Final Report

GROUP: GAMYGDALA-Integration:
B.L.L. Kreynen, bkreynen, 4331842
M. Spanoghe, mspanoghe, 4331834
R.A.N. Starre, rstarre, 4334620
Y.L. Verhoog, ylverhoog, 4155335
J.H. Wooning, jwooning, 4245318

June 24, 2015

Contents

1 Abstract

2 Introduction

In this context project four groups consisting of twenty students in total have developed a set of virtual human agents for the Tygron City Planner serious game. To accomplish this effort, tasks have been divided between these groups. One group developed the link between the GOAL[1] environment and the Tygron game, a group worked on native integration of GAMYGDALA[2] in GOAL, another developed a plug-in version of GAMYGDALA that can be used in GOAL and the fourth group developed the agent that uses GAMYGDALA to make emotional decisions and uses the link to interact with the Tygron game. We are the group responsible for GAMYGDALA's integration into GOAL.

2.1 Problem description

The research problem is stated by Tygron[4], a software company located in the Netherlands. They provide local authorities with a game in which their city is simulated. While playing this game government officials and private parties can more easily visualize and therefore better discuss urban planning projects. By playing the game they gain an understanding of different stakeholders needs and responsibilities. Tygron requested us to look into the possibility of replacing a number of human players with an Artificial Intelligence solution. Since the players of this game mostly are influenced in their interactions by emotion, the AI should feel and react to emotions as well.

The input of a real company means that to us, this project was quite exciting. That our code may one day be used by anyone other than ourselves is a better prospect for reuse than we are used to from other projects. The specific task of this subgroup is to provide the group that is responsible for the agents with an integration of GAMYGDALA in GOAL. It is not our task to implement new functionality, but rather make GAMYGDALAs functionality available in GOAL. With this added functionality programmers such as the other groups can, when creating virtual humans, let GAMYGDALA give their agents emotions and can base decisions off of those.

In this report we first give an overview of the implemented software product. Consequently, a reflection on 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 analysis will then explain how users interact with the product. Finally an outlook will list some important findings and potential future improvements.

2.2 User requirements

The user requirements are fully listed in the product planning [6]. These features are an overview of the requirements this project should fulfill. Please refer to the product planning for a complete list of the user requirements in the MoSCoW-form.[3]

The main goal of the project is giving a programmer in GOAL the ability to provide the agents he is creating with emotions. This means that the agents

have to express emotions, but can also act based on these emotions.In a few rules, a developer can define which goals cause an agent positive and negative emotions, and how strong these should be. This is achieved through editing a configuration file which is provided with documentation. To make things easier, a developers should be able to see in real-time which emotions are present for a certain agent in the development environment. With that functionality, the developer can debug the emotions and his agents response to those and see how settings impact emotions. Furthermore the programmer can query the emotions in GOAL so that he can code responses in logical rules in his agent.

All the features and settings need to be documented very thoroughly so that programmers can learn to use the basics of the GAMYGDALA integration in a few minutes. No tweaking of complicated settings is demanded of a novice user, but they are present for the advanced emotion-developer.

3 Overview of the developed and implemented software product

This section contains short descriptions of each of the parts of GOAL that we modified or extended. In short, we create a GAMYGDALA instance in the GOAL runtime, read settings from a configuration file for emotions, render input for GAMYGDALA based on these settings and the real-time input regarding goals from the GOAL environment, and insert GAMYGDALA's output into the agents database.

3.1 Emotion configuration

The emotion configuration is the most extensive part of our implementation. It is a file which holds all the GAMYGDALA settings for a GOAL program. Its content is examined in more detail in the emotionconfig documentation document.

3.2 Parser

To be able to use the EmotionConfig in GOAL we had to write a parser for the EmotionConfig. The parsing for the goal and mas2g files in GOAL is done by using the parsing tool ANTLR. However, after we made up a syntax for the emotion configuration we thought it would be overly complicated to use ANTLR for this parser.
Instead we used a simple implementation where the parser iterates over the lines and reads what settings are configured on each line, comma-seperated, e.g. one line would read ẄHITELIST, 1.̈ While parsing the EmotionConfig instance is updated according to the configured settings. For the goals, subgoals and relations a new object is created to store in the EmotionConfig.

