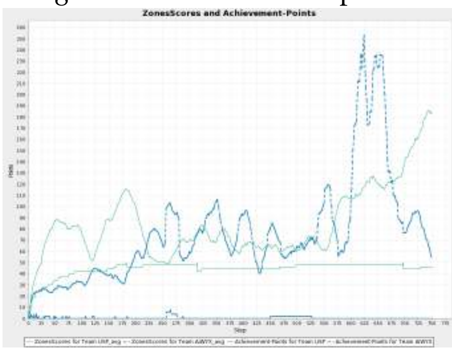


Figure 416: Zones scores and achievement points.

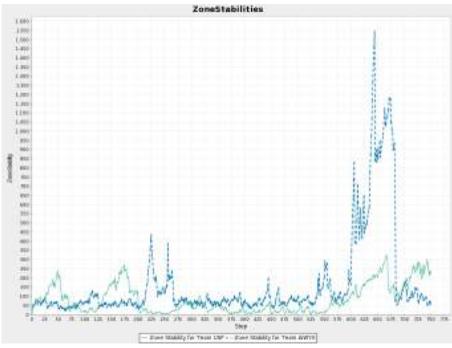


Figure 417: Zone Stabilities.

Step	USP	AiWYX
1		surveyed10, surveyed40, surveyed20
2	surveyed10, surveyed40, surveyed20	surveyed80
4	proved5	area10, proved5
5	inspected5	surveyed160
6	proved10, surveyed80, area10	inspected5
7	attacked5	proved10
10		area20, attacked5
12	area40, area20, proved20	
13		proved20
14		inspected10
18		surveyed320
19	surveyed160	
25	attacked10	
28		proved40
31		attacked10
36	proved40	
44	inspected10	
46	parried5	
47		area40
50		attacked20
55	attacked20	
59	parried10	
63		proved80
69		attacked40
75	parried20	surveyed640
85	attacked40	
106		attacked80
118		proved160
145	parried40	
159	surveyed320	
161		attacked160
163	area80	
181	attacked80	
183		area80
213	proved80	
256		area160, area320, area640
264		inspected20
287		attacked320
313	parried80	
321	attacked160	
450		attacked640
467	inspected20	
497	attacked320	
696	proved160	
728	area160	

Figure 418: Achievements.

29.2 Stability

Reason	USP	%	AiWYX	%
failed away	213	1,42		
failed parried			187	1,25
failed random	151	1,01	139	0,93
failed	169	1,13	135	0,9
failed resources	25	0,17	15	0,1
failed attacked	98	0,65	91	0,61
noAction	172	1,15	136	0,91

Figure 419: Failed actions.

29.3 Achievements

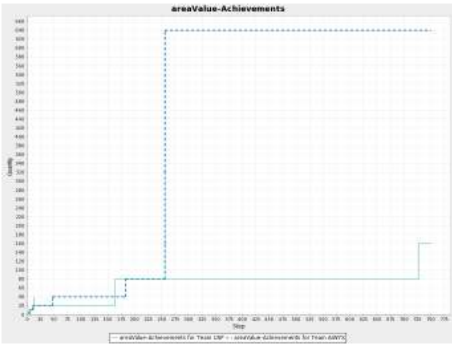


Figure 420: areaValueAchievements.

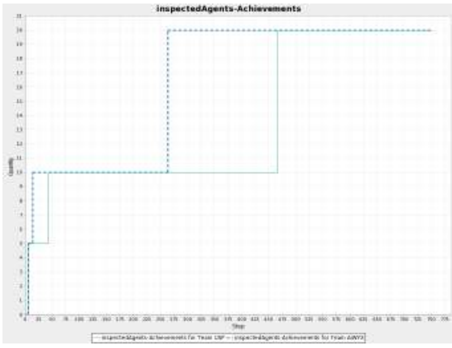


Figure 421: inspectedAgentsAchievements.

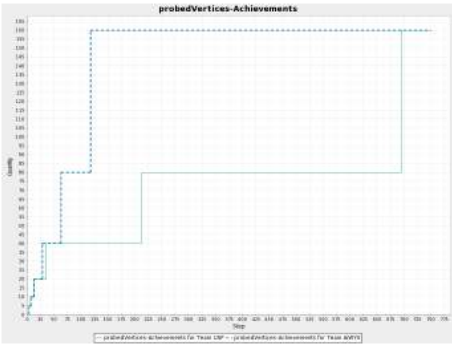


Figure 422: probedVerticesAchievements.

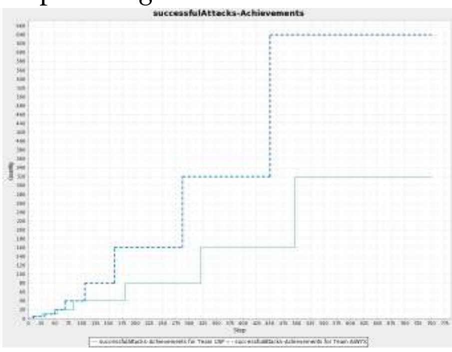


Figure 423: successfulAttacksAchievements.

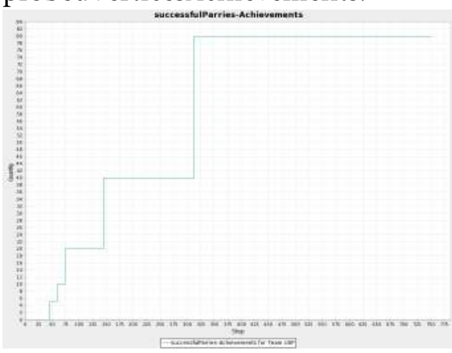


Figure 424: successfulParriesAchievements.

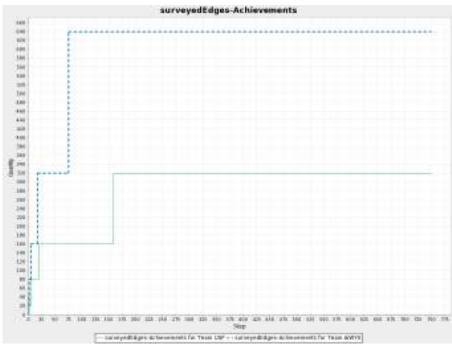


Figure 425: surveyedEdgesAchievements.

29.4 Actions per Role

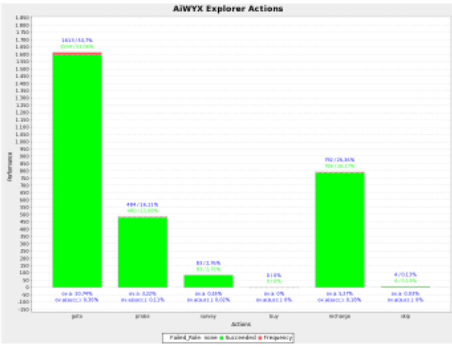


Figure 426: AiWYX vs. USP – Simulation 2 - AiWYX Explorer Actions.

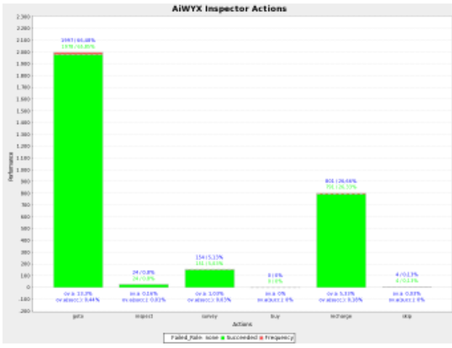


Figure 427: AiWYX vs. USP – Simulation 2 - AiWYX Inspector Actions.

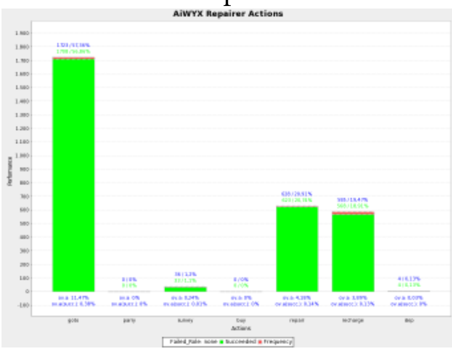


Figure 428: AiWYX vs. USP – Simulation 2 - AiWYX Repairer Actions.

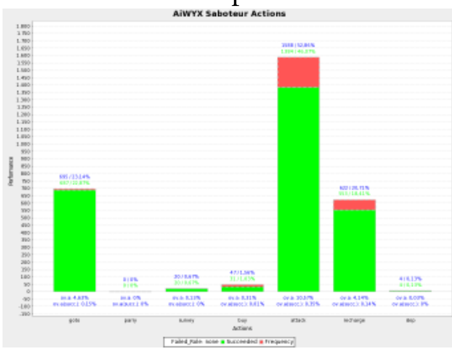


Figure 429: AiWYX vs. USP – Simulation 2 - AiWYX Saboteur Actions.

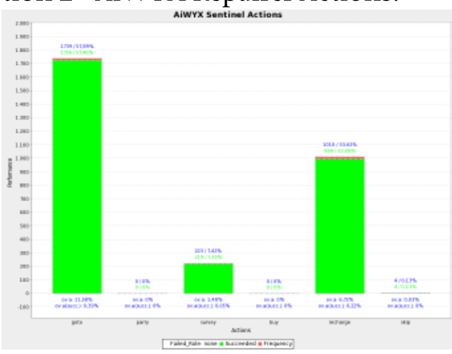


Figure 430: AiWYX vs. USP – Simulation 2 - AiWYX Sentinel Actions.

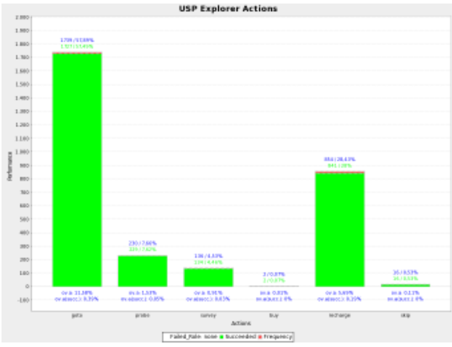


Figure 431: AiWYX vs. USP – Simulation 2 - USP Explorer Actions.

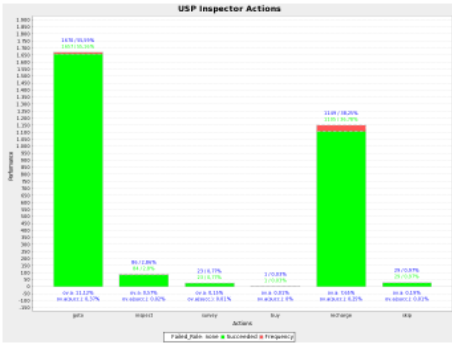


Figure 432: AiWYX vs. USP – Simulation 2 - USP Inspector Actions.

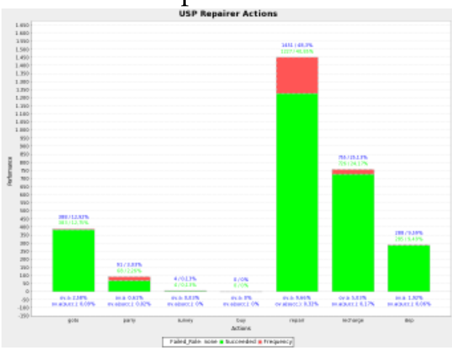


Figure 433: AiWYX vs. USP – Simulation 2 - USP Repairer Actions.

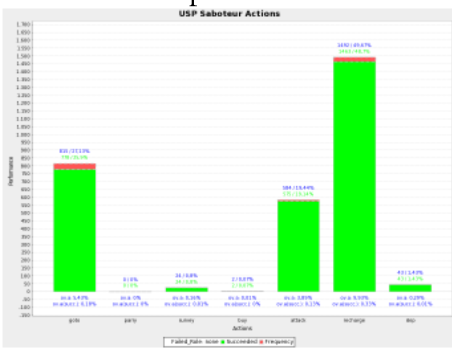


Figure 434: AiWYX vs. USP – Simulation 2 - USP Saboteur Actions.

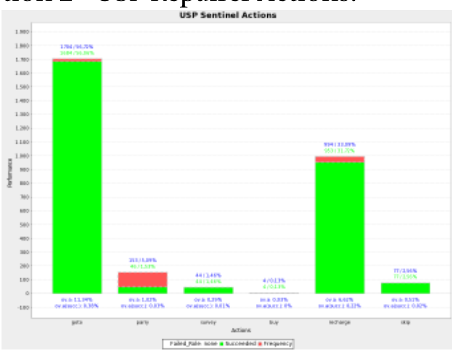


Figure 435: AiWYX vs. USP – Simulation 2 - USP Sentinel Actions.

30 AiWYX vs. USP – Simulation 3

30.1 Scores, Zone Stability and Achievements

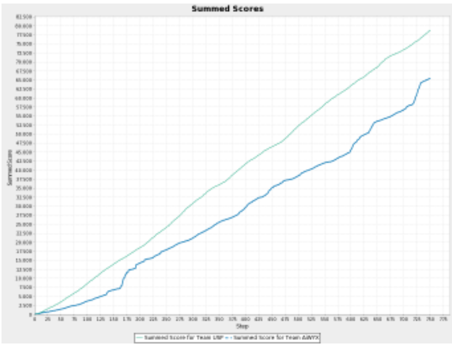


Figure 436: Summed scores.

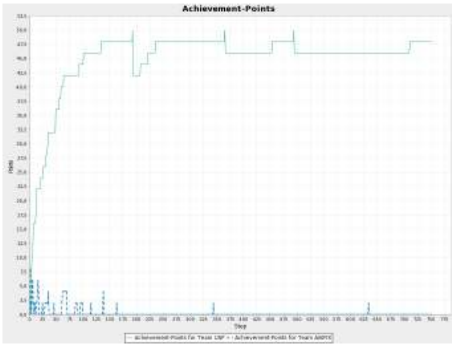


Figure 437: Achievement points.

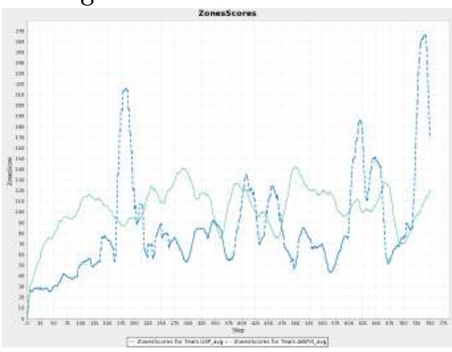


Figure 438: Zones scores.

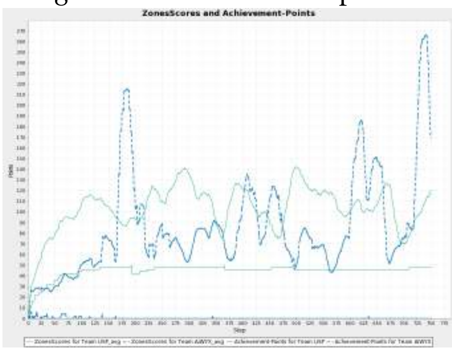


Figure 439: Zones scores and achievement points.

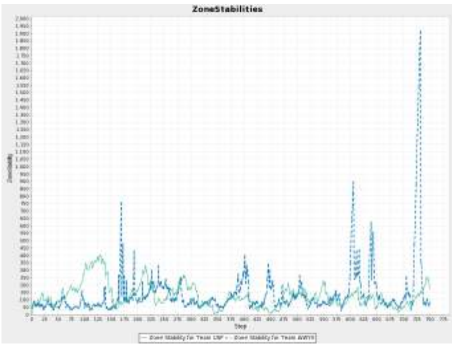


Figure 440: Zone Stabilities.

Step	USP	AiWYX
1	area10	surveyed10, surveyed40, area10, surveyed20
2	surveyed10, area20, surveyed20	surveyed80
4	surveyed40, proved5	area20, proved5, inspected5
5		surveyed160
6	proved10	
7	inspected5	proved10
9		inspected10
11	surveyed80	
12	area40, parried5	attacked5
14		proved20
15		surveyed320
19	proved20	
23		attacked10
25	parried10	
28		proved40
30	surveyed160	
33	attacked5	
34	inspected10	inspected20
45		attacked20
48	proved40, attacked10	
55	parried20	
58	area80	
60		attacked40
61		proved80
63	attacked20	
85		area40
88		attacked80
92	inspected20	
94		area80
100	parried40	
114		proved160
133	attacked40	
137		area160, area320
162		attacked160
192	proved80	
207	attacked80	
221	surveyed320	
235	parried80	
343		attacked320
364	attacked160	
453	parried160	
493	proved160	
633		attacked640
710	attacked320	

Figure 441: Achievements.

30.2 Stability

Reason	AiWYX	%	USP	%
failed away			251	1,67
failed parried	450	3		
failed random	162	1,08	156	1,04
failed	135	0,9	144	0,96
failed resources	13	0,09	30	0,2
failed attacked	82	0,55	157	1,05
noAction	136	0,91	145	0,97

Figure 442: Failed actions.

30.3 Achievements

Figure 443: areaValueAchievements.

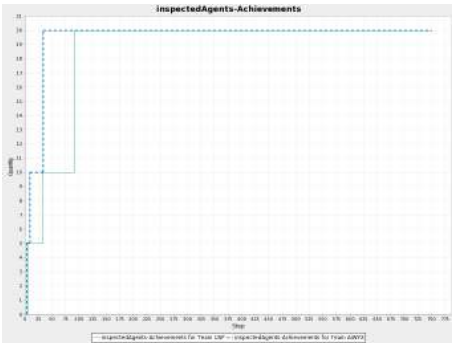


Figure 444: inspectedAgentsAchievements.

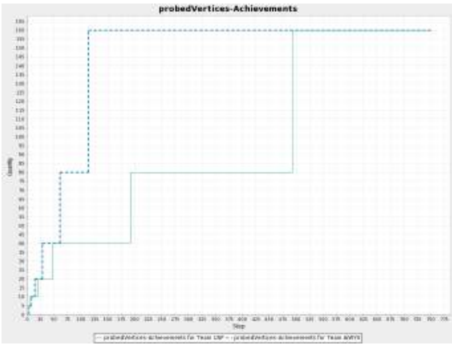


Figure 445: probedVerticesAchievements.

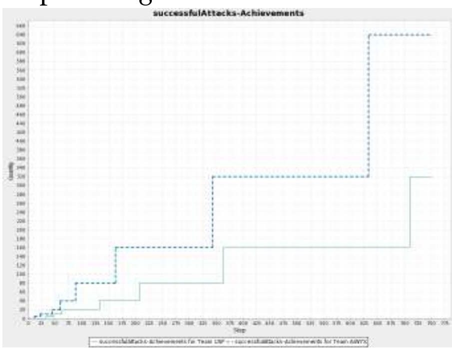


Figure 446: successfulAttacksAchievements.

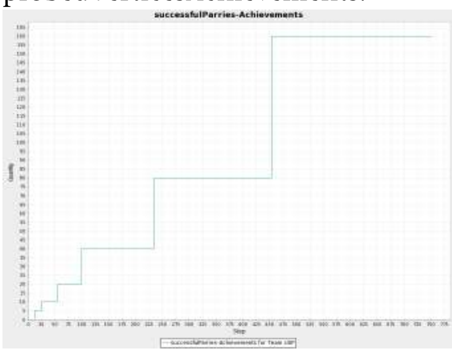


Figure 447: successfulParriesAchievements.

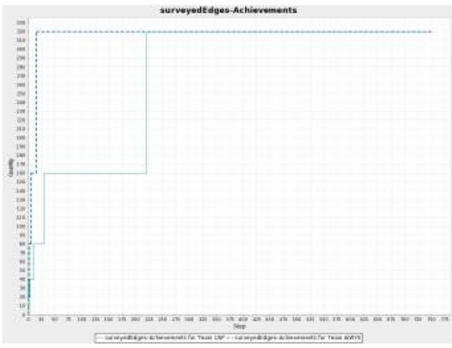


Figure 448: surveyedEdgesAchievements.

30.4 Actions per Role

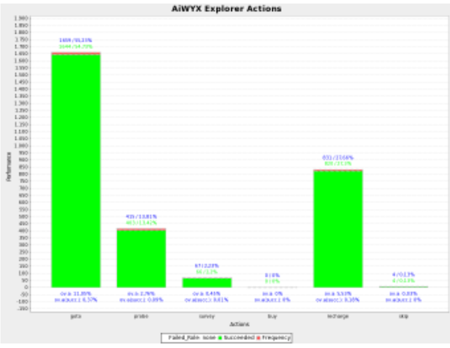


Figure 449: AiWYX vs. USP – Simulation 3 - AiWYX Explorer Actions.

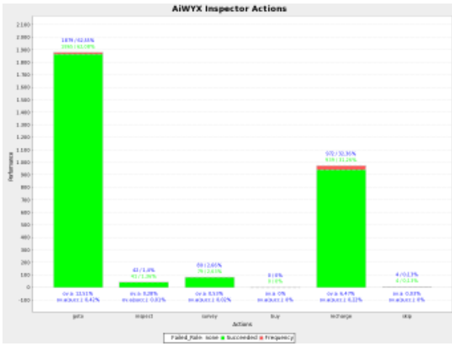


Figure 450: AiWYX vs. USP – Simulation 3 - AiWYX Inspector Actions.

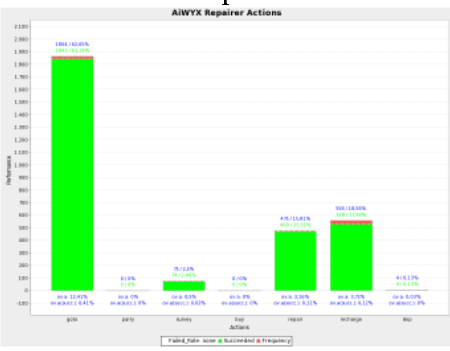


Figure 451: AiWYX vs. USP – Simulation 3 - AiWYX Repairer Actions.

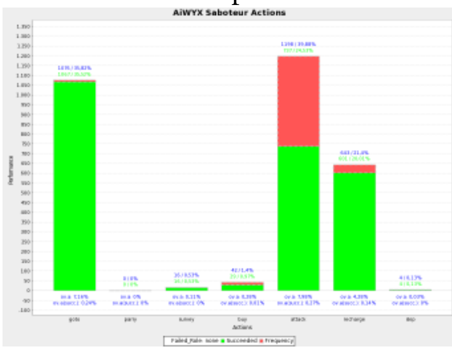


Figure 452: AiWYX vs. USP – Simulation 3 - AiWYX Saboteur Actions.

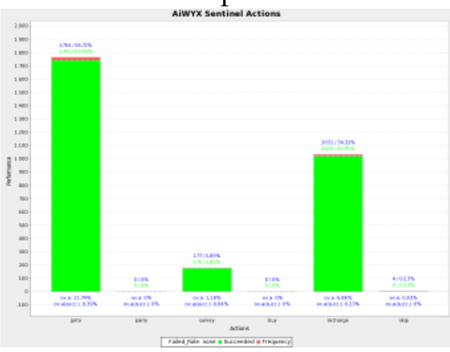


Figure 453: AiWYX vs. USP – Simulation 3 - AiWYX Sentinel Actions.

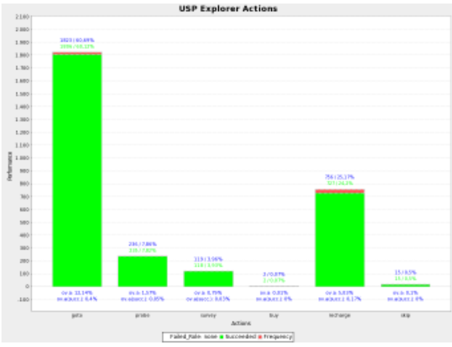


Figure 454: AiWYX vs. USP – Simulation 3 - USP Explorer Actions.

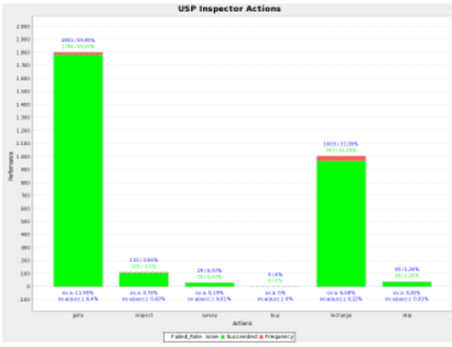


Figure 455: AiWYX vs. USP – Simulation 3 - USP Inspector Actions.

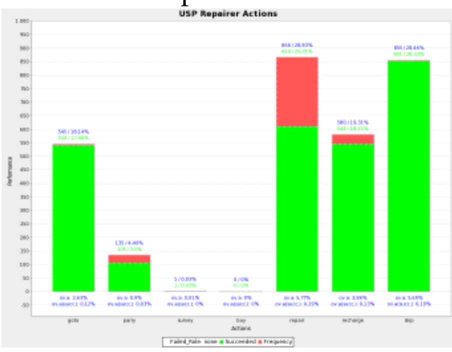


Figure 456: AiWYX vs. USP – Simulation 3 - USP Repairer Actions.

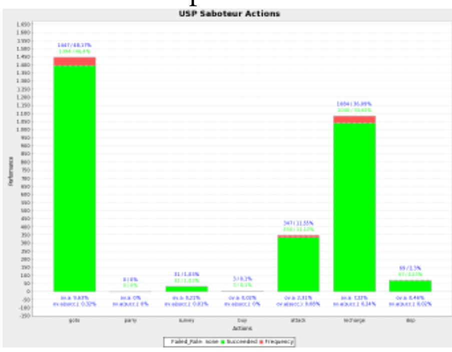


Figure 457: AiWYX vs. USP – Simulation 3 - USP Saboteur Actions.

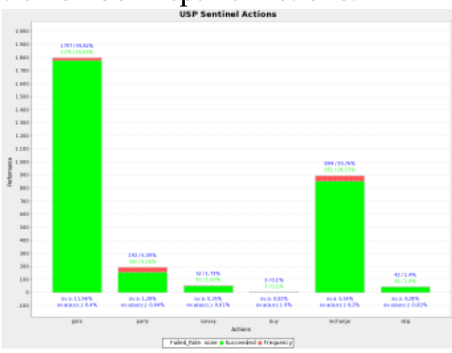


Figure 458: AiWYX vs. USP – Simulation 3 - USP Sentinel Actions.

31 PGIM vs. Python-DTU – Simulation 1

31.1 Scores, Zone Stability and Achievements

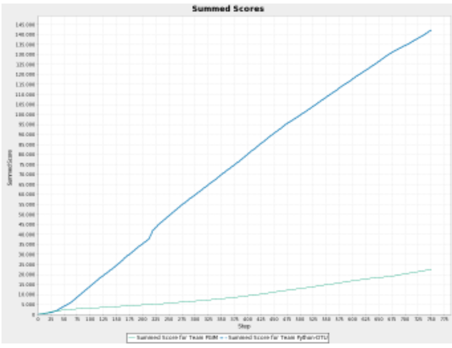


Figure 459: Summed scores.

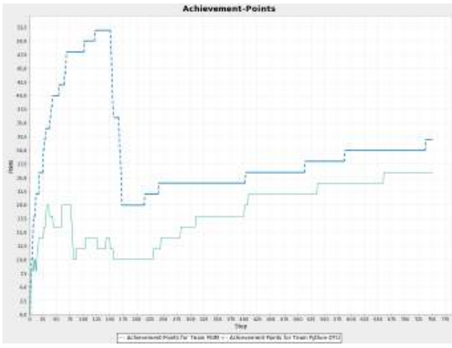


Figure 460: Achievement points.

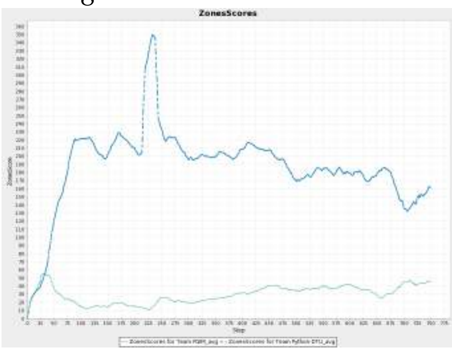


Figure 461: Zones scores.

Figure 462: Zones scores and achievement points.

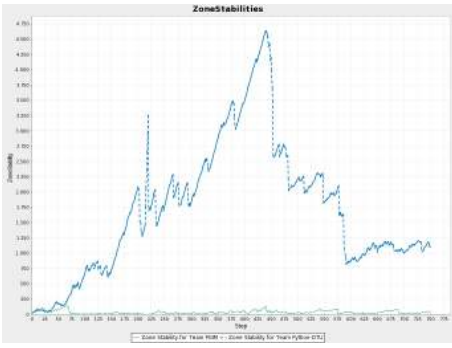


Figure 463: Zone Stabilities.

Step	PGIM	Python-DTU
1	surveyed10, surveyed20	surveyed10, surveyed40, surveyed20
2	surveyed40	
3	area10	surveyed80, proved5
5		proved10, area10
6	proved5	inspected5
7	area20	surveyed160
9	inspected5	
10		proved20
11		inspected10
13	surveyed80	
14	area40	
16	proved10	
17		surveyed320, attacked5
25		proved40, attacked10
26	proved20	
27		area20
29	surveyed160	area40
31	attacked5	
37		area80
39		attacked20
42		inspected20
54		proved80
60	inspected10, attacked10	
65		area160
67		attacked40
68		surveyed640
86	proved40	
101		proved160
104	attacked20	
121		attacked80
142	attacked40	
214		area320
230	surveyed320	
240		attacked160
244	attacked80	
281	parried5	
309	proved80	
400	attacked160	
402		attacked320
407	parried10	
513		parried5
536	attacked320	
587		parried10
660	inspected20	
738		attacked640

Figure 464: Achievements.

31.2 Stability

Reason	PGIM	%	Python-DTU	%
failed away	394	2,63	60	0,4
failed parried	15	0,1	24	0,16
failed random	152	1,01	122	0,81
failed wrong param	62	0,41		
failed	47	0,31		
failed resources	84	0,56		
failed attacked	67	0,45	161	1,07
noAction	48	0,32		
failed status	1	0,01		

Figure 465: Failed actions.

31.3 Achievements

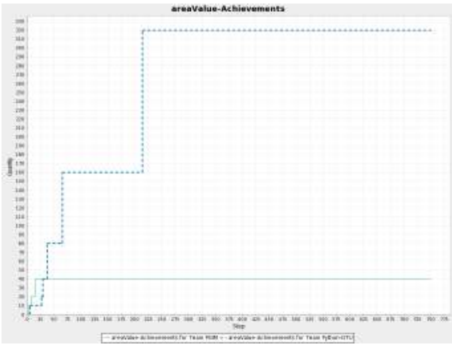


Figure 466: areaValueAchievements.

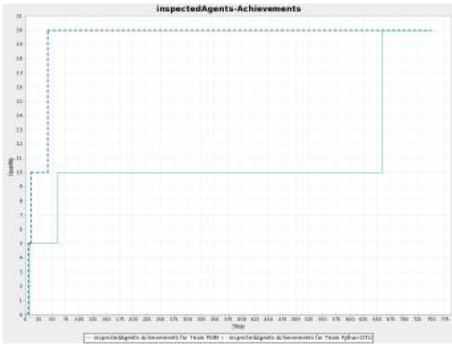


Figure 467: inspectedAgentsAchievements.

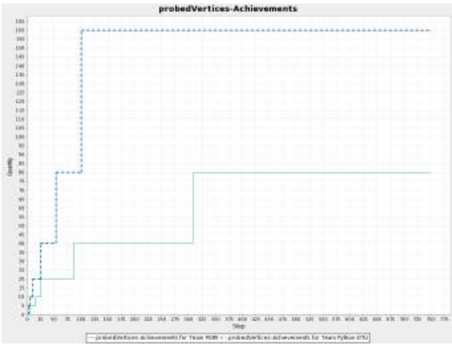


Figure 468: probedVerticesAchievements.

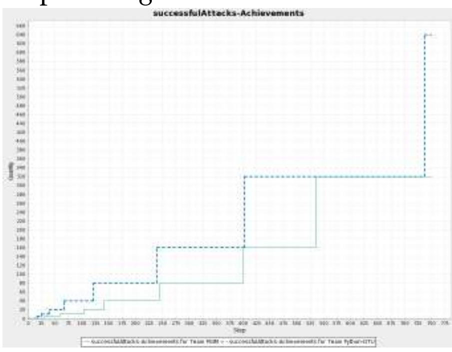


Figure 469: successfulAttacksAchievements.

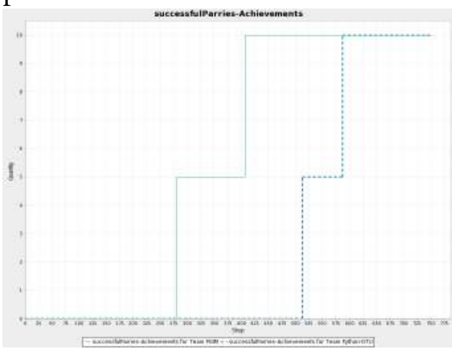


Figure 470: successfulParriesAchievements.

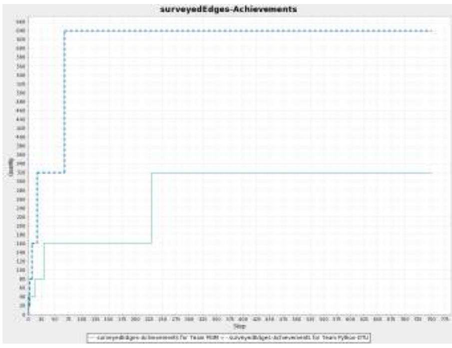


Figure 471: surveyedEdgesAchievements.

31.4 Actions per Role

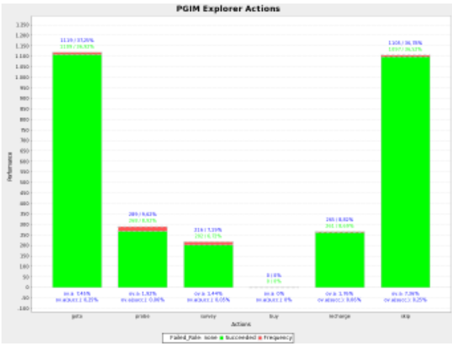


Figure 472: PGIM vs. Python-DTU – Simulation 1 - PGIM Explorer Actions.

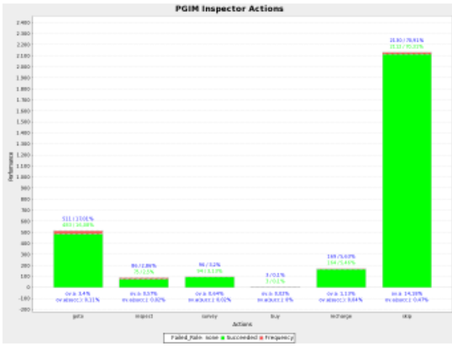


Figure 473: PGIM vs. Python-DTU – Simulation 1 - PGIM Inspector Actions.

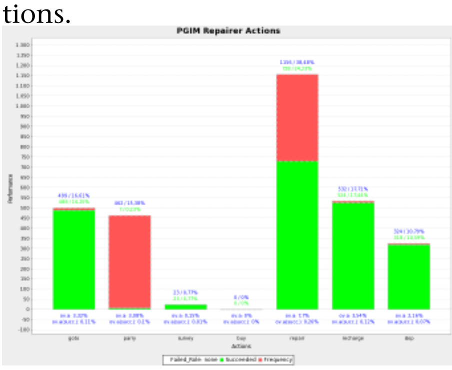


Figure 474: PGIM vs. Python-DTU – Simulation 1 - PGIM Repairer Actions.

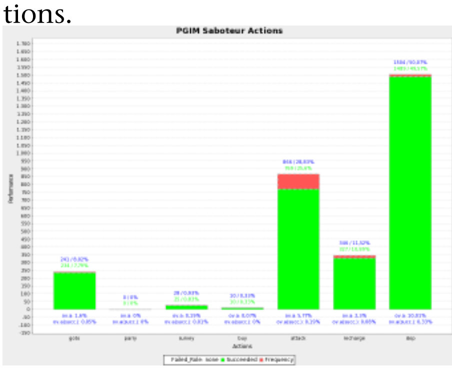


Figure 475: PGIM vs. Python-DTU – Simulation 1 - PGIM Saboteur Actions.

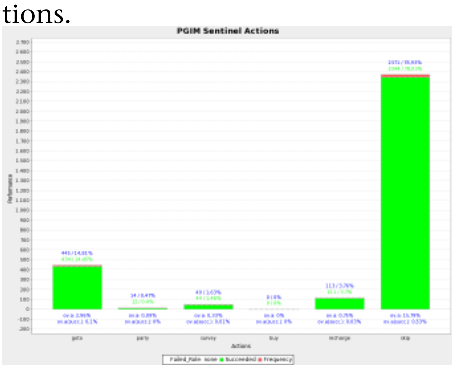


Figure 476: PGIM vs. Python-DTU – Simulation 1 - PGIM Sentinel Actions.

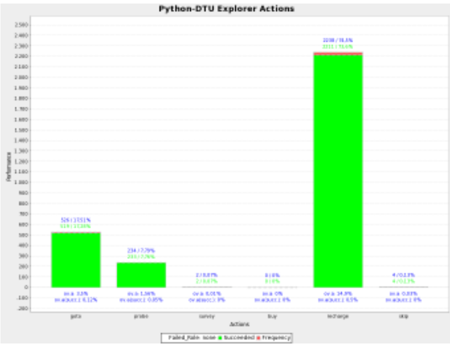


Figure 477: PGIM vs. Python-DTU – Simulation 1 - Python-DTU Explorer Actions.

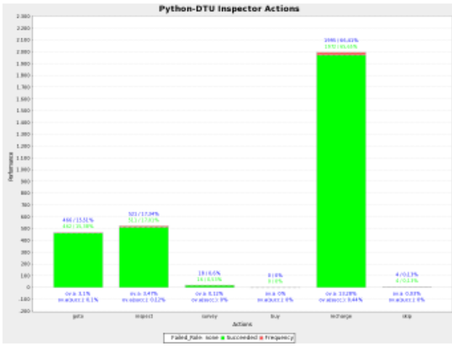


Figure 478: PGIM vs. Python-DTU – Simulation 1 - Python-DTU Inspector Actions.

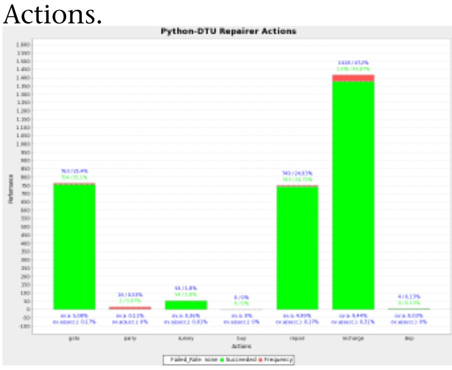


Figure 479: PGIM vs. Python-DTU – Simulation 1 - Python-DTU Repairer Actions.

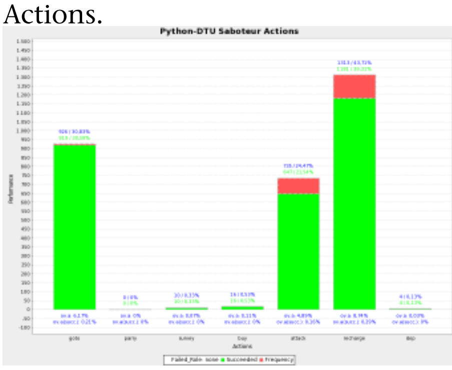


Figure 480: PGIM vs. Python-DTU – Simulation 1 - Python-DTU Saboteur Actions.

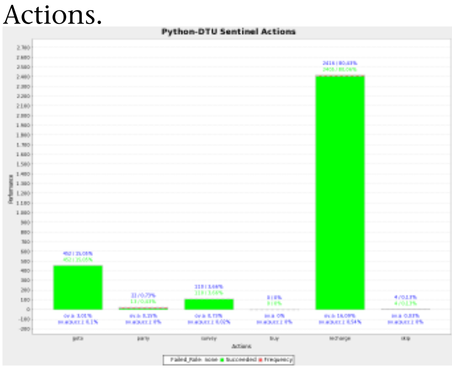


Figure 481: PGIM vs. Python-DTU – Simulation 1 - Python-DTU Sentinel Actions.

32 PGIM vs. Python-DTU – Simulation 2

32.1 Scores, Zone Stability and Achievements

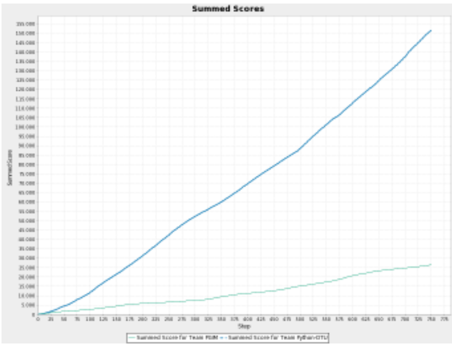


Figure 482: Summed scores.

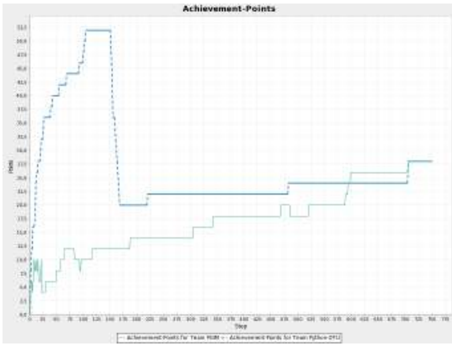


Figure 483: Achievement points.

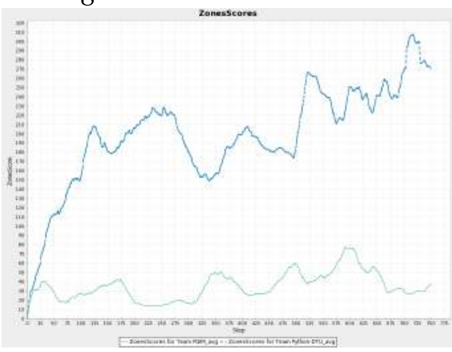


Figure 484: Zones scores.

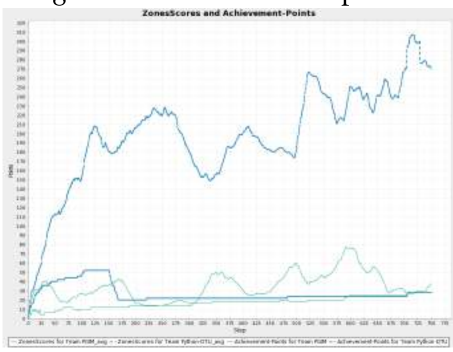


Figure 485: Zones scores and achievement points.

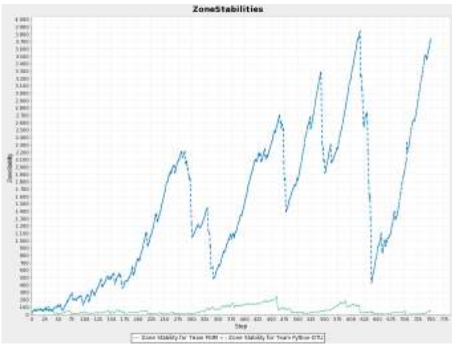


Figure 486: Zone Stabilities.

Step	PGIM	Python-DTU
1	surveyed10, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
2	surveyed40	
3	proved5	surveyed80, proved5
5	surveyed80	proved10, inspected5
7	proved10	
8	area20	
9		attacked5
10		inspected10
11	attacked5	proved20, surveyed160
12		area20
15	surveyed160	area40
19		inspected20
20	proved20	
21	area40	attacked10
24		surveyed320
26		proved40
29	attacked10	
38		attacked20
42		area80
49	attacked20	
54		proved80
57	proved40	
64	attacked40	
68		attacked40
91		surveyed640
95	surveyed320	
99		attacked80
101		area160
104		proved160
116	attacked80	
187	attacked160	
219		attacked160
304	attacked320	
342	proved80	
468	parried5	
482		attacked320
520	attacked640	
588	area80	
593	proved160	
598	parried10	
705	parried20	area320, area640

Figure 487: Achievements.

32.2 Stability

Reason	PGIM	%	Python-DTU	%
failed away	95	0,63	39	0,26
failed parried			21	0,14
failed random	157	1,05	155	1,03
failed wrong param	19	0,13		
failed resources	25	0,17		
failed attacked	37	0,25	181	1,21

Figure 488: Failed actions.

32.3 Achievements

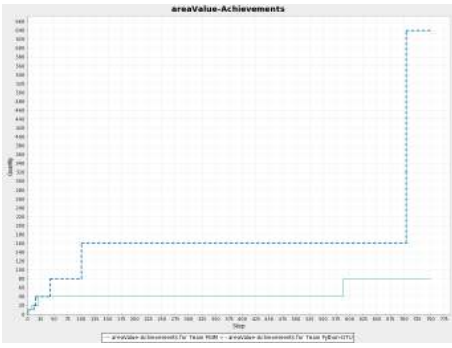


Figure 489: areaValueAchievements.

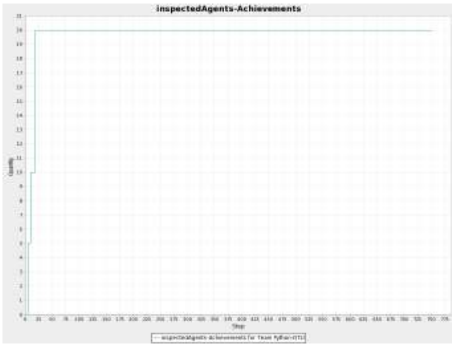


Figure 490: inspectedAgentsAchievements.

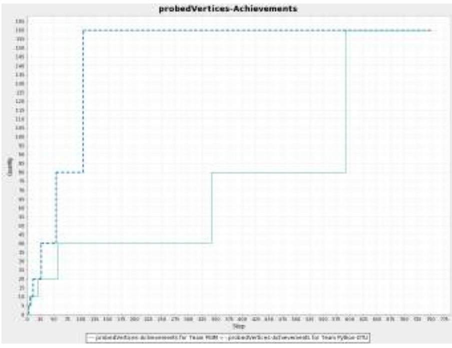


Figure 491: probedVerticesAchievements.

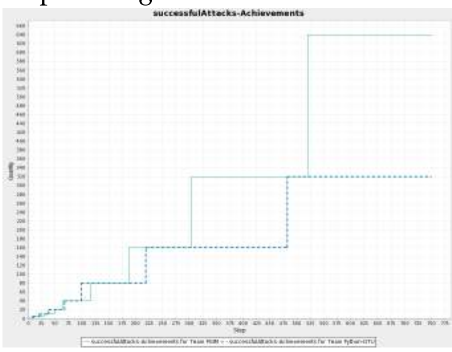


Figure 492: successfulAttacksAchievements.

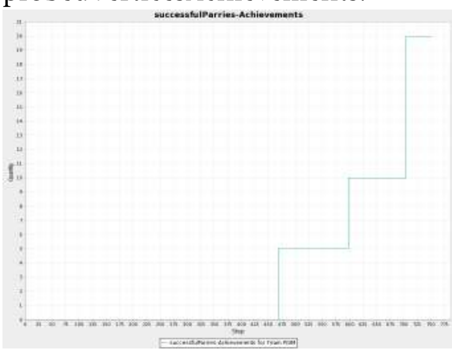


Figure 493: successfulParriesAchievements.

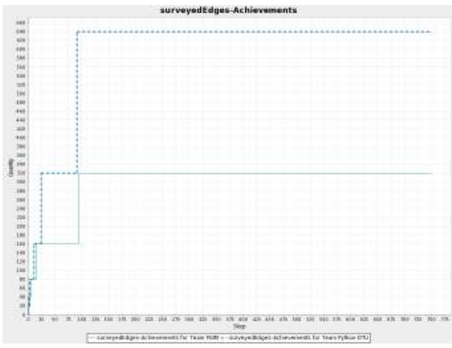


Figure 494: surveyedEdgesAchievements.

32.4 Actions per Role

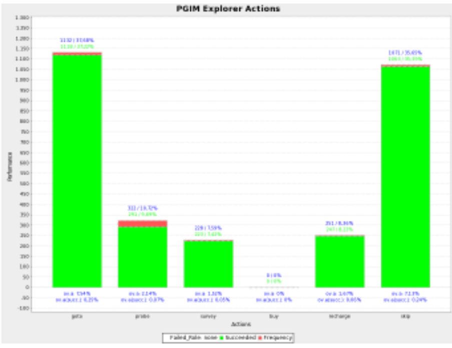


Figure 495: PGIM vs. Python-DTU – Simulation 2 - PGIM Explorer Actions.

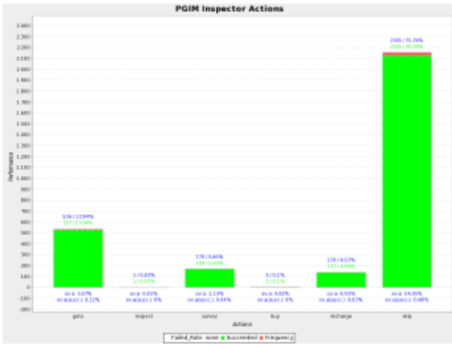


Figure 496: PGIM vs. Python-DTU – Simulation 2 - PGIM Inspector Actions.

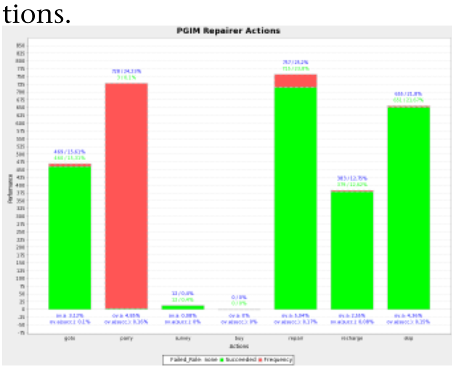


Figure 497: PGIM vs. Python-DTU – Simulation 2 - PGIM Repairer Actions.

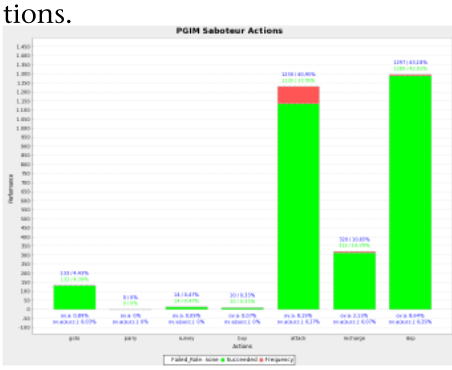


Figure 498: PGIM vs. Python-DTU – Simulation 2 - PGIM Saboteur Actions.

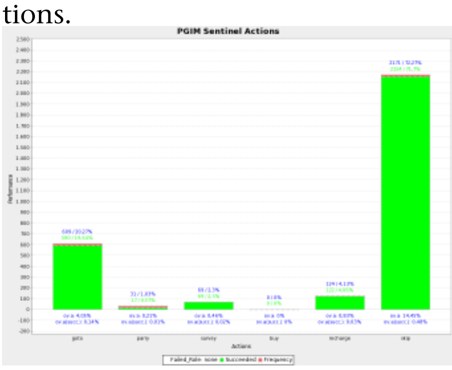


Figure 499: PGIM vs. Python-DTU – Simulation 2 - PGIM Sentinel Actions.

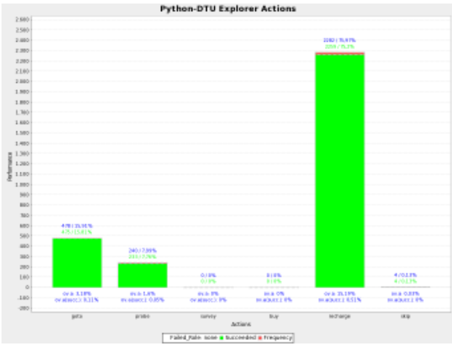


Figure 500: PGIM vs. Python-DTU – Simulation 2 - Python-DTU Explorer Actions.

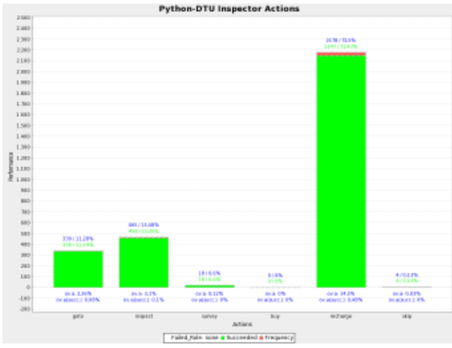


Figure 501: PGIM vs. Python-DTU – Simulation 2 - Python-DTU Inspector Actions.

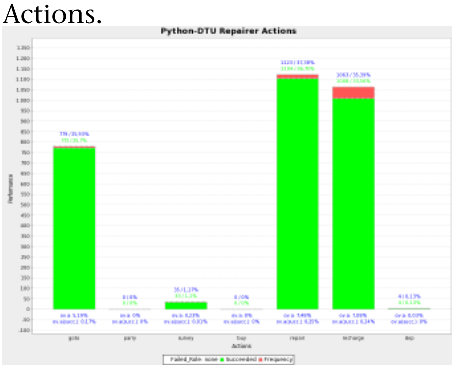


Figure 502: PGIM vs. Python-DTU – Simulation 2 - Python-DTU Repairer Actions.

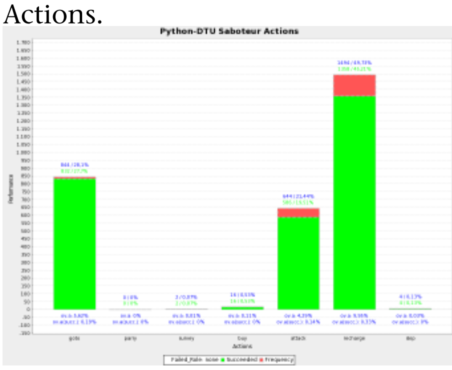


Figure 503: PGIM vs. Python-DTU – Simulation 2 - Python-DTU Saboteur Actions.

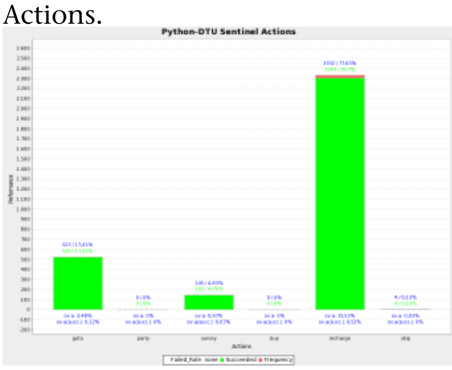


Figure 504: PGIM vs. Python-DTU – Simulation 2 - Python-DTU Sentinel Actions.

33 PGIM vs. Python-DTU – Simulation 3

33.1 Scores, Zone Stability and Achievements

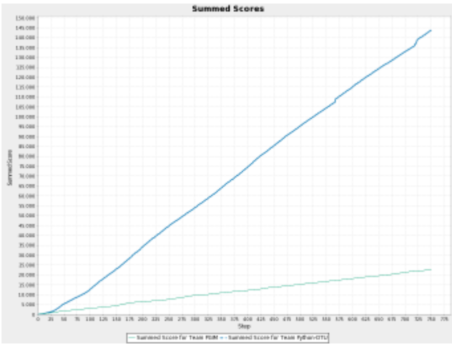


Figure 505: Summed scores.

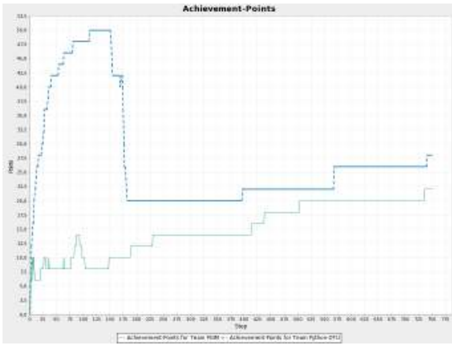


Figure 506: Achievement points.

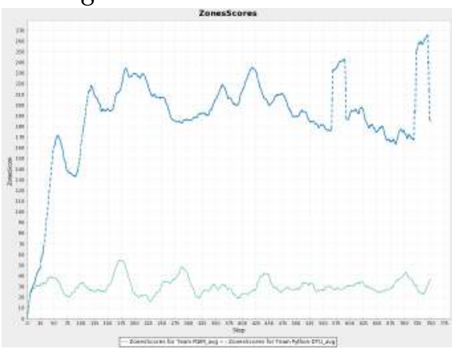


Figure 507: Zones scores.

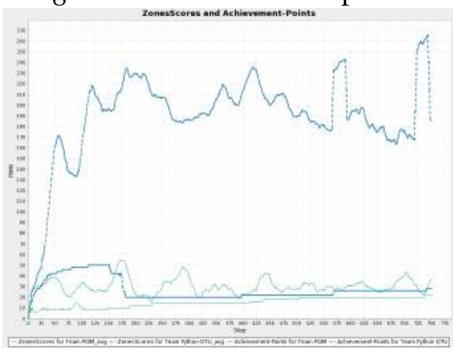


Figure 508: Zones scores and achievement points.

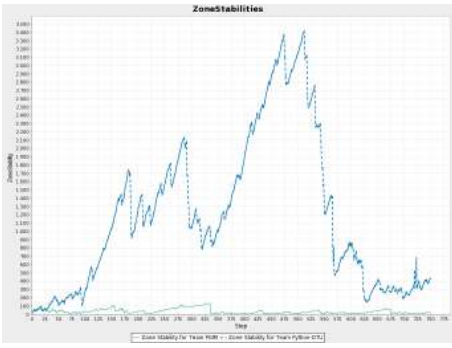


Figure 509: Zone Stabilities.

Step	PGIM	Python-DTU
1	surveyed10, surveyed20	surveyed10, surveyed20, inspected5
2	surveyed40, area10	
3		surveyed40, area10, proved5
4	area20	attacked5
5	surveyed80	surveyed80
6	proved5, attacked5	
8		inspected10, attacked10
9		proved10
11	proved10	area20
13		surveyed160
16		proved20
19	surveyed160	
22		attacked20
25		area40
26	proved20	
27		surveyed320, inspected20
33		area80
35	attacked10	proved40
36	area40	
40		area160
53		attacked40
63	attacked20	proved80
77	proved40	
80		attacked80
83	attacked40	
86	surveyed320	
111		proved160
147	attacked80	
170		attacked160
188	parried5	
229	attacked160	
396		attacked320
413	attacked320	
437	parried10	
503	proved80	
567		area320, area640
736	parried20	
741		attacked640

Figure 510: Achievements.

33.2 Stability

Reason	PGIM	%	Python-DTU	%
failed away	204	1,36	17	0,11
failed parried			23	0,15
failed random	155	1,03	134	0,89
failed wrong param	25	0,17		
failed	20	0,13		
failed resources	51	0,34		
failed attacked	62	0,41	281	1,87
noAction	20	0,13		

Figure 511: Failed actions.

33.3 Achievements

Figure 512: areaValueAchievements.

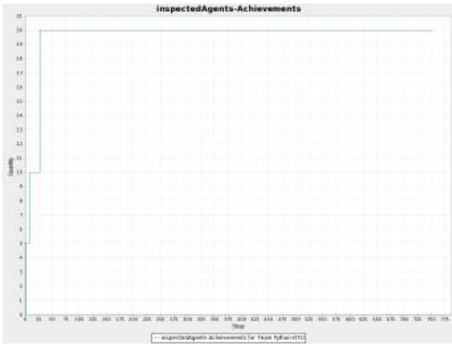


Figure 513: inspectedAgentsAchievements.

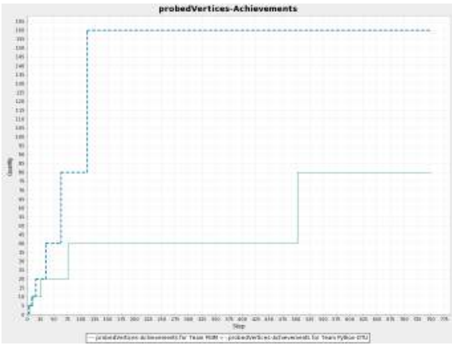


Figure 514: probedVerticesAchievements.

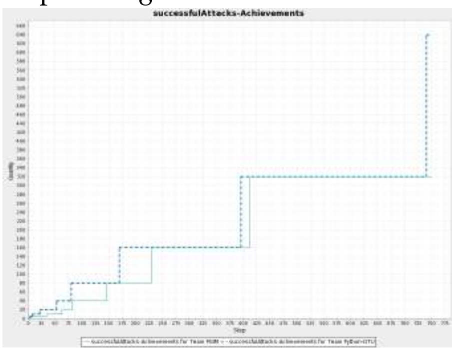


Figure 515: successfulAttacksAchievements.

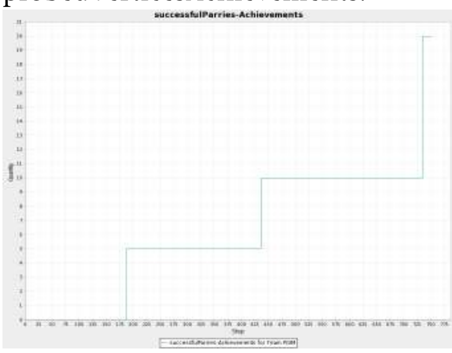


Figure 516: successfulParriesAchievements.

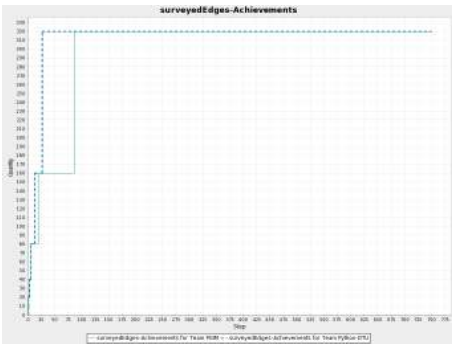


Figure 517: surveyedEdgesAchievements.

33.4 Actions per Role

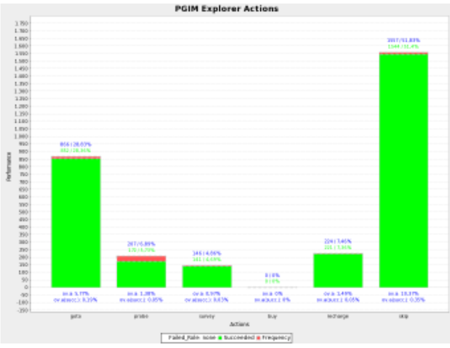


Figure 518: PGIM vs. Python-DTU – Simulation 3 - PGIM Explorer Actions.

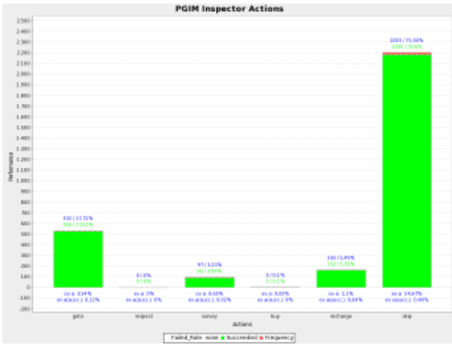


Figure 519: PGIM vs. Python-DTU – Simulation 3 - PGIM Inspector Actions.

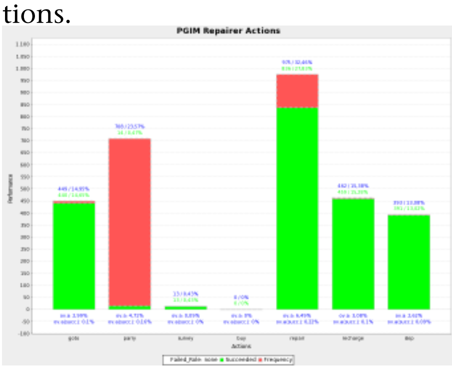


Figure 520: PGIM vs. Python-DTU – Simulation 3 - PGIM Repairer Actions.

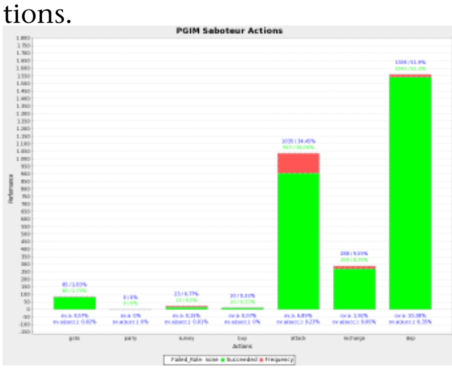


Figure 521: PGIM vs. Python-DTU – Simulation 3 - PGIM Saboteur Actions.

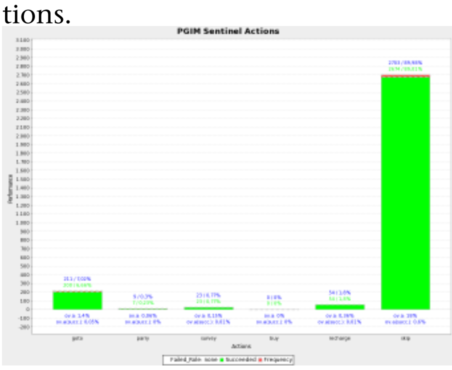


Figure 522: PGIM vs. Python-DTU – Simulation 3 - PGIM Sentinel Actions.

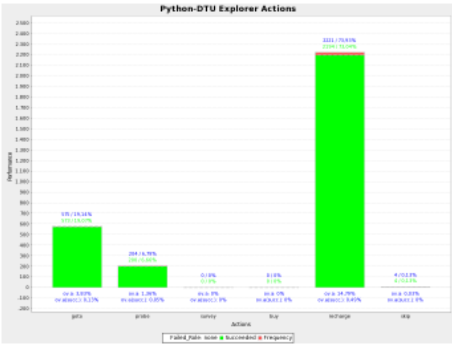


Figure 523: PGIM vs. Python-DTU – Simulation 3 - Python-DTU Explorer Actions.

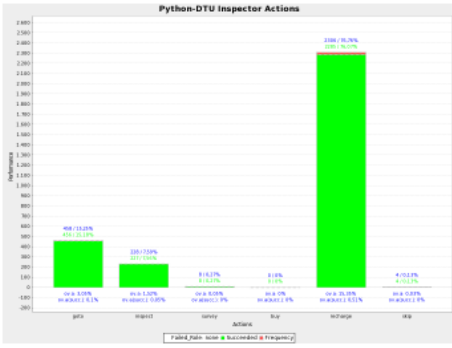


Figure 524: PGIM vs. Python-DTU – Simulation 3 - Python-DTU Inspector Actions.

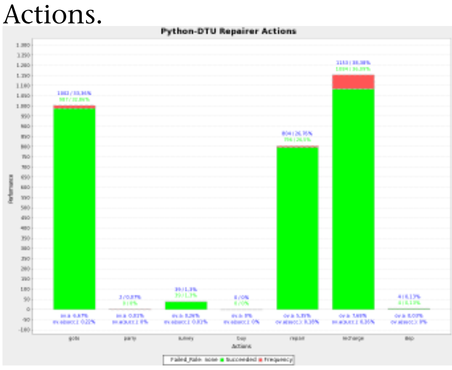


Figure 525: PGIM vs. Python-DTU – Simulation 3 - Python-DTU Repairer Actions.

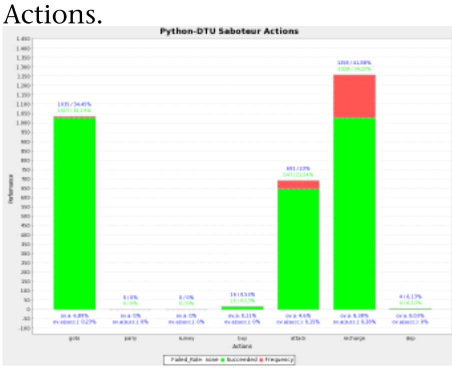


Figure 526: PGIM vs. Python-DTU – Simulation 3 - Python-DTU Saboteur Actions.

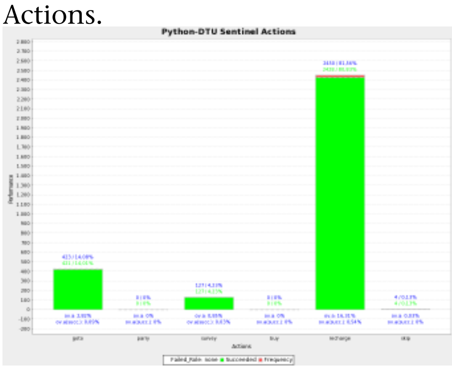


Figure 527: PGIM vs. Python-DTU – Simulation 3 - Python-DTU Sentinel Actions.

34 PGIM vs. Streett – Simulation 1

34.1 Scores, Zone Stability and Achievements

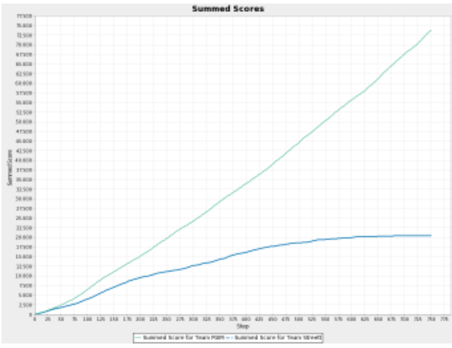


Figure 528: Summed scores.

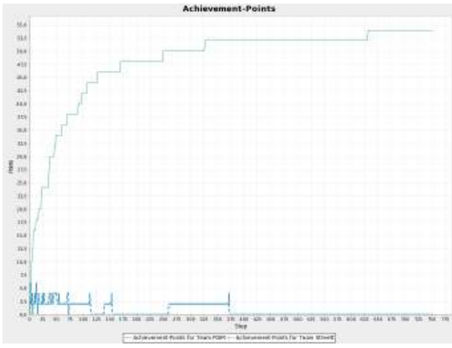


Figure 529: Achievement points.

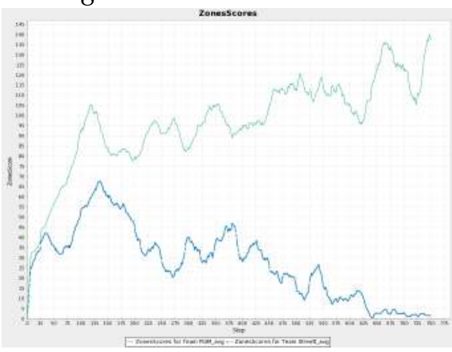


Figure 530: Zones scores.

Figure 531: Zones scores and achievement points.

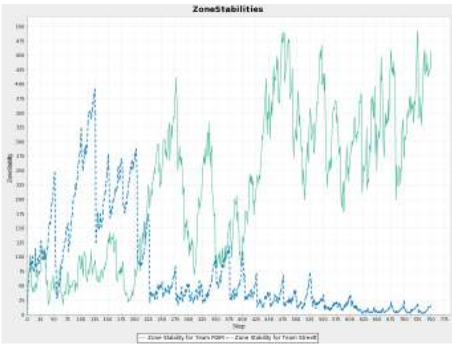


Figure 532: Zone Stabilities.

Step	PGIM	Streett
1	surveyed10, surveyed20	surveyed10, surveyed40, surveyed20
2	surveyed80, surveyed40	surveyed80
3	area10	
4		proved5
5	proved5	
6	area20, surveyed160	proved10
9		area10
11	proved10	attacked5
12		proved20
13		surveyed160
15		area20
16	attacked5	attacked10
17		inspected5
21	surveyed320	
22	attacked10	
23		attacked20
26		proved40
34	inspected5	
36	attacked20	inspected10
37	proved20	
41		surveyed320
44		attacked40
46	inspected10	
48	area40	
54		area40
60	attacked40	
69		proved80
70	parried5	
73		attacked80
90	proved40	
97	area80	
106	parried10	
112		inspected20
126	attacked80	
138		attacked160
152		proved160
169	parried20	
248	attacked160	
259		surveyed640
326	proved80	
371		attacked320
630	inspected20	

Figure 533: Achievements.

34.2 Stability

Reason	PGIM	%	Streett	%
failed away	12	0,08	62	0,41
failed parried			36	0,24
failed wrong param	223	1,49		
failed random	143	0,95	136	0,91
failed resources	2	0,01	507	3,38
failed attacked	38	0,25	94	0,63
failed status	1	0,01	3	0,02

Figure 534: Failed actions.

34.3 Achievements

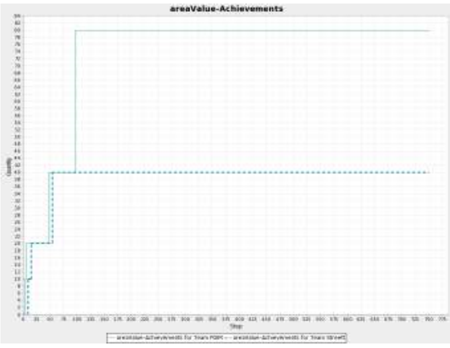


Figure 535: areaValueAchievements.

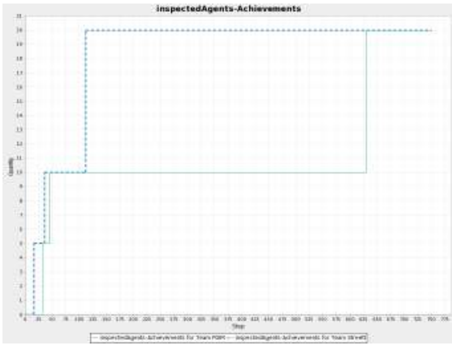


Figure 536: inspectedAgentsAchievements.

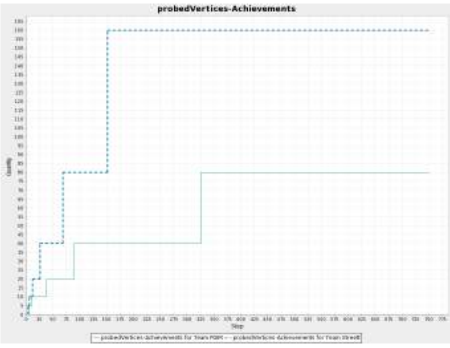


Figure 537: probedVerticesAchievements.

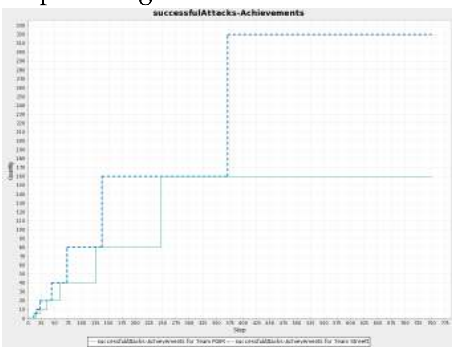


Figure 538: successfulAttacksAchievements.

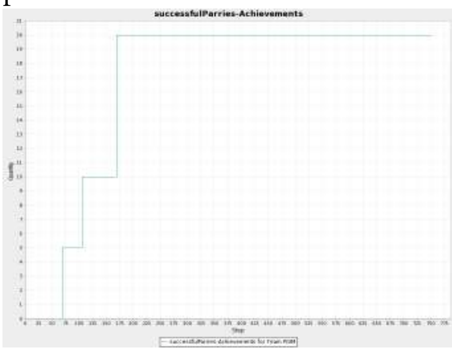


Figure 539: successfulParriesAchievements.

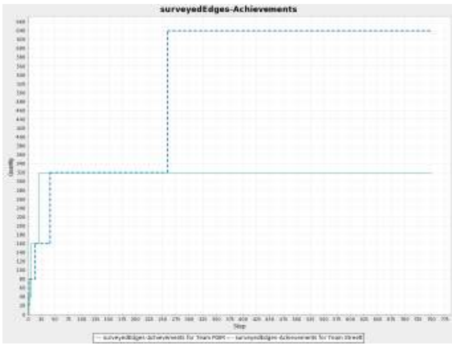


Figure 540: surveyedEdgesAchievements.

34.4 Actions per Role

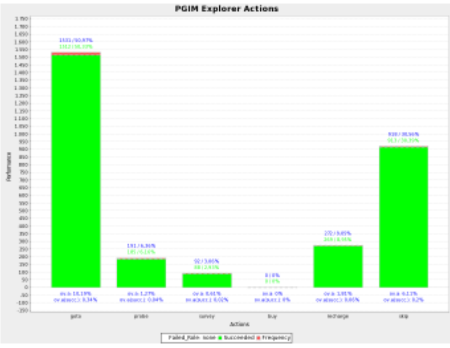


Figure 541: PGIM vs. Streett – Simulation 1 - PGIM Explorer Actions.

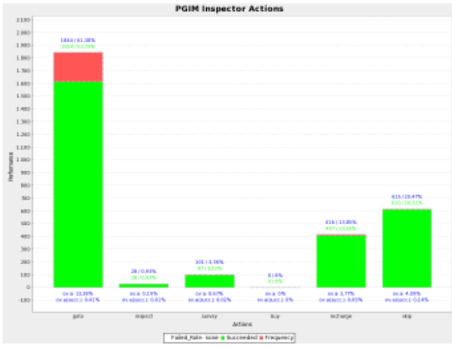


Figure 542: PGIM vs. Streett – Simulation 1 - PGIM Inspector Actions.

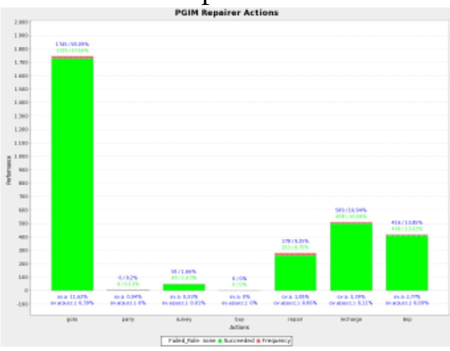


Figure 543: PGIM vs. Streett – Simulation 1 - PGIM Repairer Actions.

