Multi-Agent Programming in Jason

Pawel Drozdowski s103460
Niels Beuschau s103471

Kongens Lyngby 2014
Compute-B.Sc.-2014

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk Compute-B.Sc.-2014

Summary (English)

The primary goal of this thesis was to gain extensive knowledge about multi-agent programming - the underlying models and theory, as well as applications in practice. The main point of interest was the Jason programming platform, which is used in practice for multi-agent system implementations. The focus was put on a system developed for the multi-agent programming contest by the UFSC team from Federal University of Santa Catarina in Brazil. The main objectives were to analyse that system, identify possible improvements and implement them. After that was accomplished, series of quantifiable tests against the original version of the system were performed, in order to establish the effects of the changes. The tests showed a significant improvement in the specific parameters, which were aimed to be enhanced. Moreover, a measurable positive change in the overall system performance was seen.

Summary (Danish)

Det grundlæggende mål for denne afhandling var at opnå bred viden indenfor multiagentprogrammering - de underliggende modeller, teori, praktiske anvendelser, samt at analysere forbedringsmuligheder. Specielt interessant var Jason programmeringsplatformen, som benyttes til udvikling af multiagent systemer i praksis. Fokus var lagt på et system lavet til en multiagentprogrammerings konkurrence af holdet UFSC fra Federal University of Santa Catarina i Brazilien. Hovedmålet var at analysere dette system, identificere mulige forbedringer og implementere disse. Dette opdaterede system blev efterfølgende testet mod det oprindelige system for at vurdere effekten af de implementerede ændringer. Resultaterne viste betydelige forbedringer i de specifikke parametre, der var arbejdet på, samt en generel positiv indflydelse på systemets samlede ydeevne.

Preface

This thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfilment of the requirements for acquiring a B.Sc. in Software Technology.

The thesis deals with multi-agent programming, the Jason platform, the multi-agent programming contest and the system developed by its winners - the UFSC team from Federal University of Santa Catarina in Brazil.

Lyngby, 01-July-2014

P. Drozdowski Niels Beuschau

Pawel Drozdowski s103460 Niels Beuschau s103471

Acknowledgements

We would like to thank our supervisor Jørgen Villadsen for the guidance and help throughout the course of this project. We are also grateful to the entire UFSC team from Federal University of Santa Catarina in Brazil for allowing us to work with their system, and especially to Jomi Fred Hübner and Maicon Rafael Zatelli from that team for providing us with suggestions and advice for our work with the aforementioned system.

Contents

Chapter 1

Introduction

In this chapter, the main goals and the structure of this project are described. Furthermore, the section of computer science called 'artificial intelligence' is introduced.

1.1 Learning objectives and report structure

This projects aim is to obtain a deep insight into the world of artificial intelligence and, more specifically, multi-agent systems. The goal here is to understand the underlying theoretical models, as well as gain experience with practical implementations. Work will be carried out with the Jason programming platform and the multi-agent programming contest in order to achieve these objectives. The system developed by the winners of this contest will be examined. Once the inner workings of that system have been grasped, alterations will be made to some of the strategies it employs; new strategies will be developed as well. The ambition here is to increase the systems overall performance. A rigorous testing process will be used in order to assess the quality of the changes. Naturally, at the end of the report, perspectives and conclusions about the results and multi-agent systems in general will be presented and discussed.

Chapters 2 and 4 introduce the basic theory behind multi-agent systems and the

Jason platform, while chapter 3 outlines the multi-agent programming contest. Chapters 5 and 6 explain how the UFSC system works and what strategies it employs, as well as the details and reasoning behind the implemented changes. Chapter 7 contains the specifics of the testing process, along with its results. Chapter 8 is devoted to discussing the results, multi-agent systems, our experiences throughout the course of the project and more. Finally, chapter 9 gives a broad perspective view upon the project and artificial intelligence/multi-agent systems in general, while chapter 10 provides concluding remarks for this thesis.

1.2 Artificial intelligence - a historic perspective

The term 'artificial intelligence' (AI) was first used in the 1950s, describing it as 'the science and engineering of making intelligent machines'. Since then, research has been conducted consistently, resulting in a significant optimism, development and growth of this part of computer science; although, several severe stagnation and disillusionment periods have occurred as well.

An assertion stating that creating a machine with general intelligence, i.e. human-like intelligence, is feasible, is often considered the basis of early AI research. An excerpt from the Dartmouth Conference proposal from 1955 states:

> "The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." [SMRM55]

The Dartmouth Conference participants discussed many of the central questions and aspects of AI. The premise and proceedings of this conference are considered to have laid basic foundations for AI as a field within computer science.

Today, over half a century since the aforementioned conference, an artificial general intelligence (AGI), also referred to as 'strong AI', is, yet to be created. Regardless, specific applications of 'weak AI' in systems are nowadays commonly used for a wide range of problem solving purposes.

The ongoing research within artificial intelligence has spawned a substantial number of approaches and paradigms for solving specific problems and in the end, ideally, achieving the ultimate goal of creating the AGI. On the other hand, a multitude of questions of philosophical and ethical nature have come up. These, along with a precise description of AI research history and advances,

are all beyond the scope of this thesis. This thesis pertains to a general field called the agent-oriented programming (AOP) [Sho93] and more specifically – multi-agent programming (MAP).

Chapter 2

Multi-agent programming

This chapter introduces the basic theory behind multi-agent systems (MAS). A multi-agent system is defined to contain a number of agents (see section 2.1) placed in an environment (see section 2.3). These agents are able to communicate (see section 2.4), interact with each other and perform actions that affect and/or change the environment.

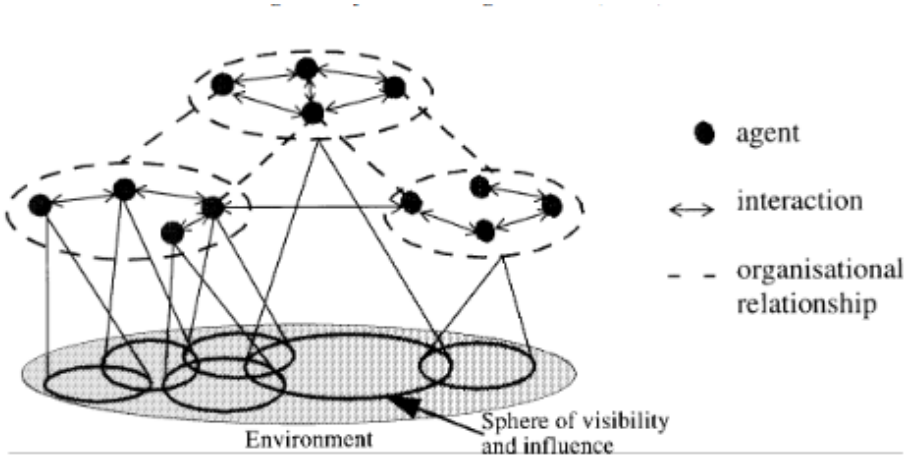

Figure 2.1: Structure of a multi-agent system (from [Jen00]).

2.1 Agent - definition

For purposes of this thesis, a precise definition of the term 'Agent' is indispensable. The continuous usage of this term in various contexts inside and outside of the AI field has caused the term to become ambiguous at best. In this case, an agent must be a software based entity, placed in a simulated, virtual environment, preferably along with other agents. An agent must also have the following properties, as defined by Michael Wooldridge, in [Woo95]:

Autonomy
: It is not necessary for humans to monitor the agent; it carries out its actions and makes decisions independently.

Social ability
: The agent is capable of communicating and interacting with other agents and/or humans.

Reactivity
: The agent is capable of perceiving its environment and reacting to the changes and events taking place in it.

Pro-activeness
: Instead of merely passively reacting to events (see previous item in the list), the agent can initiate actions by itself and thus actively pursue any number of goals.

Additionally, to make the definition of an agent stronger, it shall be extended by using one of the many existing models for expressing mentalistic notions.

2.2 The BDI model

The Belief-Desire-Intention (BDI) [RG91, RG95] was chosen as the underlying model for the agents in this thesis. Initially, the model was developed to describe parts of human behaviour [1]; however, its main parts have quickly become adopted for work with rational agents within software.

[1] In 1987, under the name "Intention, Plans, and Practical Reason" [Bra87] by Michael Bratman, a philosophy professor at the Stanford University.

2.2.1 Beliefs

An agent has a set of beliefs about itself, the environment it is placed in, about other agents and anything else of relevance. The agent can be given in code a set of initial beliefs by the programmer. The possible ways of obtaining more beliefs during runtime are twofold: by perceiving the environment and by receiving information from other agents. It is important to note, that the beliefs an agent holds are not necessarily true; for example, the agents perception may be faulty or a piece of information another agent has sent may be wrong - either voluntarily (a malicious agent) or involuntarily (the agent had wrong information, but believed it to be true).

2.2.2 Desires

The desires represent the goals (i.e. the states of matters) an agent would like to achieve. The desires do not have to be mutually exclusive; the agent may wish for conflicting, or even outright opposite outcomes. In essence, having a desire only means that it will have an influence on the agents decisions and not that an agent has decided to actively work towards achieving it. The desires can be given to the agent initially by the programmer or later obtained by the agent itself.

2.2.3 Intentions

The set of intentions consists of goals an agent is currently working on to achieve. In other words, the intentions represent what the agent has decided to commit to do. The intentions are created by an agent deciding to act upon its desires.

2.3 Environments

The agents are usually placed in an environment of some sort. It can be a real world, physical application, such as automated manufacturing, sorting or packaging plants; a slightly more abstract, but still real application, such as the Internet; or finally, a completely virtual one - simply a simulation. Despite the immense diversity of possible environments, there are a few key similarities, which apply to them all in agent-environment relations. The agents are capable of performing actions and changing the state of the environment in some,

most often limited, way. The agents desires are usually associated with the environment. Thus, an agent will typically commit to a course of action which will (hopefully) bring about a desirable state of the environment. Since in the MAS, as the name implies, multiple agents are placed in the same environment, their respective actions may counteract or assist each other. That is why cooperation and communication between agents are extremely important aspects of multi-agent systems.

2.4 Communication

Agent communication languages such as KQML (Knowledge Query and Manipulation Language) [2] and FIPA (proposed by the Foundation for Intelligent Physical Agents) [3] enable the agents to communicate with each other by exchanging messages with a predefined format. The messages can be used for information sharing, requesting actions, declaring intentions and commitments. The messages usually include following elements:

- the intended receiver(s)
- the message type (a request, an information etc.)
- the message contents

The subsection 4.1.3 describes how the agent communication is implemented on the Jason platform.

[2] http://www.csee.umbc.edu/csee/research/kqml/
[3] http://www.fipa.org/repository/aclspecs.html

Chapter 3

Multi-agent programming contest

The Multi-Agent Programming Contest (MAPC) [1] has since 2005 provided an opportunity for teams from around the world to develop complex multi-agent systems and to test their performance against each other in a beforehand specified, competitive scenario.

3.1 Scenario

The current contest scenario is entitled 'Agents on Mars'. The area the competition takes place in is represented as a weighted graph. Each vertex represents a water well, where the weight of the vertex determines how good the water source is. The edges between the vertices are weighted as well - they represent the cost of travelling between the two vertices. The graphical interface of the contest platform can be seen in figure 3.1.

Two teams are set to compete against each other in achieving the highest score possible. For details on how the scores are calculated, see subsection 3.1.3.

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

When the game begins, the teams posses no knowledge of the graph or the opposing team. The structure of the graph, as well as all values of vertices and edges have to be discovered; the same applies to the specifics of the opponent agents.

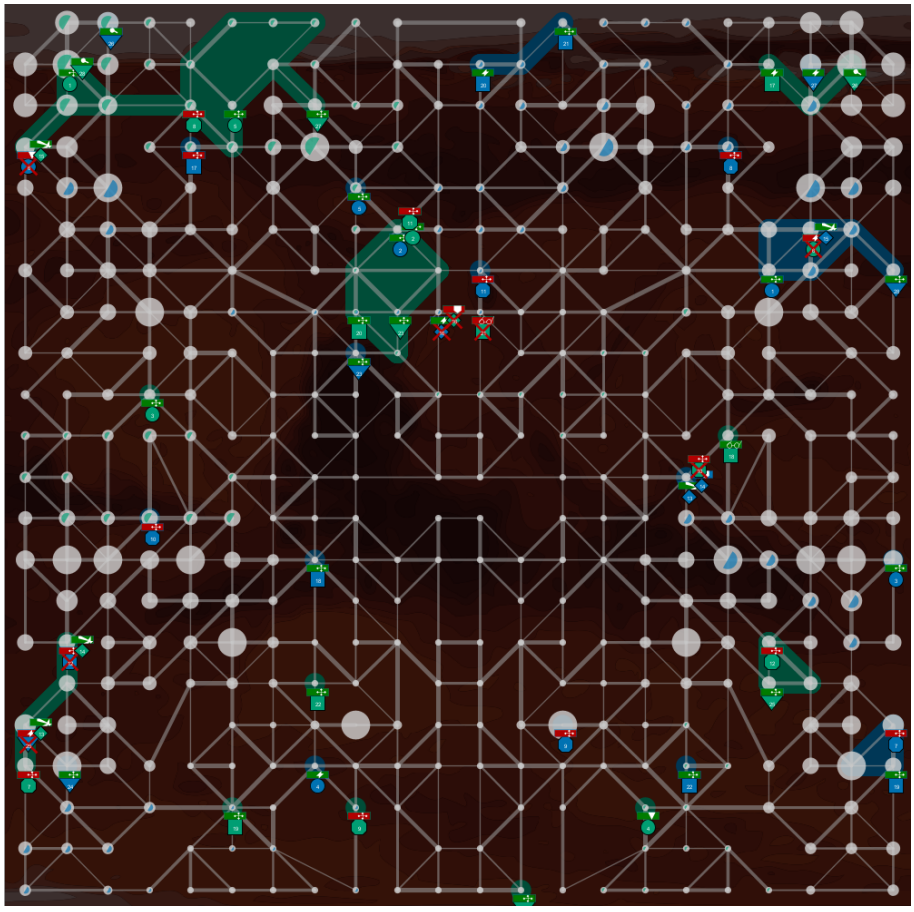

Figure 3.1: The graphical interface for the contest.

3.1.1 Agents and actions

A team consists of 28 agents. There are 5 agent types with unique features:

Agent	Energy	Health	Strength	Visibility	Available actions
Explorer	12	4	0	2	skip, goto, **probe**, buy, survey, recharge
Inspector	8	6	0	1	skip, goto, **inspect**, buy, survey, recharge
Saboteur	7	3	4	1	skip, goto, parry, buy, survey, **attack**, recharge
Repairer	8	6	0	1	skip, goto, parry, buy, survey, **repair**, recharge
Sentinel	10	1	0	3	skip, goto, parry, buy, survey, recharge

Table 3.1: Agents and their attributes.

Both teams start with 6 agents of every type, except for saboteurs, of which there are 4. The agents have following attributes:

Energy
Necessary for being able to perform most actions.

Health
Used to endure attacks; when fully depleted, an agent becomes disabled.

Strength
Defines how powerful an attack of the agent is (only applicable to saboteurs).

Visibility range
Defines how large the field of view of the agent is. It is also used when calculating the chance of success and effect of ranged actions.

The action functions are as follows:

Skip
As the name implies, simply skips the turn; doing nothing.

Recharge
Replenishes some of the agents energy.

Goto
Moves the agent to a chosen neighbouring vertex. The energy cost equals the value of the traversed edge.

Probe
Used to discover the value of a given vertex. Costs 1 energy point to use.

Survey
Reveals weights of edges close to the agent [2]. Costs 1 energy point to use.

Inspect
Used to obtain knowledge about the attributes of a rival agent. Costs 2 energy points to use.

Attack
Attacks a chosen rival agent. Costs 2 energy points to use.

Parry
Defends the agent against an enemy attack. Costs 2 energy points to use.

Repair
Restores health of a friendly agent; it is not possible for an agent to repair itself, though. Costs 2 energy points to use. If the agent is disabled, then the energy cost is 3 points.

Buy
Allows to purchase upgrades for the agents attributes. Costs 2 energy points and 2 achievement points to use.

Skip and recharge actions do not cost anything to use. If an agent becomes disabled, it loses ability to perform most actions until it is reactivated by a repairer agent. The actions available to a disabled agent are: skip, recharge (albeit with a reduced effect), goto and repair [3].
Some actions are ranged, i.e. the target does not need to be on the same vertex as the agent performing the action. However, as the range from the agent increases, the actions failure probability grows, while the effects decline. The ranged actions are: probe, inspect, attack and repair. In addition to failure possibility due to range, any action can fail with 1% probability.

[2]Depending on the visibility range attribute, the chance of revealing more edges further away from the agents position is increased.
[3]Repairer agents only.

3.1.2 Controlling vertices

Control over vertices is determined by the 'graph colouring algorithm' incorporated in the scenario - see figure 3.2 a), b) and c). The algorithm proceeds as follows, considering only agents that are not disabled:

1. The control over vertices with agents on them is distributed. A vertex is under control, if one of the teams has placed a larger number of agents than the opposing team on it; in case of equality, the vertex remains unclaimed.
2. The control is extended to empty vertices which have at least two neighbouring vertices already under control by a team. In case of an empty vertex being a neighbour to vertices controlled by both teams, the control is granted to the team with the larger number of these.
3. If the vertices controlled by one of the teams create an enclosed area of the graph without rival agents inside, then control over this entire area (called a 'zone' or a 'frontier') is granted to that team.
4. If none of the above applies to a vertex, it remains unclaimed.

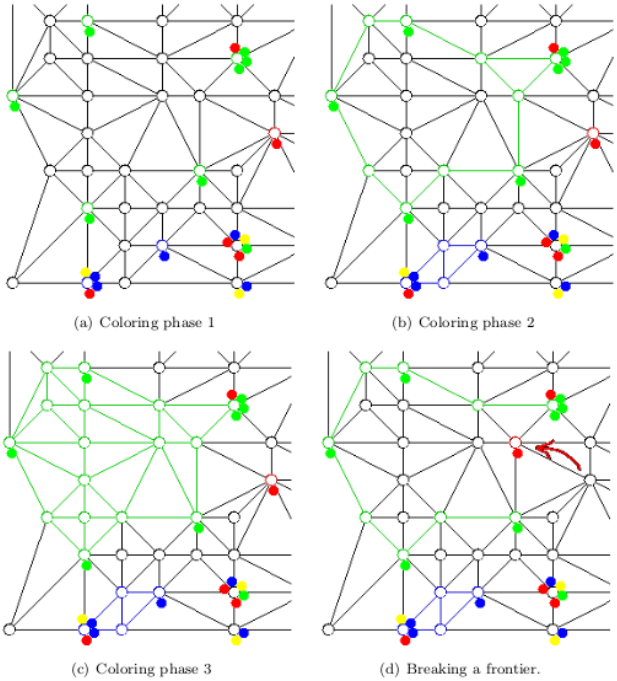

Figure 3.2: The 'graph colouring algorithm' (from [DKS13]).

Controlling frontiers may be challenging. Figure 3.2 d) shows such a situation. The green team no longer encloses a sub-graph without any rival agents inside, as one of the red agents has placed itself very cleverly and thus 'broken' the frontier.

3.1.3 Score

Two factors contribute to a teams score - the controlled zones and the money holdings. The final score is calculated by summing the values of these two factors for each turn of the simulation. The mathematical representation of the score formula looks as follows (from [DKS13]):

$$score = \sum_{s=1}^{steps} (zones_s + money_s) \tag{3.1}$$

Money is earned by completing pre-specified achievements. These achievements include surveying and exploring the map, inspecting enemy agents, successful attacks, successful parrying, and more. For a complete list of the available achievements, the reader is referred to the scenario specification [DKS13]. Money can be used to buy upgrades for the agents, or saved in order to contribute to the teams score.

3.1.4 The simulation

The simulation is turn based and handled by a dedicated server. A simulation consists of 750 steps/turns. The opposing teams connect to the server and for each turn are obligated, in a timely manner, to upload the actions their agents decided to perform. The server then processes the actions and sends the updated state of the environment, in form of agent percepts, back to the teams. Formally, the algorithm for each turn proceeds as follows (defined in [DKS13]):

1. Receive the action choices from all agents.
2. Apply the random failure chance to each action.
3. The attack and parry actions are executed.
4. Find out which agents have become disabled.

5. All remaining actions are executed.
6. The percepts are prepared.
7. The percepts are delivered to the agents.

Starting a simulation requires opening several terminal emulator windows in separate directories and invoking appropriate commands to run the server, GUI (optional) and both teams. A simple shell script has been written by us, in order to automate this process (see appendix D.4).

3.2 Future contests

In a recent announcement, a brand new scenario for the future contests has been unveiled. As of this writing, little is known about it - other than that it shall be entitled 'Policemen and Robbers' and take place in a fictional urban environment. The scenarios début is planned for the 2015 multi-agent programming contest, while the 2014 contest will only be held for entertainment purposes and use the 'Agents on Mars' scenario, with only very small changes.

Chapter 4

Jason

This chapter provides a brief introduction to the Jason programming language, along with the platform it operates on.
Jason is used for programming rational agents and multi-agent systems. Systems developed for the multi-agent programming contest (see chapter 3) in Jason have participated on numerous occasions and won the competition twice in the 'Agents on Mars' scenario, in 2012 and 2013 [BJV10, HB10, ZBS+13].
Jason uses a Java-based interpreter for an implementation and extension of the Agent Speak(L) abstract agent programming language, which was introduced in 1996 by Anand S. Rao [Rao96]. AgentSpeak(L) is based on logic-programming and the BDI architecture.
The recurring work of reference for descriptions given in this chapter is the book [BHW07].

4.1 Jason basics

The basis of any Jason project is a .mas2j configuration file, which specifies the infrastructure, environment, participating agents, graphical user interface and other settings pertaining to the given project. While some components, such as the GUI are optional, a project has to include the agents, whose code is written

in the Jason programming language itself, often supported by code written in Java. The Jason code consists of the agents initial beliefs, rules, goals and plans.

4.1.1 Beliefs and rules

The beliefs are represented by Prolog-like facts, for example: capital(Denmark, Copenhagen) or animal(cat). The rules are predicate expressions used to describe a relationship among facts by using logical implication (:-) [1].

```
1 student(N) :- attends_university(N) & owns(N, books) &
2               has(N, beer).
3 affordable(I,P) :- P < 50.
```

Listing 4.1: Example rules.

i.e. N is a student, if he/she attends university, owns some books and has beer. Item I is affordable if its price P is less than 50.

4.1.2 Goals and plans

The goals, initially given to the agent by the programmer, are present in the code. The agent may later on adopt or be delegated new goals, but these do not appear in the code.
The plans represent the possible courses of action an agent may undertake in response to an event or in order to achieve a goal. The plans intend to prepare the agent for the situations it may find itself in. In a simulated scenario, such as the multi-agent programming contest, the number of possible events is finite and thus the agents can be programmed precisely. An agent can only operate and solve problems within its capabilities, as defined by the plans. The agents can, in fact, exchange knowledge about plans between each other; however, ultimately, the scope of the plans available to a collection of agents in an environment consists of plans supplied by the programmer. This is the main limitation and a principal problem with constructing multi-agent systems for the truly complicated practical applications which are inherently non-deterministic and the possible number of events and contexts is infinite.

[1] http://www.cs.trincoll.edu/~ram/cpsc352/notes/prolog/factsrules.html

```
1 +!no_money : has(job,Me) <- +work_more.
2 +!no_money : not has(job,Me) <- !!find_job ; !!do_budget.
3 -!no_money : true <- +panic ; !call(parents).
4 +!find_job : true <- !!search_jobs ; !write_CV ; !apply.
5 +!do_budget : true <- ?spendings(Me,S) ; +reduce(S).
```

Listing 4.2: Example plans.

A plan consists of following components:

- The triggering event (when is the plan relevant?)
- The context (in which situations is the plan applicable?)
- The contents (what to do when the plan is employed?)

In the listing 4.2, there are three plans for the lack of money situation, a plan on how to find a job and a plan on how to make a budget. Suppose a +no_money event occurs; there are two plans **relevant** for handling it (the first two plans in the listing). The context defines in which situations a given plan is **applicable**, e.g. the first plan is only when the agent has a job, while the second plan only if it does not have one. The plans where context is 'true' are applicable in any situation (as long as they also are relevant in the first place).
The contents of the plan (the part after the <-) consist of steps required for completing the given plan. So, in the first example, the recipe for solving the lack of money situation is to add a mental note to work more, while the second plan calls for employing additional plans - to find a job and to create a budget. Finding a job and creating a budget may in turn require additional actions and/or plans to be employed. For instance, finding a job will require browsing through available offers and writing a CV (not necessarily one after another; these tasks are compatible as side-by-side intentions). Only when these tasks are finished, the application process may begin.
The ?spendings(Me, S) is a test goal, which retrieves desired information from the belief base - in this case, the figure for the spendings, S. Subsequently, a mental note of reducing the S value is added to the belief base with the +reduce(S) event.
Finally, -!no_money plan is a failure handler; in case the other plans for that event fail, it is used. In this example, the failure handler calls for the agent to employ a dubious strategy of panicking and calling its parents to resolve its financial problems.

An advantage of the Java implementation is that certain tasks do not have

to be handled by the AgentSpeak(L) based architecture, but can instead be delegated to be resolved with Java code. This feature allows to make use of Java and object-oriented programming to implement parts of system, which AgentSpeak(L) is not well equipped for. This includes, but is not limited to the interpreter, environments, graphical user interfaces, more complex algorithms (such as Dijkstra) and the agents internal actions. Jason includes a number of built-in standard actions for the most common purposes, such as communication between the agents, list and string manipulation, plan library manipulation, BDI, term type identification, as well as some miscellaneous ones. It is also possible for the programmers to write internal actions of their own.

4.1.3 Agent communication

The agents communicate with each other by using .send and .broadcast internal actions. Both actions contain as parameters the illocutionary force and some content, such as a string or a literal. The .send action must also include the intended receiver, while the .broadcast action simply sends the message to all agents in the environment. The illocutionary force defines the objective of the message. Jason allows for a number of possibilities, as listed below (the illocutionary forces are written with bold text).

Tell and **Untell** are used to share and remove beliefs, respectively.
Achieve and **Unachieve** delegate and ask to abandon goals, respectively.
AskOne provides a statement in the messages content and expects the receiver to reply as to whether it is true or false, according to the receivers belief base.
AskAll inquires for all answers a receiver can provide to a given statement.
TellHow shares a plan with the receiver, while **UntellHow** asks for a plan to be ignored. Finally, **AskHow** inquires for plans suitable for handling a given event.

4.2 Jason interpreter

The Jason interpreter is implemented in Java. Advantages thereof are twofold: it makes it possible to use Jason on any operating system/platform and also makes it easier to understand and modify the interpreters code.

4.2.1 Reasoning cycle

The agents repeatedly undergo a reasoning cycle, in which beliefs are updated and considerations for further actions are made. The cycle broken down into principal steps looks as follows (as listed in [BHW07]):

1. **Perception**
   The process of collecting the available information about the environment.

2. **Belief update**
   The old perceptual information gets deleted and the new data (from step 1) is added to the belief base.

3. **Receiving communication**
   Checks the agents 'mailbox' for incoming messages.

4. **Selecting acceptable messages**
   The process of making decisions on whether or not the messages are 'socially acceptable', for example based on the level of trustworthiness of the sender. By default, however, all messages are accepted.

5. **Selecting an event**
   The process of choosing an event, such as a change in the environment, which is to be handled in the current reasoning cycle. One event is chosen in every cycle.

6. **Retrieving relevant plans**
   The process of looking through the plan base in order to find plans relevant to the event chosen in step 5.

7. **Determining the applicable plans**
   From the set of relevant plans found in step 6, only plans applicable to the current situation and/or the state of the environment are found.

8. **Selecting one applicable plan**
   One of the applicable plans found in step 7 is selected using the selection function (see description 4.2.2).

9. **Selecting an intention for further execution**
   From the set of intentions, which is merely a stack of partially instantiated plans, one intention is chosen for the current cycle.

10. **Executing one step of an intention**
    One action from the chosen intention is executed.

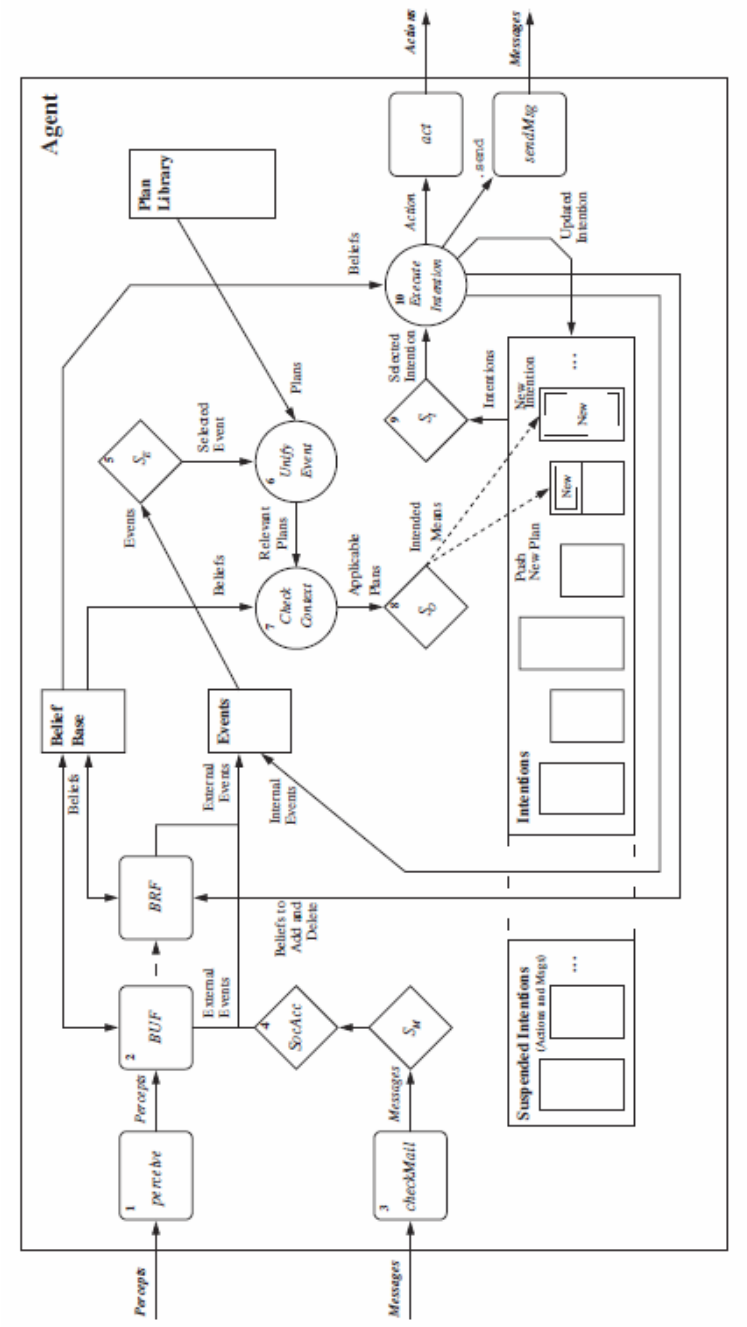

Figure 4.1: The Jason Interpreter (from [BHW07]).

4.2.2 Interpreter modifications

The figure 4.1 visualizes the steps listed in description 4.2.1 and provides a 'whole picture' abstract graphical view of the interpreter. It is important to note, that the interpreter is implemented in such a way that allows numerous customizations by the programmer. Below is a list of the interpreter parts which can be altered, with some examples of possible and common changes (as shown in [BHW07]):

Perception
It is possible to change the way an agent perceives its environment. For example, by simulating faulty percepts or limiting the number of perceived elements/parts of the environment.

Belief update and revision
It is possible to change how the percepts are added to the belief base and how the beliefs in the base are revised. For example, some beliefs may be discarded based on certain criteria, the base may or may not be allowed to hold conflicting beliefs, or the frequency of belief base revision may be changed.

Communication
It is possible to change all functions related to agent communication. For example, messages may be prioritized by their importance, messages from certain agents can be discarded, messages can be deemed socially (un)acceptable based on arbitrarily specified factors, or the agent may be disallowed to communicate with certain agents, or its communication options could be impaired, e.g. by only allowing certain message types to be sent.

Selecting events, plans and intentions
It is possible to change functions concerning selection of events and intentions. For example, events can be prioritized arbitrarily or even ignored completely. By default, if given several applicable plans, the interpreter chooses one based solely on its location in the code (i.e. first one gets picked). For some applications it could be beneficial to employ additional considerations in such situation. Intentions can be prioritized, similarly to events. Also, commitment to intentions can be simulated; for example, the agent could be forced to pick the same intention in every cycle, until that intention has been brought to completion.

