Multi-Agent Systems in StarCraft

by

H.J. Griffioen and D. Plenge

Project duration: April 18, 2016 – July 17, 2016
Supervisors: Dr. K. V. Hindriks, TU Delft, supervisor
Ir. V. J. Koeman, TU Delft, client
Dr. M. A. Larson, TU Delft, Bachelor project coordinator

An electronic version of this thesis is available at http://repository.tudelft.nl/.
Preface

This report is submitted in partial fulfillment of the requirement for the degree of Bachelor in Technical Computer Science at the Technical University of Delft. This report concludes the Bachelor project for third year students. The project was coached by Dr. K. V. Hindriks, from the Delft University of Technology. All research and development took place at the Delft University of Technology, department of Electrical Engineering, Mathematics and Computer Science. The project started at April 18th 2016 and finished at June 17th 2016.

We would like to thank our coach Dr. K. V. Hindriks, for giving us this opportunity in helping out in improving the MAS-Project course and giving us good structural feedback over the course of the project.

We would also like to thank our client Ir. V.J. Koeman, for helping us out when needed and for also providing us with very detailed feedback over the course of the project.

H.J. Griffioen and D. Plenge
Delft, June 2016
Summary

The department Interactive Intelligence of the TU Delft offer the Multi-Agent Systems (MAS) project to first year computer science students. The goal of this project is for computer science students to learn about multi-agent systems by creating such a system for the game Unreal Tournament. The system is created by using GOAL, which is an agent programming language developed by the TU Delft.

Unreal Tournament, the game that is currently being used in the project, does not challenge the computer science students enough to create an interesting strategy for the MAS. To encourage the computer science students to implement interesting strategies, it would be optimal to change the game to something more strategic. Therefore the department of Interactive Intelligence would like to switch the game for the course from Unreal Tournament to StarCraft. StarCraft is a real-time strategy game in which the player controls units which are used to defeat the opponent.

During the project, the creation of a MAS for StarCraft using GOAL was made possible. In addition to the connector that was made to make this possible, a course manual was created that holds information about the learning goals of the course and the product that computer science students should achieve.

Another interesting thing that puts StarCraft above Unreal Tournament is the Student StarCraft AI tournament. Computer science students that excel would have the opportunity to join this tournament with their MAS and compete in the yearly battles.

At the end of this project a fully functioning connector with all necessary tools and manuals is delivered. As well as a review of the robustness of the system.
Contents

1 Introduction 1

2 Methodology 2
  2.1 Workflow 2
  2.2 Quality Assurance 2
  2.3 Testing 3
  2.4 Agreements 3
  2.5 Definition of done 3

I Research 4

3 Project 5
  3.1 Problem Analysis 5
    3.1.1 Project Problem 5
    3.1.2 Aim of the Project 5
    3.1.3 Challenges in the project 6
  3.2 Why StarCraft Broodwar 6
    3.2.1 Why not StarCraft 2 6
  3.3 Project Definition 7
  3.4 Existing solutions 7
  3.5 Requirements 7

4 Education and abstraction 10
  4.1 Educational value 10
  4.2 GOAL 10
  4.3 Connector 10
  4.4 Abstraction level 11
    4.4.1 SSCAIT versus abstraction 11
    4.4.2 Abstractions 11
  4.5 Conclusion 12

5 Environment 13
  5.1 An API for StarCraft 13
  5.2 Existing EIS connector 14
  5.3 Existing bots 14
    5.3.1 Protoss bot by Andrew Smit 14
    5.3.2 Zerg bot by Marian Devecka 14
    5.3.3 Terran bot by CELab. 15
    5.3.4 Conclusion 15

II Design 16

6 Connector 17
  6.1 Architecture 17
    6.1.1 EIS 17
    6.1.2 JNI.BWAPI 18
  6.2 Design 19
# Percepts and actions

7.1 Percepts
   - 7.1.1 Types of percepts
   - 7.1.2 Factors that influence the percept structure
   - 7.1.3 Optimisation
   - 7.1.4 Problems
   - 7.1.5 Percepts for StarCraft

7.2 Actions
   - 7.2.1 Considerations
   - 7.2.2 Problems
   - 7.2.3 Set of actions for StarCraft

7.3 Conclusion

# Development tools

8.1 Features
8.2 Design
   - 8.2.1 Robustness

# Implementation

9.1 Thread structure
9.2 Problems and solutions
9.3 Final system
   - 9.3.1 Connector
   - 9.3.2 Development tools
9.4 Abstraction level
9.5 Multi-agent systems

# Education

10.1 The MAS-Project
10.2 The learning goals
10.3 The StarCraft Course
   - 10.3.1 Why StarCraft?
   - 10.3.2 Individual assignment
   - 10.3.3 Performance targets
   - 10.3.4 The end competition

# Quality

11.1 Concurrent Agents
11.2 Agent behaviour versus game speed
11.3 System run-time
11.4 System and specifications

12.1 First SIG feedback
12.2 Improvements
12.3 Second SIG feedback
Introduction

Each year, the TU Delft organises a course for first-year bachelor computer science students: Multi-Agent Systems (MAS) Project. In this project, students learn about multi-agent systems by developing a MAS using the GOAL programming language. For the past years the game: Unreal Tournament (UT) has been used for the MAS project. The students had to develop a MAS for this game which was capable in defeating the built-in AI of UT. First UT2004 was used for this project, but 3 years ago the UT3 connector was introduced for the MAS project. Unfortunately, the UT connectors have not always been functioning as expected and have brought some problems within the MAS course. Since UT has been used for quite some time now for the MAS project and students are not sufficiently challenged to create interesting strategies in UT, the time has come when replacing UT with something new.

StarCraft: Brood War is a Real-Time Strategy (RTS) game, released in 1998 and produced by Blizzard (Read appendix C for more information on StarCraft). In a RTS game, the player controls a base with defensive structures and factories that produces mobile units to form an army which can be used to attack the opponent. The big difference between a normal strategy game and a RTS game is that both players are controlling their units in real time, which means that there are no turns like in chess. This requires the player to react fast on certain situations and to make fast decisions in order to ‘outsmart’ your opponent. Because StarCraft is a game where anything can happen and many strategies can be used and implemented, it has recently become very popular for AI developers. Even Google has shown interest in developing a StarCraft bot with their AI ‘Deepmind’. After the AlphaGo project, where they managed to defeat one of the best human players in the game ‘Go’, Google has set their sights on StarCraft. Defeating one of the best human players in StarCraft could even be a harder task than defeating one of the best players of Go, because of the even greater amount of possible moves one can make in StarCraft. It has become a big challenge for AI developers to create a bot which is capable of defeating a human player without leaving out a scenario where a player would still be able to defeat the AI. With the creation of the Brood War Application Programming Interface (BWAPI) anyone can make a StarCraft bot with the programming language C++. Since the creation of the Java Native Interface Brood War Application Programming Interface (JNIBWAPI), the programming language Java can also be used to develop a StarCraft bot.

Because of the creation of JNIBWAPI, a bridge could be created between the GOAL programming language and StarCraft Broodwar, which lead to the creation of the StarCraft connector. With the StarCraft connector students can create bots using the GOAL programming language which is capable of controlling StarCraft units defined as agents. The StarCraft connector should learn students which complications developing a multi-agent system on a large scale (around 200 agents) can bring along. Developing a StarCraft bot takes a lot of scenario testing and strategy designing, which offers a fun and educational challenge for the first year students who are taking the course.
Methodology

This chapter highlights the methodology that was used to ensure the quality of the product that will be delivered. The methodology also provides a workflow where all team members know their task and can work with minimal delay. In section 2.1, the workflow is discussed. Section 2.2 highlights the agreements made to ensure the quality of the product. Section 2.3 discusses how testing was used to ensure that the product works. Finally, the agreements that the team made beforehand are stated in section 2.4.

2.1. Workflow

A smooth workflow can help the team to work better and more efficiently. The workflow is devised beforehand because this will help to prevent unwanted situations and misunderstandings. The workflow is as follows:

• The team will work at the company of the client whenever possible, this ensures that both the team members distribute the work evenly and that design decisions can immediately be addressed.

• The team will work via the agile method, sprints will be weekly. The sprints are planned on Friday and the next Friday there will be a sprint retrospective and a new sprint plan. For ease of use sprint plans will be available in Google Drive rather than Github.

• A team member will check the work of the other team member via Github pull requests.

• Comments on code, commits, and the report will all be written in English.

• There are weekly meetings with the coach where every team member will be present.

• Team members will discuss via Telegram when not working at the same place. If a team member cannot come to the client’s company, Skype will be used.

• There will be a Telegram chat with the client. The coach can be reached via email.

• The team will divide tasks based on estimated effort, this will be put in the sprint plans. If one team member finishes early, he will help the other team member or pick a new task after consulting with the rest of the team.

2.2. Quality Assurance

The quality of the product is very important. The team should strive to deliver a product with the highest possible quality. To ensure that this quality will be reached, the team set up guidelines for when it comes to quality.

• The code should be tested for at least 80%.

• The team will use various code quality tools, the code quality tools are:
  – FindBugs
2.3. Testing

- PMD
- CPD
- Checkstyle
- SonarQube

- Team members check code via pull requests. While working together, code quality can be discussed among the team before submitting a pull request.

2.3. Testing

Testing the system is important, as the team can then use test driven development and regression tests to their full potential. To ensure that the code has a vast set of tests, the coverage will be measured by using cobertura from maven. Regression tests will be conducted by using Travis continuous integration with Github.

2.4. Agreements

To make the teamwork as effective as possible, some agreements were made at the start of the project.

- The team will adhere to the methodology at all times. This includes the workflow, quality assurance and these agreements.

- Because the whole team is responsible for the delivered product, all team members should agree about the product when delivering the product.

- Each team member will spend at least 420 hours on the project, according to the load of 15 ECTS.

2.5. Definition of done

It is important that the whole team knows when a component or feature is ‘done’. To assure that everyone has the same understanding of the word ‘done’, the following definition is used:

A component or feature is done when it is:

- Implemented
- Tested
- Passing the continuous integration
- Documented
Part I

Research
This chapter defines and analyses the project. After the problem analysis and definition, goals are formulated which are needed to complete the project.

### 3.1. Problem Analysis

The project aims at revising the Multi-Agent Systems project (MAS-project) which is given to over 200 students as part of the Computer Science bachelor curriculum [14]. In this course the students have to make a multi-agent system (MAS) in groups of six students, using the programming language GOAL [2]. At the end of the project, every MAS that is created by the students will enter a competition to determine which MAS is the best. To create the groups, there is an individual assignment in the first week of the MAS-project. This assignment is used to determine the skill level of students and group students with the same level together. The current MAS-Project uses the game: Unreal Tournament 3. The tournament is held in the game-mode ‘capture the flag’, which is won by the team that has captured the enemy flag the most [24]. Despite the fact that this game is fun, using this game for the MAS-project is not optimal.

#### 3.1.1. Project Problem

One problem is the limited amount of agents that are currently being used in the MAS-project. The students are instructed to create a team of 4 agents. Despite the fact that these are ‘multiple’ agents, limiting the amount to 4 limits the coordination needed in the MAS. Some of the groups do not implement some form of tactics at all but run back and forth between flag-bases in the game. To make the MAS-project more interesting, a game where there are more agents and where there is a greater need of tactics could be used to create a more interesting challenge. The course should not be harder, as all students should be able to pass, but for smart students it would pose more depth.

Another problem is the connector that connects agent written in GOAL to Unreal Tournament 3. The connector is built on top of an API that has a refresh rate of 200 ms. For a first person shooter game, 200 ms is a rather high number. This refresh rate sometimes results in situations where agents are unable to identify enemy bots in time. As a consequence, bots controlled by these agents are killed but there is little the agent could have done to prevent this.

Finally there is also the load on a computer system. Unreal Tournament 3 is a processing heavy game. As students often bring their own laptops which are not always capable of running large games, this also poses a problem. On a Mac or Linux system it is hard to run Unreal Tournament 3, this has to be done on a virtual machine running Windows. Running on a virtual machine poses problems because of the processing power needed for the game, rendering the game unresponsive. With another game that is less processor heavy this would pose less problems.

#### 3.1.2. Aim of the Project

To focus the MAS-project on more agents and implementing interesting tactics, picking another game would be a solution. The client is interested in changing the game used in the MAS-project. The game that the client is interested in is ‘StarCraft’ (figure 3.1). This game is a so-called strategy game (C.2), where units are all individually controlled by the player. More information on StarCraft can be found in Appendix C.1.
Table 3.1: Differences between Unreal Tournament 3 and StarCraft

<table>
<thead>
<tr>
<th></th>
<th>UT3</th>
<th>StarCraft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum agents</td>
<td>4 ± 200</td>
<td></td>
</tr>
<tr>
<td>Types of agents</td>
<td>1 ± 60</td>
<td>(20 per race)</td>
</tr>
<tr>
<td>Number of actions</td>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>Number of percepts</td>
<td>23</td>
<td>23</td>
</tr>
</tbody>
</table>

For the MAS-project it is interesting if every unit in the game will be an agent, as students then deal with a real MAS with a large number of agents. Because the MAS-project uses GOAL as programming language, an environment has to be developed to connect GOAL to the game. GOAL can be connected to an environment using the Environment Interface Standard (EIS) [9], there is currently no other way to connect GOAL to an environment. Thus to connect GOAL to StarCraft, we have to create a connector using EIS.

3.1.3. Challenges in the project
To replace UT3 in the MAS-project with StarCraft, students will face a lot more challenges than they did with Unreal Tournament 3 (UT3). In Table 3.1 some of the differences between the environments are listed. The StarCraft environment will give a lot more functionality than the UT3 environment, as it will have a lot more agents with different types than the UT3 environment has. Agents can communicate with each other in GOAL by sending messages. In StarCraft, the amount of agents could lead to difficult communication between agents, because many agents are communicating at the same time. This could lead to a very complex MAS-project for first-year students. To counteract this, the abstraction level which is present in the environment should be adapted. The abstraction level is to what extend the connector takes care of things that otherwise have to be dealt with by the user itself.

3.2. Why StarCraft Broodwar
StarCraft is a Real-Time Strategy game released in 1998, developed and published by Blizzard. 9 million copies of StarCraft were sold worldwide in early 2008. Especially in South Korea the game is still very popular. Besides the fact that StarCraft is a fun game to play, creating bots for StarCraft is also becoming very popular. In 2011 an educational event was created where students can challenge each other’s bots in a tournament named: The Student StarCraft AI Tournament (SSCAIT) [17]. When this project is completed, students can submit their GOAL agents to this tournament. This may make the project even more interesting for students who like these competitions. This tournament was created because StarCraft is a big AI challenge that has not yet been solved. Google is also interested in developing AI for StarCraft; after ‘Deep Mind’ defeated masters of the game ‘Go’, Google is focussing on StarCraft [1]. Google stated that StarCraft is much harder for an AI than Chess or Go as it has a lot more possible scenario’s.

3.2.1. Why not StarCraft 2
To connect to a game, an API like BWAPI is nessecary. The problem here is that this kind of behaviour of a program can be seen as maphacking. Maphacking means that users can see the whole map and all enemies instead of only the part that they should see in game. Despite the fact that Blizzard has supported the work of the StarCraft Broodwar AI before, they do not want the creation of a similar API for StarCraft 2. Until Blizzard changes its policy about this matter, using StarCraft 2 for the AI tournament will be out of the question [25].

Other minor reasons:

- In the interest of academic development, Broodwar is cheaper, and its hardware requirements are lower than StarCraft 2, making it more accessible to university students. BWAPI is actively being used as university courseware (professors are actually using BWAPI in classrooms) [25].

- The StarCraft 2 engine is not ideal for AI development. Because Broodwar is not processor intensive, the logical framerate of Broodwar can easily be increased, enabling games can be played more quickly. Allowing some AI’s that use a form of machine learning to learn more quickly. StarCraft 2 requires more processor capacity, which means that increasing the logical frame rate will use up much more processor time and that means that the AI will receive less processing time [25].
• AI/API developers will need to handle far more information than with Broodwar, because StarCraft 2 is more complex than StarCraft Broodwar [25].

3.3. Project Definition

As stated in section 3.1.2, the aim of this project is to create a connector for StarCraft that can be used as part of an educational computer science course. Therefore, the following project GOAL can be devised.

Creating a connector for the game StarCraft that can be used for the MAS-project as well as the StarCraft competition.

3.4. Existing solutions

There is an earlier attempt to connect agents to StarCraft using EIS, this was done by Andreas Schmidt Jensen [5]. This connector is not finished and it misses a lot of functionality. The connector that he built can be used as a starting point, but does need a lot of improvement and code refactoring to adhere to the quality that the team wants to achieve.

3.5. Requirements

To define the scope of the project, the goals of the project are defined at the start. The goals have been set according to the wishes of the client and were set during the research phase of the project. The project planning, which can be found in Table A.1, has been created with these goals as major milestones. At the end
of the project, these goals should all be accomplished. When all goals are accomplished, this project can be considered a success.

1. **StarCraft connector**
   This is the most essential part of the project. The connector connects GOAL agents to StarCraft. This connector has to be developed using the EIS-interface [9]. It should be properly tested and contain all the features that are necessary to create a MAS that is capable of competing in the Student StarCraft AI Tournament.

2. **Development tools**
   In order to make developing of bots as easy as possible, dedicated development tools have to be created, to see what is happening inside the game. The development tools have to fulfill the following requirements:
   
   • It should be possible to display map data. This makes it easier to visualize what happens on the map. These are:
     - Construction sites
     - Base locations
     - Chokepoints
   
   • The game speed should be adaptable to run games as fast as possible. This will minimize the running time of the game. With this speed, debugging will be faster. Running the game in slow motion should also be included, as detailed testing while the user can easily focus on one thing at a time is then possible. The speed should range from 50% speed to as fast as possible.
   
   • Users should be able to toggle cheats on and off using buttons, these cheats can help to create scenario's in the game for testing. The cheats that should be supported are:
     - Give 10.000 from every resource, this will make it easy to test build orders and can be used to easily create scenario's that help to test the MAS.
     - Show the whole map. This helps with visualizing what the enemy does to counter the strategy of the MAS.
     - The enemy cannot harm the units of the player. With this option, the MAS can go mind his own business without worrying about the enemy. Testing will be easier because the MAS will not be defeated before a scenario is reached.

3. **Environment manual**
   To use the environment, one has to know which features are implemented and which syntax is used. This will be documented in an environment manual. The manual should contain the following sections:
   
   • StarCraft description
   • Setting up the environment
   • Connecting GOAL to StarCraft
   • Development tools
   • Actions
   • Percepts
   • Frequently asked questions

4. **Project manual**
   A project manual is needed to be able to use the StarCraft connector in the MAS-project. This project manual should contain milestones that are feasible to complete in 10 weeks. The project manual should focus on the use of StarCraft and define clearly the purpose of the course and what is needed to pass. The following chapters have to be present in the manual:
   
   • Learning Goals
   • Group assignment
3.5. Requirements

- Organisation
- Individual assignment

5. Multi-agent system
To test this product and the schedule in the project manual, a MAS has to be created that is built according to the planning in the guide. In the project, students can choose between the three races present in StarCraft. Therefore, a MAS should be created for every race, because the difference between the difficulty level of creating a MAS for a race should be minimal. The students have to learn what the environment offers and will therefore need more time to program a MAS. Therefore the MAS should be built in at most 2 weeks to ensure that the students should be able to do the same in 10 weeks. 2 weeks in the bachelor project is equal to 84 hours per person, because the team consists of two members this is 168 hours. The students have ± 10 weeks of 5 ECTS in groups (140 hours per student), thus have a lot more time that they can use to learn the environment and build a MAS.

6. Performance tests
The client would like test results of the robustness of the system. This includes the performance in terms of load:

- How many agents can be ran concurrently?
- Does the size of the map influence this?

and in terms of time:

- Do agents start to behave differently when the game runs on different speeds?
- How long can the system run?
Education and abstraction

This chapter highlights the abstraction level that has to be achieved in the system to create a fun challenge for first-year students. Abstraction level means the extent to which functionality is programmed in the connector to make creating a MAS in GOAL easier. First, the educational value that this system can offer is highlighted. Then the programming language GOAL is explained in more detail as it is important to understand the GOAL language to understand what abstraction level means. Finally, the necessity of a certain kind of abstraction is explained and the abstraction level itself is introduced. This chapter will answer the following research question:

What kind of abstraction level should be offered to make sure the difficulty level of creating a MAS in StarCraft is adequate for the first-year Bachelor MAS-project?

4.1. Educational value

The goal of the system is to accommodate the 10 week MAS-project for students. This is the main focus of the system. However, because there is an AI competition in StarCraft [17], excellent students can be offered the opportunity to join this competition. This could, for example, be a part of the honours programme of the TU Delft. Therefore the aim of the system should not solely be the MAS-project.

The goal of the MAS-project is to learn about multi-agent systems. By using StarCraft rather than UT3, the students will learn to create more complex systems with a lot more agents. Thus the transition to StarCraft will provide more learning material for the students.

4.2. GOAL

The objective of the MAS-project is to create a MAS that runs on the GOAL platform. GOAL is a programming language to create agents that act autonomously. To use GOAL, one needs a connector. The connector offers Actions and Percepts, which allow the agent to interact with StarCraft. Actions make things happen, for instance moving a unit in StarCraft. Percepts offer information about the current state of the environment. For instance, agents should be able to percept enemy units. The connector is implemented using the Environment Interface Standard [9], GOAL uses EIS to connect to an environment.

4.3. Connector

The actions and percepts for the connector will be created for StarCraft in this project. Agents are computer programs that use percepts and actions to reach their goals. Students using this connector during the MAS-project should be able to create a MAS in 10 weeks, because this is the time allocated to the MAS-project. This time could prove limited for the students to implement the minimum functionality to let the agents play the game and to implement the tactics they want the agent to have. To counteract this time limitation, the connector can handle certain functionalities that are needed for agents to function. These functionalities are then handled in the connector and do not have to be programmed using GOAL by the user. By doing so, the user does not have to worry about those functionalities and can focus on the tactics that need to be deployed. To what extent these functionalities are pre-programmed is known as the abstraction level of the connector, which will be discussed in the next part.
4.4. Abstraction level

An abstraction in the connector is the removal of details. These are details the programmer of the MAS does not have to worry about anymore. For example, in the current Unreal environment in the MAS-project, path finding is not programmed by the user, but included in the connector. This is an example of an abstraction in the connector. The users of the connector should have the freedom to change all the behaviour of units in the game as they desire, but there also needs to be enough built-in functionality to make the MAS-project fun and doable.

