Requirements and Analysis Document for the JumpyDash project (RAD)

Alex Sundbäck
Johannes Mattsson
Marcus Bertilsson
Oscar Hansson

May 2016

Contents

1 Introduction

This section gives a brief overview of the project.

1.1 Purpose of application

The project aims to create a rhythm based platform game in a two dimensional world for desktop use, called JumpyDash.

1.2 General characteristics of application

The application will be a desktop, standalone (non-networked), single player application with a graphical user interface for the Windows/Mac/Linux platforms.

The game will be in 2D with a player moving at constant speed. As a player you can only jump and shoot. The player will be able to pickup abilities that will change the way the player object act. The player will be able to fire shot against enemies that appears as the player moves. If the player gets hit by an enemy or touch a spike, the player will take damage. The player loose either by taking too much damage or by falling out of the map.

1.3 Scope of application

The application does not support multiplayer nor computer-players. The application allows the player to collect points but will not save any statistics like highscore. See *possible future directions.* If the game is interrupted the game data will not be saved.

1.4 Objectives and success criteria of the project

1. It should be possible to play a full game from start to finish. The player should at least be able to jump and shoot. During the game the player should interfere with enemies, abilities and some other neat special features.

2. The user should be able to optimise the game with some simple options like mute sound or/and effects. The user should also be able to select different maps to play.

1.5 Definitions, acronyms and abbreviations

All definitions and terms regarding the core JumpyDash game are as defined in the references section.

- **GUI**, graphical user interface

- **Java**, platform independent programming language.

- **JRE**, the Java Run time Environment. Additional software needed to run an Java application.
- **Framework**, pre-written reusable code.
- **libGDX**, a framework with a focus on game development for both desktop and mobile platforms.
- **Box2D**, a physics-based framework which is used to simulate the physics of particular bodies consisting of polygons, circles and edges.
- **Player**, the object that pictures the character controlled by the user.
- **Abilities**, items the player can pick up in order to change the behaviour of the player.
- **Health**, the players current health status.
- **Score**, the total amount of coins a player picks up during a game.

2 Requirements

2.1 Functional requirements

The user should be able to ...

1. Mute sound and/or effects
2. Select level to play
3. Start a new game
4. Control the player by
   (a) Jump
   (b) Shoot
5. Pause the game
6. Restart the game
7. Exit the application

2.2 Non-functional requirements

2.2.1 Usability

Usability is important. The user should be able to play the game within a very short period of time. The user should also be able to optimise the game (mute, unmute, etc ...).

The player should easily be able to see health, score and time passed during the game in the GUI.

2.2.2 Reliability

N/A

2.2.3 Performance

The application should run without any interference. There shall be no interruption or whatsoever during the game. Input for user shall be handled immediately.

2.2.4 Supportability

The application should be implemented using certain frameworks so that it easily can be converted to function on other platforms (Web, Mobile Apps, Tablets, etc . . .).

The application must have a good structure so that additions to the game can easily be implemented.

It should not be dependent on certain frameworks. It must be easy to replace the frameworks if needed.

2.2.5 Implementation

To achieve the usability and supportability desired the application will be implemented using JRE and the frameworks libGDX and Box2D.

2.2.6 Packaging and installation

N/A

2.2.7 Legal

N/A

2.3 Application models

2.3.1 Use case model

See APPENDIX for UML diagram and textual descriptions.

2.3.2 Use cases priority

1. Jump
2. Shoot
3. Start
4. Restart
5. Select level
6. Mute
7. Un-mute
8. Quit application

2.3.3 Domain model

See APPENDIX for domain model.

