Analyzing and Applying Agent Oriented Programming Methods for Teaching Purposes

Marc Dekker Bsc.

July 13, 2012

Abstract

In this thesis we discuss ways to improve teaching methods for agent oriented programming. The research approach consists of several steps. First data is gathered from groups of students participating in a first year agent programming project that uses the Unreal Tournament 2004 environment. The stronger and weaker groups are analyzed in a qualitative way to establish differences in programming styles between the stronger groups and weaker groups. The focus of the analysis is on software quality and code constructs. This analysis establishes what the key success factors of the stronger groups are. The results of this analysis are applied while developing the HactarV2 agent system for the Multi-Agent Programming Contest together with other students. During the Unreal Tournament project we found that stronger groups spend more time and energy on testing than weaker groups. It was also established that they pay more attention to documentation, software quality factors and the style of their code. Finally a code pattern similar to the Strategy design pattern showed up much more often with the stronger groups than with the weaker groups. When these methods were applied during the Multi-Agent Programming Contest project it was established that some of the documentation methods do not work as intended and that code style sometimes has to be sacrificed for the sake of efficiency. On the other hand the use of testing was proven again and the use of the code pattern turned out to work better than expected. The fact that the HactarV2 team won the Programming Contest is a clear indication that these methods work. The conjecture of this thesis is that the quality of an agent system can be improved by the right approach to testing and documentation during the development and by applying code patterns that make behavioral control easier. Teaching these approaches and patterns to students will improve their skill level of agent-based programming.
Preface

This Master research is in the field of agent based Artificial Intelligence and is done at the Delft University of Technology (TUD) department of Electrical Engineering, Mathematics and Computer Science (EEMCS) working from the unit of Interactive Intelligence (II).

This thesis covers research on agent based programming but it could not have been done without help for which we want to thank a few people. First we thank the direct thesis supervisors Koen Hindriks and Birna van Riemsdijk who were good enough to cope with bad spelling and sometime irregular life. Then there are the students that participated on and the student assistants that helped with the Unreal Tournament project. Without their work the research could not have been done. Also not to be forgotten are our team mates on the HactarV2 team Pieter Hameete, Michiel Hegemans, Sebastiaan Leysen, Joris van den Oever and Jeff Smits. Thank you for the wonderful experience and good luck in your continued education.
1 Introduction

Our research focuses on the field of Agent Oriented Programming (AOP) which was introduced as a paradigm by Shoham [1]. Since the first definition of AOP a lot of research and development has been done in this field. This has led to a number of AOP specific languages and language extensions. Unfortunately the teaching methods for these subjects are not developed at the same speed and are lagging behind. This means that students that are starting studies in the field of AOP are not receiving the highest quality curriculum that they could be receiving and as a result they cannot develop agent-based software at the highest quality possible.

To find out how we can improve on teaching methods we focus on the software quality factors understandability and testability and on the use of code constructs and patterns. By focusing on these things we study which development methods produce a high quality Multi Agent System (MAS).

The first part of the research takes place during a project on agent based programming in the Unreal Tournament 2004 [2] (UT2004) environment for first year students computer science. During this project the student groups had to create a number of deliverables besides which we asked them to fill out three questionnaires. Based on these deliverables and questionnaires we select the four strongest and four weakest groups. These groups are then analyzed on the use of software quality factors and code constructs and patterns by means of a checklist. The results of the strong and weak groups are then compared to find out what they do differently. The most important lessons learned during the UT2004 project are:

- A high test frequency with different kinds of test improves quality of a MAS.
- The use of a code pattern similar to the Strategy design pattern greatly improves the behavior and control flow of an agent.
- Up to date documentation during the development process improves the quality of the end product.

The lessons learned from the UT2004 project are then used by applying them in the second part of the research is a project to create a MAS for the Multi-Agent Programming Contest [3] (MAPC). This project was a collaboration between six students from the TUD to develop the agent system that would result in the HactarV2 [4] MAS. MAPC called for an agent team to control 10 bots in a competitive turn-based Mars exploration environment. By applying the lessons from the UT2004 project during the MAPC project we could immediately experience how these methods worked. The HactarV2 team took first place in the contest by winning all matches. With the experience of creating HactarV2 and the results the team had in the contest it was possible to point to development methods that produce higher quality results.

We conclude our research with a set of suggestions that can be incorporated in teaching methods to improve the education of students in the field of AOP. The suggestions range from using the Strategy pattern to improve the behavioral control over an agent to focusing more on testing during agent based development.
The thesis is organized as follows. In section 2 is described what the research questions are and what research methods were used. In section 3 the UT2004 project is described, how all the data on development methods is collected during the project, how it is analyzed and what lessons are learned from the analysis. Section 4 describes the MAPC project and in which way the lessons from the UT2004 project were applied during the project. This is followed by the results from the project and the collection of the new lessons that the project taught us. In section 5 the lessons and analysis of both projects are used to answer the research questions and ideas are supplied for future research related to this research.
2 Problem Statement

2.1 Research Targets

Our main research question is: How can we improve teaching agent-based programming. We approach this question by studying a first year programming course on agent based programming, focusing on the questions:

- Are there any programming constructs in the programming language that better performing students use that are not used by lesser performing students?
- Are there any software quality factors that better performing students focus on and that lesser performing students tend to ignore?

If we can come up with positive answers to these questions we can improve the Agent-Based Programming curriculum by giving more attention to the concepts and the software quality factors that seem to work.

2.2 Research Method

The first choice to be made is which Agent-Based Programming Language (APL) is used with our research for which we selected GOAL\(^1\). There are several reasons for this choice. First of all the Delft University of Technology’s (TUD) department of Computer Science has an agent programming project for first year students. This project uses GOAL as development language and uses the real-time shooter game Unreal Tournament \([5]\) as environment. This is an ideal project to collect data on used concepts and software quality factors. A second reason for using GOAL is that we used this APL in earlier research. And finally in the summer after the student project ran, the TUD supported a student team to compete in the international Multi-Agent Programming Contest (MAPC) organized by the Claustal University of Technology. As the TUD team was going to use GOAL for their entry we decided to join the team and use the team as a second project. In this project the team applied the results of the first project to see whether using some of the identified concepts and focusing on some identified software quality factors would actually work. More will be explained about the setting and environment of the first year project and MAPC project in sections 3 and 4 respectively.

The first of the two projects, the first year student project\(^2\), will be used to observe the students and gather information about the methods and tools they use.

We determine what good performing students are by studying the results of their work and by having the students fill in questionnaires at different stages of the project. Studying their work means looking to see what code concepts and constructs are used and looking at software quality aspects. For the software quality aspects the focus will be on understandability and testability which have been identified in earlier work and which are defined and explained in section 2.3.3. Studying code concepts and constructs we look for code patterns used

---

\(^1\)See section 2.3.1

\(^2\)Section 2.3.2 contains information about research done based on earlier iterations of this project.
by multiple groups of students to see whether good performing students either use patterns that lesser performing students do not use or whether they use the same patterns in different ways. We also analyze whether these patterns conform to any commonly used design patterns or anti-patterns as described in section 2.3.4. Questionnaires are handed out during this project to help to measure the programming skill of the students throughout the project.

One of the quality factors that Kitchenham and Pfleeger [6] define and that is focused on is testability. As will be explained in the next subsection this quality factor covers the degree to which a piece of software supports the testing of said piece of software. The work of both stronger and weaker students is studied to determine whether their code can easily be tested. We determine not only whether the MAS supports full function testing but also how much work it would be to test particular facets of the system. By comparing the testability and testing methods of the below and above average students it is determined if and how testing contributed to the quality of the software.

During the UT project the student groups had a number of documentation methods they were asked to use to document the MAS they will be developing. These documentation methods consisted out of a version control system, a Wiki page for day to day information, an ontology and a final report about the development process.

There was time between the end of the UT2004 project and the start of the second project, the MAPC project, to analyze the data gathered from the UT2004 project as it took place several weeks after the first project finishes. So it is possible to use the development methods and tools found during the analysis of the UT2004 project in the MAPC project. By using the development methods and tool together with the other team members during the development of the Multi-Agent System (MAS) for the MAPC we have a chance to see whether these methods and tools are not only usable in a different environment but also whether they again lead to good results. At the end of the project we analyze how the various tools and methods worked and determine whether they functioned as well as during the first project.

2.3 Related Work

2.3.1 GOAL

Throughout this research the projects used GOAL [7,8] as their APL. GOAL is a rule based agent language based on the Belief, Desire, Intention paradigm like Jason [9] and 2APL [10]. Where most of these APL’s see goals as plans GOAL sees them as declarative concepts. Although this is not a complete paradigm shift it is clearly different from most other Belief Desire Intent (BDI) based APL’s except 2APL. GOAL differentiates itself from 2APL by not having explicit plans but works with if then rules to decide on what actions the agent executes in which situations making the plans implicit.

GOAL as described by Hindriks et al. [8] uses Prolog as a Knowledge Representation language and is constructed as follows. The multi agent system exists in an environment and consists of one or more GOAL agents. A GOAL agent consists of six sections which are listed in code fragment 1. The first section contains the knowledge rules, the second contains the agent’s initial beliefs. The third section contains the agent’s initial goals followed by the section containing
the program rules. The fifth section contains the action specification and the last section contains the rules to handle percepts.

```plaintext
main: stackBuilder {
knowledge{
  tower([X]) :- on(X, table).
  tower([X,Y|T]) :- on(X, Y), tower([Y|T]).
}
beliefs{
  on(b, table). on(a, b).
}
goals{
  on(a, table), on(b, a).
}
program{
  # define misplaced(X) a-goal(tower([X|T])).
  if misplaced(X) then move(X, table).
}
actionspec{
  move(X,Y) {
    pre{ clear(X), clear(Y), on(X, Z), not(on(X, Y)) }
    post{ not(on(X, Z)), on(X, Y) }
  }
}
perceptrules{
  if bel(percept(on(X, Y)), not(on(X, Y))) then insert(on(X, Y)).
  if bel(on(X, Y), not(percept(on(X, Y)))) then delete(on(X, Y)).
}
}
```

**Code Fragment 1: A GOAL agent program**

Using the initial beliefs and the action rules a GOAL agent will try to accomplish its goals. This will be done by using the believes, knowledge and goals to see what actions can be executed and then to select one of the possibilities and execute it. While operating the belief base of the agent can change by the actions that are executed and by percepts from the environment.

The mental state of a GOAL agent consists of goals, beliefs and knowledge. The goals are what an agent wants to achieve and the beliefs and knowledge are what it believes about its environment. The difference between knowledge and beliefs is that knowledge in the knowledge section is static and unchanging whereas the beliefs in the beliefs section are dynamic and may change at runtime. The belief base is used to keep track of the current state of the environment and the knowledge base contains what the agent knows about how the environment works and what conclusions it can draw using beliefs from the belief base.

In the program section you can describe when actions are to be executed. In the actionspec you can write down which beliefs change because of the execution of an action and whether there is anything that has to be checked before an action is executed. Lastly in the perceptrules section you can tell the program what to do with percepts coming from the environment and how to handle
messages passed to your system.

Code fragment 1 shows the sections of a GOAL agent program. This example is not mentioned in the Multi-Agent Programming books. It is part of the example code that is part of the GOAL installation. It also contains some example code for each section. The first line contains the name of the agent: stackBuilder. On the second line the knowledge section starts with knowledge rules that define the concept of a tower. The belief section notes the beliefs where block a is on top of block b which is on the table. The goals section has a goal that a should be on the table and that b should be on a. In the program section a macro is defined that tells the agent what it means for a block to be misplaced and then uses this definition to make a program rule to move a block when it is misplaced. The move action is described in the actionspec section. To move a block it and its target should be clear and it should not already be on its target. Afterwards the block is no longer on its original location and is now on the new location. Lastly the percept rules are used to match the agent’s beliefbase to the environment.

Since its first description the GOAL agent language has undergone a lot of improvements and additions. One of the most influential additions is the use of modules [11].

In GOAL a module is a context triggered subsection of the program section. When an agent enters a module it will focus completely on the module until it exits it again. Modules can be specified in the program section along with normal program rules and can even be nested. When you define a module you give it a context which is a mental state condition. As long as this condition remains true the program will only execute program rules defined inside the module. When a belief from the context is no longer true or the goal gets accomplished the program will exit the module and continue to run as normal.

The modules concept was introduced to be able to add more structure to an agent program. Modules combined with the [order=linear] feature, introduced to eliminate non-determinism in both the whole program section and in modules, give agent programmers using GOAL a much greater control over the behavior that the agents shows.

GOAL is an open source BDI based APL that uses Prolog to represent its beliefs, knowledge and achievement goals. Developers can read more in the available documentation and use the example programs to understand the language.

2.3.2 Effective Developers

Some work has already been done on how programmers work with Agent Oriented programming languages. A good example of this is the work of van Riemsdijk and Hindriks [12] that focuses on how programmers of different familiarity with the GOAL language use the APL.

They asked three programmers with varying degrees of familiarity with GOAL ranging from one programmer with only a few lessons to an expert programmer to write a GOAL agent. The agent was set in a Blocks World environment [13] and had to be able to perform adequately in various scenarios in that environment.

The goal in a BW environment is to move blocks into a given goal configuration. A BW situation starts with an initial positioning of the blocks on the
table and the agent knows the Goal state that has to be reached. It is then the
agent’s task to move the blocks around to reach the goal situation preferably in
as few moves as possible.

After the programmers handed in their programs an analysis was done of the
programs, not only to see how well they worked but also whether and how the
programmers used the various constructs in the GOAL language. The researchers
observed some interesting facts. The first was that the use of different goal
constructs increases with the experience of the programmer. The second is
that a programmer either uses a lot of action rules or uses a lot of rules in the
knowledge base.

What we consider the most interesting observation is that there is no relation
between the experience of the programmer and the efficiency of his programs in
this research. For this simple program the programmer with the least experience
had comparable runtime results as the programmer with the most experience.
This raises the question whether this only happens in this simple environment
or will it also happen in more complex systems? Or could it be a case of chance
where the beginning programmer was working beyond his level. To find answers
to these questions more research is needed with a bigger group of test subjects
and multiple more complex environments.

Hindriks, Van Riemsdijk and Jonker [14] followed up on the work of Van
Riemsdijk and Hindriks by looking at patterns in Agent Programs. For this
study they collected data from a project for students. First year students in
project groups with five members had to write a team of GOAL agents to com-
pete in an Unreal Tournament 2004 [5] competition. Thirteen groups partici-
pated in the project which provided a lot of data to analyze.

Upon analyzing the various teams the authors found some interesting facts
about the students’ use of the GOAL language. One of the things they remarked
is that most of the teams used modules for the implementation of the different
agent roles that a single agent can take. But the teams also showed a lack of
knowledge about some of the implicit behavior of GOAL like $a$-goal($x$) that says
that there has to be a goal $x$ but that $x$ is not yet true.

They also found that students had a lot of duplicate code between various
agents and that some teams converted their messages to beliefs and then deleted
the messages which made for a lot of extra work for the agent. This may be
an indication that both the language and the way the language is taught to
students can be improved.

As we already stated above the work of Van Riemsdijk and Hindriks [12]
raised the question whether inexperienced GOAL programmers create results
comparable to that of experienced programmers. Unfortunately this new work
does not answer this question since it only analyses work of programmers with
comparable experience and training.

2.3.3 Software Quality

There is a vast amount of data that needs to be analyzed because of the scale
of the UT project. This project started with 111 students, 91 of which took
part in the group phase and were distributed over 17 groups. To structure the
results of the findings software quality concepts are used as a focus during the
group work of the UT2004 project and the MAPC project.

In this study we talk about software quality. But this raises the question:
what is software quality. In this section it is explained what we mean by software quality and how it applies to our research. Firstly note that the first step in software quality is a well designed programming language. This is itself difficult and has some special issues [15]. But in this section the focus is on software quality in development.

A quick definition of software quality is the term fitness for purpose. This means that software can be seen as high quality if it does what it is supposed to do. From this it follows that no matter how well your code is written or how great some features are, if the software does not do the work it was made for it is bad software.

In general software quality is split into two parts [16]: Quality of Design and Quality of Conformance. Quality of Design means that the design fulfills all the requirements that are put on the system by the specifications, but also adheres to some quality factors which will be explained later. Quality of Conformance means how well software conforms to the design. This definition shows that software/code quality is not only the responsibility of the programmer but of the designer as well. This means the designer has to make sure that his designs and models actually show that the system functions correctly in the situation it is developed for.

For the software quality definition we identify so called software quality factors. These are non-functional requirements. Though the factors do not directly contribute to the fitness for purpose they are very important for the quality.

List 1 contains the generally accepted software quality factors as described by Kitchenham and Pfleeger [6]. In the rest of this section it will be explained what the various factors are and how they apply to the AOP paradigm. And why this research focuses on understandability and testability.

<table>
<thead>
<tr>
<th>List 1 Software quality factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understandability The purpose of parts of the system are clear.</td>
</tr>
<tr>
<td>Completeness All parts are present and fully developed.</td>
</tr>
<tr>
<td>Conciseness There is a minimum of excess information.</td>
</tr>
<tr>
<td>Portability Can be run on various hardware and software configurations.</td>
</tr>
<tr>
<td>Consistency Internally the system is uniform. For instance variable names.</td>
</tr>
<tr>
<td>Maintainability Easily updated and expanded.</td>
</tr>
<tr>
<td>Testability Ease of testing and debugging.</td>
</tr>
<tr>
<td>Usability Ease of use.</td>
</tr>
<tr>
<td>Reliability Performs consistently.</td>
</tr>
<tr>
<td>Efficiency Runs without wasting resources.</td>
</tr>
<tr>
<td>Security Protection of data.</td>
</tr>
</tbody>
</table>

The first factor on list 1 is understandability. This factor covers how well a developer can understand the software code which covers the clarity of the code itself but also that comments are inserted in the code to explain the workings and that separate documentation is present that clarifies how the software works. The documentation present should be focused and clear for its intended target group. This means that development documentation should be clear to a developer but not necessarily to a layman while the user manual should be
understandable for the intended user, whether he/she is a layman or an expert.

The factor of completeness measures the presence and full development of all the parts of the code. This includes correct and complete references to parts of external libraries where necessary.

The quality factor of conciseness covers reduction of excessive or redundant information and code. Though often reduced to the practice of minimizing the lines of code in a system, focusing too strongly on conciseness can create conflicts with other factors like understandability.

Kitchenham and Pfleeger [6] describe portability as the ability of software to be transferred from one hardware and/or software environment to another. This can be for example from one operating system to another or from one chipset to another.

Consistency is the factor that covers the need to have uniformity within a software system. This covers all aspects like consistency in variable naming but also the use of terms in documentation and the layout of the interface.

