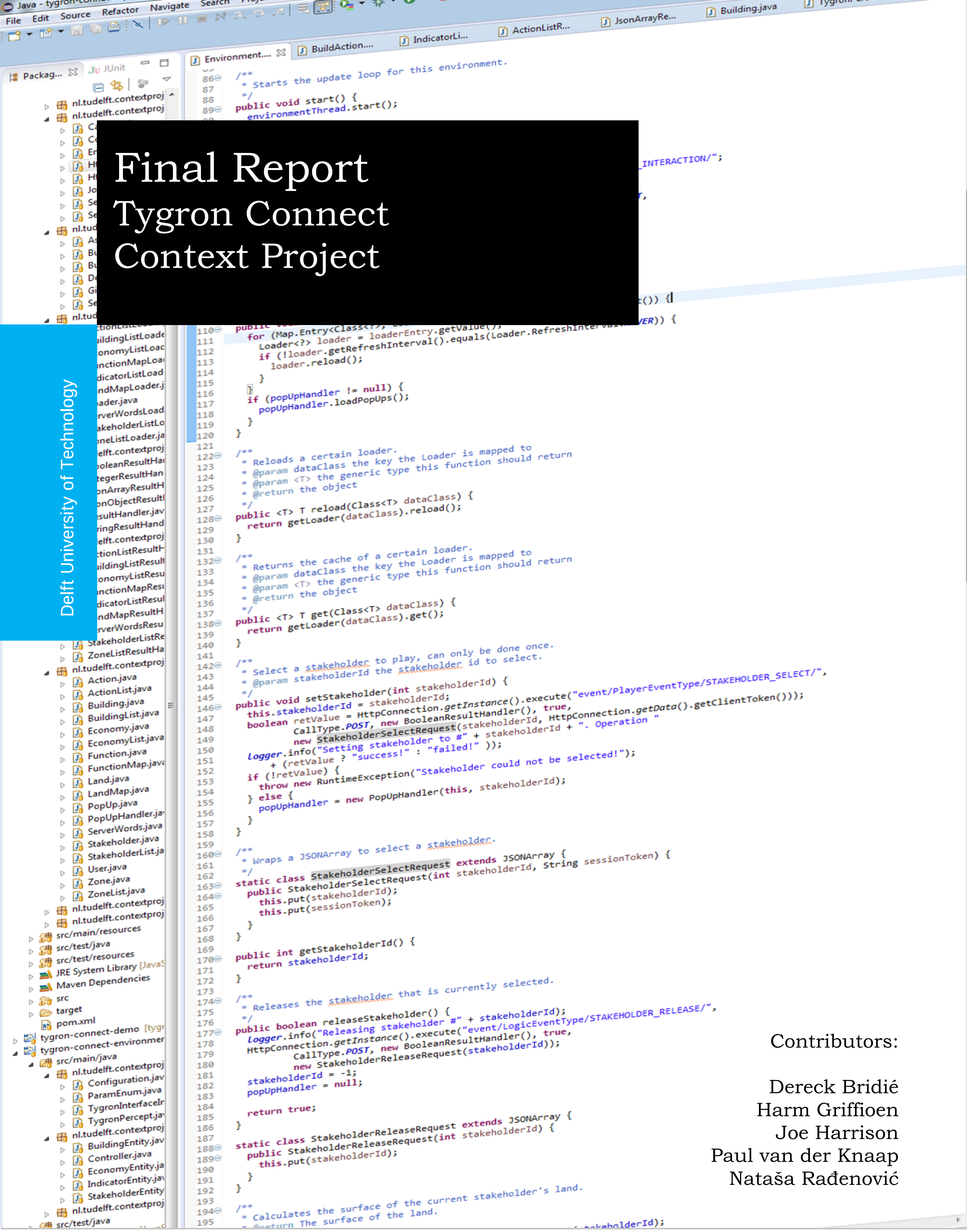


```
/**
 * Starts the update loop for this environment.
 */
public void start() {
    environmentThread.start();
```

```
_INTERACTION/";
```

```
public void ...
    for (Map.Entry<Class<?>, ...> loaderEntry ... ()) {
        Loader<?> loader = loaderEntry.getValue();
        if (!loader.getRefreshInterval().equals(Loader.RefreshInterval...ER)) {
            loader.reload();
        }
    }
    if (popUpHandler != null) {
        popUpHandler.loadPopUps();
    }
}

/**
 * Reloads a certain loader.
 * @param dataClass the key the Loader is mapped to
 * @param <T> the generic type this function should return
 * @return the object
 */
public <T> T reload(Class<T> dataClass) {
    return getLoader(dataClass).reload();
}

/**
 * Returns the cache of a certain loader.
 * @param dataClass the key the Loader is mapped to
 * @param <T> the generic type this function should return
 * @return the object
 */
public <T> T get(Class<T> dataClass) {
    return getLoader(dataClass).get();
}

/**
 * Select a stakeholder to play, can only be done once.
 * @param stakeholderId the stakeholder id to select.
 */
public void setStakeholder(int stakeholderId) {
    this.stakeholderId = stakeholderId;
    boolean retValue = HttpConnection.getInstance().execute("event/PlayerEventType/STAKEHOLDER_SELECT/",
            CallType.POST, new BooleanResultHandler(), true,
            new StakeholderSelectRequest(stakeholderId, HttpConnection.getData().getClientToken()));
    logger.info("Setting stakeholder to #" + stakeholderId + ". Operation "
            + (retValue ? "success!" : "failed!" ));
    if (!retValue) {
        throw new RuntimeException("Stakeholder could not be selected!");
    } else {
        popUpHandler = new PopUpHandler(this, stakeholderId);
    }
}

/**
 * Wraps a JSONArray to select a stakeholder.
 */
static class StakeholderSelectRequest extends JSONArray {
    public StakeholderSelectRequest(int stakeholderId, String sessionToken) {
        this.put(stakeholderId);
        this.put(sessionToken);
    }
}

public int getStakeholderId() {
    return stakeholderId;
}

/**
 * Releases the stakeholder that is currently selected.
 */
public boolean releaseStakeholder() {
    logger.info("Releasing stakeholder #" + stakeholderId);
    HttpConnection.getInstance().execute("event/LogicEventType/STAKEHOLDER_RELEASE/",
            CallType.POST, new BooleanResultHandler(), true,
            new StakeholderReleaseRequest(stakeholderId));

    stakeholderId = -1;
    popUpHandler = null;

    return true;
}

static class StakeholderReleaseRequest extends JSONArray {
    public StakeholderReleaseRequest(int stakeholderId) {
        this.put(stakeholderId);
    }
}

/**
 * Calculates the surface of the current stakeholder's land.
 * @return The surface of the land.
```

