Bilkent University

Department of Computer Engineering

**CS 319 Term Project**

*Section 1*

*Group 1A*

*Walls and Warriors*

Analysis Report

Project Group Members

1. Dawood Muzammil Malik
2. Deniz Yüksel
3. Faruk Oruç
4. Mert Soydinç
5. Muhammed Salih Altun

Supervisor: Eray Tüzün

Table of Contents

[1 Introduction 2](#_Toc531111546)

[2 Proposed System 3](#_Toc531111547)

[2.1 Overview 3](#_Toc531111548)

[2.2 Game Objects 5](#_Toc531111549)

[2.3 Gameplay 6](#_Toc531111550)

[2.4 Ways to Play 7](#_Toc531111551)

[3 Functional Requirements 7](#_Toc531111552)

[3.1 Additional Requirements 10](#_Toc531111553)

[4 Nonfunctional Requirements 11](#_Toc531111554)

[5 Pseudo Requirements 12](#_Toc531111555)

[6 System Models 13](#_Toc531111556)

[6.1 UML Diagrams 13](#_Toc531111557)

[6.1.1 Use Case Diagram 13](#_Toc531111558)

[6.1.2 State Diagram 18](#_Toc531111559)

[6.1.3 Sequence Diagrams 21](#_Toc531111560)

[6.1.4 Class Diagram 25](#_Toc531111561)

[6.1.5 Activity Diagram 28](#_Toc531111562)

[6.2 User Interface 30](#_Toc531111563)

[6.2.1 Navigation Paths 30](#_Toc531111564)

[6.2.2 Screen Mockups 31](#_Toc531111565)

[7 Improvement Summary 39](#_Toc531111566)

[8 Glossary & References 41](#_Toc531111567)

# Introduction

Walls andWarriors, by the group “404 Name Not Found”, is a software implementation of the famous classical game of the same name (*Walls and Warriors* [1]). Walls and Warriors is a single player game that challenges the players’ brain power with a wide variety of challenges.

In total, the game has one game board, seven knights (three blue, four red), four walls of different predefined shapes and one blue tower. Out of these, the walls are the only object that the player is allowed to interact with. With the help of those four walls, the player has to make sure that the castle of blue warriors is safe from the red warriors. For that, the player must arrange the walls on the game board such that the blue knights are protected within the castle and the red invaders are kept out of the castle walls. Given that each challenge has only one correct solution, *Walls and Warriors* aims to challenge both children and adults.

In our implementation of the game, we aim to add some additional features to make the game more interesting and different than the classic version. The levels will now be timed and scores will be calculated based on the time taken to complete the challenge and the difficulty level of the challenge. Game experience will be pleasing as the player will be motivated with additional bonuses like coins, diamonds and trophies. The game will include different themes/maps and characters which aims to bring versatility in the user experience (in terms of view). More information regarding themes and characters can be found in *Section 3* of this report under the *Options* description.

With the introduction of the Sandbox Mode, we aim to give power to the user and let him create his own challenges. The user will now be able to specify the dimensions of the game board, choose the number of red and blue knights, and design walls of their own choice. Once valid, the new challenge will be saved in the program and can be played anytime.

This report contains a general outline of our project including overview, functional and non-functional requirements, dynamic and object models. Following that, are the mock-ups of the game’s user interface giving a taste of the game.

# Proposed System

# Overview

We will implement a Desktop based Java application to simulate *Walls and Warriors* in our software version. The main properties of the game are as follows:

* Each user will have their own identifier name.
* The user will have a total of 80 challenges to solve of varying difficulty levels (Starter, Junior, Expert, Master, Wizard) in the classic mode, plus any user created challenges in the sandbox mode.
* Each *Walls & Warriors* puzzle grid in the Classic mode will include:
  + A 4x5 game board without the edges
  + 3 blue and 4 red knights. Not all the knights are used in each level.
  + One blue tower.
  + Four walls of predefined shapes.
* Each *Walls & Warriors* puzzle grid in the Sandbox mode will include:
  + A game board (without corners) of the dimensions specified by the user.
  + ***w*** number of red knights,
  + ***x***number of blue knight,
  + **y** blue towers,
  + **z** walls of predefined shapes.

**Note**: the values of **w, x, y** and **z** will be specified by the user while creating the level in the sandbox mode.

* **Game Algorithm**

The original Walls and Warriors game have set number of levels and a single solution for each level. This means that for the original game there is no need for an algorithm as long as you specify the solution for each level. However, our game features a sandbox mode for the player; the newly created levels in the sandbox mode can have vastly different properties like having more than one solution or having a much bigger board. This means that in order to have such a feature we are going to need an algorithm that can detect win conditions and illegal moves in unknown levels.

* **Database**

As mentioned above, the classical mode of the game will have 80 challenges. Those challenges, along with the player’s progress and rewards must be saved so that they could be used whenever needed. For that, Walls & Warriors will have a very simple and efficient data management system. Famous for its better speed as compared to MySQL, we will be using *MongoDB* [2] as our database.

# Game Objects

The game objects are the physical pieces that the player can interact on the screen.

* + 1. **Timer**

The timer is a game object that is used to measure scores of the local players. The timer starts when the user starts the game by clicking the *Start* button.

* + 1. **Board**

The classical game board is a 4x5 rectangle without the squares on the corners. Therefore, there are only 16 squares in total where the knights and tower will be placed. A knight takes 1 square whereas the tower takes 2 squares. Walls can be placed between two adjacent squares.

* + 1. **Knight**

A knight can be either red or blue. There cannot be more than one knight placed on a single square. Each level uses different number of red and blue knights. Hence, all the knights does not need to be present on the game board for each level. For example, one challenge may contain two blue knights and one red knight, whereas the other may contain just one blue and four red knights.

* + 1. **Wall**

Walls are the only game objects that player can interact. In the classic mode, there are just four walls of different shapes (taken from the original game) and cannot be modified in any way. For some levels, one single piece of wall is already placed on the grid when the game starts. In that situation, that position of that specific wall cannot be changed by the user.

In the sandbox mode will give the users the liberty to design walls of their own choice (in terms of their shapes and number).

* + 1. **Tower**

The tower is a game object that is initially placed on two squares. The user cannot interact with the tower.

# Gameplay

The original board game has 80 challenges of varying difficulty. All the challenges will be stored in the database. The user can only interact with the walls. In the game, the tower, a wall (in some challenges) and the appropriate number of knights are placed before the timer starts. The user then has the options to choose one of the four walls with a mouse click. Once the wall is selected, the user can rotate the wall 90 degrees with corresponding keyboard keys and place the wall into a space between the squares with the next mouse click. The wall then will be placed to the particular space. Once four walls are all placed and all blue knights are inside while the red knights are out, the timer will stop and the player will be presented with a message informing him about the time he took to complete the challenge, his score and his rewards in terms of coins, diamonds and trophies. If the player places all the four walls and the game is not complete, the game will continue without any warning and the timer will continue.