It should now be clear that the Jason interpreter gives vast possibilities for highly customized and specialized agents in multi-agent systems. Making changes to

the specific parts of the interpreter is fairly straightforward. The whole process consists of creating a Java class, which extends the default version and then overriding the unwanted standard procedures or implementing additional functions.

4.3 JaCaMo

Multi-agent programming paradigm consists of four distinct dimensions, as defined in [BBH+13]:

- Agent-oriented programming languages
- Interaction languages and protocols
- Environment architectures, frameworks and infrastructures
- Organisation management systems

Plenty platforms based on each of these dimensions exist, however, systems comprising multiple or all of them are scarce. JaCaMo puts three projects together: Jason, CArtAgO and Moise+, effectively covering three of the four dimensions (agents, environments and organisation). Incorporating the interaction dimension, which is currently handled by Jason communication features (see subsection 4.1.3), is being worked on by the developers of the JaCaMo platform.
CArtAgo [RPVO09] is designed for programming environments and artefacts, while Moise+ [HSB07] is used for programming organisations. Since the agents in MAS almost always are situated in different sorts of environments and often are organized in some way (teams, sub-teams, missions/goals), the unified platform provides extremely valuable resources; programming a complicated environment in a general purpose, object-oriented language (such as Java for the Jason platform alone) or a sophisticated organisation in an agent-oriented language (such as Jason) is, at best, cumbersome. The multi-agent system described in chapter 5 is written using JaCaMo. However, as the main subject of this thesis is programming using Jason alone, going into details about Moise+ and CArtAgO is refrained from.

4.4 Jason IDE

A Jason distribution comes with its own integrated development environment (see figure 4.2), which is implemented in jEdit [2]; a plug-in for Eclipse [3] is also available. These come with all tools necessary for creating multi-agent systems, agents themselves and simple environments. An extremely useful tool is the mind inspector (see figure 4.3), which is used for debugging; it allows to run agents cycle by cycle and to check their mental states at any given time during execution. The mental state encompasses the agents current beliefs, desires, intentions, rules, plans and mailbox.

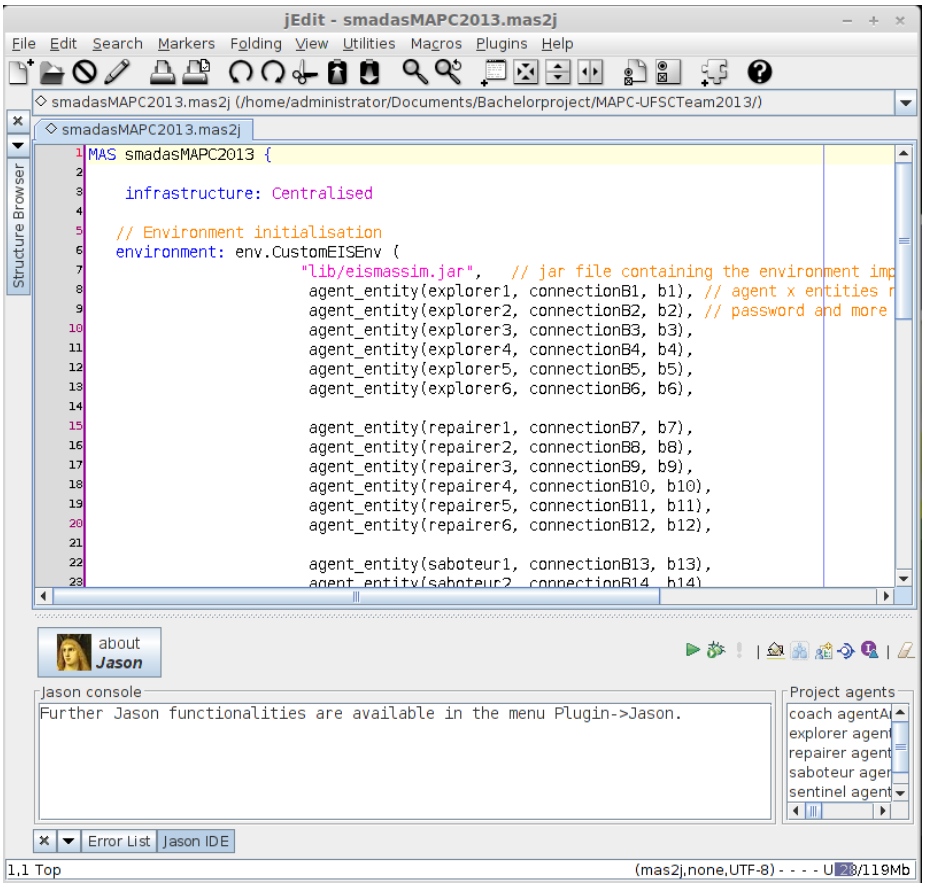

Figure 4.2: The Jason IDE.

[2] http://sourceforge.net/projects/jason/files/
[3] http://jason.sourceforge.net/eclipseplugin/juno/

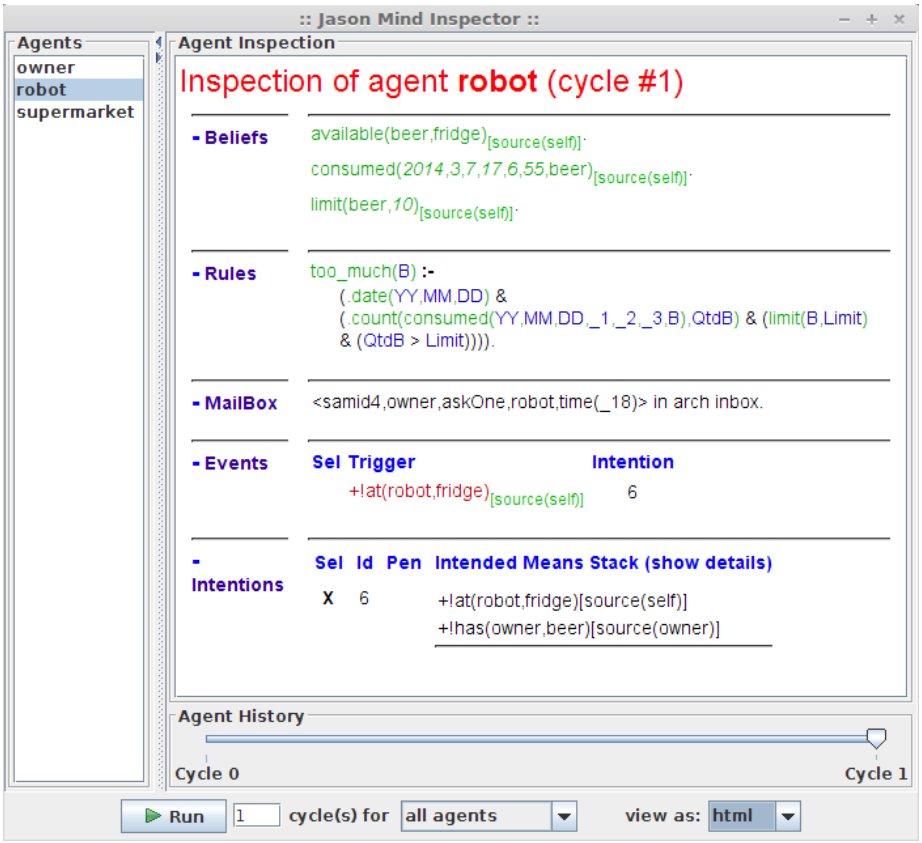

Figure 4.3: The Jason mind inspector.

Chapter 5

The UFSC system

In this chapter, the strategies employed by the UFSC team and its agents are outlined. The descriptions consist of our own analysis of the code and contest scenario, supported by the information from UFSC teams article published after the multi-agent programming contest. In this article, some of the thoughts and considerations behind the system are conveyed [ZBS+13].

5.1 Plan selection

An important matter to address first, is how the agents choose the plans in each step of the simulation. In the UFSC system, the selection of action plans is somewhat simplified. All such plans are relevant, since the triggering event for them all is the same: +select_goal. The decision is solely dependent on the applicability of the plans, as defined by the specified rules (see listing 4.1 for examples of rules). Often, more than one plan will be applicable. In such cases, it is possible to write a user-defined selection function and prioritize plans. However, in the UFSC system, a neat solution was employed - simply coding the plans in the order of their priority, knowing that the default action of the Jason interpreter in cases of multiple applicable plans is to pick the one situated earliest in the source code.

5.2 Discovering the map structure and rival team

In the beginning of each round, the agents posses no knowledge of the map. Therefore, they have to discover the values of vertices and edges as a team. Furthermore, the agents have to obtain information about rival agents to begin with, since they are not given any details about agent types and their attributes (health, strength etc.).

5.2.1 Surveying

Surveying the edges between the vertices is very important in the beginning of each contest round. Every agent has the capability to perform the survey action; however, depending on the agent type, the contribution differs. The explorers do not survey at all, since they are committed to another crucial task, which is probing the vertex values. All other agents can choose to survey if they do not have anything more important to do. Generally, the survey action has a low to middle priority amongst the agents. Additionally, no agent type has an actual commitment to surveying the map, which is hence performed in an *ad hoc* manner.

5.2.2 Probing

The explorers are the only agents capable of probing and it is therefore their duty to discover the values of all vertices as soon as possible. It is essential to do so, in order to calculate the best zones to conquer and to receive full points for the controlled vertices. Four explorers are allowed to wander around, with focus on actively seeking out unprobed vertices and uncovering their values, while the remaining two have a special assignment, which is probing the already controlled zones, in order to start receiving full points for these zones quickly. From the organizational point of view, the Moise$^+$ specification obliges the explorers to commit to the mission of probing all vertices in the graph (see section 3.1).

5.2.3 Inspecting enemies

The only agents capable of inspecting enemies are the inspectors. Any non-inspected rival agent is considered dangerous (same as a known saboteur), which disrupts allied agents, as they will typically parry or run from such an agent,

sometimes unnecessarily. Additionally, the UFSC teams saboteurs have a preference in choosing the targets based on their type. Thus, it is vital that enemy agents get inspected as quickly as possible, so that the saboteurs can harass the correct enemies and so that the rest of the teams agents know whether a nearby rival agent endangers them or not. According to the UFSC team, the strategy is to inspect each agent only once, mostly in order to determine the agent type. The UFSC team has decided not to concern itself with possible upgrades purchased by the enemies after they have been inspected. The UFSC teams inspectors will typically perform the inspect action when they are standing on the same vertex as an uninspected enemy. They may also semi-actively pursue targets to inspect; for example, if an enemy agent comes into the inspectors field of view, it may decide to go towards the agent and inspect it when possible. The UFSC team uses a CArTagO artefact, which keeps track of the inspected agents. From the organizational point of view, the Moise$^+$ specification additionally obliges the inspectors to commit to a mission of discovering all rival saboteurs.

5.3 Other strategies

The UFSC team employs a number of other strategies for a range of purposes. The preceding sections described the strategies relevant to the work carried out during this project (implementing changes in these strategies and creating new ones). This section provides a quick overview of the most important of the remaining strategies.

Attacking
: The saboteur agents never use the parry action; they attack enemy agents regardless of the risk to themselves. They prioritize targets in the following order (most important first): saboteurs, explorers, inspectors, repairers, sentinels. The saboteurs are also organized with separate roles: the 'leader' and 'helper' tend to stay close to own zones and defend them, while a 'chaser' saboteur tends to go around harassing enemies.

Repairing
: The 'repairer leader' agent coordinates the entire process. A disabled agent has to contact the 'repairer leader', who then delegates the task of repairing this agent to the nearest repairer agent. After that is done, the chosen repairer and the disabled agent start to move towards each other.

Buying upgrades
The UFSC team does not use the possibility of purchasing upgrades for its agents at all.

Controlling map zones
The UFSC team distinguishes between three types of zones: islands (zones that can be controlled by one agent), pivots (zones that can be controlled by two agents) and hills (zones requiring more agents to control). The agents employ sophisticated calculations and communication protocols in order to take and maintain control of the best zones possible. The UFSC team generally concentrates on controlling small, high value zones that are easier to protect, apart from the beginning of the simulation, where not all vertices have been probed and the hills are more profitable.

For more details about all these strategies, the reader is referred to the UFSC systems code and the article in which the UFSC team itself describes the implementations [HZB+13, ZBS+13].

Chapter 6

Changing the UFSC system

This chapter describes the alterations made to the UFSC system, as well as the reasoning behind them. Furthermore, implementation details of these changes are discussed. The first three sections (6.1, 6.2, 6.3) consist of changes whose implementation marks the minimal goal of this thesis, while the section 6.4 describes additional implementations. The statistical results of the implementations performance in tests are shown in chapter 7.

6.1 Surveying

In the preceding chapters, it has been established that quickly gaining knowledge of the graph is extremely important. Doing so benefits the team as a whole, since the agents ought to be able to use the shortest paths between vertices in the graph, thereby reducing the time and energy consumption for reaching their targets. This part of the UFSC teams strategy was the first matter looked into. The agents do not have a strict surveying scheme, and merely perform the surveys as needed - usually in the early stages of the simulation, where the agents have nothing better (i.e. the agents respective responsibilities, such as inspecting, attacking and repairing) to do. Additionally, the explorers do not perform surveys at all, while for all other agents except sentinels it is an action

of a middle to low priority, which can occur only if an agent is standing on a vertex with some unsurveyed edges to it and has nothing more important to do; in other words, **the agents do not actively look for edges to survey**. Instantly, an opportunity of possible change presented itself. It could be beneficial to allow some agents to actively pursue the task of discovering the unsurveyed edges, at least in the early phase of the simulation. Natural candidates for such task are the sentinels, as they have the largest field of view (see table 3.1). Furthermore, the sentinels do not have an unique role to play (e.g. only explores can probe etc.), unlike all other agents. It must be remembered that the agents are autonomous, i.e. they choose courses of action to commit to by themselves. Therefore, changing their behaviour consists of adding some rules and plans to their code, in such a way that the selection function (see description 4.2.1) of the agent will pick the desired plan when appropriate.
The idea is a two-step procedure, in which the agent first finds out whether the neighbouring vertices are in need of a survey action, finds the best one (i.e. the one with most unsurveyed edges), goes there and in the next turn performs a survey, unless it becomes disabled, needs to recharge or parry an attack. The procedure is employed only in the early stages of the contest, meaning no later than the 133rd step of the simulation (explanation in subsections 7.3.2 and 8.3.1). At this point, it becomes more important to use sentinels for defending the controlled zones. The listings 6.1 and 6.2 outline this strategy.

```
1 begin
2     list neighbouring vertices
3     for each vertex do
4         find number of outgoing unsurveyed edges
5     choose vertex with most unsurveyed edges
6 end
```

Listing 6.1: Rule for finding the best vertex with unsurveyed edges to go to.

```
1  begin
2      step 1:
3          if found the best vertex and turn < 133 then
4              goto vertex
5          else
6              plan not applicable
7      step 2:
8          if not has other more important tasks then
9              survey the edges
10         else
11             choose the more important task
12 end
```

Listing 6.2: The procedure for active surveying.

As the new surveying strategy has a quite high priority, the corresponding plan has been placed directly beneath the highest priority plans. These plans are:

- Recharging
- Waiting for, or going to a repairer
- Parrying
- Surveying

The chosen placement is obvious. The plan cannot be executed if the agent is lacking energy or is disabled. Executing the plan when under risk of being attacked may be ill-advised, while doing so when already standing on a vertex in need of edge surveying would be a waste of time.
The Jason code for the strategy can be found in appendix B.1.

6.2 Inspecting

Similarly to discovering the graph structure, gaining knowledge about the enemy entities is important (see subsection 5.2.3). The UFSC team generally manages to inspect enemy agents quite quickly; however, occasionally, some of the enemies elude the inspectors for a long time into the simulation, effectively delaying the team in reaching the achievement for inspecting 20 enemies and/or in inspecting all 28 enemy agents.
The possible reason for some of the enemies avoiding inspection may be the decentralized inspecting strategy. In the original system, the inspectors only inspect the enemies once they come in sight, i.e. place themselves on the same vertex as the inspector or on a vertex adjacent to it. Hence, it is clear that inspecting all enemies is, at least to some degree, luck dependent, e.g. if an enemy stays out of all inspectors' sight, it consequently will avoid inspection. The current strategy seems to work out well in the beginning of the simulation, when there are many not inspected enemies, though sometimes it does not perform too well in the later stages of the simulation. Therefore, it may be beneficial to centralize the inspecting strategy in the later stages of the simulation somewhat. It has been decided to employ such an approach after the 50th step of the simulation (explanation in subsections 7.3.3 and 8.3.1). The idea is to have all friendly agents on lookout for not inspected enemies. At the beginning of each turn, the agents send a message informing the 'inspector leader' about not inspected enemies they spotted. The 'inspector leader' then holds **auctions** [BFPW03] for all the other (available) inspectors. During an auction,

the agents 'bid' against each other in order to get assigned to a given task. In this simplified case, the inspector agents simply bid with their distance from the entity which needs to be inspected. Thus, perhaps counter-intuitively to an actual auction, the lowest bidder is chosen and delegated the task by the 'lead inspector'. This strategy could ensure quicker inspection of the last few elusive enemies. The process of agents informing the 'lead inspector' about entities (see listing 6.3) and the auctions themselves (see listings 6.4 and 6.5) do not collide with the agents actual plans and actions; they are simply performed 'in the background'. The only agent actually undertaking an action is the winner of an auction, when it commits to go to inspect the enemy entity (see listing 6.6).

```
1 begin
2     when can see unknown entity and turn > 50 do
3         tell location of entity to lead inspector
4 end
```

Listing 6.3: Informing the 'lead inspector' about an entity.

```
1 begin
2    when got reports about enemies from all agents do
3        list all entity reports and remove duplicates
4        for each report do
5            start auction and collect bids from inspectors
6            find the lowest bidder (winner)
7            inform the winner
8 end
```

Listing 6.4: Auction by the 'lead inspector'.

```
1  begin
2     when auction do
3         if has other more important tasks then
4             skip auction - place bid(max)
5         elseif already won an auction in this turn then
6             if current auction is better
7                 find shortest path to entity
8                 place bid(length of path) and await result
9             else
10                skip auction - place bid(max)
11        else
12            find shortest path to entity
13            place bid(length of path) and await result
14 end
```

Listing 6.5: Participating in auction for inspectors.

```
1 begin
2     if won auction for entity at destination vertex then
3         goto next vertex on path to destination vertex
4     else
5         choose some other plan to employ
6 end
```

Listing 6.6: Inspectors using a plan after auctions.

As with the sentinels (see bullet point listing 6.1), the strategy must be appropriately prioritized. There is a number of more important plans:

- Recharging
- Waiting for, or going to a repairer
- Inspecting an enemy in immediate vicinity
- Protecting an island

The plan for delegated inspecting is prioritized lower than the plans mentioned above. The chosen priority is entirely natural - the strategy could not be employed when the agent lacks energy or is disabled; doing so instead of inspecting an enemy within an immediate vicinity is illogical. The island protection has been given a high priority by the UFSC team itself and it has been decided not to alter on that.
The Jason code for the strategy is located in the appendix B.2.

6.3 Exploring

Probing vertex values quickly is one of the most important tasks in the simulation. Only once a vertex gets probed, the team begins to receive full points when it is under the teams control. In general, the explorers manage to probe the entire map quite efficiently; however, sometimes they tend to follow each other into same areas of the map, instead of spreading out more. Ensuring that the explorers always spread and thus all probe different regions of the map could help them with the task of probing the entire map faster. After that goal is accomplished, the explorers obviously would not need to be forced to spread out any more. The strategy would simply be deactivated after a certain number of steps into the simulation or once the map is fully explored. It has been decided

to employ such an approach until the 50th (explanation in subsections 7.3.4 and 8.3.1) step of the simulation. Afterwards, the explorers can move freely when exploring or pursuing other tasks, such as helping to control the zones. The listing 6.7 outlines this strategy.

```
1 begin
2     when not vital goal and want to move and turn < 50
3         choose destination vertex
4         find number of explorers in given range to vertex
5         if number >= 1 then
6             try different vertex
7         else
8             goto vertex
9 end
```

Listing 6.7: Forcing the explorers to spread out.

The plan is simple: when an explorer does not have an important task to perform, such as recharging or walking towards a repairer and wants to move to a vertex, it checks the number of other explorers in the immediate vicinity of that vertex. If there are some other explorers, then the agent is forced to pick another vertex to check. It was decided to force explorers to keep a distance of at least two vertices from each other. In order to do so, the agents utilize a BFS implementation. When checking a vertex, it is easy to perform such a search until the given depth and find any other explorer agents lurking nearby.
Effectively, this strategy prevents the explorers from going to same areas of the map, but notably, only in the early stages of the simulation. Later on, the agents may have to position themselves relatively close to each other, for example when probing the last few remaining vertices, or defending the zones. It is also important to note, that the strategy does not apply to the 'special explorers'. Agents with this role have the task of probing the already controlled zones early on in the simulation. In order to do so quickly, they may have to move side-by-side throughout such zones, not at all far away from each other - forcefully spreading these agents out would be counter-productive.
The rules for finding and listing nearby vertices until given depth have been implemented in Jason to begin with. Later, the Java BFS implementation of the UFSC system got looked into and modified to suit the needs of the new strategy (it needed a list of nearby vertices instead of a path from one vertex to another). The agents can call this algorithm via an internal action. The Java implementation is much more versatile than the Jason one, as it allows to select any depth for the algorithm to run, whereas the Jason rules are hard-coded for given depths. While, strictly speaking, it is enough for the purposes of the new strategy, it is prudent to provide a more complete implementation. Creating a BFS in Jason is perfectly feasible (after all, one would only have to employ

logic programming); however, it has been decided against, because it would be cumbersome to integrate with the way information about the graph is stored by the UFSC system. The code for the implementation of exploring strategy is available in appendix B.3.

6.4 Miscellaneous

In this section, a number of smaller UFSC system changes is described. The successful ones made it to the final version of the changed system, while others, albeit ineffective, are listed in order to show all the investigated possibilities.

6.4.1 Reaching higher inspect achievements

From the equation 3.1, it can be inferred that reaching more achievements faster than the rival team gives an advantage in the contest.
The highest inspect achievement the UFSC team reaches is for inspecting 20 agents. It may be beneficial for the total score to try to get the next achievement - for 40 agents, thus earning free points throughout the simulation at a low cost of using a few actions on extra inspections. Two possibilities, for when the inspectors had no important tasks to accomplish and an agent appeared nearby, were explored:

- Inspecting already inspected rival agents
- Inspecting own agents

It turned out, however, that one does not receive achievement credits for these kinds of inspections. The effort was not at all useless, though - it was a catalyst for a realization described in subsection 6.4.2.

6.4.2 Reducing the number of inspect actions

In subsection 5.2.3, it is stated that the UFSC teams strategy is to inspect every enemy agent only once (not counting failed inspect actions). It is not strictly true, though. It turns out that the agents sometimes happen to inspect same rival agents several times, which is unnecessary, since the team is only concerned

with the type (e.g. saboteur, explorer etc.) of the inspected opponents. Following situations, where an enemy agent gets inspected more than once, have been identified:

Failed actions
: If an inspect action fails, either due to being attacked or at random, the team naturally does not get the desired information, and the enemy agent has to be inspected again. These situations are unavoidable, though not very numerous.

Simultaneous inspections
: Referring to a situation where two or more inspectors choose to inspect the same agent in the same turn.

Inexplicable
: An already inspected agent being inspected again for some reason.

Nothing can be done about the first item on the list - the inspector just has to try again, until it succeeds.
The simultaneous inspections do not occur very frequently. Notwithstanding, an addition to the code, which should reduce the number of such occurrences, was devised. The method is straightforward: Once an inspector chooses to inspect an enemy at a given vertex, it informs other inspectors about it. This vertex effectively becomes locked down for other inspectors that should wish to inspect it. If that is the case, the other inspector(s) must perform a different action. The listing 6.8 outlines this strategy.

```
1 begin
2     if want to inspect vertex and not vertex has lock
3         tell other inspectors about lock on vertex
4         do inspect
5     else
6         choose some other plan to employ
7 end
```

Listing 6.8: Plan for the inspect action.

The third item on the list has been the most interesting one. For a while, it was speculated that the UFSC code specification was inaccurate and the already inspected rival agents are deliberately being inspected again by the teams agents. Upon closer look at the code and some debugging, however, it turned out that the implemented rules are sound. With that cleared, a re-examination

of the server and simulation specifications was needed. In the scenario description [DKS13] on page 15, where all actions the agents can perform are described, a minor detail in the description of the inspect action was spotted:

> "This action is used to inspect the internal attributes of an opponent agent, given as a parameter. (...) If no parameter is given, **all** opponent agents standing on the same node are inspected."

In the UFSC code, the inspect action is always performed without the optional parameter mentioned above. Subsequently, the simulation output logs from the agents were investigated. Sometimes, the following situation occurs: The agent is standing on a vertex with two or more enemy agents, at least one of which has not been inspected yet. The agent then performs the inspect action; this not only inspects the targeted unknown agent, but also all enemy agents on the vertex, despite of them having already been inspected beforehand. In this situation, instead of counting the action as one inspection only, the server counts it as several ones (the number of opponent agents on the vertex). Later, the statistics give a false impression of the agents performing unnecessary inspections, when in fact, these extra inspections are merely by-products of legitimate actions. These situations could be easily eliminated with a minor change to the UFSC code - adding the optional parameter to every inspect action. This, however, would produce an unwanted effect; if two or more not inspected enemy agents were standing on the same vertex, the agent would only inspect one of them, instead of all at once. Bearing this in mind, the code was left unaltered in this case.

6.4.3 Purchasing upgrades

The UFSC team does not purchase any upgrades for its agents, instead saving the money from achievements in order to improve the overall score.
Two purchasing strategies were tried out: acquiring attack upgrades for saboteurs and health upgrades for explorers. The necessary code was already written in the UFSC system, but it was deactivated and never tested. All that had to be done, was to adjust the parameters (which and how many upgrades to buy) in the relevant code bits and run some tests.

6.4.3.1 Saboteurs

If enough (two for each of the agents) attack upgrades were bought, the saboteurs would have enough strength to disable any rival agent by just one successful attack. It would be interesting to check, if the benefits of being able to disable enemies quicker would be enough to compensate the used achievement points.

6.4.3.2 Explorers

The explorers initial health is 4, exactly the same as the power of a standard saboteur attack, i.e. only one successful attack is needed to disable an explorer. Moreover, the explorers are one of the preferred targets for the enemy saboteurs. Purchasing just a single health upgrade for each explorer agent would mean that the enemy saboteurs would have to use two successful attacks in order to disable an explorer. Analogically to the purchasing strategy for the saboteurs, the main point of interest here was whether or not the benefits of such approach would be able to counterbalance the decrease in the final score due to use of achievement points.

Chapter 7

Testing the changed system

This chapter describes the process of testing the changes made to the UFSC system, and presents the test results.

7.1 Testing process

The tests were performed by running contest simulations, where the original and changed systems were set to play against each other. A decision was made to evaluate each of the implemented changes (i.e. the surveying, inspecting, exploring and buying strategies) separately and then test the system with all those changes active. For each of these cases, a large number of simulations against the original system was run and the results were analysed. Doing so made it possible to evaluate the impact of each change by itself, as well as the overall system impact. In order to establish baseline results, simulations where the original system was set to play against itself were run, thus providing basic scores to compare the changed system with.

7.2 Simulation statistics

After a single simulation is run, the contest system automatically outputs statistics, which include scores, the numbers of actions performed by the agents, when all the achievements have been reached and more. These statistics are the background for the test evaluation. They are output in a threefold manner:

- Visual data: graphs and plots, saved in an image format
- Raw data: long sequences of logging data, saved in .log and .txt files
- Text files: small formatted text files written in LaTeX

While the visual data appears great for the human observer, it is hardly made use of in the assessments, chiefly due to the sheer number of the simulation runs. Instead, simple programs making use of regular expressions, which extract the relevant data from the raw text and log files, have been written by us. The extracted and formatted data is then run through a R script, which produces comprehensible statistical results. Albeit unrelated to the UFSC system implementation itself, the code for these programs is available in appendix D.

Below is a list of factors deemed relevant in the testing process:

Scores
: The number of matches won, as well as the average scores are the ultimate indicators of the quality of the system.

Survey achievements
: Crucial for the changes to the sentinel and surveying strategy. They should ensure a quicker completion of these achievements.

Probe achievements
: Relevant to the changes in explorers. They should make the system reach these achievements faster. Aside from the achievements themselves, it is interesting to look at the precise turn in which the last vertex gets probed. It is also conceivable that the surveying strategy may help the explorers to get around the map quicker. Therefore, the probe achievements are also looked at in context of the surveying strategy.

Inspect achievements
: Analogous to the probe achievements, merely concerning the inspectors and their respective achievements instead. Aside from the achievements themselves, an interesting factor is the precise turn in which the last, 28th enemy entity gets inspected.

Number of survey actions
An indicator for the surveying strategy - the number of survey actions performed by the sentinels.

Number of inspect actions
An indicator of the strategy for reducing thereof - the number of inspect actions performed by the inspectors.

The aforementioned statistics were gathered for each version of the system and compared in order to assess whether the implemented changes had a positive impact on the performance of the system.

7.3 Test results

In this section, the test results obtained with the process described in section 7.1 are presented. Team A denotes the system where the changes have been implemented, while team B denotes the original system. In the base results subsection, both team A and B use the original system code.

7.3.1 The base results

The simulations between unaltered versions of the UFSC system form the base results and foundation for evaluation of the subsequent changes made to the system. Since in this case two identical systems were competing, it was expected that the statistics for scores and achievements were evenly distributed, with only small deviations from a 50-50 split in match victories and swiftness of achievement completion. The figures in tables 7.1, 7.2 and 7.3 are based on 500 simulations.

Achievement	Average turn A	Average turn B
survey320	25.778	25.892
survey640	210.51	218.00
probe160	71.52	71.504
probe320	160.35	159.22
probeAll	295.13	294.86
inspect10	36.528	35.546
inspect20	90.64	89.24
inspect28	309.95	309.8

Table 7.1: Average turns for reaching certain achievements.

Action	Average number A	Average number B
survey	52.97	53.81
inspect	76.568	76.464

Table 7.2: Average number of certain actions performed in a simulation.

Indicator	Number A	Number B
Wins	252	248
Average score	118652.70	118665.66
survey640 reached	237	246
inspect28 reached	232	238

Table 7.3: Total number of wins, average scores in a simulation and total number of simulations where the survey640 achievement was reached.

Unsurprisingly, the statistics for simulations between two unaltered instances of the system were almost identical. Nonetheless, running these tests was paramount, as the process of testing the changes required consistent base figures to compare results against.

7.3.2 Sentinel

A crucial detail to consider regarding the surveying strategy (see section 6.1) was when to stop using it (i.e. which step of the simulation). It is important to uncover the graph quickly in the beginning. Furthermore, the survey action is most effective in the early stages of the game, when all/most edge values are unknown - then, each survey will probably uncover multiple edges at a time,

while later on, when most edges have been surveyed, the chance of uncovering multiple edges simultaneously declines. Tests were performed for a few configurations: where the active surveying strategy was used in the first 25, 50, 75, 100 and 133 turns, in order to find out how long it is feasible to be using sentinels for active surveying instead of other tasks, such as protecting the zones.
In any case, it was expected that the survey achievements for 320 and 640 edges [1] to be achieved much faster by the changed system. Although the 640 achievement is not always reached, it was expected that the changed system would reach it more often than the original one. Other achievements were looked at, in order to determine if uncovering the edges faster had an impact on other agents performance (e.g. helping them achieve their goals faster), as well as on the overall simulation scores.

The statistics gathered from the test runs revealed that the active surveying strategy performs well. A large improvement in the swiftness of reaching the survey320 and survey640 achievements is noticeable (ergo, the surveying process is faster). The table C.4 shows this very clearly - on average, the changed system was a few turns ahead of the original one in reaching survey320, while the differences became truly significant with the survey640 achievement. It was also interesting to look at the number of simulations in which the survey640 was reached at all (since this not always is the case). Table C.2 shows these numbers. Again, not only did the changed system reach the achievements faster, but also more often in comparison to the original one.

One of the concerns was that the sentinels using too much time on surveying would diminish the overall score of the team, since the sentinels would not be able to protect zones as often in the early stages of the simulation. Indeed, the number of survey actions performed by the sentinels was much higher than in the original system (see figure C.3); however, it turned out that these losses were being fully made up for by receiving more achievement points and generally allowing the team as a whole to traverse the graph faster. The table C.3 proves this point - the average score differences between the systems were slightly in the favour of the changed one. As for the number of won simulations, the changes had slightly improved on that too; instead of a near 50-50 split, configurations where the strategy is employed until the 50th, 100th or 133rd step, gave a 2.5-7% improvement. The configurations employing the strategy until the 25th and 75th step were not as successful (see table C.3). Lastly, no change was recorded in the speed of reaching probe and inspect achievements with the strategy activated. It seems that the strategy had a very limited or no impact at all in this matter.