The scope of maintainability is the ease of updating and expanding an existing software system. This not only means that the code should not be too complex but also that it should be documented and that the software should be designed for additions. Maintainability shares areas of responsibility with understandability. The difference between the two is that where for understandability it is often best to keep something as easy as possible, maintainability sometimes requires something to be more complex to make it easier to change at a later time.

The ability to test the software is covered by the testability factor. It is not only important that the tests are correct but also that during the design of the system the developers take testing into account to make sure it is testable. Testability together with maintainability are the factors that are very important in the area of bug detection and fixing. Good testability improves the chance of detecting a bug while good maintainability makes it easier to remove the bug.

A software system has to be used by its users. This affects such things as the human-computer interface no matter whether it’s graphical or not but also takes manuals into account. This is all covered by the usability factor. Though software should be usable when it is released this is not enough. As Kitchenham and Pfleeger [6] note it should be usable over a period of time. This is where reliability comes in. Reliability covers the working of the system over its lifetime but also the ability to work under the conditions it is intended to.

To be efficient software should not waste resources like time or memory. Not only does the efficiency factor cover these things it also covers areas like the use of network bandwidth and space.

The last factor on list 1 is security. This factor covers the need to protect the intended user from interference by unauthorized users. It covers things like authentication, encryption and data protection. This factor is often not only a matter of software and hardware but also of physical measures in the user’s environment though the need may be different from system to system.

It is important that every factor is taken into account from the start of the development of a system until the end of its lifetime. For instance the testability is something that should be taken into account during the development to make sure that the programmers can debug the program but also that it can be expanded later. This testing and debugging is something for which there are no standard tools for agent systems yet, though important first steps have been
taken in recent years for example by de Cerqueira Gatti and von Staa [17] and has been expanded upon since by others like Salamon [18].

All eleven software quality factors can be applied to AOP since they are not focused on a particular programming paradigm. But not all factors are equally important for this research because it is focused on improving the way developers construct systems using APLs and not for instance in what way systems can be made more secure. The focus will be on two particular factors. These are understandability and testability which are noted in list 2.

List 2 Software quality factors which the research focuses on

<table>
<thead>
<tr>
<th>Understandability</th>
<th>The purpose of parts of the system are clear.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testability</td>
<td>Ease of testing and debugging.</td>
</tr>
</tbody>
</table>

There are various reasons why the focus is on understandability and testability and not on the other nine factors in list 1. Completeness will not be considered because while there are minimum requirements to what the student groups in the UT2004 project have to hand in they are free in implementing any strategy they want. Both conciseness and maintainability will not be focused on as both projects are one time projects which will not be expanded on after the completion of the project though each phase of the projects expands upon the phase before. This means that these factors are not important to the developers as they are aware of that fact. So it is impossible to compare projects on these factors. We do not consider the portability, usability, efficiency and security factors as they do not help with the focus of this research in improving teaching methods.

We will also not look at reliability during our research because all the things that fell under this factor in the two projects are either done by the environment, the environment connection or by GOAL which means the developers have very little to do with it. Consistency will not be considered because what applied to reliability also applies to interface side of the consistency factor. This is because the interface falls under the environment. The other side of consistency is consistency within the code itself. For instance with predicate names. We do pay attention to this side of consistency but only as far as it is also a part of understandability.

Both understandability and testability are very dependent on the way the code is structured and how the concepts of the programming language are used. The next subsection focuses on how code is structured and the use of language concepts.

2.3.4 Code structure and language concepts

The way code is structured and how the language concepts are used can have a significant effect on the quality of the code. Over the years there have been a lot of methodologies designed to improve the way we develop software. One of these is the design pattern methodology that was first put forward by Beck and Cunninham [19] in 1988.

Since then the methodology has matured what resulted in a book called Design Patterns [20], which is used by most schools to help teach advanced software engineering. A design pattern is a template solution for common problems in software engineering. Although the original book features 21 patterns
many new ones have been developed since then and a few more books have been written to collect them.

The original work on design patterns was done for Object Oriented Development and was a great success. There are researchers that hope to use this successful methodology in the field of agent systems. They have described multiple agent specific patterns for instance the Pyramid Pattern by Kolp and Mylopoulos [21] and the Master-Slave Pattern by Ariyod and Lange [22] though both works also describe other patterns. A down side of so many patterns is that finding the right one can be a problem. It would therefore be practical to have a system to classify patterns so that finding a pattern appropriate to a problem is simple. A promising process to do this is described by Oluyomi, Karunasekera and Sterling [23] and uses an attribute based analysis to divide the patterns into four levels. After that the patterns are further separated per level in various categories based on a different set of criteria.

It is also interesting to note that patterns are not just a way to indicate good solutions but they have also been used to indicate bad situations. These so called anti-patterns [24] identify common mistakes and give them names. An anti-pattern is a clear sign that the system that contains it has a low degree of software quality.

But anti-patterns are not only found in the models or in the code of a system but also in the development process itself. To deal with these problems Winikoff and Padgham and Harland [25] describe a system that is based on simple ideas like removing complicated steps or sub processes from the development process.

During the analysis of the UT project we will be looking for the repeated use of code structures. Not only within the code of one agent system but also code structures that appear in the work of multiple student groups.

2.3.5 Conducting qualitative research

In this section we explain qualitative research and the reasons for using it in this thesis.

To determine what programming methodologies work for agent oriented Systems we have to find a way to evaluate them. Sudeikat, Braubach, Pokahr and Lamersdorf [26] discuss how to evaluate different programming methodologies taking platform specific factors into consideration. For our evaluation we need to find a research method that will give us insight in how the developers use the methodologies and why they give the results that they do. This will be done by applying qualitative research.

There are many ways to do research in the field of computer science, for example methods like empirical studies [27] or project based evaluations [28]. Qualitative research has not yet been widely used.

Qualitative research focuses not on the what, where and when but mainly on the why and how [29]. To do this kind of research the researchers keep their sample sets smaller and more focused than with other research methods. The qualitative method of research and explanation as described by Patton [30] is based on non-numerical data. To give an example of how quantitative explanation differs from qualitative we use both methods to describe a thermal image. Using the qualitative method we would show the image in such a way that the relationship of the different colours with a logical colour scale tells us
how the temperatures relate while the quantitative method would note numbers to show the temperatures in the image.

The origin of qualitative research can be found in the sociology and anthropology disciplines but has since then spread to other fields [31]. It has even made its first steps to the field of computer science [32].

One of the problems people have with qualitative research is the selection of a useful sample set as qualitative research needs smaller focused sample sets. The mistake researchers sometimes make is that they use the normal probabilistic sampling methods used for quantitative research. This is neither efficient nor productive as this generates large unfocussed data sets which are time-consuming and difficult to compare qualitatively. There are more naturalistic sample selection methods that can be used for qualitative research [33].

We selected the qualitative research method because our data set will be relatively small. It will be based on the work of seventeen groups of first year students and so will not suffice for a quantitative approach.
3 Unreal Tournament 2004 Project

The first project of our research is a fourth quarter of the first year in which students have to create a MAS. The goal for the students is to write a team of Goal agents to compete in an Unreal Tournament 2004 (UT2004) competition. They do this in groups of five or six students. In the year of 2011-2012 seventeen groups participated in the project which provided a lot of data to collect and analyze.

The environment for which the students had to program their Goal agents was the real-time multi-player shooter Unreal Tournament 2004 (UT2004). In this environment agents control game bots that move around on the map and battle against the other agent controlled bots. During the project the students have to program their agents to compete in a Capture the Flag match in which a team wins by capturing the enemy flag from their base and returning it to their own flag. An important requirement for the agents was that they could operate on any map and did not have any map specific strategies or code in them. To test this requirement three maps were used.

The UT2004 environment was connected through an EIS connection. This connection provides the MAS with the most current percepts each reasoning cycle and allows the agents to pass on high level actions to the game bot. The actions that the Goal agents can give to the UT2004 bots are very high level. The actions cover things like going to a specific place on the map, pursuing an enemy bot as long as the pursuer knows where he is and selecting what weapon to use. This means that the students do not have to focus on basic things like making separate steps with their agents or pulling the trigger of their weapons. Instead they can focus on the high level strategy used by their agents.

Subsection 3.1 describes how the UT2004 project is set up. In subsection 3.2 an overview is given of the data that is collected based on which the stronger and weaker groups are determined in subsection 3.3. Subsection 3.4 will then compare the code and development methods used by the stronger and weaker teams. Finally subsection 3.5 provides an overview of the lessons learned during the UT2004 project.

3.1 Project Setup

The Unreal Tournament 2004 project is split into two parts for the participating students. The first is an individual assignment that lasts a week and a half. The results of the individual assignment are then used to place students in groups that took part in the second part of the project which lasted five and a half weeks. This research uses data collected from both parts of the project.

To get more data from the project we also created three Questionnaires that the students were asked to fill in at various stages of the project. The following subsections explain what the students had to do in the various parts and what kind of data is collected from the various parts.

3.1.1 Individual assignment

At the start of the project students are not immediately divided into groups. They first are given an individual assignment that is graded on a scale of zero
to five. For the individual assignment the students are provided with a basic Goal agent that does nothing but run between the two flags that are present on the map. The code of this example agent can be found in appendix A. The assignment asked students to create a Goal agent that moves around the map and collects weapons, armor and adrenalin.

List 3 The five criteria set for the individual student assignment.

**Basic functionality** The agent compiles, boots without errors, moves around in the environment and has code for all percepts with comments. It should not contain any map specific data.

**Full actions** The agent contains the action specification for all actions available in the environment with accompanying comments.

**Item collection** When run for 3 minutes in the environment the agent has gathered the following pickups: armor, four new weapons and twenty adrenalin.

**Documentation** The code should be readable and be supported by comments. The report that goes with the assignment is decently written and contains the required information.

**Match winner** When run with five other agents in a 3 vs 3 match with three minutes of playtime the agent performs best or second best depending on the amount of pickups it was able to collect.

The individual assignment was used to test the individual programming skill of each student. After the deadline the assignments were graded. The grading method was based on a set of criteria. Submitted work was judged on the five criteria which are listed in list 3. The various criteria were judged separately and meeting one criterion is not dependent on meeting another. The number of criteria on which the student got a sufficient score are counted to give a total score. When a student handed in his work late he was penalized on the total number of criteria he met though this penalty could not reduce the total to zero if the work met at least one criterion.

Students with a one to five criteria could move on to the group part of the project while students with zero criteria received a grace period of twenty four hours to improve their work and to resubmit it. After resubmission students were not able to get more than a score of one criterion.

3.1.2 Group Assignment

After the individual assignments were graded the students were divided into groups. The division was based on the scores the students got for their individual assignments. Students with the same results were placed in the same group. The idea behind this is that the groups have students with comparable ability which helps to prevent individual weaker students doing nothing within a group of better students or a very good student doing everything for his group. The groups were assigned a Student Assistant to help and monitor them. They were also given a version control environment to keep their code and documentation.

The student groups were required to submit reports on the development process and the MAS they submitted, besides submitting the code for the agents. The final grade that the groups received was a weighted average of the grades the groups received for these reports and the MAS.
3.1.3 Competition
A competition was held to find out what student groups produced the best agent teams. Instead of running this competition behind closed doors the teachers held the competition as part of the project end meeting. The agent teams of the seventeen groups that participated in the project were set up in a knock out competition where their initial opponent was decided by preliminary testing by the student assistants against bots integrated into UT2004.

The winning group got a bonus point on their final grade, a medal and the eternal honor of winning. The three runners up got a half point bonus. For these reasons the competition was an important part for the student teams. Because of its knock out set up the results were not perfect but they gave a good indication.

3.1.4 Questionnaires
To gain more insight in the agent programming skill level of the students participating in the UT project three questionnaires were made, each with a specific purpose and each was handed out to all participating students. The first questionnaire was distributed on the first day of the first week of the individual assignment, the second was distributed at the end of the fourth week of the project and the third was distributed on the last day of the last work week of the project. The goal of the questionnaires was to gather more data on the programming skills of the students and their own view on their programming skills over the whole project. The questionnaires can be found in appendix B.

To get a base line on the perceived skills of the students the first questionnaire covers their own rating on their skill in both Java and Goal. The reason for asking about their experience with Java was to get a baseline on the data. When the students start this project they should already have had several courses that teach Java programming or require its use. This means the students should have a decent grasp of the Java language and a reasonable view on their skill in using it. In the questionnaire the students are asked how well they think they know each language and how much they like working with it.

The second questionnaire was used to get a base line of the actual Goal programming skills of the students and was handed out halfway through the project. To test the programming knowledge of the students the questionnaire contained twelve multiple choice and true/false questions. These questions cover the use of various language concepts like goals and beliefs. The concepts covered by the questionnaire are slightly above basic. This was done to get a more accurate representation of their skill level. Adding questions that almost all students would answer correctly would not yield much useful information.

The final questionnaire was used to get a second reading on the data of the first two questionnaires. It was presented to the students on the last normal project day in week 7. The third questionnaire covered both the students perspective on their Goal skills and another Goal skill test with twelve multiple choice questions. In this questionnaire there are no questions about the student’s perceived skills in Java as we are not interested in how their Java knowledge had improved over the course of the project.

The Goal programming skills questions of questionnaires two and three were directly linked. Each question in questionnaire 2 addresses a specific Goal pro-
to link questionnaire 3 to questionnaire 2 there is a question in questionnaire 3 about each concept that was addressed in questionnaire 2. This means that there are 12 questions for questionnaire 3 which covered everything that questionnaire 2 covered. The sequence of questions is changed between questionnaires 2 and 3 to make less obvious that the same kinds of questions are asked again.

The benefit of linked questions between questionnaires 2 and 3 is that the final number of correct answers per questionnaire can be compared for each student but also that the results between the linked questions for each student can be compared. The object of this is that it will not only give insight in which concepts individual students are weak but will also show if certain concepts or syntax have improved from one questionnaire to the next.

The goal of the questionnaires is to measure the programming experience and skills of the students participating in the UT2004 project. Some of the questions asked the students to rate their own skills. Feigenspan et al. [36] conclude that self estimation of students is a good indicator of programming experience. They recommend mixing these questions with questions that measure programming experience. We have conformed to most of these guidelines. We mix questions that test GOAL programming knowledge, with self estimation questions and questions about experience with mainstream programming languages.

Because the questionnaires are spread out over the seven weeks of the UT project it could not be expected that all participants would be present each time the questionnaires were distributed. As many students as possible were encouraged to fill out the questionnaires each time, with the aim of creating the greatest possible overlap between the questionnaires. The number of students that filled out each questionnaire are shown in table 1.

<table>
<thead>
<tr>
<th>Questionnaire</th>
<th>number of students</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>93</td>
<td>89.19 %</td>
</tr>
<tr>
<td>2nd</td>
<td>84</td>
<td>75.68 %</td>
</tr>
<tr>
<td>3rd</td>
<td>84</td>
<td>75.68 %</td>
</tr>
<tr>
<td>1st and 2nd</td>
<td>71</td>
<td>63.96 %</td>
</tr>
<tr>
<td>1st and 3rd</td>
<td>71</td>
<td>63.96 %</td>
</tr>
<tr>
<td>2nd and 3rd</td>
<td>80</td>
<td>72.07 %</td>
</tr>
<tr>
<td>1st, 2nd and 3rd</td>
<td>68</td>
<td>61.26 %</td>
</tr>
<tr>
<td>1st, 2nd or 3rd</td>
<td>111</td>
<td>100.00 %</td>
</tr>
</tbody>
</table>

Table 1: Number of students that completed a specific questionnaire and the percentage of all students that handed in any questionnaire

3.2 Data Overview

In this section we present the various data sources and what types of data each data source contains. Observations will be made about the data and we will discuss what these observations may mean.
3.2.1 Exam Results

Before the students start the UT2004 project they received a course on the basic principles of agent programming and the GOAL language. To complete the course the students had to pass an exam. Using the results of the exam allowed us to gain an insight in the starting level of the students that would be participating in the project.

The exam consisted out of seven questions with varying weights. The results of the various questions are listed in table 2. As noted in the table the total number of points that a student could get during the exam was 101. The grade of a student was determined by taking the total of the points they collected over all the questions dividing the score by 10 and then adding 1 and maximizing the result at 10. The grades are rounded to the closest half point. This is done in accordance to the grading system that is used throughout the TUD.

<table>
<thead>
<tr>
<th>Question</th>
<th>Max points</th>
<th>Average points</th>
<th>Lowest</th>
<th>Highest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>2.20</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6.05</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>6.45</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>6.30</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5.90</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>22.37</td>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>4.37</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>101</td>
<td>53.63</td>
<td>17</td>
<td>84</td>
</tr>
</tbody>
</table>

Table 2: Per question of the exam the number of points that could be gained, the number that was gained on average, the minimum and the maximum gained points.

Before the grades were rounded the average over all students was 6.36 with grades ranging from 2.7 to 9.4. The rounding process was generally in favor of the students with the average going up to 6.39 and the grades ranging from 2.5 to 9.5. The distribution of the exam grades of the 97 students who took the exam is shown in figure 1.

Studying figure 1 and considering that a six is a passing grade leads to observation 1.

Observation 1. Of the 97 students who took the exam 68 passed the exam.

This means that 29.9% of the students did not pass the exam though this did not prohibit them from participating in the project but it could be an indication that the project will be difficult for them.

In total 87 students that took the exam that year participated in the project of which three did not pass the individual assignment. The three exam participants that did not meet any requirements for the individual assignment also did not pass the exam.

3.2.2 Individual Assignments

Though it will not give a complete picture a lot can be learned from the agent that each student had to submit as part of their individual assignment. There
Figure 1: Number of students that had a specific final grade on their exam.

were a total of ninety-five submissions. The grades that the submissions received based on the scoring conditions from list 3 give us a start in identifying the above and below average students. Figure 2 shows how many students met a specific number of criteria.

Because meeting a criterion is unrelated to meeting any other criterion we can study how often certain criteria were met by the students. The chart in figure 3 collects the numbers and percentages on criteria met where the criterion numbering corresponds to the criterion numbering in list 3.

From the data from the individual assignment as presented in the graphs in figures 2 and 3 the following observations can be made.

**Observation 2.** Not all (69 out of 95) students met the Basic Functionality criterion.

This is interesting because it is by far the easiest to meet. Most of the work you need to do is also required to have a chance at the Item Collection and Match Winner criteria. To determine the reason for the observed trends we have to go through the individual assignment entries to look for clues. When analyzing the code of students who had at least one star it becomes apparent that those students that have not met the first criterion in general had neglected to include comments in their code or have missed percepts they should have handled. Especially the former is regrettable as code comments greatly contribute to the readability and maintainability of the code. This also suggests that these students will not use comments during the group part of the project which will reduce the programming effectiveness of the students.

