TU Delft Delft University of Technology

Context Project
Computer Games Group 8

Rob van Bekkum	rvanbekkum	4210816
Thijs Brands	tlmbrands	4247132
Soheil Jahanshahi	sjahanshahi	4127617
Aidan Mauricio	amauricio	4195175
Joost van Oorschot	jjevanoorschot	4220471

Final Report

26th June 2014

Contents

1 Introduction

At the beginning of the project we were given a problem to design a solution for. The problem was that in many circumstances, a whole bunch of people have to stay together in the same place for a considerable amount of time. Often, however, these people do not even know each other, but they still stay together at the same location. A game would make some of these situations bearable, e.g. when there is waiting time involved. After playing a game in such a situation it can also give the user a sense of e.g. achievement when a user gets a new high score or solves a difficult puzzle.
For our project we were assigned to design a game to entertain such large groups of people and fulfill our end-user's need. The requirements from the end-user were that the game must be original, i.e. the game concepts must be unique. The game must also be creative, fun to play and also let people interact with each other e.g. through verbal communication with each other.
As a solution to this context problem we developed a game called Taxi Trouble which will be introduced in the next chapter.

This document is about the developed, implemented and validated software product. Chapter 2 provides an overview of the developed and implemented software product. Chapter 3 describes the developed functionalities. An HCI module that was realized for the user interaction with the developed solution is provided in chapter 4. In chapter 5 an evaluation of the functional modules and the product in its entirety, including the failure analysis is provided. And at the end of the document in chapter 6 an outlook is provided regarding the possible improvements in the future and the strategy to achieve these improvements.

2 Overview

For the past nine weeks our game development team has been developing a competitive, cooperative and interactive computer game designated with the name Taxi Trouble. In this chapter an overview is given of the developed product and its implemented functionalities. A more elaborate description of the functionalities of the game is given in the succeeding chapter.
The game Taxi Trouble is a game which can be played together by 4 to 8 players at the same time. The game is played on an Android device and uses the Google Play Game Services to allow people to connect with each other and play a game with each other. A player can select the amount of players that he or she wants to play a game with after which a waiting room is initiated. When all players are connected, the game starts and teams are formed randomly. Each team consists of two players where one gets the role of taxi-driver and the other gets the role of navigator. The taxi-driver of each team drives around in a large city and has only a limited view of the city. Therefore the role of navigator is introduced that has a complete overview of the citymap and needs to guide its partner to the right locations. For a team to win the game the players need to communicate very well to get the highest score when the time limit has exceeded.
When the game starts the main menu of the game is shown in which the player can choose to either press the start button for starting a new game or the leaderboard button to check out the leaderboard. To start a game the player logs in with his/her Google Account and chooses the amount of players with which the player wants to play. Also a player can invite other players with their Google Account. When everyone has joined the game the game starts and the roles are divided. As explained earlier each team consists of a taxi-driver and navigator which have to communicate very well to win the game. Each driver has a taxi which he or she can steer and accelerate. For the team to score points there are passengers in the game which can be picked-up by a taxi and afterwards dropped off with a certain time-limit at its destination. As well taxis can steal passengers from each other while bringing them to their destination. Also for a team to increase their chance to win the game there are power-ups in the city that can be picked up by the team's taxi and later on activated by a navigator. Further sound effects are being played when certain events in the game occur. Finally a head-up display(hud) is displayed on screen to show game state info like the team score, timer and team identifier. When the game ends each team's final score is submitted to the leaderboard.

3 Functionalities

In this section the important functionalities and key features that are developed for Taxi Trouble will be explained. Each functionally is an important part of the game and enhances the experience of the user. Functionalities are classified under four categories: the visual aspect, the game model, the sound and the multiplayer.

Visual Aspect

The visual aspect of the game relates to how the game appears to the players. It focuses on the appearance of the main game world and the components that are used to enhance the game for the player.

The driver's screen has an myopic 2D view of the map. It makes the driver's sight on the map limited so that it makes it impossible to win the game without the navigator's cooperation. There are touchscreen controls that can be used by the driver to control the taxi's acceleration and steering angle.

