Bilkent University

Department of Computer Engineering

CS 319 - Object-Oriented Software Engineering

Term Project - Final Report

Project Name : Slay The Spire

Group No : 1A

Group Name : 1A - SS

Group Members : Gamze Burcu Ayhan
Ekin Doğaç Öztürk
Farid Mirzayev
Okan Alp Ünver
Nihat Bartu Serttaş

Table Of Contents

1. Introduction **2**
2. Implemented Functions **2**
3. Design Changes **3**
3.1 Private Utility 3
3.2 Game Constants 3
3.3 Subpackages 3
3.4 Merging of Draw and Discard Piles with Play package 4
4. Lessons Learnt **4**
5. User’s Guide **6**
5.1 Build Instructions 6
5.1.1 Compile Through an IDE 6
5.1.2 Compile Using the Command line 6
5.2 System Requirements & Execution 7
5.3 How To Use 7
5.3.1 Play 7
6. Contributions **11**

1. Introduction

After the demo implementation has finished, new functionality and bug fixes have started immediately. Implementation was performed on various IDE's and we used a GitHub repository to interact synchronously with all community members contributing to the implementation as was done in the demo. By the end of the final version of the game, the implementation was completed for the fundamental requirements of regular "Slay the Spire" and with most of the additional functionalities that were presented in the Analysis Report. The implemented functionalities are further discussed in the section below.

2. Implemented Functionalities

Our purpose for the final demo was to complete the implementation in such a way that it would provide an optimized and neat UI for the fundamental and promised functionalities. Our system currently supports the following functionalities:

- Background music with adjustable volume
- Button to mute the music on the Settings screen
- Screen transitions
- Button to quit the game
- Function to hold user's progress on the map
- Function to draw map randomly for each game
- Function to keep player’s health until the floor ends
- Basic file operations (read and write challenges to file)
- Function to hold relics throughout the game
- Function to hold potions throughout the game
- Function to hold player’s deck until the game is over
- Function to animation to move card when mouse over the card
- Function to control the energy
- JSON object was used to keep cards data

- Implement the relics class
- Implement the potion class

3. Design Changes

We also found during our implementation that we were unable to implement our program exactly the way we planned it. We will discuss the differences between our design and implementation in this section. Such variations are not significant, and they do not alter our overall design structure.

3.1 Private Utility

We weren't sure of the exact content of our functions in our design process. Throughout the Implementation, we filled out their contents and found out that some other sub-functions were needed to call, to make our code cleaner. Such subfunctions have not been included in our design, as the exact algorithms have not yet been developed.

3.2 Game Constants

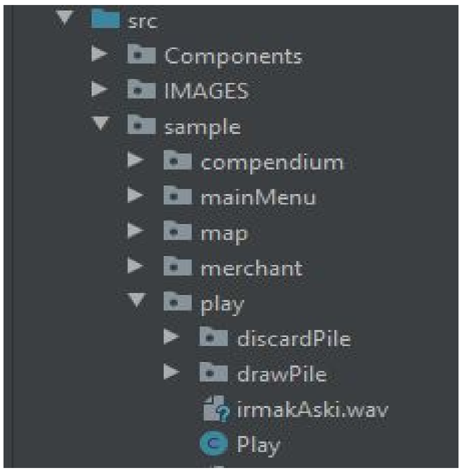
We noticed during implementation that many constants were being declared to avoid using "magic" numbers. That is why, we considered to define necessary constants to beginning of the classes as global variables such as numberOfColums, numberOfRows and so on.

3.3 Subpackages

Throughout the implementation of the game, we decided to split our packages into sub-packages. We created compendium, rest, main menu, map, merchant, play and so forth, and put into the sample package as it can be seen from the figure 1.
This division system allows us to write code in a more convenient way and trace the code easily when we encounter with a bug.

