Bilkent University

Department of Computer Engineering

CS 319 - Object-Oriented Software Engineering

Term Project - Final Report

Iteration 2

Project Name: Walls & Warriors

Group No: 2B

Group Name: OOPs

Group Members: Ali Babayev
Tunar Mahmudov
Merve Sağyatanlar
Çağla Sözen
Emin Bahadır Tülüce

Table of Contents

1. Introduction

Following the end of the first iteration, implementation for additional features and fixing bugs have immediately started. Implementation was performed on different IDE's and we used a GitHub repository to collaborate synchronously on the implementation with the contribution of all group members as it was done in the first iteration. By the end of the second iteration, the implementation was completed for the fundamental requirements of regular Walls & Warriors and with most of the additional functionalities that were presented in the Analysis Report. The implemented functionalities are discussed further in the following section *(Section 2)*.

2. Implemented Functionalities

For the second iteration, our aim was to complete the implementation such that it would provide the fundamental and the promised functionalities with an improved but neat UI. Currently, our system supports the following functionalities:

- An algorithm that checks the solutions in the GameScreen and the ChallengeEditor Screen
- GameScreen and ChallengeEditorScreen basic interactions (drag and drop elements on the grid)
- The two functions to convert between Base64 String and ChallengeData
- Basic file operations (read and write challenges to file)
- Screen transitions
- Distinguishing between Wild and Standard Challenges in CustomChallengesScreen
- Saving player progress in case of a crash or exit.
- Sound effects for GameScreen with adjustable volume
- Background music with adjustable volume
- Adding a new challenge directly to the Custom Challenges with Add to Custom Challenges button
- Option to proceed to the next challenge after a challenge is solved in Campaign Challenges
- Distinguishing between locked and unlocked challenges in Campaign Challenges
- Function to provide a hint in the GameScreen
- Game Objects are replaceable and removable in GameScreen and the ChallengeEditorScreen
- Saving preferred settings for a player
- Function to remove a challenge from CustomChallenges.
- Button to mute the music on the GameScreen.
- Continuous rendered shapes for wall pieces.

3. Design Changes

During our implementation, we have noticed that we were not able to implement our system in the exact way we have designed. In this section, we will discuss the differences between our design and implementation. These differences are not major and they do not change our general design structure.

3.1. Extended Dependencies

During our design, we were expecting the packages and subsystems to depend on each other in a restricted manner. However, we have seen that it is not easy to write independent classes while keeping the design simple. Thus, in order to keep our design simple and easy, we have introduced new dependencies between our classes. For example, screens were not designed to be dependent on ChallengeManager, however, they should display a title and update the views according to the current challenge, so they depended on ChallengeManager to get the current ChallengeData.

3.2. Private Utility Functions

In our design phase, we didn't know the exact contents of our functions. During implementation, we have filled their contents and figured out that they needed some other sub-functions to call, to make our code cleaner. These sub-functions were not included in our design since the exact algorithms were not developed yet.

3.3. Sub-Packages

During the implementation of our screens, we have thought that it would be a good idea to create sub-packages for related classes for system organization. For example, we have moved the GameScreen hierarchy (three screen classes, with one being abstract) in a sub package called "`screens.game`". See Figure 1 for the whole implemented hierarchy.

3.4. Game Constants

During the implementation, we have noticed that there were many constants being declared to avoid using "magic" numbers. We have thought that it would be a good idea to put them in a separate class called GameConstants as static final properties.

3.5. Definition Classes

We have decided to store the definitions for walls and grid in seperate classes. In `WallDefinitions` class, we have the definitions for the standard wall shapes along with their bastions. In `GridDefinitions` class, we have our standard-sized grid's definition and some other grids with different sizes which are ready to be used in possible further extensions of the game.

3.6. Merging of Game and Managers Package

We have decided to merge game and managers package since the `game` class was strongly connected to the managers package classes. This way, we had a package with higher cohesion and we avoided te high coupled situation.

