Multi-Agent Programming Contest
Scenario Description
2012 Edition

http://www.multiagentcontest.org/2012/

Tristan Behrens Michael Köster Federico Schlesinger
Jürgen Dix Jomi Hübner

April 17, 2012

1 Introduction

In the following, we provide a detailed description of the Multi-Agent Programming Contest 2012 scenario. The overall goal of the game is to control zones of a map (graph) by placing agents on appropriate positions.

2 Background 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 of them are faster and some of them have sabotage devices on board. Last but not least, your team also contains special experts, the repair agents, that are capable of fixing agents that are disabled. In general, each agent has a 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 themselves.

3 The Challenge

In this year's Contest the participants have to compete in an environment that is constituted by a graph where the vertices have an unique identifier and also a number that determines the value of that vertex. The weights of the edges on the other hand denotes the costs of traversing the edge.

A *zone* is a subgraph (with at least two nodes) whose vertices are colored by the graph coloring algorithm introduced in Section 4. If the vertices of a zone are colored with a certain team color it is said that this team occupies this area. The value of a zone determined by the sum of its vertices' values. Since the agents do not know a priori the values of the vertices, only probed vertices contribute with their full value to the zone-value, unprobed ones only contribute one point.

The goal of the game is to maximize the score. The score is computed by summing up the values of the zones and the current money (cf. Section 8) for each simulation step:

$$\texttt{score} = \sum_{s=1}^{\texttt{steps}} (\texttt{zones}_s + \texttt{money}_s)$$

Where `steps` is the number of simulation steps, and $\texttt{zones}_s$ and $\texttt{money}_s$ are the current sum of all zone values and the current amount of money respectively.

Figure 1 shows such a scenario. The numbers depicted in the vertices describe the values of the water wells while the distance of two water wells is labeled with travel costs. The green team controls the green zone while the blue team has the smaller blue zone. The value of the blue zone, assuming that all vertices have been probed by the blue team, is 25.

4 Graph Coloring Algorithm

The graph coloring algorithm is used to determine the zones that a team is occupying. We firstly present the formal definition and afterwards explain it via an example.

Definition 4.1. Let V be the set of vertices, E the set of edges, ag the set of agents, and T the set of team names. Furthermore let $ag(v)$ denote the set of agents standing on vertex $v \in V$. A *graph coloring* is a mapping

$$c : V \to T \cup \{none\}.$$

The coloring is subject to change over time. We say that a vertex v is colored if $c(v) \neq none$. The coloring is determined by the following calculation, consisting of phases that are executed sequentially:

1. The first phase of the calculation only involves the coloring of vertices that have agents standing on them. $c(v) = t$ iff $ag(v) \neq \emptyset$ and t is the name of

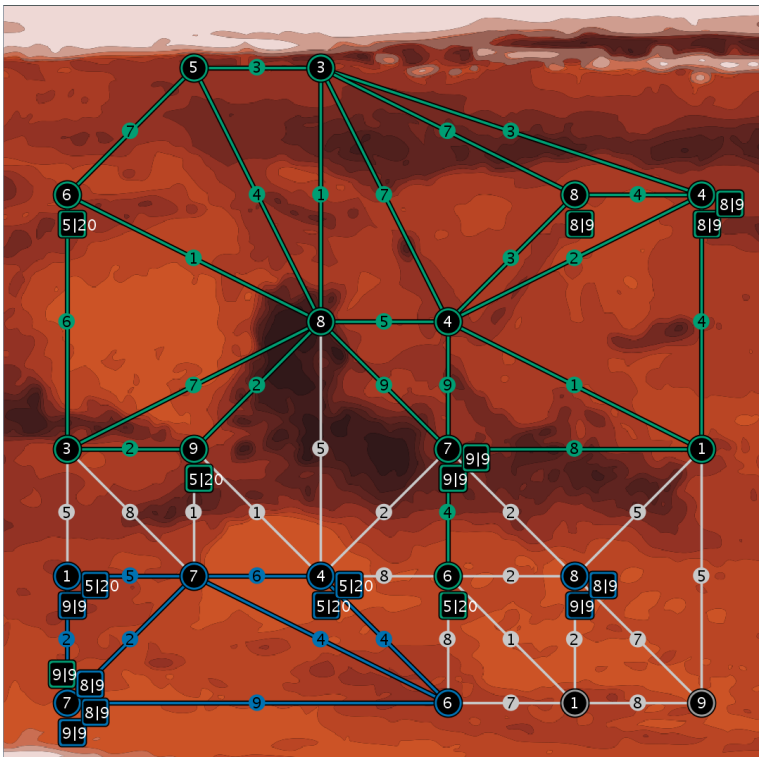


Figure 1: A screenshot.

