NTNU
Kunnskap for en bedre verden

TDT4240 - Software Architecture

"The Nearly Impossible Game" Requirements

Group 12:
Jakob E. Eikeland,
Ruben K. Rokkones,
Gaute Brandser,
Tan Chuan Xin,
Teo Chen Ning,
Sigve Sjøvold

COTS: Android, LibGDX, Firebase

Primary QA: Modifiability
Secondary QA: Usability Secondary QA: Availability
February, 2022

Contents

1 Introduction

The goal of this project is to create a functioning multiplayer game or app for mobile devices. This document provides the quality and functional requirements for the finished product.

1.1 Description of the game concept

Our group had several ideas about what type of game we wanted to create. after some discussion we decided to try to create a version of "The Impossible Game", a mobile game some of us had played when we were younger. It is a game where you play as a square going through a course made up of various obstacles appearing in its path with increased frequency and difficulty. The goal of the game is simply to reach the end of the level without dying. The basic flow of the game is that the game starts, the square (the player) starts gliding forward on the ground, and spikes start appearing. If hit from any direction by the spike, you will die, and restart the game. The player controls the square in each direction, and can jump.

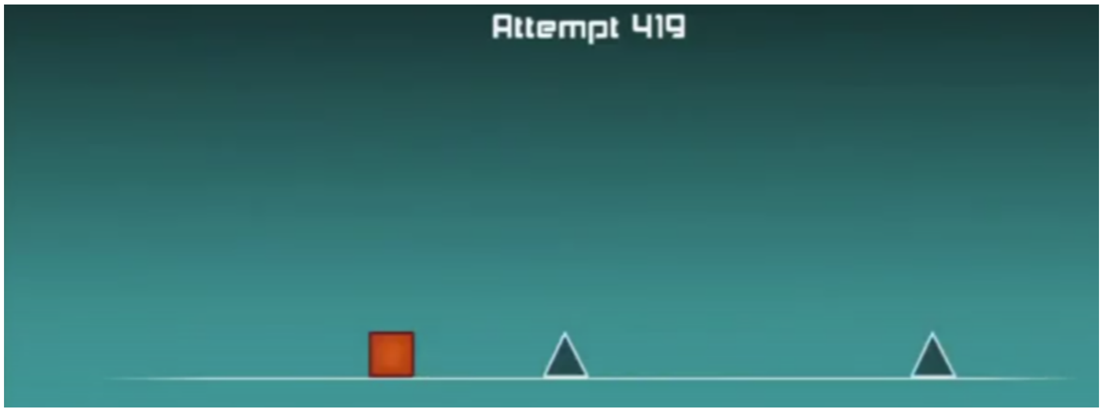

Figure 1: Original game

Our version of this game will have both single player and multiplayer modes. In single player mode, it is similar to the original Impossible Game. In multiplayer mode, two players will alternate in turns playing the game. We will also implement an online high score list. We considered various different ways of having multiplayer functionality, such as two mirrored courses on each side of the screen, or maybe two players next to each other on the same course, but decided to have a two players taking turns playing the same level. When the first player dies or wins, the game switches to the other player, who tries playing until he dies or wins. When both players have beaten the level, or died, they can type in their names and submit their scores to the database. They also have the option to skip this step, if they feel their scores are too low.

This version of this game ends up looking very different. We wanted to make our own game, and not copy the amazing original game. However we kept the core components of blocks and spikes. In our game you play as a jelly, and your goal is to get to the end of the map.

