ROLLER MADNESS-THE BALL GAME

~~"WHEN A BALL GO CRAZY"~~

SOFTWARE REQUIREMENT SPECIFICATION FOR THE ROLLER MADNESS – THE BALL GAME

SIDHARTH PADHI	*18CSE187*
ARUPA PARIDA	*18CSE013*
P.ANJALI PATRO	*18CSE183*

TABLE OF CONTENTS

1. INTRODUCTION:

1.1 PURPOSE:

This Software Requirement Specification (SRS) document is intended to give a complete overview of a Computer Game Project- Roller Madness, including the game mechanics, user interface and story therein. The SRS document details all features upon which Roller Madness have currently decided with reference to the manner and importance of their implementation.

1.2 DOCUMENT CONVENTIONS:

As the development team is responsible for the SRS document, no ambiguity arises from its usage. There is a clear distinction, however, between the use of the words “player” and” character”. The "player" is the human being interacting with the game in the real world, while the “character" is the in-game player avatar being manipulated. This document is too printed on A4 paper in Calibri font. Normal text is size 14 black, while section headings are size 16 bolded white. Subheadings are bolded size 16 black, and third subheadings are bolded size 12 black.

1.3 INTENDED AUDIENCE AND READING SUGGESTIONS:

The SRS document also gives testers and project managers a way to ensure the game's adherence to original vision. Although the document may be read from front to back for a complete understanding of the project. It was written in sections and hence can be read as such. For an overview of the document and the project itself, see Overall Description (Section 2). For a detailed description of the game play elements and their interaction with the character, see System Features (Section 3). Readers interested in the game play interface and navigation between different front-end menus should see External Interface Requirements (Section 4). Technical standards to which the team will hold the project are laid out in Other Nonfunctional Requirements (Section 5). The development schedule, meanwhile, will be maintained in the Key Milestones (Section 6).

1.4 PROJECT GOAL:

Roller Madness is a simple Computer Game in which the player has to use the character “BALL” to move around the whole environment in order to tackle the enemies using the environment and collect the required amount of coins to cross the level and move on to the next level.

Roller Madness also requires a little hand of a gamer in order to fully control the “ball” character and move throughout the environment, otherwise this game will drive you crazy. It's available for any level device. Roller Madness is a fun game to have in your library for these occasions you need a little help passing the time.

Roller Madness's main menu is minimal by design with an option to simply start the game. It contains two levels. The level becomes harder with the increase of number of enemies.

The controls are simple as you only need to use W A S D key to control the player and collect coins as fast as you can avoiding the enemies from colliding with the player and reach the goal to go to next level.

2. OVERALL DESCRIPTION:

2.1 PRODUCT PERSPECTIVE:

While the game itself is it a continuation of any predecessor, is implementation is made possible through the game development tool Unity Engine and its ability to compile and build projects for use on the Computer Devices.

Since there are tons of games in today's world, one of our goal while designing the game is to be able to let the user understand the gameplay at the first look no matter how often and how least he opens the game. That is why there no more UI or much elements to get confused with, simply enter the game and play.

2.2 PRODUCT FEATURES:

The following is a summary of the major features implemented in the game. They are separated into categories based on those that are necessary for the game function.

2.2.1 FUNCTIONAL FEATURES:

- **MOVEMENT:**
    - The central mechanic of our game, allowing the player to change their perspective and orientation through the use of W A S D keys.

- **DEATH ZONES:**
    - Areas that trigger the restart of the level that is getting out the environment plane.

- **TITLE SCREEN:**
    - The first viewable screen upon starting up the application containing buttons for play game option.

- **ENEMIES:**
    - These are the alien bots spawning at random position and follow the character. If the player collides player dies!!

2.3 USER CLASSES AND CHARACTERISTICS:

Our control scheme is designed to be intuitive and our game-play innovative and unfamiliar. Therefore, experience with games will not be a major factor in determining who is able to play: players of any age or skill level should be able to pick it up. However, as with any game with a large amount of fan base, there will be a natural divide between casual and hardcore players. These two classes will naturally differentiate themselves through the use of the game play mechanics and the structure of the levels. The abilities and characteristics distinguishing hardcore players will include:

- **CONTROL:**
  The ability to use the angular velocity, torque and forward movement with particular amount of force to move freely in the environment without falling to death zones and tackling the enemy.

- **GOAL:**
  To successfully complete the goal of collecting the coins before getting hit by the enemies.