2.3.4 User interface

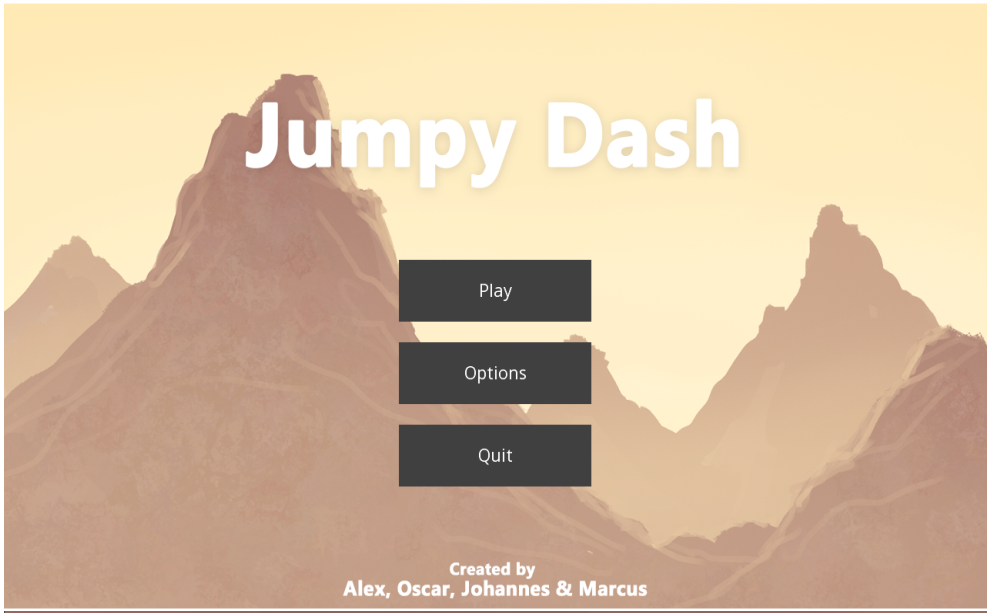
The application will be fixed at 1280 x 736 pixels with a non-scalable nor themeable GUI.