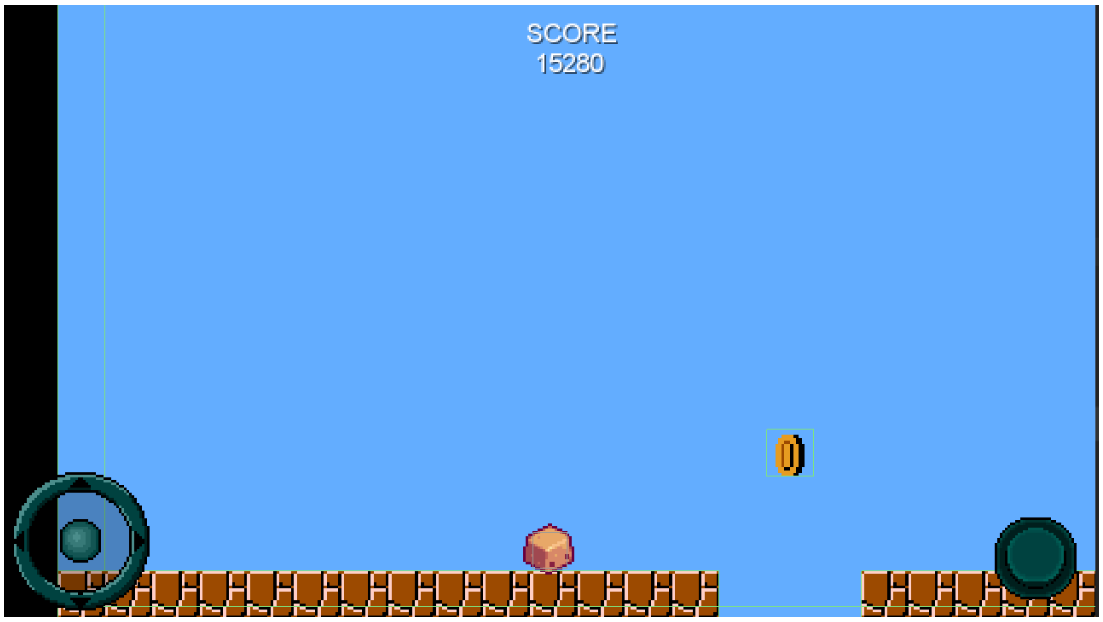

Figure 2: Beginning of game

The player has a jump button, and a joystick that controls lateral movement. On the way to the goal, the objective is to get the highest score as possible. This is achieved by picking up as many coins as you can, and going as fast as you can. The player only has one life, because this is "The Nearly Impossible Game". Another element we added to the game is fireballs. When you're gliding through the map with ease, a fireball can suddenly appear and mess up your game.

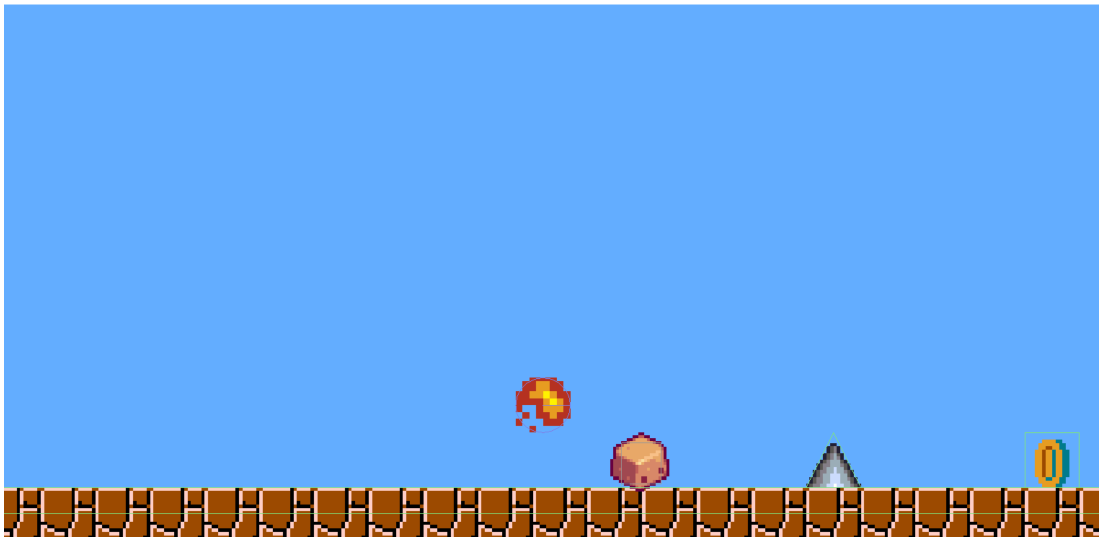

Figure 3: Fireballs and coins

Another way of messing the player's game up comes from the other player in when playing multiplayer. The player not currently playing can press a button in the top right hand corner, shoving the active player in a random direction in order to try to stop him from getting a good score.

