TI2806 Contextproject - Final Report Draft

Virtual Humans - Group 4

https://github.com/tygron-virtual-humans

Sven van Hal - 4310055

Tom Harting - 4288319

Nordin van Nes - 4270029

Sven Popping - 4289455

Wouter Posdijk - 4317149

June 18, 2015

Contents

1 Introduction

This report is the final project report for Group 4 of the Virtual Humans for Serious Gaming Contextproject. The main goal of this project was to make a virtual human that could replace an actual human in the Tygron (Tygron, 2015) urban planning game. This virtual human should be able to play the game like it is a real human, this includes making rational choices as well as making choices based on emotions. For an in-depth description of our customers and their requirements, please read our Product Vision (Contextgroups, 2015).

The focus of our group was on making the emotion part of the virtual human. In this report you can find how we made this emotional part, you can also find a description of the software engineering aspect as well as the interaction design, the failure analysis and an outlook in what is next.

2 Overview

Our final product is a GAMYGDALA (Popescu et al., 2013) plug-in for the GOAL programming language (Hindriks, 2014). The final product can be split into two separate parts: the GAMYGDALA port and the GOAL plug-in.

2.1 GAMYGDALA port

“GAMYGDALA is an emotional appraisal engine that enables game developers to easily add emotions to their Non-Player Characters (NPC)." (Popescu et al., 2013).

GAMYGDALA is written in Javascript, but for it to properly work with GOAL, which is written in Java, we needed to port GAMYGDALA to Java. This GAMYGDALA port is our biggest software product. When we finished our initial port, we noticed that the Java code did not follow the Software Engineering principles. This is why we decided to do a total code refactor.

Our final version of the port is far better then the original port on account of the Software Engineering principles and it behaves in the same way as the original GAMYGDALA code. You can read more about this refactor in the Emergent Architecture Design (VH4, 2015) and in next chapter.

2.2 GOAL plug-in

The other part of our software product is the GOAL plug-in. We had to alter the GOAL source code to enable the usage of GAMYGDALA in GOAL. We now have a fully working plug-in that developers can use in their GOAL programs. The full GAMYGDALA functionality can be used within the GOAL environment.

3 Reflection from a software engineering perspective

There are unlimited ways to approach the development of a software program. To streamline the process of developing and to ensure product quality, it is necessary to adhere to software engineering guidelines and principles. This section will discuss the measures that have been taken from a software engineering perspective and in which way they have affected the final product and the development process.

3.1 Product

From the first day on it was clear there was a challenging task to solve: porting an existing code base, the emotion engine GAMYGDALA, from JavaScript to Java. While ”JavaScript” and ”Java” sound very similar, these are actually two very different languages and require a very different approach from a programming perspective. Java is an object-oriented language pur sang, while JavaScript, in practice, uses a more procedural programming style.

Because Joost Broekens, the original author of the JavaScript version of GAMYGDALA, has programmed GAMYGDALA using a semi-object oriented approach, we initially decided to port the classes and methods to Java one on one. The Java port was therefore immediately feature complete. However, from a software engineering perspective, the code lacked proper design patterns and thus overall quality due to the fact that JavaScript simply does not support such programming structures. Using code quality assessment tool InCode Helium (Intooitus, 2015), we identified major flaws in the code and started a large-scale refactoring process to incorporate proper software design principles. During this process, we have reduced or split up very large methods, refactored methods with redundant or duplicate code and have implemented design patterns[1] where appropriate. The code review by Software Improvement Group identified additional flaws, which have been corrected in the final product.

The next and final challenge to face has been the creation of a GOAL (Hindriks, 2014) plug-in for GAMYGDALA. The plug-in acts as an interface for the GAMYGDALA Java port, allowing other GOAL programs to utilize the emotion engine.

Again, we were dealing with multiple types of language structures. GAMYGDALA is very structural, primarily based on one thread, since it is made for simple games. In GOAL, however, every agent runs in a different thread at the same time, and with that also past each other sometimes. For example, sometimes agents would create relations to other agents that were not running. We have had to make some sort of relation queue so that these relations were only processed once the target agent became active.

Creating the new GAMYGDALA actions within the GOAL source code proved to be rather difficult. Though we had been able to create our new actions, the way that we had done this seemed quite dirty. GOAL was not particularly ready for the addition of new actions and we had to violate the open/closed principle a lot to make sure that our new actions were recognized and used. To make sure that this would no longer be the case, at least for our own project, we created classloaders that would automatically recognize any new actions when we would add them.