3.4 Merging of Draw and Discard Piles with Play package

For example, we have extra sub-packages into sub-packages such as Draw and Discard pile package are in the play package and play package in the sample package as it can be seen from the figure 1. We decided to merge them because they are strongly connected to play package. This way, we had a higher-cohesion system so we avoided the high-coupled case.



```
▼ src
  ▶ Components
  ▶ IMAGES
  ▼ sample
    ▶ compendium
    ▶ mainMenu
    ▶ map
    ▶ merchant
    ▼ play
      ▶ discardPile
      ▶ drawPile
        irmakAski.wav
        Play
```

figure 1: packages

4. Lessons Learnt

We have learnt how to divide an activity into tasks and tasks into subtasks during the implementation phase. We have also learned how to collaborate, in our case, through a source control tool called Git. The important part of the second iteration before implementation starts was to evaluate the incomplete functionalities and the future changes on the existing system by considering the order of priority. Those roles were delegated to various group members after the tasks were decided.
Dependence of activities was not as strong in this iteration as in the first iteration. Thus the tasks are, in this iteration, were distributed more equitably and freely. Improving the UI and handling errors, however, were not assigned to a specific member, but rather were determined by any of the members who would have the opportunity to do so.

Our implementation phase progressed as follows:
Before the implementation began, the most crucial aspect of the second iteration was to evaluate the missing functionalities and the changes that could be made to the existing implementation and to decide their order of significance. These tasks were delegated to different group members after the tasks had been decided. In second iteration, task dependency was not as high as the first iteration. So in this iteration, activities were distributed more evenly and freely. Improving the UI and fixing errors, however, were not allocated to a single member, but rather were decided by each of the members who would have the opportunity to do so.

The implementation was divided into 14 essential tasks. The implementation was divided into the following tasks:

● Add music volume and mute	Task #1
● Add music	Task #2
● Implement Map progress	Task #3
● Implement Merchant	Task #4
● Implement Rest	Task #5
● Implement Treasure	Task #6
● Implement Draw and Discard Pile	Task #7
● Implement showing the card in player’s hand	Task #8
● Add end turn button	Task #9
● Add animation to cards	Task #10
● Implement relics	Task #11
● Implement potions	Task #12
● Handle the erroneous cases	Task #13
● Improve and fix UI	Task #14

Each participant chose the task they wanted to perform in this process.Each participant chose the task they wanted to perform in this process. However, all group members are responsible for task #13 and task #14.

For the sake of completeness, the tasks we have already completed in Iteration 1 are listed below as well:

● Implement Game Screens	Task #1
● Implement Map	Task #2
● Implement Draw and Discard Pile	Task #3
● Implement Settings	Task #4
● Read and list all challenges from a file	Task #5
● Implement Combat	Task #6
● Implement Card JSON Objects	Task #7
● Implement an algorithm to draw map	Task #8
● Implement an algorithm to attack enemy	Task #9
● Implement an algorithm to end the game	Task #10

5. User's Guide

5.1. Build Instructions

There are two ways Slay the Spire can be built from its source code. The programmer can in both cases obtain 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 Through an IDE

If you choose to use an IDE to compile the game pursue the instructions below.

- The git repository is available for download.
- Build an empty file, in your IDE, called "SlayTheSpires."
- To compile and run the game, press Run into your IDE.

5.1.2. Compile Using the Command Line

If you want the game to be compiled on command line, follow the instructions below.

- Download the git repository.
- Browse to the sample folder which contains the src folder inside it.

- To compile and run the game, execute the following commands
  javac /src/sample/Main.java
  java src.sample.Main

5.2. System Requirements & Execution

The game will run in any Java runtime environment supporting operating systems. Since the game uses only a small amount of parts of memory (~300 MB of RAM) and storage (~200 KB of HDD), there is no significant memory and storage requirement for running the game.

The player will install Java on their machine, in order to run the game. They should access it (www.java.com/download) from the official website. What the player can do after that is to execute our "SlayTheSpire.jar" document by double-clicking it on. The distribution of the game will be provided with this jar.

