# Multi-Agent Programming Contest Scenario Description (2014 Edition)

http://www.multiagentcontest.org/2013/

Tobias Ahlbrecht Jürgen Dix Michael Köster
Federico Schlesinger

July 24, 2014

**New in 2014:** Differences between the last year and 2014 are marked with boxes.

## Contents

## 1 Introduction

In the following, we provide a detailed description of the Multi-Agent Programming Contest 2014 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[1] 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 5. 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.

[1]The words *vertex* and *node* are used interchangeably in this text, except when referring to pieces of XML code (in `true type typography`).

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 9) 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.

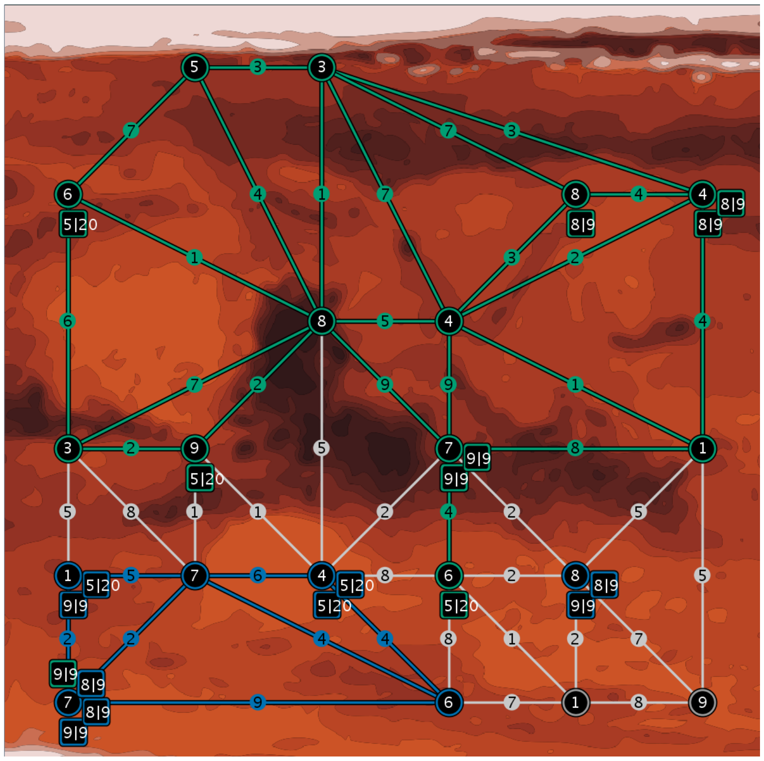


Figure 1: A screenshot.

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 Visualization

We have three different modes for the visualization, named after the year we introduced them. All of them might be useful for debugging so we will describe them in the following sections. Please note that you can watch a match while it is running as well as open a previously recorded match as well.

**Running Simulation**

```
$ ./startMarsViewer.sh localhost
```

**Recorded Simulation**

```
$ ./startMarsFileViewer.sh Mars2013_AB_2013-tournament-sim1
```

You can get details about an agent or a node by clicking on it. Additionally, you can select an agent by using the combo box. If you press pause in a running example the match on the server is still going on, however it gives you more time to look at a particular step.

The visualization for the SVGs is at the moment similar to Mode 2012, interactive features are not available. Of course we will migrate to the Mode 2013 soon.

## 4.1 Mode 2011

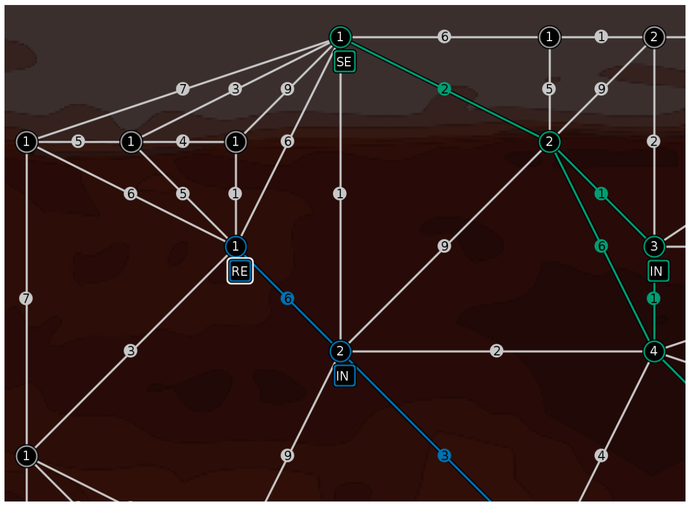


Figure 2: A screenshot of the mode 2011.