In StarCraft for example, the students should not have to worry about moving exactly to a mineral field to start gathering from it. Ideally there would be a method gather(<Mineralfield>) which automatically takes care of this, as the user then does not need to take care of locating the mineralfields by position. The students should worry about tactics and not the precise implementation of features. Tactics are all the means used to win the game, and consist of two concepts:

- Micro management (B.1).
- Macro management (B.2).

The abstractions are listed in 4.4.2.

4.4.1. SSCAIT versus abstraction

Users of the environment should also be able to join the AI competition of StarCraft [17]. To create a versatile bot it might be necessary to adapt certain details of the bot. In an ideal case for a tournament bot the user programs all functionality himself, because he then has the smallest chance to encounter undesired behaviour. Undesired behaviour is behaviour that is programmed in the environment which does not work like the user thinks and does not satisfy the user’s needs. Care needs to be taken with programming abstractions, so that users will always be able to change every behaviour of the agent.

4.4.2. Abstractions

Below is a list with abstractions following the MoSCoW principle. It is also explained which care needs to be taken when implementing the abstraction.

Must have

- **Basic pathfinding**, when a user wants an agent to move to a location, the agent should do so. StarCraft has built-in pathfinding, this pathfinding can be used. The user can create own pathfinding by sending multiple move commands, thus this abstraction does not pose undesired behaviour.

- **Gathering**, when a user wants an agent to gather a mineralfield, the user should be able to call a method gather(<Mineralfield>). This way users can easily gather from mineralfields. When the user wants to do so, he will also be able to split their workers for gathering optimisation (B.4).

Should have

- **Automatic attacking**, when an idling agent sees an enemy that is in range, the agent should attack the unit automatically. When other behaviour is wanted the user can make the agent flee or kite (B.3) the other unit.

Could have

- **Grouping units**, grouping units in squads could be accommodated by the environment. Grouping units is a part of micro-management and creating squads can also be done by GOAL code. Grouping will therefore preferably not be an abstraction because students should be able to implement this. Implementing a grouping system in the environment would remove some need for messaging in the agents. This functionality will be added if the agent messaging gets out of hand when implementing this in GOAL code.
• **Gathering optimisation**, gathering in the start of the game can be optimized to speed up the mineral gathering rate of the player. Adding this functionality could create undesired behaviour, a user might not want to gather in this way but deploy a different tactic.

• **Advanced pathfinding** (always choosing the optimal path), advanced pathfinding is not necessary, because basic pathfinding is present in the game. Implementing advanced pathfinding could pose a fun challenge for students who want to implement this, thus will be interesting for excellent students. This is not functionality that the MAS-project will require.

**Won't have**

• **Automated buildorder support**, buildorder is the order in which buildings are built. Automated buildorder could take a list in which order buildings should be built and then place them on the map near the base. This makes macro management a lot easier, but will remove important tactical decisions about where on the map to place buildings in the game. This would also give undesired behaviour because a user could want to build a building near the enemy base, which would not be supported by this functionality. Adapting the strategy to the opponent would also prove to be difficult with this functionality.

• **Kiting** (B.3), this is a form of micro-management. Despite it being a very useful function where every bot will benefit from, the environment should not have an abstraction for this. It is not very hard to implement and thus students can implement this in the MAS-project to get an edge over students who do not implement this.

**4.5. Conclusion**

The focus of the connector should be to accommodate for the 10 week MAS-project. This is not the only education that this system can offer, it could also be used in an honours programme. The abstraction level that will be used in the connector is stated in section 4.4.2.
This chapter presents our research on the environment (StarCraft). The research was used to examine what is needed to create a StarCraft environment and what features it should have. This was done by searching and comparing existing frameworks and bots. The official StarCraft AI community was very helpful and gave a lot of insight in what is needed to create a working MAS.

To create an environment for StarCraft, there needs to be a way to communicate with the game. This section answers the following two questions:

1. *Is there an existing API that can be used to make a connection to the game or should a new API be created?*

2. *What features can be found in existing bots that the connector should support?*

The answer to the second question will be used to create a list of features that have to be supported by the environment. The MAS should be able to do all of this features using the percepts and actions in the connector.

### 5.1. An API for StarCraft

There is a large community that is creating AI for StarCraft. Therefore there are already APIs available to use with StarCraft. This section will list the options available and will explain the API choice that is made. There are a few options available that can be used for this project. These are not the only APIs that are available, but because GOAL runs on Java, the APIs using another programming language are omitted as they cannot be used in this project. The APIs for Java are all wrappers of BWAPI [7]. BWAPI is an API, written in C++, that is used to communicate with the game. Creating an API like BWAPI that directly reads and writes to the memory of the game is not feasible in the scope and time of this project, therefore BWAPI is used. BWAPI has a large community behind it and is properly tested, so there will be no need to go through this process again. There are two APIs written in Java, these are listed below together with a third option.

1. **JNIBWAPI** [11]
2. **BWAPIMirror** [8]
3. **Create a connection to BWAPI**

JNIBWAPI and BWAPIMirror are both wrappers of BWAPI, thus offering the same functionality that BWAPI does, but in Java. Both these wrappers have been used and tested many times by the StarCraft community. Creating a connection directly to BWAPI is therefore not necessary, as this would mean recreating existing software. This leaves BWAPIMirror and JNIBWAPI, both of these API’s are wrappers for BWAPI and offer more or less the same functionality. JNIBWAPI runs on a 64 bit architectures as well, while BWAPIMirror does not. GOAL uses a 64 bit architecture and will therefore not be able to connect to the game with BWAPIMirror, as BWAPIMirror cannot run on a 64 bit architecture. For this reason, JNIBWAPI was chosen as the API to communicate with StarCraft.
5.2. Existing EIS connector

EIS is an interface that has been developed to facilitate connecting software agents to environments. To connect to an environment, GOAL uses this interface. The connector therefore has to implement this interface. EIS creates entities and GOAL will add agents to these entities. When an agent requests percepts, the entity in EIS will provide the agent with the percepts. Percepts are data containers which hold parameters. These parameters are then send to Prolog [16] through GOAL and allow the agent to understand the percept and reason about it.

There is an implementation of the EIS-interface using JNIBWAPI made by Andreas Schmidt Jensen [5]. While this connector is not nearly finished and contains bugs that have to be addressed, it is a good starting point. This connector has been used as a starting point for this project.

5.3. Existing bots

In order to determine the functionality a GOAL MAS should have, we have examined one of the best existing bots from the SSCAIT for each of the three races. These bots are some of the best performing bots in the SSCAIT [17] and are thereby the best representatives of what a MAS should be able to do, as a MAS written in GOAL should be able to perform at the same level. For the Terran race, the ICELab bot from the Intelligent Computer Entertainment Lab of the Ritsumeikan University was examined. This bot is very advanced in adaptive strategy decisions and micro management and also has the most wins of the whole tournament. For the Protoss race the bot of Andrew Smit was examined. This bot has also been in the tournament for quite some time, but has the second most wins of the Protoss bots in the tournament. This bot was chosen because the bot with the most wins only uses a single tactic which is not enough to determine the functionality that a GOAL bot should have. Andrew Smit's bot has lots of adaptive strategies depending on the situation and is therefore an excellent candidate for examination. For the Zerg race the bot of Marian Devecka was examined. This bot also has the most wins of all the Zerg bots and has by far the best win-loss ratio. This bot was recently updated and has lots of interesting micro strategies.

<table>
<thead>
<tr>
<th>Table 5.1: Examined bots</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terran bot</td>
</tr>
<tr>
<td>Protoss bot</td>
</tr>
<tr>
<td>Zerg bot</td>
</tr>
</tbody>
</table>

At the end of this section a list of properties is derived. Users of the environment should be able to create a MAS that has all of these properties.

5.3.1. Protoss bot by Andrew Smit

This bot makes use of scouting (B.5) in order to adapt its strategy depending on the units it perceives. It also makes use of scouting in order to prevent any proxy (B.6) strategy from the opponent. Those are strategies where one starts creating offensive structures just outside the vision range of the enemy base to quickly overwhelm the enemy with an unexpected fast attack. The bot has a different strategy for each race. When facing a Terran opponent the bot relies mostly on kiting (B.3) the Terran units, when facing an Zerg opponent the bot focuses on rushing the opponent based on the advantage of the starter units of the Protoss race in comparison of the Zerg race. When facing a Protoss bot, the bot does not immediately attack. Instead it first scouts what the opponent has and whether it is safe to attack. This bot has hard-coded that it will always attack at game time 6:30, no matter what kind of units the opponent possess.

5.3.2. Zerg bot by Marian Devecka

This bot makes use of several strategies depending on the situation. When playing on larger maps, this bot starts expanding very soon in the game. While setting up a stable economy, the bot produces some early units and waits just outside the enemy base. Every now and then the bot scours the early units to the enemy base to scout (B.5) the opponent's units and structures. Depending on the information retrieved by scouting, the bot decides whether to attack or not. When the bot has made several expansions and has a large enough army it starts attacking. When the bot faces a Terran opponent, it will make mostly use of Mutalisks. These are very mobile flying units which can attack ground and air units. The bot tries to clump up all the units as much as possible, quickly kill units one by one and fly away before the opponent has the
opportunity to return any damage. This is a very good example of making use of micro (B.1). Using micro efficiently works very well against most Terran units. Against a Protoss opponent the bot does basically the same; it heavily relies on Mutalisks, because these units can deal a lot of damage and when their army is large enough they can take on most Protoss’ armies very well. When the bot faces a Zerg opponent it uses most of the time a rush strategy, where it makes attack units as fast as possible to overwhelm the opponent before it has a proper army. When the bot fights the opponent’s early units it evaluates how many the opponent has and how many the bot has. If the opponent has more early units, the bot walks away with its units until reinforcements arrive. If the bot has more units from this point it starts attacking again because its chance to win has grown by now.

5.3.3. Terran bot by ICELab
This bot mainly focuses on macro and defending in the beginning phase of the game. When it faces a Terran or Zerg opponent it heavily relies on the siege tank unit, which is an unit with a very large shooting distance. It sets up a defensive posture, the opponent will have a hard time fighting against this posture. With a large enough army, the bot finally moves out with its units and starts attacking the enemy very slowly with siege tanks. It reinforces its army with other units and makes proper use of micro. When facing a Protoss opponent the bot focuses more on taking control of the map. It lies down mines at every base and tries to pressure its opponent. Compared to the bots discussed before, this bot relies more on winning the game in a strong defense than winning the game by quickly overwhelming the opponent with a fast attack.

5.3.4. Conclusion
There are multiple API’s that can be used to make a connection to the game. For this project, JNIBWAPI will be used to connect to the game because it is a Java API and it runs on 64 bit machines.

Most of the inspected bots have a different kind of strategy depending on the scouted units/structures and the map they are playing in. All bots are able to properly micro against all types of units. The bots are able to properly macro depending on the strategy they use and construct the right structures on the right places in their base. The GOAL-driven bot should at least be able to imitate this kind of behaviour or even more advanced behaviour. EISbot for instance is a bot that competes in the SSCAIT and has nothing to do with EIS despite the name. EISbot was able to win from 48% of the human players on a competitive StarCraft Ladder called International Cyber Cup, where players are ranked based on their performance. The bot was able to beat human players by quick decision making in specific situations [3]. With this information the following summary can be given about the properties the GOAL bots should be able to have.

• properly collect resources by sending worker units to gather minerals and gas to create buildings.
• detect what the opponent is doing by making use of scouting (B.5), and take action based on the information collected.
• react on any information, by adapting the strategy that the bot employs.
• detect all important properties of the map like size and the chokepoints of the map which can be used as good defensive places and base locations for expanding and finding the enemy.
• properly micro (B.1) its units by walking away with low health or consistently keeping a distance from an enemy with a lower attack range like kiting (B.3). Also splitting (B.4) its units where every unit keeps a distance from each other to prevent any shared damage between them like splash damage (B.7).
• properly macro (B.2) on every map by keeping up with the enemy units and expanding to other bases to increase the income of the resources.
• place its structures in such a way that it has tactical advantage over its opponent, for example by placing them near a chokepoint or near the enemy base.
Part II

Design
The connector is the bridge between GOAL and StarCraft. To establish the connection to the game, JNIBWAPI is used. The reason for this choice has been discussed in section 5.1. The high-level connector architecture is discussed in section 6.1. For the connection to GOAL, the connector will implement the Environment Interface Standard (EIS) [9]. The connection with EIS is explained in section 6.1.1. The connection with the game is explained in section 6.1.2. Finally, the design of the connector is presented in section 6.2.

6.1. Architecture

The purpose of the connector is to connect agents via EIS to StarCraft using JNIBWAPI. These components are used to create a pipeline as illustrated in figure 6.1. The figure shows that the connector connects EIS to JNIBWAPI. In this section we discuss the functionality that is provided by the EIS and JNIBWAPI components that is used for implementing the connector. The functionality that we have implemented in the connector is discussed in Chapter 9.

6.1.1. EIS

A connection from StarCraft to GOAL can be made by implementing EIS, which was briefly explained in section 5.2. In addition of what was discussed earlier, EIS handles the creation of entities. GOAL connects to EIS and creates an agent when an entity is created in EIS. GOAL also sends an event on every logic cycle of an agent to request percepts from EIS, which the EIS implementation has to provide. The connector implements the EIS-interface and therefore the connector has to implement the following functionalities from EIS:

1. **Add entities.** When a unit is created in StarCraft, an entity has to be added in EIS. GOAL will see the entity and add an agent.

2. **Delete entities.** When a unit is destroyed in StarCraft, the entity associated with that unit needs to be removed. GOAL will then also delete the agent.

3. **Send percepts.** When a GOAL agent requests percepts, the percepts for the entity associated with that agent should be provided to the agent.

4. **Execute actions.** When a GOAL agent requests execution of an action in the game, the action needs to be translated and executed inside the game.

To implement these functionalities, four interface methods from EIS need to be implemented. These methods are respectively:

![Figure 6.1: The outline of the system.](image)

---

17
1. `addEntity(String entity)`: this method is called when a unit is created. The name of the unit can be passed via the arguments. The name of the unit is unique and this is important because agents need to be distinguishable.

2. `deleteEntity(String entity)`: this method removes an agent. This method is why it is important to have unique names for agents, as an agent that has to be deleted must be specified.

3. `getAllPerceptsFromEntity(String entity)`: EIS will call this method when an agent requests percepts. This method should return a list of percepts. The percepts that will be included in the connector have been discussed in section 7.1.5.

4. `performEntityAction(String entity, Action action)`: when an agent executes an action, this method is called. The connector has to execute these actions in the game. The actions that the environment should support are listed in section 7.2.3. The connector should throw an exception with a clear message indicating the problem when an invalid action is called by an agent.

These are the four methods that need to be implemented to create a working agent system. This sounds trivial, but implementing EIS is not straightforward. Handling the percepts and actions while dealing with concurrency from StarCraft and EIS takes a lot of thought. StarCraft has a lot of units that all function differently. The difference between units has to be generalized in the connector, creating a challenge to retain speed and simplicity inside the connector.

### 6.1.2. JNIBWAPI

To connect StarCraft with EIS, JNIBWAPI is used in the connector. This API has many functions, and it is not feasible to discuss them all. This chapter highlights only the functions that are the most important to connect EIS to StarCraft. Individual actions and their appropriate functions will not be discussed.

JNIBWAPI fires events when something happens in the game. These events can be used to tell EIS to add an entity or remove one and to perform actions. The events that are most important are:

1. `matchFrame()`, this function is called every frame in StarCraft. This function is used to execute actions that have been called by agents.

2. `unitComplete; unitMorph()`, when one of these two functions is executed, a new unit was added to the game and EIS should create an entity for this unit.

3. `unitDestroy()`, when a unit is destroyed, this method is called. EIS should delete the entity that was attached to this unit.

4. `matchEnd(winner)`, when the game ends, all the data belonging to the game should be removed and all entities should be terminated.
6.2. Design

Simple pipelining architecture is used for the connector. The main complication in the design of this pipeline concerns the buffering that is needed of data that is exchanged between GOAL and StarCraft, i.e. the percepts and actions.

As can be seen from the picture, there are two methods that need a buffer to work correctly. These methods are `getAllPerceptsFromEntity` and `performEntityAction` concern respectively the percepts and actions. The percepts have to be buffered to increase the speed of the agents as explained in section 7.1.3. The actions have to be executed inside the `matchFrame` method. This is a constraint set by JNIBWAPI, therefore there has to be a buffer as the agent will not always execute the action in the `matchFrame` method.

Inside the connector there has to be more functionality, such as checking if an action is valid, calculating and storing percepts per agent and so on. This functionality is discussed in chapter 9. EIS and JNIBWAPI both use their own threads. That means that while implementing the connector, care needs to be taken to keep everything thread-safe. JNIBWAPI puts a lot of constraints on methods that have to be called from the JNIBWAPI thread. For the speed of the system it would be optimal to call the methods from the buffer thread, thus careful thought is necessary to minimize the methods called from the JNIBWAPI thread.
Every unit in the game, this includes buildings, is to be an agent. The reasoning behind this can be found in section 3.1.2. An agent is a computer program that can make decisions, and generally has certain goals that it wants to achieve. Agents try to use logic to reach those goals. [4]

The connector that is made in this project will create an agent for every unit in the game, including buildings. That means that there can be many autonomous entities present at the same time. This chapter describes the design decisions made for these entities. All these agents have to reason about something to achieve their goals, reasoning is done on the basis of percepts, this will be explained in section 7.1. To achieve their goals, agents can perform actions. Actions will be explained in section 7.2.

**7.1. Percepts**
Percepts are a result of what the agent perceives. The agent should be able perceive all information in the game that is relevant to decision making. This includes possible locations to build a new structure, enemies, etc.

**7.1.1. Types of percepts**
EIS differentiates between three types of percepts. The StarCraft connector can therefore use these three percept types as well. The percept types are:

- **Send always.** When a percept has this type, the agent will perceive this every time it asks for percepts.
- **Send once.** This type of percept is only perceived once, when this is perceived it will not be perceived again. This can be used to send static data about the game when the agent is created. Static data is data that will never change.
- **Send on change.** A percept of this type will only be sent if the percept changes. This kind of percept can be used for data that have a unique identifier, as it is important to know which percept has changed in the case that there are multiple percepts with the same name.

**7.1.2. Factors that influence the percept structure**
The percept structure displays the percepts that have to be present in the connector. Choosing the percept structure is no easy task. There are many factors that could influence the percept structure, ranging from usability to processor intensity. The factors that were used to create the structure used in this project are:

- **Usability:** Because there can be a lot of autonomous units in the game at the same time, an important goal of the project is the usability of the connector. Simply sending every piece of data as it is perceived in the game would not be feasible as it would create a large information base that would take a long time to learn for students. Therefore, the percepts should be kept as simple as possible.
- **Speed:** To ensure that the system runs fast enough to create responsive agents, even when there are a lot of agents present, the amount of percepts has to be limited. If a lot of percepts are sent, every agent has to evaluate a large amount of data. This will give a massive load on the CPU and would render the agents unresponsive.
• **Generalisation:** When a user creates a MAS for one race, he/she should also be able to create a MAS for another race without coping with a new set of percepts. Thus the percepts should be generalized so that it can be used for every race where possible.

• **Completeness:** The agents should be able to cope with every situation. This means that the percepts should enable the user to perceive every scenario that can manifest itself in StarCraft.

These four factors are used to design a percept structure for StarCraft, which can be found in section 7.1.5.

### 7.1.3. Optimisation

Perceiving everything for each agent is a very CPU intensive process when the number of agents is large. Therefore, perceiving should be optimized as much as possible. Optimizing the percept structure can be done in a number of ways. One way to optimize the mechanism for perfection is to minimize the percepts that are sent to each agent. Minimizing the percepts is done in three ways:

1. **Send less percepts.** This is straightforward. However, care needs to be taken to keep the completeness of the system intact. When removing percepts, the data that these percepts hold should be contained in other percepts as that reduces the amount of percepts that will be sent. Making percepts larger while decreasing the number of percepts will speed up the system.

2. **Change the type of percepts.** Not every percept has to be send always. Changing the type of the percept to another type (section 7.1.1) will result in the system having to send less percepts.

3. **Sending specific percepts per agent.** Attacking units do not have to know where they can build a structure, therefore the percepts can be specific per unit so that there are no percepts that are inapplicable for the unit.

Another way to optimize is to buffer the percepts of the agents in a different thread. This means that the agents do not have to calculate what they perceive, but get it returned immediately. This could be done by storing all the units and calculate their data on a time interval. When the agent requests percepts the list of percepts that was calculated can be returned. Because the agent does not have to calculate this information, it will run a lot faster. Buffering the percepts also means that the data that is the same for multiple agents in the MAS only has to be calculated once.

### 7.1.4. Problems

The knowledge representation language used by GOAL is called ‘Prolog’ [16]. Prolog is a logic programming language. Prolog offers a lot of interesting features, but because of the amount of code it has to evaluate it is the bottleneck for GOAL when considering speed. Prolog evaluates code for each agent and does this every cycle, this results in a vast amount of code that is evaluated by Prolog. Because StarCraft can host so many agents, the number of queries that Prolog is asked to evaluate is a serious issue. To counteract this, Prolog should not have a large database to reason about. This is done by minimizing the amount of percepts as described in section 7.1.3, this speeds up the time Prolog is evaluating code per unit and thus speeds up the time that Prolog will take to run the code for every unit.

Another problem is buffering of percepts. Buffering percepts is an optimization that has been explained in section 7.1.3. This poses a problem as there is some functionality that can only be loaded via the thread of JNI BWAPI, as loading this from other threads causes exceptions. These exceptions happen in the BWAPI [7] and can therefore not be handled by Java. Therefore, the percepts that cause the exceptions have to be buffered from the JNI BWAPI thread. This is not optimal, as it slows down the game, but it cannot be done another way. Only the functions that yield the exception will be buffered from the game thread, so that the game thread will have a minimal load from computations.

These problems both have an impact on the speed of the system. Therefore, optimizing the solutions for these problems is vital. The problems in their current form with the solutions described will not cause a large performance issue in normal games, but limits the maximum amount of agents that the system can handle. Because normal games in StarCraft rarely tend to have enough units to make this slowdown visible, these solutions are sufficient.
The final problem that was encountered during the design of the percept structure is the problem with build locations. The idea is to have a percept named `constructionSite`, which holds the locations where buildings can be placed. However, one of the races (Protoss), needs another percept than the other races because they can place certain buildings in spots where they cannot place other buildings. There is no way to solve this without diverging the percepts for the races. Thus the solution should diverge the percepts between the races, but only at a minimal level. The solution of choice is to give the `constructionSite` percept a new parameter when playing Protoss, which indicates what kind of building can be built in that particular spot.