**Observation 3.** A low percentage (34%) of students meet the Item Collection criterion.
Figure 2: Number of students that met a specific number of criteria.

Figure 3: Number of students that met each criterion.
There are many possible reasons for this low percentage. One of the possible reasons is that students do not take into account the possibility of the agents getting killed during a match. When looking through the code of the students this indeed proves to be the case but it is not the only reason. The second important reason is that the students do not make the agents dynamic enough to handle the presence of multiple agents in the map. They assume that everything they want to gather will be there which does not have to be the case. For instance an agent will wait until they can get armor which has just been picked up by another agent instead of getting other needed items. This means they lose a lot of time.

**Observation 4.** The Basic Functionality and Documentation criteria are met the most by the students, 69 and 64 out of 95 respectively.

The reasons for observation 4 become clear when you consider what the Basic Functionality and Documentation criteria entailed\(^1\). The Basic Functionality criterion requires the basic functionality of the agent which is the main target for the individual assignment and is an effective prerequisite for both the Item Collection and Match Winner criteria. The Documentation criterion requires the student to add comments to his code and make a two page report on the agent. From this it can be concluded that the fact that the Basic Functionality and Documentation criteria were met by most of the students was not because students put extra effort in meeting these criteria but that they were mostly part of the minimal work that students needed to do to gain access to the group part of the project.

### 3.2.3 Questionnaire 1

Now that we have established a preliminary baseline for the programming skill of the students, which would be reestablished with further questionnaires, we also want to see how the students actually perceived their own skill with Goal. Especially in comparison to how they perceived their skill with Java which is the programming language they have had the most contact with during their first year. Figure 4 shows the graph of the results of the comparison. The graph leads to the following observations.

**Observation 5.** A large majority of Students rate their Java ability higher than their Goal ability with 7 and 6 respectively being the most often chosen.

**Observation 6.** Even though they have a semester more experience on average with Java they mostly perceive their Goal skill to be only a little (one point) lower.

Observation 5 was expected before the project but that it would be such a small difference as noted in observation 6 was certainly not expected. When expressing these observations in numbers the results are even clearer: out of the 93 students that completed questionnaire 1, 69 rated their Goal skill lower than their Java skill compared to 9 students who perceived their Goal skill to be higher than their Java skill. But of the 69 students who rated themselves lower 37 perceived their skill to be only one point lower.

\(^1\)See list 3 for criterion descriptions.
Figure 4: Student count per perceived skill of GOAL and Java. 69 students rate Java higher than GOAL, 15 rate it the same and 9 students rate GOAL higher than Java.

Besides asking the students about their perspective on their skills the questionnaire also asked them about the usefulness of the tutorial courses for both Java and GOAL. Figure 5 shows a graph which notes the rating by the students on the usefulness of the tutorials for Java and GOAL. From this graph observation 7 can be made. The statistics show that students prefer Java over GOAL though this difference is very small.

**Observation 7.** Java scores somewhat better than GOAL on the usefulness of the tutorial but the difference is small with Java rating at an average of 6.8 and GOAL, rating an average of 6.5.

Another question asked in the first questionnaire was how the students rated the appeal of the programming style of both Java and GOAL. The graph that lists the resulting ratings can be seen in figure 6. Studying these numbers led to observation 8.

**Observation 8.** On average Java (7.8) scores higher than GOAL (6.5) in appeal of programming style.

Lastly the students are asked how well they thought they understand how to use the language elements of the two programming languages. Similar results to their perceived skill were expected. The comparison of the Java and GOAL scores result in the graph in figure 7. The results were as expected and led to observation 9.

**Observation 9.** A large majority of Students rate their Java ability higher than their GOAL ability with 8 and 6 respectively being the most often chosen and with average score of 7.5 and 6.1 respectively.
Figure 5: Student count per tutorial usefulness of both GOAL and Java.

Figure 6: Student count per appeal of the programming language of both GOAL and Java.
Figure 7: Student count per perceived skill of GOAL and Java. 70 students rate their skill of Java higher than that of GOAL, 17 rate it the same and 6 students rate their skill of GOAL higher then that of Java.

We think that the difference between scores for understanding the programming languages can be partially attributed to the relative inexperience of the students with the GOAL language. This is the same as with the perceived skill question.

3.2.4 Questionnaire 2

As mentioned in section 3.1 the second questionnaire consists of twelve multiple choice questions aimed at estimating the students level of skill with the GOAL language concepts. Figure 8 shows the number of times that each answer was given by students per question. The graph shows that most questions have dominant answers but it does not show whether the answers are correct. Therefore figure 9 shows the same graph again but this time showing only the correct answers.

Figure 9 shows that certain questions were answered much better than others as summarized by observation 10.

Observation 10. Questions testing very basic knowledge about GOAL, namely questions: 1, 7 and 9, have very good results (88%, 91% and 87% correct respectively) while question 2 has a very bad result (4%).

When looking at the questions and answers\(^1\) it becomes clear why question 1\(^2\) is answered correctly so often as it is a basic question about the file types

---

\(^1\)The questions and possible answers of questionnaire 2 can be found in appendix B.2.

\(^2\)In this section we only refer to the questions in section 3 of questionnaire 2
Figure 8: A comparison of the number of times each answer was chosen for each of the questions for questionnaire 2.

Figure 9: Per question of questionnaire 2 the correct answer and the number of students that correctly answered the question.
used when working with GOAL. Question 9 has a similar reasoning as it asked about a fundamental syntactic principle of the programming language.

Question 7 provides the students with code fragment 2 and asks the students to choose the correct pre-condition which means that the answer can simply be deduced from the given post-condition.

```go
goto (Route) {
  Pre {...}
  Post (not (state(X)), state(moving(Route)))
}
```

**Code Fragment 2:** The given code for question 7 of questionnaire 2. The student is asked to provide the matching pre-condition which is \(state(X)\).

While it is clear why certain questions have a high percentage of correct answers it is less clear why question 2 had such bad a result. Studying the reason of the bad performance on question 2 leads to observation 11.

**Observation 11.** A large number of students (76 out of 84) wrongly thought that the answer to question 2 was B.

The question asks the student to explain what code fragment 3 would do. As it is not allowed in GOAL to adopt a variable as a goal, this code produces an error when a programmer tries to compile it which was answer A. It is possible that a large number of students did not know this as it is an advanced language concept or that students did not read the code careful enough to spot the answer.

```go
if bel (received(X, Y)) then adopt(Y).
```

**Code Fragment 3:** The given code for question 2 of questionnaire 2. If run it will produce an error as it is not allowed to adopt a goal which is only the content of a variable.

3.2.5 Questionnaire 3

As questionnaire 3 combined parts of questionnaires 1 and 2 the progress of students could be monitored during the project. The first part consisted of a few questions to establish the level of the students GOAL programming skills as the student perceived it himself. The questions cover the student’s ability in programming in GOAL, how much the programming style appealed to them and how well they understood the language concepts used in GOAL. Figure 10 shows how the students answered these questions.

The graph in figure 10 shows that language understanding and programming ability follow each other pretty closely and both have clear peaks which gives the impression that the students rate themselves in about the same way. The ratings that the students gave to the appeal of the languages are however far more evenly distributed which shows that the students are more divided on this subject. The data behind the graph leads to observation 12.

**Observation 12.** The average appeal (5.4) of GOAL is much lower than understanding (6.9) and ability (7.0).
Figure 10: The number of students that rated themselves with a particular value for ability to program in, language appeal of and understanding of concepts of GOAL.

This reinforces the notion that students are generally less positive about the appeal of the GOAL language than they are about their understanding of its constructs and their ability to write programs in them.

In the second part of questionnaire 3 students were given twelve questions to test their skills at and knowledge of the GOAL programming language in the same way as in questionnaire 2. The graph in figure 11 shows per question how many students picked each answer. The graph does not however show how many chose the correct answer which is shown in the graph in figure 12.

Analyzing the results from questionnaire 3 leads to observation 13.

Observation 13. Questions testing very basic knowledge about GOAL, namely questions 1, 5 and 7, have very good results (85%, 94% and 94% respectively) while questions 2 and 8 have very bad results (25% and 17% respectively).

We then compared the percentages of the answers with the questions1 in the questionnaire to see whether the reason for these performances could be determined.

Question 1 asks the students to choose the correct precondition with the options being state(moving(X)), state(X) and state(pursue(X)). The correct answer is state(X) as that is the state that is negated in the post condition. As the answer is given in the code that goes with the question the high percentage of correct answers is not surprising and as all the students that answered incorrectly had chosen state(moving(X)) it seems likely that they picked the

---

1See appendix B.3
Figure 11: A comparison of the number of times each answer was chosen for each of the questions for questionnaire 3.

Figure 12: Per question of questionnaire 3 the correct answer and the number of students that correctly answered the question.
precondition they thought fitted the function name instead of what the code of the function required.

```plaintext
halt()
  pre(...)
  post(not(state(X)), state(halted))
}
```

**Code Fragment 4:** The given code for question 1 of questionnaire 3. The precondition should be `state(X)`.

The fifth question was a basic knowledge question that asked which of the three possibilities was not a file extension used by GOAL. As this questionnaire was performed at the end of the project it was actually more surprising that 5 students answered the question incorrectly then that 79 answered it correctly as students had used files of the two correct types during the project.

The high percentage of correct answers for question 7 was most surprising because while it is a basic question about syntax, the principal of macros is an advanced topic. The question asks the students to choose the correct syntax in code fragment 5 from `include, define, import`. The correct answer is `define` but while `import` is used to add external GOAL modules to the program `include` is not a reserved word in GOAL. The two most likely explanations for this are that either the students are more aware of macro’s than is anticipated or by eliminating `include` as Java code and `import` as another function the students could find the correct answer by reason. A combination of these two factors is also possible of course.

```plaintext
#include <stdio.h>
#define CONSTRUCTIVE_MOVE(X, Y)

a=goal(tower([X,Y|T]), bel(tower([Y|T]), clear),
( clear(X); holding(X))).
```

**Code Fragment 5:** The code provided for question 7 of questionnaire 3. In GOAL the `#define` syntax is used to create a macro.

After analyzing the questions with high percentages of correct answers we studied the questions with low percentages. Analyzing the data from question 2 led us to observation 14.

**Observation 14.** A large number of students (63 out of 84) wrongly thought that the answer to question 2 was A while no students picked option B.

The question provides the student with the line of code in code fragment 6 and asks the student to select what it does. The code in question looks like it will insert the belief Y if the precondition is fulfilled but it will actually produce an error. This is because it is not allowed in GOAL to use only a variable as an insert action nor is it allowed to use only a variable as a goal. With this in mind it becomes clear why all the students that had the incorrect answer picked option A but the question remains why so many picked the wrong answer. The answer to this question probably lies in one of two options. The first is that this error is an advanced code problem which is not encountered during normal work in GOAL. The second option is that students just did not read the question carefully enough to spot the error in the code.
if goal(Y), bel(received(X,Y)) then insert(Y).

Code Fragment 6: The given code for question 2 of questionnaire 3. If run it will produce an error as it is not allowed to insert only the content of a variable nor is it allowed to have only a variable as a goal() statement.

The last question with a very low number of correct answers is question 8 with only 14 out of the 84 students choosing option b. The question states: “All of the variables in the post condition of an action specification have to be bound in the precondition.” and asks the students if this statement is true or false. The answer is “false” as variables can also be bound in the parameters of the action. The best explanation for 70 students incorrectly thinking the statement is true is probably that they did not understand the statement completely. They probably thought that it meant that all the variables in the post condition should be bound, not particularly in the precondition but bound in the first place. As the question/statement is somewhat confusing this question should have been formulated in a different way to prevent this kind of confusion. Another reason could simply be they forgot or did not realize that variables could also be bound in the action parameters.

3.2.6 Group work

After the completion and grading of the individual assignment the students were formed into groups for the rest of the project. Though this meant that for the rest of the project the data set for programming code and methodologies would have only seventeen data points it would also mean that the methodologies used would be keyed to group work instead of individual development. As the groups were formed from students who met the same amount of criteria during the individual assignment we had an automatic baseline for the skill of each group. The groups are numbered from 1 to 17 with group 1 being the group with the students that met the most criteria during the individual assignment and group 17 the students that met the fewest.

Previous iterations of this project have shown that though groups with a higher number do tend to perform less well than groups with a lower number the group ranking is by no means a perfect indicator for how well the groups will perform during the project or at the competition at the end of the project. A good illustration of this can be seen in table 3 which shows the four groups that performed the best during the end competition. As shown in the table the winner is group 16 which was one of the underdogs at the start of the project.

<table>
<thead>
<tr>
<th>Place</th>
<th>Group(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winner</td>
<td>16</td>
</tr>
<tr>
<td>1st Runner up</td>
<td>10</td>
</tr>
<tr>
<td>Semi Finalists</td>
<td>6 and 13</td>
</tr>
</tbody>
</table>

Table 3: Top four groups in the competition

As part of the group portion of the project the groups had various deliverables. The first and foremost deliverable was the code for the agent team
that the groups had to develop. Besides this they had to hand in product and process reports at various stages of the development process. At the end of the project the students received a final grade per group that could be changed up or down per group member in case they had significantly contributed more or less than the other group members. The final grade was calculated by taking the weighted average of the product grade and the process grade and adding a competition bonus when applicable. The weights for the product and the process grade are seven to three respectively. The final grade for each group can be found in table 4. The table also includes the product and process grade for each group and the competition bonus for those groups that received it.

<table>
<thead>
<tr>
<th>Group</th>
<th>Product Grade</th>
<th>Process Grade</th>
<th>Bonus</th>
<th>Final Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8,75</td>
<td>9</td>
<td></td>
<td>8,8</td>
</tr>
<tr>
<td>2</td>
<td>7,75</td>
<td>8</td>
<td></td>
<td>7,8</td>
</tr>
<tr>
<td>3</td>
<td>6,75</td>
<td>6,5</td>
<td></td>
<td>6,7</td>
</tr>
<tr>
<td>4</td>
<td>7,25</td>
<td>6</td>
<td></td>
<td>6,9</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>6,5</td>
<td></td>
<td>6,9</td>
</tr>
<tr>
<td>6</td>
<td>6,25</td>
<td>6</td>
<td>0,5</td>
<td>6,7</td>
</tr>
<tr>
<td>7</td>
<td>7,75</td>
<td>8,5</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8,25</td>
<td>8,5</td>
<td></td>
<td>8,3</td>
</tr>
<tr>
<td>9</td>
<td>6,25</td>
<td>7</td>
<td></td>
<td>6,5</td>
</tr>
<tr>
<td>10</td>
<td>7,5</td>
<td>8</td>
<td>0,5</td>
<td>8,2</td>
</tr>
<tr>
<td>11</td>
<td>5,5</td>
<td>6</td>
<td>0,5</td>
<td>6,2</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>7,5</td>
<td></td>
<td>7,9</td>
</tr>
<tr>
<td>13</td>
<td>7,25</td>
<td>7</td>
<td>0,5</td>
<td>7,7</td>
</tr>
<tr>
<td>14</td>
<td>7,25</td>
<td>7</td>
<td></td>
<td>7,2</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>6</td>
<td></td>
<td>6,7</td>
</tr>
<tr>
<td>16</td>
<td>8,25</td>
<td>7,5</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>17</td>
<td>5,75</td>
<td>4</td>
<td></td>
<td>5,2</td>
</tr>
</tbody>
</table>

Table 4: Per group the final grade is shown with all the sub grades that make up the final grade.

The grades in table 4 give a clear overview of how the groups performed over the course of the project. When studying the sub-grades that make up the final grade it can be expected that there is a pattern to the grades where the groups that received a bonus in the competition would score better in general but observation 15 clearly notes that this is not the case.

Observation 15. There is no connection between getting a competition bonus and good performance in other sub-grades in any other way (ranging from \( p = 0.39 \) to \( p = 0.84 \), significantly higher than 0.05).

This lack of a connection might seem strange at first but this lack of a connection is actually supported by the way each of these grades is established. The process grade is determined by a combination of the quality of the groups’ process report and how well they handle the project/team process. The product grade is formed by taking the average of the sub grades given for the quality of the Product Report and the quality of the implementation of the agent team. In table 5 the product grade and its sub-grades are shown for each student group.
When comparing the groups that received a bonus because of the competition observation 16 can be made about the implementation grades they received.

**Observation 16.** Groups that received a bonus did neither uniformly have a higher Product Grade, with an average of 6.95 versus 7.21 for all groups, nor for any of its sub grades with the highest difference being 0.29.

The sub-grade that has the closest connection to the competition bonus is the implementation grade but where the competition bonus rewards groups that perform well the implementation grade judges the quality of the written code which means that it does not judge performance beyond the very basic functioning.

<table>
<thead>
<tr>
<th>Group</th>
<th>Implementation</th>
<th>Product Report</th>
<th>Product Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.5</td>
<td>9</td>
<td>8.75</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>7.5</td>
<td>7.75</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>6.5</td>
<td>6.75</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7.5</td>
<td>7.25</td>
</tr>
<tr>
<td>5</td>
<td>6.5</td>
<td>7.5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>6.5</td>
<td>6</td>
<td>6.25</td>
</tr>
<tr>
<td>7</td>
<td>8.5</td>
<td>7</td>
<td>7.75</td>
</tr>
<tr>
<td>8</td>
<td>8.5</td>
<td>8</td>
<td>8.25</td>
</tr>
<tr>
<td>9</td>
<td>6.5</td>
<td>6</td>
<td>6.25</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>8</td>
<td>7.5</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>5</td>
<td>5.5</td>
</tr>
<tr>
<td>12</td>
<td>7.5</td>
<td>8.5</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>7.5</td>
<td>7.25</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>6.5</td>
<td>7.25</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>8.5</td>
<td>8</td>
<td>8.25</td>
</tr>
<tr>
<td>17</td>
<td>6</td>
<td>5.5</td>
<td>5.75</td>
</tr>
</tbody>
</table>

Table 5: Per group the product grade is shown with all the sub grades that make up the final grade.

The realization that performance in the competition is not linked to the other sub-grades received by the groups makes observation 17 less unlikely but as winning the competition requires efficient and quality code it could still be expected that group 16 would have the highest product grade.

**Observation 17.** Even though group 16 won the competition\(^1\) they did not get the highest product grade which was received by group 1.

The explanation for the fact that group 16 has a lower product grade than group 1 can be found in the sub-grades of the product grade that can be seen in table 5. The table shows that groups 1 and 16 both received an 8.5 for the quality of their implementation but group 1 received a 9 for their product report while group 16 received an 8 and as the product grade is an average of the implementation and product report grades the product grade of group 1 is higher.

