TI2806 Contextproject - Final Report

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 25, 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.

There are two appendices, a manual for how to use the GAMYGDALA plug-in for GOAL and a small report on the scalability of our GAMYGDALA port.

2 Overview

The 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, there was a need to port GAMYGDALA to Java. This GAMYGDALA port is the biggest software product of this project. When the initial port was finished, we noticed that the Java code did not really follow the Software Engineering principles. This is why we decided to do a total code refactor.

The 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 the software product is the GOAL plug-in. The GOAL source code had to be altered to enable the usage of GAMYGDALA in GOAL. There now is 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 Developed functionalities

It is a great addition 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 to be affected by actions outside its own influence, and feel emotions. Therefore, we have tried to find a way to use GAMYGDALA (Popescu et al., 2013) in GOAL.

3.1 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, its functionality will be briefly discussed 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.

3.2 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.

4 Reflection from a software engineering perspective

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.

4.1 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 (SIG, 2015) 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.

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 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 this project, factories that incorporate classloaders were created, so that any new actions would instantly be recognized after they were added.

Another difficulty has been the relation management. GOAL is not a very linear language; multiple agents act in multiple threads, and they might not be active at the same time. Because of this, agents might be creating relations to other agents that are not currently active. If this call would be forwarded directly to GAMYGDALA, it would simply say that the other agent does not exist and then do nothing. To fix this, a relation will simply be saved in the GOAL-GAMYGDALA interface if its target does not yet exist. Then, when the target becomes active, the relation is autmatically applied.

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

The implementation of the Java port as well as the GOAL plug-in is thoroughly tested. JUnit, a Java unit testing framework (JUnit, 2014), was used to run unit tests which are developed for the better part of the code. The current line coverage, a metric to measure how much of the code is called in unit tests, is currently at 90%. Additionally, a test suite written in JavaScript was available for the original version of GAMYGDALA. To be able to use these tests in Java, part of the JavaScript code was altered and a Java interface for this test suite has been developed. The Java class to interface with the test suite utilizes a JavaScript engine, Nashorn, included with Java 8. Nashorn is the de-facto JavaScript engine for Java and provides high-performance (Oracle, 2015) and full ECMAScript 5.1 compliance. Alterations made to the GAMYGDALA JavaScript test suite include:

- Adapting the GAMYGDALA object instantiation to the new implementation of the Java port to be able to actually use it.
- Replacing JavaScript arrays with Java ArrayLists (a technique which is made possible by Nashorn, see: manual), to be able to use these objects as arguments in method calls.

The Java test class, which runs the JavaScript test suite, instantiates a new Nashorn JavaScript engine object which loads the test suite framework and subsequently runs all tests. The test suite returns a result string which is then parsed in a JUnit test to allow its usage in the existing unit test workflow.

4.2 Process

To streamline the development process, an iterative, incremental and agile approach was used: SCRUM (Rising & Janoff, 2000). Each week, the tasks which had to be completed were determined and ordered based on priority. Then, the most important tasks were selected for the *sprint* (a one week period in which the selected tasks are to be completed) and equally divided amongst the team members. At the end of the sprint, an evaluation takes place and the plan for the next sprint is determined. If a task has not been completed within the sprint window, it is added to the sprint plan for the next week.

To make collaborating easier, we have used GitHub (GitHub, 2015) 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 have 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.

Continuous integration has been used to ensure code quality. Each time a team member created new software functionality, unit tests were also to be provided for that particular functionality. For each commit to the repository, TravisCI (TravisCI, 2015), a continuous integration tool, runs all the unit tests and reports back with the test result. This workflow ensures no existing functionality breaks when new code is

developed. Additionally, Checkstyle, a code style checker, is automatically executed to make sure the code formatting adheres to guidelines that have been established at the beginning of the project. These measures aid other team members in determining the quality of a code commit and whether or not a pull request can be safely merged.

5 Interaction Design

This section contains the interaction design report. It contains information about the target audience of the final product, how these people have used the product and what has been learned from the results of these interactions.

5.1 Users

The final product is a fully functioning version of GAMYGDALA in the GOAL programming environment. GOAL programmers who want to use emotions for their GOAL agents are the target audience of the product.

To properly test user interaction with the GOAL plug-in, testers had to be found with appropriate background knowledge of the GOAL programming language. Because the GOAL is a language that is taught in the first year of the Computer Science bachelor, during the courses Logic Based Artificial Intelligence and Multi Agent Systems project, there are a lot of Computer Science students that are familiar with GOAL.

Second year Computer Science bachelor students have been asked to assist in the development by trying to use our GAMYGDALA plug-in in GOAL. Students who are also working on our Virtual Humans for Serious Gaming Contextproject have been selected, as well as students who work on other Contextproject subjects. This way, opinions of students could be collected who already know about the product and how it should work, as well as opinions from students that have not used our product before.