5.3. How To Use

5.3.1 Play

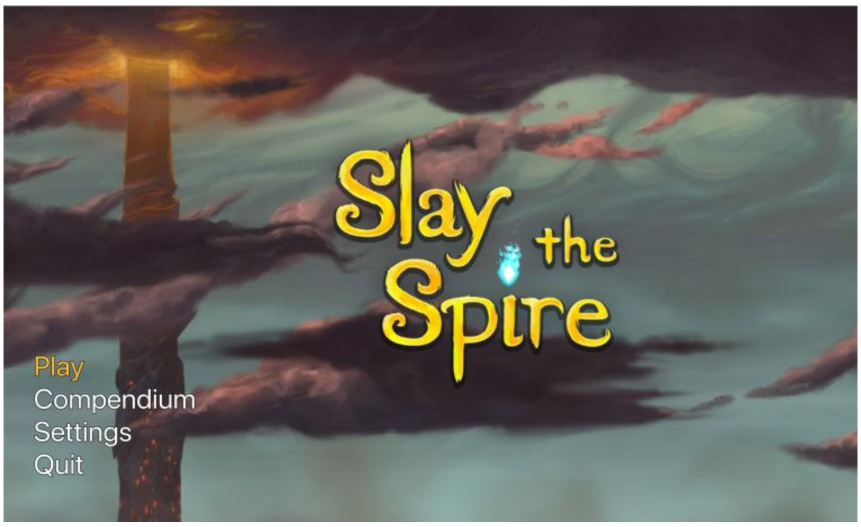


figure 2: Main Menu

In order to play the game, Player should click the play button from the main menu. Then the map screen appears. Player should choose one of the legend which is button by clicking left button over the mouse. Then combat starts, and the player shows combat window which is figure 3. On the top bar, there are 3 representations which are health of the character represented with tank on the screen, map button and settings button. Map button which is right side of the top bar allows the player to open a new window containing map informations.

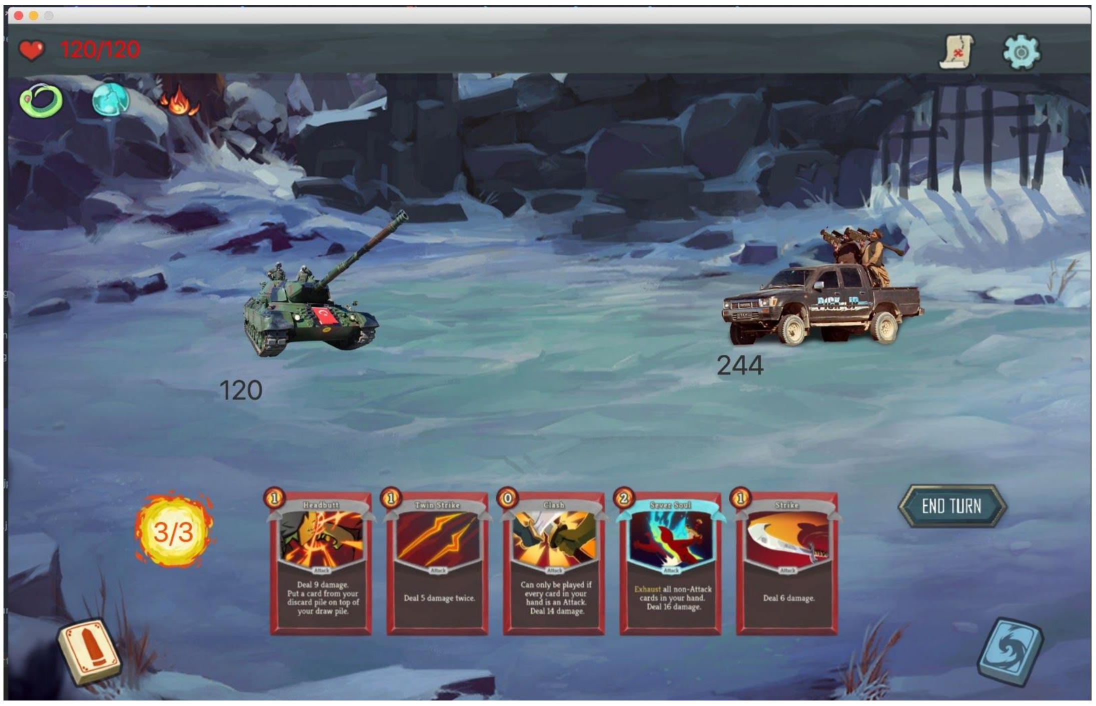

120/120
120
244
3/3
1
Headbutt
Attack
Deal 9 damage. Put a card from your discard pile on top of your draw pile.
1
Twin Strike
Attack
Deal 5 damage twice.
0
Clash
Attack
Can only be played if every card in your hand is an Attack. Deal 14 damage.
2
Sever Soul
Attack
Exhaust all non-Attack cards in your hand. Deal 16 damage.
1
Strike
Attack
Deal 6 damage.
END TURN

