Requirement Analysis Document for Ridiculous Rumble Royal

Version: 1.0

Date: 2015-05-30

Authors:
Tobias Alldén
Oscar Boking
Oskar Jedvert
David Ström

This version overrides all previous versions.

1 Introduction

This section gives a brief overview of the project.

1.1 Purpose of application

Ridiculous Rumble Royale is an attempt at making a fast-paced party game that is easy to pick up and play instantly. Combining the intense combat of classic games such as Super Smash Bro's and the accessibility of multiplayer games.

1.2 General characteristics of application

Ridiculous Rumble Royale is a fast-paced 2D multiplayer action-brawler which supports multiplayer over the internet. Up to four players fight in a closed arena until there is only one left.

1.3 Scope of application

Initially it's not possible to play the game alone, however an AI is something that might be implemented at a later point of time. Support for other devices than desktop can, but will not, be implemented. Character creation may be implemented at a later time.

1.4 Objectives and success criteria of the project

The goal of this project is to produce a fighting game which supports multiplayer over the internet.

1.5 Definitions, acronyms and abbreviations

AI = Artificial intelligence: "computer controlled players"
Action brawler = A game focused on fighting
2D = Two dimensional. For example: "Super mario" ‘

2 Requirements

2.1 Functional requirements

1. Join a game by providing an IP address and a port number.
2. Exit the application.
3. Host a game.
4. Move the character.
5. Use attacks.
6. Start a game.

2.2 Non-functional requirements

This section describes all the requirements of the system which are not vital to the execution of the program but still necessary for managing the program.

2.2.1 Usability

The game is considered “usable” when you can play a game with at least one other participant online.

2.2.2 Reliability

The application is considered reliable when the online multiplayer is stable under a decent internet connection, thus, not dropping connections. For lack of resources, time, and interest, the connection is arbitrarily deemed decent if the client have less than 100ms delay to the server.

2.2.3 Performance

The application is to perform without severe lag on a mid tier pc.

2.2.4 Supportability

Because of how the application functions: no long term support is needed. Something that might be taken into consideration is if there will be an option to use dedicated servers, by which a long term support system would be needed.

2.2.5 Implementation

The application is written in Java, using the external libraries LibGDX, Box2D and KryoNet, and uses the maven archetype for managing dependencies.

2.2.6 Packaging and installation

The installation is as simple as downloading the game jar and running it or, alternatively, one can use git to pull the project and build it oneself.

2.2.7 Legal

The program is not licenced and therefore defaults to be under copyright law (Phipps 2012).

2.3 Application models

2.3.1 Use cases

See appendix.