# Ways to Play

* **Classic Mode**

The classical game is the style of the original game, adapted to a desktop application. The game consists of a 4x5 puzzle grid, on which tower and knights (and wall(s), if applicable) are already placed when the game starts. In order to win the game, the player must place the available walls in such a way that red knights would not have a path to enter the castle. See the “2.3 Gameplay” section for more details.

* **Sandbox Mode**

The sandbox mode enables the players to create their own levels and become the editor. In this mode, the editor can specify the board dimensions and by using knights, towers and walls of their own choice, try to come up with an exciting challenge for their friends and family. This mode will enable the user to design their own walls between the grids. However, the solution has to be valid in order for the game engine to confirm it. For that, the game will make use of an algorithm that will check if the given solution is valid or not.

# Functional Requirements

We will provide the following functionalities to the game:

* **Playing game**

After completing the registration, the user will be taken on the *Play Game* menu where he will see the two game modes. The consequent screens will be based on what the user selects now. The two game modes are as follows:

* + **Classic Mode**

The levels screen will display all the available challenges, regardless of their states (locked or unlocked). When the user starts the game for the first time, he will only be able to play the first level. New levels will be unlocked as the player completes the uncompleted challenges. As mentioned above, a timer will be used to record the time it takes for the user to complete a given challenge. Depending on the time and the difficulty level of the challenge, the user will be rewarded in shape of coins/diamonds and/or trophies. This mode provides the following functionalities to the user:

* + - Place walls on the game board
    - View unlocked and locked levels
    - View available rewards
    - Leave puzzle
  + **Sandbox Mode**

The user can create his own levels in this mode. (More details about the *Sandbox Mode* can be found in section 2.4.2). The sandbox mode provides the following functionalities to the user:

* + - Create a game board of user-specified dimensions
    - Place user-specified number of blue knights on the game board
    - Place user-specified number of red knights on the game board
    - Place user-specified number of blue tower on the game board
    - Design and place walls of user’s choice on the game board
* **Settings**

Here, the user can change the settings pertaining to the gameplay features of *Walls and Warriors*. This option provides two basic functionalities:

* + **Change Audio Settings**
    - The user can adjust the volume of the sound track and/or background music.
  + **Change Video Settings**
    - The user can change settings related to the visual effects of the game.
* **Options**
  + **View Instructions**

This screen is accessed from the main menu. Here there is a number of instructions to help the player understand the game. These instructions include written sentences, informative pictures and link to tutorial videos.

* **View About Us**

Here there is general information regarding the group members and our game. Here the player can report bugs and send their criticisms about the game to us.

* **View Credits**

Here, the user can see the respective contribution of the developers.

* **Reset Game**

The user may choose to reset everything and start over from the registration window! This button will erase all information about user’s progress and the game will start assuming it was launched for the first time.

* **Quit Game**

The user will have the option to exit the game by either selecting the *Quit Game* option from the main menu or by just clicking the red cross on the game window.

# Additional Requirements

Here are the additional functional requirements added in iteration 2**.**

* **Registration**

As soon as the user downloads the game and opens it for the first time, a registration window will appear. Here, the user will personalize the game by choosing a name. From this point on, the game will save this player’s progress and information and the registration window won’t appear again unless the user decided to reset the game or create a new account.

* + **Go to My Dashboard**
    - **See Achievements**

The user will be able to see his coins, diamonds and trophies. In general, coins can be considered as total score or experience points. They cannot be spent. Diamonds are like virtual currency that can be used to unlock hints, solutions, or unlock different themes/maps. Trophies are awarded when the user achieves something for the first time. For example, if they user manages to complete the challenge in the first 30 seconds, he will receive a trophy title *The Flash*.

* **Change Themes**

The user can select from a wide range of unlocked themes. The default theme would be classic black and white theme.

* **Unlock Themes**

The user can unlock themes using diamonds to use them when they play the game. We will have the following themes in our game:

* + - Volcano
    - Ocean
    - Forest
    - Desert
* **Change Characters**

In order to make the game more attractive for children, user will be able to change the characters. For example, in place of blue and red knights, we can have cats and mice representing *Tom and Jerry*.

The ‘Go To My Dashboard’ additional requirement is part of ‘Options’ functional requirement.

# Nonfunctional Requirements

Since nonfunctional requirements have been completely revised, no additional requirements section will be provided in this part.

* **Usability**

Any user playing *Walls and* Warriors for the first time should be able to understand the dynamics of the game within an hour, without reading the user manual. This can be achieved by letting some children play the game.

* **Reliability**

Connectivity issues with the database or incorrect inputs by the user (e.g. accessing locked challenges, placing walls at invalid positions) should not cause the program to crash. Instead, errors should be handled by the program by displaying appropriate messages for the user.

* **Performance**

In order to enhance the user experience, it is important that the program performs up to a specific standard. For example,

* + The frame rate should be at least 30. Frame rate can be monitor directly from the graphic engine.
  + The maximum response time between click and reaction must be two seconds. Writing methods to calculate the response time can be used to test this requirement.
  + Data retrieval from the database must not exceed two seconds. Again, writing methods to calculate the response time can be used to test this requirement.
* **Platform**

The game must run in all systems with Windows 8 and above. This requirement can be tested by installing the game in Windows 10 environment and checking if it works properly.

# Pseudo Requirements

* The software will be developed in the Java programming language.
* Software will be handed over as a single executable file in the JAR format.

[ this space is left blank intentionally ]

# System Models

# UML Diagrams

# Use Case Diagram

*Figure 1 Use case diagram for Walls & Warriors*

Imgur link for better resolution:

<https://imgur.com/4Pmpq0q>

**Use Case #1**

Name:

* Select Game Mode

Participating Actor:

* Player

Stakeholder and Interests:

* Players wants to play the game or create his own level in the sandbox mode.

Entry Condition:

* Player has the game running.
* Player is on the main menu.
* Player clicks on the *New Game* button.

Exit Condition:

* Player clicks the *Quit Game* button.
* Player completes all the levels.

Main Flow of Events:

1. Player selects *New Game* option from the main menu.
2. The system responds by displaying the *New Game menu*.

Alternate Flow of Events:

1. If user wants to return to main menu:
   1. Player selects *Back* button.
2. If user wants to play the game:
   1. Player selects *Classic Mode* button (Use Case #2).
3. If user wants to create his own level in the sandbox mode:
   1. Player selects *Sandbox Mode* button (Use Case #2).