The detailed results, as well as visual representation of the relevant data in form

[1] 160 and earlier achievements are reached almost immediately in the first few turns of the simulation.

of boxplots for each of the 5 configurations are available in appendix C.1.

7.3.3 Inspector

Similarly to the surveying strategy, it was important to decide when the inspecting strategy should be used (see section 6.2). The natural stopping point would be inspecting all 28 enemy agents, but deciding a starting point was not obvious. In the very beginning of the simulation, the strategy could simply confuse the inspectors, especially due to the large number of auctions, since to begin with, all enemies are uninspected. Examining the results from original systems (see section 7.1) shows that, in the original system, 10 enemies were inspected on average at step 36, while 20 were inspected on average at step 90. Similarly to the case of sentinels strategy, it was decided to run tests with several configurations and then choose the best one. 4 tests were settled for: with the strategy being used after steps 25, 50, 75 and 100. The goal of these tests was to check whether the achievement for inspecting 20 enemies would be reached faster on average, finding the number of simulations in which all enemy agents were inspected, the average turn of inspecting the last enemy agent, and the possible impact on the average scores and wins. Lastly, the average number of inspect actions was looked at in order to check the effect of the change (see subsection 6.4.2) which aimed to reduce the number of these actions.

Analysing the test results showed that the active inspecting strategy was successful. Analogically to the sentinels, the respective achievements (inspect10, inspect20) and inspecting all enemy agents were completed faster than in the original system. Likewise, it was more often the case that the team managed to inspect all enemy agents (inspect20 achievement is always reached in any case). The data in tables C.6 and C.8 supports these assertions. Furthermore, the minor change in attempt of reducing the number of redundant inspect actions was successful, as clearly visible on figure C.7.

The strategy, although it forced the inspectors to pursue enemy targets instead of holding the zones, had no negative impact on the average scores. In fact, the average score differences were slightly in favour of the changed system, but it did not seem to have a direct influence on the number of won simulations in case of the strategy being employed after the 50th and 75th step. Employing the strategy after the 25th and 100th step had a slightly negative impact on the number of victories (see table C.7).

The detailed results, as well as visual representation of the relevant data in form of boxplots for each of the 4 configurations are available in appendix C.2.

7.3.4 Explorer

For the strategy in which the explorers spread out more in order to always probe different areas of the map, it was interesting to check the effect on the swiftness of reaching the achievements for probing 160 and 320 vertices, as well as the turn in which all vertices had been probed. Furthermore, the average scores were looked upon, in order to check if the strategy had any impact on the systems performance as a whole.

The strategy was tested in two configurations - employed until the 50th and 133rd step of the simulation. The reason for this was, that it is desirable for the explorers to stay spread out only in the beginning of the simulation. After the majority of vertices had been probed, the explorers may have to move closer together in order to get the remaining ones probed efficiently. Thus, restricting their movements at that point would be aimless.

The results were quite surprising: both configurations had actually increased the time for reaching the probe achievements, as well as for probing all vertices. The data in table C.11 shows that the increase was marginal in case of the strategy being employed until step 50, while in the case of the strategy being used until the 133rd step, the increase was rather significant. Even more interesting was the data from the table C.10: regardless of the probing process taking more time and thereby the achievements being reached later, the changed system actually won more often and got a higher average score in both cases. This matter is discussed in subsection 8.3.1.

The detailed results, as well as visual representation of the relevant data in form of boxplots for each of the 2 configurations are available in appendix C.3.

7.3.5 Upgrades

As it turns out, purchasing upgrades had a vastly negative impact on the system. Regardless of the two strategies (see subsection 6.4.3) being conservative in the number of upgrades bought (in total, either 6 for explorers or 12 for saboteurs per simulation), the average scores and the number of simulation wins were staggeringly lowered. In fact, the results were perfectly clear after a mere 100 simulation runs for each of the two strategies: with the average scores diminished by 5-10% and the percentage of won simulations as low as 15-20%, further testing was deemed redundant.
Section C.4 contains the precise results of the test runs for the buying strategies.

7.3.6 Final results

This section describes the final series of tests - with all the implemented changes activated (except for the inferior purchasing strategies).
It was especially interesting to see whether the quicker and more exhaustive surveying ensured by the sentinels (see subsection 7.3.2) would support the new strategy employed by the inspectors. Specifically, by having the knowledge of graph structure early on and having surveyed a larger number of edges, the inspectors could be able to find the paths to far enemies more often, and also to have a higher level of certainty that the chosen path actually is the shortest one [2]. The figures in tables 7.4, 7.5 and 7.6 are based on 500 simulations.

Achievement	Average turn A	Average turn B	Difference
survey320	21.398	26.023	-4.625
survey640	140.277	213.383	-73.106
probe160	71.702	71.554	+0.148
probe320	159.644	159.783	-0.139
probeAll	294.624	293.112	+1.512
inspect10	33.688	35.064	-1.376
inspect20	78.05	88.098	-10.048
inspect28	271.73	309.86	-38.13

Table 7.4: Average turns for reaching certain achievements.

Action	Average number A	Average number B	Difference
survey	71.872	53.89	+17.982
inspect	69.86	79.422	-9.562

Table 7.5: Average number of certain actions performed in a simulation.

Indicator	Number A	Number B	Difference
Wins	259	241	+9 [3]
Average score	119879.322	118663.466	+1215.856
survey640 reached	286	243	+43
inspect28 reached	391	247	+144

Table 7.6: Total number of wins, average scores in a simulation and total number of simulations where survey640 achievements was reached.

[2] More knowledge about the graph structure implies lower probability of that an unknown shortest path between two vertices exists.

The final results confirmed the previous findings for the respective changes. The changed system visibly outperformed the original one in the surveying and inspecting strategies. All the major surveying and inspecting achievements were reached faster. The speed of exploring was barely affected, albeit with a minor tendency to probe slower (see table 7.4). Inspecting all 28 enemy agents and surveying 640 edges does not happen in every simulation. The new strategies did have an impact on this as well: the changed system achieved these milestones more often and faster than the original one. (see table 7.6) As a consequence of the new strategies, the number of survey actions performed had increased (since the sentinels are bound to survey more often), while the number of inspect actions had decreased (see table 7.5). The implemented strategies had a positive impact on the final results as well. The average score and win frequency for the changed system had increased (see table 7.6).
The visual representation of the relevant data in form of boxplots is available in appendix C.5.

[3]The difference between the two numbers is, of course, 18, but here the point of interest is the number of simulations won above the equilibrium situation (250 victories each).

Chapter 8

Discussion

This chapter contains a discussion of the obtained test results, as well as other matters pertaining to this project, which were thought-provoking.

8.1 Pseudocode

A few sections in chapter 6 included several pseudocode listings.
The pseudocode is a very useful tool for any programmer, as it allows to convey complex ideas, in programs or algorithms in a way which closely resembles a natural language.
In this project, a simple pseudocode style is used: it shows the algorithms in a step by step manner, only including the key directives (e.g. 'find best vertex to go to'), without showing their implementation details. While the psuedocode style closely resembles an imperative programming language, such as C, the actual Jason code often looks quite different and correspondence between lines of pseudocode and code may not be apparent. Although seemingly a counter-intuitive ambiguity, this has actually been done deliberately: the reasoning cycle of an agent (see subsection 4.2.1) is fundamentally similar to a human one.
An *if-then-else* construct can be used as an example. When considered from a human logic point of view, the way it works is obvious: "if a given event occurs under certain conditions, then take some action, otherwise take a different course

of action". Jason agents do not *per se* use *if-then-else* statements, but a very similar mechanism is in place (see listing 4.2). An agent can have a number of relevant plans (courses of action analogous to an *if-elseif-...-else* construct) for a given event. Amongst these it then selects applicable plans, i.e. plans that can be employed under certain general circumstances. If the chosen plan fails, an agent may have a rescue plan for such an occasion; likewise, a human being would have [1] an alternative course of action, if the original one has failed.

Effectively, despite the differences in how the pseudocode and real code look like, the underlying ideas and thoughts are equivalent, thus rendering the chosen pseudocode style a valid way of describing parts of the implementation.

8.2 Number of simulations and result reliability

Since the agents are autonomous (see description 2.1), the simulations and results described in chapter 7 are non-deterministic. Effectively, two consecutive simulation runs on the same map will undoubtedly yield different results.
Obviously, a higher number of simulations implies a higher degree of the statistical accuracy and consistency of the results. As stated at the beginning of section 7.1, a substantial number of simulation runs was completed, although not without some inconvenience: the original simulation server configuration allows to run at most 3 simulations in sequence at a time, requiring a constant presence of a person at the computer in order to keep restarting the simulations approximately every 15 minutes (a single simulation takes around 5 minutes). Therefore, the server configuration was changed by us, in order to allow running a higher number of simulations one after another, without needing to restart. The simulations were run in batches of 100 at a time, totalling at around 8 hours per such run. In the beginning, it was assumed that these 100 simulation runs for each system configuration would be sufficient to produce precise results, but later it was decided to increase that number to 500. Naturally, one can never be fully sure when dealing with inherently non-deterministic systems. Bearing the time constraints in mind, as well as the fact of having only 1 computer capable of running the tests, the total number of **6700** simulations [2], as well as the extracted statistical results, were deemed satisfactory.

[1] Or **seamlessly come up with/create** one by himself; this, incidentally, is one of the few things that humans are capable of, but AI, as of yet, is not.

[2] Equivalent to running the simulations constantly for a little over 23 full days. The total number of simulations does not include the ones run for debugging purposes when refining the code, but only the ones run once the code for the respective implementations was completed.

8.3 Test results

Regrettably, the laborious testing process seems to be the only possible way of quantifying and objectively judging the impact of changes made to the system. It is simply not be possible for a human to go through all the test data (see section 7.2) and draw conclusions, despite some of the results being presented in a friendly, graphical manner. Furthermore, even if some conclusions could be made that way, they would be entirely subjective to the humans perception of the data plots, or, even worse, the animation of the simulations in the GUI. The raw data is fully relied on, as it is easily quantifiable (e.g. 'on average, achievement X is reached Y turns faster') and allows for a much more objective assessment. In section 8.2 it was stated, that a 100% certainty is not achievable and also explained why it is so. We are, nevertheless, confident in asserting that a large number of simulations, is at very least able to provide a sound indication of whether the changes made to the system have been beneficial or not.

8.3.1 Choosing best configurations for the strategies

Based on the extensive testing (see chapter 7), it was possible to ascertain which specific configurations of the implemented strategies perform best on their own. Subsequently, they were put to work together in the testing of the overall performance of the changed system.
For sentinels, it was decided to employ the active surveying strategy (see section 6.1) until the 133rd step of the simulation, since it performed best in general and accumulated the highest number of simulation victories.
For inspectors, it was decided for the active inspecting strategy (see section 6.2) to begin after the 50th step of the simulation, due to the swiftness of reaching the achievements, while not affecting the scores and the number of wins negatively.
The tests of exploring strategy (see section 6.3) proved interesting. Although the probing process became slightly slower, the scores and number of wins actually increased. It is uncertain why this was the case. We theorize that in some cases the explorers had 'locked up' (i.e. were mutually prevented from moving towards a desired spot on path to a vertex to probe) and instead chose to perform another action, such as helping to defend zones. Doing so multiple times during the course of a simulation would contribute to the final score in some degree, hence explaining the higher average scores. Naturally, it could also be the case that the score and number of wins increase was purely coincidental, based on the non-determinism of the systems and simulations. We are reluctant to assign the 'blame' to pure chance in this case. After all, two batches of 500 simulations for the explorer strategy were run and in both cases, the changed

system was victorious more often and reached a higher average score.
Of the two possibilities, the strategy being employed until the 50th step was deemed a better option, as it only increased the probe time marginally (in contrast to employing the strategy until the 133rd step, where the increase was significant), while still giving the benefits of winning more and scoring higher. Ultimately, the goal of the contest is to score as many points and win as many simulations as possible. Bearing that in mind, it was decided to include the modified explorers in the final version of the system.
Finally, the strategies for purchasing upgrades (see subsection 6.4.3) were a total fiasco - the benefits of having stronger agents were insufficient to compensate scores decline due to use of achievement points.
During the last contest, all the teams either did not use the upgrades at all, or only purchased very few of them. It seems that the upgrade system is not properly balanced, as the costs of purchasing upgrades clearly are much higher than any obtained benefits. Hopefully, in the upcoming contest, the parameters will be adjusted in order to make the upgrades more attractive, e.g. by lowering their costs or increasing their benefits. It could also be interesting to, instead of charging achievement points, give the teams a set, small number of upgrades they may distribute amongst their agents, thus possibly resulting in diversified choices throughout the teams.

8.3.2 The overall results

The test results showed that the implemented strategies were sound.
The better performance in reaching achievements was fully expected, based on the results of testing the new strategies separately (see chapter 7). It was, however, dismaying that the changed system had not achieved a higher number of simulation victories. It was assumed, that the combined advantages of reaching certain achievements, as well as general benefits of discovering the graph edges and enemy agent details, faster, would be reflected in more numerous simulation wins, especially since the average score had also been higher. Nevertheless, it is clear that the changes had a principally positive effect on the overall performance of the system (see subsection 7.3.6). The changes appear to have a compelling impact within their respective areas, but the influence on the final results and scores seems to be limited. This could be because the changes did not *per se* concentrate on improving the scores directly. Instead, they improved some other strategies, which in turn affected the scores indirectly. Making changes to a strategy such as zone control, would probably have a larger impact on the scores. However, the goals of this project were to improve the strategies which were either incomplete or non-existent, instead of working with a well established and refined strategy, such as the zone control.

8.4 Experiences working with Jason and UFSC system

Generally, working with the Jason platform and the UFSC system was an excellent experience. The two subsections below list and explain what the good and bad things have been, in our opinion.

8.4.1 Jason

Working with the Jason platform and multi-agent programming paradigm has been a singular experience, despite a few minor flaws.

Language and syntax
The Jason syntax is very appealing and expressive. It allows for writing of succinct programs, which nonetheless are able to perform complicated tasks. Since some of it is largely based on logic programming, it has been relatively easy to comprehend and get the general understanding of the basics quickly. The way the plans are handled is particularly clever, since it resembles human reasoning and thus one can almost seamlessly convey ideas in the code. Another great feature is, that an agent can have multiple plans for a single event, discern between contexts these plans can be applied in, and finally also have failure handling mechanisms for when the plans do not succeed (see subsection 4.1.2 for more about plans).

The platform
From the practical point of view, the Jason platform is a pleasant environment to work in. The language and IDE (see chapter 4) are simple to install, set up and understand. An extremely useful tool for the debugging process when learning the language has been the mind inspector (see figures 4.2 and 4.3). Lastly, the combination of the three different platforms into a unified platform (JaCaMo) is a phenomenal idea. Doing so allows to encompass several dimensions of the multi-agent programming paradigm (see section 4.3 for more about JaCaMo).

Interaction with Java
Supporting the Jason agent programming language with Java is an interesting approach. It has been beneficial when delegating parts of program to be handled by Java (such as data storage, algorithms and GUI), since Jason itself is not very well suited for dealing with these.

Documentation
The documentation was scarce. The basics were easy to grasp quickly, but much time was spent on understanding the more complicated parts of the platform. In some cases, the trial-and-error approach was resorted to in order to comprehend the code. This was exclusively the case with the Java part of the platform.

Other
Debugging programs running in the contest was difficult, as it was not possible to make use of the mind inspector and step-by-step runs of the programs due to the way the simulation server operates. The error and exception messages, which sometimes were not clear at all, had to be relied on. This made it troublesome to pinpoint the faults in the code. Lastly, Jason could use some more standard library functions for operating on lists and strings, similarly to what Prolog has to offer.

8.4.2 UFSC system

In general, working with the UFSC system was a pleasant experience. In the beginning, it was demanding and time consuming to understand the inner workings and interactions of the system, but once that was accomplished, making changes was relatively straightforward.

Assistance from Brazil
Upon contacting the team behind the system, they provided some excellent suggestions for the project. Notably, the team shared excerpts of their TODO list. Subsequently, some of the items from the list were implemented. Additionally, some original ideas of our own were implemented.

Jason code
The Jason code was very well written. The variable names were often self-explanatory and the commentary was decent as well. This made it easier to fathom how the system works and to write new code for it.

Java code
Working with the Java part of the system was something of a challenge. Code commentary was almost non-existent and the implementation was spread out over many small classes, whose functions and responsibilities were not always clear.

Chapter 9

Future perspectives

This chapter briefly describes the future of the multi-agent programming contest and provides examples of additional work that may be done with the UFSC system in order to try to improve its performance. Furthermore, the future of artificial intelligence and multi-agent systems in general is considered.

9.1 MAPC and the UFSC system

As mentioned in section 3.2, this year will be the last when the 'Agents on Mars' scenario is going to be used in the MAPC. Although the UFSC system will not partake in the future contests after this one (since a completely new scenario will be used), it may be of great use for educational purposes in multi-agent programming, as it has been for us during the course of this project. There still exists a number of changes that could be made and tested, for example:

Inspecting
: As a proof of concept, the inspecting strategy was implemented in such a way that the 'leader inspector' coordinates it, but this made it impossible for the coordinator to participate in the auctions. Delegating the task of

auctioneer to another agent (say, a sentinel) or changing the implementation, would most likely make the strategy a little more effective, since all 6 instead of 5 inspectors could participate.

Surveying

One could use the sentinels active surveying strategy for other agents, as a very low priority item, for example just above 'random walk', which is the lowest priority action. Instead of moving randomly, the agent could go to a neighbouring vertex, which has some unsurveyed edges. In the next turn, if the agent still has nothing better to do, it could perform a survey. On the other hand, it could also be beneficial to completely stop using the survey action after having surveyed 640 edges and having received achievement for doing so. At that point, most edges will have been surveyed and the agents would typically use survey action in order to discover value of a single edge, which may be considered a waste of time at that point.

Surveying and inspecting

Although a quite precise window for when these strategies are effective has been established, it may still be interesting to collect test data for other factors - for example depending on the type of the map, number of vertices/edges and thinning of the graph.

Exploring

A more sophisticated version of the explorers spreading strategy could be implemented, taking in consideration more variables and special cases - thus ensuring that the explorers spread out properly and do not mutually lock themselves up.

Testing against other systems

Testing against the original system has provided some excellent statistical data. Nonetheless, it may be beneficial to test against completely different systems of other MAPC participants, for example GOAL-DTU. This is because the changed system still mostly consists of the original strategies, with the few additions and changes. Testing against a system utilizing utterly different strategies would therefore be interesting.

Repairers

Having own repairers follow enemy saboteurs around (at a certain close distance, so they are not easily disabled) could be implemented. By doing so, one could ensure the repairers are very close when a friendly agent becomes disabled and can put it back into operation quickly. A disadvantage of this strategy is that the repairers are unlikely to participate in defending the zones, since they would be on the move constantly. It would be interesting to see if benefits of being able to repair friendly agents faster would

counterbalance the loss of points from the times when repairers cannot participate in controlling larger zones.

Strategies for saboteurs
An example strategy to exploit could be to make the saboteurs seek out the corners of the map and deliberately disturb enemies there. Corners of the map are good places to conquer, as they require a smaller number of agents to do so. Another option is to try organizing the saboteurs better, for example by making sure they do not all stay in the same area of the map, unless it is necessary.

Dijkstra vs BFS
Checking whether it is better to choose shortest paths (Dijkstra) and thus consume least energy possible in moving to a destination, or to use BFS and instead choose paths that are not necessarily cheapest, but consist of fewest steps.

Admittedly, the whole contest platform and the multi-agent systems participating in it encompass a completely virtual and simulated environment scenario, without any apparent practical use (apart from deepening the knowledge about MAS etc.). This could give an impression that MAS are not suited for more serious applications, but it is definitely not the case. The possibilities for using multi-agent programming in actual, useful applications are vast and have already been utilized in many areas.

9.2 AI and multi-agent systems

Despite the notions of intelligent agents and multi-agent systems being relatively new (compared to the age of computer science or AI research), numerous highly diversified real world applications have been created. A few examples are [1]:

Medical applications
A range of applications in medical care, for example helping the doctors to produce a diagnosis, managing resource allocation and providing remote healthcare [ISM10]. The remote healthcare seems particularly interesting. The project Confidence [2] has developed a prototype of a system for monitoring elderly patients living alone, both short-term (i.e. the patient falls

[1] This list does not aim to describe the systems' specifics, but merely to illustrate that these practical systems indeed do exist. For details, the reader is referred to the respective articles.
[2] `http://www.confidence-eu.org/`

- the relatives/medical staff are alerted) and long term (i.e. changes in the patients physical condition over a period of time) [KDM+11]. Another interesting application is analysis of images in order to detect retinal blood vessels and microaneurysms. It is based on agents 'exploring' the images pixel by pixel, communicating and being able to distinguish common patterns [PMG+12, PVM+14].

Mass spectrometry
A mass spectrometry experiment consists of making several subsequent test samples in order to achieve greater precision. The system is capable of performing an analysis of these independent samples of data and combining them into a unified result [RJGPnS+10].

Portfolio management
Creating an optimal asset investment portfolio. It is based on agents using different investment strategies, communicating with each other and projecting the most desirable course of actions, which ensures the best payoff-to-risk ratio [LAAM10].

Electric power systems management
Control, maintenance and management of large area electric power systems. Used for example in analysis of systems yet to be employed, in order to identify hidden errors in the infrastructure and thus prevent possible future mishaps. Also used in real time monitoring of the systems and restoring them after a fault has happened [CDM12].

Extreme programming
A development environment that enforces use of good programming techniques such as unit testing and supervises the entire process of version control, submitting updates to the code and cooperation between developers [LHC06].

An international conference dedicated to practical applications of multi-agent systems (PAAMS) is held on yearly basis. Some of the applications described above have (amongst many others) been presented at the PAAMS conferences [3]. Another large, yearly conference in this field is the international conference on autonomous agents and multi-agent systems (AAMAS) [4].
The presented examples support the assertion (see section 9.1) about possibilities for using MAS in practice. They also show a general trend in the advances within AI: the computers being able to perform increasingly difficult tasks, only recently feasible to be done exclusively by humans. More and more tasks are being computerised. While this is very often beneficial for the performance and

[3] `http://www.paams.net/`
[4] `http://www.aamas-conference.org/`

cost-effectivity of the tasks themselves, it also has an arguably negative impact on the society and its citizens. In a recent working paper [FO13], it is estimated that up to almost 50% of jobs in the USA are at risk of being computerised in the upcoming decades. The social impact of such occurrence would be immense. Naturally, one ought to be very cautious with making any precise estimates and assertions about the progress of technology (see quote 1.2). The tendency is, nonetheless, clear - the technology progresses quickly and still more tasks are subjected to computerisation.

The ability to express ideas in mentalistic notions (see section 2.2), as well as the properties of agents (see section 2.1) make it possible to convey complex ideas and perform complicated thought processes in a way not dissimilar to human reasoning. Such approach appears to have an immense potential. The humanity is still far from fully simulating (or even completely comprehending) human intelligence, but the artificial intelligence systems, regardless of technology and programming paradigms used, become increasingly sophisticated. Multi-agent programming is a growing and extremely promising area of AI research and is bound to continue being subjected to a lot of work and development in the upcoming years.

Chapter 10

Conclusions

This chapter contains concluding remarks for the thesis.

10.1 The project

During the course of this project, our goals and expectations were fulfilled. We gained a broad knowledge about artificial intelligence and multi-agent systems, as well as the Jason programming platform. In the first few chapters of this thesis, the theory and models pertaining to these subjects were laid out. This theoretical knowledge had subsequently been applied to a practical implementation of a multi-agent system. The project had dealt with the multi-agent programming contest and the system developed by its winners. By analysing the code, we understood how this system works and what strategies it employs. Possible improvements were identified and code was written to implement them. Lastly, exhaustive testing, in order to ascertain the quality of the changes, was performed.

10.2 The implementations and test results

The vigilant testing process provided conclusive data about the validity of the new implementations. Some of the newly implemented strategies had several versions. By visualizing and analysing their individual results, we believe that the best ones were chosen for the final version of the system, which contained all the successful new strategies. It turned out that the final version of the system improved a number of specific system performance parameters; most importantly, however, the overall performance and win rate had increased as well.

Bearing the time constraints in mind, we are fully satisfied with the number of tests performed for each version of the implementations and the quality of the attained statistical data. An important matter was automating the process of extracting the relevant data from simulation server output. A few scripts to do just that, as well as visualize and present the data in a clear format, were written by us. Finally, a number of further potential changes that could be made in order to make the system even better, was determined and described.

Appendix A

UFSC Code

In this appendix, attached are some of the original UFSC system code files. Since the full system consists of almost 100 separate files, totalling around 8500 lines of code, it was decided not include them all here; instead, only files in which changes were made during the course of this project are listed. These changes are shown in appendix B, while the full original code of the UFSC system is available online [HZB+13].

A.1 Sentinel

The original code for the sentinel agents.

```
1 { include("mod.common.asl") }
2
3 +!wait_and_select_goal <- !select_goal.
4 -!wait_and_select_goal[error(deadline_reached)] <- .print("
      Deadline reached"); !select_goal.
5
6 +!select_goal: is_energy_goal
7         <- !init_goal(recharge).
8 +!select_goal : is_wait_repair_goal(V)
9         <- .print("Waiting to be repaired. Recharging...");
```

```
10              !init_goal(recharge).
11  +!select_goal : not canCome(ComeT) &
        is_wait_repair_goal_nearby(V, Repairer)
12          <- .print("Waiting to be repaired (nearby). ", V , "
                ", Repairer, ". Recharging...");
13              !init_goal(recharge).
14  +!select_goal : is_goto_repair_goal_nearby(V)
15          <- .print("Goto to be repaired (nearby)...");
16              !init_goal(goto(V)).
17
18  +!select_goal: is_disabled &
        get_vertex_to_go_be_repaired_appointment(D, Path)
19          <-
20              .print("I have an appointment with some repairer.
                    I'm going to ", D, " using path: ", Path);
21              !init_goal(gotoPath(Path)).
22  +!select_goal: is_disabled &
        get_vertex_to_go_be_repaired_appointment_self(D, Path)
23          <-
24              .print("I have an self appointment with some
                    repairer. I'm going to ", D, " using path: ",
                    Path);
25              !init_goal(gotoPath(Path)).
26  +!select_goal: is_disabled & get_vertex_to_go_repair(D, Path)
27          <-
28              .print("I'm forever alone. I'm going to ", D, "
                    using path: ", Path);
29              !init_goal(gotoPath(Path)).
30
31  +!select_goal : is_parry_goal
32          <- !init_goal(parry).
33  +!select_goal: is_survey_goal
34          <- !init_goal(survey).
35
36  +!select_goal : is_good_map_conquered <-
37      .print("Good map conquered! Stopped!");
38      !init_goal(recharge).
39
40  /* Protect Island */
41  +!select_goal: is_keep_goal_island_enemy(Entity) <-
42      .print("I'm at the island and I'm going to stay here ",
            Entity);
43      !init_goal(recharge).
44  +!select_goal: there_is_enemy_nearby_island_geral(Op) <-
45      .print("I'm at the island and I'm going to ", Op, " to
            find some enemy inside");
46      !init_goal(goto(Op)).
```

```
47 +!select_goal: get_vertex_to_go_attack_search_island_geral(D,
       Path) <-
48     .print("I'm at the island and I'm going to ", D, " using
           ", Path, " to find some enemy there");
49     !init_goal(gotoPath(Path)).
50 /* End Protect Island */
51
52 +!select_goal : going_to_outside_goal(V) <-
53     .print("I'm inside a region. I can expand to ", V);
54     !init_goal(goto(V)).
55
56 +!select_goal : can_expand_to(V) <-
57     .print("I can expand to ", V);
58     !init_goal(goto(V)).
59
60 +!select_goal: is_goal_keep_aim_vertex
61         <-
62             .print("I'm at a pivot vertex. Recharging...");
63             !init_goal(recharge).
64
65 +!select_goal: is_goal_aim_vertex(Op)
66         <-
67             .print("I have a pivot vertex to go. I'm going to
                   ", Op);
68             !init_goal(goto(Op)).
69
70 +!select_goal: is_goal_aim_vertex(D, Path)
71         <-
72             .print("I have a pivot vertex to go. I'm going to
                   ", D, " using path: ", Path);
73             !init_goal(gotoPath(Path)).
74
75 //Hills
76 +!select_goal : can_expand_to_hill(V) <-
77     .print("I'm at a hill and I can expand to ", V);
78     !init_goal(goto(V)).
79
80 +!select_goal : is_at_aim_position_hill <-
81     .print("Stop! Stand still! I'm settled at the hill!");
82     !init_goal(recharge).
83
84 +!select_goal: is_goal_hill_vertex(Op)
85         <-
86             .print("I have a hill vertex to go. I'm going to
                   ", Op);
87             !init_goal(goto(Op)).
88 +!select_goal: is_goal_hill_vertex(D, Path)
```

```
89          <-
90              .print("I have a hill vertex to go. I'm going to
                    ", D, " using path: ", Path);
91              !init_goal(gotoPath(Path)).
92
93
94
95  +!select_goal
96          <- !init_goal(random_walk).
97
98  /*
99   * These functions must be dependent of each kind of agent
         because they will
100  * need to share some information with the other friends of
         the same kind
101  */
102 @do0[atomic]
103 +!do(Act):
104     step(S) & stepDone(S)
105 <-
106     .print("ERROR! I already performed an action for this step
          ! ", S).
107
108 @do1[atomic]
109 +!do(Act):
110     true
111 <-
112     !commitAction(Act).
113
114 +!initSpecific.
115 +!processBeforeStep(S).
116 +!processAfterStep(S) <-
117     !buildPivots(S);
118     !buildIslands(S).
119
120 /*
121  * CALCULATE THE BEST PLACES
122  */
123 +!buildPivots(S):
124     (not lastCalcPivot(_) & S >= 17 | lastCalcPivot(N) & S - N
          >= 17) & .my_name(MyName) & play(MyName,
         sentinelLeader,"grMain")
125 <-
126     !determinePivots;
127     -+lastCalcPivot(S).
128 +!buildPivots(S).
129
```

```
130 +!buildIslands(S):
131     not hill(_) & goalState(_,defineInitialPivots,_,_,
            satisfied) & (not lastCalcIsland(_) & S >= 23 |
            lastCalcIsland(N) & S - N >= 23) & .my_name(MyName) &
            play(MyName,sentinelLeader,"grMain")
132 <-
133     !determineIslands;
134     -+lastCalcIsland(S).
135 -!buildIslands[error(ErrorCode),error_msg(MsgError)] <-
136     .print("Error occurred to build islands! ", ErrorCode, "
            -> ", MsgError).
137 +!buildIslands(S).
```