\(^1\)See table 3
Maybe even more interesting than the groups that had high final grades are the groups that scored low. When looking in table 4 for the groups that had a low final grade observation 18 is made.

**Observation 18.** Group 17 is the only group with a failing grade for their process grade.

The reason for the low process grade of group 17 cannot be easily found in the numbers and data so far presented but requires the studying of the process reports made by group 17 and the student assistant remarks that go with them. While reading these documents it becomes clear that while group 17 started off better than was expected for one of the lowest scoring groups at the individual assignment everything fell apart near the end of the project. At the end of the project group members stopped performing assigned tasks and stopped communicating. This problem was not solved by the group members themselves and resulted in deliverables to be handed in late and incomplete. Chief among these was the final process report. When the time came to determine the group grades it was clear that their process was not good enough for a passing grade and was set to a 4 and the group got an opportunity to make up the grade during the summer.

### 3.3 Identifying Student Performance

Throughout the UT2004 project we wanted to be able to compare the programming methods of above and below average students. Before actually comparing the stronger groups with the weaker groups it first had to be determined which groups were which. Throughout the project six data sources are established that could help rank the student performance.

The six data sources are the exam results, the individual assignment, questionnaires 2 and 3, the competition results and the final grade. Over the following sections each data source is discussed ending with establishing a final ranking based on a combination and comparison of the various data sources. This final ranking will be the one used through the rest of this research. Although both questionnaires 1 and 3 ask the students to rate their goalability we will not use this data as a data source. The reason is that the data for both the questionnaires 1 and 3 are very homogeneous. Both questionnaires had almost exclusive ratings of 7 and 8 on the subject of perceived skill. Together getting almost all the votes in both cases. This makes differentiating between the groups nearly impossible.

As there were seventeen groups it would have been simple to name the middle group in the ranking average and call everything above this above average and everything below this below average. But this would result in large groups of data which might show very little difference as the difference between the eighth ranked group and the tenth ranked group could be very small. The choice was made to expand the size of the average group with four groups in either direction for each ranking to prevent this. This meant that the top four groups are designated above average and the bottom four are designated below average. This results in a reasonable sized data set to compare.
### 3.3.1 Exam Results

The first indication of the skill level of the students is given by their results of the agent programming exam. Of the students that took the exam 84 also participated in the project. To get a grasp of what are above/below average groups the average grades of the students in each group are collected. The resulting ranking can be seen in table 6 besides the number of students from each group that took the exam in the preceding quarter.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Avg</th>
<th>Members</th>
<th>Rank</th>
<th>Group</th>
<th>Avg</th>
<th>Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>7.83</td>
<td>5/5</td>
<td>10</td>
<td>4</td>
<td>6.42</td>
<td>5/5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7.66</td>
<td>5/5</td>
<td>11</td>
<td>3</td>
<td>6.37</td>
<td>5/5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>7.16</td>
<td>5/5</td>
<td>12</td>
<td>16</td>
<td>6.03</td>
<td>6/6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7.14</td>
<td>5/5</td>
<td>13</td>
<td>14</td>
<td>5.97</td>
<td>5/6</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>6.74</td>
<td>5/5</td>
<td>14</td>
<td>13</td>
<td>5.88</td>
<td>5/6</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>6.66</td>
<td>5/5</td>
<td>15</td>
<td>15</td>
<td>5.73</td>
<td>4/6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>6.58</td>
<td>5/5</td>
<td>16</td>
<td>17</td>
<td>5.64</td>
<td>5/6</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>6.54</td>
<td>5/5</td>
<td>17</td>
<td>11</td>
<td>5.06</td>
<td>4/5</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>6.52</td>
<td>5/5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Group ranking based on average exam grades with number of students that attempted last exam.

When looking closely at table 6 it is clear that with a few exceptions most groups ranking is close to the group number with an average difference between group number and rank of 3.25 and a correlation between the ranking and group number of 0.65. Using only the exam results to determine which groups were the above average four and which were the below average four would result in the selection of groups 1, 5, 6 and 10 as above average and groups 11, 13, 15 and 17 as below average. The above average groups are separated from the other groups with a grade difference of at least 0.4 while the below average groups are separate from the average by only a 0.09 grade difference. The distance between the below and above average groups is 1.26.

### 3.3.2 Individual Assignment

The second data source is the results of the individual assignment. Section 3.1.1 describes how students have to make an individual assignment before starting the group part of the project. It also shows the criteria that the assignments were graded on in list 3. The students were then placed into groups based on their score with the highest scoring students placed in group 1 and the lowest scoring students in group 17.

This system gave an instant ranking of the student groups as they could simply be ranked based on their group number with 1 being the best and 17 the worst. To get a better idea of how each group scored an overview of the average score of each group can be seen in table 7. From these averages it is clear that the middle group (9) is made up of students who had a score of 2 or 3 while the top 4 groups consist out of students who had satisfied at least 4 criteria and the bottom 4 groups only had students who had only satisfied one criterion.

Basing the selection of above and below average group only on the results of the individual ranking would result in using the students from group 1 to 4 as
above average students and the students from groups 14 to 17 as below average groups.

3.3.3 Questionnaire 2

Our third data source was the second questionnaire and to be more exact the goal skill questions it contained. When the students had filled out the questions we had a number of correct answers per student between 1 and 12. To be able to use the number of correct answers to identify good groups it would not be enough to just pick the highest result for each group because one good student does not make a successful group. Neither would be looking at the lowest scoring student be right because one bad student does not mean the whole group is going to be below average. Instead the average result of the students within a group is used. The overall average number of correct answers for all groups is 6.98. To find the below and above average groups a ranking was made of the groups based on their averages. This ranking is noted in table 8 together with the average scores.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Average</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7.8</td>
<td>5/5</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>7.8</td>
<td>5/5</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>7.8</td>
<td>5/5</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>7.67</td>
<td>6/6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>7.6</td>
<td>5/5</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7.5</td>
<td>4/5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>7.2</td>
<td>5/5</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>7.2</td>
<td>5/6</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>7.2</td>
<td>5/5</td>
</tr>
</tbody>
</table>

Table 8: Group ranking based on group average result on Questionnaire 2 including the number of students per group who filled out the questionnaire.

From table 8 it becomes clear that the above average groups in this ranking are groups 3, 5, 8 and 17. For the below average groups this turned out to be groups 9, 13, 15 and 16. Because groups 16 and 10 have the same average of 6.67 and they are on the edge of being included with the below average groups it
was considered if it was possible to switch the two groups around. But because two members of group 10 did not fill out questionnaire 2, meaning the results might have been skewed, it was decided to keep the ordering from table 8.

### 3.3.4 Questionnaire 3

The fourth data source to base a ranking on were the GOAL knowledge questions in questionnaire 3. In the same way as with questionnaire 2 the results are the average of the students of each group. This time the overall average number of correct answers for all groups was 8.1. To find the below and above average groups a ranking of the groups based on their averages was made which resulted in table 9.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Avg.</th>
<th>Students</th>
<th>Rank</th>
<th>Group</th>
<th>Avg.</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>9.75</td>
<td>4/5</td>
<td>9</td>
<td>15</td>
<td>8.2</td>
<td>5/6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>9.25</td>
<td>4/5</td>
<td>11</td>
<td>2</td>
<td>8</td>
<td>5/5</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>8.83</td>
<td>6/6</td>
<td>11</td>
<td>7</td>
<td>8</td>
<td>5/5</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>8.8</td>
<td>5/5</td>
<td>13</td>
<td>11</td>
<td>7.8</td>
<td>5/5</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>8.6</td>
<td>5/5</td>
<td>14</td>
<td>4</td>
<td>7.6</td>
<td>5/5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>8.6</td>
<td>5/5</td>
<td>14</td>
<td>16</td>
<td>7.6</td>
<td>5/6</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>8.4</td>
<td>5/5</td>
<td>16</td>
<td>1</td>
<td>6.4</td>
<td>5/6</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>8.4</td>
<td>5/6</td>
<td>17</td>
<td>9</td>
<td>5.2</td>
<td>5/5</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>8.2</td>
<td>5/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Group ranking based on group average result on Questionnaire 3.

When using the ranking in table 9 as the ranking for the final selection the ranking has groups 6, 10, 12 and 17 as above average groups while groups 1, 4, 9 and 16 are the below average groups.

### 3.3.5 Pre-competition and Competition result

The fifth data source is the results of the competition at the end of the project and the pre-competition that preceded it. The pre-competition is held by the student assistants before the competition to determine whether the student groups had met the minimal requirements of the project and to determine the starting position of the various teams for the competition. During the pre-competition each team plays 3 matches against bots that where provided with the UT2004 platform. Using the simple points system of 3 points for a win, 1 point for a tie and 0 points for losing each team got a result from 0 to 9. The point totals per team can be seen in table 10 which also includes the number of times a team Captured the enemy flag(Caps) and their own flag was captured by the bots.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Avg.</th>
<th>Students</th>
<th>Rank</th>
<th>Group</th>
<th>Avg.</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>9.75</td>
<td>4/5</td>
<td>9</td>
<td>15</td>
<td>8.2</td>
<td>5/6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>9.25</td>
<td>4/5</td>
<td>11</td>
<td>2</td>
<td>8</td>
<td>5/5</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>8.83</td>
<td>6/6</td>
<td>11</td>
<td>7</td>
<td>8</td>
<td>5/5</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>8.8</td>
<td>5/5</td>
<td>13</td>
<td>11</td>
<td>7.8</td>
<td>5/5</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>8.6</td>
<td>5/5</td>
<td>14</td>
<td>4</td>
<td>7.6</td>
<td>5/5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>8.6</td>
<td>5/5</td>
<td>14</td>
<td>16</td>
<td>7.6</td>
<td>5/6</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>8.4</td>
<td>5/5</td>
<td>16</td>
<td>1</td>
<td>6.4</td>
<td>5/6</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>8.4</td>
<td>5/6</td>
<td>17</td>
<td>9</td>
<td>5.2</td>
<td>5/5</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>8.2</td>
<td>5/6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Pre-competition and Competition result.
<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Score</th>
<th>Caps</th>
<th>Rank</th>
<th>Group</th>
<th>Score</th>
<th>Caps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>9</td>
<td>10-0</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>6-1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
<td>7-0</td>
<td>11</td>
<td>13</td>
<td>6</td>
<td>4-3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>6-0</td>
<td>12</td>
<td>12</td>
<td>6</td>
<td>5-6</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>9</td>
<td>6-1</td>
<td>13</td>
<td>17</td>
<td>6</td>
<td>2-3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>9</td>
<td>5-2</td>
<td>14</td>
<td>6</td>
<td>5</td>
<td>4-2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4-0</td>
<td>15</td>
<td>5</td>
<td>4</td>
<td>3-2</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>7</td>
<td>4-0</td>
<td>16</td>
<td>11</td>
<td>4</td>
<td>4-5</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>6</td>
<td>7-2</td>
<td>17</td>
<td>15</td>
<td>3</td>
<td>3-3</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>6</td>
<td>6-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Ranking based on pre-competition results per group in both score and flag capture difference.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Rank</th>
<th>Group</th>
<th>Rank</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>5</td>
<td>10</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>9</td>
<td>1</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>9</td>
<td>2</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>9</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 11: Group ranking based on their success in the final competition.

Determining the above average groups is quite easy as all the teams that made it to the semi finals, groups 6, 11, 13 and 16, can be selected. A problem arises with the below average groups. The lowest can be found because an actual 17th place can be found. For the other three positions eight groups are eligible because they did not make it to the quarter finals.

### 3.3.6 Final grade

The final data source was the grade the students received at the end of the project. As the final grade was determined per group and encompasses both the resulting project and the development process of the group it gives a clear picture of how the groups worked throughout the project. Using these grades for a ranking resulted in the ranking noted in Table 12.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Grade</th>
<th>Rank</th>
<th>Group</th>
<th>Grade</th>
<th>Rank</th>
<th>Group</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>7.8</td>
<td>12</td>
<td>6</td>
<td>6.7</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8.8</td>
<td>8</td>
<td>13</td>
<td>7.7</td>
<td>12</td>
<td>15</td>
<td>6.7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8.3</td>
<td>9</td>
<td>14</td>
<td>7.2</td>
<td>15</td>
<td>9</td>
<td>6.5</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>8.2</td>
<td>10</td>
<td>4</td>
<td>6.9</td>
<td>16</td>
<td>11</td>
<td>6.2</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>5</td>
<td>6.9</td>
<td>17</td>
<td>17</td>
<td>5.2</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>7.9</td>
<td>12</td>
<td>3</td>
<td>6.7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 12: Ranking of the groups based on the final grade.

Table 12 shows clearly that the top 4 groups only considering the final grade are groups 1, 8, 10 and 16. While the top is very clear cut the lowest four groups are harder to determine because while the lowest three ranks are unique
the rank above them has three groups in it.

Besides this the final grade is not completely independent from the other data sources. This dependency comes from the way the final grade is calculated. It is a weighted average of the grade for their delivered product and the grade for the group process. Added to this average is a bonus based on how well the agent team of the group performed in the end competition. If the extra bonus is not counted the result and ranking change as can be seen in table 13.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>8.8</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>8.3</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>8.3</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>7.9</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>7.8</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7.5</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>7.7</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>7.2</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>6.9</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>6.9</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>6.7</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td>6.7</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>8.8</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>7.2</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>6.9</td>
</tr>
<tr>
<td>16</td>
<td>11</td>
<td>5.7</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>6.7</td>
</tr>
</tbody>
</table>

Table 13: Ranking of the groups based on the final grade without competition bonuses.

Using the ranking from table 13 the top 4 groups no longer includes group 10 but instead includes group 7. The lowest four groups also changes as they now include groups 6, 9, 11 and 17.

The ranking from table 13 is chosen to be used over the ranking from table 12 when combining and comparing rankings with other data sources as it gives the most independent results. When referring to the ranking in table 12 at any further point it will be clearly indicated that the competition bonuses are part of the final grade.

3.3.7 Ranking comparison and group selection

With the group ranking established for each data source on its own the different rankings have to be compared so the final ranking can be determined. This final ranking could then be used to select the groups that would be used for the remainder of this research. The first step towards this final determination was collecting all the separate rankings which resulted in table 14.

At first glance the rankings in table 14 do not show a consistent connection but upon further analysis some significant connections can be found. The exam results show a distinct correlation with the individual assignment (0.634) and the final grade (0.460). Questionnaire 2 also has a correlation with the individual assignment (0.478) but also with questionnaire 3 (0.438) while the (pre)competition ranking had a positive correlation with the final grade (0.513) and a negative correlation with questionnaire 3 (-0.480). This last correlation indicates that groups that performed well in the pre-competition tended to perform bad in questionnaire 3 and vice versa. All other possible correlations are between -0.37 and 0.33.

Because of the lack of a very strong correlation between the rankings there is no indication which particular ranking or ranking combination to select for the final ranking. This meant that we had to choose between either selecting one ranking to use as the final ranking or to use a function that combines multiple rankings.

The ranking used for the final group selection has to represent the quality of the code without actually being based on directly looking at the code quality.
Table 14: All rankings per group with the final grade rank are based on the grade without competition bonus.

<table>
<thead>
<tr>
<th>Group</th>
<th>Exam</th>
<th>Individual Assignment</th>
<th>Q2</th>
<th>Q3</th>
<th>Pre Competition</th>
<th>Competition</th>
<th>Final Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td>16</td>
<td>16</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>11</td>
<td>2</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>3</td>
<td>5</td>
<td>14</td>
<td>9</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>11</td>
<td>14</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>17</td>
<td>17</td>
<td>7</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>10</td>
<td>13</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>10</td>
<td>12</td>
<td>13</td>
<td>13</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>10</td>
<td>9</td>
<td>4</td>
<td>12</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>13</td>
<td>16</td>
<td>9</td>
<td>17</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>14</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>14</td>
<td>15</td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>14</td>
<td>13</td>
<td>11</td>
<td>15</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>14</td>
<td>4</td>
<td>3</td>
<td>15</td>
<td>9</td>
<td>17</td>
</tr>
</tbody>
</table>

Because that would mean that the results of the comparison in the next part of the research would be contaminated as it would be concluding that the top groups (partially) based on code quality have a higher code quality then the bottom groups. This is why the final grade is left out as this already included a partial grade for the quality of the code.

To get a ranking that is an indication of quality of code without being directly based on the quality it will have to be based on indicators of code quality. To do this we need to look at the group’s GOAL programming skills and how well the product functions. As three rankings, the exam, the pre-competition results and the individual assignment, represented both factors already it was decided to choose a combination of these three rankings and not to combine them with other rankings.

The exam was chosen because it is the best representation of the programming skill of the students as the results of questionnaires 2 and 3 do not correlate well with any of the other results. The pre-competition ranking is selected instead of the competition ranking to represent the end product because it is more diversified and it gives a more equal comparison between the groups. While the exam and pre-competition already represent the two factors the final ranking needs the individual assignment can be added to help diversify the final ranking if needed.

The final ranking is based on the weighted average of the exam, the pre-competition and the individual assignment with weights being two, two and one respectively. The individual assignment is added because the average of the pre-competition and the exam ranking was not diversified enough resulting in three pairs of groups that have the same average. One of these spans the fourth and fifth place making determining the top four hard. Because the individual
assignment was only included just to diversify the results it was decided to make the average weighted in favor of the pre-competition and the exam so that the individual assignment did not have to much an effect on the order of the final ranking.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Group</th>
<th>Exam Results</th>
<th>Pre-Competition</th>
<th>Individual Assignment</th>
<th>Weighted Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1.8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>4.0</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>3</td>
<td>8</td>
<td>10</td>
<td>6.4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>6.6</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>6.8</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>7.0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>11</td>
<td>6</td>
<td>3</td>
<td>7.4</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>2</td>
<td>14</td>
<td>6</td>
<td>7.6</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>8.0</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>12</td>
<td>1</td>
<td>14</td>
<td>8.0</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>4</td>
<td>15</td>
<td>5</td>
<td>8.6</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>9.2</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>13</td>
<td>6</td>
<td>14</td>
<td>10.4</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>14</td>
<td>11</td>
<td>13</td>
<td>12.6</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>16</td>
<td>13</td>
<td>14</td>
<td>14.4</td>
</tr>
<tr>
<td>16</td>
<td>11</td>
<td>17</td>
<td>16</td>
<td>10</td>
<td>15.2</td>
</tr>
<tr>
<td>17</td>
<td>15</td>
<td>15</td>
<td>17</td>
<td>14</td>
<td>15.6</td>
</tr>
</tbody>
</table>

Table 15: The final ranking for each group with the exam, pre-competition and individual assignment results and the weighted average between those three rankings.