figure 3: Combat Screen

There is relic bar bottom of the top bar. It includes all of the relics that the player has. The player can see the properties of the relics by putting mouse pointer on desired relic. It boosts up the characters properties.

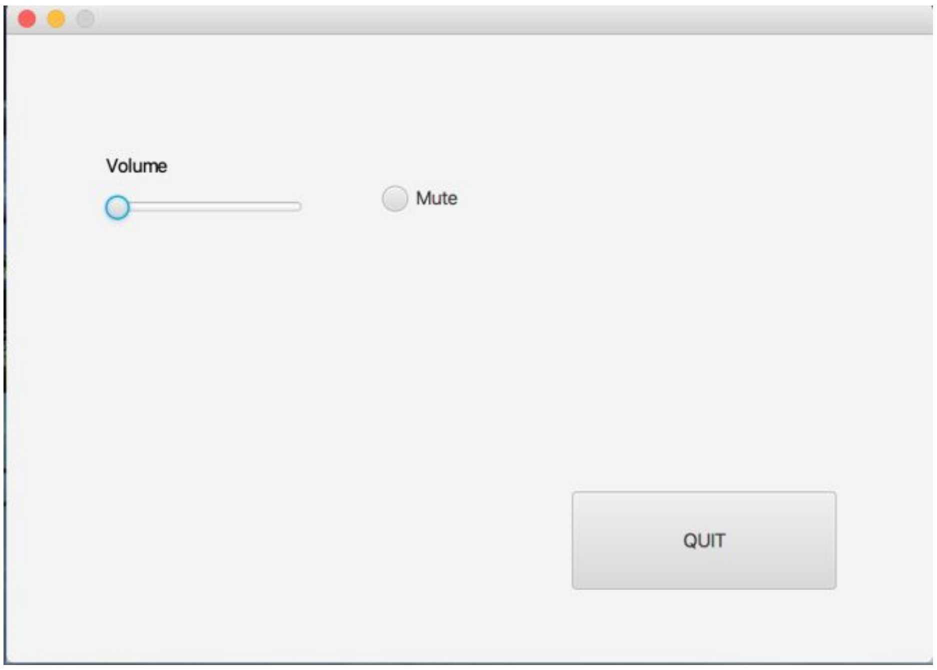
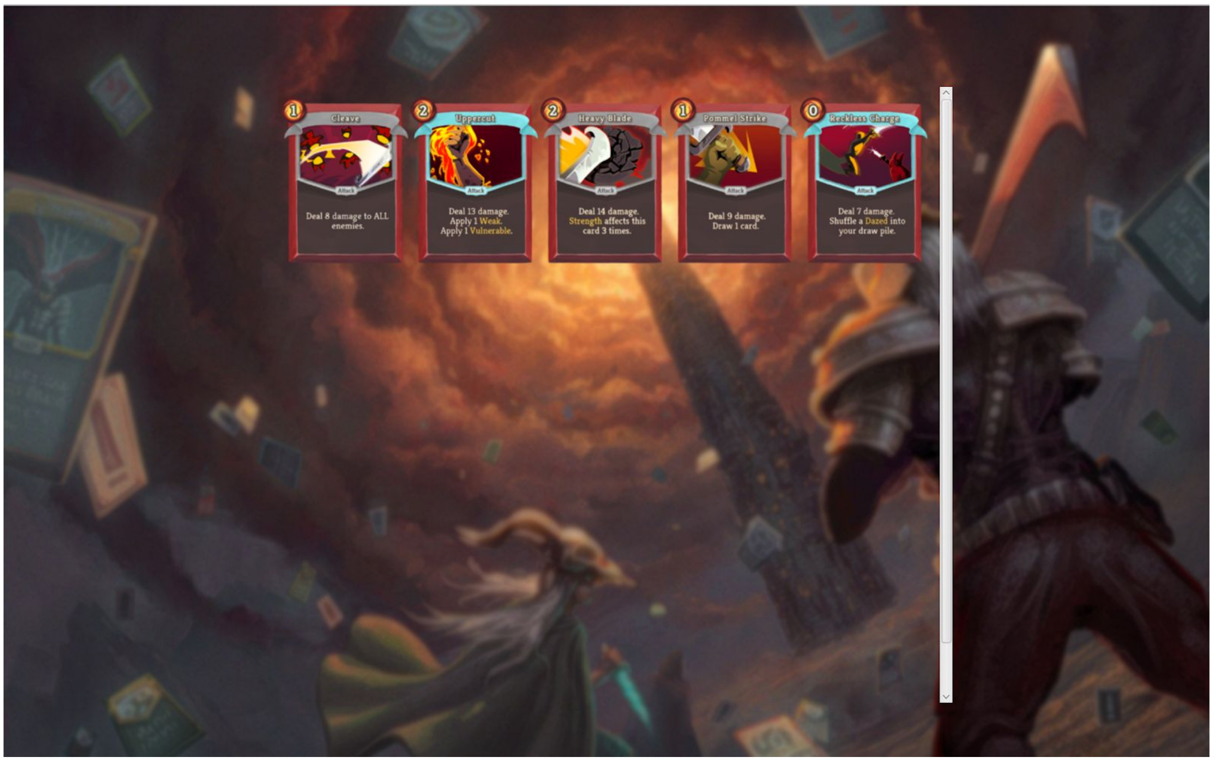