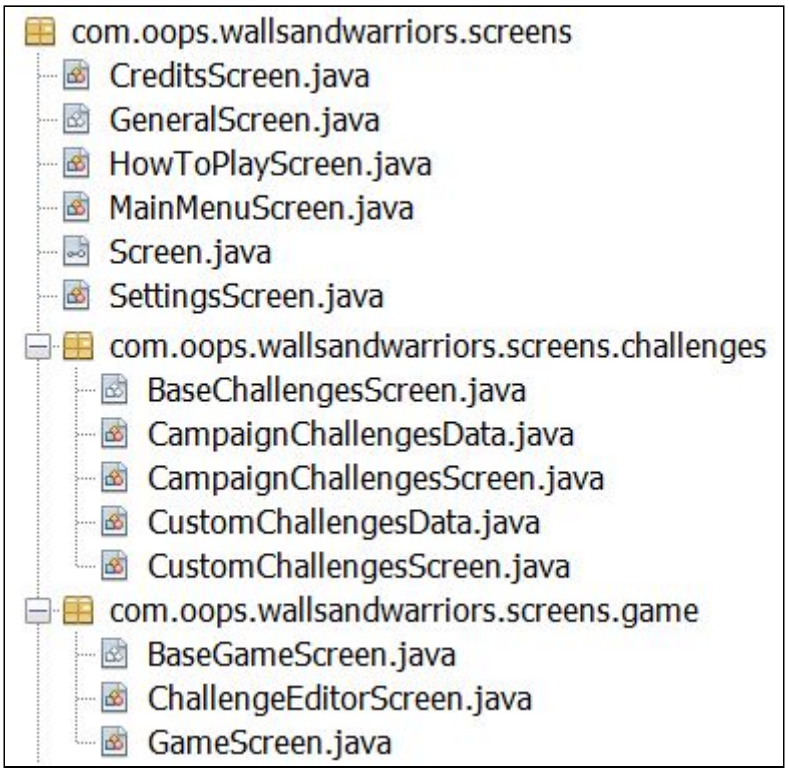

Figure 1: Revised `screens` package

4. Lessons Learnt

During the implementation phase, we have learned how to divide an activity into tasks and tasks into subtasks. Also, we have learned how to collaborate through a source control tool which is Git, in our case.

We also had the chance to experience an implementation process that is done on top of a ready design. We have seen that this kind of an implementation is fairly faster but it's also true that the minor changes might be needed in some circumstances (See Section 3).

Our implementation phase progressed as follows:

Before beginning the implementation, most significant part of the second iteration was to determine the missing functionalities and the improvements that could be done on the existing implementation and deciding upon their order of importance. After determining the tasks, these tasks were assigned to different group members. In this iteration, dependency of tasks was not as high as the first iteration. So tasks were distributed more evenly and freely in this iteration. However, improving the UI

and handling errors was not assigned to a specific member, but rather they were determined to be done by any of the members who would have the opportunity.

The implementation was divided into 18 compulsory tasks. 16 of the compulsory tasks were assigned to a single member and for the other 2, each member was to work individually and simultaneously. The implementation was divided into the following tasks:

- Implement "Add to Custom Challenges" button for export dialog. Task #1
- Implement the aftermath of a solved challenge. Task #2
- Add challenge type to the challenge preview in custom challenges Task #3
- Implement locked/unlocked for campaign challenges. Task #4
- Implement next challenge button. Task #5
- Add a hint button and implement its functionality. Task #6
- Check for solution in ChallengeData while exporting. Task #7
- Handle empty or too long name/desc/creator while exporting. Task #8
- Make placed grid pieces draggable and removable in editor. Task #9
- Save and retrieve the settings. Task #10
- Add a remove button to custom challenges. Task #11
- Handle long descriptions for custom challenges. Task #12
- Add sound effects. Task #13
- Add music and a mute button. Task #14
- Handle system crashes to save the last player progress. Task #15
- Fix the appearance of the walls (make continuous portions). Task #16
- Polish UI appearance and layout. Task #17
- Handle the erroneous cases. Task #18

Throughout this process, each member chose the task they wanted to undertake. Implementations of all tasks except Task #17 and Task #18 were assigned to a single member and all tasks except Task #5 were implemented simultaneously. However, implementation of Task #5 required the implementation of the first 4 tasks. Other than that, there were no significant dependencies among tasks. Implementation of Task #17 and Task #18 were done by all available members.