### 7.1.5. Percepts for StarCraft

The percept structure in figure 7.1 was designed based on the observations discussed above. As can be seen from the figure, there are a set of perceivers which each hold their own percepts. Units are evaluated on creation and given a set of perceivers that fit the unit, these perceivers calculate the percepts for the unit. This way there will be no calculations for percepts that the unit does not need. Explanation about the percepts can be found in the environment manual which is included in Appendix D.

### 7.2. Actions

To achieve its goals, an agent should perform actions. Actions are used to create agent behaviour. An action to move to a location will give the agent a move behaviour, where it will move to a location if possible. This section will explain the design of the actions.

#### 7.2.1. Considerations

Certain considerations have to be taken into account when designing a set of actions. The important considerations in this project are:

- **Usability**: To ensure that the system will be usable, the actions have to be simple. The name of the action should clearly indicate what the action does. The parameters of the actions should also be minimized, as a general rule, actions should never have more than three parameters.

- **Completeness**: StarCraft has a lot of different functionalities, for example the load action for bunkers and the siege action for siege tanks. These functionalities have to be present in the connector. There cannot be a functionality that cannot be done because that will have a negative impact on the user experience. Care needs to be taken that the functionalities that can be grouped in one action are grouped together to avoid redundancy. Minimizing the amount of actions as much as possible has a high priority as the simplicity otherwise decreases.

These two considerations have been made to create actions that can be used to add behaviour to the agent. The set of actions can be found in section 7.2.3.
7.2.2. Problems
To make the system robust and not error prone, actions have to be checked for input. User input is prone to errors and when an action fails it is useful to send a failure message that can help the user.

For the Terran race it is difficult to send failure messages. Some of these buildings can ascend from the ground and move like an aircraft. When a move action is called and these buildings are not lifted, the action will have no effect. This problem has been solved by checking the type of the unit and if it is capable to move. For buildings this means that it has to be lifted.

Another problem that was encountered was a problem with positions. Every unit has a position, with X and Y coordinates, and every building has a X,Y position. These X and Y of these two positions did not match, building positions were based on tile positions, with tiles of 32x32 pixels, whereas unit positions were based on pixel positions. Telling a unit to build a building at its own location did therefore not build a building in the desired spot. To solve this problem, all positions were converted to tile positions. The pixel position of a unit did not prove very useful, using tile positions speeds up the system as well as provide a much easier action set where positions do not need to be converted.

‘Attack’ and ‘attack move’ also posed problems. While testing, the agent sometimes executed the action and sometimes ignored it. After some research the problem was found, attack actions can only execute in the game every third frame. A frame from StarCraft is a state of the game, it is easiest to think about a frame as being a snapshot of the game on a particular time. BWAPI can only fire attack actions every third frame. The solution for this problem was to buffer the action and fire it on the third frame.

7.2.3. Set of actions for StarCraft
The following set of actions has been designed.

1. **Attack**, attack a unit.
2. **Attack move**, move to a position and attack everything that is encountered.
3. **Ability**, use an ability.
4. **Ability on target**, use an ability on a target.
5. **Ability on location**, use an ability on a location.
6. **Build**, build a structure.
7. **Build addon**, build an addon, only for Terran buildings.
8. **Gather**, gather resources, only for worker units.
9. **Land**, land on a position, only for Terran buildings.
10. **Lift**, lift from a position, only for Terran buildings.
11. **Load**, order a unit to load into this unit.
12. **Move**, move to a location.
13. **Research**, research a tech.
14. **Set rally point**, set the rally point on a location.
15. **Set rally point to unit**, set the rally point on a unit.
16. **Stop**, stop a unit from doing what he was doing.
17. **Train**, train a unit.
18. **Upgrade**, research an upgrade.

These are all actions present in the StarCraft connector. The syntax and exact meaning of these actions can be found in Appendix D.
7.3. Conclusion
Creating agents for StarCraft using GOAL should both be easy and complete. The agents should be responsive and there should be no noticeable performance drops. Given these constraints, percepts and actions are designed. These percepts and actions focus on usability, completeness and speed. The percepts and actions can be found in Appendix D.
Development tools

Development tools are offered to the user of the system. These tools can speed up the development by offering support to easily create certain scenarios within StarCraft for testing the MAS.

Ideally, the development tools are a standalone program which the user can start independently from the system. This is not an option however, because BWAPI (the c++ part of JNIBWAPI) only allows one connection to an instance of StarCraft. Therefore, the development tools need to be executed from the start-up of the GOAL agent.

This chapter will highlight the important features present in the tools and explain the design decisions made while creating the development tools. To do this, section 8.1 will discuss the features present inside the development tools. After that, section 8.2 will highlight the design choices made in creating the development tools.

8.1. Features

In the project requirements, a list was devised as to what functionality the development tools should have. This list can be found in section 3.5 under 2. Development tools. While developing these tools, additional functionality was added. The additional features are:

• **Show unit details.** While creating the development tools, it was clear that visualizing where a unit was moving made debugging agents more easy. In addition, showing building information that visualizes that a unit is being trained or an upgrade is being done removes the need to click on each building to see whether they are working, which also facilitates development.

• **Show agent count.** This feature was added because of the performance tests were required. Giving the user an actual count of initialized agents can help the user understand what units are not properly initialized or are bumped out of the game by using actions that cannot be executed.

Showing unit details and the agent count together with the features discussed before in section 3.5 (showing mapdata, speeding up the game and some cheats) are bundled in the development tools. Given these features, users will have an easier time prototyping their MAS.

In addition to all these functional requirements, the development tools also have some non-functional requirements. These requirements are:

• **Simplicity.** The development tools should be easy to grasp and be usable without reading any documentation. This will make it easy for the user to use the development tools.

• **Robustness.** The user should not be able to break or crash the game by using the development tools. This means that the development tools should be completely thread-safe as StarCraft and JNIBWAPI can easily crash.

8.2. Design

The development tools is a set of buttons, each providing a feature on click. The speed of the game can be changed using a slider. Because the development tools do not have to be fancy, but simple and usable, the
tools are set in a small window with descriptive button names. The user interface is illustrated in figure 8.1.

To show a user which button has been clicked, the button clicked by the user should receive another color. The other color indicates which button is active. This is shown in figure 8.2. Clicking the button again should return the color of the button to its original color.

The agent count is displayed on the game screen at all times while the development tools are active. Figure 8.3 displays the agent counter in-game.

8.2.1. Robustness
The development tools are made robust by buffering the input in the development tools and executing them in the game thread. This works similar to how actions are buffered, which is explained in section 6.2.
8.2. Design

**Figure 8.2:** The development tools window with a clicked button.

**Figure 8.3:** The agent counter in-game.
Part III

Implementation
In this chapter, the implementation of the system is discussed. The system includes the connector and the debugger and is the final software product created in this project. First, the thread structure is explained in section 9.1. After discussing the thread structure, some major problems encountered and their solutions are highlighted in section 9.2. Finally, the full system is shown in section 9.3.

9.1. Thread structure
There are only three threads in the system (apart from the threads that GOAL creates).

- **The environment thread**, this is the thread where EIS runs.
- **The game thread**, JNIBWAPI runs in this thread and communicates with the game. Actions are performed via this thread, as well as drawing actions from the development tools. This thread also buffers the constructionSite percept.
- **The update thread**, this thread updates the percept base and notifies EIS to add entities to the environment. Adding entities happens in this thread after the update cycle.

Three threads does not seem much, but careful thought was necessary to ensure the speed and robustness of the system. Buffering percepts in the game thread is not optimal, as it slows down the game, but without buffering in the game thread the system would not be robust because the system can crash on certain functions. The update thread is important because agents have to evaluate their rules as fast as possible. Calculating percepts in the GOAL threads slows down the evaluation process and therefore the whole system, rendering agents unresponsive.

9.2. Problems and solutions
The main problems encountered in the system were concurrency issues. The major issues encountered are stated in this section.

- **Game slowdown**. At some point in the project, StarCraft began to run very slow. This happened because GOAL agents are greedy and consume as much processing time they can get, pushing the processor load to 100% and ‘suffocating’ the game thread. The first solution was sleeping each agent in their update cycle for 5 milliseconds. While this decreased the processor load and solved the issue of the game running slow, it introduced a new problem; The agents slowed down. For example, when there are 100 agents, there is a total sleep time of 500 milliseconds. Half a second delay is too much as agents will not perceive enemies in time anymore and it becomes impossible to create agents that win the game. We therefore tried another solution, the sleep for the agents was removed and the thread priority of the game thread was maximized. This also did not completely solve the problem. As a final solution, we proposed to the client to change the thread priority of the agents in GOAL to a lower level. With the new thread priorities, the game runs real-time again and the agents evaluate without delay.
• **Calculating percepts.** Calculating percepts happens when agents request their percepts and new agents are added to GOAL. This lead to some agents not properly initializing due to missing percepts on the first agent cycle, this is a problem as the agents will not have all the percepts they need to function. A solution would be to calculate the percepts in the thread of the agent, but that slows down the system as explained in section 9.1. The problem was solved by adding entities to EIS after the update cycle. Agents will initialize a bit slower, but the rest of the system does not suffer from a decrease in performance.

### 9.3. Final system

This section will discuss the final implementation of the system. The system consists of two parts:

- The development tools
- The connector

The connector is the main part of the system, whereas the development tools are an addition. The UML of the full system is included in figure 9.1. The UML has been broken down into subsections, which can be found in appendix G. This appendix will be referred throughout this section to explain parts of the system.

#### 9.3.1. Connector

The connector provides percepts and actions that can be executed inside StarCraft. The system starts at the `init` function of the `StarcraftEnvironmentImpl` class. When initializing, the class `BwapiListener` creates a thread where the JNIBWAPI will run. The JNIBWAPI thread can then call the event functions in the `BwapiListener`.

**Percepts** Percepts are loaded via perceivers. Every unit will have a set of perceivers that are allocated to the unit on creation. The perceiver structure is depicted in figure G.4. The allocation of the perceivers on creation of the unit is done in the class `StarcraftUnitFactory`, which creates a `StarcraftUnit`. The created `StarcraftUnit` is then added to the `Units` class. The relation between the Factory and Unit is depicted in figure G.6. The update thread iterates over every unit currently in the game and calls the function `perceive()` on each of their perceivers. The perceivers return percepts, these percepts are calculated on runtime in the update thread. The UML of the percepts is depicted in figure G.2. Percepts all extend from the same class, this way EIS and GOAL can handle all the percepts without extra logic. The `Game` class then gets updated with the new percepts and checks whether or not the percepts should be sent by using the `PerceptFilter` class from the perceiver. When an entity requests its percepts via EIS, the `Game` class simply returns the percepts for that entity.

**Actions** Actions are initialized at the launch of the program. This is done by the `ActionProvider` class. This is depicted in figure G.1. When a GOAL agent requests the execution of a specific action, the `ActionProvider` returns the specific action class. The methods inside the action class are used to ensure that the action is valid and can be executed. When the action can be executed, it is executed in the `matchFrame()` function of the `BwapiListener`.

#### 9.3.2. Development tools

The implementation of the development tools is displayed in figure G.5. Launching the development tools is optional and can be switched on and off when running a MAS. A Swing window is created in the `DebugWindow` class and the various buttons and sliders are created in the composite classes. To make drawing functionality easily extendable, the draw package has been created. The UML of the draw package can be found in figure G.3. The `IDraw` class has functionality to draw and to toggle whether or not the drawing should occur. When a button is clicked in the development tools window, the toggle function is called, switching the draw state.

### 9.4. Abstraction level

During the creation of the system, the abstractions discussed in section 4.4.2 have been considered. Additionally, some abstractions that were deemed useful while writing agents to test the system were added. This chapter will highlight the design decisions made and list the abstractions present in the system. All the abstractions stated in the Must and Should have paragraphs stated in section 4.4.2 have been implemented in
Figure 9.1: The UML of the system.
the system. The Could have abstractions are not implemented in the final system as they proved to be nice challenges for users while implementing a MAS. The rest of this section will discuss the additional abstractions present in the system and the design choice made when adding these abstractions.

- **Using build positions.** The BWAPI offers pixel positions and build positions. Build positions are tile positions (32x32 pixels). At first the pixel positions were used to denote the position of a unit. This was not optimal because of two reasons; One being that the position of the unit got updated a lot, thus sending a lot of new data to agents every logic cycle. The other problem was that builders could not build at their own location without transforming the coordinates of the unit's own location. To counteract this, all the unit positions were also stored as build positions. This solves both problems as the unit only updates position when walking to a new tile and can build at the same coordinates as the unit believes he is located.

- **Build location percept.** Because of a difference between the races, the build locations have to be adapted for Protoss. We decided not to add a new percept only for Protoss and thus a flag was given to the build location percepts of Protoss.

- **The use action.** The use action can be used for executing upgrade types or tech types in StarCraft. This abstraction has been chosen to reduce the number of actions present in the system.

### 9.5. Multi-agent systems

During the project, a MAS was created for each race. These multi-agent systems were used to test the functionality of the system. Every MAS that was created is able to defeat the built-in AI of StarCraft at least 8 out of 10 times. All these multi-agent systems rely on communication between agents to deploy their strategy. Every MAS that was created is capable of completing the last performance target of the MAS-course. Thus, by handing in a MAS like one of these, a student will have completed the course.

- **Terran MAS**
  
  When creating a MAS for StarCraft, communication is very important. To divide the work between the worker units, an extra agent was added which handled the work division between mineral gathering, gas gathering and building structures. All worker units send their information to the manager. This will allow the manager to divide the tasks and send them back to the worker units. This MAS builds a strong economy which will help in the later stages of the game. The Terran MAS uses a strategy in which it waits for the enemy to attack. When it has fended off this attack the MAS will counterattack directly. This MAS is the most advanced of the three systems as it does not use a simple rush tactic. To implement this behaviour, extensive messaging is required. After implementing messaging at this scale, students will grasp the idea of what a MAS is and will know how to build one.

- **Protoss MAS**
  
  The Protoss MAS uses a rush strategy. This strategy is based on a commonly used build order for fast attacks on small maps. The task management is basically done by the same way as with the Terran bot. Building structures with Protoss is different than with Terran, since most structures have to be powered by a Pylon. Therefore, first a Pylon is constructed followed by two production facilities to quickly create as many units as possible to attack the opponent. When an unit is created it moves to a chokepoint near the enemy base. The units group because a group of units has a better chance of winning a fight than multiple units that attack independently. Thus, each unit waits just outside the enemy base and sends a request to the manager to attack. When the manager has received four or more requests to attack it starts sending messages to all units to attack. Because the units attack as a group, they increase their chance of overrunning the opponent before he can react properly. The MAS generally wins when there are 4 minutes of game time elapsed. If the MAS does not win on the first rush, it will most likely lose. It loses because it cannot easily recover from this strategy when it fails.

- **Zerg MAS**
  
  The Zerg MAS is based on the commonly used strategy called “4 Pool”. This means that a Spawning Pool building will be built when there are only four workers alive. The Spawning Pool building allows the player to create Zerglings. “4 Pool” means that there will be no workers produced, but only Zergling units which allow the MAS to quickly overrun the opponent. Just like with the Protoss strategy the Zergling units gather just outside the enemy base and send a request to attack to the manager. When
the manager has received 6 or more requests to attack, it starts sending messages to all units, telling the units to attack. The idea of this Zergling rush is that the Zerglings attack together, before the opponent has any defences set up. Zerglings are not strong units on their own, so if the opponent does have defences set up the Zerg Mas generally loses.

These multi-agent systems are proof that the system works and that it can produce a MAS. Due to time constraints we have not been able to participate in the SSCAIT ourselves, but we will participate after this project with one of these multi-agent systems.
Part IV

Education
In this section, the educational value of using StarCraft in the MAS-project is discussed. The StarCraft environment should offer a sufficient challenge for the MAS-project. This chapter will discuss what the design of the current MAS-Project discuss how replacing UT3 used now with StarCraft would improve the project without removing any important aspects of the current course.

10.1. The MAS-Project

Each year the first year students of the TU Delft complete the course Logic Based Artificial Intelligence (LBAI) where they learn how to program using the Prolog programming language [12]. Students are taught how to develop a multi-agent system that uses knowledge representation to reason about the environment in which the multi-agent system operates. Using this knowledge, students will develop an army of intelligent agents that control bots in the game StarCraft Broodwar. Besides the implementation of the multi-agent system, students also have to create a technical report in which they explain how their multi-agent program operates and how they have tested it. At the end of the course there is a big competition where the developed MAS programs will compete against each other for an additional bonus point on their grade.

10.2. The learning goals

After completing the MAS-course students will have gained knowledge regarding to the following items [13].

- **Agents and Multi-agents Systems** The student is capable of recognizing concepts as: Agent, multi-agent system, mental state, beliefs, goals, actions, communication and coordination in the given exercises and program provided by the course, and how to use these concepts in a multi-agent program and report.

- **Agent-Environment Interaction** The student is capable of recognizing concepts as: actions and percepts in the given exercises and program provided by the course, and how to use these concepts in a multi-agent program and report.

- **Agent and Multi-Agent Programming** The student is able to develop in a team using the GOAL language. The student is able to apply relevant programming constructs consistently and effectively when writing a multi-agent program that acts in a dynamic real-time environment that requires coordination between agents.

- **Navigation in the environment** The student is capable of analysing whether a search-algorithm is the best solution for a given problem, make a decision out of multiple search-algorithms and knows how to implement these algorithms.

- **Project Skills** The student is able to collaborate on solving complex issues in an efficient and result-oriented manner. The student is able to determine an appropriately balanced task division in a team and react on his or her own contribution to the group process and resulting product.
- **Technical Writing Skills** The student has appropriate skills in writing professional and scientific texts, providing insight into the factors that influence the quality, usefulness and persuasiveness of corporate and scientific texts.

### 10.3. The StarCraft Course

For the last few years, the UT (Unreal Tournament) Connector has been used in the MAS-Project. Students were asked to develop a multi-agent program within a group using the UT Connector. After 10 weeks of development, each student would have gained the knowledge of the learning goals provided in section 10.2. Using the UT connector however, limited the amount of agents that the students had to control. Some students did not feel the need to create a complex strategy as simple strategies worked fine as well. This did not motivate the students to create a real intelligent MAS. Additionally, there were some problems with the UT-Environment regarding the stability of the Environment and the performance of the programmed agents. Because of these problems the main focus of the StarCraft Environment was to prevent these problems and create a stable and fast environment where students can keep improving their multi-agent program over the whole course.

#### 10.3.1. Why StarCraft?

The UT-course has always been a fun and informative course for the students regarding the teaching of multi-agent systems. However, the course also had some minor unresolved issues where students kept complaining about. The first problem of the course was the instability of the environment, which lead to many different run-time errors. To learn from the mistakes that were made in the past the StarCraft connector has been intensively tested for any error it could contain. Also each percept and action has been tested by unit tests and by a GOAL agent to test its functionality. Foreseeing every scenario in the game and testing it is not feasible as there are close to infinite game states. Therefore there is no guarantee that the system will work in every scenario, but the connector has more unit tests than the UT environment and is built to be easily extensible to be able to fix bugs easily. Another problem with the UT course is that the multi-agent systems that have to be developed are rather small. This reduces the need for developing a truly complex MAS coordination strategy. The UT environment was able to run 4 agents where the student were supposed to make them communicate and let them work together. One related issue is that programming 4 agents with a group of 6 people is not optimal. Using the StarCraft Environment there are over 20 different types of agents per race and the students can choose which ones they want to use. All these unit-types can be implemented separately and later be adjusted to let them work together, which makes it easier for the work distribution within a group. Instead of 4 different agents, the StarCraft Environment offers a good performance with 200 running agents. Making use of this many agents and different agent types should also increase the new knowledge and expertise that students gain during the project.

#### 10.3.2. Individual assignment

In the first 2 weeks of the MAS-Project all students have to hand in an individual assignment regarding the StarCraft Environment based on the StarCraft course manual (See appendix E). This will be an exercise where the students have to create a small multi-agent program which consists of some basic functionality. After this assignment each student should have a basic knowledge of StarCraft, the environment and how to develop a MAS using the StarCraft environment. Just like the individual assignment of the UT course, there will be 5 stars which students can achieve by completing the assigned tasks. First the students have to choose which race they want to build a MAS for. This step is very important, since the students will be using this race until the end of the project. After choosing a race, the students has to create a basic MAS to earn stars. The first few stars are basic and easy to achieve, the last few stars are more complicated to achieve. For the fifth star the students are asked to defend their base as long as they can against a given opponent. Only the best students will achieve this fifth star as there is a limited amount of students that will get this star and it is based on performance in the game. The students will be divided in groups based on the amount of stars they got based on their performance in the individual assignment.

#### 10.3.3. Performance targets

After the individual assignment the students will be working together in a group to create a more advanced multi-agent program, also based on the StarCraft course manual (See appendix E). For the UT course students had to develop a MAS which was capable of defeating 3 unreal bots with skill level 3. For each week there is a
description which functionality they should be working on and a performance target which their MAS should be able to beat. The StarCraft course will make use of the same general structure, however with a bit of a different approach to the performance targets. To keep the same amount of workload as the UT course, the first 2 performance targets do not involve defeating an opponent. StarCraft is a game where students can make use of many different strategies. This means that there are also a lot of strategies which one should be able to defend against. Therefore, scenario testing is even more important for the StarCraft environment than it was for the Unreal Environment. The last performance target is set in such a way that it requires the same amount of work that a group of students would have spent on the UT course. However after meeting the last performance target there are still many things that can be done, which makes the course more interesting for students.

