Emergent Architecture Design Document
DuoDrive

Pim van den Bogaerdt, pvandenbogaerd, 4215516
Ramin Erfani, rsafarpourerfa, 4205502
Robert Luijendijk, rluijendijk, 4161467
Mourad el Maouchi, melmaouchi, 4204379
Kevin van Nes, kjmvannes, 4020871

June 6, 2014

TI2805 Contextproject, 2013/2014, TU Delft
Group 5, Computer Games
Version 2 (updated draft)

Abstract

This document describes the architecture of the system of which our product consists. Firstly, we will describe the design goals for our final product. After that, our Software Architecture Views will be expanded upon. Finally, a glossary can be found at the end of the document. This document will be edited and changed throughout the duration of the project to fit our current system architecture. The latest class diagram can be found in the Appendix.

Contents

1 Introduction

In this document we will describe the architecture of the system that we will be creating during the Computer Games Contextproject. The architecture will be explained in a form containing high level components with the sub-components and sub-systems. This will give more insight into what the system is composed of and how the several elements of the system cohere.

1.1 Design goals

As a team, we have set four design goals that will be maintained throughout the development phase of our product:

Availability

During this project our product will be *continuously integrated* with the goal of having a working product running at any time. This allows us and the client to test and work with the product at any time, even before the final release. After the final release we will want to make sure that the server has an uptime as high as possible, especially during the time that people are visiting our client's company or area.

Replayability

A design goal that we consider to be specifically important for our product is replayability. We want the players to enjoy the game in such a way that the game will be replayable in both the short and long term. We want the players to think of our game as a memorable one, so they may look forward to playing the game the next time. To reach this goal, the game should make a solid first impression, after which elements such as randomness and other unpredictable events will keep the game interesting to play.

Usability

The game is designed in such a way that the player will be able to start playing the game with few instructions. Those players who do not want to or are not able to figure out the controls are offered a small tutorial, which they can choose to follow at the start screen. After a few minutes, or even seconds, the player will have learned how to work with the system and play the game. The physics in the game are set to make the user experience better.
Moreover, with feedback given during the game, the user will be able to see progress and see how well he is playing the game. An example of the given feedback is a timer.

Performance

The system will have a central server which is situated in the same room/building/area as the players who are playing the game. Also, all the players should be connected on the same network as the server is. Since the server will be the machine sending and receiving the data of the players, it is necessary that the connection between the player and the server is fast enough. This is necessary to ensure that the players will not encounter any *latency issues*.
In a future release we might implement a system where some of the server's calculations will be calculated locally on the player's device. This will take load off the server and may improve the connection speed throughout the game.

2 Software Architecture Views

This chapter will discuss the architecture of the system and how it is decomposed. In the first paragraph the subsystems will be decomposed and each subsystem will be explained. In the second paragraph we will elaborate upon the relations and mapping between hardware and software. In the last paragraph the data management will be explained.

2.1 Subsystem Decomposition

The software architecture of the system consists of three different subsystems: the server, the system for the steering player and the system for the player who controls the throttle:

- Server
  The server in the system maintains the data flow. It will send the data of the positions of the cars to all players, so that every player has a near real-time experience, where they can see other players' positions. All data sent by a player will first be sent to the server, which distributes it to the other players. There is no player to player data flow at this moment.
- Steering player system
  The system for the steering player will only contain the view that the steering player sees. This will be a limited view of the track. The steerer will only have the ability to steer.
- Throttler system
  In the throttler system the abilities to perform certain actions are also limited. The player will only be able to control the throttle, i.e. accelerating and decelerating. By limiting the view and the possibilities to perform actions, communication will be enforced.

2.2 Hardware/Software Mapping

The hardware used is very varied in terms of the player's device. This device must have Android running. The only other piece of hardware that will be required is a central computer that will serve as a central server. The players will be able to connect as clients to this server, which is conform to our Client-Server architecture.
The software that is being used on the described hardware is the same. Depending on the role (i.e. being a steering player, throttler, or server) the interface will be different. The server will only have a start button, while a player (client) will be able to choose from different options in a start screen. These options are 'Start Game' and 'How-To-Play'. After pressing 'Start Game' a player will be able to choose his role, after which the game starts.

2.3 Persistent Data Management

Persistent data management is not applicable for our product. There is no data that needs to be saved persistently.

3 Glossary

Continuous Integration (Wikipedia, 2014)

Continuous integration is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day. This type of integration is used to make sure a working copy is always available at any moment during a software engineering project.

Latency Issues

In a network, latency is the amount of time a packet needs to get from one point to another, i.e. from a server to a client. Latency issues arise when a packet takes too long to arrive at the endpoint, the fluency of the program may be compromised, because packets may take too long or be dropped completely.

References

Continuous integration. (n.d.). In *Wikipedia*. Retrieved May 15, 2014, from http://en.wikipedia.org/wiki/Continuous_integration

Appendix

//TODO: Add class diagram