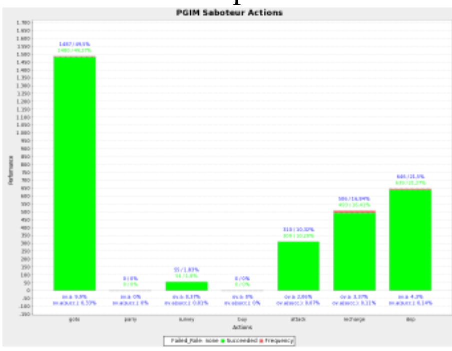


Figure 544: PGIM vs. Streett – Simulation 1 - PGIM Saboteur Actions.

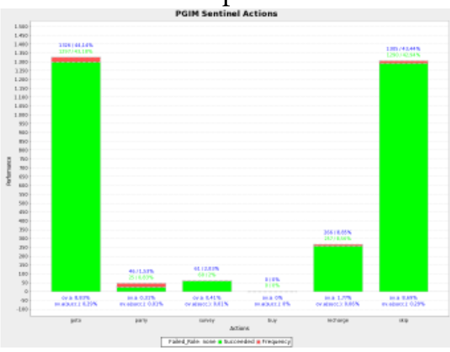


Figure 545: PGIM vs. Streett – Simulation 1 - PGIM Sentinel Actions.

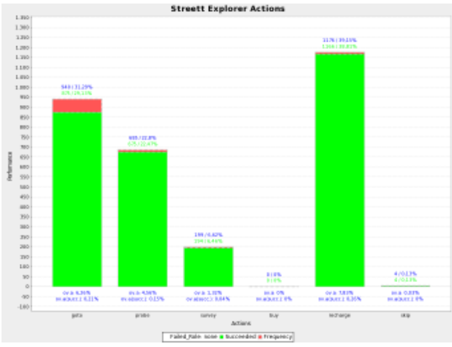


Figure 546: PGIM vs. Streett – Simulation 1 - Streett Explorer Actions.

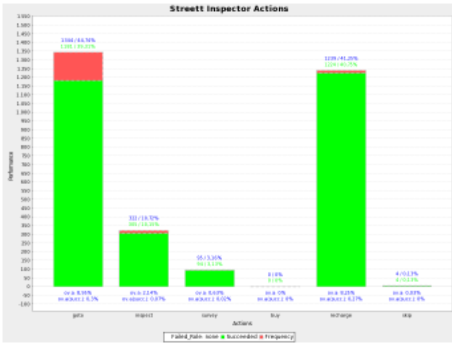


Figure 547: PGIM vs. Streett – Simulation 1 - Streett Inspector Actions.

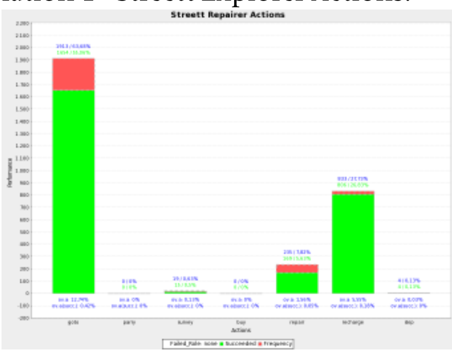


Figure 548: PGIM vs. Streett – Simulation 1 - Streett Repairer Actions.

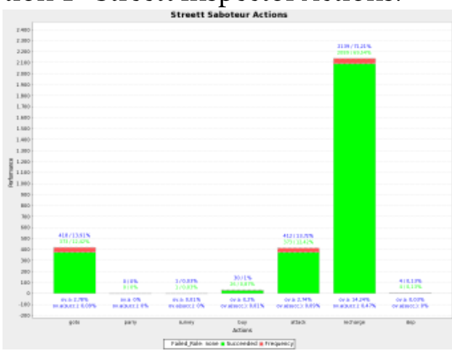


Figure 549: PGIM vs. Streett – Simulation 1 - Streett Saboteur Actions.

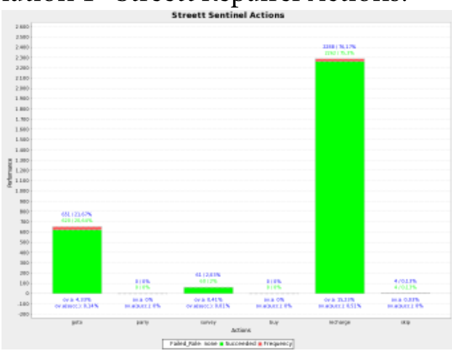


Figure 550: PGIM vs. Streett – Simulation 1 - Streett Sentinel Actions.

35 PGIM vs. Streett – Simulation 2

35.1 Scores, Zone Stability and Achievements

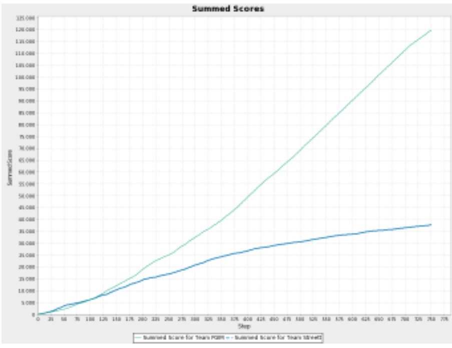


Figure 551: Summed scores.

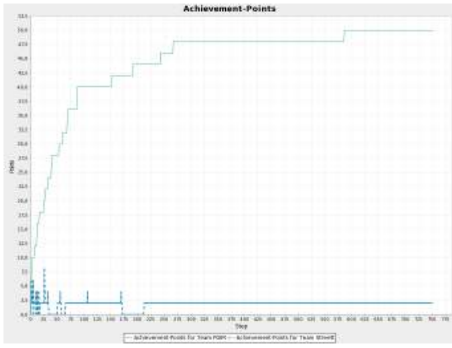


Figure 552: Achievement points.

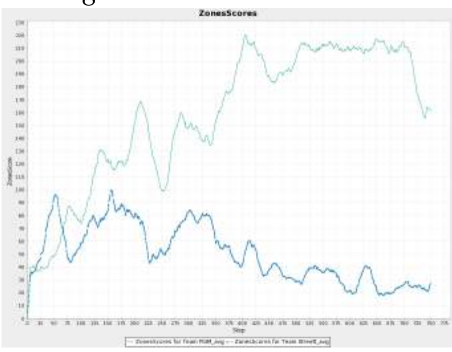


Figure 553: Zones scores.

Figure 554: Zones scores and achievement points.

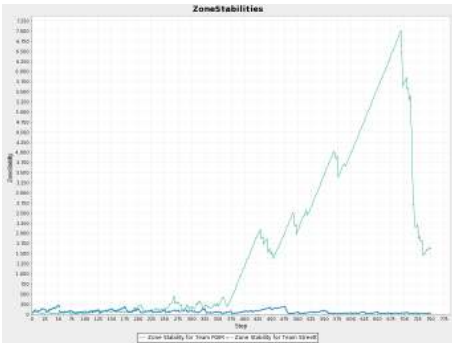


Figure 555: Zone Stabilities.

Step	PGIM	Streett
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed20, area10
2	surveyed80	surveyed40
3		proved5
4		area20, surveyed80
6		proved10
7	proved5	
9		attacked5
11	area20	proved20
12	surveyed160	inspected5
15		surveyed160, attacked10
16	proved10	
17		area40
25	attacked5	inspected10, proved40, attacked20
27	area40	
32	inspected5	area80
38	proved20	
40	attacked10	
50		attacked40
53	attacked20	
55		proved80
59	inspected10	
65		surveyed320
68	area80	
69	surveyed320	
86	attacked40, proved40	
106		attacked80
151	parried5	
169		proved160
191	attacked80	
212		inspected20
242	proved80	
266	area160	
586	attacked160	

Figure 556: Achievements.

35.2 Stability

Reason	PGIM	%	Streett	%
failed away	3	0,02	9	0,06
failed parried			11	0,07
failed random	141	0,94	154	1,03
failed wrong param	131	0,87		
failed	62	0,41		
failed resources	2	0,01	605	4,03
failed attacked	33	0,22	62	0,41
noAction	62	0,41		
failed status	2	0,01	1	0,01

Figure 557: Failed actions.

35.3 Achievements

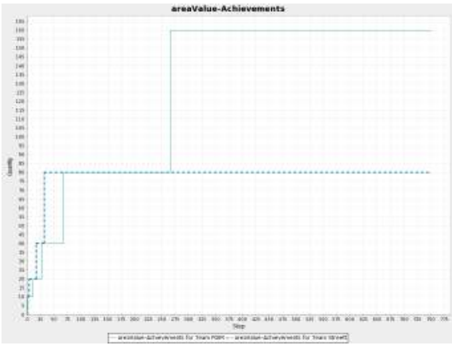


Figure 558: areaValueAchievements.

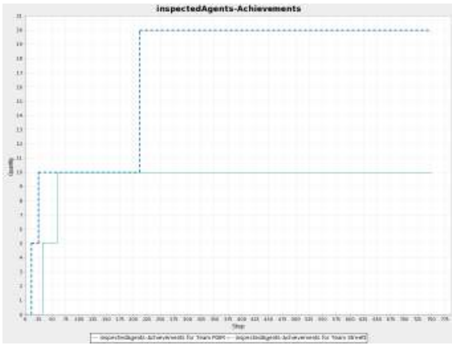


Figure 559: inspectedAgentsAchievements.

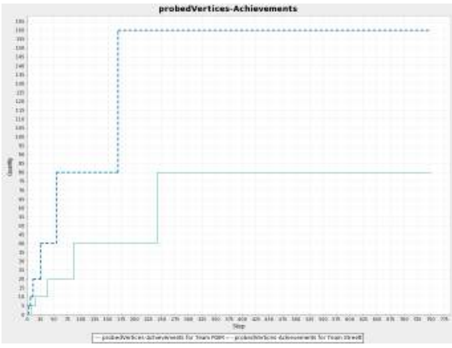


Figure 560: probedVerticesAchievements.

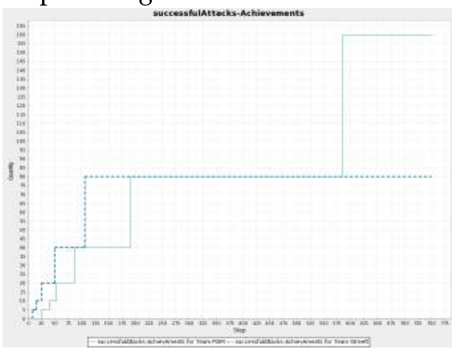


Figure 561: successfulAttacksAchievements.

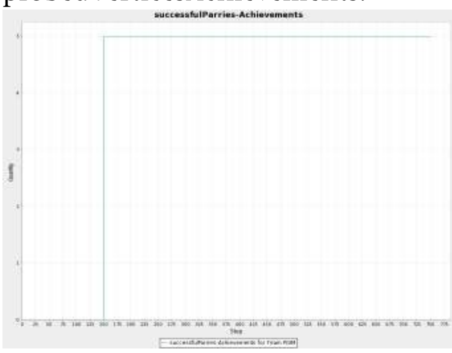


Figure 562: successfulParriesAchievements.

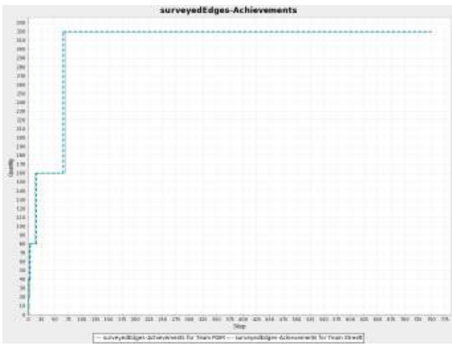


Figure 563: surveyedEdgesAchievements.

35.4 Actions per Role

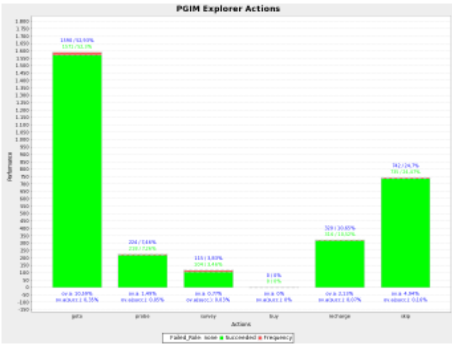


Figure 564: PGIM vs. Streett – Simulation 2 - PGIM Explorer Actions.

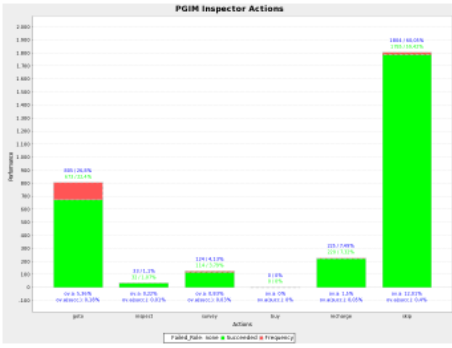


Figure 565: PGIM vs. Streett – Simulation 2 - PGIM Inspector Actions.

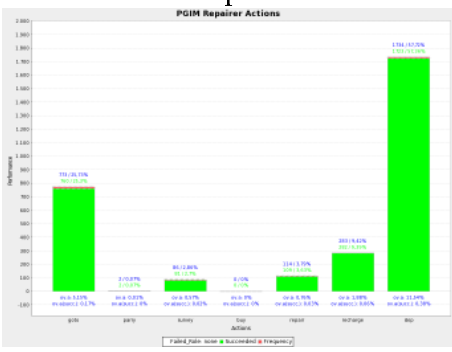


Figure 566: PGIM vs. Streett – Simulation 2 - PGIM Repairer Actions.

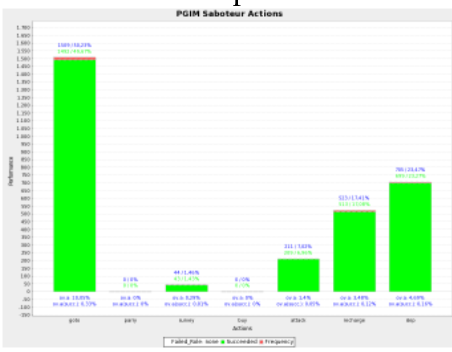


Figure 567: PGIM vs. Streett – Simulation 2 - PGIM Saboteur Actions.

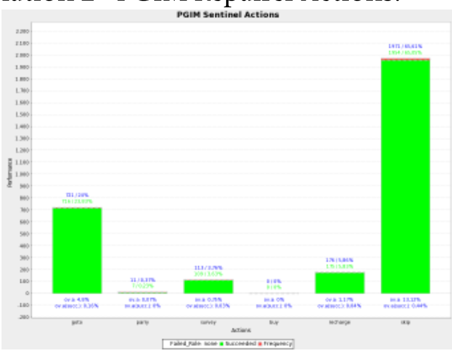


Figure 568: PGIM vs. Streett – Simulation 2 - PGIM Sentinel Actions.

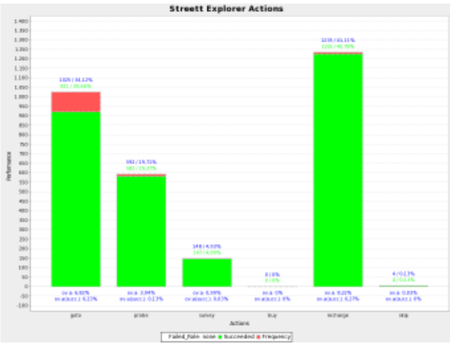


Figure 569: PGIM vs. Streett – Simulation 2 - Streett Explorer Actions.

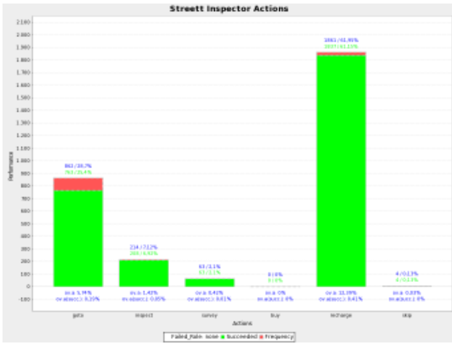


Figure 570: PGIM vs. Streett – Simulation 2 - Streett Inspector Actions.

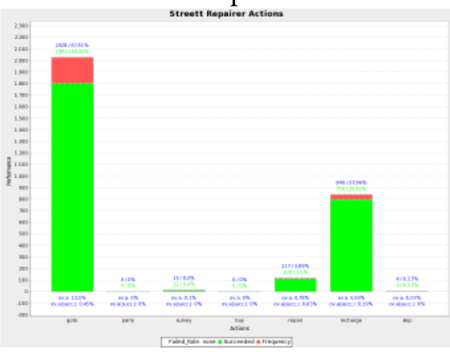


Figure 571: PGIM vs. Streett – Simulation 2 - Streett Repairer Actions.

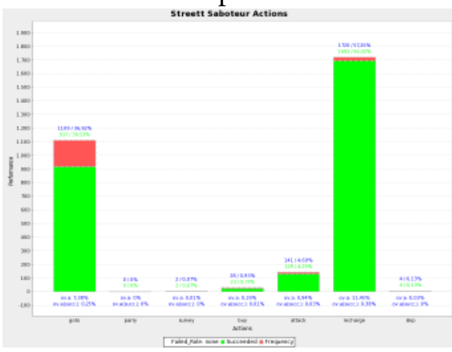


Figure 572: PGIM vs. Streett – Simulation 2 - Streett Saboteur Actions.

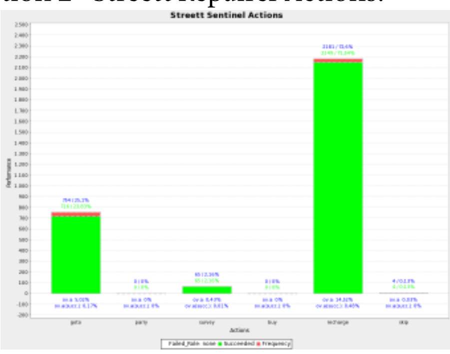


Figure 573: PGIM vs. Streett – Simulation 2 - Streett Sentinel Actions.

36 PGIM vs. Streett – Simulation 3

36.1 Scores, Zone Stability and Achievements

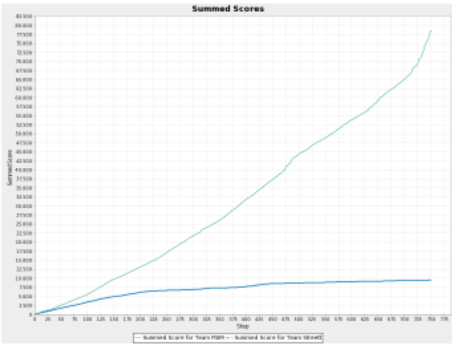


Figure 574: Summed scores.

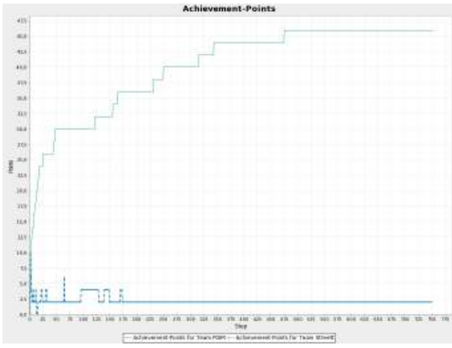


Figure 575: Achievement points.

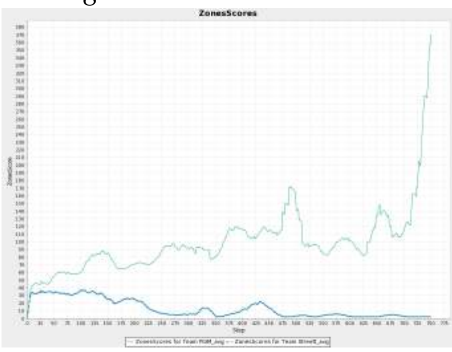


Figure 576: Zones scores.

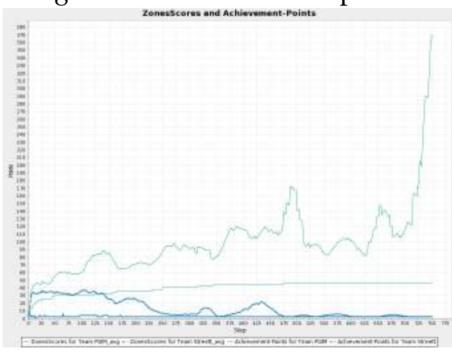


Figure 577: Zones scores and achievement points.

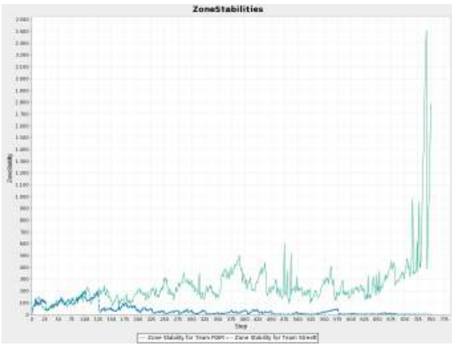


Figure 578: Zone Stabilities.

Step	PGIM	Streett
1	surveyed40, surveyed10, area20, surveyed20, area10	surveyed10, area20, surveyed20, area10, inspected5
2	surveyed80	surveyed40
3		proved5
4	proved5	surveyed80
6		proved10
7	proved10	inspected10
9	surveyed160	
11		proved20
12	area40	
15	proved20	attacked5
17	attacked5	
21		surveyed160
22		attacked10
25	attacked10	
29		proved40
31		attacked20
44	proved40	
47	attacked20	
64		attacked40, proved80
95		surveyed320
122	attacked40	
138		inspected20
155	surveyed320	
164	proved80	
169		proved160
231	attacked80	
249	area80	
314	area160	
343	proved160	
474	area320	

Figure 579: Achievements.

36.2 Stability

Reason	PGIM	%	Streett	%
failed away	3	0,02	22	0,15
failed parried			4	0,03
failed random	138	0,92	153	1,02
failed wrong param	19	0,13	1	0,01
failed resources			367	2,45
failed attacked	10	0,07	52	0,35

Figure 580: Failed actions.

36.3 Achievements

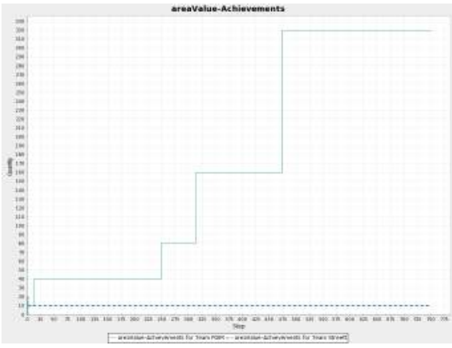


Figure 581: areaValueAchievements.

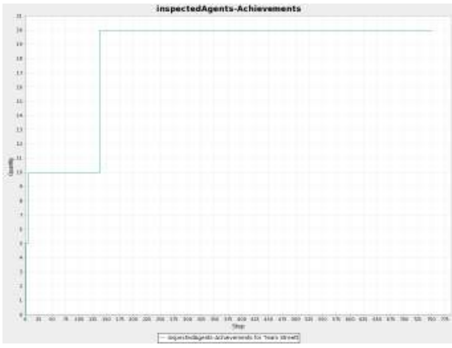


Figure 582: inspectedAgentsAchievements.

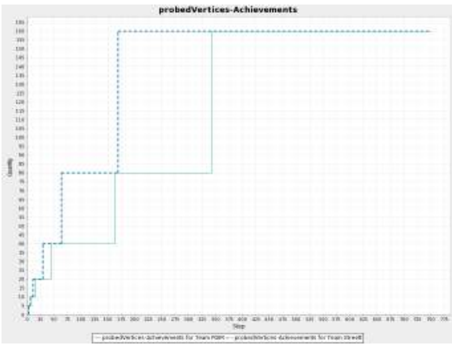


Figure 583: probedVerticesAchievements.

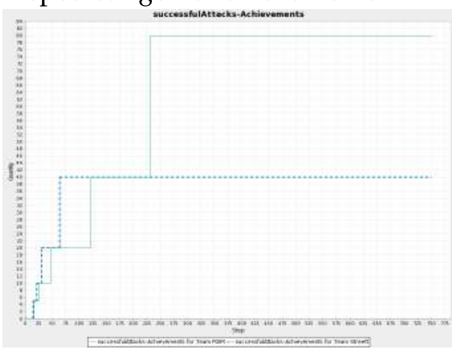


Figure 584: successfulAttacksAchievements.

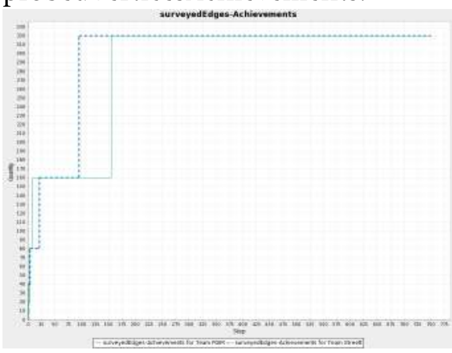


Figure 585: surveyedEdgesAchievements.

36.4 Actions per Role

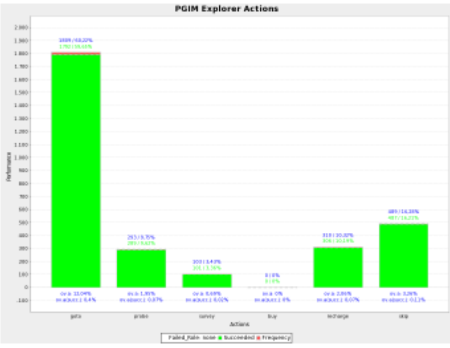


Figure 586: PGIM vs. Streett – Simulation 3 - PGIM Explorer Actions.

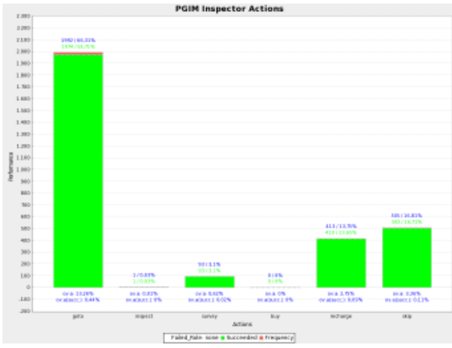


Figure 587: PGIM vs. Streett – Simulation 3 - PGIM Inspector Actions.

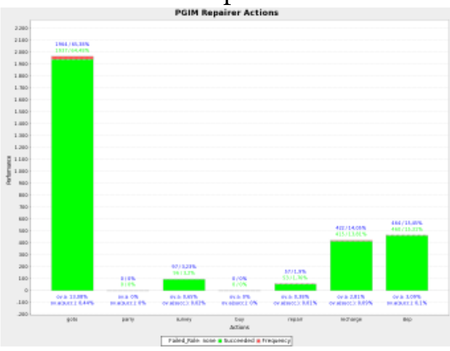


Figure 588: PGIM vs. Streett – Simulation 3 - PGIM Repairer Actions.

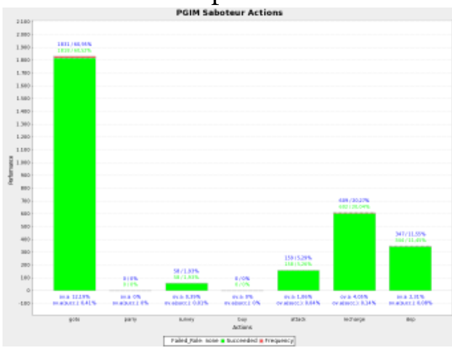


Figure 589: PGIM vs. Streett – Simulation 3 - PGIM Saboteur Actions.

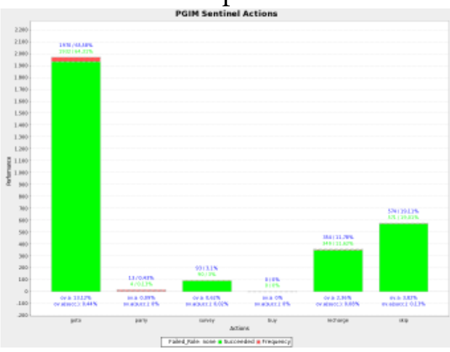


Figure 590: PGIM vs. Streett – Simulation 3 - PGIM Sentinel Actions.

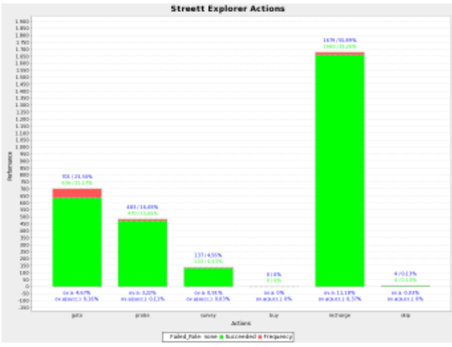


Figure 591: PGIM vs. Streett – Simulation 3 - Streett Explorer Actions.

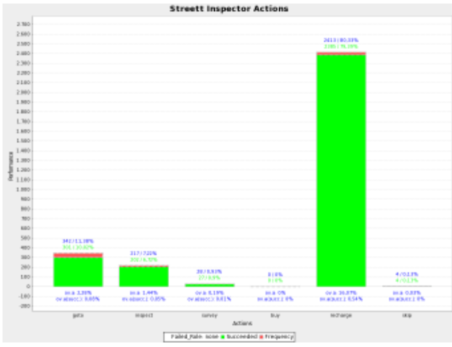


Figure 592: PGIM vs. Streett – Simulation 3 - Streett Inspector Actions.

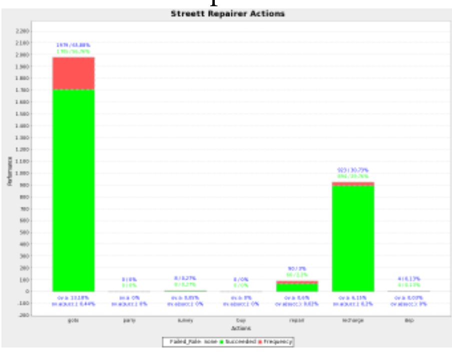


Figure 593: PGIM vs. Streett – Simulation 3 - Streett Repairer Actions.

Figure 594: PGIM vs. Streett – Simulation 3 - Streett Saboteur Actions.

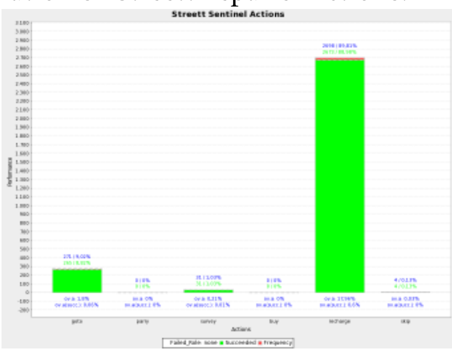


Figure 595: PGIM vs. Streett – Simulation 3 - Streett Sentinel Actions.

37 PGIM vs. TUB – Simulation 1

37.1 Scores, Zone Stability and Achievements

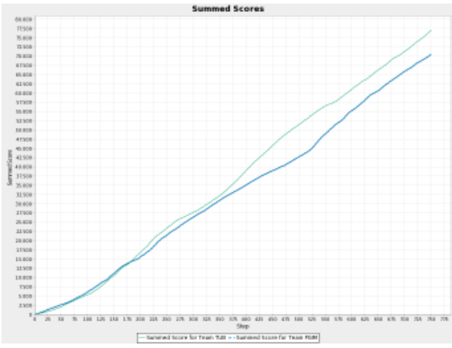


Figure 596: Summed scores.

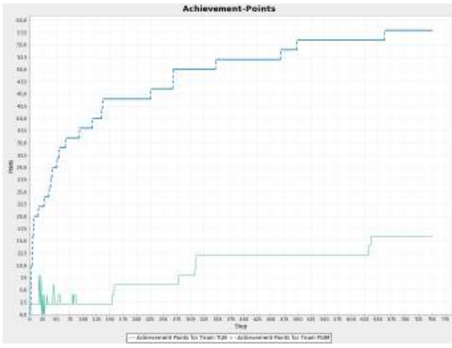


Figure 597: Achievement points.

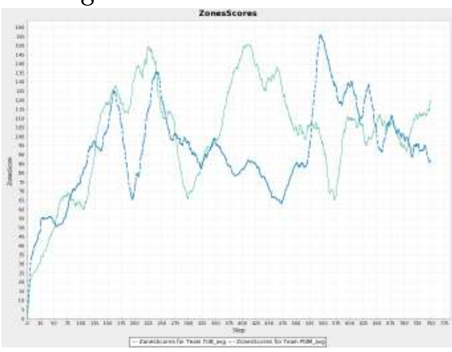


Figure 598: Zones scores.

Figure 599: Zones scores and achievement points.

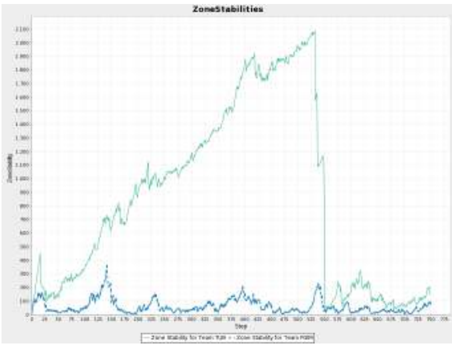


Figure 600: Zone Stabilities.

Step	TUB	PGIM
1	area10	surveyed10
2		surveyed40, surveyed20
3		surveyed80, area10
5		area40, area20, proved5
7		surveyed160
8		proved10
16		proved20
17	surveyed10, surveyed40, surveyed20	
18	area20	
20	surveyed80, attacked5, inspected5	
23	inspected10, proved5	
26	area40, attacked10	
27		inspected5
28	proved10	
32	surveyed160	
33	attacked20	
36		inspected10
39		attacked5
42		proved40
43	proved20, inspected20	
51		surveyed320
53	attacked40	
55		attacked10
57	area80	
67		attacked20
79	attacked80	
82	proved40	
93		attacked40
117		area80
134		proved80
136		attacked80
154	area160	
157	attacked160	
226		parried5
267		parried10, attacked160
277	attacked320	
308	surveyed320	
310	proved80	
346		parried20
468		attacked320
498		proved160
631	attacked640	
636	proved160	
662		parried40

Figure 601: Achievements.

37.2 Stability

Reason	TUB	%	PGIM	%
failed away	1	0,01	14	0,09
failed parried	59	0,39		
failed wrong param	425	2,83	196	1,31
failed random	158	1,05	152	1,01
failed resources	1	0,01	3	0,02
failed	725	4,83		
failed attacked	108	0,72	68	0,45
noAction	731	4,87		
failed status			2	0,01

Figure 602: Failed actions.

37.3 Achievements

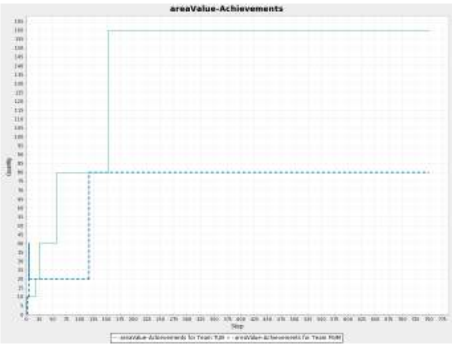


Figure 603: areaValueAchievements.

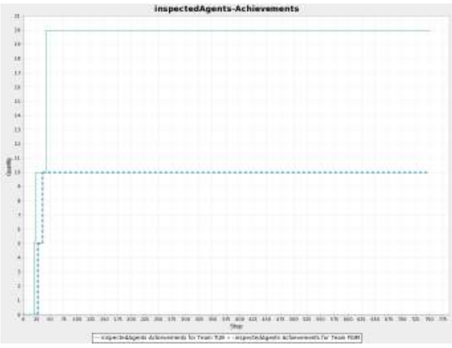


Figure 604: inspectedAgentsAchievements.

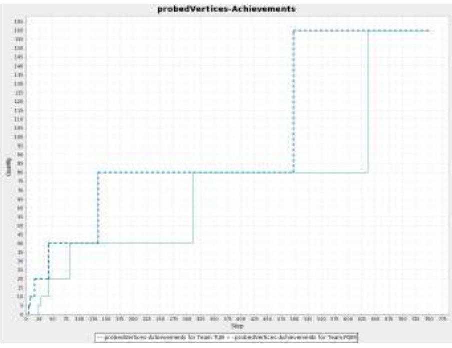


Figure 605: probedVerticesAchievements.

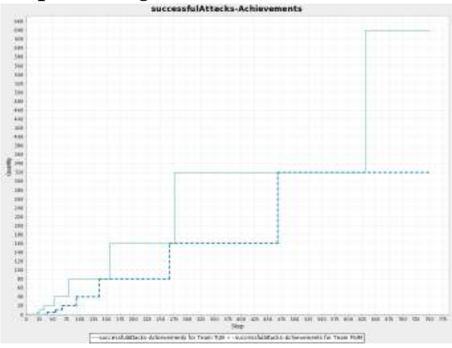


Figure 606: successfulAttacksAchievements.

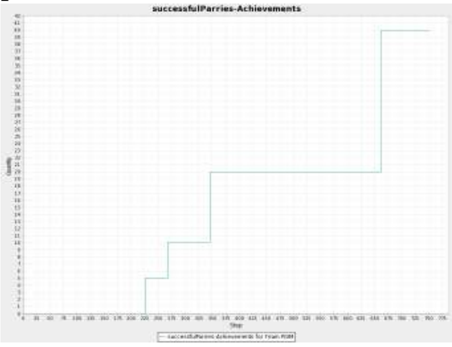


Figure 607: successfulParriesAchievements.

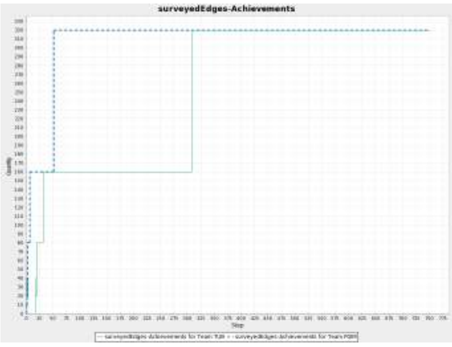


Figure 608: surveyedEdgesAchievements.

37.4 Actions per Role

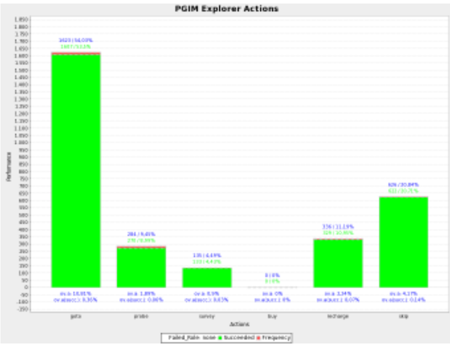


Figure 609: PGIM vs. TUB – Simulation 1 - PGIM Explorer Actions.

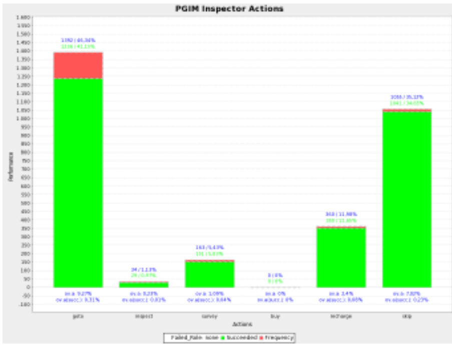


Figure 610: PGIM vs. TUB – Simulation 1 - PGIM Inspector Actions.

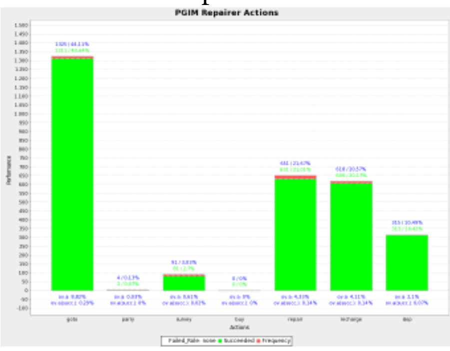


Figure 611: PGIM vs. TUB – Simulation 1 - PGIM Repairer Actions.

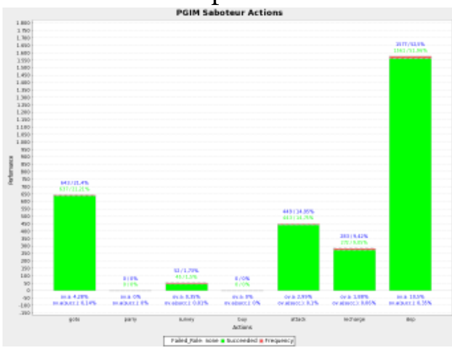


Figure 612: PGIM vs. TUB – Simulation 1 - PGIM Saboteur Actions.

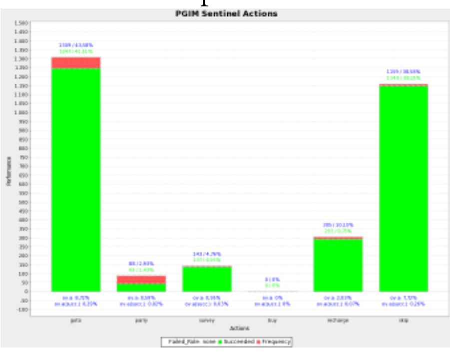


Figure 613: PGIM vs. TUB – Simulation 1 - PGIM Sentinel Actions.

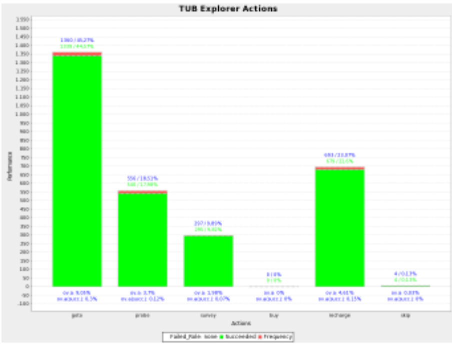


Figure 614: PGIM vs. TUB – Simulation 1 - TUB Explorer Actions.

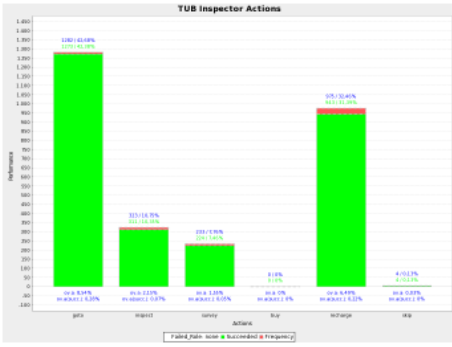


Figure 615: PGIM vs. TUB – Simulation 1 - TUB Inspector Actions.

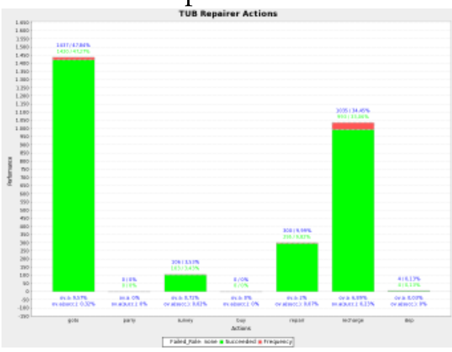


Figure 616: PGIM vs. TUB – Simulation 1 - TUB Repairer Actions.

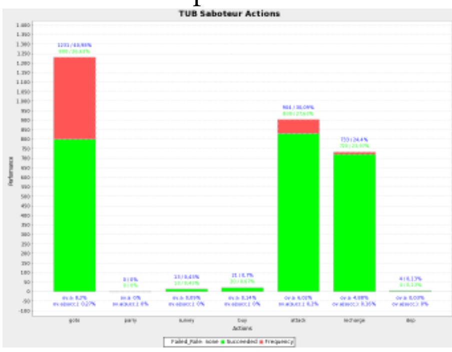


Figure 617: PGIM vs. TUB – Simulation 1 - TUB Saboteur Actions.

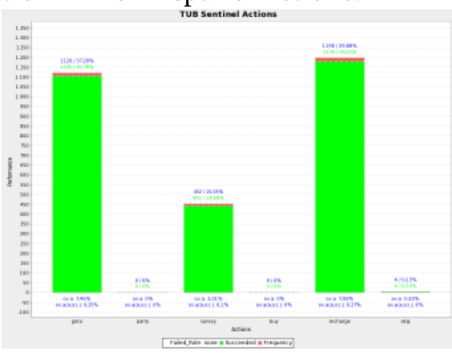


Figure 618: PGIM vs. TUB – Simulation 1 - TUB Sentinel Actions.

38 PGIM vs. TUB – Simulation 2

38.1 Scores, Zone Stability and Achievements

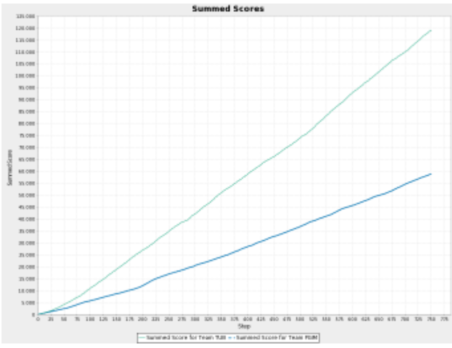


Figure 619: Summed scores.

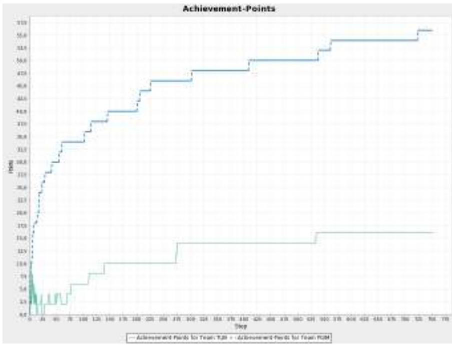


Figure 620: Achievement points.

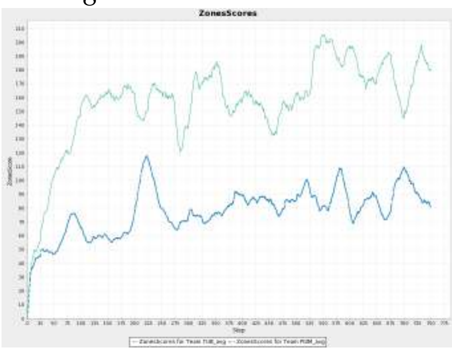


Figure 621: Zones scores.

Figure 622: Zones scores and achievement points.

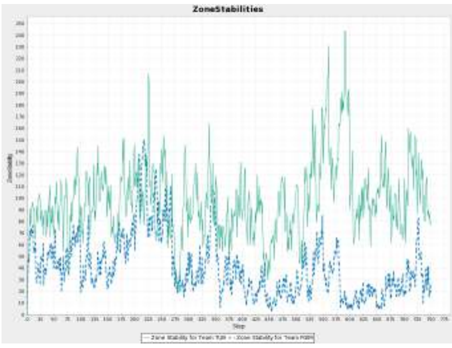


Figure 623: Zone Stabilities.

Step	TUB	PGIM
1	surveyed10, surveyed80, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
2		surveyed80
4	area40, area20, proved5	proved5
5		area20, surveyed160
7	inspected5	
8	proved10, surveyed160, attacked5	proved10
10	inspected10	
12	area80	
14		area40
15	attacked10	
17		proved20, surveyed320
21	proved20	
22		inspected5
27	attacked20	
28		attacked5
34	surveyed320	
35	inspected20	
41		attacked10
47	attacked40	
51	proved40	
55		proved40
60		attacked20
69	area160	
77	attacked80	
102		parried5
110	proved80	
114		attacked40
139	attacked160	
145		parried10
200		attacked80
205		area80
225		proved80
273	attacked320	
275	proved160	
302		parried20
408		attacked160
534	attacked640	
538		parried40
561		proved160
723		attacked320

Figure 624: Achievements.

38.2 Stability

Reason	TUB	%	PGIM	%
failed away	1	0,01	14	0,09
failed parried	84	0,56		
failed random	148	0,99	153	1,02
failed wrong param	2	0,01	53	0,35
failed attacked	42	0,28	68	0,45

Figure 625: Failed actions.

38.3 Achievements

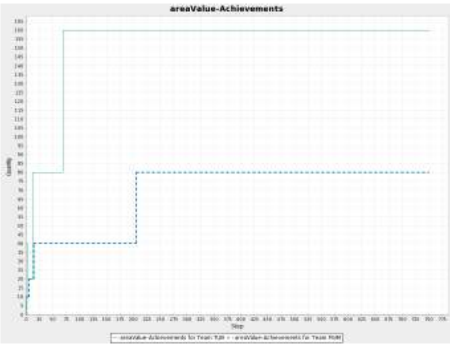


Figure 626: areaValueAchievements.

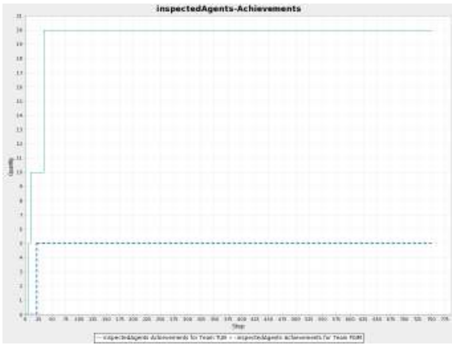


Figure 627: inspectedAgentsAchievements.

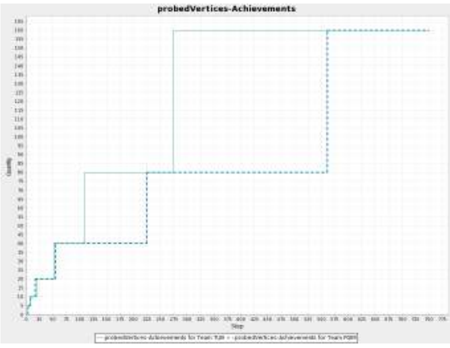


Figure 628: probedVerticesAchievements.

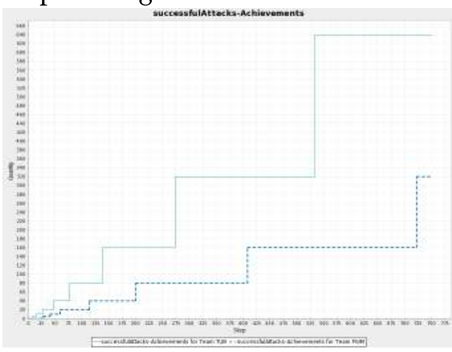


Figure 629: successfulAttacksAchievements.

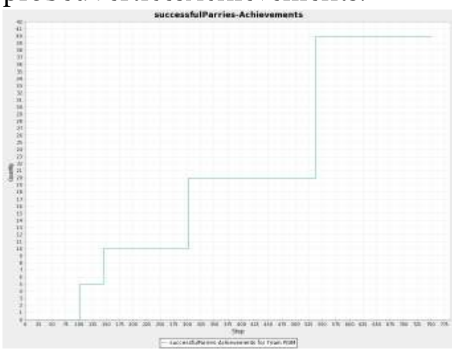


Figure 630: successfulParriesAchievements.

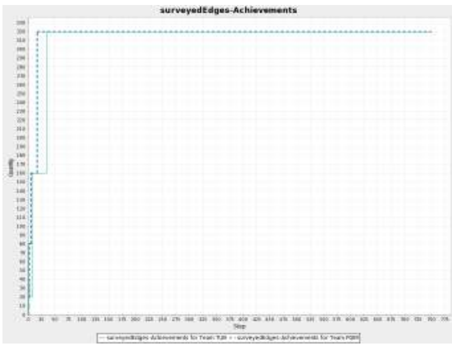


Figure 631: surveyedEdgesAchievements.

38.4 Actions per Role

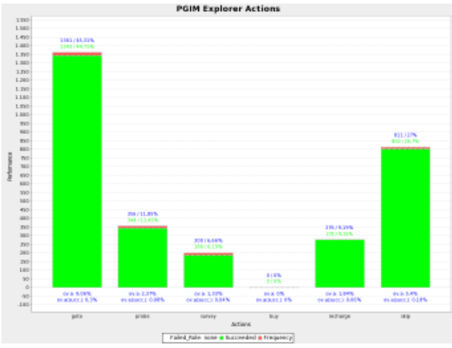


Figure 632: PGIM vs. TUB – Simulation 2 - PGIM Explorer Actions.

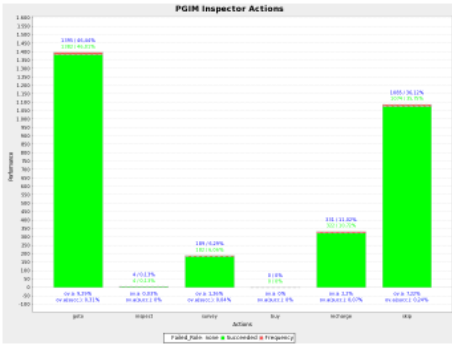


Figure 633: PGIM vs. TUB – Simulation 2 - PGIM Inspector Actions.

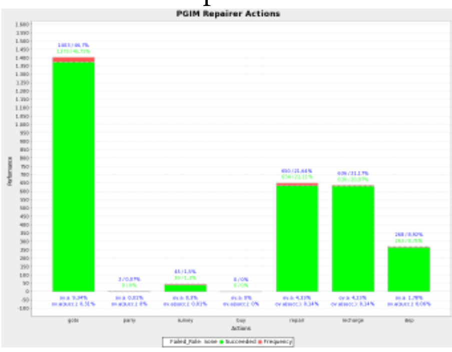


Figure 634: PGIM vs. TUB – Simulation 2 - PGIM Repairer Actions.

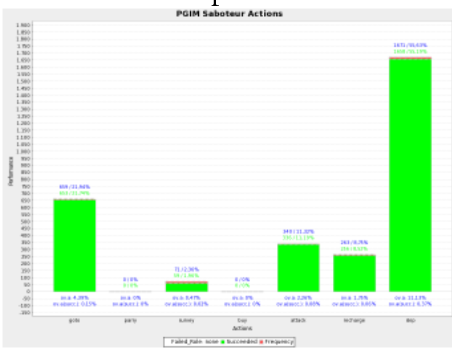


Figure 635: PGIM vs. TUB – Simulation 2 - PGIM Saboteur Actions.

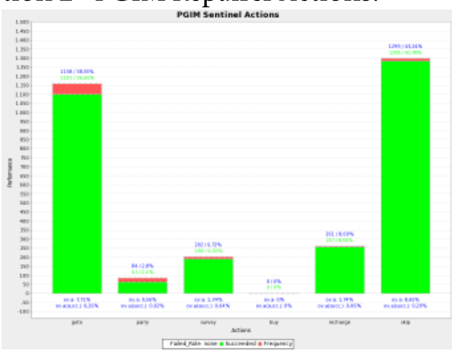


Figure 636: PGIM vs. TUB – Simulation 2 - PGIM Sentinel Actions.

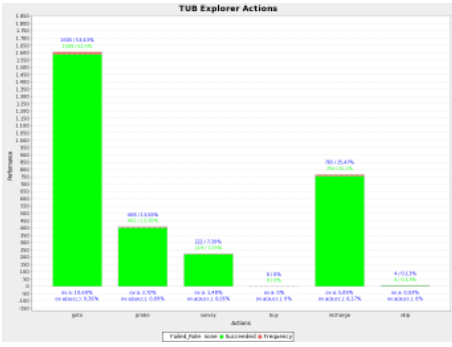


Figure 637: PGIM vs. TUB – Simulation 2 - TUB Explorer Actions.

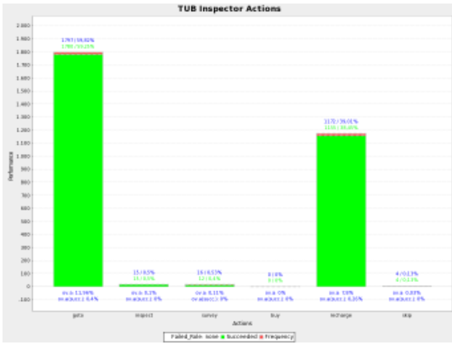


Figure 638: PGIM vs. TUB – Simulation 2 - TUB Inspector Actions.

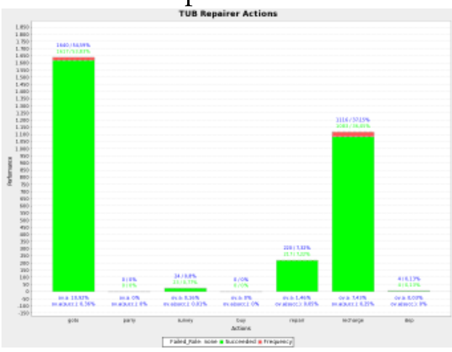


Figure 639: PGIM vs. TUB – Simulation 2 - TUB Repairer Actions.

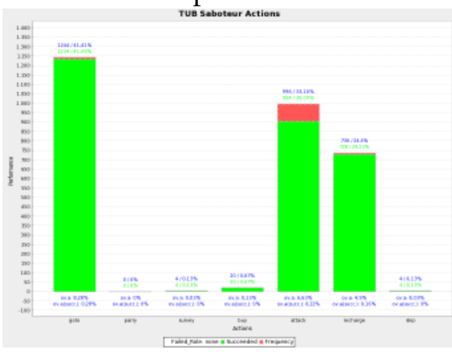


Figure 640: PGIM vs. TUB – Simulation 2 - TUB Saboteur Actions.

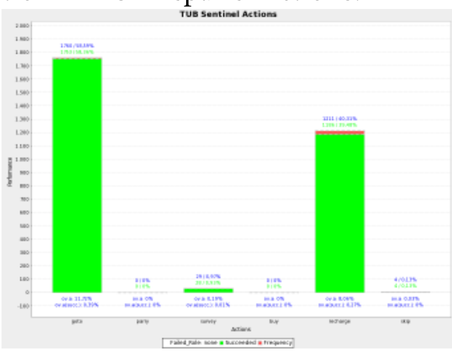


Figure 641: PGIM vs. TUB – Simulation 2 - TUB Sentinel Actions.

39 PGIM vs. TUB – Simulation 3

39.1 Scores, Zone Stability and Achievements

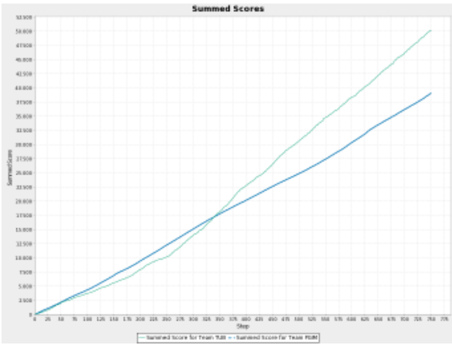


Figure 642: Summed scores.

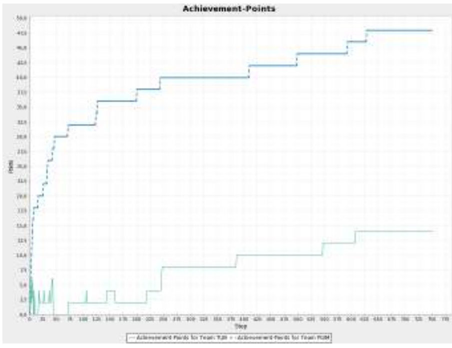


Figure 643: Achievement points.

Figure 644: Zones scores.

Figure 645: Zones scores and achievement points.

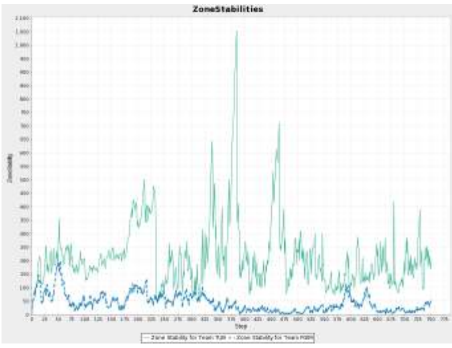


Figure 646: Zone Stabilities.

Step	TUB	PGIM
1	surveyed10, surveyed40, surveyed20	surveyed40, surveyed10, surveyed20
2	surveyed80, area10	surveyed80
3	inspected5	area10
4	proved5	area20, proved5
5	area20	surveyed160
7		proved10
8	proved10, surveyed160	
15		proved20
16	attacked5	
17	inspected10	
18	proved20	
25		attacked5
26	attacked10	
27	area40	
32		surveyed320
33		attacked10
36	proved40	
41	attacked20, area80	
42		attacked20
46		proved40
71		attacked40
72	attacked40	
105	proved80	
123		proved80
126		attacked80
144	attacked80	
199		parried5
218	inspected20	
243		attacked160
245	surveyed320	
246	attacked160	
385	attacked320	
409		parried10
498		parried20
546	area160	
592		attacked320
607	attacked640	
628		proved160

Figure 647: Achievements.

39.2 Stability

Reason	TUB	%	PGIM	%
failed away	5	0,03	11	0,07
failed parried	42	0,28		
failed random	153	1,02	168	1,12
failed wrong param	358	2,39	55	0,37
failed	404	2,69		
failed resources	7	0,05	3	0,02
failed attacked	82	0,55	49	0,33
noAction	411	2,74		

Figure 648: Failed actions.

39.3 Achievements

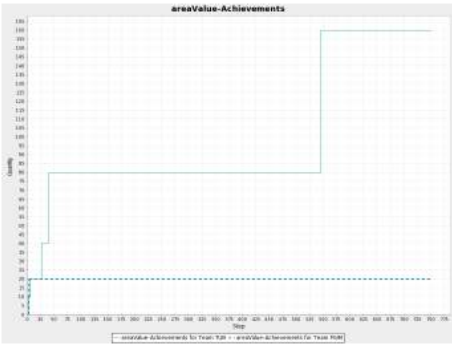


Figure 649: areaValueAchievements.

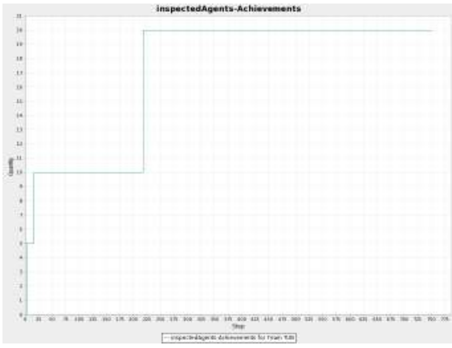


Figure 650: inspectedAgentsAchievements.

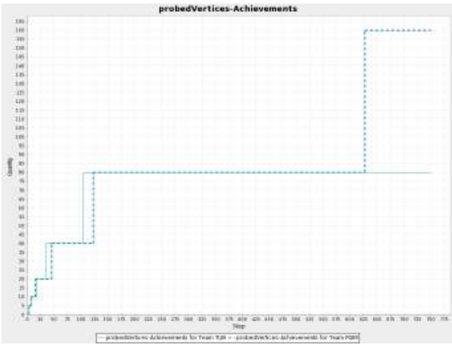


Figure 651: probedVerticesAchievements.

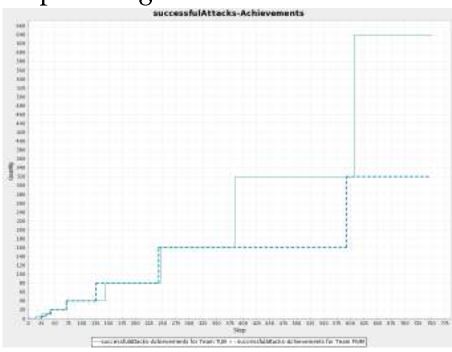


Figure 652: successfulAttacksAchievements.

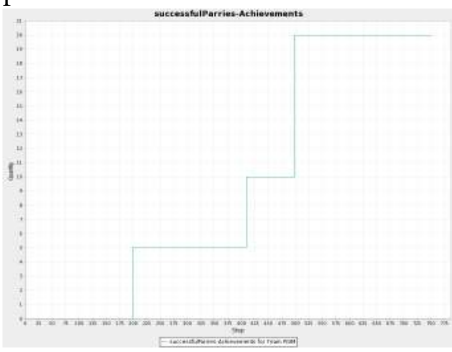


Figure 653: successfulParriesAchievements.

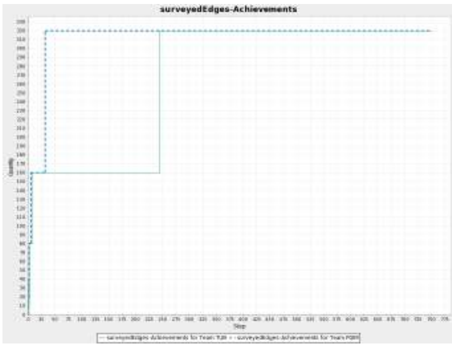


Figure 654: surveyedEdgesAchievements.

39.4 Actions per Role

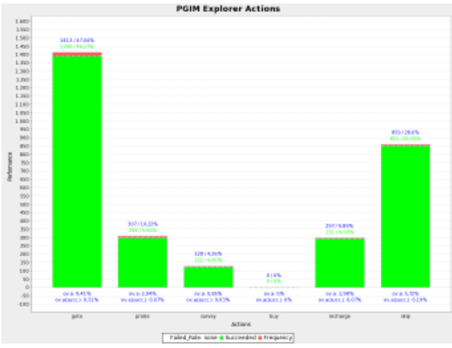


Figure 655: PGIM vs. TUB – Simulation 3 - PGIM Explorer Actions.

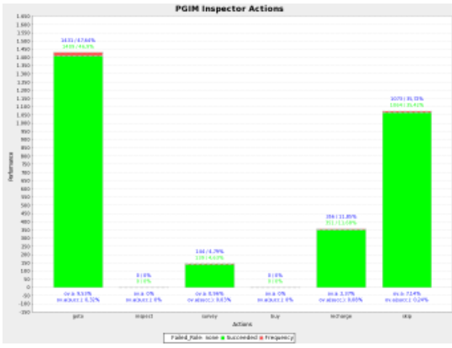


Figure 656: PGIM vs. TUB – Simulation 3 - PGIM Inspector Actions.

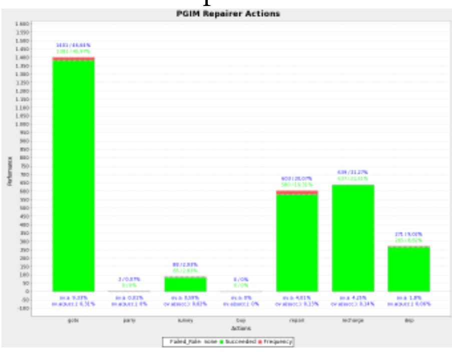


Figure 657: PGIM vs. TUB – Simulation 3 - PGIM Repairer Actions.

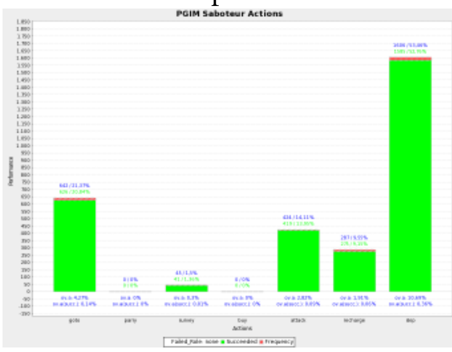


Figure 658: PGIM vs. TUB – Simulation 3 - PGIM Saboteur Actions.

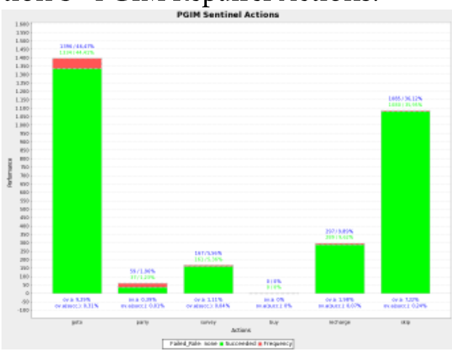


Figure 659: PGIM vs. TUB – Simulation 3 - PGIM Sentinel Actions.

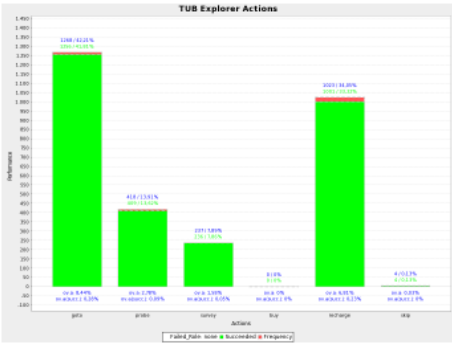


Figure 660: PGIM vs. TUB – Simulation 3 - TUB Explorer Actions.

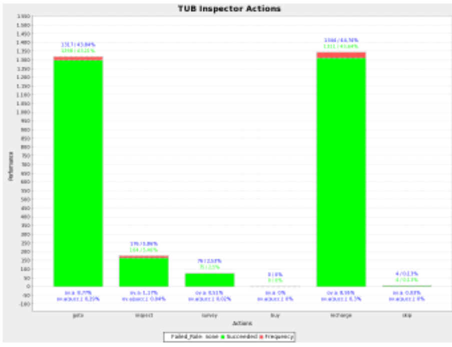


Figure 661: PGIM vs. TUB – Simulation 3 - TUB Inspector Actions.

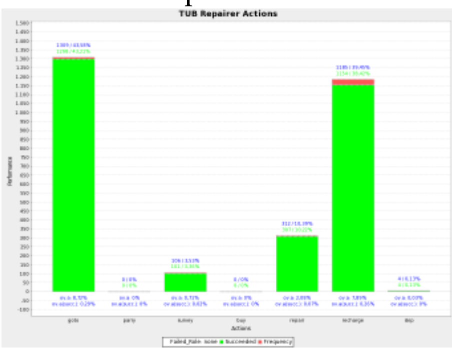


Figure 662: PGIM vs. TUB – Simulation 3 - TUB Repairer Actions.

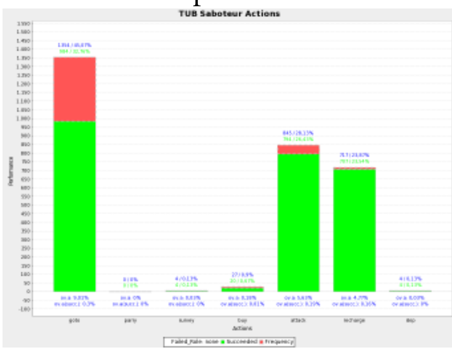


Figure 663: PGIM vs. TUB – Simulation 3 - TUB Saboteur Actions.

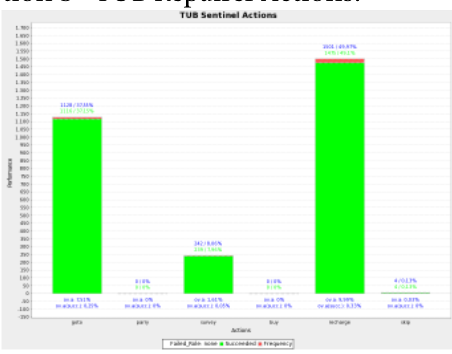


Figure 664: PGIM vs. TUB – Simulation 3 - TUB Sentinel Actions.

40 PGIM vs. UFSC – Simulation 1

40.1 Scores, Zone Stability and Achievements

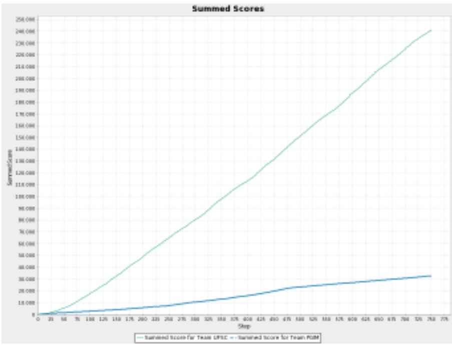


Figure 665: Summed scores.

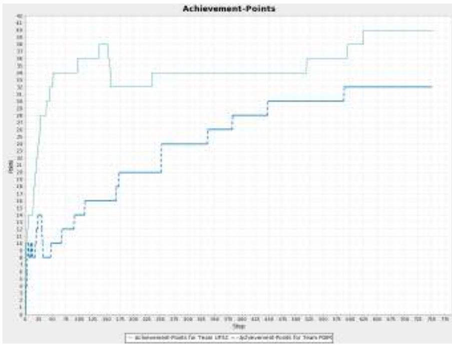


Figure 666: Achievement points.

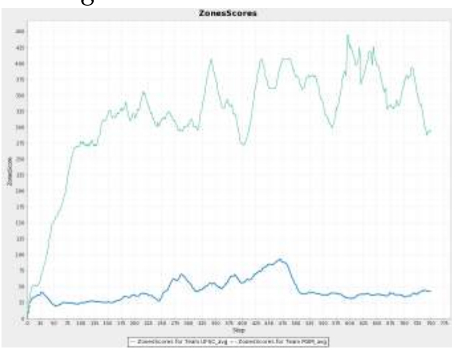


Figure 667: Zones scores.

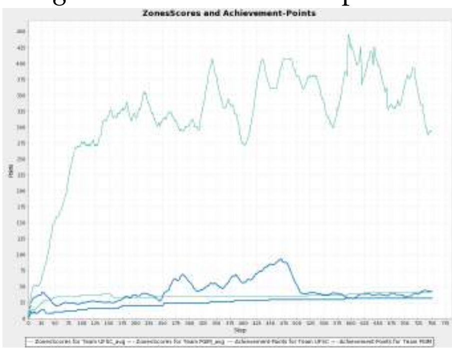


Figure 668: Zones scores and achievement points.

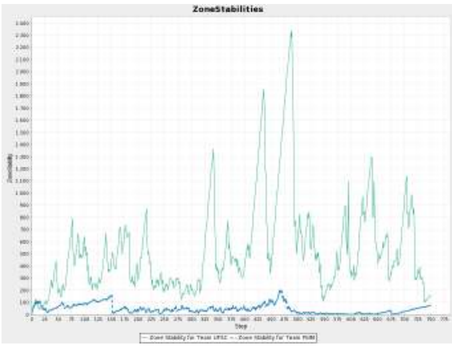


Figure 669: Zone Stabilities.

Step	UFSC	PGIM
1	surveyed10, area20, surveyed40, area10, surveyed20	surveyed10, area10
3	surveyed80, proved5	surveyed20
4		area20, proved5
5	proved10, surveyed160	
7	area40	
9	proved20	area40
12		surveyed40
14	attacked5	
15	surveyed320	
18	attacked10	proved10
20		inspected5
21	proved40	
23	area80	surveyed80
26	attacked20	
28	inspected5	
39	area160	
45	proved80	
47		proved20
51	attacked40	
67		surveyed160
90		inspected10
96	proved160	
110		proved40
136	area320	
168		surveyed320
173		attacked5
233	attacked80	
251		attacked10, proved80
336		attacked20
382		attacked40
447		area80
519	inspected10	
588		parried5
594	area640	
624	attacked160	

Figure 670: Achievements.

40.2 Stability

Reason	UFSC	%	PGIM	%
failed away			1625	10,83
failed parried	5	0,03		
failed random	151	1,01	163	1,09
failed wrong param			10	0,07
failed	3	0,02		
failed resources			24	0,16
failed attacked	9	0,06	57	0,38
noAction	3	0,02		
failed status			535	3,57

Figure 671: Failed actions.

40.3 Achievements

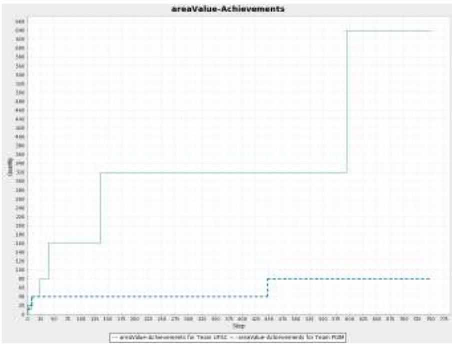


Figure 672: areaValueAchievements.

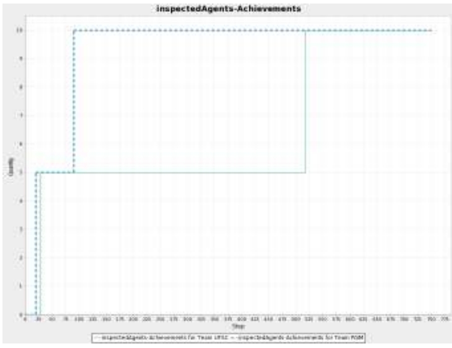


Figure 673: inspectedAgentsAchievements.

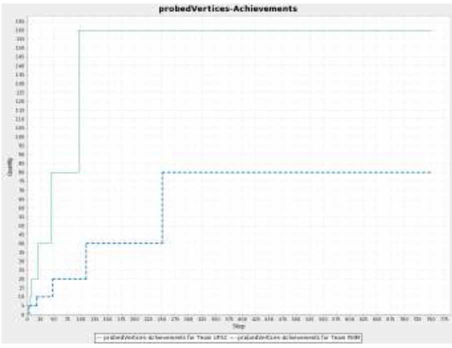


Figure 674: probedVerticesAchievements.

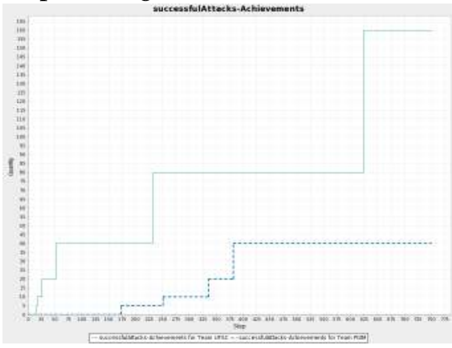


Figure 675: successfulAttacksAchievements.

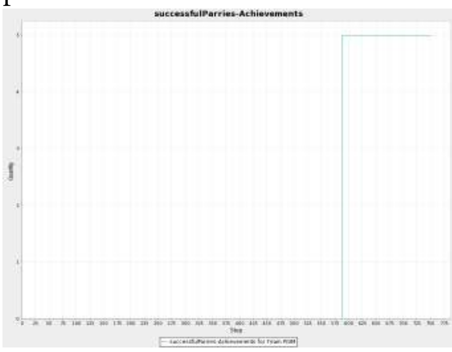


Figure 676: successfulParriesAchievements.