3.3 Mentalstate

We had to implement changes to the mentalstate to be able to create a emotionbase for an agent. With this emotionbase the programmer using goal can easily see what emotions an agent currently has. Prolog can only reason about one database at a time, so beforeit starts a reasoning cycle, all beliefbases for an agent are merged, including the base for its emotions. At the time of writing this report, the emotionbase was functional, but the information in it was not inserted correctly into the belief base before prolog queries are executed.

3.4 SimpleIDE

We also changed the simpleIDE to make it able to edit the emotion configuration file. This includes changes in the filepanel to let the emo2g files show as a child of the mas2g file, parsing errors and a tab to be able to debug the emotion base. We unfortunately did not have the time to add syntax highlighting for the emotion configuration file or the emotionFile block in the mas2g file.

4 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.

4.1 Product

We started off with an existing code base (GOAL[1]) on which we had to build. Since the goal was integrate GAMYGDALAs emotions in the GOAL environment and keep working in this version of GOAL 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.

Grammar

The Grammar repository is where we had to add code to give goal programmers the option to give an agent emotions without editing any settings 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 wrote an EmotionConfig document to help developers find their way through this file. We have included this document in the appendices.

Runtime

In the Runtime repository the emotions had to be handled during runtime. This was done by connecting the adding and dropping of goals in GOAL 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 a seperate EmotionBase for emotions rather than inserting them into the PerceptBase we ran into some trouble at the mentalstate level.

MentalState

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 the complexity

of the classes managing the interface between GOAL and prolog. Because the existing code base was large and marginally tested and comments did not explain every quirk or error this made it difficult to comprehend why our addition 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 and following conventions already put into practice in the existing code rather than trying to implement by the book. An obvious downside of this method is that we had no test structure to fall back on when for example the emotionbase did not function as we expected. We believe this trade-off for functionality was worth it, as we do now have all GAMYGDALAs functionality implemented with not a lot of time to spare.

SimpleIDE

In the simpleIDE we wanted to be able to show the emotions from the EmotionBase similar to how percepts, messages, and beliefs are shown in their own tab. We were able to add the emotion tab, where facts inserted into the emotionbase are displayed, but these facts are not visible to an agent at prolog level, so we cannot show this function in our release version, where emotion is added to the percepts.

4.2 Process

We made use of SCRUM, 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 was caused by the tests trying to open a gui which was not possible on TravisCI. 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 and which functions to unit test. Unit tests took significant time to write because there were usually no tests already included in GOAL for similar functions.

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 GOAL repositories do not include a checkstyle template and do not seem to follow many style conventions that checkstyle checks for. If we followed checkstyle conventions only for the functions we added that would not be a significant change in checkstyle compliance and fixing entire classes (often hundreds of lines) would take a lot of time and would not contribute much to the project. We concluded that formatting our code to checkstyle would not be worthwile.
When we enabled the Maven Javadoc tool included in most GOAL repositories as a dependency the code would not compile because of the warnings it generates. We disabled javadoc checking for most projects.

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.

5 Interaction Design

This section will describe how we tried to validate the interaction of our users with our product. First there will be a description of the method used to gain feedback then the results from this will be discussed after which there will be an explanation of how we processed this feedback. Finally there will be a short discussion on what could be done to gain more valuable feedback.

5.1 Used Method

The way we gained feedback from users on our product was to ask the project group that is assigned to code the virtual human in GOAL to complete a tutorial we wrote, which is included as Appendix B. This is a perfect group to do this with since they will have to work with our program to complete their own, they have experience with GOAL and know a little bit about GAMYGDALA. They are a perfect example of our typical end user. They were asked to think out loud so that it was possible to identify any issues they were having. The recording of this session is attached to this report.

Sadly enough only three people of the other group were available, they worked on the tutorial in one team. This felt like the best setup because thinking out loud feels more natural when there are other people to talk to.