3. PROJECT CONTEXT:

This includes Milestones, Development Strategy, Risk Analysis and Tools, and Version Control Procedures.

3.1 MILESTONES:

MILESTONE	**START**	**FINISH**
Pre Study	20th OCT, 2020	27th OCT, 2020
Design	27nd OCT, 2020	31th OCT, 2020
Implementation	1st NOV, 2020	10th NOV, 2020
Content and Demonstration preparation	10th NOV, 2020	16th NOV, 2020
Demonstration	17th NOV, 2020	21st NOV, 2020

Evaluation	22nd NOV, 2020	25th NOV,2020
Finalization	26th NOV, 2020	30th NOV , 2020

3.1.1 PRESTUDY:

This phase consists of gaining an insight into different subjects that concern this project. This preliminary study will also include creating the research part of the written report.

3.1.2 DESIGN:

This phase consists of gathering requirements and creating a software architecture for the game. The phase will be revisited several times during the implementation phase.

3.1.3 IMPLEMENTATION:

This phase consists of the iterative process of creating the game. The overall architecture should be implemented and then the actual code should be written, along with tests and a thorough documentation. The implementation phase is divided into three sprints.

MILESTONE	**START**	**FINISH**
Sprint 1	20th OCT, 2020	30th OCT, 2020
Sprint 2	2nd NOV, 2020	15th NOV, 2020
Sprint 3	16th NOV, 2020	30th NOV, 2020

3.1.4 CONTENT AND DEMONSTRATION PREPARATION:

This phase consists of preparing for the demonstration. The content of the game must also be prepared. This includes the creation and finalizing of task.

3.1.5 DEMONSTRATION:

This is a one day test consisting of letting several groups compete against each other. The experiences and results from this demonstration will form the basis for the evaluation.

3.1.6 EVALUATION:

This phase consists of evaluating the experiences and results from the demonstration, and will form part of the basis for the platform design.

3.1.7 PLATFORM DESIGN:

This phase consists of designing the platform. Based on the evaluation of the results and experiences from our game, as well as any conclusions made on the background of our research questions.

3.1.8 FINALIZATION:

This phase consists of finalizing the thesis document.

3.2 DEVELOPMENT STRATEGY:

In order to implement the prototype, we are going to use a combination of the waterfall and the iterative software development strategy. We will make an overall design of the application using a waterfall approach which means we will have a good idea about what the system should look like, and what we are going to implement. The implementation will be based on an iterative software development strategy. This means that the implementation process will be divided into smaller segments of design, implementation, and testing thus allowing for a better pumping of the time used on implementation.

3.3 RISK ANALYSIS:

This section is about the risks in this project. They are separated into two parts, general risks and demonstration specific risks.

3.3.1 GENERAL RISKS:

- **INTERNAL CONFLICTS:**

  Conflicts between team members can occur, and is even more likely since we will be working closely together over a long period of time. It is therefore important to communicate openly, and handle issues internally before they become too much to handle, Should this be insufficient one could always seek help from the supervisors, or in the worst case scenario- dissolve the group.

- **COMPUTER DEVELOPMENT DIFFICULTY:**

Developing a computer application game is something new for us and most of us are not completely familiar with it. If we spend too much time in understanding the framework, development will take more time than expected and other parts of the project will suffer. We can avoid this by planning accordingly and allocate sufficient time to understand Android, as well as start development early.

- **BAD PROTOTYPE DESIGN:**

  The survey and evaluation tells us that the concept for the prototype is bad. This is a risk that can make us unable to design a satisfactory platform, which would have a huge impact on the project. One way to reduce the probability of this occurring is to interview experts to verify the learning aspect of the concept. The impact can be reduced by making a survey that will give us useful information even if the design of the prototype is bad.

3.3.2 DEMOSTRATION SPECIFIC RISKS:

- **BATTERY:**

  The low battery for some devices may be a well-known issue. We need to keep in mind the usage of battery with using other things as low as possible.

- **UNCLEAR TAKS:**

  When creating task descriptions, one runs the risk of taking tasks that are unclear. This can bad to frustration from participant and they may get stuck. This can be avoided by scrutinizing tasks and making sure that they are clear and have no ambiguity. Should that not work, the participants could always contact a demonstration supervisor and receive clarification.

3.4 TOOLS:

Programming Language: Processing

Programming IDE: processing development environment (PDE)

Software: Unity Engine, Visual Studio & Blender.

