Final Report Draft

Contributors:

Dereck Bridié
Harm Griffioen
Joe Harrison
Natasa Radenovic
Paul van der Knaap

Table of contents

1.1 Introduction

The goal of this year's context-project is to develop a software product for users in a non-ICT context. Our project consists of the collaboration of four groups in creating a realistic Virtual Human for Serious Gaming for the company Tygron. Tygron is a company that aids in the collaboration of multiple stakeholders in city-planning tasks.

The project's group structure is a follows: group 1 will build the connection between the Tygron API and GOAL, group 2 will build a virtual human in GOAL, group 3 and group 4 will handle the emotion simulation.

This document contains information on the development, implementation and validation of the software product. The document also contains a section on the HCI tests that have been executed with respects to the user interaction with the developed solution.

1.2 Problem description

Tygron is normally played by a group of human stakeholders. These stakeholders usually have busy schedules and it is hard for them to meet at the same time. A common problem is that not all stakeholders are present while playing the Tygron game. This causes massive disruptions in the gameplay as e.g. the stakeholder that grants permit is not present. A solution would be to create an agent that can replace this particular stakeholder so that the game can still be played

The final product will be a virtual human in the serious game Tygron that can fulfill the role of a stakeholder: the virtual human. The main focus of our group will lie on developing a connection between the Tygron API and GOAL.

1.3 End-user requirements

For this context-project the main end-user is the Virtual Human group. The Virtual Human group will use our connector between GOAL and the Tygron environment to perform actions in GOAL and percept the Tygron environment for data acquisition. The Virtual Human group has provided us with a list of all GOAL actions they need to create a virtual human.

Another important requirement is that the connector needs to work fast. GOAL is already slow and the user would not want to the connector to make the entire system even slower.

There might be other users than the Virtual Human group in the future that want to build their own virtual human. These users will most probably need other actions and percepts than we've provided with this project. For this reason the connector needs to be easy to expand in functionality. The users need to be able to understand the connector and work with it and even add functionality themselves. In order to achieve this, the documentation needs to be very clear. In order to test this we will perform a test as documented in section 5.1.

2.1 Overview developed and implemented software product

The connector consists of two main components: the connection between EIS and the Tygron API and the connection between EIS and GOAL. EIS is the environment interface standard that facilitates the connection between agent-platforms and environments. EIS was ideal to use in our case because a connection between GOAL, an agent-platform, and the Tygron API, an environment, was exactly what we needed.

The connection between EIS and Tygron is used to extract data from the Tygron environment. In order to achieve this we first set up a connection on launch to the Tygron API. After establishing this connection we use a thread that request all available data every 2.5 seconds. The Tygron environment is prone to continuous changes because there can be multiple human players of this serious game that fulfill their role as stakeholder and thus change the environment by e.g. granting permits. We deemed that 2.5 seconds is fast enough because the Tygron game is of slow pace. After the extraction of data an agent can use this data to change its belief base.

The connection between EIS and GOAL is used to let the agent perform an action and subsequently change the environment. For this project we only let the agent perform actions on a high level of abstraction such as placing a building without the agent providing the dimensions. The reason behind this high level of abstraction is that it is difficult for a GOAL agent to do simple mathematical calculations.

3.1 Reflection on the product

The most important requirement of the software is that it's scalable. We ensured that this requirement was met from a software engineering perspective by making the Environment open to extension. It is simple to add a new action to our Environment because all actions are modular. The same goes for the percepts.

3.2 Reflection on the process

Every week we put together a sprint plan for the next week. In each SCRUM meeting, Paul, our group's manager, would reiterate the groups individual tasks and inform the group if new unforeseen task appear and delegated it to a group member. At the end of the week, our group would come together and discuss what had been done and how that went, which was noted in each Sprint Reflection.

To ensure the quality of the code we used pull requests in a separate branch when committing code to Github. At least two other members of the group would have to check the pull request's commits. In case the code is faulty a comment is placed and discussed or the pull request is retracted in its entirety. In case the code is correct the pull request is merged and the branch is deleted. We created a web-hook bot that sent a message to our private chat everytime a new pull request was created so that the commits could easily be discussed outside of a meeting.

We used both Jenkins and Travis as continuous integration tools. The tools added a indicator to the pull request that indicated whether all maven tests passed. The maven tests include: PMD, checkstyle and our own JUnit-tests. Jenkins also provided a building platform with which other groups could grab the latest version of our code without having to build it themselves.