The resulting final ranking can be found in table 15. Using this ranking to determine the bottom four groups results in the selection of groups 11, 13, 15 and 17. These four have a big gap from the other groups with a difference of 2.2 on the average. Selecting the top 4 groups is equally easy because of the differentiating effect of the individual assignment. The top four groups are groups 1, 2, 4 and 10 though the difference between the fourth and fifth group is still small at only 0.2.

3.4 Group Comparison

With the eight groups selected for comparison the comparison method has to be determined. To do this a checklist is made with items for each subject that is to be studied. Section 3.4.1 will explain which items are in the checklist, what they mean and how subjective scoring is prevented. After explaining the checklist we will go through each subject and compare the stronger and weaker groups and what conclusions can be drawn from this comparison.
3.4.1 Checklist composition

As explained in section 2.2 the selected eight groups are compared based on software quality factors\(^1\) and code constructs. In particular the focus of the software quality part of the comparison will be on Understandability and Testability. To be able to easily compare the groups a checklist was created. The checklist contains elements that contributed to the two software quality factors and the code constructs. The complete checklist is divided into four subjects: Code Quality, Code Structures and Patterns, Documentation and Test Protocols. In the rest of this subsection we will go through the checklist per subject and highlight and clarify some of the items on it. All items on the checklist can be marked either true or false for a group except where noted otherwise.

To determine what items are included in the checklist for the subject of software quality chapter 7: Design of Agent Programs of the GOAL programming guide [37] is used. These items are split into two parts: Code Style and Comments. Code Style includes nineteen items which are listed in list 4 and the Comments include eight items which are listed in list 5.

The first item in list 4 states that the ‘-’ or “don’t care” symbol should not be used in mental state conditions. If programmers do use the ‘-’ symbol in a mental state the behavior of the agent can become unpredictable because non matching values are bound to the ‘-’ during run-time. The second item states that if two modules are not strongly connected to another module they should be placed in separate files. For example an attacker module and a defender module should be in separate files but a module that helps the attacker with target selection can be included in the same file.

Items seven to ten state the rules about the use of goals. The seventh item asks that goals are used. The eighth states that goals are used for things the agent wants to achieve. The ninth means that goals are declarative meaning that the goal to get the flag is not \textit{needFlag} but is \textit{haveFlag} instead. The tenth item states that goals are concrete meaning that a goal is not abstract like for instance \textit{win} but deterministic like \textit{haveLead}.

The eleventh and twelfth items state that all actions specified in an agent should have matching actions in the environment and that they should always be specified in the init module because they are then available to every module while any other module makes them only available in that module. The thirteenth item states that the pre-condition specified for an action should match the conditions that the environment puts on the action and variable bindings for the post-condition. It should not include any conditions that are convenient for the developer to influence the behavior of the agent. These conditions should be included in the action rules that are used to call the action. If the condition was included in the action specification and a case comes up where the extra conditions do not apply all the code has to be rewritten to prevent unwanted behavior. For example the halt action in UT2004 can be executed at any time so the pre-condition should always be true, though it might include a variable that needs to be bound for the post-condition. The pre-condition should not include a condition that prevents it from executing when it is inconvenient. An example of this is when the agent is the flag carrier and should stop for nothing. This kind of extra restrictions should be included in the action rules to decide on a different action while being the flag carrier. Further in the development

\(^1\)See section 2.3.3.
List 4 Checklist items for comparing the Code Style which is part of Code Quality.

- ‘-‘ is not used in mental state conditions (goals).
- Independent modules are placed in separate files.
- Labels are declarative.
- Beliefs only show current state of the agent.
- There are no redundant predicates.
- Knowledge is only used for environment or basic logic.
- Goals are used.
- Goals only express desires.
- Goals are declarative.
- Goals are concrete.
- There are no non environment actions.
- All environment actions are declared in the init module.
- Action pre-condition matches the environment.
- Action rules are not used to insert knowledge in belief base.
- Action rules are only in the main module or linked modules.
- Percepts are only used in the event module.
- Percepts are handled in forall rules.
- Communication rules are written in the event module preferably after percepts.
- Any goal management rules are at the end of the event module.
of the MAS it could turn out that there is a case in which halting as the flag carrier is the correct action.

List 5 Checklist items for comparing the Comments which is part of Code Quality.

- % comments before knowledge predicates
- Comments before knowledge rules explain the meaning of those rules.
- % comments before action specifications
- Comments before actions explain the pre and post conditions.
- % comments before modules
- Comments before modules explain the meaning/use.
- % comments before Rule groups
- Comments before action rule groups explain the purpose of those rules.

The items in the Comments checklist, list 5, determine how well comments are used in a MAS. The list checks the use of comments for four language elements. These elements are knowledge rules, action specifications, modules and action rules. For each language element it is checked what percentage is preceded with comments (values from 0 to 100 inclusive) and whether the comments are used in the appropriate way to explain what the code it describes is used for. The later items, items 2, 4, 6 and 8, are marked true if most of the appropriate comments are valid and useful, though one meaningless comment does not automatically marks an item as false.

The checklist for Code structures and patterns, list 6, contains only two items which are both open questions. The first item asks if there are any code patterns used in the code of the team and if so what kind. The second item checks how the groups handle rules that should both add and remove beliefs. Section 3.4.3 explains what different methods there are and why the choice is important.

List 6 Checklist items for comparing the Code structures and patterns used.

- Are there any code patterns used in the working of the team?
- Is insert+delete or insert with negation used when beliefs are both added and removed?

List 7 shows the items that are part of the checklist for Documentation. The actual checklist has separated lists for the ontology and for DrProject. While studying the code they are checked of separately. As the items for each documentation method are the same they will be discussed here at the same time. The first item is marked true if at the end of the project that documentation type is up to date to the last version of the code.

The second item is true if the group attempted to keep the documentation system up to date throughout the project. As we did not keep track of all the
List 7 Checklist items for comparing the use of the Ontology or DrProject for the Documentation subject.

- The ontology/DrProject is complete/up to date at the end of the project.
- The ontology/DrProject was kept up to date throughout the project.
- The ontology/DrProject is understandable.
- The ontology/DrProject was used during the project.

groups during the whole project the marking of this item is based on the notes of the student assistant that was responsible for a certain group. The third item is marked true if someone who has not seen the documentation before can navigate the system without problems and find basic information like a specific predicates in the ontology. The last item is marked of if the groups actually use the system for its purpose and did not just keep it filled because they were asked to do so. It is again hard to determine whether this was true and we had to rely on the notes of the student assistants.

The last subject of the checklist is test protocols which has five items as listed in list 8. As with some of the documentation items it is impossible to keep track of all the tests done by all the groups during the project so the notes of the student assistants and the test reports made by the groups are used as extra information source when judging these items.

List 8 Checklist items for comparing Test protocols.

- What is the noted test frequency?
- Were module test executed?
- Were full system tests executed?
- Was the system fully tested on all maps?
- Was the system tested against (non native) bot teams?

The first item on list 8 is the frequency of testing done by the students. This can be rated high, medium or low where a medium rating was roughly equivalent to doing an average of a full system test each project day and an occasional module test. Doing less than that was rated as low while doing more was rated as high.

If groups did module/unit tests or full system tests regularly respectively the second and the third item would be marked as true. As the UT2004 environment includes three maps the fourth item on the checklist is marked true if the group tested each map to make sure that their MAS worked as expected on each map. It is not necessary to spend equal time on all maps as long as the MAS functionality is fully tested on each map. The last item is marked true if the group tested their team against a GOAL agent team. This could be a team of another group or an old version of their own team.

Then the code and reports of each group were studied and each item that
was found in the work of the groups was checked of. After we went through the checklist for each group the results of the bottom four groups was compared with the work of the top four groups to identify differences between both groups.

Tables 16, 17, 18, 19 and 20 contain the results of the groups for the different items on the checklists. In the coming subsections the results will be discussed for each section of the checklist.

<table>
<thead>
<tr>
<th>Item</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>10</th>
<th>11</th>
<th>13</th>
<th>15</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>'-.' is not used in mental state conditions (goals).</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Independent modules are placed in separate files.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Labels are declarative.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Beliefs only show current state of the agent.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>There are no redundant predicates.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Knowledge is only used for environment or basic logic.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Goals are used.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Goals only expresses desires.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Goals are declarative.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Goals are concrete.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>There are no non environment actions.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>All environment actions are declared in the init module.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Action pre condition match the environment.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Action rules are not used to insert knowledge in belief base.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Action rules are only in the main module or linked modules.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Percepts are only used in the event module.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Percepts are handled in forall rules.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Communication rules are written in the event module preferably after percepts.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Any goal management rules are at the end of the event module.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 16: The results of the checklist on Code style for each group.
Comments before knowledge predicates explain the meaning of those rules.

Comments before actions explain the pre and post condition.

Comments before modules explain the meaning/use.

Comments before action rule groups explain the purpose of those rules.

<table>
<thead>
<tr>
<th>Group</th>
<th>Patterns</th>
<th>Insert and Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strategy pattern based on roles.</td>
<td>insert + delete</td>
</tr>
<tr>
<td>2</td>
<td>Strategy pattern based on roles.</td>
<td>insert with negation</td>
</tr>
<tr>
<td>4</td>
<td>Strategy pattern based on goals.</td>
<td>both</td>
</tr>
<tr>
<td>10</td>
<td>Strategy pattern based on tasks.</td>
<td>insert + delete</td>
</tr>
<tr>
<td>11</td>
<td>None</td>
<td>insert + delete</td>
</tr>
<tr>
<td>13</td>
<td>None</td>
<td>insert with negation</td>
</tr>
<tr>
<td>15</td>
<td>Strategy pattern based on roles.</td>
<td>insert with negation</td>
</tr>
<tr>
<td>17</td>
<td>None</td>
<td>insert with negation</td>
</tr>
</tbody>
</table>

Table 17: The results of the checklist on Comments for each group.

Table 18: The results of the checklist on Code patterns for each group.
The ontology is complete/up to date at the end of the project.
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

The ontology was kept up to date throughout the project.
✓ ✓ ✓ ✓ ✓ ✓ ✓

The ontology is understandable.
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

The ontology was used during the project.
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

The DrProject is complete/up to date at the end of the project.
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

The DrProject was kept up to date throughout the project.
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

The DrProject is understandable.
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

The DrProject was used during the project.
✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

Table 19: The results of the checklist on Documentation for each group.

<table>
<thead>
<tr>
<th>Item</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>10</th>
<th>11</th>
<th>13</th>
<th>15</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the noted test frequency?</td>
<td>H</td>
<td>H</td>
<td>M</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>Were module tests executed?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Were full system tests executed?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Was the system fully tested on all maps?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Was the system tested against (non native) bot teams?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 20: The results of the checklist on Testing protocol for each group. (H = High, M = Medium, L = Low)
3.4.2 Code Quality

The first checklist subject to be discussed is Code Quality. This subject had a total of twenty-seven items. Twenty-three of the items were code style principals. The other four items had to be filled in with a percentile as they represented how much of a certain type of language construct (action rules, knowledge rules, action specifications and modules) were accompanied with comments. The Code Quality checklist is separated into 2 parts. The items can be found in lists 4 and 5 in the previous subsection.

<table>
<thead>
<tr>
<th>Group</th>
<th>Style</th>
<th>Comments</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>6</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>10</td>
<td>7</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 21: Number of items marked of per group on the Code quality checklist.

In table 21 can be seen how many items of code style and comments were used by each group. This number was established by counting per sub-subject how many items were deemed true per group. When comparing the results of the groups it becomes clear that the top four groups perform better on code style while comments are reasonable close with averages of 14.75 and 5.75 versus averages of 11 and 4.5. These differences explain for a large part the bad performance of group 11 which only had a total of 9. But even when not counting their performance the bottom four groups average on style is 12. This is almost three lower than the average of the top groups though the difference in the comments part becomes negligible. The results of the checklist suggest that paying attention to code quality in both the code style and the use of comments leads to better performance.

3.4.3 Code Structures and Patterns

The second subject is Code Structures and Patterns which contain two open questions. The first question asks after any recurring code pattern that can be found in the code of the agent teams. The second question asks what syntax the students use when they have to remove beliefs from the beliefbase and add beliefs. There are two ways to accomplish this. The first and standard way is by deleting the unwanted belief and inserting the new belief. The second is inserting the new belief and inserting the negation of the unwanted belief. For example if the agent dies and has to reset his state from what ever it is now to the standard blank state it could be done as in code fragment 7 with delete and insert or as in code fragment 8 with insert and a negation. Because of how GOAL works the second option is a bit faster but it is a bit less understandable when adding and removing multiple beliefs.

Table 22 contains the collection of the results of the code patterns checklist which shows that multiple groups used a construct that closely resembled the
if bel(died, state(X))
   then delete(state(X)) + insert(state(reached([]))).

Code Fragment 7: Adding and removing code by using insert + delete.

if bel(died, state(X))
   then insert(not(state(X)), state(reached([]))).

Code Fragment 8: Adding and removing code by using insert with negation.

Strategy design pattern [20] to determine what code module they would be executing. The strategy pattern supports the selection of algorithms at runtime. To be exact the strategy pattern defines a family of algorithms and makes them interchangeable by encapsulating them individually. This means that the Strategy lets the algorithm vary independently from the code that uses it. In the case of AOP multiple modules can be made that cover the same basic behavior in different way which then can be selected based on the current state of the agent, for instance on the current roll or task of the agent.

Code fragment 9 shows the use of the Strategy pattern by group 15. This pattern allows the developers to easily control the behavior of the agent by selecting what module the agent will execute and then letting the different modules handle any other situation. In most cases where these patterns occur any changes to the selecting criteria, for instance role changes, are handled in the event module. When comparing which groups used the Strategy pattern it became clear that it was used by all top four groups and that only one group from the bottom four used it. This is a clear indication that the use of the Strategy pattern improves the quality of the agent.

The other entry column in table 22 shows that the use of the insert and delete syntax was more divided as three groups used insert + delete, four groups used insert with negations and group 4 actually used both syntax constructs in different parts of their code. The use of insert + delete is slightly higher in the top four groups than in the bottom four groups but as the UT2004 environment is not a high performance environment it cannot be concluded whether this is significant or just a case of preference. The use of the insert syntax is probably something that should be decided on on a project by project basis though we advice against mixing insert + delete with insert with negations as this can make the code confusing to read which happened somewhat with the code of group 4.
main module{
  program{
    % The agent chooses the right module depending on his role
    if bel(role(defender)) then defender.
    if bel(role(attacker)) then attacker.
    if bel(role(helpAttacker)) then helpAttacker.
    if bel(role(tempAttacker)) then tempAttacker.
  }
}

Code Fragment 9: Strategy like code patterns in the main module of group 15.

<table>
<thead>
<tr>
<th>Group</th>
<th>Patterns</th>
<th>Insert and Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strategy pattern based on roles.</td>
<td>insert + delete</td>
</tr>
<tr>
<td>2</td>
<td>Strategy pattern based on roles.</td>
<td>insert with negation</td>
</tr>
<tr>
<td>4</td>
<td>Strategy pattern based on goals.</td>
<td>both</td>
</tr>
<tr>
<td>10</td>
<td>Strategy pattern based on tasks.</td>
<td>insert + delete</td>
</tr>
<tr>
<td>11</td>
<td>None</td>
<td>insert + delete</td>
</tr>
<tr>
<td>13</td>
<td>None</td>
<td>insert with negation</td>
</tr>
<tr>
<td>15</td>
<td>Strategy pattern based on roles.</td>
<td>insert with negation</td>
</tr>
<tr>
<td>17</td>
<td>None</td>
<td>insert with negation</td>
</tr>
</tbody>
</table>

Table 22: The use of code patterns and the insert syntax per group.

3.4.4 Documentation

The third section on the checklist is the use of documentation by the various groups. This section dealt with the two main documentation methods the groups used which cover the ontology and the use of DrProject¹ and specifically its ticket tracking system². Though comments are normally part of the documentation they are not included with this subject as they are already covered in the checklist in the subject of Code Quality.

The ontology was used to keep track of beliefs, goals, percepts, actions and knowledge predicates with the goal of having a convenient place for all group members to look up any term used in the agent team code without having to browse through all the code used. Besides listing the names of the predicates a good ontology also includes at least a description of what each item in the ontology is used for and is ordered in such a way that it is clear whether an item is a belief, goal, percepts, action or knowledge predicate. An extract from the ontology of group 1 can be seen in figure 13 showing a few of the beliefs they used in their agent team.

The second documentation type looked into was the use of the ticket tracking system found in DrProject. This system was used by the student groups to keep track of the development process. This is done by making tickets both for new functionalities that had to be added to the agent teams and for bugs that where found and had to be fixed by the students. Using this system the students could keep track of the progress they made and had an overview of what still had to be done.

¹https://drproject.twi.tudelft.nl/drproject2011/All/wiki/DrProject
²Derived from the Trac system. http://trac.edgewall.org/
Each of the documentation methods has the same items to check. The items
do not only try to assert whether the students managed to deliver sufficient and
clear documentation at the end of the project but also whether they kept the
documentation up to date throughout the project. Lastly it is checked whether
the groups actually used the documentation method or only paid lip service to
it. Table 23 notes for each group the number of items that could be marked of
based on their documentation. From the data it becomes clear that the top four
groups paid more attention to their documentation than the bottom four groups
with an average of 5.75 items marked of against only 3.5 items. This suggests
that paying attention to documentation improves the performance of a group.
It is interesting to note that the bottom four groups have on average spend
slightly more attention to the ontology while the top four groups on average
have spend slightly more attention on DrProject.

<table>
<thead>
<tr>
<th>Group</th>
<th>Ontology</th>
<th>DrProject</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Top 4</td>
<td>2.5</td>
<td>3.25</td>
<td>5.75</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Bottom 4</td>
<td>2</td>
<td>1.5</td>
<td>3.5</td>
</tr>
<tr>
<td>Total</td>
<td>18</td>
<td>19.5</td>
<td>37</td>
</tr>
</tbody>
</table>

Table 23: The number of marked items in the documentation section of the
checklist for both the ontology and DrProject.
### 3.4.5 Testing Protocol

The last part of the checklist is concerned with the method of testing used by the students. The items in this part of the checklist are aimed towards finding how the different groups organized and executed the testing of their agent team. The items cover the frequency of testing done by the students, whether they did module/unit tests, whether they did full system tests, whether they tested fully on all three available maps and whether they tested their bot team against other bot teams, either their own or the team of another group, as opposed to the bots that are part of UT2004. Table 24 lists the resulting data.