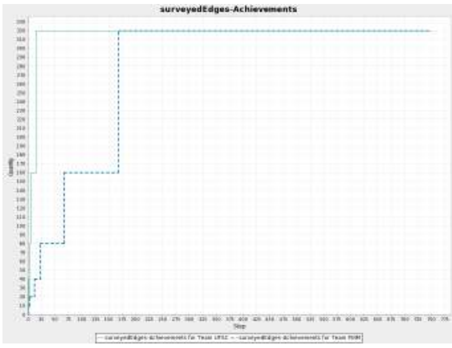


Figure 677: surveyedEdgesAchievements.

40.4 Actions per Role

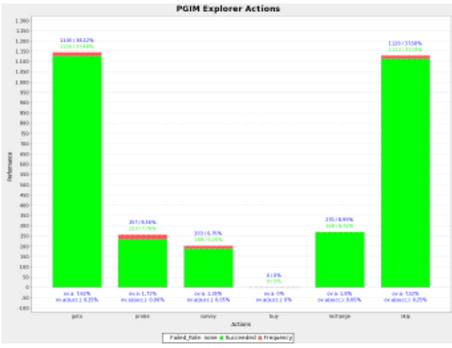


Figure 678: PGIM vs. UFSC – Simulation 1 - PGIM Explorer Actions.

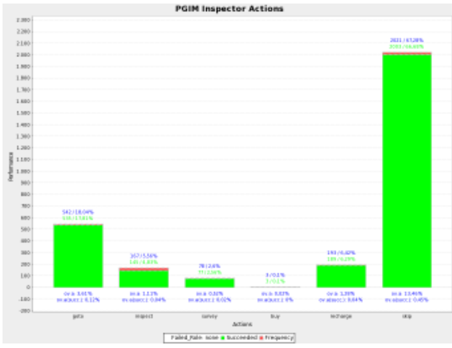


Figure 679: PGIM vs. UFSC – Simulation 1 - PGIM Inspector Actions.

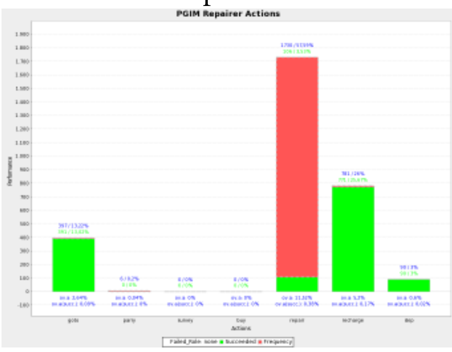


Figure 680: PGIM vs. UFSC – Simulation 1 - PGIM Repairer Actions.

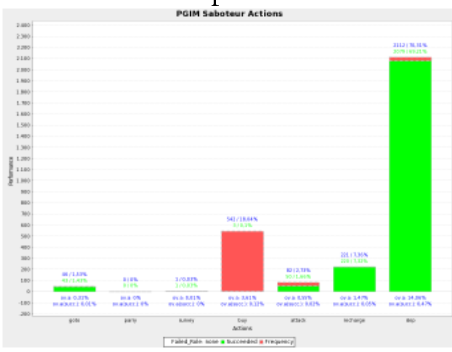


Figure 681: PGIM vs. UFSC – Simulation 1 - PGIM Saboteur Actions.

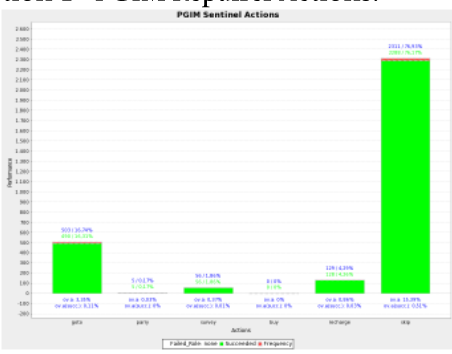


Figure 682: PGIM vs. UFSC – Simulation 1 - PGIM Sentinel Actions.

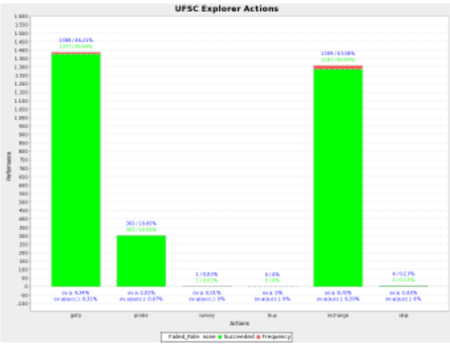


Figure 683: PGIM vs. UFSC – Simulation 1 - UFSC Explorer Actions.

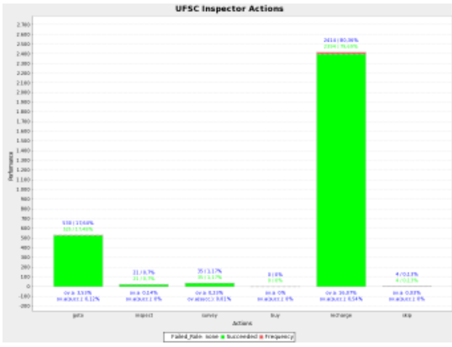


Figure 684: PGIM vs. UFSC – Simulation 1 - UFSC Inspector Actions.

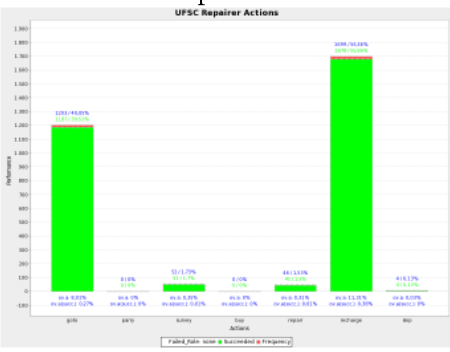


Figure 685: PGIM vs. UFSC – Simulation 1 - UFSC Repairer Actions.

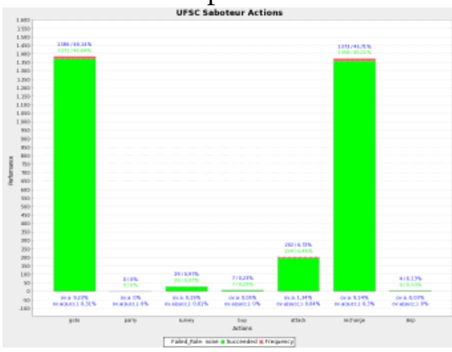


Figure 686: PGIM vs. UFSC – Simulation 1 - UFSC Saboteur Actions.

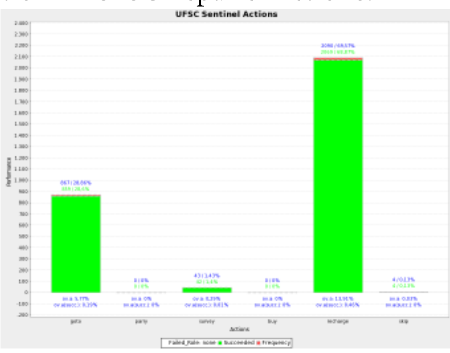


Figure 687: PGIM vs. UFSC – Simulation 1 - UFSC Sentinel Actions.

41 PGIM vs. UFSC – Simulation 2

41.1 Scores, Zone Stability and Achievements

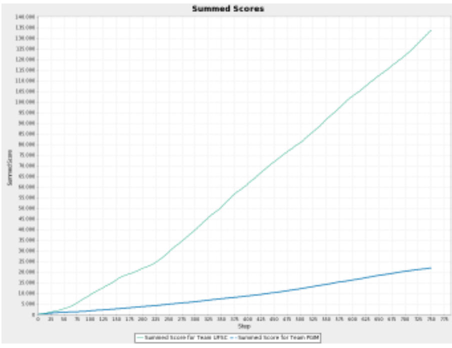


Figure 688: Summed scores.

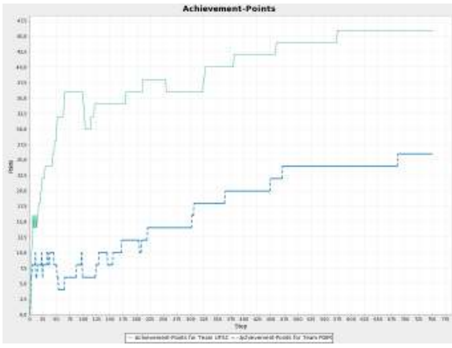


Figure 689: Achievement points.

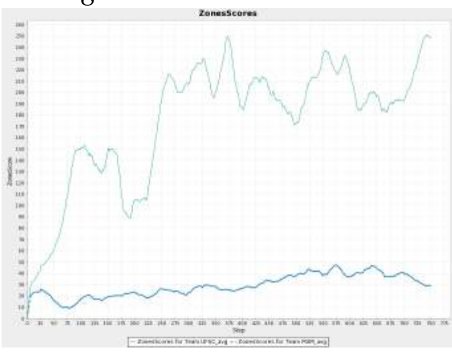


Figure 690: Zones scores.

Figure 691: Zones scores and achievement points.

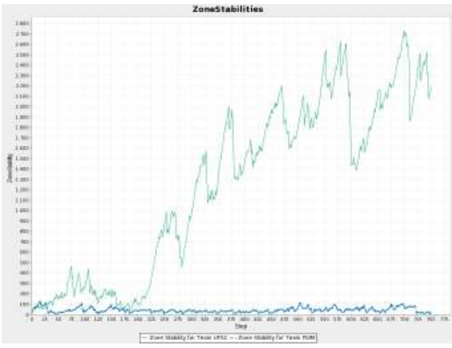


Figure 692: Zone Stabilities.

Step	UFSC	PGIM
1	area10	surveyed10
2	surveyed10, surveyed40, surveyed20	area10, surveyed20
3	proved5	
4	surveyed80	area20
5	proved10, attacked5	
7	surveyed160	
10	proved20	surveyed40
12	inspected5	
14	attacked10	proved5
16	surveyed320	
20	area20	
22	proved40	proved10
24		surveyed80
28	attacked20	
32		attacked5
37		inspected5
42	area40	
46	proved80	
49	inspected10	
50		surveyed160
51	attacked40	
63	parried5	
65	area80	attacked10
87		proved20
96		attacked20
114	proved160	
120	attacked80	
123		attacked40
128		proved40
155		attacked80
171		inspected10
179	attacked160	
208		attacked160
211	parried10	
219		parried5
302		parried10
306		surveyed320
323	attacked320	
326	area160	
364		proved80
380	parried20	
449		attacked320
459	inspected20	
471		parried20
473	attacked640	
686		parried40

Figure 693: Achievements.

41.2 Stability

Reason	UFSC	%	PGIM	%
failed away			164	1,09
failed parried	49	0,33	27	0,18
failed random	151	1,01	151	1,01
failed wrong param			78	0,52
failed resources			64	0,43
failed	25	0,17		
failed attacked	160	1,07	49	0,33
noAction	25	0,17		

Figure 694: Failed actions.

41.3 Achievements

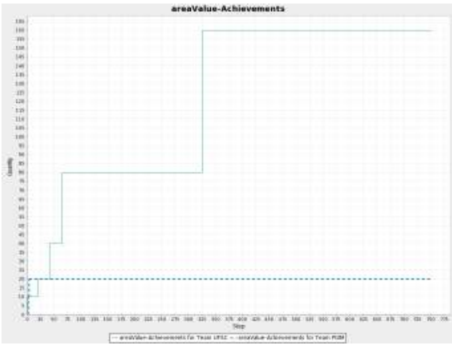


Figure 695: areaValueAchievements.

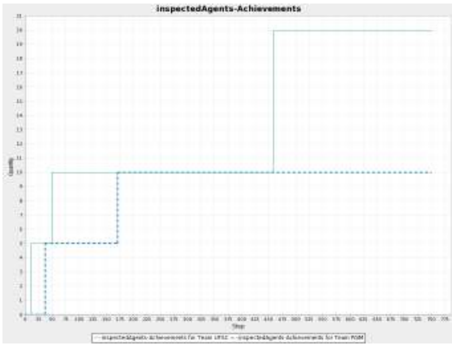


Figure 696: inspectedAgentsAchievements.

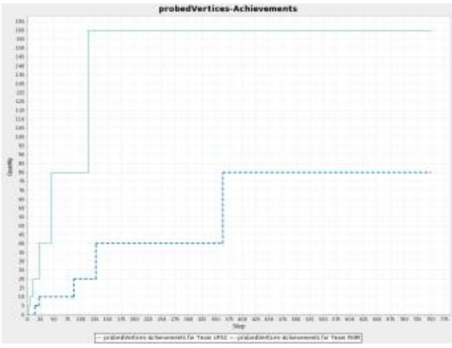


Figure 697: probedVerticesAchievements.

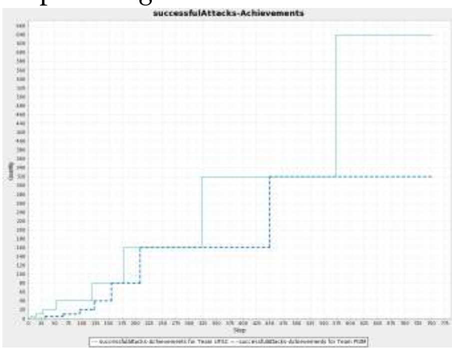


Figure 698: successfulAttacksAchievements.

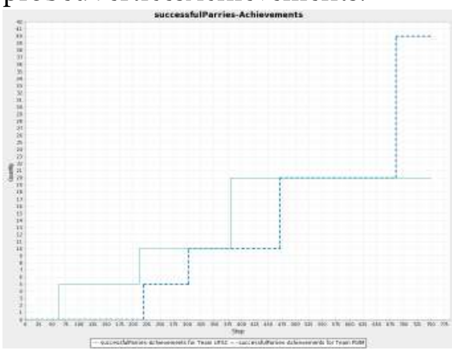


Figure 699: successfulParriesAchievements.

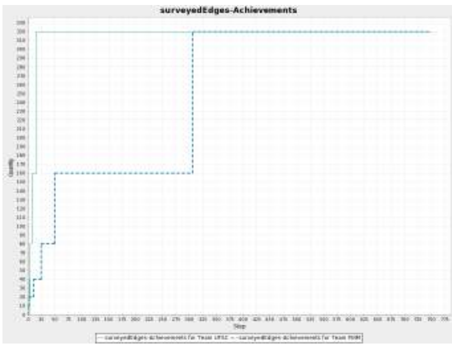


Figure 700: surveyedEdgesAchievements.

41.4 Actions per Role

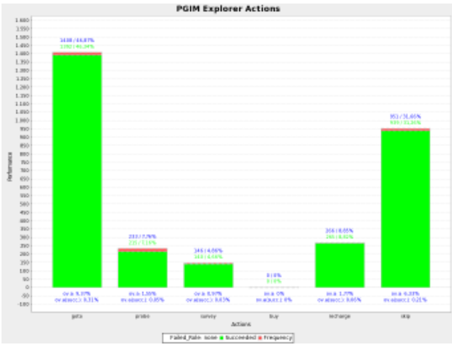


Figure 701: PGIM vs. UFSC – Simulation 2 - PGIM Explorer Actions.

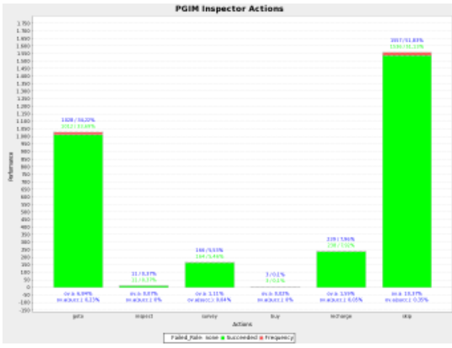


Figure 702: PGIM vs. UFSC – Simulation 2 - PGIM Inspector Actions.

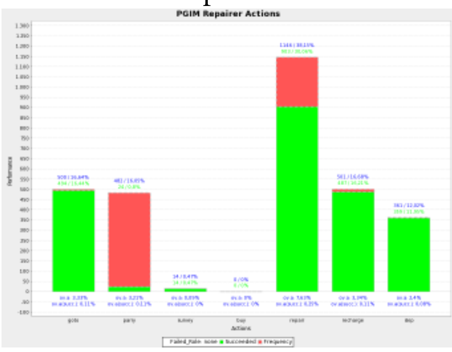


Figure 703: PGIM vs. UFSC – Simulation 2 - PGIM Repairer Actions.

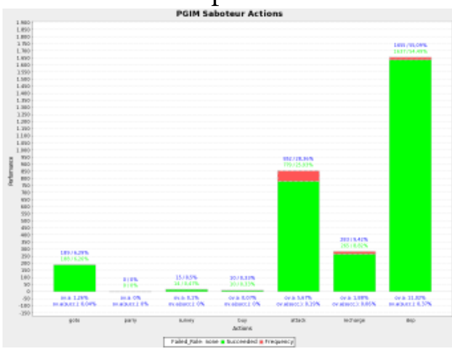


Figure 704: PGIM vs. UFSC – Simulation 2 - PGIM Saboteur Actions.

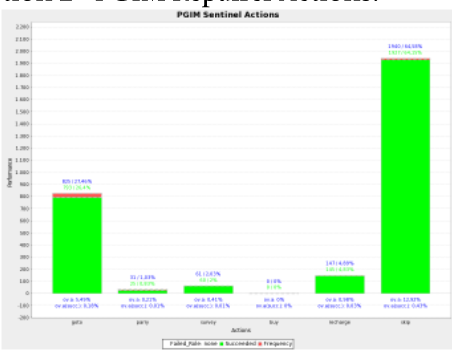


Figure 705: PGIM vs. UFSC – Simulation 2 - PGIM Sentinel Actions.

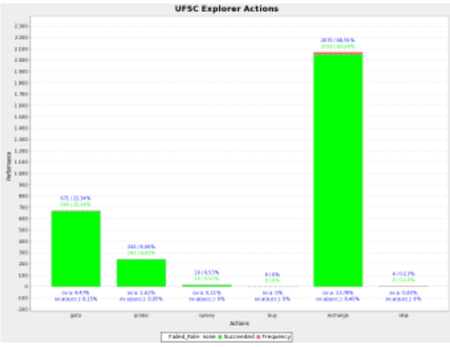


Figure 706: PGIM vs. UFSC – Simulation 2 - UFSC Explorer Actions.

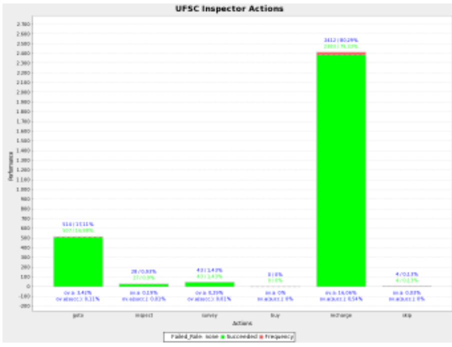


Figure 707: PGIM vs. UFSC – Simulation 2 - UFSC Inspector Actions.

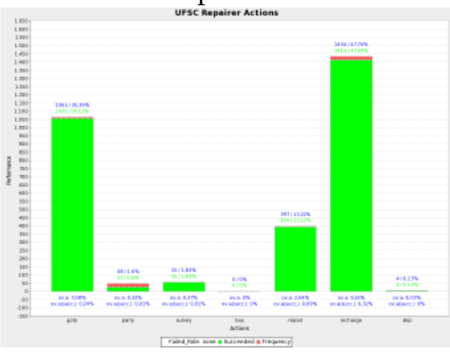


Figure 708: PGIM vs. UFSC – Simulation 2 - UFSC Repairer Actions.

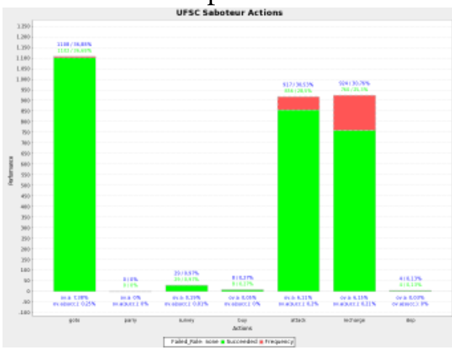


Figure 709: PGIM vs. UFSC – Simulation 2 - UFSC Saboteur Actions.

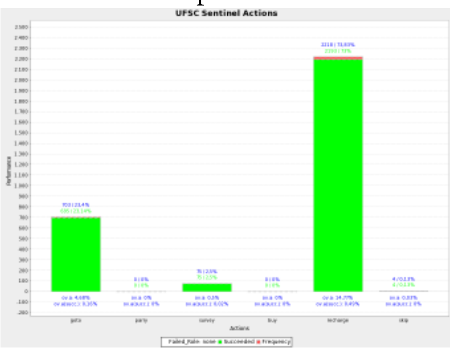


Figure 710: PGIM vs. UFSC – Simulation 2 - UFSC Sentinel Actions.

42 PGIM vs. UFSC – Simulation 3

42.1 Scores, Zone Stability and Achievements

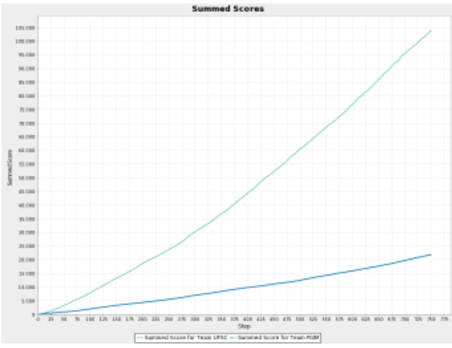


Figure 711: Summed scores.

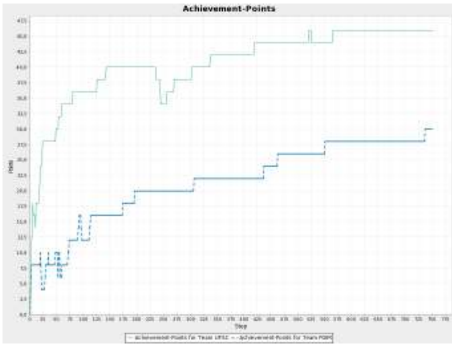


Figure 712: Achievement points.

Figure 713: Zones scores.

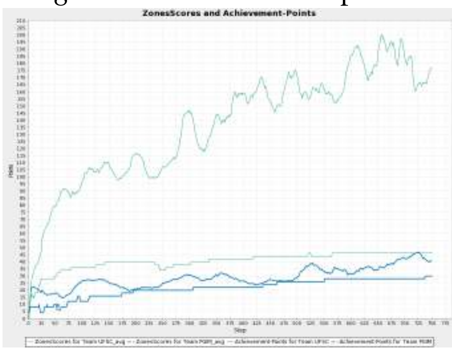


Figure 714: Zones scores and achievement points.

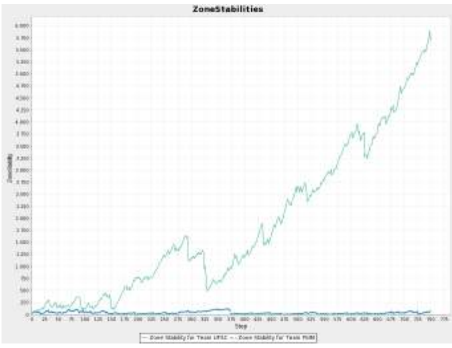


Figure 715: Zone Stabilities.

Step	UFSC	PGIM
1	surveyed10, surveyed20	surveyed10, surveyed20
2	surveyed40, inspected5	area10
3	inspected10, proved5	area20
4	surveyed80, area10	
5	proved10, attacked5	
8	surveyed160	
11	proved20, attacked10	
18	area20, attacked20	
19	area40	proved5
23	proved40, surveyed320	
28		surveyed40
29		proved10
34		surveyed80
47		attacked5
48	proved80	
53	attacked40	attacked10
54		proved20
59	area80	
60		attacked20
71		attacked40
73		surveyed160
79	attacked80	
90		parried5
92		attacked80
111		parried10
113		proved40
125	proved160	
142	attacked160	
173		attacked160
195		parried20
255	parried5	
269	parried10	
302	attacked320	
306		attacked320
336	parried20	
418	area160	
436		proved80
462		attacked640
520	inspected20	
550		parried40
564	attacked640	
737		surveyed320

Figure 716: Achievements.

42.2 Stability

Reason	UFSC	%	PGIM	%
failed away			116	0,77
failed parried	48	0,32	39	0,26
failed random	153	1,02	137	0,91
failed wrong param			68	0,45
failed resources	6	0,04	73	0,49
failed	9	0,06	77	0,51
failed attacked	218	1,45	76	0,51
noAction	9	0,06	78	0,52
failed status			1	0,01

Figure 717: Failed actions.

42.3 Achievements

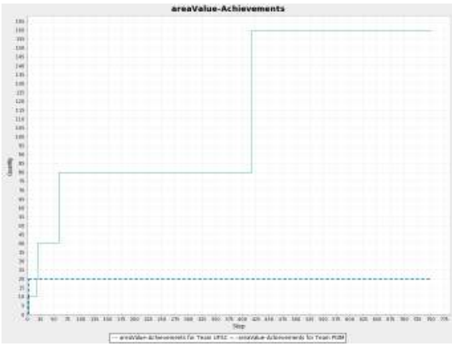


Figure 718: areaValueAchievements.

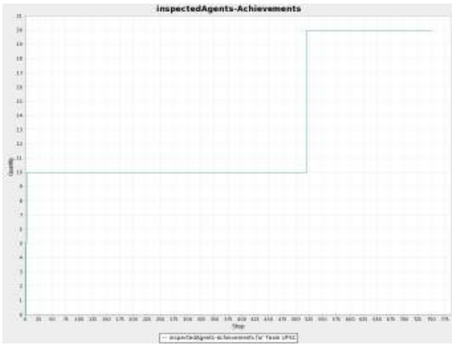


Figure 719: inspectedAgentsAchievements.

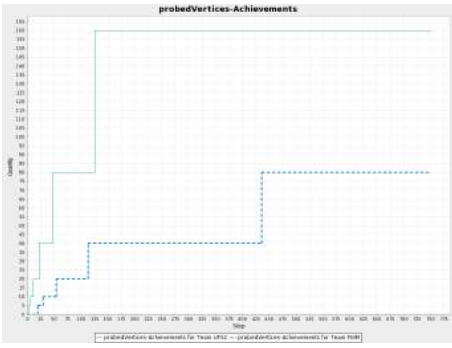


Figure 720: probedVerticesAchievements.

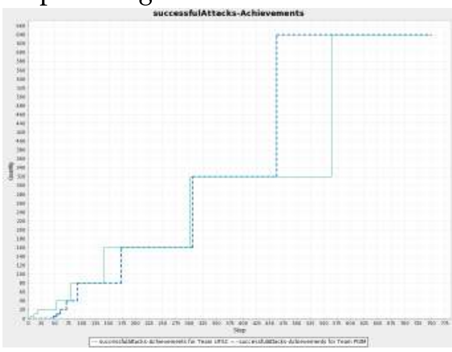


Figure 721: successfulAttacksAchievements.

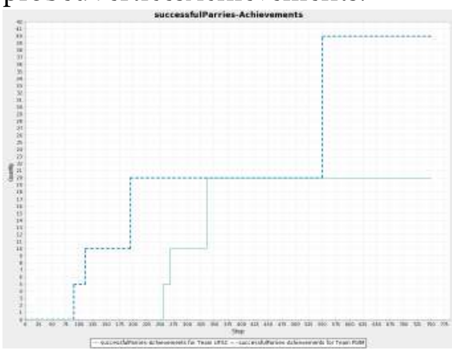


Figure 722: successfulParriesAchievements.

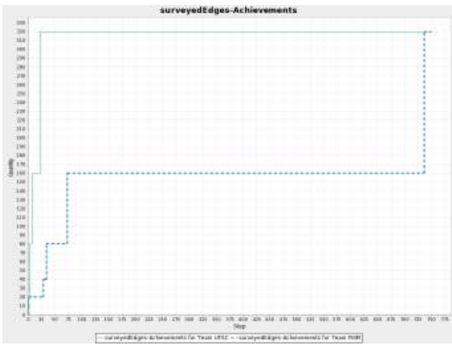


Figure 723: surveyedEdgesAchievements.

42.4 Actions per Role

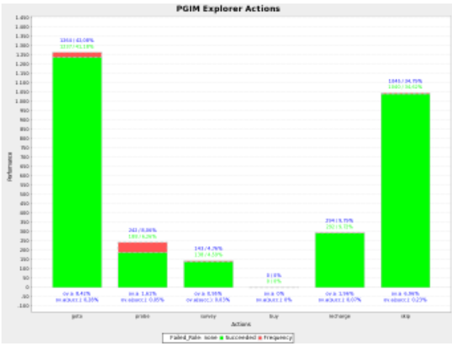


Figure 724: PGIM vs. UFSC – Simulation 3 - PGIM Explorer Actions.

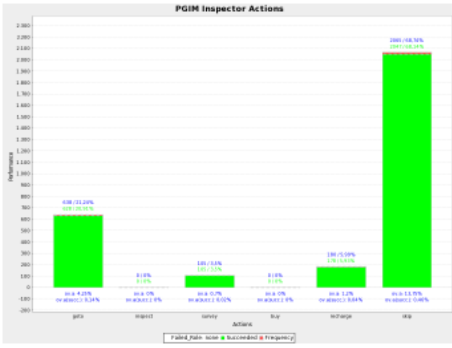


Figure 725: PGIM vs. UFSC – Simulation 3 - PGIM Inspector Actions.

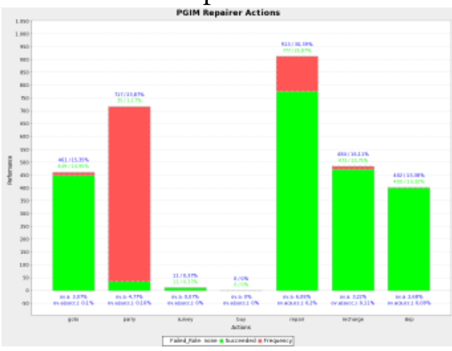


Figure 726: PGIM vs. UFSC – Simulation 3 - PGIM Repairer Actions.

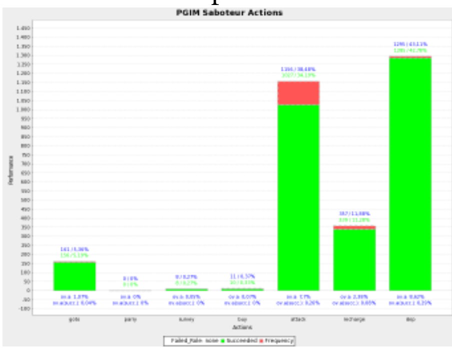


Figure 727: PGIM vs. UFSC – Simulation 3 - PGIM Saboteur Actions.

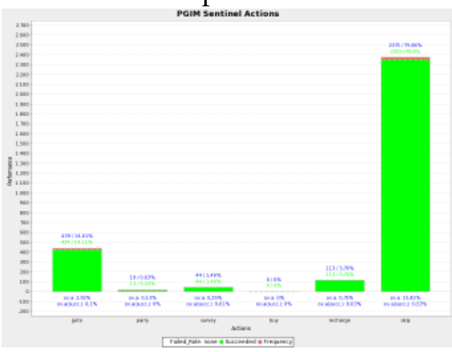


Figure 728: PGIM vs. UFSC – Simulation 3 - PGIM Sentinel Actions.

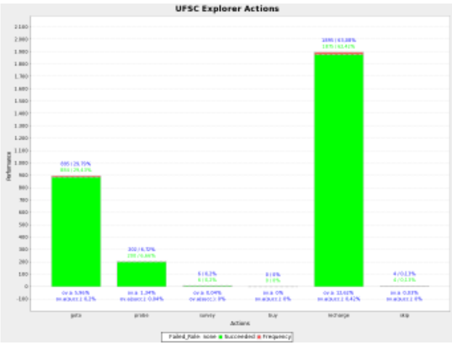


Figure 729: PGIM vs. UFSC – Simulation 3 - UFSC Explorer Actions.

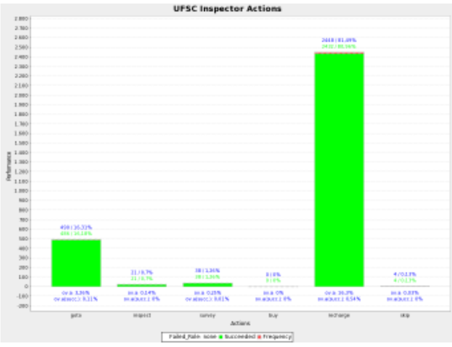


Figure 730: PGIM vs. UFSC – Simulation 3 - UFSC Inspector Actions.

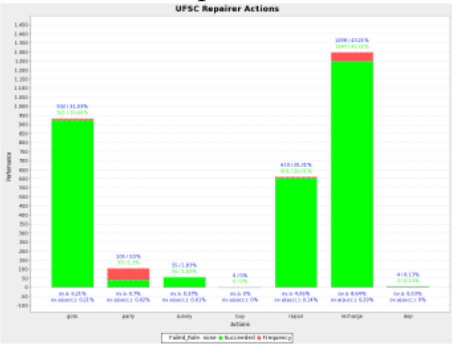


Figure 731: PGIM vs. UFSC – Simulation 3 - UFSC Repairer Actions.

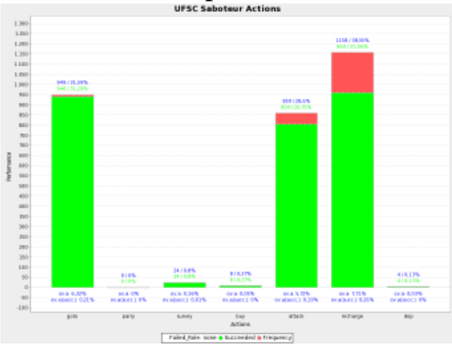


Figure 732: PGIM vs. UFSC – Simulation 3 - UFSC Saboteur Actions.

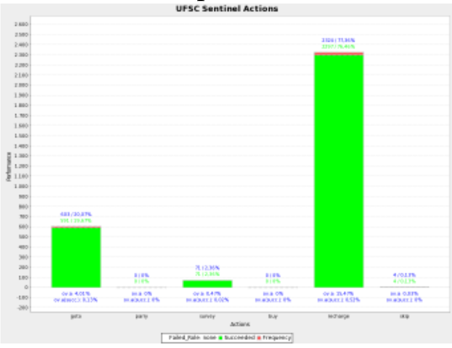


Figure 733: PGIM vs. UFSC – Simulation 3 - UFSC Sentinel Actions.

43 PGIM vs. USP – Simulation 1

43.1 Scores, Zone Stability and Achievements

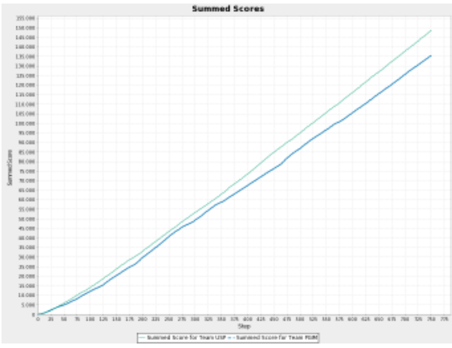


Figure 734: Summed scores.

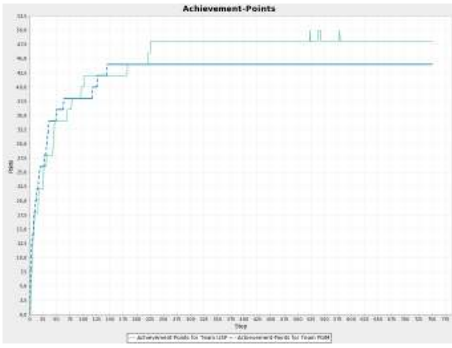


Figure 735: Achievement points.

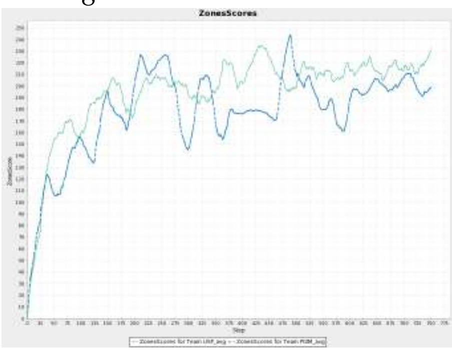


Figure 736: Zones scores.

Figure 737: Zones scores and achievement points.

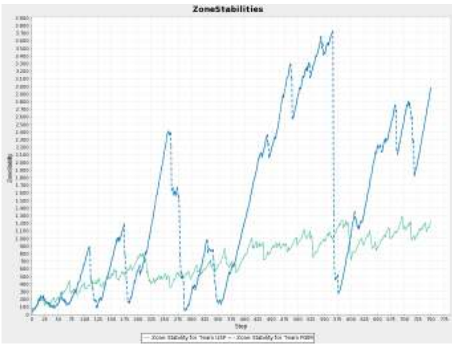


Figure 738: Zone Stabilities.

Step	USP	PGIM
1	area10	surveyed40, surveyed10, surveyed20, area10
2	surveyed10, surveyed20	surveyed80
3		area20
4	area20, surveyed40, proved5	proved5
6	proved10	
7		proved10
8	area40	surveyed160
9	surveyed80	
10		area40
13		inspected5
16	proved20, area80	proved20
18		area80
25	inspected5	
26	surveyed160	
27		surveyed320
31	attacked5	attacked5
33		inspected10
35		proved40
43	attacked10	
44	proved40	
46	inspected10	
49		attacked10
63		attacked20
70	area160	
78	attacked20	
95	surveyed320	
102	parried5	
117		proved80
126		area160
144		attacked40
182	attacked40	
221	parried10	
225	proved80	
522	parried20	
537	attacked80	
577	proved160	

Figure 739: Achievements.

43.2 Stability

Reason	USP	%	PGIM	%
failed away	15	0,1	2	0,01
failed parried	2	0,01	23	0,15
failed random	141	0,94	151	1,01
failed wrong param			10	0,07
failed resources	71	0,47		
failed attacked	22	0,15	7	0,05

Figure 740: Failed actions.

43.3 Achievements

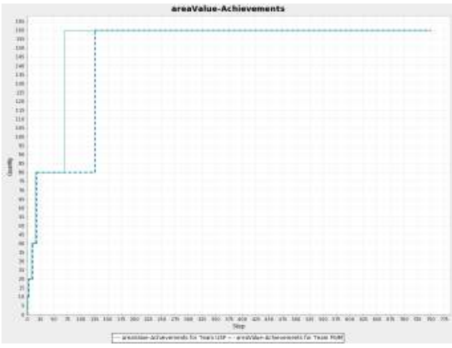


Figure 741: areaValueAchievements.

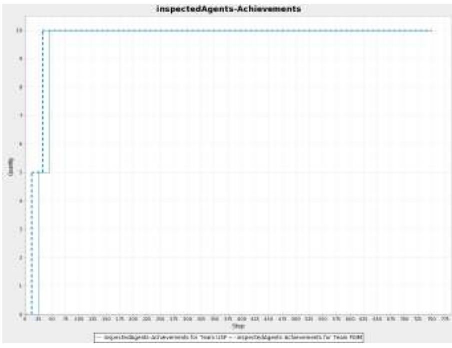


Figure 742: inspectedAgentsAchievements.

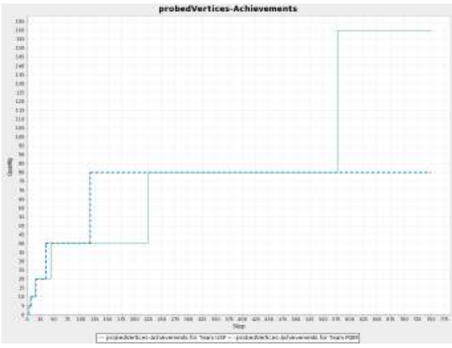


Figure 743: probedVerticesAchievements.

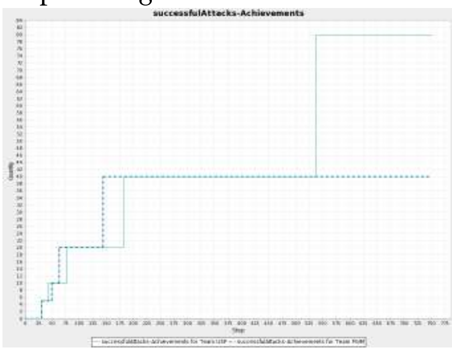


Figure 744: successfulAttacksAchievements.

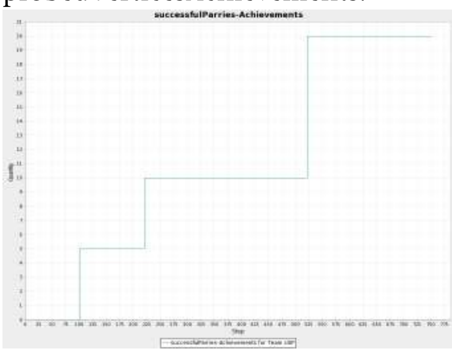


Figure 745: successfulParriesAchievements.

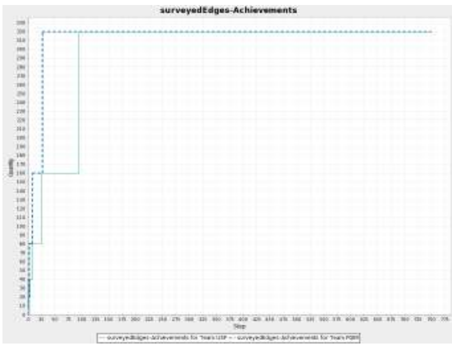


Figure 746: surveyedEdgesAchievements.

43.4 Actions per Role

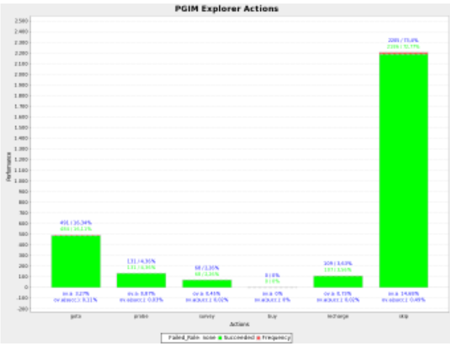


Figure 747: PGIM vs. USP – Simulation 1 - PGIM Explorer Actions.

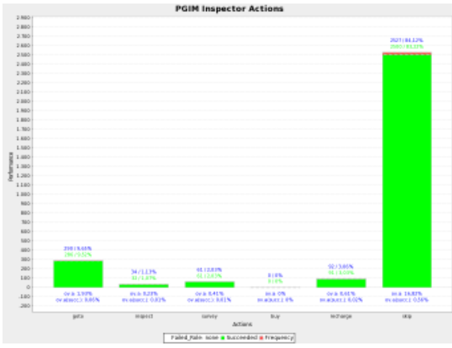


Figure 748: PGIM vs. USP – Simulation 1 - PGIM Inspector Actions.

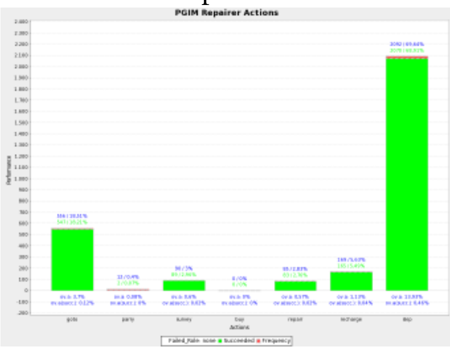


Figure 749: PGIM vs. USP – Simulation 1 - PGIM Repairer Actions.

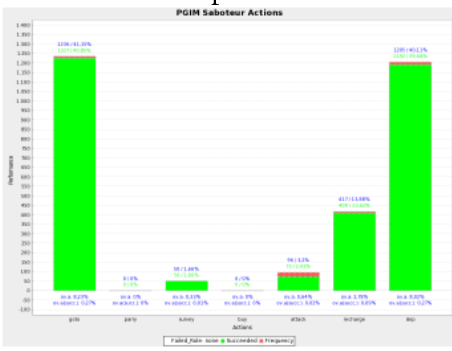


Figure 750: PGIM vs. USP – Simulation 1 - PGIM Saboteur Actions.

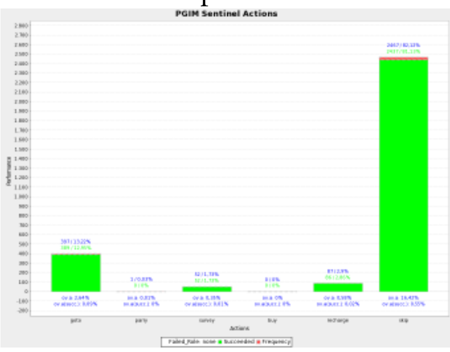


Figure 751: PGIM vs. USP – Simulation 1 - PGIM Sentinel Actions.

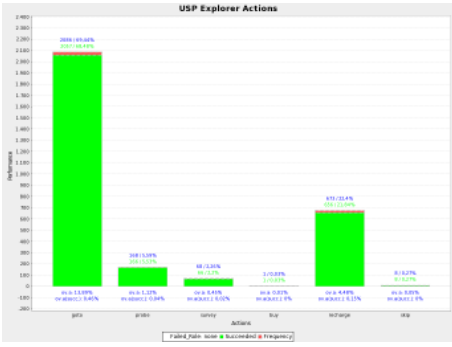


Figure 752: PGIM vs. USP – Simulation 1 - USP Explorer Actions.

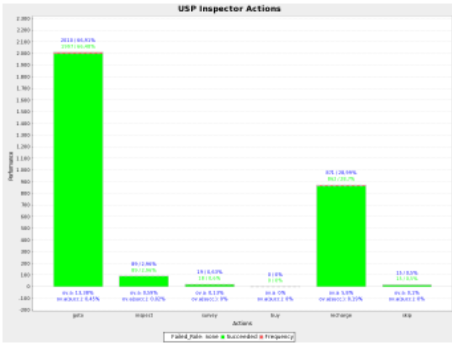


Figure 753: PGIM vs. USP – Simulation 1 - USP Inspector Actions.

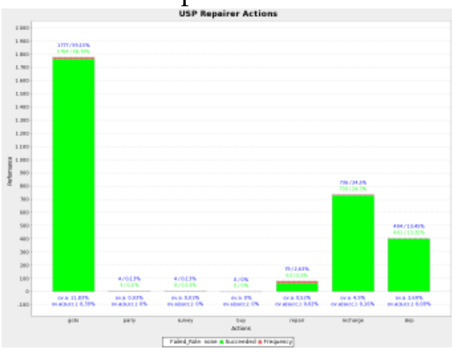


Figure 754: PGIM vs. USP – Simulation 1 - USP Repairer Actions.

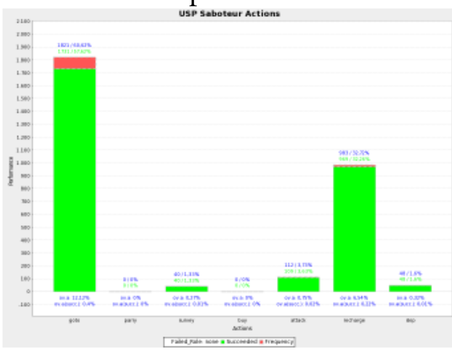


Figure 755: PGIM vs. USP – Simulation 1 - USP Saboteur Actions.

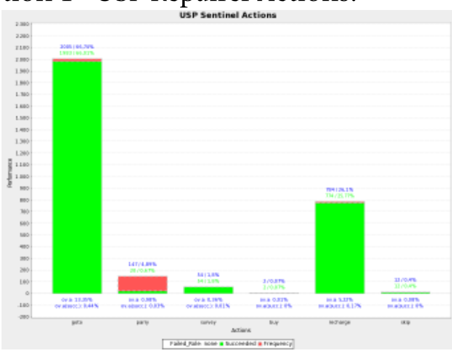


Figure 756: PGIM vs. USP – Simulation 1 - USP Sentinel Actions.

44 PGIM vs. USP – Simulation 2

44.1 Scores, Zone Stability and Achievements

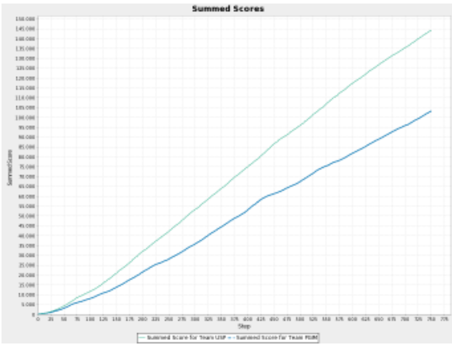


Figure 757: Summed scores.

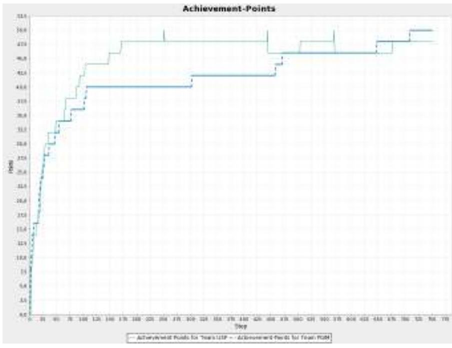


Figure 758: Achievement points.

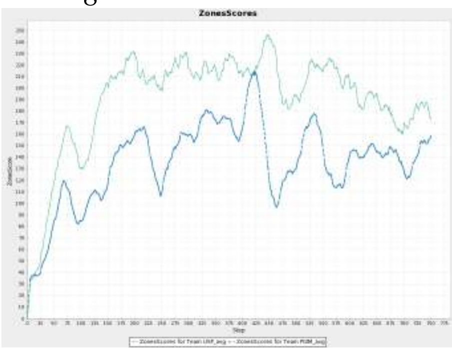


Figure 759: Zones scores.

Figure 760: Zones scores and achievement points.

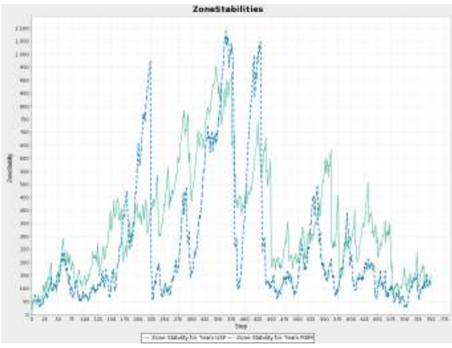


Figure 761: Zone Stabilities.

Step	USP	PGIM
1		surveyed10, surveyed40, area10, surveyed20
2	surveyed10, area10, surveyed20	surveyed80
3	surveyed40	
4	proved5	surveyed160, proved5
6	proved10	
7	surveyed80	proved10
13	proved20	
16	area20	surveyed320
17		proved20
18		inspected5
19	surveyed160	
20	attacked5	attacked5
21	area40	
23	inspected5	
24		area20
26	area80	
27		area40
28	proved40	
34	inspected10	
36		proved40
47		attacked10
48	attacked10	
55		area80
65	area160	
67	parried5	
77		attacked20
86	surveyed320	
93	attacked20	
102		proved80
103	parried10	
105		attacked40
147	parried20	
170	proved80	
250	attacked40	
302		area160
443	parried40	
458		attacked80
471		proved160
504	proved160	
567	attacked80	
647		inspected10
676	parried80	
709		attacked160

Figure 762: Achievements.

44.2 Stability

Reason	USP	%	PGIM	%
failed away	54	0,36	5	0,03
failed parried	2	0,01	102	0,68
failed random	145	0,97	162	1,08
failed wrong param			8	0,05
failed resources	64	0,43		
failed attacked	67	0,45	22	0,15

Figure 763: Failed actions.

44.3 Achievements

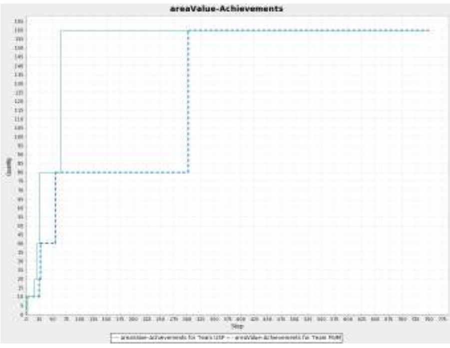


Figure 764: areaValueAchievements.

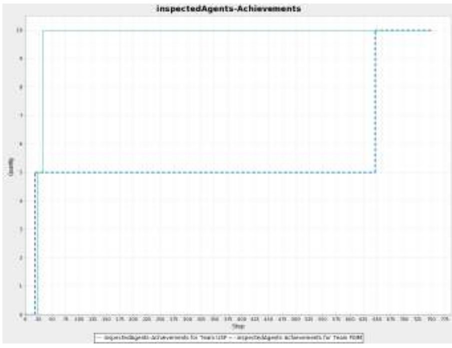


Figure 765: inspectedAgentsAchievements.

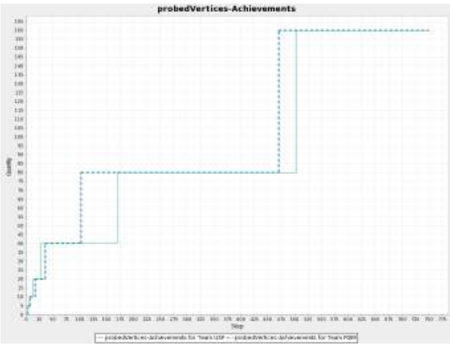


Figure 766: probedVerticesAchievements.

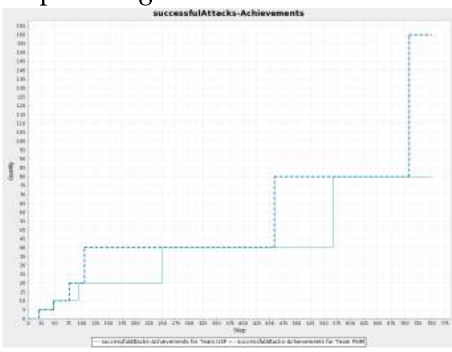


Figure 767: successfulAttacksAchievements.

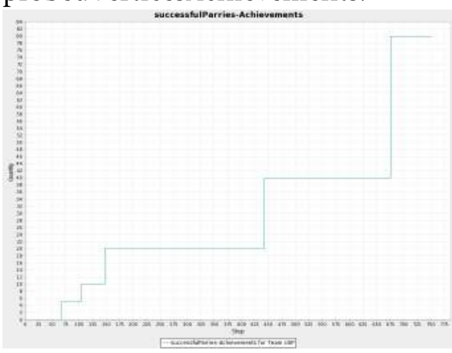


Figure 768: successfulParriesAchievements.

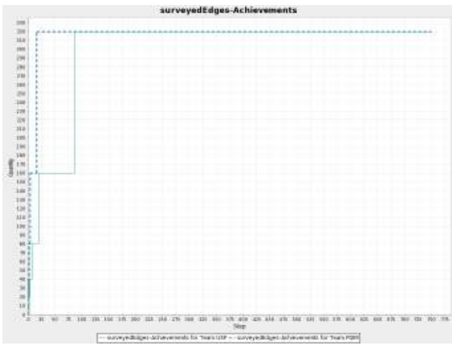


Figure 769: surveyedEdgesAchievements.

44.4 Actions per Role

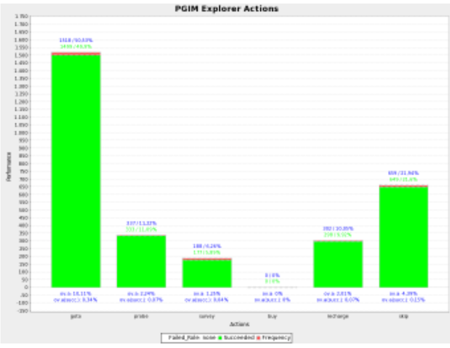


Figure 770: PGIM vs. USP – Simulation 2 - PGIM Explorer Actions.

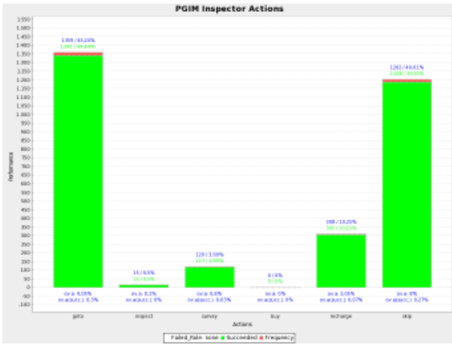


Figure 771: PGIM vs. USP – Simulation 2 - PGIM Inspector Actions.

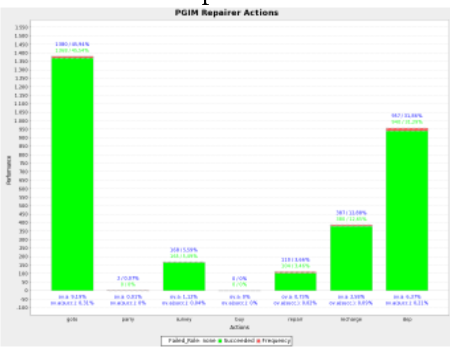


Figure 772: PGIM vs. USP – Simulation 2 - PGIM Repairer Actions.

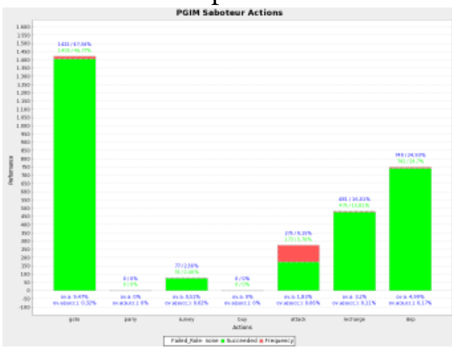


Figure 773: PGIM vs. USP – Simulation 2 - PGIM Saboteur Actions.

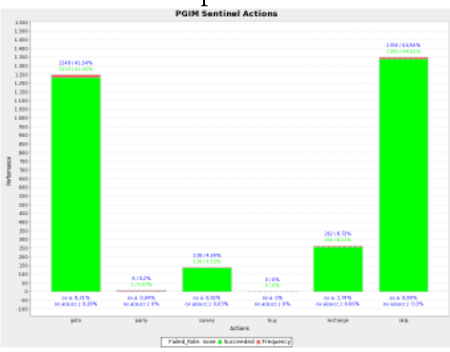


Figure 774: PGIM vs. USP – Simulation 2 - PGIM Sentinel Actions.

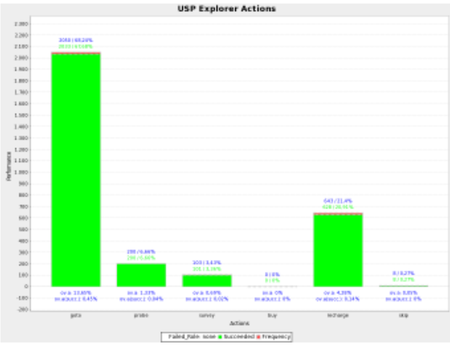


Figure 775: PGIM vs. USP – Simulation 2 - USP Explorer Actions.

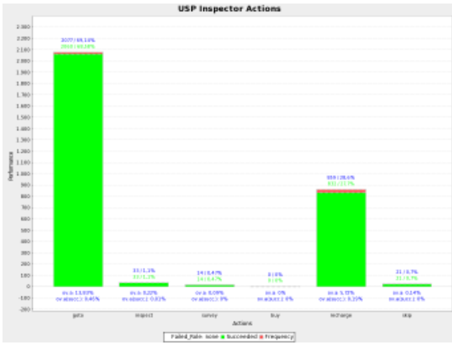


Figure 776: PGIM vs. USP – Simulation 2 - USP Inspector Actions.

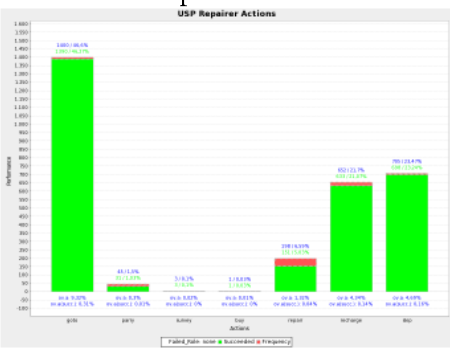


Figure 777: PGIM vs. USP – Simulation 2 - USP Repairer Actions.

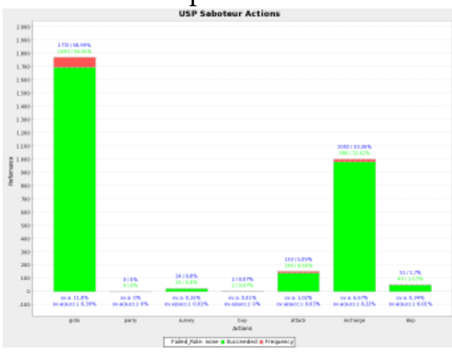


Figure 778: PGIM vs. USP – Simulation 2 - USP Saboteur Actions.

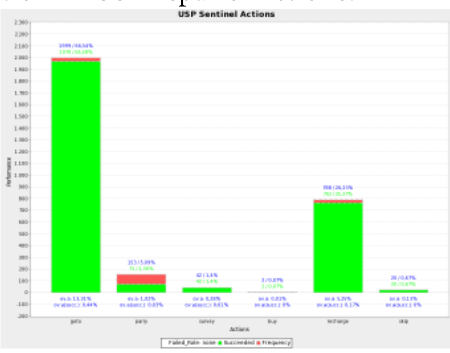


Figure 779: PGIM vs. USP – Simulation 2 - USP Sentinel Actions.

45 PGIM vs. USP – Simulation 3

45.1 Scores, Zone Stability and Achievements

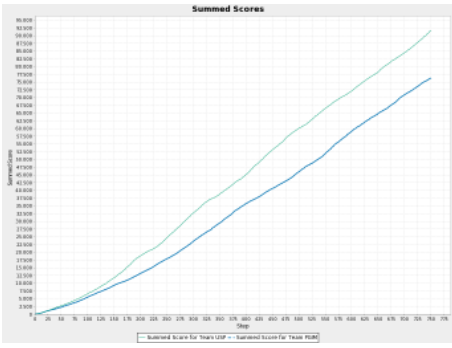


Figure 780: Summed scores.

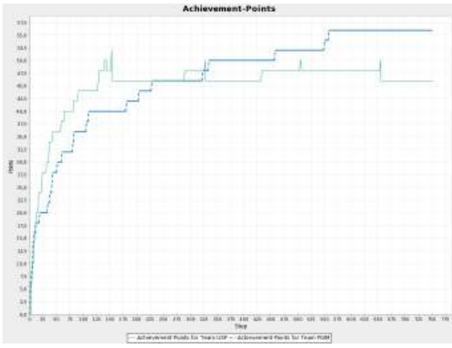


Figure 781: Achievement points.

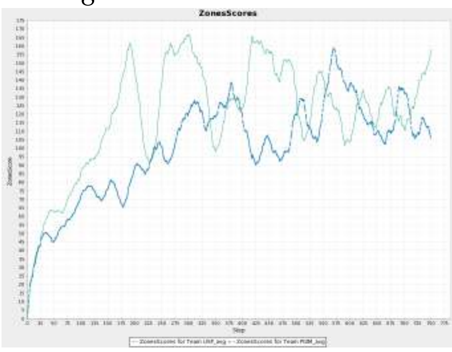


Figure 782: Zones scores.

Figure 783: Zones scores and achievement points.

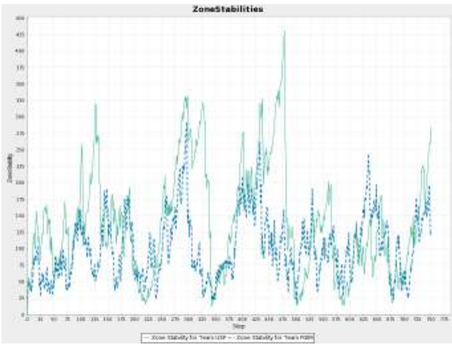


Figure 784: Zone Stabilities.

Step	USP	PGIM
1		surveyed10, surveyed40, surveyed20
2	surveyed10, surveyed40, surveyed20	surveyed80
4	proved5	proved5
5	surveyed80	area10
6		surveyed160
7	proved10	proved10
8	area10	
9	area20	
10	inspected5	
11		area20
13	proved20	
16	inspected10, attacked5	
18		proved20
22	surveyed160	
23	attacked10	
31	attacked20	
33		inspected5
36	area40, proved40	
37		attacked5
41		inspected10
42	inspected20	proved40
51		attacked10
58	attacked40	
60		inspected20
64	parried5	
80		area40
81	parried10	
82		attacked20
90	parried20	
105		surveyed320
109		attacked40
127	area80	
129	proved80	
138	parried40	
151	attacked80	
152	surveyed320	
180		proved80
203		attacked80
227		area80
288	attacked160	
322		parried5
326	parried80	
333		attacked160
432	area160	
457		parried10
504	proved160	
549		proved160
557		attacked320
653	parried160	
654	attacked320	

Figure 785: Achievements.

45.2 Stability

Reason	USP	%	PGIM	%
failed away	143	0,95	12	0,08
failed parried	17	0,11	173	1,15
failed random	146	0,97	171	1,14
failed wrong param			26	0,17
failed	140	0,93		
failed resources	40	0,27		
failed attacked	178	1,19	69	0,46
noAction	140	0,93		

Figure 786: Failed actions.

45.3 Achievements

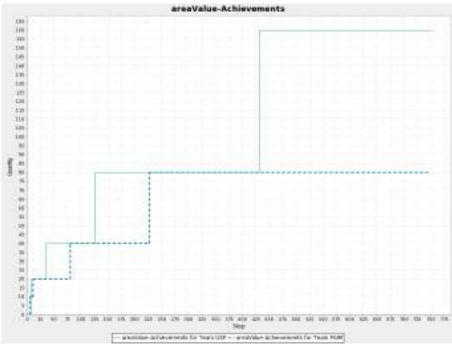


Figure 787: areaValueAchievements.

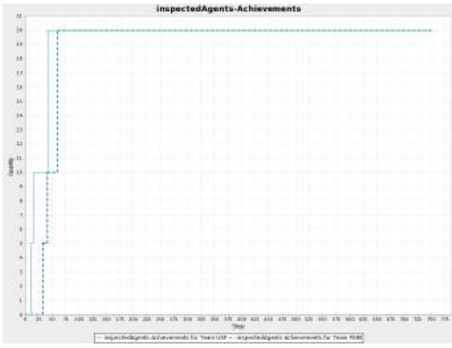


Figure 788: inspectedAgentsAchievements.

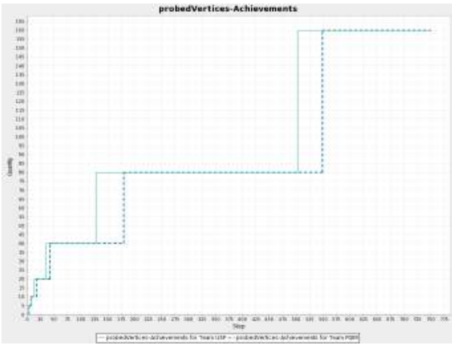


Figure 789: probedVerticesAchievements.

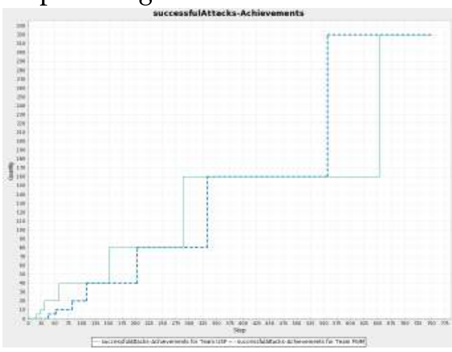


Figure 790: successfulAttacksAchievements.

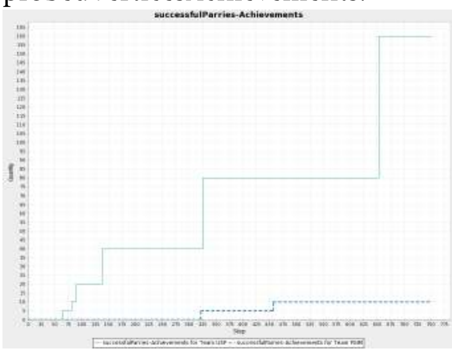


Figure 791: successfulParriesAchievements.

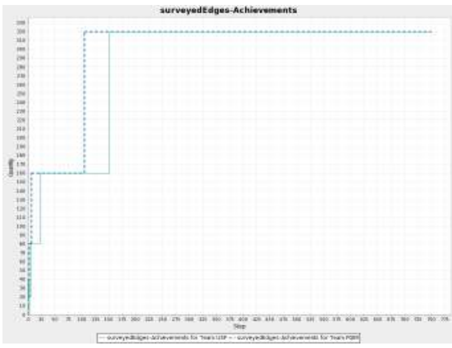


Figure 792: surveyedEdgesAchievements.

45.4 Actions per Role

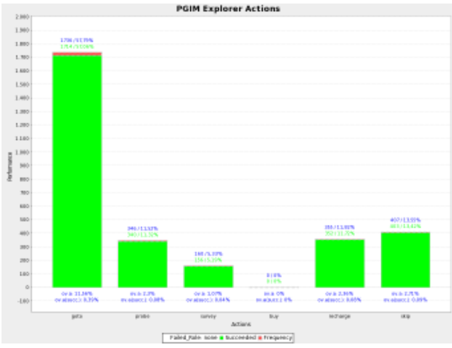


Figure 793: PGIM vs. USP – Simulation 3 - PGIM Explorer Actions.

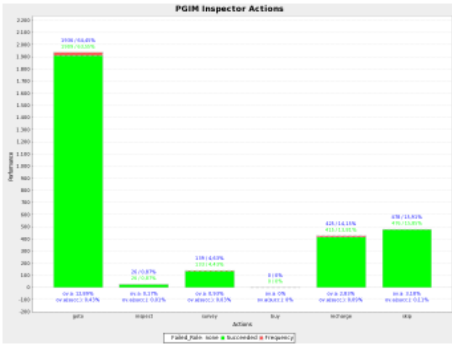


Figure 794: PGIM vs. USP – Simulation 3 - PGIM Inspector Actions.

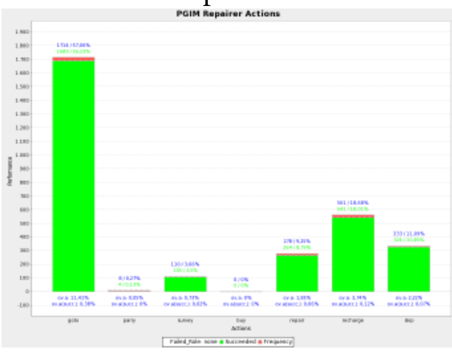


Figure 795: PGIM vs. USP – Simulation 3 - PGIM Repairer Actions.

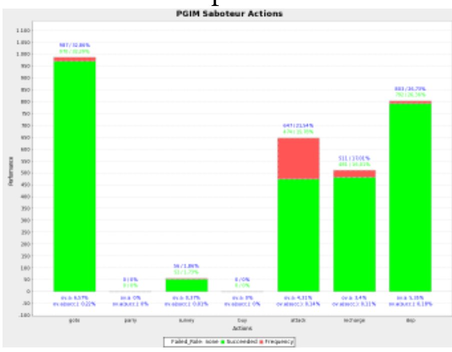


Figure 796: PGIM vs. USP – Simulation 3 - PGIM Saboteur Actions.

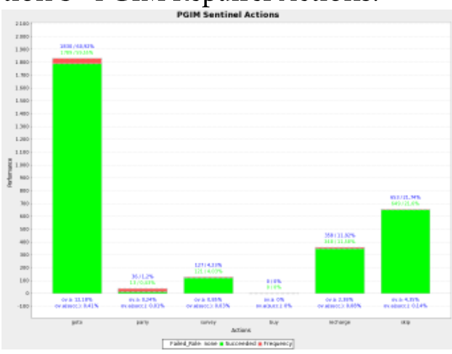


Figure 797: PGIM vs. USP – Simulation 3 - PGIM Sentinel Actions.

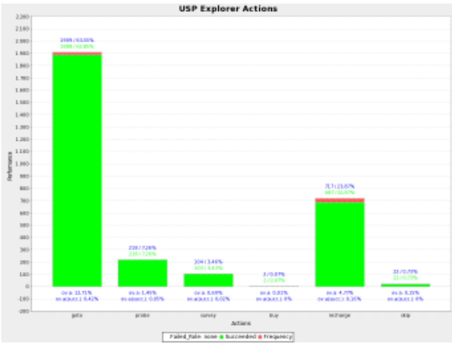


Figure 798: PGIM vs. USP – Simulation 3 - USP Explorer Actions.

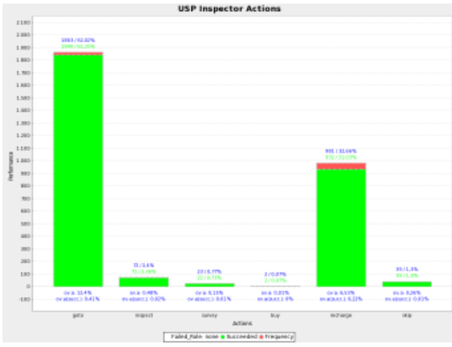


Figure 799: PGIM vs. USP – Simulation 3 - USP Inspector Actions.

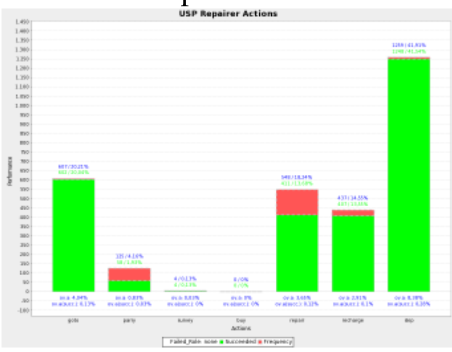


Figure 800: PGIM vs. USP – Simulation 3 - USP Repairer Actions.

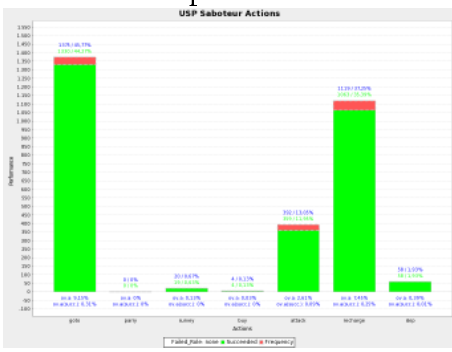


Figure 801: PGIM vs. USP – Simulation 3 - USP Saboteur Actions.

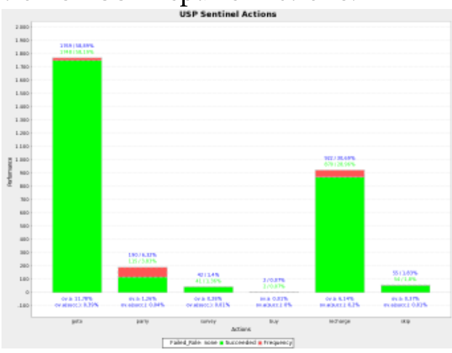


Figure 802: PGIM vs. USP – Simulation 3 - USP Sentinel Actions.

46 Python-DTU vs. UFSC – Simulation 1

46.1 Scores, Zone Stability and Achievements

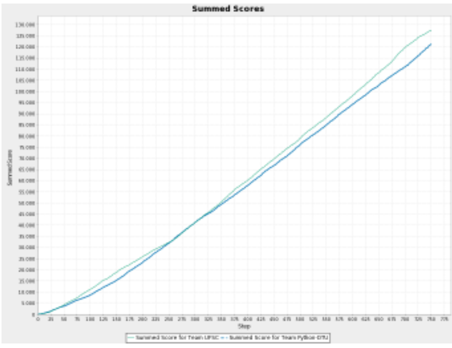


Figure 803: Summed scores.

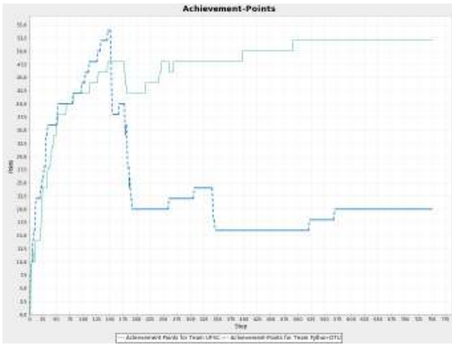


Figure 804: Achievement points.

Figure 805: Zones scores.

Figure 806: Zones scores and achievement points.

Figure 807: Zone Stabilities.

Step	UFSC	Python-DTU
1	surveyed10, surveyed40, surveyed20	surveyed10, surveyed40, area10, surveyed20
2	area10	
3	surveyed80, proved5	proved5
5	proved10, surveyed160	proved10, surveyed80
7		area20
9	proved20	
10	area20, attacked5	proved20, attacked5
11		surveyed160
19	parried5	area40
21	proved40	
22	area40	
23	surveyed320, area80	proved40
26		area80
29		attacked10, inspected5
30		surveyed320
32	attacked10	
33	inspected5	inspected10
38	parried10	
41	attacked20	
45	inspected10	
50	proved80	
51		attacked20
52	attacked40	proved80
68	parried20	
80		attacked40
81	attacked80	
96		surveyed640
103		proved160
110		attacked80
111	proved160	
126		parried5
127	attacked160	
132		parried10
145	parried40	parried20
166		area160
179		parried40
186		attacked160
215	inspected20	
241	parried80	
245	attacked320	
259		parried80
268	area160	
306		attacked320
396	attacked640	
490	parried160	
521		parried160
568		attacked640

Figure 808: Achievements.

46.2 Stability

Reason	UFSC	%	Python-DTU	%
failed away	11	0,07	2	0,01
failed parried	221	1,47	266	1,77
failed random	148	0,99	143	0,95
failed	3	0,02		
failed resources	3	0,02		
failed attacked	86	0,57	235	1,57
noAction	3	0,02		

Figure 809: Failed actions.