This is the first visualization that was used in 2011 (cf. Fig. 2). The number in the nodes describe the value of the node while number on edges denote the traveling costs. The roles of agents are depicted inside the agent symbol. Finally, blue and green lines describe the team zones. For the role names the following abbreviations are used:

| | |
|---|---|
| RE | Repairer |
| SE | Sentinel |
| IN | Inspector |
| SA | Saboteur |
| EX | Explorer |

Lastly, disabled agents are filled with a Grey tone.

### 4.2 Mode 2012

This is the visualization for 2012 (see Fig. 3).

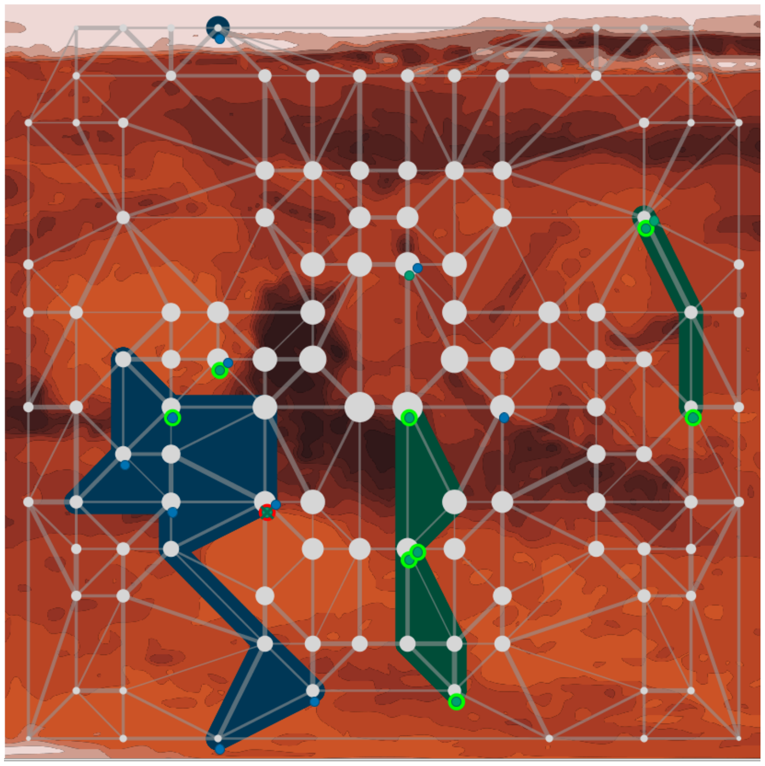


Figure 3: A screenshot of the mode 2012.

The thickness of the grey lines denotes the weights of the respective edges. A thin line represents a small weight and a thick line a big value. The sizes of the grey circles denote the nodes' values. A small circle stands for a low value, a big circle on the other hand represents a high value. Both teams color individual nodes, edges between nodes and zones with their respective color.

On top of that the last action is rendered. The agents are decorated with the following color/shape scheme:

**green circle** - the agent performed a successful sense action (probe, survey, inspect),

**red circle** - the last action failed,

**yellow star** - the last action was a successful attack,

**indigo star** - the last action was a successful parry,

**pink star** - the agent executed a repair successfully, and

**crossed out** - the agent is currently disabled

Additionally, disabled agents are crossed out.

Please note that this color scheme is only representing the results of the last action not the action currently performed.