For the sake of completeness, the tasks that we have already finished in Iteration 1 are also listed below:

- Implement an algorithm to determine if a solution is valid. Task #1.1
- Implement an algorithm to determine invalid parts of the wrong solution. Task #1.2
- Implement an algorithm to determine if a ChallengeData is valid. Task #1.3
- Implement GameScreen interactions. Task #2.1
- Implement ChallengeEditorScreen interactions. Task #2.2
- Implement a function to convert ChallengeData into a Base64 String. Task #3.1
- Implement a function to convert Base64 String into ChallengeData object. Task #3.2
- Read and list all challenges from a file. Task #4.1
- Write an imported custom challenge to file. Task #4.2
- Keep track of solved/unsolved challenges. Task #4.3

- Implement SettingsManager. Task #5.1
- Implement settings screen. Task #5.2
- Implement how to play screen. Task #5.3
- Implement credits screen. Task #5.4

5. User's Guide

5.1. Build Instructions

There are two ways for building Walls & Warriors from its source code. In both cases, the programmer should get a JDK with Java 8 (or later versions).
JDK download link: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

5.1.1. Compile Using an IDE

If you prefer to compile the game with an IDE, follow the below instructions.

- Download the git repository.
- Create an empty project with the name "WallsAndWarriors", in your IDE.
- Copy the `com` folder of the repository into the `src` folder of your empty project.
- If necessary, set the main class of the project in the configurations as "`com.oops.wallsandwarriors.GameLauncher`".
- Click run in your IDE to compile and run the game.

5.1.2. Compile Using the Command Line

If you prefer to compile the game on the command line, follow the below instructions.

- Download the git repository.
- Browse to the `src` folder which contains the `com` folder inside it.
- Execute the following commands to compile and run the game:

```
javac com/oops/wallsandwarriors/GameLauncher.java
java  com.oops.wallsandwarriors.GameLauncher
```

5.2. System Requirements & Execution

The game can run in any operating systems that support Java runtime environment. Since the game uses only a small amount of memory (~300 MB of RAM) and storage (~200 KB of HDD) portions, there is not any serious requirement on memory and storage to run the game.

To run the game, the player should install Java on their system. They can download it from the official site (www.java.com/download). After that, all the player should do is executing our "`WallsAndWarrior.jar`" file by double-clicking on it. This jar will be provided with the distribution of the game.

