Carried Away

Luka Miljak	lmiljak	4368916
Marcel Kuipers	mekuipers	4389042
Damian Voorhout	dvoorhout	4388550
Nils Hullegien	nhullegien	4389069
Remi Flinterman	rflinterman	4362950

An interactive game using Virtual Reality

Table of Contents

---

1. Introduction

The world of Virtual Reality opens up many possibilities for interactive gameplay. This report serves as background for the game *Carried Away*, an interactive multiplayer game that focuses on the interaction between players using Virtual Reality and players that are not using Virtual Reality. The delivered game, which can be played in casual, social situations, attempts to entertain a group of people in a unique way, using unconventional gameplay mechanics.

1.1 End-user requirements

The product attempts to satisfy a number of requirements that are intentionally not very clearly defined, this was to allow for a lot of freedom in the design. Below, a short description is given of all the end-user requirements of the game.

Firstly, there needed to be one player that lives immersed in the virtual world and has a special role in that world. Since this requirement forces (at least) two very distinct roles, the game can provide multiple different types of experiences. The usage of Virtual Reality allows for a lot of experimentation with how the player experiences the world. Virtual reality can be overwhelming and it is necessary to keep in mind that some players can get nauseous from using it.

Secondly, multiple other players need to be able to join the game and act upon the virtual world. Since these players are unable to see the virtual world, they have to work with less, or at least different information. This makes the giving and withholding of information an important aspect of the game.

Thirdly, all players need to be actively engaged in the gameplay. This requirement forces the game to have a certain level of intensity. This makes the user's attention is a very important aspect of the game, which requires research. Each player needs a diverse role, to ensure that he always stays involved and does not get bored.

Lastly, all players need to be in the same room. Having players in the same room allows them to talk to each other. Communication is a very important tool and should play a central role in the game.

2. Overview

This chapter provides a description of the developed and implemented software product. Firstly, a description of the game is given. This description will be expanded upon in the chapter *Functionalities*. After that, some information is given about the hardware used during the game. Lastly, some information regarding the software used during the development of said game is provided.

2.1 Game description

The game, which is played with 5 players, implements asynchronous gameplay by not assigning everybody the same role.

One player, called the *commander*, is wearing an Oculus Rift and is immersed in the virtual world of *Carried Away.* In this world he is located on top of a platform, which is moving forward on an infinitely long path. He can look around, but he can not interact with the world himself.

The other four players, who are playing on their smartphones, are called the *carriers*. They cannot see the virtual world, but they can interact with it. By synchronously tilting their devices they can steer the platform. Steering is needed to avoid deadly obstacles on the road.

Besides obstacles, enemies spawn randomly on the side of the road. An enemy targets a certain player and chooses from which side he will attack him. Each carrier has 3 buttons on their device that show the directions in which the carrier can attack. He has to rely on the commander to tell him where enemies are located. If a carrier attacks in a direction where no enemy is located, his attack will go on cooldown, leaving him vulnerable to attack.

Carriers start with 3 lives, losing a life each time he is hit by an obstacle or enemy. When a carrier dies he becomes immobilized for a short period of time, meaning he will not be able to steer the platform. The game ends when two carriers are dead at the same time.

The platform wobbles and shakes carrier. The stability of the platform depends on the number of carriers and how synchronously they tilt their phone. An unstable platform can be very disorienting for the commander.

The game gets harder the longer it progresses. The speed of the platform increases, making it more difficult to dodge obstacles. At the same time, the score also increases. The goal of the game is to get the highest score.

2.2 Hardware details

The game requires an Oculus Rift and four Android devices for the carriers. These five devices are all connected to a pc, which functions as a server as well as the visual provider for the Oculus user. The server receives information from the android users and processes it in the game. The server can also send information to the android users in order to influence what is displayed on their devices.

Games that make use of Virtual Reality typically require a powerful computer to run. Relatively speaking though, *Carried Away* is a pretty lightweight game. It is recommended to use at least something similar to a Nvidia Geforce 670 GTX in order to make the game run without performance issues.