2.3.1 Use case model

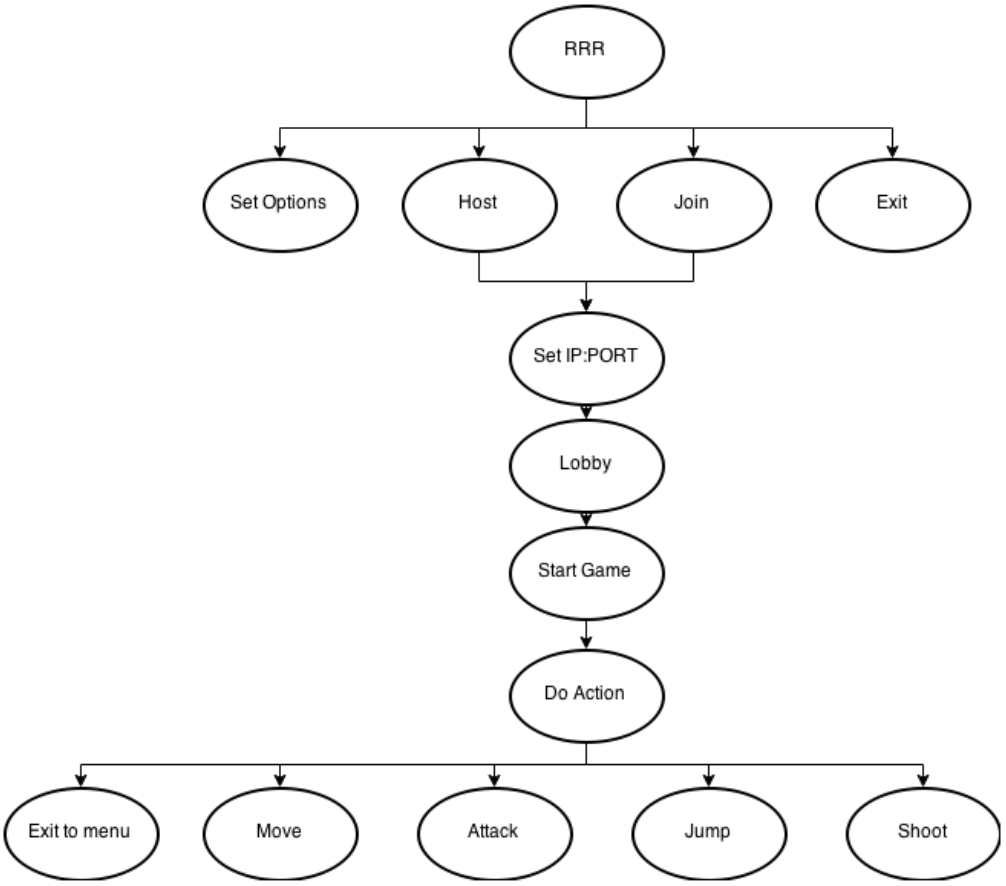

Figure 1. Use case Model

1. Exit button pressed at startmenu
2. Player starts application and hosts a game
3. Player starts application and joins a hosted game
4. W pressed during play state
5. A pressed during play state
6. D pressed during play state
7. Player HP is below 0
8. Player Respawn
9. Player performs a special attack
10. Player gets hit by a projectile
11. Player performs a base attack
12. Player gets hit by a base attack
13. Set options

2.3.2 Use cases priority

1. Exit button pressed at startmenu
2. W pressed during play state
3. A pressed during play state
4. D pressed during play state
5. Player starts application and hosts a game
6. Player starts application and joins a hosted game
7. Player performs a special attack
8. Player gets hit by a projectile
9. Player HP is below 0
10. Player performs a base attack
11. Player gets hit by a base attack
12. Player Respawn
13. Set options