The navigator's screen has a 2D top-down view of the map. The navigator can zoom and pan to explore the game world. The navigator can use his overview of the game to guide the taxi driver. There is a button provided on the navigator's screen, which can be used to activate power-ups that are picked up by the driver.

The head-up display is part of the graphical user interface. It is used to show information to the player during gameplay. The following information is represented textually:

- The team number, which indicates to which team the player belongs.
- A scoreboard that represents the score achieved by the team.
- The time left until the game ends.
- The time remaining for the driver to drop off a passenger to his/her destination.

The main menu is the first interactive screen shown after starting the game. It provides two buttons: a play button and a leaderboard button. The player can start the game by pressing play. The player can view the top scores of the current day or week, or the all-time top scores by pressing the leaderboard button.

Game model

The game model lies at the heart of the design of the game. The model consists of all entities that populate the game. In this section we will give an overview of the most important entities that make up Taxi Trouble's model.

Taxi
: The taxi entity is the most important entity in the game, as one would guess by Taxi Trouble's name. A taxi contains a body and wheels, which are separate entities on their own. Each taxi also has a team assigned to it. Furthermore, the taxi entity contains a set of properties such as the passenger that is in the taxi and its current speed, maximum speed, angle, and acceleration.

Team
: A team is made of a driver and a navigator. Each team has a corresponding taxi and the team contains the state that the team is in. The state of the team is made up of the team score, whether the team owns a power-up and the appearance of the team.

Passenger
: Passengers are spawned in random locations on the map. They are meant to be picked up by the driver and to be dropped off at their destinations before a timer runs out. Whenever a passenger is dropped off to his destination, a new passenger is spawned at a random location on the map. Passengers can be stolen by other taxi while they are on their way to their destination.

Power-ups

Power-ups are randomly spawned on the map and they give a driver a temporal gameplay advantage. The driver can pick up a power-up, after which the navigator has to activate the power-up. There are three types of power-ups, which are invincibility, a speed boost, and an increase for the drop off timer. The invincibility power-up prevents a taxi against passenger stealing. The speed boost power-up increases a taxi's speed for ten seconds. Lastly, the timer increasing power-up grants an extra ten seconds of drop-off time to a taxi.

Sound

Having good sound effects can create a positive impact on user's experience. Sound effects are designed to immerse the player into the virtual game world, making the game more entertaining and satisfying.
In the game, sound effects are used when certain events take place. These events are car collision, passenger pick up and drop off, power up pick up and activation, and passenger stealing.

Multiplayer

One of the primary requirements for the design of a multiplayer game is to develop a sense of shared space among the players. The architecture that is chosen is the peer to peer(p2p) architecture.
A multiplayer game can be started by inviting friends or choosing to be joined with random players. Once enough players have joined a game room, the game will commence. The game can be played with four, six our eight players. During the game, the illusion of a shared space is maintained through network messages which are being broadcasted between players. These messages contain information about the game model and tell the receiving client to update its own model accordingly.

4 Interaction Design

After seven weeks of development we conducted a user test among visitors of the TU Delft Science Center. This section will describe the Interaction Design aspects of the usability evaluation that we conducted there. Firstly, we will discuss our evaluation methods and what part of the system we tested. Secondly, we will give an overview of how the testing was done, by discussing the setting and location of the user test, a description of the users that tested the game, and the methods that we used during the test. Lastly, we will give a summary of our findings.

For the usability evaluation, we have chosen to use the empirical 'experiment' practice. An experiment was best suited for our user test, because we wanted to observe users interact with the game in order to discover flaws in the game, and to identify gameplay elements that were not considered fun. A big aspect of Taxi Trouble is communication, which resulted in users practicing Think Aloud without being asked by us. This was very useful for our evaluation of the user test, as it gave us a lot of information about how the users perceived the game.

The usability evaluation was done right before the release of the beta release of Taxi Trouble, so that we could incorporate our findings into the beta version. This means that we evaluated the usability of the alpha version of Taxi Trouble. The alpha version was missing a lot of features compared to the final version, but it was stable enough to conduct a user test with.