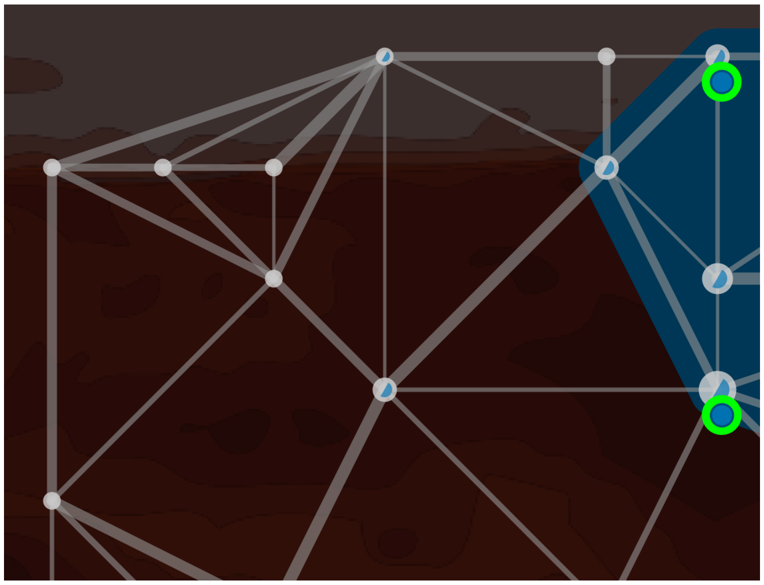


Figure 4: A screenshot of the mode 2012.

We also added a new feature to this mode, namely the half circles inside the vertices that describe whether a node was probed by a particular team. Fig 4 shows a setting were some nodes where probed by the blue team while the green team did not probe any vertex so far.

## 4.3 Mode 2013

This is the newest iteration of our visualization (Fig. 5) and still the most up-to-date version in 2014. As before the thickness of edges describe the traveling costs while the size of the vertices denote the value of a node. Probed nodes are filled with the color of the corresponding team. A red cross means that an agent is disabled. The shape of an agent describes its role:

| | |
|---|---|
| Triangle | Inspector |
| Diamond | Saboteur |
| Circle | Explorer |
| Octagon | Repairer |
| Square | Sentinel |

Inside of the agent is its identifier. Above an agent is its last action depicted. If it was successful it is green, if it was failing in range it is yellow, otherwise red. The symbol describes the action that was executed:

| | |
|---|---|
| +1 | describes a `buy` action. The item that was bought is depicted afterwards. A heart means `health`, a flash corresponds to `energy`, glasses to `visibility` and a sword to `strength`. |
| Cross | This shows the `goto` action. |
| Drill | The `probe` action. |
| Glasses | The `survey` action. |
| Flash | The `recharge` action. |
| Shield | The `parry` action. |
| Wrench | The `repair` action. |
| Sword | The `attack` action. |
| Magnifying Glass | The `inspect` action. |
| Nothing | The skip `action`. |

A line from one agent to another agent is drawn with the following color:

| | |
|---|---|
| Orange | if it was an `attack` action. |
| Magenta | if it was an `inspect` action. |
| Cyan | if it was an `probe` action. |
| Pink | if it was an `repair` action. |