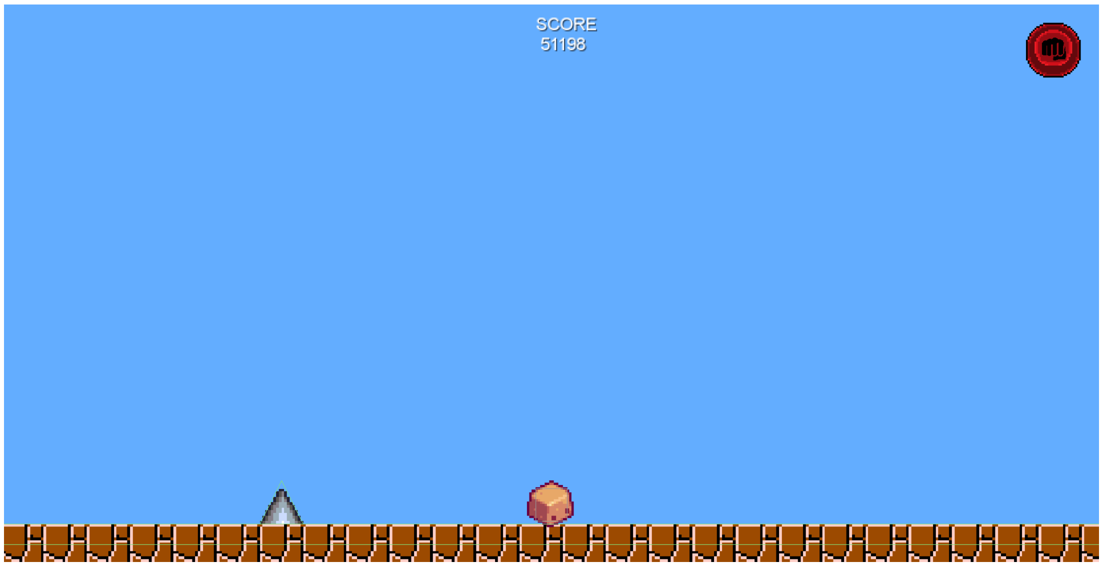

Figure 4: Button to mess with opponent

1.2 Structure of the document

This document gives an overview of the requirements we have identified for our game. section 2 outlines the functional requirements, while section 3 discusses the quality requirements. Section 4 details our choices concerning COTS, and how they impact the project. Finally, we show the individual contribution of each member, changes in the project from beginning to end, and references

2 Functional requirements

This section shows the functional requirements we have identified for the project. Table 1 shows the functional requirements with a measure of their priority. High priority means we consider the requirement mandatory, medium means it would make the final product better in a significant way, and low means it could be a good thing, but is not needed in any way.

ID	**Requirement**	**Priority**
FR1.1	The game should allow and respond to touches from the user	High
FR1.2	The user should be able to make their character jump by tapping the jump button	High
FR1.3	The user should be able to make their character jump higher by holding down the jump button	Low
FR1.4	The user should be able to move forwards and backwards by pressing the left and right buttons	High
FR2.1	The game should produce obstacles like spikes and boxes	High
FR2.2	The game should produce obstacles that increase in difficulty, creating a more complex environment	Medium
FR3	The game should cause the character to die if it hits a damaging obstacle (spikes)	High
FR4	The game should have support for two players alternating to run, allowing for offline turn by turn multiplayer	High
FR5	The second player should be able to create obstacles with a touch	Low
FR6	The game should have the music inspired by the original game	Low
FR7	The game should give the player a score based on the amount of deaths and time taken to complete a course	High
FR8	The user should have a unique character model as compared to the other player	Low
FR9	The game should have gravity, such that after a jump, the user falls downward	High
FR10	The game should have different courses/maps, like the original game	Medium
FR11	The game should have an option for creating save-points in the course, giving up high score	Low
FR12	The game's background color should change a different points throughout the course	Low
FR13	The game should have support for an online high score list	High
FR14	The game should have support for a co-op mode	Low
FR15	The game should spawn power-ups (e.g. revive on the spot)	Low
FR16	The players will get to pick their characters at the start of each run	Low

3 Quality Requirements

Here we will look at some quality attributes we have chosen for our system. Our primary quality attribute will be modifiability, and we have included usability as our secondary quality attribute.