For the setting of the user test we chose the TU Delft Science Center. The Science Center is a good location for testing because it receives a lot of visitors that fit into the user demographic of Taxi Trouble, and there are sufficient facilities for conducting a user test. We were appointed a large, open room, with two racing chairs in the center of the room. The racing chairs made the user test more fun for the younger users, and fit the theme of Taxi Trouble well.

The users that tested Taxi Trouble ranged in age from 8 to 24. We took great care during the user tests with the younger users. The parents of the younger users were present during the user tests at all times and we made sure to mention that the users could stop the test at any time.

The user test was performed in groups of two users. At the start of the test we asked the users to sit next to each-other and to pick a role in the game (either navigator or driver). Then, we let the users play the game for about five minutes. During this time we logged what the users said to each-other and we did not interact with the users. After playing the game, we interviewed the users about the game, asking open questions about the art style of the game, the controllability of the game, and about the gameplay.

We did a formative evaluation of the test resuslts and we will give a summary of our findings in this section. Overall, the users were very pleased with our game. Even though all users stated that they found the controllability of the game sufficient, we noticed that most users struggled to understand what the buttons to control the car meant, and they had to take some time to figure that out by trial-and-error. This resulted in us creating buttons that mimicked the look of an actual gas pedal and brake.

During the evaluation it became apparent that users did not immediately understand the navigator view. To make the navigator screen more understandable, we created a HUD that shows to which team the navigator belongs.

Another thing that we found out, is that when a taxi is carrying a passenger, it is hard for the users to distinguish the front of the taxi from the back. In response to this we created new sprites for the taxi that clearly show the difference between the front and back of the car.

The results of the usability evaluation helped us identify a number of flaws, which allowed us to fix them. Additionally, we were able to pinpoint which gameplay features the users liked and which features they disliked.

5 Evaluation of Functional Modules

5.1 User evaluation

Aspect	Average score (1 to 5)
Fun	4.25
Understandability	3.42
Visual presentation	4

The table above shows the average score we got from users for three of our most important aspects of the game. From the table it becomes apparent that the understandability of the game leaves something to be desired. Users did not know what to do at the start of the game. Some started driving, but both driver and navigator didn't know where they should drive to. After a short period of time we stepped in to explain how the game works to those who were truly lost. Many users suggested making a short tutorial for new players. Unfortunately this tutorial has not made the current release of the game.

As soon as users understood how to play the game, drivers and navigators started to communicate and a lot of laughter was to be heard. We are very pleased with the score we obtained from our users for the fun aspect of the game. Most users were pleased with the overall visual presentation of our game. We were happy to hear this, as a significant amount of work went in creating the world of Taxi Trouble.

5.2 Evaluation of individual functional modules

We will now give a brief evaluation on our functional modules. It should be noted that some subcomponents have been left out of the description of the modules and their components. This is for the sake of overview.

5.2.1 GUIs

Our game has two main graphical user interfaces, one for the driver and one for the navigator. The GUIs consist of the following components:

ViewObserver	Takes care of basic rendering functionality for both GUIs.
DriverScreen	Takes care of special rendering functionality for the driver GUI.
DriverControlsUI	Provides the input buttons needed for the DriverScreen.
DriverControls	Provides functionality for the buttons of the DriverControlsUI.
NavigatorScreen	Takes care of special rendering functionality for the navigator GUI.
NavigatorControlsUI	Provides the input buttons needed for the NavigatorScreen.
NavigatorControls	Provides functionality for the buttons of the NavigatorControlsUI.

This module consists of the view and controller components of our MVC architecture. The DriverScreen and NavigatorScreen are both extensions of the abstract ViewObserver. They use their respective Controllers and ControllerUIs to enable the user to give inputs to the game's model. The game's model is in turn used by the ViewObserver to render the game.

During user tests it became clear that the navigator's GUI was intuitive to use for almost all users. However, the driver's GUI had varying results. While most users were able to use the GUI as intended, a significant amount did not have the hand-eye coordination that is required. A probable cause suggested by the results of our interviews is that the users that had difficulties using the GUI rarely played games in general.