10.3.4. The end competition
At the end of the course all students have to hand in their project for evaluation and for the end competition. The groups that end at the top of the competition will receive bonus points. This has always been a fun final activity of the UT course. The problem was that most groups would implement the same kind of strategies which resulted in a game where luck played a significant role in which team wins. Since StarCraft offers a lot more strategies which can be used and has three different races where students can choose from, the matches will probably be more diverse than with UT. To make this competition more interesting for the students, the groups can also represent the TU Delft in the StarCraft AI competition. This is a tournament where students from different universities can participate with their MAS to let it compete in the competition with other bots. These matches are live streamed every day of the year where the best bots will fight each other for the title of best AI of the year.
Part V

Quality
Performance Testing

The performance of the system is vital. If the system does not perform fast enough, agents will be unresponsive and fail to execute actions in time. The performance and robustness of the system are tested in numerous ways. The tests that have been conducted are highlighted in this chapter. The first test that is discussed in section 11.1 is how many agents the system can run while maintaining a decent speed. The second test that is conducted is explained in section 11.2 and tests the behaviour of the agents when the game runs at different speeds. The final test is explained in section 11.3 and determines how long the system can run without slowing down. The specifications of the system that was used to conduct these tests can be found in section 11.4.

11.1. Concurrent Agents
This test will determine how many agents can be ran concurrently on the system without rendering the agents unresponsive. To test this, there has to be a measure for which the agents are called unresponsive. The definition of an unresponsive agent for this system has been defined as follows:

*An agent is called ‘unresponsive’ when it performs less than 10 update cycles each second.*

This means that agents are responsive when their update cycle has a frequency of 10 Hz or more.

11.1.1. Test setup
The test will be executed by setting the game speed to the normal game speed, creating many agents and toggling the zero damage cheat so the agents cannot die. When the average update frequency of the agents drops below 10 Hz, the test will be over. The MAS that will be used for this test has been created for this single purpose. Another MAS might yield different test results. When the MAS is very complex and uses many complicated calculations it will naturally run slower than a simple MAS. The description that is used for this test as follows:

- **Name:** ZergAgentTest
- **Race:** Zerg
- **Description:** This MAS builds as many Zerglings as possible. To enable this MAS to build Zerglings, the resources cheat should be activated.

This MAS is tested on a small, medium and large sized map to determine the difference between map sizes.

11.1.2. Test results
The results of the test can be found in table 11.1. The difference in map size is very minimal. The difference that occurs between map sizes is because of the map percepts. Because the map contains more chokepoints and baselocations, querying will take longer. 10 Hz is reached just after the 100 agent mark, but this is not visually noticable. The agents become visually unresponsive when there are around 170 agents present in
Table 11.1: Concurrent agent speed

<table>
<thead>
<tr>
<th>Mapsize</th>
<th>Agents until 10 Hz</th>
<th>Agents until visual difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>105</td>
<td>175</td>
</tr>
<tr>
<td>Medium</td>
<td>104</td>
<td>173</td>
</tr>
<tr>
<td>Large</td>
<td>102</td>
<td>170</td>
</tr>
</tbody>
</table>

the game. As a game of StarCraft generally does not have that many units, the game will be playable in most cases.

Figure 11.1 shows the profiler results from running an agent until it becomes unresponsive. This shows how much CPU time a function has used. As can be seen from the picture, the function that is by far the most time consuming is `jpl.fli.Prolog.attach_pool_engine[native]()`. This function executes Prolog code. To reduce the time that this function takes in respect to the other functions, the system already uses various optimisation techniques which can be found in section 7.1.3 and 9.2. Nonetheless this function is still the bottleneck of the system.

![Figure 11.1: The CPU profiler results](image)

11.2. Agent behaviour versus game speed

The game can run in different speeds, the development tools offer functionality to do so. However, changing the speed of the game will also change the speed in which agents need to evaluate. Testing at different speeds is therefore vital, as the users of the system should know whether or not this changes the behaviour of the MAS they are creating.

11.2.1. Test setup

Testing behaviour versus game speed is difficult, as the behaviour can vary in very subtle ways. Because there can be unforeseen behaviour, testing this automatically would be very hard. Therefore, the test will be manual. The test that has been devised is as follows:

Multiple MASses are run on low, normal, tournament and high speed. Low speed is game speed 50, normal speed is game speed 30, tournament speed is game speed 20 and high speed is game speed 0. While running the agents, various factors are tested, these are:

1. Does the MAS build structures at the same game times (Low speed is the baseline)?
2. Does the MAS beat the built-in AI at every speed level?
3. Is there any behaviour that does not comply with the scripting that shows at different speed levels?
Table 11.2: Behaviour versus game speed

<table>
<thead>
<tr>
<th>Speed</th>
<th>Build time</th>
<th>Game won</th>
<th>Unwanted behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>-</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Normal</td>
<td>Same</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Tournament</td>
<td>Same</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>High</td>
<td>Not always the same</td>
<td>Most of the time</td>
<td>The agents are sometimes unresponsive</td>
</tr>
</tbody>
</table>

11.2.2. Test results

Results of the test can be found in table 11.2. There was no difference found between different races, therefore the table is generalized over all races. From the table it can be concluded that running the game on high speed will sometimes result in a different outcome. When the game runs that fast, the agents cannot keep up. Agents become unresponsive because the game runs a lot faster because the speed of the agents does not change. At tournament speed, an update frequency of 10 Hz will result in the same outcome of the game as with normal speed, because the update frequency is still faster than the game speed.

N.B. running more agents concurrently does change the outcome of this test, as it renders agents unresponsive. The test was conducted using example agents that execute one strategy. At no point in time were there more than 100 agents in the game, thus the update frequency did not drop below 10 Hz.

11.3. System run-time

Testing the amount of time the system can run is important to ensure that the system is able to run full games of StarCraft. To test this, the system has to be checked on memory leaks and whether data is efficiently stored as well as deleted.

11.3.1. Test setup

This test will be conducted by running a MAS that creates 100 agents and then stays idle. Memory profiling will be used to detect leaks, whereas the speed of the game will be tested by using the framerate of the game.

11.3.2. Test results

The program can run without problems for the whole time of the game. One small memory leak has been detected, but the memory leak is caused by Prolog. As this is not part of the system and does not pose problems when running a game, this does not need a fix.

11.4. System and specifications

The tests were run using a system with the following specifications:

- Intel Core i5-4200U 1.6 GHz with Turbo Boost up to 2.6 GHz
- NVIDIA Geforce GT 750M with 4 GB Dedicated VRAM
- 8 GB DDR3 L Memory
- Windows 10 64-bit

Running these tests on a similar system should yield the same results.
In this chapter the feedback of SIG is discussed and the improvements made to the code after the first SIG code review are highlighted.

12.1. First SIG feedback
The first feedback of SIG was received at June 3, 2016. This section will explain the feedback in English, the raw feedback can be found in Appendix H.2.

The system scores almost 4.4 out of 5 on the SIG scale. This is above average. The highest score possible was not achieved because of small duplications in the code. The test coverage of the code is promising and SIG expects the test coverage to grow even further when new functionality gets added. The test coverage was at 80% at this point in time.

12.2. Improvements
To improve the code even further, more tests were written. Every new piece of code was heavily tested where possible. The final system has tests for every part of code except for parts of the Development Tools as that is a GUI. The total test coverage after the improvements is 84%. The duplication that was present in the code was removed by creating abstract classes with the duplicated methods already implemented. With these improvements the goal is to get the maximum score of 5 on the next code evaluation.

12.3. Second SIG feedback
The second feedback of SIG was received at June 28, 2016. This section will explain the feedback in English, the raw feedback can be found in Appendix H.2.

The system has the same score as last time: 4.4 out of 5 on the SIG scale. SIG has recognized that there were indeed improvements for maintainability and that all the feedback was processed. However the score has not grown due the fact that the code volume has grown, which makes it harder to maintain a quality code structure and also because the last score was already very high. SIG has recognized that the amount of tests has grown and that the ratio code volume / production code is still fine.
Part VI

Conclusion and recommendations
Within ten weeks, the team completed a connector to connect GOAL to StarCraft. The connector can be used to connect a Multi-Agent System (MAS) to StarCraft. This connector is planned to be used in the MAS-project course for first year students in the Computer Science curriculum. This means that the course also has to have new manuals. Thus, in addition to the connector and its documentation, a project manual had to be created for the MAS-project.

One of the remarkable features is the amount of agents that the system can run. GOAL had not yet been tested using such a large amount of agents, but after optimizing as much as possible the system ran better than expected.

To be able to finish everything in 10 weeks, the team used iterative development. Sprints of one week were used to quickly create the connector and its documentation. Features that needed to be added were planned ahead and careful thought was necessary to ensure that the system would have the functionality and speed that it needs.

Quality of the code was also one of the main criteria that the team deemed important. To ensure the quality of the code the team used various static analysis tools, as well as code reviews with GitHub pull requests.

During the ten weeks, the team encountered a lot of unexpected challenges. Despite the challenges that were encountered, the team managed to finish all the deliverables for the client.

At the end of the project, there were six deliverables which have all been completed in time. These deliverables were:

- The connector
- A course manual
- Development tools to make the creation of a MAS as easy as possible
- Example MASses for all three races in StarCraft
- Tests to ensure the robustness and performance of the system
- An environment manual where all the functionality of the connector is listed

With these deliverables all being completed, the project can definitely be called a success.
Future work and Recommendations

14.1. Future work
During the development, we encountered a few issues which were translated into recommendations for the client. The issues are listed below:

- **Performance.** Because GOAL is not used for a project of this scale before, performance in GOAL became an issue. There are a lot of optimisations that can be done in a connector, but these are still limited. With so many agents it might be interesting to create a shared belief base between multiple agents to lower the insertion time of beliefs for every agent and reduce the memory usage of the system. It might also prove useful, when implementing a shared belief base, to buffer results from querying to the shared belief base. Because agents generally have the same logic if they have the same type, this could speed up the system.

- **Percept types.** The client would like to delete the send on change with negation from GOAL. However, the team thought that this type of percept was very useful and thought that it would be useful to have a solid replacement for these kinds of percepts.

- **GOAL timeout.** GOAL will terminate if there is no agent present after the first 10 seconds of the program launch. This means that the game has to be started within 10 seconds. On slower systems this poses problems as the game does not start that fast. This can be overcome by starting the game before starting GOAL. Starting the game before GOAL does however mean that the user cannot use the autostart functionality of the game. If the timeout can be adapted inside GOAL this problem would be solved.

When these functionalities would be added to GOAL, the connector should be adapted to reflect these changed and be as fast as possible. We believe that this will boost the performance of the system.

Another issue is that BWAPI currently does not have an option to use the auto menu to load save games. Save games can be loaded manually, but for testing purposes it would be interesting to load save games dynamically. If this functionality is added to BWAPI, it would be useful to add it to the connector.

14.2. Recommendations
For the tournament we would advise to run the games at at least two computers. Running two multi-agent systems on the same system is possible, but not recommended as it decreases the performance of both the systems. Running on two different systems via the internet or a LAN network will yield the best results.

Using the save game functionality in StarCraft while testing the MAS makes it easy to test certain scenario’s. After playing the game as a human and invoking a certain scenario, a user can save the game and test his/her MAS on this specific scenario. Using this method, one can test scenario’s that do not happen often more easily.
Part VII

Evaluation and reflection
At the end of the project, all the deliverables have been completed. For the main part, the project went smoothly. However, some issues were encountered. Concurrency issues happened a lot due to the nature of the API. To increase the performance of the system, an update thread was created that calculated information. The creation of this thread gave unforeseen issues which had to be resolved. Adding the update thread was a big improvement, but we should have calculated more time to do so as it was a larger job than originally thought.

Starting with the EIS implementation of Andreas Schmidt Jensen helped us to prototype faster, but also slowed us down. The code contained a lot of bugs and was prone to many errors. Fixing these errors was hard and it also took a while to learn about the code.

Overall, the project went very well. Apart from some minor code issues the work got done quickly and efficiently. Test driven development and working with Scrum ensured that we could advance quickly in the project and deliver the system with its manuals in time.

The best part of this project was the teamwork. There were frequent discussions about code quality and both team members had their own particular skill set that complemented the skill set of the other person. The discussions ensured that every part of the code was well structured and carefully examined before release. The agreements made in the first week of the project were upheld which made the teamwork easy. Because of this teamwork the project was fun to work on and we could motivate each other to push hard and finish all deliverables in time.

The worst part of this project were the frequent concurrency bugs. The bugs happened in the BWAPI and could therefore not be handled in Java. Because of the time limitations we had for completing this project, we could not afford to learn the code of the BWAPI to handle the bugs there. Fortunately, the bugs could be avoided by carefully thinking about methods and when to call them. By sacrificing some performance of the system, the concurrency bugs could be resolved. In the end we have managed to remove all concurrency bugs by sacrificing a very small amount of performance. This problem can be avoided in future projects by using an API from someone that has enough time to look into our problems. For this project, this was the only API available and therefore the problem could not be avoided.

We are very pleased with the final deliverables of this project.
16.1. Danny Plenge

I started with this project because I got invited by Harm. I have always enjoyed the StarCraft game and I also really enjoyed the MAS-Project, so it seemed as a good opportunity to start working on a really fun project. I did not have any experience with EIS, so it helped a lot that Harm had already worked with EIS before during his context project. It took some time before working with the connector that Andreas Schmidt Jensen had already created went smoothly. At the end of the project nearly everything was rewritten or removed, however the connector of Andreas Schmidt Jensen did give us a very nice start.

Creating new MASs was also a fun way to test all the functionality we added to the environment, however testing functionality which only happens in the later stages of the game seemed to be tedious task. It took a lot of time to do this for all the three races, however at the end it provided us with multiple MASs that were capable of defeating the in-game AI.

I really enjoyed working on the project and I am really satisfied with the results. Doing this project with Harm was a very fun experience and I am really grateful that he invited me to this project.

16.2. Harm Griffioen

Before the project started, I had basic understanding of EIS. I developed this understanding during the context project in which I also created a connector. Understanding EIS helped a lot during the course of this project, as we did not have to research EIS. I have spent most of my time programming the connector and not much time programming a MAS. The only MAS that I have created is a MAS where I could test new functionalities from the connector. We used the project of Andreas Schmidt Jensen as a start [5], in retrospect I would do this differently. The state of the connector that we used was not good and it took a lot of refactoring and bug-fixing to get the connector at a quality level that we wanted the connector to have. However, if we would have started without the connector of Andreas Schmidt Jensen we would not have prototyped the system as fast as we did in this project.

I usually work very individualistic, but from this project I have learned that people have different skill sets and that combining these skill sets is very useful. Even though it is the first time that we work together, Danny and I have worked together without issues and our skill sets complemented each other very well. I am very pleased with the course of this project and I have really enjoyed working on this together with Danny.
Part VIII

Appendix
A

Project management

In the first week of the project, a planning was devised which the client and coach approved. The planning can be found in this chapter.

A.1. Planning

Table A.1: Planning

<table>
<thead>
<tr>
<th>Task</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. API of choice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. StarCraft competition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Debugger</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Existing bots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Study goals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Terminology</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Implementing all races</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Testing and bugfixing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1 Performance testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2 Testing robustness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3 JUnit testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing GOAL code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Zerg MAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Terran MAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Protoss MAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Competition MAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Work on SIG feedback</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creating manuals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluate with client and coach</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.1. Micro
Micro is the ability to control your units individually, in order to make up for pathing or otherwise imperfect AI. For example, controlling two weak units with a long attack range killing 1 strong unit without a long attack range is considered “Micro”. The general theory of micro is to keep as many units alive as possible. For example it is better to have four half-dead Dragoons after a battle, rather than to have two Dragoons at full health and two dead ones [21].

B.2. Macro
Macro is your ability to produce units, and keep all of your production buildings busy. Generally, the player with the better macro will have the larger army. The other element of macro is your ability to expand at the appropriate times to keep your production of units flowing. A good macro player is able to keep increasing his or her production capability while having the resources to support it [21].

B.3. Kiting
Kiting (to kite) is to move units around to make the enemy chase them and thus not be able to attack as much, or not at all. This is often used by ranged units against melee units in combination with move-shoot micro, but can also be used to let other units standing still deal damage, or to buy time for example in an early game rush [23].

B.4. Splitting
Splitting refers to the skill of sending your first four workers to different mineral patches. A perfect split is getting all four workers on different patches on the first run. In addition, it involves rallying your workers to go towards your mineral lines so they get there in the quickest possible time. Although having a perfect split is ideal, it generally has no effect on the outcome of the match. However, splitting three of the four initial workers properly is important, because it allows you to start building your sixth worker faster [20].

B.5. Scouting
Scouting is the act of revealing (either with a unit’s line of sight or by using Comsat) remote areas of the map to gain information about your opponent [22].

B.6. Proxy
Proxy strategies are strategies where one starts creating offensive structures like photon cannons or a barracks just outside the scout range of the enemy base to quickly overwhelm the enemy with an unexpected fast attack [18].
B.7. Splash Damage

Splash damage is damage taken by units surrounding the point of impact of an attack. Splash damage never stacks with the original damage with one exception: the Firebat. There are four types of splash damage in StarCraft, radial, line, spell, and special. The properties of splash damage directly effect normal damage, so the splash damage is dealt as a percentage of the original damage before armor and damage type are applied. Therefore, the formula for all splash damage is: \[(\text{Original Damage} \times \text{Splash Modifier}) - \text{Armor}\] \times \text{Damage Type} \]. The splash modifier is generally determined by the distance away from the point of impact, but there are some special cases where the splash modifier is 1, meaning that the splash damage is exactly the same as the original damage [19].
C

StarCraft

C.1. What is StarCraft

StarCraft is a real-time strategy (RTS) game, like chess is to board games, with a lot of players and lots of professional competitions. The game has three different but very well balanced teams, or so called “races”, allowing for varied strategies and tactics without any dominant strategy, and requires both strategic and tactical decision-making roughly equally matchups [3]. StarCraft has received over 50 game industry awards, including over 20 “game of the year” awards. Some professional players have reached celebrity status, and prize money for tournaments total in the millions of dollars annually. The game revolves around players collecting resources to construct a base, upgrade their militaries, and ultimately conquer opponents. StarCraft Brood War’s gameplay remains fundamentally unchanged from that of StarCraft, although it made small alterations to unit costs and some abilities. These changes make rushing tactics a factor that gained some criticism in the original StarCraft less practical. The single-player campaign has an increased difficulty; missions are no longer entirely linear, and a greater focus on strategy is needed to complete missions. In addition, the game’s Artificial Intelligence (AI) has been augmented so that AI-controlled players are more intelligent and tend to use tactics more effectively. StarCraft has always been very popular by amateur and professional players in South Korea. Because of the creation of the Brood War Application Programming Interface (BWAPI), the popularity of the game significantly rose by AI developers. By observing professional players, AI developers are trying to make bots which mimic the behaviour of the professional players to win in the AI tournaments. Because StarCraft is a game where anything can happen lots of different strategies have to be implemented which makes StarCraft AI complex and interesting.

C.2. Real-Time Strategy

In an RTS (Real-Time Strategy) game, a player indirectly controls many units and structures by issuing orders from an overhead perspective in “real-time” in order to gather resources, build an infrastructure and an army, and destroy the opposing player’s forces [3]. The difference between any other strategy game here is that instead of taking turns, both players can perform as many actions as they want, while the game simulation runs at a constant frame rate. The idea is that both players start with a minimal amount of units and resources, and from there start gathering resources to create more units and buildings and eventually an army to defeat the other player.
Environment Manual
Contents

1 Environment 5
  1.1 Chaoslauncher 5
  1.2 Installation 5
  1.3 The Mas2g 6
    1.3.1 Map 6
    1.3.2 Own Race 7
    1.3.3 Starcraft_Location 7
    1.3.4 Debug 7
    1.3.5 Auto Menu 7
    1.3.6 Enemy Race 7
    1.3.7 Defining an agent 8
  1.4 The Development Tool 9
    1.4.1 Game Speed 9
    1.4.2 Cheat Actions 9
    1.4.3 Map drawing 10

2 Percepts 11
  2.1 Percepts for all units 12
    2.1.1 Available Resources 12
    2.1.2 Unit Information 13
    2.1.3 Player Percepts 14
    2.1.4 Map Percepts 15
    2.1.5 Unit percepts 16
  2.2 Building percepts 18
    2.2.1 Research and Upgrade percepts 18
    2.2.2 Production Buildings 18
    2.2.3 Loadable Buildings 19
  2.3 Worker percepts 20
    2.3.1 Worker Management 20
2.3.2 Builder Percepts

2.4 Conditions

2.4.1 Worker Conditions

2.4.2 Building Conditions

2.4.3 Generic Conditions

2.4.4 Zerg Conditions

2.4.5 Terran Conditions

2.4.6 Protoss Conditions

2.4.7 Unit Percept Conditions

3 Actions

3.1 Attack action

3.2 Move action

3.3 Attack move action

3.4 Upgrade action

3.5 Build action

3.6 Gather action

3.7 Train action

3.8 Stop action

3.9 Ability action

3.10 Ability on target action

3.11 Ability on location action

3.12 Research action

3.13 Set rally point action

3.14 Set rally point to unit action

3.15 Lift action

3.16 Land action

3.17 Build addon action

3.18 Load action

4 TechTypes

4.1 Terran Units

4.1.1 Battle Cruisers

4.1.2 Command Centers

4.1.3 Ghosts

4.1.4 Marines and Firebats

4.1.5 Medics

4.1.6 Science Vessels

4.1.7 Siege Tanks

4.1.8 Vultures
# Unit Types

6.1 Terran Units .............................. 40
   6.1.1 Terran Ground Units ............... 40
   6.1.2 Terran Air Units .................. 40
   6.1.3 Terran Building Units ............. 41
   6.1.4 Terran Addons ...................... 41

6.2 Protoss Units ............................ 42
   6.2.1 Protoss Ground Units ............... 42
   6.2.2 Protoss Air Units .................. 42
   6.2.3 Protoss Building Units ............. 42

6.3 Zerg Units ............................... 43
   6.3.1 Zerg Ground Units ................... 43
   6.3.2 Zerg Air Units ...................... 43
   6.3.3 Zerg Building Units ................. 44
Chapter 1

Environment

This section will explain how to set up and start a bot with the starcraft environment using the GOAL programming language.

1.1 Chaoslauncher

In order to make use of all the starcraft brood war plugin, you can make use of the application: the chaoslauncher. With this application several plugins can be used like the: BWAPI Injector which is neccessary for using the BWAPI library. It is also recommended to make use of the plugin: APMAlert, which shows the current actions per minute of all your units together. When the APM of your bot is suddenly very high, your agents might be using to many actions in a row. At last it is also recomendded to make use of the W-Mode plugin. This plugin automatically sets your Starcraft game in windowed mode which makes it easier for debugging.