**Use Case #2**

Name:

* Select Classic Mode

Participating Actor:

* Player

Stakeholder and Interests:

* Player wants to play a challenge in the classic mode.

Entry Condition:

* Player is on the *New Game* menu.
* Player clicks *Classic Mode* button.

Exit Condition:

* Player clicks the *Back* button to return to main menu.
* Player quits the game.

Main Flow of Events:

1. Player clicks *Classic Mode* button.
2. System shows the levels the player is eligible to play.
3. Player selects the level he wishes to play.
4. The system loads and displays the game board and starts the timer.
5. Player tries to complete the challenge by placing walls in appropriate places.
6. System displays player’s score and rewards on screen when the game ends.

Exceptional Cases:

* User can modify audio settings while playing the game.
* User can view game instructions while playing the game.

**Use Case #3**Name:

* Select Sandbox Mode

Participating Actor:

* Player

Stakeholder and Interests:

* Players wishes to create a level of his own.

Entry Condition:

* Player is on the *New Game* menu.
* Player clicks *Sandbox Mode* button.

Exit Condition:

* Player clicks the *Back* button to return to main menu.
* Player quits the game.

Main Flow of Events:

1. Player clicks *Sandbox Mode* button.
2. System asks user to enter dimension of the game board.
3. User enters the dimension of board he wants.
4. System responds by displaying the board of the dimensions entered.
5. User places knights, towers and self-designed walls on the board.
6. When the user is done, he submits his challenge.
7. System runs the algorithm to check if the challenge is valid.
   1. If valid, the challenge is saved in the system.
   2. If invalid, user is asked to try again until his challenge is accepted.

**Use Case #4**Name:

* Modify Settings

Participating Actor:

* Player

Stakeholder and Interests:

* Players wishes to view/modify audio and/or video settings.

Entry Condition:

* Player is on the *Main* *menu*.
* Player clicks *Settings* button.

Exit Condition:

* Player successfully changed the settings.
* Player clicked the *Back* button.

Main Flow of Events:

1. Player clicks the *Settings* button.
2. Player selects the type of settings he wishes to modify.
3. System responds to by displaying the appropriate page.
4. Player applies the changes.

**Use Case #5**Name:

* View Options

Participating Actor:

* Player

Stakeholder and Interests:

* Players wishes to view information regarding the game or its creators.

Entry Condition:

* Player is on the *Main* *menu*.
* Player clicks *Options* button.

Exit Condition:

* Player clicks *Back* button.

Main Flow of Events:

1. Player selects the *Options* button.
2. System responds by displaying the *Options* menu.

Alternate Flow of Events:

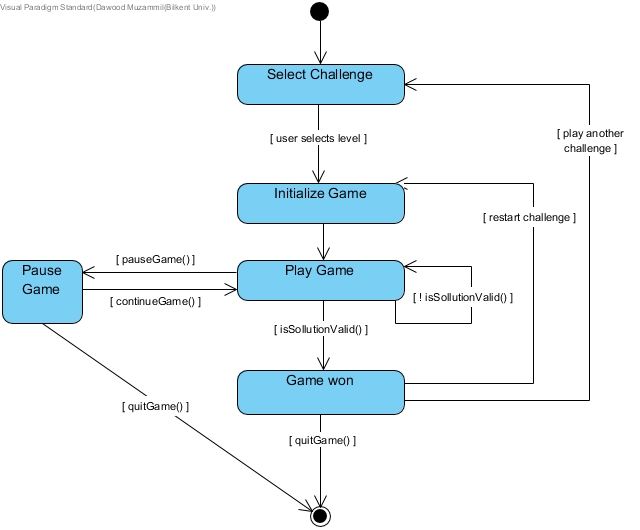
1. Player wants to see his achievements and/or change themes or characters.
   1. Player clicks *My Store* button.
2. Player wants to reset his progress.
   1. Players clicks *Reset Progress* button.
3. Player wishes to see how to play the game.
   1. Players clicks *Game Instructions* button.
4. Player wants to see information about the game creators.
   1. Players clicks *About Us* button.
5. Player wants to see the game credits.
   1. Players clicks *View Credits* button.

# State Diagram

* **Classic Mode**

The diagram below (figure 2) represents the state diagram of the Classic Mode of our game. When playing the classic mode, the user will have to select an unlocked challenge. Once selected, the system will initialize the game and take the system to the *Play Game* state. Here, the player will interact with the four predefined walls and try to come up with the correct solution. The player will stay in this state until he solves the challenge. Being in the *Play Game* state, the user can pause the game (and then continue the game) at any time. He can also quit the game from the *Pause Game* menu.

As soon as the player comes up with the correct solution, he will be taken to the *Game Won* state where he’ll see his time, score and rewards. From this state, the user can quit the game, play the same level again or play a different level.



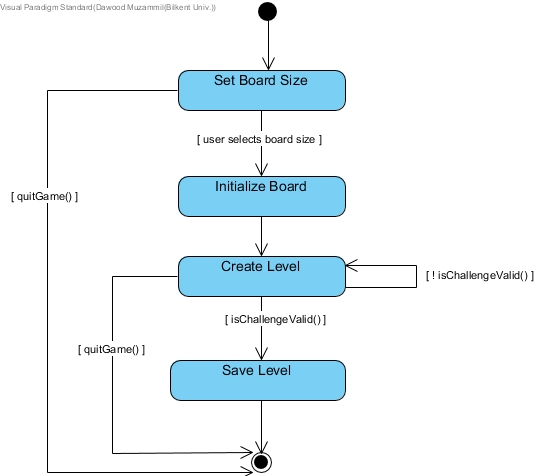
*Figure 2 State diagram for Classic mode*

*Imgur* link for better resolution:

<https://imgur.com/NR2IXyQ>

* **Sandbox Mode**

The diagram below (figure 3) represents the state diagram of the Sandbox Mode of our game. When the player starts the Sandbox mode, he will be in the state where he’ll be asked to enter the game board dimensions. When he submits his preferred dimensions, the game board will be initialized (next state) and the player will automatically be taken to the *Create Level* state. Here, the player will try to come up with a valid solution. The player will stay in the same state unless he has created a valid challenge. When the user submits a valid challenge, his new challenge will be saved (*Save Challenge* state). The user will be able to quit the game/mode at any point.



