Implementation Documentation

The Pendulum Paradox

GROUP 22

Rune Strøm Brekke
Tore Dybdahl
Andrea Falk Lind
Henning Einar Luick
Matěj Mňouček
Simon Smeets

COTS: Android Studio, LibGDX and Google Play Services
Primary Attribute: Modifiability
Secondary Attributes: Performance, Usability and Availability

May 12, 2019

Contents

List of Figures

List of Tables

1 Introduction

1.1 Project Description

This implementation documentation is written as part of a project in the course TDT4240 Software Architecture at the Norwegian University of Science and Technology. The project work will be conducted during the spring semester 2019. The participants are Rune Strøm Brekke, Tore Dybdahl, Andrea Falk Lind, Henning Einar Luick, Matěj Mňouček and Simon Smeets.

The goal of this project is to make a functional multiplayer game that can run on a smartphone. Our group use Android as our game platform, and the we have implemented a multiplayer real-time game. In our work, we will use several different architecture and design patterns. During the process, we will focus on designing the game, implementing it correctly, evaluate it and conduct testing of the result. We focus on making a high quality game with respect to important qualities in software architecture. Our targets are described in quality attribute targets are modifiability, performance, usability, and availability, thoroughly described in the requirement documentation.

The project officially started 12th of February and has continued for approximately 12 weeks, with end on the 12th of May. During these weeks, the project was divided into three phases: "Requirements and Architecture", "Evaluation (ATAM)" and "Design, Implementing and Testing". What is more, we also contributed with feedback to other groups taking the same course, and we received feedback.

In the current phase, Implementation, we the focus is on implementing the architecture specified in the architectural documentation, and doing this in a way which will fulfil our requirements as specified in the requirement documentation. The team will write the code, combine the different classes, and perform several tests in order to see if the architecture fulfils our requirements.

1.2 Game Description

The Pendulum Paradox is a platform game, and therefore a subgenre of action game. The player controls a sprite object, and advances through different levels, starting in a forest. During the game, the player faces several obstacles and enemies will attack and attempt to kill him. The basics of the game dynamics are borrowed and inspired from the games Jazz Jackrabbit, Super Mario Bros., and A Great Adventure.

The game can be played in single player and multiplayer mode over a real time network. During each level the two players will have sci-fi characters that are lost in time.

Both players are equipped with a basic shooting device in order to defend themselves.

The players move through the game world, where they get points for shooting enemies and collecting gears. They can also collect artefacts that can improve their survival skills. The artifact lasts for a specified amount of time after the player has collected it.

Both players have to survive to the end of the current level in order to advance to the next one. If one of the players loses all of his lives, both players will die. The players lose lives when they collide with an enemy or fall in the water, where the last will result in total death. The game works as an endless runner game, and will proceed through levels until one player dies.

The game is played on a smartphone. Buttons in the left and right corner of the screen will provide functionality for the player to move his play character left and right, and give him the possibility to jump and shoot.

Figure 1 to 3 show screenshots from three different platform games. These are all used as inspiration for our game.

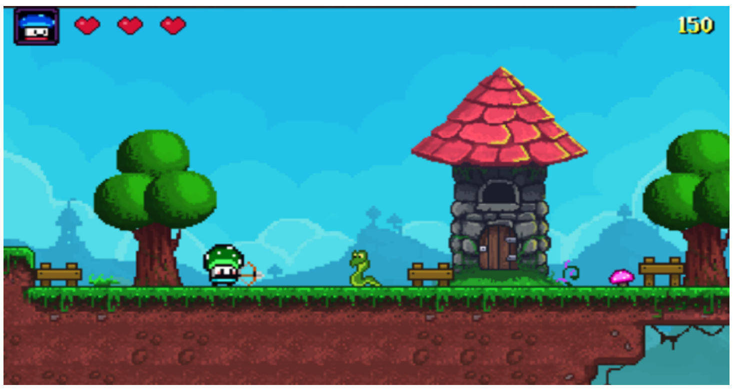

Figure 1: Screenshot from the platform game A Great Adventure

Figure 2: Screenshot from the platform game Super Mario Bros.

Figure 3: Screenshot from the platform game Jazz Jackrabbit

1.3 Document Structure

This document contains a description of the implementation of the architecture of our game. The primary focus is on the implementation process. Chapter 2 contains details regarding design and implementation. We present the platforms used, as well as an overview of the architectural and design patterns used. Furthermore, a class-diagram is included, with a description of each of the classes. Chapter 3 contains a user manual with how to install and play the game. Test reports are included in chapter 4. Here we give the results of the test performed for all requirements specified in the requirement documentation. Chapter 5 describes the architectural tactics used and how the patterns were implemented, and thereby the implementation's relationship to the architecture, as described in the architectural documentation. We finish with chapter 6 including problems during both the documentation process and during the implementation process, as well as what we have learned and take with us to future projects.

2 Design and Implementation Details

2.1 Implementation

The Pendulum Paradox is implemented with the use of Android Studio and Java, libGDX, GitHub, and Google Play services. COTS are thoroughly documented in the requirement documentation, section 4, and we will here give a brief overview.

