DuoDrive

Final Report

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 20, 2014

TI2805 Contextproject, 2013/2014, TU Delft
Group 5, Computer Games
Version 1 (draft)

1 Introduction

We wanted a game that would force interaction, real communication between players which involved a lot of humor and entertainment. We therefore developed DuoDrive, a game that can be classified as a racegame which ensures a lot of communication and interaction. The main idea behind the game is to divide a classical car driver into two parts: a driver and a throttler. In real life these two parts are controlled by one person, by splitting these, a lot of cooperation needs to be done. Not only to reach the finish, but to make sure you're going to win. To ensure our game will entertain the end-user, the player. We had to create end-user requirements

The end-user requirements contained that communication is mandatory, we heavily focused on the communication and thought of features like limited visibility. Another requirement was that the game would be playable on an Android device, we therefore have build an .apk file that can be installed and played on an Android device. Furthermore users had to be able to virtually cooperate, a network server was therefore needed. We implemented networking with the help of Unity. Last but not least, the graphics of the game had to be recognizable, users had to recognize tracks, cars, mud and grass. We created custom sprites to ensure that the player would recognize their environment. The requirements above are the ones we had high-priority's on. These features are not all the features our game provides, for more features please take a look at **??**

During the project, testing was a pre. There were literally hundreds of problems that could occur. We therefore used integration testing, to make sure most of the features still worked as expected once we added and/or modified pieces of code. Whenever a test failed the logs gave us information where and what went wrong, giving us the oppurtunity to quickly resolve problems. For more info on our testing please take a look at **??**

2 Overview of the Developed and Implemented Software Product

The product we have implemented is a top-down, 2D racing game for Android. In this game, pairs of people each control one car. Specifically, the game is designed for waiting rooms where two lines of two chairs are placed, placed back-to-back. One or two pairs of people team up and each pair chooses two chairs placed back-to-back. Each pair will control one of the cars the one person controls the steering wheel (he/she is the driver) and the other controls the throttle (he/she is the throttler).

This is what we described in our design document as well. During development, we have implemented this concretely. The throttler and driver do not see the same thing, instead, the driver sees the car with a small area of light in front of it. The rest of the space is dark. On the other hand, the throttler sees an overview of the whole track, but the car is shown as a dot only. Both players also see signs: the throttler is given signs on how the driver should control the steering wheel (through driver signs of e.g. a U-turn), and the driver occasionally sees signs when the throttler should slow down. Because a player does not see the screen of his teammate, physical communication is enforced in order to reach the finish.

When two pairs are playing, the competitive aspect is apparent: the goal of the game then is to reach the finish at the end of the track first. It is also possible for only one pair to play the game. In that case, the collaborative aspect is apparent but not the competitive one. The goal is then just to reach the finish.

3 Description of the Developed Functionalities

The software product that has been developed, like stated in the previous section, has several functionalities. Every functionality contributes to the bigger part. Moreover, the functionalities have a certain purpose which makes the game more interesting and fun to play.

As stated, there have been several functionalities developed in the game, which will be given below with a brief description.

1. **Tutorial:**
   When the game is being started, there is a possibility to get a tutorial. This will contain a short description of the tasks possible and the main goal of the game.

2. **Steering:**
   The functionality of steering is a solo functionality. The player who will have the task to steer, will only be able to do that. The player will not have the ability to throttle.

3. **Throttling:**
   The functionality of throttling is, like the steering functionality, a solo functionality. This player will only have the ability to throttle, but cannot steer. However, the player will have an overview of the whole track.

4. **Limited visibility:**
   This functionality applies to the steering player. The player will not have a clear view of where he/she is at the moment or where to steer at. This is achieved by both creating a top-down view on the car and making the game dark with only headlights which cannot reach far.

5. **Deceiving road marks:**
   During the game, the track has certain road marks which are deceiving for the steering player. It might look, for example, that the road will continue straight while in fact a turn is coming up.

6. **Mud tiles:**
   On certain tiles, randomly spread over the track, mud has been placed. Depending on the speed of the car, the car will be slowed down. Thus, if the speed is very high, the car will be slowed down a lot immediately. While driving slowly on it will make you proceed over it with the same amount of speed.

7. **Finish screen:**
   At the end of the game, a finishing screen appears. When the first car has reached the finish, the players will get a view of their finishing time. At the moment that the other car finishes, players of both teams will get a view containing each others finishing times.

By means of all these functionalities, combined into a single game, several aspects are achieved. The aspects are communication, amusement and achievement. The first functionality, has been made so that for those who are not familiar with the game or it does not feel intuitive, they will still have a tutorial to go through.