Final Report
Tygron Connect
Context Project

Delft University of Technology

Contributors:

Dereck Bridié
Harm Griffioen
Joe Harrison
Paul van der Knaap
Nataša Rađenović

Challenge the future

Final Report

Tygron Connect – Context Project

By

Dereck Bridié
Harm Griffioen
Joe Harrison
Paul van der Knaap
Nataša Rađenović

Contents

1 Introduction

In the second year of the bachelor programme of the Computer Science study at TU Delft, students are given the course TI2806 - Contextproject. In this project students will develop a usable software product for a customer who is not familiar with the inner workings of the product.

Our project consists of the collaboration of four groups in creating realistic Virtual Humans in GOAL[1] for Serious Gaming for the company Tygron. Tygron[2] is a company that aids in the collaboration of multiple stakeholders in city-planning tasks. Tygron provides an online environment in which stakeholders can easily visualize their goals through e.g. placing buildings. Tygron also provides the stakeholders with information about the location that they are discussing so that they can make informed decisions.

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 of group 1, the connector between the Tygron API and GOAL.

The document also contains a section on the Human-Computer Interaction (HCI) tests that have been executed with respects to the user interaction with the developed solution. With these tests, group 1 has made sure their software would be easy to use for other developers.

1.1. Problem Description

Tygron is normally played by a group of human stakeholders, such as the municipality, the contractor and city-planners. They usually meet in a room with computers running the Tygron setup where they discuss their plans and visualize these plans using Tygron. These stakeholders have busy schedules and it is difficult 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 when, for example, the stakeholder that grants permits is not present. This affects the stakeholders as they are experiencing loss of productivity and may also affect inhabitants who are waiting for an important building to be built in their neighbourhood.

It is important to solve this problem so that both the stakeholders and civilians spend less time waiting. A solution would be to create an agent that can replace a 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 will be able to perform all actions of the selected stakeholder. The main focus of our group is on developing a connection between the Tygron API and GOAL.

[1] http://ii.tudelft.nl/trac/goal
[2] http://www.tygron.com/

1.2. End-user requirements

For this Contextproject 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 the game and do data acquisition so they can base their actions on real-time game information. The Virtual Human group has provided a list of all GOAL requirements they need to create a virtual human e.g. a percept for the availability of land and the action to buy land.