[1] For an overview of all design patterns that have been used, please review the Emergent Architecture Design (VH4, 2015).

We have tested our implementation, the Java port as well as the GOAL plug-in, thoroughly. For the GAMYGDALA port, a test suite written in JavaScript was available. In order to be able to execute these tests in a Java environment, we have adapted part of the JavaScript code and have written an interface for this test suite in Java. The result is then interpreted in a JUnit test to be able to use this in continuous testing.

3.2 Process

To streamline the development process, we have used an iterative, incremental and agile approach: SCRUM. Every week, we've determined which tasks we would like to have completed which of these tasks have to highest priority. The selected tasks for that "sprint" (a one week period in which the selected tasks are to be completed) are then, roughly equally, divided amongst team members. At the end of the sprint, an evaluation takes place and the plan for the next sprint is determined. If we were unable to complete a particular task, it is added to the sprint plan for the next week.

TODO: Meer over sprint.

To make collaborating easier, we have used GitHub to host our code. We have used multiple repositories for different parts of the project. The GAMYGDALA port, GOAL plug-in and reports all used a different repository. This enabled us to work seamlessly together at different parts of the final product. Furthermore, we've utilized GitHub using a pull-based development approach. This means that whenever a team member finished and tested a piece of code on a separate branch, a pull request has to be created. Another team member has to review the code and check it for possible errors, before the request can be merged in the master branch. Two sets of eyes on a particular piece of code increases the likelihood of finding bugs before they make their way into the final product.

To ensure code quality, we have used continuous integration. Each time a team member has created new software functionality, he must also provide unit tests for that particular functionality. For each commit to the repository, Travis, a continuous integration tool, runs all the unit tests and generates a test report, to make sure no existing functionality breaks. Also, Checkstyle is automatically executed to make sure the code formatting adheres to guidelines we've established at the beginning of the project. These measures aid other team members in determining the quality of a commit and whether or not a pull request can be merged.

4 Developed functionalities

It is essential for Non-Playable Characters (NPC) in games to have and feel emotions. Emotions are crucial in respect to gameplay decisions, especially in serious games. The virtual human to which we contribute with this project has to be able be affected by actions outside it's own influence, and feel emotions. Therefore, we have tried to find a way to use GAMYGDALA (Popescu et al., 2013) in GOAL.

4.1 Gamygdala Java Port

GAMYGDALA is an emotion engine for games originally written in JavaScript. A large part of the work done during this project has been porting the JavaScript version to Java while simultaneously optimizing code quality. While we have not developed GAMYGDALA ourselves, we will briefly discuss its functionality for completeness.

GAMYGDALA operates in an environment where actors, "agents", interact with each other. Every agent has goals to achieve. In the environment, events can occur which influence the likelihood of goal achievement for a particular agent. Taken into account all the events that have happened, one or more emotions are deduced from the agents internal emotional state. Furthermore, agents have reciprocal relations: agents "feel" for other agents and adjust their emotions accordingly.

4.2 Gamygdala in GOAL

The primary goal of the GOAL programming language is to allow people to program agents from a human point of view, so that they can simulate human behaviour with ease. GOAL is very good at modelling the logical side of humans, but it lacks the very reason for our right brain's existence: Emotion. To allow GOAL to support this vital part of human simulation, we have added GAMYGDALA functionality to GOAL. All actions that are available in GAMYGDALA are now also available in goal. Using the GAMYGDALA plugin, agents can do the following things:

- Tell GAMYGDALA it has a goal.
- Tell GAMYGDALA it no longer has a goal.
- Appraise an event. This is what makes it gain emotions.
- Establish a relationship with another agent.
- Recalculate the emotions based on the passage of time.

Using these actions, programmers can completely take emotion into account while programming agents that need to mimic human behaviour.

5 Interaction Design

This section contains our interaction design report. You can read who will be the users of our final product, how we let these users test our product and what we learned from the results of these tests.

5.1 Users

As stated before, our final product is a fully functioning version of GAMYGDALA in the GOAL programming environment. Therefore, our final users will be GOAL programmers who want to use emotions for their GOAL agents.

To find testers we looked for students who were familiar with the GOAL programming language. Because the GOAL programming language is a language that is taught during the first year of the Computer Science bachelor (Logic Based Artificial Intelligence and the Multi Agent Systems project) there are a lot of Computer Science students that are familiar with GOAL.

We decided to ask second year Computer Science bachelor students to help us by trying to use our GAMYGDALA plug-in in GOAL. We chose some students who are also working on our Virtual Humans for Serious Gaming Contextproject, as well as some students who work on other Contextproject. This way we could collect the opinions of students who already know about our product and how it should work, as well as the opinions from students that did not use our product before.

