

# Architecture design

Team Krokot

13-05-2016

Irene van der Blij	ivanderblij	4385691
Jochem Heijltjes	jheijltjes	1534041
Mayke Kloppenburg	mlkloppenburg	4383265
Alan van Rossum	alanvanrossum	4293932
Harvey van Veltom	hvanveltom	4350073

# Table of contents

- 1 Introduction
  - 1.1 Design goals
    - 1.1.1 Code quality
    - 1.1.2 Availability
    - 1.1.3 Performance
    - 1.1.4 Maintainability
- 2 Software architecture views
  - 2.1 Subsystem decomposition
    - 2.1.1 The Oculus Rift
    - 2.1.2 The Wireless Controller
    - 2.1.3 Desktop Machine
    - 2.1.4 Game Host
    - 2.1.5 Android Devices
  - 2.2 Hardware/software mapping
  - 2.3 Persistent data management
  - 2.4 Concurrency
- 3 Glossary

# 1 Introduction

This document will provide an outline of the system that is going to be built by Team Kroket during the Context Project in the context Computer Games. This chapter will elaborate the design goals of the system.

## 1.1 Design goals

The design of a system is very important for every developer working on it. Therefore this section will discuss the desired design goals for the system.

### 1.1.1 Code quality

In order to have a system that is easily altered, extended and understood, the system should have adequate code quality. The master branch should always be of good quality; this means that everyone will do their work on a separate branch. Continuous integration will be used, so whenever someone thinks the work on their branch is finished and the code quality is on the same level as the master branch, he or she can create a pull request. At least two people should review this pull request, not only looking at functionality, but also verifying the code quality. In this manner cleanness of the code in the master branch is ensured, and thus that the system that is released after every sprint has adequate code quality. The code should be neat, clear and as bug-free as possible. To this end, Checkstyle, PMD and Findbugs will be used.

### 1.1.2 Availability

In order to create a system that meets the requirements of the stakeholders, a system with high availability has many advantages. At the end of each week a playable release is made available to the stakeholders. This is done by using Scrum and weekly sprints. This allows the stakeholders to give feedback during the development process, and adjust their wishes and requirements. When there is the possibility to give feedback during the development process instead of only at the end, you can make sure that the product you are developing is what the customer/stakeholders want and require. In this manner, the stakeholders also have a better overview of the product they are going to receive at the end, so they will not be surprised (and disappointed).

### 1.1.3 Performance

In every computer game, performance is of great importance. Game players do not want to wait long amounts of time for the game to respond; they want to keep on playing. This also needs to be incorporated in the to be build system. The performance of the system will also depend on the actual hardware that is going to be used.

### 1.1.4 Maintainability

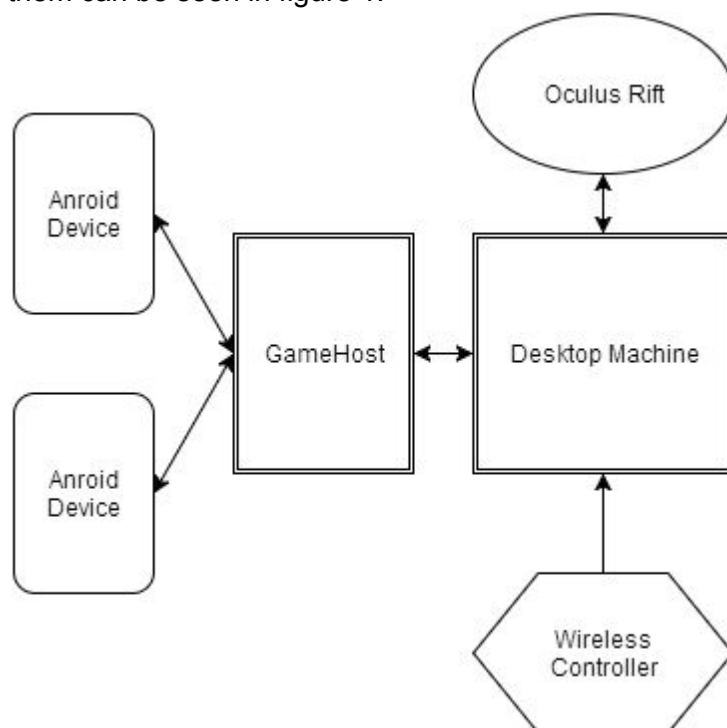
For a system to be maintainable it should be easy to modify the system and to correct faults. It will also ease improving the performance of the system and make it easier for the system to adapt to another environment. Iterative development and regular feedback, as mentioned earlier, already contribute to the maintainability. The code of the system should also be readable and easy to understand. This can be achieved by documentation and adequate comments explaining the code.