1.2 Installation

- Download the starcraft environment MSI installer. After running the installer you can find in the installation folder the environment jar and a folder with 3 examples bots (one for each race).
- You can import one of the Example bots in your eclipse IDE. In the mas2g are some specified values, this will be discussed in 1.3, but for now you can run the bot for testing purposes.
- After running the bot, the chaoslauncher should be automatically launched. If this is not the case please read 1.3. When running the
chaoslauncher, please check the BWAPI Injector (and W-Mode for your own convenience) if this is not already done. From here you should be able to click on start which will launch the game. The running GOAL bot will automatically connect to the game and start controlling your units. If the game does not start immediately, it could be generating the map data. If so please wait up to 2 minutes before doing anything, generating the map data only has to happen once for each map you want to use.

1.3 The Mas2g

The starcraft environment offers multiple parameters to be set up in the mas2g. Within the mas2g you can specify which map you want to play, specify your own race, give up the map location of your starcraft game, turn the development tool on or off, enable the automenu script and specify which race you want to play against.

```java
use StarcraftEnvironment.jar as environment with
map="(2)Destination.scx",
own_race="terran",
starcraft_location="C:\\Starcraft",
debug="true",
auto_menu="Single_Player",
enemy_race="zerg".
```

1.3.1 Map

It is possible to specify which map the chaoslauncher will automatically load when starting the game. This can be done by inserting the following line: `map = <filename>`, where `<filename>` is the exact filename of the map (with extension). Please note that the environment will only choose maps in the directory: `Starcraft/maps/sscai/`. The installer provides the `mapData` of 1 map, however you can use as many maps as you want. When choosing another map in the `sscai` folder please note that the first time running the environment will take some time (around 2 minutes) to generate the data of the given map. This only has to happen once, so it won’t have to generate more than once.
1. Environment

1.3.2 Own Race
You may also specify the race of your bot in the mas2g. This will automatically launch the chaoslauncher with the specified race. You can do this by inserting the following line: `own_race = <RaceName>`, where `<RaceName>` can either be `zerg`, `protoss`, `terran` or `random`. The option `random` will choose one race with 1/3 of a chance for each race.

1.3.3 Starcraft_Location
It is also possible to specify the location of the source map of the starcraft game. When using the starcraft game provided by the environment installer, this feature will automatically start the chaoslauncher when launching the GOAL bot. When the chaoslauncher is already running it won’t start again until you close it. When the Choaslauncher is automatically started by the environment, an automatic script will be written with all the necessary information to run the GOAL bot (so it is recommended to use this feature). You can use this feature by inserting the line: `starcraft_location = <FilePath>`, where `<FilePath>` is the absolute path to the starcraft source folder.

1.3.4 Debug
The Environment also offers a development tool for debugging purposes. With this development tool you can increase or decrease the game speed, enable cheats and draw unit and map details on screen. More information about the development tool can be found at 1.4. For using the development tool you can insert the following line: `debug=<Boolean>`, where `<Boolean>` will indicate for enabling or disabling the development tool.

1.3.5 Auto Menu
The auto menu parameter can be used to quickly go through the menu of the game when starting your agent. This can be used for single player games and multi player games. For using the auto menu function you can insert the following line: `auto_menu=<MenuChoice>`, where `<MenuChoice>` is either `Single_Player` for a single player game or `Multi_Player` for a multi player game.

1.3.6 Enemy Race
The enemy race parameter can be used for specifying which race you want to play against. When an actual enemy race is chosen like: `zerg`, `protoss` or
terran the enemyRace percept will indicate against which race you are playing, while when not specifying an enemy race, so when the option: random is chosen, the enemyRace percept will be Unknown until the opponent is scouted for the first time. For using the enemy race parameter you can insert the following line: enemy_race=<RaceName>, where <RaceName> can either be zerg, protoss, terran or random. The option random will choose one race with 1/3 of a chance for each race.

1.3.7 Defining an agent

When defining an agent it is important that the right type is given to the agent. This has to be the same type of the starcraft unit where the first letter is non-capital. So for example when you want to add a terran SCV agent, this can be done by defining the type of this agent as: terranSCV. Note that each unit type first begins with the race of the unit and is followed by the exact name of the unit type.

define myAgent as agent {
    use MyAgentInit as init module.
    use MyAgent as main module.
    use MyAgentEvent as event module.
}

launchpolicy {
    when type = terranSCV launch myAgent.
}
1.4 The Development Tool

![Figure 1.1: Example of the Development Tool](image)

1.4.1 Game Speed

The Game Speed slider can be found at the top of the development tool window. This can be used to quickly change the speed of the game. The initial game speed is set to 20. The fastest game speed is 50 and the slowest game speed is 0. Please note that the agent is supposed to play normally at game speed 30 which is the default game speed for AI tournaments. When the speed is set to 50, the agents can react slower than they would be on the tournament gamespeed. Setting the game speed on 50 should only be used for quick testing purposes.

1.4.2 Cheat Actions

The development tool offers 3 buttons which instantly enable cheats. Note that these cheats should be used for testing purposes only. The first cheat is called: *Give resources* which gives the player 10000 minerals and 10000 gas. The second cheat is called: *Enemy attacks deal 0 damage* which makes the
units of the player immune for damage. The last cheat is called: *Show map* which makes the whole map visible for the player. Note that also all your agents will be perceiving everything on the map.

1.4.3 Map drawing

The development tool can also be used to show map or unit details. There are 4 buttons which can be used. First there is the *Unit Details* button which shows the health and *ID* of every unit. There is also the *Base Locations* button which shows all the starting locations of the map and also all the base locations on the map where players could be expanding to. There is also the *Chokepoints* button which shows all the chokepoints (which are the narrow points where not many units can go through at the same time) on the map. At last there is the *Build Locations* button which shows all the non-obstructed and explored building locations of the map which the worker units perceive with the *constructionSite* percept.
This section will list all the percepts that are usable in the Starcraft environment. The percepts vary per unit, for example: an attacking unit will not perceive the amount of resources available to the player as he does not need them. For the implementation of these percepts in your GOAL code, please refer to the GOAL manual.
2. Percepts

2.1 Percepts for all units

These percepts are available to all the units and buildings.

2.1.1 Available Resources

Resources percept

<table>
<thead>
<tr>
<th>Description</th>
<th>The amount of minerals, gas and supply available to the player. NOTE: supply is multiplied by 2, so 10 supply in game corresponds with 20 supply in the environment.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>send on change</td>
</tr>
<tr>
<td>Syntax</td>
<td>resources(&lt;M&gt;, &lt;G&gt;, &lt;CS&gt;, &lt;TS&gt;)</td>
</tr>
<tr>
<td>Example</td>
<td>resources(350, 100, 25, 41)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;M&gt;</td>
<td>The current amount of minerals available to the player.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;G&gt;</td>
<td>The current amount of gas available to the player.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;CS&gt;</td>
<td>The supply of the player which is currently in use.</td>
<td>Positive Integer</td>
<td>[0–400]</td>
</tr>
<tr>
<td>&lt;TS&gt;</td>
<td>The total amount of supply the player can currently use. Note that &lt;TS&gt; is always greater or equal to &lt;CS&gt;</td>
<td>Positive Integer</td>
<td>[0–400]</td>
</tr>
</tbody>
</table>
2.1.2 Unit Information

Self percept

Description The (unique) ID and type of the unit. Also gives information about the maximum health, shield and energy of the unit.

Type Send once

Syntax self(<ID>, <UnitType>, <MaxHealth>, <MaxShield>, <MaxEnergy>)

Example self(21, Terran SCV, 60, 0, 0)

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID&gt;</td>
<td>The (unique) ID of the unit.</td>
<td>Positive Integer [0–∞]</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;UnitType&gt;</td>
<td>The type of the unit. The type of a unit consists of a string with the race of the unit and the name of the unit parted by a space. See Section 6 for the list of all the unit types.</td>
<td>String</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;MaxHealth&gt;</td>
<td>The maximum amount of health of the unit.</td>
<td>Positive Integer [0–2500]</td>
</tr>
<tr>
<td>Type</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;MaxShield&gt;</td>
<td>The maximum amount of shield of the unit.</td>
<td>Positive Integer [0–2500]</td>
</tr>
<tr>
<td>Type</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;MaxEnergy&gt;</td>
<td>The maximum amount of energy of the unit.</td>
<td>Positive Integer [0–2500]</td>
</tr>
<tr>
<td>Type</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>Range</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Defensive Matrix percept

Description Information about how much health the defensive matrix has left on a unit.

Type Send on change

Syntax defensiveMatrix(<health>)

Example defensiveMatrix(200)

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;health&gt;</td>
<td>The amount of health left of the defensive matrix.</td>
<td>Positive Integer [0–250]</td>
</tr>
</tbody>
</table>
2. Percepts

## Status percept

**Description**
The current amount of health, shield and energy of the unit. The status percept also shows the conditions of the unit and the current position.

**Type**
Send on change

**Syntax**
`status(<Health>, <Shield>, <Energy>, <Cond>, <X>, <Y>)`

**Example**
`status(250, 0, 0, [moving, carrying], 24, 36)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Health&gt;</td>
<td>Type</td>
<td>Positive Integer</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>([0–&lt;MaxHealth&gt;]) where (&lt;MaxHealth&gt;) is the maximum health of the given unit.</td>
</tr>
<tr>
<td>&lt;Shield&gt;</td>
<td>Type</td>
<td>Positive Integer</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>([0–&lt;MaxShield&gt;]) where (&lt;MaxShield&gt;) is the maximum shield of the given unit.</td>
</tr>
<tr>
<td>&lt;Energy&gt;</td>
<td>Type</td>
<td>Positive Integer</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>([0–&lt;MaxEnergy&gt;]) where (&lt;MaxEnergy&gt;) is the maximum energy of the given unit.</td>
</tr>
<tr>
<td>&lt;Cond&gt;</td>
<td>Type</td>
<td>List of Strings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The current condition of the unit. Each unit can have multiple or no conditions depending on the unit and situation. See Section 2.4 for the list of all the conditions.</td>
</tr>
<tr>
<td>&lt;X&gt;</td>
<td>Type</td>
<td>Positive Integer</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>([0–∞])</td>
</tr>
<tr>
<td>&lt;Y&gt;</td>
<td>Type</td>
<td>Positive Integer</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>([0–∞])</td>
</tr>
</tbody>
</table>

### 2.1.3 Player Percepts

## Enemy Race percept

**Description**
The race of your opponent.

**Type**
Send once

**Syntax**
`enemyRace(<Race>)`

**Example**
`enemyRace(protoss)`
2. Percepts

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Race&gt;</td>
<td>The enemy race which can take the value: protoss, terran, zerg or unknown when the enemy race is not yet known.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

2.1.4 Map Percepts

Map percept

Description: The width and the height of the map.

Type: Send once

Syntax: `map(<Width>,<Height>)`

Example: `map(96, 128)`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Width&gt;</td>
<td>The width of the map.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;Height&gt;</td>
<td>The height of the map.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
</tbody>
</table>

Base percept

Description: All the base locations of the map.

Type: Send once

Syntax: `base(<X>,<Y>,<IsStart>,<RegionID>)`

Example: `base(28, 32, true, 8)`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;X&gt;</td>
<td>The x-coordinate of the base location.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;Y&gt;</td>
<td>The y-coordinate of the base location.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;IsStart&gt;</td>
<td>Indicates whether the location is a starting location or not.</td>
<td>Boolean (true or false)</td>
<td></td>
</tr>
<tr>
<td>&lt;RegionID&gt;</td>
<td>The ID of the region this location is in. The vespene geyser and all mineral fields will share this region ID.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
</tbody>
</table>
2. Percepts

Chokepoint percept

Description  All the chokepoints on the map. These are the narrow points on the map where only a limited amount of units can go through at the same time.

Type  Send once

Syntax  `chokepoint(<X>,<Y>)`

Example  `chokepoint(12, 15)`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;X&gt;</code></td>
<td>The x-coordinate of the chokepoint.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td><code>&lt;Y&gt;</code></td>
<td>The y-coordinate of the chokepoint.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
</tbody>
</table>

2.1.5 Unit percepts

Attacking percept

Description  Shows the units which are attacking and which units they have targeted.

Type  Send always

Syntax  `attacking(<ID>,<TargetID>,<X>,<Y>)`

Example  `attacking(123, 177, 120, 96)`

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ID&gt;</code></td>
<td>The (unique) ID of the unit which is attacking.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td><code>&lt;TargetID&gt;</code></td>
<td>The (unique) ID of the targeted unit which is being attacked.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td><code>&lt;X&gt;</code></td>
<td>The x-coordinate of the (attacking) unit.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td><code>&lt;Y&gt;</code></td>
<td>The y-coordinate of the (attacking) unit.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
</tbody>
</table>
Unit percept

Description  Shows all units that are currently visible to the player.
Type  Send always
Syntax  unit(<IsFriendly>,<Type>,<ID>,<Health>,<Shield>,<Condition>)
Example  unit(true, Protoss Gateway, 26, 255, 255, [isBeingConstructed])

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;IsFriendly&gt;</td>
<td>Indicates whether the unit is friendly or not. Type Boolean (true or false)</td>
</tr>
<tr>
<td>&lt;Type&gt;</td>
<td>The type of the unit. The type of a unit consists of a string with the race of the unit and the name of the unit parted by a space. See Section 6 for the list of all the unit types. Type String</td>
</tr>
<tr>
<td>&lt;ID&gt;</td>
<td>The (unique) ID of the unit. Type Positive Integer Range [0–∞]</td>
</tr>
<tr>
<td>&lt;Health&gt;</td>
<td>The current amount of health of the unit. Type Positive Integer Range [0–&lt;maxHealth&gt;] where &lt;maxHealth&gt; is the maximum health of the given unit.</td>
</tr>
<tr>
<td>&lt;Shield&gt;</td>
<td>The current amount of shields of the unit. Type Positive Integer Range [0–&lt;maxShield&gt;] where &lt;maxShield&gt; is the maximum shield of the given unit.</td>
</tr>
<tr>
<td>&lt;Cond&gt;</td>
<td>The current condition of the unit. Each unit can have multiple or no conditions depending on the unit and situation. See Section 2.4 for the list of all actual conditions. Type List of Strings</td>
</tr>
</tbody>
</table>
2.2 Building percepts

These percepts are available to buildings.

2.2.1 Research and Upgrade percepts

HasResearched percept

Description: Indicates which tech is already researched. See Section 4 for the list of all actual tech types.

Type: send once

Syntax: hasResearched(<TechType>)

Example: hasResearched(Stim Packs)

Parameters:

<table>
<thead>
<tr>
<th>&lt;TechType&gt;</th>
<th>The tech which is currently researched.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

Upgrading percept

Description: Indicates which upgrade is currently being upgraded. See Section 5 for the list of all actual tech types.

Type: Send always

Syntax: upgrading(<UpgradeType>)

Example: upgrading(Stim Packs)

Parameters:

<table>
<thead>
<tr>
<th>&lt;UpgradeType&gt;</th>
<th>The upgrade which is currently upgraded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

2.2.2 Production Buildings

Queue Size percept

Description: Shows how many units are in queue of the production building.

Type: Send on change

Syntax: queueSize(<Size>)

Example: queueSize(2)

Parameters:

<table>
<thead>
<tr>
<th>&lt;Size&gt;</th>
<th>The size of the current queue.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Positive Integer</td>
</tr>
<tr>
<td>Range</td>
<td>[0–5]</td>
</tr>
</tbody>
</table>
Rally point percept

Description The exact position of the rallypoint in map coordinates.

Type Send on change

Syntax rallyPoint(<X>,<Y>)

Example rallyPoint(76, 45)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;X&gt;</td>
<td>The x-coordinate of the rallypoint.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;Y&gt;</td>
<td>The y-coordinate of the rallypoint.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
</tbody>
</table>

Rally unit percept

Description Shows on which unit the rallypoint is set.

Type Send on change

Syntax rallyUnit(<UnitID>)

Example rallyUnit(145)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;UnitID&gt;</td>
<td>The (unique) ID the rallypoint points to.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
</tbody>
</table>

2.2.3 Loadable Buildings

Space Provided percept

Description Shows how many units are currently loaded in the building and how the maximum amount of units that can be loaded in the building.

Type Send on change

Syntax spaceProvided(<CSize>, <MSize>)

Example spaceProvided(2, 4)
2. Percepts

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CSize&gt;</td>
<td>The amount of currently loaded units.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;MSize&gt;</td>
<td>The maximum amount of units that can be loaded.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
</tbody>
</table>

**Unitloaded percept**

<table>
<thead>
<tr>
<th>Description</th>
<th>Send always</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>unitLoaded(&lt;ID&gt;, &lt;Type&gt;)</td>
</tr>
<tr>
<td>Example</td>
<td>unitLoaded(154, Terran Marine)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID&gt;</td>
<td>The (unique) ID of the loaded unit.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;Type&gt;</td>
<td>The type of the loaded unit.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

2.3 Worker percepts

These percepts are available to worker units.

2.3.1 Worker Management

**Worker Activity Percept**

<table>
<thead>
<tr>
<th>Description</th>
<th>Shows the current activity of all friendly workers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>workerActivity(&lt;ID&gt;, &lt;Activity&gt;)</td>
</tr>
<tr>
<td>Example</td>
<td>workerActivity(146, gatheringGas)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID&gt;</td>
<td>The (unique) ID of the worker unit.</td>
<td>Positive Integer</td>
<td>[0–∞]</td>
</tr>
<tr>
<td>&lt;Activity&gt;</td>
<td>The current activity of the worker unit. Can take values: gatheringGas, gatheringMinerals, constructing or idling.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>
2. Percepts

Gathering Percept

Description  Shows which mineral or vespene ID the worker is currently gathering from.
Type  Send always
Syntax  gathering(<ID>)
Example  gathering(110)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID&gt;</td>
<td>The (unique) ID of the mineral or vespene geyser.</td>
<td>Positive Integer [0–∞]</td>
</tr>
</tbody>
</table>

2.3.2 Builder Percepts

Vespene Geyser percept

Description  Information about a visible vespene geyser on the map.
Type  Send on change
Syntax  vespeneGeyser(<ID>, <Resources>, <ResourceGroup>, <X>, <Y>)
Example  vespeneGeyser(57, 5000, 6, 22, 32)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ID&gt;</td>
<td>The (unique) ID of the vespene geyser.</td>
<td>Positive Integer [0–∞]</td>
</tr>
<tr>
<td>&lt;Resources&gt;</td>
<td>The amount of resources left in the vespene geyser.</td>
<td>Positive Integer [0–5000]</td>
</tr>
<tr>
<td>&lt;ResourceGroup&gt;</td>
<td>The resource group of the vespene geyser.</td>
<td>Positive Integer [0–∞]</td>
</tr>
<tr>
<td>&lt;X&gt;</td>
<td>The x-coordinate of the vespene geyser.</td>
<td>Positive Integer [0–∞]</td>
</tr>
<tr>
<td>&lt;Y&gt;</td>
<td>The y-coordinate of the vespene geyser.</td>
<td>Positive Integer [0–∞]</td>
</tr>
</tbody>
</table>
### ConstructionSite percept

**Description**  Shows all construction sites on the map, which are explored and not obstructed.

**Type**  Send always

**Syntax**
- (If Protoss) `constructionSite(<X>,<Y>,<InPylonRange>)`
- (If Zerg/Terran) `constructionSite(<X>,<Y>)`

**Example**
- `constructionSite(66, 98, false)`
- `constructionSite(66, 98)`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;X&gt;</code></td>
<td>The x-coordinate of the construction site.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Positive Integer</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td><code>[0–∞)</code></td>
</tr>
<tr>
<td><code>&lt;Y&gt;</code></td>
<td>The y-coordinate of the construction site.</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Positive Integer</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td><code>[0–∞)</code></td>
</tr>
<tr>
<td><code>&lt;InPylonRange&gt;</code></td>
<td>Indicates whether the construction site is in range of a pylon (this is only for protoss)</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Boolean (true or false)</td>
</tr>
</tbody>
</table>
2.4 Conditions

These are all the possible conditions agents can have in the condition percept.

2.4.1 Worker Conditions

These are the conditions only worker units can have.

- **carrying**
  - Indicates when the worker unit is carrying minerals or vespene gas.

- **constructing**
  - Shows that the worker unit is busy constructing a building.

2.4.2 Building Conditions

These are the conditions only building units can have.

- **beingConstructed**
  - Indicates when a building is being constructed.

- **lifted**
  - Indicates when the building is lifted.

- **<addonName>**
  - Indicates when an addon of the building is present, gives the exact addonname.

2.4.3 Generic Conditions

These are the conditions all units can have.

- **idle**
  - Indicates when the unit is idle (not doing anything).

- **cloaked**
  - Indicates when a unit is cloaked.

- **moving**
  - Shows that a unit is moving.

- **following**
  - Shows that a unit is following an other unit.

- **loaded**
  - Indicates when a unit is loaded.

2.4.4 Zerg Conditions

These are the conditions caused by zerg units.
2. Percepts

2.4.5 Terran Conditions

These are the conditions caused by terran units.

- **burrowed**
  Indicates when a zerg unit is burrowed.
- **ensnared**
  Shows that the unit is ensnared by a Queen unit.
- **parasited**
  Shows that the unit is parasited by a Queen unit.
- **plagued**
  Indicates that the unit is plagued by a Defiler unit.
- **darkSwarmed**
  Indicates that the unit is under a Dark Swarm from a Defiler unit.

2.4.6 Protoss Conditions

These are the conditions caused by protoss units.

- **underStorm**
  Shows when a unit is under a storm from a High Templar unit.
- **inStatis**
  Indicates when a unit is stuck in stasis.
- **maelstrommed**
  Indicates when a unit is maelstrommed by a Dark Archon.
- **disruptionWebbed**
  Shows when a unit is in a disruption web from a Corsair.

2.4.7 Unit Percept Conditions

These are the conditions which are visible within the friendly and enemy percept.

- **flying**
  Indicates whether a unit is flying or not.
- **morphing**
  Shows when a unit is morphing. (NOTE that sieging and unsieging is also considered morphing)
- **cloaked**
  Indicates when a unit is cloaked.
- **beingConstructed**
  Indicates when a unit is being constructed.
Chapter 3

Actions

This section will list all the actions that are usable in the Starcraft environment.

3.1 Attack action

Description: This action makes a unit which is attack capable, attack the chosen unit.

Syntax: \texttt{attack(<TargetID>)}

Parameters:
\<TargetID>: The ID of the target that will be attacked.

Pre: The targeted unit is attack capable.

Post: The targeted unit is being attacked by your unit.

3.2 Move action

Description: Instruct a unit to move to a chosen location.