Report Writing: MS Word

3.5 VERSION CONTROL PROCEDURES:

The following guidelines should be followed when using Git:

a) Perform updates before committing any changes
b) Only commit working code
c) Several people should not work in the same file at once.

4. SYSTEM FEATURES:

4.1 MOVEMENT:

4.1.1 DESCRIPTION AND PRIORITY:

This mechanic gives full control over the orientation of the level relationship to the character. Using the movement keys the player controls the forward and angular movements.

4.1.2 STIMULUS/ RESPONSE SEQUENCES:

Step 1: The player only have to use W A S D keys to move the player.

4.2 DEATH ZONES:

4.2.1 DESCRIPTION AND PRIORITY:

Death zones are the opposite of level completion zones in that when touched, they send the character back to the starting point of the level.

The only death zone is the outside the plane. This zone increase the game's difficulty and incentivized good performance via negative reinforcement. They should be prioritized in level design, as they bring the game-play from "playable" to "enjoyable".

4.2.2 STIMULUS/ RESPONSE SEQUENCES:

Step 1: The character comes into contact with a Death Zone.
Step 2: The character is immediately returned to the beginning of the level.

4.3 TITLE SCREEN:

4.3.1 DESCRIPTION AND PRIORITY:

The title screen is the screen the player will see every time upon playing the game. Through this interface the player can start the level and play again the same level or move to next level.

4.3.2 STIMULUS/ RESPONSE SEQUENCES:

Step 1: The player launches the game from their portable device.
Step 2: The start screen starts and appears, prompting the player with a button: "Play Game”.
Step 3: The player presses one of the buttons, triggering tits respective function.

5. EXTERNAL INTERFACE REQUIREMENTS:

5.1 HARDWARE INTERFACES:

Roller Madness is a computer game application designed specifically for the computer platform and is functional only on computer, laptop or desktop.

Roller Madness game is built for any low end desktop or computer system and all subsequent releases.

5.2 SOFTWARE INTERFACES:

Roller Madness is to be developed under the Windows operating system using a series of game development tools

a) Universal Windows Platform
b) Unity Engine: Primary game development tool using C#
c) Visual Studio: Used for programming C# integrated with Unity Game Engine.
d) PDE: Processing Development Environment

6. OTHER NON-FUNCTIONAL REQUIREMENTS:

6.1 PERFORMANCE REQUIREMENTS:

As this game uses very medium level graphics and hence there will be no issue of performance. But with very old Windows we might incur some game lags.

6.2 SAFETY REQUIREMENTS:

Roller Madness will not affect or damage any of the other apps installed on player’s device. The game will not cause any overheating issue. It won’t give any potential harm to the player.

6.3 SECURITY REQUIREMENTS:

Roller Madness doesn’t ask for any personal information. No authentication required from player’s side. The player simply has to download and play,

7. IMPLEMENTATION PHASE:

7.1 DESIGN:

7.1.1 GAME START

When the game is installed, everything is set up. The game will start at the first level every time the player reopens the game.

7.1.2 TASK FLOW:

The user initially encounters the welcome screen containing the game name and a button to play. Once the play button is triggered the game starts and the player need to use the movement keys to play.

7.1.3 COHESIVENESS:

High cohesion leads to the reliability, robustness and reusability. Since it is a game project therefore each module, each class and each file in the entire project is somehow dependent upon one another and can't be completed independently. In order to detect the collision of the main character with the enemies or walls and any other obstacles we would have to access the location of both the actual character and any of the colliding objects.

Similarly if we have to take any coins we have to detect the collision and trigger the respective behavior in the game.
In almost every game the code is most of the time cohesive because each object in the game has to obtain the information or fields of the other objects in order to make the game physics work therefore the cohesion of this project is high in comparison to the usual software projects.
Reducing the cohesiveness might Lead the game to look like a fake physics and not a can work, hence high cohesiveness in this project can be said as mandatory.

7.2 IMPLEMENTATION:

7.2.1 GAME START:

After the game is installed the game is all set up with the environment. When the user starts the game, the game state is loaded by calling the loadGameState() method. Then the new game page gets loaded and welcome screen appears. The welcome screen contains only a canvas on which the text field and the background image is set up. The buttons are also placed upon the canvas in each screen.