## 2 Software architecture views

This chapter will describe the architecture of the system. The first section is about the subsystem decomposition, the subsystems and dependencies between them. The second section is about the hardware/software mapping. In the third section the database and the database design will be discussed. The last section is about concurrency.

### 2.1 Subsystem decomposition

The architecture can be divided into different subsystems. As of now it consists of 5 different subsystems. The Oculus Rift, The Wireless controller, two android devices, a game host and a desktop machine. An overview of the different subsystems and the interactions between them can be seen in figure 1.



*Figure 1: Architecture Design of the game*

#### 2.1.1 The Oculus Rift

The Rift will interact with the Desktop Machine. The desktop machine will send the Rift the image of the virtual world so the user can look around in said virtual world. In turn the Oculus Rift will send back the coordinates of the headset so the view can be updated accordingly. For example if the user in the Oculus Rift looks left the new coordinates are sent to the Desktop Machine and in turn the updated image of the virtual world is sent back to the Rift.

#### 2.1.2 The (Wireless) Controller

The controller does not have to be wireless. The (wireless) controller is a way for the oculus rift user to send input to the desktop machine. For example if the Rift user needs to inspect an element he can press a button on the (wireless) controller. Then the Desktop machine will

know what the Rift user wants and will respond accordingly. The (wireless) controller is only used to send input to the desktop machine it doesn't receive any kind of input itself.

### 2.1.3 Desktop Machine

The desktop machine interacts with several subsystems. Namely the Oculus Rift, the Wireless controller and the game host. The desktop machine receives input from both the wireless controller and the Oculus Rift. The desktop machine will check the input of both and will update the image of the virtual world and send it to the Oculus Rift based on the coordinates it received from the Rift. It will also send these inputs to the Game Host.

### 2.1.4 Game Host

The game host interacts with both the Desktop machine and the android devices. The desktop machine and android devices will send information to the game host. This information could be clues the players gathered, potential answers to a puzzle etc. The game host will process these inputs and send information back to the subsystems. The information sent back could be how much time is left in the game, a new puzzle the players have to solve, a victory screen if the players have won etc.

### 2.1.5 Android Devices

The android devices will interact with the Game Host. The android devices will send input to the game host. This could be solutions to a puzzle for example. These inputs could be information how much time is left, what the android players need to do next, potential clues, victory screen if the players have won etcetera.

## 2.2 Hardware/software mapping

The software and hardware are different for the users, as illustrated in figure 2. When the game is started, a start screen will appear. The start screen is the same for all users. In this screen, the player can see if everyone is connected and start the game.

There is one Oculus Rift user. This user has the game on his computer. He can look around in the environment (the interface) with the oculus rift and move around and interact with the environment using a wireless controller. The other two users have an app on their smartphone. The smartphones and computer are connected to a server, so there can be communication between the Oculus user and the smartphone users.

A single application running on a desktop computer operates as host. The clients mainly communicate about whether a minigame on the smartphone can be initiated and whether a minigame is finished. All the clients interact with this host through TCP. For the game, it is necessary to exchange string messages to initialise events in the app and Oculus game. This means it is very important to have a reliable protocol, so it is certain messages have been received. This is why TCP is used.