Syntax: \texttt{move(<X>,<Y>)}

Parameters:
\<X>: The x-coordinate of the chosen location
\<Y>: The y-coordinate of the chosen location

Pre: The unit is capable of moving to the chosen location.

Post: The unit moves to the chosen location (ignoring any other unit it might pass by).
3. Actions

3.3 Attack move action

Description Go to a location and attack everything you encounter.
Syntax `attack(<X>,<Y>)`
Parameters `<X>`: The x-coordinate of the chosen location
`<Y>`: The y-coordinate of the chosen location
Pre The unit is capable of moving to the chosen location.
Post The unit moves to the chosen locations and attacks any attack capable enemy unit it encounters.

3.4 Upgrade action

Description Starts working on the chosen upgrade.
Syntax `upgrade(<UpgradeName>)`
Parameters `<UpgradeName>`: The name of the upgrade you want to upgrade.
Pre The unit is capable of upgrading and has sufficient resources to do so.
Post The unit starts upgrading the chosen upgrade.

3.5 Build action

Description Build a building on a given, not obstructed location.
Syntax `build(<Type>,<X>,<Y>)`
Parameters `<Type>`: The Type of the building that has to be built.
`<X>`: The x-coordinate of the chosen build location
`<Y>`: The y-coordinate of the chosen build location
Pre The unit is capable of constructing the chosen building and the chosen location is not obstructed.
Post The unit starts constructing the chosen building at the chosen location.
3. Actions

3.6 Gather action

Description Instruct a unit to gather the chosen resource. This can either be minerals or vespene gas.

Syntax `gather(<ID>)`

Parameters `<ID>`: The ID of the chosen resource.

Pre The unit is capable of performing the gather action and a valid resource unit is selected.

Post The unit starts gathering the chosen resource.

3.7 Train action

Description Train a chosen unit with a production facility capable of producing the chosen unit.

Syntax `train(<Type>)`

Parameters `<Type>`: The type of unit to train.

Pre The production facility is capable of producing the chosen unit and has sufficient resources to do so.

Post The production facility starts producing the chosen unit.

3.8 Stop action

Description The unit stops performing the action he was busy with.

Syntax `stop`

Pre The unit is performing some kind of action.

Post The unit stops performing the action.

3.9 Ability action

Description Use an (researched) ability.

Syntax `use(<Type>)`

Parameters `<Type>`: The type of technology to use.

Pre The chosen tech type is researched and the unit is capable of performing the chosen tech type.

Post The unit performs the chosen tech ability.
3. Actions

3.10 Ability on target action

Description Use an (researched) ability on a target.
Syntax use(<Type>, <Target>)
Parameters
- **<Type>**: The type of technology to use.
- **<Target>**: The target to use the technology on.
Pre The chosen tech type is researched, the unit is capable of performing the chosen tech type and the chosen target is attack capable.
Post The unit performs the chosen tech ability on the chosen target.

3.11 Ability on location action

Description use an (researched) ability on a location.
Syntax use(<Type>, <X>, <Y>)
Parameters
- **<Type>**: The type of technology to use.
- **<X>**: The x-coordinate of the chosen location
- **<Y>**: The y-coordinate of the chosen location.
Pre The chosen tech type is researched, the unit is capable of performing the chosen tech type and the chosen location is valid to perform an action on.
Post The unit performs the chosen tech ability on the chosen location.

3.12 Research action

Description Research a chosen tech type.
Syntax research(<Type>)
Parameters **<Type>**: The type of tech to research.
Pre The building is capable of researching the chosen tech type and has sufficient resources to do so.
Post The building starts researching the chosen tech type.
3. Actions

3.13 Set rally point action

Description: Set the rally point of a building on a specific location. When the rally point is set, produced units of this production facility will automatically move to this location.

Syntax: `setRallyPoint(<X>, <Y>)`

Parameters:
- `<X>`: The x-coordinate of the chosen rally location
- `<Y>`: The y-coordinate of the chosen rally location.

Pre: The building is capable of setting up a rally point and the chosen location is a valid location where units can move to.

Post: The building sets the rally point on the chosen location.

3.14 Set rally point to unit action

Description: Set the rally point of a building on a unit. When the rally point is set, produced units of this production facility will automatically move to this unit.

Syntax: `setRallyPoint(<Unit>)`

Parameters:
- `<Unit>`: The unit to set the rally point on.

Pre: The building is capable of setting up a rally point and the chosen unit is on a valid location where units can move to.

Post: The building sets the rally point on the chosen unit.

3.15 Lift action

Description: Lifts a building which is capable of lifting.

Syntax: `lift`

Pre: The building is capable of flying and is not busy performing any other action.

Post: The building starts flying.

Note: Only for Terran buildings.
3.16 Land action

Description: Land the unit on a specific, not obstructed location.
Syntax: `land(<X>, <Y>)`
Parameters:
- `<X>`: The x-coordinate of the chosen land location
- `<Y>`: The y-coordinate of the chosen land location.
Pre: The unit is currently flying and is capable of landing on the chosen location.
Post: The unit lands on the chosen location.
Note: The location has to be visible.

3.17 Build addon action

Description: Order a building to build a chosen addon.
Syntax: `buildAddon(<Name>)`
Parameters:
- `<Name>`: The name of the chosen addon.
Pre: The building is capable of building the addon and does not already have the addon.
Post: The building starts constructing the addon.
Note: Only for Terran buildings.

3.18 Load action

Description: Order a unit to load into this (loadable) unit.
Syntax: `load(<ID>)`
Parameters:
- `<ID>`: The ID of the unit to load into this (loadable) unit.
Pre: The unit is capable of loading other units inside it and still has enough space prodvided for the targeted unit.
Post: The targeted unit starts walking to the loadable unit and loads into it.
Chapter 4

TechTypes

Here is the list of all tech types that can be researched.

4.1 Terran Units

These are all the Terran tech types.

4.1.1 Battle Cruisers

These are the tech type(s) for Battle Cruiser units.
Yamato Gun

4.1.2 Command Centers

These are the tech type(s) for Command Center units.
Scanner Sweep

4.1.3 Ghosts

These are the tech type(s) for Ghost units.
Lockdown
Personel Cloaking
Nuclear Strike

4.1.4 Marines and Firebats

These are the tech type(s) for Marine and Firebat units.
Stim Packs
4. TechTypes

4.1.5 Medics
These are the tech type(s) for Medic units.
Healing
Restoration
Optical Flare

4.1.6 Science Vessels
These are the tech type(s) for Science Vessel units.
Defensive Matrix
EMP Shockwave
Irradiate

4.1.7 Siege Tanks
These are the tech type(s) for Siege Tank units.
Tank Siege Mode

4.1.8 Vultures
These are the tech type(s) for Vulture units.
Spider Mines

4.1.9 Wraith
These are the tech type(s) for Wraith units.
Cloaking Field

4.2 Protoss Units
These are all the Protoss tech types.

4.2.1 Arbiters
These are the tech type(s) for Arbiter units.
Cloaking Field
Recall
Stasis Field
4.2.2 Corsairs

These are the tech type(s) for Corsair units.

Disruption Web

4.2.3 Dark Archons

These are the tech type(s) for Dark Archon units.

Feedback
Maelstrom
Mind Control

4.2.4 Dark Templars

These are the tech type(s) for Dark Templar units.

Dark Archon Meld

4.2.5 High Templars

These are the tech type(s) for High Templar units.

Archon Warp
Psionic Storm
Hallucination

4.3 Zerg Units

These are all the Zerg tech types.

4.3.1 Generic

These are the tech type(s) which all ground units can use.

Burrowing

4.3.2 Defilers

These are the tech type(s) for Defilers units.

Dark Swarm
Plague
Consume
4.3.3 Hydralisks
These are the tech type(s) for Hydralisk units.
Lurker Aspect

4.3.4 Lurkers
These are the tech type(s) for Lurker units.
Burrowing (Can be used without having it researched)

4.3.5 Queens
These are the tech type(s) for Queen units.
Infestation
Parasite
Ensnare
Spawn Broodlings
Chapter 5

UpgradeTypes

Here is the list of all upgrade types that can be upgraded.

5.1 Terran Units

These are all the Terran upgrade types for Terran units.

5.1.1 Academy

These are the upgrade type(s) the Academy offers.
U 238 Shells
Caduceus Reactor

5.1.2 Armory

These are the upgrade type(s) the Armory offers.
Terran Vehicle Weapons
Terran Vehicle Plating
Terran Ship Weapons
Terran Ship Plating

5.1.3 Covert Ops

These are the upgrade type(s) the Covert Ops offers.
Ocular Implants
Moebius Reactor
5. UpgradeTypes

5.1.4 Engineering Bay
These are the upgrade type(s) the Engineering Bay offers.
Terran Infantry Weapons
Terran Infantry Armor

5.1.5 Machine Shop
These are the upgrade type(s) the Machine Shop offers.
Ion Thrusters
Charon Boosters

5.1.6 Physics Lab
These are the upgrade type(s) the Physics Lab offers.
Colossus Reactor

5.1.7 Science Facility
These are the upgrade type(s) the Science Facility offers.
Titan Reactor

5.1.8 Control Tower
These are the upgrade type(s) the Control Tower offers.
Apollo Reactor

5.2 Protoss Units
These are all the Protoss upgrade types for Protoss units.

5.2.1 Arbiter Tribunal
These are the upgrade type(s) the Arbiter Tribunal offers.
Khaydarin Core

5.2.2 Citadel of Adun
These are the upgrade type(s) the Citadel of Adun offers.
Protoss Plasma Shields
5. Upgrade Types

Leg Enhancements

5.2.3 Cybernetics Core
These are the upgrade type(s) the Cybernetics Core offers.
Singularity Charge
Protoss Air Weapons
Protoss Air Armor

5.2.4 Fleet Beacon
These are the upgrade type(s) the Fleet Beacon offers.
Apial Sensors
Gravitic Thrusters
Argus Jewel
Carrier Capacity

5.2.5 Forge
These are the upgrade type(s) the Forge offers.
Protoss Plasma Shields
Protoss Ground Armor
Protoss Ground Weapons

5.2.6 Observatory
These are the upgrade type(s) the Observatory offers.
Gravitic Boosters
Sensor Array

5.2.7 Robotics Support Bay
These are the upgrade type(s) the Robotics Support Bay offers.
Reaver Capacity
Scarab Damage
Gravitic Drive
5. UpgradeTypes

5.2.8 Templar Archives
These are the upgrade type(s) the Templar Archives offers.
Argus Talisman
Khaydarin Amulet

5.3 Zerg Units
These are all the Zerg upgrade types for Zerg units.

5.3.1 Defiler Mound
These are the upgrade type(s) the Defiler Mound offers.
Metasynaptic Node

5.3.2 Evolution Chamber
These are the upgrade type(s) the Evolution Chamber offers.
Zerg Melee Attacks
Zerg Missile Attacks
Zerg Carapace

5.3.3 Hydralisk Den
These are the upgrade type(s) the Hydralisk Den offers.
Muscular Augments
Grooved Spines

5.3.4 Lair and Hive
These are the upgrade type(s) the Lair and Hive offers.
Ventral Sacs
Antennae
Pneumatized Carapace

5.3.5 Queen’s Nest
These are the upgrade type(s) the Queen’s Nest offers.
Gamete Meiosis
5. UpgradeTypes

5.3.6 Spawning Pool
These are the upgrade type(s) the Spawning Pool offers.
Metabolic Boost
Adrenal Glands

5.3.7 (Greater) Spire
These are the upgrade type(s) the (Greater) Spire offers.
Zerg Flyer Carapace
Zerg Flyer Attacks

5.3.8 Ultralisk Cavern
These are the upgrade type(s) the Ultralisk Cavern offers.
Chitinous Plating
Anabolic Synthesis
Chapter 6

Unit Types

Here is the list of all unit types you can specify within the mas2g. Note that when you bind your agent to a specific unit type in the mas2g, the first letter of the name unit type should always be non-capital!

6.1 Terran Units

These are all the terran unit types.

6.1.1 Terran Ground Units

These are all the terran ground units.

Terran Firebat
Terran Ghost
Terran Goliath
Terran Marine
Terran Medic
Terran SCV
Terran Siege Tank
Terran Vulture
Terran Vulture Spider Mine

6.1.2 Terran Air Units

These are all the terran air units.
6. Unit Types

Terran Battlecruiser
Terran Dropship
Terran Science Vessel
Terran Valkyrie
Terran Wraith

6.1.3 Terran Building Units

These are all the terran building units.

Terran Academy
Terran Armory
Terran Barracks
Terran Bunker
Terran Command Center
Terran Engineering Bay
Terran Factory
Terran Missile Turret
Terran Refinery
Terran Science Facility
Terran Starport
Terran Supply Depot

6.1.4 Terran Addons

These are all the terran addon units. Note that terran is the only race capable of making addons.

Terran Comsat Station
Terran Control Tower
Terran Covert Ops
Terran Machine Shop
Terran Nuclear Silo
Terran Physics Lab
6.2 Protoss Units

These are all the protoss unit types.

6.2.1 Protoss Ground Units

These are all the protoss ground units.

Protoss Archon
Protoss Dark Archon
Protoss Dark Templar
Protoss Dragoon
Protoss High Templar
Protoss Probe
Protoss Reaver
Protoss Scarab
Protoss Zealot

6.2.2 Protoss Air Units

These are all the protoss air units.

Protoss Arbiter
Protoss Carrier
Protoss Corsair
Protoss Interceptor
Protoss Observer
Protoss Scout
Protoss Shuttle

6.2.3 Protoss Building Units

These are all the protoss building units.

Protoss Arbiter Tribunal
Protoss Assimilator
Protoss Citadel of Adun
Protoss Cybernetics Core
Protoss Fleet Beacon
6. Unit Types

- Protoss Forge
- Protoss Gateway
- Protoss Nexus
- Protoss Observatory
- Protoss Photon Cannon
- Protoss Pylon
- Protoss Robotics Facility
- Protoss Robotics Support Bay
- Protoss Shield Battery
- Protoss Stargate
- Protoss Templar Archives

6.3 Zerg Units

These are all the zerg units.

6.3.1 Zerg Ground Units

These are all the zerg ground units.

- Zerg Broodling
- Zerg Defiler
- Zerg Drone
- Zerg Egg
- Zerg Hydralisk
- Zerg Infested Terran
- Zerg Larva
- Zerg Lurker
- Zerg Lurker Egg
- Zerg Ultralisk
- Zerg Zergling

6.3.2 Zerg Air Units

These are all the zerg air units.

- Zerg Cocoon
- Zerg Devourer
6. Unit Types
6.3.3 Zerg Building Units

These are all the zerg building units.

Zerg Creep Colony
Zerg Defiler Mound
Zerg Evolution Chamber
Zerg Extractor
Zerg Greater Spire
Zerg Hatchery
Zerg Hive
Zerg Hydralisk Den
Zerg Infested Command Center
Zerg Lair
Zerg Nydus Canal
Zerg Queens Nest
Zerg Spawning Pool
Zerg Spire
Zerg Spore Colony
Zerg Sunken Colony
Zerg Ultralisk Cavern
Course manual
PROJECT MULTI-AGENT SYSTEMS
(TI1606)


June 17, 2016

Delft University of Technology
Faculty of Electrical Engineering, Mathematics, and Computer Science
# Contents

## 1 Introduction

2

## 2 Learning Goals

3

## 3 Individual Assignment

4

### 3.1 Description of the assignment

4

#### 3.1.1 Requirements

4

### 3.2 Deliverable

5

### 3.3 Assessment

5

### 3.4 Deadline is Strict

6

## 4 Group Assignment

6

### 4.1 Approach

6

#### 4.1.1 Plan

7

#### 4.1.2 Execute

7

#### 4.1.3 Review

8

### 4.2 Performance Targets

8

#### 4.2.1 Game Settings and Evaluation Setup

9

### 4.3 Functional Requirements and Sprints

9

#### 4.3.1 Sprint Week 3: The Basics

10

#### 4.3.2 Sprint Week 4: Defend and Scout

10

#### 4.3.3 Sprint Week 5: Attacking, Upgrading and Researching

10

#### 4.3.4 Sprint Week 6: Improve attack strategy and Improve resources income

11

#### 4.3.5 Sprint Week 7: Race and Map specific strategies

11

#### 4.3.6 Sprint Week 8: Test and Improve

11

#### 4.3.7 Sprint Week 9: Refinement

11

### 4.4 Deliverable Guidelines

11

#### 4.4.1 Handing in Deliverables

12

#### 4.4.2 Report

12

#### 4.4.3 Project Management Tool Trac

16

### 4.5 Grading and Evaluation of Your Performance

17

#### 4.5.1 Competition and End of Project Review Meetings

17

#### 4.5.2 Grading

18

#### 4.5.3 Individual Contribution

18

#### 4.5.4 Product Grade

18

#### 4.5.5 Process

19

#### 4.5.6 Who is evaluating what?

19

## 5 Organization

19

### 5.1 Supervision

20

#### 5.1.1 Role Assistants and Supervisors

20

### 5.2 The Formation of Groups

20

#### 5.2.1 The First Project Group Meeting

20

### 5.3 Course Materials

21
Supervisors:
Dr. Nils Bulling
Dr. Koen Hindriks
Dr. M. Birna van Riemsdijk
Ir. Vincent Koeman
Dr. Ester Sorm

Assistants:
Cas Bilstra, Ege de Bruin, Nick Cleintuar,
Michiel Doesburg, Maiko Goudriaan, Ricardo Jongerius,
Maikel Kerkhof, Mathias Meuleman, Menno Oudshoorn,
Danny Plenge, Wouter Smit, Daphne van Tetering,
Paul Verkooijen

Contact Email: ti1606.mas@gmail.com
1 Introduction

The gaming industry is developing rapidly, and techniques from Artificial Intelligence are becoming more and more common place to control so-called “non-player characters” (NPCs). NPCs are the characters in a game that are not controlled by humans. In addition, serious games are an emerging application in which a game is not just used for entertainment but for educational means as well, for example, for training disaster response teams. In the future, a fire chief who is being trained by means of a serious game will cooperate with policemen and paramedics in the game that are controlled by the NPCs.

Artificial intelligence is a field in computer science that is rapidly growing and improving. After Google defeated the top ranked Go players, Google searched for a new challenge. The new challenge that arose was the strategy game StarCraft. StarCraft is deemed far more complex than Go and is a big challenge. Google is not alone in this, for many years research is being done about AI in StarCraft. Every year a StarCraft tournament is hosted for students, in which the AI’s play against each other for the ultimate goal of being the best AI.

This project has been designed within this context. Every student group is asked to program an agent system that controls bots in StarCraft. The agents will be programmed in the GOAL agent programming language, which has been taught and used also in the Logic-Based AI course. GOAL supports control of bots on a high level of abstraction. For example, by using GOAL, one can instruct the bot to move to a certain location on a map. The bot itself will execute this action on a lower level of abstraction; in other words, the bot will autonomously determine how to move from one point to the other without getting stuck (whenever possible).

Generally speaking, the assignment is to create a team of GOAL agents that control multiple bots in a game of StarCraft in a group of six students, aiming for a well performing team (see Section 4 for more details). At the end of the project, a competition will be organized in which the teams of different groups will battle each other for the grand prize! However, before starting this group assignment, a (small) individual assignment will need to be completed (see Section 3).

Note that attendance during scheduled contact hours is compulsory (for both the individual and the group assignment parts). In case there are compelling reasons for not being present at any time, you need to notify Teaching Assistants and supervisors well in advance. Only if they agree with your reason for being absent, you can be absent. In this case, you will need to discuss how you will compensate for this absence with your group, Teaching Assistant, and the supervisors. Unauthorized absence may influence your grade negatively. Note that a holiday does not count as a valid reason for not being present.

Also note that scheduled contact hours only account for half of the time that you are expected to spent on the project. On average, every week, you should spend at least 7 more hours on this project. Also note that the Technical Writing course has been fully integrated into this project. You should on average each spend 20% of your time on writing the report (including contact hours for Technical Writing, which accounts for 1 ECTS out of the total of 5 ECTS for the project). The focus of this project is on programming an agent system that controls bots, which you should document in the report.

An overview of deadlines and deliverables for the project can be found in Table 1.1. The dates associated with deliverables are strict deadlines for handing in these deliverables. The time line for the project is tight and we cannot allow for any delays. Rows in italic font in the table indicate important dates that require your attendance or action; all other rows correspond with deliverables.

The following terminology is used in this project. With the term agent we refer to the GOAL agents. Depending on the context this means the GOAL program itself or the process that is started when the agent program is run.
Read this manual carefully so it is clear to you what is expected of you during the project! The remainder of this manual is organized as follows:

- Section 2: presents the **learning goals**;
- Section 3: describes the **individual assignment**;
- Section 4: describes the **group assignment**;
- Section 5: provides information about the **organization and supervision** of this project.

## 2 Learning Goals

In this project, students will learn to apply the concepts and techniques they have learned in the accompanying Logic Based AI course in practice. At the end of the project, a student will be able to analyse requirements and how to manage a project as part of a team, and will be able to select, specify, implement and test solutions based on agent technology. Below, the specific learning goals have been specified.

1. **Agents and Multi-Agent Systems**
   
   **Concepts:** agent, multi-agent system, cognitive state, beliefs, goals, actions, communication, coordination.
   
   The student is able to recognize these concepts in given assignments and program code, and apply these concepts consistently and effectively in a multi-agent program and reports.

2. **Agent-Environment Interaction**
   
   **Concepts:** action, percept
   
   The student is able to recognize these concepts in given assignments and program code, and apply these concepts consistently and effectively in a multi-agent program and accompanying reports.
3. **Agents and Multi-Agent Programs**  
*Concepts:* action rule, module, MAS file, launch rule  
The student is able to apply relevant programming constructs consistently and effectively when writing a multi-agent program that acts in a dynamic real-time environment that requires coordination between agents.

4. **Agent-oriented Software Development**  
*Concepts:* scenario, testing.  
The student is able to construct a scenario and use it to test the software under development using a testing framework for agents.

5. **Project Skills**  
The student is able to collaborate on solving complex issues in an efficient and result-oriented manner. The student is able to determine an appropriately balanced task division in a team and reflect on his or her own contribution to the group process and resulting product.

6. **Technical Writing Skills**  
The student has appropriate skills in writing professional and scientific texts, providing insight into the factors that influence the quality, usefulness and persuasiveness of corporate and scientific texts.

