TI2806 Contextproject - Product Planning

Virtual Humans - Group 4 - Port

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

Sven van Hal - 4310055

Tom Harting - 4288319

Nordin van Nes - 4270029

Sven Popping - 4289455

Wouter Posdijk - 4317149

April 28, 2015

Contents

1 Introduction

In this report you can find the product planning for the Virtual Humans for Serious Gaming Context project group 4.

In section 2, you can find a high-level product backlog in which we will give a global overview of what we will do and what our users can expect from us during this project.

Section 2 also contains a roadmap, in this roadmap you can see what we will work on during the next sprints.

In section 3 you can find our detailed product backlog. Our product backlog is divided in different subsections, these are user stories for features, technical improvements and know-how acquisition. We will use the MoSCoW method to define our product backlog. Section 3 also contains an initial release plan.

Section 4 will contain our definition of done. In this section you can find what we need to achieve to successfully finish this project.

2 Product

2.1 High-level Product Backlog

We as a group are part of the bigger context project, which consists of four groups that all work on a different part of the final product. Our part of this final product is making a GAMYGDALA plugin for the GOAL programming language. To achieve this, we first need to get a good understanding of the GOAL programming language. Then we need to find out how we can make a (simple) plugin for GOAL. We also need to have a certain understanding of how GAMYGDALA works, so we can port GAMYGDALA to Java. Then we need to bring it all together so we can create a working GAMYGDALA plugin for GOAL.

Our final plugin will be used by the other groups to make a virtual human for the Tygron engine. You can see the Product View for an elaborate description of our user and the final product.

2.2 Roadmap

In this subsection, you can find a global planning per sprint/week.

Sprint 1 The first week was used to start up the project. We followed seminars about the context project in general and about our specific context project. We also divided the groups within our project. We found out what was expected of us during this project.

Sprint 2 Sprint 2 was used to gain some understanding for the way that the GOAL programming language works 'under the hood'. We did this by implementing a simple calculator plugin for GOAL.

Sprint 3 This sprint will be used to port GAMYGDALA to Java. To do this, we first need a global understanding of how GAMYGDALA works. We then need to port GAMYGDALA, which is written in Javascript, to Java. Finally, we need to refactor the Java code so it follows the Software Engineering principals.

Sprint 4 The fourth sprint will be used to make an action/percept base integration of GAMYGDALA into the GOAL programming language. This will be similar to the Javascript interface, but in the action/percept interface of GOAL.

Sprint 5 Sprint 5 will be used to make a simple appraisal module in the GOAL programming language which is based on actions/percepts.

Sprint 6 to 9 These sprints will be used to make the full GAMYGDALA appraisal module. These sprints will also be used to work on the final project report. Furthermore, we can also use these sprints to finalize tasks that weren't finished during the previous sprints.

3 Product Backlog

In this section you can find the product backlog for our part of the project. It is made using the MoSCoW method.

3.1 User stories of features

Must have

- There needs to be a fully functional port for GAMYGDALA to Java.
- The GAMYGDALA plugin needs to work in GOAL.
- The plugin must work without fatal problems.

Should have

- The plugin should work efficient, so that the programmer won't notice the calculation time.
- The plugin should work without any errors or failures.

Could have

- There could be a plugin framework for GOAL, so that new plugins can easily be added.

Won't have

- There won't be new customizable emotions in GAMYGDALA on initialization of the plugin.
- There won't be automatic usage of the plugin. The programmer needs to use the plugin manually

3.2 User stories of technical improvements

Must have

- The code must be implemented according to the Software Engineering principals.
- The code must have a test percentage of at least 70%.
- There must be unit tests as well as integration tests.

Should have

- There should be proper Javadoc for the code.
- The code should pass the checkstyle demands.

Could have

- There could be a higher overall test percentage.

Won't have

- There won't be an improvement over the standard GOAL framework.
- The functionality of GAMYGDALA will not be changed.

3.3 User stories of know-how acquisition

Must have

- There must be a proper understanding of the GOAL code.
- There must be a proper understanding of the GAMYGDALA code.
- We must obtain and use the proper SCRUM skills.
- We must obtain and finish the informationskills section.
- We must obtain knowledge about interactive design.

Should have

- We should have a global understanding of the other parts of this project.
- We should have an understanding of the GOAL programming language.
- We should have a good understanding of Git/GitHub.

Could have

- We could obtain knowledge about EIS.
- We could obtain general knowledge about emotions in AI.

Won't have

- We won't obtain a severe understanding of the Tygron engine.
- We won't obtain knowledge about other programming languages then the ones used in the plugin.

3.4 Initial release plan

Our project is divided in three separate parts. We started by finding out what was expected from us and how we could start at working on the final product (getting an understanding of the GOAL programming language code). Then we ported GAMYGDALA to Java. The final and biggest part of this project is designing and implementing the final plugin using the previously acquired knowledge.

4 Definition of Done

For out project to be done we need to have a fully functional GAMYGDALA plugin in GOAL. The plugin should not have any errors or failures that prevent the user from using the plugin.

The code needs to have a test percentage of at least 70%. There need to be unit tests as well as integration test for this project to be done. The code also needs to follow the software engineering principals.