<table>
<thead>
<tr>
<th>Group</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Frequency</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Unit tests</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>System tests</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>All Maps tested</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Other Bot team</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Group</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>Test Frequency</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Unit tests</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>System tests</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>All Maps tested</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Other Bot team</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 24: The results of the checklist on testing protocol for each group. Questions are short versions for those in list 8.

Looking at the test frequency it becomes clear that the top four groups in general have a higher test frequency than the bottom four groups and they also on average test in more varied ways. Strangely enough only group 11 tested their agent team against older versions of their own team as part of their testing protocol. This suggests that high frequency testing and testing in various different ways help to improve the quality of the agent team. In environments where you are in competition against other teams it is a good thing to test against several bot teams to see how a team performs in comparison because this can lead to situations the groups had not thought about yet and prevents over training the team against one particular opponent. Alternatively the current version of an agent team can be tested by running it against an older version of the same team to see whether changes to the team improved the performance of the team. Another important point is that the behavior of the native UT2004 bots is distinctly different from the behavior of agent teams made in GOAL. This means that that teams that only test against native bot teams can be unprepared for the actual competition against the agent teams of other students groups as student teams can have “irrational” strategies that the native bots will never use like all bots defending the flag.

While group 13 did do module tests during their production process the initial set up of their MAS would not allow them to do so easily. During development they had a controlling agent not connected to a bot that made all the decisions for all agents. This means that the control flow of the agents came from outside the agents. Furthermore the controlling agent was hard coded.
to expect three agent to be present while running and could not function with fewer agents being present. Because of this set up all the module tests had to be set up as full system test because all decision making required the whole system to operate. The students themselves recognized this as a problem during the development process and they changed the set up of their MAS so that most decisions were made by the agents themselves. This made module testing a lot simpler and increased the testability of their MAS.

3.5 Lessons Learned

Analyzing the work of the various students groups the last question that remains is what can be learned during the UT2004 project that can be used in the HactarV2 project.

First the analysis of code quality in section 3.4.2 suggests it is important that the HactarV2 team keep to the code style set forth in the GOAL programming guide [37] and that they all use the same code style so that they can all easily understand the code as it is written. A part of this is that they had to agree on what syntax is going to be used: insert + delete or insert with negation.

Secondly as noted in section 3.4.3 the HactarV2 will want to use a Strategy design pattern to control the behavior and role selection of the agents. Part of this is that the agent has modules for each separate role/strategy option so that the function of the module is absolutely clear.

Thirdly in section 3.4.4 we find that keeping accurate and complete documentation improves the quality of an agent system when working in groups. This applies to documentation in the form of an ontology to keep track of the predicates used and in the form of a project/issue management system to facilitate the cooperation between group members.

The second to last lesson to take away from this project is from section 3.4.5 which is that repeatedly testing an agent team in both module/unit and full system test improves the quality of the agent. With these tests it is important that test cases account for as many as possible scenario’s that can occur during the actual running of the agent team.

The last lesson is that testing agents is hard if the agent is dependent on other agents to make their decisions for them. This means that to test a single module the whole system has to be run. The example of group 13 in subsection 3.4.5 is a clear illustration of this.

Table 25 shows a clear overview of all the lessons per subject. The lessons for Code Quality come from section 3.4.2 and start with the general lesson that following the code style described in the GOAL programming guide will improve the understandability of GOAL code. The next lessons on Code Quality point to the fact that clear comments and declarative predicate naming improve the understandability of a GOAL MAS. Placing each module in its own .mod2g file will also help with the understandability. By having goals that state concrete desires the use of the goals becomes clearer which does also enhance the understandability. By matching the pre and post conditions to the environment and placing all goal management in the event module the testability and understandability of a MAS are improved as it guarantees that without looking up the code of the action the developer knows what it does from the environment description and that the agent will always do its goal management respectively.
The subject of Design Patterns in table 25 has a lesson that advices a developer to choose between using either Insert + Delete or Insert with negation but not both as this makes code more confusing. The other lesson advises to use the strategy pattern that was found during the analysis of the agent teams. This pattern is described more fully in section 3.4.3.

On the subject of Documentation the table contains that documentation should be kept up to date and that the use of an ontology reduces look up time during development though it should not be seen as a replacement of comments.

The lessons on Testing in table 25 note that it is a good idea to have a high testing frequency which includes unit test, full system test and test against multiple bot teams. It also contains the lesson that if other agents make too many decisions for an agent it becomes very hard to do unit tests. Lastly it suggests that as many scenarios as possible should be tested.

With table 25 in hand we can start on the MAPC project to see whether the HactarV2 team can apply these lessons and to test whether the lessons are transferable to other environments and development teams.
4 Multi-agent programming contest

After the completion of the UT project and the analysis of the data it provided the methods found during the analysis could be tested with a second project. This second project involved working with five other students to create a MAS that we could enter into the Multi-agent programming contest. The team was called HactarV2 and the development process was done in August 2011.

4.1 Project Description

The Multi-agent programming contest is a yearly international competition organized by the German Clausthal University of Technology. The competition was first organized in 2005 [38] and has since then stimulated research in the field of multi-agent systems and languages. Originally there were ten competing teams but one had to forfeit before the actual competition started. Most of these teams came from Europe (Denmark, Ireland, Germany and Holland), but there were also several teams from Iran and a team from Argentina.

4.1.1 Mars Exploration

The basis of the 2011 MAPC is that the agent teams will explore Mars [3]. The exploration in this scenario is based on the fact that water wells have been discovered on Mars. The goal the agent teams have is to locate the locations with water wells and to occupy those places. Of course multiple companies and their agent teams want to profit the most from this discovery and the teams will have to compete for the possession of the water wells.

A graph represents the surface of Mars where nodes denote locations and they have a value which indicates the amount of water that is present in the water well. The graph is mirrored to provide a fair symmetric map on which ten agents from each team can move around. These agents are evenly distributed over five different roles. The Explorers can determine the amount of water at nodes, Sentinels have a better vision to provide more information about what happens on the planet, Inspectors can determine the roles and status of enemy agents, Saboteurs have the ability to attack and disable opponent agents and Repairers are able to restore disabled agents back to a working state.

Two teams play a match over three games each with a duration of 750 steps. The final score of a game is the total of all the step scores in that game. The step score is determined by adding the number of unspent achievement points to the zone score of that step. Each team starts the game with ten achievement points which can be spent on upgrades. A team can collect more points by gaining achievements for actions like attacking enemies and exploring the map. The zone score is determined each step by the nodes that are controlled/guarded by the agents of a team and is simply the sum of the values of all the nodes in the controlled area. This means that a zone with higher value nodes will provide a better score.

4.1.2 HactarV2

The TUD team is named HactarV2 as a reference to the super computer Hactar. Hactar is a fictional computer from the book “Life, the Universe and Everything” by author Douglas Adams [39]. Hactar is a computer whose components
reflect the pattern of the whole computer. Its original purpose was to design the ultimate weapon. It completed this task by creating a device that would make all the stars in the universe go supernova at once. But it also made the device inoperable because it had decided that such a weapon should not exist. Hactar’s creators saw this as a failure of the super computer and pulverized Hactar into small dust and scattered its remains throughout the universe. Because all the components reflect the whole Hactar was still operational. And the pieces proceeded to slowly recombine and became a cloud of particles. Reasoning that it had erred by disabling the device Hactar then tried to destroy the universe only to be thwarted by the main protagonist’s terrible cricket skills.

The team behind HactarV2 found it a fitting name for their MAS because of two reasons. The first is that at the start of a match all agents that are part of HactarV2 are scattered all over the map as their positions are randomly allocated. During the match they will move towards each other and form a cloud of agents. The second reason is that all the agents in the system contained the complete code of the system and in this way an agent as a component reflects the whole.

4.1.3 Team Strategy

In this section a short explanation will be given of the strategy of the HactarV2 team. For a more complete explanation of the team strategy we refer you to the work of Dekker et al. [4].

The strategy of HactarV2’s MAS distinguishes two phases. In the first phase agents do not yet act as a team. In the second phase agent’s act as a team in order to occupy high valued zones on the map.

At the beginning of a game all agents move and act on their own. The goal of an explorer agent at the start of the game is to find the highest valued node on the map. We call this node the optimum. This strategy works because the map generator produces maps that have one cluster of higher valued nodes at the center of the map. Once the optimum is found, the name of this node is sent to the other agents and a swarm can be formed to occupy the zone around this node. The first phase ends when the optimum has been found.

When explorer agents have found the optimum and all other agents have been informed about this, these other agents will start moving toward the optimum. This moves the game to the second phase in our strategy, the swarming phase. The zone of nodes which is owned by our team and contains the optimum is called the optimum zone. Once an agent has arrived in the optimum zone, the agent determines the highest valued node directly outside the optimum zone and moves towards that node. By using this tactic the swarm will always expand in the direction of the highest valued nodes not yet owned by our MAS. Figure 14 shows a situation in which the HactarV2 agents have created a swarm around the nodes with the highest value.

When the agent reaches a position on the edge of the swarm it will consider all nodes it can move to. In addition it will predict to which nodes its connected agents may want to move. Based on this information, an agent can determine without communication whether it can make the best move compared to any of the agents it is connected to. If that is the case, an agent will move to expand the zone, and otherwise it will stay on its current node in order to ensure that the agents will still be connected.
Figure 14: HactarV2 (green) agents forming a swarm around the nodes with the highest value.
Agents that are located on the same node are ranked and assigned a unique number called the *agent’s rank*. This rank is used when multiple agents are predicted to perform the same action. In that case, based on their rank it is decided which agent will perform that action, the other agents will then perform another action. This rank-based mechanism allows agents to divide tasks among themselves without the need for communication while at the same time ensuring that each agent performs an unique action whenever possible.

Agents that are disabled will ask for help from a repairer agent. Disabled agents will move towards the repairer that is closest after informing it that the agent wants to be repaired by the repairer. If a repairer is not already committed to another agent, it will also start moving towards the disabled agent that requested help. Disabled agents send a path to the repairer they are moving to, to prevent the repairer from having to calculate a path towards the disabled agent to save time.

The saboteurs start the match in search and destroy mode. They receive information from all agents on the location of opposing agents. They will move towards a last known location of an opponent agent that is closest to their own position and attack that agent until it is disabled. Then they will look for the closest opponent agent that is still active. They will repeat this until they have disabled every enemy agent or until the explorer has found the optimum location. At that point the saboteurs will join the swarm and will start protecting it from opposing agents.

Our buying strategy is designed to keep our achievement points as high as possible and to spend less money than the opponent does. As the amount of money available has quite an impact on the points scored each round we decided to only upgrade our saboteurs. The upgrading of saboteurs is aimed at ensuring that (I) our saboteurs have one health point more then the highest strength of any of the opponent saboteurs and (II) the strength of our saboteurs is equal to the highest number of health points of any of the opponent saboteurs. If both of these goals are realized, then our saboteurs will survive a blow of an opponent saboteur while destroying them in a single hit.

The actual competition of the 2011 MAPC was held in the beginning of September and encompassed four days of simulations. And although it was allowed to change the code of the MAS between matches the only thing the HactarV2 team changed was adding special clauses to their agents for when they had disabled all enemy agents.

The HactarV2 team performed exceptionally well during the competition and did not only manage to win every match but also managed to win every single game. With this undefeated record they firmly placed them self as winners of the Multi-agent programming contest. Although the victory of the HactarV2 team is in a big part the result of hard work from a dedicated team of great developers, the application of the methods gathered from the UT project have helped the team to succeed.

### 4.2 Application of lessons

The idea behind participation on the MAPC project is to use it as a means to check the conclusions of the UT2004 project. At the end of UT2004 project there is a set of tools and methodologies, as described in section 3.5, that the above average groups used, or used better, than the below average groups. In
this section it is described how these methodologies and tools are used during the MAPC project. Section 4.3 is used to discuss whether we benefited from the application of these methodologies and tools.

### 4.2.1 Documentation

During the UT2004 project setting up and maintaining an ontology and a project management system is mandatory for the student groups. But as noted before in section 3.4.4 the groups that have clearer documentation and were better at maintaining them have on average a better result than groups that did not.

Even before this point about project management systems was brought up the development team of HactarV2 decided to use the agile software development approach Scrum [40]. Scrum is an iterative, incremental framework for project management. It emphasizes short development cycles in which clear targets are set to build and extend the functionality of a system. The open-source platform iceScrum [41] is used for managing and maintaining the development process and for collecting all ideas concerning strategy choices, implementation issues and optimization problems. iceScrum supports working with virtual sticky notes for representing tasks that can be assigned to people and for keeping track of progress.

To confirm whether there was indeed a connection between good ontology’s and good agent systems we made sure that an ontology was used during the MAPC project. But just using an ontology would not be enough to prove or disprove its usefulness. To prove that an ontology helps to improve the result of a project the ontology had to be clear, up to date and used by the developers. While team members could only be encouraged to use the ontology we assigned somebody to keep it up to date and that the information it contained was useful and readable.

The ontology was managed in the form of a table with each row containing a concept. To make sure that the ontology was readable and useful it was kept in alphabetical order and had the following columns: Name, Type, Description, Parameters and File. Figure 15 shows an extract from the ontology used during the MAP contest.

![Figure 15: Extract from the ontology used by the HactarV2 team during the MAPC project.](image)
ically sorted concepts with the same name would be sorted on the number of parameters with the one with the fewest parameters first.

The Type column is used to indicate what kind of concept a row contains. The option are: Action, Knowledge, Belief, Percept and Knowledge/Percept. This last one is used to indicate Percepts that are directly used as knowledge rules.

The third column contains a brief description of the function of the concept. This choice to not expand the description was made because long descriptions would interfere with the understandability of the ontology. Instead if more explanation was needed it would be included as comments at the same place as the concept was defined.

In the Parameters column was noted what the exact function of each parameter was. This was often no more than one word and parameters were separated by commas. The reason for this very short notation was the same as for the description as it kept the ontology understandable.

The last column contains the name of the file in which the concept is defined. This was added to help developers to easily find a concept when they wanted to edit it or find more information. The decision was made to not add line numbers as these would change often and would make keeping the ontology up to date a time intensive chore with very little gain.

4.2.2 Testing

In section 3.4.5 it is suggested that frequent and varied testing improves the quality of a MAS. This is why the HactarV2 team chose to spend a lot of effort during the developing of their MAS with testing their agent system and analyzing the test results. A great benefit of extensive testing has been that it quickly made the team familiar with all the details of the simulation scenario.

Because it is not practical to test every aspect of the MAS with a full system test a variety of test strategies were used. Initial testing focused on whether the MAS behavior was coherent. At this stage dummy agents were used as opponents that did nothing during a simulation (performed skips each round). In subsequent stages, when the performance of the MAS was at a reasonable level, the Java teams that were supplied by the organizers were introduced as opponents. These tests were particularly interesting as they yielded information about the way the agents reacted to very different agents. Another test strategy used was tests in which the most recent MAS played against older versions in order to verify whether the agent team actually improved over time. During these tests suboptimal behavior was observed that could probably only have been found because the strategy of the opposing MAS of the earlier versions was still quite similar. The development team also had the agent team, though not always the most recent one, participating in all test matches that were organized by the contest organizers which helped to sort out connection problems.

One testing tactic that was used while debugging the MAS involved the use of an edited XML configuration file for a simulation which granted the teams 2 million seconds to send their actions. This made it easier to pause the agent system and study the state of the MAS at a time when something went wrong. As a result, bugs were found more easily.
4.2.3 Code Quality

In section 3.4.2 the student groups of the UT2004 project are compared on their adherence to the Goal code style as noted in the Goal programming guide. The results of the comparison suggested that groups that adhered closely to the code style produced higher quality code and performed better than groups that only loosely adhered to the code style. The development team for HactarV2 has tried to follow these guidelines as closely as possible. Unfortunately because of the high performance requirements of the environment some corners had to be cut and section 4.3 shows exactly which guidelines are not followed.

4.2.4 Code Structures and Patterns

When deciding between the options for inserting beliefs and choosing between the insert + delete and the insert with negation syntax as discussed in section 3.4.3 the development team chose to use insert with negation. This decision was made on the grounds that the increase in time efficiency outweighed the loss in understandability.

The HactarV2 team implemented the Strategy patterns as every agent in the MAS has a similar structure and in order to prevent version problems and code copying, it was decided to make one master agent which can handle all different roles. By using the Strategy pattern and a modular organization of code, each agent uses role specific modules while sharing standard modules related to e.g. navigation. One shared main module dealt with all the role-independent information and strategy choices that are common to all agents. This included, for example, the processing of percepts that are received from the environment, basic communication and navigation tasks. The function of the main module is to connect all the smaller sub-modules and jump back and forth between them. Once all the common tasks have been executed the main module checks the role of the agent in question and selects role-specific modules that handle all tasks and decisions that concern that specific role. The module selection can be seen happening in code fragment 10 which displays the main program section for the agents. The first condition checked is that no action has been selected yet this round as the environment accepts only one action each round. So after an action is executed the program section is done for this round. Next the agent checks whether it is disabled right now. If so it will start the disabled module. Next the agent checks whether the team controls the whole map because then the agents will not have to swarm but can do other things. If this is not the case the role specific module of the agent is run and if this still does not yield an action it will go exploring.

4.3 Evaluation and Analysis

After the HactarV2 agent team was completed and the actual contest was held it was time to analyze the results and to evaluate the usage of the programming methods gathered from the UT2004 project. As mentioned at the end of section 4.1.3 the HactarV2 team won the contest for which the team had created the MAS and this is a good indicator that the programming methods used work.

To evaluate the HactarV2 team the same method is used as used to compare the UT2004 teams. This made it possible to check the use of all the methods that
program {
  %Only try to find a new action if none was executed yet
  if bel(not(doneAction)) then {

    %If disabled get yourself fixed as soon as possible
    if bel(disabled, not(role('Repairer'))) then disabled.

    %Perform specific behavior when we have the entire map
    if bel(allMapAreBelongToUs) then superioritySelect.

    %Enter your role specific module
    if bel(role('Repairer')) then repairerAction.
    if bel(role('Inspector')) then inspectorAction.
    if bel(role('Explorer')) then explorerAction.
    if bel(role('Saboteur')) then saboteurAction.
    if bel(role('Sentinel')) then sentinelAction.

    %Apparently you had nothing role specific to do so go explore
    if bel(true) then explore.

    %Otherwise skip (should never happen)
    if bel(true) then skip.
  }
}
of updating the belief base. Code fragment 11 shows the knowledge rules that handle percepts. While some like money(M) give only a limited time gain by removing the storing of the percepts others like visibleEdge(Node1,Node2) and vertexOwner(Node,Team) can prevent a large number of saving operations especially when the agents are swarming and controlling a large number of nodes. But this means that knowledge rules are used for beliefs. And that percepts are used outside of the event module by something else as forall rules. Though you could say that this does not count against “Percepts are handled in forall rules.” but in the event module a lot of the percepts are also handled by if then rules.