the team that dominates the vertex. We say that a vertex v is dominated by t if t has the majority of agents on v. If no team dominates the vertex, then $c(v) = none$.

2. The coloring is extended to empty vertices that are direct neighbors of dominated vertices. Formally, $c(v) = t$ if $ag(v) = \emptyset$, t is the name of the team that dominates the largest subset of neighbors

$$S_t = \{v_n \mid (v, v_n) \in E, c(v_n) = t, c(v_n) \neq none, ag(v_n) \neq \emptyset\}$$

of v, with $\mid S_t \mid > 1$. Note that a team needs to dominate at least two neighboring vertices of an empty vertex to be able to color that empty vertex.

3. Some of the vertices that where colored with a team name t in the previous two steps might represent a *frontier* that isolates a part of the graph from all the other teams' agents. We say that an empty vertex v has been isolated by a team t (and thus $c(v) := t$) iff for all agents ag belonging to a team t', where $t' \neq t$, there is no path from ag_n to v that does not include a vertex v' such $c(v') = t$.

4. $c(v) := none$ iff the other conditions are not satisfied.

For the coloring algorithm, we are only consider agents that are not `disabled`. The definition of disabled agents is given later in Section 7.

An example of graph coloring in an hypothetical world configuration is depicted in Figure 2. Pictures (a), (b) and (c) show the result of executing the coloring calculation phases 1, 2 and 3 respectively. For the sake of improving visibility, all edges whose two vertices are colored in the same team's color, are also shown in that same color, but internally this has neither meanings nor implications.

In detail, phase 1 colors such vertices in a certain color regarding the color of the majority of agents. For instance, in Figure (a) the top right vertex is colored in green because there are three green agents but only one red agent standing on that vertex. When there is a draw the vertex does not belong to a team.

In phase 2 (Figure (b)) we look at the direct neighbors of the already colored vertices. We color such a neighbor in a certain team color when there is an edge from this uncolored vertex to at least two other vertices that are colored in that particular team color. We are taking again the majority into account, i.e., the color of the vertex is finally determined by counting for each team color the connected vertices and choosing the best result. If there is a draw the vertex is not colored at all.