Another important requirement is that the connector needs to work fast. The connector should not slow down the running speed of the agent significantly. For example when they want to percept something, they don’t want to wait a few seconds for a response. The impact of the connector should be brought to a minimum in all possible configurations it will be used in.

There might be users other than the Virtual Human group that want to build their own virtual human in the future. These users will most probably need other actions and percepts than those that have been provided with this project. For this reason the connector needs to be easy to extend in functionality.

The users need to be able to understand the connector and be able to 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, some tests have been performed as documented in Section 5.1.

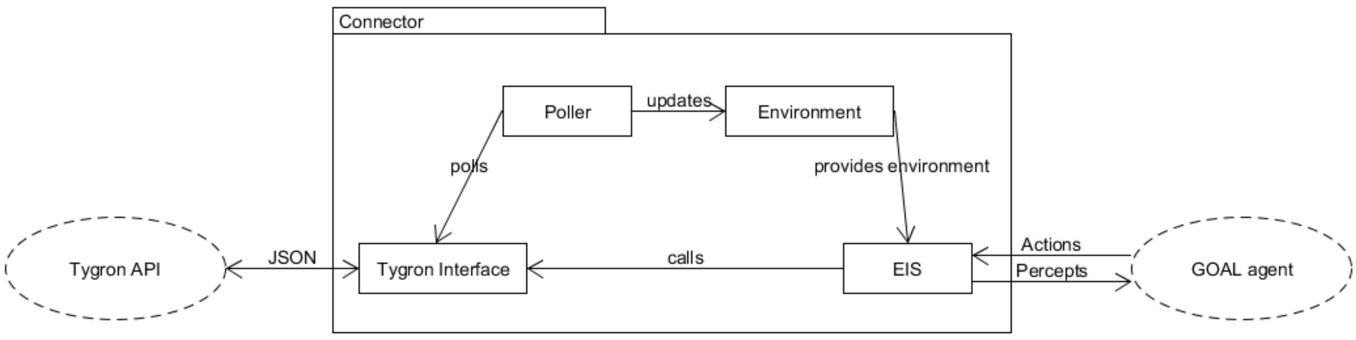
Finally, the actual produced source code needs to be of high quality. This means that it must be easy to extend and modify the product later, for example in the case that changes or improvements at Tygron are done. The software also needed to be stable. Performing certain actions should always follow the same pattern and give the same result, the end users need the reliability that when they use the product, it’ll do as expected, in any case.

2

Overview of product

2.1. Overview developed and implemented software product

The connector consists of two main components: the connection between EIS[3] 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 was needed.



The connection between EIS and Tygron is used to extract data from the Tygron environment. In order to achieve this, a connection is setup on launch to the Tygron API. After establishing this connection, a thread is used that requests all data that have a Loader defined in the Environment class every 10 seconds. The Tygron environment is prone to continuous changes because there can be multiple human players of this serious game who fulfill their role as stakeholder and thus change the environment by e.g. granting permits. After the extraction of data an agent can use this data to change its belief base.

[3] https://github.com/eishub

Another option of loading data is on request base. It has been decided to do so because it might cause the agent to use other outdated data that did not get loaded in the request and would cause the agent to make incorrect decisions. Another issue of not reloading constantly would be that if you start requesting data when the agent wants to percept something, you'll bring in much more delay as then you still need to make a full HTTP request. 10 seconds was deemed fast enough because the Tygron game is of slow pace.

The connection between EIS and GOAL is used to let the agent perform an action and subsequently change the environment. For the scope of this project, it has been decided that the agent is only allowed to 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 the GOAL agent should not be burdened with simple mathematical calculations which are irrelevant to the decision making process. Calculating the amount of available land or selecting a piece of land should be done by the connector as GOAL is developed to focus on higher cognitive strategies.

3 Reflection

3.1. Reflection on the product

The most important requirement of the software is that it is expandable. This means that contributors in the future must be able to add new features easily. This is accomplished by keeping loading modules simple and adjustable. After creating a loader it only has to be registered and then the information can be provided to the Agent.

Another feature of the project is that the GOAL module can be swapped out if another engine is decided upon in the future. The project’s environment is independent of GOAL which means that the GOAL engine can be replaced by a different one if the customer desires it.

Another vital aspect is the code quality. During the development of the project, the developers have been keeping the quality of the code high by continually testing new and old features and having discussions about design choices.