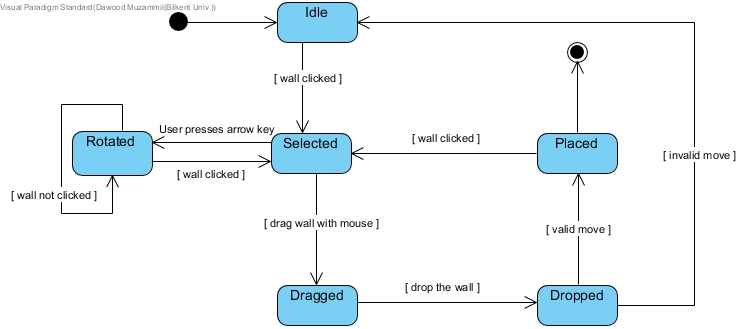
*Figure 3 State diagram of the Sandbox Mode*

*Imgur* link for better resolution:

<https://imgur.com/mJMC6oQ>

* **Wall Object**

The diagram below (figure 4) represents the states of a single wall object as the user interacts with it. While playing or creating a challenge, the state of each wall change as the user interacts with it. It starts from the *Idle* state. When a user clicks on a wall, it is *Selected* and can be manipulated (can be rotated by arrow button on the keyboard or moved by dragging with mouse). To go to the next states, the user must drag the wall (*Dragged*) and drop it in appropriate place (*Dropped*). If the wall is dropped at a valid position, it goes to the *Placed* state, otherwise, the state of the wall is taken back to the *Idle* state.



*Figure 4 State diagram of the wall object*

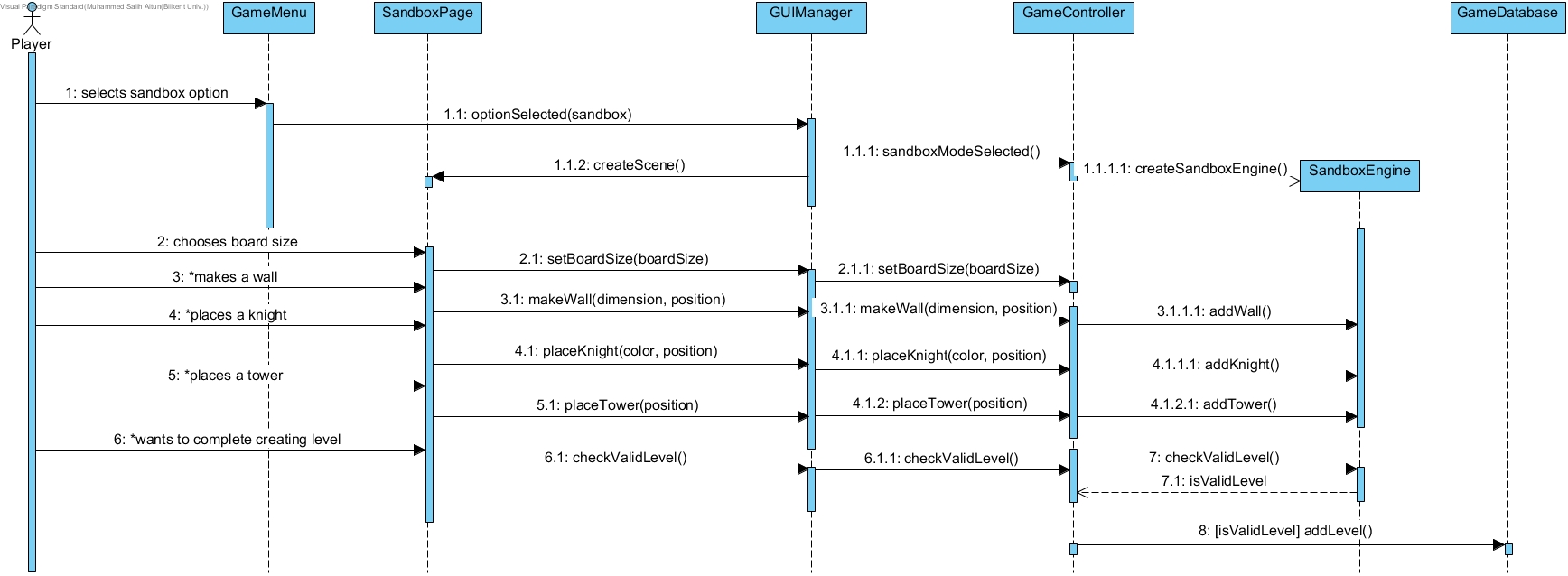
*Imgur* link for better resolution:

<https://imgur.com/jfYCOJf>

[ this space is left blank intentionally ]

# Sequence Diagrams

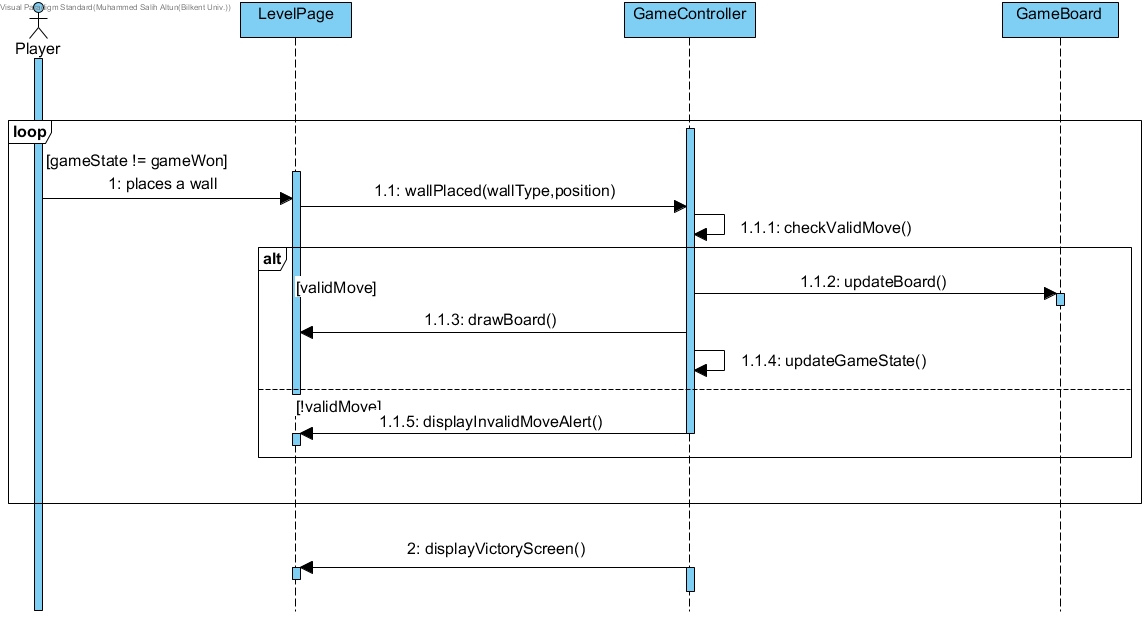
* **Create Level Diagram**

In Create Level Diagram (figure 5), the player decides that they want to create a new level of their own using the sandbox mode. They choose sandbox option from the menu. The menu sends the information to the GUIManager which tells the sandbox page to create a scene. When the player is creating a level, they need to choose their board size first. After that, they are free to place however many of the game objects on the board. The player can also introduce their own new wall shapes to tackle these new puzzles. At any point, they can try to make their level a reality by asking the game to create the level. In this case the controller checks if the self-made level has a valid solution or not. If the level has a solution, it is added to the game database.