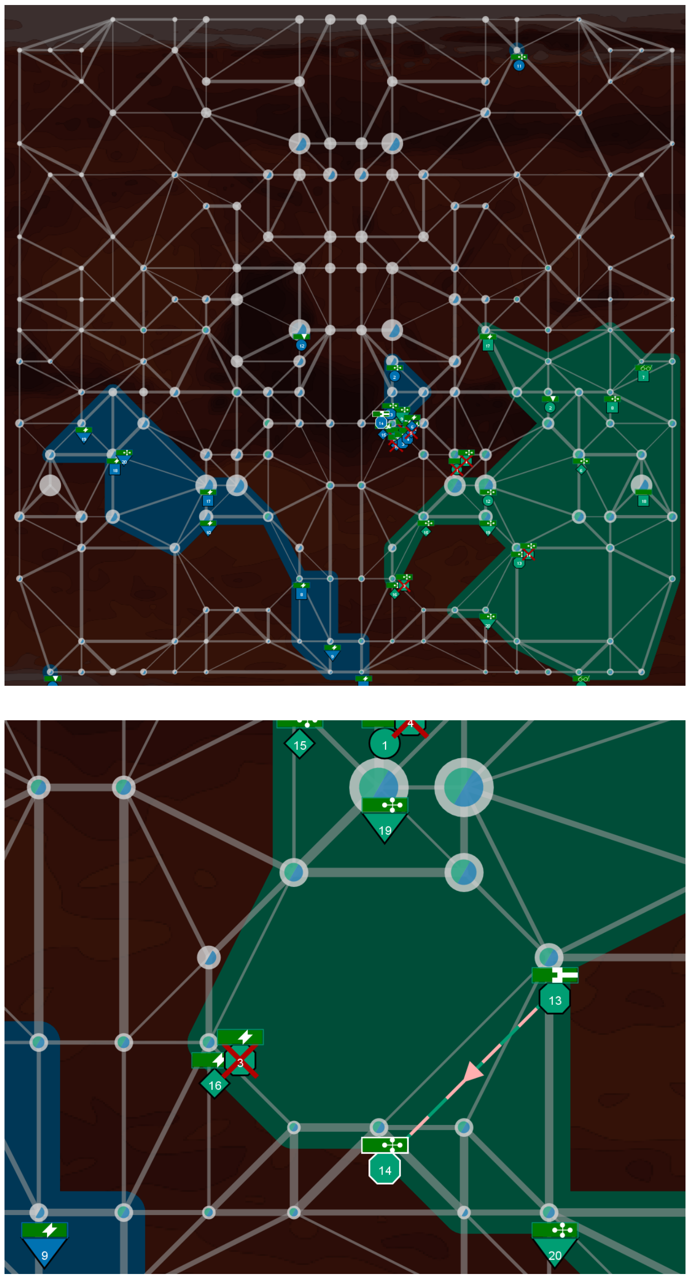


Figure 5: Screenshots of the mode 2013.

# 5 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 5.1.** 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 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 8.

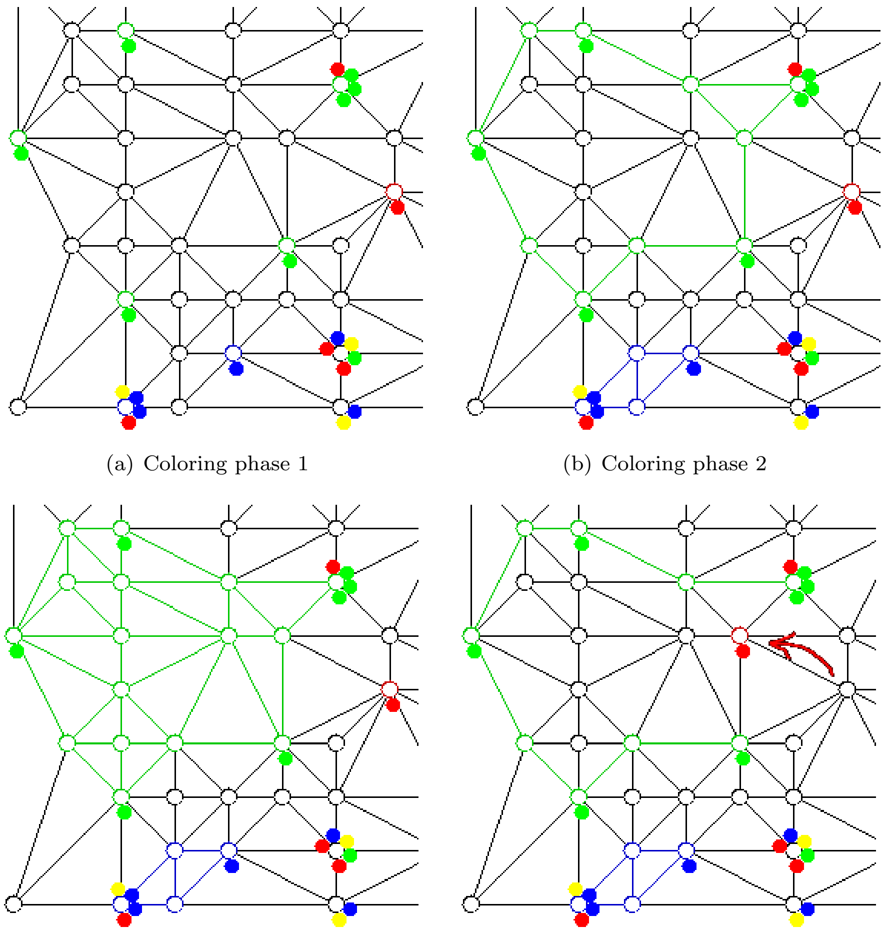
An example of graph coloring in an hypothetical world configuration is depicted in Figure 6. Pictures (a), (b) and (c) show the result of executing the coloring calculation phases 1, 2 and 3 respectively. For the sake of improving visibility, all edges whose two vertices are colored in the same team's 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.