2.3 Software details

The server requires a Windows pc to run the desktop application. The android users require an Android version of 4 (Ice Cream Sandwich, API 19) or higher to run the mobile application. For android users to be connected to the server, both the server and mobile users need to be connected to the same network, preferably a private one. If the network is not private, port forwarding may be required.

3. Reflection

This goal of this chapter is to describe this project from a software engineering perspective. A rundown is given of a number of software engineering topics, how they were relevant to the project and what lessons were learned.

3.1 Scrum

The Scrum framework was used for the development of this project, with sprints of one week and four or five face-to-face meetings every sprint. Each sprint, there was one fixed meeting. These meetings took place at the beginning of each new sprint. During these meetings, progress and plans for the upcoming sprint were discussed and afterwards formalized in the sprint retrospective and sprint backlog. The decision to have only one fixed meeting was made because of the very short length of the sprints and the fact that there were so many face-to-face meetings, which made communication easy.

In hindsight, there was a bit too much freedom, which sometimes led to people not knowing what to do, missing information and some tasks being neglected. The scrum master giving more structure to the meetings could have prevented some of these things from happening. On the other hand, the group members felt comfortable with the freedom and were never completely at a loss.

3.2 Project design

At the start of the project it was agreed that the SOLID principles would be used. However, mainly due to technical debt, one out of the five principles were ignored. This caused the maintainability of the project to drop drastically. Having to rush finishing certain features before the deadline mainly caused violation to the “Single Reponsiblity Principle”, as it made the code more compact, allowing the developer to quickly implement the feature.

One way this problem could have been solved from the start, was through Responsibility Driven Design. It would have ensured that the SOLID principles were being followed, giving the project a better maintainability, while not reducing too much of the time.

3.3 Design patterns

The usage of design patterns allows programmers to solve commonly occurring problems in an elegant manner. Below a quick rundown is given of the design pattern that were used during the production of this project.

Singleton: This is a controversial pattern and was used as little as possible. Still, there were a few situations in which using the pattern was justifiable. For example: a class which functions as a container for the jMonkey-implemented class *AssetManager* was made a singleton. The only use of the *AssetManager* was to retrieve models for the *Assets* folder, after it had been created at startup. Any visual object needs to be able to access the *AssetManager*, only to retrieve its model. It was argued that the container class, called *ProjectAssetManager*, was a utility class and that making it a singleton did not break any object-oriented programing principles.

Abstract factory: Used while generating obstacles. This pattern makes it easy to switch between the types of obstacles that are created. After selecting an initial type of obstacle to spawn in the *ObstacleSpawner*, it is possible to switch to spawning different types of obstacles by instantiating the *obstacleFactory* field with an *ObstacleFactory* of a different type. This makes it possible to, for instance, first have a section of static obstacles and following that up with a section of moving obstacles.

3.4 Lessons Learned

The project taught the members of the production team a lot, mainly through failure; trial and error. This section will provide insight into what exactly went wrong during the project and what lessons this provided.

One of the main recurring issues was the lack of planning regarding the structure of the code. Many features on their own were thought out and implemented without paying attention to the interaction between the features. This resulted in a lot of refactoring, some of which was unnecessary if some more time was spent on designing the structure. This could have been avoided by using more responsibility driven design. It would ensure that features would be separated from the structure. The creation of the structure would then be easier since it wouldn't affect the individual features.

The use of android devices, an Oculus rift, and an unfamiliar game engine resulted in a lot of technical difficulties, especially in the first 4 weeks of the project. Every project would have had these issues, they were not unique to this project. Still, it resulted in more problems than were anticipated, which in turn created a large backlog in the first few weeks. Planning for the worst case scenario when it comes to said problems, and making sure the backlog of last weeks sprint gets cleared as soon as possible would have made it so more features reached their respective deadlines.

The last major problem goes hand in hand with the one stated above. When a backlog piles up, it is expected to fix the backlog as soon as possible so new planned features aren't compromised. In order to meet those deadlines, some older features in the backlog were rushed to meet the upcoming deadline. Features were placed in places where they don't belong, responsibility of classes was given to classes which didn't have that responsibility, tests were either rushed or not implemented at all, etc. Better planning and realising how much of an impact an ever growing backlog has on development could have avoided these major problems.