3.1 Primary QA: Modifiability

M1: Developer should be able to add another level within 120 minutes

ID	M1
Source	Developer
Stimulus	Wants to create another level for the players
Artifacts	Level module
Environment	Design time
Response	The player should be able to play the new level once they update the game, and the level should be correctly integrated into the architecture and tested
Response Measure	Done within 120 minutes without adjusting other modules

Table 2: Modifiability requirement 1

M2: Developer should be able to add new character designs in 30 minutes

ID	M2
Source	Developer
Stimulus	Add new character design to the list of available characters
Artifacts	Character module
Environment	Design time
Response	Player should be able to play with the new character after they update the game, and character should be correctly integrated in the architecture and tested
Response Measure	Done within 30 minutes given that the design already exists

Table 3: Modifiability requirement 2

M3: Developer should be able to create new obstacles in 90 minutes

ID	M3
Source	Developer
Stimulus	Create a new type of obstacle which can be implemented into current and new levels
Artifacts	Level module
Environment	Design Time
Response	Player should experience levels with a new type of obstacles and new obstacle should be correctly integrated in the architecture and tested
Response Measure	Done within 90 minutes.

Table 4: Modifiability requirement 3

M4: Developer should be able to add new game mode within 240 min.

ID	M4
Source	Developer
Stimulus	Create a new game mode which allows for a different gaming experience within the limitations of the existing game
Artifacts	GameMode module
Environment	Design Time
Response	Player should have the opportunity to select preferred game mode, and play both of them after update. New game mode should be correctly integrated in the architecture and tested
Response Measure	New game mode should be functional after 240 minutes

Table 5: Modifiability requirement 4

M5: Developer should be able to add new music within 20 minutes

ID	M5
Source	Developer
Stimulus	Implement new music to an existing level.
Artifacts	Level module
Environment	Design Time
Response	After update the level should have the new music implemented during play time. New music should be correctly integrated in the architecture and tested
Response Measure	Done within 20 minutes

Table 6: Modifiability requirement 5

M6: Developer should be able to add power-ups and environmental changes within 150 minutes

ID	M6
Source	Developer
Stimulus	Create a new element for the second player to interact with while the first player is playing the game, and power ups for the player.
Artifacts	Environment module
Environment	Design Time
Response	After update the level should have the new power-ups and environmental changes correctly integrated in the architecture and tested
Response Measure	Done within 150 minutes

Table 7: Modifiability requirement 6

3.2 Secondary QA: Usability

U1: The end user should be able to understand the mechanics of the game by playing it the first time.

ID	U1
Source	End user
Stimulus	Understand the game by playing it
Artifacts	The game
Environment	During first play
Response	The end user should understand the game mechanics just by playing the game
Response Measure	Learn the game mechanics by playing for 2 minutes

Table 8: Usability requirement 1

U2: User configuring settings

ID	U2
Source	User
Stimulus	Wants to change settings, most likely volume
Artifacts	System
Environment	Runtime, but before play state
Response	User finds settings button, and changes desired setting
Response Measure	User changes a given setting within 10 seconds

Table 9: Usability requirement 2

U3: User starts a game with a friend

ID	U3
Source	User
Stimulus	Wants to start a game playing versus friend
Artifacts	System
Environment	Runtime, but before play state
Response	User intuitively finds buttons for play, player select(1 or 2), map select and character select to open a new game
Response Measure	User goes from opening app to starting a game within 2 minutes

Table 10: Usability requirement 3

3.3 Secondary QA: Availability

A1: Unexpected game crash should be handled

ID	A1
Source	Game
Stimulus	Handle game crash
Artifacts	Process
Environment	Runtime
Response	If game crashes/freezes/collapses for unexpected reason, the app should return to main menu without a problem
Response Measure	Game goes from unexpected crash happening to main menu ready to go again in less than 40 seconds

Table 11: Availability requirement 1

A2: Game should respond correctly to player input

ID	A2
Source	End-user
Stimulus	Player action
Artifacts	Game-state
Environment	Runtime
Response	Game should respond to player input with intended action
Response Measure	The user's input is handled correctly at least 99 percent of the time

Table 12: Availability requirement 2

4 COTS

This section discusses our choices of COTS, and what possible constraints they place on the project as a whole.