A.2 Inspector

The original code for the inspector agents.

```
1  { include("mod.common.asl") }
2
3  //There is no priority because one of them can fail
4  is_inspect_goal(V, Entity) :- not is_disabled & position(MyV)
       & myTeam(MyTeam) & visibleEntity(Entity, V, Team, _) &
       Team \== MyTeam & ia.edge(MyV,V,_)
5                      & not entityType(Entity, _, _, _, _).
6
7  is_inspect_goal(Entity) :- not is_disabled & position(MyV) &
       myTeam(MyTeam) & visibleEntity(Entity, MyV, Team, _) &
       Team \== MyTeam
8                     & not entityType(Entity, _, _, _, _).
9
10 +!wait_and_select_goal <- !select_goal.
11 -!wait_and_select_goal[error(deadline_reached)] <- .print("
       Deadline reached"); !select_goal.
12
13 +!select_goal: is_energy_goal
14         <- !init_goal(recharge).
15 +!select_goal : is_wait_repair_goal(V)
16         <- .print("Waiting to be repaired. Recharging...");
17             !init_goal(recharge).
18 +!select_goal : not canCome(ComeT) &
       is_wait_repair_goal_nearby(V, Repairer)
19         <- .print("Waiting to be repaired (nearby). ", V , "
               ", Repairer, ". Recharging...");
20             !init_goal(recharge).
21 +!select_goal : is_goto_repair_goal_nearby(V)
22         <- .print("Goto to be repaired (nearby)...");
23             !init_goal(goto(V)).
24
25 +!select_goal: is_disabled &
       get_vertex_to_go_be_repaired_appointment(D, Path)
26         <-
27             .print("I have an appointment with some repairer.
                   I'm going to ", D, " using path: ", Path);
28             !init_goal(gotoPath(Path)).
29 +!select_goal: is_disabled &
       get_vertex_to_go_be_repaired_appointment_self(D, Path)
30         <-
31             .print("I have an self appointment with some
                   repairer. I'm going to ", D, " using path: ",
                   Path);
```

```
32              !init_goal(gotoPath(Path)).
33 +!select_goal: is_disabled & get_vertex_to_go_repair(D, Path)
34          <-
35              .print("I'm forever alone. I'm going to ", D, "
                    using path: ", Path);
36              !init_goal(gotoPath(Path)).
37
38 +!select_goal: is_inspect_goal(Entity)
39          <-
40          .print("I'm going to inspect ", Entity);
41          !init_goal(inspect).
42
43 +!select_goal : is_good_map_conquered <-
44     .print("Good map conquered! Stopped!");
45     !init_goal(recharge).
46
47 /* Protect Island */
48 +!select_goal: is_keep_goal_island_enemy(Entity) &
       is_survey_goal <-
49     .print("I'm at the island and I'm going to stay here (
           survey) ", Entity);
50     !init_goal(survey).
51 +!select_goal: is_keep_goal_island_enemy(Entity) <-
52     .print("I'm at the island and I'm going to stay here ",
           Entity);
53     !init_goal(recharge).
54 +!select_goal: there_is_enemy_nearby_island_geral(Op) <-
55     .print("I'm at the island and I'm going to ", Op, " to
           find some enemy inside");
56     !init_goal(goto(Op)).
57 +!select_goal: get_vertex_to_go_attack_search_island_geral(D,
       Path) <-
58     .print("I'm at the island and I'm going to ", D, " using
           ", Path, " to find some enemy there");
59     !init_goal(gotoPath(Path)).
60 /* End Protect Island */
61
62 +!select_goal: is_inspect_goal(Op, Entity)
63          <-
64          .print("I'm going to ", Op, " to inspect ", Entity);
65          !init_goal(goto(Op)).
66 +!select_goal: is_survey_goal & not is_leave_goal
67          <- !init_goal(survey).
68
69 +!select_goal : going_to_outside_goal(V) <-
70     .print("I'm inside a region. I can expand to ", V);
71     !init_goal(goto(V)).
```

```
72
73 +!select_goal : can_expand_to(V) <-
74     .print("I can expand to ", V);
75     !init_goal(goto(V)).
76
77 +!select_goal: is_goal_keep_aim_vertex
78         <-
79             .print("I'm at a pivot vertex. Recharging...");
80             !init_goal(recharge).
81
82 +!select_goal: is_goal_aim_vertex(Op)
83         <-
84             .print("I have a pivot vertex to go. I'm going to
                ", Op);
85             !init_goal(goto(Op)).
86 +!select_goal: is_goal_aim_vertex(D, Path)
87         <-
88             .print("I have a pivot vertex to go. I'm going to
                ", D, " using path: ", Path);
89             !init_goal(gotoPath(Path)).
90 //Hills
91 +!select_goal : can_expand_to_hill(V) <-
92     .print("I'm at a hill and I can expand to ", V);
93     !init_goal(goto(V)).
94
95 +!select_goal : is_at_aim_position_hill <-
96     .print("Stop! Stand still! I'm settled at the hill!");
97     !init_goal(recharge).
98
99 +!select_goal: is_goal_hill_vertex(Op)
100         <-
101             .print("I have a hill vertex to go. I'm going to
                ", Op);
102             !init_goal(goto(Op)).
103 +!select_goal: is_goal_hill_vertex(D, Path)
104         <-
105             .print("I have a hill vertex to go. I'm going to
                ", D, " using path: ", Path);
106             !init_goal(gotoPath(Path)).
107
108 +!select_goal
109         <- !init_goal(random_walk).
110
111 +inspectedEntity(Entity, Team, Type, V, Energy, MaxEnergy,
    Health, MaxHealth, Strength, VisRange):
112     step(S) & not myTeam(Team) & artifact(inspectArtifact,
        IdInspectArtifact)
```

```
113 <-
114     addEntity(Entity, Type, MaxHealth, Strength, VisRange)[
            artifact_id(IdInspectArtifact)];
115     .print("The entity ", Entity, " was inspected Energy (",
            Energy, ",", MaxEnergy, ") Health (", Health, ",",
            MaxHealth, ") Strength ", Strength).
116 /*
117  * These functions must be dependent of each kind of agent
        because they will
118  * need to share some information with the other friends of
        the same kind
119  */
120 @do0[atomic]
121 +!do(Act):
122     step(S) & stepDone(S)
123 <-.print("ERROR! I already performed an action for this step!
        ", S).
124
125 @do1[atomic]
126 +!do(Act):
127     true
128 <- !commitAction(Act).
129
130 +!initSpecific.
131 +!processBeforeStep(S).
132 +!processAfterStep(S).
```

A.3 Explorer

The original code for the explorer agents.

```
1  { include("mod.common.asl") }
2
3  //Verify if the explorer is at an unprobed vertex, and he is
       the explorer with the highest priority when there are
       other explorers at the same vertex
4  is_probe_goal  :- not is_disabled & position(MyV) & not ia.
       probedVertex(MyV,_) &
5            myTeam(MyTeam) & myNameInContest(MyName) &
6            .my_name(MyAgentName) &
7            not (
8              visibleEntity(Entity, MyV, MyTeam, normal) &
9              friend(AgentName, Entity, explorer, _) &
10             Entity \== MyName &
11             priorityEntity(AgentName, MyAgentName)
12           ).
13 is_probe_goal(Op) :- not is_disabled &
       there_is_unprobed_vertex_next_to_mine(Op).
14
15
16 /*
17  * Probe inside hill just special explorers
18  */
19 there_is_unprobed_vertex_next_to_mine_special(Op) :- .my_name(
       MyName) & play(MyName,specialExplorer,"
       grSpecialExploration") &
20           not is_disabled & step(S) & position(MyV) & myTeam(
                 MyTeam) & hill(Neighborhood) &
21           .setof(V,
22                ia.edge(MyV,V,_) &
23                .member(V, Neighborhood) &
24                not ia.probedVertex(V, _) &
25                not (visibleEntity(Entity, V, MyTeam, normal) &
26                     friend(_, Entity, explorer, _))
27           & not nextStepExplorer(_, V, S), Options)
28          & .length(Options, TotalOptions) & TotalOptions > 0 &
29          .nth(math.random(TotalOptions), Options, Op).
30
31 get_path_to_unprobed_probe_special(D, Path) :-
32        .my_name(MyName) & play(MyName,specialExplorer,"
               grSpecialExploration") &
33        not is_disabled &
34        position(MyV) &
35        hill(Neighborhood) &
```

```
36          .setof(Vertex,
37          .member(Vertex, Neighborhood) &
38          ia.probedVertex(Vertex,-1) &
39          not nextStepExplorer(_, Vertex, _)
40          , List) &
41          not .empty(List) &
42          ia.shortestPathDijkstraCompleteTwo(MyV, List, D, Path,
                Lenght) &
43          Lenght > 2.
44   /*
45    * End special explorers
46    */
47
48
49
50  /*
51   * ########################################################
52   * Check if there is some unprobed vertex around mine
53   * ########################################################
54   */
55   //Test if the vertex is not probed and there is other
        explorer there and also no one of the other explorers
        will go there in this step
56   //First try the vertices at the hill
57  there_is_unprobed_vertex_next_to_mine(Op) :- step(S) &
        position(MyV) & maxWeight(INF) & myTeam(MyTeam) & hill(
        Neighborhood) &
58          .setof(V,
59                ia.edge(MyV,V,W) & W \== INF &
60                .member(V, Neighborhood) &
61                not ia.probedVertex(V, _) &
62                not (visibleEntity(Entity, V, MyTeam, normal) &
63                     friend(_, Entity, explorer, _))
64           & not nextStepExplorer(_, V, S), Options)
65          & .length(Options, TotalOptions) & TotalOptions > 0 &
66          .nth(math.random(TotalOptions), Options, Op).
67  there_is_unprobed_vertex_next_to_mine(Op) :- position(MyV) &
        infinite(INF) & myTeam(MyTeam) & hill(Neighborhood) &
68          .setof(V,
69                ia.edge(MyV,V,W) & W \== INF &
70                .member(V, Neighborhood) &
71                not ia.probedVertex(V, _) &
72                not (visibleEntity(Entity, V, MyTeam, normal) &
73                     friend(_, Entity, explorer, _))
74           & not nextStepExplorer(_, V, S), Options)
75          & .length(Options, TotalOptions) & TotalOptions > 0 &
76          .nth(math.random(TotalOptions), Options, Op).
```

```
77
78 //Test if the vertex is not probed and there is other explorer
       there and also no one of the other explorers will go
       there in this step
79 there_is_unprobed_vertex_next_to_mine(Op) :- step(S) &
       position(MyV) & maxWeight(INF) & myTeam(MyTeam) &
80           .setof(V,
81                ia.edge(MyV,V,W) & W \== INF &
82                not ia.probedVertex(V, _) &
83                not ( visibleEntity(Entity, V, MyTeam, normal) &
84                     friend(_, Entity, explorer, _))
85            & not nextStepExplorer(_, V, S), Options)
86           & .length(Options, TotalOptions) & TotalOptions > 0 &
87           .nth(math.random(TotalOptions), Options, Op).
88 there_is_unprobed_vertex_next_to_mine(Op) :- position(MyV) &
       infinite(INF) & myTeam(MyTeam) &
89           .setof(V,
90                ia.edge(MyV,V,W) & W \== INF &
91                not ia.probedVertex(V, _) &
92                not (visibleEntity(Entity, V, MyTeam, normal) &
93                friend(_, Entity, explorer, _)) &
94                not nextStepExplorer(_, V, S), Options)
95           & .length(Options, TotalOptions) & TotalOptions > 0 &
96           .nth(math.random(TotalOptions), Options, Op).
97
98 //Probe hills first
99 get_path_to_unprobed_probe(D, Path) :-
100          not is_disabled &
101          position(MyV) &
102          hill(Neighborhood) &
103          .setof(Vertex, .member(Vertex, Neighborhood) & ia.
                 probedVertex(Vertex,-1) &
104          not nextStepExplorer(_, Vertex, _), List) &
105          not .empty(List) &
106          ia.shortestPathDijkstraCompleteTwo(MyV, List, D, Path,
                 Lenght) &
107                   Lenght > 2.
108
109 //Try to probe some far vertex, it is better
110 get_path_to_unprobed_probe(D, Path) :-
111      not is_disabled & position(MyV) &
112      .setof(Vertex, ia.probedVertex(Vertex,-1) &
113      not nextStepExplorer(_, Vertex, _) , List) &
114      not .empty(List) &
115      ia.shortestPathDijkstraCompleteTwo(MyV, List, D, Path,
           Lenght) &
116               Lenght > 2.
```

```
117
118
119 +!wait_and_select_goal:
120     not numberWaits(_)
121 <-
122     +numberWaits(0);
123     !wait_and_select_goal.
124
125 +!wait_and_select_goal:
126     (numberWaits(K) & K >= 15) | step(0)
127 <-
128     .print("I can't wait anymore!");
129     -+numberWaits(0);
130     !select_goal.
131
132 +!wait_and_select_goal:
133     .my_name(MyName) & number_agents_higher_priority_same_type
        (MyName, NumberRequired) &
134     step(S) & .count(nextStepExplorer(_, _, S), Number) &
135     Number < NumberRequired &
136     numberWaits(K)
137 <-
138     .wait(50);
139     -+numberWaits(K+1);
140     !wait_and_select_goal.
141 +!wait_and_select_goal <- !select_goal.
142
143 +!select_goal: is_energy_goal
144         <- !init_goal(recharge).
145 +!select_goal : is_wait_repair_goal(V)
146         <- .print("Waiting to be repaired. Recharging...");
147             !init_goal(recharge).
148 +!select_goal : not canCome(ComeT) &
    is_wait_repair_goal_nearby(V, Repairer)
149         <- .print("Waiting to be repaired (nearby). ", V , "
                ", Repairer, ". Recharging...");
150             !init_goal(recharge).
151 +!select_goal : is_goto_repair_goal_nearby(V)
152         <- .print("Goto to be repaired (nearby)...");
153             !init_goal(goto(V)).
154
155 +!select_goal: is_disabled &
    get_vertex_to_go_be_repaired_appointment(D, Path)
156         <-
157             .print("I have an appointment with some repairer.
                    I'm going to ", D, " using path: ", Path);
158             !init_goal(gotoPath(Path)).
```

```
159 +!select_goal: is_disabled &
        get_vertex_to_go_be_repaired_appointment_self(D, Path)
160         <-
161             .print("I have an self appointment with some
                repairer. I'm going to ", D, " using path: ",
                Path);
162             !init_goal(gotoPath(Path)).
163 +!select_goal: is_disabled & get_vertex_to_go_repair(D, Path)
164         <-
165             .print("I'm forever alone. I'm going to ", D, "
                using path: ", Path);
166             !init_goal(gotoPath(Path)).
167
168 +!select_goal: is_probe_goal & not is_leave_goal
169         <- !init_goal(probe).
170
171 +!select_goal : is_good_map_conquered <-
172     .print("Good map conquered! Stopped!");
173     !init_goal(recharge).
174
175 /* Special explorers */
176 +!select_goal: there_is_unprobed_vertex_next_to_mine_special(
        Op) <-
177     .print("I'm a special explorer and I'm going to probe ",
            Op);
178     !init_goal(goto(Op)).
179 +!select_goal: get_path_to_unprobed_probe_special(D, Path) <-
180     .print("I'm a special explorer and I'm going to probe ", D
            , " using ", Path);
181     !init_goal(gotoPath(Path)).
182 /* End special explorers */
183
184 +!select_goal: is_probe_goal(Op)
185         <- !init_goal(goto(Op)).
186 +!select_goal: get_path_to_unprobed_probe(D, Path)
187         <- !init_goal(gotoPath(Path)).
188 +!select_goal: is_probe_goal //Probe even if I need to leave
189         <- !init_goal(probe).
190
191
192 /* Protect Island */
193 +!select_goal: is_keep_goal_island_enemy(Entity) &
        is_survey_goal <-
194     .print("I'm at the island and I'm going to stay here (
            survey) ", Entity);
195     !init_goal(survey).
196 +!select_goal: is_keep_goal_island_enemy(Entity) <-
```

```
197     .print("I'm at the island and I'm going to stay here ",
            Entity);
198     !init_goal(recharge).
199 +!select_goal: there_is_enemy_nearby_island_geral(Op) <-
200     .print("I'm at the island and I'm going to ", Op, " to
            find some enemy inside");
201     !init_goal(goto(Op)).
202 +!select_goal: get_vertex_to_go_attack_search_island_geral(D,
        Path) <-
203     .print("I'm at the island and I'm going to ", D, " using
            ", Path, " to find some enemy there");
204     !init_goal(gotoPath(Path)).
205 /* End Protect Island */
206
207 +!select_goal : going_to_outside_goal(V) <-
208     .print("I'm inside a region. I can expand to ", V);
209     !init_goal(goto(V)).
210
211 +!select_goal : can_expand_to(V) <-
212     .print("I can expand to ", V);
213     !init_goal(goto(V)).
214
215 +!select_goal: is_goal_keep_aim_vertex
216         <-
217             .print("I'm at a pivot vertex. Recharging...");
218             !init_goal(recharge).
219
220 +!select_goal: is_goal_aim_vertex(Op)
221         <-
222             .print("I have a pivot vertex to go. I'm going to
                    ", Op);
223             !init_goal(goto(Op)).
224 +!select_goal: is_goal_aim_vertex(D, Path)
225         <-
226             .print("I have a pivot vertex to go. I'm going to
                    ", D, " using path: ", Path);
227             !init_goal(gotoPath(Path)).
228
229 //Hills
230 +!select_goal : can_expand_to_hill(V) <-
231     .print("I'm at a hill and I can expand to ", V);
232     !init_goal(goto(V)).
233
234 +!select_goal : is_at_aim_position_hill <-
235     .print("Stop! Stand still! I'm settled at the hill!");
236     !init_goal(recharge).
237
```

```
238 +!select_goal: is_goal_hill_vertex(Op)
239         <-
240             .print("I have a hill vertex to go. I'm going to
                ", Op);
241             !init_goal(goto(Op)).
242 +!select_goal: is_goal_hill_vertex(D, Path)
243         <-
244             .print("I have a hill vertex to go. I'm going to
                ", D, " using path: ", Path);
245             !init_goal(gotoPath(Path)).
246
247 +!select_goal
248         <- !init_goal(random_walk).
249
250 /*
251  * Percept a new probed vertex and share with friends
252  */
253 +probedVertex(V,Value) [source(percept)]:
254     true
255 <-
256     .print("Vertex probed: ", V, " with value ", Value);
257     ia.setVertexValue(V, Value);
258     !broadcastProbe(V,Value).
259 +probedVertex(V,Value) [source(self)]: true <- .abolish(
    probedVertex(V,Value)).
260
261 +!broadcastProbe(V,Value):
262     .findall(Agent, friend(Agent, _, _, _), SetAgents)
263 <-
264     .print("Sending probed vertex in broadcast ", V, " ",
        Value);
265     .send(SetAgents, tell, probedVertex(V,Value)).
266 +!broadcastProbe(V,Value).
267
268 /*
269  * These functions must be dependent of each kind of agent
     because they will
270  * need to share some information with the other friends of
     the same kind
271  */
272 +!do(Act):
273     step(S) & stepDone(S)
274 <-
275     .print("ERROR! I already performed an action for this step
        ! ", S).
276
277 +!do(Act):
```

```
278     step(S) & .my_name(MyName) &
            get_agents_lower_priority_same_type(MyName, SetAgents)
279     & is_disabled
280 <-
281     .print("Sending action to ", S, " ", SetAgents);
282     .send(SetAgents,tell,nextStepExplorer(MyName, none, S));
283     !commitAction(Act);
284     !!clearNextStepExplorer.
285
286 +!do(goto(V)):
287     step(S) & .my_name(MyName) &
            get_agents_lower_priority_same_type(MyName, SetAgents)
288 <-
289     .print("Sending action to ", S, " ", SetAgents);
290     .send(SetAgents,tell,nextStepExplorer(MyName, V, S));
291     !commitAction(goto(V));
292     !!clearNextStepExplorer.
293
294 +!do(Act):
295     step(S) & position(V) & .my_name(MyName) &
            get_agents_lower_priority_same_type(MyName, SetAgents)
296 <-
297     .print("Sending action to ", S, " ", SetAgents);
298     .send(SetAgents,tell,nextStepExplorer(MyName, V, S));
299     !commitAction(Act);
300     !!clearNextStepExplorer.
301
302 +!clearNextStepExplorer <-
303     .abolish(nextStepExplorer(_, _, _)).
304
305 +!initSpecific.
306 +!processBeforeStep(S).
307 +!processAfterStep(S) <-
308     !!calculateTotalSumVertices;
309     !buildAreas(S).
310
311 /*
312  * CALCULATE THE BEST PLACES
313  */
314 +!buildAreas(S):
315     goalState(_,concludeFirstPhase,_,_,enabled) & obligation(
            MyName,_Norm,achieved(_Scheme,concludeFirstPhase,_),_)
             &
316     (not lastCalcCoverage(_) & S >= 7 | lastCalcCoverage(N) &
            S - N >= 13) & .my_name(MyName)
317 <-
318     !determineHills;
```

```
319     -+lastCalcCoverage(S).
320 +!buildAreas(S).
321
322 /* Update the sum of all vertices */
323 +!calculateTotalSumVertices:
324     .my_name(MyName) & play(MyName,explorerLeader,"grMain") &
            ia.sumVertices(Total) &
325     .findall(Agent, friend(Agent, _, _, _), SetAgents)
326 <-
327     .print("Calculating the sum of all vertices: ", Total);
328     !updateTotalSumVertices(Total);
329     .send(SetAgents, achieve, updateTotalSumVertices(Total)).
330 +!calculateTotalSumVertices.
```

A.4 Common rules

Prolog-like rules shared by all agents are located in this file.

```
1  // Test if I need energy
2  is_energy_goal :- energy(MyE) & minEnergy(Min) & MyE < Min
3      |
4    .my_name(MyName) & friend(MyName, _, repairer, _) &
         is_disabled & energy(MyE) & MyE < 3
5      |
6    .my_name(MyName) & friend(MyName, _, saboteur, _) & energy(
         MyE) & minEnergy(Min) & visRange(Vis) & Vis > 1 & MyE <
         5. //4 is the max distance that saboteurs attack
7
8  // Some edge to adjacent vertex is not surveyed
9  is_survey_goal :- not is_disabled & position(MyV) &
10     (
11       infinite(INF) & ia.edge(MyV,_,INF)
12     |
13       maxWeight(MAXWEIGHT) & ia.edge(MyV,_,MAXWEIGHT)
14     ).
15
16 // Test if the agent is disabled
17 is_disabled :- (health(MyHealth)[source(percept)] | health(
       MyHealth)[source(self)]) & MyHealth <= 0.
18
19 // Check if Agent1 has higher priority than Agent2
20 priorityEntity(Agent1, Agent2) :- friend(Agent1, _, _,
       Priority1) &
21     friend(Agent2, _, _, Priority2) &
22     Priority1 > Priority2.
23
24
25 // Number of agents higher priority
26 number_agents_higher_priority_same_type(Agent, C) :-
27     .count(friend(Agent, _, Type, Priority) &
28             friend(Agent2, _, Type, Priority2) &
29             Priority > Priority2, C).
30
31 // Get the set of agents with lower priority
32 get_agents_lower_priority_same_type(Agent, SetAgents) :-
33     .findall(Agent2,friend(Agent, _, Type, Priority) &
34             friend(Agent2, _, Type, Priority2) &
35             Priority < Priority2, SetAgents).
36
37 // Test if there is no more vertices to probe
38 noMoreVertexToProbe :- not ia.thereIsUnprobedVertex.
```

```
39
40 /*
41  * If I'm at some vertex with an enemy saboteur, so I count
        how many friends are there too.
42  * I choose to leave or to parry using probability.
43  * The probability to execute parry is: 1.0 / N, where N is
        the amount of friends
44  */
45 is_parry_goal :- not is_disabled & position(MyV) & myTeam(
       MyTeam)
46  & visibleEntity(Entity, MyV, Team, normal) &
47       Team \== MyTeam & entityType(Entity, "Saboteur", _, _, _)
             &
48       there_is_more_enemies_than_friends(MyV) &
49       .count(visibleEntity(_, MyV, MyTeam, normal), N) &
50       .random(K) & K <= (1.0 / N).
51 there_is_more_enemies_than_friends(V) :-
52     myTeam(MyTeam) &
53     .count((visibleEntity(Entity, V, MyTeam, normal)
54   & friend(_, Entity, saboteur, _)), NFriend) &
55     .count((visibleEntity(EntityEnemy, V, Team, normal)
56   & Team \== MyTeam & entityType(EntityEnemy, "Saboteur", _, _
       , _)), NEnemy) &
57     NEnemy > NFriend.
58 //That's the same idea for agents who can't parry, but they
       need to leave
59 is_leave_goal :- not is_disabled & position(MyV) & myTeam(
       MyTeam)
60     & visibleEntity(Entity, MyV, Team, normal) &
61          Team \== MyTeam & entityType(Entity, "Saboteur", _, _,
                _) &
62          there_is_more_enemies_than_friends(MyV).
63
64 //Check if there are visible saboteurs at some vertex
65 there_are_friends_saboteurs_at(V) :-
66              myTeam(MyTeam) &
67              myNameInContest(MyName) &
68              visibleEntity(Entity, V, MyTeam, normal) &
69              MyName \== Entity &
70              friend(_, Entity, saboteur, _).
71
72 there_are_friends_saboteurs_at(V) :-
73              .my_name(MyAgentName) &
74              friend(AgentName, _, saboteur, Id) & MyAgentName
                    \== AgentName & ia.getAgentPosition(Id, V).
75
76 //Check if there are visible saboteurs near at some vertex
```

```
77 there_are_friends_saboteurs_near(U) :-
78     .my_name(MyAgentName) &
79     myNameInContest(MyName) &
80     myTeam(MyTeam) &
81     (
82         (
83             visibleEntity(Entity, U, MyTeam, _) & MyName \==
                    Entity & friend(_, Entity, saboteur, _)
84             |
85             friend(AgentName, _, saboteur, Id) & MyAgentName
                    \== AgentName & ia.getAgentPosition(Id, U)
86         )
87     |
88         ia.edge(U,V,_) &
89         (
90             visibleEntity(Entity, V, MyTeam, _) & MyName \==
                    Entity & friend(_, Entity, saboteur, _)
91             |
92             friend(AgentName, _, saboteur, Id) & MyAgentName
                    \== AgentName & ia.getAgentPosition(Id, V)
93         )
94     ).
95
96 //Verify if there is some dangerous enemies at vertex Op. A
       dangerous enemy is an unknown enemy or a Saboteur
97 there_is_enemy_at(Op) :-
98     myTeam(MyTeam) &
99     visibleEntity(Entity, Op, Team, normal) &
100    Team \== MyTeam &
101    (entityType(Entity, "Saboteur", _, _, _) | not entityType(
           Entity, _, _, _, _)).
102
103 //Verify if there is any kind of enemy at vertex Op
104 there_is_any_enemy_at(Op) :-
105    myTeam(MyTeam) &
106    visibleEntity(Entity, Op, Team, normal) &
107    Team \== MyTeam.
108 there_is_any_enemy_at(Op) :-
109    ia.visibleEnemy(Entity, Op).
110
111 //Verify if there is some friend at vertex Op
112 there_is_friend_at(Op) :- myTeam(MyTeam) & visibleEntity(
       Entity, Op, MyTeam, normal).
113 there_is_friend_at(Op) :-
114     .my_name(MyAgentName) &
115     friend(AgentName, _, _, Id) &
116     MyAgentName \== AgentName &
```

```
117       ia.getAgentPosition(Id, Op).
118
119 is_good_map_conquered :- step(S) & S > 300 & is_free_to_walk &
        sumVertices(Total) & zonesScore(Score) & Score >= Total *
        0.45 & at_some_good_zone_score(0.34).
120 is_good_map_conquered :- step(S) & S > 150 & is_free_to_walk &
        sumVertices(Total) & zonesScore(Score) & Score >= Total *
        0.47 & at_some_good_zone_score(0.35).
121 is_good_map_conquered :- step(S) & S > 80 & is_free_to_walk &
        sumVertices(Total) & zonesScore(Score) & Score >= Total *
        0.49 & at_some_good_zone_score(0.35).
122 is_good_map_conquered :- step(S) & S > 25 & is_free_to_walk &
        sumVertices(Total) & zonesScore(Score) & Score >= Total *
        0.55 & at_some_good_zone_score(0.35).
123
124 at_some_good_zone_score(Perc) :- not is_disabled & zonesScore(
        Score) & zoneScore(MyZoneScore) & (MyZoneScore / Score) >=
         Perc.
125
126 there_are_enemies_saboteurs_near(U) :-
127     .my_name(MyAgentName) &
128     myNameInContest(MyName) &
129     myTeam(MyTeam) &
130     (
131         (
132             visibleEntity(Entity, U, Team, _) & Team \==
                    MyTeam & entityType(Entity, "Saboteur", _, _,
                    _)
133             |
134             ia.visibleEnemy(Entity, U) & entityType(Entity, "
                    Saboteur", _, _, _)
135         )
136     |
137         ia.edge(U,V,_) &
138         (
139             visibleEntity(Entity, V, Team, _) & Team \==
                    MyTeam & entityType(Entity, "Saboteur", _, _,
                    _)
140             |
141             ia.visibleEnemy(Entity, V) & entityType(Entity, "
                    Saboteur", _, _, _)
142         )
143     ).
```

A.5 New step

Code in this file is responsible for handling each new step of the simulation.

```
1  +step(S):
2      steps(S+1) & .my_name(inspector1)
3  <-
4      .print("Current step is ", S, " the last one! I'm
           inspector1 and I'm doing noAction on the last step!");
5      !finishSimulation.
6
7  +step(S):
8      steps(S+1) | (lastStep(LastS) & LastS > S)
9  <-
10     .print("Current step is ", S, " the last one!");
11     .wait(100); //wait a bit because I can receive more
           information about the other agents
12     !wait_and_select_goal;
13     !finishSimulation.
14
15 +step(S):
16     true
17 <-
18     .print("Current step is ", S);
19     !processBeforeStep(S);
20     !!testCleanIsland;
21     !!testCleanPivot;
22     .wait(100); //wait a bit because I can receive more
           information about the other agents
23     !wait_and_select_goal;
24     !sendToken(S);
25     !evaluateHealth(S);
26     -+lastStep(S);
27     !processAfterStep(S);
28     !evaluateCanCome(S);
29     !!recoverySystem;
30     !!addLineLog(S);
31     !!synchronizeGraph.
32
33 @commitAction[atomic]
34 +!commitAction(Act):
35     step(S)
36 <-
37     -+stepDone(S);
38     Act.
39
40 /* Which goal I'm going to follow */
```

```
41 +!init_goal(G):
42     money(M) & step(S) & position(V) & energy(E) & maxEnergy(
           Max) & lastActionResult(Result) & score(Score) &
           health(Health) & lastAction(LastAction) &
           lastStepScore(LastScore) & zonesScore(ZoneScore) &
           zoneScore(MyZoneScore)
43 <-
44     .print("I am at ",V," (",E,"/",Max,"), my health is ",
           Health, " the goal for step ",S," is ",G, " and I have
            ", M, " of money. My last result was ", Result, ". My
            last action was ", LastAction, ". The score is ",
           Score, " and my last score was ", LastScore, " with
           zones ", ZoneScore, " and my zone is ", MyZoneScore);
45     !G.
46
47 +!init_goal(G):
48     step(S) & position(V) & energy(E) & maxEnergy(Max)
49 <-
50     .print("Something wrong... I'going try to don't lose the
           step. I'm at ",V," (",E,"/",Max,"). My action for step
            ",S," is ", G);
51     !G.
52
53 +!init_goal(_)
54 <-
55     .print("No step yet... wait a bit");
56     .wait(500);
57     !wait_and_select_goal.
58
59 /* Log System */
60 +!addLineLog(S):
61     lastActionResult(Result) & lastAction(LastAction) & .
           my_name(MyName) & friend(MyName, _, _, AgentId) &
           myTeam(MyTeam)
62 <-
63     ia.addLastAction(MyTeam, AgentId, S, LastAction, Result).
64 +!addLineLog(S):
65     .my_name(MyName) & friend(MyName, _, _, AgentId) & myTeam(
           MyTeam)
66 <-
67     ia.addLastAction(MyTeam, AgentId, S, error, error).
68 +!addLineLog(_).
69
70 /* Graph synchronization */
71 +!synchronizeGraph:
72     step(S) & lastSync(Last) & S - Last > 2
73 <-
```

```
74      .print("Synchronizing...");
75      -+lastSync(S);
76      ia.synchronizeGraph.
77  +!synchronizeGraph:
78      not lastSync(_)
79  <-
80      +lastSync(0).
81  +!synchronizeGraph.
82
83  +!recoverySystem:
84      not steps(_) | not edges(_) | not vertices(_)
85  <-
86      .send(coach, askOne, steps(_));
87      .send(coach, askOne, edges(_));
88      .send(coach, askOne, vertices(_)).
89  +!recoverySystem.
```