5.2 Results

From the feedback gained it became obvious that some of our documentation was not yet clear enough. There were three identifiable issues with the documentation:

- Sometimes details about the behaviour of an option were missing.
- Sometimes information was not easily found. For example, details users were looking for were mentioned in a global entry in the documentation but was not repeated in the entry for a specific option.
- Some information in the guide was outdated. The configuration options had been changed after the documentation was written.

Furthermore we identified one bug in the program when the users were working on the tutorial.

5.3 Reaction

The details that were still missing in the documentation have been added and we have restructured the document to ensure all information about an aspect of the config is included in one section. The outdated information in the document was updated. The bug that was identified has since been fixed.

5.4 Commentary

The feedback we were given was somewhat superficial, it was certainly useful feedback, but sadly the other teams did not have enough time to use the program for an extended period of time and then have an interview with us. Pretty much all of the received feedback was feedback on the documentation, we feel that if they would have spent more time using our program that we would have received more valuable feedback on missing features and the usability of our implemented features. However, we understand they could not, given the time constraints imposed by project deadlines. The agent group still had work to do on implementing their strategy. We had initially negotiated to have this session a few days earlier while they still had more time.

Only having one run with three people also renders the feedback more limited. We had hoped to have a session with all five members of the other group. We should have scheduled another session with other users of the GOAL environment to gain more diverse feedback.

6 Evaluation and failure analysis

This section is split up in two main subsections. The evaluation of the functional modules and the failure analysis. The evaluation is about how our functionalities were evaluated and in the failure analysis section a list of failures during the process is given.

6.1 Evaluation

To evaluate the functional modules this group used the tutorial and the HCI meeting with the other group. In this evaluation is was clear that the main functionalities that were needed were present and working. The other group successfully finished the tutorial and was able to implement their emotional agent with the functionalities provided by us. However, they did not have enough time to use the implementations extensively. This means that the feedback is not very in depth and thus mainly based on small bugs and lack of documentation. The evaluation of the HCI can be found in its dedicated section.

6.2 Failure analysis

In this section the major failures that our group encountered will be explained. These involve the problems, how they arose and how they have been fixed or why they were not easily fixed. As the definition of failure states, a failure occurs when the delivered service no longer complies with the specifications according to Laprie et al.[8]

Empty EmotionConfig

An empty EmotionConfig file caused some default values to be 0 instead of their intended default values. The cause of this failure was mainly because of the way the parser created an EmotionConfig object. When the parser finds no lines an

empty EmotionConfig was returned instead of creating an instance through our singleton design which would automatically set values to the correct defaults. This failure was present for a long time but during the tutorial our group found out it was there. It can be defined as a failure since it does not comply with the specifications. An empty EmotionConfig should use the default settings. It was easy to solve this failure. More unittests would have caught this.

Agent stuck

During the implementations of the achievements of goals a certain failure arose. 2 agents were ran with the same GOAL code, but for some reason one of the bots got stuck when performing actions. The problem was synchronization. The lack of integration tests from both GOAL programmers as this group caused this failure to get into the code. It was an easy problem to solve, but the fact that it was present was bad. Our group handled accordingly to this problem by making more integration tests.

EmotionBase

Our group specified the requirements that the programmer in GOAL can look at an Emotionbase, just like the beliefbase, and see the emotion present for a certain agent. This feature was harder to implement than expected. The EmotionBase is created and added to an agent, and emotions in it can be displayed in the GOAL IDE emotion tab, but adding the emotion facts to the prolog database, similar to how percepts are added, was a feature we could not complete on time. We have tried to ask for help on some occasions, but did not get a hold on the developer that supposedly knew the most about this part of GOAL. Vincent Koeman could not help us with this, and Koen Hindriks was engaged in more important matters. We feel that we should have tried harder to contact the relevant developer at an earlier stage, because we did not get an opportunity to ask for information. We have a strong suspicion that with some help it would not take more than a few minutes to identify our error.
Emotions are inserted into the percept base in our final release, so they are displayed in the IDE and inserted into the mental state of the agent in the syntax percept(emo(type, intensity)). Functionally, this solution is equivalent to having a seperate base, with the exception that the emotions are not displayed in their own tab in the gui.