4.1 Android

We are going to use Android as the platform for this application. Android is an open-source platform lacking standardization, and because of that it allows for many different devices, with many different screen sizes. This presents a signifcant constraint on our project. All visual elements of the game have to be able to adapt to a multitude of different screen sizes, which needs to be taken in to account in textures, graphics, and code. We have to account for these screen resolutions in addition to different DPI's, and thus need to scale the GUI accordingly. Fortunately, LibGDX takes care of the different screen sizes for us. Since we are developing for android it is natural to use Android Studios for the project. The IDE uses Gradle as a build automation tool, which requires Java 11 for the newest version (7.x.x). Additionally, we are targeting Android 10 (API 32) as Google Play recommends at least Android 10 for new apps in the play store, otherwise users will see a warning on app boot.

4.2 LibGDX

For our project, we chose to use LibGDX which is an OpenGL based framework for Java used for creating games. It has a large base of various libraries and tools that make game development easier. In addition, it's what we were encouraged to use in the early exercises in this course, so it's something all the members of the group are in some way familiar with. This was the main factor in choosing LibGDX. One constraint this framework presents comes from the way it handles screens. LibGDX places all available screens in a sort of stack, and brings different ones to the top of the stack as active. This means that our architecture needs multiple screens, all with the built in Screen interface. This will be the menus and the actual game. Also, by using LibGDX we are forced into a gameloop architecture.

4.3 Firebase and Multiplayer

For database/multiplayer support we chose to use Firebase. Firebase is a full stack managed platform including a realtime NoSQL database, authentication and more. It also includes many client libraries which allows us to interact with the platform from our game directly. The reason we chose to use Firebase in the project is that it will help us implement multiplayer capabilities like a high score list without having to host our own database infrastructure. Since we are only using Firebase to store high scores it will not affect our architecture in a major way since it only requires a simple interface for pushing and getting high score data.

5 Issues

The most significant issue we had around requirements is that we struggled to be on the same page in the beginning about exactly what the game would be, coming up with new ideas expanding the scope of the game, such as dynamic maps, fireballs shooting through the map and more. This lead to some misunderstandings and confusion during the project, because some members had different ideas of what the game would look like. By the end, a lot of the big ideas we had were discarded due to immense time constraints.

6 Changes

1 Introduction

- fixed punctuation errors
-

2 Functional Requirements

- altered FR6 to not describe how good the music is

3 Quality Requirements

- fixed naming error for M6
- added to response in M1-M6 that they should be integrated and tested
- added u2 and u3
- removed old u2 due to feedback on it being more of a functional requirement
- added quality requirements A1 and A2 after deciding to focus on an additional quality attribute

4 COTS

- rewritten as a whole to discuss constraints more in depth

7 Individual Contribution

Everybody contributed equally.

References

[1] Android vs ios: Which platform to build your app for first? https://medium.com/@the_manifest/android-vs-ios-which-platform-to-build-your-app-for-first-22ea8996abe1. (accessed: 22.02.2021).

[2] Top android os versions. https://www.appbrain.com/stats/top-android-sdk-versions. (accessed: 21.02.2021).

[3] Use java 8 language features and apis. https://developer.android.com/studio/write/java8-support. (accessed: 21.02.2021).

[4] Vlad Fatu. Build a serverless multiplayer game with firebase. https://levelup.gitconnected.com/build-a-serverless-multiplayer-game-with-firebase-bf50454a1b69. (accessed: 25.02.2021).

[5] Features. https://libgdx.com/features/. (accessed: 21.02.2021).