5.2 How do we test

The best way to let our testers use our product is by just letting them play with it. This is why we asked them to try to give an agent some simple emotions.

For this test we used the Thinking-aloud method (Nielsen, 2012). This means that we asked our testers to say anything that came up in their mind during the test. This was the easiest way to find out if the testers enjoyed using our product and if the usage was easy enough.

5.3 Results

The results will be discussed here in the final report. This draft does not contain these results yet.

6 Evaluation

In this section you can read our evaluation about our experiences during this Contextproject. It consist of an evaluation of the product, a failure analysis, an evaluation of collaboration between our group members as well as an evaluation of the collaboration between the separate groups of our Contextprojext.

6.1 Product

This subsection is divided into two parts, the GAMYGDALA port and the GOAL plug-in.

6.1.1 GAMYGDALA port

This was the first project in which we had to work with code that we did not make ourselves. We had to work with the GAMYGDALA engine that was already made, this was an entirely new experience. The initial idea was to literally port the GAMYGDALA Javascript code to Java, this was not a lot of work. When we finished this initial port, we found out that it did not really follow the Software Engineering principles. Because our Software Engineering skills are also rated during this project, we decided to start refactoring the port. You can read more about the problems that we encountered during this refactor in the Failure analysis subsection.

Now that the port is finished, we are quite happy we the result. Although the code is not perfect yet, it is a lot better then the port that we started with. It now follows the Software Engineering principles a lot better, it uses design patterns and for example the god classes are eliminated now. You can read more about this in our Emergent Architecture Design (VH4, 2015).

It was a great experience to work on someone else's code because we encountered problems that we did not encounter in previous projects. We learned a lot of new skills that will be useful in the future.

6.1.2 GOAL plug-in

A lot of what we wrote about the GAMYGDALA port also holds for our GOAL plug-in. We spent a lot of time in understanding how GOAL works "under the hood". We needed to truly get how GOAL works before we could start making a plug-in for it. Over time we gained more knowledge about GOAL and we started making a simple calculator plug-in. This took more time then we first anticipated, but we learned a lot from making it. With help from Vincent Koenen and Koen Hindriks, who are both major collaborators on the GOAL project, we have gained a good understanding of the GOAL source code. This knowledge was used later on to make the actual plug-in for GAMYGDALA.

Working with GOAL was a very useful experience. It has taught us about the difficulties of working with someone else's code. We have seen cases of beautifully engineered code that we have learnt from, as well as cases where the implementation lacked adherence to important design principles. By having been the people who had to add new code to someone else's code, we will be able to write better code for someone else to use in turn. Also, we have been able to learn how to adhere to most design principles, even if you are modifying code that does not, which is a rather difficult task.

Although, similarly to working with the port, we also worked on code that we did not write ourselves, there are also some differences between making the GAMYGDALA port and this plug-in. In the port code we worked on every class, but in the GOAL code we only added and changed code in certain classes. Making the plug-in was more about adding code and functionality and the port was more about copying the code and functionality.

A problem we have encountered with the GOAL plug-in is that GAMYGDALA primarily assumes that every agent is omniscient, while in GOAL each agent thinks and perceives for itself and therefore their beliefs about certain events may differ. Agents are able to appraise events for themselves, luckily, so we do get most of the GAMYGDALA functionality. Relations, however, do not work optimal. For example, if one agent would appraise that it has done something that is positive for itself and negative for another agent, this other agent would feel all kinds of negative emotions. However, when these two things are separated into the first agent appraising the positive side, and the second agent appraising the negative side, these negative emotions aren't nearly as intense. This is because the face that the other agent gets happy is also supposed to create negative emotions, but in this way it does not. If we were to do this anyway, we would be conflicting the GOAL philosophy. Also, solving this problem would require significant changes to GAMYGDALA itself.

It was very nice to have these two experiences within one project.

6.2 Failure analysis

Not everything went perfectly according to plan during this project. In this subsection you can read about the biggest problems that we encountered.

In the beginning of the project, our biggest focus was on gaining knowledge about how GOAL works. Here we also encountered our first problem. Until that point we did not have much experience in working with complex code from other people. We might have underestimated the time it would take to really understand what was going on in GOAL. This is why the calculator plug-in that was planned to take about a week actually took us a lot longer.