A.6 Graph

Code in this file has the main responsibilities of the systems graph library.

```
1  package graphLib;
2
3  import java.util.ArrayList;
4  import java.util.LinkedList;
5  import java.util.List;
6
7  public class Graph {
8      public static final int INF = 10000;
9      public static final int MAXWEIGHT = 10;
10     public static final int NULL = -1;
11     public static final int MAXVERTICES = 1000;
12     public static final int MAXEDGES = 10000;
13     public static final int MAXVERTEXVALUE = 10;
14
15     private int edgeCounter = 0;
16
17     public int values[] = new int[MAXVERTICES];
18     public int grade[] = new int[MAXVERTICES];
19     public int w[][] = new int[MAXVERTICES][MAXVERTICES];
20     public int adj[][] = new int[MAXVERTICES][MAXVERTICES];
21     public String teams[] = new String[MAXVERTICES];
22     public int visited[] = new int[MAXVERTICES];
23     public boolean known[] = new boolean[MAXVERTICES];
24     public String integer2vertex[] = new String[MAXVERTICES];
25
26     private int maxVerticesSim = MAXVERTICES-1;
27     private int maxEdgesSim = MAXEDGES-1;
28
29     private boolean sumVertexCalculated = false;
30     private int sumVertex = 1;
31     private int averageVertex = 1;
32     private boolean allVertexProbed = false;
33
34
35     public Graph() {
36         //Graph initialization
37         for (int i = 0; i < MAXVERTICES; i++) {
38             values[i] = NULL; //Every vertex not probed has -1
39             grade[i]  = 0; //The initial grade of each vertex
                   is 0
40             teams[i] = "none";
41             integer2vertex[i] = "v" + i;
42             visited[i] = NULL;
```

```
43              known[i] = false;
44              for (int j = 0; j < MAXVERTICES; j++) {
45                  w[i][j] = INF; //The initial weight of each
                        edge if INFINITE
46              }
47          }
48      }
49
50      public int vertex2Integer(String vertex) {
51          return Integer.valueOf(vertex.substring(1, vertex.
                length()));
52      }
53
54      public void addVertex(String vertexV, String team) {
55          int v = vertex2Integer(vertexV);
56
57          teams[v] = team;
58      }
59
60      public void resetTeamsVertice() {
61          for (int v = 0; v <= getSize(); v++) {
62              teams[v] = "none";
63          }
64      }
65
66      public void addEdge(String vertexU, String vertexV, int
            weight) {
67          //Get the id of each vertex
68          int u = vertex2Integer(vertexU);
69          int v = vertex2Integer(vertexV);
70
71          if (w[u][v] == INF && weight != INF) {
72              //Add the weight of the edge
73              w[u][v] = w[v][u] = weight;
74              //Add the edge into the graph and increase the
                    grade of each vertex
75
76              adj[u][grade[u]++] = v;
77              adj[v][grade[v]++] = u;
78              edgeCounter++;
79
80              known[u] = true;
81              known[v] = true;
82          } else if (weight < w[u][v]) {
83              //Add the weight of the edge
84              w[u][v] = w[v][u] = weight;
85
```

```
86              known[u] = true;
87              known[v] = true;
88          }
89      }
90
91      public void addEdgeSync(int u, int v, int weight) {
92          if (w[u][v] == INF && weight != INF) {
93              //Add the weight of the edge
94              w[u][v] = w[v][u] = weight;
95              //Add the edge into the graph and increase the
                    grade of each vertex
96
97              adj[u][grade[u]++] = v;
98              adj[v][grade[v]++] = u;
99              edgeCounter++;
100
101             known[u] = true;
102             known[v] = true;
103         } else if (weight < w[u][v]) {
104             //Add the weight of the edge
105             w[u][v] = w[v][u] = weight;
106
107             known[u] = true;
108             known[v] = true;
109         }
110     }
111
112     public void setVertexValue(String vertexV, int value) {
113         //Get the id of each vertex
114         int v = vertex2Integer(vertexV);
115
116         //Update the value of the vertex
117         values[v] = value;
118     }
119
120     public void setVertexVisited(String vertexV, int step) {
121         //Get the id of each vertex
122         int v = vertex2Integer(vertexV);
123
124         //Update the value of the vertex
125         visited[v] = step;
126     }
127
128     public int getSize() {
129         return maxVerticesSim;
130     }
131
```

```
132     public List<String> getShortestPath(String vertexS, String
            vertexD) {
133         LinkedList<String> result = null;
134         DijkstraAlgorithm dijkstra = new DijkstraAlgorithm();
135 
136         int s = vertex2Integer(vertexS);
137         int d = vertex2Integer(vertexD);
138 
139         List<Integer> resultDijkstra = dijkstra.execute(this,
                s, d);
140 
141         result = new LinkedList<String>();
142 
143         if(resultDijkstra != null) {
144             for (int i : resultDijkstra) {
145                 result.addFirst(integer2vertex[i]);
146             }
147         }
148 
149         return result;
150     }
151 
152     public List<String> getShortestPathDijkstraComplete(String
            vertexS, List<String> vertexD) {
153         LinkedList<String> result = null;
154         DijkstraAlgorithmComplete dijkstra = new
                DijkstraAlgorithmComplete();
155 
156         List<Integer> listD = new LinkedList<Integer>();
157 
158         int s = vertex2Integer(vertexS);
159 
160         for (String v : vertexD) {
161             listD.add(vertex2Integer(v));
162         }
163 
164         List<Integer> resultDijkstra = dijkstra.execute(this,
                s, listD);
165 
166         result = new LinkedList<String>();
167 
168         if(resultDijkstra != null) {
169             for (int i : resultDijkstra) {
170                 result.addFirst(integer2vertex[i]);
171             }
172         }
173 
```

```
174         return result;
175     }
176 
177     public List<String> getShortestPathBFSComplete(String
            vertexS, List<String> vertexD) {
178         LinkedList<String> result = null;
179         BFSAlgorithm bfs = new BFSAlgorithm();
180 
181         List<Integer> listD = new LinkedList<Integer>();
182 
183         int s = vertex2Integer(vertexS);
184 
185         for (String v : vertexD) {
186             listD.add(vertex2Integer(v));
187         }
188 
189         List<Integer> resultBFS = bfs.execute(this, s, listD);
190 
191         result = new LinkedList<String>();
192 
193         if(resultBFS != null) {
194             for (int i : resultBFS) {
195                 result.addFirst(integer2vertex[i]);
196             }
197         }
198 
199         return result;
200     }
201 
202     public List<String> getBestCoverage(int depth) {
203         List<String> result = null;
204         BestCoverageInterface bestCoverage;
205         int paramInt;
206         if (getMaxEdges() / (double) getMaxVertices() > 2) {
207             bestCoverage = new BestCoverage();
208             paramInt = depth;
209         } else {
210             bestCoverage = new BestCoverageFewEdges();
211             paramInt = depth * 10;
212         }
213 
214         List<Integer> list = bestCoverage.execute(this,
                paramInt);
215 
216         if (list.size() > 0) {
217             result = new ArrayList<String>();
218 
```

```
219             result.add(0, integer2vertex[list.get(0)]);
220             result.add(1, String.valueOf(list.get(1)));
221         }
222 
223         return result;
224     }
225 
226     public List<String> getBestCoverage(int depth, String
        vertexIgnore) {
227         List<String> result = null;
228         BestCoverageInterface bestCoverage;
229         int paramInt;
230         if (getMaxEdges() / (double) getMaxVertices() > 1) {
231             bestCoverage = new BestCoverage();
232             paramInt = depth;
233         } else {
234             bestCoverage = new BestCoverageFewEdges();
235             paramInt = depth * 10;
236         }
237 
238         List<Integer> list = bestCoverage.execute(this,
            paramInt, vertex2Integer(vertexIgnore));
239 
240         if (list.size() > 0) {
241             result = new ArrayList<String>();
242 
243             result.add(0, integer2vertex[list.get(0)]);
244             result.add(1, String.valueOf(list.get(1)));
245         }
246 
247         return result;
248     }
249 
250     public List<String> getNeighborhood(String vertex, int
        depth) {
251         List<String> result = null;
252         NeighborhoodInterface bestCoverage;
253 
254         int paramInt;
255         if (getMaxEdges() / (double) getMaxVertices() > 1) {
256             bestCoverage = new Neighborhood();
257             paramInt = depth;
258         } else {
259             bestCoverage = new NeighborhoodFewEdges();
260             paramInt = (depth + 1) * 6;
261         }
262 
```

```
263
264         int s = vertex2Integer(vertex);
265         List<Integer> list = bestCoverage.execute(this, s,
                paramInt);
266
267         if (list.size() > 0) {
268             result = new LinkedList<String>();
269             for (int i : list) {
270                 result.add(integer2vertex[i]);
271             }
272         }
273
274         return result;
275     }
276
277     public String getTeamAtVertex(String vertexV) {
278         //Get the id of the vertex
279         int v = vertex2Integer(vertexV);
280
281         return teams[v];
282     }
283
284     public int getGrade(String vertexV) {
285         //Get the id of the vertex
286         int v = vertex2Integer(vertexV);
287
288         return grade[v];
289     }
290
291     public int getVertexValue(String vertexV) {
292         //Get the id of the vertex
293         int v = vertex2Integer(vertexV);
294
295         return values[v];
296     }
297
298     public int getVertexVisited(String vertexV) {
299         //Get the id of the vertex
300         int v = vertex2Integer(vertexV);
301
302         return visited[v];
303     }
304
305     public String getAdj(String vertexU, int index) {
306         //Get the id of the vertex
307         int u = vertex2Integer(vertexU);
308
```

```
309         int v = adj[u][index];
310         return integer2vertex[v];
311     }
312
313     public int getWeight(String vertexU, int index) {
314         //Get the id of the vertex
315         int u = vertex2Integer(vertexU);
316
317         int v = adj[u][index];
318         return w[u][v];
319     }
320
321     public List<String> getVertexByValue(int value) {
322         List<String> list = new ArrayList<String>();
323
324         for (int v = 0; v < getSize(); v++) {
325             if (values[v] == value) {
326                 list.add(integer2vertex[v]);
327             }
328         }
329
330         return list;
331     }
332
333     public void setMaxEdges(int maxEdges) {
334         this.maxEdgesSim = maxEdges;
335     }
336
337     public void setMaxVertices(int maxVertices) {
338         this.maxVerticesSim = maxVertices;
339     }
340
341     public int getMaxVertices() {
342         return maxVerticesSim;
343     }
344
345     public int getMaxEdges() {
346         return maxEdgesSim;
347     }
348
349     public int getEdges() {
350         return edgeCounter;
351     }
352
353     public boolean thereIsUnprobedVertex() {
354         if (allVertexProbed)
355             return false;
```

```
356
357         if (edgeCounter == 0)
358             return true;
359
360         for (int v = 0; v < getSize(); v++) {
361             if (values[v] == Graph.NULL) {
362                 return true;
363             }
364         }
365
366         allVertexProbed = true;
367         return false;
368     }
369
370     private void calcStatVertices() {
371         if (sumVertexCalculated)
372             return;
373
374         int total = 0;
375         int qtde = 0;
376         for (int v = 0; v < getSize(); v++) {
377             if (values[v] != Graph.NULL) {
378                 total += values[v];
379                 qtde++;
380             }
381         }
382
383         sumVertex = total;
384         if (qtde > 0)
385             averageVertex = total / qtde;
386         if (allVertexProbed || !thereIsUnprobedVertex())
387             sumVertexCalculated = true;
388     }
389
390     public int getSumVertices() {
391         calcStatVertices();
392         return sumVertex;
393     }
394
395     public int getAverageVertex() {
396         calcStatVertices();
397         return averageVertex;
398     }
399
400     public List<String> getAllVertices() {
401         ArrayList<String> vertexList = new ArrayList<String>()
                ;
```

```
402
403             for (int v = 0; v < getSize(); v++) {
404                 if (grade[v] > 0) {
405                     vertexList.add(integer2vertex[v]);
406                 }
407             }
408
409             return vertexList;
410         }
411
412         public List<PairPivot> getAllPivots(int amount) {
413             PivotAlgorithm p = new PivotAlgorithm();
414             return p.execute(this, amount);
415         }
416
417         public List<PairPivot> getAllPivotsIgnoringVertices(int
                amount, List<String> listVerticesIgnore) {
418             List<Integer> listVerticesIgnoreInt = new LinkedList<
                    Integer>();
419
420             for (String v : listVerticesIgnore) {
421                 listVerticesIgnoreInt.add(vertex2Integer(v));
422             }
423
424             PivotAlgorithm p = new PivotAlgorithm();
425             p.setVerticesToIgnore(listVerticesIgnoreInt);
426             return p.execute(this, amount);
427         }
428
429         public List<PairPivot> getAllPivotsJustSomeVertices(int
                amount, List<String> listVertices) {
430             List<Integer> listVerticesInt = new LinkedList<Integer
                    >();
431
432             for (String v : listVertices) {
433                 listVerticesInt.add(vertex2Integer(v));
434             }
435
436             PivotAlgorithm p = new PivotAlgorithm();
437             p.setVerticesToJustUse(listVerticesInt);
438             return p.execute(this, amount);
439         }
440
441         public List<Island> getAllIslands(int amount) {
442             IslandAlgorithm p = new IslandAlgorithm();
443             return p.execute(this, amount);
444         }
```

```
445
446     public int getDistance(String vertexS, String vertexD) {
447         DistanceAlgorithm distance = new DistanceAlgorithm();
448
449         int s = vertex2Integer(vertexS);
450         int d = vertex2Integer(vertexD);
451
452         return distance.execute(this, s, d);
453     }
454 }
```

Appendix B

Changes to the UFSC code

This appendix shows the changes made to the UFSC system code files listed in appendix A.

B.1 Sentinel

The changed code for the sentinel agents. New code on line numbers: 3-17, 52-54, 133-135 and 140.

```
1 { include("mod.common.asl") }
2
3 /*Test if neighbouring vertices have unsurveyed edges
4  * and pick the best one (the one with the highest number of
       such edges).
5  */
6 unsurveyed_edge_nearby(Op) :- infinite(INF) & maxWeight(
      MAXWEIGHT) & position(MyV) & not is_disabled & not
      is_survey_goal
7         & .setof(NearbyVertex, (ia.edge(MyV,NearbyVertex,W) & W
              \== INF & W \== MAXWEIGHT), Nearby)
8         & .length(Nearby,Length) & Length > 0 & .member(Op,
              Nearby) & test(Nearby,0,Op).
```

```
9
10 /*Helper function for unsurveyed_edge_nearby(Op). Given a list
       of vertices, finds one with most unsurveyed edges.*/
11
12 test([],L,_) :- L > 0.
13 test([H|T],L,H) :- infinite(INF) & maxWeight(MAXWEIGHT) & .
       setof(W,(ia.edge(H,W,INF)
14 | ia.edge(H,W,MAXWEIGHT)),Unknowns) & .length(Unknowns,L1) &
       L1 > L & test(T,L1,H).
15 test([H|T],L,Res) :- infinite(INF) & maxWeight(MAXWEIGHT) & .
       setof(W,(ia.edge(H,W,INF)
16 | ia.edge(H,W,MAXWEIGHT)),Unknowns) & .length(Unknowns,L1) &
       L1 <= L & test(T,L,Res).
17
18
19 +!wait_and_select_goal <- !select_goal.
20 -!wait_and_select_goal[error(deadline_reached)] <- .print("
       Deadline reached"); !select_goal.
21
22 +!select_goal: is_energy_goal
23         <- !init_goal(recharge).
24 +!select_goal : is_wait_repair_goal(V)
25         <- .print("Waiting to be repaired. Recharging...");
26             !init_goal(recharge).
27 +!select_goal : not canCome(ComeT) &
       is_wait_repair_goal_nearby(V, Repairer)
28         <- .print("Waiting to be repaired (nearby). ", V , "
               ", Repairer, ". Recharging...");
29             !init_goal(recharge).
30 +!select_goal : is_goto_repair_goal_nearby(V)
31         <- .print("Goto to be repaired (nearby)...");
32             !init_goal(goto(V)).
33
34 +!select_goal: is_disabled &
       get_vertex_to_go_be_repaired_appointment(D, Path)
35         <-
36             .print("I have an appointment with some repairer.
                  I'm going to ", D, " using path: ", Path);
37             !init_goal(gotoPath(Path)).
38 +!select_goal: is_disabled &
       get_vertex_to_go_be_repaired_appointment_self(D, Path)
39         <-
40             .print("I have an self appointment with some
                  repairer. I'm going to ", D, " using path: ",
                  Path);
41             !init_goal(gotoPath(Path)).
42 +!select_goal: is_disabled & get_vertex_to_go_repair(D, Path)
```

```
43          <-
44              .print("I'm forever alone. I'm going to ", D, "
                    using path: ", Path);
45              !init_goal(gotoPath(Path)).
46
47  +!select_goal : is_parry_goal
48          <- !init_goal(parry).
49  +!select_goal : is_survey_goal
50          <- !init_goal(survey).
51
52  +!select_goal : unsurveyed_edge_nearby(Op) & step(S) & S < 133
53      <- .print("Vertex ", Op, " has some unsurveyed edges! I'm
            going there");
54          !init_goal(goto(Op)).
55
56  +!select_goal : is_good_map_conquered <-
57      .print("Good map conquered! Stopped!");
58      !init_goal(recharge).
59
60  /* Protect Island */
61  +!select_goal: is_keep_goal_island_enemy(Entity) <-
62      .print("I'm at the island and I'm going to stay here ",
            Entity);
63      !init_goal(recharge).
64  +!select_goal: there_is_enemy_nearby_island_geral(Op) <-
65      .print("I'm at the island and I'm going to ", Op, " to
            find some enemy inside");
66      !init_goal(goto(Op)).
67  +!select_goal: get_vertex_to_go_attack_search_island_geral(D,
        Path) <-
68      .print("I'm at the island and I'm going to ", D, " using
            ", Path, " to find some enemy there");
69      !init_goal(gotoPath(Path)).
70  /* End Protect Island */
71
72  +!select_goal : going_to_outside_goal(V) <-
73      .print("I'm inside a region. I can expand to ", V);
74      !init_goal(goto(V)).
75
76  +!select_goal : can_expand_to(V) <-
77      .print("I can expand to ", V);
78      !init_goal(goto(V)).
79
80  +!select_goal: is_goal_keep_aim_vertex
81          <-
82              .print("I'm at a pivot vertex. Recharging...");
83              !init_goal(recharge).
```

```
84
85 +!select_goal: is_goal_aim_vertex(Op)
86         <-
87             .print("I have a pivot vertex to go. I'm going to
                ", Op);
88             !init_goal(goto(Op)).
89
90 +!select_goal: is_goal_aim_vertex(D, Path)
91         <-
92             .print("I have a pivot vertex to go. I'm going to
                ", D, " using path: ", Path);
93             !init_goal(gotoPath(Path)).
94
95 //Hills
96 +!select_goal : can_expand_to_hill(V) <-
97     .print("I'm at a hill and I can expand to ", V);
98     !init_goal(goto(V)).
99
100 +!select_goal : is_at_aim_position_hill <-
101     .print("Stop! Stand still! I'm settled at the hill!");
102     !init_goal(recharge).
103
104 +!select_goal: is_goal_hill_vertex(Op)
105         <-
106             .print("I have a hill vertex to go. I'm going to
                ", Op);
107             !init_goal(goto(Op)).
108 +!select_goal: is_goal_hill_vertex(D, Path)
109         <-
110             .print("I have a hill vertex to go. I'm going to
                ", D, " using path: ", Path);
111             !init_goal(gotoPath(Path)).
112
113 +!select_goal
114         <- !init_goal(random_walk).
115
116 /*
117  * These functions must be dependent of each kind of agent
       because they will
118  * need to share some information with the other friends of
       the same kind
119  */
120 @do0[atomic]
121 +!do(Act):
122     step(S) & stepDone(S)
123 <-
124     .print("ERROR! I already performed an action for this step
```

```
        ! ", S).
125
126 @do1[atomic]
127 +!do(Act):
128     true
129 <-
130     !commitAction(Act).
131
132 +!initSpecific.
133 //Tell lead inspector about spotted uninspected entities
134 +!processBeforeStep(S) : S > 50 & unknown_enemy_visible(Op,
        Entity)
135 <- .print("Told lead inspector about entity ", Entity, " at ",
        Op, ".") ; .send(inspector6, tell, enemy(Op, Entity)).
136 +!processBeforeStep(S).
137 +!processAfterStep(S) <-
138     !buildPivots(S);
139     !buildIslands(S).
140 +!auction. //Ignore inspector auctions
141 /*
142  * CALCULATE THE BEST PLACES
143  */
144 +!buildPivots(S):
145     (not lastCalcPivot(_) & S >= 17 | lastCalcPivot(N) & S - N
            >= 17) & .my_name(MyName) & play(MyName,
            sentinelLeader,"grMain")
146 <-
147     !determinePivots;
148     -+lastCalcPivot(S).
149 +!buildPivots(S).
150
151 +!buildIslands(S):
152     not hill(_) & goalState(_,defineInitialPivots,_,_,
            satisfied) & (not lastCalcIsland(_) & S >= 23 |
            lastCalcIsland(N) & S - N >= 23) & .my_name(MyName) &
            play(MyName,sentinelLeader,"grMain")
153 <-
154     !determineIslands;
155     -+lastCalcIsland(S).
156 -!buildIslands[error(ErrorCode),error_msg(MsgError)] <-
157     .print("Error occurred to build islands! ", ErrorCode, "
            -> ", MsgError).
158 +!buildIslands(S).
```

B.2 Inspector

The changed code for the inspector agents. New code on line numbers: 11-103, 130-134, 160-163, 237-239.

```
1  { include("mod.common.asl") }
2
3  //There is no priority because one of them can fail
4  is_inspect_goal(V, Entity) :- not is_disabled & position(MyV)
       & myTeam(MyTeam) & visibleEntity(Entity, V, Team, _) &
       Team \== MyTeam & ia.edge(MyV,V,_)
5                        & not entityType(Entity, _, _, _, _).
6
7  is_inspect_goal(Entity) :- not is_disabled & position(MyV) &
       myTeam(MyTeam) & visibleEntity(Entity, MyV, Team, _) &
       Team \== MyTeam
8                       & not entityType(Entity, _, _, _, _).
9
10
11 //The auction algorithm
12 @pb1[atomic]
13 +place_bid(_) //Case 1: all inspectors bid 99 - they are busy,
       too far or cannot find a way. No winner for this auction
14     :   .findall(bids(Bid,Ag), (place_bid(Bid)[source(Ag)] & Bid
           == 99), List) &
15         .length(List, 5)
16     <- .print("No winner for this auction!");
17         .abolish(place_bid(_)); +single_auction_done ; .abolish(
               single_auction_done).
18
19 @pb2[atomic]
20 +place_bid(_) //Case 2: at least one inspector has made a bid
       - a winner can be decided
21     :   .findall(bids(Bid,Ag), place_bid(Bid)[source(Ag)], List)
           &
22         .length(List, 5)
23     <- .min(List, bids(Bid, W));
24         .print("Winner is ", W, " with ", Bid);
25        +winner(W);
26         .abolish(place_bid(_)); +single_auction_done ; .abolish(
               single_auction_done).
27
28 //More important things to do than the auction
29 cannot_participate_in_auction :- is_energy_goal | is_disabled
       | is_inspect_goal(_) | is_good_map_conquered
30 | is_keep_goal_island_enemy(_) | is_inspect_goal(_, _) |
       there_is_enemy_nearby_island_geral(_) |
```

```
    get_vertex_to_go_attack_search_island_geral(_, _).
31
32 //More important things to do, skip the auction
33 +auction(Op)[source(Ag)] : cannot_participate_in_auction
34    <- .print("Not participating in auction, important stuff to
          do!") ; .send(Ag, tell, place_bid(99)) ; .abolish(
          auction(Op)).
35
36 //Already won one auction, current one is farther away
37 +auction(Op)[source(Ag)] : winner(_, Op1) & position(MyV) & ia
     .shortestPath(MyV, Op, _, Length) & ia.shortestPath(MyV,
     Op1, _, Length1) & Length1 <= Length
38    <- .print("I already won an auction and the path to current
          one is longer than the one I won!") ; .send(Ag, tell,
          place_bid(99)) ; .abolish(auction(Op)).
39
40 //Already won one auction, but the current one is closer
41 +auction(Op)[source(Ag)] : winner(_, Op1) & position(MyV) & ia
     .shortestPath(MyV, Op, _, Length) & ia.shortestPath(MyV,
     Op1, _, Length1) & Length1 > Length & Length > 1
42    <- .print("Bidding ", Length, "!") ; .send(Ag, tell,
          place_bid(Length)) ; .abolish(auction(Op)).
43
44 //Placing a bid - the distance between me and the entity which
      needs to be inspected
45 +auction(Op)[source(Ag)] : position(MyV) & ia.shortestPath(MyV
     , Op, _, Length) & Length < 6 & Length > 1 <- .print("
     Bidding ", Length, "!") ; .send(Ag, tell, place_bid(Length
     )) ; .abolish(auction(Op)).
46
47 //Path too long, not participating
48 +auction(Op)[source(Ag)] : position(MyV) & ia.shortestPath(MyV
     , Op, _, Length) & Length >= 6 <- .print("Not
     participating in auction, path too long!") ; .send(Ag,
     tell, place_bid(99)) ; .abolish(auction(Op)).
49
50 //Could not find path, not participating
51 +auction(_)[source(Ag)] : true <- .print("Not participating in
      auction, cannot find way!") ; .send(Ag, tell, place_bid
     (99)) ; .abolish(auction(Op)).
52
53 //Lead inspector: Gather all received messages about unknown
     entities and remove duplicates
54 +!auction: step(S) & S > 50 & .my_name(MyName) & play(MyName,
     inspectorLeader, _) <-
55 .setof(Op, enemy(Op, _), List);
56 .print("Entities to auction ", List);
```

```
57 .length(List, L);
58 !auction2(L, List).
59
60 //Normal inspectors: All auctions done, proceed
61 +!auction: step(S) & S > 50 & .my_name(MyName) & play(MyName,
       inspector, _) & auction_done <- true.
62
63 //Normal inspectors: Auctions are being held, await
       information about the next one
64 +!auction: step(S) & S > 50 & .my_name(MyName) & play(MyName,
       inspector, _) & auctions <- .wait("+auction(_,_)", 25) ; !
       auction.
65
66 //Normal inspectors: Wait for information from lead inspector
67 +!auction: step(S) & S > 50 & .my_name(MyName) & play(MyName,
       inspector, _) <- .wait(25) ; !auction.
68
69 +!auction.
70 -!auction.
71
72 //Lead inspector: No entities to be auctioned, let the
       inspectors know that they may proceed
73 +!auction2(L, List): L == 0 <-
74 .print("No auctions in this step");
75 .send([inspector1, inspector2, inspector3, inspector4,
       inspector5], tell, auction_done).
76
77 //Lead inspector: Initialise - tell other inspectors that some
        auctions are going to be held
78 +!auction2(L, List):
79     not number(_)
80 <-
81   .print("Auctions in this step ", L);
82   .send([inspector1, inspector2, inspector3, inspector4,
         inspector5], tell, auctions);
83    +number(0);
84    !auction2(L, List).
85
86 //Lead inspector: All auctions completed, let the inspectors
       know that they may proceed
87  +!auction2(L, List):
88     number(K) & K >= L
89 <-
90   .abolish(number(_));
91   .print("All auctions for this step finished");
92   .send([inspector1, inspector2, inspector3, inspector4,
         inspector5], tell, auction_done).
```

```
93
94 //Lead inspector: Pick the next entity from the list and start
       an auction for it
95 +!auction2(L, List) : number(K) <- .nth(K, List, Op);
96 .send([inspector1, inspector2, inspector3, inspector4,
       inspector5], tell, auction(Op));
97 .print("Current: ", Op, "."); -+number(K+1) ; .wait("+
       single_auction_done") ; ?winner(W) ; .send(W, tell, winner
       (W,Op)) ; .abolish(winner(W)) ; !auction2(L,List).
98
99 //Lead inspector: For when there is no winner of an auction -
       the ?winner(W) call then fails
100 -!auction2(L, List) <- !auction2(L, List).
101
102 +!wait_and_select_goal: is_inspect_goal(_) <- .random(K) ; .
       wait(K*50) ; !select_goal.
103 +!wait_and_select_goal <- !select_goal.
104 -!wait_and_select_goal[error(deadline_reached)] <- .print("
       Deadline reached"); !select_goal.
105
106 +!select_goal: is_energy_goal
107         <- !init_goal(recharge).
108 +!select_goal : is_wait_repair_goal(V)
109         <- .print("Waiting to be repaired. Recharging...");
110             !init_goal(recharge).
111 +!select_goal : not canCome(ComeT) &
       is_wait_repair_goal_nearby(V, Repairer)
112         <- .print("Waiting to be repaired (nearby). ", V , "
                ", Repairer, ". Recharging...");
113             !init_goal(recharge).
114 +!select_goal : is_goto_repair_goal_nearby(V)
115         <- .print("Goto to be repaired (nearby)...");
116             !init_goal(goto(V)).
117
118 +!select_goal: is_disabled &
       get_vertex_to_go_be_repaired_appointment(D, Path)
119         <-
120             .print("I have an appointment with some repairer.
                   I'm going to ", D, " using path: ", Path);
121             !init_goal(gotoPath(Path)).
122 +!select_goal: is_disabled &
       get_vertex_to_go_be_repaired_appointment_self(D, Path)
123         <-
124             .print("I have an self appointment with some
                   repairer. I'm going to ", D, " using path: ",
                   Path);
125             !init_goal(gotoPath(Path)).
```

```
126 +!select_goal: is_disabled & get_vertex_to_go_repair(D, Path)
127         <-
128             .print("I'm forever alone. I'm going to ", D, "
                    using path: ", Path);
129             !init_goal(gotoPath(Path)).
130 +!select_goal: is_inspect_goal(Entity)  & position(MyV) & not
    insp(MyV)
131         <-
132         .send([inspector1, inspector2, inspector3, inspector4,
                 inspector5, inspector6], tell, insp(MyV));
133         .print("I'm going to inspect ", Entity);
134         !init_goal(inspect).
135
136 +!select_goal : is_good_map_conquered <-
137     .print("Good map conquered! Stopped!");
138     !init_goal(recharge).
139
140 /* Protect Island */
141 +!select_goal: is_keep_goal_island_enemy(Entity) &
    is_survey_goal <-
142     .print("I'm at the island and I'm going to stay here (
            survey) ", Entity);
143     !init_goal(survey).
144 +!select_goal: is_keep_goal_island_enemy(Entity) <-
145     .print("I'm at the island and I'm going to stay here ",
            Entity);
146     !init_goal(recharge).
147 +!select_goal: there_is_enemy_nearby_island_geral(Op) <-
148     .print("I'm at the island and I'm going to ", Op, " to
            find some enemy inside");
149     !init_goal(goto(Op)).
150 +!select_goal: get_vertex_to_go_attack_search_island_geral(D,
    Path) <-
151     .print("I'm at the island and I'm going to ", D, " using
            ", Path, " to find some enemy there");
152     !init_goal(gotoPath(Path)).
153 /* End Protect Island */
154
155 +!select_goal: is_inspect_goal(Op, Entity)
156         <-
157         .print("I'm going to ", Op, " to inspect ", Entity);
158         !init_goal(goto(Op)).
159
160 //Plan for the active inspecting strategy
161 +!select_goal:winner(W,Op) & position(MyV) & ia.shortestPath(
    MyV, Op, Path, _) <-
162 .print("I won the auction for ", Op, " going there using ",
```

```
    Path);
163 !init_goal(gotoPath(Path)).
164
165 +!select_goal: is_survey_goal & not is_leave_goal
166         <- !init_goal(survey).
167
168 +!select_goal : going_to_outside_goal(V) <-
169     .print("I'm inside a region. I can expand to ", V);
170     !init_goal(goto(V)).
171
172 +!select_goal : can_expand_to(V) <-
173     .print("I can expand to ", V);
174     !init_goal(goto(V)).
175
176 +!select_goal: is_goal_keep_aim_vertex
177         <-
178             .print("I'm at a pivot vertex. Recharging...");
179             !init_goal(recharge).
180
181 +!select_goal: is_goal_aim_vertex(Op)
182         <-
183             .print("I have a pivot vertex to go. I'm going to
                    ", Op);
184             !init_goal(goto(Op)).
185 +!select_goal: is_goal_aim_vertex(D, Path)
186         <-
187             .print("I have a pivot vertex to go. I'm going to
                    ", D, " using path: ", Path);
188             !init_goal(gotoPath(Path)).
189 //Hills
190 +!select_goal : can_expand_to_hill(V) <-
191     .print("I'm at a hill and I can expand to ", V);
192     !init_goal(goto(V)).
193
194 +!select_goal : is_at_aim_position_hill <-
195     .print("Stop! Stand still! I'm settled at the hill!");
196     !init_goal(recharge).
197
198 +!select_goal: is_goal_hill_vertex(Op)
199         <-
200             .print("I have a hill vertex to go. I'm going to
                    ", Op);
201             !init_goal(goto(Op)).
202 +!select_goal: is_goal_hill_vertex(D, Path)
203         <-
204             .print("I have a hill vertex to go. I'm going to
                    ", D, " using path: ", Path);
```

```
205             !init_goal(gotoPath(Path)).
206
207 +!select_goal
208         <- !init_goal(random_walk).
209
210 +inspectedEntity(Entity, Team, Type, V, Energy, MaxEnergy,
        Health, MaxHealth, Strength, VisRange):
211     step(S) & not myTeam(Team) & artifact(inspectArtifact,
            IdInspectArtifact)
212 <-
213     addEntity(Entity, Type, MaxHealth, Strength, VisRange)[
            artifact_id(IdInspectArtifact)];
214     .print("The entity ", Type, " ", Entity, " was inspected
            at ", V, " Energy (", Energy, ",", MaxEnergy, ")
            Health (", Health, ",", MaxHealth, ") Strength ",
            Strength).
215
216 /*
217  * These functions must be dependent of each kind of agent
        because they will
218  * need to share some information with the other friends of
        the same kind
219  */
220 @do0[atomic]
221 +!do(Act):
222     step(S) & stepDone(S)
223 <-
224     .print("ERROR! I already performed an action for this step
            ! ", S).
225
226 @do1[atomic]
227 +!do(Act):
228     true
229 <-
230     !commitAction(Act).
231
232
233 +!initSpecific.
234
235 +!processBeforeStep(S).
236
237 +!processAfterStep(S): true <- //Clear all auction related
        beliefs before proceeding to next step
238 .abolish(auction(_)) ; .abolish(enemy(_,_)) ; .abolish(
        auction_done) ;
239 .abolish(place_bid(_)) ; .abolish(winner(_,_)) ; .abolish(
        auctions) ; .abolish(insp(_)).
```