Android Studio and Java
Android studio is the official integrated development environment for developing Android applications (Android Developer, 2019). It has a folder structure making it easy to navigate, and it provides an emulator to run the code. One of the languages that Android studio supports is Java, which is the programming language used in our game implementation. Java supports many design and architectural patterns, which has been useful for the implementation of our game. This object oriented language makes it easy to create and implement classes and interfaces in a consistent way.

libGDX
libGDX is an open source platform game development framework (LibGDX, n.d.). It provides several tools that simplifies game development. Especially, it made the creation of sprites and textures easier, and collision handling was simplified. Using libGDX also had the advantage of making it easier to handle game states, including the creation, rendering and disposal of them.

GitHub
GitHub is a version control system that keeps revisions of the code and store modifications (Brown, 2017). Every member can see what changes have been made, and update the changes that they selves make. The use of GitHub was very convenient due to the fact that we were six people working on implementing the same game. It made collaboration easier, and every one could keep up to date with changes made to the code.

Google Play Services
In order to create a real-time multiplayer game, we used the real-time multiplayer API in Google play games services. The API helps with the management of network connections, provides an UI to invite players to join, stores participants and update all players (Android Developer, 2017). This simplified the implementation of a real-time multiplayer game.

2.2 Architectural Patterns

This section presents used architectural patterns and shows how they define application structure.

Model View Presenter (MVP)
The Model View Presenter pattern is the nuts and bolts of the game. It connects together data stored in the model with views presenting them on screen. During the implementation process it was decided to use a not so common variant of MVP which relies on only one Presenter. This approach is similar to Model View Controller (MVC) but still preserves mutual interactions of the standard MVP pattern. Furthermore, the role MVP plays in terms of developed architecture is even bigger. In fact, it also defines the top level of hierarchical structure in which Model is further divided into Entities Components and Systems and Views are managed by State pattern.

Entity Component System (ECS)
As was pointed out in the previous section, Entity Component System pattern divides the model of MVP. The game mechanics are stored in the model which makes the majority of the whole implementation reside in the ECS and the model respectively. All objects in the game are in fact Entities and the Components assigned to them carry all their data. Entities with particular components are processed by various Systems which handle the actual game logic and possible interactions with game Presenter.

State
The State design pattern turned out to be really useful and, therefore, got upgraded to an almost fully-featured architectural pattern on its own. The Presenter uses it for View management. A ViewState is a class which encapsulates UI and game level combination. More precisely, it pairs BaseScreen subclasses for rendering UI and Scene subclasses for rendering game levels. One ViewState therefore renders the UI and game level at the same time. For management and transitions between views the actual State pattern (also commonly called State machine) implementation is used.

2.3 Design Patterns

This section presents implemented design patterns and discusses their role in the implementation.

2.3.1 Creational patterns

Abstract factory
The Abstract factory proved its usefulness primarily in connection with the ECS architectural pattern. Thanks to the fact that Entities mostly contain a wide variety of Components, tearing them up takes a substantial amount of effort. That is where the factory came in handy. It abstracts from the complex entity creation and provides ready to use Entities wherever in the game needed. Also possible changes in Entity creation need to happen only in one place and the whole game gets affected. The Abstract factory pattern is defined by the AbstractEntityFactory abstract class and implemented by the EntityFactory class.

Builder
The Builder pattern is closely related to the Abstract factory. The problem with complex Entity creation also partially applies to Components creation. Some of the Components contain a lot of data fields or event objects and therefore, their creation and initialization is also a bit of a challenge. Hence, most of the components possesses several fluent Builder methods which simplify their creation. They are mostly used inside of EntityFactory.

2.3.2 Behavioral patterns

Chain of responsibility
The Chain of responsibility pattern serves well for player attribute modifiers (also called Enhancements) handling. The InteractionSystem class (which is responsible for handling player's interaction with environment) is where the pattern is used. Basically, the implementation includes several different modifiers that can be chained in a list. When the handling of the first modifier is triggered the whole chain gets iterated through and all included modifiers are applied to player's attributes. The base abstract class for defining a new modifier is called Enhancement.

Observer
The Observer pattern is alpha and omega of the game event system. The implementation was inspired by the standard C# built-in event system. The pattern is used for decoupled communication of Presenter with its Views, communication of various Systems with Presenter, communication of ControlModule class with ControlSystem and also communication of Presenter with Android platform specific code used for multiplayer.

State
Aside from the use of the State pattern in the View part of MVP, the pattern also performs well in several other tasks. The game further uses it for player state management in StateSystem (which can changed by ControlSystem and is read by AnimationControlSystem) and level management in the LevelManager class.

Command
The Command pattern turned out to be real life-saver while defining routines for game controlled enemies. Together with the CommandQueue implementation, various Command classes can be queued in order to define enemy behaviour. The queues are encapsulated in ControlModule classes which make them act as autonomous input providers/generators.

2.3.3 Sequencing patterns

Game loop
The game loop is implicitly provided by LibGDX and therefore easy to use. The basic rendering operations happens in this method.

Update method
The Update method is called from the game loop which makes this concept almost identical to the Game loop pattern. However, it separates bare logic operations (such ECS update and physics world update) from rendering operations.

