Bilkent University

Department of Computer Engineering

CS 319 - Object-Oriented Software Engineering

Term Project - Final Report

Iteration 1

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

After designing the classes for Walls & Warriors during the Design Phase, implementation was 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. For the moment, the implementation was completed for the fundamental requirements of regular Walls & Warriors with some additional functionalities that are discussed more on the following section *(Section 2)*.

2. Implemented Functionalities

For the first iteration, our aim was to complete the implementation such that it would provide the basic functionalities of the game with a simple UI. Currently, our system supports the following functionalities:

- An algorithm that checks solutions and prints the invalid part of a wrong solution
- 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)
- Screens with simple design and their transitions between

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.

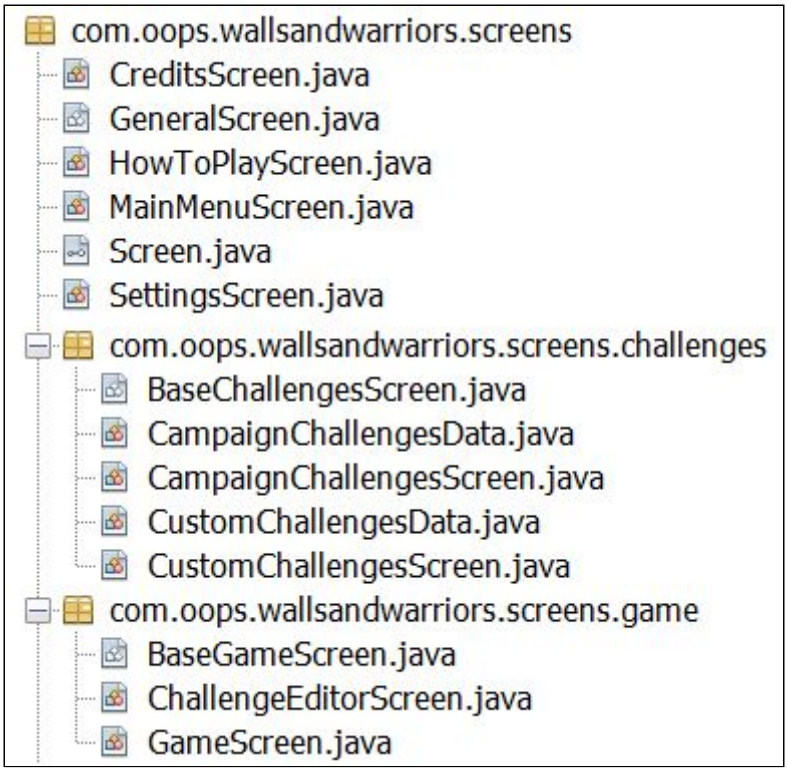

Figure 1: Revised `screens` package

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.

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 4).

Our implementation phase progressed as follows:

The implementation was divided into tasks to be completed by different group members. The most significant part of the implementation was to divide the implementation into tasks that were not depending on each other since the implementation of classes having relationships should be consistent. The dependent classes were treated as one task and given to only one group member to provide this required consistency. The implementation was divided into 14 sub-tasks, in 5 main tasks, to assign one main task for each member. The implementation was divided into the following sub-tasks:

- Implement an algorithm to determine if a solution is valid. Task #1
- Implement an algorithm to determine invalid parts of the wrong solution. Task #1
- Implement an algorithm to determine if a ChallengeData is valid. Task #1

- Implement GameScreen interactions. Task #2
- Implement ChallengeEditorScreen interactions. Task #2

- Implement a function to convert ChallengeData into a Base64 String. Task #3
- Implement a function to convert Base64 String into ChallengeData object. Task #3

- Read and list all challenges from a file. Task #4
- Write an imported custom challenge to file. Task #4
- Keep track of solved/unsolved challenges. Task #4

- Implement SettingsManager. Task #5
- Implement settings screen. Task #5
- Implement how to play screen. Task #5
- Implement credits screen. Task #5

Throughout this process, each member chose the task they wanted to undertake. The first step of the implementation was to come up with a simple User-Interface (UI) to construct the general structure of the program and add the regarding functionalities in the appropriate screens. The second step was to apply the designs of the objects, screens, controllers providing the essential functionalities. Among these tasks, implementation of the drag and drop interactions was prioritized above all in order to be able to test other gameplay functionalities.

5. User's Guide

5.1 System Requirements & Installation

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.

