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

Emergent Architecture Design
Final Version

Contents

1 Introduction

This document has been written with the goal of providing insight in the architecture design of our software solution. The architecture design will be subjected to change frequently due to the need for additional features or problems encountered with the current design. This document will be updated accordingly when these changes are made.

1.1 Design goals

This section will contain the goals we keep in mind while making decisions about our architecture design. We believe that these goals are needed to create a product we can be proud of.

Loose Coupling

We have divided our classes mostly over two categories: model and view. All classes that represent the game world or objects of the game world are part of the model. All classes that construct a way to view a part of the game world are part of the view category. One of our design goals was to keep coupling between these two categories as loose as possible. This way we can easily make changes to the game world our implement a new view without having to adjust a lot of classes.

High Cohesion

It is very common that the strive for loose coupling is accompanied with the desire for high cohesion. We are no exception and want to keep cohesion as high as possible to ensure robustness and understandabillity of the code. This is why we always discuss where in the software we should add new pieces of functionality, before actually implementing.

Quality Code

We strive for our code to have a certain level of quality. Writing good code makes it easier to expand the working prototype without having to fix emerging errors in the existing code. We want our code to approach the Exemplary Source Code Quality criteria as described in the Software Engineering Rubrics (2014). This implies proper use of design patterns, very few design flaws and proper use of all software engineering principles.

Working Prototype

We want to have a new working prototype at the end of every week. This will ensure that the features implemented in that week actually work when they are all integrated in one prototype. It also gives us the opportunity to show our client early versions of our work on which we can get feedback. This feedback loop helps us make the product the client actually wants. The working prototype also gives an indication of our current development stage and can give our client a feeling of progress.

2 Software architecture views

2.1 Subsystem decomposition
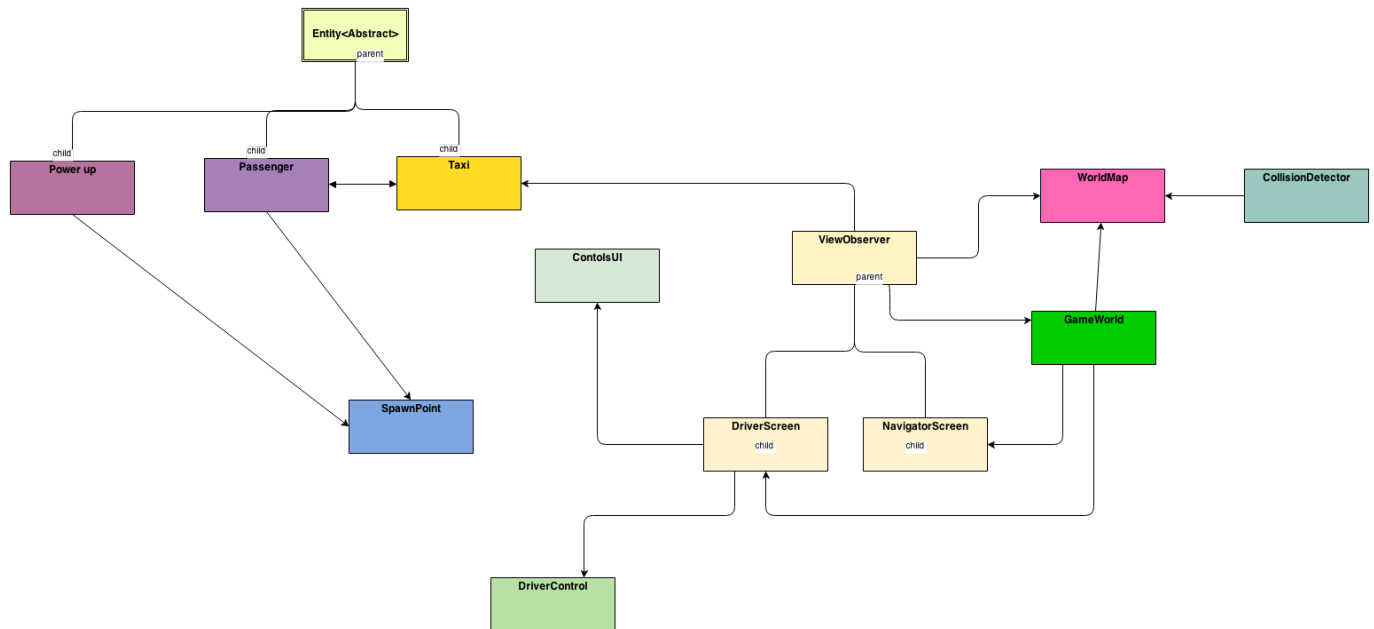
    The architecture of the system is divided into subsystems. These subsystems are the Game Model, the Driver Interface, the Navigator Interface and the Network Interface. All subsytems will be explained in this section.

The Game Model
    The Game Model consists of the game world and all data related to the game. The actual game takes place in this subsystem. All other subsystems are interfaces that are used to alter the Game Model.

The Driver Interface
    This is one of the two interfaces that will be directly used by the players to interact with the Game Model. The Driver Interface enables the user to see the car and its direct surroundings. It also gives the user the means to control the car (steering and acceleration).

The Navigator Interface
    The second way for players to interact with the Game Model is the Navigator Interface. This interface gives the user an overview of the game world in the form of a map. This enables the user of this interface to (verbally) guide the user using the Driver Interface through the game world. The Navigator Interface can be used to interact with the Game Model through the activation of powerups, which will alter specific parts of the game world.

The Network Interface
    This interface is used to connect players with each other. It is responsible for the connection of the two players within each team as well as the connection between all the teams. The Network Interface is also responsible for the concurrency between all Game Models.