3.5 Testing

In this project user tests were done to test the full game, as well as the playability. One goal of these tests was to find out if the game didn't have any breaking bugs. Some of these bugs were encountered, so it was definitely worth it to do these tests. Another goal of these test was to determine if the game was playable. This last part was all about tweaking values in the game and balancing the different factors. Examples of these factors are the spawn rate of the enemies, the speed of the game and the interval at which a random event was triggered.

To test the code itself, JUnit was used for unit testing, as well as Mockito and PowerMockito to test the more difficult parts of the code. Some parts of the code weren't tested by unit testing since this was either impossible or too difficult to do. These parts of the code can be found in a document on the github repository of this project, as well as a short description about why this part of the code wasn't tested.

4. Functionalities

During the process of creating this project many functionalities were devised. In this chapter the functionalities that were added are described. Focus is put on how these functionalities satisfy the needs of the users. This chapter gives a high-level description of the most important functionalities.

The game features a virtual environment in which the player wearing the Oculus Rift is immersed. He can look around in all directions and has an overview of everything that is happening. The environment is made up of level pieces that all share a forest theme. The world aims to give the player a feeling of immersion.

Users with an Android device that are connected to the server can tilt their device in such a way that the platform, on which the virtual reality user is standing, actually moves.

Depending on how the Android users are steering, the platform may get very unbalanced. The less synchronized the users are holding their device, the harder the platform is rotating, giving a slight feeling of nausea to the user wearing the Oculus Rift.

The level is infinitely long. This allows users to challenge themselves to keep breaking their old score. To prevent the environment from being bland, multiple different level pieces are created and spawn at random.

Enemies are spawned into the world at a specified interval. They pick a carrier to attack and move to his location. For each carrier, there are three directions from which an enemy can attack. This direction can be communicated by the commander, allowing carriers to attack back.

By steering the platform, the carriers can avoid moving obstacle. If an obstacle hits one of the carriers of the platform, the carrier takes damage.

To make the game more fun for the Android users, rather than just listening to the commanding Oculus user, there are random events. The only implemented event is the *bug spray* event. This event randomly appears for the Android users, which does not allow them to attack any enemies while the event is active. To end the event, a bug must be sprayed. One of the Android users is plagued by a bug and one of the users contains the spray. By swiping the screen in a certain direction, an Android user can give the spray to another user. If the user with the bug has the spray, it can kill the bug and the game can continue.

5. Interaction design

Our project is a video game, a digital piece of entertainment that gives users an enjoyable, interactive experience. Since its main purpose is to entertain, HCI (Human Computer Interaction) is a very important aspect in the development process. The game needs to be user friendly, accessible and ‘fun’. Since it is very easy to make mistakes in fulfilling these requirements, especially since ‘fun’ is such a subjective thing, we need to have users play our game and see how they interact with it. And of course since this game uses the Oculus Rift VR headset, motion sickness needs to be prevented as much as possible.

5.1 Design Process:

Throughout the design process, users had to be taken into account. “What would they enjoy?”, “What would they get annoyed by?” and “What would be confusing for them and how can it be fixed?” were questions that continuously had to be asked. When planning to implement features in the backlogs, user stories were used, looking at the features and why they’d want them from their perspective. Of course this alone cannot guarantee a user friendly experience, but there are more tricks for that.

There are seven design principles revolving around interaction design, called the design principles of Norman. Of these seven, four were followed:

- *Simplify the structure of the task.* The carriers have simple tasks: steering through motion controls (like a steering wheel), attacking (through three big buttons on their screen) and dealing with random events with clear instructions. The commander also has one simple task: to look out for obstacles and enemies and tell the carriers what to do to avoid them.
- *Make everything clear and visible.* The buttons on the android device have different colours and black text telling them in which direction their attacks are. There are red hearts that represent health for the carrier, which turn grey. The commander can clearly see what and where the obstacles and enemies are, thanks to their notable design.
- *Make sure the images look good in relation to each other.* On the carrier’s screen, the buttons are the most important mechanic, so they are large in the middle of the screen, with the hearts underneath. The character models in the 3D environment also have logical proportions.
- *Take the mistakes of users into account.* Carriers might want to spam the attack buttons to efficiently defend against enemies, without having to listen to the commander. However when they miss, they get a sound effect signalling that they messed up and they get a two second penalty.