5.2. How To Use

5.2.1. Start a campaign challenge

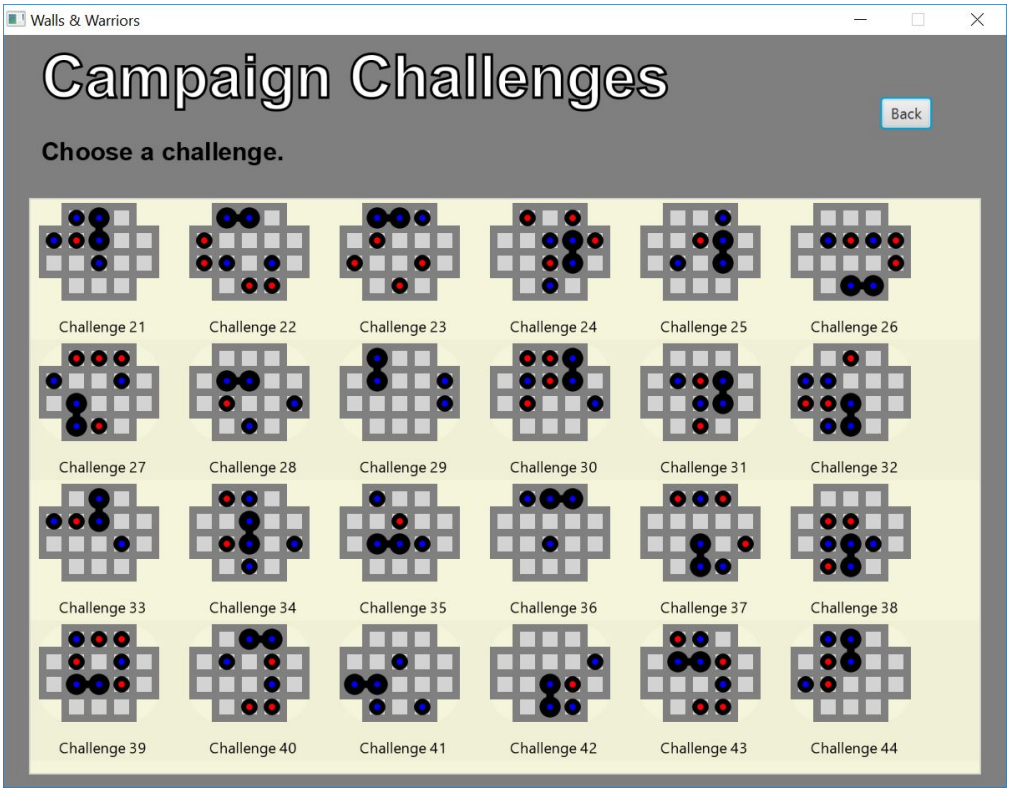

Figure 2: Campaign challenges screen

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.

5.2.2. Start a custom challenge

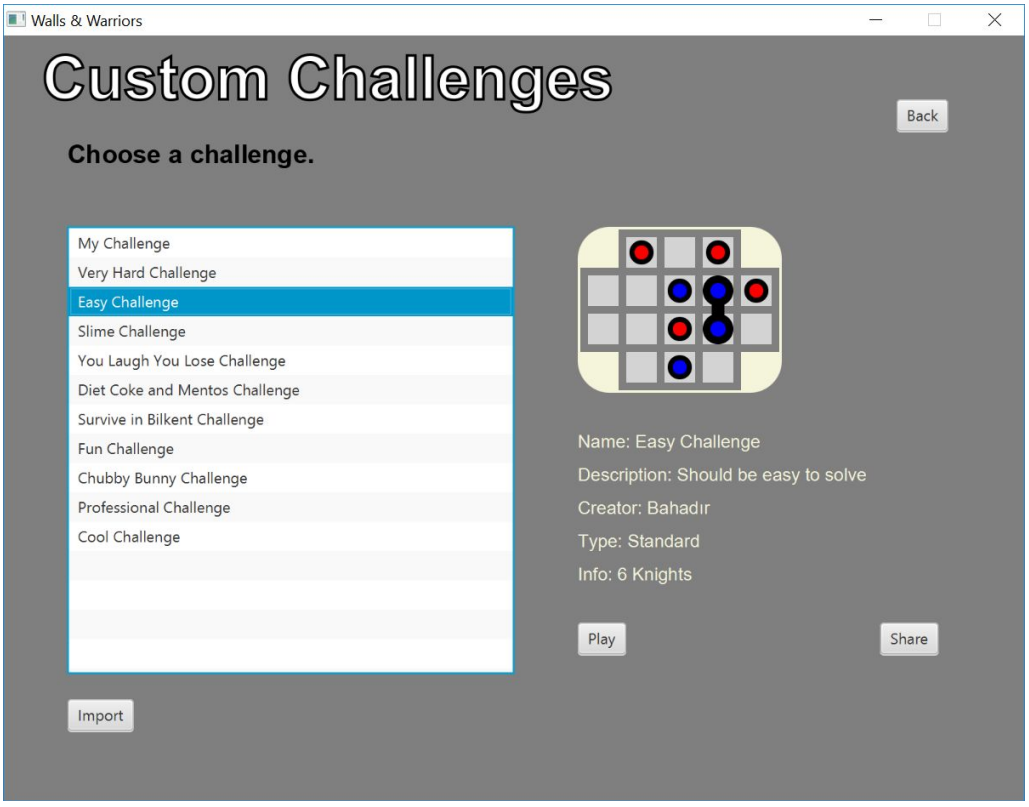

Figure 3: Custom challenges screen

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.