% Extraction some information about this agent from the percepts
money(M) :- percept(money(M)).
energy(E) :- percept(energy(E)).
maxEnergyWorking(E) :- percept(maxEnergy(E)).
maxEnergyDisabled(E) :- percept(maxEnergyDisabled(E)).
strength(S) :- percept(strength(S)).
maxHealth(H) :- percept(maxHealth(H)).

% Extracting visible entities, vertices and edges from the percepts
visibleEntity(Id,Node,Team,Status) :- percept(visibleEntity(Id,Node,Team,Status)).
vertexOwner(Node,Team) :- percept(visibleVertex(Node,Team)).
visibleEdge(Node1,Node2) :- percept(visibleEdge(Node1,Node2)).
visibleEdge(Node1,Node2) :- percept(visibleEdge(Node2,Node1)).

% Extracting the agents current position from the percepts
currentPos(Agent,Node) :- percept(visibleEntity(Agent,Node,e_,_)).

% Extracting round information from the percepts
lastAction(Action) :- percept(lastAction(Action)).
lastActionResult(Result) :- percept(lastActionResult(Result)).

Code Fragment 11: The use of knowledge rules to see percepts as beliefs in a more time efficient way. Lines beginning with % are comments.

This leaves the question why the HactarV2 team did not handle the item of “Any goal management is at the end of the event module,” correctly. This stems from the limited use of goals in the code. To be precise goals are used only three times. The first and most important is the swarm goal used by all agents to express the desire to be part of the swarm and to expand it. The second goal optimum is used by the explorer role to show the desire to find the optimum node. The last goal hunt(ID) is the desire of the saboteur to hunt a specific enemy agent when the HactarV2 team controls the whole map. And while the swarm goal is neatly handled in a sub module of the event module and the optimum goal does not need to be managed the hunt(ID) goal is managed in the saboteur modules because it was added at the last minute on the first day of the contest to implement an extra functionality and because of this hasty implementation the code style was forgotten.

Another case of choosing for time efficiency before understandability comes from the second item of the code patterns section of the checklist which was the choice at the start of the project to use insert with negations instead of insert + delete. Analyzing the code at the end of the project it becomes clear that the original idea to only use insert with negation was not followed consistently.
as in some parts of the code insert + delete is used either for the sake of un-
derstandability or personal taste. Checking on where insert + delete is used, it
becomes clear that the use of insert + delete is restricted to less time critical
parts of the code which are not used repeatedly.

The other item in the code patterns section asks if there is any code pattern
used in the code. The answer is of course yes as described in section 4.2. The
HactarV2 used the strategy pattern not just in the main module as can be
seen in code fragment 10 but uses it again twice in the event module once to
handle role specific percepts and once to handle role specific messages. The code
for the main message handling module is included in code fragment 12. This
wide spread use of the strategy pattern saves the agent the time of evaluating
a large number of rules that are not relevant to its role and it also keeps a clear
separation between the event module and the normal agent instructions.

% Makes sure agents process mail that is relevant to their role
module selectReceiveMail{
  program[order=linearall]{
    % Handle mails that everyone uses.
    if true then commonReceiveMail.
    % Handle mails specific for your role.
    if bel(role('Explorer')) then explorerReceiveMail.
    if bel(role('Saboteur')) then saboteurReceiveMail.
    if bel(role('Repairer')) then repairerReceiveMail.
    if bel(role('Inspector')) then inspectorReceiveMail.
    if bel(role('Sentinel')) then sentinelReceiveMail.
    % Handle mails that disabled agents need.
    if bel(disabled) then disabledReceiveMail.
    % Clean up mailbox.
    if true then clearMailbox.
  }
}

Code Fragment 12: The use of a pattern similar to the Strategy design
patterns in the message handling system of the HactarV2 team. Lines beginning
with % are comments.

The next section of the checklist covers the use of documentation during the
project. As described above the documentation consisted out of two parts: the
ontology and the project management system. At the start of the project it
was decided to use iceScrum as the project management system. Unfortunately
half way through the project technical difficulties with the server running the
iceScrum system made access unreliable. This forced the team to work without
the project management system. By the time the problems where resolved the
team had no inclination to use the system again as all project members knew
what their general tasks were until the end of the project and all incidental
tasks were divided in conference. While the iceScrum server was working the
use of the system varied per team member, some used it to its full potential
while others only used a few features. This means that for the purpose of our
evaluation it is impossible to give a reasonable analysis of the use of the project
management system.

In contrast with iceScrum the ontology was used throughout the life of the
project though again the use per team member differed. When looking at the items for the ontology in the checklist in section 3.4.1 and checking them against the use of the ontology during the project all items were marked off. This means that the ontology was kept up to date from the start of the MAPC project all the way to the end. It was used to look up the function or syntax of the predicates, to ensure that there were no multiple predicates with the same function and to keep naming of predicates consistent. All these uses where made easy by the clear overview of the ontology as can be seen in figure 15 in section 4.2. This clear overview and the fact that the ontology was kept up to date throughout the project greatly enhanced the understandability of the code though it is hard to say exactly how much it contributed to the success of the HactarV2 team in the competition.

The last section of the checklist covers the use of testing throughout the project. Just as with the ontology all items from the checklist that deal with this subject are marked off for the testing procedures of the HactarV2 team. This meant that the HactarV2 team did unit tests on individual modules and full system tests. The team also did not only test against the basic bots that were supplied with the environment but also tests through the organizers of MAPC against other participating teams. Besides this the team tested new versions of their team against older versions to make sure that changes in their behavior improved the overall performance. Because the maps are randomly generated it is practically impossible to test on every possible map but any tests concerning navigation and exploration was not considered passed until it was repeated successfully on multiple generated maps. With all these different tests throughout the whole development process the test frequency was very high for this project with about 40 % of the project time spent testing. This amount of testing is only useful and doable if the code has a high level of testability. The modular set up of the agent and the clear flow of behavior through the Strategy patterns helps to provide this.

The members of the development team of HactarV2 acknowledge that testing has contributed to the success of their team in section 2.1 from their paper [4]. With the amount of time spent on testing by the HactarV2 team and the results they had during the competition there is a clear indication that consistent testing improves the quality of a multi-agent system.

4.4 Lessons learned

After analyzing the HactarV2 code and evaluating the performance of the agents and the development process we have learned some lessons that can be used to answer the research question.

The first lesson is not so much a repeat of one of the lessons from the UT2004 project as emphasizing and expanding on the lesson and it concerns the use of testing during the development of a MAS. While it is already suggested in section 3.5 that testing could improve the quality of the MAS, based on the results and experience of testing the HactarV2 MAS it can safely be concluded that frequent, consistent and varied testing helps the creation of a high quality MAS. An important note here is that not only the code of the team should be tested but also the connection to the environment, especially if the computer running the environment is not in the same physical location as the computer running the agent team. Because HactarV2 started early with testing against
other groups the few problems with the connection could be fixed in time for the actual competition while teams that did not test this connection in time encountered problems during the competition with bad connections and time-outs.

On the subject of the ontology it needs to be added that including an explanation of the arguments that are part of the predicate can greatly help developers to work on the code. It means there is one place they can find all the needed information on all predicates when they want to use one without the need to look them up in the code.

The most important lesson that goes against the lessons from the UT2004 project is that while the code style described in the GOAL programming guide supports understandability and testability it sometime has to give for efficiency. The MAPC project is a clear example off this situation because if the team had kept perfectly to the code style their agent team would have been to slow on some occasions. This would have meant the team had to simplify their strategy or the system would have had problems with missed actions. Both of which would have reduced the outstanding results the team achieved.

The HactarV2 code again shows the advantages of the Strategy design pattern by not only implementing it in the main module but also applying it in the event module. This multiple use of the strategy pattern could not be achieved without the clear separation of common modules from role specific modules. This was done not only for the main behavioral modules but also for modules for percept and message handling that are different for specific roles.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Lessons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Quality</td>
<td>Sometimes efficiency is more important than code style.</td>
</tr>
<tr>
<td></td>
<td>Combine dependent modules for a role in one file.</td>
</tr>
<tr>
<td></td>
<td>Keep comments clear and concise.</td>
</tr>
<tr>
<td>Design Pattern</td>
<td>Use multiple Strategy patterns for different module types.</td>
</tr>
<tr>
<td></td>
<td>Separate common modules from role specific modules.</td>
</tr>
<tr>
<td>Documentation</td>
<td>Include argument explanation in the ontology.</td>
</tr>
<tr>
<td>Testing</td>
<td>It is useful to test against an old version of a team.</td>
</tr>
<tr>
<td></td>
<td>Do connection tests in an early stage when connecting to physically distant systems.</td>
</tr>
</tbody>
</table>

Table 26: A short overview of the lessons learned from the analysis and evaluation of the HactarV2 agents and the development process that created it.

Table 26 gives a short overview of all the lessons learned. In case of Code Quality it reiterates the point that sometimes the efficiency of an agent is more important that the code style as there are cases where adherence to the normal code style can make the agent incapable of making deadlines required by the environment. The next lesson on Code Quality is that not all modules should have their own file as they are just a sub module that is called in multiple locations by another (master) module. By keeping these dependant modules in the same .mod2g file you prevent the MAS from having too many files and keep all the code of the master module in one file. The last lessons advises to keep the comments used clear and concise as to long comments do not enhance understandability as much as concise comments.
On the subject of Code Constructs and Patterns the lessons listed in table 26 include the advice that the strategy pattern can not only be used for the main module but also for other modules like the event module and can be used multiple times in the same agent program. The other lesson teaches that modules like event and knowledge modules can be split in separate modules for the parts that are role specific and those that are shared by all roles.

The only lesson in the table on Documentation is the advice to explain the argument of a predicate in the ontology to prevent developers from having to look up what the different arguments are used for in the code.

The subject of Testing teaches us that a lot can be learned on the progress of a system by testing against older version of the team. The older version are also a good benchmark to see whether the newer version is actual an improvement over the old ones. The second lesson is that if the machines that run the environment and the agents are physically at a great distance it is a good idea to do connection tests before actually using the system as a connection with high delays can influence the shorten the deadlines that a team has to meet in production.

With the lessons collected in table 26 it is time to conclude this research and to answer the research question.
5 Conclusions and Future Work

5.1 Conclusions

After analyzing and evaluating the results of two projects we have learned various lessons about what good GOAL code is. These will help to answer the initial research question: “How can we improve on teaching agent-based programming?” To answer this main research question the following two sub-questions will be discussed first:

- Are there any programming constructs in the programming language that better performing students use that are not used by lesser performing students?

- Are there any software quality factors that better performing students focus on and that lesser performing students tend to ignore?

To find the answer to these questions research was done by comparing the strongest and weakest teams participating in the UT2004 project. The comparison focused on the use of programming constructs and software quality factors. Instead of comparing the top and bottom half of the seventeen groups that participated in the UT2004 project only the top and bottom four groups are compared. The results of this analysis were then applied in the MAPC project. This was done to check whether the research results could be confirmed.

To start answering the first sub-question, “Are there any programming constructs in the programming language that better performing students use that are not used by lesser performing students?”, we found both a code construct and a code pattern.

The code construct concerns the use of the GOAL syntax as it is used to add and remove beliefs. The construct details the use of an insert clause with negated beliefs instead of a separate insert and delete clauses as explained in section 3.4.3. While using only an insert clause with negation provides a slight improvement in performance the use of both insert and delete clauses provides easier to read code. But in general the differences between these two uses of the syntax are so small as to be unimportant and the construct used can be selected on project or personal preference though it is best to stick to one method during a project.

The code pattern found during this research is a pattern that looks like the Strategy design pattern. It consists of the use of modules to separate role or task specific behavior for each role/task from behavior that is common between all roles/tasks. And it consists of a module selection method based on the role/task the agent has at that point in time. Code examples of these selection methods can be seen in earlier sections in code fragments 9 and 12. The use of this code pattern greatly differed between the strongest and the weakest four groups of the UT2004 project as all the four top groups used it while only one of the bottom groups used it. This code pattern was used again during the MAPC project for the HactarV2 team which did not only use it for the normal behavior but also used it for percept and message handling. This meant that there were separate modules for the handling of common and role/task specific percepts and messages. This indicates that the modularization of the HactarV2 code was done with forethought.
As the Strategy patterns is a well known OOP design pattern [20] it is interesting that it is found in an AOP project. Aridor and Lange [22] and Oluyomi, Karunasekera and Sterling [23] mention numerous AOP specific design patterns but do not mention the use of any OOP patterns to solve AOP problems.

The concluding answer to the first sub-question is that better groups make successful use of the Strategy code pattern and the separation of role specific and common modules that is needed to use it. The use of this pattern in the MAPC project confirmed that this is indeed a good programming practice.

The second sub-question “Are there any software quality factors that better performing students focus on and that lesser performing students tend to ignore?” deals with software quality factors focusing on Understandability and Testability as explained in section 2.3.3.

A part of understandability in GOAL programming is knowing what each predicate does and how to use it. To help the groups of the UT2004 project keep track of their predicates a part of the assignment is to keep track of them using an ontology. Section 3.4.4 shows that the various groups used this system to various degrees. But when comparing the top four groups with the bottom four groups it became clear that the top groups put more effort keeping the ontology up to date and made more use of it in general though the difference was not that big. This difference was enough to entice further research which was done by creating an ontology for the HactarV2 project and though it was used in different amounts by the team members it was generally seen as a useful thing which enhanced the understandability of the agent.

To have a high testability factor a program should be structured in such a way that it is easy to test in different ways and that the test results from those tests can be used to improve the program. Section 3.4.5 shows that the testing done by the various teams differs greatly with some teams only doing sporadic full system tests while other groups had high test frequencies, with testing of individual modules and testing against older versions of the team to see whether the strategy had improved. When comparing the testing of the top and bottom four groups of the UT2004 it is immediately clear that the top groups have a far higher test frequency and more often do multiple types of tests. In contrast the bottom four groups on average have a low to medium test frequency and mostly stick to full system tests. Group 13 mentions in their report that they found it very hard to do unit tests at one point during the project. This shows that their code at that point was not structured for testing indicating a low testability.

As part of the analysis of the software quality factors the UT2004 teams were compared using the checklist found in section 3.4.1. When looking at the average number of items marked off for both the top four and the bottom four groups it is clear that the top groups have more items marked off. But the difference is not significant especially as we consider that a big part of the difference is caused by the poor performance of group 11 on the subject of code style as can be seen in section 3.4.

The answer to the second sub-question is that while better groups generally score better at the various software quality factors the difference is not that great where understandability is concerned but the top four groups do a lot better on the subject of testability. This pattern is confirmed by the HactarV2 team which does not have a higher understandability than some of the UT2004 groups but has a very high testability as shown in section 4.3.
Considering the answers to the two sub-questions we conclude that there are several suggestions to answer the overall question, “How can we improve on teaching agent-based programming?” These suggestions are listed in short in the following list and will be explained more fully in the rest of this section.

- Teach the use of the Strategy Pattern.
- Teach the use of an Ontology.
- Teach the use of a clear Code style.
- Teach the use of a testing with a high frequency and different types of tests.

The first suggestion is to use the Strategy patterns and the modularization that is needed to use the patterns. It is especially effective in multi-agent systems where agents don’t know what their role will be at start up or in systems where the role of an agent can change (dynamically) during its life time.

The second suggestion is to set up an ontology at the start of the project and keep it up to date. This is especially useful for projects where the developers work in groups or when they only work on the project on and off. It will save a lot of time searching through code for predicates and will keep their agents understandable.

The third suggestion is to show students a concise and clear code style that can be used during programming and show them what the advantages are of using this style. But students should also be made aware that there are cases where the code style should be sacrificed for performance reasons.

The last suggestion is to emphasize the need for and use of repeated testing in various forms. In section 4.3 we show that the HactarV2 team spent about 40% of their time on testing. This seems much but in Dekker et al. [4] the team describes that they would have had worse results if they had spend less time on testing. This idea can be applied to the development of any MAS and should therefore be a clear focus point. An important part of this idea is that it is explained to the students how they can perform different kinds of tests like unit tests and full system tests and how they can structure their agent in a way that makes testing easy as opposed to how group 13 from the UT2004 initially structured their agent team\(^1\). In section 5.1.2 we provide guidelines to structure test procedures for a MAS.

Following up on these suggestions will contribute to the improvement of teaching methods for agent-based programming and future students will benefit from this research when they attempt to learn to program multi-agent systems.

5.1.1 GOAL IDE improvement

Besides suggestions for the improvement of teaching methods we gather suggestions to improve the GOAL IDE. These suggestions were gathered during the UT2004 and MAPC projects. In the comments on the questionnaires and while talking about the GOAL language with students we received a lot of feedback on how the IDE currently works and what kind of improvements would be a good idea and while some of this feedback was non constructive and some of

\(^1\)See section 3.4.5.
it was contradictory there were several suggestions that kept returning. In the following list the most repeated comments that can reasonably be implemented are listed.

- Add auto completion for the names of beliefs, actions, modules and knowledge predicates.
- Add the ability to the IDE that lets you double click on the name of an action, module or predicate and then jumps the cursor directly to the definition in the code, in the same manner as already possible in common development environments like Eclipse and NetBeans.
- Add the possibility to click on an error to jump directly to the line and place in the code that created the error.
- Currently GOAL has its own IDE but as some of the other suggestions in this appendix are functionalities that are already part of common development environments like Eclipse and NetBeans it is suggested that instead of adding all these functions to the GOAL IDE it might be easier to integrate GOAL in the common development environments. (At the end of our project a project has been started to integrate GOAL in NetBeans.)
- Split the use of breakpoints into two different types. The first has the current function which activates when a line of code is used and the second the function that activates when the rule is evaluated no matter whether the rule would actually fire or not.
- Add the ability of adding log files to the IDE which can be played back in later stages on a step by step basis. This means that evaluation of tests can be done after the tests are completed without the need to pause the environment. In this log it should also be clear which action was executed and preferably which line was used to activate it.
- Add the ability to go into sleep mode when no percepts are being received.
- Add a way to automatically generate documentation like ontology’s out of the code and the comments. This could be done in a similar way as JavaDoc is generated for Java programs.

Implementation of the listed suggestions will improve the ease of use of the GOAL IDE. It will also improve the general understandability and testability of GOAL programs. For instance the suggestion to be able to click on an error to jump to its origin will improve testability by making debugging easier. Another example is the ability to automatically generate documentation in a similar manner as JavaDoc. This will improve understandability as it makes it easier to keep all documentation consistent and up to date.