2.3.4 Optimization patterns

Object pool
The Object pool pattern defines a pool which maintains a collection of reusable objects. Instead of destroying old objects and creating new ones again when needed, they can be stored in object pool and reused. We use this approach for creating Entities in order to make the creation less resource intensive.

2.4 Classes

We here provide a brief description of the classes used in the game implementation. Including all the classes resulted in a quite extensive diagram *that can be viewed here*. For easier viewing we provide it here As we use

LibGDX to initiate the game, the launcher class PendumlumParadoxGame is not shown in our class diagram

Presenter
The GamePresenter contains all the main class necessary for the game to function. Upon creation it initializes all the other classes that it needs, sets EventHandlers, and runs the main game loop updating ECS, rendering the current view and synchronizing with another device in multiplayer.

View
The view package contains a ViewState class which implements the IState interface. One ViewState consists of one Screen and one Scene class, and it renders the Scene and Screen in every game loop interation. The Screens extends the abstract BaseScreen and has the UI elements in a LibGDX provided Stage. The Stage is 2D scene graph and also input processor used to register button presses. The Scene is one layer bellow the Screen and holds the game map and creates enhancements, players and enemies.

State
There are interfaces for state, transition and a state machine. IState is implemented both by the ViewState mentioned above as well as the TaggedState used in the StateComponent to keep track of the state a player or enemy Entity is in. The StateMachine class is used to hold ViewStates such as ingame and main menu, and it uses Transition to switch between the states.

Multiplayer
The multiplayer package holds a single interface that allows the app to connect to Google Play Services. It contains functions to log in with a Google account, load and send High scores, start a multiplayer game and synchronise the game states.
This interface is implemented in the Android package, by a class called MultiplayerApplication. It handles all communication with the Google Play Services API, deals with room management, synchronisation, high scores and matchmaking.

Observer
Event holds a list of EventHandlers. When an event is invoked it will toggle the handle method of all it's handlers. The EventHandler is deliberately vaguely defined in an interface with only a single handle method which is defined when a concrete handler is created elsewhere in the code. EventArgs is included for the ability to send some data along with the event invocation.

Events are used several places in the code, but one example is when one of the buttons in the Screen class gets pressed. This will cause Event to call it's invoke method. In GamePresenter, all button-press events have been assigned handlers, and these handlers' handle methods have been defined. The invoke call will then trigger the appropriate response, for example a bullet will be fired from the player if the shoot button was pressed.

Model
The model package consists of three additional packages; component, entity, and system which together is responsible for the game's logic. Our code is based on LibGDX's Ashley framework for this. Instances of Entity, which in essence are merely bags of components, are created by the EntityFactory. Each Component defines a property of the Entity that is holding it. EntitySystem processes entities and components. Finally the Engine class updates every entity system registered to it every game loop. The ComponentMapper is used to access Component of one type fast (O(1)).

Levels
LevelManager keeps track of the current level as well as a list of levels that the player can potentially reach, as well as an instance of IStateMachine to manage states and transitions.

Control
The different ControlModule classes in this package are used to control entities. When an event is triggered the corresponding handler's handle method uses a control module to execute. Control modules can be used to assert inputs from the user on the first player entity, control the second player entity based on events performed on another device in multiplayer, using the NetworkControlModule, or control an enemy using one of the AI control modules.

Command
The DelayCommand and InvokeCommand are used by the AI control modules to define actions on the enemy entities. The AI control modules have a CommandQueue which contains different delay commands to make the enemy wait, and invoke commands to make the enemy perform an action. The command queue will then iterate over the commands in the command queue.

3 User Manual

This section describes the game in detail and includes all the things needed in order to get it up an running.

3.1 Prerequisites

The game requires Android operating system with API version 19 or newer for all functionality to be available. There is also a possibility to run it in Desktop mode on Windows, Mac and Linux but multiplayer mode is not supported on these platforms.

3.2 Installation and running the game

The game APK file is available on this link: *https://bit.ly/2PW6dN1* and, therefore, can be easily downloaded and installed to your Android smartphone. The multiplayer mode also requires the Google Play Games app and an account. This app is either already installed (so you can easily log in) or the game will guide you through its download and account creation. An internet connection is also needed on startup.

In order to run the game in desktop mode you need to compile it from the source directly. The source code is available in GitHub repository: *https://github.com/matej-mnoucek/the-pendulum-paradox*. The repository contains standard Android Studio project (for more information about Android Studio refer to its documentation).

3.3 Game features and functionality

After the game is run, the Main menu appears. From the Main menu you can navigate to 6 different places. The first two buttons called **New Game** and **New Multiplayer game** start the game in Single player and Multi player mode respectively. In order to play Multi player you at first need to make sure you are already logged in with your Google Play Games account and have internet access. Clicking on the **Log in to Google** button will guide you through this process. Under the **High score** button you can find a table with results of the best players and the **Settings** button opens menu which allows you to change some basic game settings. In case you are lost, you can always find some help under the **Tutorial** button.