B.3 Explorer

The changed code for the explorer agents. New code - the Java files and Jason file on line numbers: 135-170, 242, 244 and 365-367.

```
1  { include("mod.common.asl") }
2
3
4  //Verify if the explorer is at an unprobed vertex, and he is
       the explorer with the highest priority when there are
       other explorers at the same vertex
5  is_probe_goal  :- not is_disabled & position(MyV) & not ia.
       probedVertex(MyV,_) &
6        myTeam(MyTeam) & myNameInContest(MyName) &
7        .my_name(MyAgentName) &
8        not (
9          visibleEntity(Entity, MyV, MyTeam, normal) &
10         friend(AgentName, Entity, explorer, _) &
11         Entity \== MyName &
12         priorityEntity(AgentName, MyAgentName)
13       ).
14 is_probe_goal(Op) :- not is_disabled &
       there_is_unprobed_vertex_next_to_mine(Op).
15
16 /*
17  * Probe inside hill just special explorers
18  */
19 there_is_unprobed_vertex_next_to_mine_special(Op) :- .my_name(
       MyName) & play(MyName,specialExplorer,"
       grSpecialExploration") &
20       not is_disabled & step(S) & position(MyV) & myTeam(
             MyTeam) & hill(Neighborhood) &
21      .setof(V, ia.edge(MyV,V,_) & .member(V, Neighborhood) &
22           not ia.probedVertex(V, _) &
23           not (visibleEntity(Entity, V, MyTeam, normal) &
24                friend(_, Entity, explorer, _)
25           ) & not nextStepExplorer(_, V, S), Options)
26      & .length(Options, TotalOptions) & TotalOptions > 0 &
27      .nth(math.random(TotalOptions), Options, Op).
28
29 get_path_to_unprobed_probe_special(D, Path) :-
30             .my_name(MyName) & play(MyName,specialExplorer,"
                   grSpecialExploration") &
31             not is_disabled &
32             position(MyV) &
33             hill(Neighborhood) &
34             .setof(Vertex,  .member(Vertex, Neighborhood) &
```

```
35              ia.probedVertex(Vertex,-1) & not nextStepExplorer(_
                    , Vertex, _)
36              , List) & not .empty(List) &
37    ia.shortestPathDijkstraCompleteTwo(MyV, List, D, Path,
          Lenght) &
38              Lenght > 2.
39   /*
40    * End special explorers
41    */
42
43
44
45  /*
46   * ########################################################
47   * Check if there is some unprobed vertex around mine
48   * ########################################################
49   */
50   //Test if the vertex is not probed and there is other
        explorer there and also no one of the other explorers
        will go there in this step
51   //First try the vertices at the hill
52  there_is_unprobed_vertex_next_to_mine(Op) :- step(S) &
       position(MyV) & maxWeight(INF) & myTeam(MyTeam) & hill(
       Neighborhood) &
53      .setof(V,
54           ia.edge(MyV,V,W) & W \== INF &
55           .member(V, Neighborhood) &
56           not ia.probedVertex(V, _) &
57           not (
58                visibleEntity(Entity, V, MyTeam, normal) &
59                friend(_, Entity, explorer, _)
60           ) &
61           not nextStepExplorer(_, V, S)
62           , Options
63      )
64      & .length(Options, TotalOptions) & TotalOptions > 0 &
65      .nth(math.random(TotalOptions), Options, Op).
66  there_is_unprobed_vertex_next_to_mine(Op) :- position(MyV) &
       infinite(INF) & myTeam(MyTeam) & hill(Neighborhood) &
67        .setof(V,
68             ia.edge(MyV,V,W) & W \== INF &
69             .member(V, Neighborhood) &
70             not ia.probedVertex(V, _) &
71             not (
72                  visibleEntity(Entity, V, MyTeam, normal) &
73                  friend(_, Entity, explorer, _)
74             ) &
```

```
75              not nextStepExplorer(_, V, S)
76              , Options
77          )
78          & .length(Options, TotalOptions) & TotalOptions > 0 &
79          .nth(math.random(TotalOptions), Options, Op).
80
81  //Test if the vertex is not probed and there is other explorer
        there and also no one of the other explorers will go
        there in this step
82  there_is_unprobed_vertex_next_to_mine(Op) :- step(S) &
        position(MyV) & maxWeight(INF) & myTeam(MyTeam) &
83      .setof(V,
84              ia.edge(MyV,V,W) & W \== INF &
85              not ia.probedVertex(V, _) &
86              not (
87                  visibleEntity(Entity, V, MyTeam, normal) &
88                  friend(_, Entity, explorer, _)
89              ) &
90              not nextStepExplorer(_, V, S)
91              , Options
92      )
93      & .length(Options, TotalOptions) & TotalOptions > 0 &
94      .nth(math.random(TotalOptions), Options, Op).
95  there_is_unprobed_vertex_next_to_mine(Op) :- position(MyV) &
        infinite(INF) & myTeam(MyTeam) &
96      .setof(V,
97              ia.edge(MyV,V,W) & W \== INF &
98              not ia.probedVertex(V, _) &
99              not (
100                 visibleEntity(Entity, V, MyTeam, normal) &
101                 friend(_, Entity, explorer, _)
102             ) &
103             not nextStepExplorer(_, V, S)
104             , Options
105     )
106     & .length(Options, TotalOptions) & TotalOptions > 0 &
107     .nth(math.random(TotalOptions), Options, Op).
108
109 //Probe hills first
110 get_path_to_unprobed_probe(D, Path) :-
111       not is_disabled &
112       position(MyV) &
113       hill(Neighborhood) &
114       .setof(Vertex,
115               .member(Vertex, Neighborhood) &
116               ia.probedVertex(Vertex,-1) &
117               not nextStepExplorer(_, Vertex, _)
```

```
118       , List) &
119       not .empty(List) &
120      ia.shortestPathDijkstraCompleteTwo(MyV, List, D, Path,
             Lenght) &
121      Lenght > 2.
122
123 //Try to probe some far vertex, it is better
124 get_path_to_unprobed_probe(D, Path) :-
125       not is_disabled &
126       position(MyV) &
127       .setof(Vertex,
128                ia.probedVertex(Vertex,-1) &
129                not nextStepExplorer(_, Vertex, _)
130       , List) &
131       not .empty(List) &
132         ia.shortestPathDijkstraCompleteTwo(MyV, List, D, Path,
              Lenght) &
133       Lenght > 2.
134
135 //List nearby vertices up to depth 2 from position V
136 vertices_up_to_depth2(V, List) :- .setof(D1Vertex, ia.edge(V,
      D1Vertex,_), D1Nearby)
137 & .setof(D2Vertex,
138    (.member(X, D1Nearby) & ia.edge(X,D2Vertex,_) & D2Vertex \==
          V
139      & not .member(D2Vertex, D1Nearby)), D2Nearby)
140 & .setof(X, (.member(X, D1Nearby) | .member(X, D2Nearby)),
      List).
141
142 //List nearby vertices up to depth 3 from position V
143 vertices_up_to_depth3(V, List) :- .setof(D1Vertex, ia.edge(V,
      D1Vertex,_), D1Nearby)
144 & .setof(D2Vertex,
145    (.member(X, D1Nearby) & ia.edge(X,D2Vertex,_) & D2Vertex \==
          V
146      & not .member(D2Vertex, D1Nearby)), D2Nearby)
147 & .setof(D3Vertex, (.member(X, D2Nearby) & ia.edge(X,D3Vertex,
      _)
148    & D3Vertex \== V & not (.member(D3Vertex, D2Nearby)
149       | .member(D3Vertex, D1Nearby))), D3Nearby)
150 & .setof(X, (.member(X, D1Nearby) | .member(X, D2Nearby) | .
      member(X, D3Nearby)), List).
151
152 //Check if there are any friendly explorers on one of the
      vertices in the list (true if none are present)
153 check_for_other_explorers(Vertices) :-
154    .setof(X, (.member(X, Vertices)
```

```
155   & there_are_friends_explorers_at(X)), List) & .length(List,
          L) & L == 0.
156
157 //Check if there is a friendly explorer at the given vertex
158 there_are_friends_explorers_at(V) :-
159       .my_name(MyAgentName) & friend(AgentName, _, explorer, Id)
160        & MyAgentName \== AgentName & ia.getAgentPosition(Id, V).
161
162 //I can go to a vertex if I am a special explorer, or no other
        friendly explorers are nearby, or early phase has been
        concluded
163 //Call to ia.shortestPathBFSTwo can be replaced with call to
        one of the vertices_up_to_depth rules
164 can_go_to(V) :- position(MyV) & ia.edge(MyV, V, _) &
165 ((.my_name(Name) & play(Name, specialExplorer,_)) |
166 (ia.NewBFSAlgorithm(MyV,V,2,List) & check_for_other_explorers(
        List)) |
167 not early_phase).
168
169 //Early phase of simulation definition
170 early_phase :- step(S) & S < 50 & ia.thereIsUnprobedVertex.
171
172 +!wait_and_select_goal:
173       not numberWaits(_)
174 <-
175       +numberWaits(0);
176       !wait_and_select_goal.
177
178 +!wait_and_select_goal:
179       (numberWaits(K) & K >= 15) | step(0)
180 <-
181       .print("I can't wait anymore!");
182       -+numberWaits(0);
183       !select_goal.
184
185 +!wait_and_select_goal:
186       .my_name(MyName) & number_agents_higher_priority_same_type
            (MyName, NumberRequired) &
187       step(S) & .count(nextStepExplorer(_, _, S), Number) &
188       Number < NumberRequired &
189       numberWaits(K)
190 <-
191       .wait(50);
192       -+numberWaits(K+1);
193       !wait_and_select_goal.
194
195 +!wait_and_select_goal <- !select_goal.
```

```
196
197 +!select_goal: is_energy_goal
198         <- !init_goal(recharge).
199 +!select_goal : is_wait_repair_goal(V)
200         <- .print("Waiting to be repaired. Recharging...");
201             !init_goal(recharge).
202 +!select_goal : not canCome(ComeT) &
        is_wait_repair_goal_nearby(V, Repairer)
203         <- .print("Waiting to be repaired (nearby). ", V , " "
                , Repairer, ". Recharging...");
204             !init_goal(recharge).
205 +!select_goal : is_goto_repair_goal_nearby(V)
206         <- .print("Goto to be repaired (nearby)...");
207             !init_goal(goto(V)).
208
209 +!select_goal: is_disabled &
        get_vertex_to_go_be_repaired_appointment(D, Path)
210         <-
211             .print("I have an appointment with some repairer.
                    I'm going to ", D, " using path: ", Path);
212             !init_goal(gotoPath(Path)).
213 +!select_goal: is_disabled &
        get_vertex_to_go_be_repaired_appointment_self(D, Path)
214         <-
215             .print("I have an self appointment with some
                    repairer. I'm going to ", D, " using path: ",
                    Path);
216             !init_goal(gotoPath(Path)).
217 +!select_goal: is_disabled & get_vertex_to_go_repair(D, Path)
218         <-
219             .print("I'm forever alone. I'm going to ", D, "
                    using path: ", Path);
220             !init_goal(gotoPath(Path)).
221
222 +!select_goal: is_probe_goal & not is_leave_goal
223         <- !init_goal(probe).
224
225 /*
226 +!select_goal: is_survey_goal & not is_probe_goal & step(S) &
        S < 50
227         <- !init_goal(survey).
228 */
229 +!select_goal : is_good_map_conquered <-
230     .print("Good map conquered! Stopped!");
231     !init_goal(recharge).
232
233 /* Special explorers */
```

```
234 +!select_goal: there_is_unprobed_vertex_next_to_mine_special(
        Op) <-
235     .print("I'm a special explorer and I'm going MyV, V, _to
            probe ", Op);
236     !init_goal(goto(Op)).
237 +!select_goal: get_path_to_unprobed_probe_special(D, Path) <-
238     .print("I'm a special explorer and I'm going to probe ", D
            , " using ", Path);
239     !init_goal(gotoPath(Path)).
240 /* End special explorers */
241
242 +!select_goal: is_probe_goal(Op) & can_go_to(V)
243         <- .print("Going to ", Op, " to probe it") ; !
                init_goal(goto(Op)).
244 +!select_goal: get_path_to_unprobed_probe(D, Path) & .nth(1,
        Path, V) & can_go_to(V)
245         <- .print("Going to ", D, " using ", Path) ; !
                init_goal(gotoPath(Path)).
246
247 +!select_goal: is_probe_goal //Probe even if I need to leave
248         <- !init_goal(probe).
249
250 /* Protect Island */
251 +!select_goal: is_keep_goal_island_enemy(Entity) &
        is_survey_goal <-
252     .print("I'm at the island and I'm going to stay here (
            survey) ", Entity);
253     !init_goal(survey).
254 +!select_goal: is_keep_goal_island_enemy(Entity) <-
255     .print("I'm at the island and I'm going to stay here ",
            Entity);
256     !init_goal(recharge).
257 +!select_goal: there_is_enemy_nearby_island_geral(Op) <-
258     .print("I'm at the island and I'm going to ", Op, " to
            find some enemy inside");
259     !init_goal(goto(Op)).
260 +!select_goal: get_vertex_to_go_attack_search_island_geral(D,
        Path) <-
261     .print("I'm at the island and I'm going to ", D, " using "
            , Path, " to find some enemy there");
262     !init_goal(gotoPath(Path)).
263 /* End Protect Island */
264
265 +!select_goal : going_to_outside_goal(V) <-
266     .print("I'm inside a region. I can expand to ", V);
267     !init_goal(goto(V)).
268
```

```
269 +!select_goal : can_expand_to(V) <-
270     .print("I can expand to ", V);
271     !init_goal(goto(V)).
272
273 +!select_goal: is_goal_keep_aim_vertex
274         <-
275             .print("I'm at a pivot vertex. Recharging...");
276             !init_goal(recharge).
277
278 +!select_goal: is_goal_aim_vertex(Op)
279         <-
280             .print("I have a pivot vertex to go. I'm going to
                ", Op);
281             !init_goal(goto(Op)).
282 +!select_goal: is_goal_aim_vertex(D, Path)
283         <-
284             .print("I have a pivot vertex to go. I'm going to
                ", D, " using path: ", Path);
285             !init_goal(gotoPath(Path)).
286
287 //Hills
288 +!select_goal : can_expand_to_hill(V) <-
289     .print("I'm at a hill and I can expand to ", V);
290     !init_goal(goto(V)).
291
292 +!select_goal : is_at_aim_position_hill <-
293     .print("Stop! Stand still! I'm settled at the hill!");
294     !init_goal(recharge).
295
296 +!select_goal: is_goal_hill_vertex(Op)
297         <-
298             .print("I have a hill vertex to go. I'm going to "
                , Op);
299             !init_goal(goto(Op)).
300 +!select_goal: is_goal_hill_vertex(D, Path)
301         <-
302             .print("I have a hill vertex to go. I'm going to "
                , D, " using path: ", Path);
303             !init_goal(gotoPath(Path)).
304
305 +!select_goal
306         <- !init_goal(random_walk).
307
308 /*
309  * Percept a new probed vertex and share with friends
310  */
311 +probedVertex(V,Value) [source(percept)]:
```

```
312     true
313 <-
314     .print("Vertex probed: ", V, " with value ", Value);
315     ia.setVertexValue(V, Value);
316     !broadcastProbe(V,Value).
317 +probedVertex(V,Value) [source(self)]: true <- .abolish(
        probedVertex(V,Value)).
318
319 +!broadcastProbe(V,Value):
320     .findall(Agent, friend(Agent, _, _, _), SetAgents)
321 <-
322     .print("Sending probed vertex in broadcast ", V, " ",
            Value);
323     .send(SetAgents, tell, probedVertex(V,Value)).
324 +!broadcastProbe(V,Value).
325
326 /*
327  * These functions must be dependent of each kind of agent
        because they will
328  * need to share some information with the other friends of
        the same kind
329  */
330 +!do(Act):
331     step(S) & stepDone(S)
332 <-
333     .print("ERROR! I already performed an action for this step
            ! ", S).
334
335 +!do(Act):
336     step(S) & .my_name(MyName) &
            get_agents_lower_priority_same_type(MyName, SetAgents)
337     & is_disabled
338 <-
339     .print("Sending action to ", S, " ", SetAgents);
340     .send(SetAgents,tell,nextStepExplorer(MyName, none, S));
341     !commitAction(Act);
342     !!clearNextStepExplorer.
343
344 +!do(goto(V)):
345     step(S) & .my_name(MyName) &
            get_agents_lower_priority_same_type(MyName, SetAgents)
346 <-
347     .print("Sending action to ", S, " ", SetAgents);
348     .send(SetAgents,tell,nextStepExplorer(MyName, V, S));
349     !commitAction(goto(V));
350     !!clearNextStepExplorer.
351
```

```
352 +!do(Act):
353     step(S) & position(V) & .my_name(MyName) &
            get_agents_lower_priority_same_type(MyName, SetAgents)
354 <-
355     .print("Sending action to ", S, " ", SetAgents);
356     .send(SetAgents,tell,nextStepExplorer(MyName, V, S));
357     !commitAction(Act);
358     !!clearNextStepExplorer.
359
360 +!clearNextStepExplorer <-
361     .abolish(nextStepExplorer(_, _, _)).
362
363 +!initSpecific.
364
365 +!processBeforeStep(S) : S > 50 & unknown_enemy_visible(Op,
        Entity) <-
366 .print("Told lead inspector about entity ", Entity, " at ", Op
        , ".") ;
367 .send(inspector6, tell, enemy(Op, Entity)).
368
369 +!processBeforeStep(S).
370 +!processAfterStep(S) <-
371     !!calculateTotalSumVertices;
372     !buildAreas(S).
373 +!auction.
374 /*
375  * CALCULATE THE BEST PLACES
376  */
377 +!buildAreas(S):
378     goalState(_,concludeFirstPhase,_,_,enabled) & obligation(
            MyName,_Norm,achieved(_Scheme,concludeFirstPhase,_),_)
             &
379     (not lastCalcCoverage(_) & S >= 7 | lastCalcCoverage(N) &
            S - N >= 13) & .my_name(MyName)
380 <-
381     !determineHills;
382     -+lastCalcCoverage(S).
383 +!buildAreas(S).
384
385 /* Update the sum of all vertices */
386 +!calculateTotalSumVertices:
387     .my_name(MyName) & play(MyName,explorerLeader,"grMain") &
            ia.sumVertices(Total) &
388     .findall(Agent, friend(Agent, _, _, _), SetAgents)
389 <-
390     .print("Calculating the sum of all vertices: ", Total);
391     !updateTotalSumVertices(Total);
```

```
392         .send(SetAgents, achieve, updateTotalSumVertices(Total)).
393 +!calculateTotalSumVertices.
```

```
1  package ia;
2
3  import java.util.LinkedList;
4  import java.util.List;
5
6  import env.MixedAgentArch;
7  import graphLib.Graph;
8  import jason.asSemantics.*;
9  import jason.asSyntax.*;
10
11 public class newBFSAlgorithm extends DefaultInternalAction {
12
13     @Override
14     public Object execute(TransitionSystem ts, Unifier un,
           Term[] terms) throws Exception {
15         MixedAgentArch arch = (MixedAgentArch)ts.getUserAgArch
               ();
16         Graph graph = arch.getGraph();
17
18         String vertexS = ((Atom) terms[0]).getFunctor();
19         int maxDist = (int) ((NumberTerm) terms[2]).solve();
20         VarTerm bfsList = ((VarTerm) terms[3]);
21
22         String vertexD = ((Atom) terms[1]).getFunctor();
23
24         List<String> bfs = graph.getNewBFS(vertexS, vertexD,
               maxDist);
25
26         if (bfs != null && bfs.size() > 0) {
27             ListTerm list = new ListTermImpl();
28             ListTerm tail = list;
29             for (String s : bfs) {
30                 tail = tail.append(new Atom(s));
31             }
32
33             un.bind(bfsList, list);
34
35             return true;
36         } else {
37             return false;
38         }
39     }
40 }
```

```
1  package graphLib;
2
3  import java.util.LinkedList;
4  import java.util.List;
5
6  public class newBFSAlgorithm {
7    private int pred[] = new int[Graph.MAXVERTICES];
8    private int dist[] = new int[Graph.MAXVERTICES];
9    private int queue[] = new int[Graph.MAXVERTICES];
10
11   public List<Integer> execute(Graph g, int s, int maxDist) {
12     List<Integer> path = new LinkedList<Integer>();
13     for (int i = 0; i <= g.getSize(); i++) {
14       pred[i] = Graph.NULL;
15       dist[i] = Graph.INF;
16     }
17     dist[s] = 0;
18     pred[s] = Graph.NULL;
19     int fbegin = 0;
20     int fend = 0;
21     int u;
22     int v;
23     int i;
24     queue[fend++] = s;
25     path.add(s);
26     while (fbegin < fend) {
27       u = queue[fbegin++];
28       if(!(dist[u] >= maxDist))
29       {
30         for (i = 0; i < g.grade[u]; i++) {
31           v = g.adj[u][i];
32           if (dist[v] == Graph.INF) {
33             dist[v] = dist[u] + 1;
34             pred[v] = u;
35             queue[fend++] = v;
36             path.add(v);
37           }
38         }
39       }
40     }
41
42     return path;
43   }
44 }
```

B.4 Other

A few more minor (albeit important) changes to a couple files have been made; instead of listing the whole files again, the code below is shown with explanations as to where each code snippet has been added.

```
1  //Added to code for each agent, other than inspectors; can be
       seen in sentinel code in appendix B1 at lines 133-135 and
       140
2  +!processBeforeStep(S) : S > 50 & unknown_enemy_visible(Op,
       Entity)
3  <- .print("Told lead inspector about entity ", Entity, " at ",
        Op, ".");
4  .send(inspector6, tell, enemy(Op, Entity)).
5  +!auction.
6
7  //Added as a separate rule in the common rules file (appendix
       A3). Used in the situations in which an agent sees an
       uninspected enemy agent nearby
8  unknown_enemy_visible(Op, Entity) :- myTeam(MyTeam) &
       visibleEntity(Entity, Op, Team, _)
9            & Team \== MyTeam & not entityType(Entity, _, _, _,
                 _).
10
11 //Added in new step file (appendix A4), line 22. Adds the goal
        of completing an auction; all agents simply succeed this
       goal, while inspectors engage in auctions when appropriate
12 !auction.
13
14 //Added as a method in the Graph library (appendix A6)
15 public List<String> getNewBFS(String vertexS, String vertexD,
       int maxDist) {
16     LinkedList<String> result = null;
17     newBFSAlgorithm bfs = new newBFSAlgorithm();
18     List<Integer> listD = new LinkedList<Integer>();
19     int s = vertex2Integer(vertexS);
20     int d = vertex2Integer(vertexD);
21     List<Integer> resultBFS = bfs.execute(this, d, maxDist);
22     result = new LinkedList<String>();
23     if(resultBFS != null) {
24         for (int i : resultBFS) {
25             result.addFirst(integer2vertex[i]);
26        }
27     }
28     return result;
29 }
```

Appendix C

Test results

In this appendix, all the test results are attached. The first three sections contain tests where each strategy is tested separately, while the last section shows results where all strategies are used.

C.1 Sentinel

The results of simulations with original system against the system with the sentinel surveying strategy implemented. The results are shown for 5 configurations, where the strategy is employed until a given turn into the simulation. Each of the 5 configurations was subjected to 500 simulation runs against the original system.

configuration	survey320	survey640
sentinel25	21.516	180.9
sentinel50	20.136	157.3
sentinel75	20.476	146.5
sentinel100	20.452	142.1
sentinel133	20.476	141.5

Table C.1: Average turns for reaching certain achievements.

configuration	survey640 number	survey640 faster	survey320 faster
sentinel25	252	154	323
sentinel50	270	202	312
sentinel75	277	201	309
sentinel100	279	212	287
sentinel133	302	229	321

Table C.2: Number of simulations in which survey640 achievement was reached, how many were reached faster and number of simulations where it and survey320 were reached faster than the opposing (original system) team.

configuration	wins	av. score	av. score difference
sentinel25	240	119199.55	+540
sentinel50	265	119248.21	+589
sentinel75	228	118475.092	-184
sentinel100	256	119123.34	+470
sentinel133	268	118948.49	+290

Table C.3: Number of won simulations, average scores and differences between average scores of the changed and original team.

configuration	survey320	survey640
sentinel25	-4.319	-33.355
sentinel50	-5.699	-56.955
sentinel75	-5.359	-67.755
sentinel100	-5.383	-72.155
sentinel133	-5.359	-72.755

Table C.4: Differences between average turns of reaching survey achievements for the changed and original system.

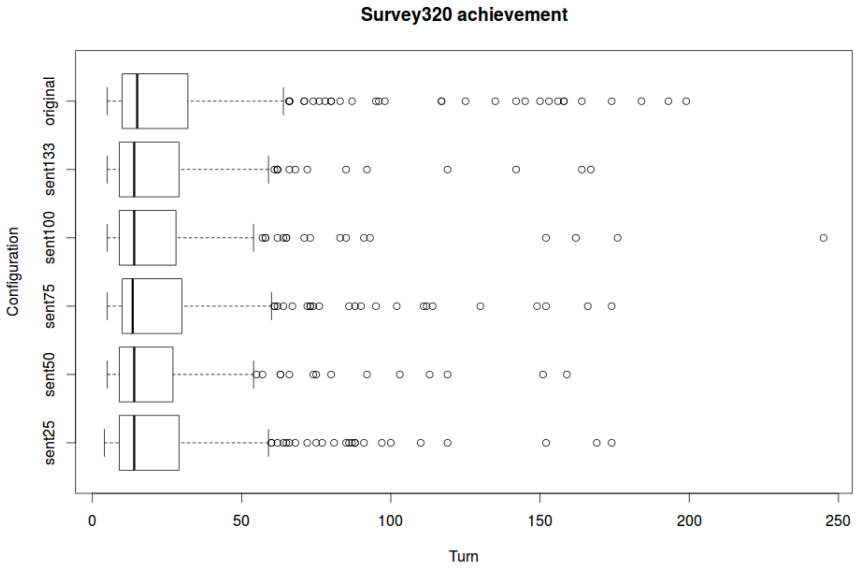

Figure C.1: Reaching the survey320 achievement.

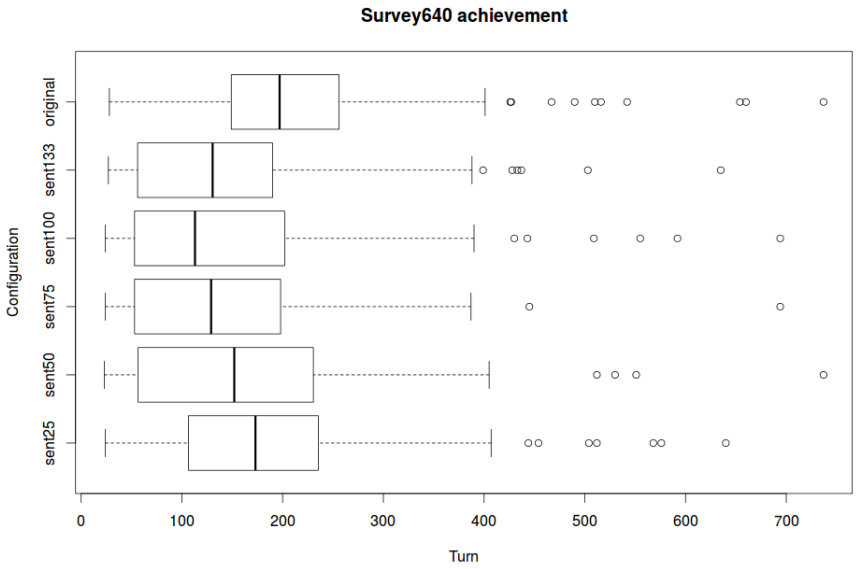

Figure C.2: Reaching the survey640 achievement.

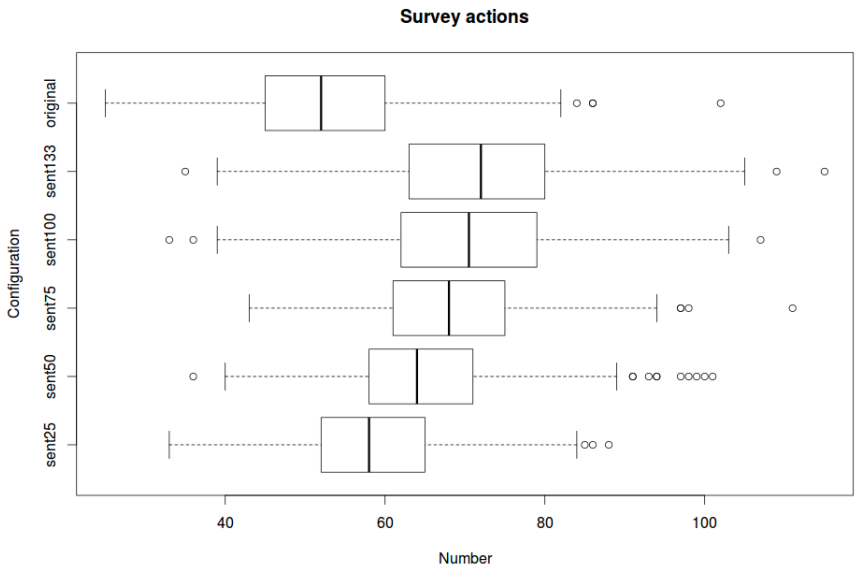

Figure C.3: The number of survey actions.

C.2 Inspector

The results of simulations with original system against the system with the inspecting strategies implemented. The results are shown for 4 configurations, where the strategy is employed starting from a given turn into the simulation. Each of the 4 configurations was subjected to 500 simulation runs against the original system.

configuration	inspect10	inspect20	inspect28
inspector25	33.28	73.52	260.44
inspector50	33.0	78.8	275.75
inspector75	35.69	85.206	280.123
inspector100	35.542	89.08	284.5

Table C.5: Average turns for reaching certain achievements and milestones.

configuration	inspected all	inspected all faster	inspected 20 faster
inspector25	407	306	318
inspector50	405	310	272
inspector75	405	297	256
inspector100	367	272	250

Table C.6: Number of simulations in which all 28 enemy agents were inspected and number of simulations in which all and 20 enemies were inspected faster than the opposing (original system) team.

configuration	wins	av. score	av. score difference
inspector25	243	119168.578	+510
inspector50	251	119263.696	+605
inspector75	252	119594.75	+935
inspector100	236	119103.286	+445

Table C.7: Number of won simulations, average scores and differences between average scores of the changed and original team.

configuration	inspect10	inspect20	inspect28
inspector25	-2.757	-16.42	-49.435
inspector50	-3.037	-11.14	-34.125
inspector75	-0.347	-4.734	-29.752
inspector100	-0.495	-0.86	-25.375

Table C.8: Differences between average turns for reaching inspect achievements and milestones for the changed and original system.

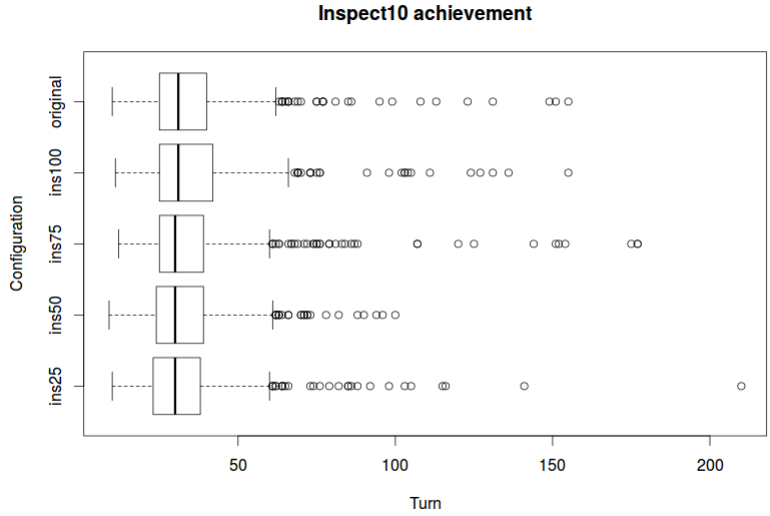

Figure C.4: Reaching the inspect10 achievement.

