Requirements 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 requirement 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 uses Android as our game platform, and the intention is to implement 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 we respect to important qualities in software architecture. Our targets are described in sections 2 and 3.

The project officially started 12th of February and will continue for approximately 12 weeks, to the 29th of April. During these weeks, the project will be divided into three phases: "Requirements and Architecture", "Evaluation (ATAM)" and "Design, Implementing and Testing". What is more, we will also contribute with feedback to other groups taking the same course, as well as receiving feedback.

In the current phase, Requirements, will focus on specifying the requirements of our game. We specify the functional requirements with assigned priority level. Further more, the quality attributes of importance will be described. Our main quality attributes are Modifiability, Performance, Usability, and Availability. We end with a description of Components and Technical Constraints, which will affect the design of our game.

1.2 Game Description

The Pendulum Paradox is a platform game, and therefore a subgenre of action game. The player will control a sprite object, and advance through different levels, starting in a forest. During the game, the player will face 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 will be equipped with a basic shooting device in order to defend themselves.

The players will 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 will only last 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.

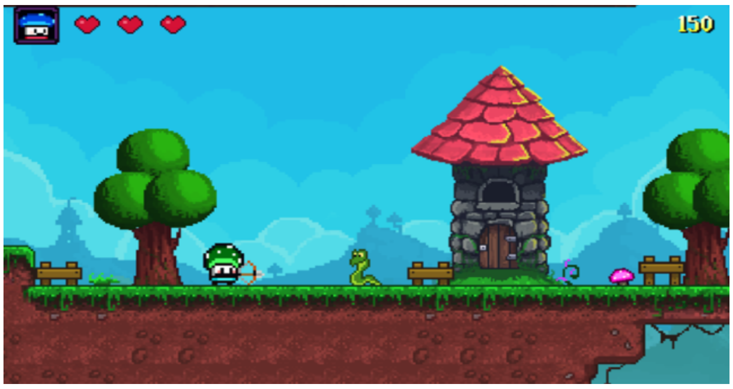

Figure 1: Screenshot from the platform game A Great Adventure

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

Figure 3: Screenshot from the platform game Jazz Jackrabbit

1.3 Structure of Document

This paper focuses on presenting our game, the different quality requirements and attributes we wish to fulfil, as well as describing our technology choices in the project. We will start by presenting our functional requirements in chapter 2. They will be given with an ID and a priority level. Further, in chapter 3, our quality attribute requirements are described. Our main quality attributes are Modifiability, Performance, and Usability. Each of these are presented in detail with ID, stimulus source, stimulus, artifact, environment, response, and response measure. Chapter 4 considers our components and technical constraints, focusing on Android OS, Android Studio and JAVA, LibGDX, GitHub, and Google Play Services. In the end of the document, you will find issues and a change log.

2 Functional Requirements

Table 1 presents a list of functional requirements that our game has to fulfil. Each requirement is given an unique ID, and a priority level.

ID	Requirement	Priority
FR1	The user should be able to move around in the play world using transparent control buttons	High
FR1.1	The user should be able to move left and right using a joystick arrow function	High
FR1.2	The user should be able to jump using a button	High
FR1.3	The user should be able to shoot at enemies using a button	High
FR1.4	The user should be able to use other firearms for a short while after he has collected enhancement	Medium
FR2	The user should be able to start a multiplayer game	High
FR3	The user should be notified if no other player is found	High
FR4	The user should be able to start a single player game	High
FR5	The user should be able to see the other user when playing multiplayer	High
FR6	The user should be able to turn on/off music and sound	Low
FR7	The user should be able to choose his own play character	Low
FR8	The player should be able to choose preferred language	Low
FR9	The player should be able to quit the game whenever he wants to	High
FR10	The player should be able to access the menu during the game	Low
FR11	The player should be able to access a highscore menu with his previous highscores	Medium
FR12	The player should see a real time score during the game	High
FR13	The player should be able to choose a game character name before he starts playing	Low
FR14	The players should be able to access a small text tutorial from the menu state	High
FR15	The players should see his results when game is finished	Medium
FR16	The players should be shown placement on the highscore list when a game is finished	Medium
FR17	The player should be able to start a new game with one touch gesture from the menu state	High
FR18	Game should end for both players if one of them loose three lives	High
FR19	Player should lose a life if play character is hit by an enemy	High
FR20	Player should lose a life when an enemy's bullet hits the play character	High
FR21	Player should die when falling into the water	High