46.3 Achievements

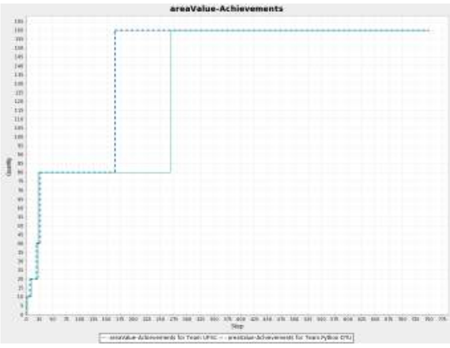


Figure 810: areaValueAchievements.

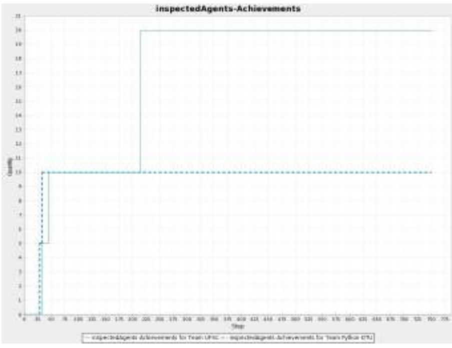


Figure 811: inspectedAgentsAchievements.

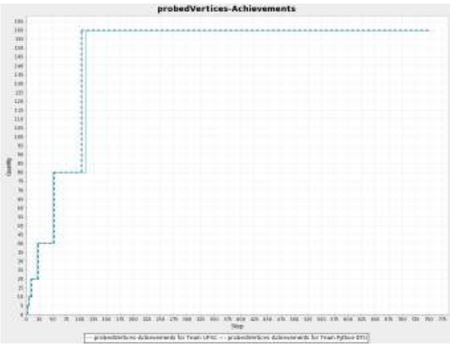


Figure 812: probedVerticesAchievements.

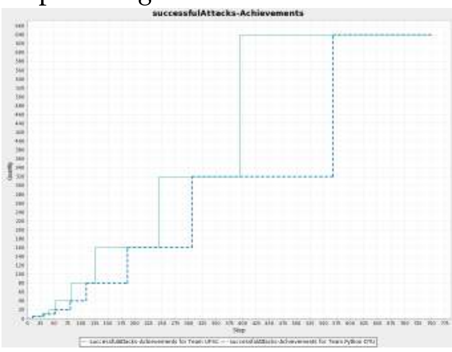


Figure 813: successfulAttacksAchievements.

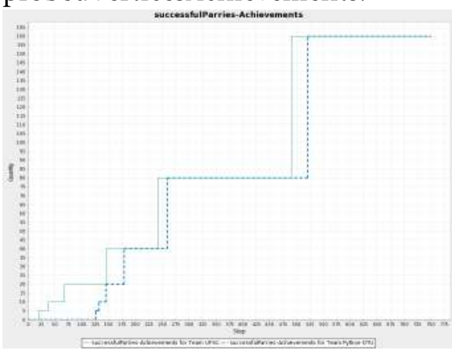


Figure 814: successfulParriesAchievements.

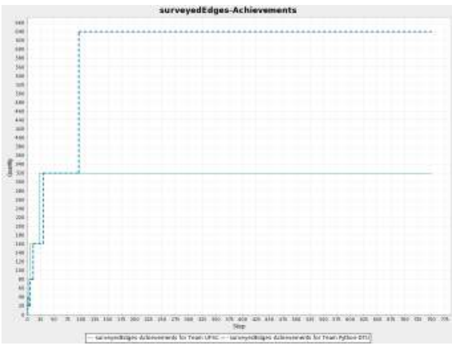


Figure 815: surveyedEdgesAchievements.

46.4 Actions per Role

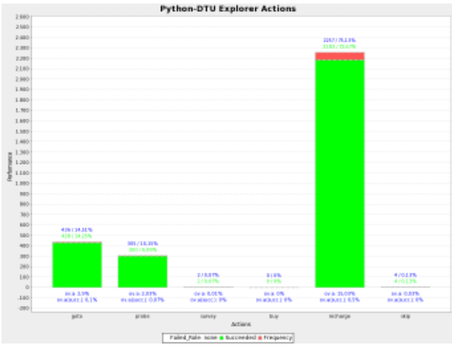


Figure 816: Python-DTU vs. UFSC – Simulation 1 - Python-DTU Explorer Actions.

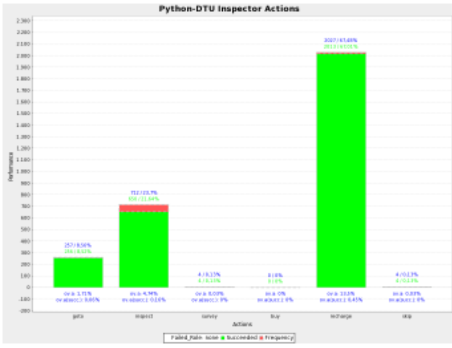


Figure 817: Python-DTU vs. UFSC – Simulation 1 - Python-DTU Inspector Actions.

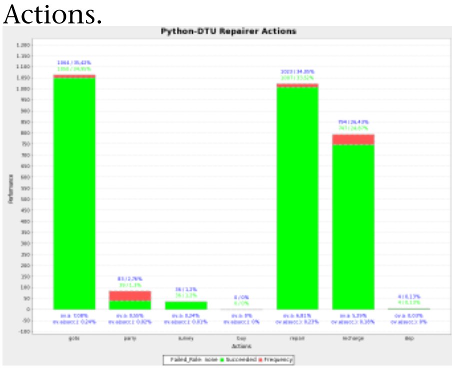


Figure 818: Python-DTU vs. UFSC – Simulation 1 - Python-DTU Repairer Actions.

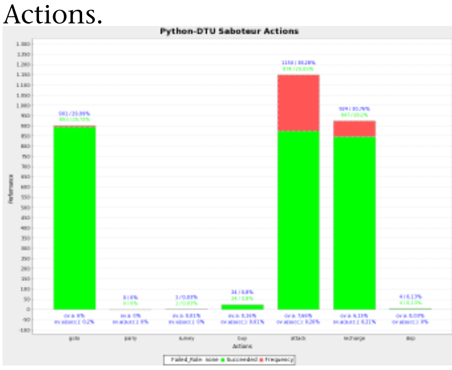


Figure 819: Python-DTU vs. UFSC – Simulation 1 - Python-DTU Saboteur Actions.

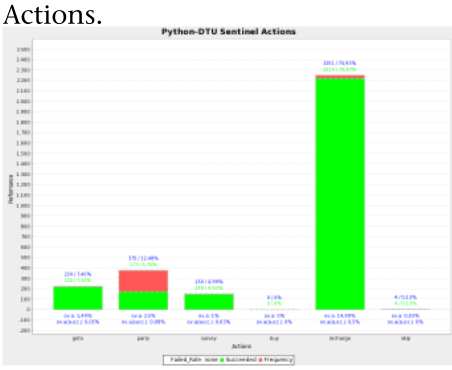


Figure 820: Python-DTU vs. UFSC – Simulation 1 - Python-DTU Sentinel Actions.

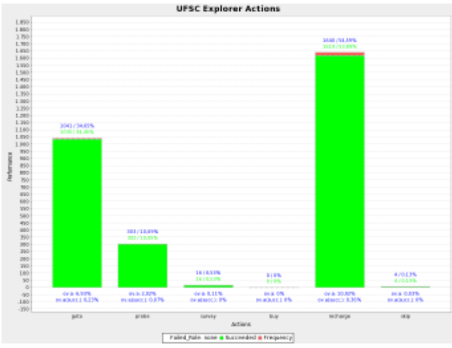


Figure 821: Python-DTU vs. UFSC – Simulation 1 - UFSC Explorer Actions.

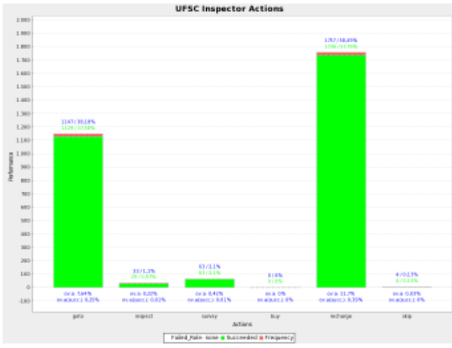


Figure 822: Python-DTU vs. UFSC – Simulation 1 - UFSC Inspector Actions.

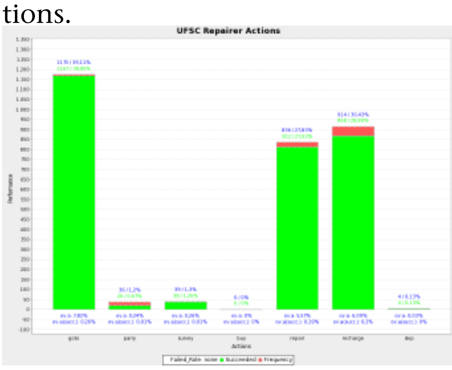


Figure 823: Python-DTU vs. UFSC – Simulation 1 - UFSC Repairer Actions.

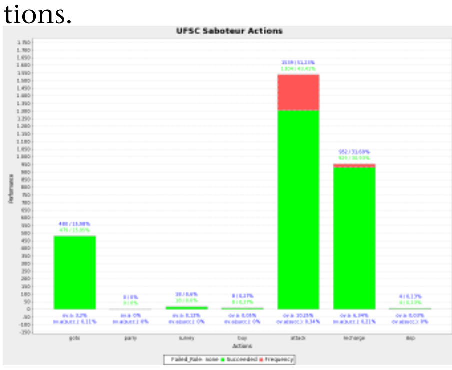


Figure 824: Python-DTU vs. UFSC – Simulation 1 - UFSC Saboteur Actions.

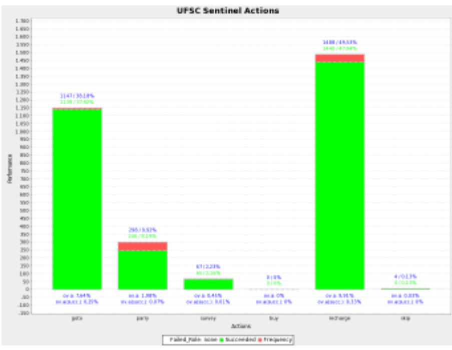


Figure 825: Python-DTU vs. UFSC – Simulation 1 - UFSC Sentinel Actions.

47 Python-DTU vs. UFSC – Simulation 2

47.1 Scores, Zone Stability and Achievements

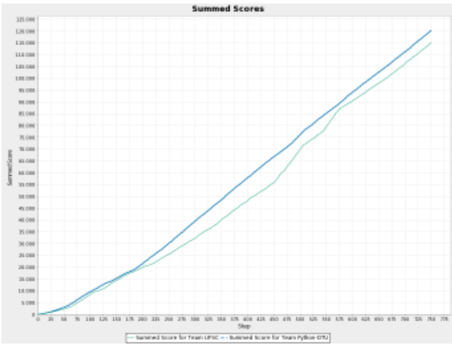


Figure 826: Summed scores.

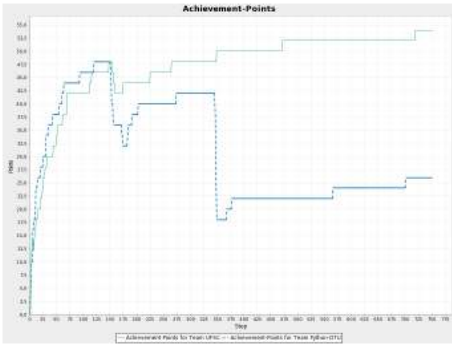


Figure 827: Achievement points.

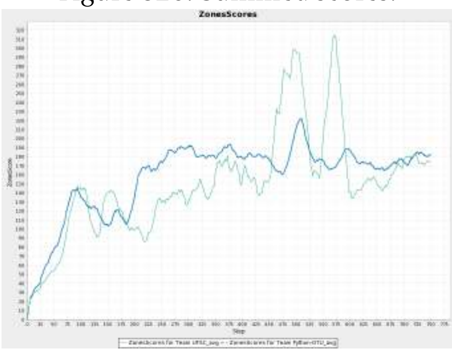


Figure 828: Zones scores.

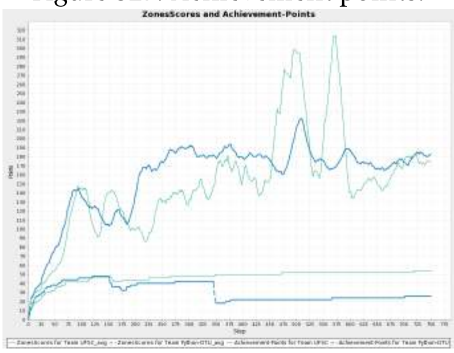


Figure 829: Zones scores and achievement points.

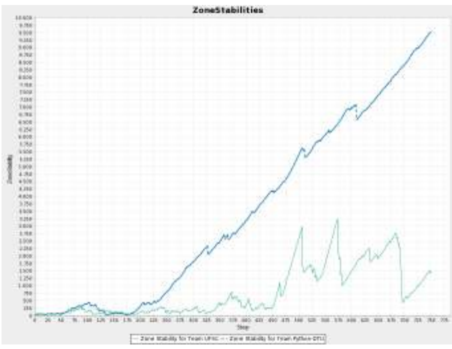


Figure 830: Zone Stabilities.

Step	UFSC	Python-DTU
1	surveyed10, surveyed20	surveyed10, surveyed20
2	surveyed40, inspected5	surveyed40
3	area10, proved5	proved5, inspected5
4	surveyed80	
5	proved10	proved10, surveyed80, attacked5
7		inspected10
8	attacked5	
9	inspected10, surveyed160	
10		area10
11	proved20	proved20, attacked10
12	parried5	
14		surveyed160
15	attacked10	
19	parried10	area20
23	proved40	
24	attacked20	attacked20
28	surveyed320	
30		proved40, inspected20
32	area20	
34		area40
42		surveyed320
43	attacked40	
49	proved80	
52	parried20	
55		attacked40
60		area80
61	area40	
63		proved80
69	attacked80, area80	
93		attacked80
112	proved160	
115	inspected20	
119		proved160
147	attacked160	
174	parried40	
182		parried5
183		attacked160
190		parried10
202		parried20
224	parried80	
264	attacked320	
273		parried40
348	area160	
367		attacked320
376		parried80
471	attacked640	
565		attacked640
701		parried160
719	parried160	

Figure 831: Achievements.

47.2 Stability

Reason	UFSC	%	Python-DTU	%
failed away	10	0,07		
failed parried	168	1,12	163	1,09
failed random	152	1,01	161	1,07
failed resources	2	0,01		
failed	13	0,09		
failed attacked	68	0,45	206	1,37
noAction	13	0,09		

Figure 832: Failed actions.

47.3 Achievements

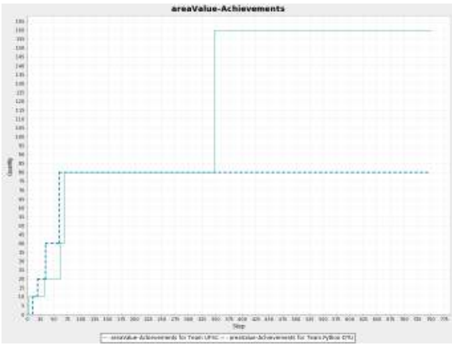


Figure 833: areaValueAchievements.

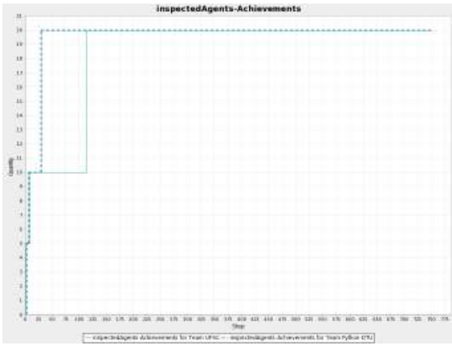


Figure 834: inspectedAgentsAchievements.

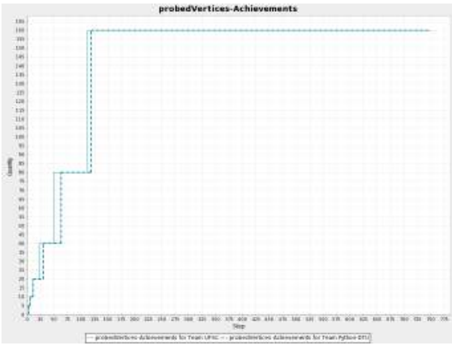


Figure 835: probedVerticesAchievements.

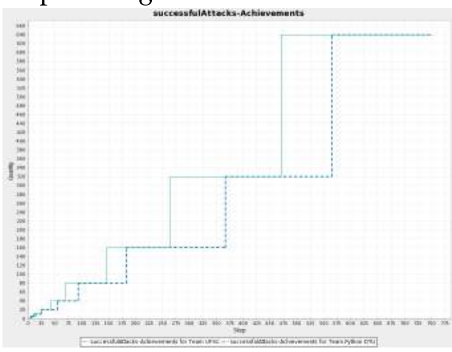


Figure 836: successfulAttacksAchievements.

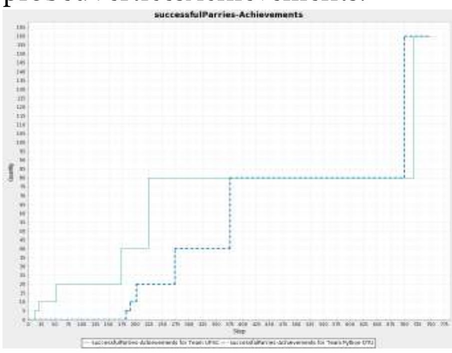


Figure 837: successfulParriesAchievements.

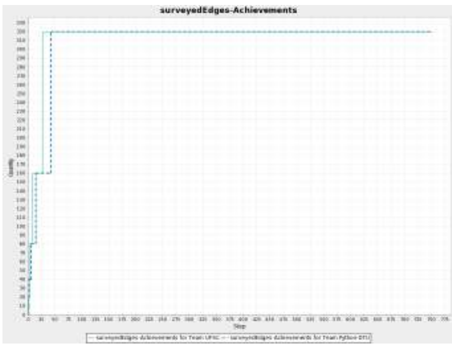


Figure 838: surveyedEdgesAchievements.

47.4 Actions per Role

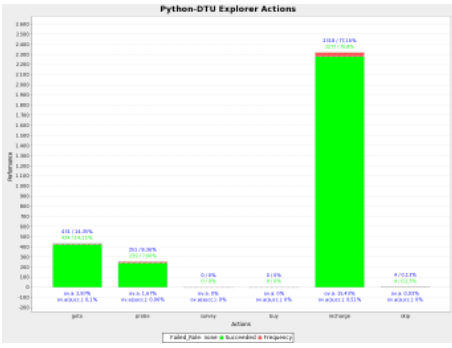


Figure 839: Python-DTU vs. UFSC – Simulation 2 - Python-DTU Explorer Actions.

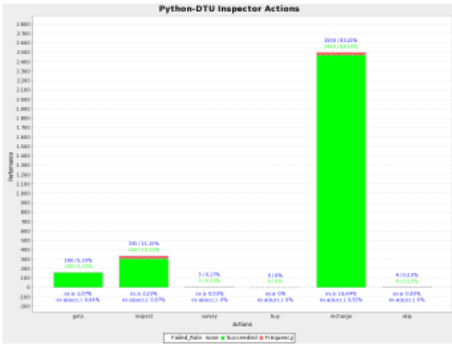


Figure 840: Python-DTU vs. UFSC – Simulation 2 - Python-DTU Inspector Actions.

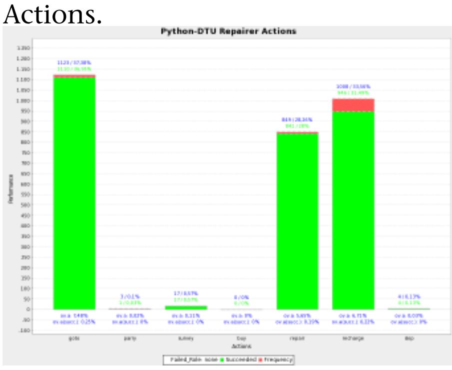


Figure 841: Python-DTU vs. UFSC – Simulation 2 - Python-DTU Repairer Actions.

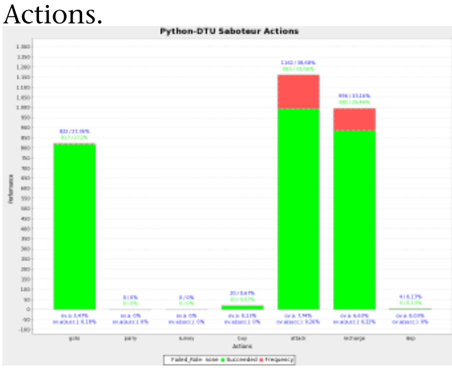


Figure 842: Python-DTU vs. UFSC – Simulation 2 - Python-DTU Saboteur Actions.

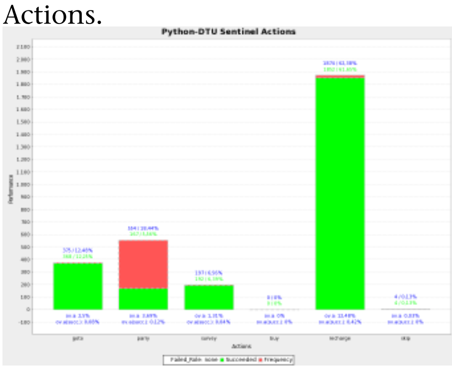


Figure 843: Python-DTU vs. UFSC – Simulation 2 - Python-DTU Sentinel Actions.

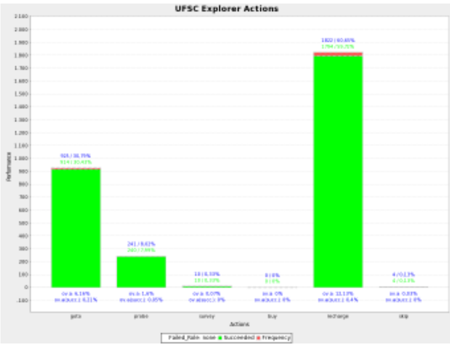


Figure 844: Python-DTU vs. UFSC – Simulation 2 - UFSC Explorer Actions.

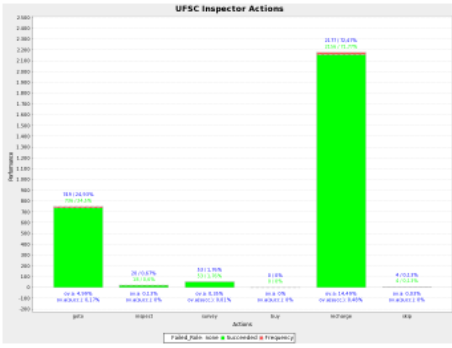


Figure 845: Python-DTU vs. UFSC – Simulation 2 - UFSC Inspector Actions.

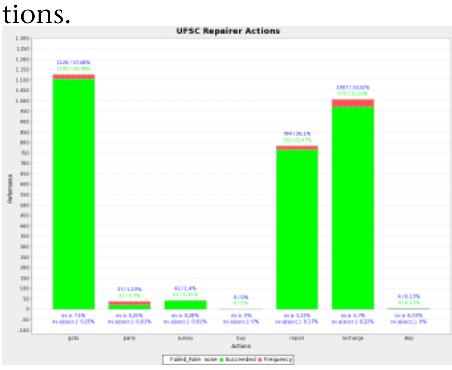


Figure 846: Python-DTU vs. UFSC – Simulation 2 - UFSC Repairer Actions.

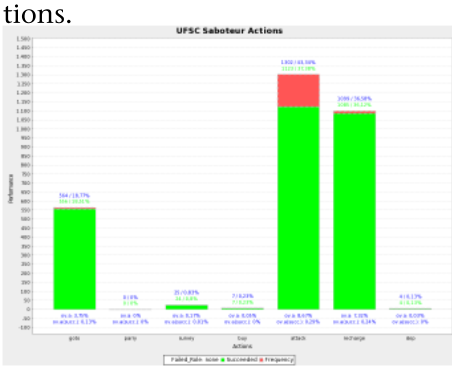


Figure 847: Python-DTU vs. UFSC – Simulation 2 - UFSC Saboteur Actions.

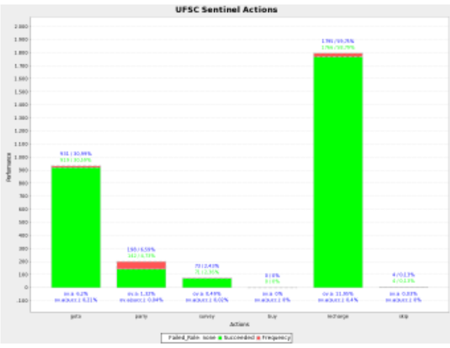


Figure 848: Python-DTU vs. UFSC – Simulation 2 - UFSC Sentinel Actions.

48 Python-DTU vs. UFSC – Simulation 3

48.1 Scores, Zone Stability and Achievements

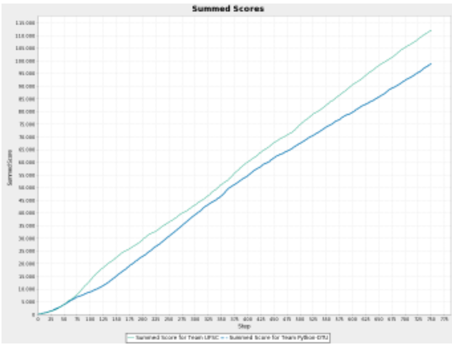


Figure 849: Summed scores.

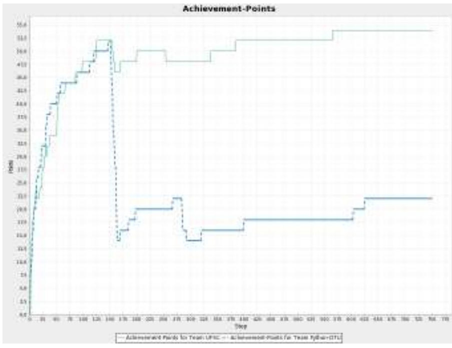


Figure 850: Achievement points.

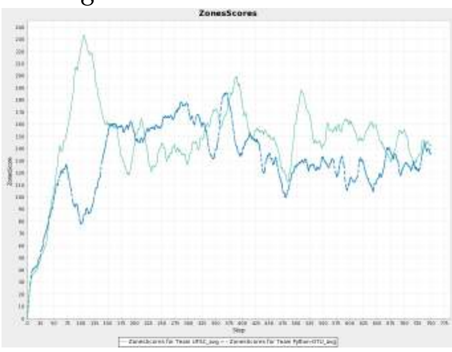


Figure 851: Zones scores.

Figure 852: Zones scores and achievement points.

Figure 853: Zone Stabilities.

Step	UFSC	Python-DTU
1	surveyed10, area20, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
3	surveyed80, proved5	proved5
4		inspected5
5	proved10, inspected5	proved10, surveyed80
7	surveyed160	area20, attacked5
8	attacked5	
9	proved20	
11	attacked10	proved20
13		inspected10, attacked10
16	inspected10	surveyed160
18	attacked20	
21		area40
22		attacked20
23	area40, proved40	
27	surveyed320	
29		proved40
30	attacked40	area80
32	parried5	inspected20
37	area80	
38		attacked40
51	attacked80, proved80	surveyed320
52	inspected20	
54	parried10	
57		proved80
66	area160	
85	parried20	
88		attacked80
98	attacked160	
112		proved160
119		parried5
123	proved160	
124	parried40	
146		parried10
162		parried20
168		area160
169	attacked320	
184		attacked160
197		parried40
199	parried80	
266		parried80
320		attacked320
337	parried160	
384	attacked640	
398		parried160
564	parried320	
603		parried320
624		attacked640

Figure 854: Achievements.

48.2 Stability

Reason	UFSC	%	Python-DTU	%
failed away	14	0,09	1	0,01
failed parried	404	2,69	445	2,97
failed random	129	0,86	153	1,02
failed resources	8	0,05		
failed	17	0,11		
failed attacked	104	0,69	310	2,07
noAction	17	0,11		

Figure 855: Failed actions.

48.3 Achievements

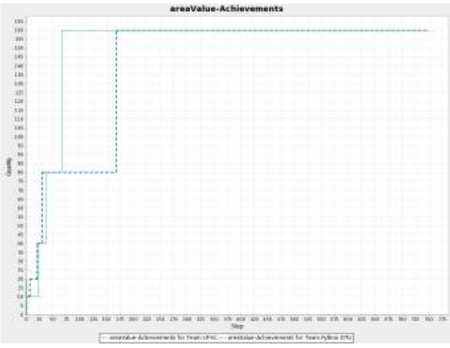


Figure 856: areaValueAchievements.

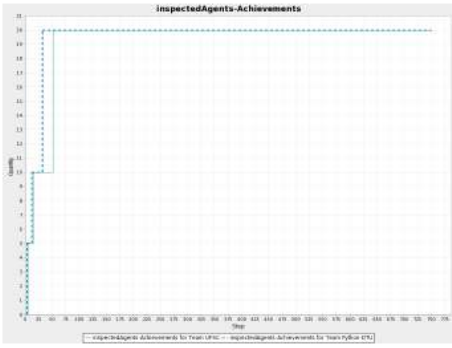


Figure 857: inspectedAgentsAchievements.

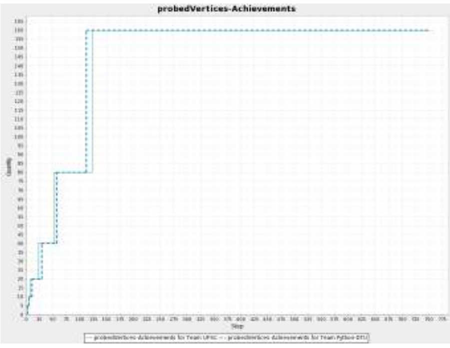


Figure 858: probedVerticesAchievements.

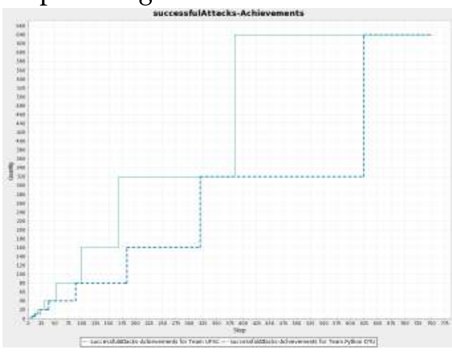


Figure 859: successfulAttacksAchievements.

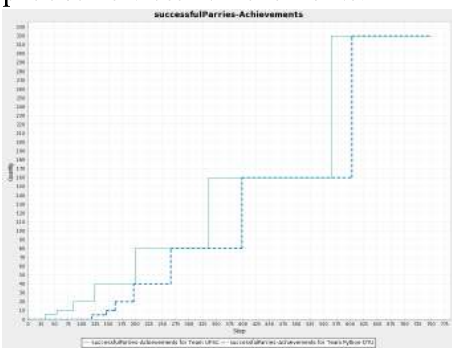


Figure 860: successfulParriesAchievements.

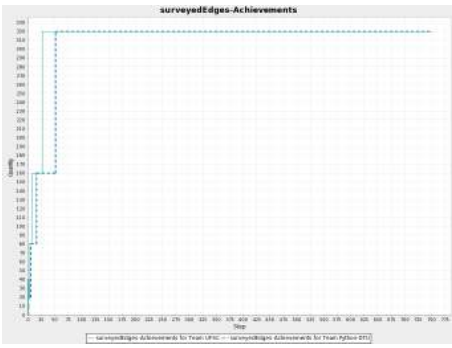


Figure 861: surveyedEdgesAchievements.

48.4 Actions per Role

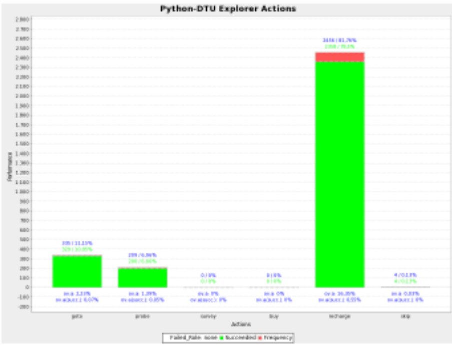


Figure 862: Python-DTU vs. UFSC – Simulation 3 - Python-DTU Explorer Actions.

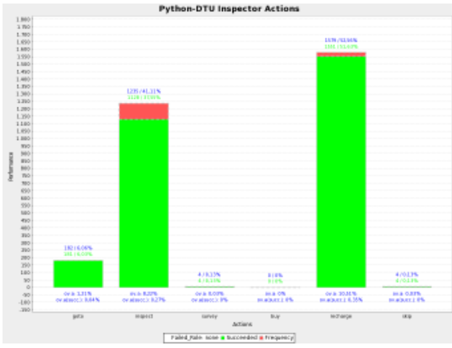


Figure 863: Python-DTU vs. UFSC – Simulation 3 - Python-DTU Inspector Actions.

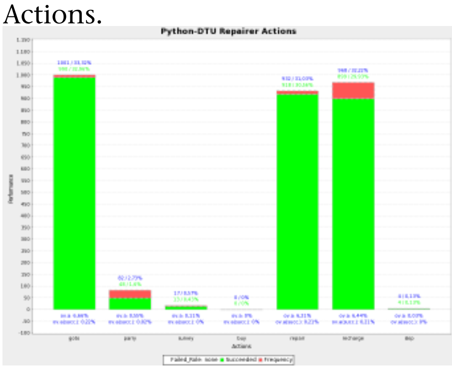


Figure 864: Python-DTU vs. UFSC – Simulation 3 - Python-DTU Repairer Actions.

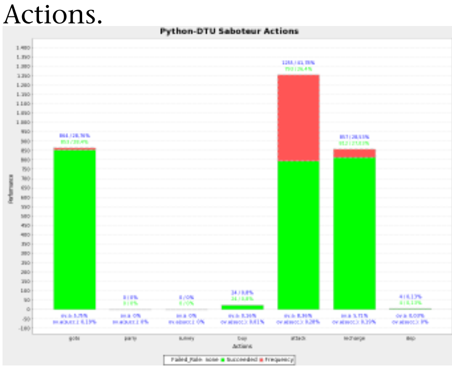


Figure 865: Python-DTU vs. UFSC – Simulation 3 - Python-DTU Saboteur Actions.

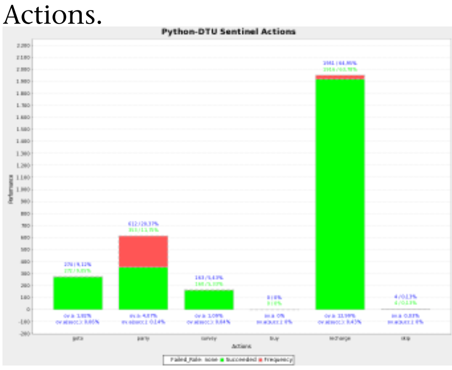


Figure 866: Python-DTU vs. UFSC – Simulation 3 - Python-DTU Sentinel Actions.

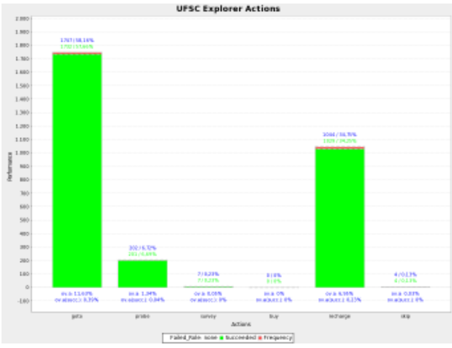


Figure 867: Python-DTU vs. UFSC – Simulation 3 - UFSC Explorer Actions.

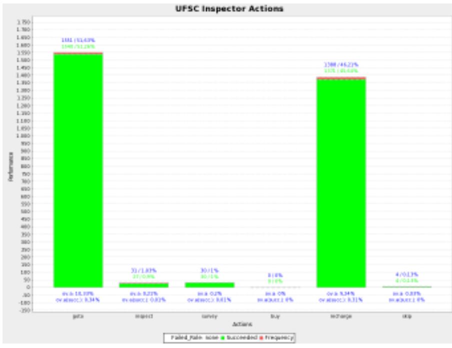


Figure 868: Python-DTU vs. UFSC – Simulation 3 - UFSC Inspector Actions.

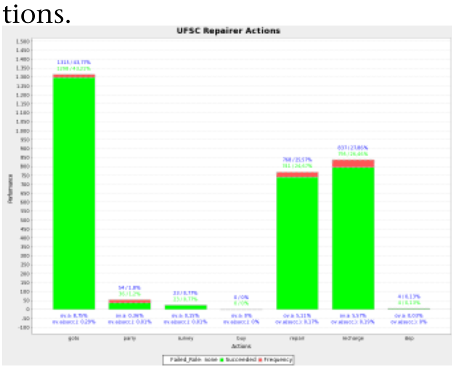


Figure 869: Python-DTU vs. UFSC – Simulation 3 - UFSC Repairer Actions.

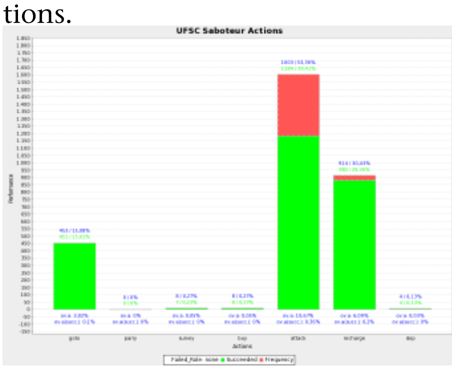


Figure 870: Python-DTU vs. UFSC – Simulation 3 - UFSC Saboteur Actions.

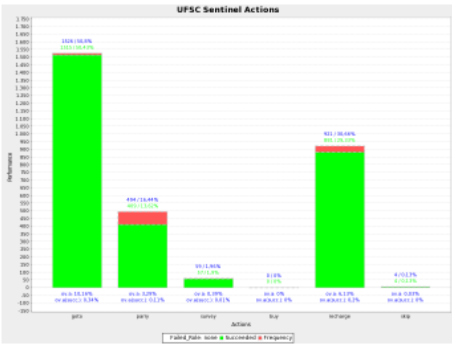


Figure 871: Python-DTU vs. UFSC – Simulation 3 - UFSC Sentinel Actions.

49 Streett vs. Python-DTU – Simulation 1

49.1 Scores, Zone Stability and Achievements

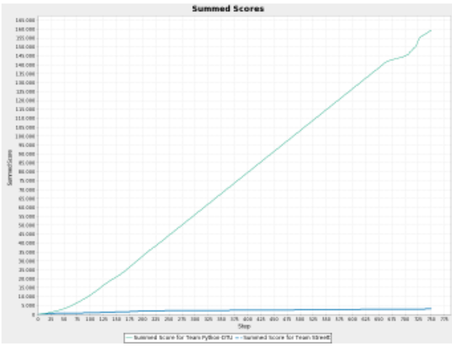


Figure 872: Summed scores.

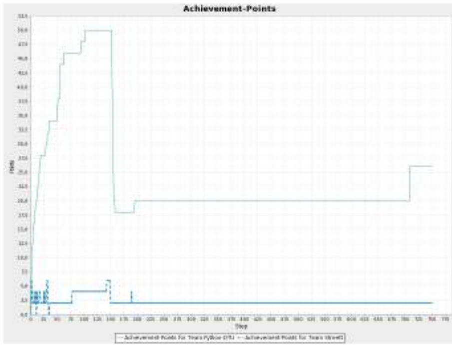


Figure 873: Achievement points.

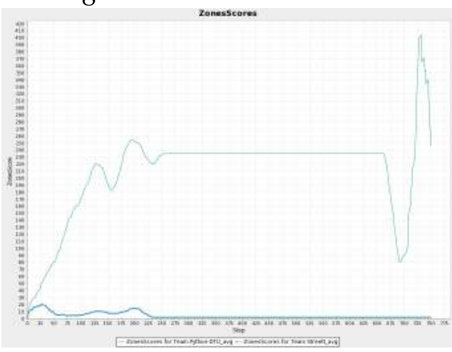


Figure 874: Zones scores.

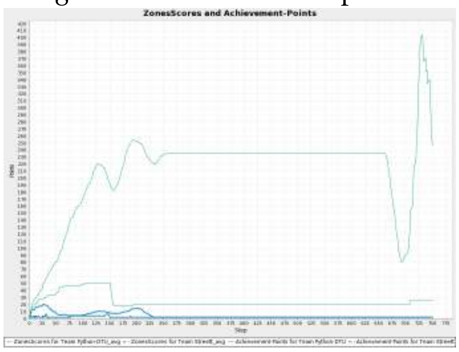


Figure 875: Zones scores and achievement points.

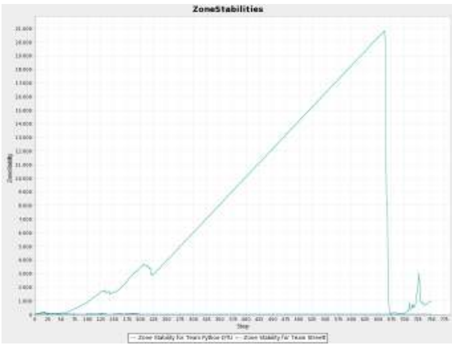


Figure 876: Zone Stabilities.

Step	Python-DTU	Streett
1	surveyed10, surveyed40, area10, surveyed20	surveyed40, surveyed10, surveyed20
2		surveyed80
3	surveyed80, proved5	proved5
4	inspected5	
5	proved10	
6		proved10
8	attacked5	
9	surveyed160	attacked5
11		proved20
12	inspected10	area10
14	attacked10	
16	proved20	attacked10
18	area20	
24		surveyed160
26		inspected5
28	attacked20	
29		inspected10
30		proved40
31	proved40	
34	surveyed320	
35		attacked20
49	area40	
51	inspected20	
55	attacked40, proved80, area80	
62	parried5	
77		proved80
94	surveyed640	
102	proved160	
142		surveyed320
188		area20
194	attacked80	
708	area160, area320, area640	

Figure 877: Achievements.

49.2 Stability

Reason	Python-DTU	%	Streett	%
failed away	42	0,28	14	0,09
failed parried			5	0,03
failed random	149	0,99	167	1,11
failed	211	1,41	20	0,13
failed resources			504	3,36
failed attacked	6	0,04	30	0,2
noAction	214	1,43	20	0,13

Figure 878: Failed actions.

49.3 Achievements

Figure 879: areaValueAchievements.

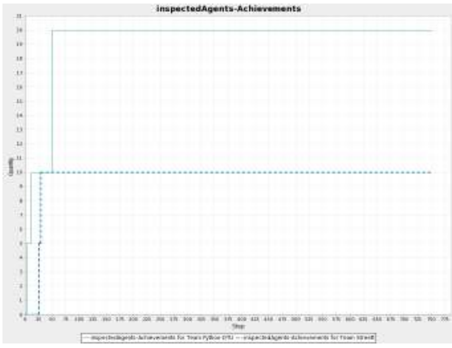


Figure 880: inspectedAgentsAchievements.

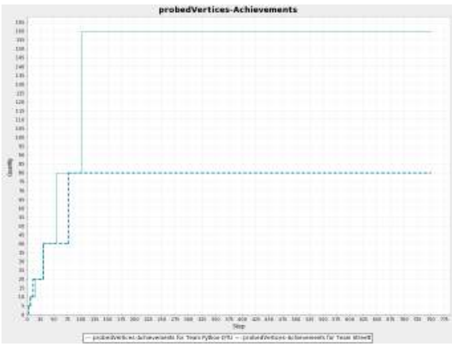


Figure 881: probedVerticesAchievements.

Figure 882: successfulAttacksAchievements.

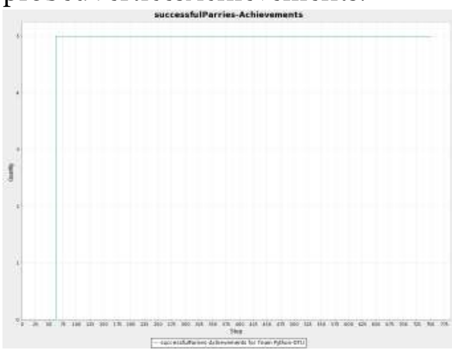


Figure 883: successfulParriesAchievements.

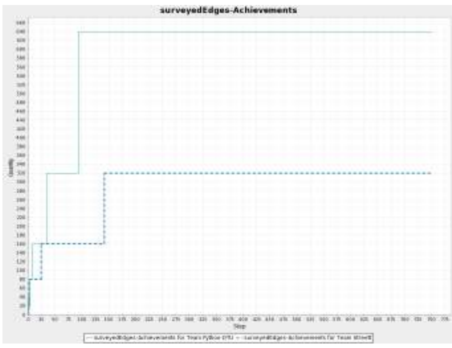


Figure 884: surveyedEdgesAchievements.

49.4 Actions per Role

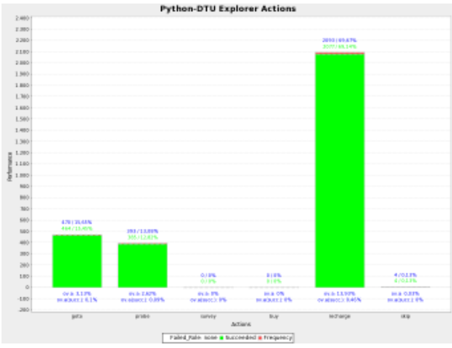


Figure 885: Streett vs. Python-DTU – Simulation 1 - Python-DTU Explorer Actions.

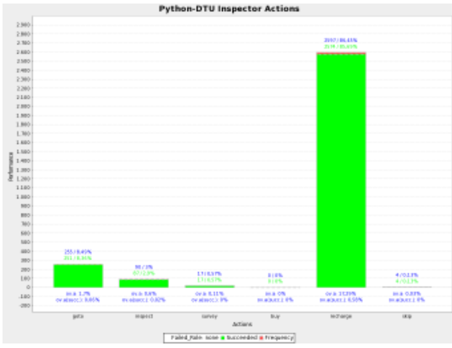


Figure 886: Streett vs. Python-DTU – Simulation 1 - Python-DTU Inspector Actions.

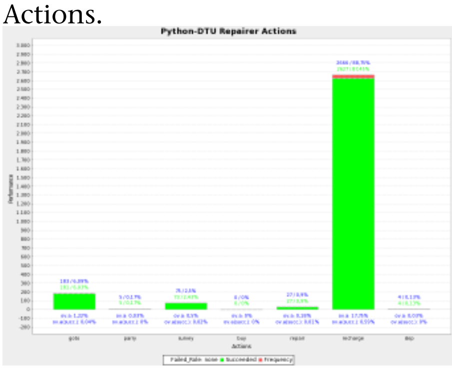


Figure 887: Streett vs. Python-DTU – Simulation 1 - Python-DTU Repairer Actions.

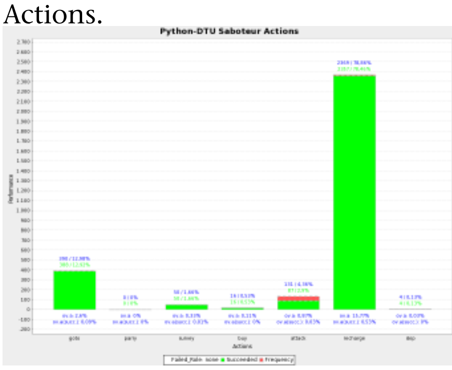


Figure 888: Streett vs. Python-DTU – Simulation 1 - Python-DTU Saboteur Actions.

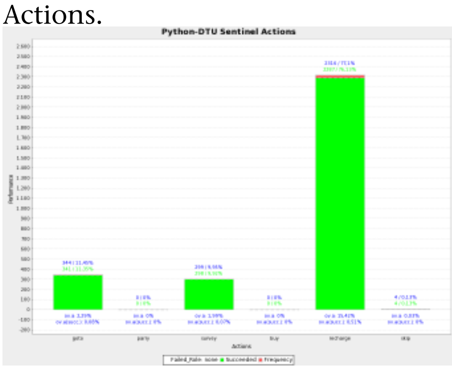


Figure 889: Streett vs. Python-DTU – Simulation 1 - Python-DTU Sentinel Actions.

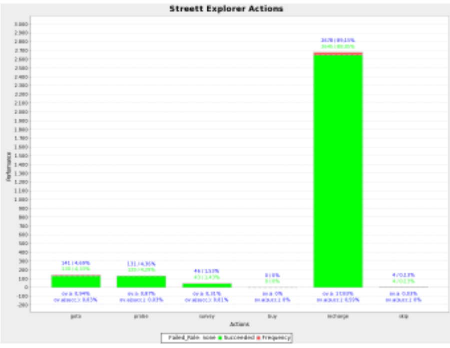


Figure 890: Streett vs. Python-DTU – Simulation 1 - Streett Explorer Actions.

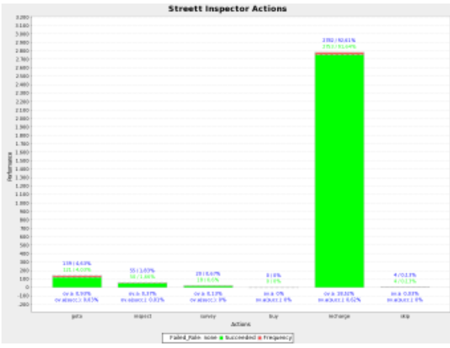


Figure 891: Streett vs. Python-DTU – Simulation 1 - Streett Inspector Actions.

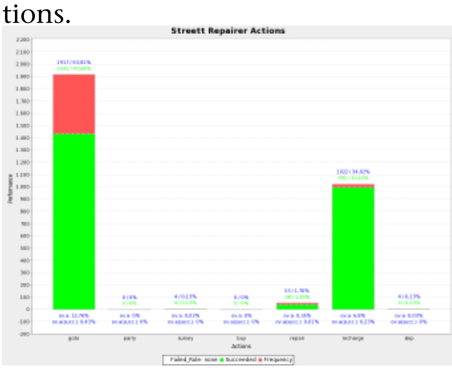


Figure 892: Streett vs. Python-DTU – Simulation 1 - Streett Repairer Actions.

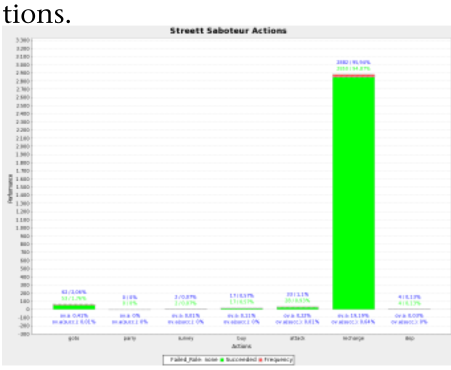


Figure 893: Streett vs. Python-DTU – Simulation 1 - Streett Saboteur Actions.

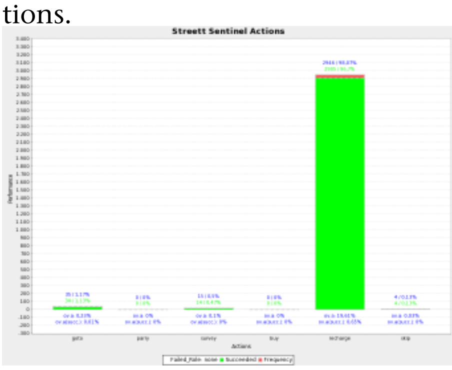


Figure 894: Streett vs. Python-DTU – Simulation 1 - Streett Sentinel Actions.

50 Streett vs. Python-DTU – Simulation 2

50.1 Scores, Zone Stability and Achievements

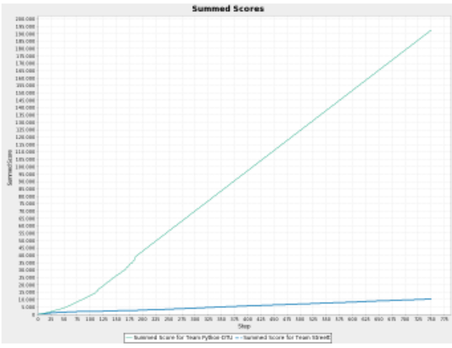


Figure 895: Summed scores.

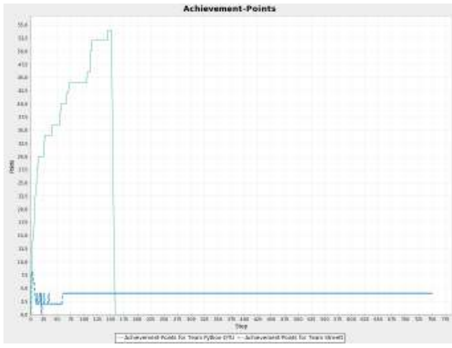


Figure 896: Achievement points.

Figure 897: Zones scores.

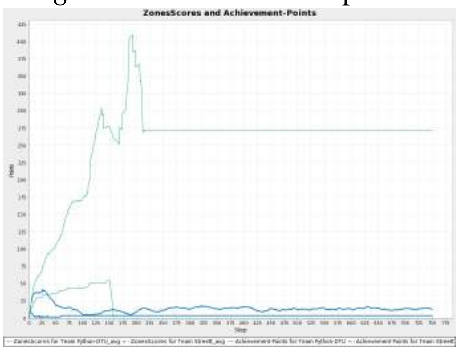


Figure 898: Zones scores and achievement points.

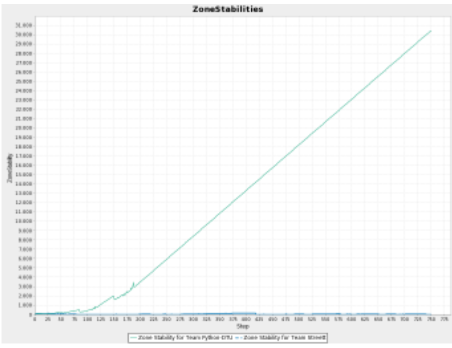


Figure 899: Zone Stabilities.

Step	Python-DTU	Streett
1	surveyed10, area20, area10, surveyed20	surveyed10, area20, surveyed20, area10
2	surveyed40	surveyed40
3	proved5, inspected5	proved5
5	proved10	inspected5
6	surveyed80	proved10
7	area40, attacked5	
8		surveyed80
9	proved20	
11	inspected10	proved20
12	surveyed160	
14	attacked10	
16		inspected10
19		attacked5
21		area40
24	proved40	
25		proved40
26	surveyed320	surveyed160
33		attacked10
40	attacked20	
54	proved80	
56	inspected20	
59		proved80
67	area80	
71	attacked40	
105	proved160	
111	area160, area320	
114	area640	
144	attacked80	

Figure 900: Achievements.

50.2 Stability

Reason	Python-DTU	%	Streett	%
failed away	28	0,19		
failed random	157	1,05	161	1,07
failed resources			174	1,16
failed attacked	2	0,01	24	0,16

Figure 901: Failed actions.

50.3 Achievements

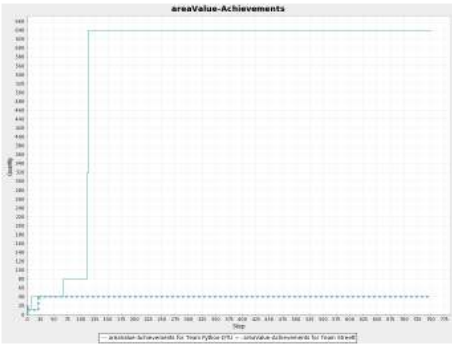


Figure 902: areaValueAchievements.

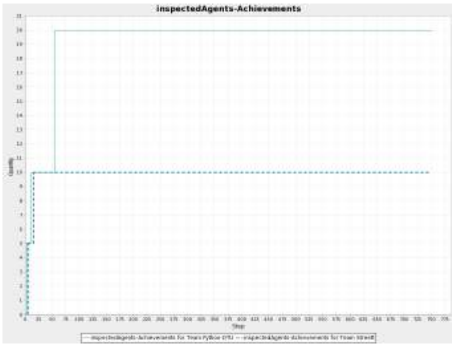


Figure 903: inspectedAgentsAchievements.

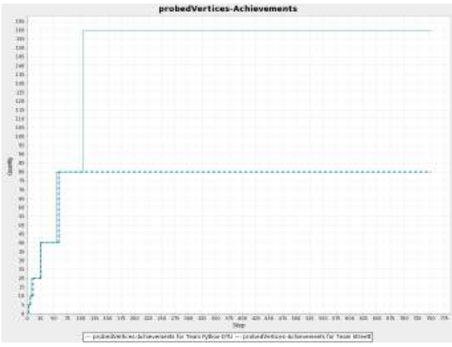


Figure 904: probedVerticesAchievements.

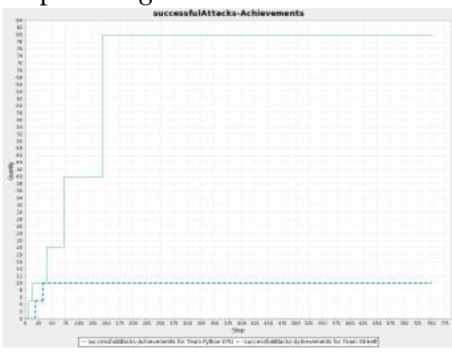


Figure 905: successfulAttacksAchievements.

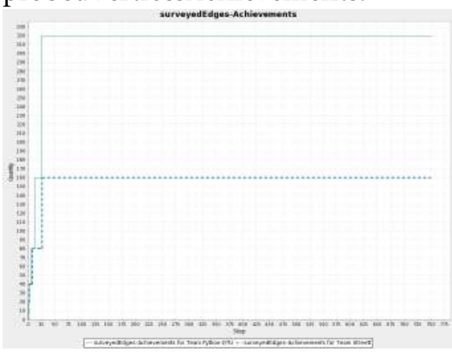


Figure 906: surveyedEdgesAchievements.

50.4 Actions per Role

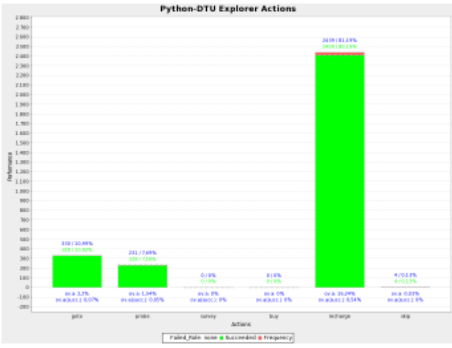


Figure 907: Streett vs. Python-DTU – Simulation 2 - Python-DTU Explorer Actions.

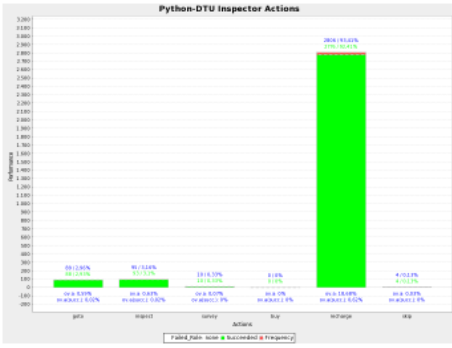


Figure 908: Streett vs. Python-DTU – Simulation 2 - Python-DTU Inspector Actions.

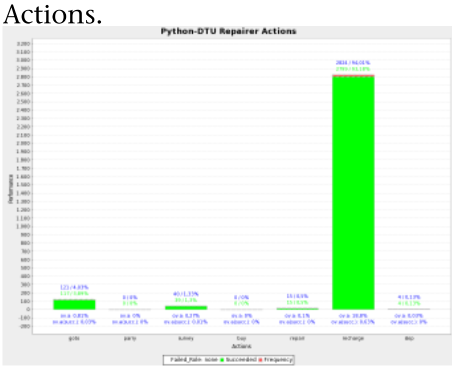


Figure 909: Streett vs. Python-DTU – Simulation 2 - Python-DTU Repairer Actions.

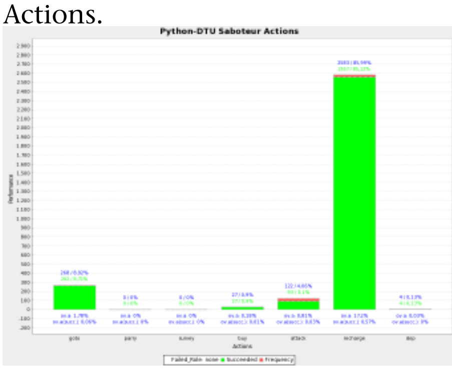


Figure 910: Streett vs. Python-DTU – Simulation 2 - Python-DTU Saboteur Actions.

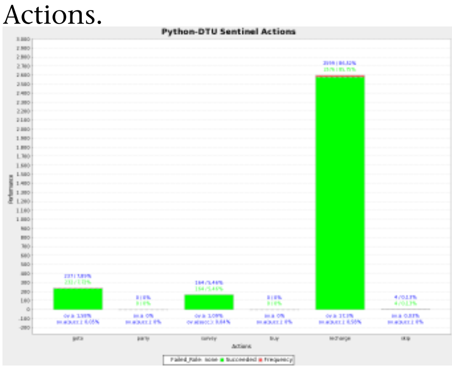


Figure 911: Streett vs. Python-DTU – Simulation 2 - Python-DTU Sentinel Actions.

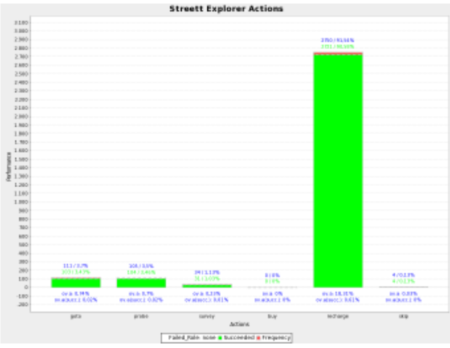


Figure 912: Streett vs. Python-DTU – Simulation 2 - Streett Explorer Actions.

Figure 913: Streett vs. Python-DTU – Simulation 2 - Streett Inspector Actions.

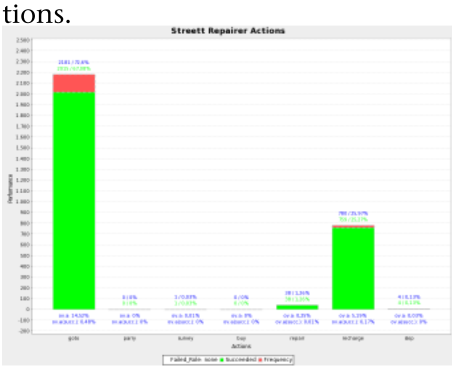


Figure 914: Streett vs. Python-DTU – Simulation 2 - Streett Repairer Actions.

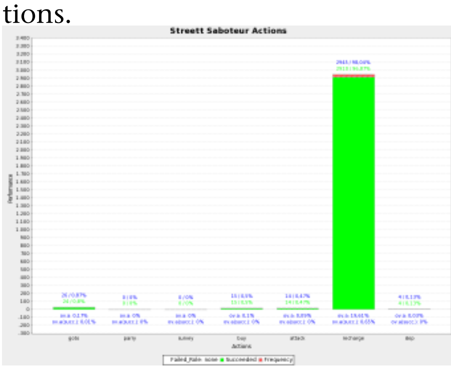


Figure 915: Streett vs. Python-DTU – Simulation 2 - Streett Saboteur Actions.

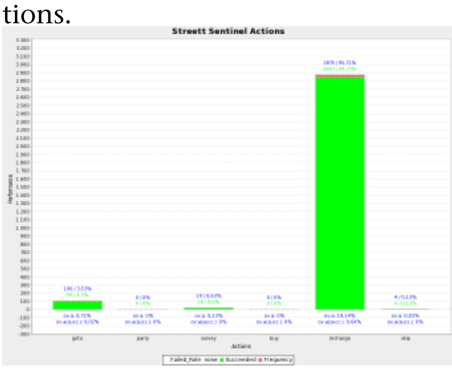


Figure 916: Streett vs. Python-DTU – Simulation 2 - Streett Sentinel Actions.

51 Streett vs. Python-DTU – Simulation 3

51.1 Scores, Zone Stability and Achievements

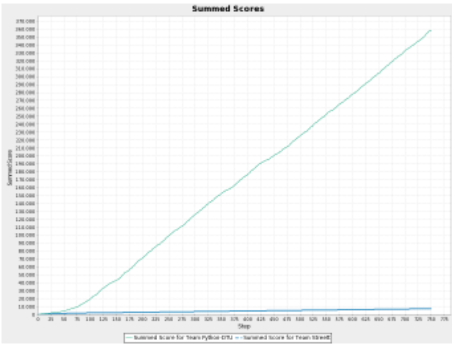


Figure 917: Summed scores.

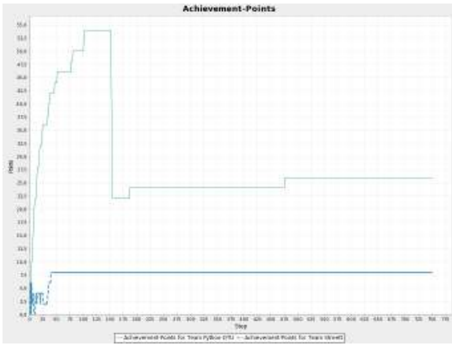


Figure 918: Achievement points.

Figure 919: Zones scores.

Figure 920: Zones scores and achievement points.

Figure 921: Zone Stabilities.

Step	Python-DTU	Streett
1	surveyed10, area10, surveyed20	surveyed40, surveyed10, surveyed20
3	surveyed40, proved5	area10, proved5, surveyed80
4		attacked5
5	area20, proved10	
6	attacked5	proved10
7	surveyed80, inspected5	
9	proved20	
10		attacked10
11		proved20
12	attacked10	area20
13	area40	
14	inspected10	inspected5
17	surveyed160	
18	attacked20	
20		surveyed160
23	proved40, area80	
33	inspected20	attacked20
35	attacked40	proved40
37	surveyed320	
39		inspected10
46	proved80	
51	area160	
77	area320	
80	attacked80	
100	area640	
102	proved160	
186	attacked160	
475	attacked320	

Figure 922: Achievements.

51.2 Stability

Reason	Python-DTU	%	Streett	%
failed away	1	0,01		
failed random	135	0,9	134	0,89
failed			3	0,02
failed resources			239	1,59
failed attacked	12	0,08	116	0,77
noAction			4	0,03
failed status			1	0,01

Figure 923: Failed actions.

51.3 Achievements

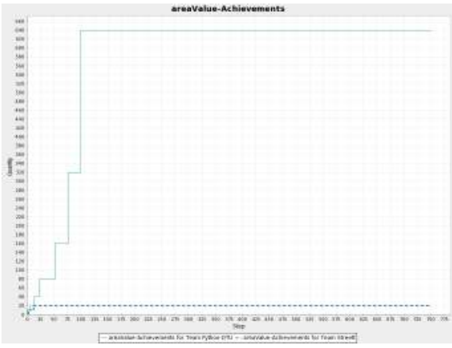


Figure 924: areaValueAchievements.

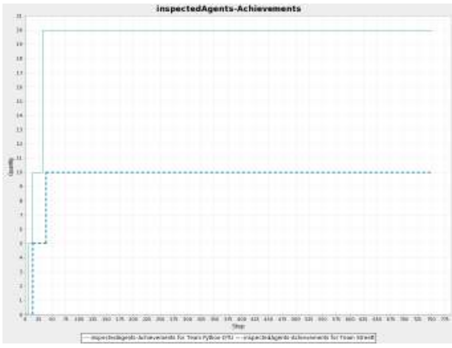


Figure 925: inspectedAgentsAchievements.

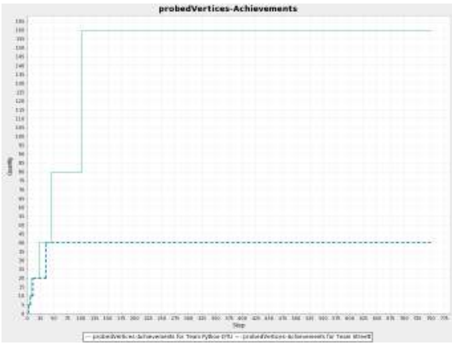


Figure 926: probedVerticesAchievements.

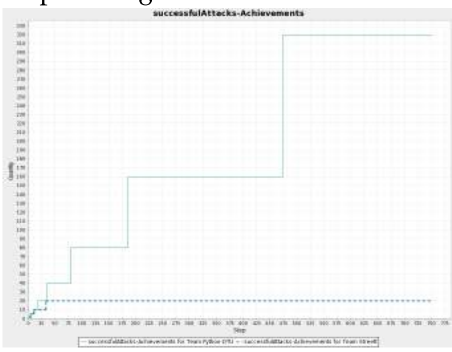


Figure 927: successfulAttacksAchievements.

Figure 928: surveyedEdgesAchievements.

51.4 Actions per Role

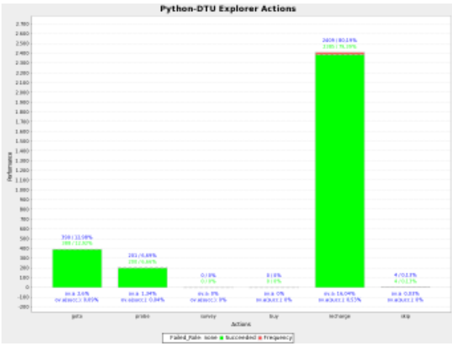


Figure 929: Streett vs. Python-DTU – Simulation 3 - Python-DTU Explorer Actions.

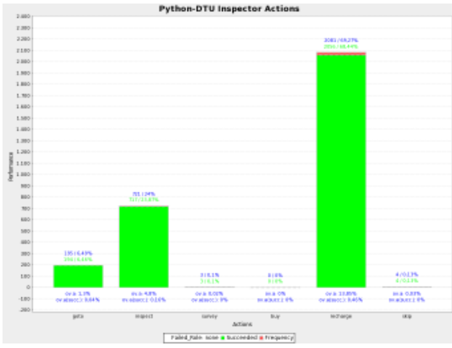


Figure 930: Streett vs. Python-DTU – Simulation 3 - Python-DTU Inspector Actions.

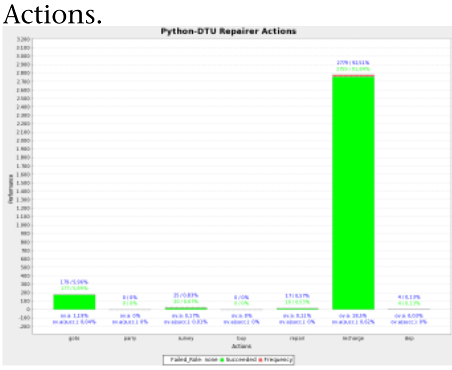


Figure 931: Streett vs. Python-DTU – Simulation 3 - Python-DTU Repairer Actions.

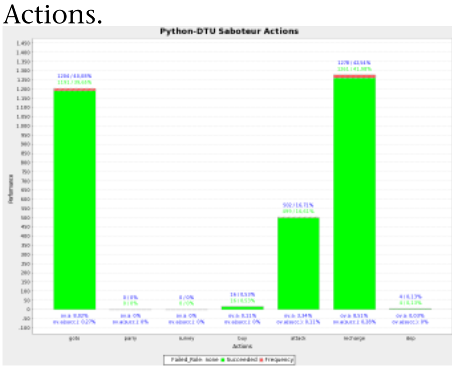


Figure 932: Streett vs. Python-DTU – Simulation 3 - Python-DTU Saboteur Actions.

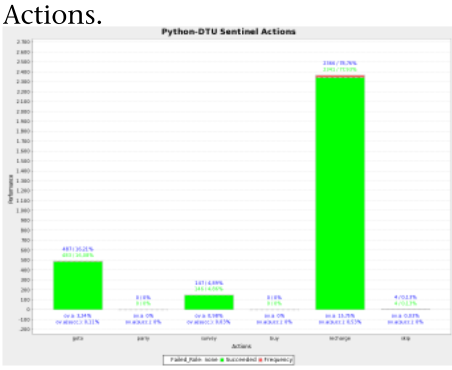


Figure 933: Streett vs. Python-DTU – Simulation 3 - Python-DTU Sentinel Actions.

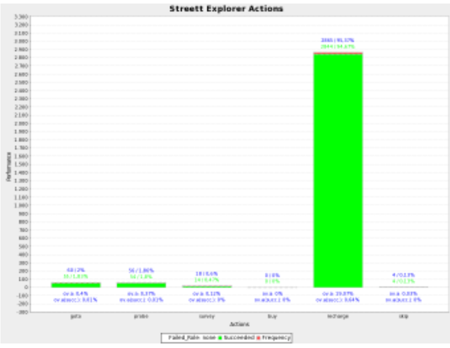


Figure 934: Streett vs. Python-DTU – Simulation 3 - Streett Explorer Actions.

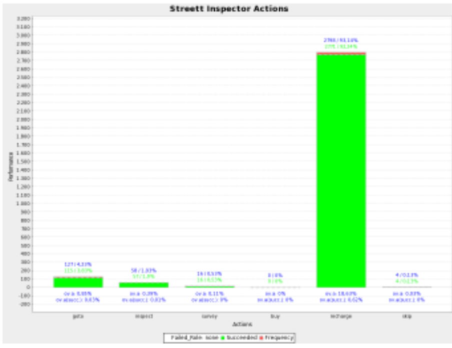


Figure 935: Streett vs. Python-DTU – Simulation 3 - Streett Inspector Actions.

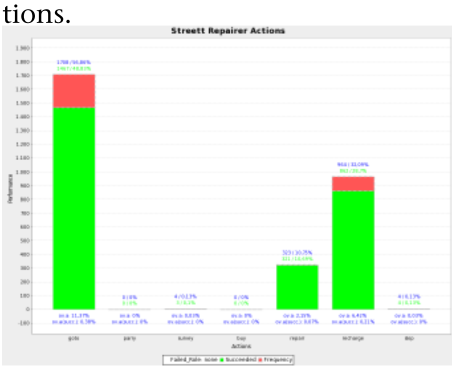


Figure 936: Streett vs. Python-DTU – Simulation 3 - Streett Repairer Actions.

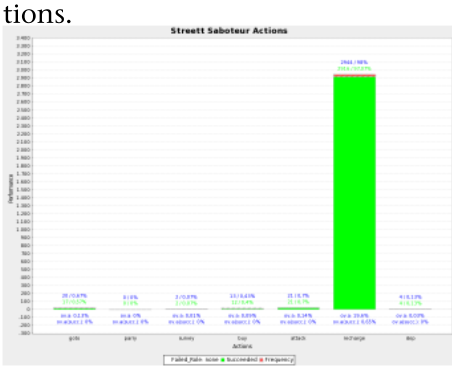


Figure 937: Streett vs. Python-DTU – Simulation 3 - Streett Saboteur Actions.

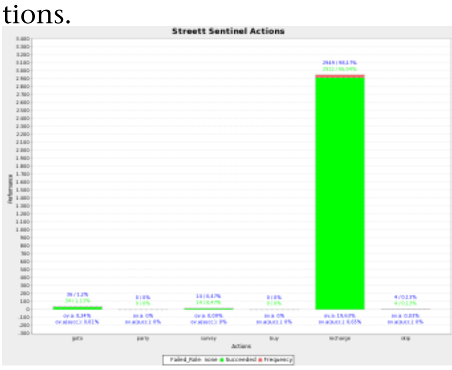


Figure 938: Streett vs. Python-DTU – Simulation 3 - Streett Sentinel Actions.

52 Streett vs. TUB – Simulation 1

52.1 Scores, Zone Stability and Achievements

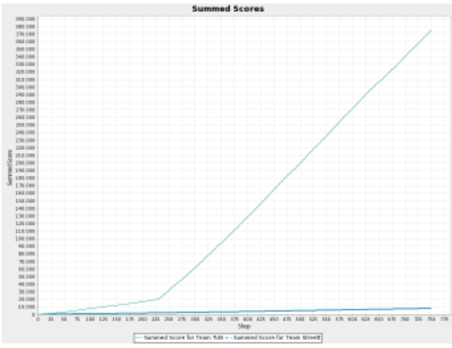


Figure 939: Summed scores.

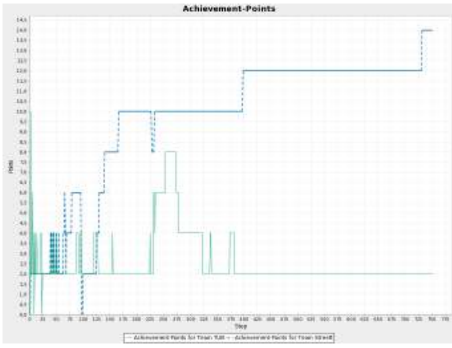


Figure 940: Achievement points.

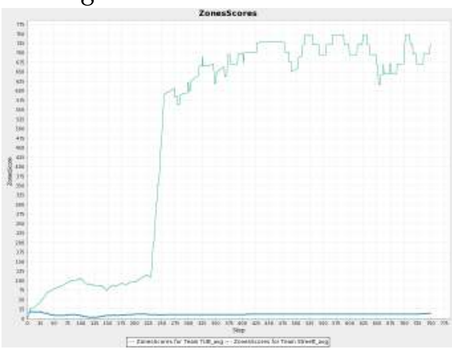


Figure 941: Zones scores.

Figure 942: Zones scores and achievement points.

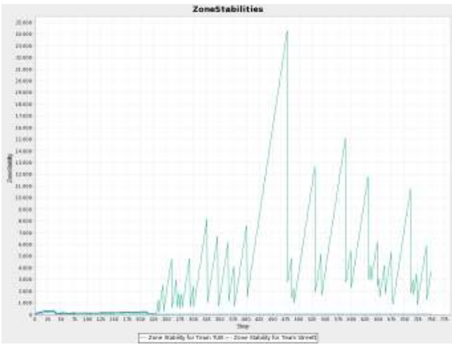


Figure 943: Zone Stabilities.