5.3. How To Use

5.3.1. Start a campaign challenge

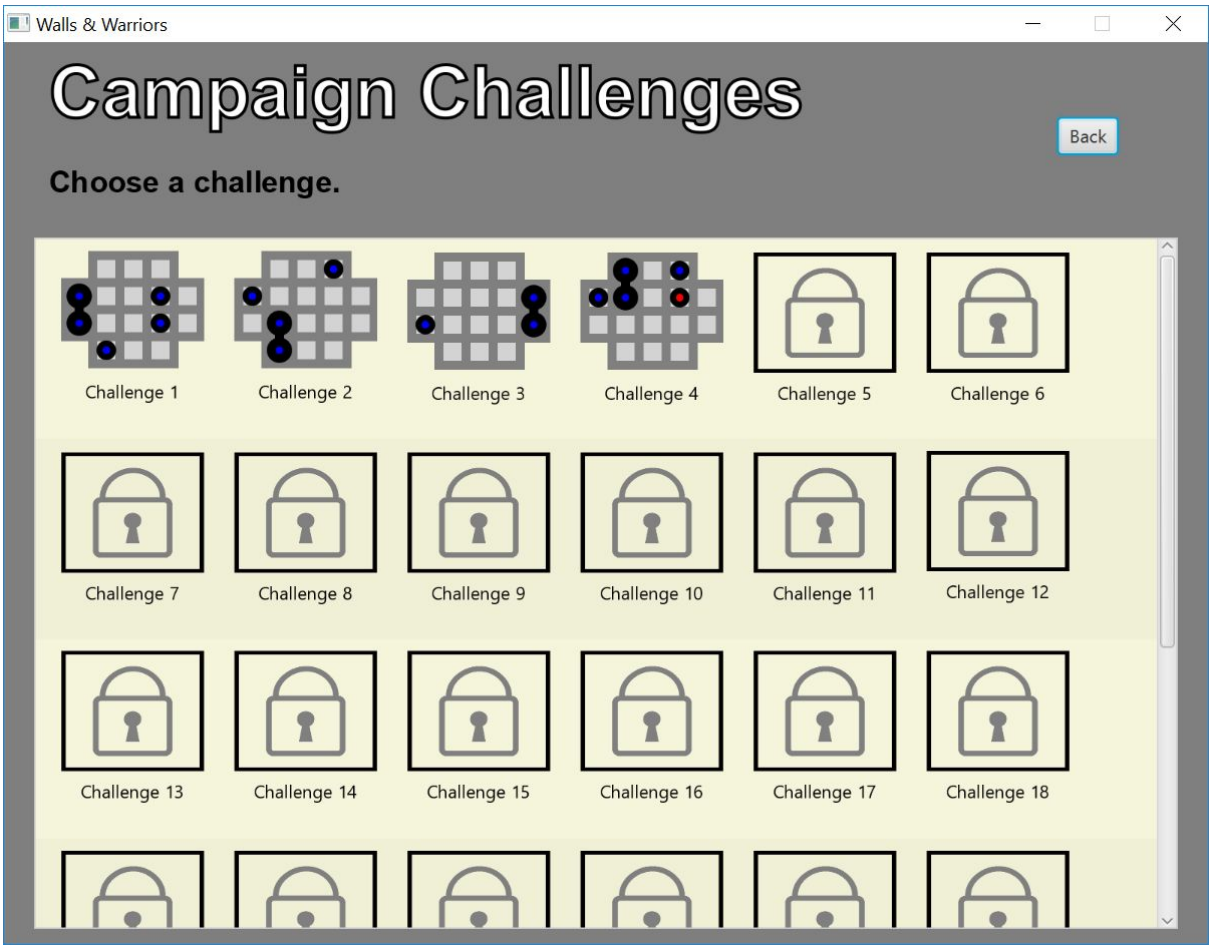

Figure 2: Campaign challenges screen

In order to start a campaign challenge, the user needs to click “Campaign Challenges” button in the main menu. In the next window, the user will choose one of the pre-loaded challenges by the left-click of the mouse over the desired challenge’s preview. In the following window, the user will see the challenge’s grid and in the left of the grid, available walls are given to use in order to solve the challenge. The playing process is described in Section 5.2.3.