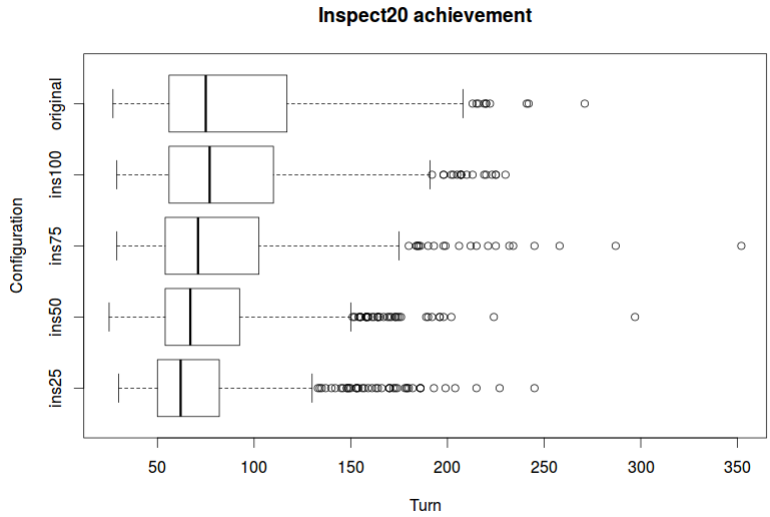

Figure C.5: Reaching the inspect20 achievement.

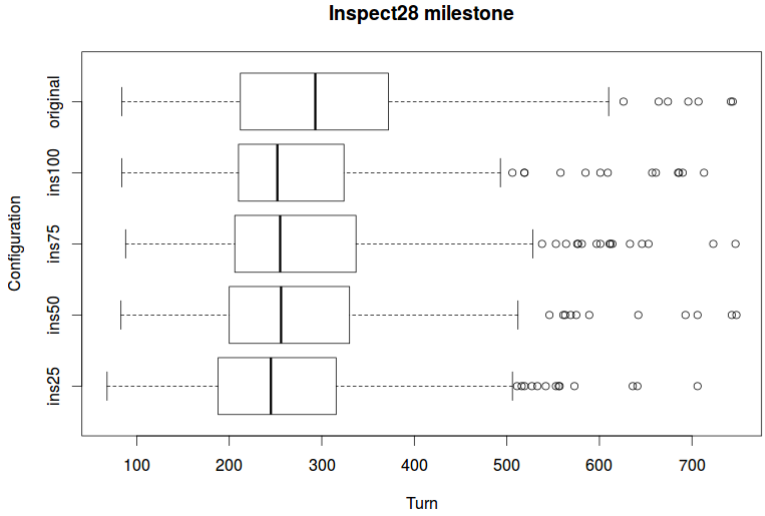

Figure C.6: Inspecting all 28 enemy agents.

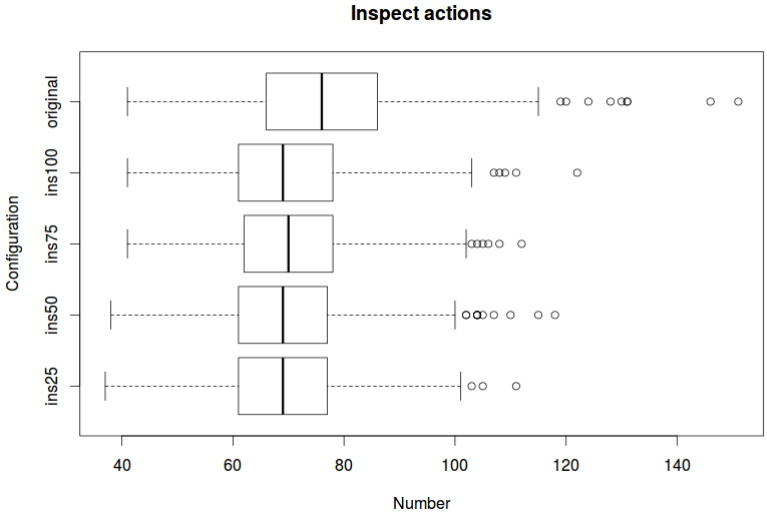

Figure C.7: The number of inspect actions.

C.3 Explorer

The results of simulations with original system against the system with the explorer spreading strategy implemented. The results are shown for 2 configurations, where the strategy is employed until a given turn into the simulation. Each of the 2 configurations was subjected to 500 simulation runs against the original system.

configuration	probe160	probe320	probeAll
explorer50	72.07	159.946	295.856
explorer133	72.97	163.20	297.19

Table C.9: Average turns for reaching certain achievements and milestones.

configuration	wins	av. score	av. score difference
explorer50	254	119787.1	+1128
explorer133	258	119362.7	+703.5

Table C.10: Number of won simulations, average scores and differences between average scores of the changed and original team.

configuration	probe160	probe320	probeAll
explorer50	+0.558	+0.761	+0.861
explorer133	+1.46	+3.415	+2.195

Table C.11: Differences between average turns of reaching probe achievements and milestones for the changed and original system.

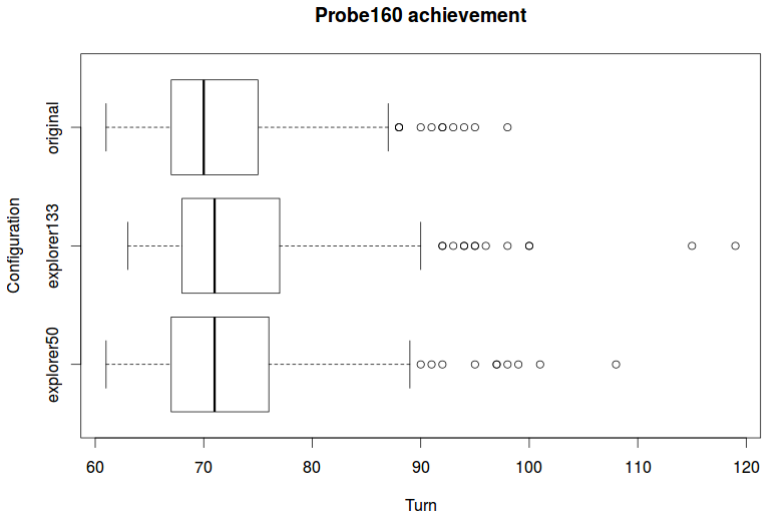

Figure C.8: Reaching the probe160 achievement.

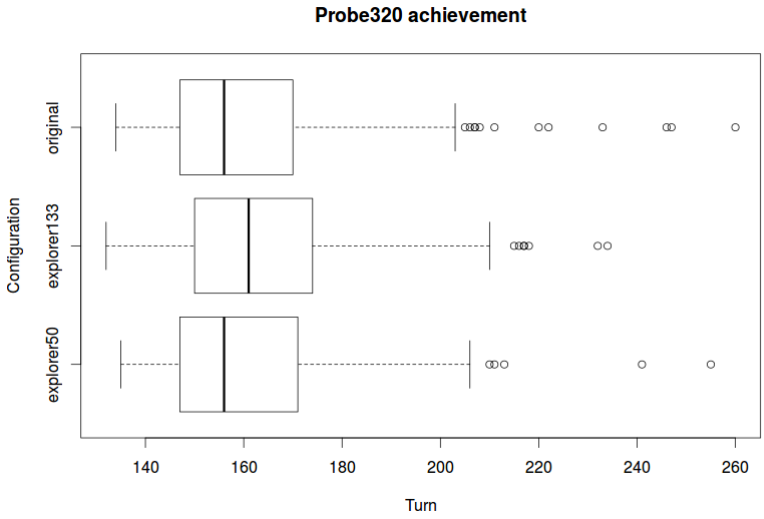

Figure C.9: Reaching the probe320 achievement.

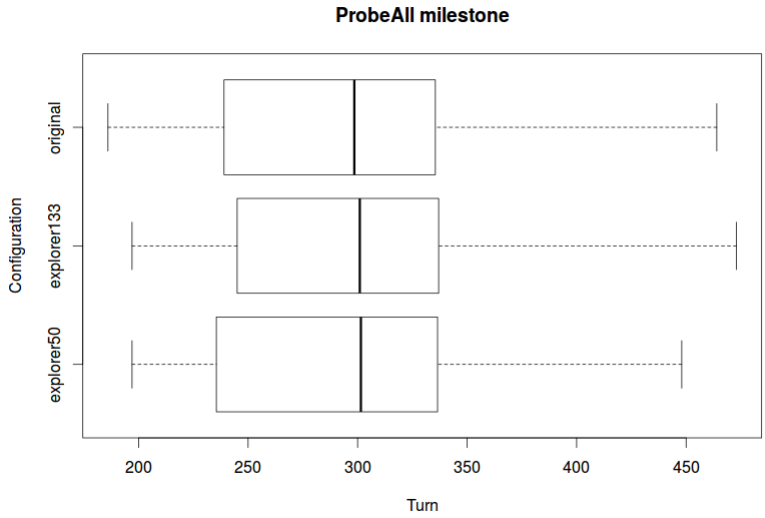

Figure C.10: Probing all vertices.

C.4 Buying strategies

The results of simulations for the two buying strategies - attack for saboteurs and health for explorers. Each one was subjected to 100 test runs - the results had already become abundantly clear by then and further testing was deemed irrelevant.

upgrade	wins	av. score	av. score difference
attack	15	102730.6	-15929
health	20	111170.2	-7489

Table C.12: Number of won simulations, average scores and differences between average scores of the changed and original team.

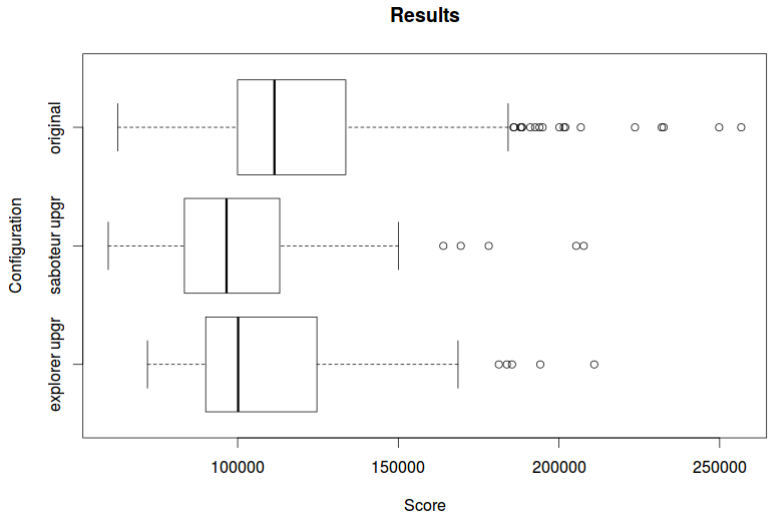

Figure C.11: The scores.

C.5 Final system

The results of simulations with all implemented strategies active (not the buying strategies, though - as they were unsuccessful). For sentinel, inspector and explorer strategies, where there are multiple possibilities (the step from/until which they are employed) the best performing ones were chosen - sentinel until the 133rd, inspector from the 50th and explorer until the 50th step of the simulation.

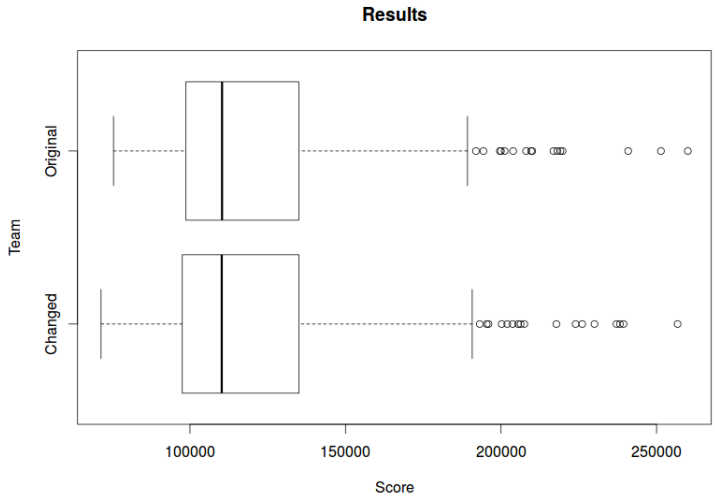

Figure C.12: The scores.

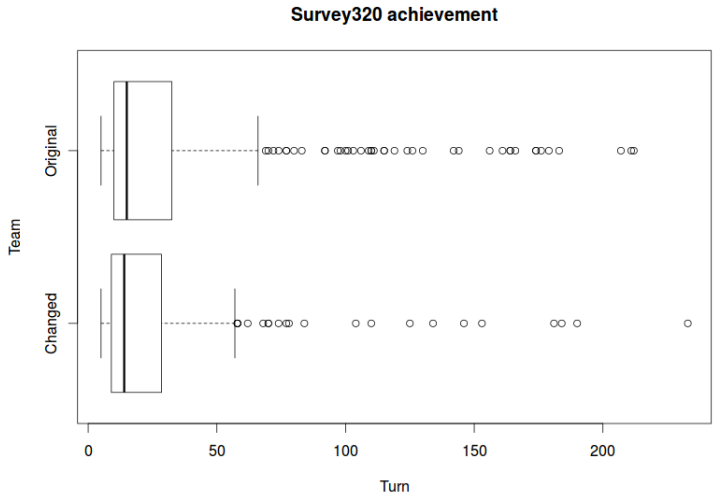

Figure C.13: Reaching the survey320 achievement.

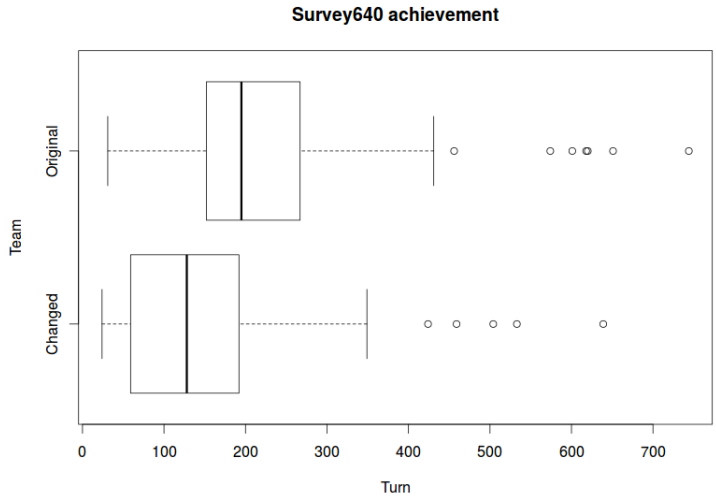

Figure C.14: Reaching the survey640 achievement.

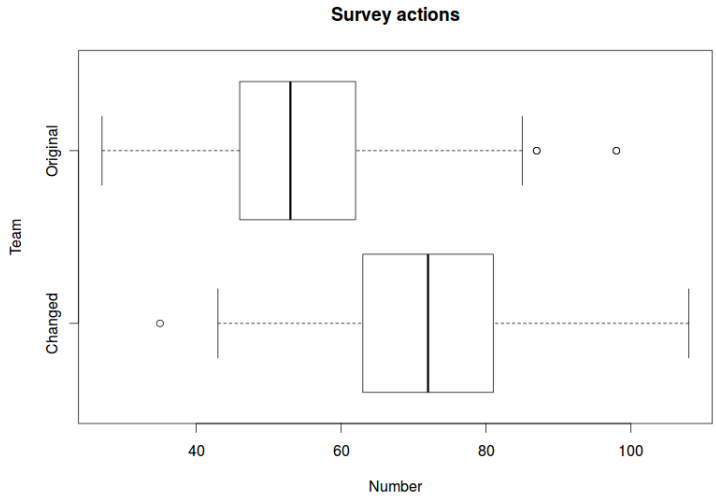

Figure C.15: The number of survey actions.

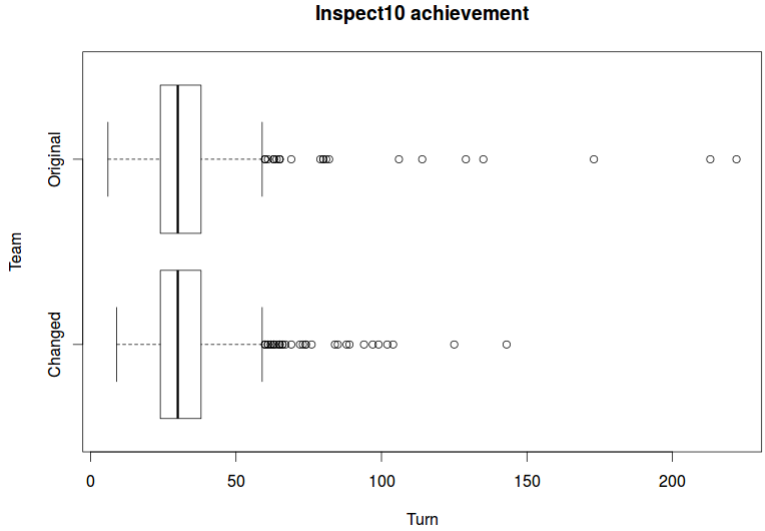

Figure C.16: Reaching the inspect10 achievement.

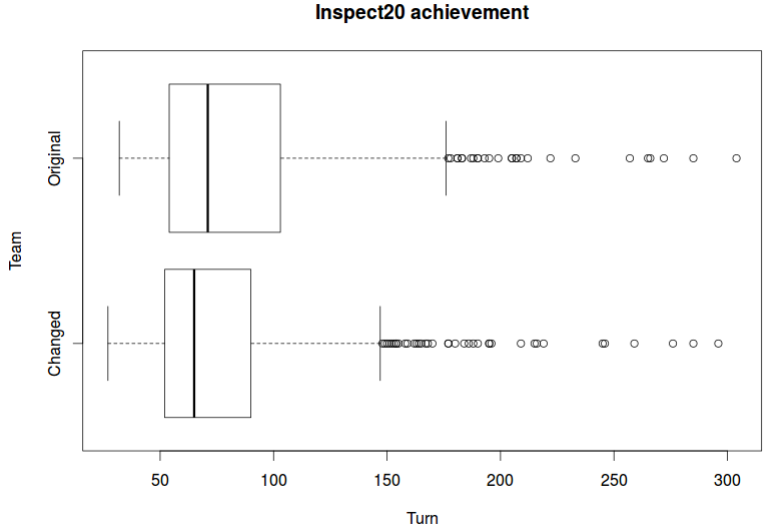

Figure C.17: Reaching the inspect20 achievement.

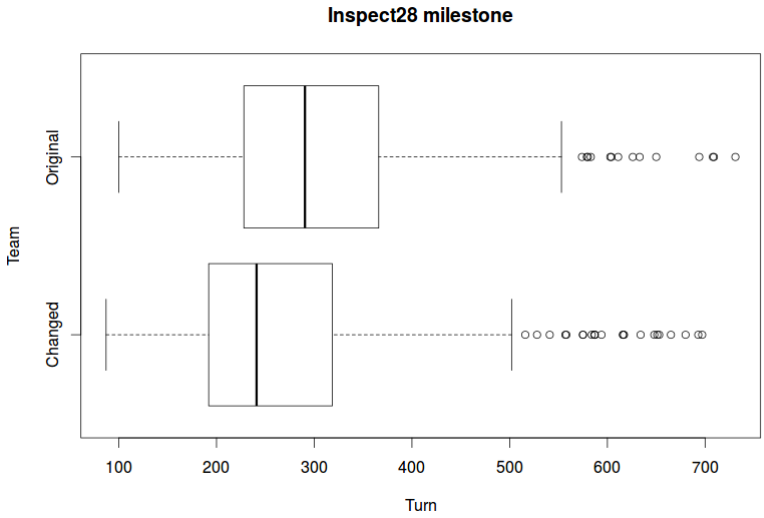

Figure C.18: Inspecting all 28 enemy agents.

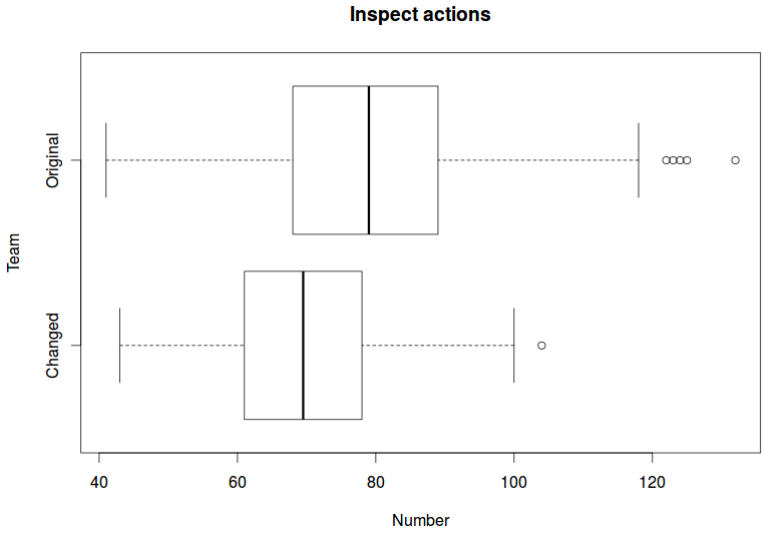

Figure C.19: The number of inspect actions.

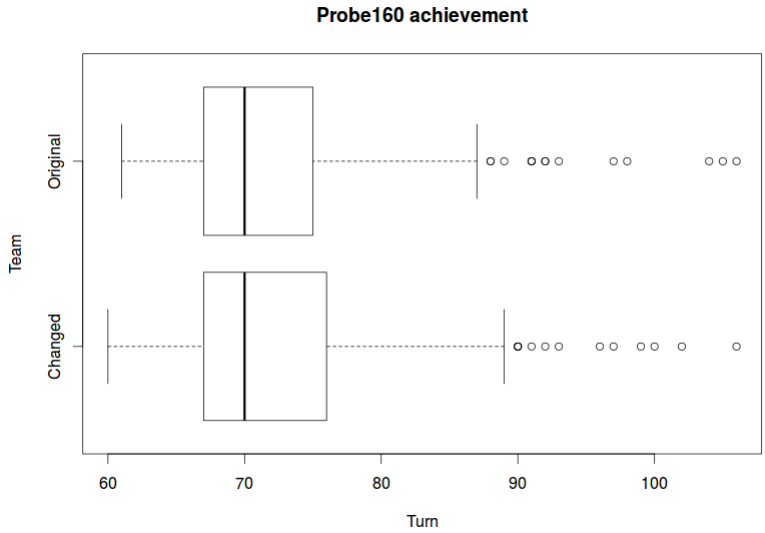

Figure C.20: Reaching the probe160 achievement.

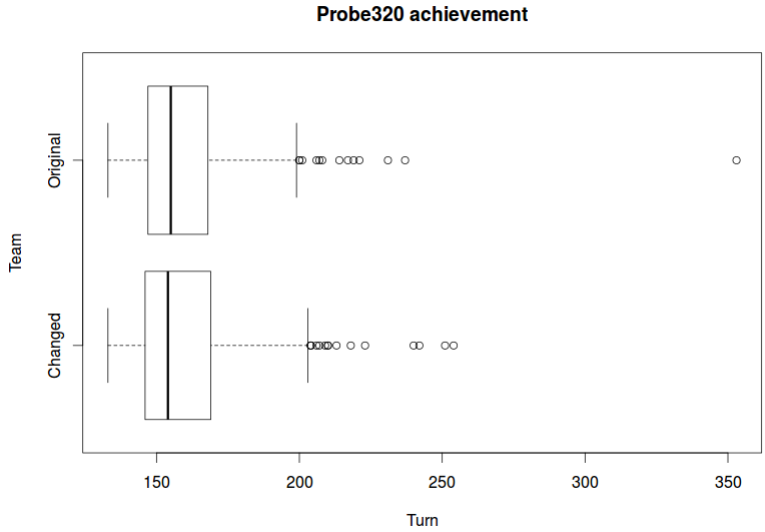

Figure C.21: Reaching the probe320 achievement.

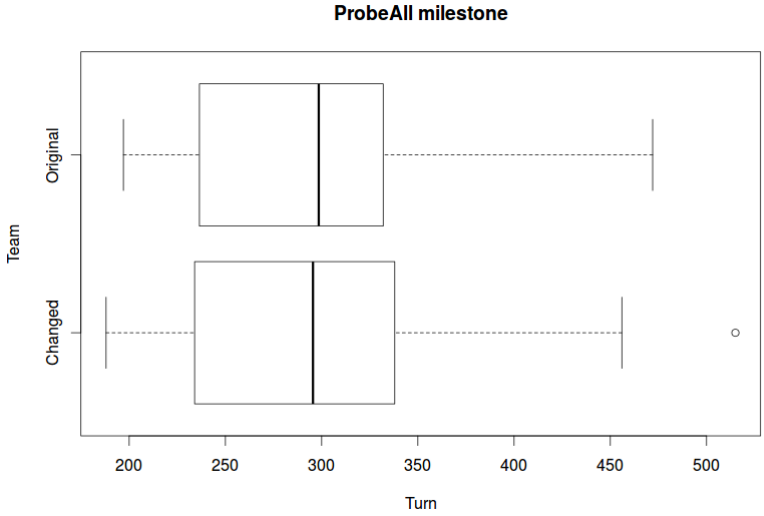

Figure C.22: Probing all vertices.

Appendix D

Other code files

This appendix lists other code files, which were used during the course of this project.

D.1 Parser for raw statistics text files

```
1  package testing;
2
3  import java.io.BufferedOutputStream;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.io.FileOutputStream;
7  import java.io.IOException;
8  import java.util.ArrayList;
9  import java.util.Map.Entry;
10 import java.util.Scanner;
11 import java.util.TreeMap;
12 import java.util.regex.Matcher;
13 import java.util.regex.Pattern;
14
15 //Used to extract relevant results from MAPC statistics files
16 class parser {
```

```
17   private final static String s1 = "surveyed320";
18   private final static String s2 = "surveyed640";
19   private final static String p1 = "proved160";
20   private final static String p2 = "proved320";
21   private final static String i1 = "inspected10";
22   private final static String i2 = "inspected20";
23   private final static Pattern matchTurnNr = Pattern.compile("
         ^[\\d]{1,3}");
24   private final static Pattern matchScore = Pattern.compile("^
         Score: [\\d]* Ranking: -1$");
25   private final static Pattern extractScore = Pattern.compile(
         "^[\\d]*");
26   private final static Pattern matchSentinels = Pattern.
         compile("^Statistics for Agent [a-b][1][7-9]|[2][0-2] of
          Team [A-B]:$");
27   private final static Pattern matchInspectors = Pattern.
         compile("^Statistics for Agent [a-b][2][3-8] of Team [A-
         B]:$");
28   private final static Pattern patternAsurvey320 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*"+s1);
29   private final static Pattern patternAsurvey640 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*"+s2);
30   private final static Pattern patternAprobe160 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*"+p1);
31   private final static Pattern patternAprobe320 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*"+p2);
32   private final static Pattern patternAinspect10 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*"+i1);
33   private final static Pattern patternAinspect20 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*"+i2);
34   private final static Pattern patternBsurvey320 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*[&]{1}[\\w\\s,]*"+s1
         );
35   private final static Pattern patternBsurvey640 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*[&]{1}[\\w\\s,]*"+s2
         );
36   private final static Pattern patternBprobe160 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*[&]{1}[\\w\\s,]*"+p1
         );
37   private final static Pattern patternBprobe320 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*[&]{1}[\\w\\s,]*"+p2
         );
38   private final static Pattern patternBinspect10 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*[&]{1}[\\w\\s,]*"+i1
         );
39   private final static Pattern patternBinspect20 = Pattern.
         compile("^[\\d]{1,3}[&]{1}[\\w\\s,]*[&]{1}[\\w\\s,]*"+i2
```

```
    );
40  private final static Pattern[] patternsA = {
        patternAsurvey320, patternAsurvey640, patternAprobe160,
        patternAprobe320, patternAinspect10, patternAinspect20};
41  private final static Pattern[] patternsB = {
        patternBsurvey320, patternBsurvey640, patternBprobe160,
        patternBprobe320, patternBinspect10, patternBinspect20};
42  private final static String achievementsTex = "
        achievementsTable.tex";
43  private final static String scoresTxt = "txtfile";
44  private final static String testName = "Inspector50";
45  private final static String directory = "/home/admin/
        workspace/Testing/"+testName+"/";
46  private static ArrayList<String> filesTex = new ArrayList
        <>();
47  private static ArrayList<String> filesTxt = new ArrayList
        <>();
48  private static int counterTex = 0;
49  private static int counterTxt = 0;
50  private static int counter = 0;
51  private static int lineInput = 0;
52  private static int lineOutput = 0;
53  private static int exceptions = 0;
54  private static TreeMap<String, Integer>[] mapsA = null;
55  private static TreeMap<String, Integer>[] mapsB = null;
56
57  public static void main(final String[] args) {
58    TreeMap<String, Integer> achievA = new TreeMap<>();
59    TreeMap<String, Integer> achievB = new TreeMap<>();
60    traverse(new File(directory));
61    assert (counterTxt == counterTex);
62    mapsA = new TreeMap[counterTex];
63    mapsB = new TreeMap[counterTex];
64    for (int i = 0; i < counterTex; i++) {
65      parseTxtFile(filesTxt.get(i), achievA, achievB);
66      parseTexFile(filesTex.get(i), achievA, achievB);
67    }
68    assert (exceptions == 0);
69    writeResultsToFiles();
70    System.out.println("Success.");
71    System.out.println("Parsed files: " + counter);
72    System.out.println("Input lines: " + lineInput);
73    System.out.println("Output lines: " + lineOutput);
74    System.out.println("Exceptions: " + exceptions);
75  }
76
```

```
77  //Go through the scores txt file and save relevant
        information
78  private static void parseTxtFile(String path, TreeMap<String
        , Integer> achievA, TreeMap<String, Integer> achievB) {
79    boolean found1 = false;
80    int sentinelsCounted = 0;
81    int inspectorsCounted = 0;
82    File file = new File(path);
83    Scanner reader = null;
84    String line = null;
85    String agent = null;
86    int score1 = 0;
87    int score2 = 0;
88    int surveys1 = 0;
89    int surveys2 = 0;
90    int inspects1 = 0;
91    int inspects2 = 0;
92    try {
93      reader = new Scanner(file);
94      while (reader.hasNext()) {
95        line = reader.nextLine();
96        lineInput++;
97        if (match(matchScore, line)) {
98          Matcher match = extractScore.matcher(line.substring
                (7, 13));
99          while (match.find()) {
100           if(!found1) {
101             score1 = Integer.parseInt(match.group());
102             found1 = true;
103           } else {
104             score2 = Integer.parseInt(match.group());
105             break;
106           }
107         }
108       }
109     }
110     if (Math.abs(score1 - score2) > 50000) {
111       System.err.println("Score difference more than 50.000
              - investigate: " + path);
112     }
113     achievA.put("Score: ", score1);
114     achievB.put("Score: ", score2);
115     reader.close();
116     reader = new Scanner(file);
117     while (reader.hasNext()) {
118       agent = reader.nextLine();
119       lineInput++;
```

```
120         if (match(matchSentinels, agent)) {
121           do {
122             line = reader.nextLine();
123             lineInput++;
124           } while (!line.startsWith("survey:"));
125           Matcher match = extractScore.matcher(line.substring
                (14, 17));
126           while (match.find()) {
127             if (sentinelsCounted < 6) {
128               sentinelsCounted++;
129               surveys1 = surveys1 + Integer.parseInt(match.
                    group());
130             } else {
131               sentinelsCounted++;
132               surveys2 = surveys2 + Integer.parseInt(match.
                    group());
133             }
134           }
135         }
136         if (match(matchInspectors, agent)) {
137           do {
138             line = reader.nextLine();
139             lineInput++;
140           } while (!line.startsWith("inspect:"));
141           Matcher match = extractScore.matcher(line.substring
                (15, 18));
142           while (match.find()) {
143             if (inspectorsCounted < 6) {
144               inspectorsCounted++;
145               inspects1 = inspects1 + Integer.parseInt(match.
                    group());
146             } else {
147               inspectorsCounted++;
148               inspects2 = inspects2 + Integer.parseInt(match.
                    group());
149             }
150           }
151         }
152       }
153       achievA.put("surveys", surveys1);
154       achievB.put("surveys", surveys2);
155       achievA.put("inspects", inspects1);
156       achievB.put("inspects", inspects2);
157     } catch (Exception e) {
158       exceptions++;
159       System.err.println(path);
160       System.err.println(agent);
```

```
161         System.err.println(line);
162         e.printStackTrace();
163       } finally {
164         reader.close();
165       }
166     }
167
168     //Go through the achievements tex file and save relevant
          information to maps
169     private static void parseTexFile(String path, TreeMap<String
          , Integer> achievA, TreeMap<String, Integer> achievB) {
170       File file = new File(path);
171       Scanner reader = null;
172       Matcher match = null;
173       try {
174         reader = new Scanner(file);
175         String line = null;
176         while (reader.hasNext()) {
177           line = reader.nextLine();
178           lineInput++;
179           for (int i = 0; i < patternsA.length; i++) {
180             if (match(patternsA[i], line)) {
181               match = matchTurnNr.matcher(line);
182               if (match.find()) {
183                 int turn = Integer.parseInt(match.group());
184                 switch (i) {
185                 case 0:
186                   achievA.put(s1, turn);
187                   break;
188                 case 1:
189                   achievA.put(s2, turn);
190                   break;
191                 case 2:
192                   achievA.put(p1, turn);
193                   break;
194                 case 3:
195                   achievA.put(p2, turn);
196                   break;
197                 case 4:
198                   achievA.put(i1, turn);
199                   break;
200                 case 5:
201                   achievA.put(i2, turn);
202                   break;
203                 default:
204                   System.err.println("Something went wrong in
                        switch A!");
```

```
205                 break;
206               }
207             }
208           }
209         }
210         for (int i = 0; i < patternsB.length; i++) {
211           if (match(patternsB[i], line)) {
212             match = matchTurnNr.matcher(line);
213             if (match.find()) {
214               int turn = Integer.parseInt(match.group());
215               switch (i) {
216               case 0:
217                 achievB.put(s1, turn);
218                 break;
219               case 1:
220                 achievB.put(s2, turn);
221                 break;
222               case 2:
223                 achievB.put(p1, turn);
224                 break;
225               case 3:
226                 achievB.put(p2, turn);
227                 break;
228               case 4:
229                 achievB.put(i1, turn);
230                 break;
231               case 5:
232                 achievB.put(i2, turn);
233                 break;
234               default:
235                 System.err.println("Something went wrong in
                      switch B!");
236                 break;
237               }
238             }
239           }
240         }
241       }
242       TreeMap<String, Integer> tempA = (TreeMap<String,
            Integer>) achievA.clone();
243       TreeMap<String, Integer> tempB = (TreeMap<String,
            Integer>) achievB.clone();
244       tempA.putAll(achievA);
245       tempB.putAll(achievB);
246       mapsA[counter] = tempA;
247       mapsB[counter] = tempB;
248       counter++;
```

```
249         achievA.clear();
250         achievB.clear();
251       } catch (Exception e) {
252         exceptions++;
253         e.printStackTrace();
254       } finally {
255         reader.close();
256       }
257     }
258
259     //Match a regex pattern to a line of text
260     private static boolean match(Pattern pattern, String line){
261       Matcher matcher = pattern.matcher(line);
262       boolean found = false;
263       while (matcher.find()) {
264         found = true;
265       }
266       return found;
267     }
268
269     //Traverse a directory and save paths to achievements and
          scores text files
270     private static void traverse(File file) {
271       if (file.isDirectory()) {
272         String entries[] = file.list() ;
273         if (entries != null) {
274           for (String entry : entries) {
275             traverse(new File(file,entry)) ;
276           }
277         }
278       }
279       if (file.getName().equals(achievementsTex)) {
280         filesTex.add(file.getAbsolutePath());
281         counterTex++;
282       }
283       if (file.getName().equals(scoresTxt)) {
284         filesTxt.add(file.getAbsolutePath());
285         counterTxt++;
286       }
287     }
288
289     //Go through the array with results and extract information
          about achievements to file
290     private static void writeResultsToFiles() {
291       File survey320A = new File(directory+"survey320A.txt");
292       File survey640A = new File(directory+"survey640A.txt");
293       File probe160A = new File(directory+"probe160A.txt");
```

```
294     File probe320A = new File(directory+"probe320A.txt");
295     File inspect10A = new File(directory+"inspect10A.txt");
296     File inspect20A = new File(directory+"inspect20A.txt");
297     File survey320B = new File(directory+"survey320B.txt");
298     File survey640B = new File(directory+"survey640B.txt");
299     File probe160B = new File(directory+"probe160B.txt");
300     File probe320B = new File(directory+"probe320B.txt");
301     File inspect10B = new File(directory+"inspect10B.txt");
302     File inspect20B = new File(directory+"inspect20B.txt");
303     File resultA = new File(directory+"resultA.txt");
304     File resultB = new File(directory+"resultB.txt");
305     File surveysA = new File(directory+"surveysA.txt");
306     File surveysB = new File(directory+"surveysB.txt");
307     File inspectsA = new File(directory+"inspectsA.txt");
308     File inspectsB = new File(directory+"inspectsB.txt");
309     for (int i = 0; i < mapsA.length; i++) {
310       for (Entry<String, Integer> entry : mapsA[i].entrySet())
              {
311         String key = entry.getKey();
312         Integer value = entry.getValue();
313         switch (key) {
314         case "inspected10":
315           writeToFile(inspect10A, value.toString(), true);
316           break;
317         case "inspected20":
318           writeToFile(inspect20A, value.toString(), true);
319           break;
320         case "proved160":
321           writeToFile(probe160A, value.toString(), true);
322           break;
323         case "proved320":
324           writeToFile(probe320A, value.toString(), true);
325           break;
326         case "Score: ":
327           writeToFile(resultA, value.toString(), true);
328           break;
329         case "surveyed320":
330           writeToFile(survey320A, value.toString(), true);
331           break;
332         case "surveyed640":
333           writeToFile(survey640A, value.toString(), true);
334           break;
335         case "surveys":
336           writeToFile(surveysA, value.toString(), true);
337           break;
338         case "inspects":
339           writeToFile(inspectsA, value.toString(), true);
```

```
340             break;
341           default:
342             System.err.println("Something went wrong in switch A
                    !");
343             break;
344           }
345       }
346     }
347     for (int i = 0; i < mapsB.length; i++) {
348       for (Entry<String, Integer> entry : mapsB[i].entrySet())
              {
349         String key = entry.getKey( );
350         Integer value = entry.getValue();
351         switch (key) {
352         case "inspected10":
353           writeToFile(inspect10B, value.toString(), true);
354           break;
355         case "inspected20":
356           writeToFile(inspect20B, value.toString(), true);
357           break;
358         case "proved160":
359           writeToFile(probe160B, value.toString(), true);
360           break;
361         case "proved320":
362           writeToFile(probe320B, value.toString(), true);
363           break;
364         case "Score: ":
365           writeToFile(resultB, value.toString(), true);
366           break;
367         case "surveyed320":
368           writeToFile(survey320B, value.toString(), true);
369           break;
370         case "surveyed640":
371           writeToFile(survey640B, value.toString(), true);
372           break;
373         case "surveys":
374           writeToFile(surveysB, value.toString(), true);
375           break;
376         case "inspects":
377           writeToFile(inspectsB, value.toString(), true);
378           break;
379         default:
380           System.err.println("Something went wrong in switch B
                    !");
381           break;
382         }
383       }
```

```
384     }
385   }
386
387   //Standard method for writing text to file
388   private static void writeToFile(File file, String text,
          boolean append){
389     BufferedOutputStream output = null;
390     try{
391       output = new BufferedOutputStream(new FileOutputStream(
              file, append));
392       lineOutput++;
393       output.write(text.getBytes());
394       output.write("\n".getBytes());
395     }catch(FileNotFoundException e){
396       exceptions++;
397       e.printStackTrace();
398     }catch(IOException e){
399       exceptions++;
400       e.printStackTrace();
401     }finally{
402       try{
403         if(output != null){
404           output.flush();
405           output.close();
406         }
407       }catch(IOException e){
408         exceptions++;
409         e.printStackTrace();
410       }
411     }
412   }
413 }
```