(a) Coloring phase 1 (b) Coloring phase 2

(c) Coloring phase 3 (d) Breaking a frontier.

Figure 6: Coloring phases

## 6 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 7. The teams consist of 28 agents (6 Explorers, 6 Repairers, 6 Sentinels, 6 Inspectors and only 4 Saboteurs).

| | | |
|---|---|---|
| **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.

# 7 Agent Actions

**New in 2014:** Ranged actions were changed to be less likely to fail and overall more effective.

In this section we present all the actions that are available for the agents. Availability of an action for a particular agent depends on that agent's role, as stated in previous section. Table 2 presents a condensed version of the general characteristics. Following, we present an explanation of every action.

`skip` The agent does not do anything. Note, most often performing a recharge action is more useful than executing the skip action.

`recharge` This action increases the current energy of the agent by 50 percent of the total.

`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.

`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 after 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 percepts). **This is a *ranged* action**, meaning that the target node does not need to be on the same node (more details in Section 7.1).

`survey` With this action the agent can get the weights of the edges (in the next percept). All edges are perceived up to a certain distance from the agent, which is determined randomly based on the visibility range.

`inspect` This action is used to inspect the internal attributes of an opponent agent, given as a parameter. **This is a *ranged* action**, meaning that the opponent does not need to be on the same node (more details in Section 7.1). If no parameter is given, all opponent agents standing on the same node are inspected.

`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). **This is a *ranged* action**, meaning that the opponent does not need to be on the same node (more details in Section 7.1).

`parry` This action parries an attack, conserving the health of the agent for the step in spite of opponent actions.

`repair` This action repairs a teammate. **Note that an agent cannot repair itself**. The parameter determines which agent gets repaired. **This is a *ranged* action**, meaning that the opponent does not need to be on the same node (more details in Section 7.1).

`buy` The `buy` action is more complex. It's purpose is to increase your agent's maximum health, maximum energy, visibility range or maximum strength by spending money (cf. Section 9) 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).

The result of an action is perceived explicitly by the agent, i.e., the information is sent to it in the next percept. Last action result is `successful` when the action is effectively executed; for more information on failure of actions see section 9.

| | | |
|---|---|---|
| `skip` | Parameter: | - |
| | Status: | Any |
| | Cost: | - |
| | Failure: | - |
| `recharge` | Parameter: | - |
| | Status: | Any |
| | Cost: | - |
| | Failure: | - |
| `goto` | Parameter: | Vertex (required) |
| | Status: | Any |
| | Cost: | energy pts. equal to traversed edge value |
| | Failure: | `resources`, `attacked`, `ureachable`, `wrong_param` |
| `probe` | Parameter: | Vertex (optional) |
| | Status: | `enabled` |
| | Cost: | 1 energy pt. |
| | Failure: | `resources`, `attacked`, `out_of_range`, `in_range`, `wrong_param`, `role`, `status` |
| `survey` | Parameter: | - |
| | Status: | `enabled` |
| | Cost: | 1 energy pt. |
| | Failure: | `resources`, `attacked`, `status` |
| `inspect` | Parameter: | Agent (opponent - optional) |
| | Status: | `enabled` |
| | Cost: | 2 energy pts. |
| | Failure: | `resources`, `attacked`, `out_of_range`, `in_range`, `wrong_param`, `role`, `status` |
| `attack` | Parameter: | Agent (opponent - required) |
| | Status: | `enabled` |
| | Cost: | 2 energy pts. |
| | Failure: | `resources`, `parried`, `out_of_range`, `in_range`, `wrong_param`, `role`, `status` |
| `parry` | Parameter: | - |
| | Status: | `enabled` |
| | Cost: | 2 energy pts. |
| | Failure: | `resources`, `role`, `status` |
| `repair` | Parameter: | Agent (teammate - required) |
| | Status: | any |
| | Cost: | 2 energy pts. if `enabled`, 3 energy pts. if `disabled` |
| | Failure: | `resources`, `out_of_range`, `in_range`, `wrong_param`, `role` |
| `buy` | Parameter: | Attribute (`battery`, `sensor`, `shield` or `sabotageDevice`) |
| | Status: | `enabled` |
| | Cost: | 2 energy pts. & 2 achievement pts. |
| | Failure: | `resources`, `wrong_param`, `status`, `limit`, |