3.2. Reflection on the process

Every week a sprint plan was put together 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 pull requests were made of separate branches 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 bot using GitHub webhooks that would send 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. This ensured that pull requests were quickly handled and encouraged discussion.

We used both Jenkins and Travis as continuous integration tools. The tools added an indicator to the pull request that indicated whether all maven tests passed. The maven tests include: PMD[4], checkstyle[5] and our own JUnit-tests. Jenkins[6] also provided a building platform with which other groups could grab the latest version of our code without having to build it themselves.

Using Github in this way was an improvement for our workflow because everyone could easily work on their individual tasks each week without having to worry about other code changing all of a sudden. The bot that forwarded the pull request messages kept everyone notified that they had to check the pull request, which was a major improvement because we did not have to ask people to check the pull request. The continuous integration ensured that our code would still build and all tests still succeeded. This was not always the case when it got pushed and pull requested, so that saved us many time we would have lost reverting changes and finding what part of the code did not work anymore.

[4] http://pmd.sourceforge.net/
[5] http://checkstyle.sourceforge.net/
[6] http://jenkins.buildwise.eu/job/tygron-connect/

4

Developed Functionalities

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. The HttpConnection class sets up the connection for this purpose.

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. The Connector class can send both GET requests and POST requests.

To process the results of our requests, there are ResultHandlers that specify what should be done with a result. These Handlers can transform Strings to objects such as BuildingLists.

All entities that can exist on a map of a Tygron session, such as buildings or parks, are sent in the Well-Known Text format[7]. A standalone string in WKT format is unusable. The PolygonUtil class was used to convert these strings into polygon objects. The conversion was needed because you cannot do calculations on string representations of polygons. 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. function that returns whether a polygon contains another polygon. The Tygron API sends buildings and lands in array form. Each stakeholder can own multiple pieces of land. The PolygonUtil functions are used in actions to calculate for every stakeholder the available land, the land occupied with buildings, and to select a piece of land for building or demolishing.

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 agent. This can easily be altered to send the information to the agent first and let it make the decision. The PopUpHandler also keeps track of the number of request sent by the agent that have not been answered yet, such as buy land requests or permit requests. The agent does not receive any popup information, but instead receives the amount of open requests it still has.

[7] https://en.wikipedia.org/wiki/Well-known_text

5

Interaction Design

5.1. Interaction design evaluation

In this context project only one group that used this software (the Tygron Environment). However, other users might use the software in the future. The usability of this software is evaluated in an experiment. It is expected that the participant is familiar with programming and has some knowledge on how programming in GOAL works.

Goal of the experiment

This project does not have any visual components on its own, instead programmers have to use the API to build their own virtual human. It is of great importance that the user understands how to work with this software through the available documentation, the Tygron Manual[8]. The goal of the experiment was to find out what the participants thought of the manual.

Setup of the experiment

A couple of programming tasks have been devised that our participants have to perform. The programming is done in GOAL and the participants had received documentation with all available actions and percepts. Furthermore they had been provided with a basic template in GOAL to save time, with all the available percepts and actions are already in place. This way the participants only have to worry about their implementation. During their performance they are asked to think out loud. All of their important comments are noted. After each task is completed the duration and the correctness of the performed task is noted.

The tasks

In the first task the participants were asked to simply buy some land. This is a simple task to familiarize the participants with the documentation.

In the second task the participants were asked 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 the participants were asked to check and see if the agent has sufficient funds and no pending permits. The agent in the game that was set up purposely had no funds. Now the participant has to figure out a way to get money. After the participant has money, the participant has to buy land. This task is considered difficult.

[8] https://github.com/tygron-virtual-humans/tygron-connect/blob/master/SE%20Deliverables/Manual/TygronManual.pdf

Results
Participant 2, 3 and 5 had used GOAL recently, and participant 1 and 4 had to refresh their knowledge first.

	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5
Task 1	9:01	3:41	5:20	8:25	5:28
Task 2	2:20	1:24	2:05	2:34	1:50
Task 3		4:03	3:47	6:10	4:35

Conclusion
During the beginning of task 1 where the participant was getting familiar with the documentation, a participant mentioned that a startup guide to freshen one's GOAL knowledge would be handy helping him to perform the task faster. The manual is however made for people with GOAL knowledge, so there was decided not to add a GOAL section. The participants that had recently used GOAL did not need time to see how GOAL actually worked and performed the task seemingly faster.