*Figure 5 Sequence diagram for creating a level in sandbox mode*

*Imgur* link for better resolution: <https://imgur.com/a/o5fiZMi>

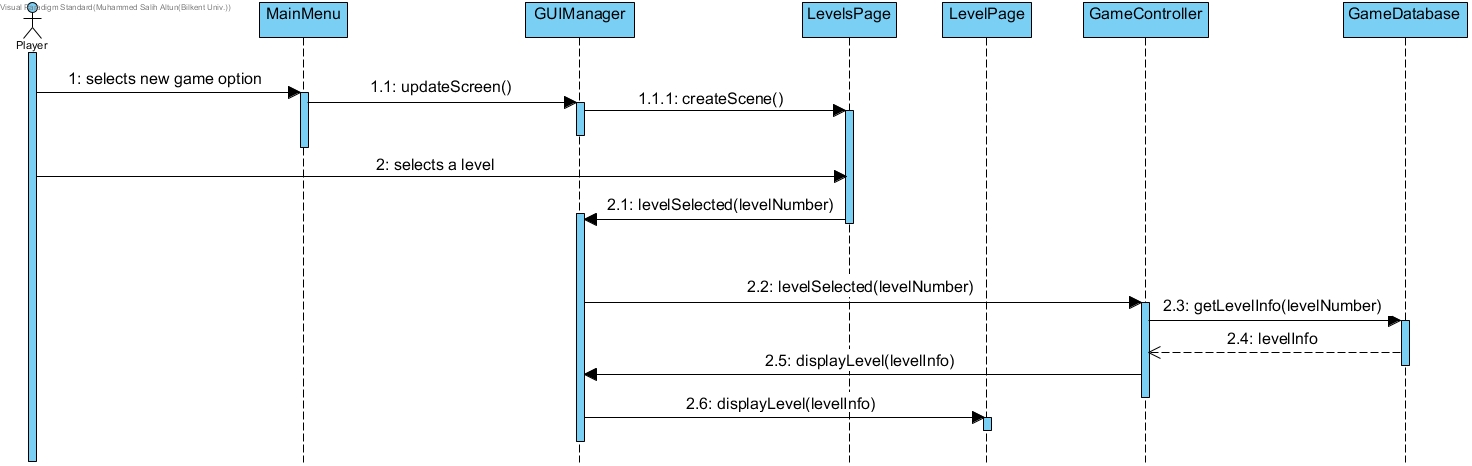
* **Play Game Diagram**

In Play Game Diagram (figure 6), the player has chosen a level to play and they are trying to solve the puzzle. In a loop, they try to place walls in the right places. Every time a wall is placed by the player, the level page informs the game controller which checks if the move was valid or not. There are two alternate scenarios in this case. If the move was valid, the board has to be updated and drawn on the page and the game state needs to be updated. Otherwise, the game needs to display an alert telling the player that their move was invalid. If the move was valid and the game state is equal to game won, the game exits the loop and displays the victory screen.

*Figure 6 Sequence diagram for playing the game in classic mode*

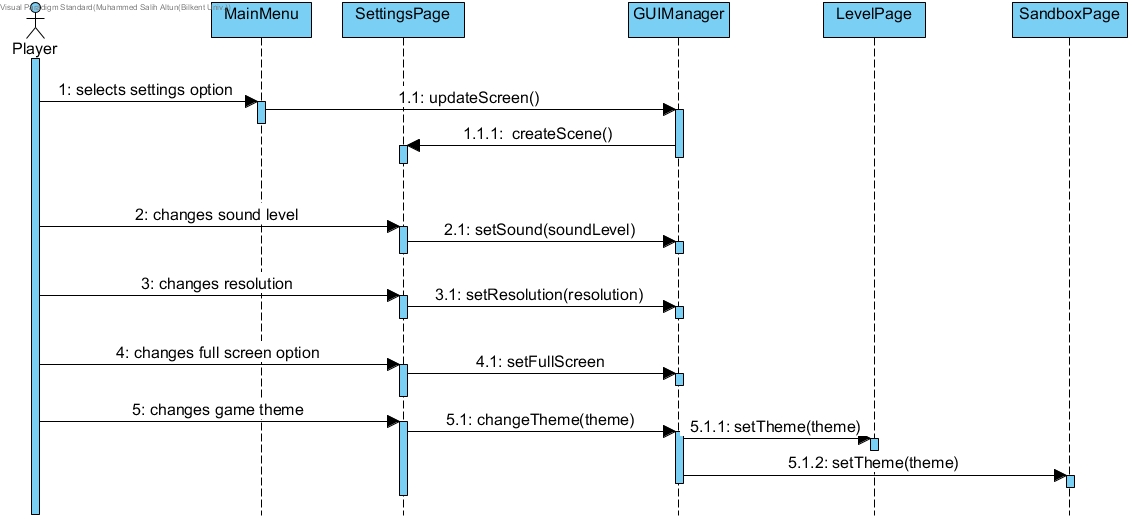
*Figure 5 CAPTION FOR SALIH*

* **Start New Game Diagram**

 In Start New Game Diagram (figure 7), the player is in the main menu. They select the new game option. The main menu informs the GUIManager of this choice, which tells the LevelsPage to create a scene. The LevelsPage displays the levels to the player as locked or unlocked. The player selects a level they can play. LevelsPage tells the player's choice to GUIManager, which informs the GameController of this selection. The controller requests the level information from the GameDatabase and tells the GUIManager to display a level with this information. GUIManager tells the LevelPage to display the level.

*Figure 7 Sequence diagram for starting a new game*

* **Change Settings Diagram**

In Change Settings Diagram (figure 8), the player is in the main menu. They select the settings option. MainMenu informs GUIManager who tells the SettingsPage to create a scene. The player changes the sound level. SettingsPage tells GUIManager to set the sound level to the chosen level. The player then changes the resolution. SettingsPage tells GUIManager to set the resolution. Similarly, the player then changes the fullscreen option. SettingsPage tells GUIManager whether the game should be set to fullscreen or not. The player then changes the game theme. Settings page tells GUIManager to change the theme which then informs both LevelPage and SandboxPage to change the game theme.