5.2 Test method

User interaction is all about users actually using our product and taking opinions on usability and user-friendliness. Because code writing is fairly hard to measure in a meaningful way, the Thinking-aloud method (Nielsen, 2012) has been used to evaluate user interaction. Testers have been asked to say anything that came up in their minds while using the product. Positive and negative emotions could therefore easily be distinguished. Users were encouraged to express their feelings about all aspects of the product including overall usability, but also about missing features, naming convention and minor other facets. All remarks were noted and the results are discussed in paragraph 5.3.

The test set-up consisted of a laptop with a working version of SimpleIDE and a BlocksWorld for Teams[2] instance running on a GOAL version extended with GAMYGDALA. Students of Computer Science, familiar with GOAL, were asked to evaluate a sample program and write with additional code for it afterwards. A manual describing GOAL actions, a short GAMYGDALA introduction and a list of all emotions in GAMYGDALA was provided.

5.3 Test results

This section contains the test results. Firstly, the overall reaction on the product will be reviewed. Secondly, the positive feedback about the GAMYGDALA plug-in is examined

[2] https://github.com/eishub/BW4T

and finally the user remarks regarding possible improvements are discussed.

5.3.1 Overall reaction

The overall reaction on the product was very positive. At first, however, certain aspects of GAMYGDALA such as the inner workings with goals, beliefs and likelihoods, were unclear. After clarification, most of the confusion was resolved, but from this can be concluded that a more encompassing introduction to GAMYGDALA is desirable. Users were immediately interested in GAMYGDALA after being fully instructed about the possibilities. They already started to imagine what it could mean for computer games if GAMYGDALA was used in it. Because of their interest, it was very easy to discuss certain topics of the product with the users. They gave very clear and targeted feedback. Most of it was positive, as discussed in section 5.3.2, but some feedback was about possible improvements, which is explained in section 5.3.3. The overall conclusion is that GAMYGDALA is very promising, but a user would need a thorough understanding of the engine to really get the most out of it.

5.3.2 Positive feedback

The positive feedback was generally based on the purpose of GAMYGDALA and the possibilities it opens for the (serious) gaming industry. Relations and emotions are fundamental elements of life which computers are not capable of simulating, and GAMYGDALA was generally seen as a step in the right direction by the users. Some people directly mentioned the positive addition of a more human touch of non-player characters in-game, would emotions be added.

Additional positive feedback included the following.

- The GAMYGDALA manual is clear for the users. There were few questions about the purpose and content.
- The percepts about emotions returned by GAMYGDALA are clear and concise.
- The logic behind GAMYGDALA, the emotion generation, is understandable. The emotions resulting from particular actions are as expected in that situation.
- No visible delay was observed while using GAMYGDALA, indicating all necessary calculations are performed quickly.

5.3.3 Possible improvements

With the feedback from the users came a few points for improvement.

Firstly, the naming of actions and variables was unclear. For example, users did not intuitively understand the meaning of the action “ga-appraise”. “Appraise” might be the wrong choice of words, as for example “ga-event” or “ga-value” are more meaningful action names. Furthermore, the “isMaintenanceGoal” variable, which determines whether or not an achieved goal is to be removed from the goal base, was hard to understand by users.

Being the project team which has worked for weeks with the code, the meaning of above action and variable is clear. For new users, however, these names are not

descriptive at all. Whether or not this is to be changed in a future version is up for debate, because the working of all actions, methods and variables can be looked up in the manual. It is all the same a good idea to evaluate the naming of unintuitive actions, methods and variables.

Another point of confusion among users was the *likelihood* of goals and beliefs. When appraising an event, the belief as well as the affected goals have a *likelihood*, but each with a different meaning. The difference of these is not clear from the manual or the variable names. Explaining the difference proved to be rather difficult, so a clear explanation in the manual is required.

Finally, some points of improvement about the working of GAMYGDALA itself were mentioned.

- The sensitivity of agents for certain emotions is currently not adjustable. In real life, some people are more affected by positive emotions than by negative emotions, and vice versa.
- The intensity of a relation between two agents in a GAMYGDALA environment is fixed, and does not change based on positive of negative actions by the causal agent of the relation. If another agent consistently undermines your own goals, the intensity of the relation should decrease.

6 Evaluation

In this section we describe 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. In addition, 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.

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 chose this method to make sure that every task 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. We do not have users that extensively use the product. There can still be some bugs that we have not found that will show up while using the product. Bug-fixing is an ongoing process.

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. An example of a part that can be further improved is the way we keep track of relations. It now initializes a relation within a certain agent, but it would be better to make a central place in which the relations are defined.