5.2.3. Play a challenge

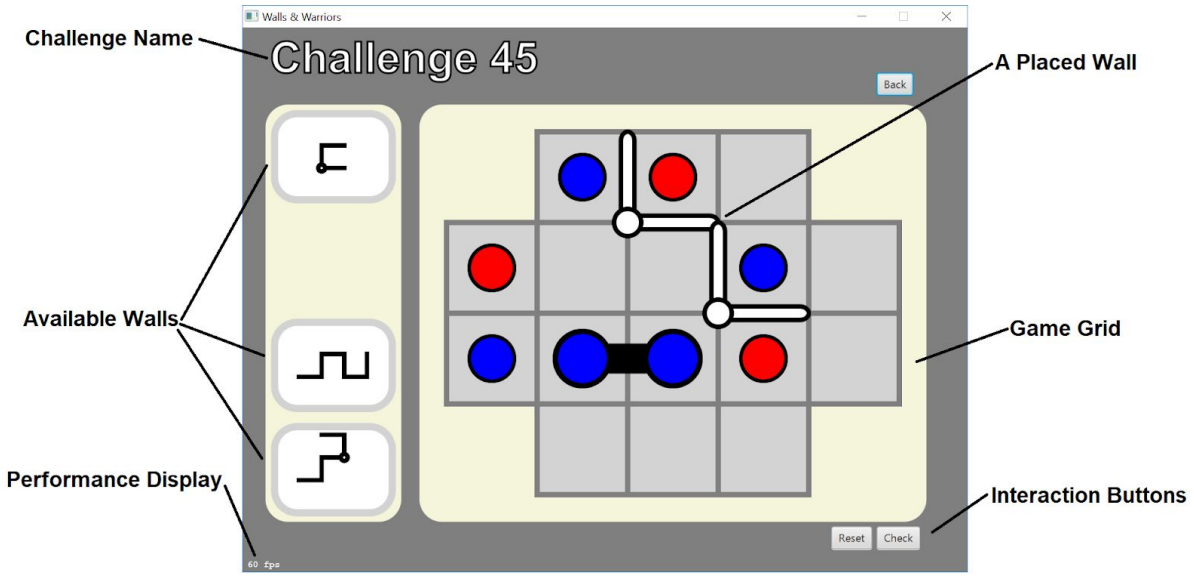

Figure 4: Game screen components

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 despite placing all the walls on the map, he/she can learn the incorrect attempt by clicking “check” button.

5.2.4. 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.

5.2.5. 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.

5.2.6. Create your own challenge and share

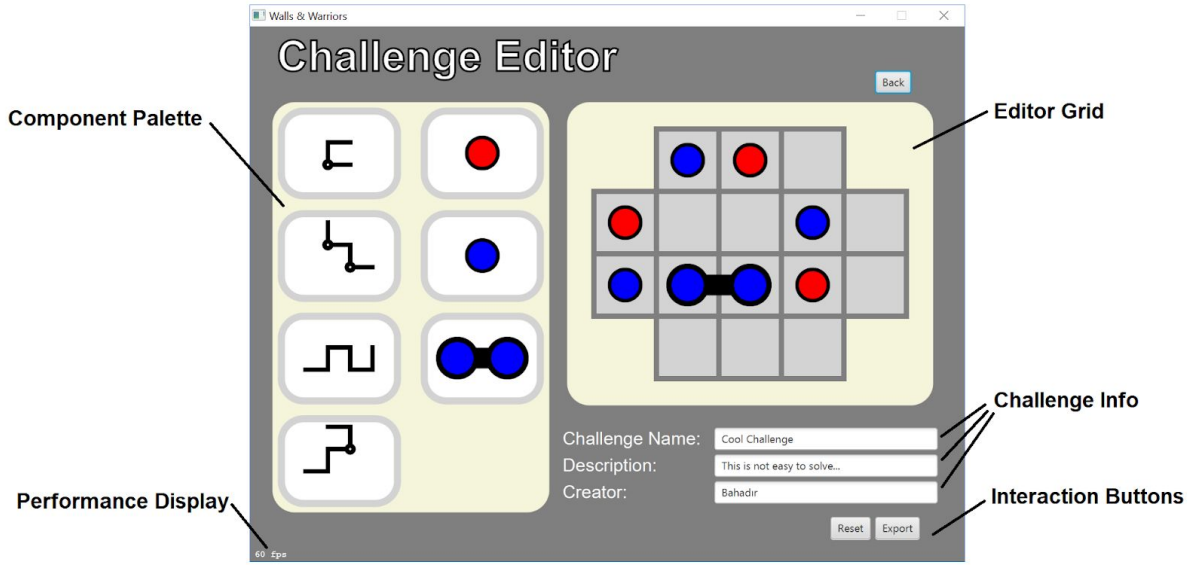

Figure 5: Challenge editor screen components

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.).

5.2.7. 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.

5.2.8. 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.

5.2.9. See Credits

To see the credits, a user needs to click the “Credits” button on the main menu. In the next window, credits of the game will appear.