The second task was more difficult, but because the participants knew the documentation they finished much more quickly. This task was deemed more easy by the participants. Some participants used the stakeholder ID, in the description there was stated this was an unique number. This was seen as an unclear description and therefore all ID descriptions have been changed.

The third task was experienced as difficult by the participants. None of the participants knew what permits were because they were not familiar with the Tygron engine. Because of that the manual got updated with a clearer explanation of permits. Also some distance units where unclear. Such as the type of the surface parameter and the price parameter. To combat these unclarities units are added to some parameters, e.g. price is per m^2. There was also a problem with the budget of the player, which was unclear in the manual and has been updated.

6 Evaluation

6.1. Evaluation of functional modules

Approximately 87% of our code base is JUnit tested. For the actual implementation and evaluation of the rest there are programs that can be run to go over the full (Java) implementation of the given actions. One can then visually, in the game, validate that the actions are executed properly.

The same applies for the GOAL implementation. The VH-group did lots of integration tests with our environment, and we've also run several tests with GOAL for the Human-Computer Interaction evaluation. Our conclusion was that the actions we've currently implemented work flawlessly.

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 our product 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

As our software is primarily used for integration, it is hard to evaluate the product as a separate 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.

We have conducted a small interview which can be found on the next page.

Did the requested functionalities function as expected?
Yes, sometimes there was a small bug but those were fixed quickly every time.

What did not function correctly or not as expected?
In the beginning there were some struggles, mainly because no one knew what was possible in regard to the Tygron API. We expected some things that were not possible.

Does the implementation work fast enough for your agent?
There was a small delay, which makes it difficult, but in general it is fast enough.

How did you use the implementation and was the implementation clear?
We used the implementation to create an agent. The implementation was always clear and could be found in the manual. Clarification could be requested on whatsapp or face to face.

Conclusion
We conclude from this interview that the implementation is good enough for the agent to function correctly and in a timely manner. We also conclude that our manual is comprehensive. The end user thinks our product is good enough and they can build a virtual human that meets their expectations. Our manual has improved after the HCI experiment, the other group had good use of that manual and we think that the manual is clear and representative.

7 Outlook

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.

The code can be edited so that pop ups and messages are sent to the agent instead of answering them automatically in the connector. This gives the agent more freedom to devise a good strategy and even refuse requests, making it a better replacement for a real stakeholder. This would make it difficult to track which pop up is referenced by the agent and get the relevant information linked to the pop ups/messages.

A Appendices

A.1. Sprint Plans

The sprint plans can be found at the following link.
https://github.com/tygron-virtual-humans/tygron-connect/tree/master/SE%20Deliverables

A.2. Sprint Reflections

The sprint reflections can be found at the following link.
https://github.com/tygron-virtual-humans/tygron-connect/tree/master/SE%20Deliverables

A.3. Recommendations Tygron

To ensure proper working agents for Tygron in the future, precautions need to be taken to keep up with any changes in Tygron without having to restructure the whole connector code. During the project development, changes in the Tygron API have brought to light some issues that may occur. Many issues could have been fixed more quickly if a log was kept of all API changes. An alternative to keeping a changelog is to freeze a version of an API. A client can specify which version it is coded for and the server will keep this into account

Some changes in the API included id changes in data lists. A list that caused significant delay was server words. These words have random id's and, when requested from the API, do not have their name as an attribute. This makes it difficult to correct code if id's are changed. It requires a full revision of all server words to get the correct id's associated with their content. This could be avoided by providing names and types of server words when they are requested from the API.

When sending action requests to the API, null given back as an answer. This causes uncertainties, because it is difficult to determine whether an action succeeded or not. If the code does not work properly, debugging becomes a time-consuming process. It is better to receive a meaningful response, stating whether the action failed, and if so, what the issue was.

The game is very player-oriented with popups and messages. These contain text which has to be parsed in order to extract meaning from them. A more computer-oriented approach would be to attach a type to each popup and message, along with the price and land surface associated with these messages.
Another useful improvement would be to use a push based approach instead of the current pull based approach. Information should be sent as it becomes available and not requested by the code in random intervals. This would reduce the needless amount of reloading done by the environment and would significantly speed up connector.

A last useful improvement would be to provide a price per square meter for every function type available in the API. This would enable the agent to request a building of a certain cost instead of a certain surface, making it easier to create a good strategy.

A.4. Manual

The manual for the environment can be found at the following link.
https://github.com/tygron-virtual-humans/tygron-connect/raw/master/SE%20Deliverables/Manual/TygronManual.pdf