4.1 Description developed functionalities

Tygron is played by multiple stakeholders over an internet connection between the user and the Tygron API. The virtual human must be able to connect to a gaming session so it can play. In order to do this we’ve built a class that sets up the connection.

The virtual human must also be able to interact with the API. The Tygron API is based on requests. The agent sends a request for data to the server and in return the server sends back an answer in JSON format. We’ve built a class that can both send get requests and post requests. The server’s answer is not always in the same format. Sometimes the server sends back in list format, sometimes in boolean format, and sometimes nothing at all; for these cases we’ve added a list type of every entity e.g. a building class and a building list class.

All entities that can exist on a map of a Tygron session such as buildings or parks are sent in WKT format. A standalone string in WKT format is unusable so we built a class that can convert these strings into polygon objects. In order to place buildings the system must know where a large enough space exists on the map. For this purpose the class also contains high level polygon functions e.g. a functions that returns if a polygon contains another polygon.

Communication between stakeholders is done through popups. These appear when land transactions are initiated or when buildings are placed. The popuphandler contains and parses all the pop up information. Every few seconds, new popups are gathered from the server and directly used by either sending an automatic reply or sending information to the agent. In the final product, all requests sent to the selected stakeholder are automatically accepted instead of being sent to the stakeholder. This can easily be altered to send the information to the stakeholder first and let it make the decision. The pop up handler also keeps track of the amount of open requests, requests sent by the agent that have not yet been answered, such as buy land requests or permit requests.

5.1 Interaction design evaluation

For this context project we had only one group that used our software. However other users might use the software in the future. We thought it would be interesting to evaluate the usability in an experiment. We expect from the test subject that he is familiar with programming and has some knowledge on how programming in GOAL works. Our project does not have any visual components on its own instead programmers have to use the code to build their own virtual human. It is of great importance that the user understands how to work with our software through documentation.

We devised a couple of programming tasks that our test-subjects have to perform. The programming is done in GOAL and the participants receive our documentation with all available actions and percepts. Furthermore we will provide the participants with a template in GOAL to save time. During their performance we will ask them to think out loud and we will write down any important comments. After each task is completed we note down the duration and the correctness of the performed task.

In the first task we asked the participants to simply buy some land. We started off with a simple task to familiarize the participants with everything.

In the second task we asked the participants to buy land and to build a building on that land if the land is available. This task is of moderate difficulty.

In the third task we asked the participants to check and see if the agent has sufficient funds and a permit to build. The agent in the game we had set up purposely had no funds and permits. When the participant thought out loud that it had no permit we granted him the permit. Now the participant had to figure out a way to get money. After the participant has money it will again buy land. This task is considered difficult.

For this week’s draft we’ve only been able to have one participant evaluate the product. In the upcoming week we will have the other participants evaluate the product.

	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5
Task 1	9:01				
Task 2	2:20				
Task 3					

During the beginning of task 1 where the participant was getting familiar with the documentation he mentioned that a startup guide to freshen one's GOAL knowledge would be handy helping him to perform the task faster. Also some parameter units where unclear. Such as the type of the server parameter and the surface parameter. To combat these unclarities we’ve added units to some parameters. The concept of permits seemed a tad vague to the participant so we’ve added a better explanation to the documentation.

6.1 Evaluation of functional modules

Because of the organization of this project there is no real good evaluation of the functional modules other than that the Virtual Human group is able to use each module well and the code coverage. Most modules were used in the HCI evaluation and worked very well.

The part of the software that connects the connector to the Tygron API is simple to use. Users merely have to fill out on which map they want to play and the software will do the rest. Without this function the user would have to do complex preparations to set up a session. Our goal therefore was to make the setting up of the connection as simple as possible which turned out quite well.

6.2 Evaluation of entire product

Again because of the organization of this project it is hard to evaluate the product that we have built as it is a part of a bigger project. The best well-justified method is to check whether our customer, the Virtual Human group, finds that the product performs as needed.

The Virtual Human group provided us with a list of functions that it needed to create an agent. We adhered to their desires on their list and added or changed functions on their demand. The only way to test if they also believe that this is true is by taking an interview with this group and discussing whether the product performs as needed or not. Next week we will take this interview and put the result in the report.

7.1 Outlook

A possible improvement in the future is to add all components that exist in the Tygron API so that any type of user can directly use them. This would however massively slow down the system because all components in the environment would be requested. A strategy to combat this problem would be letting the user selecting only the components his agent needs in its implementation.

Appendix A Sprintplans

Will add all sprintplans in final report

Appendix B Sprintreflections

Will add all sprintreflections in final report