5.2.2 Entities

All physical objects in the game world are subclasses of the abstract Entity class. Entities can interact with each other when colliding with each other. The detection of collisions is a task of the Management module. Our game has the following Entities:

Taxi	A taxi that is controlled by the driver of a team.
Passenger	A passenger that can be picked up by a taxi.
Destination	A destination that at which a passenger has to be dropped off by A taxi.
PowerUp	A power-up that can be picked up by a taxi.
SolidBox	A solid object no taxis can drive through. This is used to create walls for example.

During user tests almost every user was able to recognize and understand all entities in the game. Power-ups were an exception to this rule. Users that played games on a regular basis were often quick to recognize and use power-ups. However, most users that didn't game on a regular basis completely missed the presence of power-ups. Most of these users noticed the power-ups after playing for 3-4 minutes. This is also something that can be helped by having a tutorial in the game. Once users picked up a power-up most of them were able to understand what it would do on activation, which made it easy for them to activate the power-up at the right moment. Interaction with and between other entities was experienced as logical and self-explanatory.

5.2.3 Game and Entity management

The entities in the game are managed by a set of specialized classes:

Spawner	Responsible for creating and removing entities.
CollisionDetector	Responsible for invoking interaction between colliding entities, if any.
GameWorld	Notifies the Spawner when a new entity is needed.

This module combined with the Entities module make up the model part of our MVC architecture. The GameWorld is the backbone of the game's management. It starts the correct GUI and controls the spawning behavior of the Spawner. The CollisionDetector manages the interaction between colliding entities. This interaction often results in the removal of one of the colliding entities. This is done by telling the Spawner to despawn an entity from the world.

5.2.4 Multiplayer implementation

The multiplayer implementation enables the game to actually be played with other people. The implementation exists of the following classes:

AndroidLauncher	Used to create and manage multiplayer connections.
AndroidMultiplayerAdapter	Used to send information to other clients playing the game.
MessageAdapter	Handles incoming messages and alters the game world accordingly.

The AndroidLauncher invokes methods from the Google Play Services Library to connect players to each other. Once a connection with all clients is established, one client will be assigned the role of "host". The host will act as a central authority during the game. Clients are only allowed to send the state of their own car and the activation of acquired power-ups. All other events (e.g. picking up a passenger or power-up) can only be sent by the host. This basically means that the state of the host's game is the true state of the game. Clients will alter their game's state to match that of the host's game.

During user tests it became clear that the multiplayer implementation does have a shortcoming. While the latency of the game is generally within playable limits, it sometimes tends to have a brief moment of increased latency, a so called "lagspike". Lagspikes are extremely annoying to the user because it makes the game feel unresponsive for a short duration. This can cause the user to lose (part of) the game while they should have won. Lagspikes are mostly caused by unstable internet connections. We have tried some techniques to reduce the impact of lagspikes (e.g. client-side prediction) but these proved insufficient.

5.3 Failure analysis

During the project, we encountered a number of critical failures that we have fixed. We will discuss some of these in the upcoming section. Some lesser failures are still unsolved and will be described in section 5.3.2.

5.3.1 Encountered problems

One of the most problematic failures we encountered seemed to crash the game at random moments in the game. Upon looking for a stack trace, the only error message found was a single line indicating a segmentation violation. After some research and experimenting, we found the cause of this failure. When adding or removing entities to the game world while the game world was performing a physics step, the engine of our game would try to access memory outside the process's address space. We still do not know why the engine does this, but we suspect that the engine's developer made a mistake when porting it from C to Java. We made sure that the game will always wait until it is done with its physics step before entities are added or removed.

An even more problematic failure caused the game not to crash, but to freeze. Because the game did not crash, the debugger did not show us any useful information. Because of the freezing of the game, we hypothesized that the failure was due to a deadlock. We searched the LibGDX community and found conformation for our hypothesis. Because our MessageAdapter, operated by the thread responsible for network events, executed methods in the LibGDX game world, that should only be accessed by the LibGDX game thread, a deadlock could occur when both threads tried to perform certain operations. The solution for this failure was to, instead of executing the method, let the MessageAdapter pass a method call as a runnable to the LibGDX game thread. After applying these changes the failure has not occurred again.