### 3 Individual Assignment

The individual assignment is intended to give students a head start on programming a GOAL agent that controls a StarCraft bot. In addition, the aim of this assignment is to familiarize students with the StarCraft-GOAL environment. After completing this assignment, a student will know which actions can be executed, which percepts can be received, and is familiar with one of the StarCraft Brood War maps that is used in the project.

#### 3.1 Description of the assignment

The given agent: *SimpleAgent* is a very basic agent that contains some initial code that makes the bot collect minerals. The objective of this assignment is to add the following functionality:

- build more worker units
- start constructing multiple structures (depending on which race you are playing)
- begin scouting the enemy base.

These skills are needed for your bots in able to effectively start producing units and play the game.

#### 3.1.1 Requirements

Your agent code must meet the following requirements:

- **Hardcoding is not allowed.** For example, the use of specific information on a location that is present in a map in your GOAL agent program is not allowed. See also the remarks in Section 4.3 about hardcoding.

- **It is mandatory to provide comments for all code** that you hand in. The comments should clearly explain what a code fragment does and why it has been implemented.

- **Make sure your program does not contain any commented code.**

Note that if your deliverable, i.e., agent code, does not meet these requirements this will affect your ranking; in case of clear violations of these requirements we will assign no stars at all (see below).
3.2 Deliverable

The deliverable for the individual assignment is:

- A .zip file `<netID>.zip` containing your agent program. The agent code must include comments that provide sufficient explanation of how the agent is supposed to work.

The deliverable has to be handed in on the blackboard page of this course.

3.3 Assessment

The assessment of the individual assignment will not directly influence your grade for this project but will be used to create a ranking. This ranking will be used to assign students to groups, as students with similar rankings will be put in the same group.

Your work will be assessed based on the agent program code and its run-time behaviour. This behaviour will be tested by evaluating the correct behaviour as described below. You can achieve a star for each of the 5 criteria that are described below. These criteria are evaluated in an incremental manner. In other words, a star for a higher (numbered) criterion can only be achieved when the stars for the lower (numbered) criteria have been achieved as well.

1. Compilation, Workers gather minerals, and “send once” percepts:
   - **Compilation**: The code compiles and is able to run (without errors) using the accompanying `.mas2g` file, where the right launch rules are applied and the necessary agent(s) are properly defined;
   - **Workers gather minerals**: At least one worker unit starts gathering minerals;
   - **Send once percepts**: Any “send once” percepts that are needed for the agent program are handled in the init module such that they appear in the belief base correctly. At least the percept `self/5` should be processed (even if not used) and any other percept that is necessary for gathering minerals.

2. Training workers and percepts:
   - **Training workers**: Define a Command Center, Nexus or Hatchery agent in the `.mas2g` and program it in such a way that it trains multiple workers;
   - **Percepts**: Any percepts that are needed for the agent program (i.e., besides “send once” percepts) are handled in the event module such that they appear in the belief base correctly at all times. At least the percepts `resources/4` and `status/6` should be processed (even if not used) and any other percept that is necessary for the requirements above.

3. Minimal functionality:
   - **Gas collecting**: The worker units can build a structure to collect gas and collect gas from this building;
   - **Scouting**: One of the units will move towards the enemy base location and scout at least 1 enemy building;
   - **Task managing**: The worker units can divide the mineral and gas gathering tasks in such a way that there are 1 or 2 workers gathering gas (at least one and at most two) and 1 worker scouting the enemy base.

4. Building and Upgrading:
   - **Building**: The worker unit(s) build the following buildings:
     - **Terran**: At least one Barracks, one Engineering Bay and one Academy.
     - **Protoss**: At least one GateWay, one Forge and one Cybernatics Core.
3.4 Deadline is Strict

The deadline for handing in the individual assignment is strict. Solutions that are handed in after the deadline will not be graded. As the formation of groups depends on the assessment of the individual assignments, if you fail to hand in a solution on time, you will not be assigned to a group and you will not be able to participate in the group assignment part.

4 Group Assignment

During the group assignment phase, groups of 6 students need to program a team of Goal agents that each control a bot in StarCraft (see also Section 1). The main objective is to design and develop an agent system that is able to beat the final performance target, which can be found in chapter 4.2. Important learning goals during this phase are to design and develop an agent system that can effectively compete in a StarCraft match, to effectively work as a team, and plan this work and allocate tasks.

4.1 Approach

A scrum-based approach based on weekly “sprints” and allocation of tasks needs to be used in the group assignment phase. A sprint in scrum is a time-boxed unit of development effort, in our case a week. Scrum is an iterative and incremental agile software development methodology. Important aspects of this development methodology are continuous re-assessment and re-planning of the concrete tasks that a development team defines for itself, self-organization of the team by close collaboration of all team members, and daily communication among all team members in the project. A key principle of scrum is the recognition that during a project one can change one’s mind about what is needed (also called “requirements churn”) based on continuous evaluation of the product that is being developed. For the purpose of evaluating the software developed, it is important to have a working product at the end of each week that can be tested.

Each group should follow a basic schedule each sprint, i.e., each week, which is the basic unit taken as a sprint in this project. Each sprint has a planning event at the start of the week, during the week an execution phase in which the plan is executed by the team, and a review that evaluates and discusses progress at the end of the week.

Advice: Program your agent in an incremental fashion. First, create a program that complies with the requirements of star 1, then for star 2, etcetera. Save a version of each agent so you will always be able to hand-in a correctly functioning version for a certain star level (i.e., hand-in only one working deliverable). Note, however, that for example whilst implementing functionality for the third star, the criteria of the stars below (e.g., correct percept handling when introducing new percepts) should still be met.
4.1 Plan

At the start of each week, in a sprint planning event, the requirements and functionality (also called the backlog) and the corresponding set of tasks to be performed in that week needs to be agreed upon. The aim of planning is to identify the work for the sprint and to define the goals that should be achieved during the sprint. The main activities and results are:

- **Activity**: Specify new and refine existing requirements and functionality that will be implemented during the weekly sprint; 
  **Output**: List of required functionality.

- **Activity**: Identify which tasks are to be done during the weekly sprint by which team member; 
  **Output**: List of assigned tasks (who is going to do which tasks).

A few additional important remarks on how to execute the planning event:

- A planning event does not need to take a lot of time but should always have concrete outputs.

- Do not only identify tasks related to implementing functionality but also consider that testing is very important and takes significant time. Make sure you also plan to test the performance of your agent system (see Section 4.2).

- All requirements and functionality identified during the planning event must be documented on the Trac wiki.

- For all (assigned) tasks tickets must be created on Trac.

- Each team member on average should spent 14 hours per week. Divide the work equally and try to take into account how much time a task will take.

4.1.2 Execute

The activities during the weekly sprint are to actually execute but also to monitor and discuss progress of the tasks done by the team. The main activities and results are:

- **Activity**: Each team member should execute his or her tasks and monitor progress with own tasks; task status should be shared within the team: in case a task is completed (earlier) or difficulties are met that may prevent a task from being completed, the team must be informed; 
  **Output**: Updates of assigned tasks.

- **Activity**: Monitor and discuss progress within the team: evaluate whether assigned tasks can (still) be achieved within the sprint and make a decision whether to postpone or remove a task from the list of assigned tasks; also discuss whether there is room for implementing additional functionality and perform additional tasks during the sprint; 
  **Output**: Updated lists of assigned tasks and/or required functionality.

A few additional important remarks on how to execute tasks within the team:

- Individual team members should not spend a lot of time trying to solve a difficulty that they encounter themselves but ask for help from the team instead.

- Even though progress should be monitored and shared within the team during the spring, most time should be spent on completing tasks and implementing requirements! The activity is mainly there to not continue working on tasks when that does not make sense any more.

- Updates of assigned tasks must also result in updated Trac tickets for the tickets associated with those tasks.

- Updates of required functionality must result in corresponding updates of the Trac wiki.
4.1.3 Review

At the end of every week, in a review meeting the team needs to evaluate whether tasks are successfully completed and planned functionality has been implemented. The main activities and results are:

- **Activity**: As preparation, evaluate the performance of the system developed so far (see Table 4.2 in Section 4.2 for details). Also identify how your agent system can be improved;
  

- **Activity**: Review progress: what went well, what went wrong? How can the team improve? For each task that is not completed, decide whether to postpone (continue next sprint) or abandon the task; in case a task is continued, re-evaluate who should work on the task;
  
  **Output**: Updated lists of assigned tasks and/or required functionality.

A few additional important remarks on how to report system performance and conduct the review:

- Reports on system performance and a brief evaluation of this performance (what went well, what went wrong? why?) for every week must be documented on the Trac wiki.

- Updates of assigned tasks must also result in updated Trac tickets for the tickets associated with those tasks. For each task that is abandoned (i.e., a decision is made in the review meeting to not finish and remove the task), an explanation why the task is abandoned needs to be added to the corresponding ticket on Trac (which then should be closed as wontfix).

- Updates of required functionality must result in corresponding updates of the Trac wiki.

4.2 Performance Targets

In order to pass the project, the most important evaluation metric will be whether your agent system performs sufficiently well against the given test bots. It is the final performance of your agent system that counts here: Beat every race of the build-in AI on all 5 maps in a StarCraft match. Note that it might be harder to win on a big map than on a smaller map, so you should test your bot on each map extensively.

Concrete performance requirements will help you assess whether you are making the progress you should be making while developing your agent system. To this end, we have created the table below that you should use to evaluate every week whether you are on track and the performance of your agent system is improving as expected over time.

<table>
<thead>
<tr>
<th>Week</th>
<th>Minimum Required Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Make use of at least one build order which works on all 5 maps</td>
</tr>
<tr>
<td>4</td>
<td>Defend your base from the first 2 attacks of the build-in AI on all 5 maps</td>
</tr>
<tr>
<td>5</td>
<td>Beat test bot 1 on all 5 maps</td>
</tr>
<tr>
<td>6</td>
<td>Beat test bot 2 on all 5 maps</td>
</tr>
<tr>
<td>7</td>
<td>Beat test bot 3 on all 5 maps</td>
</tr>
<tr>
<td>8</td>
<td>Beat test bot 4 on all 5 maps</td>
</tr>
<tr>
<td>9</td>
<td>Beat every race of the build-in AI on all 5 maps</td>
</tr>
</tbody>
</table>

Table 4.2: Weekly performance requirements that should be met

To be precise, beating means that the game actually ends with your bot as the winner. If there is no winner for more then 30 minutes in-game time then it is a draw (a draw does not count as a win!).

Two remarks are important:

- A tie is not sufficient; you need to consistently beat the opponent. The win should not be due to luck, or, in other words, if we test your agents, our test should also result in a win for your agent system.
4.3 Functional Requirements and Sprints

- Do not underestimate what needs to be done by your team because we start with very easy to meet requirements in the first few sprints; things will quickly get much more difficult! Therefore, if you meet the weekly performance requirement don’t stop working, but work towards next week’s requirement! Continuously investing the required time and effort is crucial to successfully finishing the course.

4.2.1 Game Settings and Evaluation Setup

We briefly discuss the game settings that will be used to test and evaluate the performance of your agent system. Of course, you should test your system using these same settings but you can also test with different settings if you like. In the final competition run at the end of the project we will also use these settings.

- Game speed is set to 20 (other game speeds can be used for testing purposes).
- Your agent system must be able to perform well at the end of the project on all 5 maps named: (2) Benzene, (2) Destination, (3) Tau Cross, (4) Circuit Breaker and (4) Jade.
- The initial starting location of bots on a map is determined (randomly) by that map.
- The maximum duration of a game will be set to 30 minutes in-game time (which can take up to 15 minutes in real time).

4.3 Functional Requirements and Sprints

There are no strict functional requirements that you must have met. You are free to design your agent system the way you want to, as long as it is effective. In order to help you, we have provided several suggestions for each weekly sprint below (see Sections 4.3.1-4.3.7).

There are a few strict requirements that your agent system must meet. Every bot that you control is controlled by exactly one agent. It is allowed that your agent system uses additional Goal agents that do not control a bot but, for example, assist in the coordination between agents (see also Section 4.3.4).

Second, you must document the ontology used in your agent system on the Trac wiki. The ontology is a specification of the predicates and action names that are used within your code base. It should also specify the (type of) parameters associated with each predicate and action name. The purpose of the ontology is to communicate and share within the team which predicates and actions have been defined and specified and how they can be used. You should update your ontology every week and make sure it matches with your actual code base.

Third, hardcoding is not allowed. That is, it is not allowed to explicitly mention map specific items in your code base. This means that you are not allowed to use terms in your code that are different for every map and are otherwise received via percepts. For example, specific chokepoint locations cannot be explicitly mentioned in your code because these are different for each map. In case you are not sure, please discuss with teaching assistants. If doubt remains after discussion, the course supervisors will decide on the specific case at hand.

In the remainder of this section, we provide several suggestions of functionality that you can implement. We provide different suggestions for every weekly sprint during the project. You have 7 weeks to implement your agent system but week numbers start at 3 as we assume that the first two weeks will for the most part be used for completing the individual assignment. Every week it is your task as a team to continuously refine functionality already implemented, derive additional requirements, add new functionality, to identify specific tasks to implement functionality, test functionality, and to allocate these tasks within the team. See also Section 4.1.1.

Please remember that the performance targets in Section 4.2 are minimum requirements; your agent team will need to be able to play against other agent teams as well. Testing your agent team against other groups is certainly allowed (and even encouraged).
4.3 Functional Requirements and Sprints

4.3.1 Sprint Week 3: The Basics
This is the first week that your start coding on your agent system. We suggest you to implement the following functionality during this sprint:

- Basic resource gathering of minerals and vespene gas.
- The construction of the first structures and necessary units.
- The beginning of the task management between the workers.

The gathering of the resources and the construction of the structures are one of the most important parts of the game. The way you gather resources and construct buildings in the early part of the game will be a very important factor of your strategy.

**Important:** As a group, at the start of the project, you also need to agree on how you will work together and create a group cooperation contract. Make sure you discuss this and create the contract; for details, see the explanation for Appendix B in Section 4.4.2. This is also the time where you should set up and start using the project management tool Trac. Also maintain your group’s Trac wiki. This is a weekly recurring task which you should do at the beginning of every sprint after the planning meeting and at the end of every sprint after the review meeting (see also Section 4.4.3).

4.3.2 Sprint Week 4: Defend and Scout
In the second sprint week, we suggest you to implement the following functionality:

- Start producing attack capable units.
- Begin implementing a defensive strategy for the early part of the game.
- Start scouting the opponents base and learn what the opponent is doing.

In order survive the first waves of attacks a proper defence has to be set up. There are a lot of ways you may be attacked, for instance on smaller maps there will be a good chance that the opponent chooses to rush you, while on bigger maps your opponent will most likely first build up his economy. Thereby it is important to know what the opponent is doing and adapt your strategy based on the gained information gained by scouting your opponent.

4.3.3 Sprint Week 5: Attacking, Upgrading and Researching
In the third sprint week, we suggest you to implement the following functionality:

- A basic general attack strategy.
- Start researching Tech and Upgrade types from your structures.

When you’ve scouted the opponent and gathered an army together, than at some point you want to attack your opponent. It is important to know that the army you have is capable of winning from your opponent’s army and not lose your whole army in case you are not able to defeat him yet. Researching upgrade types can be a key factor in defeating your opponent’s army. When your opponent has an attack upgrade and you don’t, it will become a lot harder to defeat his army. The bigger your army becomes, the more important upgrades also become. However you could also for example quickly rush your opponent and overrun his forces before upgrades have any effect at all. It is very important to have a solid strategy when attacking your opponent. Having a strong army composition is an important factor of an attack strategy, likewise the way your army attacks together. Try to get the best out of every unit and make them work together to get an advantage over your opponent’s army.
4.3.4 Sprint Week 6: Improve attack strategy and Improve resources income

In the fourth sprint week, we suggest you to implement the following functionality:

- A more advanced attack strategy.
- Improve the resource gathering from your bot by expanding at some point.

Now that you have a basic attack strategy it is time to improve it a little bit. At this point you should be focussing on using micro with your units like letting them kite or use any researched Tech-types in a strategic way. Communication between your units can also be very important for letting them work together. To maintain a solid army count, it is important to keep up a stable economy at your base. When your army is big enough, the opponent’s army is less of a threat, so you might consider expanding to the nearest base location. The size of the map is also an important factor for when or why you should be expanding. When you have successfully expanded you can increase your unit production at a higher rate.

4.3.5 Sprint Week 7: Race and Map specific strategies

In the fifth sprint week, we suggest you to implement the following functionality:

- Strategies for specific units or races.

During this sprint, we suggest you to implement more specific strategies for more specific scenarios. For instance when facing a Zerg opponent in a small map, you might want to prepare for a fast attack from your opponent, where if you are facing a Protoss opponent you might want to check the map for any proxy attacks. Note that when you are playing against a random opponent, the enemyRace percept will only return: ‘Unknown’ until the first enemy unit is scouted.

4.3.6 Sprint Week 8: Test and Improve

In the sixth sprint week, we suggest you to implement the following functionality:

- Test your bot extensively to handle any scenario.
- Evaluate the tested scenarios and improve your bot based on the scenarios which it is not handling well.

The end of the project is near and your bot should already have multiple strategies. However covering all possible scenarios can be a tedious task to do. For this week, think of the most common scenarios you can think of and test how your bot handles these situations. Try to improve your bot for each situation it does not handle well.

4.3.7 Sprint Week 9: Refinement

In the seventh and last sprint week, we suggest you to finish up your project and test how your bot performs against the bots of other groups. Make sure that you have a stable version ready which is capable of defeating every race of the build-in AI on each of the 5 maps.

4.4 Deliverable Guidelines

An overview of deliverables and associated deadlines can be found in Table 1.1. The main deliverables are the implemented agent system, the report, and online documentation made available through the project management tool Trac. Performance targets and requirements for the implemented agent system have been discussed in Sections 4.2 and 4.3. In this section, we briefly discuss handing in deliverables in 4.4.1, some guidelines for writing the report in 4.4.2, and expectations on and guidelines for using Trac in 4.4.3.
4.4 Deliverable Guidelines

4.4.1 Handing in Deliverables

All versions of the report need to be made available as pdf files on git (accessible via Trac) and at the given deadlines (see Table 1.1) should also be handed in via blackboard. It is not sufficient to only make them available as attachment on a Trac wiki page. All code of the implemented agent team needs to be made available as a snapshot (e.g., as a ZIP-file) on git. In addition, links to deliverables must be created and made available on the Trac wiki homepage of your group. Finally, we will also look at the wiki pages and tickets you created and maintained on Trac, so make sure these are presentable.

4.4.2 Report

The report should document the product, i.e., the implemented agent system as well as the process, i.e., how the group cooperated as a team and how the project was managed. The report is a design report that documents product design as well as the process that led to the design.

Please write the report in English. There is a strict page limit of 15 pages that you are allowed to use for chapters 1-6. In case this limit is not met, this may impact your report grade negatively. We have provided some suggestions for the number of pages you use for individual chapters as a rough guideline below. In the remainder we provide guidelines for the content of report chapters and appendices; see also the overview in Table 4.3.

<table>
<thead>
<tr>
<th>Report Part</th>
<th>Brief Content Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cover and Title Page</td>
<td>Report title, group member names, TA, Mentor Technical Writing, date</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>List of chapter and section titles in order</td>
</tr>
<tr>
<td>Preface</td>
<td>Acknowledgements and story how report came about</td>
</tr>
<tr>
<td>Summary</td>
<td>Executive summary</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>Introduction</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Program of Requirements</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Analysis of the StarCraft Environment</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Design of Agent System</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Testing Approach and Results</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Conclusions and Recommendations</td>
</tr>
<tr>
<td>References</td>
<td>Literature references in APA-style</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Ontology definition</td>
</tr>
<tr>
<td>Appendix B</td>
<td>A report on the process (project management and group cooperation)</td>
</tr>
<tr>
<td>Appendix C</td>
<td>Account of hours spent on the project per group member</td>
</tr>
</tbody>
</table>

Table 4.3: Content Overview of Report

Chapter 1: Introduction  Questions to be addressed:

- What is the reason for your design project?
- What is the aim of your design project?
- What is the broad outline of the report?

Suggested length: about 1 page. Use literature references to motivate why there is a need for this project (from a gaming company’s point of view, see also below).

Imagine that you are writing this report for an IT manager of a gaming company that asked you, as a group, to program agents that can effectively control bots in StarCraft. The manager would like these agents to demonstrate some intelligent control of the bots in the game and to show nice game play. This manager knows about software engineering and does not need an explanation

You should use 11pt font size, and small margins on a4 pages.
of basic techniques used in developing software. The main purpose of this report, rather, is to convince this manager that the agent software that you deliver upon his request is meeting his expectations. The manager would also like to be informed about the limitations of your product and how it could be improved.

Chapter 2: Program of Requirements Questions to be addressed:

- Which reasons did you have for prioritising requirements using the MoSCoW method?
- What are the minimum requirements/functionalities (‘Must haves’)?
- What are the advanced functionalities (‘Should haves’)?
- What are the extended functionalities? (‘Could haves’ and ‘Won’t haves’)

Suggested length: 1-2 pages. Use literature references that support your prioritisation of requirements/functionalities according to the MoSCoW method.

The only hard requirement set for this project is a performance target and evaluation and grading looks mainly at software related aspects such as readability of code. Here you should formulate requirements that concern the design choices that you make and the behaviour of your agent system. For example, a requirement could be that your agents that have a particular role use specific powerups that you explicitly mention here. Of course, this would require your agent system to implement different types of roles. See for more examples Section 4.3.

Chapter 3: Analysis of the StarCraft Environment Questions to be addressed:

- What are the characteristics of the scenarios (see below) that you designed (describe at least ten scenarios)?
- How do you justify your design choices?

Suggested length: 3-4 pages. Use literature references to motivate your design choices, i.e., to motivate why you design your agent system as you plan to.

As part of the design of your agent system, you need to create testable scenarios for a StarCraft match that will help you analyse and develop a system that will perform as expected, i.e., according to what you specify in each scenario. Test results for each scenario that you specify should be discussed in Chapter 5. For example, one scenario you can analyse would be that one of your bots encounters another unit and gets chased by an enemy bot that tries to kill your bot. Another example would be that a bot perceives a enemy and needs to decide whether to kill him or not. A scenario should specify: (1) relevant percepts, (2) actions used, and (3) a basic story that specifies how these percepts and actions are used.

Chapter 4: Design of Agent System Questions to be addressed:

- What is the organizational structure of the agent team (agent roles)?
- How do you justify the organizational structure?
- What is the strategy of the agent team?
- How do you justify the strategy?
- What is the validation of the agent team?
4.4 Deliverable Guidelines

Suggested length: about 3 pages. Use literature references to motivate the (organizational) structure of your agent system and to motivate the strategy that you chose to implement.