Table 2: The different actions.

## 7.1 Ranged Actions

**New in 2014:** Ranged actions were changed to be less likely to fail and overall more effective.

Certain actions of the agents can act at a distance, i.e., having a target node that is different than the one where the agent stands (`probe`), or having a target agent that stands on a different node (`inspect`, `attack` and `repair`), as long as the target is within the visibility range. Of course, this comes at some cost: the energy required to execute the action increases with the distance; a factor of randomness is introduced and the probability of success is decreased the further away the action is attempted; and some actions (`attack` and `repair`) also decrease their effects.

We define the *distance to the target* consistently with the definition of visibility range, as the minimum amount of edges that needs to be traversed to reach it (edge's values are ignored). The distance is added to the cost of the action when the action is executed successfully. The distance is also added, if the action fails. Only if the target does not exist (`wrong_param`), the base costs are taken. If the target exists and is outside the visibility range, the visibility range is added instead of the distance.

The visibility range not only affects how far away a ranged action can be executed, but also its probability of reaching the target at a certain distance. The effectiveness is decreased as a linear function, with the maximum possible effect (that is, the attacker's `strength` in case of `attack` and the target's `maxHealth` in case of `repairs`) at distance 0, an effect value of 2 at distance 4 and an effect of 1 afterwards. The function that we use for this calculation is defined as follows:

$$\text{Effect} = \max\{\text{MaxValue} - 2 \times \text{Distance}, 1\}$$

Figure 7 illustrates this formula in function of the distance for an attack with `strength` 10. Note that values beyond the agent's current visibility range are irrelevant.

For determining the outcome of a `survey` action, we calculate an *effective range* randomly every time this action is attempted. This effective range is somewhere between 1 and the agent's visibility range. The weights of all the edges within the effective range are perceived. It is given by the following formula:

$$\text{EffectiveRange} = (\text{VisRange - 1}) \cdot \text{rand}^2 + 1$$

were rand is a random value evenly distributed between 0 and 1. Figure 8 shows the function with visibility ranges of 3 and 5. Note that, since values are rounded to integers, with a Visibility range of 3 there is a 0,59 probability of the effective range being $\geq 1$, and of 0,09 of it being $\geq 3$; whereas with a Visibility range of 5, the probability is 0,68 of it being $\geq 1$, is 0,29 of it being $\geq 3$, and is 0,05 of it being $\geq 5$. These are equivalent to say, the probabilities of an action targeted at those distances reaching its target. Note that, since the function is always $\geq 0$, the actions targeted at the same node always pass this filter.