D.2 Parser for agent simulation logs

The program used to extract the turn in which all 28 enemy agents were successfully inspected.

```
1  package testing;
2
3  import java.io.BufferedOutputStream;
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.io.FileOutputStream;
7  import java.io.IOException;
8  import java.util.Scanner;
9  import java.util.regex.Matcher;
10 import java.util.regex.Pattern;
11
12 //Used to extract data from agent output log file
13 class parser2 {
14   private static int exceptions = 0;
15   public static void main(final String[] args) {
16     final Pattern patternStep = Pattern.compile("Current step
           is[\\s][\\d]");
17     final Pattern patternInspected28 = Pattern.compile("Number
           of entities in artefact:[\\s]28");
18     final Pattern p = Pattern.compile("\\d{2,3}");
19     File f = new File("outputA.log");
20     File f2 = new File("outputB.log");
21     File resA = new File("inspected28A.txt");
22     File resB = new File("inspected28B.txt");
23     Scanner s = null;
24     String line = null;
25     try {
26       s = new Scanner(f);
27       while (s.hasNextLine()) {
28         line = s.nextLine();
29         if (match(patternInspected28, line)) {
30           do{
31           line = s.nextLine();
32           }while (!match(patternStep, line));
33           Matcher m = p.matcher(line);
34           if (m.find()) {
35           writeToFile(resA, m.group(), true);
36           }
37         }
38       }
39     } catch (FileNotFoundException e) {
40       exceptions++;
```

```
41        e.printStackTrace();
42      } finally {
43        s.close();
44      }
45      try {
46        s = new Scanner(f2);
47        while (s.hasNextLine()) {
48          line = s.nextLine();
49          if (match(patternInspected28, line)) {
50            do{
51            line = s.nextLine();
52            }while (!match(patternStep, line));
53            Matcher m = p.matcher(line);
54            if (m.find()) {
55            writeToFile(resB, m.group(), true);
56            }
57          }
58        }
59      } catch (FileNotFoundException e) {
60        exceptions++;
61        e.printStackTrace();
62      } finally {
63        s.close();
64      }
65      assert (exceptions == 0);
66      System.out.println("Success.");
67    }
68
69    //Match a regex pattern to a line of text
70    private static boolean match(Pattern pattern, String line){
71      Matcher matcher = pattern.matcher(line);
72      boolean found = false;
73      while (matcher.find()) {
74        found = true;
75      }
76      return found;
77    }
78
79    //Standard method for writing text to file
80    private static void writeToFile(File file, String text,
          boolean append){
81      BufferedOutputStream output = null;
82      try{
83        output = new BufferedOutputStream(new FileOutputStream(
              file, append));
84        output.write(text.getBytes());
85        output.write("\n".getBytes());
```

```
86      }catch(FileNotFoundException e){
87        exceptions++;
88        e.printStackTrace();
89      }catch(IOException e){
90        exceptions++;
91        e.printStackTrace();
92      }finally{
93        try{
94          if(output != null){
95            output.flush();
96            output.close();
97          }
98        }catch(IOException e){
99          exceptions++;
100         e.printStackTrace();
101       }
102     }
103   }
104 }
```

D.3 R statistics script

The program used to read output text files from D.1 and D.2 and subsequently generate box plots and other statistics.

```
#Directory with text files
setwd('/home/admin/workspace/tests/') #Add name of
    configuration after /tests/; e.g. Inspector25

#Extract data from text files
ins10A=as.vector(as.matrix(read.table("inspect10A.txt")))
ins10B=as.vector(as.matrix(read.table("inspect10B.txt")))
ins20A=as.vector(as.matrix(read.table("inspect20A.txt")))
ins20B=as.vector(as.matrix(read.table("inspect20B.txt")))
ins28A=as.vector(as.matrix(read.table("inspected28A.txt")))
ins28B=as.vector(as.matrix(read.table("inspected28B.txt")))
probe160A=as.vector(as.matrix(read.table("probe160A.txt")))
probe160B=as.vector(as.matrix(read.table("probe160B.txt")))
probe320A=as.vector(as.matrix(read.table("probe320A.txt")))
probe320B=as.vector(as.matrix(read.table("probe320B.txt")))
survey320A=as.vector(as.matrix(read.table("survey320A.txt")))
survey320B=as.vector(as.matrix(read.table("survey320B.txt")))
survey640A=as.vector(as.matrix(read.table("survey640A.txt")))
survey640B=as.vector(as.matrix(read.table("survey640B.txt")))
surveysA=as.vector(as.matrix(read.table("surveysA.txt")))
surveysB=as.vector(as.matrix(read.table("surveysB.txt")))
inspectsA=as.vector(as.matrix(read.table("inspectsA.txt")))
inspectsB=as.vector(as.matrix(read.table("inspectsB.txt")))
resultA=as.vector(as.matrix(read.table("resultA.txt")))
resultB=as.vector(as.matrix(read.table("resultB.txt")))

#Extract number of simulations
simulationsNumber=length(resultA)

#Draw boxplots for each pair of data
boxplot(ins10A,ins10B,names=c("Changed","Original"),horizontal
    =TRUE,xlab="Turn",ylab="Team",main="Inspect10 achievement"
    )
boxplot(ins20A,ins20B,names=c("Changed","Original"),horizontal
    =TRUE,xlab="Turn",ylab="Team",main="Inspect20 achievement"
    )
boxplot(ins28A,ins28B,names=c("Changed","Original"),horizontal
    =TRUE,xlab="Turn",ylab="Team",main="Inspect28 milestone")
boxplot(probe160A,probe160B,names=c("Changed","Original"),
    horizontal=TRUE,xlab="Turn",ylab="Team",main="Probe160
    achievement")
boxplot(probe320A,probe320B,names=c("Changed","Original"),
```

```
    horizontal=TRUE, xlab="Turn", ylab="Team", main="Probe320
    achievement")
boxplot(survey320A, survey320B, names=c("Changed","Original"),
    horizontal=TRUE, xlab="Turn", ylab="Team", main="Survey320
    achievement")
boxplot(survey640A, survey640B, names=c("Changed","Original"),
    horizontal=TRUE, xlab="Turn", ylab="Team", main="Survey640
    achievement")
boxplot(surveysA, surveysB, names=c("Changed","Original"),
    horizontal=TRUE, xlab="Number", ylab="Team", main="Survey
    actions")
boxplot(inspectsA, inspectsB, names=c("Changed","Original"),
    horizontal=TRUE, xlab="Number", ylab="Team", main="Inspect
    actions")
boxplot(resultA, resultB, names=c("Changed","Original"),
    horizontal=TRUE, xlab="Score", ylab="Team", main="Results")

#Extract average values
averageIns10A=mean(ins10A)
averageIns10B=mean(ins10B)
averageIns20A=mean(ins20A)
averageIns20B=mean(ins20B)
averageIns28A=mean(ins28A)
averageIns28B=mean(ins28B)
averageProbe160A=mean(probe160A)
averageProbe160B=mean(probe160B)
averageProbe320A=mean(probe320A)
averageProbe320B=mean(probe320B)
averageSurvey320A=mean(survey320A)
averageSurvey320B=mean(survey320B)
averageSurvey640A=mean(survey640A)
averageSurvey640B=mean(survey640B)
averageSurveysA=mean(surveysA)
averageSurveysB=mean(surveysB)
averageInspectsA=mean(inspectsA)
averageInspectsB=mean(inspectsB)
averageScoreA=mean(resultA)
averageScoreB=mean(resultB)

#Extract median values
medianIns10A=median(ins10A)
medianIns10B=median(ins10B)
medianIns20A=median(ins20A)
medianIns20B=median(ins20B)
medianIns28A=median(ins28A)
medianIns28B=median(ins28B)
medianProbe160A=median(probe160A)
```

```
medianProbe160B=median(probe160B)
medianProbe320A=median(probe320A)
medianProbe320B=median(probe320B)
medianSurvey320A=median(survey320A)
medianSurvey320B=median(survey320B)
medianSurvey640A=median(survey640A)
medianSurvey640B=median(survey640B)
medianSurveysA=median(surveysA)
medianSurveysB=median(surveysB)
medianInspectsA=median(inspectsA)
medianInspectsB=median(inspectsB)
medianScoreA=median(resultA)
medianScoreB=median(resultB)

#Extract number of survey640 and inspect28 reached
survey640NumberA=length(survey640A)
survey640NumberB=length(survey640B)
inspect28NumberA=length(ins28A)
inspect28NumberB=length(ins28B)

#Extract number of victories in each pair
insFaster10A=length(which(ins10A<ins10B))
insFaster10B=length(which(ins10A>ins10B))
insFaster10Draw=length(which(ins10A==ins10B))
insFaster20A=length(which(ins20A<ins20B))
insFaster20B=length(which(ins20A>ins20B))
insFaster20Draw=length(which(ins20A==ins20B))
t=length(ins28A)
t1=length(ins28B)
if(t>t1) {
  t2=t-t1
  for(i in 1:t2) {ins28B <- c(ins28B,750)}
}else if(t<t1){
  t2=t1-t
  for(i in 1:t2) {ins28A <- c(ins28A,750)}
}
insFaster28A=length(which(ins28A<ins28B))
insFaster28B=length(which(ins28A>ins28B))
insFaster28Draw=length(which(ins28A==ins28B))
inspectsLessA=length(which(inspectsA<inspectsB))
inspectsLessB=length(which(inspectsA>inspectsB))
inspectsLessDraw=length(which(inspectsA==inspectsB))
probeFaster160A=length(which(probe160A<probe160B))
probeFaster160B=length(which(probe160A>probe160B))
probeFaster160Draw=length(which(probe160A==probe160B))
probeFaster320A=length(which(probe320A<probe320B))
probeFaster320B=length(which(probe320A>probe320B))
```

```
118 probeFaster320Draw=length(which(probe320A==probe320B))
119 surveyFaster320A=length(which(survey320A<survey320B))
120 surveyFaster320B=length(which(survey320A>survey320B))
121 surveyFaser320Draw=length(which(survey320A==survey320B))
122 t=length(survey640A)
123 t1=length(survey640B)
124 if(t>t1) {
125   t2=t-t1
126   for(i in 1:t2) {survey640B <- c(survey640B,750)}
127 }else if(t<t1){
128   t2=t1-t
129   for(i in 1:t2) {survey640A <- c(survey640A,750)}
130 }
131 surveyFaster640A=length(which(survey640A<survey640B))
132 surveyFaster640B=length(which(survey640A>survey640B))
133 surveyFaster640Draw=length(which(survey640A==survey640B))
134 surveysLessA=length(which(surveysA<surveysB))
135 surveysLessB=length(which(surveysA>surveysB))
136 surveysLessDraw=length(which(surveysA==surveysB))
137 resultWinA=length(which(resultA>resultB))
138 resultWinB=length(which(resultA<resultB))
139 resultDraw=length(which(resultA==resultB))
140
141 #Clear unnecessary values
142 remove(t,t1,t2,i,ins10A,ins10B,ins20A,ins20B,ins28A,ins28B,
        probe160A,probe160B,probe320A,probe320B,
143         survey320A,survey320B,survey640A,survey640B,inspectsA,
                inspectsB,surveysA,surveysB,resultA,resultB)
144
145 #Draw several boxplots in one graph for comparison
146 setwd('/home/admin/workspace/tests/Inspector25')
147 ins10A1=as.vector(as.matrix(read.table("inspect10A.txt")))
148 ins20A1=as.vector(as.matrix(read.table("inspect20A.txt")))
149 ins28A1=as.vector(as.matrix(read.table("inspected28A.txt")))
150 inspectsA1=as.vector(as.matrix(read.table("inspectsA.txt")))
151
152 setwd('/home/admin/workspace/tests/Inspector50')
153 ins10A2=as.vector(as.matrix(read.table("inspect10A.txt")))
154 ins20A2=as.vector(as.matrix(read.table("inspect20A.txt")))
155 ins28A2=as.vector(as.matrix(read.table("inspected28A.txt")))
156 inspectsA2=as.vector(as.matrix(read.table("inspectsA.txt")))
157
158 setwd('/home/admin/workspace/tests/Inspector75')
159 ins10A3=as.vector(as.matrix(read.table("inspect10A.txt")))
160 ins20A3=as.vector(as.matrix(read.table("inspect20A.txt")))
161 ins28A3=as.vector(as.matrix(read.table("inspected28A.txt")))
162 inspectsA3=as.vector(as.matrix(read.table("inspectsA.txt")))
```

```
163
164 setwd('/home/admin/workspace/tests/Inspector100')
165 ins10A4=as.vector(as.matrix(read.table("inspect10A.txt")))
166 ins20A4=as.vector(as.matrix(read.table("inspect20A.txt")))
167 ins28A4=as.vector(as.matrix(read.table("inspected28A.txt")))
168 inspectsA4=as.vector(as.matrix(read.table("inspectsA.txt")))
169
170 setwd('/home/admin/workspace/tests/Sentinel25')
171 survey320A1=as.vector(as.matrix(read.table("survey320.txt")))
172 survey640A1=as.vector(as.matrix(read.table("survey640.txt")))
173 surveysA1=as.vector(as.matrix(read.table("surveys.txt")))
174
175 setwd('/home/admin/workspace/tests/Sentinel50')
176 survey320A2=as.vector(as.matrix(read.table("survey320.txt")))
177 survey640A2=as.vector(as.matrix(read.table("survey640.txt")))
178 surveysA2=as.vector(as.matrix(read.table("surveys.txt")))
179
180 setwd('/home/admin/workspace/tests/Sentinel75')
181 survey320A3=as.vector(as.matrix(read.table("survey320.txt")))
182 survey640A3=as.vector(as.matrix(read.table("survey640.txt")))
183 surveysA3=as.vector(as.matrix(read.table("surveys.txt")))
184
185 setwd('/home/admin/workspace/tests/Sentinel100')
186 survey320A4=as.vector(as.matrix(read.table("survey320.txt")))
187 survey640A4=as.vector(as.matrix(read.table("survey640.txt")))
188 surveysA4=as.vector(as.matrix(read.table("surveys.txt")))
189
190 setwd('/home/admin/workspace/tests/Sentinel133')
191 survey320A5=as.vector(as.matrix(read.table("survey320.txt")))
192 survey640A5=as.vector(as.matrix(read.table("survey640.txt")))
193 surveysA5=as.vector(as.matrix(read.table("surveys.txt")))
194
195 setwd('/home/admin/workspace/tests/Buying Explorer')
196 resultA2=as.vector(as.matrix(read.table("resultA.txt")))
197
198 setwd('/home/admin/workspace/tests/Buying Saboteur')
199 resultA3=as.vector(as.matrix(read.table("resultA.txt")))
200
201 setwd('/home/admin/workspace/tests/Explorer50')
202 probe160A13=as.vector(as.matrix(read.table("probe160A.txt")))
203 probe320A13=as.vector(as.matrix(read.table("probe320A.txt")))
204 probeAllA13=as.vector(as.matrix(read.table("probeAllA.txt")))
205
206 setwd('/home/admin/workspace/tests/Explorer133')
207 probe160A14=as.vector(as.matrix(read.table("probe160A.txt")))
208 probe320A14=as.vector(as.matrix(read.table("probe320A.txt")))
209 probeAllA14=as.vector(as.matrix(read.table("probeAllA.txt")))
```

```
210
211 setwd('/home/admin/workspace/tests/Original')
212 ins10A5=as.vector(as.matrix(read.table("inspect10A.txt")))
213 ins20A5=as.vector(as.matrix(read.table("inspect20A.txt")))
214 inspectsA5=as.vector(as.matrix(read.table("inspectsA.txt")))
215 ins28A5=as.vector(as.matrix(read.table("inspected28A.txt")))
216 survey320A6=as.vector(as.matrix(read.table("survey320.txt")))
217 survey640A6=as.vector(as.matrix(read.table("survey640.txt")))
218 surveysA6=as.vector(as.matrix(read.table("surveys.txt")))
219 probe160A5=as.vector(as.matrix(read.table("probe160A.txt")))
220 probe320A5=as.vector(as.matrix(read.table("probe320A.txt")))
221 probeAllA5=as.vector(as.matrix(read.table("probeAllA.txt")))
222 resultA1=as.vector(as.matrix(read.table("resultA.txt")))
223
224 boxplot(ins10A1,ins10A2,ins10A3,ins10A4,ins10A5,names=c("ins25
      ","ins50","ins75","ins100","original"),horizontal=TRUE,
      ylab="Configuration",xlab="Turn",main="Inspect10
      achievement")
225 boxplot(ins20A1,ins20A2,ins20A3,ins20A4,ins20A5,names=c("ins25
      ","ins50","ins75","ins100","original"),horizontal=TRUE,
      ylab="Configuration",xlab="Turn",main="Inspect20
      achievement")
226 boxplot(ins28A1,ins28A2,ins28A3,ins28A4,ins28A5,names=c("ins25
      ","ins50","ins75","ins100","original"),horizontal=TRUE,
      ylab="Configuration",xlab="Turn",main="Inspect28 milestone
      ")
227 boxplot(inspectsA1,inspectsA2,inspectsA3,inspectsA4,inspectsA5
      ,names=c("ins25","ins50","ins75","ins100","original"),
      horizontal=TRUE,ylab="Configuration",xlab="Number",main="
      Inspect actions")
228
229 boxplot(survey320A1,survey320A2,survey320A3,survey320A4,
      survey320A5,survey320A6,names=c("sent25","sent50","sent75"
      ,"sent100","sent133","original"),horizontal=TRUE,ylab="
      Configuration",xlab="Turn",main="Survey320 achievement")
230 boxplot(survey640A1,survey640A2,survey640A3,survey640A4,
      survey640A5,survey640A6,names=c("sent25","sent50","sent75"
      ,"sent100","sent133","original"),horizontal=TRUE,ylab="
      Configuration",xlab="Turn",main="Survey640 achievement")
231 boxplot(surveysA1,surveysA2,surveysA3,surveysA4,surveysA5,
      surveysA6,names=c("sent25","sent50","sent75","sent100","
      sent133","original"),horizontal=TRUE,ylab="Configuration",
      xlab="Turn",main="Survey actions")
232
233 boxplot(resultA2,resultA3,resultA1,names=c("explorer upgr","
      saboteur upgr","original"),horizontal=TRUE,ylab="
      Configuration",xlab="Score",main="Results")
```

```
234
235 boxplot(probe160A13, probe160A14, probe160A5, names=c("explorer50
        ","explorer133","original"), horizontal=TRUE, ylab="
        Configuration", xlab="Turn", main="Probe160 achievement")
236 boxplot(probe320A13, probe320A14, probe320A5, names=c("explorer50
        ","explorer133","original"), horizontal=TRUE, ylab="
        Configuration", xlab="Turn", main="Probe320 achievement")
237 boxplot(probeAllA13, probeAllA14, probeAllA5, names=c("explorer50
        ","explorer133","original"), horizontal=TRUE, ylab="
        Configuration", xlab="Turn", main="ProbeAll milestone")
```

D.4 Script for starting simulations

The small shell script used for starting up the simulation server, GUI and both the teams, as well as redirecting the agents output from console to log files.

```
1 #!/bin/bash
2
3 gnome-terminal \
4   --tab-with-profile=Default --working-directory="/home/admin/
        workspace/massim-2013-1.4/massim/scripts" -e "bash -c '
        sleep 5; ./startMarsMonitor.sh; exec $SHELL'" \
5   --tab-with-profile=Default --working-directory="/home/admin/
        workspace/MAPC-UFSCTeam2013changed" -e "bash -c 'sleep
        5; ant runA > "/home/admin/workspace/massim-2013-1.4/
        massim/scripts/backup/outputA.log"; exec $SHELL'" \
6   --tab-with-profile=Default --working-directory="/home/admin/
        workspace/MAPC-UFSCTeam2013original" -e "bash -c 'sleep
        5; ant runB > "/home/admin/workspace/massim-2013-1.4/
        massim/scripts/backup/outputB.log"; exec $SHELL'" \
7   --tab-with-profile=Default --working-directory="/home/admin/
        workspace/massim-2013-1.4/massim/scripts" -e "bash -c
        './startServer.sh; exec $SHELL'" \
```

Bibliography

[BBH+13] Olivier Boissier, Rafael Bordini, Jomi Fred Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. In *Science of Computer Programming*, volume 78, pages 747–761. Elsevier, 2013.

[BFPW03] Rafael Bordini, Michael Fisher, Carmen Pardavila, and Michael Wooldridge. Model checking AgentSpeak. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 409–416. ACM Press, 2003.

[BHW07] Rafael Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason.* Wiley, 2007.

[BJV10] Niklas Skamriis Boss, Andreas Schmidt Jensen, and Jørgen Villadsen. Building multi-agent systems using Jason. In *Annals of Mathematics and Artificial Intelligence*, volume 59, pages 373–388. Springer, 2010.

[Bra87] Michael Bratman. *Intention, plans, and practical reason.* Harvard University Press, 1987.

[CDM12] Victoria Catterson, Euan Davidson, and Stephen McArthur. Practical applications of multi-agent systems in electric power systems. In *European Transactions on Electrical Power*, volume 22, pages 235–252. Wiley, 2012.

[DKS13] Jürgen Dix, Michael Köster, and Federico Schlesinger. Scenario description for the multiagent contest. `https://multiagentcontest.org/scenario`, 2013.

[FO13] Carl Frey and Michael Osborne. *The Future of Employment: How Susceptible are Jobs to Computerisation?* Oxford Uiversity Press, 2013.

[HB10] Jomi Fred Hübner and Rafael Bordini. Using agent- and organisation-oriented programming to develop a team of agents for a competitive game. In *Annals of Mathematics and Artificial Intelligence*, volume 59, pages 351–372. Springer, 2010.

[HSB07] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. In *International Journal of Agent-Oriented Software Engineering*, volume 1, pages 370–395. InderScience Publishers, 2007.

[HZB+13] Jomi Fred Hübner, Maicon Rafael Zatelli, Michael de Brito, Tiago Luiz Schmitz, Daniela Maria Uez, Marcelo Menezes Morato, and Kaio Siqueira de Souza. UFSC system code. `https://multiagentcontest.org/downloads/Multi-Agent-Programming-Contest-2013/Sources/UFSC/`, 2013.

[ISM10] David Isern, David Sánchez, and Antonio Moreno. Agents applied in health care: A review. In *International journal of medical informatics*, volume 79, pages 145–166. Elsevier, 2010.

[Jen00] Nicholas Jennings. On agent-based software engineering. In *Artificial Intelligence*, volume 117, pages 277–296. Elsevier, 2000.

[KDM+11] Boštjan Kaluža, Erik Dovgan, Violeta Mirchevska, Božidara Cvetkovic, Mitja Lušterek, and Matjaž Gams. A Multi-Agent System for Remote Eldercare. In *Trends in Practical Applications of Agents and Multiagent Systems*, volume 90, pages 33–40. Springer, 2011.

[LAAM10] Vivian López, Noel Alonso, Luis Alonso, and María Moreno. A Multiagent System for Efficient Portfolio Management. In *Trends in Practical Applications of Agents and Multiagent Systems*, volume 71, pages 53–60. Springer, 2010.

[LHC06] Yong-Feng Lin, Ming-Wei Huang, and Jason Jen-Yen Chen. Agent-based unit testing environment for extreme programming. In *Journal of Computational Methods in Sciences and Engineering*, volume 6, pages 1–8. IOS Press, 2006.

[PMG+12] Carla Pereira, Jason Mahdjoub, Zahia Guessoum, Luís Gonçalves, and Manuel Ferreira. Using MAS to Detect Retinal Blood Vessels. In *Highlights on practical applications of agents and multi-agent systems*, volume 156, pages 239–246. Springer, 2012.

[PVM+14] Carla Pereira, Diana Veiga, Jason Mahdjoub, Zahia Guessoum, Luís Gonçalves, Manuel Ferreira, and João Monteiro. Using a multi-agent system approach for microaneurysm detection in fundus images. In *Artificial intelligence in medicine*, volume 60, pages 179–188. Elsevier, 2014.

[Rao96] Anand Rao. AgentSpeak (L): BDI agents speak out in a logical computable language. In *Lecture Notes in Computer Science*, volume 1038, pages 42–55. Springer, 1996.

[RG91] Anand Rao and Michael Georgeff. Modeling rational agents within a BDI-architecture. In *Principles of Knowledge Representation and Reasoning (KR)*, pages 473–484. Morgan Kaufmann Publishers, 1991.

[RG95] Anand Rao and Michael Georgeff. BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multiagent Systems*, pages 312–319. AAAI Press, 1995.

[RJGPnS+10] Miguel Reboiro-Jato, Daniel Glez-Peña, Hugo Santos, Mário Diniz, Carlos Lodeiro, José Capelo, and Florentino Fdez-Riverola. Multi-agent System for Mass Spectrometry Analysis. In *Trends in Practical Applications of Agents and Multiagent Systems*, volume 71, pages 87–95. Springer, 2010.

[RPVO09] Alessandro Ricci, Michele Piunti, Mirko Vireli, and Andrea Omicin. Environment Programming in CArtAgO. In *Multi-agent programming : languages, platforms and applications*, pages 259–288. Springer, 2009.

[Sho93] Yoav Shoham. Agent-oriented programming. In *Artificial Intelligence*, volume 60, pages 51–92. Elsevier, 1993.

[SMRM55] Claude Elwood Shannon, Marvin Lee Minsky, Nathaniel Rochester, and John McCarthy. A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. `http://www.formal.stanford.edu/jmc/history/dartmouth/dartmouth.html`, 1955.

[Woo95] Michael Wooldridge. Intelligent agent: theory and practice. In *Knowledge Engineering Review*, volume 10, pages 115–152. Cambridge University Press, 1995.

[ZBS+13] Maicon Rafael Zatelli, Maiquel de Brito, Tiago Luiz Schmitz, Marcelo Menezes Morato, Kaio Siqueira de Souza, Daniela Maria Uez, and Jomi Fred Hübner. SMADAS: A Team for MAPC Considering the Organization and the Environment as First-class Abstractions. In *Engineering Multi-Agent Systems*, pages 319–328. Springer, 2013.