7 Functionality Description

7.1 Default functionality

The purpose of the integration of GAMYGDALA into the GOAL runtime is to provide all of GAMYGDALAs functionality continuously in the GOAL multi-agent system. This is possible because GAMYGDALA is like GOAL in the aspect that it is goal-oriented. This implementation was chosen so that developers do not need to code for every call to GAMYGDALA, but rather every

update relevant to an agent's (specified) goals is evaluated by GAMYGDALA automatically. This is a notable distinction from the GAMYGDALA plugin developed by group four. After evaluation, an agents updated emotional state is inserted into the perceptbase, which can be queried by GOAL and is displayed in the GUI of the SimpleIDE GOAL development environment. Developers can use a large portion of the functionality of GAMYGDALA without having to write one line of extra code. A developer can write agents that reason about their emotions just as if they were another percept from an environment.

In the release version of our project, emotions are instered as percepts into the agents mental state in the following syntax: percept(emo(type, intensity)).

There are default values for all the parameters available to GAMYGDALA and many other options relevant to the function of GAMYGDALA within GOAL, which can be modified by developers in the emotion configuration file.

7.2 Emotion configuration options

In this section an overview is given of the options in the emotion configuration file. A selection of these options will be examined more closely.For more detailed descriptions of every option and their grammar, please refer to the EmotionConfig Documentation in Appendix C.

The emotion configuration file is optional. If it is not present in the same folder as the agent file, only default values will be used. If any options are not specified in the emotion configuration file, default values will be used.

It contains settings for the default parameters given to GAMYGDALA, such as default goal utility, default goal congruence and the speed of decay of emotions and a whitelist option. There are sections for the definition of goals and their individual utility which overrides the default, the definitions of subgoals, which include the unique congruence and likelihood for the subgoal, and the relations between agents.

GAMYGDALA parameters

Default Goal Utility
This value specifies how valuable achievement of a goal is for an agent. A positive value means achieving the goal is valuable and will cause positive emotions, a negative value implies achieving it will cause negative emotions. Normal setting is 1.

Default achieved congruence
This value controls the congruence of the event that an agent achieves a goal. In simple environments it is usually set to 1, so achieving a goal simply registers as a positive event in GAMYGDALA, but for more complex MASs it might prove necessary to lower the congruence setting for subgoals that contribute to a larger goal relative to their importance.

Default dropped congruence
The congruence of the event that a goal is dropped. Very similar to the achieved

congruence, but with a different trigger. By default set to -1, so an agent will receive negative emotions from GAMYGDALA when a goal is dropped.

Default belief likelihood
This is set to 1 in most cases because in GOAL beliefs are true or not, for example when a goal is achieved or dropped, and there is no probability. GAMYGDALA can do calculations on beliefs with probabilities and requires a likelihood parameter with every update. This setting can be overridden for subgoals of which it is not certain whether they contribute to a larger goal.

Goal definitions

Goal types
There are two types of goals that can be specified in this section of the configuration. Common goals that are shared by all agents with that goal, and individual goals that are only pursued by a singular agent. If a goal is common, reaching it will cause the emotional states of all agents with that goal to be evaluated. If it is individual, only one agent will update and the rest will remain unaffected. For every goal, a utility can be specified that overrides the default setting. Individual goals can be specified without an agent name parameter so the individual goal is given to all agents. Goals should be named the same as they are in GOAL.

Subgoal definition
In this section of the config file the congruence of goals towards other goals can be specified, creating a tree-like structure of goals that contribute toward each other. The likelihood that the achievement of a subgoal will contribute toward the larger goal can also be changed.

Relation definition

Here the relations between agents in a MAS can be specified. A positive number equals a good relationship, zero is indifference and a negative number means animosity. When an agents emotional state is updated, all agents it has a relation with update their emotional state proportional to the change and the strength and sign of the relationship. The names of agents given must be the same as their names in GOAL.

8 Outlook

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

8.1 Emotion configuration parser

In our implementation 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.