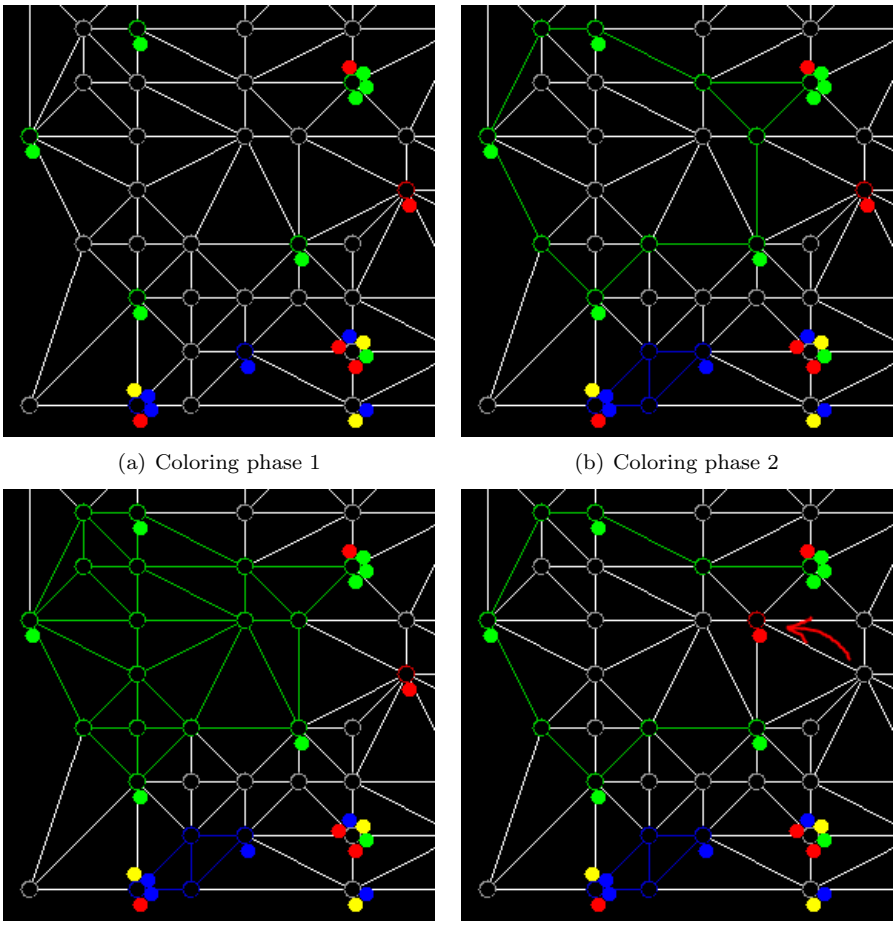
Phase 3, finally, colors all vertices that are not reachable by other teams without crossing the already colored vertices. One can see it as a border that is separating parts of the graph. After executing phase 3 we have defined the zones of all teams.

Picture (b) clearly shows how the green team has built a closed frontier around a set of empty vertices, which are then colored in picture (c). In picture (d), an agent of the red team has "broken" the frontier, making some of the vertices inside of it not isolated anymore.

5 Teams & All Terrain Planetary Vehicles

We define five roles (see Table 1), where each role describes the available actions (actions an agent can perform) for the All Terrain Planetary Vehicle (ATPV), its maximum energy, its maximum health, its strength and its visibility range. While the energy is important for executing actions, the health determines whether an agent is still able to perform all actions or just a small subset. The strength defines how strong a sabotage will be and the visibility range describes how far an agent can see. The concrete actions are described in Section 6. The teams consist of 20[1] agents and for each group of four agents we assign the following roles:

[1] This is new in this version.



(a) Coloring phase 1

(b) Coloring phase 2

(c) Coloring phase 3

(d) Breaking a frontier.

Figure 2: Coloring phases

6 Agent Actions

An agent can perform some of the following actions regarding its role. The result of that action is perceived by the agent automatically, i.e., the information is sent to it in the next percept.

`skip` This action is always `successful`, costs no resources and the effect is that the agent does not do anything.

`recharge` This action increases the current energy of the agent by 20 percent. The result can be `successful`, or `failed` if attacked by an opponent. Of course, it does not cost any resources.

Explorer	Actions:	`skip`, `goto`, `probe`, `survey`, `buy`, `recharge`
	Energy:	12
	Health:	4
	Strength:	0
	Visibility range:	2
Repairer	Actions:	`skip`, `goto`, `parry`, `survey`, `buy`, `repair`, `recharge`
	Energy:	8
	Health:	6
	Strength:	0
	Visibility range:	1
Saboteur	Actions:	`skip`, `goto`, `parry`, `survey`, `buy`, `attack`, `recharge`
	Energy:	7
	Health:	3
	Strength:	4
	Visibility range:	1
Sentinel	Actions:	`skip`, `goto`, `parry`, `survey`, `buy`, `recharge`
	Energy:	10
	Health:	1
	Strength:	0
	Visibility range:	3
Inspector	Actions:	`skip`, `goto`, `inspect`, `survey`, `buy`, `recharge`
	Energy:	8
	Health:	6
	Strength:	0
	Visibility range:	1

Table 1: The different roles.

`attack` If an agent wants to sabotage some other agent it has to perform this action. The action requires a parameter (the identifier of the target) and can be `successful`, or `failed` because of lack of energy. Also, it can be `parried`. Lastly, it can fail because of a wrong parameter, i.e., `wrongParameter` is the result. Also, the current energy is decreased about 2 points.