Step	TUB	Streett
1	surveyed10, surveyed80, surveyed40, area10, surveyed20	area10
4	area20, proved5	
7	inspected5	
9	proved10, surveyed160	
10	attacked5	
13	inspected10	
14	area40	
20	proved20	
21	area80	
23	attacked10	
38		surveyed10
39		surveyed20
40	proved40	
42		inspected5
44		surveyed40
45	attacked20	
49	surveyed320	proved5
53		inspected10
62		proved10
65		attacked5
68		surveyed80
78		proved20
86	proved80	
94	attacked40	
99		attacked10
119	attacked80	
124		attacked20
129		proved40
139		surveyed160
154	attacked160	
165		attacked40
224	proved160	
231	area160, area320	
232		attacked80
234	attacked320	
253	area640	
336	surveyed640	
373	attacked640	
397		attacked160
731		attacked320

Figure 944: Achievements.

52.2 Stability

Reason	TUB	%	Streett	%
failed away			428	2,85
failed wrong param	2	0,01		
failed random	152	1,01	160	1,07
failed resources	1	0,01	112	0,75
failed			730	4,87
failed attacked	53	0,35	89	0,59
noAction			740	4,93

Figure 945: Failed actions.

52.3 Achievements

Figure 946: areaValueAchievements.

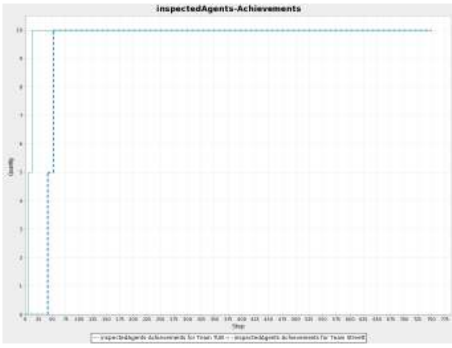


Figure 947: inspectedAgentsAchievements.

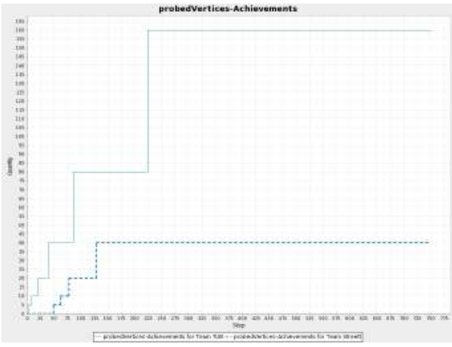


Figure 948: probedVerticesAchievements.

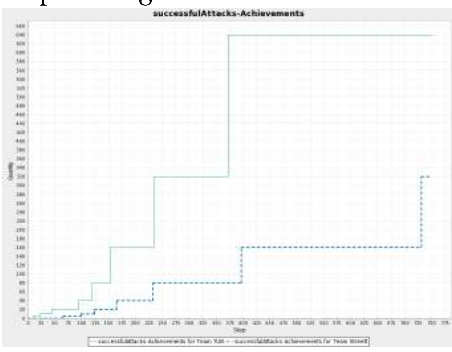


Figure 949: successfulAttacksAchievements.

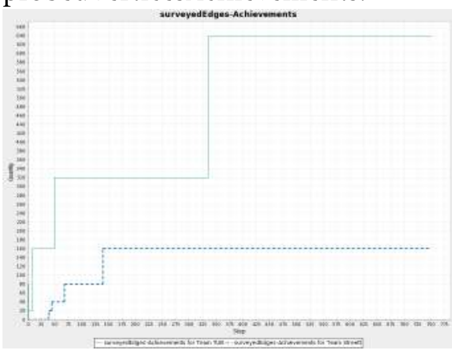


Figure 950: surveyedEdgesAchievements.

52.4 Actions per Role

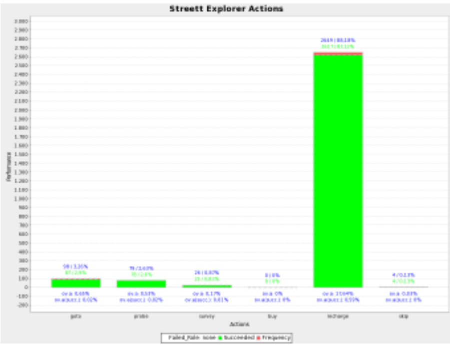


Figure 951: Streett vs. TUB – Simulation 1 - Streett Explorer Actions.

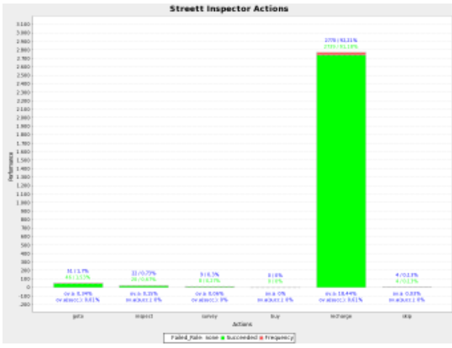


Figure 952: Streett vs. TUB – Simulation 1 - Streett Inspector Actions.

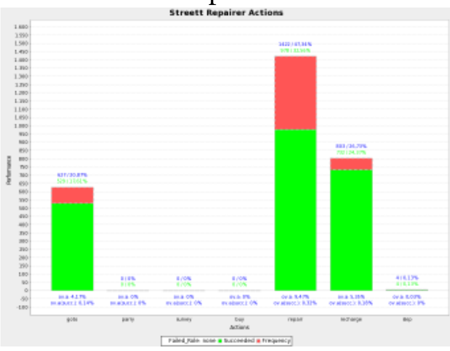


Figure 953: Streett vs. TUB – Simulation 1 - Streett Repairer Actions.

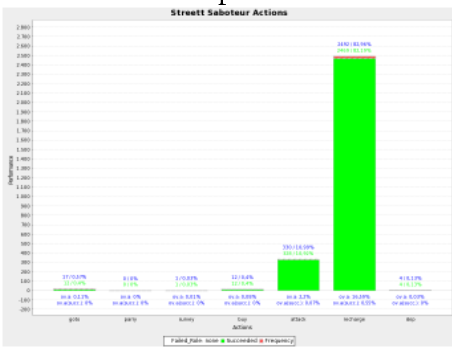


Figure 954: Streett vs. TUB – Simulation 1 - Streett Saboteur Actions.

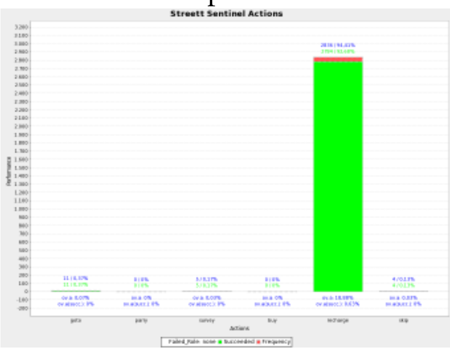


Figure 955: Streett vs. TUB – Simulation 1 - Streett Sentinel Actions.

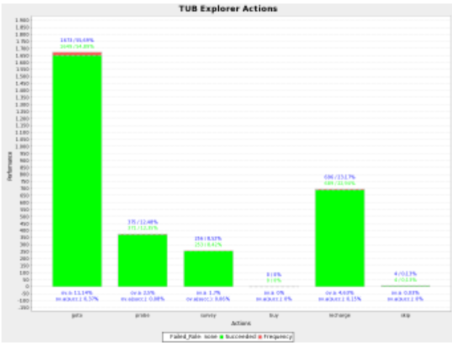


Figure 956: Streett vs. TUB – Simulation 1 - TUB Explorer Actions.

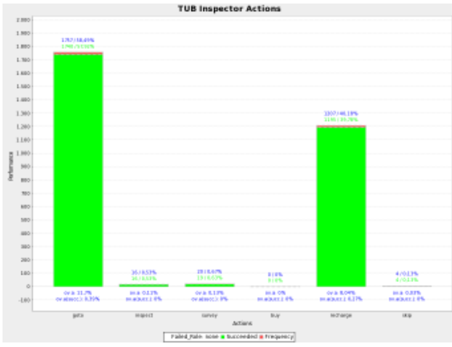


Figure 957: Streett vs. TUB – Simulation 1 - TUB Inspector Actions.

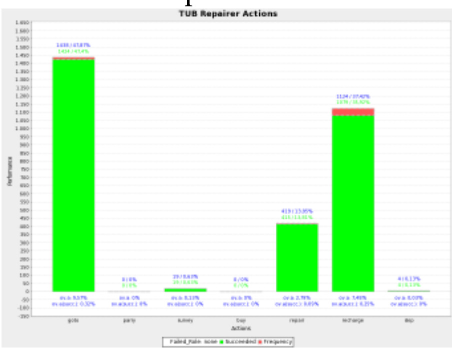


Figure 958: Streett vs. TUB – Simulation 1 - TUB Repairer Actions.

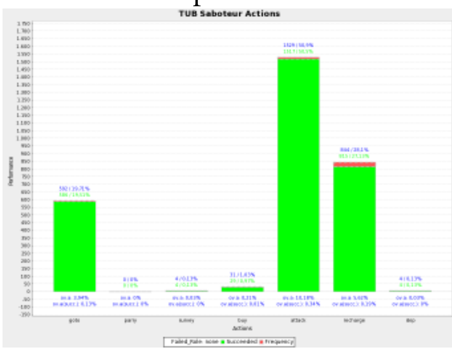


Figure 959: Streett vs. TUB – Simulation 1 - TUB Saboteur Actions.

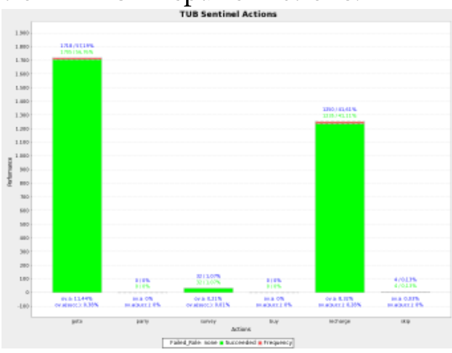


Figure 960: Streett vs. TUB – Simulation 1 - TUB Sentinel Actions.

53 Streett vs. TUB – Simulation 2

53.1 Scores, Zone Stability and Achievements

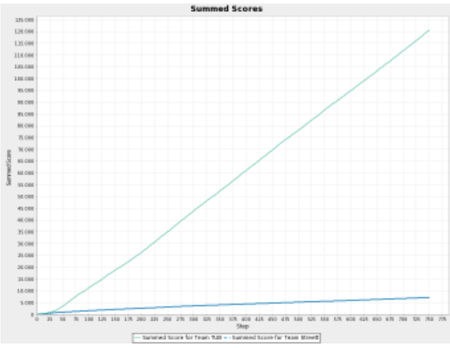


Figure 961: Summed scores.

Figure 962: Achievement points.

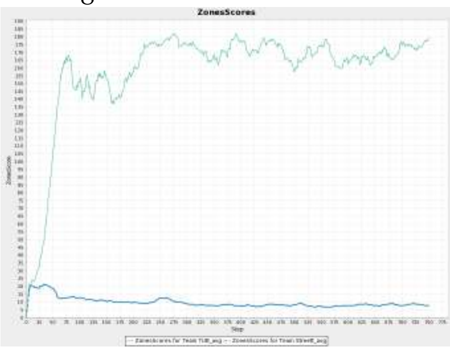


Figure 963: Zones scores.

Figure 964: Zones scores and achievement points.

Figure 965: Zone Stabilities.

Step	TUB	Streett
1	surveyed40, surveyed10, surveyed20	surveyed10, surveyed20
2	area10, surveyed80	surveyed40
3	proved5	area10, proved5
5		inspected5
6	inspected5	proved10
9	proved10, attacked5	area20
10		surveyed80, attacked5
11	inspected10, area20	
12		proved20
14	attacked10	
15	surveyed160	
18	area40, proved20	
32		proved40, surveyed160
34	area80	
44	proved40	
46	attacked20	
53	area160	
110	proved80	
130	surveyed320	
249		surveyed320

Figure 966: Achievements.

53.2 Stability

Reason	TUB	%	Streett	%
failed away			623	4,15
failed random	138	0,92	141	0,94
failed resources	1	0,01	331	2,21
failed attacked	2	0,01	13	0,09

Figure 967: Failed actions.

53.3 Achievements

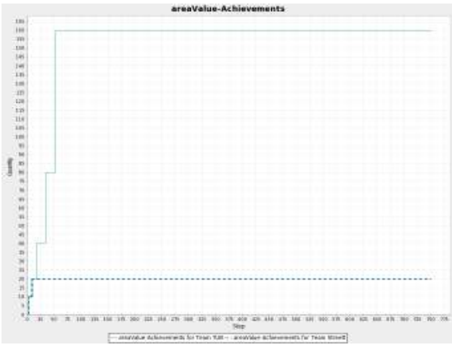


Figure 968: areaValueAchievements.

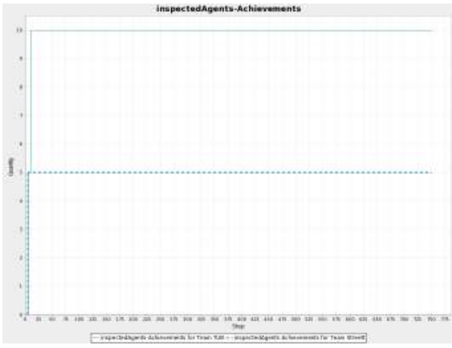


Figure 969: inspectedAgentsAchievements.

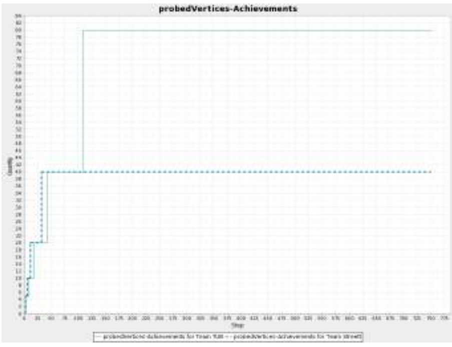


Figure 970: probedVerticesAchievements.

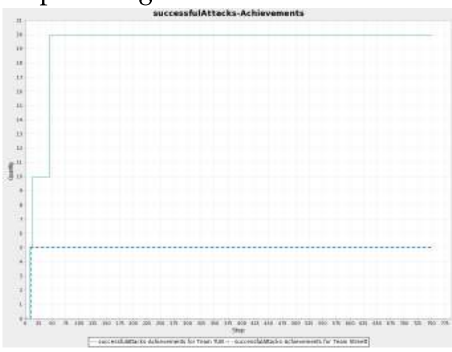


Figure 971: successfulAttacksAchievements.

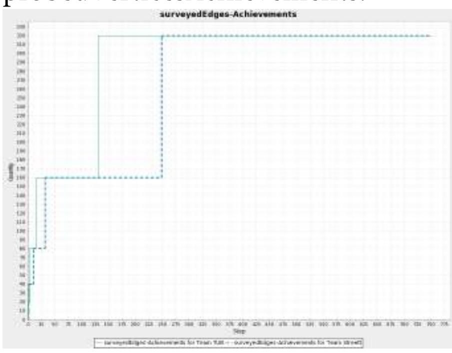


Figure 972: surveyedEdgesAchievements.

53.4 Actions per Role

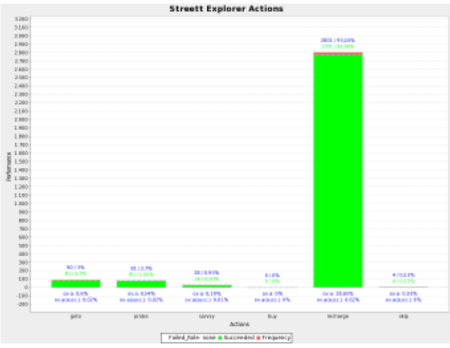


Figure 973: Streett vs. TUB – Simulation 2 - Streett Explorer Actions.

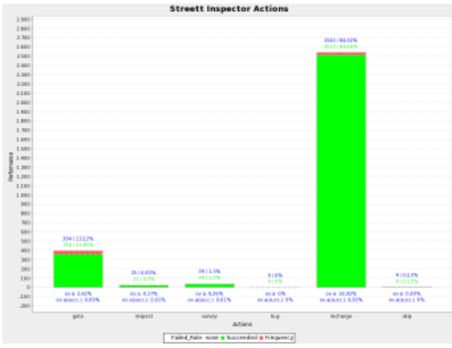


Figure 974: Streett vs. TUB – Simulation 2 - Streett Inspector Actions.

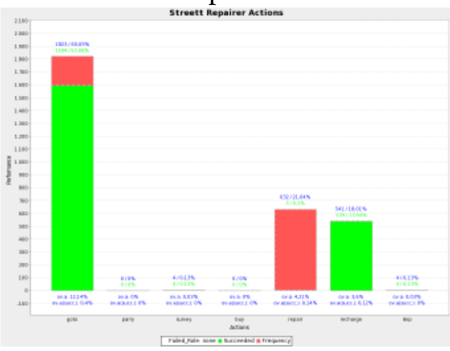


Figure 975: Streett vs. TUB – Simulation 2 - Streett Repairer Actions.

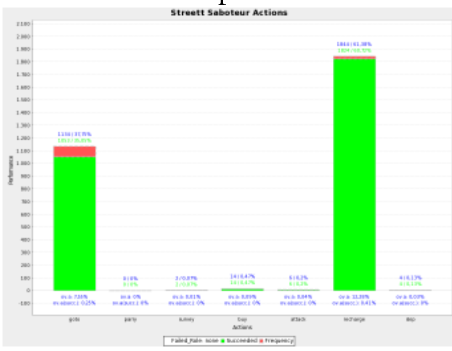


Figure 976: Streett vs. TUB – Simulation 2 - Streett Saboteur Actions.

Figure 977: Streett vs. TUB – Simulation 2 - Streett Sentinel Actions.

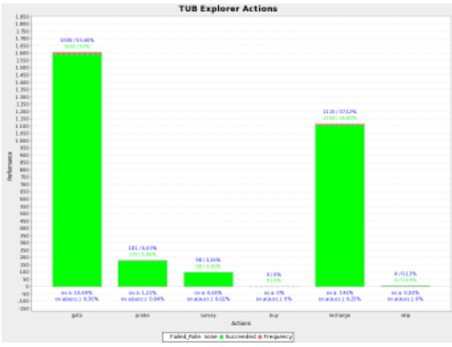


Figure 978: Streett vs. TUB – Simulation 2 - TUB Explorer Actions.

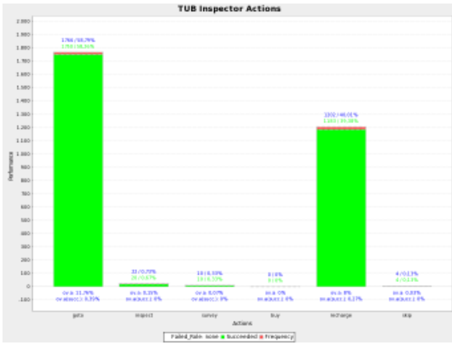


Figure 979: Streett vs. TUB – Simulation 2 - TUB Inspector Actions.

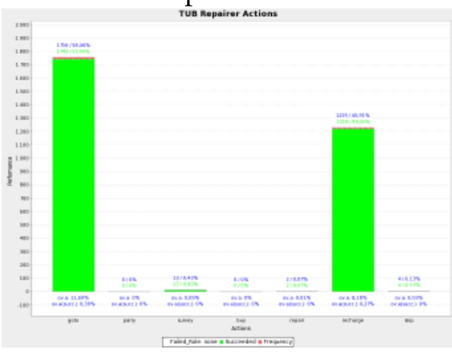


Figure 980: Streett vs. TUB – Simulation 2 - TUB Repairer Actions.

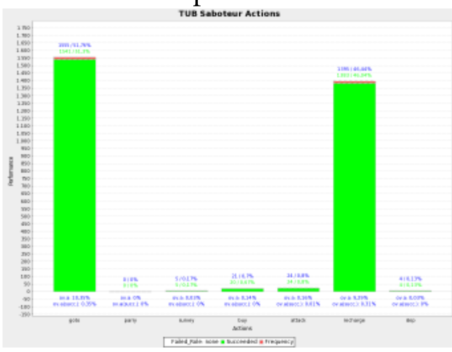


Figure 981: Streett vs. TUB – Simulation 2 - TUB Saboteur Actions.

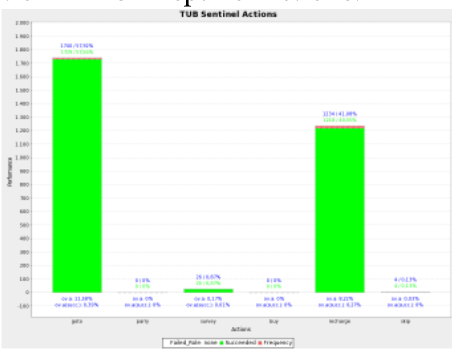


Figure 982: Streett vs. TUB – Simulation 2 - TUB Sentinel Actions.

54 Streett vs. TUB – Simulation 3

54.1 Scores, Zone Stability and Achievements

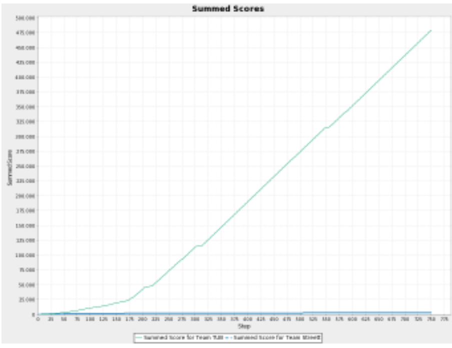


Figure 983: Summed scores.

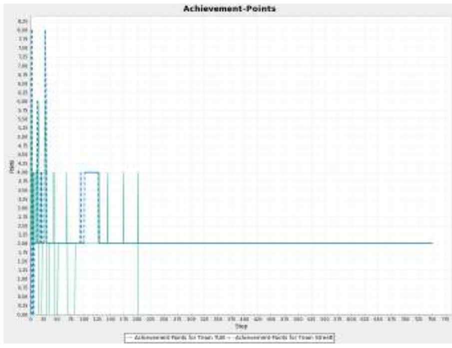


Figure 984: Achievement points.

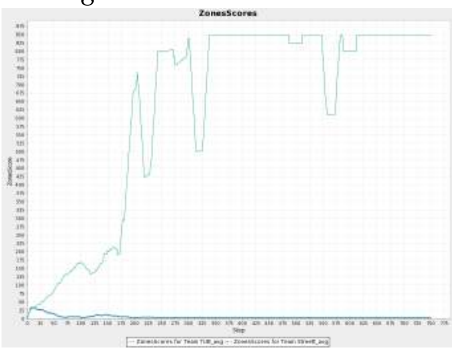


Figure 985: Zones scores.

Figure 986: Zones scores and achievement points.

Figure 987: Zone Stabilities.

Step	TUB	Streett
1	surveyed10, surveyed40, surveyed20	surveyed10, surveyed20, area10, inspected5
2	area10	surveyed40, surveyed80
3	surveyed80	
4	area20, proved5	area20, proved5
6		proved10
8	proved10	
9	attacked5, inspected5	
10	surveyed160	
11		proved20
12		attacked5
13	inspected10	
14	area40, attacked10	
18		attacked10
20		surveyed160
21	proved20	
26		attacked20
27	attacked20, inspected20	inspected10, proved40
34	area80	
42	surveyed320	
43	proved40	
51	area160	
67	attacked40	
83	proved80	
93		proved80
100		attacked40
126	attacked80	
143	area320	
173	area640	
200	proved160	

Figure 988: Achievements.

54.2 Stability

Reason	TUB	%	Streett	%
failed away	1	0,01	20	0,13
failed wrong param	16	0,11		
failed random	176	1,17	152	1,01
failed	13	0,09		
failed resources	5	0,03	265	1,77
failed attacked	21	0,14	31	0,21
noAction	13	0,09		

Figure 989: Failed actions.

54.3 Achievements

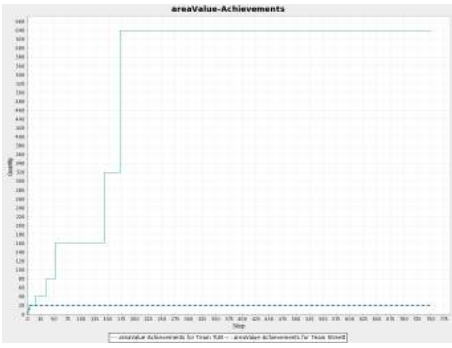


Figure 990: areaValueAchievements.

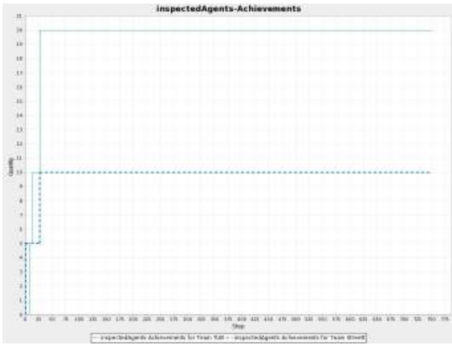


Figure 991: inspectedAgentsAchievements.

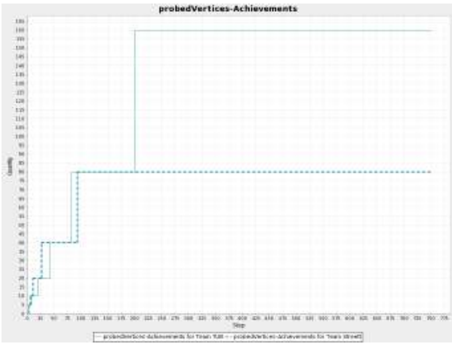


Figure 992: probedVerticesAchievements.

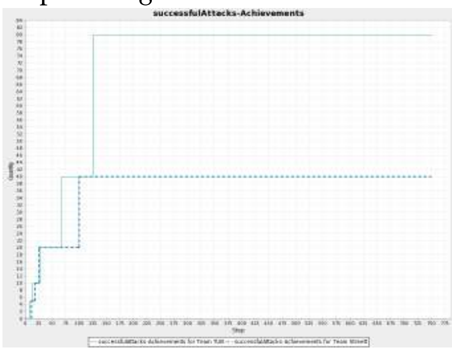


Figure 993: successfulAttacksAchievements.

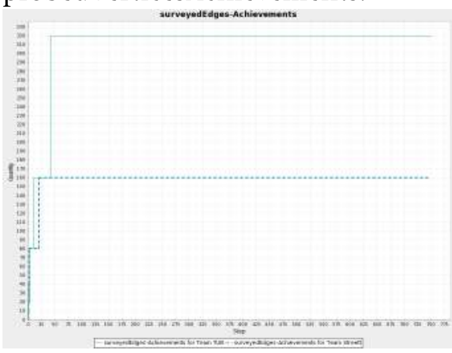


Figure 994: surveyedEdgesAchievements.

54.4 Actions per Role

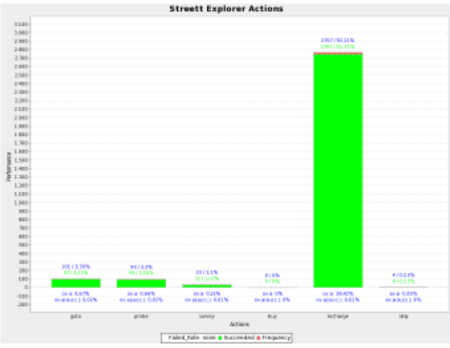


Figure 995: Streett vs. TUB – Simulation 3 - Streett Explorer Actions.

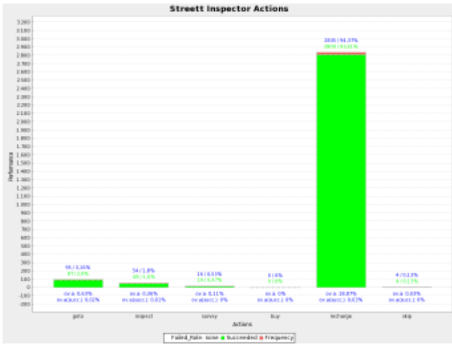


Figure 996: Streett vs. TUB – Simulation 3 - Streett Inspector Actions.

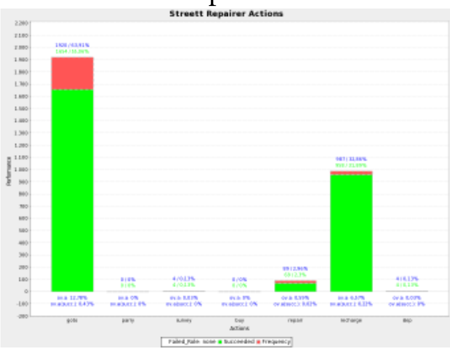


Figure 997: Streett vs. TUB – Simulation 3 - Streett Repairer Actions.

Figure 998: Streett vs. TUB – Simulation 3 - Streett Saboteur Actions.

Figure 999: Streett vs. TUB – Simulation 3 - Streett Sentinel Actions.

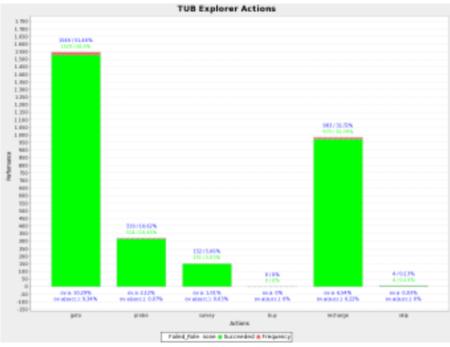


Figure 1000: Streett vs. TUB – Simulation 3 - TUB Explorer Actions.

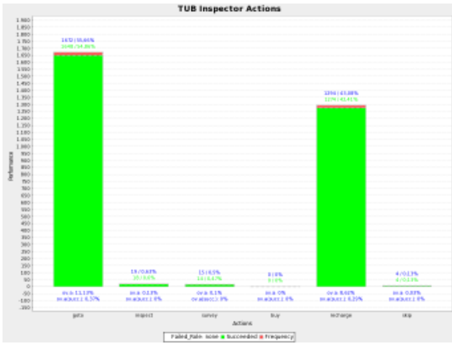


Figure 1001: Streett vs. TUB – Simulation 3 - TUB Inspector Actions.

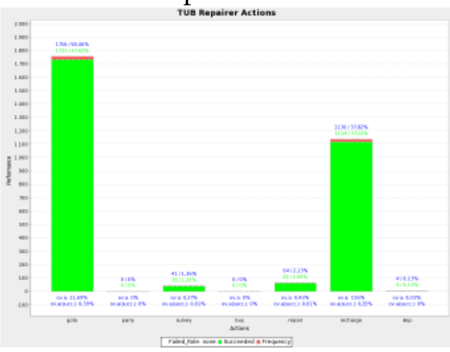


Figure 1002: Streett vs. TUB – Simulation 3 - TUB Repairer Actions.

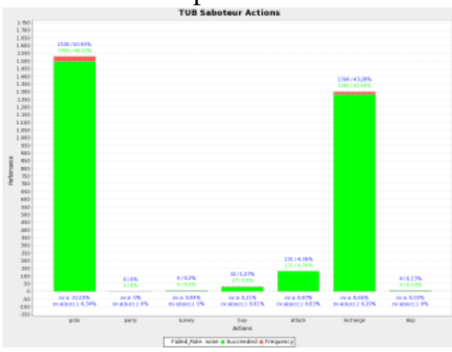


Figure 1003: Streett vs. TUB – Simulation 3 - TUB Saboteur Actions.

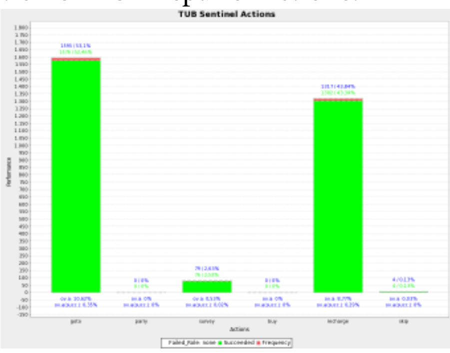


Figure 1004: Streett vs. TUB – Simulation 3 - TUB Sentinel Actions.

55 Streett vs. UFSC – Simulation 1

55.1 Scores, Zone Stability and Achievements

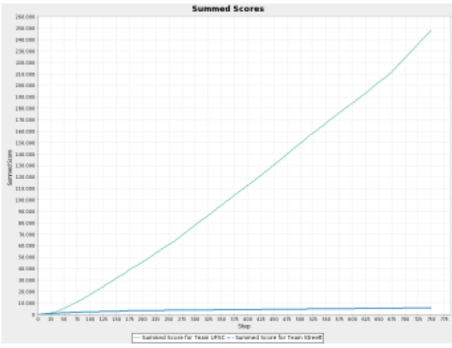


Figure 1005: Summed scores.

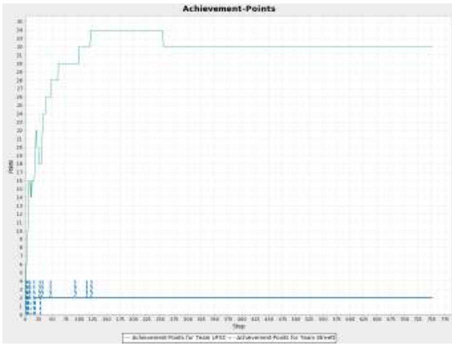


Figure 1006: Achievement points.

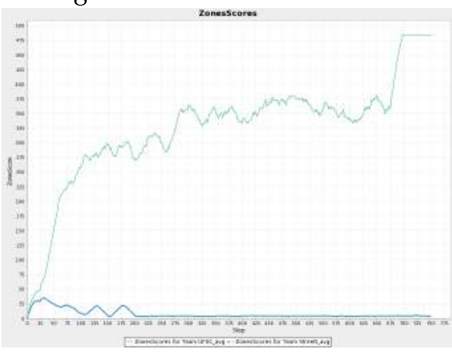


Figure 1007: Zones scores.

Figure 1008: Zones scores and achievement points.

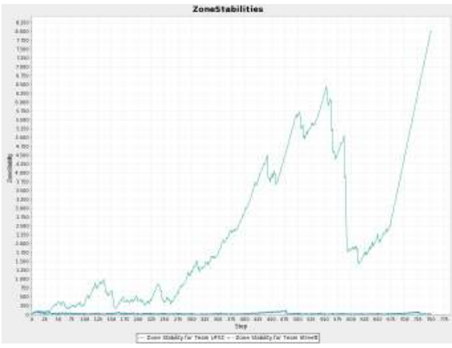


Figure 1009: Zone Stabilities.

Step	UFSC	Streett
1	surveyed40, surveyed10, surveyed20	surveyed10, surveyed20
2		surveyed40, surveyed80
3	area10, proved5, surveyed80	
4		area10, proved5
5	area20, proved10, attacked5	
6		proved10
7	surveyed160	
8		area20
10	proved20	
12	attacked10	
16		area40, proved20
17	surveyed320	
18	area40	attacked5
19	inspected5	
23	proved40	
26		surveyed160
28		attacked10
30	inspected10	
31	attacked20	inspected5
32	area80	
37	area160	
46		proved40
47	proved80	
61	attacked40	
91		inspected10
99	proved160	
113		surveyed320
120	area320	
121		proved80

Figure 1010: Achievements.

55.2 Stability

Reason	UFSC	%	Streett	%
failed away			2	0,01
failed parried			2	0,01
failed random	145	0,97	128	0,85
failed resources			247	1,65
failed attacked	3	0,02	21	0,14

Figure 1011: Failed actions.

55.3 Achievements

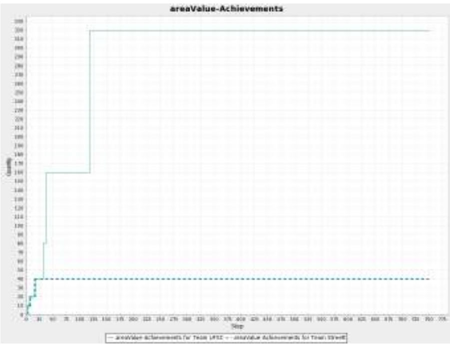


Figure 1012: areaValueAchievements.

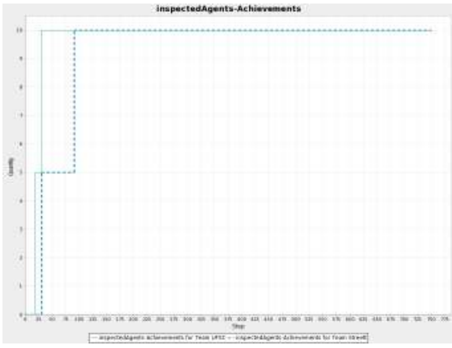


Figure 1013: inspectedAgentsAchievements.

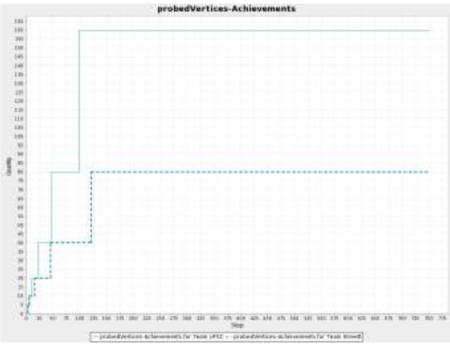


Figure 1014: probedVerticesAchievements.

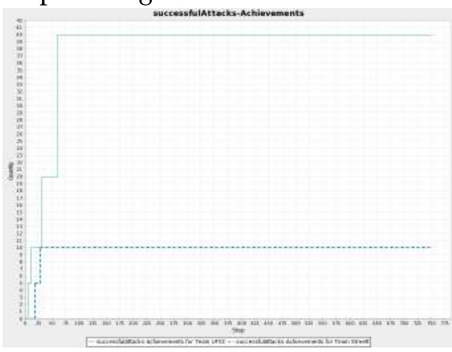


Figure 1015: successfulAttacksAchievements.

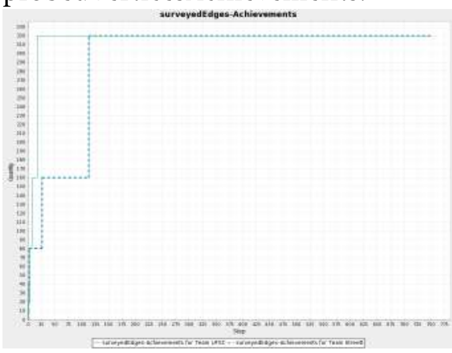


Figure 1016: surveyedEdgesAchievements.

55.4 Actions per Role

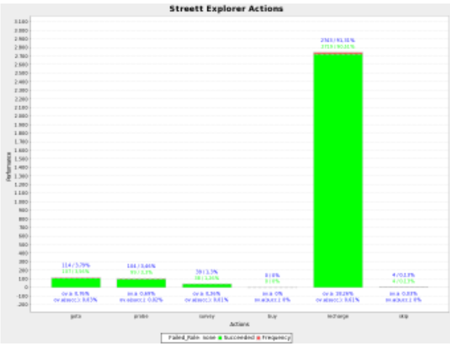


Figure 1017: Streett vs. UFSC – Simulation 1 - Streett Explorer Actions.

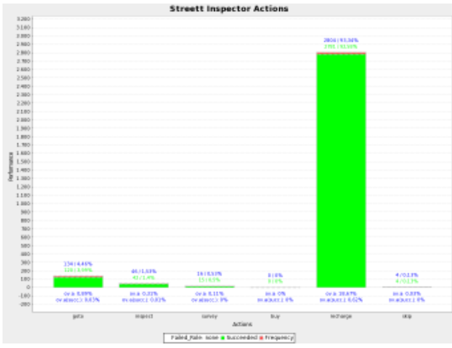


Figure 1018: Streett vs. UFSC – Simulation 1 - Streett Inspector Actions.

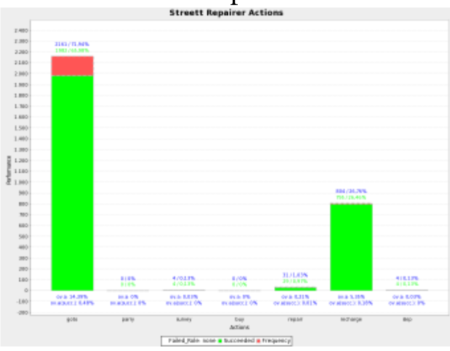


Figure 1019: Streett vs. UFSC – Simulation 1 - Streett Repairer Actions.

Figure 1020: Streett vs. UFSC – Simulation 1 - Streett Saboteur Actions.

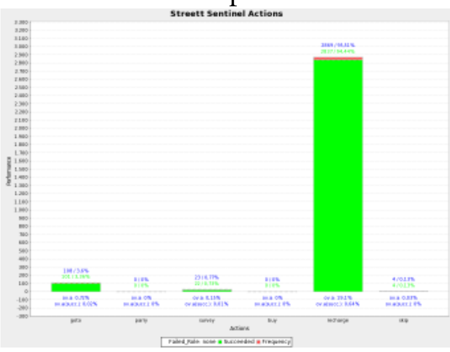


Figure 1021: Streett vs. UFSC – Simulation 1 - Streett Sentinel Actions.

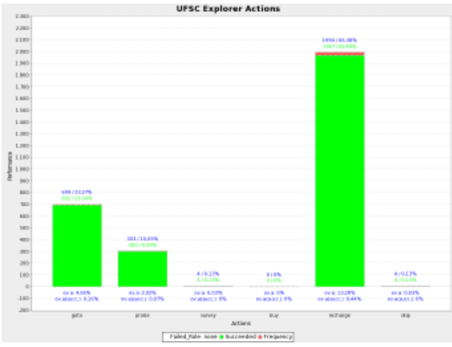


Figure 1022: Streett vs. UFSC – Simulation 1 - UFSC Explorer Actions.

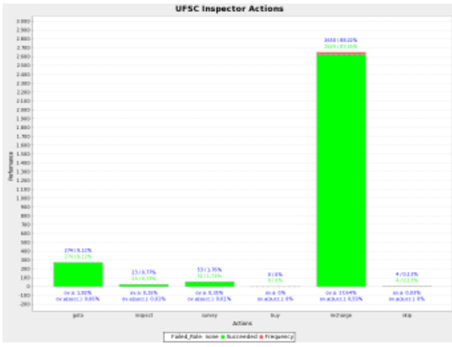


Figure 1023: Streett vs. UFSC – Simulation 1 - UFSC Inspector Actions.

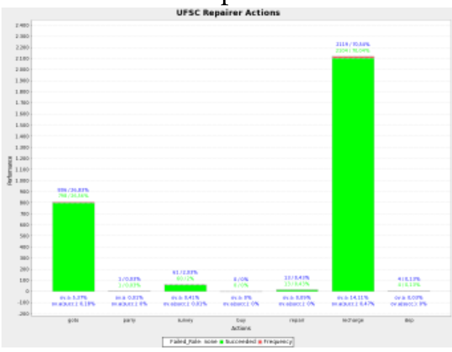


Figure 1024: Streett vs. UFSC – Simulation 1 - UFSC Repairer Actions.

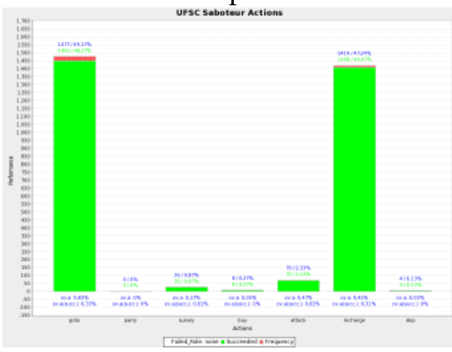


Figure 1025: Streett vs. UFSC – Simulation 1 - UFSC Saboteur Actions.

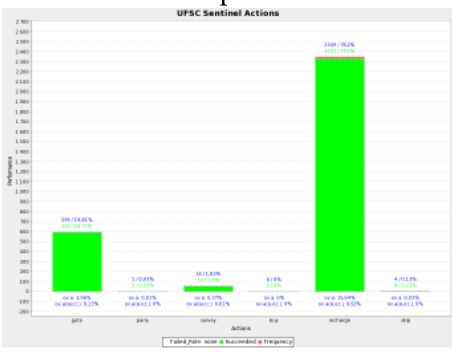


Figure 1026: Streett vs. UFSC – Simulation 1 - UFSC Sentinel Actions.

56 Streett vs. UFSC – Simulation 2

56.1 Scores, Zone Stability and Achievements

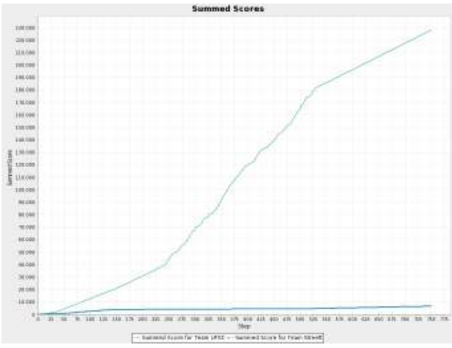


Figure 1027: Summed scores.

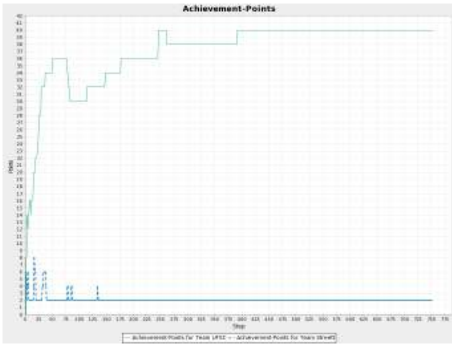


Figure 1028: Achievement points.

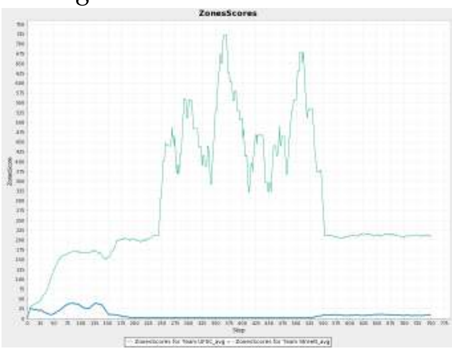


Figure 1029: Zones scores.

Figure 1030: Zones scores and achievement points.

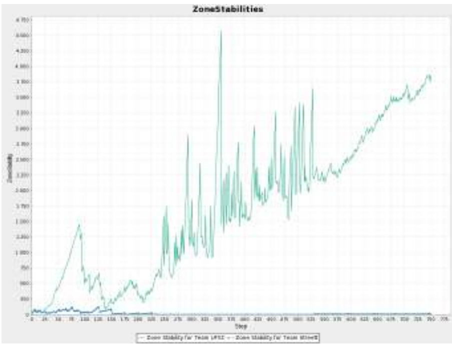


Figure 1031: Zone Stabilities.

Step	UFSC	Streett
1	surveyed10, surveyed40, area10, surveyed20	surveyed40, surveyed10, surveyed20
3	surveyed80, attacked5, proved5	area10
4		proved5, attacked5
5	proved10	area20, surveyed80
6	area20	proved10
7	surveyed160	
10	inspected5	
11	proved20	
14	parried5	
15	attacked10	proved20
16		attacked10, inspected5
18	parried10	surveyed160
22	surveyed320	
24	proved40	
25	inspected10	
29	area40, attacked20	
31		inspected10
33		proved40
36	area80	
50	proved80	
77		area40
84		proved80
113	proved160	
133		attacked20
147	area160	
176	attacked40	
245	area320, area640	
391	attacked80	

Figure 1032: Achievements.

56.2 Stability

Reason	UFSC	%	Streett	%
failed away			3	0,02
failed parried			17	0,11
failed random	156	1,04	140	0,93
failed resources			248	1,65
failed attacked	3	0,02	30	0,2

Figure 1033: Failed actions.

56.3 Achievements

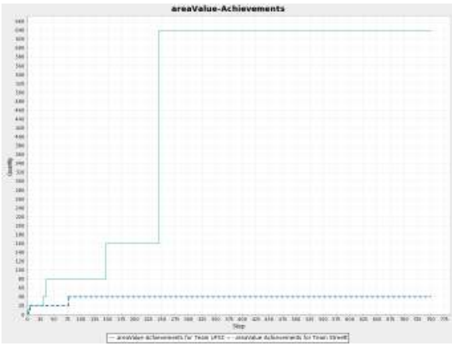


Figure 1034: areaValueAchievements.

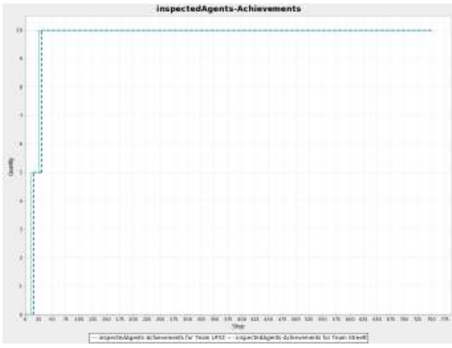


Figure 1035: inspectedAgentsAchievements.

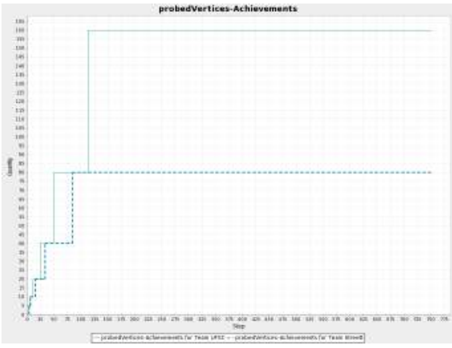


Figure 1036: probedVerticesAchievements.

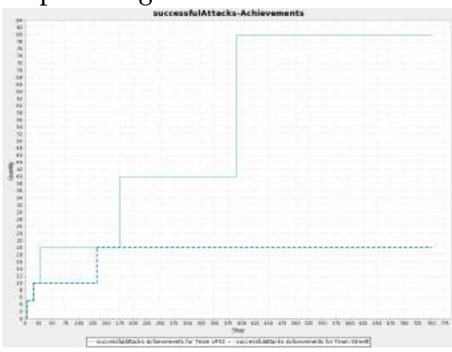


Figure 1037: successfulAttacksAchievements.

Figure 1038: successfulParriesAchievements.

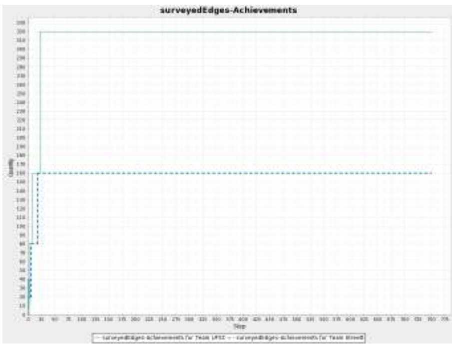


Figure 1039: surveyedEdgesAchievements.

56.4 Actions per Role

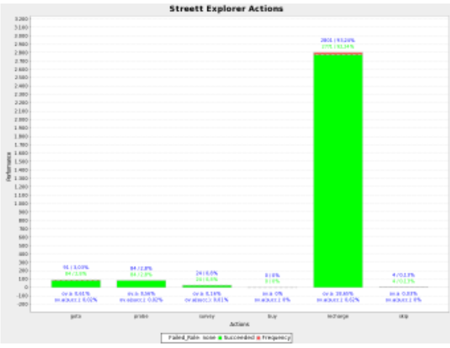


Figure 1040: Streett vs. UFSC – Simulation 2 - Streett Explorer Actions.

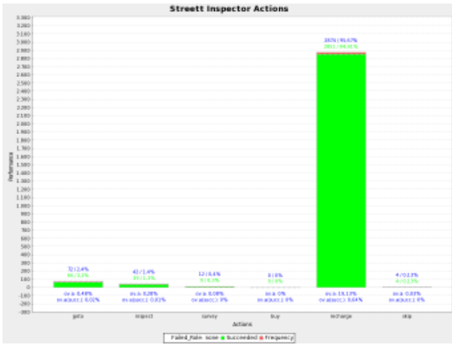


Figure 1041: Streett vs. UFSC – Simulation 2 - Streett Inspector Actions.

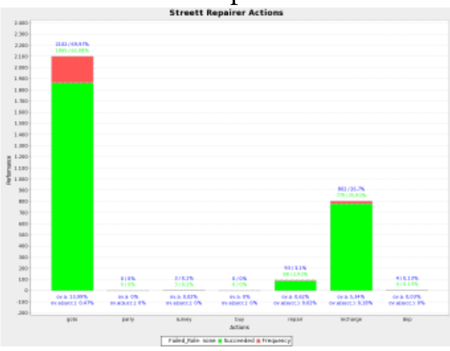


Figure 1042: Streett vs. UFSC – Simulation 2 - Streett Repairer Actions.

Figure 1043: Streett vs. UFSC – Simulation 2 - Streett Saboteur Actions.

Figure 1044: Streett vs. UFSC – Simulation 2 - Streett Sentinel Actions.

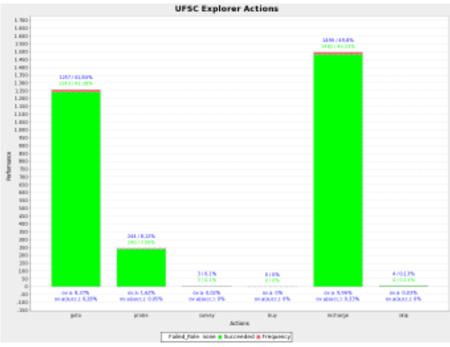


Figure 1045: Streett vs. UFSC – Simulation 2 - UFSC Explorer Actions.

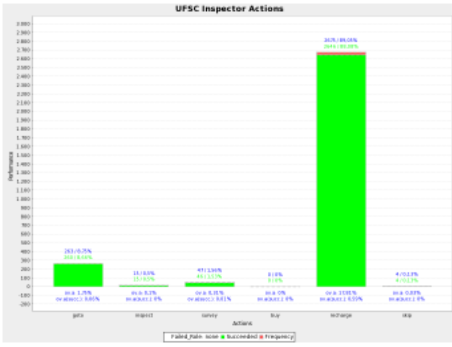


Figure 1046: Streett vs. UFSC – Simulation 2 - UFSC Inspector Actions.

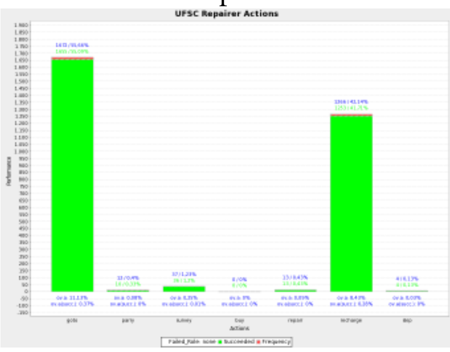


Figure 1047: Streett vs. UFSC – Simulation 2 - UFSC Repairer Actions.

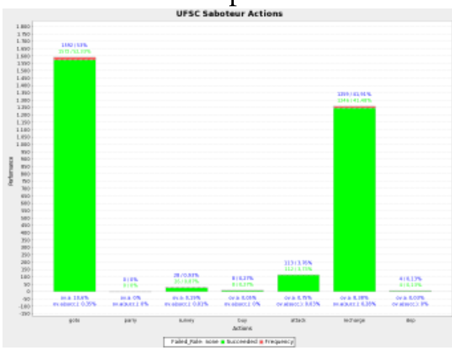


Figure 1048: Streett vs. UFSC – Simulation 2 - UFSC Saboteur Actions.

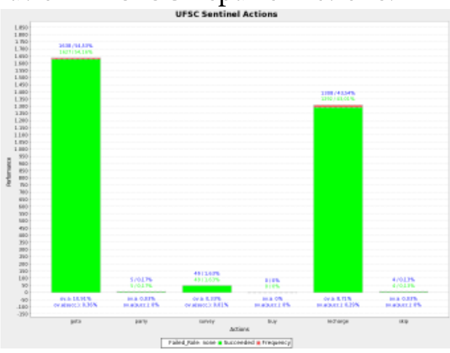


Figure 1049: Streett vs. UFSC – Simulation 2 - UFSC Sentinel Actions.

57 Streett vs. UFSC – Simulation 3

57.1 Scores, Zone Stability and Achievements

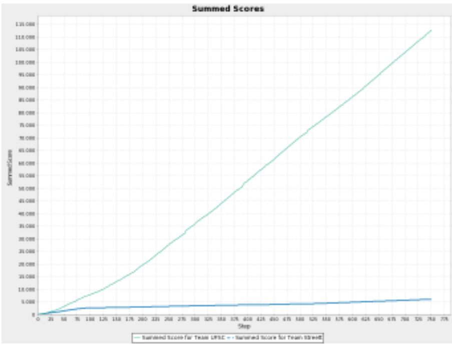


Figure 1050: Summed scores.

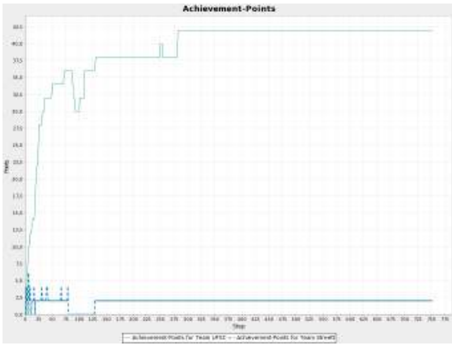


Figure 1051: Achievement points.

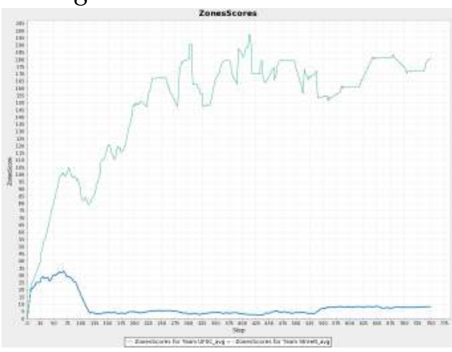


Figure 1052: Zones scores.

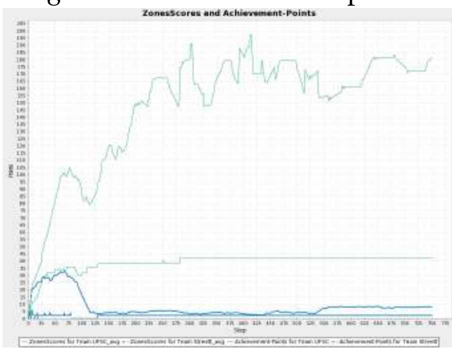


Figure 1053: Zones scores and achievement points.

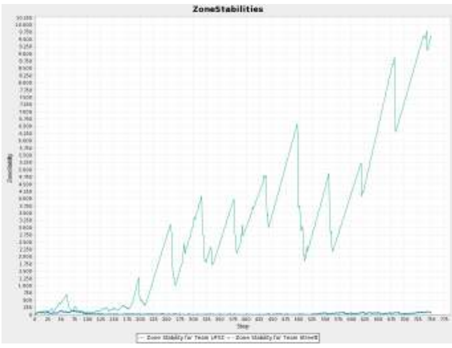


Figure 1054: Zone Stabilities.

Step	UFSC	Streett
1		surveyed10, surveyed20
2	surveyed10, surveyed40, surveyed20	surveyed40
3		proved5
4	surveyed80, proved5	
5	area10	area10, surveyed80, inspected5
6	proved10	proved10
8	surveyed160, inspected5	area20
10	proved20	
12	attacked5	proved20
16		surveyed160
17	inspected10	
18	attacked10	attacked5
19	area20	
21	surveyed320	
23	proved40	
24	parried5	
25	area40	
30		inspected10
31	parried10	
35	attacked20	
39		proved40
49	proved80	
66		attacked10
71	attacked40	
78		proved80
100	parried20	
109	proved160, attacked80	
128		surveyed320
129	area80	
249	attacked160	
281	area160, area320	

Figure 1055: Achievements.

57.2 Stability

Reason	UFSC	%	Streett	%
failed away			7	0,05
failed parried			21	0,14
failed random	146	0,97	150	1
failed resources			291	1,94
failed	19	0,13	1	0,01
failed attacked	5	0,03	83	0,55
noAction	19	0,13	1	0,01

Figure 1056: Failed actions.

57.3 Achievements

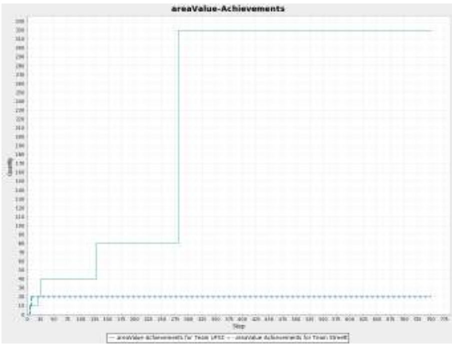


Figure 1057: areaValueAchievements.

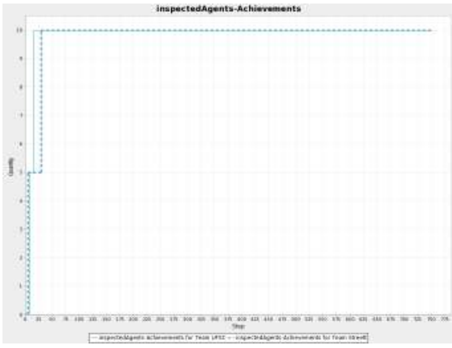


Figure 1058: inspectedAgentsAchievements.

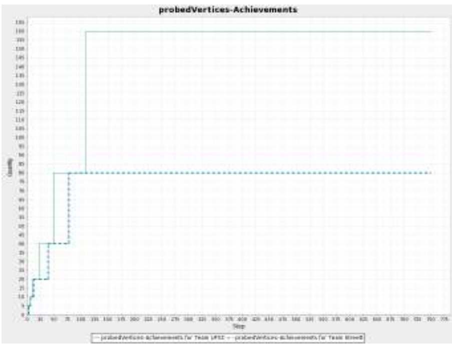


Figure 1059: probedVerticesAchievements.

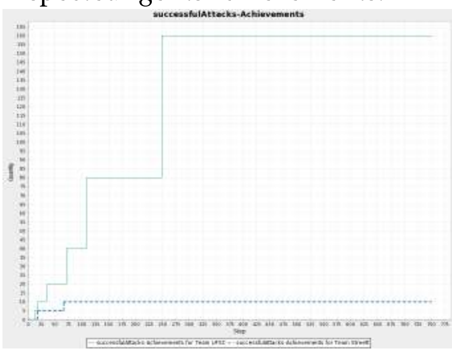


Figure 1060: successfulAttacksAchievements.

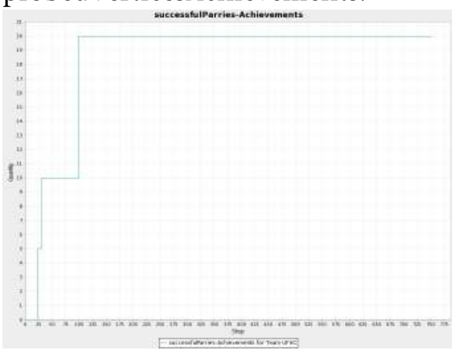


Figure 1061: successfulParriesAchievements.

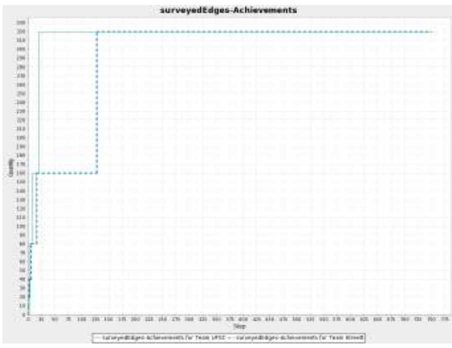


Figure 1062: surveyedEdgesAchievements.

57.4 Actions per Role

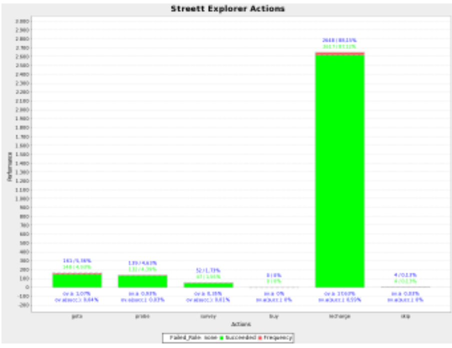


Figure 1063: Streett vs. UFSC – Simulation 3 - Streett Explorer Actions.

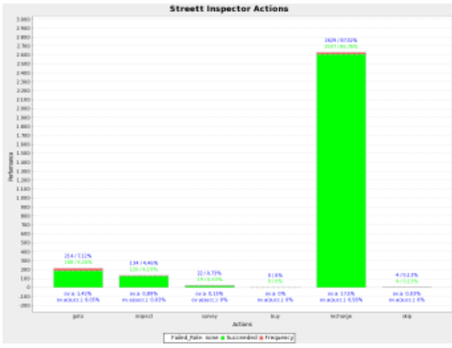


Figure 1064: Streett vs. UFSC – Simulation 3 - Streett Inspector Actions.

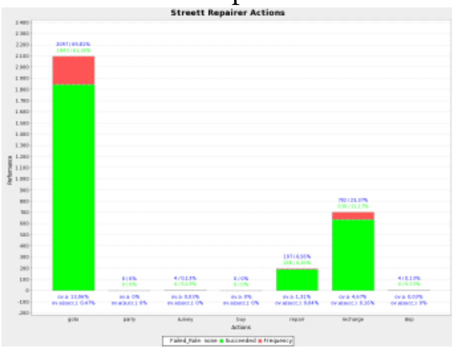


Figure 1065: Streett vs. UFSC – Simulation 3 - Streett Repairer Actions.

Figure 1066: Streett vs. UFSC – Simulation 3 - Streett Saboteur Actions.

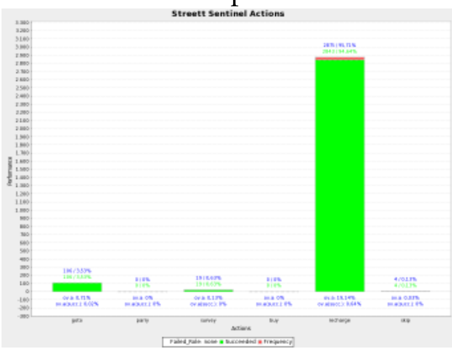


Figure 1067: Streett vs. UFSC – Simulation 3 - Streett Sentinel Actions.

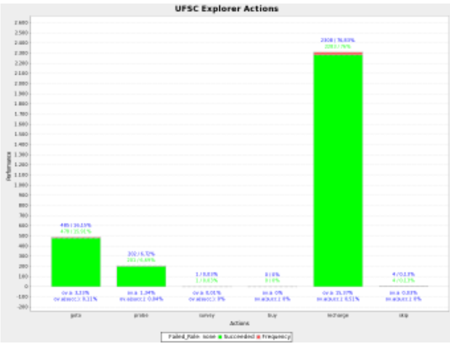


Figure 1068: Streett vs. UFSC – Simulation 3 - UFSC Explorer Actions.

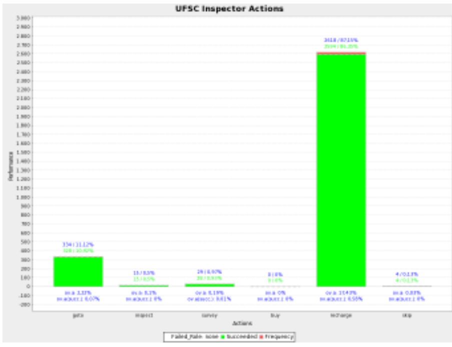


Figure 1069: Streett vs. UFSC – Simulation 3 - UFSC Inspector Actions.

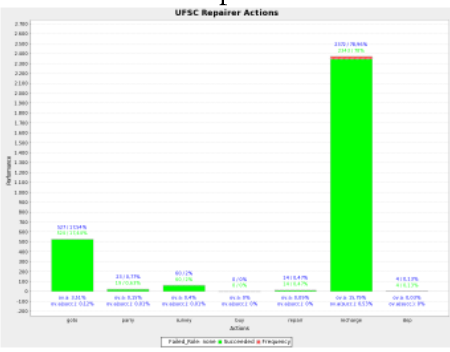


Figure 1070: Streett vs. UFSC – Simulation 3 - UFSC Repairer Actions.

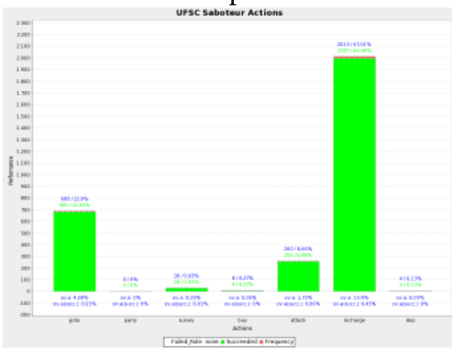


Figure 1071: Streett vs. UFSC – Simulation 3 - UFSC Saboteur Actions.

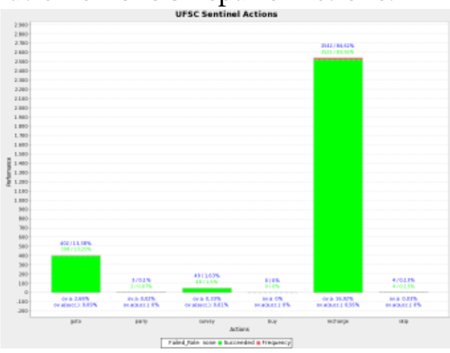


Figure 1072: Streett vs. UFSC – Simulation 3 - UFSC Sentinel Actions.

58 Streett vs. USP – Simulation 1

58.1 Scores, Zone Stability and Achievements

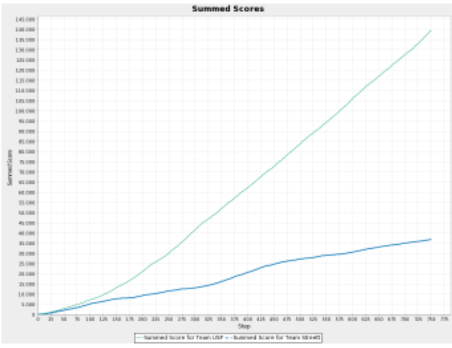


Figure 1073: Summed scores.

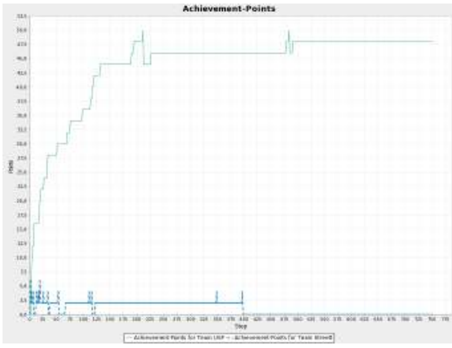


Figure 1074: Achievement points.

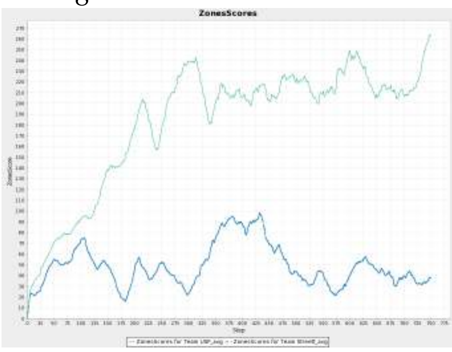


Figure 1075: Zones scores.

Figure 1076: Zones scores and achievement points.

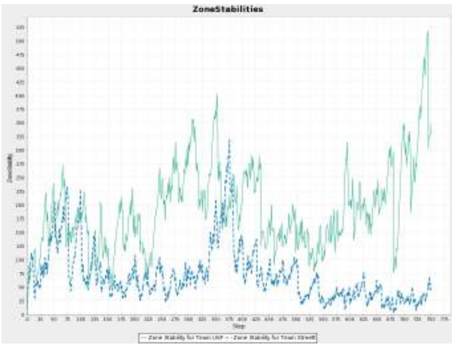


Figure 1077: Zone Stabilities.

Step	USP	Streett
1		surveyed10, area10, surveyed20
2	surveyed10, surveyed20	surveyed40
3	surveyed40, area10	proved5
4	area20, proved5	surveyed80
6		proved10
7	surveyed80	
8	proved10	
11		proved20
13		inspected5
16		attacked5
17	attacked5	
18	proved20	area20, attacked10
20	surveyed160	surveyed160
25		proved40
26	attacked10	area40
32	area40	
33	inspected5	inspected10
37		attacked20
51	attacked20	
52		proved80
53		surveyed320
66		attacked40
70	proved40	
75	attacked40	
98	parried5	
110		proved160
113	inspected10	
115		inspected20
116	surveyed320	
119	area80	
121		attacked80
131	attacked80	
189	proved80	
194	area160	
211	parried10	
226	parried20	
348		area80
396		surveyed640
478	parried40	
483	attacked160	
490	proved160	

Figure 1078: Achievements.

58.2 Stability

Reason	USP	%	Streett	%
failed away	39	0,26	16	0,11
failed parried			51	0,34
failed random	151	1,01	121	0,81
failed	151	1,01		
failed resources	52	0,35	576	3,84
failed attacked	49	0,33	74	0,49
noAction	152	1,01		
failed status			23	0,15

Figure 1079: Failed actions.

58.3 Achievements

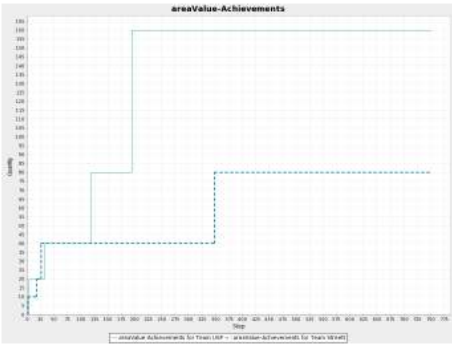


Figure 1080: areaValueAchievements.

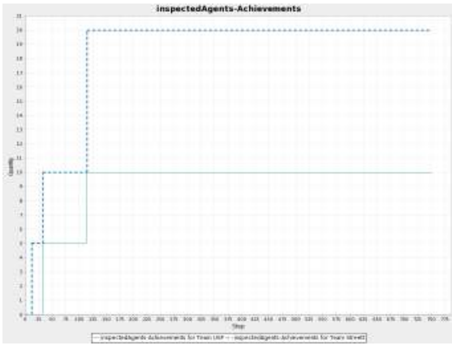


Figure 1081: inspectedAgentsAchievements.

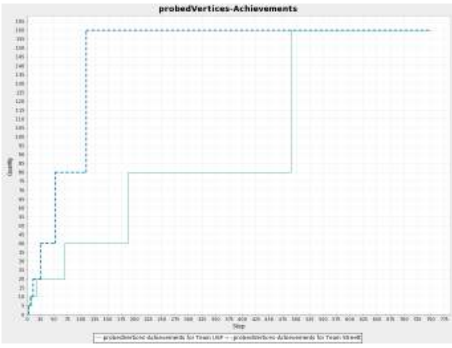


Figure 1082: probedVerticesAchievements.

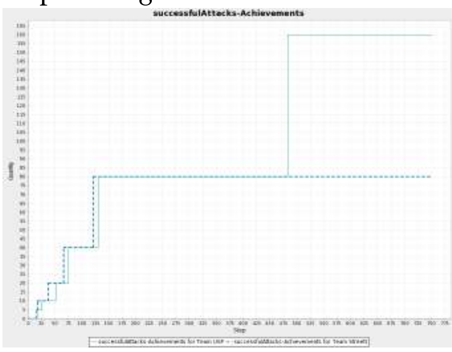


Figure 1083: successfulAttacksAchievements.

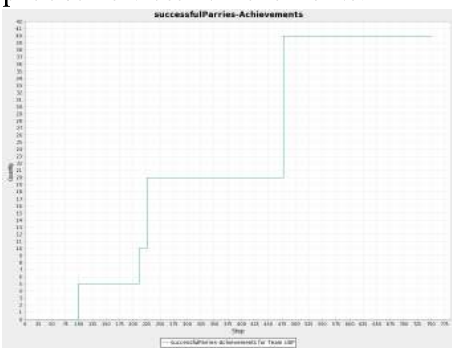


Figure 1084: successfulParriesAchievements.

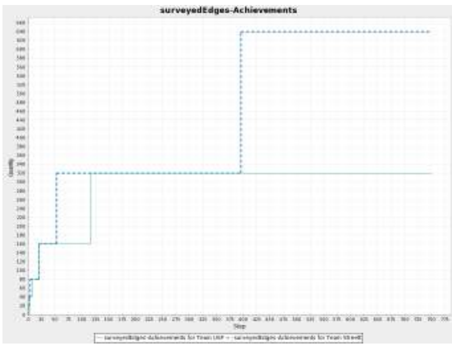


Figure 1085: surveyedEdgesAchievements.

58.4 Actions per Role

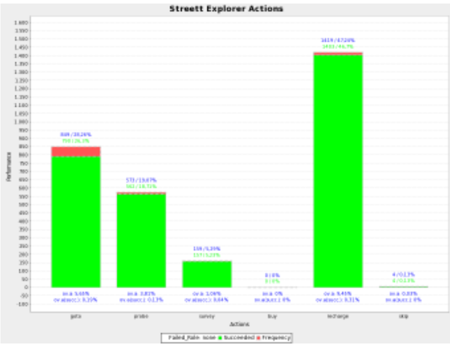


Figure 1086: Streett vs. USP – Simulation 1 - Streett Explorer Actions.