Table 1: Table with functional requirements

3 Quality Requirements

3.1 Descriptions

Quality attribute requirements are qualifications of the functional requirements or for the product as a whole (Bass, Clements, & Kazman, 2013). The quality attribute requirements should be specified in a concrete fashion and you should be able to test whether the system satisfies them. Bass et al. (2013) specify the following way to express the quality attributes:

- *Stimulus source*: The source of the stimulus.
- *Stimulus*: Used for an event arriving at the system. It triggers the specified scenario.
- *Environment*: Describes the state of the system when the scenario takes place.
- *Artifact*: Tells which part of the system is affected. It can be the whole or only part of the system. In modifiability requirements we have specified which classes are affected
- *Response*: How the system should respond. What should be done as a response to the stimulus.
- *Response measure*: How we decide whether the response is satisfactory. Makes the requirement measurable.

For our game modifiability is our primary quality attribute, while performance, usability and availability are our secondary quality attributes. Each attribute will be given an unique ID.

3.2 Modifiability

ID:	M1
Stimulus Source:	Developer
Stimulus:	Wishes to add a new level
Artifact:	GameScene, LevelManager
Environment:	Design time
Response:	The new mode should be available for players after they successfully complete the last level as of now
Response measure:	1 hour to deploy a new level

Table 2: Add new game level

ID:	M2
Stimulus Source:	Developer
Stimulus:	Wishes to add a new enemy
Artifact:	GameScene, EntityFactory
Environment:	Design time
Response:	New enemy will be available for the developer to deploy in play state
Response measure:	30 minutes to create

Table 3: Add a new enemy to the game

ID:	M3
Stimulus Source:	Developer
Stimulus:	Wishes to create a new game character
Artifact:	GamePresenter, EntityFacotory
Environment:	Design time
Response:	Deployed character will be available for player to select
Response measure:	30 minutes to create

Table 4: Add new play character

ID:	M4
Stimulus Source:	Developer
Stimulus:	Wishes to add a new powerup to the game
Artifact:	GameScene, EntityFactory
Environment:	Design time
Response:	New powerups can be deployed to different difficulty modes by the developer
Response measure:	1 hour to create

Table 5: Add a new powerup to the game

ID:	M5
Stimulus Source:	Code
Stimulus:	Wishes to add a new game state background
Artifact:	GameScene
Environment:	Design time
Response:	New game state backgrounds will be available in the game
Response measure:	1 hour to create

Table 6: Add a new play state background

3.3 Performance

ID:	P1
Stimulus Source:	User
Stimulus:	Two users play a multiplayer game
Artifact:	System
Environment:	Normal mode
Response:	There is consistency between what the two players observe at their screens
Response measure:	At least 30 FPS

Table 7: Consistency between what two collaborating players see

ID:	P2
Stimulus Source:	User
Stimulus:	User searches for a teammate to play a multiplayer game
Artifact:	System
Environment:	Normal mode
Response:	Other player is searched for
Response measure:	Search for another player starts within 2 seconds

Table 8: Search for another player

3.4 Usability

ID:	U1
Stimulus Source:	User
Stimulus:	Downloads the game for the first time
Artifact:	System
Environment:	Runtime
Response:	User is able to successfully play the game
Response measure:	Successfully playing the game after 3 minutes

Table 9: Playing the game for the first time

ID:	U2
Stimulus Source:	User
Stimulus:	User wishes to figure out how to control the play character
Artifact:	User interface
Environment:	Runtime
Response:	User can access a short text tutorial with pictures from the menu state
Response measure:	User is able to understand how to control the play character within 1 minute of first game