5.3.2. Start a custom challenge

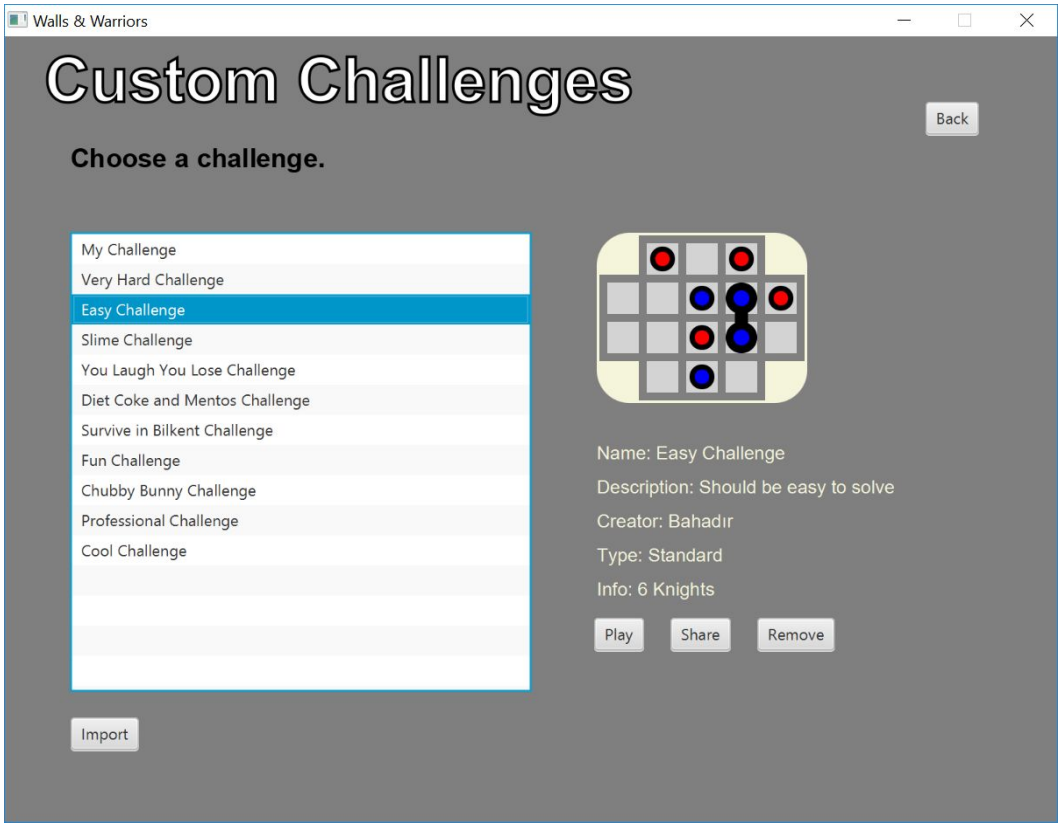

Figure 3: Custom challenges screen

In order to start a custom challenge, the user needs to click “Custom Challenges” button in the main menu. In the next window, the user will see the list of available custom challenges, which are made by other players or the player themselves. Initially, there will be no any challenge in this list, so they should be imported. To learn how to import the challenge, see Section 5.2.4. In order to play the challenge, the user needs to choose the challenge he/she wants to play by clicking the name of the challenge from the given list and press “Play” button in the given window. In the following window, the user will see the challenge’s grid and in the left of the grid, available walls are given to use in order to solve the challenge. The playing process is described in Section 5.2.3.