Consider the following functionality and design of your system in this chapter: (1) Building units, (2) building build order, (3) defence and attack strategy, (4) navigation, (5) roles and organization structure of the agent system, and (6) communication and coordination. Important is that you motivate the design choices that you made.

Chapter 5: Testing Approach and Results  Questions to be addressed:

- What kinds of bugs were identified by the tests (present examples)?
- How were the bugs identified?
- What are the problems that occurred during testing?
- How do you evaluate the functioning of the implementation?

Suggested length: 4-6 pages. Use literature references that explain the methods you used to identify bugs.

This chapter should document your test approach and concrete test results. Results for each scenario that you specified in Chapter 3 should be analysed here, i.e., what went well, what went wrong during (initial) tests of the scenario; use decision tables (cf. course Software Quality and Testing) but also discuss results in text. Consider the following aspects related to your testing approach in this chapter: (1) frequency of testing during a sprint, (2) kinds of tests, (3) who is performing the tests.

Chapter 6: Conclusions and Recommendations

- What was the main question (should be clear from the introduction)?
- What is your answer to the main question?
- What are the main arguments for the answer?
- What is the significance of your conclusions?
- What are your recommendations for the implementation process?
- What are the reasons for giving your recommendations?

Suggested length: 1 page.

Consider the lessons you learned during the project. Reflect on how the IT manager should take these into account when implementing his own agent-based system.

Appendix A  This appendix should document the ontology used in your agent system (which should match both with the code and the wiki page that documents your ontology). An ontology item consists of the following items:

- predicate or action label;
- specification of the (type of) parameters of the predicate or action;
- informal explanation of the meaning in natural language;
- a specification of the use of the label (knowledge, beliefs, etc.).

The purpose of specifying an ontology is that all group members that work on developing the same agent system have access and share the definition of predicates. This should facilitate shared coding on the same system and prevent duplication effort on defining (similar) predicates.
Appendix B  This appendix needs to consist of the group cooperation contract and a discussion and reflection part on how your group cooperated and managed the project, i.e., organized and planned tasks, dealt with problems encountered, etc, and a statement on how you used Trac, a brief reflection on your individual contribution, and an evaluation of the project. We briefly discuss each item below:

1. Group cooperation contract The purpose of the group contract is to agree on how the group will work together during the project. You should document specific agreements at the start of the project on:
   - Task Allocation Agree on how tasks, e.g., writing code, testing, etc., will be assigned to group members during the project. Consider which project roles you will have during the project (chair of meetings, secretary, planner, wiki page maintainer, task coordinator, review meeting chair, etc.), who will take on these roles, and how this will affect task allocation. Also consider different types of tasks that have to be done, e.g., putting meeting minutes on Trac, preparing meeting agendas, testing, etc.
   - Decision Making You will need to make many decisions during the project about who will do what, how a group member compensates for absence, how to make design choices and revise design choices that did not work out, and other decisions on how to resolve problems. Think about the way you as group want to make these decisions. What is important to you? That the group reaches consensus? That everyone participates equally in a discussion? If so, how can you make sure that this happens? Or do you care more about reaching the best decision according to most even though not everyone agrees? Will there be a special role for the meeting chair in which decisions are made?
   - Presence and Availability Every group member must be present during scheduled hours but also available outside these hours (contact hours make up only part of the hours you need to spent on the project!). In case someone is late, or for some special reason cannot be present or available, how will you deal with this as a group? How should other group members be notified about this? How will your Teaching Assistant be notified? How do you make sure this does not happen repeatedly? And how does someone compensate for this? Also agree on how you can make sure that everyone contributes equally and what you do if this is not the case. Which channels of communication will you use?
   - Meetings and Schedules Every group must have at least one planned meeting with agenda with its Teaching Assistant every week. Of course, a Teaching Assistant can provide help and feedback also at other moments; just ask for it. Agree on how you will conduct meetings, which issues should be discussed, and how issues for discussion can be put on the agenda. Keep in mind that at the start of every sprint you need to have a meeting to plan what you will do every week, and at the end of every sprint you need to have a meeting to review what has been achieved during the sprint and how you can improve. Minutes of each of these meetings must be made available on the Trac wiki of the group.

2. Evaluation of the cooperation during the project In this part of the appendix, discuss at the end of the project whether your group cooperated and worked together as agreed at the start of the project. Did everyone meet agreements made, would you revise agreements in hindsight, would you have allocated roles and tasks differently, did everyone contribute equally? Briefly document which tasks were performed by each group member and how much effort was spent on it by that group member. Discuss group issues that you needed to resolve and how you resolved these.

3. Use of Trac Briefly describe how you use the project management tool Trac. Discuss in particular the use of tickets and how that helped you to manage the project; did it help
you meet deadlines, achieve performance targets? Did you create a ticket for every agreed action item in a meeting. Given your experience at the end of the project, how would you use Trac in a next project?

4. Individual Reflection At the end of the project, each group member must reflect on his or her individual contribution to the project. Address the following questions. Which contributions did you make to the final product? To organizing the group effort? Did you actively participate in decision making and discussions within the group? Which roles did you have during the project? What went well and what can be improved? What did you learn and what would you do differently in a next project? You should use 0.5 a page to summarize your reflection (use a heading Summary) and 1.5 page to discuss these questions in more detail (use a heading Reflection).

5. Evaluate the project As a group, provide constructive feedback on the project organization, supervision, design, and software used in the project. Try to be as concrete as possible. The more concrete suggestions you can provide for resolving issues that you feel are present can help us improve the project!

Appendix C Document per week how many hours each group member spent on each task.

4.4.3 Project Management Tool Trac

During the project (group assignment), you should use the project management tool Trac. Trac provides a shared workspace for a group. Files and documents can be shared and made available to other group members or Teaching Assistants. Trac also provides a version management system. Trac is aimed at supporting teamwork, but it also supports management by the assistants and supervisors. It offers the following tools that facilitate working together effectively:

- **Wiki pages.** You should use these pages to make meeting agendas available, share meeting minutes, document the ontology, and other relevant information on your design, product, and group cooperation (e.g., create a page for your group contract). Wiki pages are easy to create, view, and modify. A version management system that is able to compare different versions of a page is available as well.

- **Tickets.** You should use tickets for each assigned task and for reporting bugs and/or feature requests that should be assigned to someone. Closing a ticket means that no more work will be invested in the task; specify the reason for closing a ticket (fixed, etc). Tickets should have an owner, and a due date. It is recommend to reference tickets in commit messages.

- **Version Management Tool A git repository is available for each group that you should use for version management of code as well as the report (and possibly other documents).**

We briefly discuss a few specific requirements on the use of wiki pages. Each group should have a main **front page** that provides the following information:

- Basic project information: the name of the group & a group logo, a link to your group’s repository, and the mail address your group uses.

- Group member information: name, student number, and email address.

- Deliverables: links to the project deliverables (made available via your group’s repository, not as attachments of a wiki page).

- Ontology: a link to your wiki page that documents your ontology.

- Meeting agenda and minutes: a link to a wiki page with agenda and corresponding minutes of all of your weekly meetings (organized from most recent at the top to oldest at the bottom).
4.5 Grading and Evaluation of Your Performance

The following information should be present in meeting agendas:

1. Meeting date and time.
2. Opening and fixing agenda items.
3. Approval of meeting minutes of previous meeting & discussion of progress on agreed action items. Discuss how you will resolve any issues with action items.
4. Announcements.
5. Other items.
6. Any other issues that group members want to mention or discuss.
7. Closing.

Your meeting minutes should at least report the following:

- Date, time, group members present, name of meeting chair and secretary.
- A brief report for each agenda item that was discussed. Explicitly mention decisions that were made.
- List of action items: which actions were agreed upon during the meeting; make sure you also create tickets for each action item and link these to the item in the meeting minutes on the wiki page.

4.5 Grading and Evaluation of Your Performance

The grading and evaluation will be based on the following main items:

- product (see Section 4.5.4)
- process (see Section 4.5.5)
- individual contribution (see Section 4.5.3)
- report grade (information will be made available on blackboard)

In the sections mentioned, more detailed information is provided on how the grading will be performed and how a final grade is determined (see Section 4.5.2). We briefly discuss in the next Section 4.5.1 how the competition and project review meetings at the end of the project contribute to the process of determining a final grade.

4.5.1 Competition and End of Project Review Meetings

At the end of the project, agent systems will compete against each other in a project competition for bonus points and review meetings with each group will be conducted in which a final group grade will be determined.

Competition A project competition is organised at the end of the project in which all agent systems compete with each other. Games between agent teams will be displayed using beamers for viewing. A similar set up as described in Section 4.2.1 will be used. However, the competition will start with a ‘qualifying round’, for which random pools of groups will be made (win: 3 points, tie: 1 point). The top two of each pool will then continue to the ‘final tournament’, which will be a knock-out competition. In this stage, if there is a tie at the end of a match, the group with the highest number of kills will be declared the winner. The group whose agent system wins the competition receives a price as well as, of course, eternal fame. Bonus points can be earned if your agent system is ranked amongst the top teams (see Section 4.5.4).
4.5 Grading and Evaluation of Your Performance

End of Project Review Meetings  Your deliverables and performance during the project will be discussed with each group at the end of the project in a review meeting (also see Section 4.5.2). Each review meeting will take 30 minutes. Before this meeting a preliminary group grade will be determined which can be adjusted based on the discussion during the review meeting. Reasons for doing so include clarification provided in the discussion on items that were not clear or a lack of knowledge about deliverables of (specific) team members.

4.5.2 Grading

The final grade for your work is based on two components. The first component consists of a group grade for your work on designing and developing the agent system. The second component is based on a report grade for the report that you write.

The group grade for your work as a team on the agent system again is based on two components. It is based on a product grade for the agent system that you delivered (the “product”) and a process grade for the way you organized the effort that you have put into creating the product during the group assignment part (the “process”). In principle, each student receives the same (group) grade, but in specific cases there may be reasons to deviate from this (see Section 4.5.3).

Only if product and process grades are both sufficient, a group grade is determined as a weighted mean of these grades as follows:

\[ 0.7 \times \text{product grade} + 0.3 \times \text{process grade} \]

In case grades are insufficient, a retake is offered but only if both product and process grades are at least a 4. The nature of the retake is determined by the supervisor(s). If a retake is not offered, you will need to redo the project the next year.

The final grade is a weighted mean of the group and report grade determined as follows:

\[ 0.8 \times \text{group grade} + 0.2 \times \text{report grade}. \]

Again, both grading components must be sufficient to pass the project.

4.5.3 Individual Contribution

The evaluation of the individual contribution of team members is based on the part of the report that reflects on individual participation during the project, the results of the peer review, (tickets on) Trac, and feedback from teaching assistants. No explicit grade will be determined for the individual contribution but this evaluation will be taken into account to decide whether to deviate from the group grade for individual team members.

4.5.4 Product Grade

The product consists of the final agent system that has been implemented in GOAL, documented in the accompanying report (parts related to the product, which includes everything except for content reporting on the process).

The minimum requirement for passing the project is that your agent system meets the performance target for the last sprint as documented in Section 4.2.

The product grade is based on grades for the following parts:

- An agent system grade is determined based on code quality, taking into account readability, quality of action specs and handling of percepts, proper use of GOAL constructs and design guidelines, quality of code comments, and on the performance of your agent system, taking into account, e.g., the behaviour generated by the bots in a game, robustness.

- A product report grade is determined based on the quality of report sections, taking into account criteria such as clear style of writing, depth of description, and motivation of choices made in design and implementation.
• A bonus can be obtained based on the ranking in the competition at the end of the project (see Section 4.5.1). The winner of the competition will get 1 bonus point and runner-up groups ranked 2-4 (the semi-finalists) will get 0.5 bonus point.

If the minimum requirement is met, the **product grade** will be determined as follows:

\[
\text{Grade} = \frac{\text{agent system grade} + \text{product report grade}}{2} + \text{bonus}. 
\]

Grades > 10 are not possible; in that case, the grade will be determined to be a 10.

### 4.5.5 Process

The process concerns the organization, planning, approach, and cooperation within a group. The **minimum requirement** for passing the project is that all group members contributed equally to delivering the product.

The process grade is based on grades for the following parts:

- the **process report** (Appendix B, see Section 4.4.2).
- how effectively the group has used **Trac**.
- **feedback** of teaching assistants.

The fact that you within your group will have encountered problems does not need to negatively impact your grade. In fact, the way you resolved problems may positively influence your grade. It is important to **identify and handle problems as soon as possible**. Important other factors are the **level of reflection** on what went well and what went not so well in your report and how you as a group ensured **everyone contributed equally**.

### 4.5.6 Who is evaluating what?

Table 4.4 indicates who will evaluate a deliverable; below, TA stands for Teaching Assistant.

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Teaching Assistant</th>
<th>Supervisor</th>
<th>Group Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implemented system</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Report (product)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Process</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report (process)</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Trac</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input TA</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual contribution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report (process)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Peer review</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trac</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input TA</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final interviews</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4.4: Who evaluates which deliverable?

### 5 Organization

This section discusses supervision (Section 5.1), how to get started as a group (Section 5.2), and which course materials are made available to you (Section 5.3).
5.1 Supervision

You, as a group, have the main responsibility for making this project a success. You are responsible for organizing and managing your own project, for monitoring the progress you make, for the quality of the product that you develop and the decisions that you make during this development, and for making sure that each group member contributes equally to the development of the product and the group process. Having said that, we are available for supporting you where we can. You can always consult with your Teaching Assistant and supervisors whenever you have questions or other issues to discuss.

5.1.1 Role Assistants and Supervisors

Each group will be supervised by a Teaching Assistant. Your assistant will provide feedback on the deliverables that you create (product) and on how you organize yourselves and manage the project (process). A Teaching Assistant, for example, will indicate if you are not working effectively. Make sure you involve your Teaching Assistant in important discussions and keep the information on your Trac wiki pages up to date so your Teaching Assistant can adequately assist you.

Each project session all Teaching Assistants and one supervisor will be present. Each group should interact with their Teaching Assistant every session (make sure you arrange for a minimum of 15 minutes for discussion and interaction with your Assistant). In addition, each group’s Assistant will be present at least once a week during a group meeting in order to understand how you manage the project and to be able to provide feedback on whether you are making the progress that you are expected to make.

If your group has any questions or is in need of assistance on any matter, the Teaching Assistant is the first person to contact. Do not hesitate to contact them and ask questions. If, for example, a group observes that one of its members is free-riding, the group should first try to discuss this with the student who is not putting in enough effort. Ask your Teaching Assistant for help when this has no effect, or when the student does not keep up to his end of the bargain repeatedly (e.g., no improvements are made). Do not wait until the end of the project with reporting problems (group, planning, technical or other)! That is too late, and we may not be able to help you in finding a solution any more for your problem.

A Teaching Assistant can provide valuable support in helping you to get organized (process) but also provide advise on how to develop your system (product). Of course, which choices are made remains the group’s responsibility. Assistants will have regular contact with the supervisors, and, if necessary, a supervisor will provide support to a group together with the Assistant in order to resolve any issue.

If you or one of your group members is considering quitting the project, or if circumstances do not allow you to contribute sufficiently (any more), a supervisor needs to be contacted immediately. An appropriate solution will be found in consultation with all involved.

5.2 The Formation of Groups

The project group that you will be assigned to will be determined by the evaluation and assessment of your solution that you handed in for the individual assignment.

5.2.1 The First Project Group Meeting

The first meeting with your project group is a very important meeting to make a good start! Make sure that your Teaching Assistant is present during this meeting. This way, you can get introduced to each other and make some initial agreements on how to work together. Please note that most assistants will assist more than one project group and will need to fit your meeting into his or her schedule. Start preparing the meeting as soon as possible by reading this manual and consult other course materials (see Section 5.3) to understand what the project is about.
It is important to discuss the following items during your first group meeting; although we advise you to generally try to have short meetings, there are quite some topics to discuss during your first meeting and it may take more time:

- **Introduction**: Everyone should briefly introduce him/herself to get acquainted;
- **Expectations**: Have a round in which everyone indicates what they expect to get out of the project, what they like and/or dislike about working in a project group; be open and clear right from the start to prevent misunderstandings later;
- **Assistant**: The role of the Teaching Assistant (see also Section 5.1);
- **Process**: Start creating a group cooperation contract (see also Section 4.4.2);
- **Name and Logo**: Think about a name and logo for your group, and put this on your Trac wiki page (see also Section 4.4.3);
- **Plan**: start creating a plan and assign tasks to each group member (see also Section 4.1.1).

### 5.3 Course Materials

The following course materials are provided to you via (links on) blackboard:

- This project manual;
- The SC-GOAL manual, which documents the actions and percepts and other relevant information about the interface between SC and GOAL; this manual also explains the use of the **visualizer** that provides a 3D overview of the SC map;
- The SC game manual;
- The SC installation page (SCLINK); this page provides instructions for installing and running SC in combination with GOAL;
- Documentation on GOAL and IDE (http://goalhub.github.io/eclipse);
- Additional material for the Technical Writing course.

---

*Good luck and have fun!*
Original project description

F.1. Project description
This project aims at revising the existing Multi-Agent Systems project (1st year Computer Science Bachelor). Currently, Unreal Tournament is used in the MAS project. This bachelor project should create an Environment Interface Standard implementation for StarCraft which is intended to be used in the MAS project in the future. Additionally, a plan has to be made for the MAS project that specifies what students should do in the 10 weeks of the project. To test the bots of the students and the functionality of the environment, three bots (one of each race) have to be created. Research has to be done into what debugging functionalities can be added to the game to help students in debugging their agents. Design decisions include: What API to use to connect to the game, what terminology to use in the various percepts and actions, how to implement EIS with the API, how to display the functionality of the game in such a way that writing agents is as easy as possible, and how to port the multi agent systems to the game so it can be used in the StarCraft AI competition?

Research is needed to verify that the revised project would be feasible for first year students to complete in 10 weeks.

F.1.1. Final products
- A fully working StarCraft EIS environment with according manual
- A student guide for the MAS project
- One (example) multi agent system per race (3 total)
- A debugging tool for StarCraft (focused on agents)

F.2. Company description
The Interactive Intelligence group aims to engineer empathy. To achieve this aim we combine research from different fields:

- Agent-based reasoning to develop cognitive frameworks for various domains of applications, focusing on robots, human-agent/robot teamwork, serious gaming, agent-based simulation and negotiation.
- Computational Intelligence to study machines that can learn through interaction.
- Perceptual Intelligence.
- User-centered design.
Figure G.1: The UML of the action package. Only two actions are shown, all actions have the same structure.
Figure G.2: The UML of the percept package. Only two percepts are shown, all percepts have the same structure.

Figure G.3: The UML of the draw package.
Figure G.4: The UML of the perceiver package.

Figure G.5: The UML of the development tools.
Figure G.6: The UML of the units package.
H.1. First feedback

De code van het systeem scoort bijna 4.4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Duplicatie.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er duplicatie te vinden tussen bijvoorbeeld ‘isValid’-methode van Land.java, Patrol.java, Move.java, AttackMove.java en RightClickPosition.java. Het is aan te raden om dit soort duplicaten op te sporen en te verwijderen.

Tot nu toe is de code van het systeem schoon en gestructureerd schreven en over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

H.2. Second feedback

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid ongeveer gelijk is gebleven.

Dat de totaalscore niet hoger is hoeft niet te betekenen dat jullie niets aan de onderhoudbaarheid gedaan hebben. Bij Duplicatie, wat in de feedback op de eerste upload als verbeterpunt werd genoemd, zien we een duidelijke verbetering. Jullie hebben daar niet alleen de genoemde voorbeelden aangepast, maar er ook voor gezorgd dat er in de nieuwe code geen nieuwe duplicaten zijn geïntroduceerd. Dat de totaalscore desondanks niet omhoog is gedaan heeft twee oorzaken: ten eerste zaten jullie al vrij hoog, waardoor je meer werk moet doen om nog verder te stijgen. Ten tweede is het normaal dat als een systeem gaat groeien het meestal wat minder onderhoudbaar wordt. Jullie verbeteringen hebben die daling weer ongedaan gemaakt, waardoor je “netto” dus ongeveer gelijk bent gebleven.

Naast nieuwe code hebben jullie ook nieuwe tests toegevoegd, de verhouding testcode/productiecode ziet er dan ook net als bij de eerste upload prima uit. Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.
The project

**Project title**: StarCraft MAS-project

**Client Organization**: TU Delft

**Final Presentation**: Friday July 1, at 12:00 PM

**Description**:
The Interactive Intelligence section of the TU Delft would like to replace the game UT3 that is used in the Multi-Agent Systems project (MAS-project), which is a part of the bachelor curriculum of computer science. The game that has been selected for this purpose is StarCraft. StarCraft is called the ‘next challenge in AI development’ by Google. To create a multi-agent system (MAS) for StarCraft, a connection is needed from the programming language GOAL to the game. During the research phase, we evaluated existing solutions and chose an API to work with. Additionally, the current MAS-project was researched. One of the biggest challenges encountered was the performance of the system. Because of the large number of agents present in the game, the agents became unresponsive. This issue was overcome by optimizing as much as possible and collaborating with the client to find solutions. We used an agile development methodology together with test-driven development which enabled us to quickly prototype the system and identify bugs as soon as possible. There were many unexpected concurrency issues due to the API in conjunction with GOAL. These issues were solved by sacrificing some performance for robustness, with careful considerations not to hurt the system. The final system that was created connects GOAL to StarCraft and is tested using a variety of unit test and coverage frameworks. The product can be used in next years MAS-project.

The project team

**Harm Griffioen**

**Interests**: Artificial intelligence, cyber security, reverse engineering, algorithms.

**Roles**: Repository manager, Lead Java developer, GOAL developer, Quality assurance, Tester.

**Danny Plenge**

**Interests**: Artificial intelligence, Machine learning.

**Roles**: Sprint manager, Lead GOAL developer, Java developer, Documenter, Lead tester.

Contact information

**Team members**  Harm Griffioen  (harmgriffioen@gmail.com)

Danny Plenge  (dannyplenge@gmail.com)

**Coach**  Dr. K. V. Hindriks  (Interactive Intelligence, TU Delft)

**Client**  Ir. V. Koeman  (Interactive Intelligence, TU Delft)

The final report for this project can be found at: http://repository.tudelft.nl
Bibliography