2.3.3 Domain model

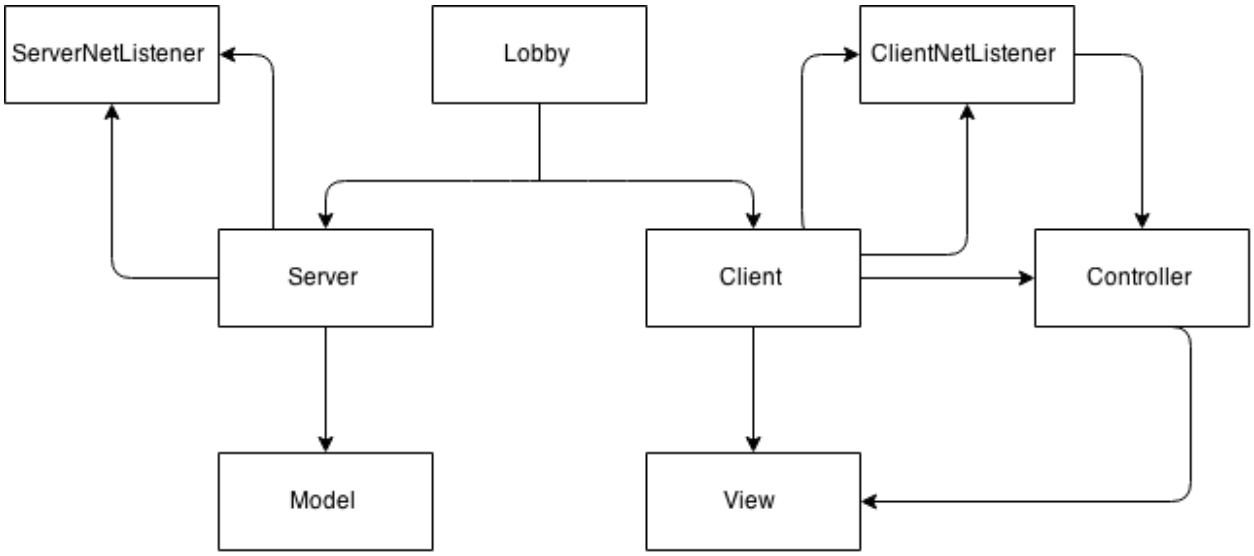

Figure 2. Domain Model

UML, possible some text.

2.3.4 User interface

The interface has five buttons: a play button for local games, a exit button to quit, a join button to join a hosted game, a host button to host a game, and a options button to set the game options. Possible future versions may include a character editor. The running game has no user interface at all but the thought is for a hp indicator, round counter, scoreboard to be implemented in a future version.

2.4 References

Phipps, S. (2012). Github needs to take open source seriously. I Infoworld hämtad 2015-05-27 från http://www.infoworld.com

Appendix - Use cases

MENU STATE

1. Exit button pressed at startmenu

USER	SYSTEM
User clicks Exit	
	Detects input, Exits program

2. Player starts application and hosts a game

USER	SYSTEM
Starts program	
	Displays main menu
Clicks on “host” button	
	Displayes port input screen
	Queries user for port number
Enters port number	
	Displayes the lobby
	Creates a server with the specified port
	Creates a client with the specified port and the ip “127.0.0.1” (localhost).

3. Player starts application and joins a hosted game

USER	SYSTEM
Starts program	
	Displays the main menu
Clicks on “join” button	
	Displays the ip input screen
	Queries the user to enter the ip
Enters the ip	
	Displayes the port input screen
	Queries the user to enter the port
Enters the port	
	Displayes the lobby
	Creates a client with the specified ip and port

PLAY STATE

4. W pressed during play state

USER	SYSTEM
User presses W	
	Maps input
	Checks if player is grounded
	Apply an upward force to the player’s body, specified by the constant “JUMP_FORCE_MULTIPLIER”

5. A pressed during play state

USER	SYSTEM
User presses A	
	Maps user's input
	Check if the specified player has a body in the world(not dead)
	Set horizontal speed of player body to the negative constant MOVEMENT_SPEED

6. D pressed during play state

USER	SYSTEM
User presses D	
	Maps user's input
	Check if the specified player has a body in the world(not dead)
	Set horizontal speed of player body to the positive constant MOVEMENT_SPEED

7. Player HP is below 0

USER	SYSTEM
	Removes player from world, increment player deaths, sets player alive boolean to false, sets should respawn if player respawn is enabled to true

8. Player Respawn

USER	SYSTEM
	checks if If respawn is enabled and player respawn timer is greater than the respawn time
	sets player hp to 100, sets player alive boolean to true, create player body in world

9. Player performs a special attack

USER	SYSTEM
Player presses attack button k	
	Checks if player can attack
	Adds player special attack to active attack lists, creates a body in the world for the attack with a linear velocity set to 10 if player is facing right and -10 if player is facing left

10.Player gets hit by a projectile

USER	SYSTEM
	Adds projectile body to impacted attacks, adds player to player damage taken map
	Body is removed from the world, attack resets, players health is reduced by the attack constant SPECIAL_ATTACK_DMG.
	Empties maps

11. Player performs a base attack

USER	SYSTEM
Presses attack button j	
	Checks if player can attack
	Adds player special attack to active attack lists, creates a body in the world for the attack.

12. Player gets hit by a base attack

USER	SYSTEM
	Adds attack body to impacted attacks, adds player to player damage taken map
	Body is removed from the world, attack resets, players health is reduced by the attack constant BASE_ATTACK_DMG.
	Empties maps