By means of the 2nd until the 6th functionalities the communication and amusement are achieved. Since the jobs are split into two, a first way of communication is needed. Depending on the directions of the throttling player should steer and vice versa depending on instructions about gassing or driving backwards the steering player should react. Moreover, the limited visibility helps the communication because the steering player does not know where to steer to, it again depends on the throttle. The deceiving road marks forces the players to communicate since the steering player cannot rely on what he sees. The last functionality, the mud tiles, will make the steering player communicate fast towards the throttle to slow down when SLOW DOWN appears on his/her screen.

With also those functionalities another aspect has been achieved: amusement. Especially because of the 5th and 6th functionality, a lot of things can go wrong. Because of the chaos and yelling at certain times because of crashes, both players (of a single car) will experience a time of amusement while playing. This has been concluded of the user tests that have been done throughout the development of the product.

The final functionality, the seventh, creates the feeling of achievement. The winner, the first car to finish, will have an immediate notice of having won the game. However, the players of the other car will not know that. They will still attempt to finish in the fastest time possible, making the competition go on. When both cars have finished, they will see their times and compare them against each other. The feeling of achievement is gained by successfully complete a track with difficulties and by communication.

With the presented functionalities stated above, the crucial product attributes in the Design Document have been achieved. By having user tests throughout the product it has been assured that these functionalities are the most important ones which make the product achieve those crucial attributes.

4 Human-Computer Interaction

In this section we discuss what kind of measures we took to keep in mind the Human Computer Interaction (HCI) principles.

4.1 Analysis

In a game there are many features that need attention when the subject is Human Computer Interaction (HCI). The startscreen at the very beginning of the game is a good example. All of the buttons on there must be clear for the student. Then there is the game itself, because it is a racegame the controls of the car must feel responsive. Because our game will be mostly played on touchscreen, players must know how to give gas or steer without explaining to them how to do that. These are just a few main issues with this game regarding HCI.

4.1.1 Startscreen

We have made the buttons fairly large, and made them so that on each screen they adjust to the size of the screen. This way you will not have to touch tiny buttons when your screen has more pixels then the screen we tested on. We also made tutorial for people, this will help the game to be clearer.

4.1.2 Race game

In the game itself we have two cars with different colors, so that the players will know in which car they are. This avoids confusion. Because it is a racegame the separation between the track and the grass must be clear. The responsiveness of the car must feel good. If these features are not in there, the player will lose interest.

4.1.3 Touchscreen

On the touchscreen itself there are also buttons, to drive or steer. These buttons must be visible and intuitive to the user. For the throttler there are arrows visible in the screen, these arrows must be obvious for what they are for and be clear.

4.2 User Testing

To get to all of these HCI information we did have some thoughts of our own. But these are in most cases not enough, because we are the creators of game and know the game well. For us it is obvious, but for another person it may be really strange. This is why we also did user tests. When we did these user tests we let them play the game and afterwards we asked them questions, from a premade questionsheet. All of the answers of the user tests we have analysts and we have took them with us as useful feedback.

5 Evaluation of the functional modules and the Product in its entirety

5.1 Evalutaion of the functional modules

In the software system there are several components which are put together, or even apart, based on their responsibilities. These components have each a main responsibility, but still communicate with other components. In this section the components will be evaluated.

5.1.1 Behaviours

This component exists out of classes which handle the behaviour of the car and the track that is created. In the sense that a car should collide smoothly when colliding with the bounds of the track. The component is mainly responsible for the events of the car. The elements associated with the car, like position, rotation and the notification if a car has finished are all handled within this component. The component has also been named 'Behaviours' since it contains the classes and/or scripts which handle behaviour of GameObjects.

5.1.2 Car

The Car-component consists of classes associated with a Car-object. It contains a Car -and Player-class, where the Player-class can be split up into a Throttler and a Driver. A Car consists of a maximum of two Players. The Car has a behaviour attached to it, but does not handle the actions which should be done by the behaviour. The Driver -and Throttler-classes are focused on their functionalities and perform actions based on their Role in the game. In this component there has been carefully looked so that it does not violate the Single Responsibility Principle.

5.1.3 Controllers

Like the name suggests, this component exists of the Controllers that have been made for the software product. To take off some functionalities from, for example the Network, this component has been made. In this way it was possible to have more control over what happens and have the classes withing the components focus on their responsibility.