After the first weeks of the project we also started porting GAMYGDALA. The initial version of the port did not give us to many problems, it just took a lot of time to work trough all the Javascript code. The first big problem that we encountered was when we ran InCode Helium (Intooitus, 2015) over this first version. The results were quite bad from a software engineering perspective. As stated above in the report, this problem was solved by an extensive refactor of the entire code.

Another problem that we encountered during this project was that is very hard to plan the right tasks every week. A great advantage of this Contextproject over the others was that we did not have a tight schedule in which every step was clear from week 1. Although we really liked the reality side of this project, it made it also a lot harder to foresee which tasks would take a lot of time during each week. We just had to plan in any task that we could think of in the beginning and then see what came up during the week.

An ongoing problem during the refactoring was that if you change a certain class, all of the classes and tests that depend on it also need to be changed. This often led to a 'domino' effect of problems, solving one problem led to another which also led to another and so on.

6.3 Collaboration between our group members

Our group consists of five members. During the project we decided to involve everybody with all the different parts of the project as much as we could. This way everybody learnt something about all the different aspects of our product.

We did have a distribution of the responsibilities amongst the team members. This means that for example Wouter spend more time working on the GOAL plug-in and Sven H spend more time working on the port. We did this to make sure that every task was would be finished.

We did not encounter problems within our group during this project. We had clear rules that every team member followed and we all worked equally hard on this project. We are very happy with our group process during this project.

6.4 Collaboration between the groups

Our Contextproject differs from the other Contextprojects because the groups within our Contextproject are all working together on one product, instead of all the groups working on their own product.

This is why the collaboration between the different groups was also a huge part of this project. We worked together on code as well as on reports. The communication between the groups was great, we could ask each other questions and help each other with the project. We really liked this part of the project because it gives a realistic view on actual jobs that include working on a software project with different people and groups.

7 Outlook

In this section we will give an outlook on where our product can be used and on the future improvements to our product.

7.1 Usage of our product

In this project, our product will be used by one of the other groups to create a virtual human for the Tygron Engine. However, this is not the only field in which our product can be used. Our plug-in supports the full GAMYGDALA functionality, this means that it can be used in almost every GOAL project.

For example, our product can be used in the first year Multi Agent Systems project, were students get the opportunity to program a team of bots in the Unreal Tournament environment (EpicGames, 2015). The bad feelings for an opponent bot can grow worse every time it scores a point. This way your own bots can focus more on the bots that they hate more which are the best bots on the other team (the bots that are scoring the most points).

7.2 Improvements to our product

Our final product is not perfect yet. The time we had for the entire project was ten weeks, which is not enough to make a perfect product. The functionality of our port and plug-in passes all our tests and seems to be working like it should, but we do not yet have a 100% test coverage. This might be impossible to achieve, but our test coverage can be approved.

Another part that we can improve is the refactor. Refactoring, and making sure that all the functionality is kept the same, takes a lot of time. We managed to fix many design flaws that were in our initial port, but there are still some parts that could be improved.

The documentation could also be extended, we could create examples and assignments in which you could train yourself in using the plug-in.

References

Contextgroups. (2015, May). *Virtual humans for serious gaming product vision* (Tech. Rep.). Delft University of Technology. Retrieved from https://github.com/tygron-virtual-humans/port-deliverables/blob/master/report/Deliverables/Product%20Vision%20Nieuwe%20versie.pdf

EpicGames. (2015, June). *Unreal tournament.* Retrieved from https://www.unrealtournament.com

Hindriks, K. V. (2014, March). *Programming cognitive agents in goal* (Tech. Rep.). Delft University of Technology. Retrieved from http://ii.tudelft.nl/trac/goal/raw-attachment/wiki/WikiStart/Guide.pdf

Intooitus. (2015, June). *Incode helium.* Retrieved from https://www.intooitus.com/products/incode

Nielsen, J. (2012, January). *Thinking aloud: The #1 usability tool.* Retrieved from http://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/

Popescu, A., Broekens, J., & van Someren, M. (2013, January). *Gamygdala: an emotion engine for games* (Tech. Rep.). Delft University of Technology and The Informatics Institute of the University of Amsterdam. Retrieved from http://www.joostbroekens.com/files/Popescu_Broekens_Someren_2013.pdf

Tygron. (2015, June). *The tygron website.* Retrieved from http://www.tygron.com

VH4. (2015, June). *Virtual humans for serious gaming group 4 emergent architecture design* (Tech. Rep.). Delft University of Technology. Retrieved from https://github.com/tygron-virtual-humans/port-deliverables/blob/master/report/Deliverables/Emergent%20architecture%20design%20draft.pdf