figure 4: Settings

Settings button allows the player to open new window which is figure 4. The player can adjust the music volume or mute it.

Furthermore, the first turn is given to the player. In order to attack the enemy represented with terrorist pick up one of the 5 cards on the screen should be clicked by left button over the mouse. In each turn player have 3 energy, and plays the cards accordingly. Each card has different energy cost which are 0, 1, 2 and so forth. The player does not have to spend all of his or her energy in order to pass to the next turn. The player passes to next turn by clicking the end turn button on the right side of the screen with left button of the mouse.



Deal 8 damage to ALL enemies.

Deal 13 damage. Apply 1 Weak. Apply 1 Vulnerable.

Deal 14 damage. Strength affects this card 3 times.

Deal 9 damage. Draw 1 card.

Deal 7 damage. Shuffle a Dazed into your draw pile.

figure 5: Draw and Discard Piles

Besides, the player can see the cards that can be given into his or her hand by clicking with the left button of the mouse on draw pile button which is left bottom side of the screen or discard pile which is right bottom side of the screen. Then new window appears which is figure 5.

At the end of the turn, the enemy damages to the character, and the health of the character is refresh on the top bar and left bottom side of the tank. The combat lasts until either health of the character or enemy becomes 0. If the enemy health becomes 0 wins the combat, and can choose another location on the map to continue the game. Otherwise, the game is over, and the player should start from the beginning.

6. Contributions

Ekin Doğaç Öztürk

1. Analysis Report Overview (Iteration 1)
2. Analysis Report Functional Requirements (Iteration 1)
3. Analysis Report System Models - Gameplay Sequence Diagram (Iteration 1)
4. Design Report High Level Software architecture (Iteration 1)
5. Design Report Final Object Design (Iteration 1)
6. Implemented Draw Pile (Iteration 1)
7. Implemented Discard Pile (Iteration 1)
8. Implemented Rest class (Iteration 1)
9. Implemented Merchant.fxml (Iteration 1)
10. Implemented Map class (Iteration 1)
11. Analysis Report Overview (Iteration 2)
12. Analysis Report Functional Requirements (Iteration 2)
13. Analysis Report System Models - Gameplay Sequence Diagram (Iteration 2)
14. Analysis Report System Models - Combat Sequence Diagram (Iteration 2)
15. Final Report Contributions (Iteration 2)
16. Final Report Lessons Learnt(Iteration 2)
17. Final Report Design Changes (Iteration 2)
18. Contributed Play class (Iteration 2)
19. Handled several alerts (warning, information type).
20. Fixed several kinds of bugs.
21. Improve Quality of code

