Final Report

GROUP: Gamygdala-Integration:
B.L.L. Kreynen
M. Spanoghe
R.A.N. Starre
Yannick Verhoog
J.H. Wooning

June 18, 2015

Contents

1 Abstract

2 Introduction

In this section you can find a brief introduction to the context problem and to the solutions applied by group 3.

2.1 Problem description

The research problem is stated by a company called Tygron. They provide local authorities with a game in which their city is fully simulated. In this game the council members can discuss difficult matters on where to build certain structures. By doing so they gain an understanding of everybody's needs and responsibilities. The main question that rises is the possibility of replacing one of those players with an Artificial Intelligence solution. Since the players of this game mostly interact in an emotional way, as real humans do, these bots should feel emotions too.

This topic is very interesting because it has many applications. Both further research and different industries can profit from this. For this context project the students work in a large group that is subdivided into different smaller groups each with specific tasks. In total there are 4 groups of 5 students that work together. Together the whole group will make a proof of concept by creating a game in the Tygron engine[4], creating an interface between the Tygron engine and GOAL and creating both a plug-in and an integration of the Gamygdala emotional engine in GOAL. The specific task of this subgroup is to provide the group that will create an agent with an integration of Gamygdala in GOAL. It is not our task to implement the engine itself, since it is already present, but it is our job to integrate it in GOAL in a way that programmers, like the other groups, can make use of this when creating virtual humans.

In this report there is given a overview of the user requirements first. Then a list of the implemented software product is explained in detail. After that, you can find a reflection of the product and process from a software engineering perspective. This is followed by a detailed list of implemented features. A section on the Human Computer Interaction will then explain how the product and users interact. Finally a conclusion and outlook will list all the important findings and future improvements.

2.2 User requirements

In this section you can find the specific tasks the product should fulfill. By using the MoSCoW method we can subdivide the high level specifications into the following. The subsections decrease in importance or priority. This means that for example the MUST-haves are way more important to implement than the COULD-haves. This method is very handy and more information can be found in the glossary or references.

Must haves

- An agent in GOAL must have an equivalent in Gamygdala.
- The agent's goals in GOAL should also be present in Gamygdala for that agent. This means that if you define a goal for an agent in GOAL that the Gamygdala engine must also know of this. New goals must be communicated to Gamygdala.
- It must be possible to define how good or bad a belief is for certain goals.
- It must be possible to define relations between agents in GOAL.
- It must be possible to have these relations also present in Gamygdala. Again, it must be the case that when a relation is created in GOAL, the Gamygdala engine knows about this relation. Otherwise no emotions between two agents can be calculated.
- It must be possible to retrieve the emotional state of an agent in the same way as his beliefs.
- It must be possible to setup an initial emotion of an agent.

Should haves

- It should be possible to set the gain to a specific value.
- It should be possible to set the decay to a specific function.
- The Gamygdala goals and its properties should be possible to change. This also includes the relations regarding beliefs and goals.
- It should be possible to display the emotional states in the Simple IDE[5] provided with GOAL for debugging.

Could haves

- It could be possible to set a custom decay function.
- It could be possible to change relations during runtime.
- It could be possible for an agent to reason about other agents emotional bases.

Will not haves

- Eclipse plug-in

3 Reflection on the product and process from a software engineering perspective

In this section we will reflect in on our product and our process from a software engineering perspective. We will describe some of the problems we encountered and what we learned from them as well as how things could have been improved. We will start with a look at the product and after that we will look at the process.

3.1 Product

We started off with an existing code base (GOAL) on which we had to build. Since the goal was to make working with emotions in the GOAL environment intuitive and similar to working with the existing GOAL language we tried to keep true to the existing architecture. There were four main repositories in which we had to make changes, we will start with the Grammar repository. The Grammar repository is where we had to add code to give goal programmers the option to use emotions without knowing what exactly is happening while also giving options for programmers to tinker with the emotion settings that are available in the Gamygdala engine. To facilitate this we made a file with configurations. Since we found that it was difficult to use without any knowledge about the Gamydgala engine we made a document to help configuring this file.

In the Runtime repository the emotions had to be handled during runtime. This was done by connecting the adding and dropping of goals to the Gamygdala engine and by making sure that the emotion updates from the Gamydala engine were handled and added to a base similar to how percepts are updated every cycle. We implemented this by adding calls in the existing code to functions that we made to handle the emotions. For the structure of our functions we looked at how the existing code handled similar situations. After some initial time to get to know the existing code base we were able to implement the functionality we wanted in this repository relatively easily. However, when we tried to add to add an EmotionBase rather than use the PerceptBase to process the emotion we learned that this was a lot harder to do.

In the simpleIDE we wanted to be able to show the emotion from the EmotionBase similar to how percepts, messages, and beliefs are shown in their own tab. While we were able to do this the problem was not to make emotions visible in the EmotionBase but rather to get these changes functional in prolog and goal files. Which brings us to the changes in the Mentalstate repository. To create an EmotionBase we had to make changes to the Mentalstate repository. Since the functionality of the EmotionBase was going to be very similar to that of the PerceptBase the code that we added in this repository was very similar to the already existing code. This repository is also where we had to make sure that the emotion information was inserted into the prolog knowledge base, and while this seemed to be simple enough this is where we ran into troubles with the EmotionBase.

The difficulties with the addition of the EmotionBase mostly seemed to

stem from issues with the dependencies between the repositories, and since the existing code base was pretty large this made it difficult to comprehend why it was not working as we intended. This showed that when writing software it is critical to maintain good testing coverage and integration testing, which the existing code lacked. We also learned that when you want to implement new functionality into an existing code base with some hurry, it is best to focus solely on functionality rather than trying to implement it in a "nice" way. Especially if you aren't sure the code will still work with the "nicer" implementation.