8.2 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.

8.3 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 are 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. Possibities nclude the definition of setting a standard for these messages so that programmers can set it and forget it, or a seperate section in the config file dedicated to this.

8.4 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.

8.5 Code quality

Finally, in terms of code quality there can always be improvements of course. Some parts of our additional code would benefit from being re-factored a bit. Most of the it 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

. 4

GOAL A programming language developed at TU Delft for creating Virtual Humans[1].. 4

Appendices

A HCI

In this appendix you can find a full report on the HCI meeting and all things that happened. Also with this appendix an audio file with a length of 1h16min is included which is a full recording of the tutorial. Here are the notes our group took during the introduction of our implementations.

"From the start it seems that importing the project is an uncertain problem. For 2 of the 3 members of the group is it not clear what to do with it. One knows what to do and is helping. Cloning from Github directly into Eclipse seems to do weird things. Somehow the project is imported in that way but the folders are not known as source folders thus running Java code is not working. Now they are trying to clone to the harddisk and then import it into Eclipse. This seems to work. Now they mention that converting it to a maven project is not an option when you right click. It is under the tab configuration which is not mentioned clearly in the guide it seems. Now they tell the same thing for maven install. It is under the tab Run As which is somehow unclear for the members of the group. One of them is still in the goal perspective and this is bugging Eclipse. Bernd is helping them now and telling they should better be in the Java perspective instead of the GOAL perspective.

They have come to the point of running an agent with an empty emotion-Config configuration file. They are running the agent and they look at the current feelings of the agent and say that nothing is present. No emotions are given. They take a look at the mas2g file but it seems to be set correctly. They think it is a bug. It is a bug indeed. We tell them that we will fix this because it should give emotions when an empty file is set. Now they are trying out the whitelist option our group implemented. The information is not that clear. They are arguing what the intention is and what it does. They are not reading the documentation very thoroughly. They have gotten it working and now they ask us how you can make actions alter emotions. They can do this by adding goals to do the actions and then when the action is performed a goal is achieved and an emotion can be given. They are happy with this answer and think it is a good solution. They wonder why emotions like distress are coming. They did not read through all the documentation saying that dropping goals also cast emotions. They are getting a version working and they are excited that their agents get emotions. They wonder now what the isIncremental does. They can find this in the GAMYGDALA paper written by Propescu et al.[9] but it is a bit

unclear that the last parameter(boolean) controls this setting. We will change this.

Now they have a version working with subgoals. They tried to run with a goal with a congruence of 3. This cannot happen since the congruence must be between -1 and 1. It is mentioned at a certain section but not in the section of the subgoals. Now they have finished the single bot tasks and will see if they can get two bots working together.

It seems unclear they have to change the masfile to run the second agent with the same GOAL code. After a while our group helped them because this is just a little change in the commands in the mas2g files. Now they want to query an emotion and alter an action. They try to use the gam() predicate but it is not correct. They should use the emo() predicate. This is being mentioned incorrectly in the tutorial. They tell us they will try to let one agent chill out in the Dropzone when he is happy enough. They get this working after a while because first they let him perform an action but the logical rule was not high enough in the calling order so other logical rules are getting called before that one. This is something you should think about when programming in GOAL. They set the relations which works the first time they try it. They reflect on what the did and are done."

Together with these notes the full audio file can be found here: reference

Glossary

. 4

GOAL A programming language developed at TU Delft for creating Virtual Humans[1].. 4

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

[6] Product plan: GROUP 3 https://github.com/tygron-virtual-humans/Integration-documents/blob/master/Product%20plan/Product-plan.pdf

[7] SCRUM method https://nl.wikipedia.org/wiki/Scrum_(softwareontwikkelmethode)

[8] J. C. Laprie (Ed.).
Dependability: Basic Concepts and Terminology.
Springer-Verlag, Wein, New York, 1992.

[9] Propescu, Broekens, Someren GAMYGDALA: an Emotion Engine for Games JOURNAL OF IEEE TRANSACTIONS ON AFFECTIVE COMPUTING, 2014 vol. 5