`parry` This action parries an attack and costs 2 points of energy points. If actually an attack is taking place it is `successful`, otherwise it is `useless`. The action can also fail because of too low energy, i.e., the result is `failed`.

`goto` The agent moves from one vertex to another by executing this action. The reduction of the current energy is determined by the traveling costs, i.e., the weight of an edge. The action needs a parameter, namely the id of the vertex it wants to go to. The result of that action is, `failed` when the current energy is too low and the current energy is decreased by 1, `wrongParameter` if the parameter is incorrect or `successful` otherwise.

`probe` Without the team knowing the exact value of the node, it is set to 1

when it comes to the computation of the zone score. Only if at least one agent of the team analyzes the water well the team gets the full value of that vertex (the value is then incorporated in the next percept). The action costs 1 points of energy. A `probe` action can fail (result: `failed`) for different reasons: lack of energy or being attacked by another agent. Otherwise it is `successful`.

`survey` With this action (costs: 1) the agent can get the weights of the edges (in the next percept). If the action is not `successful` it `failed` because of too low energy or the agent was attacked in that moment.

`inspect` This action (costs: 2) inspects all opponents (the internals) on the vertex the agent is standing at the moment as well as all direct neighbors. The result can be again `failed` because of lack of energy or being attacked, and is `successful` otherwise even if there is no agent at all.

`buy` The `buy` action (costs: 2) is more complex. It's purpose is to increase your agent's maximum health, maximum energy, visibility range or maximum strength buy spending money (cf. Section 8) on extension packs. The possible values for the parameter are: `battery` (increases maximum energy and current energy by 1), `sensor` (increases visibility range by 1), `shield` (increases maximum health and current health 1) or `sabotageDevice` (increases the strength by 1). Of course, it fails if your agent is not allowed (determined by the role) to wear a `sabotageDevice`. Also, it fails when being attacked or if the current energy is too low. Finally, if the parameter is syntactically wrong the result is `wrongParameter` and otherwise it is `successful`.

`repair` This action (costs: 2) repairs a teammate. Note that an agent cannot repair itself. The parameter determines which agent gets repaired. If the value is syntactically wrong the result is `wrongParameter`. If there is no such agent it is `failed`. It also fails because of too low energy. Otherwise it is `successful`.

In general, an action can fail with a certain probability (1 percent). In this case the action is considered as the `skip` action (and the percieved result will be `failed`). Actions that are performed by an agent but do not correspond to the agent's role fail as well.

7 Disabled Agents

Agents whose health drops to zero, are disabled, i.e., only the action `goto`, `repair`, `skip` are executable (if the role allows that). The `recharge` action is also allowed to be performed, but its recharge rate is set to 10 percent.

8 Money

If a team reaches a milestone, its money is increased. We have different achievements, for example:

- having zones with fixed values, e.g. 10 or 20,
- fixed numbers of probed vertices, e.g. 5 or 10,
- fixed numbers of surveyed edges, e.g. 10 or 20,
- fixed numbers of inspected vehicles, e.g. 5 or 10,
- fixed numbers of successful attacks, e.g. 5 or 10, or
- fixed numbers of successful parries, e.g. 5 or 10.

Note that the exact achievements are defined in the configuration. The exact details about achievements are still object to change, in the upcoming phase of balancing. We are going to announce them later.

9 Percepts

In every step, the agents get these percepts:

- state of the simulation, i.e. the current step,
- state of the team, i.e. the current scores and money,
- state of the vehicle, i.e. its internals as described above,
- visible vertices, i.e. identifier and team,
- visible edges, i.e. its vertices' identifiers,
- visible vehicles, i.e. its identifier, vertex, team,
- probed vertices, i.e. its identifier and its value,
- surveyed edges, i.e. its vertices' identifiers and weight, and
- inspected vehicles, i.e. its identifier, vertex, team and internals.

Please refer to the protocol description for the details about percepts.

We also have the notion of *shared percepts*. Agents of the same team that are in the same zone share their percepts, that is visible vertices, edges and vehicles, and probed vertices, surveyed edges and inspected vehicles.

10 Simulation state transition

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.