5.1.2 MAS Testing

Testing a MAS is difficult as the behavior is usually not predictable because of the interaction of the agents which all make their own decisions. This means that testing all possible situations and configurations is virtual impossible. So instead of a test plan that covers all possible situations a test protocol is needed.
for testing a MAS. This test protocol facilitates both the finding of bugs and the testing of the strategy of the MAS. In this section we provide the basic setup for such a test protocol based on the experience gained during the UT2004 and MAPC projects. This protocol is based on the Three-Layer Approach by Salamon [18]. In this approach the three layers of testing are unit tests, agent tests and full system tests.

Unit tests should be the first tests done on new pieces of code. They cover the separate testing of modules and should be conducted each time something is changed in a module. This layer also covers tests to check interaction between modules when this is an important part of the functioning of a module. For instance a communication module should be checked to correctly interact with another communication module.

The second layer is agent tests in which individual agents are tested to see if they function as they are supposed to and their modules work correctly. These tests should be done each time an agent has been changed and when all the module tests belonging to the agent are completed without problems. It can also be necessary to test the interaction between agents in this layer in a similar manner as interaction between modules is done in the first layer and for similar reasons.

The last layer is full system tests. The tests in this layer are focused on finding any problem in the interaction between agents and to check that the MAS functions as is should. In competitive settings it is important to test against different opposing teams to prevent the system from being over trained against a particular opponent. It is also a good idea to test against old versions of your own team to check if the new team is an actual improvement over the old one. If not it might be necessary to change the team further or to revert to the old team and try another strategy change. The full system tests should be done whenever an agent has being changed and the agent and module tests belonging to the agent are successful. It can also be useful to run extra full system test to find flaws in the strategy of the MAS which are not really bugs but are places where the agent can be improved.

5.2 Future work

My research has answered some questions, revealed some interesting information but there are still questions left. In this section we will highlight a few that warrant future attention.

The first question follows from the concept of design patterns in Agent oriented languages. We have already noted in section 2.3.4 that there is already research in the field of agent oriented design patterns. The research shows much promise but more research is needed to find out if and how these patterns can be used in the GOAL language. Another avenue of research is finding out whether more patterns can be identified that can be seen as good programming practices. Lastly it is interesting to find out whether more design patterns that originate from OOP can be converted to the use of AOP.

This may be done by comparing the various teams in the MAPC project in a similar way to how we compared the UT2004 teams to find out why certain teams perform better than other teams. This is possible as the code of all participating teams can be found on the MAPC site [3] and the teams have written papers on the development process of their MAS. It will be harder than
our comparison because the teams use different agent oriented languages which complicates the comparison.

As our research focuses on two software quality factors out of eleven as described in section 2.3.3 another question is how the other software quality factors contribute to the performance of a MAS. An extra dimension to this is the question which quality factor contributes the most to the performance or does it depend on certain aspects of the context of the MAS.
References


A  Example Bot Code

In this appendix we have listed the code of the example bot that was provided to the students that participated in the Agent programming project at the Delft University of Technology as described in section 3. To use the code you will need to have GOAL installed and a connection to the UT2004 environment. Installation files and instructions can be found at the UT2004 environment page [2].

A.1 MAS file

The MAS or .mas2g file describes how the agent should be set up.

```goal
environment {
  "UnrealGoal2.jar".
  init {
    botNames = ["SimpleRed 1"],
    visualizerServer = "rmi://127.0.0.1:1099",
    botServer = "ut://127.0.0.1:3000",
    logLevel = "OFF",
    showFocalPoint = "true",
    skill = 1,
    skin = "BotA",
    team = "red",
    waitForInterpretation = "false"
  }.
}
agentfiles {
  "betweenFlags.goal" [name=agent] .
}
launchpolicy {
  when [type=bot, max=1]@env do launch *:agent .
}
```

A.2 GOAL file

The .goal file contains all the code for this simple example agent without the need of separate modules.

```goal
init module {
  knowledge {
    beliefs{
      state(reached([''])).
      moving(unknown, unknown, unknown).
      % list of weapons that the bot picked up so far.
      pickedupWeapons([''])..
      maxLevelOfArmour(0).
    }
  }
  goals{
    visitFlags.
  }
  actionspec{
    goto(Route) {
      pre { state(reached(X)) }
    }
```
post { not(state(reached(X))), state(moving(Route)) }
}
}

main module {

program{
  if bel(flagPos(UnrealLocID)) then goto([UnrealLocID]).
}
}

event module {

program {
  forall bel( percept(flagBase(Team,NavPoint)) )
    do insert( flagPos(NavPoint) ).

  forall bel(moving(OLoc,ORot,OVel), percept(moving(Loc, Rot, Vel)))
    do insert( not(moving(OLoc,ORot,OVel)), moving(Loc,Rot,Vel) ).

  forall bel( percept(reached(Route)) )
    do insert( state(reached(Route)), not(state(moving(Route))) ).
  forall bel( percept(stuck(On, Route)) )
    do insert( state(reached(Route)), not(state(moving(Route))) ).

  % do not remove or modify percept rules below
  forall bel( percept(status(_,_,Armour,_)),
    maxLevelOfArmour(LastMaxArmour), Armour > LastMaxArmour )
    do insert( not(maxLevelOfArmour(LastMaxArmour)),
      maxLevelOfArmour(Armour) ).

  if bel(percept(weapon(Weapon,_,_)), not(Weapon=translocator),
    pickedupWeapons(PUW), not(member(Weapon, PUW)))
    then insert( not(pickedupWeapons(PUW)),
      pickedupWeapons([Weapon | PUW] ).
}
}
B Questionnaire questions

These are the questionnaires distributed among the first year students that participated in the Unreal Tournament 2004 project of 2011 as described in section 3. The questionnaires are listed here in a chronological order. The first questionnaire was distributed and collected on the first day of the project, the second in the fourth week and the last was filled in on the last regular day of the project.

B.1 First Questionnaire

Programming questionnaire

1. Introduction

The purpose of this questionnaire is to get an idea of the perceived skill level of first year students with respect to writing programs in both Java and GML. The results will be used to improve agent programming methodologies and teaching methods.

We will conduct a lottery where questionnaire participants can win prizes. There are six prizes of €25 that we will be giving away. The winners of the prizes will be announced the session after the questionnaire is filled in. The questionnaire will take about 10 minutes to complete.

Important: At the start of the questionnaire you will be asked to fill in both your name and student id. This information is needed to connect the results of the questionnaire with your project results and to award the prizes. This information will be treated as completely confidential, will not be shared with any third parties and no information that can be used to identify individual participants will be included in the reported results. We want to stress that the answers to this questionnaire will NOT influence your grade for the project in any way.

The questionnaires are part of a Master thesis research project into programming methodologies used for agent programming languages.

NA = Not Applicable

2. General Questions

1. Name: ________________________________________________

2. Student-number: _______________________________________

3. Did you have any experience in programming before you started your study at the TU Delft? If so, which languages have you used to write programs?

4. How many tutorials (werfcolleges) for the TH1609-A course did you attend?


3. Java

5. In how many courses in which Java is used have you participated?

6. On a scale of 1-10 how do you rate your ability to write a program in Java:
   
<table>
<thead>
<tr>
<th>Basic skill</th>
<th>Expert skill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>NA</td>
</tr>
</tbody>
</table>

7. On a scale of 1-10 how do you rate the usefulness of the tutorials for Java:

<table>
<thead>
<tr>
<th>Not useful</th>
<th>Very useful</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>NA</td>
</tr>
</tbody>
</table>

8. What would you need to improve your skills in Java?

9. On a scale of 1-10 indicate how much the programming style of Java appeals to you:

<table>
<thead>
<tr>
<th>Not appealing</th>
<th>Very appealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>NA</td>
</tr>
</tbody>
</table>

10. On a scale of 1-10 rate your understanding on how to use the language elements in Java:

<table>
<thead>
<tr>
<th>Basic understanding</th>
<th>Expert understanding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>NA</td>
</tr>
</tbody>
</table>

4. GOAL

11. On a scale of 1-10 how do you rate your ability to write a program in GOAL:

<table>
<thead>
<tr>
<th>Basic Skill</th>
<th>Expert Skill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>NA</td>
</tr>
</tbody>
</table>

12. On a scale of 1-10 how do you rate the usefulness of the tutorials for GOAL:

<table>
<thead>
<tr>
<th>Not useful</th>
<th>Very useful</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>NA</td>
</tr>
</tbody>
</table>

13 On a scale of 1-10 how do you rate the course material for GOAL (slides, manuals, etc.):

<table>
<thead>
<tr>
<th>Not useful</th>
<th>Very useful</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td>NA</td>
</tr>
</tbody>
</table>
14. What would you need to improve your skills in GOAL?

15. On a scale of 1-10 indicate how much the programming style of GOAL appeals to you:
   Not at all appealing 0 0 0 0 0 0 0 0 0 Very appealing 0
   1 2 3 4 5 6 7 8 9 10 NA

16. On a scale of 1-10 rate your understanding on how to use the language elements in GOAL:
   Basic understanding 0 0 0 0 0 0 0 0 0 Expert understanding 0
   1 2 3 4 5 6 7 8 9 10 understanding NA

17. Do you want to provide any other comments on programming in GOAL or the agent-oriented style of programming in general?

   5. Other
   18. Any other comments you would like to add?

   Thank you for your time.
B.2 Second Questionnaire

Programming questionnaire

1. Introduction
The purpose of this questionnaire is to get an idea of the basic skill level of first year students with respect to writing programs in GOAL. Besides this we will look at programming methods used during the individual assignment. The results will be used to improve agent programming methodologies and teaching methods.

We will conduct a lottery where questionnaire participants can win prizes. The winners of the prizes will be announced the session after the questionnaire is filled in. The questionnaire will take about 10 minutes to complete.

Important: At the start of the questionnaire you will be asked to fill in both your name and student-id. This information is needed to connect the results of the questionnaire with your project results and to award the prizes. This information will be treated as completely confidential, will not be shared with any third parties and no information that can be used to identify individual participants will be included in the reported results. We want to stress that the answers to this questionnaire will NOT influence your grade for the project in any way.

The questionnaires are part of a Master thesis research project into programming methodologies used for agent programming languages.

NA = Not Applicable

2. General Questions

1. Name: __________________________

2. Student number: __________________________

3. Group Number: __________________________

3. Programming Questions
This part of the questionnaire will feature twelve multiple choice or true/false questions. Please circle the answer you think is correct.

1. Which of the following file-extensions is not a GOAL extension?
   a. .goal
   b. .mas
   c. .mas2g
2. What does the following code line do?
   if bel(received(X, Y)) then adopt(Y).
   a. Produce an error
   b. Create the goal Y when a message Y is received.
   c. Create the belief Y when a message Y is received.

3. In what module do you place action specifications if you want to use them throughout the whole agent?
   a. main module
   b. any module
   c. init module

4. True or False: In action specifications variables are only bound in the pre-condition.
   a. True
   b. False

5. True or False: Under all circumstances sendone(X, Y) will only be sent once while an agent is running.
   a. True
   b. False

6. Which file type can not be imported with the #import directive in GOAL?
   a. goal
   b. mod2g
   c. .pl

7. What should the precondition of the following code be?
   goto(Route) {
     Pre{...
       Post{not(state(X)), state(moving(Route))} }
   }
   a. state(moving(X))
   b. state(X)
   c. state(halted)

8. What does the following line of code do?
   if a-goal(have(X)), bel(have(X)) then delete(have(X)).
   a. If the goal have(X) is fulfilled it drops the belief that the goal is fulfilled.
   b. If there is no goal have(X) but the belief exists the belief is deleted.
   c. Nothing, the condition of the if-statement is never true.

9. True or False: In GOAL every rule should be terminated with a point.
   a. True
   b. False
10. How many bots will be automatically started by the following code?

```java
environment {
    "UnrealGoal.jar".
    init [  
        bots = ["SimpleRed ",
                botServer = "ut://127.0.0.1:3000"
            ];
    }
systemfiles [  
    "example-goal.goal" [name=agent].
]
lsvspolicy [  
    when [type=bot, max=3]@env do launch *agent .
]
test  
    a. 0  
    b. 1  
    c. 3  
```

11. True or False: If you want to perform an action in an agent’s environment, then you need to write an action specification for that action.

a. True  
b. False  

12. Which of the following program rules from agent sender will place the belief `received(sender, imp harass(link gun))` in the mailbox of agent simple?

a. if bel(agent(X), X=simple) then send(X, have(link gun)) .

b. if bel(agent(X), X=simple) then send(X, !have(link gun)).

c. if bel(agent(X), X=simple) then send(X, ?have(link gun)).

5. Other

1. Any other comments you would like to add?

Thank you for your time.
B.3 Third Questionnaire

Programming questionnaire

1. Introduction
The purpose of this questionnaire is to get an idea of the effect of the project on the skill level of first year students with respect to writing programs in GOAL. Besides this we will look the effect the project has had on the perceived skill level of students. The results will be used to improve agent programming methodologies and teaching methods.

We will conduct a lottery where questionnaire participants can win prizes. The winners of the prizes will be announced the session after the questionnaire is filled in. The questionnaire will take about 10 minutes to complete.

**Important:** At the start of the questionnaire you will be asked to fill in both your name and student id. This information is needed to connect the results of the questionnaire with your project results and to award the prizes. This information will be treated as completely confidential, will not be shared with any third parties and no information that can be used to identify individual participants will be included in the reported results. We want to stress that the answers to this questionnaire will **NOT** influence your grade for the project in any way.

The questionnaires are part of a Master thesis research project into programming methodologies used for agent programming languages.

NA = Not Applicable

2. General Questions

1. Name:

2. Student-number:

3. Group Number:

3. GOAL perceived skill level
This section features some scale questions and some yes/no questions.

1. On a scale of 1-10 how do you rate your ability to write a program in GOAL:

<table>
<thead>
<tr>
<th>Basic Skill</th>
<th>O</th>
<th>O</th>
<th>O</th>
<th>O</th>
<th>O</th>
<th>O</th>
<th>O</th>
<th>O</th>
<th>O</th>
<th>Expert Skill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>NA</td>
</tr>
</tbody>
</table>
2. Have your programming skills for programming GOAL agents improved during this project?
   Yes / No

3. On a scale of 1-10 indicate how much the programming style of GOAL appeals to you:
   Not appealing:  O  O  O  O  O  O  O  O  O  Very appealing
   1  2  3  4  5  6  7  8  9  10

4. Do you like programming in GOAL now more than at the start of the project?
   Yes / No

5. On a scale of 1-10 rate your understanding on how to use the language elements in GOAL:
   Basic understanding:  O  O  O  O  O  O  O  O  O  Expert understanding
   1  2  3  4  5  6  7  8  9  10

6. Has your understanding of GOAL language elements improved during the project?
   Yes / No

4. Programming Questions
This part of the questionnaire will feature twelve multiple choice or true/false questions. Please circle the answer you think is correct.

1. What should the precondition of the following code be?
   \[
   \text{halt();}
   \quad \text{pres[...],}
   \quad \text{post[not(state(X)), state(halted])}
   \]
   a. state(moving(X))
   b. state(X)
   c. state(pursue(X))

2. What does the following code line do?
   \[\text{if goal(Y), bel(received(X,Y)) then insert(Y).}\]
   a. Create the belief Y when a message Y is received and goal Y exists.
   b. Create the goal Y when a message Y is received and goal Y exists.
   c. Produce an error.
3. True or False: You must always give an action specification of every environmental action even if you do not use it in your agent.
   a. True
   b. False

4. What does the following line of code do?
   if goal-a(have(X)), bel(have(X)) then delete(have(X)).
   a. If the goal have(X) is fulfilled it deletes the belief that the goal is fulfilled.
   b. If there is no goal have(X) but the belief exists the belief is deleted.
   c. Nothing, the condition of the if-statement is never true.

5. Which of the following file extensions is not a GOAL extension?
   a. .max2g
   b. .goal2g
   c. .mod2g

6. Which of the following program rules from agent sender will place the belief received(sender, have(link_gun)) in the mailbox of agent simple?
   a. if bel(agent(X), X=simple) then send(X, :have(link_gun)).
   b. if bel(agent(X), X=simple) then send(X, !have(link_gun)).
   c. if bel(agent(X), X=simple) then send(X, ?have(link_gun)).

7. You can specify a macro in GOAL code as follows:
   #... constructiveMove(X, Y)
   a-goal(tower([X, Y | T]), bel(tower([Y | T]), clear(Y),
   (clear(X); holding(X))).
   What word is supposed to be at the ...?
   a. include
   b. define
   c. import

8. True or False: All of the variables in the postcondition of an action specification have to be bound in the precondition.
   a. True
   b. False
9. In what module do you place beliefs in your agent file?
   a. init module
   b. any module
   c. main module

10. How many bots will be automatically started by the following code?
    ```
    environment {
      "UnrealGoal2.jar",
      init [  
        botNames = ["SimpleRed", "SimpleGreen"],
        botServer = "127.0.0.1:3000".
      ]
    }
    agentfiles [  
      "example-goal.goal" [name=agent].
    ]
    launchpolicy [  
      when [type=bot,max=3]@env do launch *:agent .
    ]
    ```
    a. 3
    b. 2
    c. 1

11. True or False: One of the types of rules in GOAL is the forall .. then .. rule.
    a. True
    b. False

12. True or False: When the agent sender uses the send(all, finish(questions)) action it will have the received(sender, finish(questions)) in its mailbox next round.
    a. True
    b. False

5. Other

1. Any other comments you would like to add?

Thank you for your time.
C  UT2004 project Data

This appendix contains graphs and tables showing data collected throughout the UT2004 project that has not been presented in the main sections. The data is ordered in sections based on the data source to which it belongs.

C.1  Questionnaire 1

The following graph shows the number of students that perceived their skill at programming Java and GOAL at the various ratings. On average the students perceived their Java skill to be 7.12 and their GOAL skill to be 5.97.

The following graph shows how the students rated their understanding of how to use the language elements of Java and GOAL. On average they rated their understanding of GOAL at 6.1 and their understanding of Java at 7.5.
C.2 Questionnaires 2 and 3

The following graph shows the results of linking questions of questionnaire 3 with the questions of questionnaire 2.

The following table shows whether students thought whether they improved on various area’s over the course of the UT2004 project. It also shows how much
their average rating on the subject changed.

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Have your GOAL programming skills improved?</td>
<td>80</td>
<td>4</td>
<td>+1.03</td>
</tr>
<tr>
<td>Do you like GOAL more?</td>
<td>32</td>
<td>51</td>
<td>-1.2</td>
</tr>
<tr>
<td>Has your understanding of GOAL improved?</td>
<td>80</td>
<td>3</td>
<td>+0.8</td>
</tr>
</tbody>
</table>