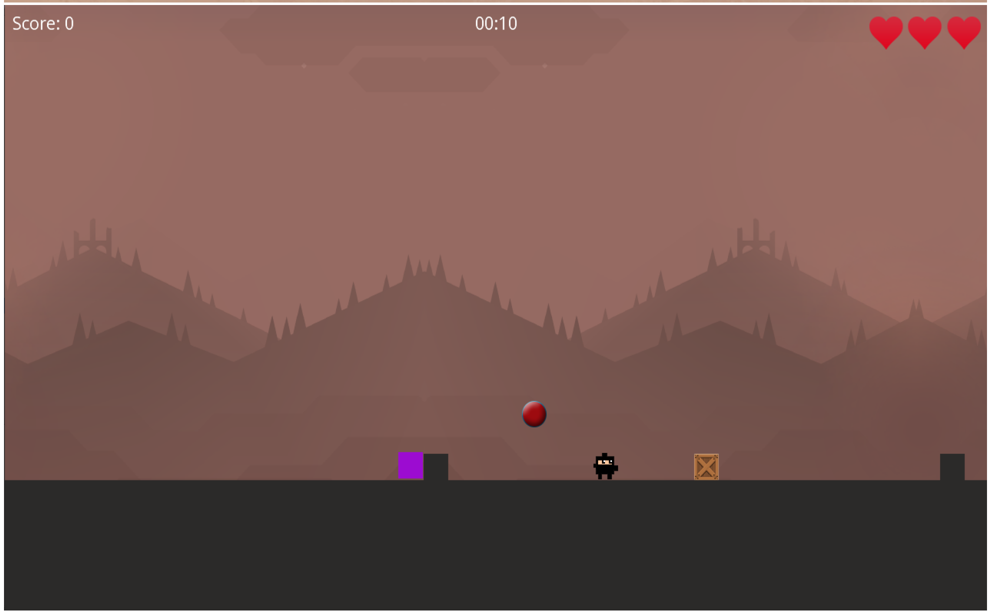
2.3.5 References

N/A

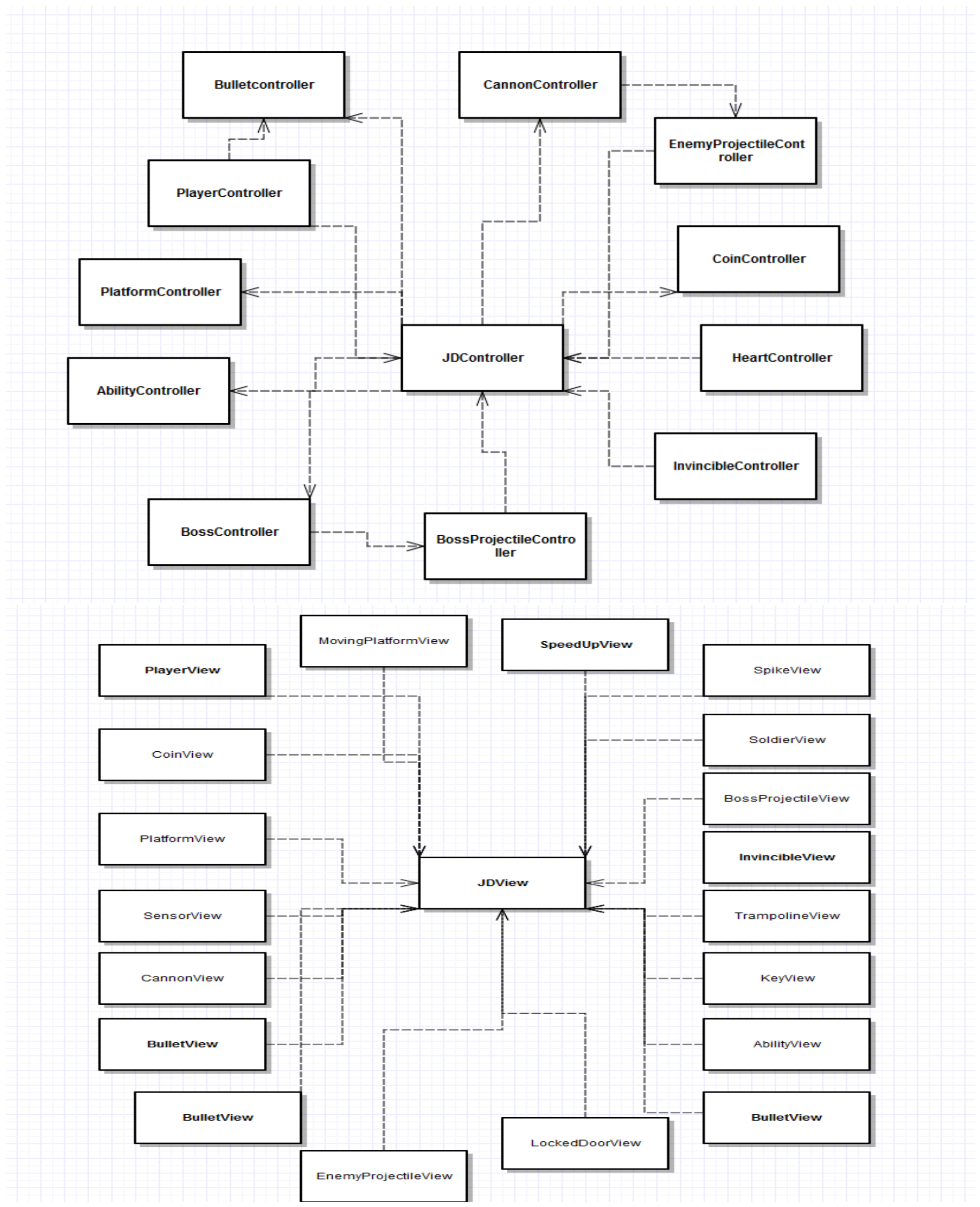
APPENDIX

GUI





Domain model


Bulletcontroller
CannonController
EnemyProjectileController
PlayerController
CoinController
PlatformController
JDController
HeartController
AbilityController
InvincibleController
BossController
BossProjectileController
MovingPlatformView
SpeedUpView
PlayerView
SpikeView
SoldierView
CoinView
BossProjectileView
PlatformView
JDView
InvincibleView
SensorView
TrampolineView
CannonView
KeyView
BulletView
AbilityView
BulletView
BulletView
LockedDoorView
EnemyProjectileView