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

The GOAL plug-in works like it should and it is tested. Like with the GAMYGDALA port, there might be some bugs left that can only be discovered during extensive usage. There is an improvement that can be made. Emotions that come from relations currently do not show the person they are aimed at, this can be added to 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

GitHub. (2015, June). *The github website.* Retrieved from https://github.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

JUnit. (2014, December 4). *The junit website.* Retrieved from http://junit.org

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

Oracle. (2015, June). *The project nashorn page.* Retrieved from http://openjdk.java.net/projects/nashorn

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

Rising, L., & Janoff, N. S. (2000, August). *The scrum software development process for small teams.* Retrieved from http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/scrum/s4026.pdf

SIG. (2015, June). *The sig website.* Retrieved from https://www.sig.eu/nl/

TravisCI. (2015, June). *The travisci website.* Retrieved from https://travis-ci.org

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

A Appendix A: Plug-in Manual

A.1 Introduction

This is the manual for the GAMYGDALA extension of the GOAL programming language. Before reading this manual, you will have to understand how the GOAL language itself works. If you still need to read up on this, it is suggested that you read the GOAL programming manual and try out some of the exercises.

To offer you a full understanding of how our plugin in works, you will find in this document an overview of how GAMYGDALA works, definition of all actions that can be used, how to use them, and the results that are yielded when using the plugin.

A.2 Gamygdala overview

In GAMYGDALA, an agent gains emotions by adopting goals and appraising beliefs that influence the likelihoods of these goals. For example, if an agent has a goal to become rich, and he signs a large deal, its likelihood to become rich will become higher, and the agent will become hopeful. As in the real world, the intensity of these emotions decays over time.

Agents may also have relations; they can like or hate each other to a certain extent. These relations will inflict emotions such as happy-for or resentment when something happens to the agents that they feel liking or hate towards. For example, if Mario kills a Goomba, Peach may start feeling happy-for and bowser may start feeling resentment.

A.3 Actions

The GAMYGDALA plugin for goal is based on new actions. These actions are very similar to the environment actions that are generally defined in the actionspec. No actionspec is needed for this plugin, though. This section covers the actions that can be used. The next section will cover the results that these actions yield.

A.3.1 ga-adopt

ga-adopt(term *goal*, double *likelihood*, boolean *isMaintenanceGoal*)

Adopts a goal in the GAMYGDALA engine.

goal	The goal that is being adopted.
likelihood	The likelihood of the goal being achieved, from 0 to 1.
isMaintenanceGoal	Whether or not the goal can be achieved multiple times during the session.

A.3.2 ga-drop

ga-drop(term *goal*)

Drops a goal in the GAMYGDALA engine.

goal	The goal that is being adopted.

A.3.3 ga-create-relation

ga-create-relation(string *otherAgent*, double *relation*)

Creates a relation towards another agent. This can already be called when the other agent does not 'exist' yet.

otherAgent	The name of the other agent.
relation	The intensity of the relations, from -1 to 1, where -1 is full hate, and 1 is full love.

A.3.4 ga-appraise

ga-appraise(double *likelihood*, string *agent*, list *affectedGoals*, list *goalCongruences*, boolean *isIncremental*)

Appraises a belief that influences the likelihoods of goals. This is how agents gain emotions.

likelihood	The likelihood of the belief to be true.
agent	The agent that caused the belief.
affectedGoals	The goals that were affected.
goalCongruences	How the goals were affected respectively, from -1 to 1, -1 for very negatively, 1 for very positively.
likelihood	The likelihood of the belief to be true.
isIncremental	Whether this belief enforces the likelihood of these goals (*true*), or defines the absolute likelihood (*false*).

A.3.5 ga-decay

ga-drop(any *any*)

Decays all emotions and updates them in the belief base.

any	Any parameter. This does not have any influence and will be removed in the future.

A.4 Results

When the appraise and decay actions are called, the emotions for the agent are fetched from GAMYGDALA, and they are stored in the belief base, in the following form:

```
emotion(EmotionName,Intensity)
```

Basic emotions can be the following:

Hope	Fear
Satisfaction	Fear-Confirmed
Joy	Distress
Disappointment	Relief

And relational emotions can be the following:

Happy-for	Resentment
Pity	Gloating
Gratitude	Anger
Gratification	Remorse

The intensity ranges from 0 to 1.

B Scalability GAMYGDALA

The developers of GAMYGDALA choose to not implement some of the Software Engineering Method, because most of the time it will impact the scalability of the code. But the port of GAMYGDALA has been refactored and some Software Engineering Methods has been applied. So the GAMYGDALA port had to be tested if the refactor has impact on the scalability.

We tested GAMYGDALA by calculating the execution time of the `appraise()` and the `decayAll()`. This because those two methods are affected by the amount of entities with in engine. Because are so many entities within GAMYGDALA the test has been split into three smaller once which are different in the amount of relation between Agents, the amount of Beliefs that have to be appraised and the amount of Goals that are affected by the Beliefs.