3.3.1 The main game mode

The game is fairly straightforward. The basic goal is to complete as many levels as possible while staying alive and get as high a score as possible. One game round runs as long as all involved players are alive. Each player has three lives but also health attribute which means one hit from an enemy might or might not lead to one life subtraction (depending on the enemy strength). Moreover, there are also defense and damage attributes that define player's resistance against attacks and damage applied to enemies. Players damage enemies by shooting at them, eventually leading to their death.

3.3.2 Controlling the player

In the Android version of the game, the player character is controlled by UI buttons. The left part of the screen contains buttons for moving left and right and the right part has buttons for shooting and jumping. The desktop version uses the arrow keys for movement and jumping and space key for shooting.

3.3.3 Levels

The game contains two medieval levels. The goal is always to reach the end of the current level. When all players finish the last level the game starts again with the first level, resulting in an endless loop. This reset to the first level does not impact the score or lives of the players, these are never reset during a playthrough.

3.3.4 Score

Players get five points for killing enemies and 10-20 points for collecting time machine parts. Scores of individual players are counted separately. The more points the better. After the player dies, its score is sent to online High score ladder managed by Google Play Game services.

3.3.5 Collectables

The game has two different types of collectable time machine parts. The first, a copper wheel, is worth 10 points while the second one, a gold wheel, is worth 20 points.

3.3.6 Enhancements

In the levels you can also find three different kinds of enhancements (also commonly known as power-ups).

- *Life enhancement (red)* - permanently adds one extra life to the player who has collected it.
- *Defense enhancement (green)* - increases defense of all players for a short period of time (15 s)
- *Damage enhancement (blue)* - increases damage of all players for a short period of time (15 s)

3.3.7 Enemies

The game contains three different kinds of enemies:

- *Smeets the Knight* - Aggressive guy that is constantly trying to kill someone with his well-forged sword. There are rumors that he came from Netherlands.
- *Tore the Ninja Boy* - Impatient being, constantly jumping around and trying avoid work. He was born in Japan but is not welcome there anymore.
- *Henning the Ninja* - A close friend of Tore. His hearing is not one of the best and that is why it is so hard to get in touch with him. You can often find him just running around in panic.

4 Test Report

The Requirement documentation establishes several functional and quality requirements. This section documents test to verify these requirements. These tests were mainly done during the third phase of the project

4.1 Functional Requirements

FR1: The user should be able to move around in the play world using transparent control buttons	
Executor	Andrea
Date	01.05.2019
Time used	1 min
Evaluation	Success
Comment	It was easy to move around in the game using buttons in the left and right bottom corners of the screen.

Table 1: FR1

FR1.1: The user should be able to move left and right using a joystick function	
Executor	Andrea
Date	01.05.2019
Time used	30 sec
Evaluation	Success
Comment	Left and right movement was easily accomplished using two buttons in the left bottom corner. It is worth noticing that "joystick" function might not be the right description, but as the game solution provides identical functionality, we still consider it a success.

Table 2: FR1.1

FR1.2: The user should be able to jump using a button	
Executor	Andrea
Date	01.05.2019
Time used	30 sec.
Evaluation	Success
Comment	Jump was accomplished using a button in the right bottom corner. Several jumps were accomplished by pressing the same button several times.

Table 3: FR1.2

FR1.3: The user should be able to shoot at enemies using a button	
Executor	Andrea
Date	01.05.2019
Time used	2 min
Evaluation	Success
Comment	Play character shot at enemies when pressing a button in the right bottom corner. One bullet was fired for each time the button was pressed

Table 4: FR1.3

FR1.4: The user should be able to use other firearms for a short while after he has collected enhancement	
Executor	Andrea
Date	01.05.2019
Time used	2 min. 30 sec.
Evaluation	Success
Comment	Regular firearm was automatically changed with an upgraded version after collecting such an upgrade. This upgrade was used for a limited time, before the regular firearm was switched back

Table 5: FR1.4

FR2: The user should be able to start a multiplayer game	
Executor	Andrea
Date	01.05.2019
Time used	3 min
Evaluation	Success
Comment	With the use of Google Play services, a multiplayer room was created and the search for a second player was started. A new multiplayer game was started after a new player was found

Table 6: FR2

FR3: The user should be notified if no other player is found	
Executor	Andrea
Date	Date
Time used	1 min. 30 sec.
Evaluation	Success
Comment	Google Play services searched for another player for a while. When none was found, a massage of this was shown on the screen

Table 7: FR3

FR4: The user should be able to start a single player game	
Executor	Andrea
Date	01.05.2019
Time used	1 min
Evaluation	Success
Comment	A single player game was easily started by pressing a button

Table 8: FR4

FR5: The user should be able to see the other user when playing multiplayer game	
Executor	Simon
Date	06.05.2019
Time used	1 min
Evaluation	Success
Comment	When in a multiplayer game, the other user can be seen as the same robot as the user, but with a blue colour.