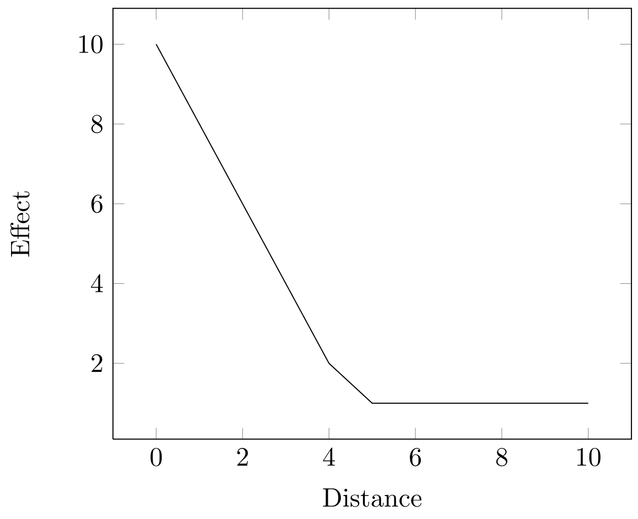


Figure 7: Effect of an Attack at a Distance

The success rate (regarding the range factor) of the remaining actions `attack`, `probe`, `repair` and `inspect` is depending on the distance to the target and the executing agent's visibility range. At a distance of 0 (in the same node), the probability of succeeding is 100%, while it decreases to 20% at a distance equivalent to the visibility range. In between, we have a linear progress. The gradient is illustrated in Figure 9 for the visibility ranges 1, 3 and 5 respectively. So, the more visibility range an agent has, the more likely he can execute a ranged action at a certain (fixed) distance.

### 7.2 Actions' Failure Codes

Actions can fail due to diverse reasons. The possible causes of failure for each action are presented in Table 2. The perceived last action result will be the prefix `failed_` followed by the failure reason. Here we explain what each possible failure means:

`failed_resources` The agent does not have enough resources to execute the action. In most cases, resources mean energy points, although for the `buy` action it can also mean money (i.e., achievement points).

`failed_attacked` The attempted action was interrupted because the agent was successfully attacked. Not that only some actions can be prevented by an attack (see Table 2).

`failed_parried` The attempted `attack` was parried by the target.

`failed_unreachable` The agent attempted to move (`goto`) to a node that is not connected to its current one.

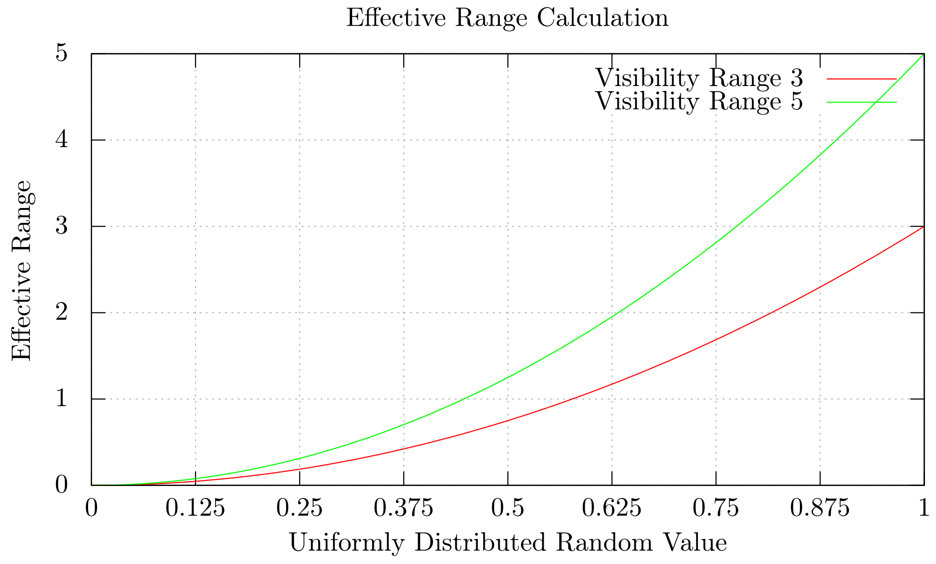


Figure 8: Effective Range Calculation

`failed_out_of_range` The target of the attempted ranged action was outside the visibility range of the agent.

`failed_in_range` The ranged action was missed because of the distance, even though the target of the attempted ranged action was within the visibility range of the agent (a random factor was involved - see Section 7.1).

`failed_wrong_param` The parameter given was not recognized as a valid identifier.

`failed_role` Agent belongs to a role that is not capable of executing the attempted action.