7.2.2 TASK FLOW:

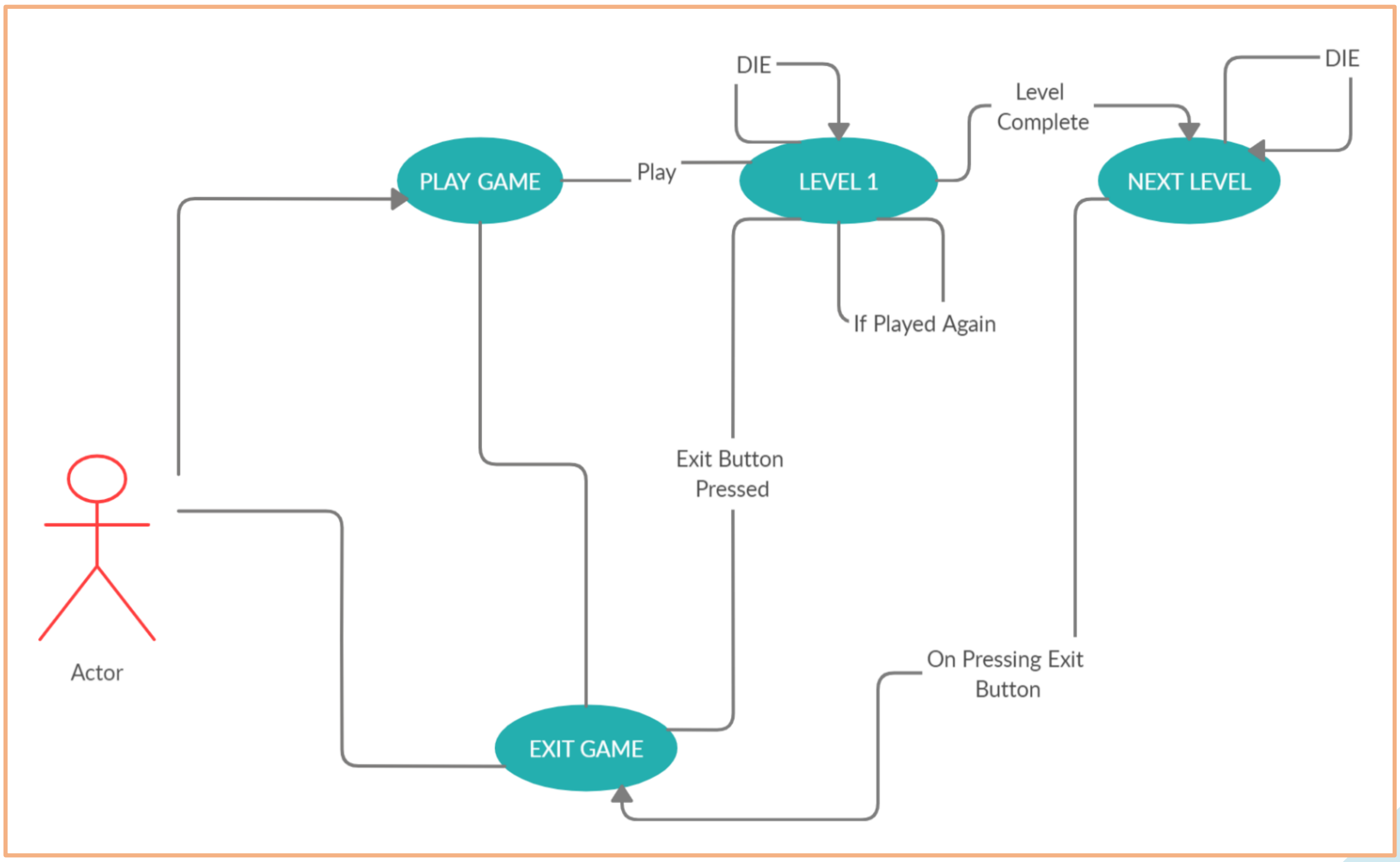
When the player push the play button then the method running behind gets trigger and load the next scene which it is directed through code. Each and every elements are triggered likewise.

7.3 TESTING:

7.3.1 TASK FLOW:

Testing if the task flow was done incrementally. The overall task flow was created and tested first.

APPENDIX A: USE CASE MODELLING


DIE
Level Complete
DIE
PLAY GAME
Play
LEVEL 1
NEXT LEVEL
If Played Again
Exit Button Pressed
Actor
On Pressing Exit Button
EXIT GAME