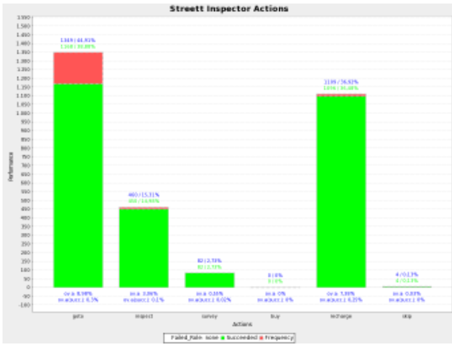


Figure 1087: Streett vs. USP – Simulation 1 - Streett Inspector Actions.

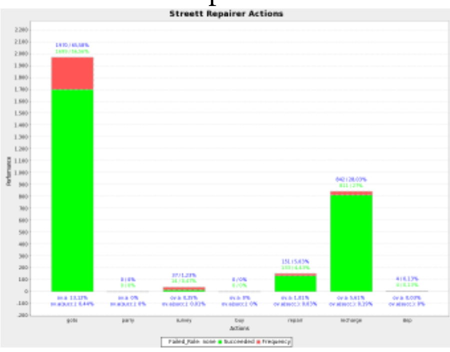


Figure 1088: Streett vs. USP – Simulation 1 - Streett Repairer Actions.

Figure 1089: Streett vs. USP – Simulation 1 - Streett Saboteur Actions.

Figure 1090: Streett vs. USP – Simulation 1 - Streett Sentinel Actions.

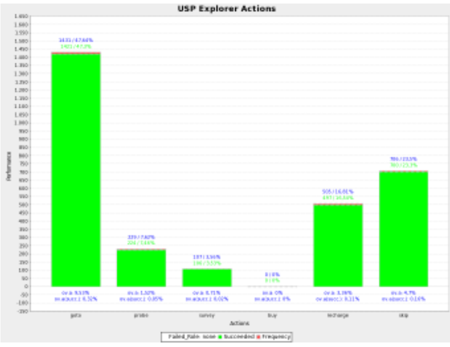


Figure 1091: Streett vs. USP – Simulation 1 - USP Explorer Actions.

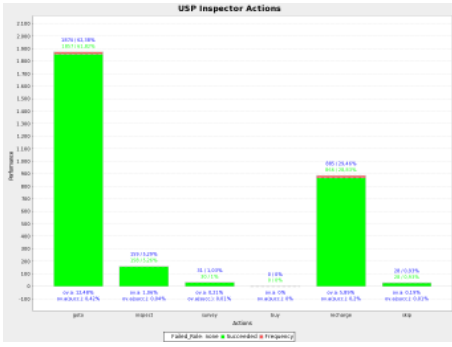


Figure 1092: Streett vs. USP – Simulation 1 - USP Inspector Actions.

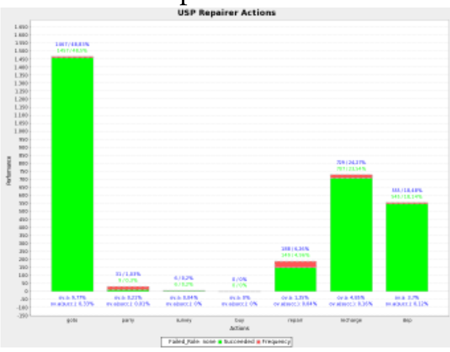


Figure 1093: Streett vs. USP – Simulation 1 - USP Repairer Actions.

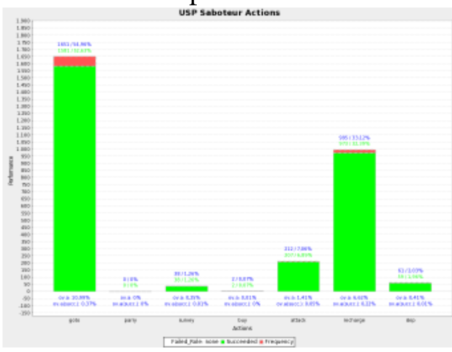


Figure 1094: Streett vs. USP – Simulation 1 - USP Saboteur Actions.

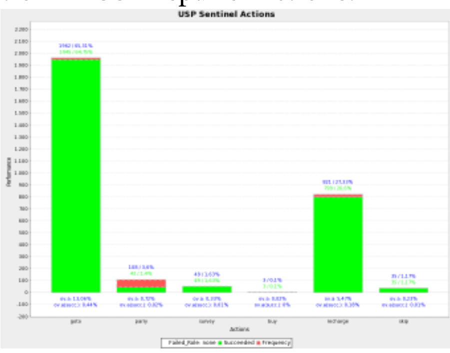


Figure 1095: Streett vs. USP – Simulation 1 - USP Sentinel Actions.

59 Streett vs. USP – Simulation 2

59.1 Scores, Zone Stability and Achievements

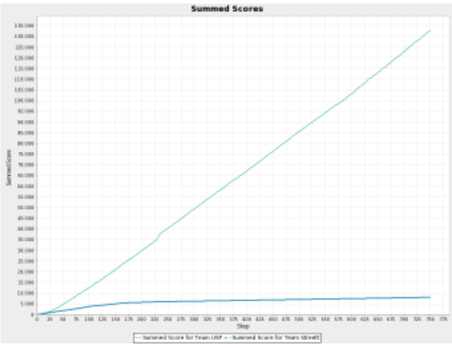


Figure 1096: Summed scores.

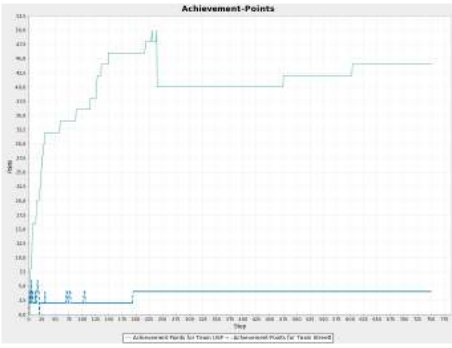


Figure 1097: Achievement points.

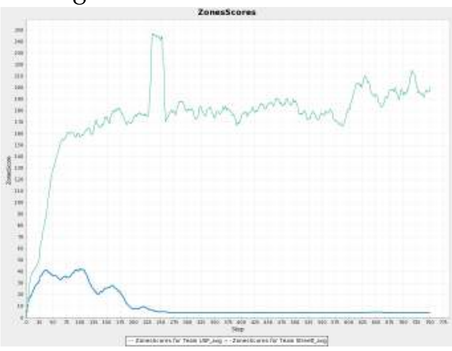


Figure 1098: Zones scores.

Figure 1099: Zones scores and achievement points.

Figure 1100: Zone Stabilities.

Step	USP	Streett
1		surveyed10, surveyed20
2	surveyed10, area10, surveyed20	surveyed40
3	surveyed40	attacked5
4		area10, proved5
5	attacked5	
6	area20, proved5	proved10
7	surveyed80	surveyed80
8		area20
12		inspected5
14	proved10, inspected5	
15		proved20
16		area40
17		inspected10
18		surveyed160
19	area40	
20		attacked10
22	surveyed160	
23	proved20	
24	attacked10	
27	area80	
30	inspected10	proved40
58	proved40	
70		proved80
75		surveyed320
88	attacked20	
103		attacked20
114	parried5	
126	area160	
127	attacked40	
135	surveyed320	
148	parried10	
194		proved160
217	attacked80	
229	area320	
238	proved80	
474	attacked160	
603	proved160	

Figure 1101: Achievements.

59.2 Stability

Reason	USP	%	Streett	%
failed away	4	0,03		
failed parried			10	0,07
failed random	148	0,99	142	0,95
failed			13	0,09
failed resources	70	0,47	344	2,29
failed attacked	11	0,07	73	0,49
noAction			13	0,09

Figure 1102: Failed actions.

59.3 Achievements

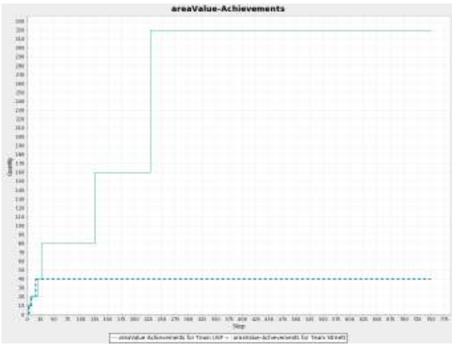


Figure 1103: areaValueAchievements.

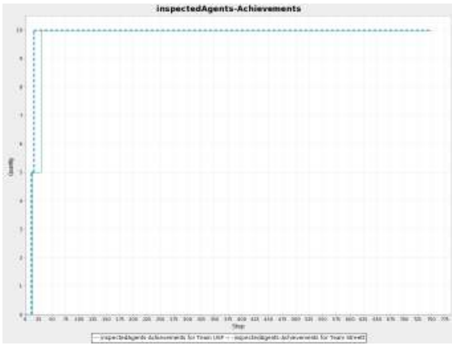


Figure 1104: inspectedAgentsAchievements.

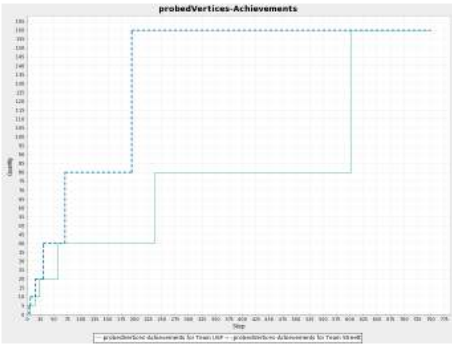


Figure 1105: probedVerticesAchievements.

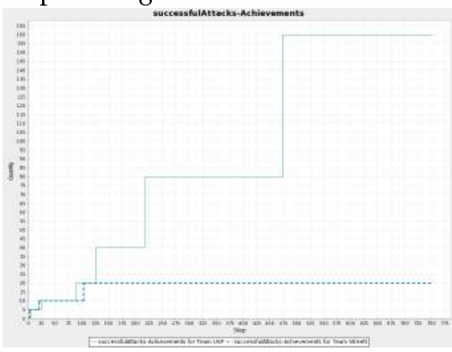


Figure 1106: successfulAttacksAchievements.

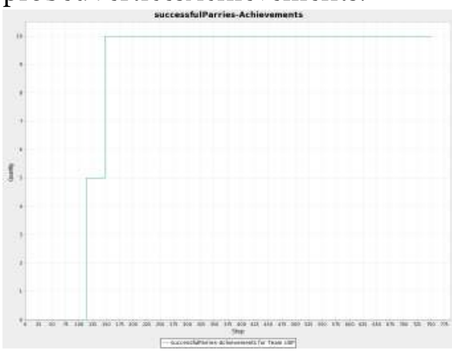


Figure 1107: successfulParriesAchievements.

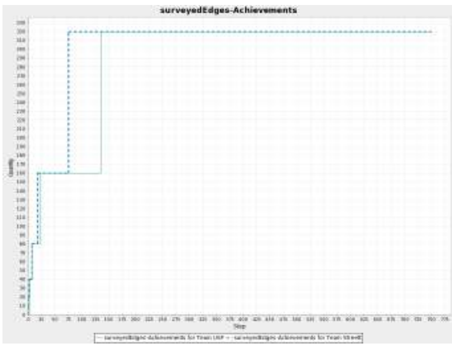


Figure 1108: surveyedEdgesAchievements.

59.4 Actions per Role

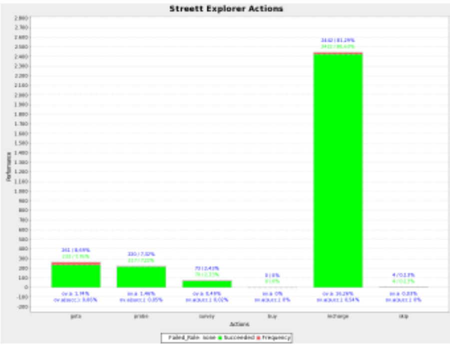


Figure 1109: Streett vs. USP – Simulation 2 - Streett Explorer Actions.

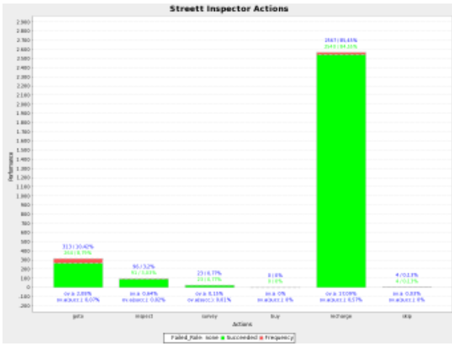


Figure 1110: Streett vs. USP – Simulation 2 - Streett Inspector Actions.

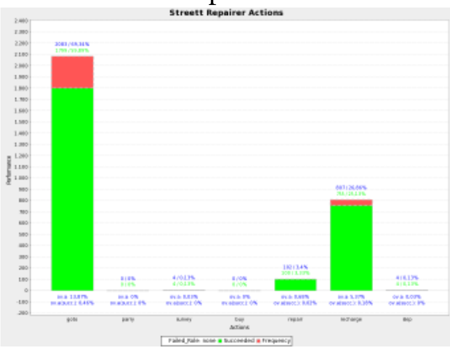


Figure 1111: Streett vs. USP – Simulation 2 - Streett Repairer Actions.

Figure 1112: Streett vs. USP – Simulation 2 - Streett Saboteur Actions.

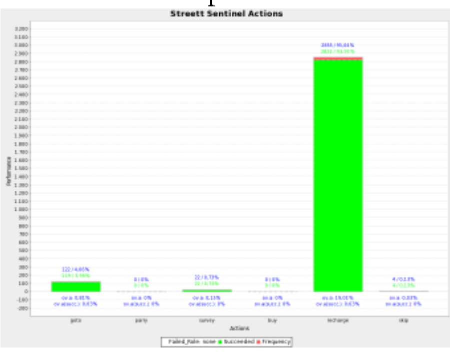


Figure 1113: Streett vs. USP – Simulation 2 - Streett Sentinel Actions.

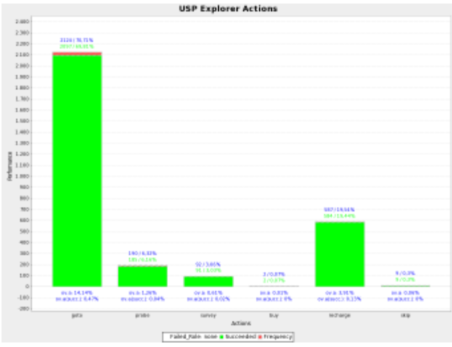


Figure 1114: Streett vs. USP – Simulation 2 - USP Explorer Actions.

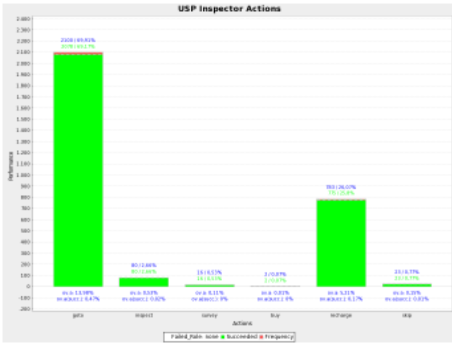


Figure 1115: Streett vs. USP – Simulation 2 - USP Inspector Actions.

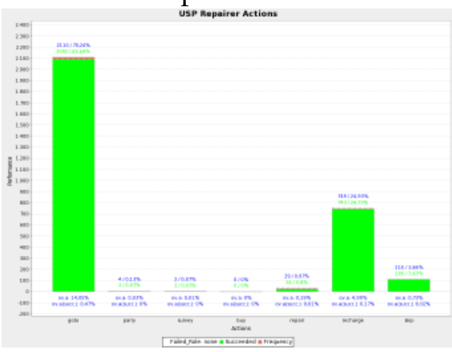


Figure 1116: Streett vs. USP – Simulation 2 - USP Repairer Actions.

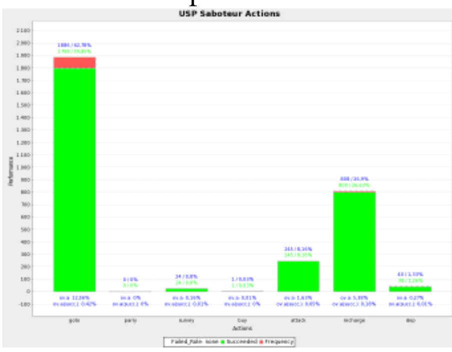


Figure 1117: Streett vs. USP – Simulation 2 - USP Saboteur Actions.

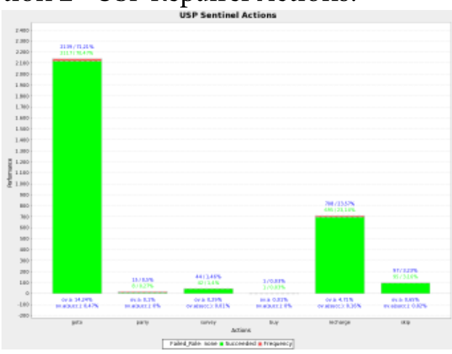


Figure 1118: Streett vs. USP – Simulation 2 - USP Sentinel Actions.

60 Streett vs. USP – Simulation 3

60.1 Scores, Zone Stability and Achievements

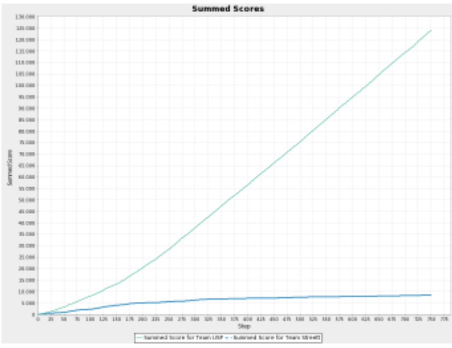


Figure 1119: Summed scores.

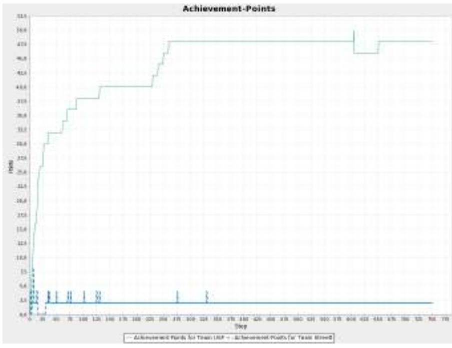


Figure 1120: Achievement points.

Figure 1121: Zones scores.

Figure 1122: Zones scores and achievement points.

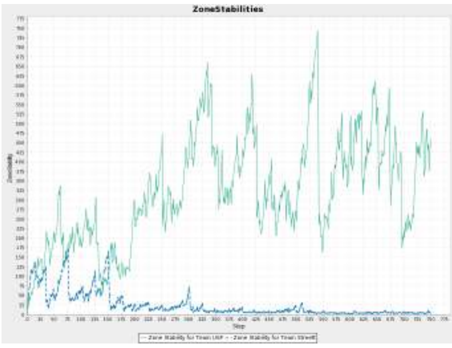


Figure 1123: Zone Stabilities.

Step	USP	Streett
1	surveyed10, surveyed20	surveyed10, surveyed20
2	surveyed40	surveyed40
4	area10, proved5	proved5
6	proved10	proved10, area10, inspected5
7	surveyed80	surveyed80
9	area20	
13	area40	attacked5
14	attacked5	inspected10, proved20
15	proved20, inspected5	
18	attacked10	
24	surveyed160	
26	inspected10	
29		attacked10
34	attacked20	surveyed160
36		proved40
50		area20
61	proved40	
70	attacked40	
71		attacked20
76		proved80
87	area80	
101		attacked40
124		area40
130	attacked80	surveyed320
229	area160	
239	proved80	
249	surveyed320	
259	attacked160	
275		proved160
330		inspected20
604	parried5	
650	attacked320	

Figure 1124: Achievements.

60.2 Stability

Reason	USP	%	Streett	%
failed away	23	0,15	5	0,03
failed parried			6	0,04
failed random	170	1,13	173	1,15
failed	114	0,76		
failed resources	77	0,51	437	2,91
failed attacked	21	0,14	133	0,89
noAction	116	0,77		

Figure 1125: Failed actions.

60.3 Achievements

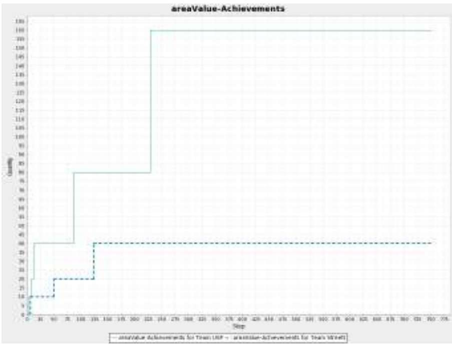


Figure 1126: areaValueAchievements.

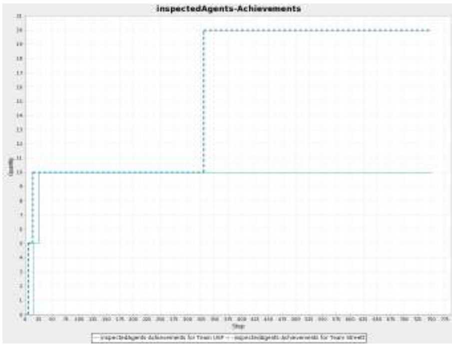


Figure 1127: inspectedAgentsAchievements.

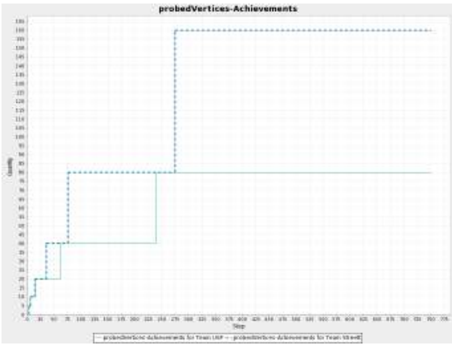


Figure 1128: probedVerticesAchievements.

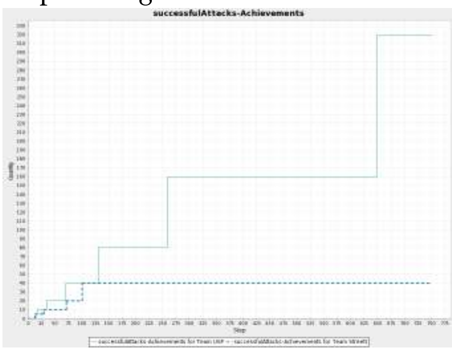


Figure 1129: successfulAttacksAchievements.

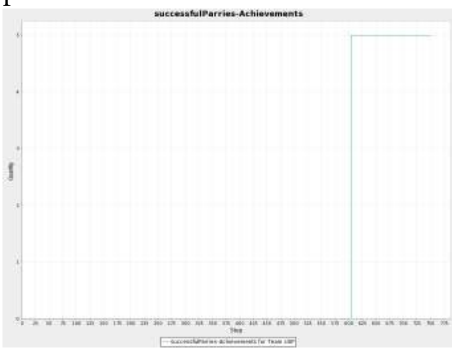


Figure 1130: successfulParriesAchievements.

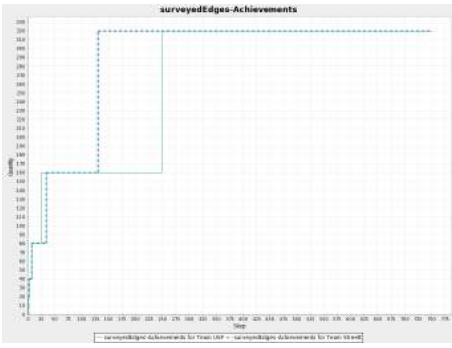


Figure 1131: surveyedEdgesAchievements.

60.4 Actions per Role

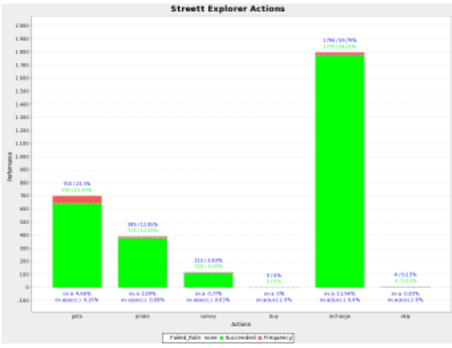


Figure 1132: Streett vs. USP – Simulation 3 - Streett Explorer Actions.

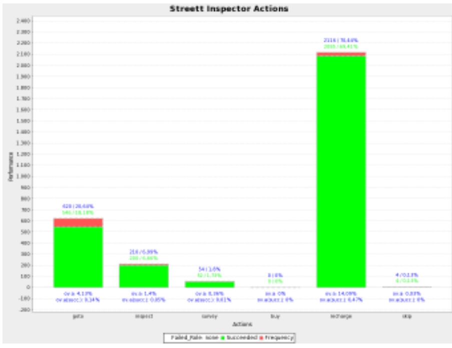


Figure 1133: Streett vs. USP – Simulation 3 - Streett Inspector Actions.

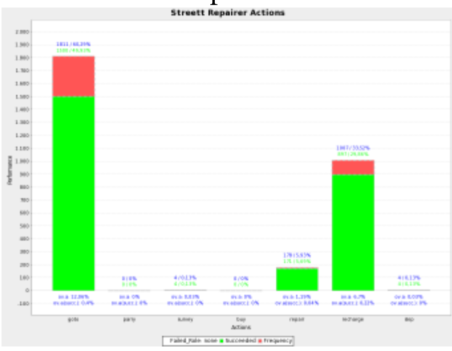


Figure 1134: Streett vs. USP – Simulation 3 - Streett Repairer Actions.

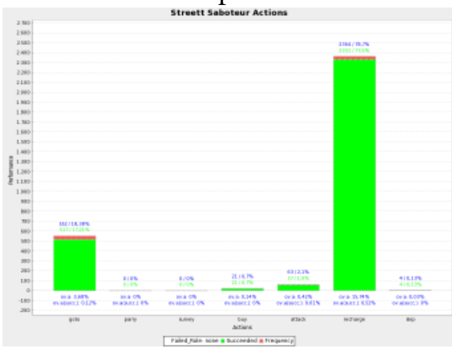


Figure 1135: Streett vs. USP – Simulation 3 - Streett Saboteur Actions.

Figure 1136: Streett vs. USP – Simulation 3 - Streett Sentinel Actions.

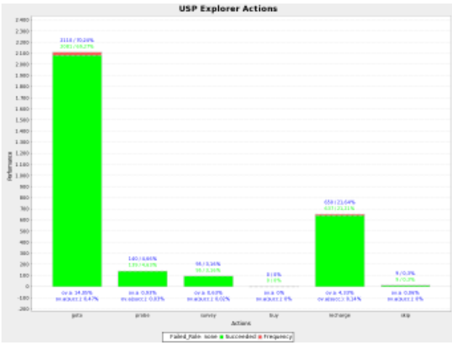


Figure 1137: Streett vs. USP – Simulation 3 - USP Explorer Actions.

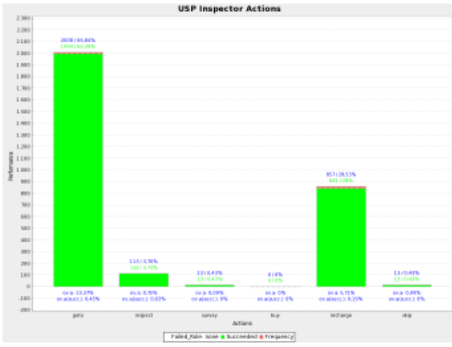


Figure 1138: Streett vs. USP – Simulation 3 - USP Inspector Actions.

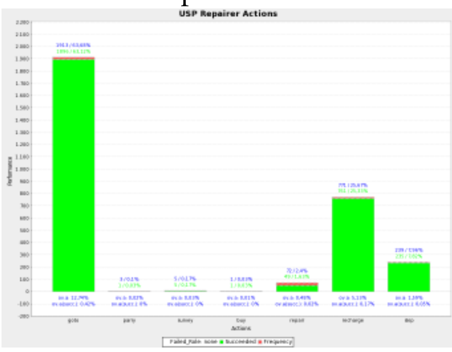


Figure 1139: Streett vs. USP – Simulation 3 - USP Repairer Actions.

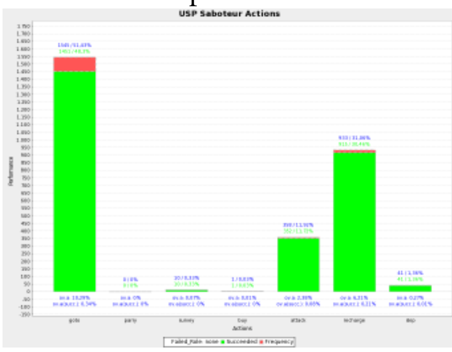


Figure 1140: Streett vs. USP – Simulation 3 - USP Saboteur Actions.

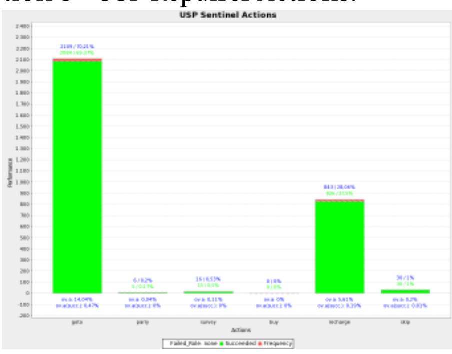


Figure 1141: Streett vs. USP – Simulation 3 - USP Sentinel Actions.

61 TUB vs. Python-DTU – Simulation 1

61.1 Scores, Zone Stability and Achievements

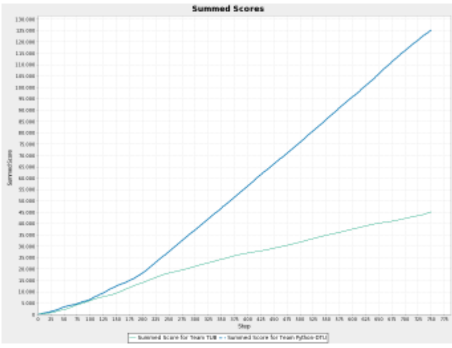


Figure 1142: Summed scores.

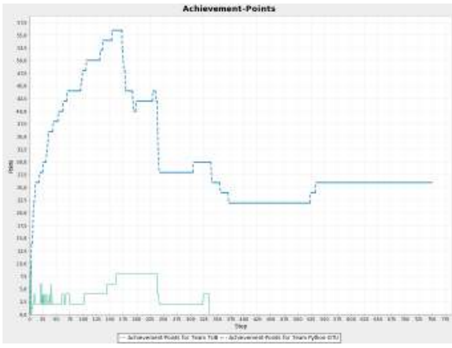


Figure 1143: Achievement points.

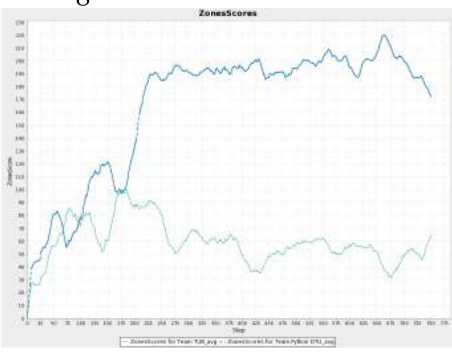


Figure 1144: Zones scores.

Figure 1145: Zones scores and achievement points.

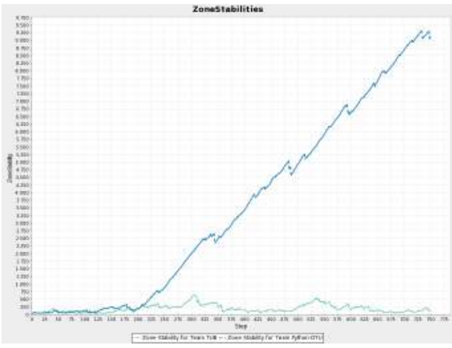


Figure 1146: Zone Stabilities.

Step	TUB	Python-DTU
1	surveyed40, surveyed10, surveyed20, area10, surveyed80	surveyed10, surveyed40, area10, surveyed20
3		surveyed80, proved5, inspected5
4	proved5	
5		proved10
6		area20, attacked5
7		surveyed160
8	area20	
9		inspected10
10	attacked5	proved20
18		attacked10
19	attacked10, inspected5	
20	proved10	
23	area40	
24		proved40
26	surveyed160	
30	attacked20	
31		inspected20
33		surveyed320
34		area40
36	inspected10	
37	proved20	
40	area80, attacked40	
43		attacked20
53		proved80
60	attacked80	
62		attacked40
67	proved40	
70		parried5
95		attacked80
98		surveyed640
101	attacked160	
105		proved160
131		area80
136		parried10
143	proved80	
153		attacked160
161	attacked320	
198		parried20
229		parried40
305		attacked320
323	attacked640	
523		parried80
533		attacked640

Figure 1147: Achievements.

61.2 Stability

Reason	TUB	%	Python-DTU	%
failed away	6	0,04		
failed parried	170	1,13		
failed random	160	1,07	155	1,03
failed wrong param	14	0,09		
failed resources	2	0,01		
failed	12	0,08		
failed attacked	60	0,4	38	0,25
noAction	12	0,08		

Figure 1148: Failed actions.

61.3 Achievements

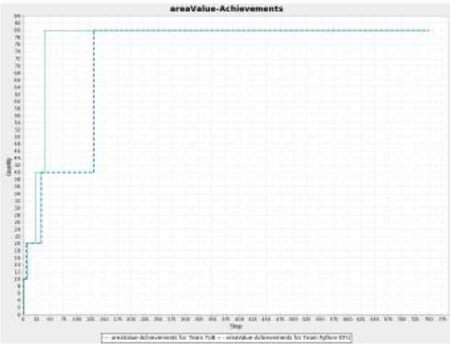


Figure 1149: areaValueAchievements.

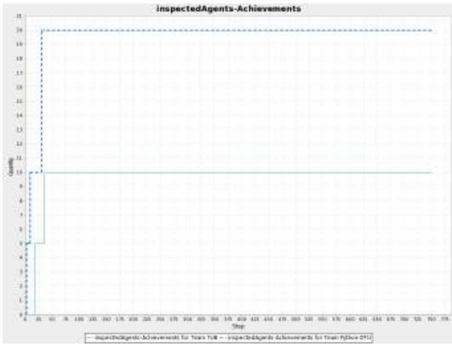


Figure 1150: inspectedAgentsAchievements.

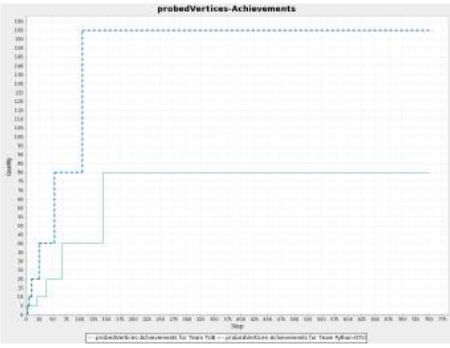


Figure 1151: probedVerticesAchievements.

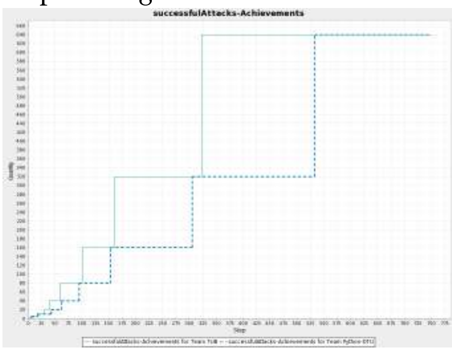


Figure 1152: successfulAttacksAchievements.

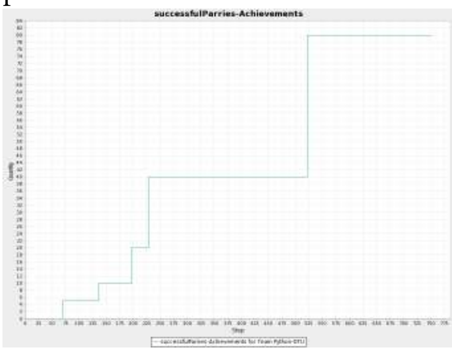


Figure 1153: successfulParriesAchievements.

Figure 1154: surveyedEdgesAchievements.

61.4 Actions per Role

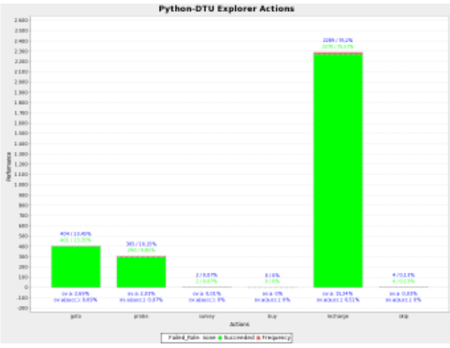


Figure 1155: TUB vs. Python-DTU – Simulation 1 - Python-DTU Explorer Actions.

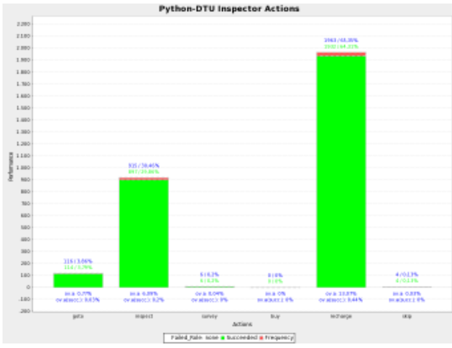


Figure 1156: TUB vs. Python-DTU – Simulation 1 - Python-DTU Inspector Actions.

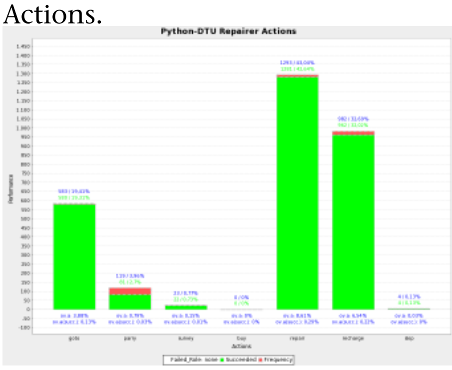


Figure 1157: TUB vs. Python-DTU – Simulation 1 - Python-DTU Repairer Actions.

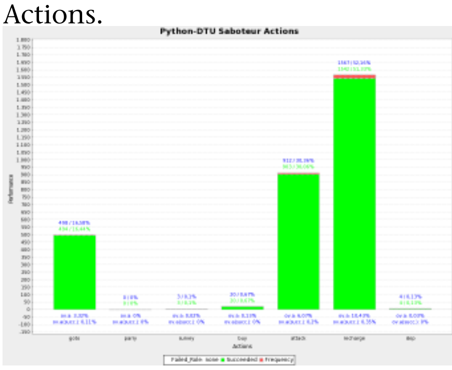


Figure 1158: TUB vs. Python-DTU – Simulation 1 - Python-DTU Saboteur Actions.

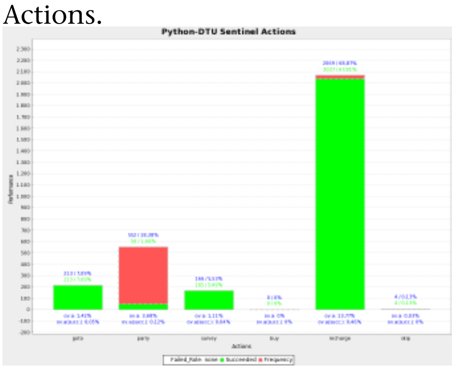


Figure 1159: TUB vs. Python-DTU – Simulation 1 - Python-DTU Sentinel Actions.

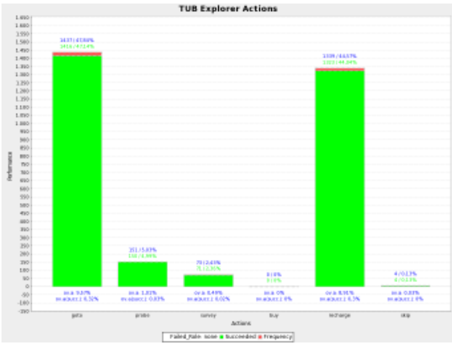


Figure 1160: TUB vs. Python-DTU – Simulation 1 - TUB Explorer Actions.

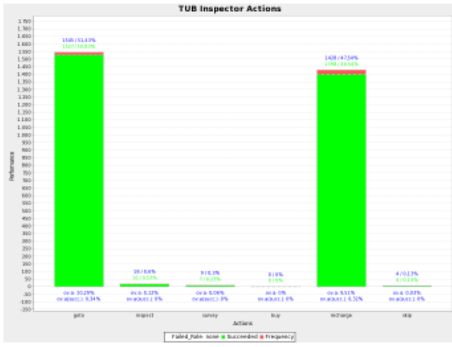


Figure 1161: TUB vs. Python-DTU – Simulation 1 - TUB Inspector Actions.

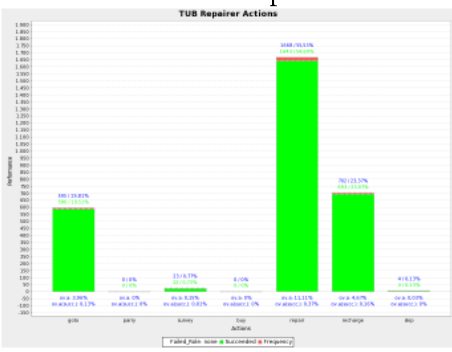


Figure 1162: TUB vs. Python-DTU – Simulation 1 - TUB Repairer Actions.

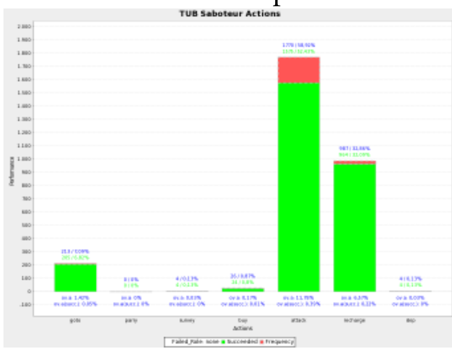


Figure 1163: TUB vs. Python-DTU – Simulation 1 - TUB Saboteur Actions.

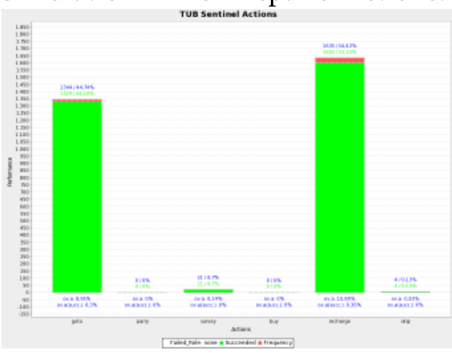


Figure 1164: TUB vs. Python-DTU – Simulation 1 - TUB Sentinel Actions.

62 TUB vs. Python-DTU – Simulation 2

62.1 Scores, Zone Stability and Achievements

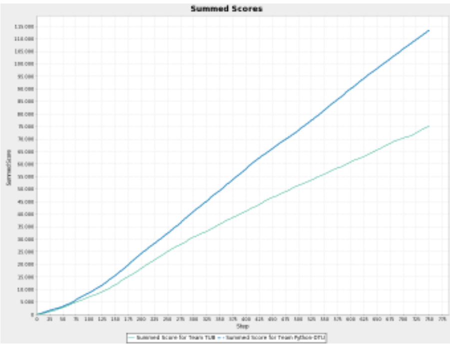


Figure 1165: Summed scores.

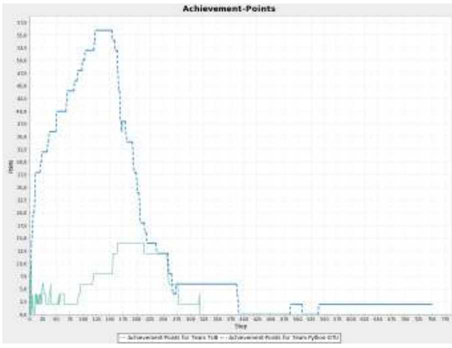


Figure 1166: Achievement points.

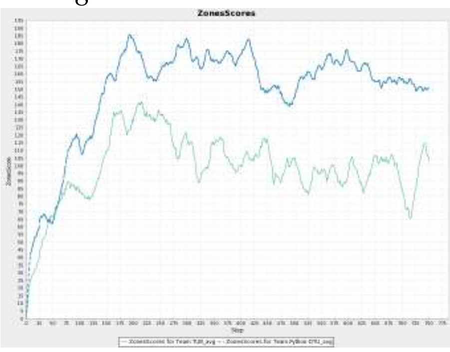


Figure 1167: Zones scores.

Figure 1168: Zones scores and achievement points.

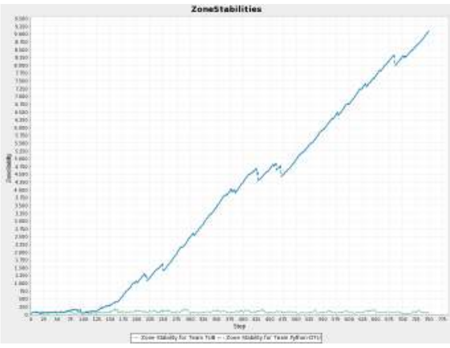


Figure 1169: Zone Stabilities.

Step	TUB	Python-DTU
1	surveyed10, surveyed80, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
3		area20, surveyed80, proved5
4	proved5	inspected5
5	area20	proved10
6		attacked5
9		proved20, surveyed160
10	proved10, attacked5	inspected10, attacked10
11	area40	
13	attacked10	
16	inspected5	
19	surveyed160	attacked20
21	inspected10	
22	proved20	proved40
23	attacked20	
33		attacked40
36	attacked40	surveyed320
38	area80	
49		proved80
50		area40
53	proved40	
56	attacked80	
68		attacked80
70		area80
83		parried5
89		parried10
90	attacked160	
94	inspected20	
98		proved160
103		inspected20
118	proved80	
120		surveyed640
122		attacked160
155	surveyed320, attacked320	
163	area160	
172		parried20
268		attacked320
272		parried40
316	attacked640	
486		parried80
539		attacked640

Figure 1170: Achievements.

62.2 Stability

Reason	TUB	%	Python-DTU	%
failed away	2	0,01		
failed parried	161	1,07		
failed wrong param	8	0,05		
failed random	155	1,03	135	0,9
failed resources	3	0,02		
failed	3	0,02		
failed attacked	73	0,49	32	0,21
noAction	3	0,02		

Figure 1171: Failed actions.

62.3 Achievements

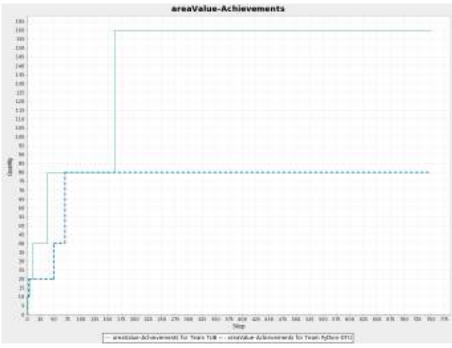


Figure 1172: areaValueAchievements.

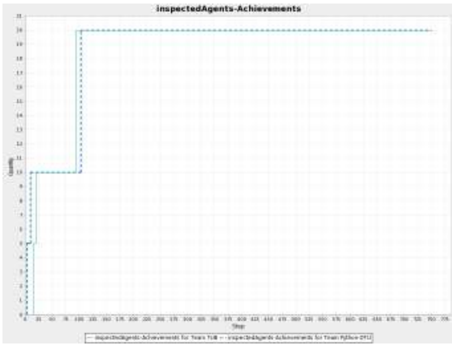


Figure 1173: inspectedAgentsAchievements.

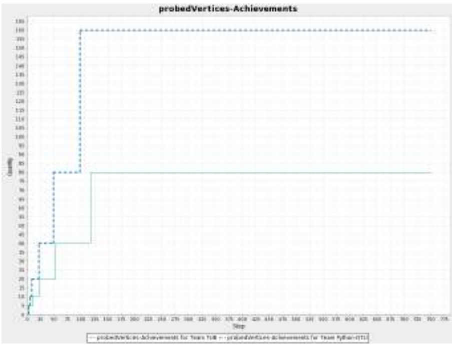


Figure 1174: probedVerticesAchievements.

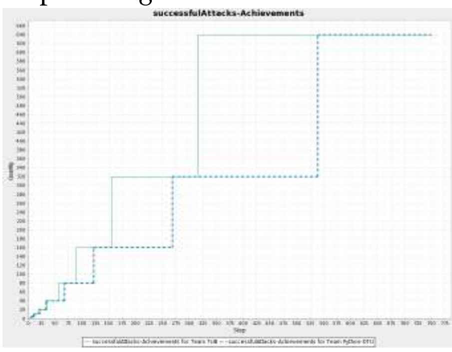


Figure 1175: successfulAttacksAchievements.

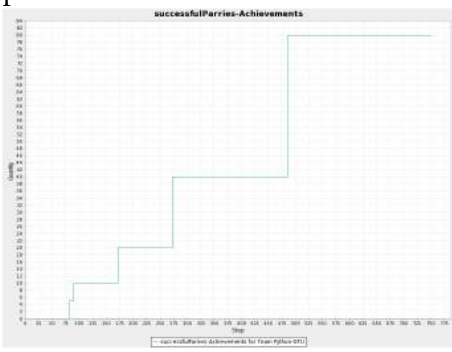


Figure 1176: successfulParriesAchievements.

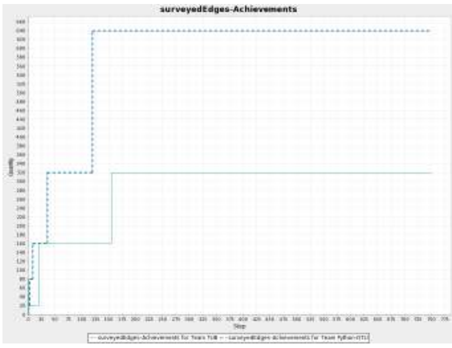


Figure 1177: surveyedEdgesAchievements.

62.4 Actions per Role

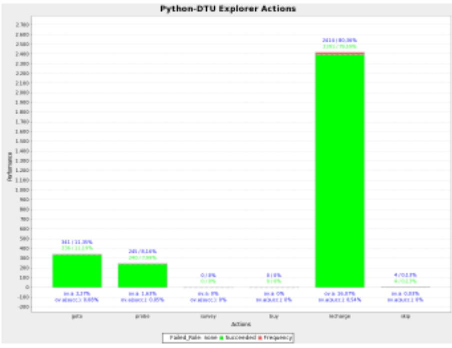


Figure 1178: TUB vs. Python-DTU – Simulation 2 - Python-DTU Explorer Actions.

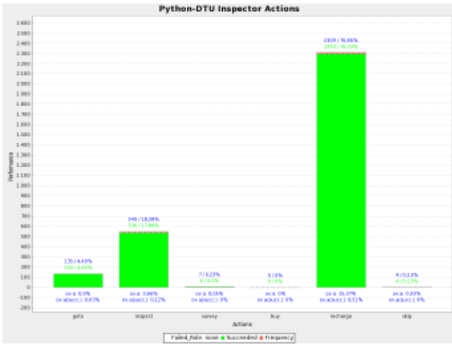


Figure 1179: TUB vs. Python-DTU – Simulation 2 - Python-DTU Inspector Actions.

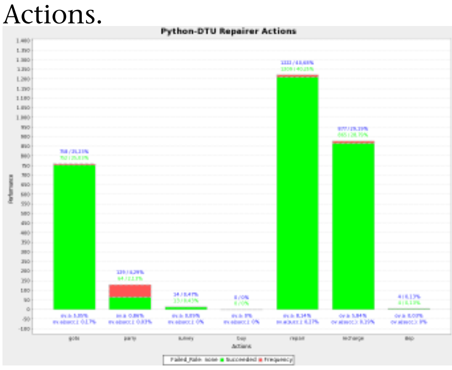


Figure 1180: TUB vs. Python-DTU – Simulation 2 - Python-DTU Repairer Actions.

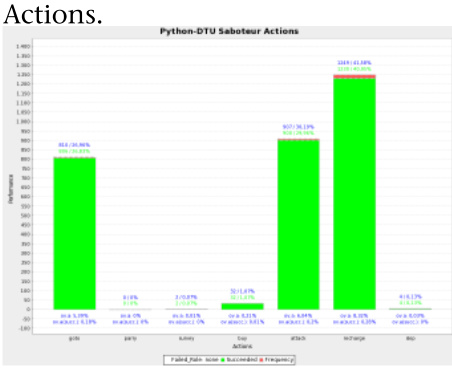


Figure 1181: TUB vs. Python-DTU – Simulation 2 - Python-DTU Saboteur Actions.

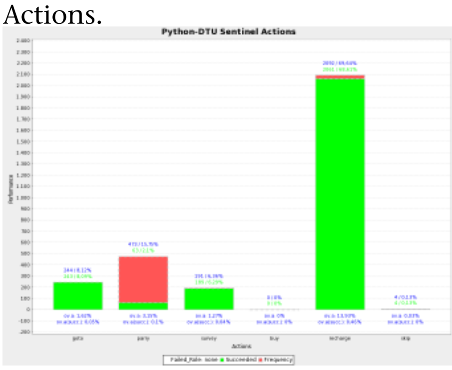


Figure 1182: TUB vs. Python-DTU – Simulation 2 - Python-DTU Sentinel Actions.

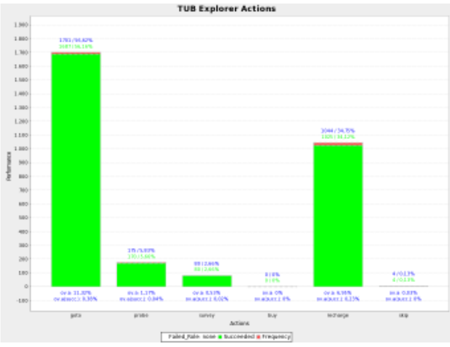


Figure 1183: TUB vs. Python-DTU – Simulation 2 - TUB Explorer Actions.

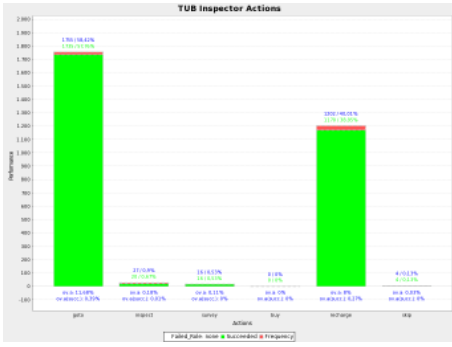


Figure 1184: TUB vs. Python-DTU – Simulation 2 - TUB Inspector Actions.

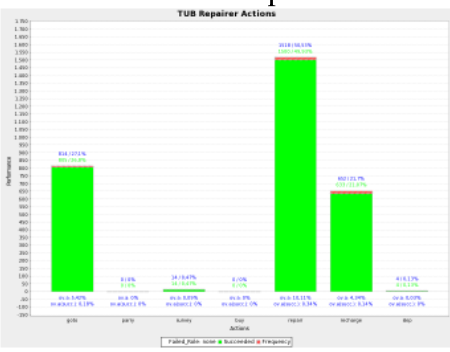


Figure 1185: TUB vs. Python-DTU – Simulation 2 - TUB Repairer Actions.

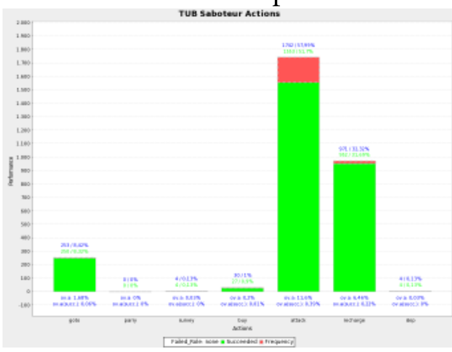


Figure 1186: TUB vs. Python-DTU – Simulation 2 - TUB Saboteur Actions.

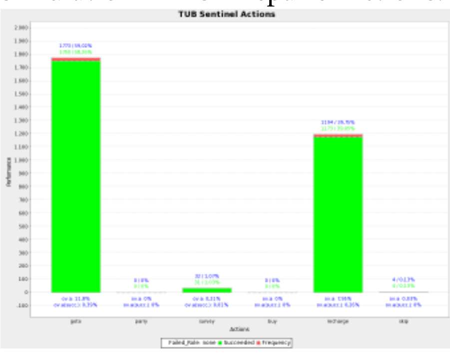


Figure 1187: TUB vs. Python-DTU – Simulation 2 - TUB Sentinel Actions.

63 TUB vs. Python-DTU – Simulation 3

63.1 Scores, Zone Stability and Achievements

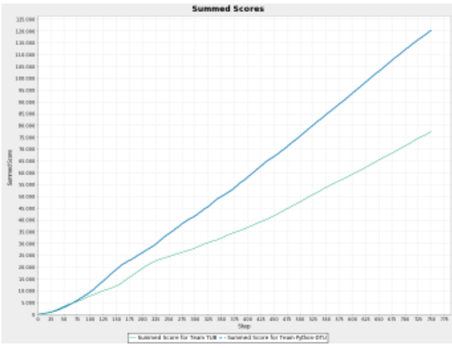


Figure 1188: Summed scores.

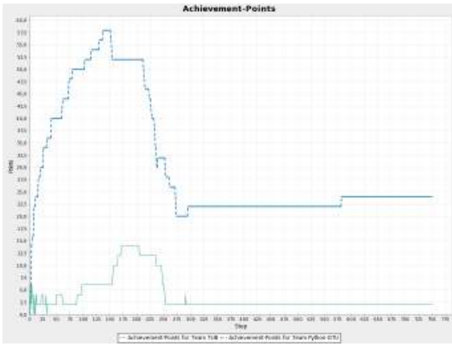


Figure 1189: Achievement points.

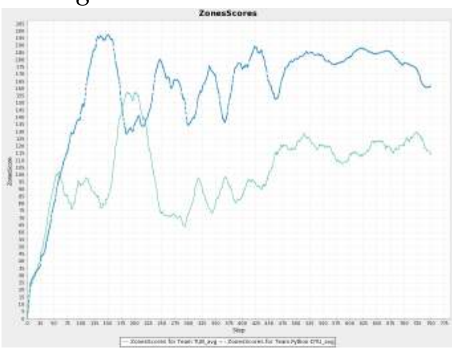


Figure 1190: Zones scores.

Figure 1191: Zones scores and achievement points.

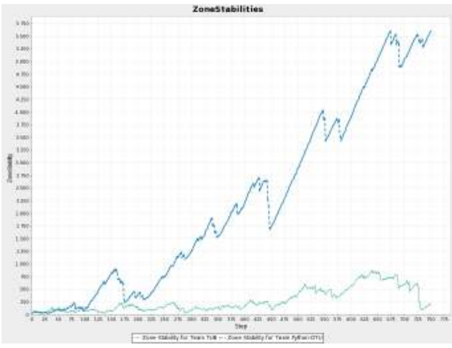


Figure 1192: Zone Stabilities.

Step	TUB	Python-DTU
1	surveyed10, surveyed40, surveyed20	surveyed10, surveyed40, surveyed20
3	surveyed80, area10	surveyed80, attacked5, proved5, inspected5
4	area20, proved5	area10
7	inspected5	proved10, surveyed160
8	attacked5	attacked10
10		inspected10
11	area40, proved10	
12	inspected10, surveyed160	
13	attacked10	
15		inspected20
16		proved20
19		attacked20
21	attacked20	
22	proved20	
25		area20, surveyed320
30	area80	
32	attacked40	proved40
40		area40, attacked40
50	proved40	
51	attacked80	
60		area80
61		proved80
72		attacked80
73		parried5
79		parried10
88	attacked160	
97	inspected20	
102		parried20
114		proved160
129		attacked160
136		area160
154	attacked320	
156	proved80	
164	area160	
170	surveyed320	
238		parried40
290	attacked640	
294		attacked320
581		attacked640

Figure 1193: Achievements.

63.2 Stability

Reason	TUB	%	Python-DTU	%
failed away	2	0,01	3	0,02
failed parried	72	0,48		
failed wrong param	4	0,03		
failed random	133	0,89	148	0,99
failed resources	3	0,02		
failed attacked	51	0,34	76	0,51

Figure 1194: Failed actions.

63.3 Achievements

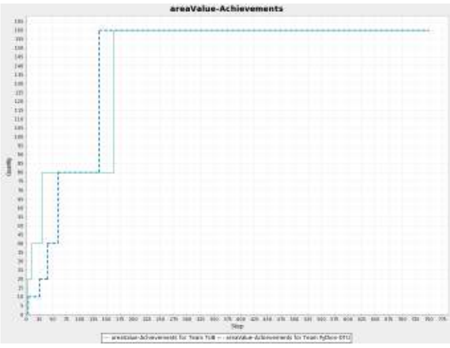


Figure 1195: areaValueAchievements.

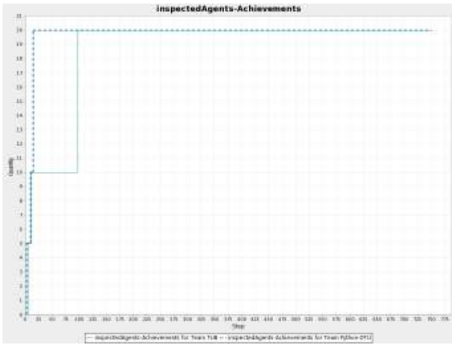


Figure 1196: inspectedAgentsAchievements.

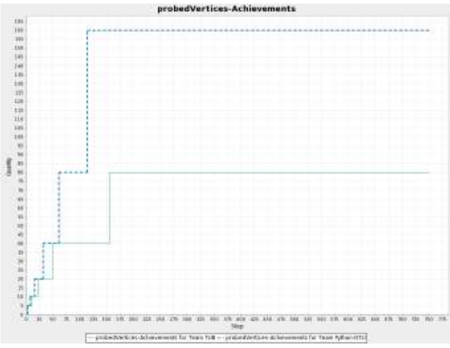


Figure 1197: probedVerticesAchievements.

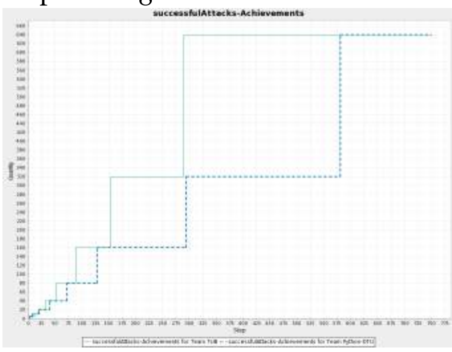


Figure 1198: successfulAttacksAchievements.

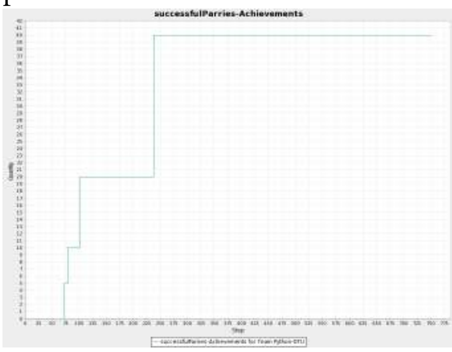


Figure 1199: successfulParriesAchievements.

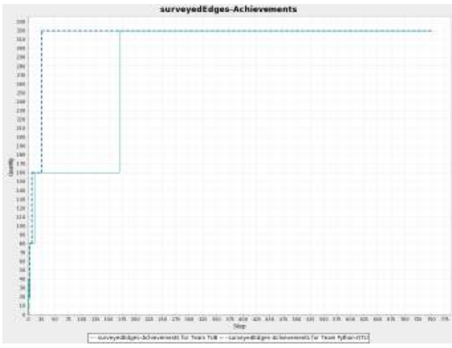


Figure 1200: surveyedEdgesAchievements.

63.4 Actions per Role

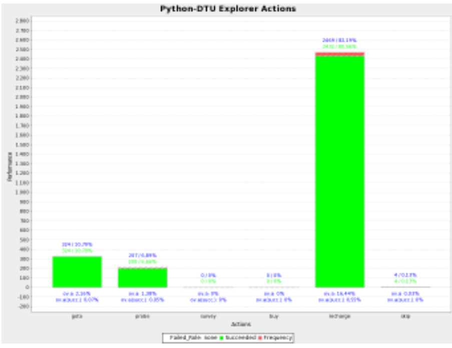


Figure 1201: TUB vs. Python-DTU – Simulation 3 - Python-DTU Explorer Actions.

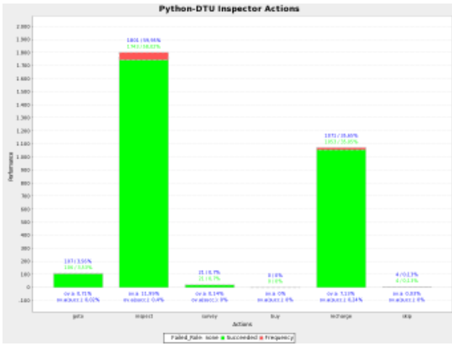


Figure 1202: TUB vs. Python-DTU – Simulation 3 - Python-DTU Inspector Actions.

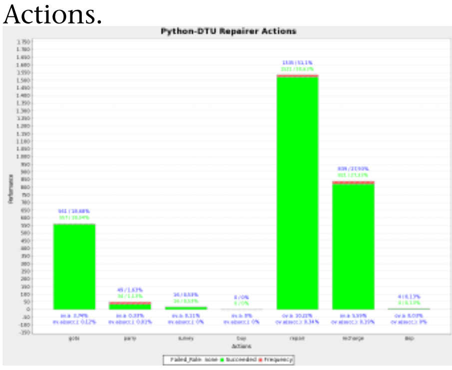


Figure 1203: TUB vs. Python-DTU – Simulation 3 - Python-DTU Repairer Actions.

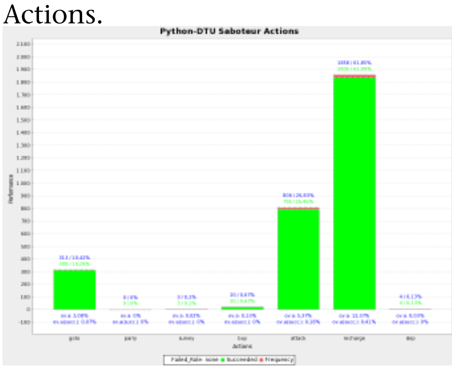


Figure 1204: TUB vs. Python-DTU – Simulation 3 - Python-DTU Saboteur Actions.

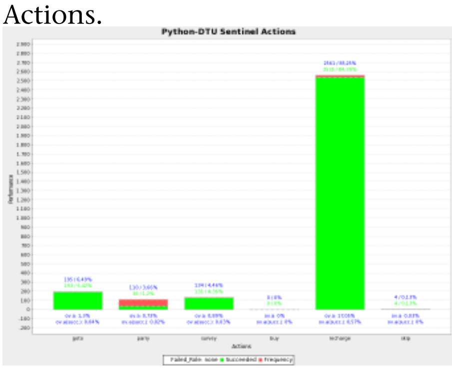


Figure 1205: TUB vs. Python-DTU – Simulation 3 - Python-DTU Sentinel Actions.

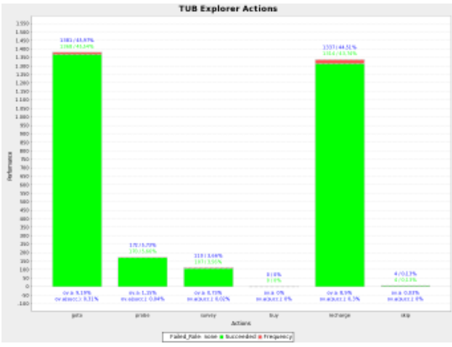


Figure 1206: TUB vs. Python-DTU – Simulation 3 - TUB Explorer Actions.

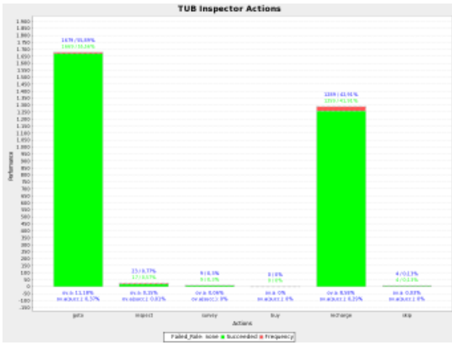


Figure 1207: TUB vs. Python-DTU – Simulation 3 - TUB Inspector Actions.

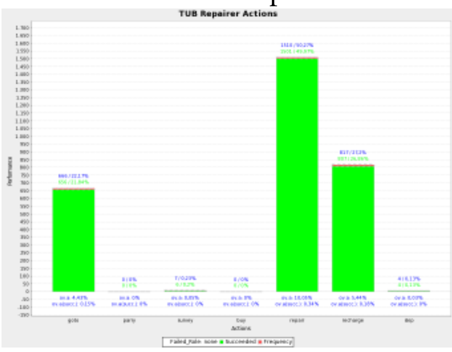


Figure 1208: TUB vs. Python-DTU – Simulation 3 - TUB Repairer Actions.

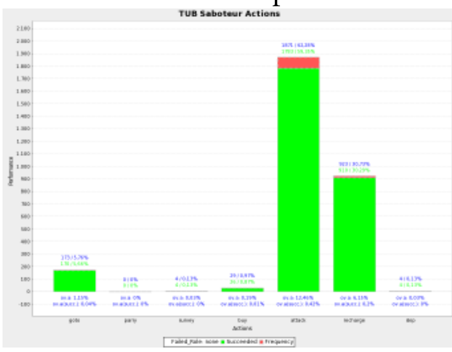


Figure 1209: TUB vs. Python-DTU – Simulation 3 - TUB Saboteur Actions.

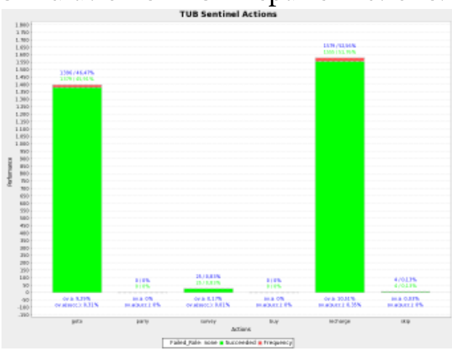


Figure 1210: TUB vs. Python-DTU – Simulation 3 - TUB Sentinel Actions.

64 TUB vs. UFSC – Simulation 1

64.1 Scores, Zone Stability and Achievements

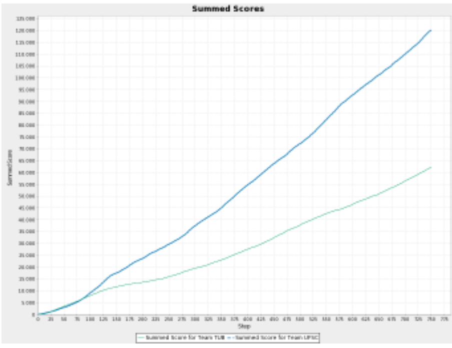


Figure 1211: Summed scores.

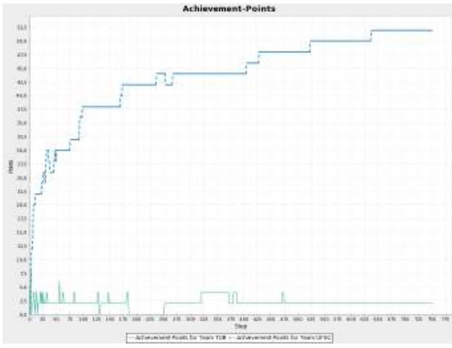


Figure 1212: Achievement points.

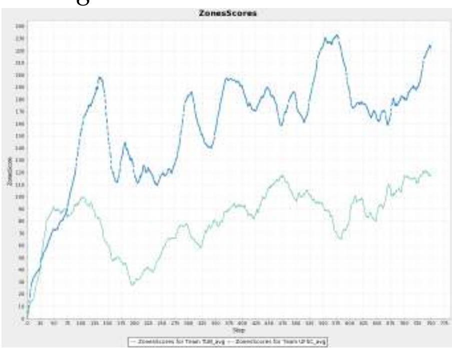


Figure 1213: Zones scores.

Figure 1214: Zones scores and achievement points.

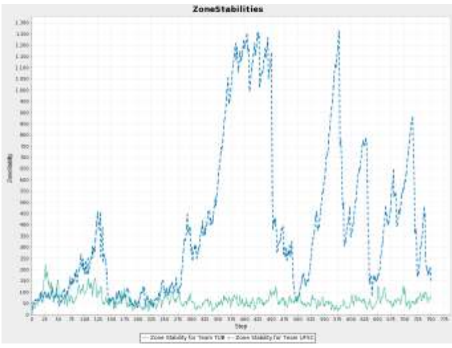


Figure 1215: Zone Stabilities.

Step	TUB	UFSC
1	surveyed10, surveyed80, surveyed40, surveyed20	surveyed10, surveyed40, surveyed20
3		surveyed80, area10, proved5
4	proved5	
5		proved10
6		surveyed160, attacked5, inspected5
7	area10	
8	attacked5	
10	inspected5	proved20
12	area20	
13	area40	
15	proved10	
17		surveyed320
19	attacked10	attacked10
22	area80	proved40
24		area20
25	proved20	
26		parried5
29		attacked20
30		parried10
31	surveyed160	area40
32	attacked20	
45		inspected10
47		attacked40
49		proved80
54	proved40	
55	attacked40	
61	inspected10	
75		area80
81	attacked80	
92		attacked80
93		area160
98		proved160
126	surveyed320	
132	attacked160	
146	proved80	
169		attacked160
172		parried20
180	inspected20	
236		inspected20
251	attacked320	
266		attacked320
319	proved160	
379	attacked640	
404		parried40
427		attacked640
471	surveyed640	
523		parried80
636		surveyed640

Figure 1216: Achievements.

64.2 Stability

Reason	TUB	%	UFSC	%
failed away	4	0,03	8	0,05
failed parried	206	1,37		
failed wrong param	255	1,7		
failed random	165	1,1	145	0,97
failed resources			5	0,03
failed	108	0,72		
failed attacked	103	0,69	94	0,63
noAction	108	0,72		

Figure 1217: Failed actions.

64.3 Achievements

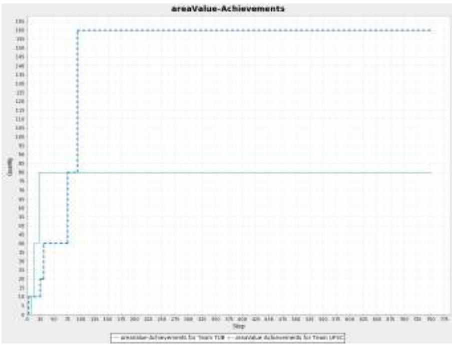


Figure 1218: areaValueAchievements.

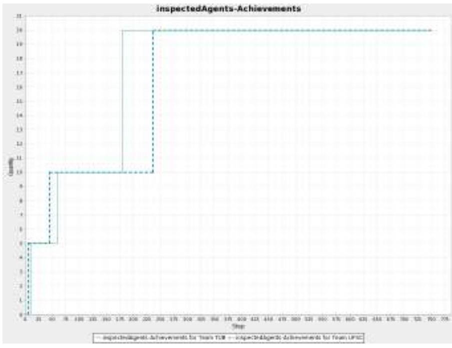


Figure 1219: inspectedAgentsAchievements.

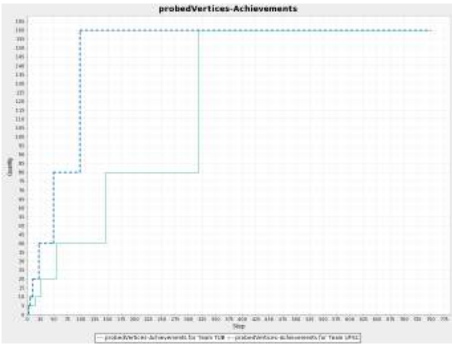


Figure 1220: probedVerticesAchievements.

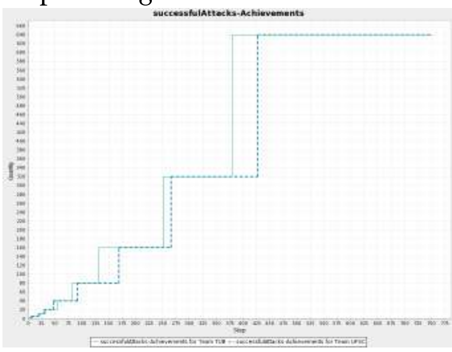


Figure 1221: successfulAttacksAchievements.

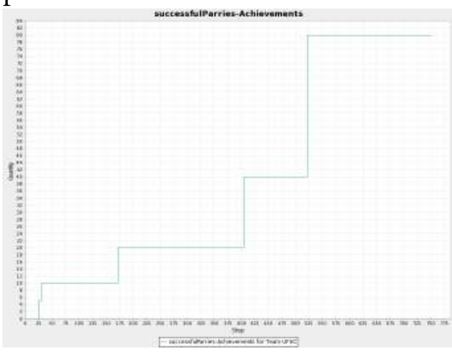


Figure 1222: successfulParriesAchievements.

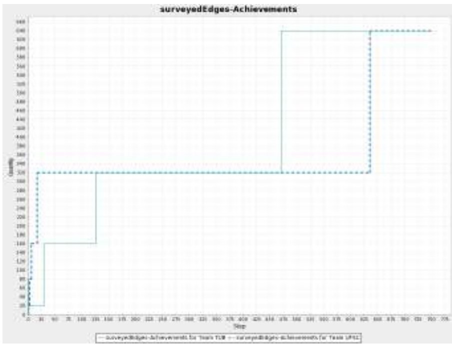


Figure 1223: surveyedEdgesAchievements.