Table 10: Understanding how to control play character

ID:	U3
Stimulus Source:	User
Stimulus:	User wishes to change settings during play state
Artifact:	User interface
Environment:	Runtime
Response:	User can access the settings menu from the play state
Response measure:	User is able to access and change settings from the play state with no error and without having to quit the game

Table 11: Changing settings during play state

3.5 Availability

ID:	A1
Stimulus Source:	Network
Stimulus:	Network is not responding
Artifact:	Process
Environment:	Normal operation
Response:	Let multiplayer game mode be unavailable while the problem is fixed
Response measure:	No downtime

Table 12: The network should always be availble in order to start a multiplayer game

4 COTS Components and Technical Constraints

In the development of *The Pendulum Paradox*, we have chosen a multi-tier architecture. This makes it possible to group together components of similar functionality. This n-tier architecture focuses on separating the processing, data management and presentation functions (Stackify, 2017). Using this architecture creates flexibility for the developer, and is important for meeting the modifiability quality attribute. We will now give a description of our COTS in relation to this architecture.

4.1 Android OS and API

The Android Operating System was developed by Google, and was initially released in 2008. Its primary design is for smartphones and tablets. Choosing Android as the operating system for our game, makes it necessary for the user to possess an Android device or have a device with an Android API. Having the Android API on the computer during the game development will also be preferred so that the developers can run the game during development. Android has the largest market share as operating system on smartphones. In 2008, 88% of all smartphones used Android OS, while only 11.9% used iOS (Statista, 2018). By using Android as the platform for our game, we are able to target over 2 billion devices (Popper, 2017).

Since we are using Android, we have to be aware of some constraints. For example, we have to be aware that different Android devices have different screen resolution. This is important to have in mind, since it will influence the graphics. It also means that we have to be able to scale the GUI. In addition to having different screen resolutions, the devices also have different screen sizes. During development, we must make sure that the game character is easy to navigate also when the game is played on devices with smaller screens. Google periodically releases new versions of Android, so different devices might run on different versions. Since some classes and methods do not run on all versions, we have to consider which versions our game should support.

Using android also means that an Activity has to be implemented to start the application. This puts a constraint on the architecture as it forces the initialisation of the application to take place here.

Applications written for Android is executed using a Dalvik Virtual Machine. This makes the application use less physical space and a smaller memory footprint (Android Java Point, 2017). This constraints how complex our game can be.

4.2 Android Studio and Java

Android Studio is the official integrated development environment (IDE) for development of Android applications(Android Developer, 2019). Android Studio supports several different programming languages, for example Java. We use Java as our primary programming language in developing our game. Depending on the Java SDK version we use, we will be restricted in using functionalities from higher versions, but lower versions are more widely supported. Java supports many of the design and architectural patterns we wish to use in the development of the game. It is object oriented language, making it easy to create classes and interfaces. This is an important and helpful feature for implementing several patterns. The patterns we choose to use will help us fulfil the listed quality attributes, but will also come with certain trade-offs.

4.3 LibGDX

LibGDX is an open source cross platform game development framework (LibGDX, n.d.). By using LibGDX, we get several high level tools making it easier to create a game application. It is cross-platform, and supports among others Windows, Linux, Mac OS X, and Android. A stack based state machine is used. It controls what state is active. The active state can call methods for changing state and/or updating information for other objects. When using libGDX, we have to be aware of the fact that libGDX do not clear states nor dispose of graphics. In the development of our game, we have to implement methods for this to make sure that the device does not run out of memory and crashes. If we do not do this, it could potentially reduce the availability, usability and performance of our game.
As libGDX provides us with high level tools, it must obscure some of the low level details and thus constraint our architecture and what is possible. It for example provides a screen class that has to be extended to make menu screens and such.