*Figure 8 Sequence diagram for displaying and changing settings of the game*

[ this space is left blank intentionally ]

# Class Diagram

*Figure 9 Class diagram*

*Imgur* link for better resolution:

<https://imgur.com/a/YvaCzs0>

* **Controller Class**
* **GameController**: This class is the overseer of the whole class system. It lets other classes to talk to each other and gives necessary commands to them. Starting, loading and ending the levels are issued by this class.
* **Entity Classes**
* **GamePiece**: This is the superclass of all game pieces. Which are the things that can be put on the game board in the physical game. It has all the common functionalities of the pieces such as position and the ability to move.
* **Knight**: Child class of the GamePiece, both the Red and the Blue knights are represented by this class. It has a colour attribute to distinguish the different knights. As well as the functionality to get and set their colours. There are different knights in the game but we have chosen to represent them with the same class because there are only two types and checking their colour attribute is easier to code and more efficient than creating entirely different objects. The only difference between red and blue knights is their one attribute, color.
* **Wall**: Child class of the GamePiece, all walls are made up of one side lines put together. The shape property specifies the formation of these lines to create a wall. The rotate functionality rotates the wall while keeping its shape intact.
* **GameBoard**: Represents the actual board of the game but in our game it is more like the map of each level. It has dimensions, which are 5 to 4 in regular levels but can be changed in sandbox mode. It also has the main game algorithm in it to keep track of the state of the game and block illegal moves.
* **GameTimer**: Keeps the track of time for each individual level. Has standard functionalities of a regular stopwatch.
* **Database Class**
* **GameDatabase**: Keeps all the data regarding the created levels and the progress of the player. It saves and loads data as controller instructs it.
* **View Classes**
* **GUIManager**: This class is a control class of all UI components and pages. It is connected to the rest of the game logic by the game controller. It has methods to update the screens and scenes in the game.
* **MenuPage**: This class is the parent class of all menu pages. Its attributes are the common properties of all the menu pages.
* **MainMenu**: This class has the views that belong to the main menu. Includes all game options like playing a new game, display options, change settings and quit game.
* **NewGamePage**: This class has the views that belong to new game page.
* **LevelsPage**: This class has the views that belong to levels page. Levels page shows the current levels in the game, they are locked and unlocked based on player's current progression.
* **OptionsPage**: This class has the views that belong to options page. Options page shows the options of the game like credits and about us.
* **SettingsPage**: This class has the views that belong to settings page. Settings page shows the current settings of sound level, resolution and fullscreen.
* **GamePage**: This class has the views that belong to game page. Game page displays walls and the board of a level.

# 6.1.5 Activity Diagram

*Figure 10 Activity diagram of Walls & Warriors*

*Imgur* link for better resolution:

<https://imgur.com/H1wUEIa>

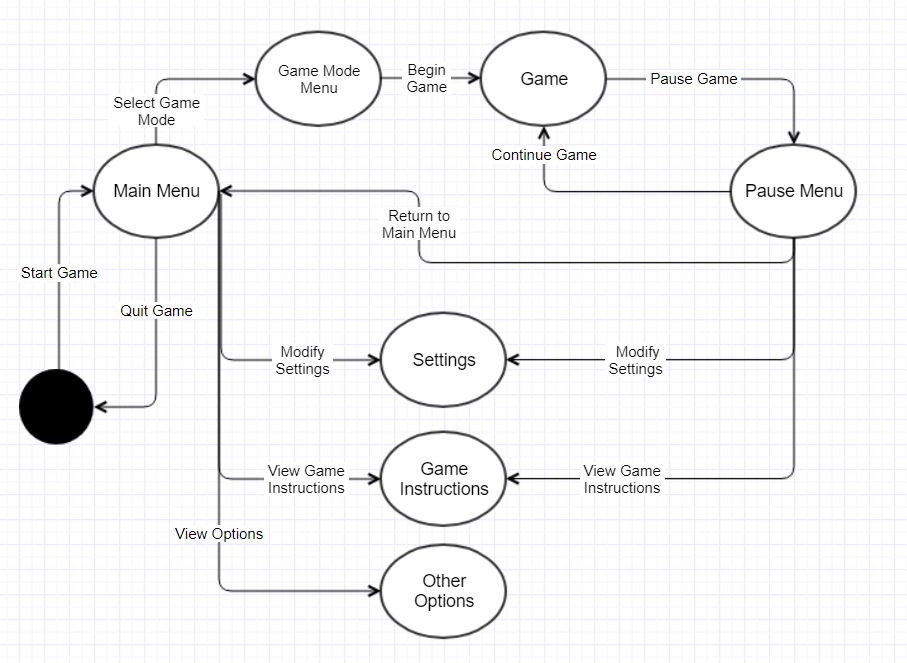
The activity diagram above (figure 10) illustrate how the system runs the game. In the beginning, the system waits for the user to select what he wishes to do. If the user decides to play the classic mode, the system initializes the game board and starts the timer. It then waits for user to drag and drop all four wall pieces on the board. Once the fourth wall piece is placed, an algorithm is run and to check if the solution is valid. If the solution is valid, the game returns stops the timer and retrieves the time elapsed, calculate score and distribute the rewards based on the score. The system is then returned to the *Classis Mode Menu* so that the user can continue playing the game. However, if the solution is invalid, the game continues.

In the Sandbox mode, the game allows players to create their own challenges. First, the system asks the user to specify board dimensions and then it initializes the game board. The system then lets the user sets all the objects on the board and tries to submit the challenge. Once the user submits the challenge, the same algorithm is run to check if the challenge is valid. If it is valid, the system saves the level. However, if the challenge is not valid, the system stays in the Sandbox Mode until the user either submits a valid challenge or cancels.

[ this space is left blank intentionally ]

# User Interface

# Navigation Paths



*Figure 11 Navigation Paths of Walls & Warriors* [3]

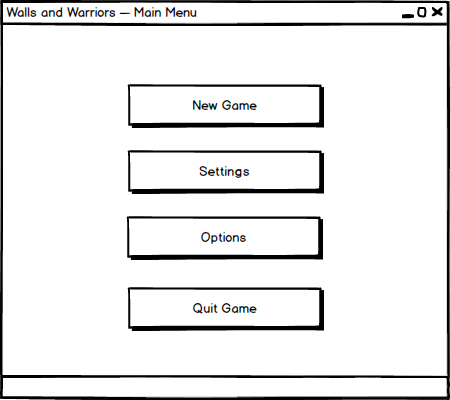
*Imgur* link for better resolution:

<https://imgur.com/a/AGVsLmJ>

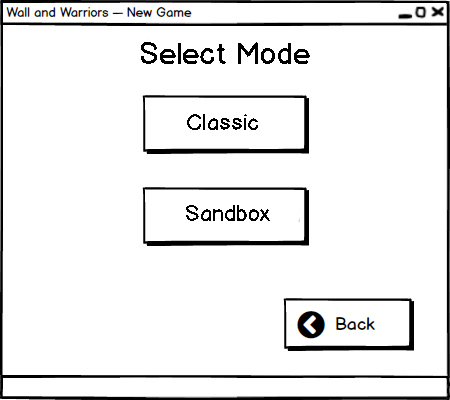
The figure above (figure 11) shows the navigation paths of *Walls and Warriors*. *Game Menu* is the first thing displayed to the user. Here, the user can go on to select the game mode, modify settings, view game instructions as well as other options such as *My Store, About Us* and etc. The reason why *Game Instructions* and *Other* Options are mentioned separately is because the user can not view any options from the *Pause Menu* except the game instructions.

