RAD - Requirements Analysis Document

1 - Introduction

1.1 - Purpose of the system

Aims to create a 2D Racing-game with multiple game-modes. Solely for entertainment purposes.

1.2 - General characteristics of the application

2D racing game with the camera from above. It has multiple game modes, including time-trial, player vs player and player vs computer-racing. Players drive around a track x amount of

laps. There will be a GUI to select game modes, maps, etc. Time and amount of laps will be displayed on the screen while racing.

Desktop stand-alone application written in Java for Windows/Mac/Linux

1.3 - Scope of application

Only simple controls, no power-ups or speed boosts. Can be played with arrow keys only. Se possible future extensions.

1.4 - Objectives and success criteria of the project

- Being able to drive a timed lap around a track.
- Being able to race against the computer
- Being able to select different maps and different vehicles.
- Being able to play against another player, using online multiplayer

1.5 - Definitions, acronyms and abbreviations.

Map - A driving track with surroundings

GUI - a graphical user interface

Java - platform independent programming language.

2D - Two dimensional graphics

Game mode - different ways of playing the game, with different goals

FPS (Frames Per Second) - how many times the screen is drawn upon per second

Time-Trial - A gamemode where the aim is to complete a lap around the track as fast as possible, trying to beat the fastest recorded time.

JAR - A runnable version of the program code, containing eventual resources

ZIP - A compressed archive containing a set of files

README - Text file containing solutions to common problems

JRE - Java Runtime Environment, software needed to run Java application.

AppWarp - Platform for online games, a place for games to connect people to each other.

2 - Requirements

2.1 Functional requirements

Players should be able to:

1. Start a new game
    a. choose map
    b. select game mode
    c. select a vehicle
2. Race
    a. Control the car's speed
    b. Turn the car
    c. Finish the race (enter the goal)
3. Receive race result and statistics
    a. Check time for lap and whole race
    b. See placement
4. Exit the program

2.2 Non-functional requirements

2.2.1 Usability

Application should be able to be used by users with all types of experience from novice to expert. The application will have a similar look to real life and therefore easy to understand.

2.2.2 Reliability

For online multiplayer the application will be relying on AppWarp servers for connecting players to each other.

2.2.3 Performance

The program should run smooth without any noticeable delay between user interaction and system response. A reasonable goal is that the game should never drop below 60 FPS.

2.2.4 Supportability

Each use case should have corresponding automated tests; verifying that it’s working as intended.

The program will be implemented to make releases to other platforms (e.g iOS/HTML5/etc) a possible future feature. The same applies for multiplayer-support.

2.2.5 Implementation

The application will use Java and therefore is platform independent. It also means that users will have to install JRE to run the application. For playing over network users will need a working network connection.

2.2.6 Packaging and installation

The application will be delivered in a ZIP-archive containing:

- Application code in a JAR-file (which contains resources such as icons e.t.c)
- A README-file containing brief instructions for installation and running the program

2.2.7 Legal

All source-code will be released as open source under the MIT license, excluding eventual third-party textures/resources.