Foreach test the duration had been calculated for 1, 10, 100, 1000 and 10000 Agents with 5, 50 and 500 Goals per Agent. Below you can see the result of the test plotted into three different graph. Below each graph you can see the settings of the test. The x and y-axis are logarithmic.

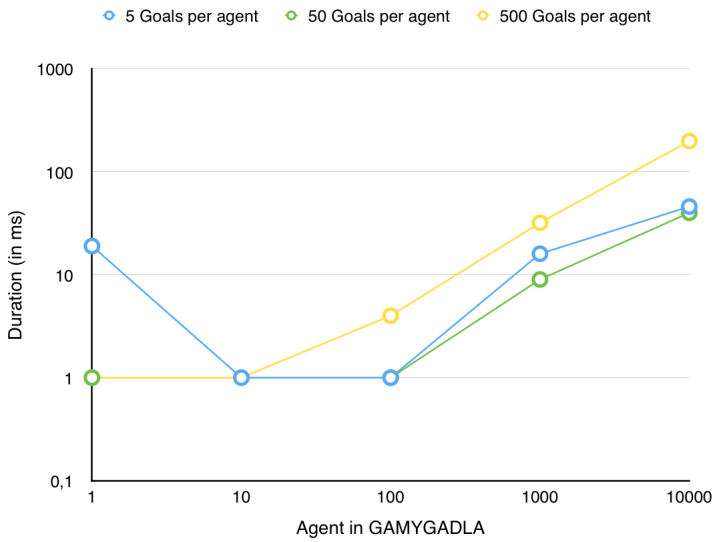

Figure 1: Scalability, Relation: 1, Belief: 1 and Affected Goals: 1

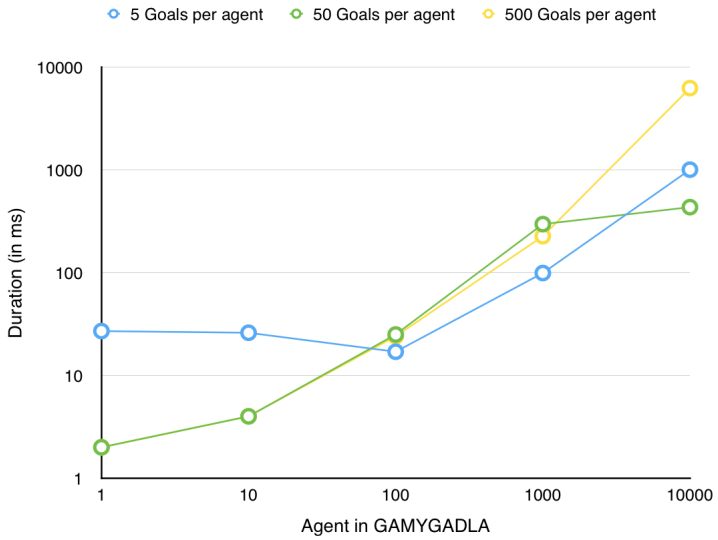

Figure 2: Scalability, Relation: 10, Belief: 10 and Affected Goals: 10

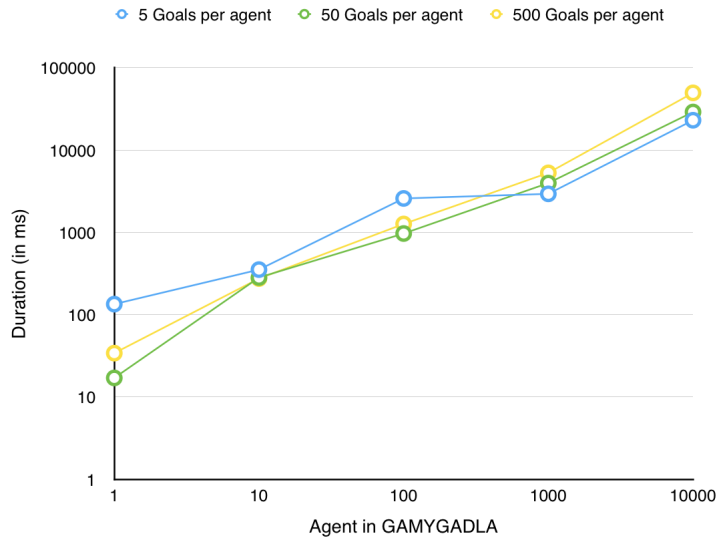

Figure 3: Scalability, Relation: 100, Belief: 100 and Affected Goals: 100

From these graphs you can see that their is a linear relation between the amount of Agents and the execution time. From this you can conclude that the port of GAMYGDALA is scalable. At least till 10000 Agents, 500 Goals per Agent, 100 Relations, 100 Beliefs and 100 Affected Goals. We think that this is enough, because 1000 Agents with 500 Goals is a really really big system.