If the user wants to play the game, he will be taken to the *Game Mode Menu* where he’ll select his desired mode and from there he’ll be taken to the actual *Game*. The user can pause the game at any time. From the *Pause Menu*, the user can continue the game, modify *Settings* or view *Game Instructions*. He can also return to the main menu, from where the user can quit the game.

# Screen Mockups

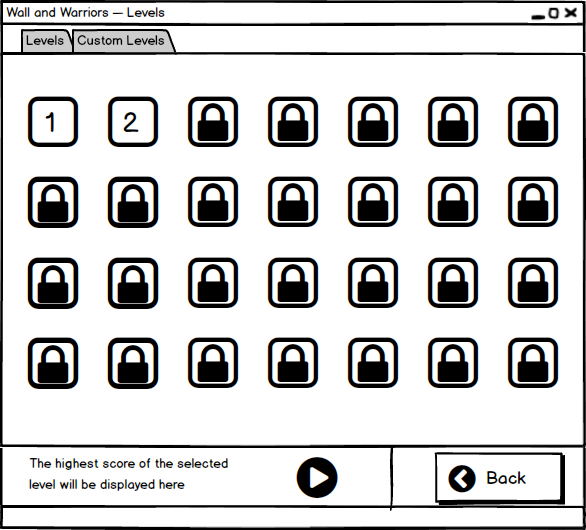
* **Main Menu**

*Figure 12 Screen mockup of the Main Menu*

* **New Game**

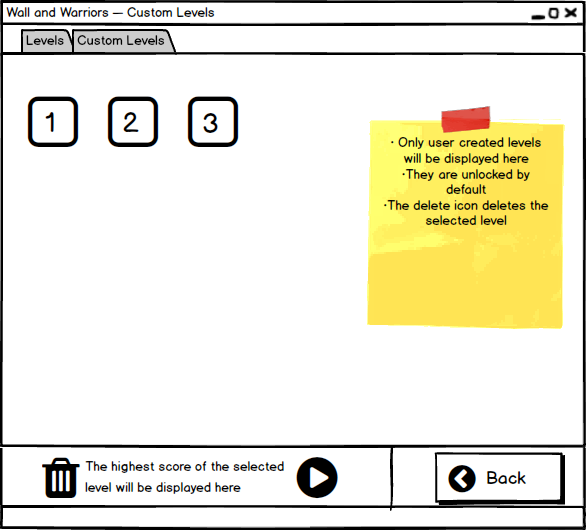
*Figure 13 Screen mockup of the New Game menu*

* **Select Level**

****

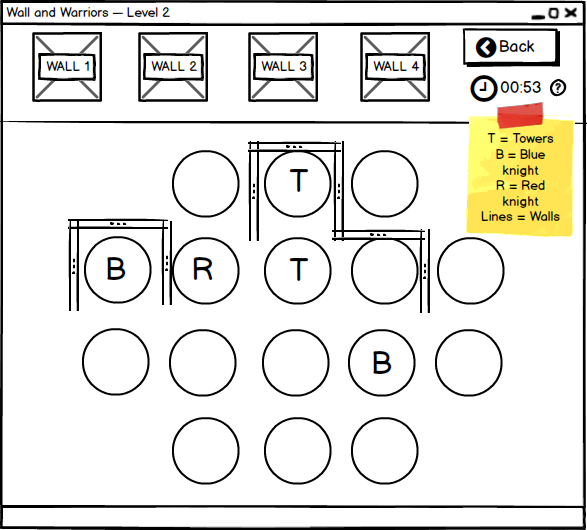
*Figure 14 Screen mockup of level selection in Classic Mode*

* **Select Custom Level**



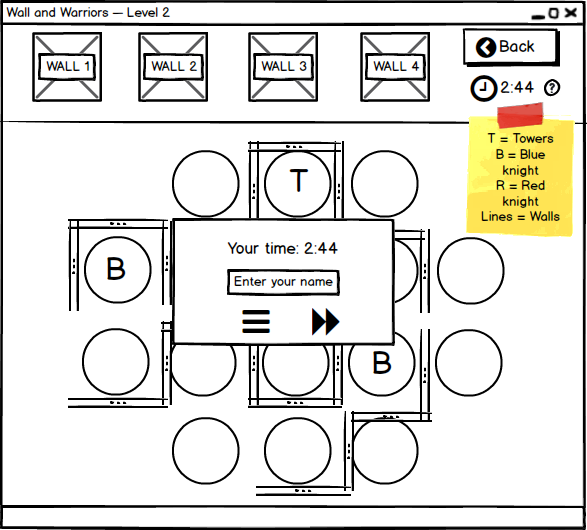
*Figure 15 Screen mockup viewing levels in Sandbox mode*