5.3.3. Play a challenge

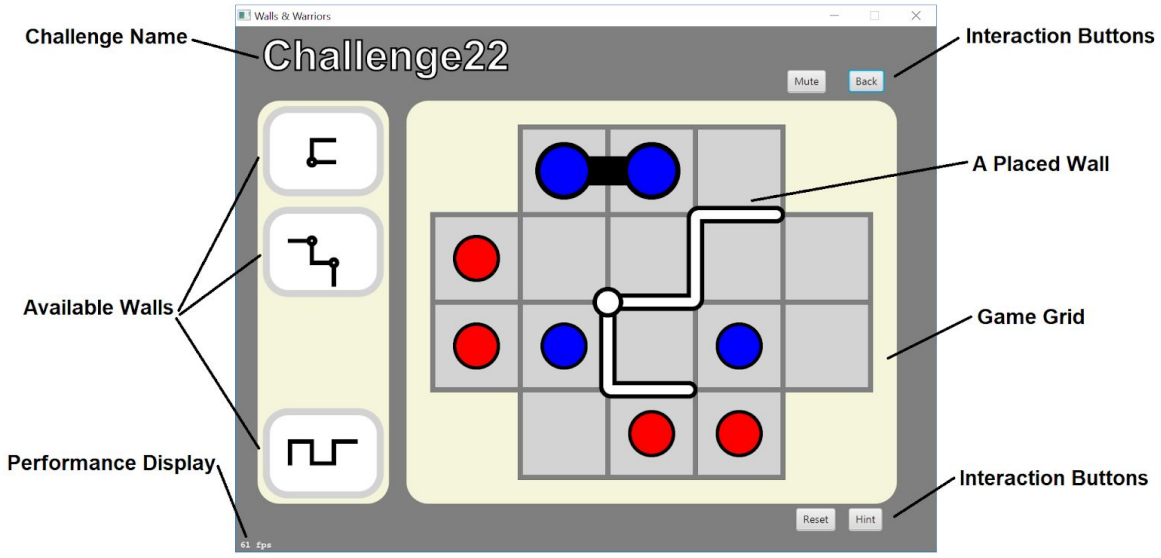

Figure 4: Game screen components

In order to solve a challenge, the user needs to drag the given walls on the left of the screen and place them to the map by clicking the left button of the mouse on the walls. The user can rotate the walls by clicking the right button of the mouse while a wall is selected. If the user placed the walls incorrectly, then he/she can change its place on the map or return to its default place by just dragging with the left button of the mouse. If all walls correctly are placed, then a “Congratulations” message will appear on the screen. The user can also reset all placements of the walls, by just clicking the reset button on the current window. On the other hand, if the user can not solve the challenge, he/she can get hint by clicking “Hint” button. One of the walls will be placed by game for hint.

5.3.4. Import a challenge

If the desired challenge is not among the given challenges in the “Custom Challenges” list, then users can import their own, which is done by entering the challenge code. In order to import the new challenge, the user needs to press the “Import” button and enter the code of the challenge in the dialog window. After pressing “Ok”, the new challenge will be added and it will be seen on the list of the custom challenges.

5.3.5. Share a custom challenge

To share a custom challenge with other players, the user should click the “Share” button in the right of the window under the challenge’s preview. Then the user can copy the unique code of the challenge and share it with others. To learn how to import a challenge, see Section 5.2.4.