4.4 GitHub

Since we are several people working on developing the game, it is important that we are able to share code on a common platform. We use GitHub for this purpose. GitHub is a version control system that keeps revisions of the code and store modifications (Brown, 2017). Every member can see what changes that have been made and upload the changes they make. In addition, GitHub keeps a record of previous changes, making it easy to switch back to an older version if some new feature is making the code unrunnable. By using branching, we can test out new code without affecting the code that is working properly so far.

4.5 Google Play Services

Our game is a real-time multiplayer game. To create this real-time multiplayer function, we use the real-time multiplayer API in Google play games services. This lets us connect players together in a single game session. In addition we can transfer data messages between players (Android Developer, 2017). By using the API, the game development is simplified because it helps us with the management of network connections, provides an UI for the waiting room, stores participants and update all players (Android Developer, 2017). By developing a real-time multiplayer networking game, we are increasing the complexity of our game, and availability becomes a critical quality attribute.
Using Google play also contstraints the architecture, as it needs to be instantiated on setup and be able to control the multiplayer character. Yet the game should also work if one is not logged in. This means that there has to be some loose coupeling between the game and the network.

4.6 Other constraints

In our game design, we intend to use several design and architectural patterns. These will have some constraints associated with them. We intend to use the Observer pattern. This pattern is used to make sure that all players are automatically notified of any changes and pushing the correct players' states to the GameStateManager. This pattern will introduce several layers that the code must traverse. This increases latency, and thereby also affects performance.

We also intend to use the Abstract Factory pattern for creating different play characters, enemies and powerups during the play state. By using this pattern we encapsulate individual characters without specifying the classes. The play state is unaware of how to initialise the characters, the enemies or the powerups, and also about which parameters to provide. Again the use of this pattern introduce more layers, which can affect latency and performance.

We also wish to deploy the model-view-controller (MVC) pattern. This pattern helps us divide the program into data (*model*) and user interface (*view*). This makes us able to do changes in the user interface without affecting how the data is treated, and vice versa. We use a *Controller* to transfer data. This will simplify the code by avoiding replication. The constraints of using this patterns, is that it increases complexity, requires multiple programmers and knowledge of multiple technologies (Inter server, n.d.).

5 Issues

No issues so far

6 Changes

Our first draft of this document, included a description of the next phase (Architectural) in the introduction. In our final version we have changed the introduction to only include what the current phase (Requirement) involves.

The game description in chapter 1 has been updated in order to make it consistent with how the game came to be.

Table 1 has been updated when new functional requirements have been discovered during the project.

The Artifact section for each of our Modifiability quality attributes have been updated in order to be more specific. In first draft, we only said "Application", not which part of it. In the final iteration we are specifying which classes in the code are affected

Functional requirement FR1.2 was change from "The user should be able to jump and duck using a joystick arrow function" to "The user should be able to jump using a button" due to time limitations and the fact that we saw that having another button for ducking would take too much screen space.

Functional requirement FR1.4 was changed from "The user should be able to use other firearms after he had collected them using a swipe function" to "The user should be able to use other firearms for a short while after he has collected them" due to time constraints and a wish to limit complexity

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/

Android Java Point. (2017). *Closer Look At Android Runtime: DVM vs ART.* Retrieved 16-02-2019, from http://androidjavapoint.blogspot.com/2017/04/closer-look-at-android-runtime-dvm-vs.html

Bass, L., Clements, P., & Kazman, R. (2013). *Software architecture in practice.* Pearson Education, Inc.

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/

Inter server. (n.d.). *What is MVC? Advantages and Disadvantages of MVC.* Retrieved 16-02-2019, from https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/

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

Popper, B. (2017). *Google announces over 2 billion monthly active devices on android.* Retrieved 16-02-2019, from https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users

Stackify. (2017). *What is N-Tier Architecture? how It Works, Examples, Tutorials, and More.* Retrieved 16-02-2019, from https://stackify.com/n-tier-architecture/

Statista. (2018). *Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2018.* Retrieved 16-02-2019, from https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/