`failed_status` The agent is currently `disabled`, and the action can only be executed when `enabled` (see Section 8).

`failed_limit` The agent attempted to `buy` an extension pack to improve an attribute for which it has already reached the maximum value allowed.

`failed` This code is used when the agent did not send an action on time, or when the action sent was not recognized by the server.

Besides each action's intrinsic failure possibilities, any action can fail randomly with a 1 percent probability. In this case the action is considered as the `skip` action (and the perceived result will be `failed_random`).

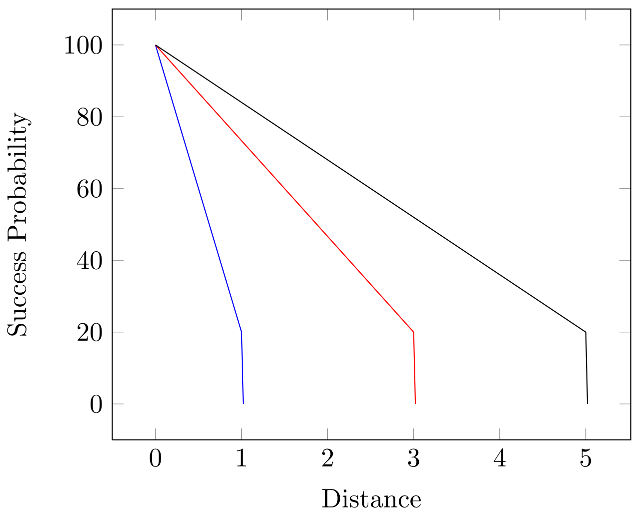


Figure 9: Success Probability for Ranged Actions

## 8 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 30 percent.

## 9 Money

If a team reaches a milestone, its money is increased by 2 units. We have different achievements:

- Conquered a zone valued more than: 10, 20, 40, 80, 160, 320, 320, 640 or 1280.
- Reached a number of probed vertices: 5, 10, 20, 40, 80, 160, 320, 320 or 640.
- Reached a number of surveyed edges: 10, 20, 40, 80, 160, 320, 320, 640 or 1280.
- Reached a number of inspected vehicles: 5, 10, 20, 40, 80, 160, 320, 320 or 640.
- Reached a number of successful attacks: 5, 10, 20, 40, 80, 160, 320, 320 or 640.
- Reached a number of successful parries: 5, 10, 20, 40, 80, 160, 320, 320 or 640.

## 10 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.

## 11 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.

## 12 Statistics

The server generates also some statistics that are useful for debugging. In the following we provide a short overview of the generated files:

**Achievement-Points:** The chart depicts the achievement-points of both teams in every step of the current simulation. The points increase, when a team gets an achievement and decrease, when the buy-action is used.

**TEAMNAME ROLENAME Actions** The chart show the actions of all four agents of the respective role in the given team. Every bar represents one action the role is allowed to use. The whole bar (green and red) relates to the frequency the action was sent by the agents. The absolute and relative numbers for this frequency are given in blue above the chart. The percentage relates to all actions that were sent by this specific role's agents in the current team. The green part of the bar represents the number of succeeded actions, that is actions which did not fail. Again, the number of succeeded actions is given in green above the respective bar. The blue numbers under the bar give relative frequencies of each action in relation to all actions that were sent by every agent of the current team in the course of the simulation. Finally, if one or more agents of this role try to send an action they aren't allowed to use, this fact is mentioned in the legend under the chart.

**ACHIEVEMENTNAME-Achievements** There is one chart for every category of achievements. It shows, in which step which quantity of the respective achievement was reached by all teams which participate in the current simulation.

**Summed Scores** The chart depicts the summed score of each teams in each step of the current simulation.

**ZonesScores** The chart depicts the ZonesScores of each teams in each step of the current simulation. The ZonesScore derives from the number and value of the currently dominated nodes.

**ZonesScores and AchievementPoints** This chart is just a combination of both ZonesScores- and Achievement-Points-chart.

**ZoneStabilities** The chart depicts the ZoneStabilities of each teams in each step of the current simulation. The ZonesStability 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.