Table 9: FR5

FR6: The user should be able to turn on/off music and sound	
Executor	Rune
Date	29.4.2019
Time used	3 minutes
Evaluation	Success
Comment	Pressing the sound on/off button in settings results in sound being correctly being turned on or off both in menu and in game play, as it should. Likewise pressing the sound on/off button in game play results in sound being turned on or off correctly in both menu and game play

Table 10: FR6

FR7: The user should be able to choose his own play character	
Executor	Tore
Date	02.5.2019
Time used	N/A
Evaluation	Failure
Comment	We did not prioritise making multiple characters since it wasn't an important feature for the game

Table 11: FR7

FR8: The player should be able to choose preferred language	
Executor	Rune
Date	28.4.2019
Time used	N/A
Evaluation	Failure
Comment	This feature has not been implemented. It was low priority, and given the ammount of time used on the rest of the project, this was not prioritized

Table 12: FR8

FR9: The player should be able to quit the game whenever he wants to	
Executor	Simon
Date	06.05.2019
Time used	1 min
Evaluation	Success
Comment	The user can quit the game at any time and return to the main menu by pressing the button "Main Menu" on the top of the screen.

Table 13: FR9

FR10: The player should be able to access the menu during the game	
Executor	Rune
Date	01.05.2019
Time used	1 min
Evaluation	Success
Comment	Pressing the menu button while in game play brought me to the menu screen as intended. However this also ends the current game.

Table 14: FR10

FR11: The player should be able to access a highscore menu with his previous highscores	
Executor	Simon
Date	06.05.2019
Time used	1 min
Evaluation	Success
Comment	If the user is logged in with a Google account, the top 10 highscores will be displayed. Otherwise a blank highscore table is displayed.

Table 15: FR11

FR12: The player should be able to see a real time score during the game	
Executor	Andrea
Date	01.05.2019
Time used	30 sec.
Evaluation	Success
Comment	A real-time score was shown on the screen from the start of the game

Table 16: FR12

FR13: The player should be able to choose a game character name before he starts playing	
Executor	Simon
Date	06.05.2019
Time used	1 min
Evaluation	Success
Comment	The game uses the name of the Google play account that is used to log in. As this name can be changed, the player can change their name in the game as well.

Table 17: FR13

FR14: The player should be able to choose access a small text tutorial from the menu state	
Executor	Rune
Date	01.05.2019
Time used	1 min
Evaluation	Success
Comment	Pressing the tutorial button from main menu brings me to the tutorail screen where I can read a short tutorial of the game

Table 18: FR14

FR15: The player should see his result when game is finished	
Executor	Simon
Date	09.05.2019
Time used	1 min
Evaluation	Success
Comment	The score is shown in the game over screen when the player dies. It is not shown when a player decides to leave a game before dying.

Table 19: FR15

FR16: The player should be shown placement on the highscore list when a game is finished	
Executor	Simon
Date	06.05.2019
Time used	10 min
Evaluation	Success
Comment	If the player is logged in to a Google play account, his/her highscore is submitted after the player dies. If this highscore is in the top 10 best highscores, it can be viewed in the highscore menu. The updating of these scores might take some time.

Table 20: FR16

FR17: The player should be able to start a new game with one touch gesture from the menu state	
Executor	Andrea
Date	01.05.2019
Time used	15 sec.
Evaluation	Success
Comment	A new game was easily started from the menu.

Table 21: FR17

FR18: Game should end for both players if one of them loose three lives	
Executor	Andrea and Rune
Date	07.05.2019
Time used	1 min. 30 sec.
Evaluation	Success
Comment	Game ended for both players when Andrea lost all her lives

Table 22: FR18

FR19: Player should lose a life if play character is hit by an enemy	
Executor	Andrea
Date	07.05.2019
Time used	1 min
Evaluation	Success
Comment	The play character ran into an enemy, and a life was extracted

Table 23: FR19

FR20: Player should lose a life when an enemy's bullet hits the play character	
Executor	Andrea
Date	07.05.2019
Time used	1 min
Evaluation	Failure
Comment	Due to time limitations, we did not implement any enemies with a shooting functions. Therefore, the play character will never encounter an enemy shooting at him.

Table 24: FR20

FR21: Player should die when falling into the water	
Executor	Andrea
Date	07.05.2019
Time used	1 min
Evaluation	Success
Comment	The player dies when his character falls in the water. Game ends.

Table 25: FR21

4.2 Quality Attribute Requirements

4.2.1 Modifiability

M1: Add new game level	
Executor	Henning
Date	05.05.2019
Environment	Design time
Stimulus	Wishes to add a new level
Expected response measure	1 hour to deploy a new level
Observed response measure	Three hours
Evaluation	Success
Comment	The total time it took to design, build, and playtest the new level is about three hours. These steps took about one hour each. Adding the level to the game was almost immediate. Video game levels vary greatly in both length and complexity, so naturally the development time of a new level will vary just as much. It may have been better to split this requirement in three separate to measure them individually, but even then there would be no guarantee that the measured time would reflect an average scenario, as a simple level can be perhaps be made in 30 minutes from scratch while more complex ones take, well, three hours. So even if this particular level took way longer than the estimate, I still consider it a success

Table 26: M1: Add new game level

M2: Add a new enemy to the game	
Executor	Tore
Date	07.05.2019
Environment	Design time
Stimulus	Developer wishes to add a new enemy
Expected response measure	30 minutes to create
Observed response measure	15 minutes
Evaluation	Success
Comment	Adding a new enemy only involves a few lines of code. This is because we can reuse much of the code written for the other enemies in the game, since we use entity factory. If the enemy need to have more advanced mechanics, this will obviously take longer time.