2.2 Hardware/software Mapping

Taxi Trouble is a real-time multi-player game where multiple users can join and interact with the game instantaneously using their Android devices. The game will start in the main menu where the user will choose whether he/she wants to start a new game or to view the leaderboard top scores of the game. When the user chooses to play a new game, he/she will get a role of either taxi driver or navigator. These roles are assigned randomly at the beginning of the match.
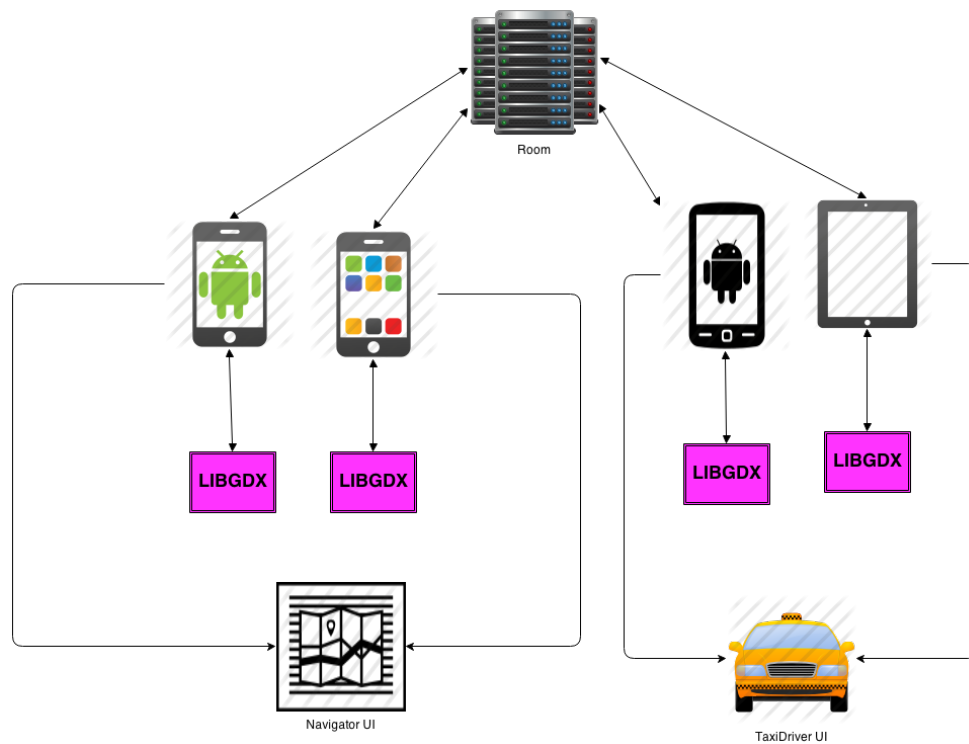
2.3 Persistent Data Management

The only persistent data in our game is the global leaderboard. This data is automatically handled by the Google Play Services API and stored on the Google Play Services Servers.

2.4 Network Architecture

Our network architecture makes use of the Google Play Services (GPS) API. This API sets up a peer to peer network between all clients playing the game. After establishing the peer to peer network, one of the clients is chosen to act as a server. The server will act as an authority, its state will be the "true" state of the game. Note that the client with the server role is still to be considered a functional client, although this client does have the advantage of playing directly on the server world. The table below lists all events that are sent over the network during normal gameplay.

Event	Sent by	Client-side prediction	notes
Car movement	Client	Yes	Car movement is send by the client driving the car.
Passenger pickup / dropoff / stealing	Server	Yes	To prevent passengers being assigned to other taxis at the same time, only the server can send these events
Powerup pickup	Server	Yes	To prevent power-ups being assigned to different teams at the same time, only the server can send this event.
Powerup usage	Client	If needed	We will implement client-side prediction and server reconciliation for powerups that are in need of this.
Create passenger / powerup	Server	No	New instances can only be spawned by the server, the clients should not try to predict this, as most spawning is random.

If an event makes use of client-side prediction, the client (visually) simulates the event before receiving the event over the network. Upon receiving the actual event, the client can finalize and/or correct the simulated event.

2.5 Concurrency

The only real type of concurrency we have to deal with is the concurrency of incomming messages. For sending critical game data, we use GPS's reliable messaging protocol which insures delivery, integrity and correct order of receival. For sending non-critical game data (ie. The location of a taxi), we use GPS's unreliable messaging protocol. One might wonder what happens when two messages are received in such a way that two taxis are at the same location. The LibGDX game engine will handle this as a normal collide event and correct the location of the taxis accordingly.

3 Glossary

- **Database** : A database is an organized collection of data.

- **Interface** : An interface is a shared boundary across which two separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans and combinations of these.

- **Prototype** : A prototype is an early sample, model or release of a product built to test a concept or process or to act as a thing to be replicated or learned from.

- **Software Engineering Rubrics** is a document consisting of the different criteria for evaluation of our product. The document can be found as well in the references at the end of this document. The document implies proper use of design patterns, very few design flaws and proper use of all software engineering principles.

- **Subsystem Decomposition**: Also known as factoring, refers to the process by which a complex problem or system is broken down into parts that are easier to conceive, understand, program, and maintain.

- **LibGDX**: LibGDX is a game-development application framework written in the Java programming language with some C and C++ components for performance dependent code.

- **UI**: The user interface, in the industrial design field of human–machine interaction, is the space where interaction between humans and machines occurs.

4 References

Software Engineering Rubrics : https://blackboard.tudelft.nl/bbcswebdav/pid-2235317-dt-content-rid-7624672_2/courses/30183-131404/SE%20Rubrics.pdf

Google Play Services : https://developers.google.com/games/services/common/concepts/realtimeMultiplayer