5.3.2 Unsolved failures

A few problems that cause our game to fail are still unsolved. An example is the lagspike problem described in section 5.2.4. Although this is not a critical failure, the game can become unplayable when many lagspikes occur

during the game.

A more important failure sometimes happens when pressing the play button in the main menu more than once, without giving the game time to load. This can cause the network connection of the game to fail at the start of the game. We don't think this issue is really hard to solve (disabling the button once it has been pressed will probably do wonders) but we found this bug to have a relatively low priority when compared to other problems, such as the ones described in the previous section.

The most important failure of our game is the lack of anti-cheat measures. The game does not contain any way to validate messages from other clients and the messages are also unencrypted. If someone were to implement a "hacked" client, he could most likely influence the game by sending messages that are beneficial to him, or crash the game by flooding the network with messages. Google Play Services offers some form of security by only allowing applications signed by our digital key to connect to their services, but we doubt this will be enough to keep really eager hackers out.

6 Outlook

After delivering a product that satisfies our expectations and our end user's needs there is always room for improvement. Taxi Trouble is playable and lots of fun to play, but it still contains a few bugs. Also there are tons of features that didn't make the first release of the game which could still be implemented in the future. Implementing these will greatly increase the quality and the experience of the user who plays the game.

Bugs

A bug is a flaw in the software. The two bugs that are present in Taxi Trouble are the following:

1. When the host of the game locks his phone or minimizes the game all messages stop getting through meaning that most of the features of the game won't work until the host resumes the game again.
2. If you want to restart the game to play again you need to completely kill the app or the game will not start correctly anymore. The resources of the game will not load correctly.

Solving these bugs will greatly increase the quality and the playability of the game. A few strategies to solve these bugs are:

1. Implementing a pause function. When someone lowers or locks his phone the game should pause. Pausing is not always the answer here. If someone locks his phone or lowers the game they might have had to leave, so implementing a leave function is also viable. But if the host leaves the game then a function should be implemented to switch hosts otherwise no messages will be getting through and the game stops working.
2. A viable strategy for this is implementing a restart function that returns you to the start menu of the game or to the lobby.

Features

Implementing a new feature's difficulty depends on the structure of your software and how much the developer understands this structure. That being said a new developer that can understand the structure of Taxi Trouble will have no problem implementing a few of the features mentioned below. A few ideas of features that we had are:

1. Choosing who you want to be in team with. Right now the game has an auto pick feature which randomly assigns people in a team.

2. Adding more collidable objects e.g. traffic, walking pedestrians, cones, etc. Adding more collidables to the map will increase the challenge of the game and the focus a player has to the game.

3. Cops. Adding this feature will make the game more immersive and realistic.

4. Health for the taxi. Colliding with objects will decrease the health of the taxi. When the taxi's health is decreased there can be a lot of side-effects e.g. the taxi's speed gets decreased, turning radius is increased, etc.

5. More powerups. A few new powerups that can be implemented are e.g. increasing the health of the taxi, slowing down other taxis, calling cops on other taxis, etc.

Adding these new features will increase the experience a user has with the game and will ultimately make it more fun and challenging to play. A few ways to implement these features are:

1. Implement a choose a team function which puts you in a lobby where a user can switch between teams.

2. Collidables can be fixed on the map or their spawnpoints can be defined on the map. After that you need to define an event that is triggered when a collision is detected for them. If you want to add traffic one must add some kind of agent that can control the traffic so that they don't collide in any objects and stay on the streets. For walking pedestrians one can extend the passenger class and add a walking animation for this. The same counts here for pedestrians as for traffic. The pedestrians may not collide with other entities and they have to stay off the streets.

3. Just like implementing traffic, one must define an agent to control the cops so that they can chase after a taxi.

4. Everytime a taxi collides with an entity a function must be called to decrease the taxi's health.

5. Powerups were implemented using the strategy pattern, so implementing these is just applying the Powerup-Behaviour to the new powerup and adding its new features.

As you can see, implementing some of these features can be pretty easy to do, but there are also a few where it is pretty difficult to implement, e.g. agents for traffic and cops. And a lot of these features depend on one another which is why we did not have enough time to add these to our first release.