5.3.6. Create your own challenge and share

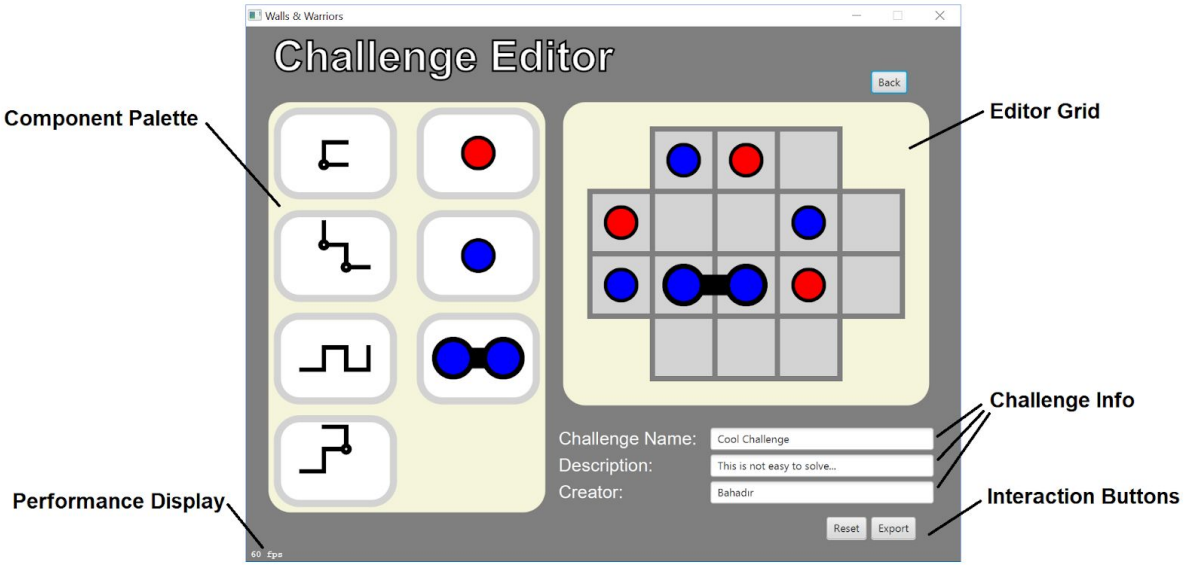

Figure 5: Challenge editor screen components

To create the new challenge, initially, the user needs to click “Challenge Editor” button in the main menu. In the next window, the user will face with the window that is similar to the window of solving a challenge, but additionally, in the left part of the window, there are also blue and red knight and high tower to place on the map in the right. In order to create the new challenge, the user needs to put the desired number of blue and red knights, high tower as well as walls on the map by dragging with the left click of the mouse. However, once user placed the object on the map, the user can not move it or return to default. Instead, the user can reset all placements by simply clicking “Reset” button. After finishing the construction of the new challenge, the user can fill the information lines, such as challenge name, description, creator, and extract the challenge by pressing the “Export” button. As a result, the user will get the unique code of the challenge to copy in order to share or import it (see Section 5.2.4 for import.).

5.3.7. Continue last game

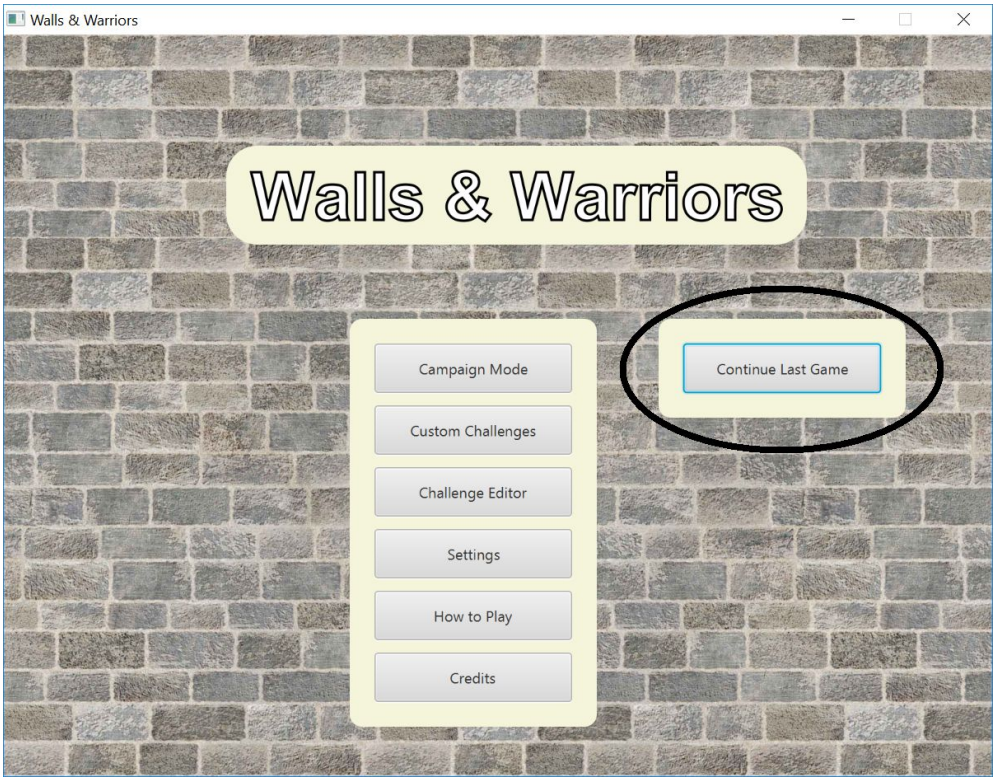

Figure 6: Main menu when the game is relaunched after a sudden exit

In case of a system crash or sudden close during the game screen, the user can restore their last progress on the game by clicking the “Continue Last Game” button from the Main Menu when the game is relaunched.