64.4 Actions per Role

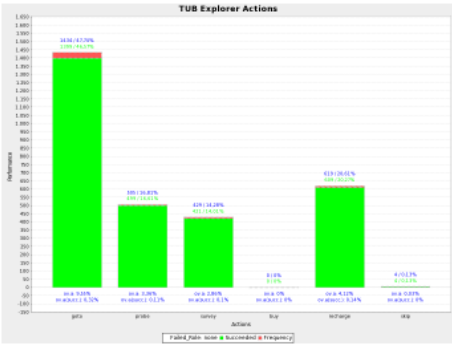


Figure 1224: TUB vs. UFSC – Simulation 1 - TUB Explorer Actions.

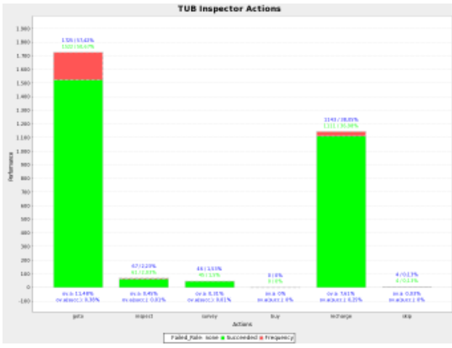


Figure 1225: TUB vs. UFSC – Simulation 1 - TUB Inspector Actions.

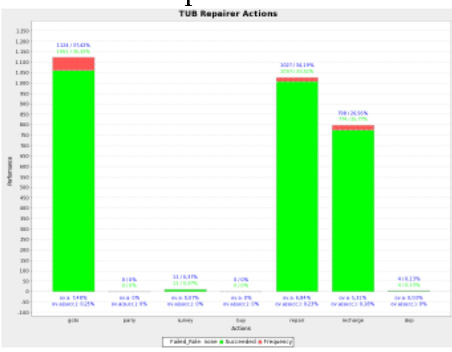


Figure 1226: TUB vs. UFSC – Simulation 1 - TUB Repairer Actions.

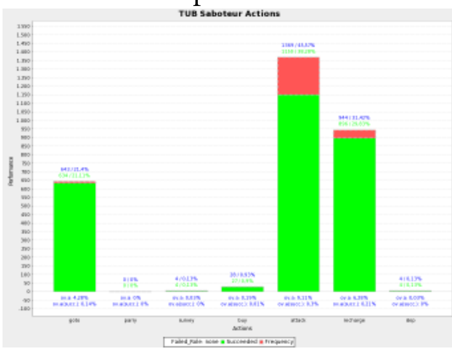


Figure 1227: TUB vs. UFSC – Simulation 1 - TUB Saboteur Actions.

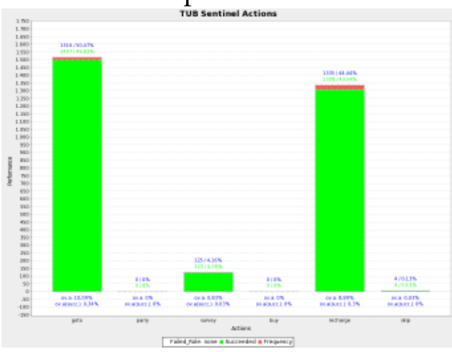


Figure 1228: TUB vs. UFSC – Simulation 1 - TUB Sentinel Actions.

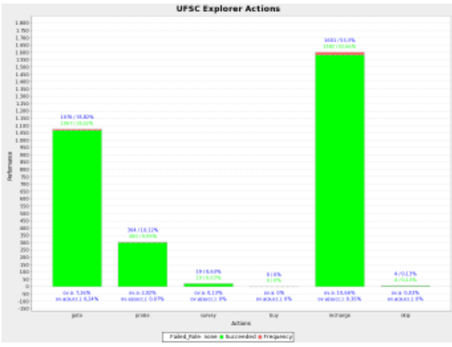


Figure 1229: TUB vs. UFSC – Simulation 1 - UFSC Explorer Actions.

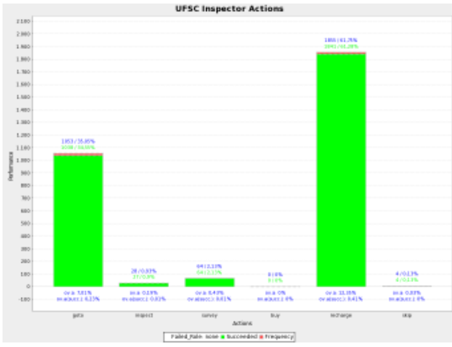


Figure 1230: TUB vs. UFSC – Simulation 1 - UFSC Inspector Actions.

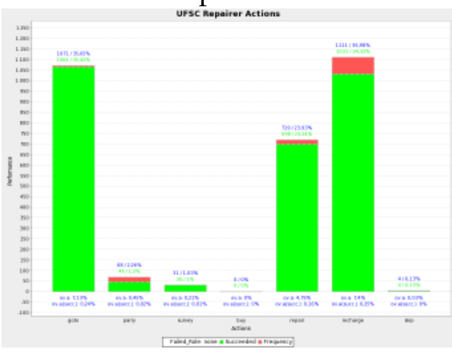


Figure 1231: TUB vs. UFSC – Simulation 1 - UFSC Repairer Actions.

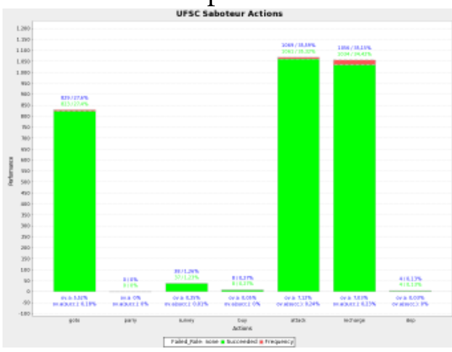


Figure 1232: TUB vs. UFSC – Simulation 1 - UFSC Saboteur Actions.

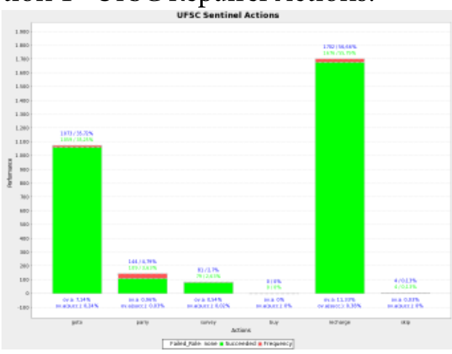


Figure 1233: TUB vs. UFSC – Simulation 1 - UFSC Sentinel Actions.

65 TUB vs. UFSC – Simulation 2

65.1 Scores, Zone Stability and Achievements

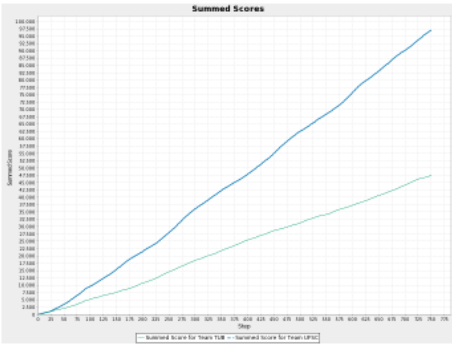


Figure 1234: Summed scores.

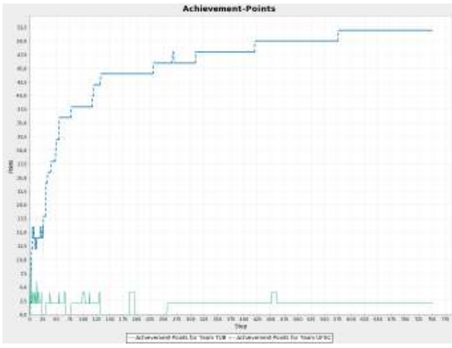


Figure 1235: Achievement points.

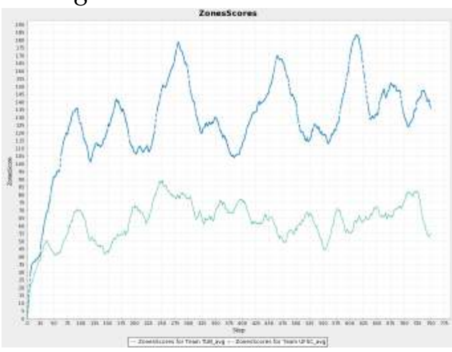


Figure 1236: Zones scores.

Figure 1237: Zones scores and achievement points.

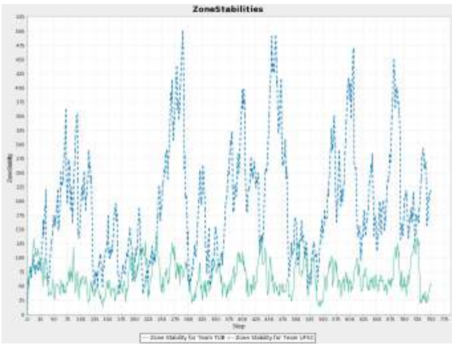


Figure 1238: Zone Stabilities.

Step	TUB	UFSC
1	surveyed10, surveyed80, surveyed40, area10, surveyed20	surveyed10, area10
2		surveyed40, surveyed20
3		surveyed80, proved5
4	proved5	
5		area20, proved10
7		surveyed160
8	attacked5	
10	area20	
11		proved20
13	proved10, surveyed160	attacked5
15	attacked10	
16	inspected5	
17	area40	
19		surveyed320, attacked10
21	proved20	
23		proved40
25		area40, parried5
29	inspected10	
30		parried10, attacked20, inspected5
33		inspected10
37	attacked20	
39		parried20
48		attacked40
49		proved80
54	proved40	
55		area80, parried40
64	attacked40	
77	area80	attacked80
98	attacked80	
111	surveyed320	
117		proved160
119		parried80
129	proved80	
132		attacked160
185	inspected20, attacked160	
231		attacked320
256	attacked320	
266		area160
309		inspected20
420		attacked640
451	attacked640	
575		parried160

Figure 1239: Achievements.

65.2 Stability

Reason	TUB	%	UFSC	%
failed away	10	0,07	2	0,01
failed parried	254	1,69		
failed wrong param	4	0,03		
failed random	162	1,08	138	0,92
failed resources	2	0,01	2	0,01
failed	2	0,01	22	0,15
failed attacked	140	0,93	62	0,41
noAction	2	0,01	22	0,15

Figure 1240: Failed actions.

65.3 Achievements

Figure 1241: areaValueAchievements.

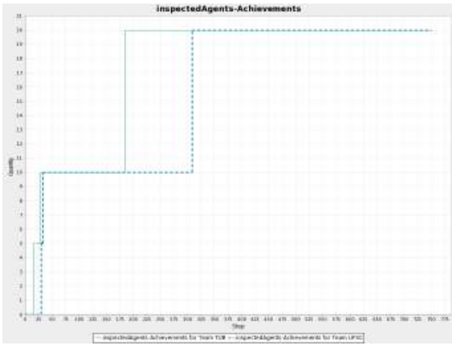


Figure 1242: inspectedAgentsAchievements.

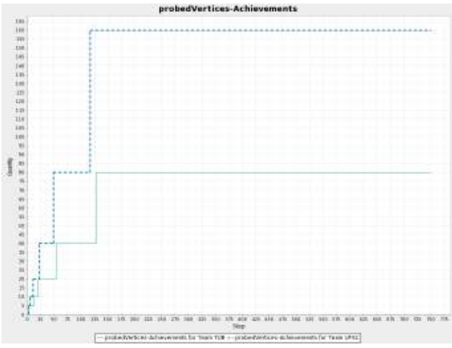


Figure 1243: probedVerticesAchievements.

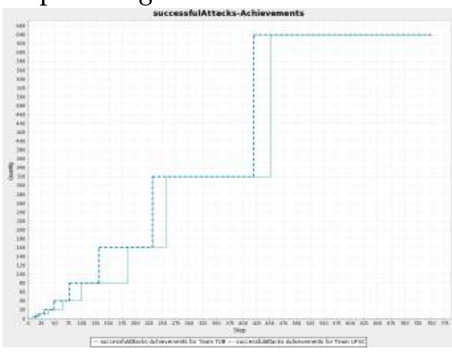


Figure 1244: successfulAttacksAchievements.

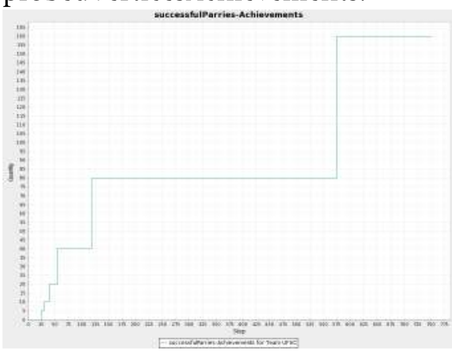


Figure 1245: successfulParriesAchievements.

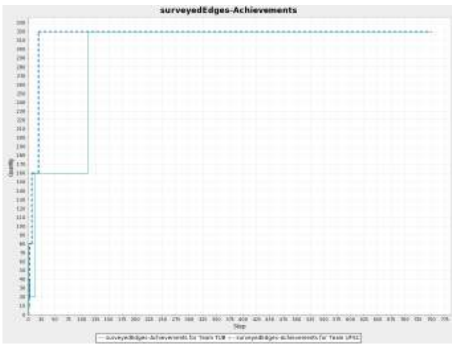


Figure 1246: surveyedEdgesAchievements.

65.4 Actions per Role

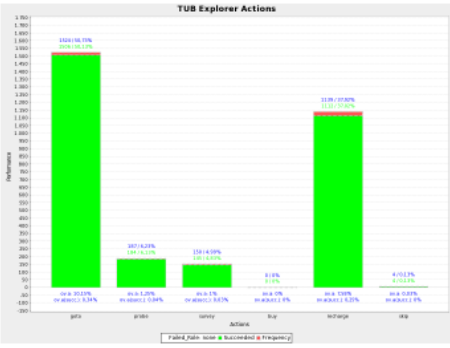


Figure 1247: TUB vs. UFSC – Simulation 2 - TUB Explorer Actions.

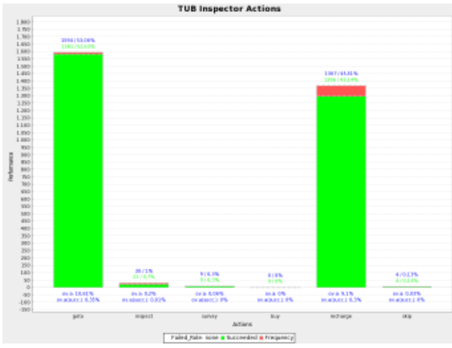


Figure 1248: TUB vs. UFSC – Simulation 2 - TUB Inspector Actions.

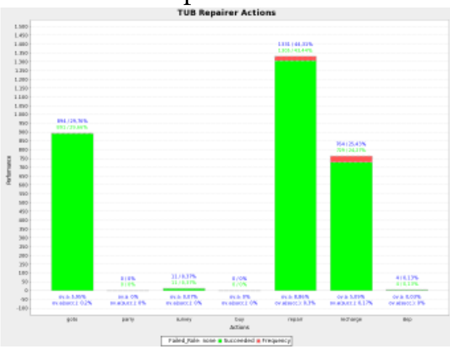


Figure 1249: TUB vs. UFSC – Simulation 2 - TUB Repairer Actions.

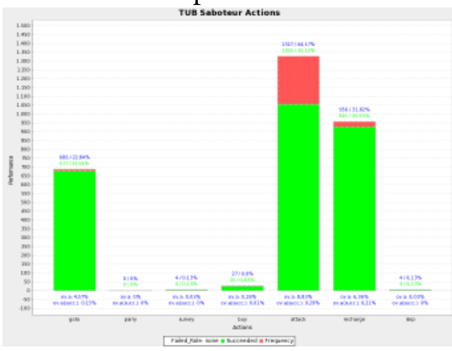


Figure 1250: TUB vs. UFSC – Simulation 2 - TUB Saboteur Actions.

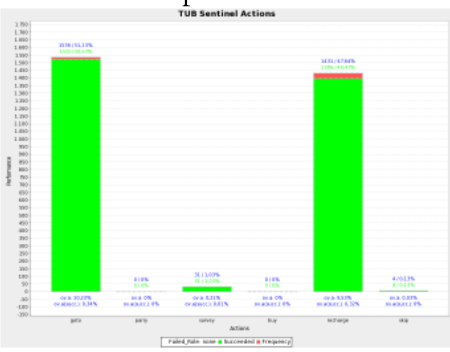


Figure 1251: TUB vs. UFSC – Simulation 2 - TUB Sentinel Actions.

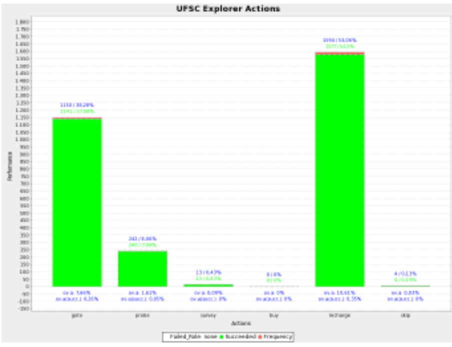


Figure 1252: TUB vs. UFSC – Simulation 2 - UFSC Explorer Actions.

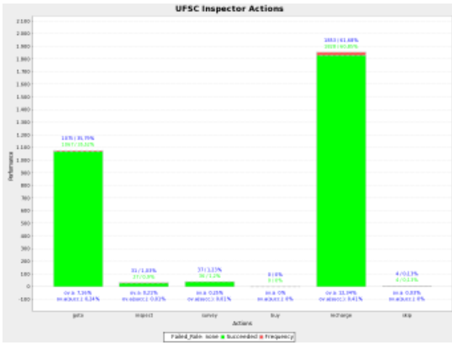


Figure 1253: TUB vs. UFSC – Simulation 2 - UFSC Inspector Actions.

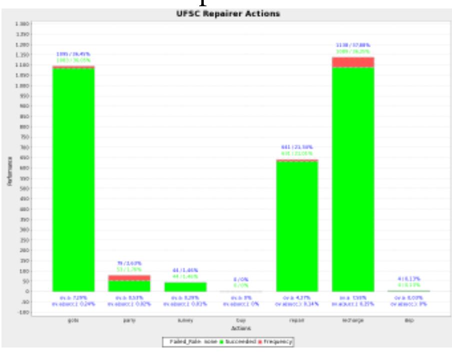


Figure 1254: TUB vs. UFSC – Simulation 2 - UFSC Repairer Actions.

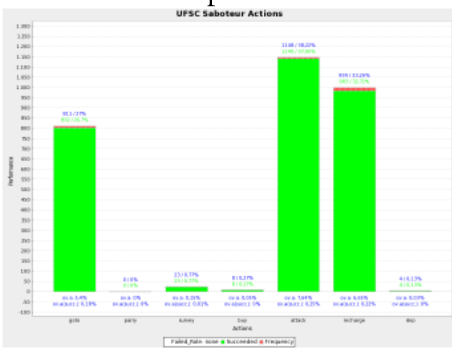


Figure 1255: TUB vs. UFSC – Simulation 2 - UFSC Saboteur Actions.

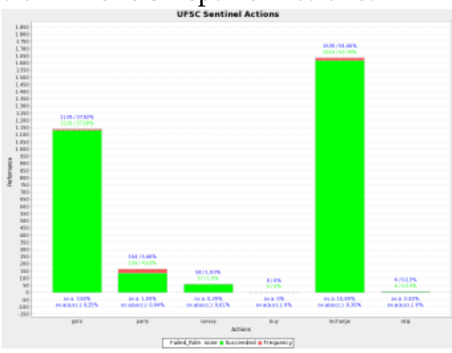


Figure 1256: TUB vs. UFSC – Simulation 2 - UFSC Sentinel Actions.

66 TUB vs. UFSC – Simulation 3

66.1 Scores, Zone Stability and Achievements

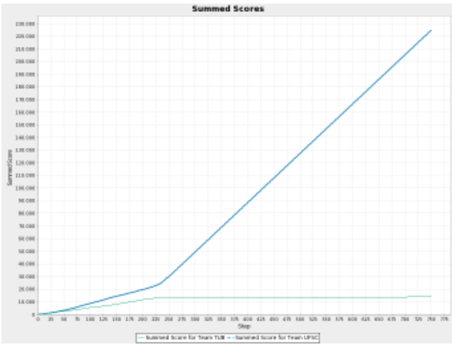


Figure 1257: Summed scores.

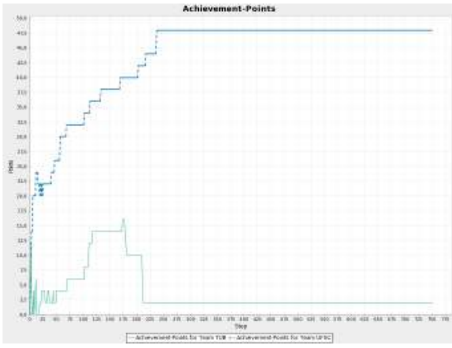


Figure 1258: Achievement points.

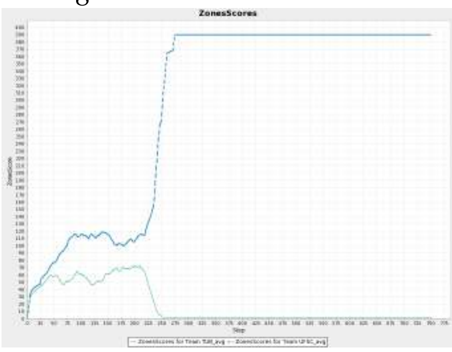


Figure 1259: Zones scores.

Figure 1260: Zones scores and achievement points.

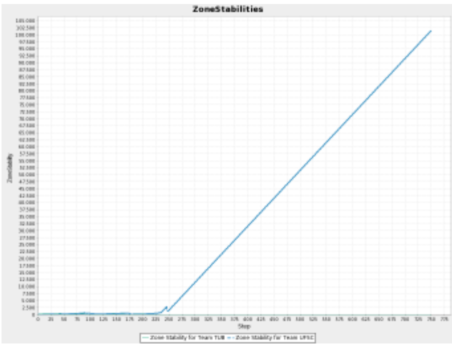


Figure 1261: Zone Stabilities.

Step	TUB	UFSC
1	surveyed10, area20, surveyed80, surveyed40, area10, surveyed20	surveyed10, area20, area10
2	inspected5	surveyed20, inspected5
3	proved5	surveyed40, proved5
5		proved10, surveyed80, attacked5
6	proved10	
7	area40	
9	surveyed160	
10	inspected10	surveyed160
11	attacked5	inspected10, proved20
12		attacked10
18	proved20	
19		attacked20
22	attacked10	proved40
24		parried5
26		area40
33	area80	
35	attacked20	
40		attacked40
43	proved40	
46		proved80
49	attacked40	
56		attacked80, area80
68		surveyed320
70	attacked80	
101		attacked160
102	proved80	
109	attacked160	
110	inspected20	
112		proved160
116	surveyed320	
132		inspected20
168		parried10
172	attacked320	
202		attacked320
216		parried20
236		area160, area320

Figure 1262: Achievements.

66.2 Stability

Reason	TUB	%	UFSC	%
failed away	1	0,01	3	0,02
failed parried	24	0,16		
failed wrong param	15	0,1		
failed random	143	0,95	161	1,07
failed resources	2	0,01		
failed	1790	11,93	9	0,06
failed attacked	40	0,27	19	0,13
noAction	1801	12,01	9	0,06

Figure 1263: Failed actions.

66.3 Achievements

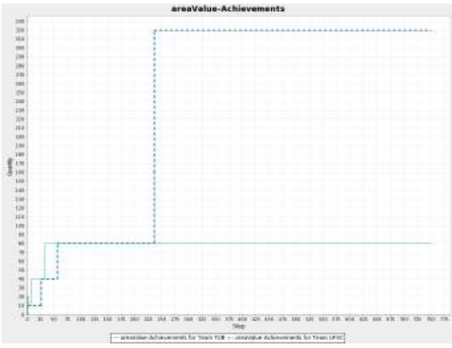


Figure 1264: areaValueAchievements.

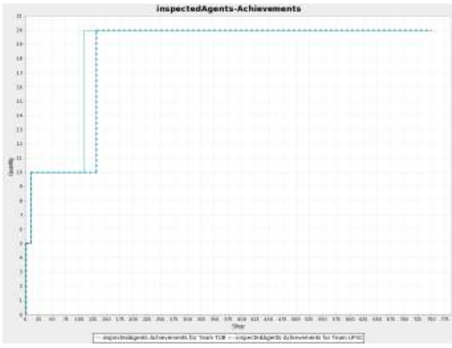


Figure 1265: inspectedAgentsAchievements.

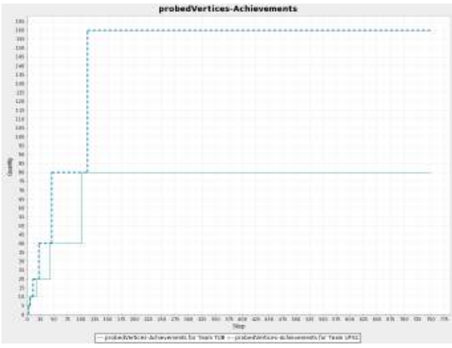


Figure 1266: probedVerticesAchievements.

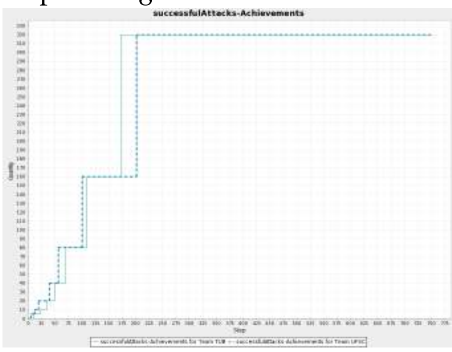


Figure 1267: successfulAttacksAchievements.

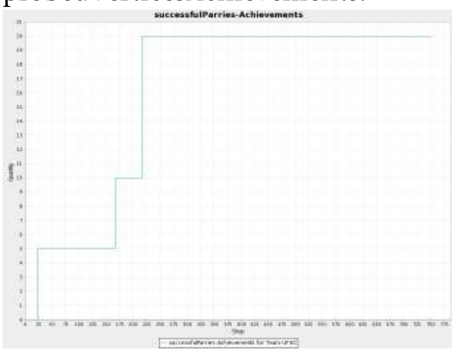


Figure 1268: successfulParriesAchievements.

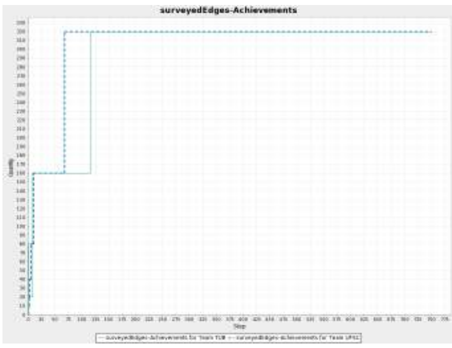


Figure 1269: surveyedEdgesAchievements.

66.4 Actions per Role

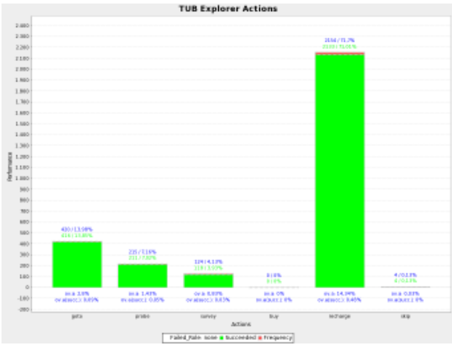


Figure 1270: TUB vs. UFSC – Simulation 3 - TUB Explorer Actions.

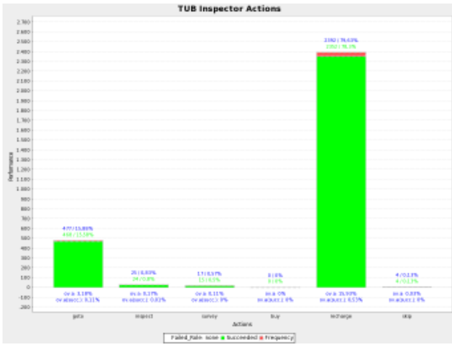


Figure 1271: TUB vs. UFSC – Simulation 3 - TUB Inspector Actions.

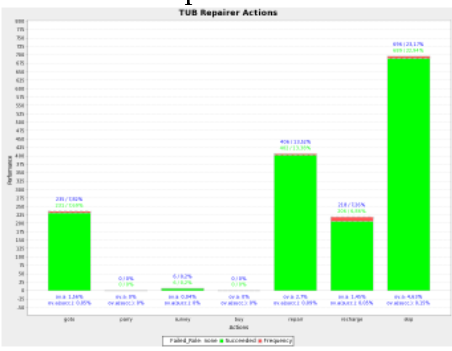


Figure 1272: TUB vs. UFSC – Simulation 3 - TUB Repairer Actions.

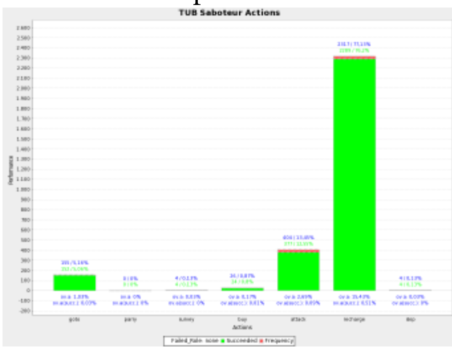


Figure 1273: TUB vs. UFSC – Simulation 3 - TUB Saboteur Actions.

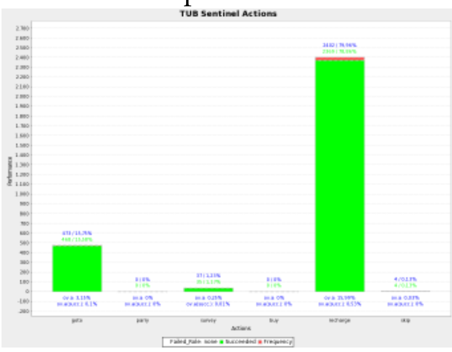


Figure 1274: TUB vs. UFSC – Simulation 3 - TUB Sentinel Actions.

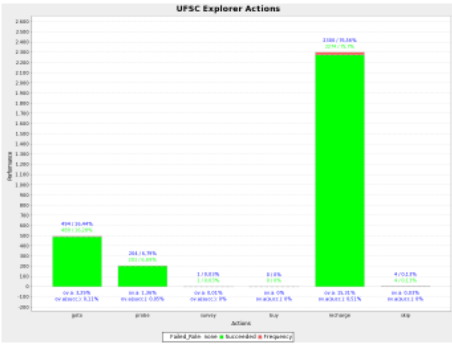


Figure 1275: TUB vs. UFSC – Simulation 3 - UFSC Explorer Actions.

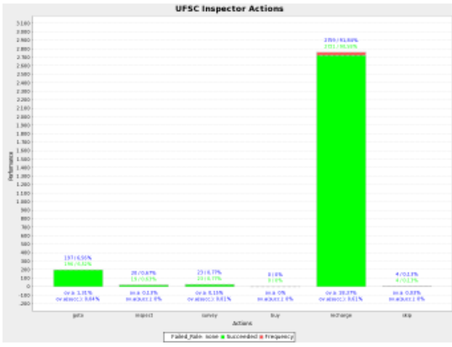


Figure 1276: TUB vs. UFSC – Simulation 3 - UFSC Inspector Actions.

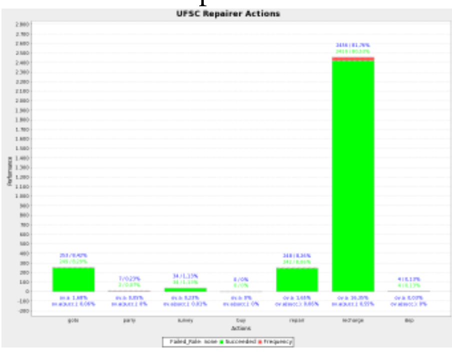


Figure 1277: TUB vs. UFSC – Simulation 3 - UFSC Repairer Actions.

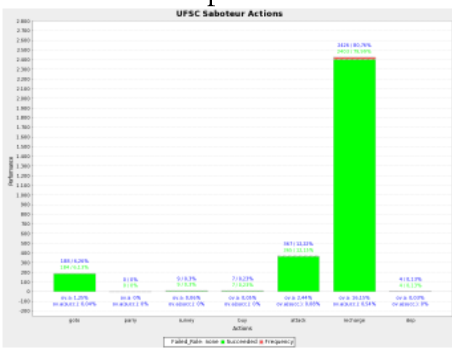


Figure 1278: TUB vs. UFSC – Simulation 3 - UFSC Saboteur Actions.

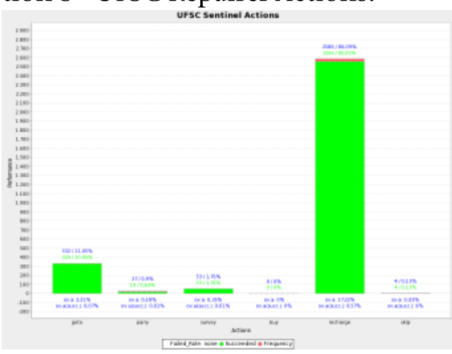


Figure 1279: TUB vs. UFSC – Simulation 3 - UFSC Sentinel Actions.

67 TUB vs. USP – Simulation 1

67.1 Scores, Zone Stability and Achievements

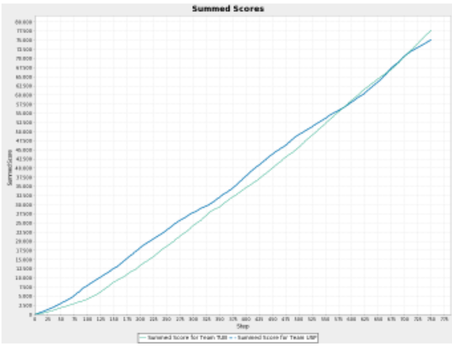


Figure 1280: Summed scores.

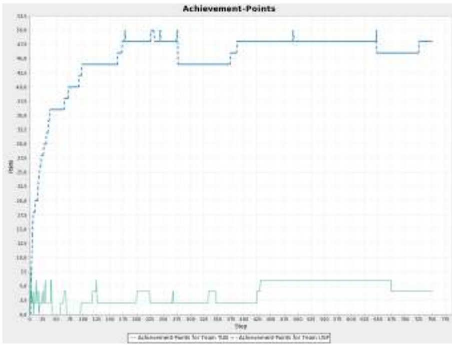


Figure 1281: Achievement points.

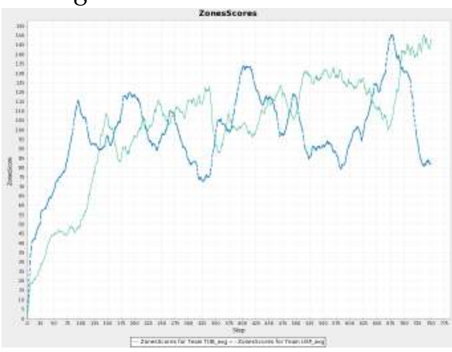


Figure 1282: Zones scores.

Figure 1283: Zones scores and achievement points.

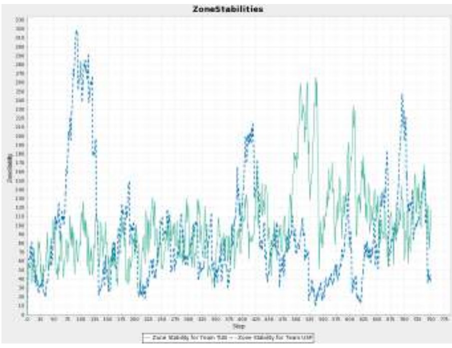


Figure 1284: Zone Stabilities.

Step	TUB	USP
1	surveyed10, surveyed80, surveyed40, surveyed20	
2	area10	surveyed10, surveyed40, surveyed20
3	proved5	area10
4		area20, surveyed80, proved5
5		area40
6	surveyed160	proved10
8	proved10	
10		attacked5
11	area20, inspected5	
13	attacked5	
15	inspected10	inspected5
16		surveyed160
18	attacked10	proved20
21		attacked10
23	proved20	
24	attacked20	
26		parried5
28	area40, inspected20	
31		attacked20
34		parried10
37		inspected10
39	attacked40, surveyed320	
57	attacked80	
63	proved40	
64		attacked40
72		area80
92		parried20
95	attacked160	
97		proved40
117	area80	
124	proved80	
164		attacked80
172		parried40
177		surveyed320
199	attacked320	
226		proved80
242		inspected20
266	area160	
274		parried80
333	proved160	
374		attacked160
386		parried160
423	surveyed640	
430	attacked640	
491		proved160
646		surveyed640
725		attacked320

Figure 1285: Achievements.

67.2 Stability

Reason	TUB	%	USP	%
failed away	1	0,01	327	2,18
failed parried	446	2,97		
failed wrong param	12	0,08		
failed random	144	0,96	160	1,07
failed resources	1	0,01	42	0,28
failed	4	0,03		
failed attacked	70	0,47	159	1,06
noAction	4	0,03		

Figure 1286: Failed actions.

67.3 Achievements

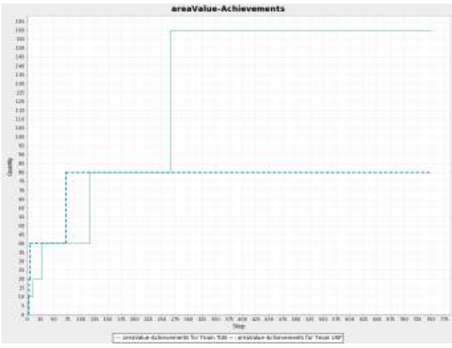


Figure 1287: areaValueAchievements.

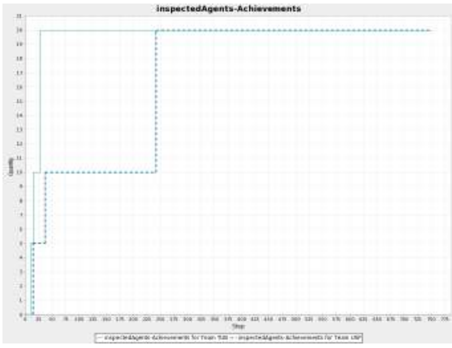


Figure 1288: inspectedAgentsAchievements.

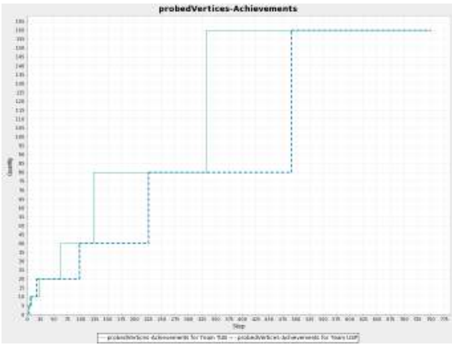


Figure 1289: probedVerticesAchievements.

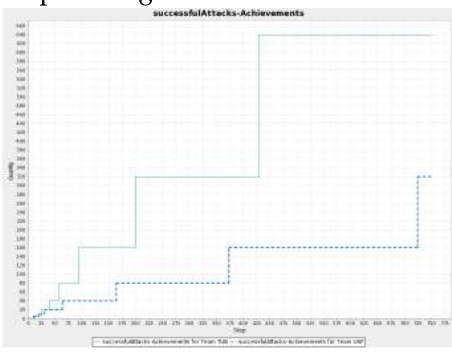


Figure 1290: successfulAttacksAchievements.

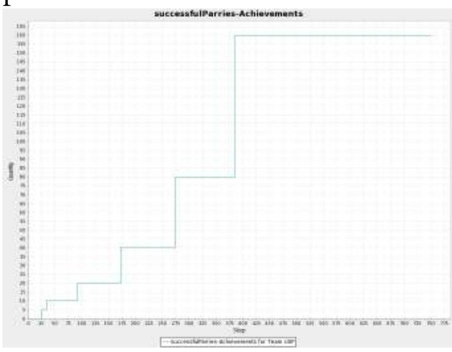


Figure 1291: successfulParriesAchievements.

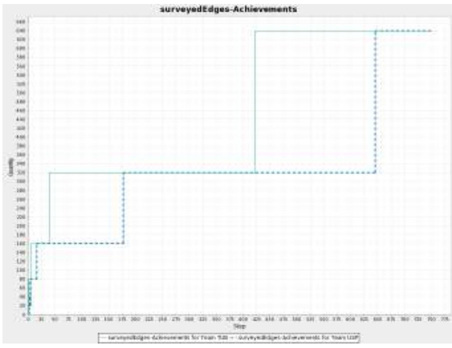


Figure 1292: surveyedEdgesAchievements.

67.4 Actions per Role

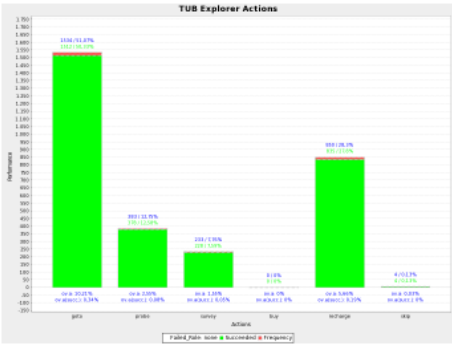


Figure 1293: TUB vs. USP – Simulation 1 - TUB Explorer Actions.

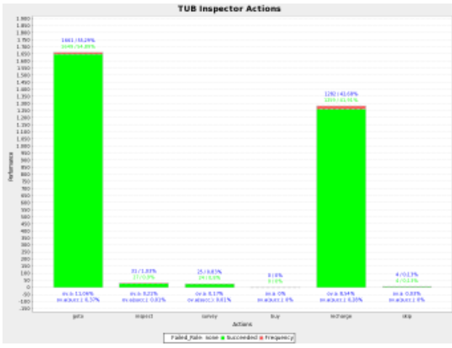


Figure 1294: TUB vs. USP – Simulation 1 - TUB Inspector Actions.

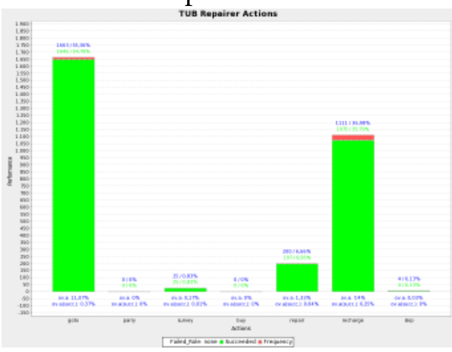


Figure 1295: TUB vs. USP – Simulation 1 - TUB Repairer Actions.

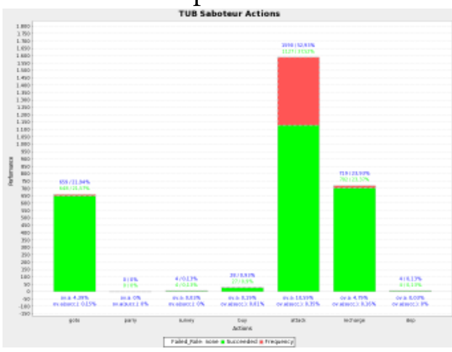


Figure 1296: TUB vs. USP – Simulation 1 - TUB Saboteur Actions.

Figure 1297: TUB vs. USP – Simulation 1 - TUB Sentinel Actions.

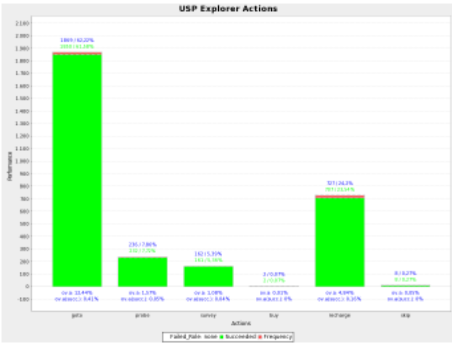


Figure 1298: TUB vs. USP – Simulation 1 - USP Explorer Actions.

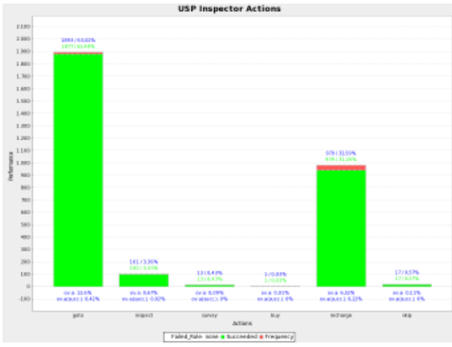


Figure 1299: TUB vs. USP – Simulation 1 - USP Inspector Actions.

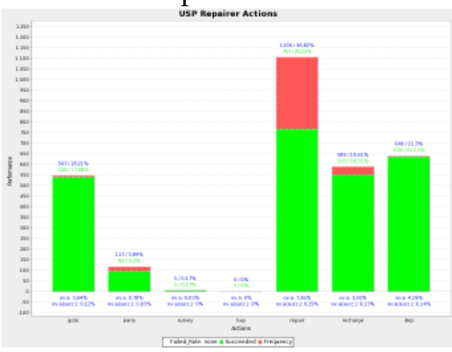


Figure 1300: TUB vs. USP – Simulation 1 - USP Repairer Actions.

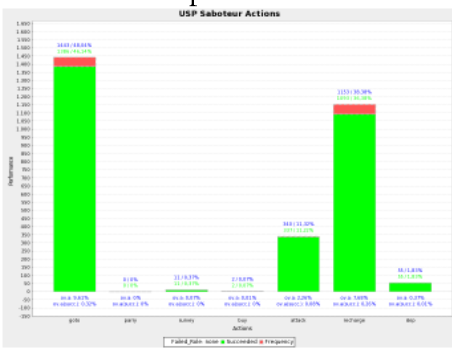


Figure 1301: TUB vs. USP – Simulation 1 - USP Saboteur Actions.

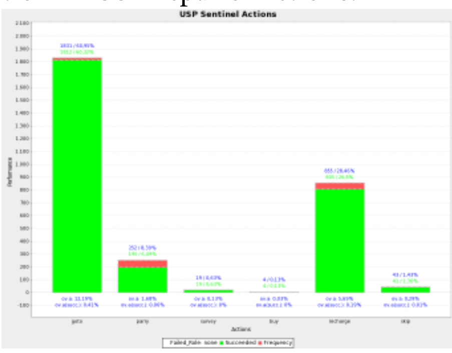


Figure 1302: TUB vs. USP – Simulation 1 - USP Sentinel Actions.

68 TUB vs. USP – Simulation 2

68.1 Scores, Zone Stability and Achievements

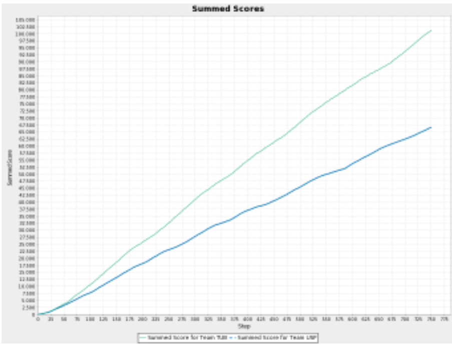


Figure 1303: Summed scores.

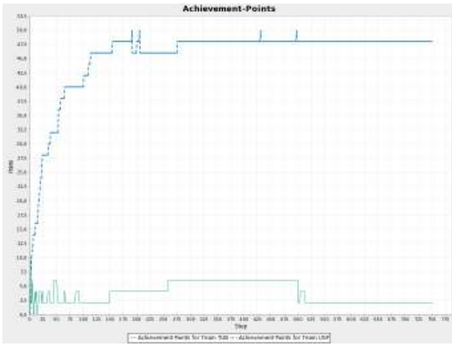


Figure 1304: Achievement points.

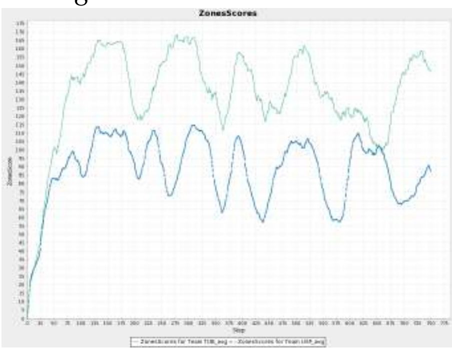


Figure 1305: Zones scores.

Figure 1306: Zones scores and achievement points.

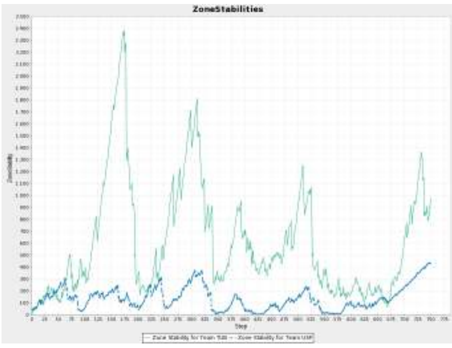


Figure 1307: Zone Stabilities.

Step	TUB	USP
1	surveyed40, surveyed10, surveyed20, area10, surveyed80	surveyed10, area10
2		surveyed40, surveyed20
3		surveyed80
4	proved5, inspected5	proved5
6		proved10
8	area20	
9	attacked5	
10		area20
11	area40	
12	inspected10, surveyed160	
15	proved10	proved20
16		surveyed160
17	attacked10	
18		parried5
19		inspected5
22		attacked5
23	proved20	area40
24	area80	
33	attacked20	
34		attacked10
38		proved40
44	attacked40	
45	proved40	
53		inspected10, attacked20
57		parried10
64	area160	
65		parried20
85	attacked80	
100		parried40
109		area80
113		surveyed320
149	attacked160	
154		attacked40
190		proved80
199		inspected20
204		parried80
257	attacked320	
275		attacked80
430		parried160
497		attacked160
504	attacked640	

Figure 1308: Achievements.

68.2 Stability

Reason	TUB	%	USP	%
failed away	1	0,01	283	1,89
failed parried	383	2,55		
failed random	120	0,8	172	1,15
failed			272	1,81
failed resources	3	0,02	45	0,3
failed attacked	47	0,31	171	1,14
noAction			275	1,83

Figure 1309: Failed actions.

68.3 Achievements

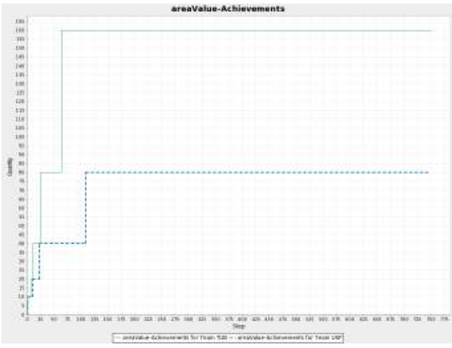


Figure 1310: areaValueAchievements.

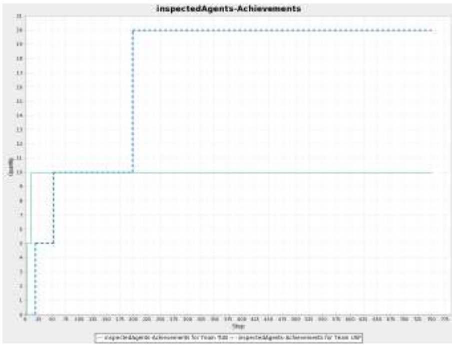


Figure 1311: inspectedAgentsAchievements.

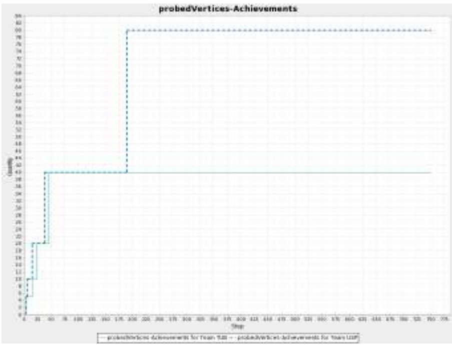


Figure 1312: probedVerticesAchievements.

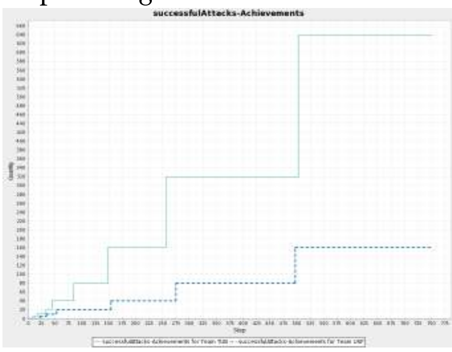


Figure 1313: successfulAttacksAchievements.

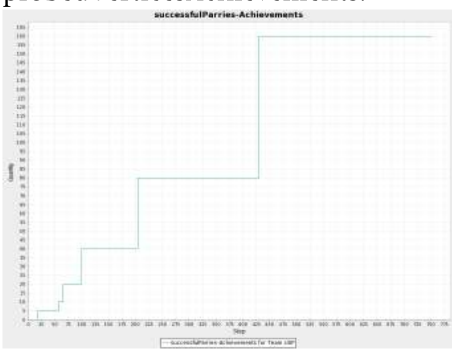


Figure 1314: successfulParriesAchievements.

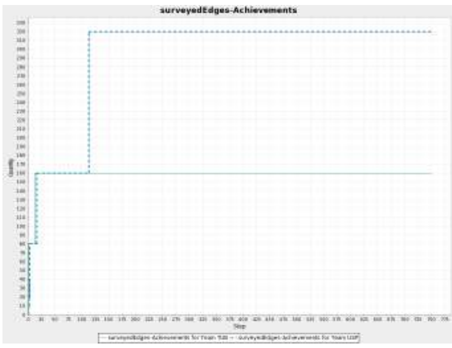


Figure 1315: surveyedEdgesAchievements.

68.4 Actions per Role

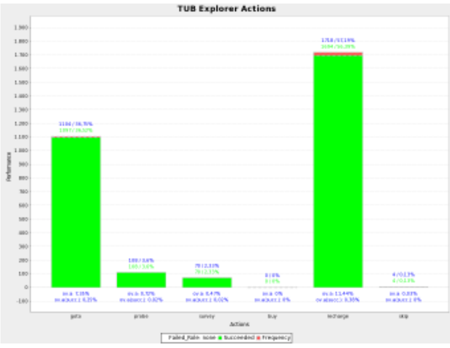


Figure 1316: TUB vs. USP – Simulation 2 - TUB Explorer Actions.

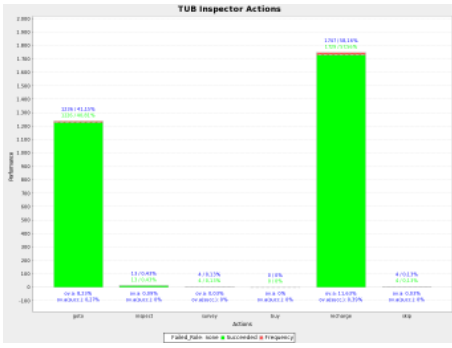


Figure 1317: TUB vs. USP – Simulation 2 - TUB Inspector Actions.

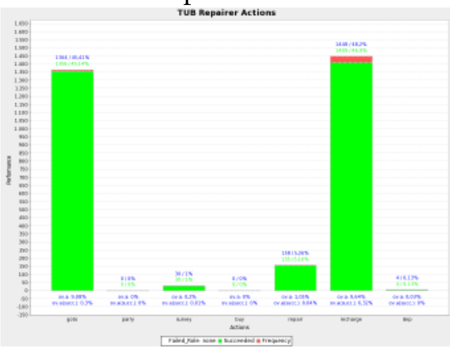


Figure 1318: TUB vs. USP – Simulation 2 - TUB Repairer Actions.

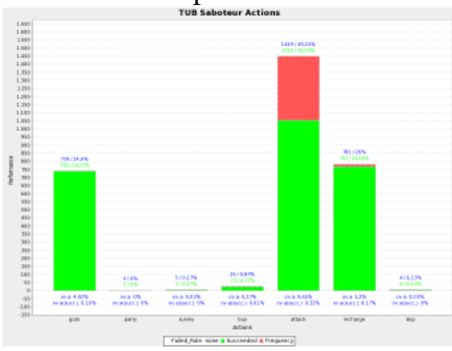


Figure 1319: TUB vs. USP – Simulation 2 - TUB Saboteur Actions.

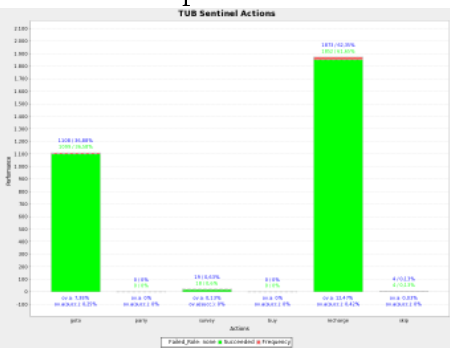


Figure 1320: TUB vs. USP – Simulation 2 - TUB Sentinel Actions.

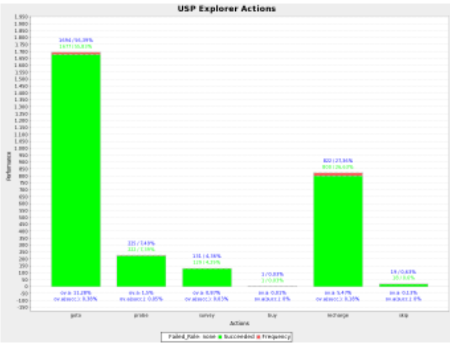


Figure 1321: TUB vs. USP – Simulation 2 - USP Explorer Actions.

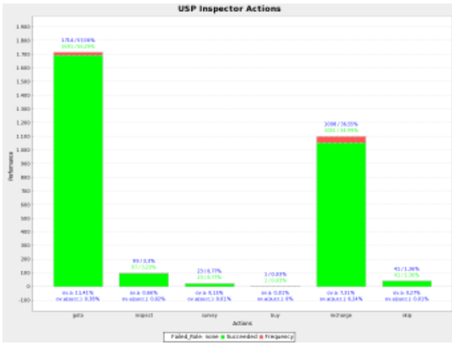


Figure 1322: TUB vs. USP – Simulation 2 - USP Inspector Actions.

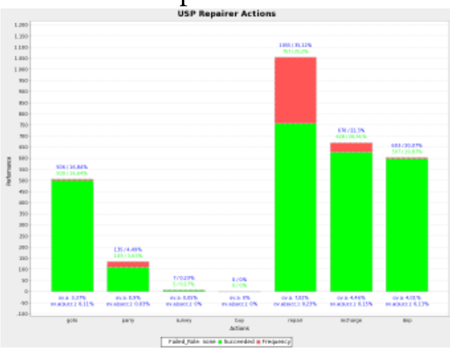


Figure 1323: TUB vs. USP – Simulation 2 - USP Repairer Actions.

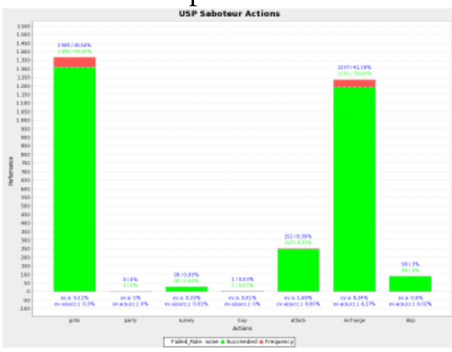


Figure 1324: TUB vs. USP – Simulation 2 - USP Saboteur Actions.

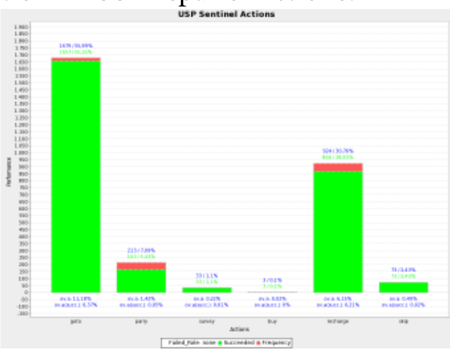


Figure 1325: TUB vs. USP – Simulation 2 - USP Sentinel Actions.

69 TUB vs. USP – Simulation 3

69.1 Scores, Zone Stability and Achievements

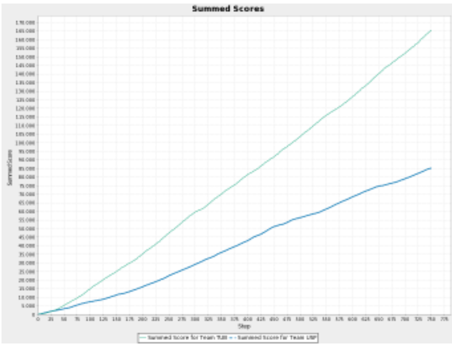


Figure 1326: Summed scores.

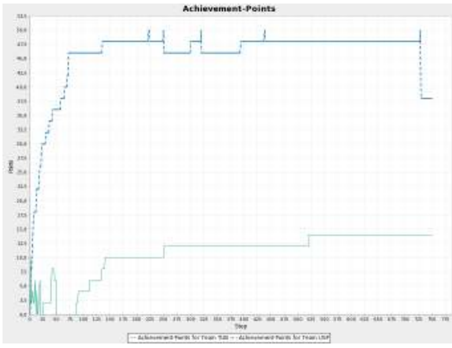


Figure 1327: Achievement points.

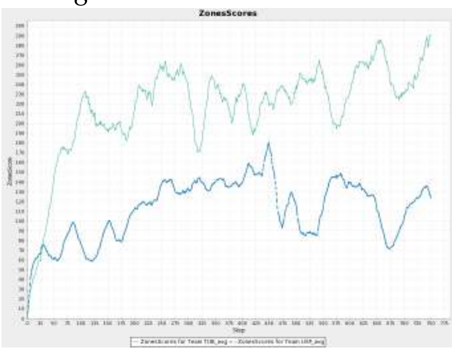


Figure 1328: Zones scores.

Figure 1329: Zones scores and achievement points.

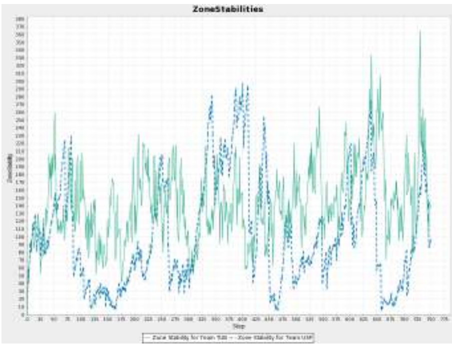


Figure 1330: Zone Stabilities.

Step	TUB	USP
1	surveyed10, surveyed80, surveyed40, area10, surveyed20	surveyed10
2		surveyed40, area10, surveyed20
4	proved5	area20, proved5
5	inspected5	surveyed80
6	area20	proved10
7		area40
9	area40, proved10	
10	inspected10	
12	surveyed160	attacked5, inspected5
13	attacked5	
16	area80	
17	attacked10	proved20
18	proved20	attacked10
21		inspected10
22		surveyed160
24	attacked20	
29		parried5
36		parried10
39	attacked40, proved40	
41	area160	
42		parried20
57		attacked20
65		area80
70		parried40
72		proved40, inspected20
86	attacked80	
90	area320	
112	proved80	
134	surveyed320	
135		attacked40
140	attacked160	
221		parried80
249		attacked80
250	attacked320	
299		surveyed320
319		proved80
393		parried160
437		attacked160
438		area160
520	attacked640	
728		parried320

Figure 1331: Achievements.

69.2 Stability

Reason	TUB	%	USP	%
failed away			305	2,03
failed parried	468	3,12		
failed wrong param	2	0,01		
failed random	129	0,86	150	1
failed			147	0,98
failed resources	4	0,03	76	0,51
failed attacked	40	0,27	183	1,22
noAction			148	0,99
failed status			1	0,01

Figure 1332: Failed actions.

69.3 Achievements

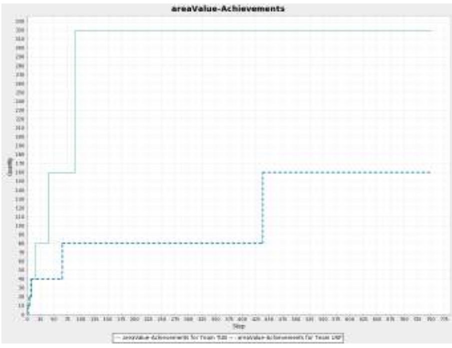


Figure 1333: areaValueAchievements.

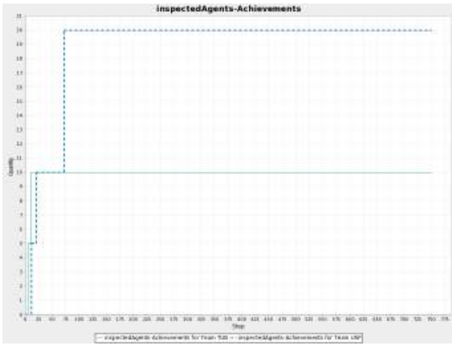


Figure 1334: inspectedAgentsAchievements.

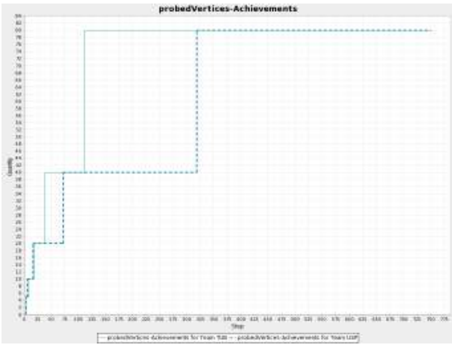


Figure 1335: probedVerticesAchievements.

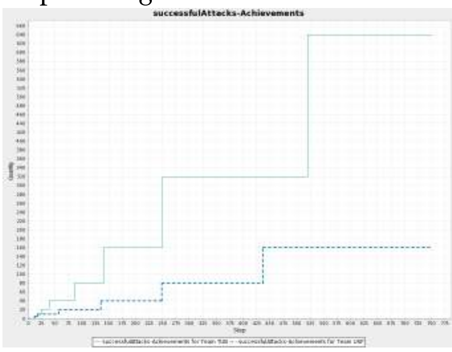


Figure 1336: successfulAttacksAchievements.

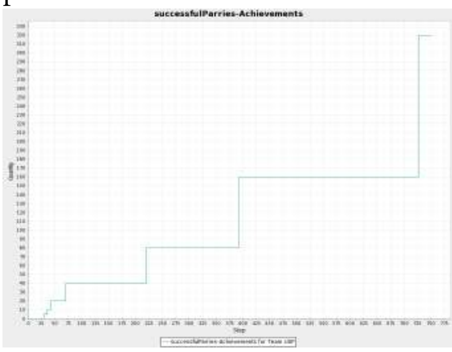


Figure 1337: successfulParriesAchievements.

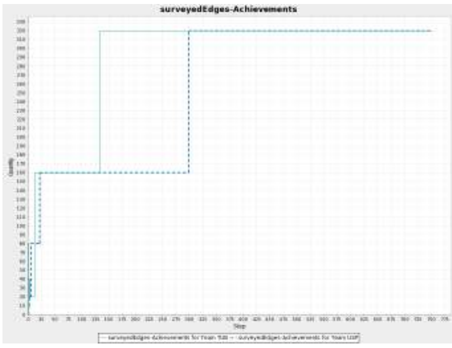


Figure 1338: surveyedEdgesAchievements.

69.4 Actions per Role

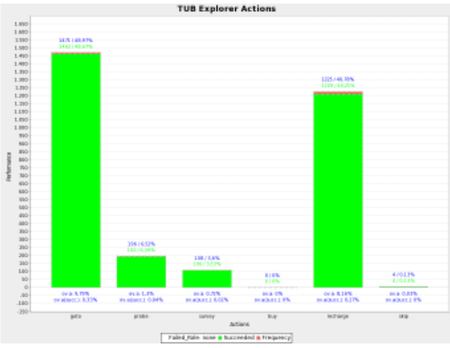


Figure 1339: TUB vs. USP – Simulation 3 - TUB Explorer Actions.

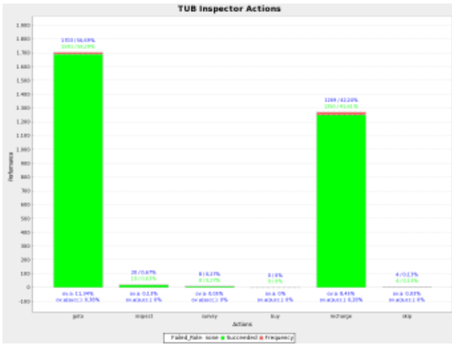


Figure 1340: TUB vs. USP – Simulation 3 - TUB Inspector Actions.

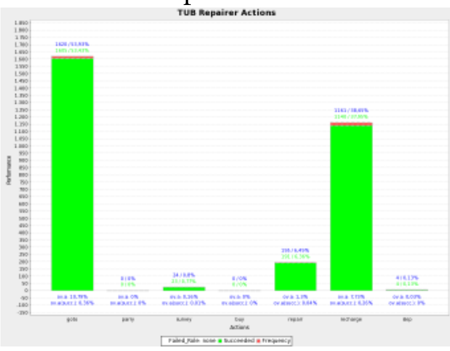


Figure 1341: TUB vs. USP – Simulation 3 - TUB Repairer Actions.

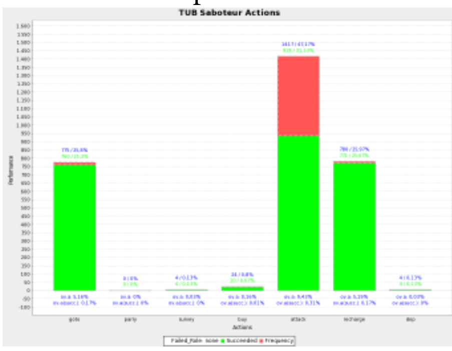


Figure 1342: TUB vs. USP – Simulation 3 - TUB Saboteur Actions.

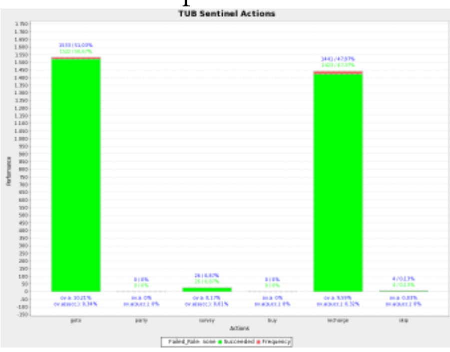


Figure 1343: TUB vs. USP – Simulation 3 - TUB Sentinel Actions.

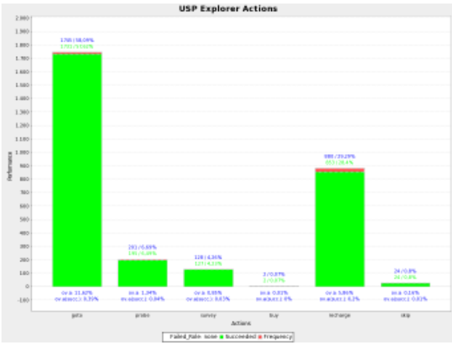


Figure 1344: TUB vs. USP – Simulation 3 - USP Explorer Actions.

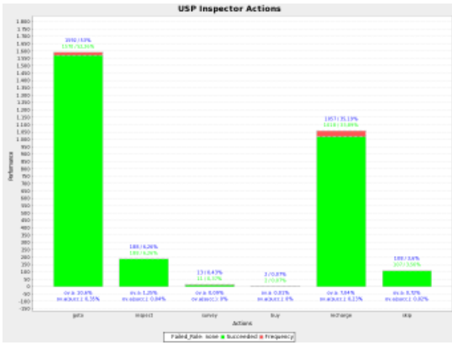


Figure 1345: TUB vs. USP – Simulation 3 - USP Inspector Actions.

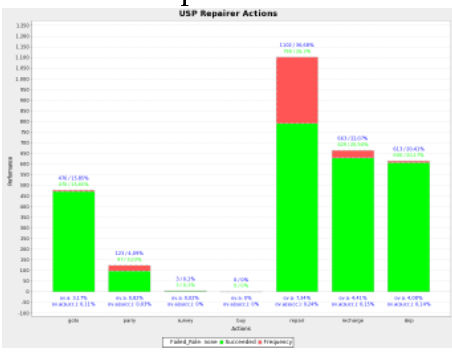


Figure 1346: TUB vs. USP – Simulation 3 - USP Repairer Actions.

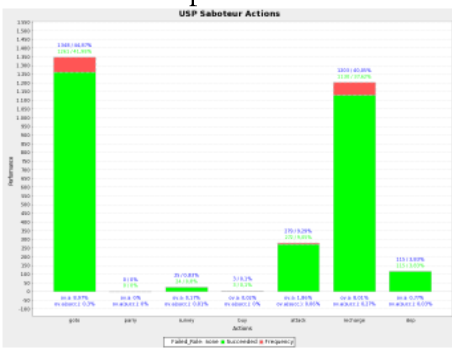


Figure 1347: TUB vs. USP – Simulation 3 - USP Saboteur Actions.

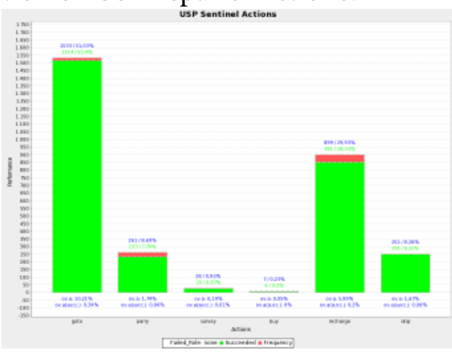


Figure 1348: TUB vs. USP – Simulation 3 - USP Sentinel Actions.

70 USP vs. Python-DTU – Simulation 1

70.1 Scores, Zone Stability and Achievements

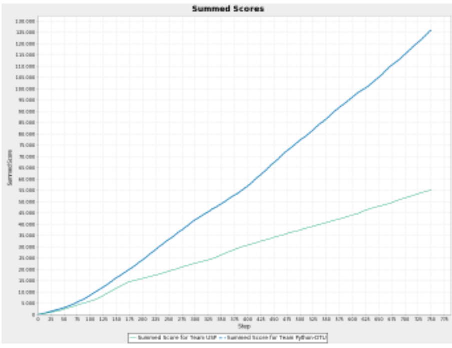


Figure 1349: Summed scores.

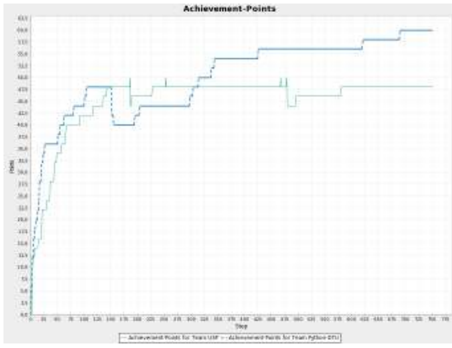


Figure 1350: Achievement points.

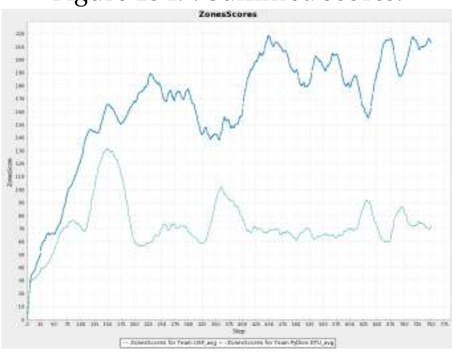


Figure 1351: Zones scores.

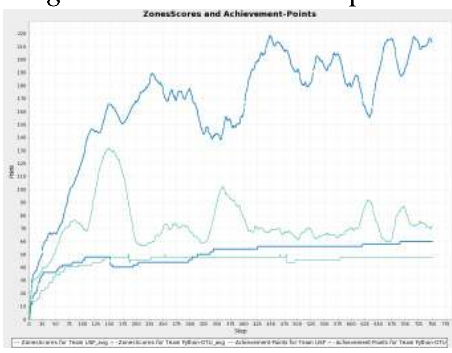


Figure 1352: Zones scores and achievement points.

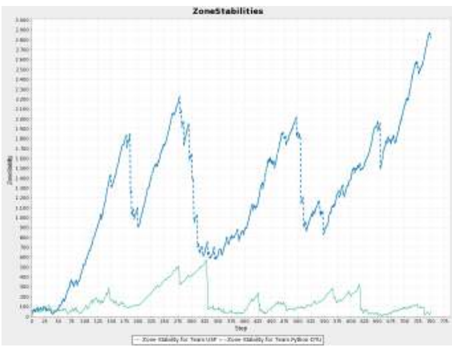


Figure 1353: Zone Stabilities.

Step	USP	Python-DTU
1		surveyed10, surveyed40, surveyed20
2	surveyed10, surveyed40, surveyed20	
3	surveyed80, area10	surveyed80, area10, proved5
4	proved5	
5		proved10, surveyed160
7	proved10	attacked5
9		proved20
13		attacked10
14	proved20	
15		inspected5
16		inspected10, area20
19		area40
20	area20	surveyed320
21	surveyed160, attacked5	
23		proved40
26		attacked20
29	parried5	
36	attacked10, inspected5	
43	proved40	
44	attacked20	
48	parried10	
51		proved80
55		attacked40
57	area40	
62		inspected20
65	parried20	
66	inspected10	
80		surveyed640
92	parried40	
100		proved160
104		attacked80
117	area80	
135	parried80	
142	attacked40	
185	surveyed320	
188	proved80	
194		attacked160
203		area80
227	parried160	
252	attacked80	
297		parried5
303		parried10
313		parried20
337		attacked320
343		parried40
425		parried80
467	proved160	
478	parried320	
495	attacked160	
579	surveyed640	
620		parried160
690		attacked640

Figure 1354: Achievements.

70.2 Stability

Reason	USP	%	Python-DTU	%
failed away	266	1,77		
failed parried	211	1,41	498	3,32
failed random	148	0,99	154	1,03
failed resources	25	0,17		
failed attacked	218	1,45	77	0,51

Figure 1355: Failed actions.