Table 27: M2: Add new enemy level

M3: Add new play character	
Executor	Tore
Date	02.05.2019
Environment	Design time
Stimulus	Developer wishes to add a new play character
Expected response measure	30 minutes to create
Observed response measure	4 minutes
Evaluation	Success
Comment	Just need to change the sprite sheet the character entity refers to. It includes animations for all the different movements the character does.

Table 28: M3: Add new play character

M4: Add a new powerup to the game	
Executor	Tore
Date	05.05.2019
Environment	Design time
Stimulus	Developer wishes to add a new powerup
Expected response measure	1 hour to create
Observed response measure	20 minutes
Evaluation	Success
Comment	Adding a new powerup is a little more complex than adding an enemy character. But is still simple, since we are using entity factory. Again, if you need to implement new mechanics to the game, it will be more complex and take longer time.

Table 29: M4: Add a new powerup

M5: Add a new play state background	
Executor	Henning
Date	05.05.2019
Environment	Design time
Stimulus	Developer wishes to add a new game state background
Expected response measure	1 hour to create
Observed response measure	10 minutes
Evaluation	Success
Comment	Adding a new background takes very little time, as one only has to add the new background to the intended level through "Tiled".

Table 30: M5: Add new play state background

4.2.2 Performance

P1: Consistency between what two collaborating payers see	
Executor	Matthew
Date	06.05.2019
Environment	Normal mode
Stimulus	Two players play a multiplayer game
Expected response measure	At least 30 FPS
Observed response measure	The worst case scenario is 50+ FPS, usually 60+ FPS
Evaluation	Success
Comment	The framerate is more than 2x higher on most of the tested devices. The worst observed case was 50-56 FPS which is still way above the required rate. Therefore, the framerate is sufficient and allows good synchronization and consistency.

Table 31: P1: Consistency between what two collaborating player see

P2: Search for another player	
Executor	Simon
Date	06.05.2019
Environment	Normal mode
Stimulus	User searches for a teammate to play a multiplayer game
Expected response measure	Search for another player starts within 2 seconds
Observed response measure	1-5 sec
Evaluation	Success
Comment	Searching for a teammate usually starts in 1-2 seconds, but slow networks can increase the time needed.

Table 32: P2: Search for another player

4.2.3 Usability

U1: User successfully playing game for the first time	
Executor	Andrea
Date	01.05.2019
Environment	Runtime
Stimulus	User plays the game for the first time
Expected response measure	Successfully playing the game after 3 minutes
Observed response measure	2 min.
Evaluation	Success
Comment	User was able to play the game during first try. Lost one life in the beginning, but was quick to recover and to get an understanding of how to play it.

Table 33: U1: User successfully playing the game for the first time

U2: User easily understands how to control play character	
Executor	Andrea
Date	01.05.2019
Environment	Runtime
Stimulus	User wishes to figure out how to control the play character
Expected response measure	User is able to understand how to control the play character within 1 minute of first game
Observed response measure	30 sec.
Evaluation	Success
Comment	How to use the buttons are intuitive for most people who has played a mobile game before. The test person quickly got a hold of it, and moved effortlessly through the game world.

Table 34: U2: User easily understands how to control play character

U3: User can change settings during play state	
Executor	Andrea
Date	01.05.2019
Environment	Runtime
Stimulus	User wishes to change setting during play state
Expected response measure	User is able to access and change settings from the play state with no error and without having to quit the game
Observed response measure	30 sec
Evaluation	Success
Comment	There is only one setting our game: sound. This can easily be turned on or off during the game.

Table 35: U3: User can change settings during play state

4.2.4 Availability

A1: The network should always be available in order to start a multiplayer game	
Executor	Simon
Date	06.05.2019
Environment	Normal operation
Stimulus	Network is not responding
Expected response measure	Let multiplayer game mode be unavailable while the problem is fixed
Observed response measure	Multiplayer button does not initiate a multiplayer game
Evaluation	Failure
Comment	If the user is not connected to Google play, the multiplayer button will have no effect. However if the user has no connection to the internet at all, the multiplayer button will cause a fatal crash.

Table 36: A1: The network should always be available in order to start a multiplayer game

5 Relationship with architecture

This chapter describes which of the planned tactics and patterns were used, and how some of them were implemented. There was little deviation between the plan and the implementation, which can perhaps be attributed to the fact that a lot of time was spent planning the architecture, with a huge chunk of the code skeleton planned out beforehand. Most of the deviations stem from the original multiplayer architecture. This is presumed to be the cause because the team had very little to no experience with Google Play Games Services beforehand, and were not sure how to implement it.

5.1 Architecture Tactics

5.1.1 Modifiability

High cohesion/Low coupling
We executed this tactic by proper implementation of several design patterns. The most central patterns for this tactic are the architectural patterns ECS and MVP. Smaller patterns, like the state patterns, also increased cohesion. The nature of ECS makes making high cohesion come naturally for all classes related to that pattern. Strict enforcement of this tactic has ensured that other modules have a similarly high cohesion. We have made good use of clearly defined interfaces to ensure their role before making the implementations.