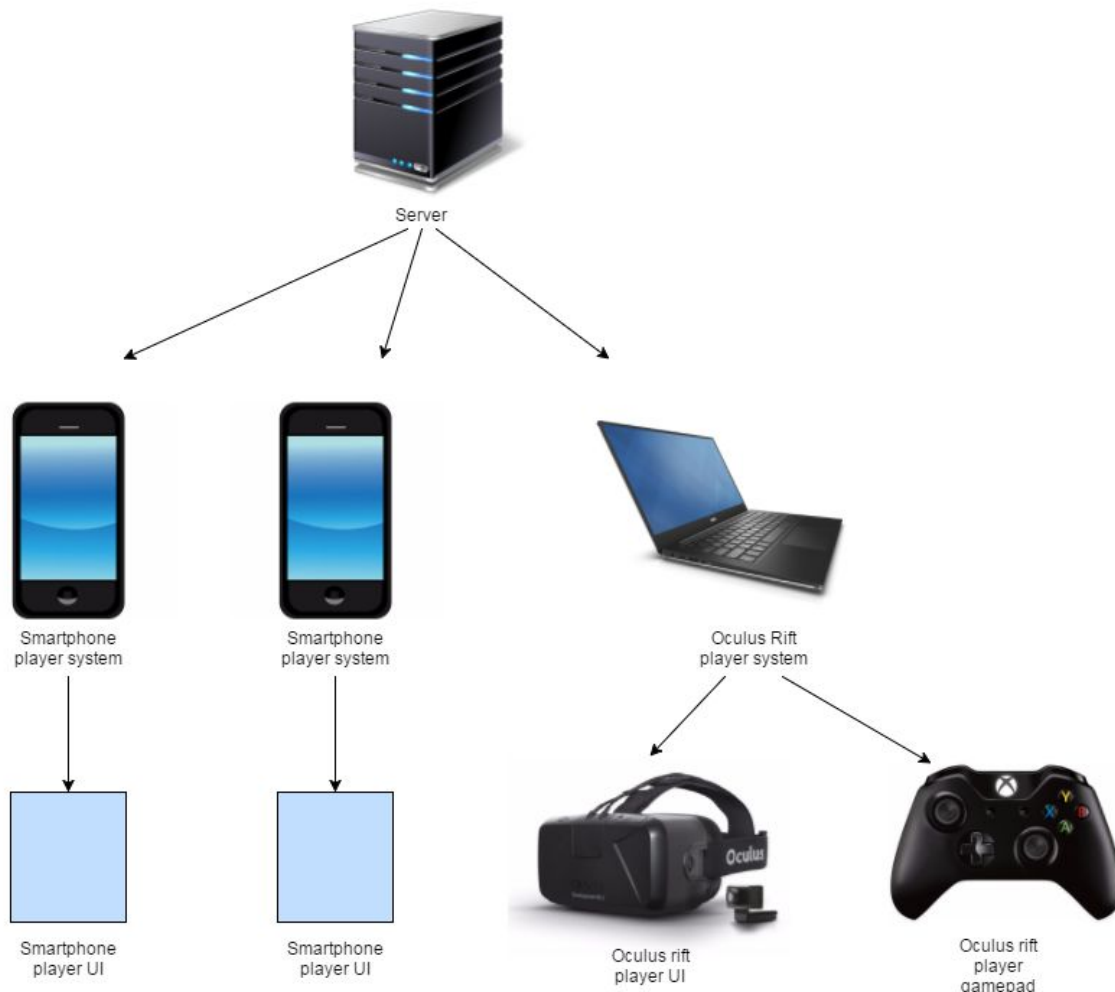


Figure 2: hardware/software

## 2.3 Persistent data management

The game concept does not require user data to be stored. It is a one time game without user profiles etcetera. Therefore, there is no need for a database design.

However, our system could have a form of high scores in the future. The scores would have to be stored throughout different sessions and thus require a persistent form of a database for data storage. The use of an actual RDBMS would be unnecessary, as a simple JSON or XML file would suffice. A simple method would read the data and output the 3 or 10 teams with the highest scores when prompted.

## 2.4 Concurrency

The system consists of multiple clients interacting with a gamehost, through TCP. Because the gamehost/server will be responsible for hosting an entire game session and ordering each client what to do, all data will flow through the server. The clients won't communicate with each other directly. The host is also responsible for using shared resources. We chose TCP over UDP for now because we want our data transmissions to be reliable and all we are exchanging is events. For position data, we might use UDP.

In the gamehost/server, the accepting part is in its own thread. For each incoming connection, a separate thread is spawned so that each player/user is dealt with in its own instance.

Each connected client has to wait for incoming data from the server, such as events. To make sure the clients can keep running while waiting for incoming data, the receiving part of each client is dealt with in its own separate thread.

Because threading is potentially dangerous, we have to be extra careful when dealing with shared variables and incoming data.

### 3 Glossary

**Android** - Operating system for Mobile Phones and tablets. Makes use of the Linux Kernel and the Java programming platform.

**Branch** - A parallel version of a repository. A branch is contained within a repository, but does not affect the primary or master branch.

**Client** - An application that relies on a server or host to perform operations.

**Controller** - An input device primarily used to play video games on a console, like Xbox or Playstation.

**Host** - A computer system that can be accessed by clients at a remote location, often through the internet.

**Master branch** - The default and main branch. Should contain the latest working version of the system.

**Oculus Rift** - A pair of Virtual Reality goggles with a stereoscopic field of view, designed for gamers. The device measures the movements of its wearer.

**Pull request** - A proposed change to a repository, submitted by a user, and accepted or rejected by the repository's collaborators.

**RDBMS** - Relational Database Management System, a database management system based on the relational model. Easy to understand choice of storing record data.

**Repository** - Git's name for the 'main folder' that contains all project files and each files' revision history. Collaborators can contribute to the state of the repository by committing and pushing changes to the project files.

**Smartphone** - A phone with many features running an advanced operating system, similar to a desktop computer.

**Socket** - An endpoint of a connection in computer networking.

**SQL** - Structured Query Language, a standardised language that's used to ask and alter data from a database management system (see: **RDBMS**).

**Virtual Reality** - An artificial world, generated by a computer.

**TCP** - Transmission Control Protocol, often used to establish connections via the internet.

**UDP** - User Datagram Protocol, a stateless/connectionless protocol for exchanging data over networks.