2.3 - Application models

2.3.1 - Use case model

See APPENDIX for UML

2.3.2 - Use case priority

1. Vehicle moving
2. Move forward
3. Break/reverse
4. Turn left/right
5. Lap

2.3.3 - Analysis model

See APPENDIX for analysis model

2.3.4 - User interface

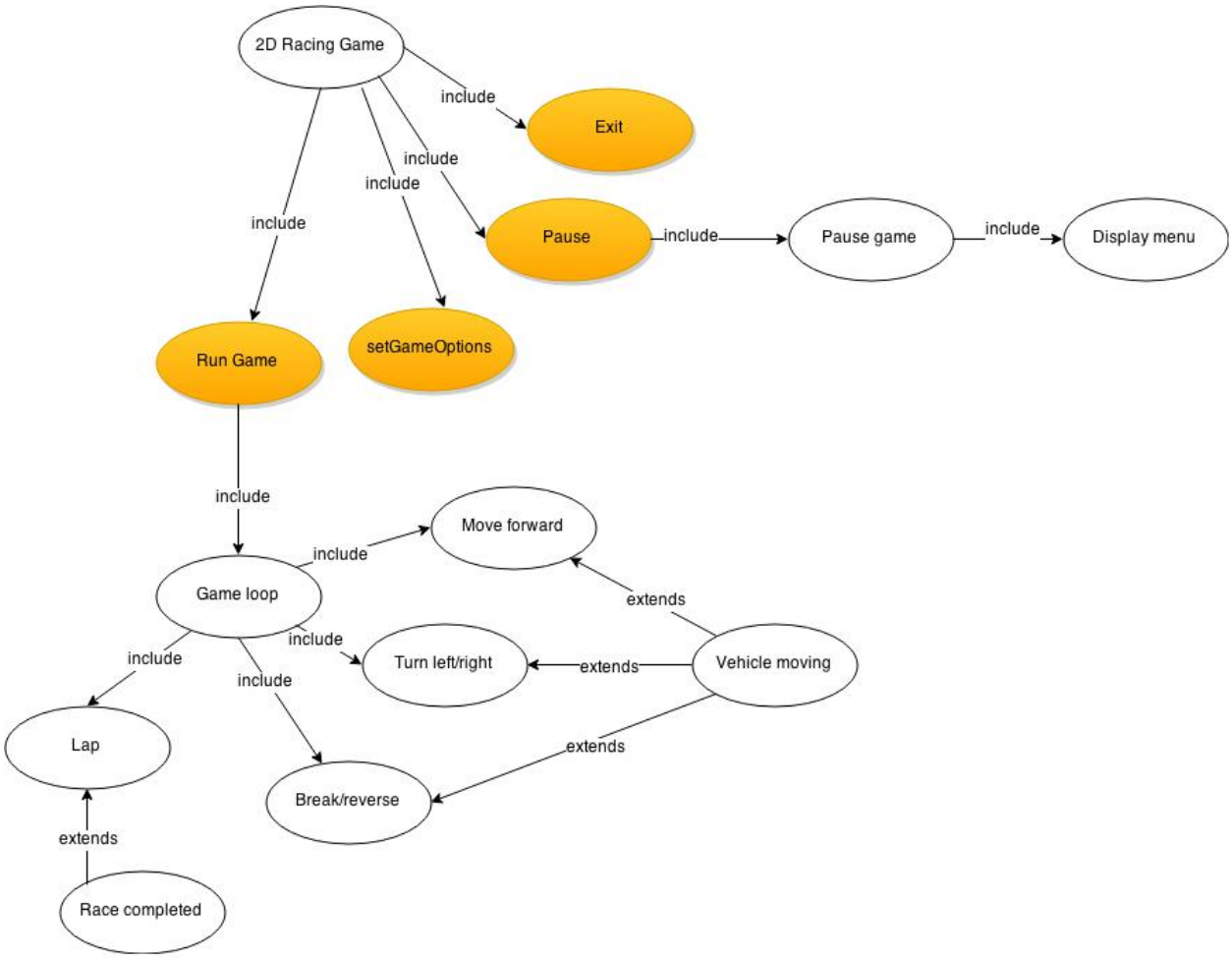
Application will have a standard GUI startup menu with choices and when a game is played the GUI will be a map with realistic racing environment.

3 - References

Badlogicgames. (2015). *LibGDX*
http://libgdx.badlogicgames.com/
Box2D (2015). *A 2D physics engine for games*
http://box2d.org/

4 - APPENDIX

4.1 Use cases overview



4.2 Use cases texts

Base case: Vehicle moving

Summary: Base logic which applies to all moving cases.
Priority: high

Normal

1	Actor	System
2	Performs case which results in vehicle moving	
3		Moves vehicle, with respect to collision, forward.
4		Move camera to follow vehicle

Alternate 1:

1	Actor	System
2	Stops performing user case which results in vehicle moving	
3		Progressively eventually apply inverse action, e.g decrease speed

Alternate 2:
Finishes a lap.

1	Actor	System
2	Player drives across the finish line, haven't driven all required laps (e.g. lap ¾)	
3		If player have crossed all required checkpoints; increase lap counter

Alternate 3:

Finishes race.

1	Actor	System
2	Player drives across the finish line and has driven all required laps (e.g. 4/4)	
3		Store score/time.
		Display a summary of the race and the player's score and buttons to navigate further.