Information hiding/encapsulation
All general modules are made behind interfaces. The interfaces were made early on to ensure that we knew which functionality each module would have. Most member variables were also private as described in the architecture document. We did, however, elect to keep most of the member variables of the components public, as a means to avoid the overhead of every single "getter" and "setter", as the primary purpose of components is to expose data about its entity in the first place.

Intermediaries
We did not find any use for intermediaries.

5.1.2 Usability

The tutorial is implemented as a simple screen explaining all the details of the game, including every input option and all interactable objects. The

gameplay is intuitive enough to be played without the tutorial if the user chooses to skip it. The buttons are made intentionally large to be easily touchable even on small displays.
Many interactions play a sound to give the player feedback that something happened. The UI also helps the player keep track of changes in the game state such as their lives and score.

5.1.3 Performance

Our game did not suffer any performance hits. The simple design of the game is among the biggest preventive measures for this. Graphics and heavy physics are usually some of the main culprits for performance loss, but we elected to go for a very simple style with no fancy effects or physics. In addition to that, we all have very modern phones, and never had the opportunity to test it on old ones. We can thus not guarantee that old phones will reach a satisfactory frame rate. We could have done some performance analysis to uncover bottlenecks to boost performance on older hardware just in case the game runs choppy on those, but we deemed it unnecessary due to the unlikelihood of anyone playing our game on an ancient device.

Asset management
All assets apart from projectiles are loaded at the start of the level. There is a short delay at the start of each level, but we consider the advantages, such as no load time during play and the guarantee to have all required assets, to outweigh this small disadvantage. The only entity that the ECS creates during playtime is the bullet for the player, but even that asset is stored on the component for the player, and as such, no assets are loaded during playtime. An expansion of the architecture to support other types types of games would probably benefit from a stricter way of handling assets.

Resource management
We experienced no performance issues, and elected to not spend time implementing parallel processing as we knew it would take some development time and can cause hard to find bugs.

Networking tactics
Our network tactics changed significantly from our initial plan, and does not take advantage of interpolation and extrapolation, but instead ensures that the player is on the correct position at the time they make any sort of input, while only estimating the position in the meanwhile. The packets are

still processed FIFO, but there are way less packages to process due to only changes in input being sent across google play games services.

5.1.4 Availability

The Google Play Games Services worked as expected, even though the integration took more time than initially estimated, and a few other complications arose. As availability is mainly controlled by the Google play servers, no further tactics for the availability were implemented. It's worth noting that even after this was integrated, implementing it was no easy task either.

5.2 Architectural Patterns

This section confronts planned use of architectural patterns with the actual implementation.

Model View Presenter (MVP)
The Model View Presenter pattern was utilized as expected and without any issues. No modifications of the plan were necessary.

Entity Component System (ECS)
Entity Component System turned out to be even more helpful than was anticipated. The initial attitude was skeptical and even not using it completely was discussed. In the end, this pattern is one of the most important building blocks of the implementation.

Mobile Backend as a Service (MBaaS)
Google Play Game Services library provided what its documentation promised and was utilized as planned. On the other hand, the peer to peer communication was slower than expected which needed to be taken into account during development.

5.3 Design patterns

This section confronts planned use of design patterns with the actual implementation.

5.3.1 Creational patterns

Abstract factory
The Abstract factory worked really well but for a slightly different purpose

than was planned. Initially, its role was to create components for later entity creation. During development a better pattern fit was found which was the direct creation of entities. This new purpose made the pattern more useful and also an important part of the architecture.

Builder
The Builder pattern shares a similar story with the Abstract factory pattern. The original idea was building entities from already created components, nonetheless, using it for component creation and initialization instead worked significantly better. That is also the pattern's current role in the project.

Dependency Injection
Due to not really having a service oriented architecture, the Dependency Injection system showed up as unnecessary. Even though it might be a good extension of the implementation in the future, it is not used in the current version of the project.

Singleton
Singleton originally had only one purpose and that was implementation of the Dependency Injection container. Therefore, it is not implemented.

5.3.2 Structural patterns

Proxy
The idea of proxied multiplayer game instances yielded to be more complicated than previously thought. That is why the pattern was not used for this approach but only for smaller tasks such as input module multiplexing.

Marker
The intended use of the Marker pattern was to mark classes as services that can be injected. Because of the fact that Dependency Injection pattern was not utilised, this pattern is also not present.

5.3.3 Behavioral patterns

Chain of responsibility
The Chain of responsibility pattern is used as planned before and worked well.

Observer
The Observer was utilized as anticipated and turned out to be even more

helpful in decoupling parts of the project and therefore significant for the whole design.

State
The State pattern worked as expected and was also used by more classes than initially expected.

5.3.4 Sequencing patterns

Game loop
The Game loop is implicitly provided by LibGDX and was used as planned.

Update method
The Update method was implemented as planned.

6 Problem, issues and points learned

This section presents problems and issues the group experienced during the project. We also reflect on what we have learned and would take with us when working in the future.