3.2 Process

We made use of an incremental and iterative software development process. Every week we made a scrum plan and at the end of the week we reflected on this plan and made a new one. Initially the tasks we made were not very fitting for scrum, they were somewhat too generic and did not lead to something we could demo. We were able to use the feedback given by the TA and the problems we encountered in our weekly scrums to improve our scrum process over the course of the project.

We used git on the social platform Github in combination with TravisCI and Maven for our code versioning and continuous integration. The code bases we used were already on github and after cloning them to our own repositories we later had some troubles with the POM-files which were still referring to the original repositories. This problem came back several times, and we should probably have fixed this initially so any new issues would have been smaller and therefor easier to solve.

Another problem we encountered here was that the original code already failed when doing integration tests, this had to do with the tests trying to open windows which was not possible and to solve this we had to ignore a few tests. Regarding testing, the original code lacked good test coverage and integration testing. Due to the difficulty of integration testing the whole code base including our new code we decided to focus our testing efforts on making unit tests involving the code we added to the existing code. However, the lack of integration tests available often make it hard to find where errors originated.

The original code also had a ton of Checkstyle errors and a lot of errors from the Maven Javadoc tool. We did not use Checkstyle because the errors were legion and if we used the Maven Javadoc tool the code would not compile.

This was also the first project where we made use of a pull-based development model. Initially we made some mistakes by pulling changes in the master onto our own base instead of rebasing. We also learned to squash a lot of small changes into fewer more substantial commits to make it easier to track the changes that were made.

4 Outlook

This section gives a recommendation as to what possible improvements there still are for future expansion of the project.

4.1 Emotion configuration parser

First off we define the properties of the emotion configuration in a regular text file that is parsed by a simple parser. It's not a bad idea to improve upon this and to create a new file with the ANTLR framework that is used for the other files in GOAL as well. Furthermore while our parser does throw errors which mention which line numbers are still incorrect it would be nice if this could be statically checked and displayed in the SimpleIDE or the GOAL plug-in for eclipse.

4.2 Emotion configuration

Secondly there are still a few things that could be added to the emotion configuration. It is possible to define common goals and individual goals and it is possible to define that the individual goals only apply to certain agents. However it is not possible to define a notion of teams for the common goals, if a common goal is defined and two agents adopt this goal then it is assumed that they are working together on this goal in some instances it might be useful to have an optional parameter that allows you to define multiple teams for these common goals. However before adding something like this it should also be considered whether this does not complicate the emotion configuration too much for a feature that might not be all that widely used.

4.3 Aspects of GOAL that influence emotions

There are still some aspects in GOAL that are not being taken into account for the evaluation of emotions but which could potentially be interesting. For example bots can send messages to each other and bots can also try to reason about the goals and beliefs that other agents are holding. These parts of GOAL have no effect on emotions in the current implementation. An example of how this could affect emotions is that needing to drop a GOAL because of a message given by another agents is less bad than having to drop a GOAL because of your own observations (the idea being that you were notified beforehand and had to waste less time trying to achieve this goal before realizing you couldn't complete it anymore). While we're not sure how this should affect emotions exactly it would be interesting to take a look at what could be done in these areas. Although, again, adding definitions for these things might make the emotion configuration overly complicated for a feature that might not have that big of an effect, this should be considered when thinking of these features. Either a smart way of setting a standard for these messages (so that programmers just have to send the correct message and not worry about any other configuration)

or a very easy way of defining them in the emotion configuration should be figured out.

4.4 Causal agents

At the moment determining the causal agent of dropping or achieving a goal is fairly simplistic, for individual goals the causal agent is always the agent itself and for common goals it is the agents that first achieved that goal. This does not always reflect the real world and it might be interesting to see what can be done to improve this. Again just like with the other sections it should be carefully thought out so that it does not complicate the use of gamygdala within GOAL too much. For this an potentially interesting solution would be to define a new drop and insert predicate that not only takes the goal/belief to drop/insert as input but that also allows the programmer to enter which agent caused this drop/insert.

4.5 Code quality

Finally, in terms of code quality there can always be improvements of course. Some parts of the code would benefit from being re-factored a bit. Most of the code would also benefit from better integration testing, but this is not only a problem in our own code but also in the code of GOAL. In terms of unit testing the code written by our group scores pretty high but throughout the project we noticed a few times that some changes created serious issue in GOAL but none of our tests notified us of this. This was caused by a shortage of integration testing, all the individual components still seemed to work perfectly fine but when combined in certain scenarios they failed and these scenarios were not always tested. As mentioned, this would also be a recommendation to GOAL itself, at one time for example a modification to updating the goal base caused one of our two agents to not perform any actions anymore at all, this was not caught by integration nor unit tests of GOAL.

Glossary

agent A virtual identity that uses logical rules to derive the wanted actions and can percept events from the environment. It is AI.. 4

Gamygdala An emotional engine created to simulate emotions for virtual identities[2].. 3

GOAL A programming language developed at TU Delft for creating Virtual Humans[1].. 3

MoSCoW A method used to fill in requirements for a project based on difference in priority regarding the final goal of the project.[3]. 3

References

[1] GOAL programming language `http://ii.tudelft.nl/trac/goal`

[2] Gamygdala emotion engine `http://ii.tudelft.nl/~joostb/gamygdala/index.html`

[3] MoScoW method `http://en.wikipedia.org/wiki/MoSCoW_method`

[4] Tygron engine `http://www.tygron.com`

[5] Simple IDE `https://github.com/goalhub/simpleIDE`