Use case: Move forward

Summary: User presses arrow key forward to move vehicle forward

Priority: high

Extends: Vehicle moving [Always performs the logic of the base case]

Normal

1	Actor	System
2	Presses arrow key forward	
3		Increase vehicle speed

Alternate

1	Actor	System
2	Holds arrow key forward down	
3		Progressively increase vehicle speed

Exceptional

1	Actor	System
2	Holds arrow key forward down while moving at max speed	
3		Do not alter speed

Use case: Break/reverse

Summary: User presses arrow key back to reduce speed, or alternatively reverse if standing still

Extends: Vehicle moving [Always performs the logic of the base case]

Priority: high

Normal

1	Actor	System
2	Presses arrow key back while moving	
3		Reduce vehicle speed

Alternate

1	Actor	System
2	Presses arrow key back while standing still	
3		Decrease speed so it becomes negative

Exceptional

1	Actor	System
2	Presses arrow key back while reversing at max speed	

3		Do not alter speed

Use case: Turn left/right

Summary: User presses the left or right key to change direction.

Priority: High

Normal

1	Actor	System
2	User presses right/left while moving forward	
3		wheels turn (this is something we have to look more into when implementing, hard to specify exactly)
4		angle the car turns (next update) is increased.

Alternate

1	Actor	System
2	User presses right/left while at maximum turn angle	
3		Wheels don’t turn.
4		The angle which the car turns next update is the same as it was before.

Exceptional

1	Actor	System

2	User presses both left and right at the same time	
3		Nothing happens as both actions cancel each other out.

Use case: Pause

Summary: Pauses the game when the escape key / p key is pressed

Priority: Medium

Normal

1	Actor	System
2	User presses the escape key during a race/lap	
3		The game pauses. Stop all actions
4		Menu with options like start over, quit ect appears.

Alternate

1	Actor	System
2	User presses the escape key at another time	
3		Nothing happens.

Exceptional

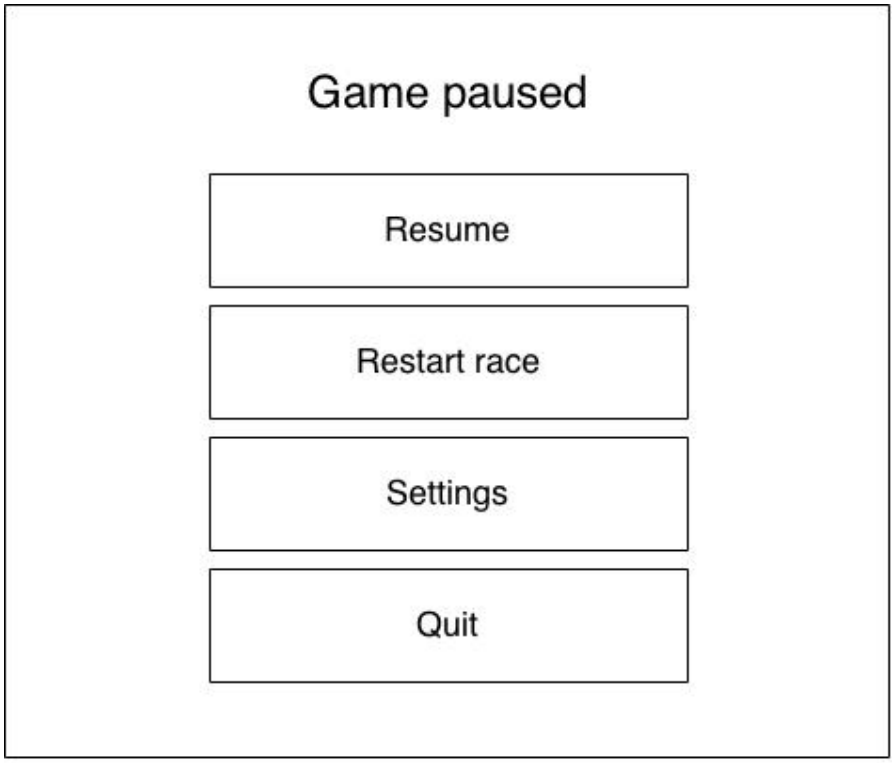
1	Actor	System
2	User presses the pause key while the game is paused	

3		Game unpauses
		Menu goes away.

4.3 GUI

Preliminary GUI.

Ingame gui



Paused race menu mockup

4.3 Analysis model

Preliminary analysis model.