70.3 Achievements

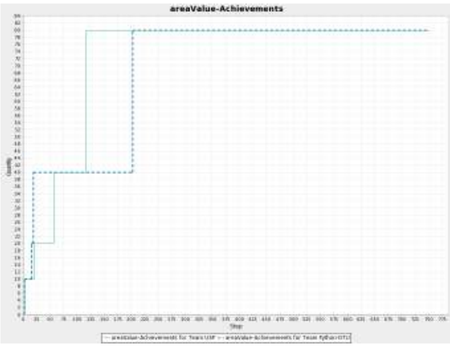


Figure 1356: areaValueAchievements.

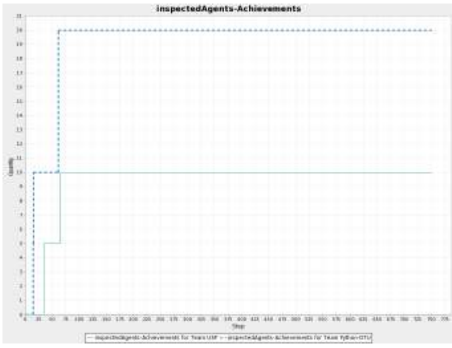


Figure 1357: inspectedAgentsAchievements.

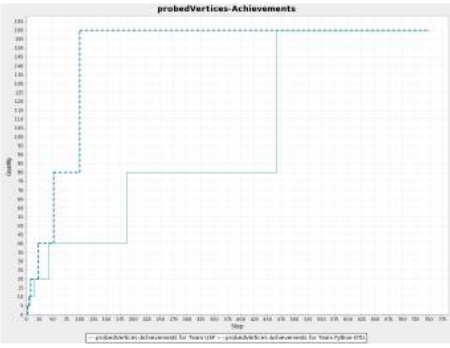


Figure 1358: probedVerticesAchievements.

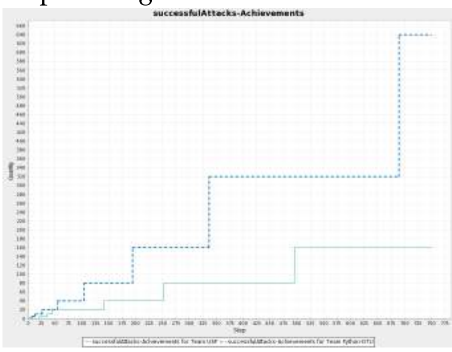


Figure 1359: successfulAttacksAchievements.

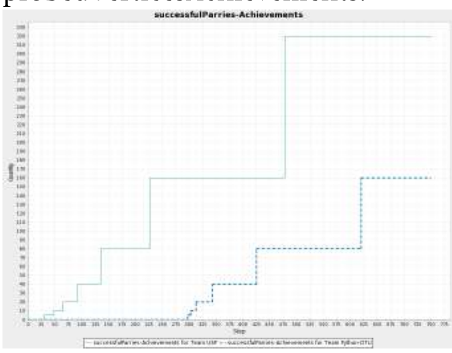


Figure 1360: successfulParriesAchievements.

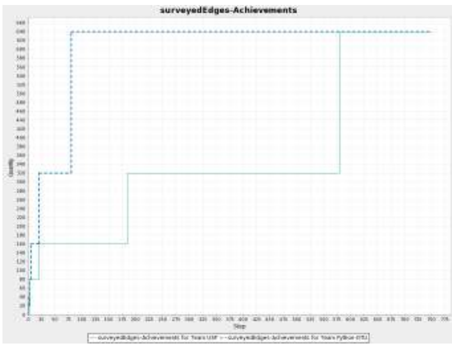


Figure 1361: surveyedEdgesAchievements.

70.4 Actions per Role

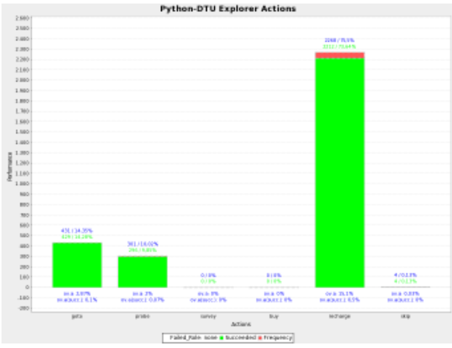


Figure 1362: USP vs. Python-DTU – Simulation 1 - Python-DTU Explorer Actions.

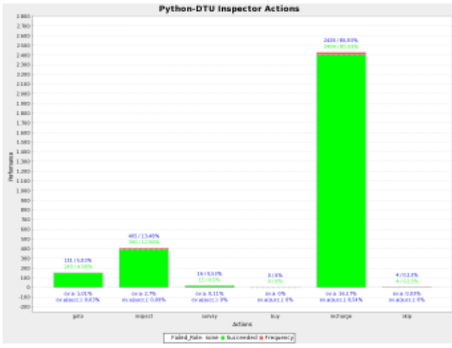


Figure 1363: USP vs. Python-DTU – Simulation 1 - Python-DTU Inspector Actions.

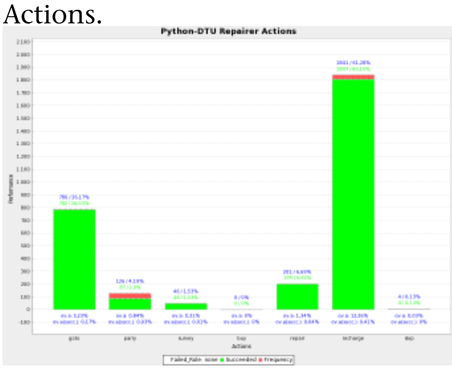


Figure 1364: USP vs. Python-DTU – Simulation 1 - Python-DTU Repairer Actions.

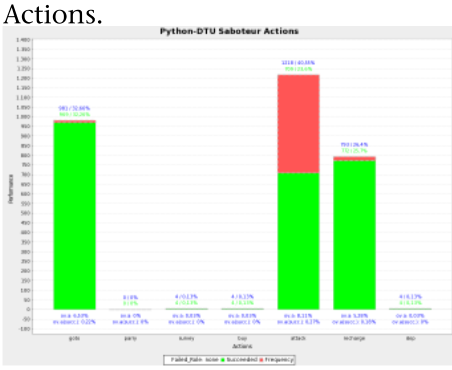


Figure 1365: USP vs. Python-DTU – Simulation 1 - Python-DTU Saboteur Actions.

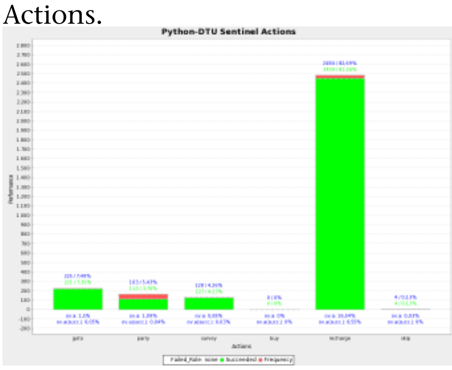


Figure 1366: USP vs. Python-DTU – Simulation 1 - Python-DTU Sentinel Actions.

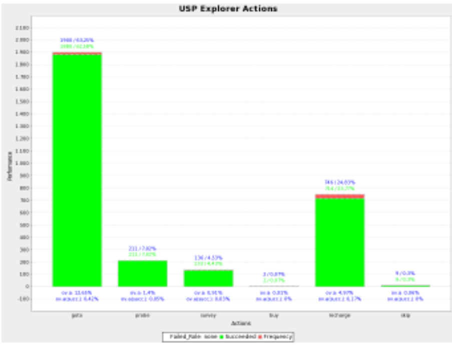


Figure 1367: USP vs. Python-DTU – Simulation 1 - USP Explorer Actions.

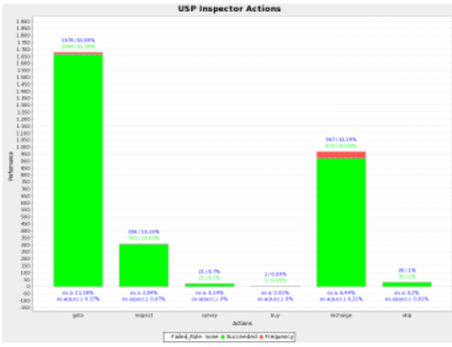


Figure 1368: USP vs. Python-DTU – Simulation 1 - USP Inspector Actions.

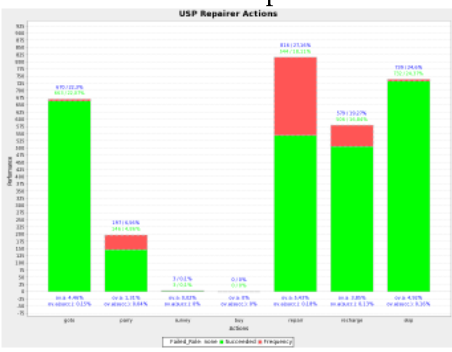


Figure 1369: USP vs. Python-DTU – Simulation 1 - USP Repairer Actions.

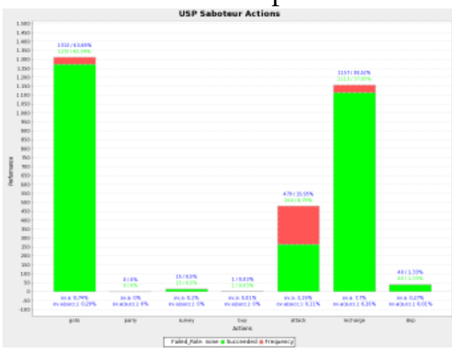


Figure 1370: USP vs. Python-DTU – Simulation 1 - USP Saboteur Actions.

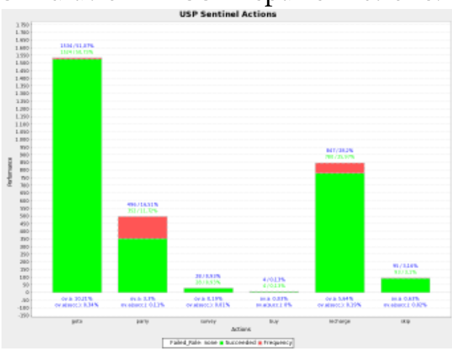


Figure 1371: USP vs. Python-DTU – Simulation 1 - USP Sentinel Actions.

71 USP vs. Python-DTU – Simulation 2

71.1 Scores, Zone Stability and Achievements

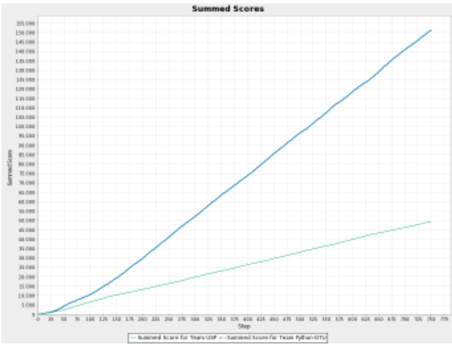


Figure 1372: Summed scores.

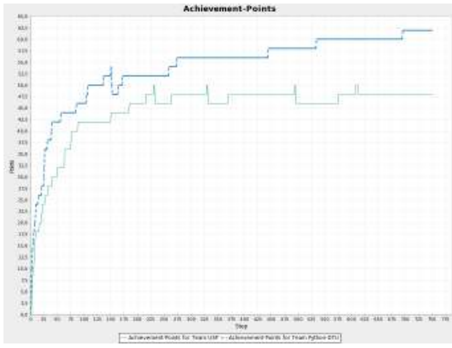


Figure 1373: Achievement points.

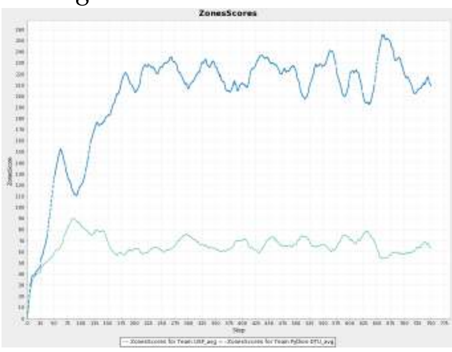


Figure 1374: Zones scores.

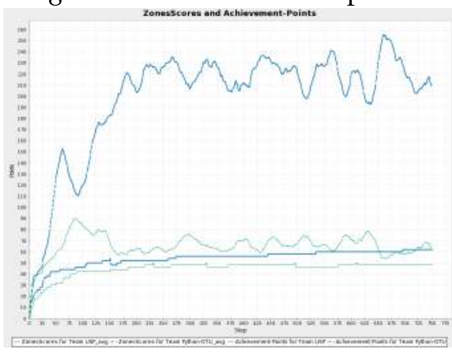


Figure 1375: Zones scores and achievement points.

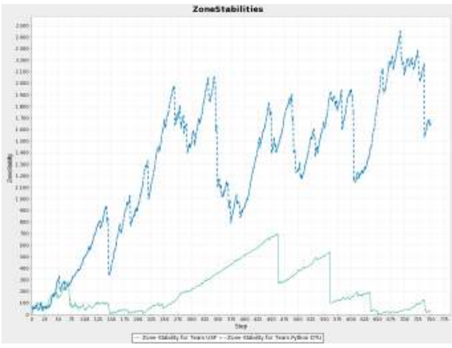


Figure 1376: Zone Stabilities.

Step	USP	Python-DTU
1	area10	surveyed10, area20, surveyed40, area10, surveyed20
2	surveyed10, surveyed20	
3	surveyed40	surveyed80, proved5
4	proved5	inspected5
5		proved10
6	area20	
7	proved10	inspected10
8	surveyed80	
9	area40	proved20, surveyed160
14		attacked5
16	proved20	
19	attacked5	
20		attacked10
22	inspected5	
24		surveyed320
25		inspected20
26		proved40, attacked20
27	surveyed160	
31		area40
32	attacked10	
38		area80
39		attacked40
40	parried5	
50	parried10	
56		proved80
63	inspected10, attacked20	
75	parried20	
77	proved40	
85		attacked80
88	attacked40	
104		proved160
107		surveyed640
136		parried5
150	parried40	attacked160
151		parried10
163		area160
171		parried20
184	attacked80	
215	parried80	
230	surveyed320	
258		parried40
263	proved80	
273		attacked320
329	parried160	
369	attacked160	
444		parried80
493	inspected20	
534		attacked640
575	proved160	
607	parried320	
695		parried160

Figure 1377: Achievements.

71.2 Stability

Reason	USP	%	Python-DTU	%
failed away	395	2,63		
failed parried	196	1,31	403	2,69
failed random	144	0,96	146	0,97
failed resources	40	0,27		
failed attacked	247	1,65	48	0,32
failed status	1	0,01		

Figure 1378: Failed actions.

71.3 Achievements

Figure 1379: areaValueAchievements.

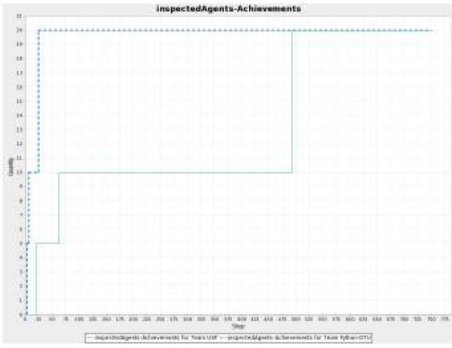


Figure 1380: inspectedAgentsAchievements.

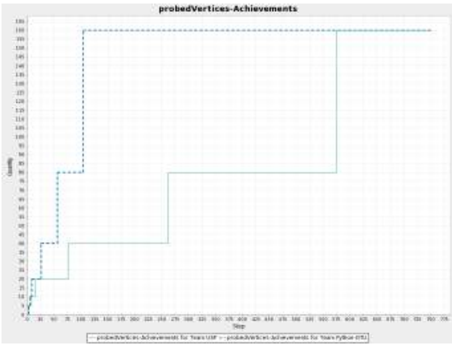


Figure 1381: probedVerticesAchievements.

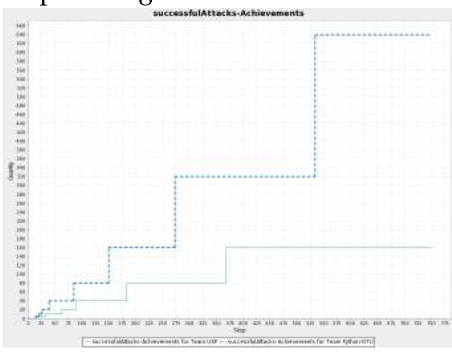


Figure 1382: successfulAttacksAchievements.

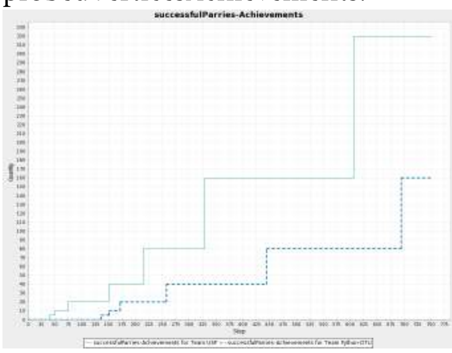


Figure 1383: successfulParriesAchievements.

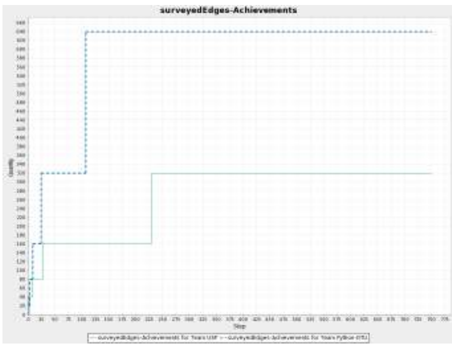


Figure 1384: surveyedEdgesAchievements.

71.4 Actions per Role

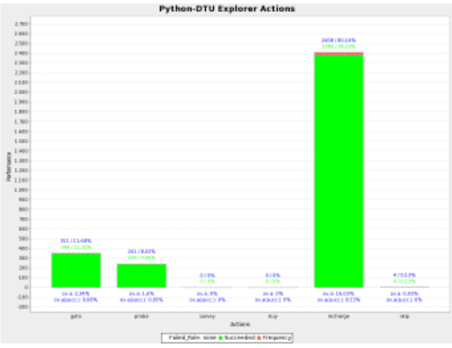


Figure 1385: USP vs. Python-DTU – Simulation 2 - Python-DTU Explorer Actions.

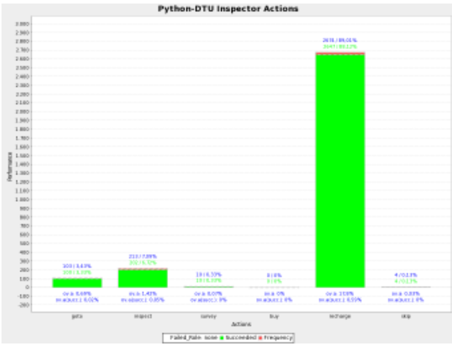


Figure 1386: USP vs. Python-DTU – Simulation 2 - Python-DTU Inspector Actions.

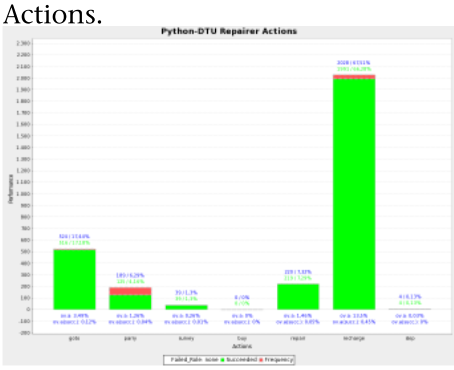


Figure 1387: USP vs. Python-DTU – Simulation 2 - Python-DTU Repairer Actions.

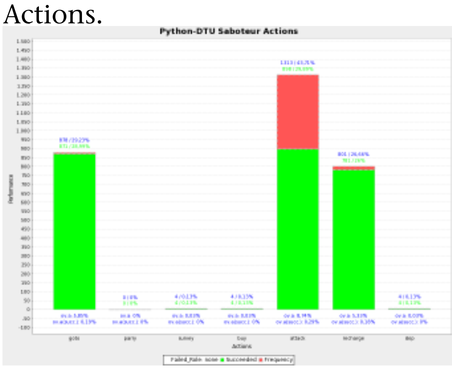


Figure 1388: USP vs. Python-DTU – Simulation 2 - Python-DTU Saboteur Actions.

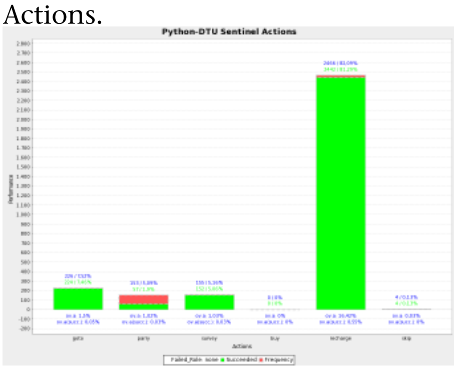


Figure 1389: USP vs. Python-DTU – Simulation 2 - Python-DTU Sentinel Actions.

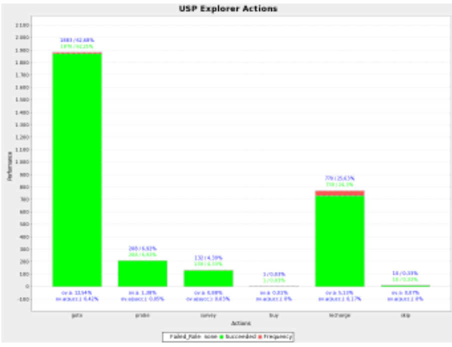


Figure 1390: USP vs. Python-DTU – Simulation 2 - USP Explorer Actions.

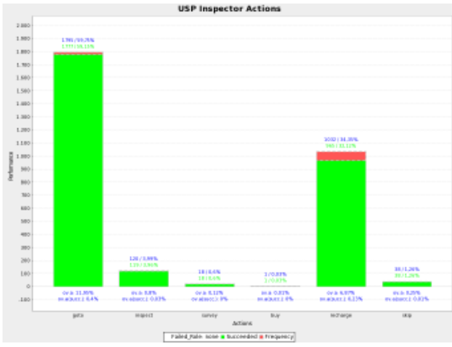


Figure 1391: USP vs. Python-DTU – Simulation 2 - USP Inspector Actions.

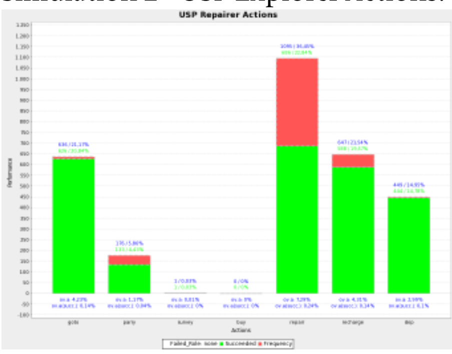


Figure 1392: USP vs. Python-DTU – Simulation 2 - USP Repairer Actions.

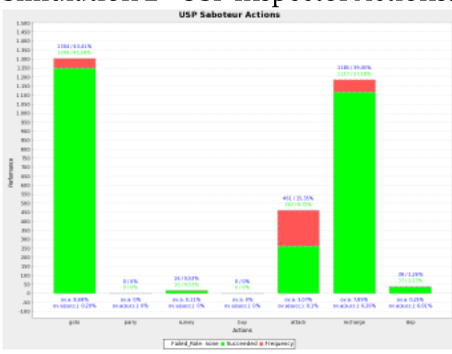


Figure 1393: USP vs. Python-DTU – Simulation 2 - USP Saboteur Actions.

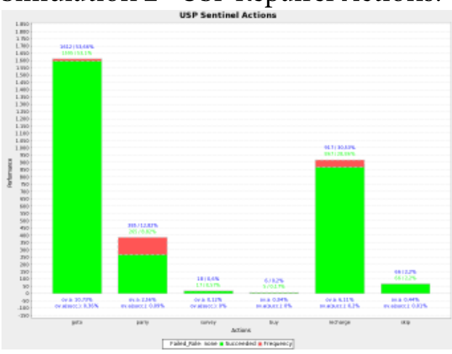


Figure 1394: USP vs. Python-DTU – Simulation 2 - USP Sentinel Actions.

72 USP vs. Python-DTU – Simulation 3

72.1 Scores, Zone Stability and Achievements

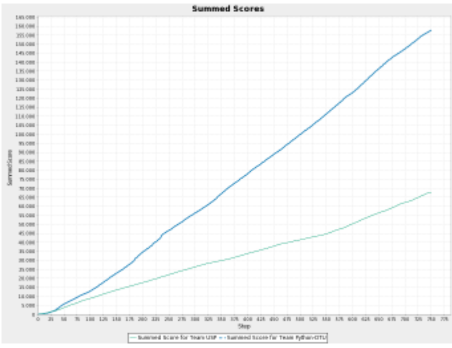


Figure 1395: Summed scores.

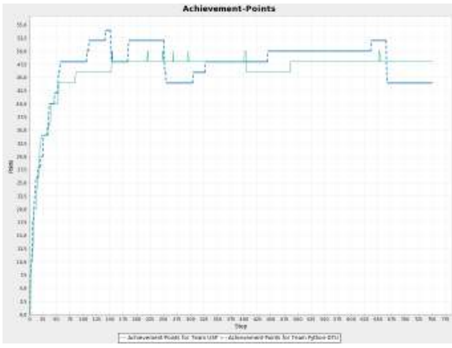


Figure 1396: Achievement points.

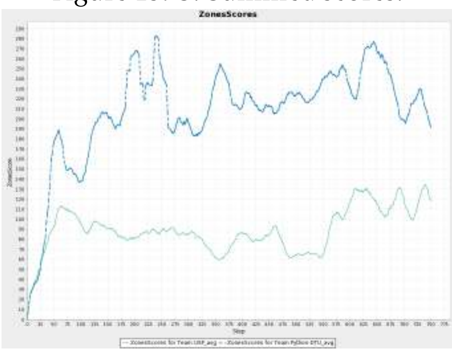


Figure 1397: Zones scores.

Figure 1398: Zones scores and achievement points.

Figure 1399: Zone Stabilities.

Step	USP	Python-DTU
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, surveyed40, area10, surveyed20
3	proved5	proved5
4		surveyed80, inspected5
5	proved10	proved10, attacked5
6	area20	
7	attacked5, inspected5	
8	surveyed80	attacked10
9		inspected10, area20
11		proved20
12	parried5, attacked10	
15	parried10	surveyed160
17	proved20	
18	inspected10, parried20	
20		area40
21	area40	
25		proved40, attacked20
33		surveyed320
35	parried40	area80
36		area160
38	surveyed160	
40	attacked20	
46		inspected20
52		attacked40
53	proved40, area80	proved80
56		parried5
85	parried80	
106		proved160
110		attacked80
141		parried10
152	attacked40	
153		parried20
183		attacked160
184		area320
219	parried160	
247	attacked80	
267	proved80	
295	surveyed320	
304		parried40
327		attacked320
400	attacked160	
444		parried80
486	parried320	
636		attacked640
651	proved160	

Figure 1400: Achievements.

72.2 Stability

Reason	USP	%	Python-DTU	%
failed away	317	2,11		
failed parried	158	1,05	468	3,12
failed random	155	1,03	154	1,03
failed resources	63	0,42		
failed attacked	220	1,47	86	0,57

Figure 1401: Failed actions.

72.3 Achievements

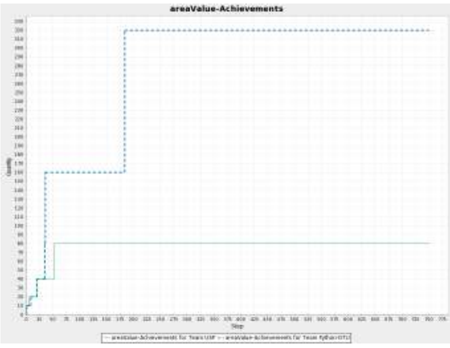


Figure 1402: areaValueAchievements.

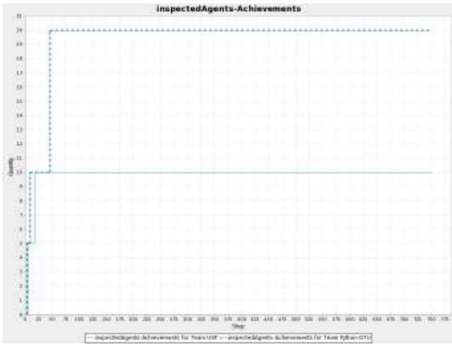


Figure 1403: inspectedAgentsAchievements.

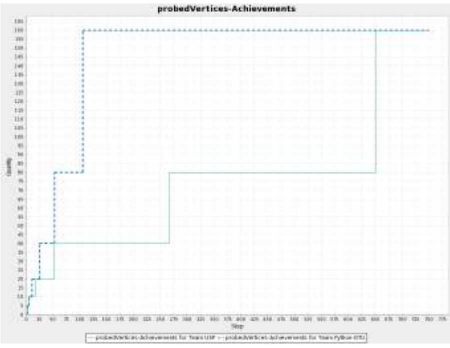


Figure 1404: probedVerticesAchievements.

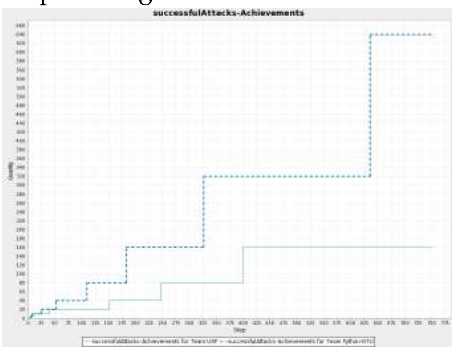


Figure 1405: successfulAttacksAchievements.

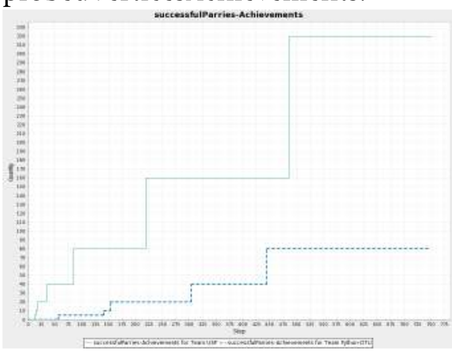


Figure 1406: successfulParriesAchievements.

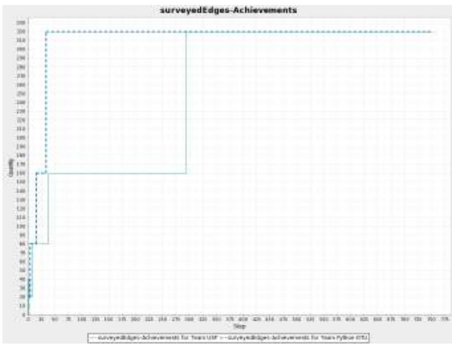


Figure 1407: surveyedEdgesAchievements.

72.4 Actions per Role

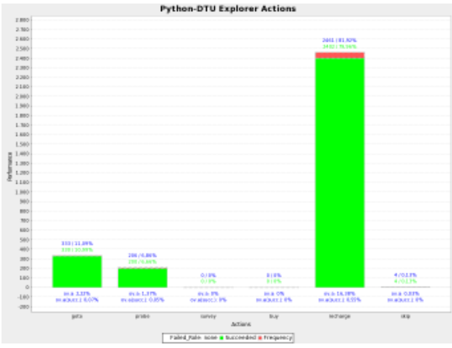


Figure 1408: USP vs. Python-DTU – Simulation 3 - Python-DTU Explorer Actions.

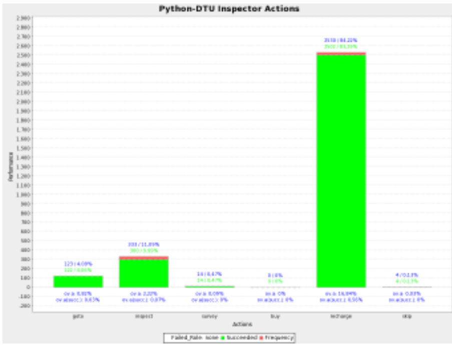


Figure 1409: USP vs. Python-DTU – Simulation 3 - Python-DTU Inspector Actions.

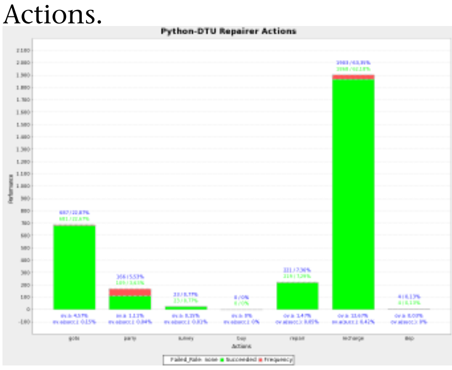


Figure 1410: USP vs. Python-DTU – Simulation 3 - Python-DTU Repairer Actions.

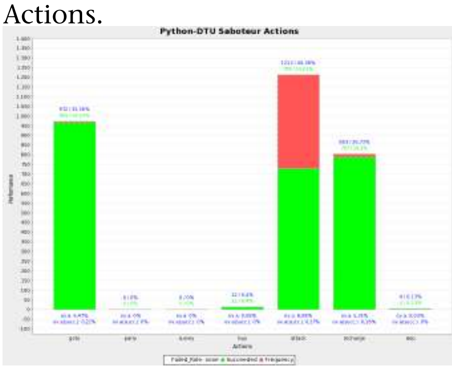


Figure 1411: USP vs. Python-DTU – Simulation 3 - Python-DTU Saboteur Actions.

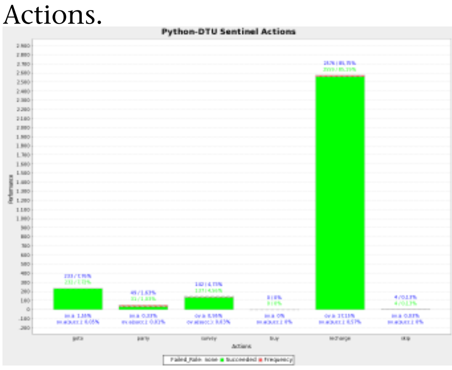


Figure 1412: USP vs. Python-DTU – Simulation 3 - Python-DTU Sentinel Actions.

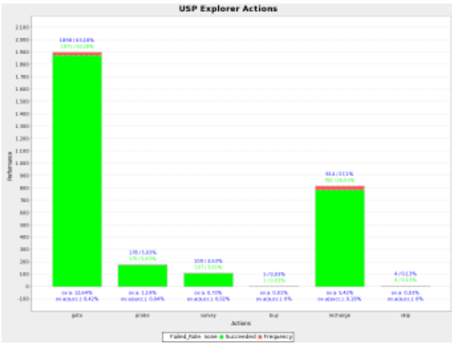


Figure 1413: USP vs. Python-DTU – Simulation 3 - USP Explorer Actions.

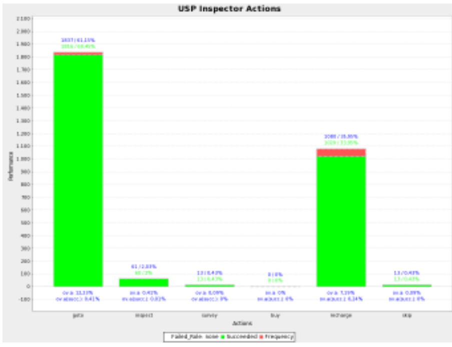


Figure 1414: USP vs. Python-DTU – Simulation 3 - USP Inspector Actions.

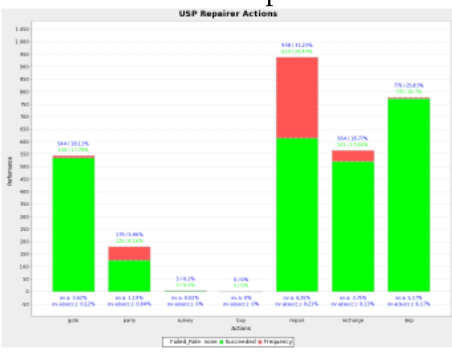


Figure 1415: USP vs. Python-DTU – Simulation 3 - USP Repairer Actions.

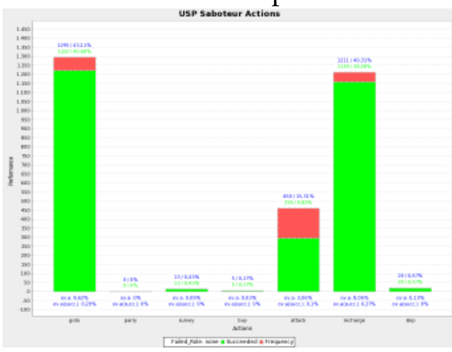


Figure 1416: USP vs. Python-DTU – Simulation 3 - USP Saboteur Actions.

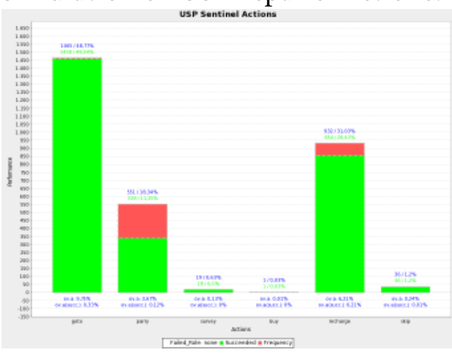


Figure 1417: USP vs. Python-DTU – Simulation 3 - USP Sentinel Actions.

73 USP vs. UFSC – Simulation 1

73.1 Scores, Zone Stability and Achievements

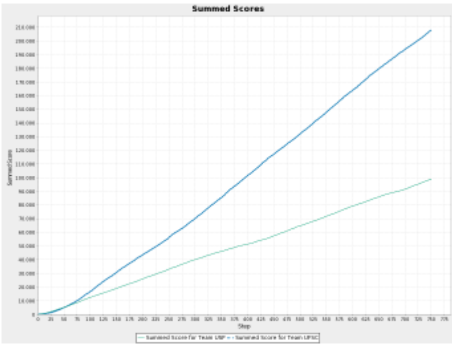


Figure 1418: Summed scores.

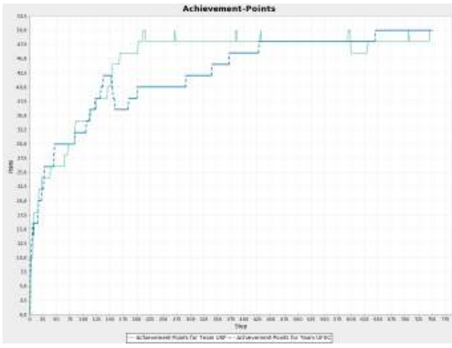


Figure 1419: Achievement points.

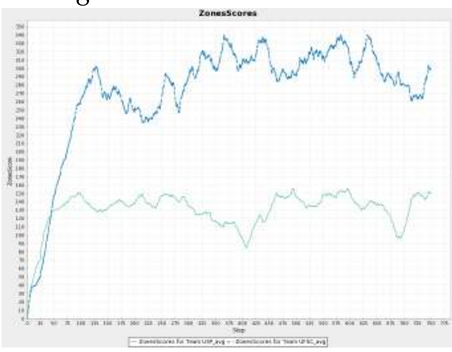


Figure 1420: Zones scores.

Figure 1421: Zones scores and achievement points.

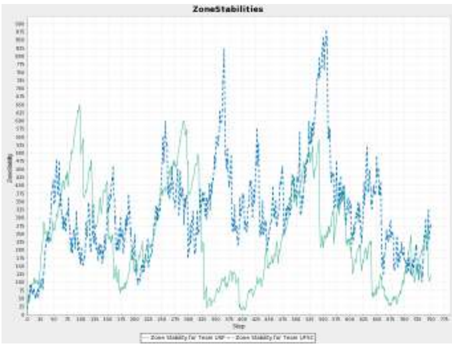


Figure 1422: Zone Stabilities.

Step	USP	UFSC
1	surveyed10, area10, surveyed20	surveyed10, area20, surveyed40, area10, surveyed20
2	surveyed40	
3	proved5	surveyed80, proved5
4		inspected5
5		proved10, surveyed160
6	area20, proved10, surveyed80	
8	area40	inspected10
9		proved20
15		surveyed320
16	proved20, area80	attacked5
22	surveyed160	proved40
26		area40
27		area80
38	proved40	
45		area160
46		proved80
64	parried5	
71	parried10	
84	attacked5	attacked10
85	inspected5	
105		attacked20
111		proved160
113	parried20	
122		area320
124	attacked10	
132		attacked40
137		inspected20
145	attacked20	
151	surveyed320	
153	proved80	
167	inspected10	
184		attacked80
200		surveyed640
202	parried40	
211	attacked40	
270	parried80	
291		attacked160
340		parried5
372		parried10
384	attacked80	
427		attacked320
430	parried160	
593	inspected20	
630	proved160	
644		parried20
706	parried320	
745	attacked160	

Figure 1423: Achievements.

73.2 Stability

Reason	USP	%	UFSC	%
failed away	241	1,61		
failed parried	31	0,21	347	2,31
failed random	144	0,96	147	0,98
failed			7	0,05
failed resources	52	0,35		
failed attacked	178	1,19	19	0,13
noAction			7	0,05

Figure 1424: Failed actions.

73.3 Achievements

Figure 1425: areaValueAchievements.

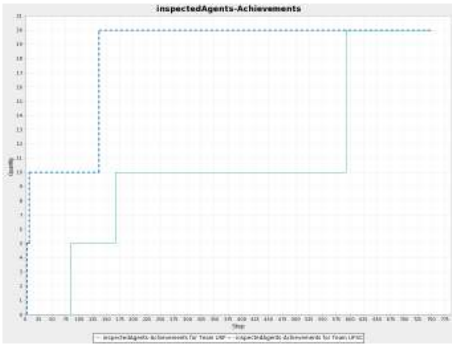


Figure 1426: inspectedAgentsAchievements.

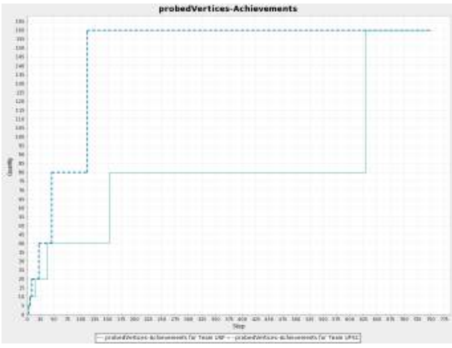


Figure 1427: probedVerticesAchievements.

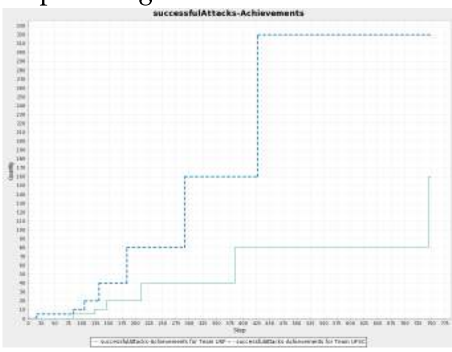


Figure 1428: successfulAttacksAchievements.

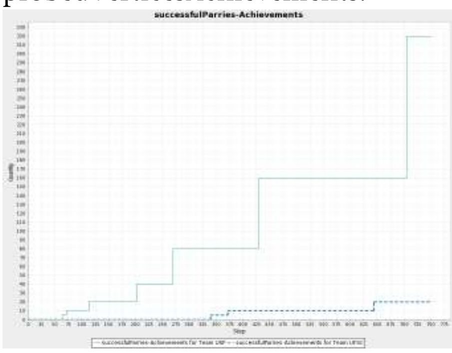


Figure 1429: successfulParriesAchievements.

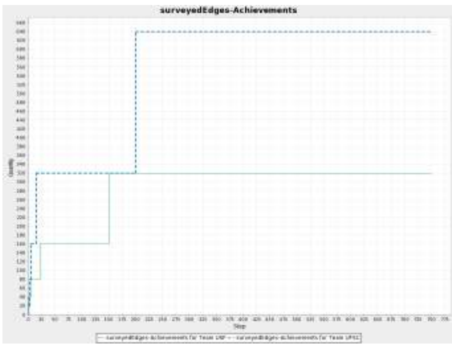


Figure 1430: surveyedEdgesAchievements.

73.4 Actions per Role

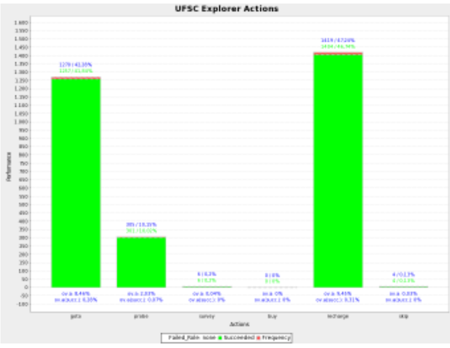


Figure 1431: USP vs. UFSC – Simulation 1 - UFSC Explorer Actions.

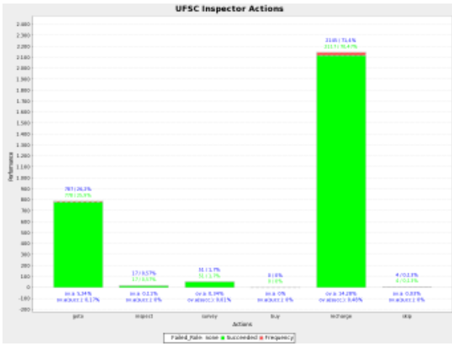


Figure 1432: USP vs. UFSC – Simulation 1 - UFSC Inspector Actions.

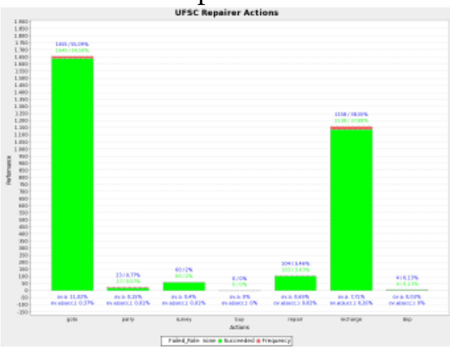


Figure 1433: USP vs. UFSC – Simulation 1 - UFSC Repairer Actions.

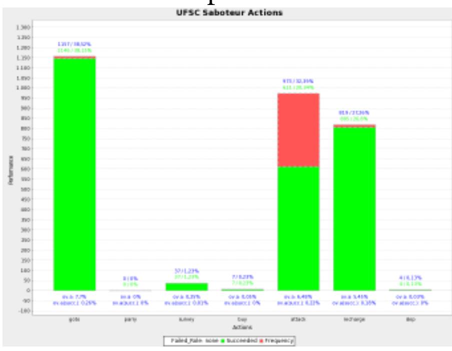


Figure 1434: USP vs. UFSC – Simulation 1 - UFSC Saboteur Actions.

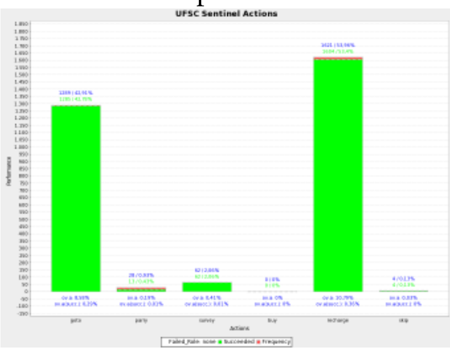


Figure 1435: USP vs. UFSC – Simulation 1 - UFSC Sentinel Actions.

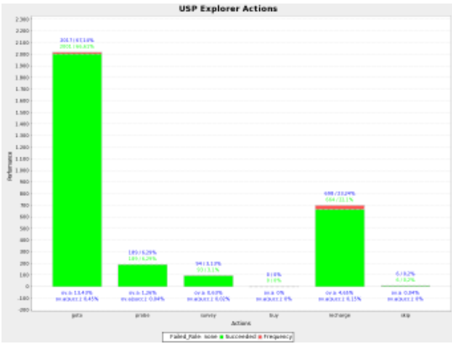


Figure 1436: USP vs. UFSC – Simulation 1 - USP Explorer Actions.

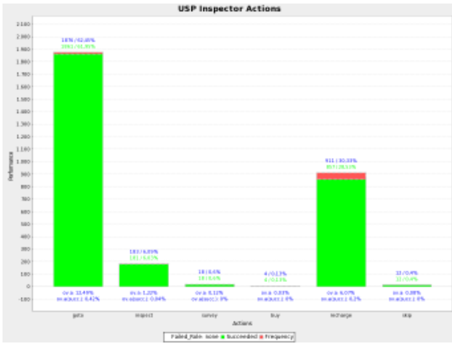


Figure 1437: USP vs. UFSC – Simulation 1 - USP Inspector Actions.

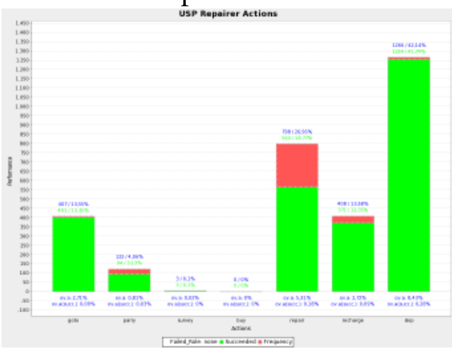


Figure 1438: USP vs. UFSC – Simulation 1 - USP Repairer Actions.

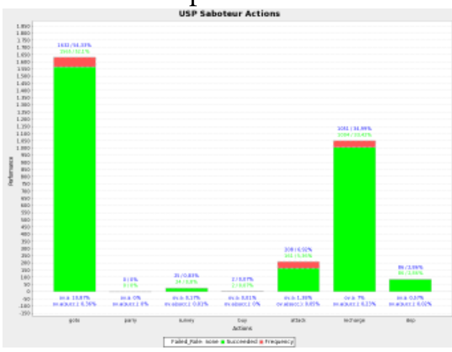


Figure 1439: USP vs. UFSC – Simulation 1 - USP Saboteur Actions.

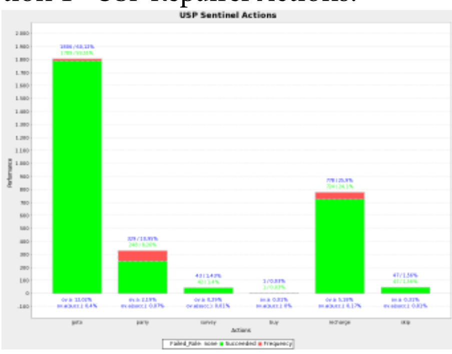


Figure 1440: USP vs. UFSC – Simulation 1 - USP Sentinel Actions.

74 USP vs. UFSC – Simulation 2

74.1 Scores, Zone Stability and Achievements

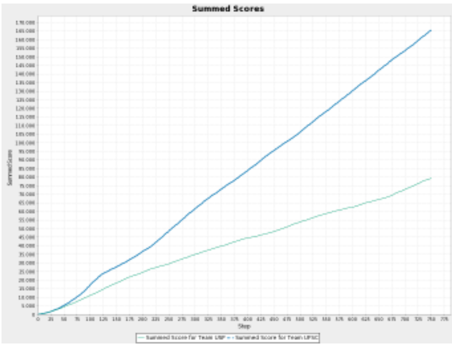


Figure 1441: Summed scores.

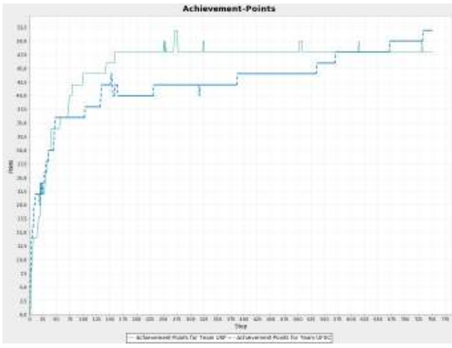


Figure 1442: Achievement points.

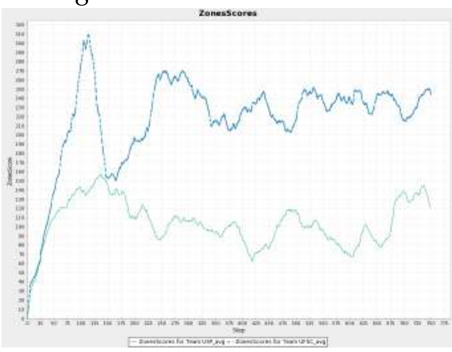


Figure 1443: Zones scores.

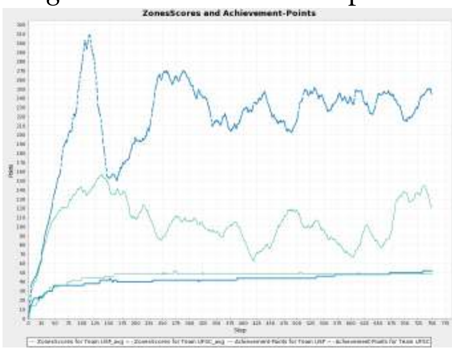


Figure 1444: Zones scores and achievement points.

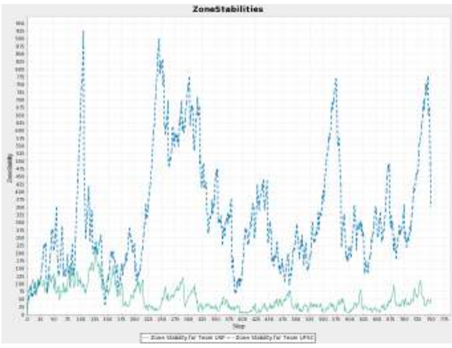


Figure 1445: Zone Stabilities.

Step	USP	UFSC
1	area10	surveyed10, area20, area10, surveyed20
2	surveyed10, surveyed40, surveyed20	surveyed40
3		surveyed80, proved5
4	area20, proved5	
5		proved10
6	proved10	
7		surveyed160
8		attacked5
10		proved20
13		attacked10
14	area40	
16	surveyed80	
19	proved20	area40, surveyed320
20	area80	
22		proved40
24		attacked20
27		inspected5
28	inspected5	
29	parried5	
30		area80
32	attacked5	
35	parried10	inspected10
39	parried20	
40	attacked10	
45		attacked40
46		area160
47		proved80
56	surveyed160	
71	inspected10	
73	parried40	
79	proved40	
99	attacked20	
103		proved160
132		parried5
133		attacked80
142	parried80	
151		parried10
158		parried20
159	attacked40	
231		attacked160
250	inspected20	
253	proved80	
269	attacked80	
270	surveyed320	
317		parried40
323	parried160	
387		attacked320
503	attacked160	
535		inspected20
570		parried80
613	parried320	
671		attacked640
731	proved160	
733		parried160

Figure 1446: Achievements.

74.2 Stability

Reason	USP	%	UFSC	%
failed away	324	2,16		
failed parried	182	1,21	433	2,89
failed random	155	1,03	133	0,89
failed resources	45	0,3	1	0,01
failed			17	0,11
failed attacked	204	1,36	41	0,27
noAction			17	0,11

Figure 1447: Failed actions.

74.3 Achievements

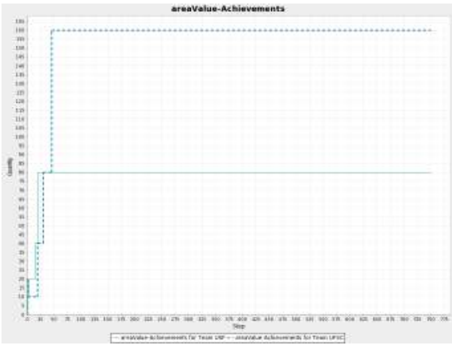


Figure 1448: areaValueAchievements.

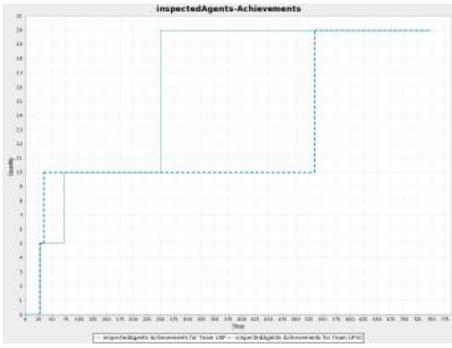


Figure 1449: inspectedAgentsAchievements.

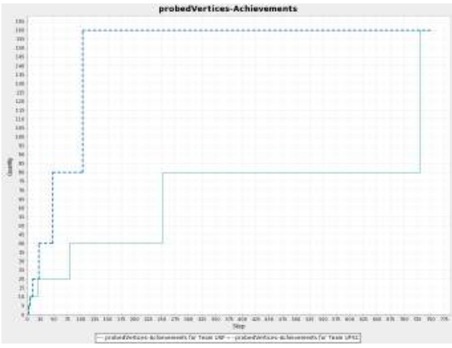


Figure 1450: probedVerticesAchievements.

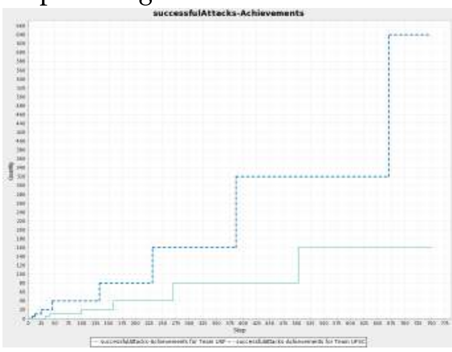


Figure 1451: successfulAttacksAchievements.

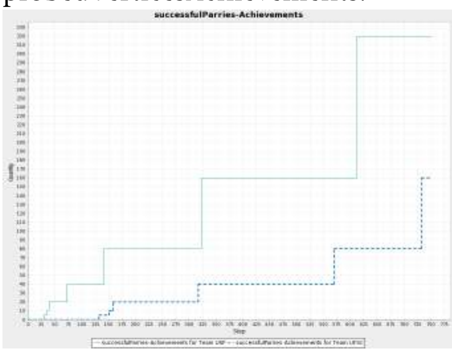


Figure 1452: successfulParriesAchievements.

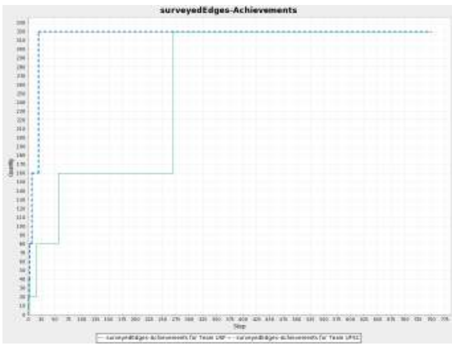


Figure 1453: surveyedEdgesAchievements.

74.4 Actions per Role

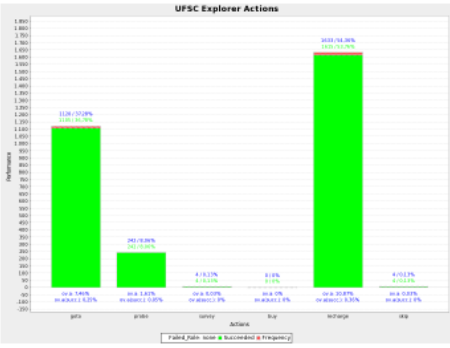


Figure 1454: USP vs. UFSC – Simulation 2 - UFSC Explorer Actions.

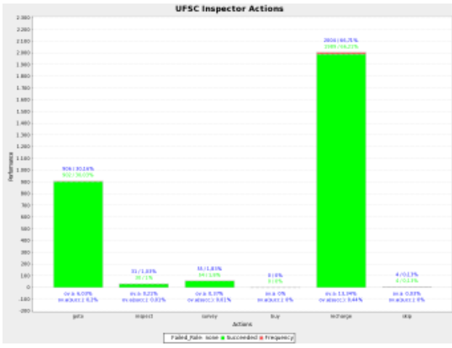


Figure 1455: USP vs. UFSC – Simulation 2 - UFSC Inspector Actions.

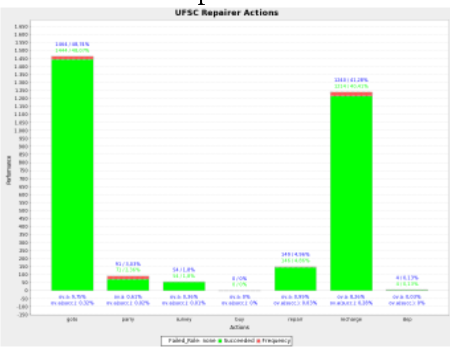


Figure 1456: USP vs. UFSC – Simulation 2 - UFSC Repairer Actions.

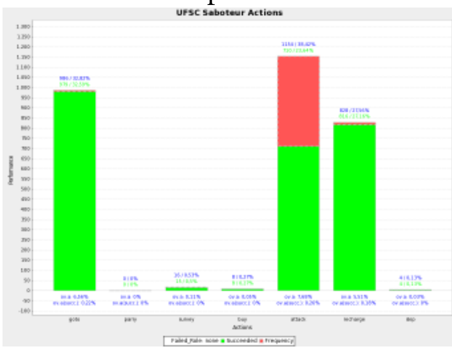


Figure 1457: USP vs. UFSC – Simulation 2 - UFSC Saboteur Actions.

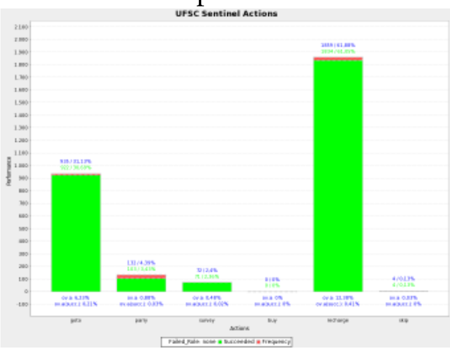


Figure 1458: USP vs. UFSC – Simulation 2 - UFSC Sentinel Actions.

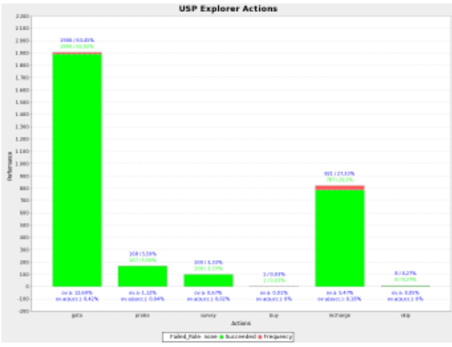


Figure 1459: USP vs. UFSC – Simulation 2 - USP Explorer Actions.

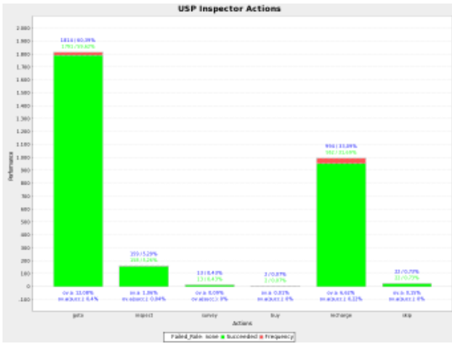


Figure 1460: USP vs. UFSC – Simulation 2 - USP Inspector Actions.

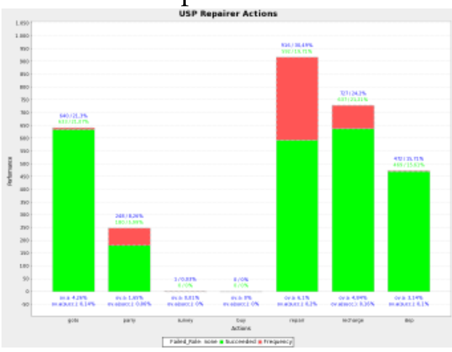


Figure 1461: USP vs. UFSC – Simulation 2 - USP Repairer Actions.

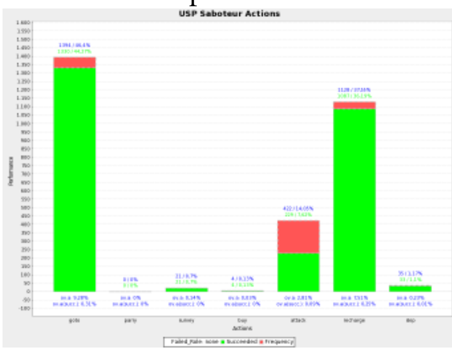


Figure 1462: USP vs. UFSC – Simulation 2 - USP Saboteur Actions.

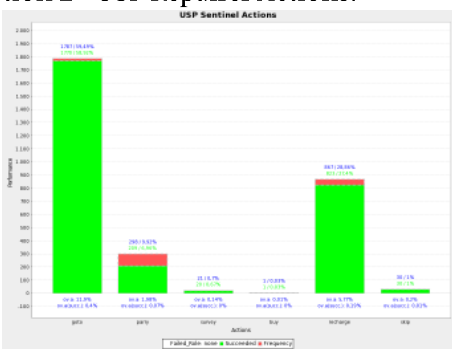


Figure 1463: USP vs. UFSC – Simulation 2 - USP Sentinel Actions.

75 USP vs. UFSC – Simulation 3

75.1 Scores, Zone Stability and Achievements

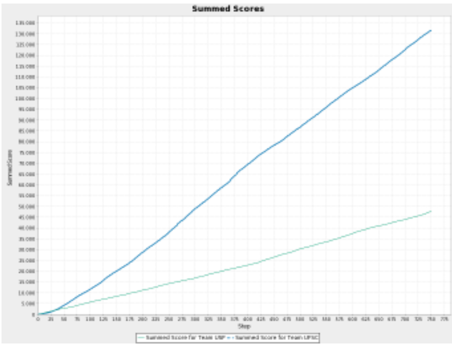


Figure 1464: Summed scores.

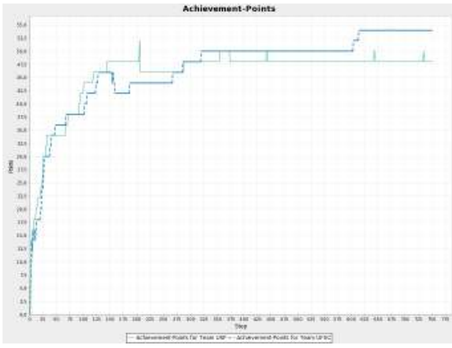


Figure 1465: Achievement points.

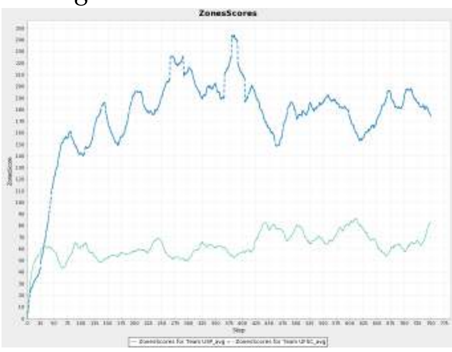


Figure 1466: Zones scores.

Figure 1467: Zones scores and achievement points.

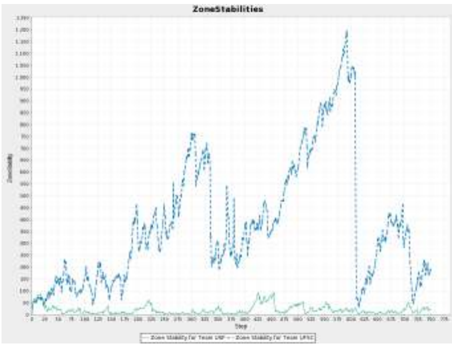


Figure 1468: Zone Stabilities.

Step	USP	UFSC
1	surveyed10, surveyed40, area10, surveyed20	surveyed10, area10, surveyed20
2		surveyed40
3	surveyed80	area20, surveyed80, proved5
4	proved5	
5	area20	proved10, inspected5
6	area40	
7	proved10	attacked5
8		surveyed160
11	attacked5	proved20
12		inspected10
14	surveyed160	
19		attacked10
20	proved20	
22		area40, proved40
23	parried5	
24	inspected10, inspected5	
25		inspected20
26		attacked20
27		surveyed320
29	parried10	
32	attacked10	
37		area80
40		attacked40
47		proved80
67	attacked20	attacked80
71	parried20	
91	inspected20	
94	parried40	
100	proved40	
102		parried5
106		proved160
118	parried80	
123		area160
127		parried10
143	attacked40	
155		attacked160
186		parried20
204	surveyed320, parried160	
266		area320
285		parried40
287	proved80	
319		attacked320
354	attacked80	
441	parried320	
603		parried80
613		attacked640
641	attacked160	
734	proved160	

Figure 1469: Achievements.

75.2 Stability

Reason	USP	%	UFSC	%
failed away	367	2,45	1	0,01
failed parried	143	0,95	548	3,65
failed random	174	1,16	156	1,04
failed resources	17	0,11	1	0,01
failed			43	0,29
failed attacked	268	1,79	29	0,19
noAction			43	0,29

Figure 1470: Failed actions.

75.3 Achievements

Figure 1471: areaValueAchievements.

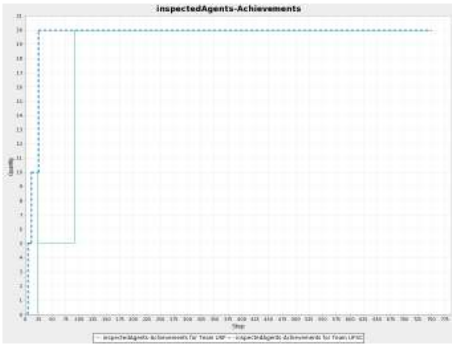


Figure 1472: inspectedAgentsAchievements.

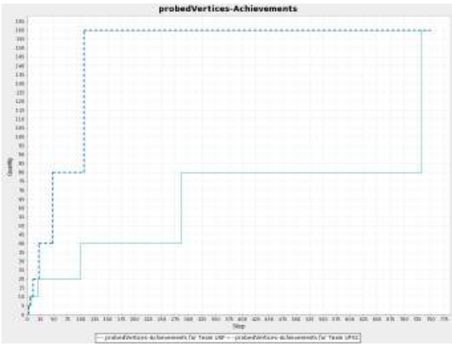


Figure 1473: probedVerticesAchievements.

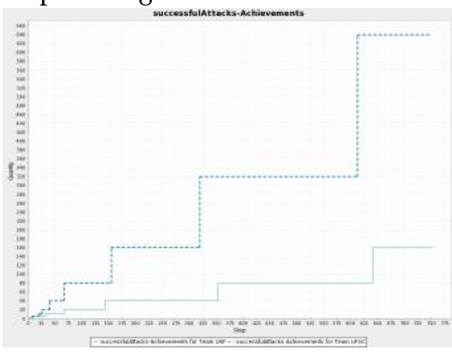


Figure 1474: successfulAttacksAchievements.

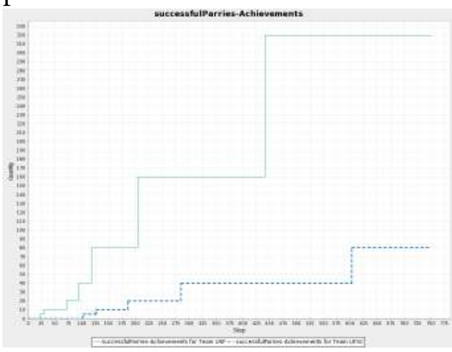


Figure 1475: successfulParriesAchievements.

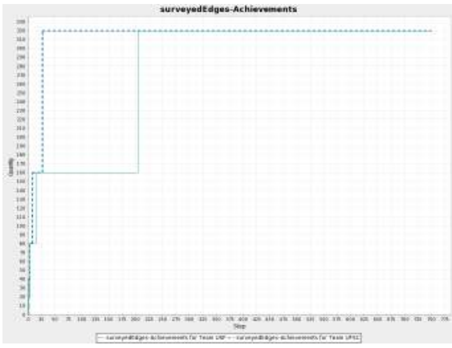


Figure 1476: surveyedEdgesAchievements.

75.4 Actions per Role

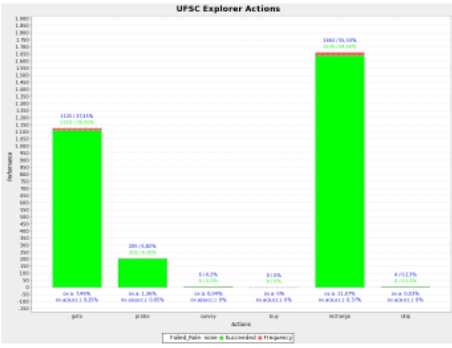


Figure 1477: USP vs. UFSC – Simulation 3 - UFSC Explorer Actions.

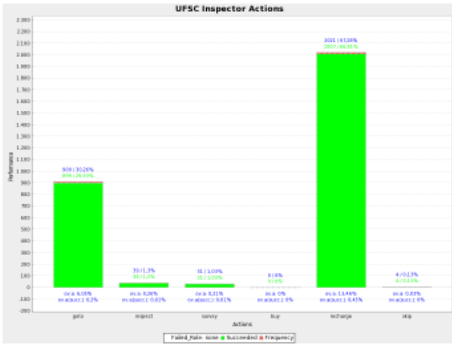


Figure 1478: USP vs. UFSC – Simulation 3 - UFSC Inspector Actions.

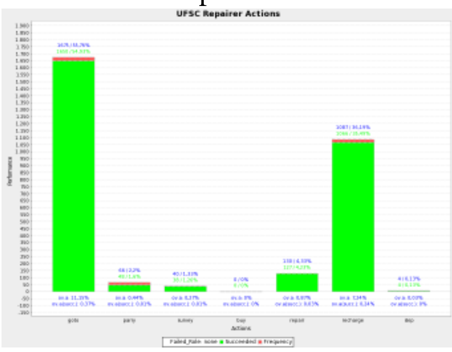


Figure 1479: USP vs. UFSC – Simulation 3 - UFSC Repairer Actions.

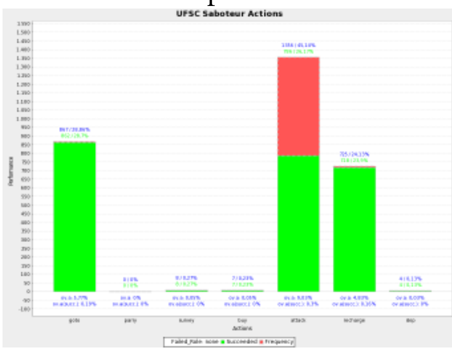


Figure 1480: USP vs. UFSC – Simulation 3 - UFSC Saboteur Actions.

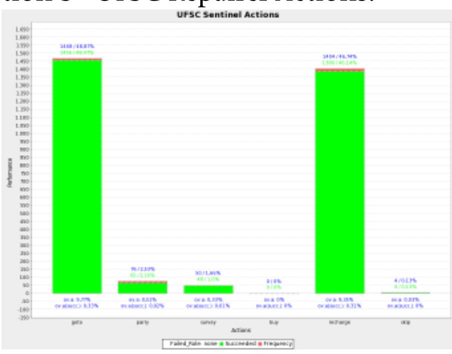


Figure 1481: USP vs. UFSC – Simulation 3 - UFSC Sentinel Actions.

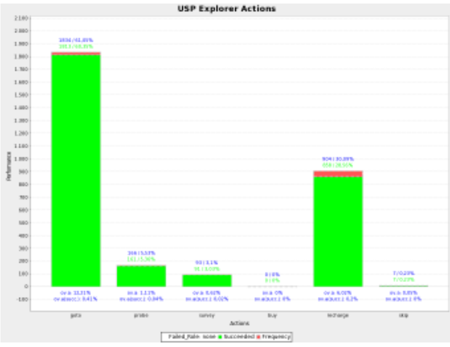


Figure 1482: USP vs. UFSC – Simulation 3 - USP Explorer Actions.

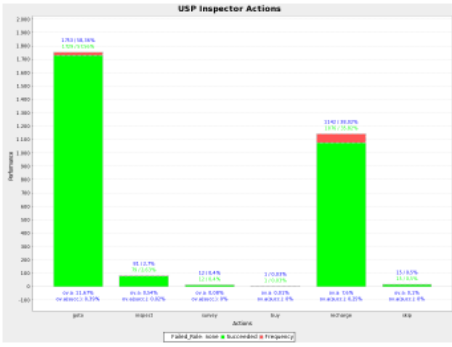


Figure 1483: USP vs. UFSC – Simulation 3 - USP Inspector Actions.

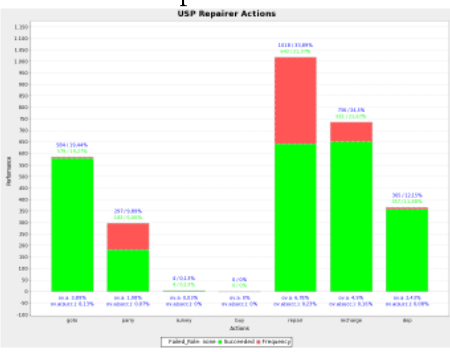


Figure 1484: USP vs. UFSC – Simulation 3 - USP Repairer Actions.

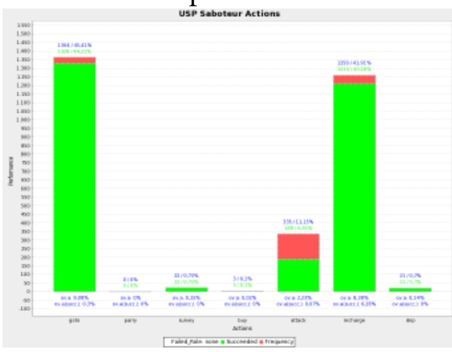


Figure 1485: USP vs. UFSC – Simulation 3 - USP Saboteur Actions.

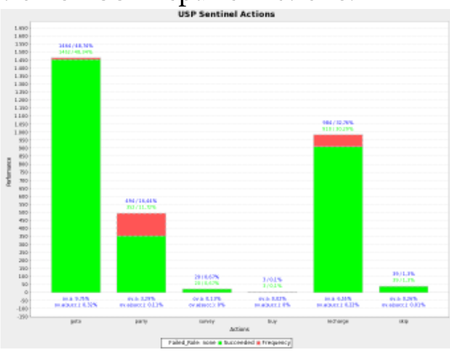


Figure 1486: USP vs. UFSC – Simulation 3 - USP Sentinel Actions.