5.1.4 GraphicalUI

This is the largest component of all, but has been created in such a way that it is possible to create additional GUI-elements in the future. The component's main focus is the Graphical User Interface. No actions or events are triggered within the component which have nothing to do with the GUI. By splitting up the GUI in smaller parts it has also given more control and maintainability. By use of the GUI-parts it has been simple to adjust, tweak and improve the elements within the component.

5.1.5 Main

This component has not a very active responsibility, but has an overcoupling view of the other components. The MainScript-class within the component contains a Game-object. By passing through

the MainScript-class it is possible to gain information about the Game. Moreover, the other components have the ability to reach several attributes in the Main-component to get fast and accurate information.

5.1.6 NetworkManager

The NetworkManager-component manages everything with and around the Network. By means of the Network, it is meant that a Server and Client are split up. The server here controls which player should get what role. It also makes it possible to spawn the players and everything around it. The Client-class is being attached to a Player, since every Client is in fact a new Player.

5.1.7 Utilites

This component contains classes with either constant data which does not change in the software or classes which contain methods that are used very often. By putting much used methods together in appropriate classes, it was much easier to gain access to them and reduce code duplication.

5.2 Failure Analysis

Analyzing possible failures of our system has been crucial throughout the whole duration of the project. From the moment we picked our game concept (and even before that), we had to do a lot of thorough research on whether Android devices would be able to offer a real-time online multiplayer experience in terms of their hardware capabilities.
We also decided to use Unity as a development tool, because we figured that Unity offered a lot of freedom and potential for creating a game in a rather short time span. As the first few weeks passed by we found out that Unity could at some points be rather limiting instead of simplifying. Two of the major limitations that were points that could, and still might, make our system fail will be discussed below.

One of the first possible things that could fail is the fact that the server our game creates has to communicate with Unity's 'Master Server'. Our clients (the Android devices) also have to communicate with this Master Server. The problem with this is that we are dependent of the fact whether Unity has their Master Server running at all times or not. Something that happened not earlier than last week is that Unity's Master Server went offline for two days. This meant that we were unable to test or play our game in those two days. As a solution we changed the code in such a way that a server can now be run locally. However, because of the fact that this happened last week, we have not been able to properly test our solution to this problem yet. This means that the server might fail at some point, although we think the odds that this happens are not big enough to pose a problem.

The second feature of Unity that is perceptible to failure is the use of Remote Procedure Calls (RPC). As explained in the Concurrency paragraph of our Emergent Architecture Design document, our first idea to implement network play was to use state synchronization. This is a feature in Unity that allows for unreliable state synchronization of Unity GameObjects. This posed a problem in combination with our main game idea (two players controlling one car) and so we were forced to come up with a different idea: using RPC. Sadly enough, RPC can cause congestion if too many calls are made at the same time.

Congestion would cause latency issues and this would completely break the game experience at any time. We have tested our game pretty thoroughly with users and by playing it ourselves and we have noticed that while playing on certain networks RPC congestion actually poses a problem. We have noted this as a problem that we should try to fix or at least try to reduce in the future, for congestion might be the main reason for our system to fail.

To conclude, our system is perceptible to failure in a few ways. These failures might be caused by Unity on one hand, but we have done our best to work around these problems as to reduce the chance of failure to a minimum.

6 Outlook

In this section we will discus the future of our game.

6.1 Improvements

There are a few improvements our game should get. One of those improvements is that the network capabilities of the game can be improved. Also the game should have a random track generator. This was the people do not learn the track, thus the races become more interesting. The game should also have a better Graphical user interface. The game should also run without first starting the server on a different device.

6.2 Speculations

If our game where to release at the end of this project, we do not think it would gain much popularity. In the first place we do not think that people would play with strangers in a waiting room for example. In the second place because there are a lot of things to improve and there is much to add to the game, for example: leaderboards with competetive tracks.

6.3 Achieving the improvements and additional features

There are a few options for this matter: We could work on it by our selfs after the project, give it to the TUDelft, let students from a gaming study work on it and last let a company work on it.

6.3.1 Keep working on it

This is a scenario that is not likely to happen, because we have our own jobs and we have also study. To keep working on it would be very time consuming.

6.3.2 TUDelft

This is a scenario that could work. We give it to the TUDelft and the say to a group of people that they must give it more additional features. This can be done for a course, or maybe even for a bachelor end project.

6.3.3 Other students

This is also a possible scenario. The TUDelft will give students who actually study a gaming related subject the game, under the supervision of the TUDelft. This will be a great learning process for them.

6.3.4 Company

This is also a scenario that is not likely to happen, because companies want to make their own game. It will be more like that they buy/get the concept of the game and then make it from scratch.