5.2 Evaluation:

The first round of playtesting resulted in some valuable feedback. Fortunately the game was fun, despite several bugs hindering the experience. Basically the game needed to be refined and bugs need to be removed and a decent party game would come to existence.

The only major problem that came to light was that it was not always conveyed well how certain elements in the game worked. Sadly due to time constraints, a way to solve this could not be implemented into the game. The only thing that could be done was delivering additional information with the game.

5.3 Emotion and social aspects:

Carried Away is a fun, hectic experience, which can cause both happiness and frustration. The game is pretty fair, so the expected response is that this frustration will lead people to sort of get angry with their own performance and try to improve. Not enough playtesting has been done to see this actually happen.

Something that was imperative to take into account during development is motion sickness. People often seem to have trouble with it while wearing an Oculus Rift. Our experience is designed so that the main motivation for players not to lose is that the commander will fall over in the game. It cause motion sickness, which essentially is punishment for losing. However, people shouldn't feel disgusted or sick while playing our happy little game. The camera movements are purposefully not too wild, so that the commander won't get sick during gameplay. Falling over is a short experience after which the game is over, so the commander immediately gets a break right after.

During playtesting it was apparent that the group dynamic during gameplay is the way intended. All players are actively involved at all times. The commander has has fun telling the carriers what to do to survive. He also seems to grow a sense of responsibility for the other members, as he tells them which enemy is targeting which player from which direction. The carriers have fun carrying out the commander's instructions and defending against enemies, and get frustrated when their attack doesn't hit an enemy and they have to take a hit. Bug spray events surprise the carriers and causes a funny moment in which they frantically yell "I've got a bug, where's the bug spray!?", etc. In short: all players are immersed in the game and enjoy working together to achieve their goal, which is getting as far as possible.

6. Evaluation of modules

Evaluation of the functional modules and the product in its entirety, including failure analysis.

This chapter is dedicated to discussing the performance of the product and how this performance was assessed.

6.1 Play Testing

As mentioned before in the Interaction Design segment, playtesting has also resulted in useful feedback. Several bugs were discovered and a few design issues came to light, so they could be taken care of. The reason why this is a good way to evaluate the project, is that certain issues are too hard to discover during programming. When an error is made in the design process, the designers think that their code is correct, until during playtesting is revealed that their thinking was wrong or that they overlooked something. In addition to this, none of our pc's were able to use the Oculus Rift, so issues caused by VR could only be discovered during playtesting.

6.2 Failure analysis

Being quite an unconventional game, this project consists of multiple modules that needed to be linked together. Making a number of technologies work together can be difficult and can result in behaviour that is difficult to predict. This section is split up in two parts: one part discusses how individual modules were checked for failures, while the second part talks about how this was done for the communication between multiple modules.

During this project various software tools were used. The software itself was mostly written in the Eclipse IDE using the JMonkey engine. In order to convert blender objects to jmonkey model files, the JMonkey IDE was used, since this cannot be done with the Eclipse IDE.

In Eclipse, several tools like pmd, findbugs, and checkstyle were used to ensure the creation of a product which is up to standards regarding code quality and code readability. Version control was done using git. Travis was used to ensure failing builds would never make it to the master.

Logging was important for checking the behaviour of modules. In the desktop application, logs to console were used extensively for this purpose. Using logcat, the same can be achieved on Android, as long as the device is connected to a pc. Logging was particularly useful while processing network messages.

7. Outlook

This chapter takes a look at the potential of the project and and puts it in perspective with other game within the same category. A small section is dedicated to describing, in broad lines, what actions could be taken to improve the product. After that, some final remarks are made about the project.

7.1 Future improvements

The created game is very simplistic. For the project, this was intentional, because most focus was put on making a simple concept work well. Many additional functionalities can be created to make the game more diverse and interesting. The project is very open to extension and also

improvements.