* **Play an Original Level**

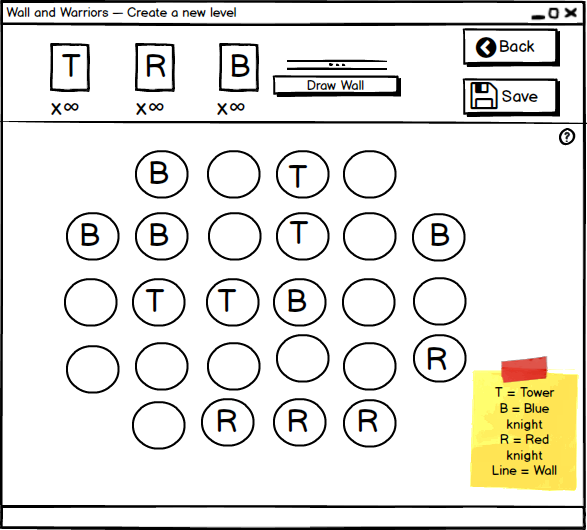
****

*Figure 16 Screen mockup of a Classic game*

* **End of an Original Level**

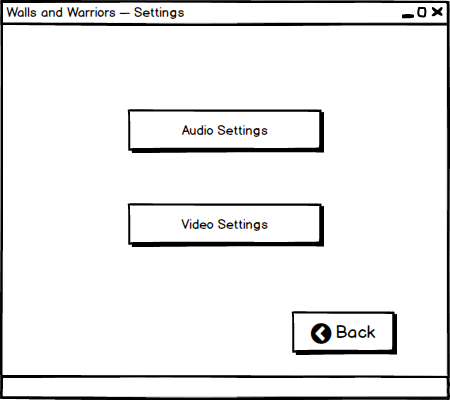
****

*Figure 17 Screen mockup of the classic game*

* **Create a New Level**

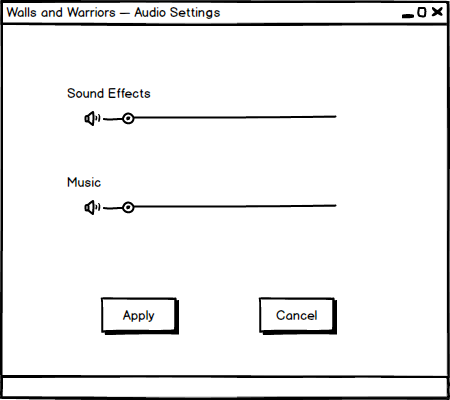
*Figure 18 Screen mockup of the Sandbox mode*

* **Settings**

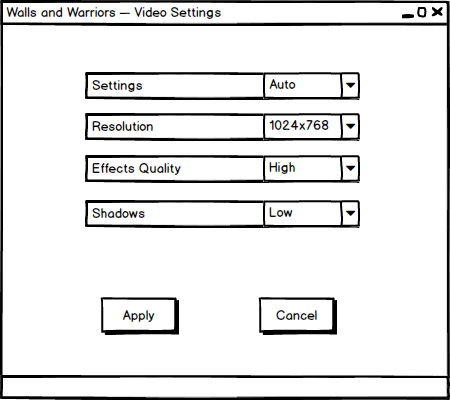


*Figure 19 Screen mockup of Settings menu*

* **Audio Settings**

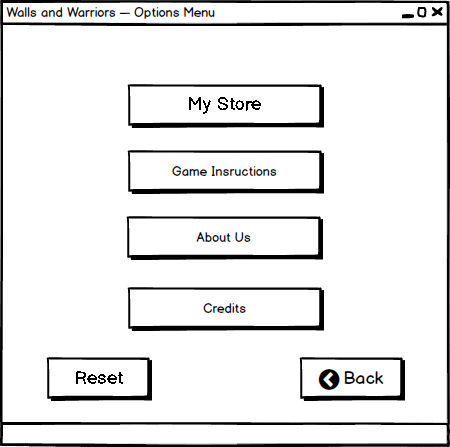
****

*Figure 20 Screen mockup of the Audio Settings*

* **Video Settings**

*Figure 21 Screen mockup of the Video Settings*

* **Options**

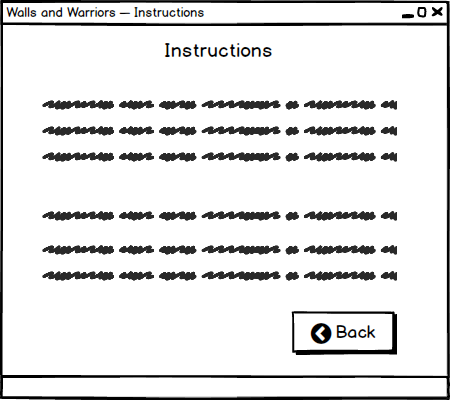
****

*Figure 22 Screen mockup of the Options menu*

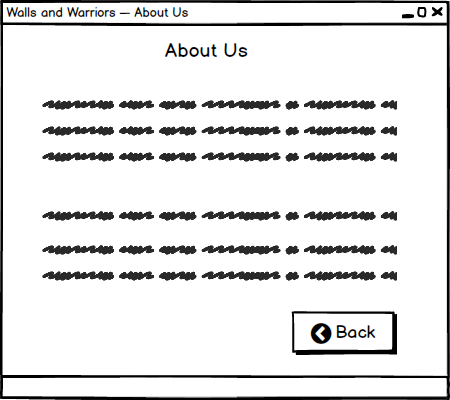
* **High Scores**

*Figure 23 Screen mockup of the My Achievements screen*

* **Game Instructions**

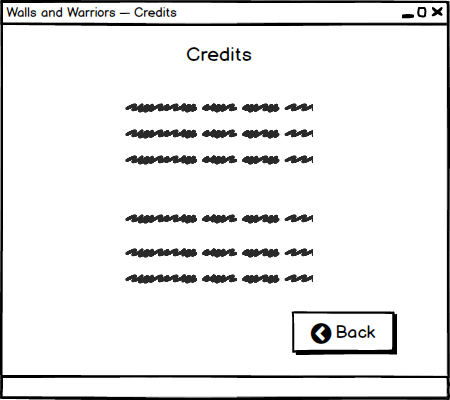
****

*Figure 24 Screen mockup of Game Instructions*

* **About Us**

*Figure 25 Screen mockup of About Us*

* **Credits**

****

*Figure 26 Screen mockup of the Credits Screen*

[ this space is left blank intentionally ]

# Improvement Summary

1. Two new functional requirements have been added. One is the registration system; the other is the ability to change themes and characters provided to the users.

1. Example of themes: Volcano, Forest, Ocean, Desert, etc.
2. We intent to replace red and blue knights with cartoon characters and animals which children could relate to easily. For example, cats and mice representing *Tom & Jerry*.

2. The non-functional requirements have been revised and made into new, measurable requirements.

3. The use case diagram has been revised to be a more accurate and specific to the game. It has been changed to include the additional functionality of the system.

4. The activity diagram has been revised

5. The state diagram has been completely revised into three different state diagrams, two for the system and one for the object state diagram of “wall”.

6. The sequence diagrams have been revised to more accurately represent how objects of the system communicate between themselves and the user. Sequence diagram for displaying and changing settings has been added. Textual descriptions of all sequence diagrams have been revised to reflect these changes.

7. The class diagram has been revised to accurately represent the relationships between objects in the system and be synchronized with the low-level design of the system. Textual descriptions of classes have been updated to reflect these changes.

8. Navigation path for the system added along with its textual description.

9. The introduction and overview parts of the report have been updated to improve clarity and remove ambiguities in text.

# Glossary & References

[1] [Online] <https://www.smartgames.eu/uk/one-player-games/walls-warriors>

[2] [Online] [www.mongodb.org](http://www.mongodb.org/)

[3] [Online] <https://www.gliffy.com/>