6.1 During documentation process

The documentation process started quite smoothly. By looking at previous years' deliveries and reading the documentation description, the group was able to get a good impression of what the process would involve. We started with the requirement documentation, then followed up with the architectural documentation, and ended with the implementation documentation.

All members agree that the group from the start did a good job in dividing documentation responsibilities among each other. Each member was assigned different sections, and we made sure that sections in different documents with similar topic were given to the same group member. This created a nice work environment, where none of the members felt overloaded with documentation work.

However, we did encounter a couple of problems and issues with the documentation process. Especially, many of us found it challenging to do the design documentation before we started implementing. Most of us had no experience with starting with a detailed architectural design description and then implementing this. We usually started implementing our design based on a brief description and requirements. It therefore took use a while to establish a detail design documentation for the architecture that was able to fulfil our requirements, before we started writing the code. However, all group members agreed that having the architectural documentation made it easier to collaborate, and the views gave us a common understanding.

Furthermore, we had some problems with keeping the documentation up to date. Changes were made continuously, and these were often not logged in the documentation. For example, during the process some of the functional requirements were change due to time limitations and unforeseen complications, however, only in the end many of these changes were recorded. Instead of doing this continuously, the group had to use a lot of time in the end to make sure the requirement documentation was up-to-date.

6.2 During the implementation process

In general, there were not many problems during the implementation process but still a few thithe ngs stood out. Initially, the main issue was the lack of knowledge and experience with the LibGDX framework. It was hard to figure out which things were easy and which solutions might be hard to implement, so it was necessary to make some guesses and hope for the best. This was further worsened by the decision of using two more libraries that no one in the team was familiar with: Ashley and Box2D.

An event which also influenced the development process was a period of time around Easter when more than half of the team members were on holiday trips. This made team communication and coordination of work more difficult and lowered the development pace during that period. In the end, it was necessary to speed up the pace during last weeks of the project in order to compensate for that.

Finally, there was also an issue with the chosen Peer-to-peer solution, Google Play Games services. The integration of Google play services into the project took more time than expected. One of the reasons was that quite a bit of code was needed to have a testable setup. This in conjunction with a type of coding that was unfamiliar to the person implementing, getting a working prototype took some time.

After initial testing, it was found that the message sending was much slower than expected. This meant that the original synchronisation scheme was no longer feasible and a new one had to be designed.
The scheme that the group came up with was sending all actions that are made by the player, such as starting and stopping a movement to the left/right, and calculating all other interactions from these actions.
This strategy was tested and found to be lacking, so to combat some of its weaknesses, the position of the character is send together with some of the actions. This makes the character's movement look jumpy at times, but that is preferable over desynchronisation.

Finally there were some problems during design with being able to log into Google with certain accounts. Some team members accounts worked, others did not. This should be fixed by releasing the game on the google play store where all accounts are valid.

6.3 Points learned

All group members agree that working on this project has been educational and interesting. Several of the members had no experience in architectural design from before, and during this project we gained insight in how this is done in practice. In stead of reading about it in a textbook, we felt we were exposed to pitfalls of a real software project that would not have been fully comprehended without hands-on experience. We felt that it was both interesting and challenging to develop a full architectural design, before we started to code. What is more, several of the members were exposed to new platforms like libGDX, Android Game Play services, and application making in general. This was partly frustrating when we encountered unfamiliar problems, but also very satisfying when we felt we were able to master the new technology.

The group consisted of new members. In the beginning, it took us a little time to get to know each other. However, we used quite some time on creating a safe and positive working environment, something we believe was critical for the work we did afterwards. All members agree that they would take with them this experience into future work. Creating a safe environment made everyone feel safe enough to express their thoughts and ask for help. However, being a group of people with different background also created some frustration. There was a lot of different opinions, and we used a lot of time on merging the different ideas in order to satisfy everyone. We believe though that this will be the case many future projects as well, and feel that we all have gained more insight in how to handle group members with different perspectives.

During the project, we soon understood how important communication was for the project's success. We started out with no clear task-communication platform, and everyone worked on whatever they felt like. This soon became a problem, due to the fact that we did not get any sense of the project's progress. We therefore created a Trello-board to keep track of tasks that had to be done, and a overview of what everyone was currently working on. In future projects, we would have created such a board much earlier on.

The group members were assigned different responsibilities. One took the role as leader and coordinator, while another became lead architect. Members felt this was positive. They knew who to turn to for questions, and that someone was taking responsibility for scheduling meetings, assigning documentation tasks, etc.

Overall, all members agree that the project has given them new insight and has been a valued experience. We saw how important a good architectural design was for the implementation, that members with good communication skills are key, and that building a trusting environment can have a great impact on the team's work.

References

Android Developer. (2017). *Real-time Multiplayer.* Retrieved 16-02-2019, from https://developers.google.com/games/services/common/concepts/realtimeMultiplayer

Android Developer. (2019). *Meet Android Studio.* Retrieved 16-02-2019, from https://developer.android.com/studio/intro/

Brown, K. (2017). *What Is GitHub, and What Is It Used For.* Retrieved 16-02-2019, from https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/

LibGDX. (n.d.). *Itroduction to LibGDX.* Retrieved 16-02-2019, from https://libgdx.badlogicgames.com/documentation/