5.3.8. Change settings

To change game settings, first, the user needs to click the “Settings” button in the main menu. In the next window, the user can change the volume level of the music and sound by just dragging the “level point/circle” to left or right. The user can also change colors of the knights by just choosing one of the pairs from the given list. Settings will be saved automatically after leaving the page.

5.3.9. Visit “How to Play” page

To see how to solve page, the user needs to click on the “How to Play” button on the main menu. As a result, in the next window, detailed information about the rules, game modes, control editor and controls will appear in adjacent tabs.

6. Contributions

Ali Babayev

- Analysis Report - Use-Case Diagram and textual descriptions (Iteration 1)
- Final Report (Iteration 1)
- Analysis Report (Iteration 2)
- Final Report (Iteration 2)
- Demo Video (Iteration 2)
- Implemented algorithm using DFS to determine solved/unsolved state of challenges.
- Implemented several operations on coordinates to use for the solution algorithm
- Implemented an algorithm to determine wrong grid component/coordinate.
- Implemented "Add to Custom Challenges" button for export dialog.
- Handled empty or too long name/desc/creator while exporting.
- Added a remove button to custom challenges and implemented its functionality.
- Handled operations on file manager to implement session and remove files
- Handled system crashes to save the last player progress.
- Handled several alerts (warning, information type).
- Fixed several kinds of bugs.

Tunar Mahmudov

- Analysis Report - Object-Class Model (Iteration 1)
- Analysis Report - Functional Requirements (Iteration 1)
- Created a storage manager to store session, data and settings files.
- Implemented campaign and custom challenges screens.
- Implemented campaign and custom challenges data classes.
- Wrote, read and listed campaign and custom challenges from/to a file.
- Wrote an imported custom challenge to file.
- Implemented locked/unlocked for campaign challenges.

Merve Sağyatanlar

- Analysis Report - Overview (Iteration 1)
- Design Report (Iteration 1)
- Final Report (Iteration 1)
- Analysis Report (Iteration 2)
- Design Report (Iteration 2)
- Final Report (Iteration 2)
- Demo Slides (Iteration 2)
- Demo Video (Iteration 2)
- Implemented how to play screen content.
- Implemented credits screen content.
- Implemented settings screen content.
- Implemented theme settings.
- Write and read settings from file.
- Added sound effects.
- Added music and a mute button.

Çağla Sözen

- Analysis Report - Introduction (Iteration 1)
- Analysis Report - Dynamic Models (Iteration 1)
- Design Report (Iteration 1)
- Final Report (Iteration 1)
- Demo Slides (Iteration 1)
- Analysis Report (Iteration 2)
- Design Report (Iteration 2)
- Final Report (Iteration 2)
- Demo Video (Iteration 2)
- Demo Slides (Iteration 2)
- User interface mockups
- Converting ChallengeData into Base64 and vice-versa
- Added challenge type to the challenge preview in custom challenges.
- Added a hint button and implement its functionality.
- Checking for solution in Challenge Data while exporting

Emin Bahadır Tülüce

- Analysis Report - Overview (Iteration 1)
- Analysis Report - Non-Functional Requirements (Iteration 1)
- Analysis Report - Dynamic Models (Iteration 1)
- Design Report (Iteration 1)
- Final Report (Iteration 1)
- Demo Video (Iteration 1)
- Analysis Report (Iteration 2)
- Design Report (Iteration 2)
- Final Report (Iteration 2)
- Demo Video (Iteration 2)
- Demo Slides (Iteration 2)
- Class design and system architecture design
- User interface for game screen and editor screen
- Drag and drop interactions
- Screen transitions
- Graphics of the walls (Displaying continuous wall portions)
- Fixed various kinds of bugs.
- Managed GitHub repository.