Farid Mirzayev

1. Analysis Report Introduction (Iteration 1)
2. Analysis Report Functional Requirements (Iteration 1)
3. Analysis Report System Models - Use Case Diagram (Iteration 1)

4. Analysis Report System Models - State Diagram (Iteration 1)
5. Design Report High Level Software architecture (Iteration 1)
6. Design Report Subsystem services (Iteration 1)
7. Implemented Main Menu class
8. Designed Combat screen
9. Implemented Play class
10. Contributed Map class
11. Implemented Draw pile class
12. Implemented Discaard pile class
13. Contributed card class
14. Contributed character class
15. Contributed merchant class
16. Contributed rest class
17. Contributed Monster
18. Implemented necessary methods for the functionality (Victory and Game over)
19. Contributed Potion class
20. Contributed Main class
21. Implemented necessary methods for the functionality (refreshScreen(), changeScreen() etc.)
22. Handled several alerts (warning, information type).
23. Fixed several kinds of bugs.
24. Improve Quality of code
25. Analysis Report (Iteration 2)
26. Final Report Contributions (Iteration 2)
27. Final Report Implemented Functionalities (Iteration 2)

Nihat Bartu Serttaş

1. Analysis Report Functional Requirements (Iteration 1)
2. Analysis Report System models - Class Diagrams (Iteration 1)
3. Analysis Report Overview (Iteration 1)
4. Design Report High Level Software architecture (Iteration 1)

5. Design Report Subsystem services (Iteration 1)
6. Implemented Card class
7. Implemented potion class
8. Implemented relic class
9. Implemented monster class
10. Handled several alerts (warning, information type).
11. Fixed several kinds of bugs.
12. Improve Quality of code
13. Handled several alerts (warning, information type).
14. Fixed several kinds of bugs.
15. Improve Quality of code
16. Analysis Report (Iteration 2)
17. Final Report Contributions (Iteration 2)
18. Final Report Implemented Functionalities (Iteration 2)
19. Final Report Design Changes (Iteratioon 2)
20. Contributed map class
21. Implemented Character class
22. Contributed Play class

Gamze Burcu Ayhan

1. Analysis Report Overview (Iteration 1)
2. Analysis Report Functional Requirements (Iteration 1)
3. Analysis Report Non-Functional Requirements (Iteration 1)
4. Analysis Report Page layout (Iteration 1)
5. Design Report Low-Level Design (Iteration 1)
6. Design Report Subsystem services
7. Design Report Page Layout (Iteration 1)
8. Implemented Settings class (Iteration 1)
9. Analysis Report (Iteration 2)
10. Final Report Contributions (Iteration 2)
11. Final Report Introduction (Iteration 2)

12. Final Report Implemented Functionalities (Iteration 2)
13. Final Report Design Changes (Iteration 2)
14. Final Report Lessons Learnt (Iteration 2)
15. Contributed Play class (Iteration 2)
16. Added animation to cards (Iteration 2)
17. Fixed several kinds of bugs (Iteration 2)

Okan Alp Ünver

1. Analysis Report Introduction (Iteration 1)
2. Analysis Report Functional Requirements (Iteration 1)
3. Analysis Report System Models - Activity Diagram (Iteration 1)
4. Design Report High-Level Software Architecture (Iteration 1)
5. Design Report Overview (Iteration 1)
6. Design Report Subsystem Services (Iteration 1)
7. Implemented Potion JSON Object (Iteation 1)
8. Final Report Implemented Functionalities (Iteration 2)
9. Final Report Contributions (Iteration 2)
10. Final Report Design Changes (Iteration 2)
11. Contributed Play class (Iteration 2)
12. Fixed several kinds of bugs (Iteration 2)
13. Contributed card class
14. Contributed main menu class
15. Contributed Character class
16. Improve Quality of code