In The Name Of God

Sharif University of Technology

Computer Science Department

Advanced Programming Project
Phase 2

Hearthstone

Amirhossein Afsharrad

95101077

May 6, 2020

Contents

1 Introduction

In this document we briefly explain the important points of the implementation of Hearthstone project's second phase. The goal for this phase is to implement the graphical interface of the game, so the game logic has been through slight changes compared to the previous phase; and the game is not ready to be completely played yet. You may find all packages and classes attached to this report, or you can check them using the following link on GitHub:

`https://github.com/AmirAfsharrad/Hearthstone`

The java version of this project is java11 and it has been built using *Gradle*. Apart from java standard packages, only the two following dependencies have been added to the `build.gradle` file for the `gson` and `json` respective packages:

- `compile 'com.googlecode.json-simple:json-simple:1.1.1'`
- `compile group: 'com.google.code.gson', name: 'gson', version: '2.8.5'`

2 What has changed since phase 1?

Apart from full implementation of the graphical user interface, the game logic has only had small changes. The most important item to be mentioned is the change in the identity of a *deck*, which used to an array list for each hero; but in this phase has turned into an independent object who may have a hero and a name. To be more specific, the class `Deck` has been declared containing a name, a hero, and an array list of cards. Each user may have as many decks as he wishes, and can edit all the attributes of a deck. Having to save user's all decks also resulted in some modifications of the code responsible for saving and loading the game.

Another change to be mentioned is the addition of more places to the game. In phase 1 we had designed the abstract class `Place` which was an abstraction of all places a user can visit throughout the game. These places included `SignInOrSignUp`, `MainMenu`, `Store` and `Collections`. For this phase we have add more places: `Playground`, `Status`, and `InfoPassive`.

Apart from the two points just mentioned, there are no major differences in the game logic compared to the previous phase.

3 The GUI General Structure

As mentioned before, the major goad to this phase was to implement the graphical user interface of the game. To achieve this goal, we have several new classes, the most important of which is the *Mainframe. The Mainframe acts as a controller and an intermediary link between the logic and the GUI.* To be more specific, no graphical panel makes a change in the game logic, unless making in through the mainframe. This makes the logic and the GUI quite separate, which is a key feature in the program's structure.

The next three sections are dedicated to a brief review of the five packages designed for the GUI. After that, we also review the resources we have used for this phase of the project. Before starting that, to gain a better understanding fo the general structure, note that the main frame is managed be the swing's `CardLayout` layout manager, and each place of the game (which has a panel) is a card in this card layout. As a result, the game controller brings the appropriate card to the front whenever an event happens in the game.

4 Package `GamePanels`

This package is the most important package responsible for the game graphics. Inside that, there exists an abstract class called `GamePanel`, which is the abstraction of a panel existing inside a game. To be more specific about the panels, note that there is a one to one correspondence between the game places and the game panels. For each place, we have designed a panel to let the user what he can do when he is inside that place. As a result, there exist seven classes in this package who extend the `GamePanel` abstract class.

While designing these panels, we have tried to make the most use of swing's layouts, and we also have used the absolute positioning method whenever needed. We shall review these panels briefly in the following subsections.

4.1 `LoginPanel`

This panel is responsible for taking the username and password as the input, and to log in for a previously-registered user, or to create a new account for a new one. Figure 1 shows this panel.

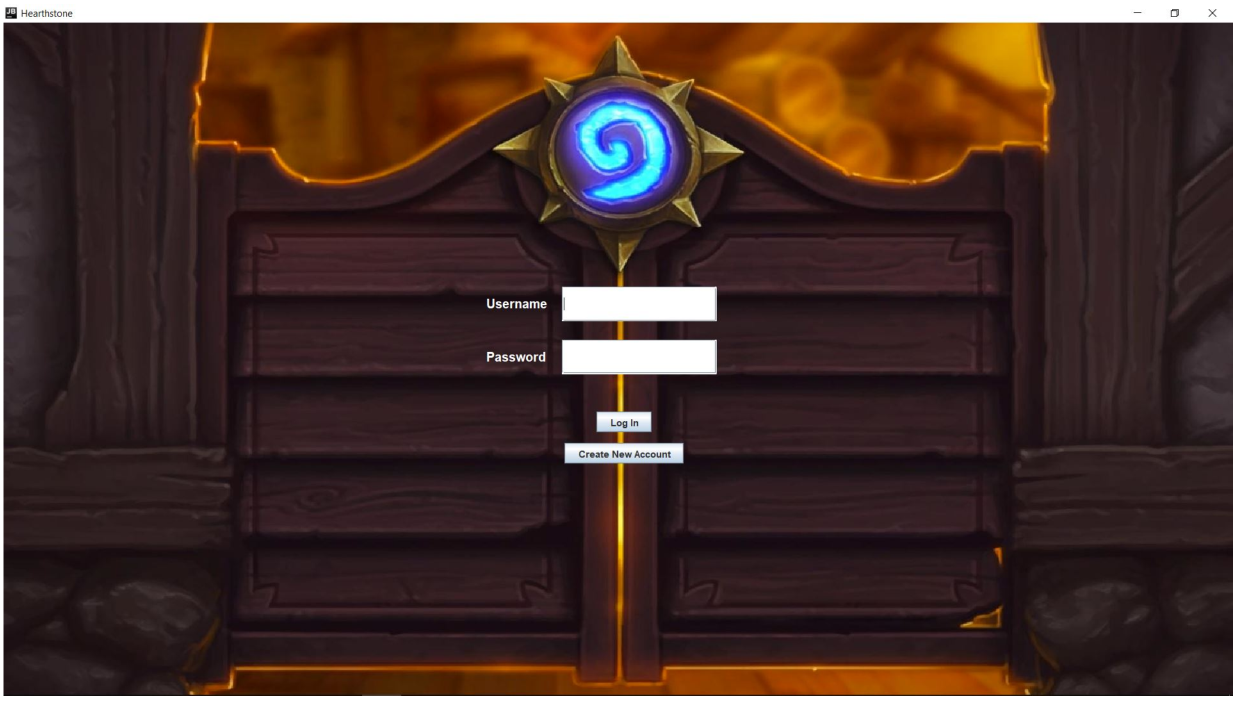


Figure 1: The login panel

4.2 `MainMenuPanel`

This panel is responsible for the main menu of the game, which leads the user to collections, store, status, or playground. It consists of a background and simple buttons as depicted in figure 2.

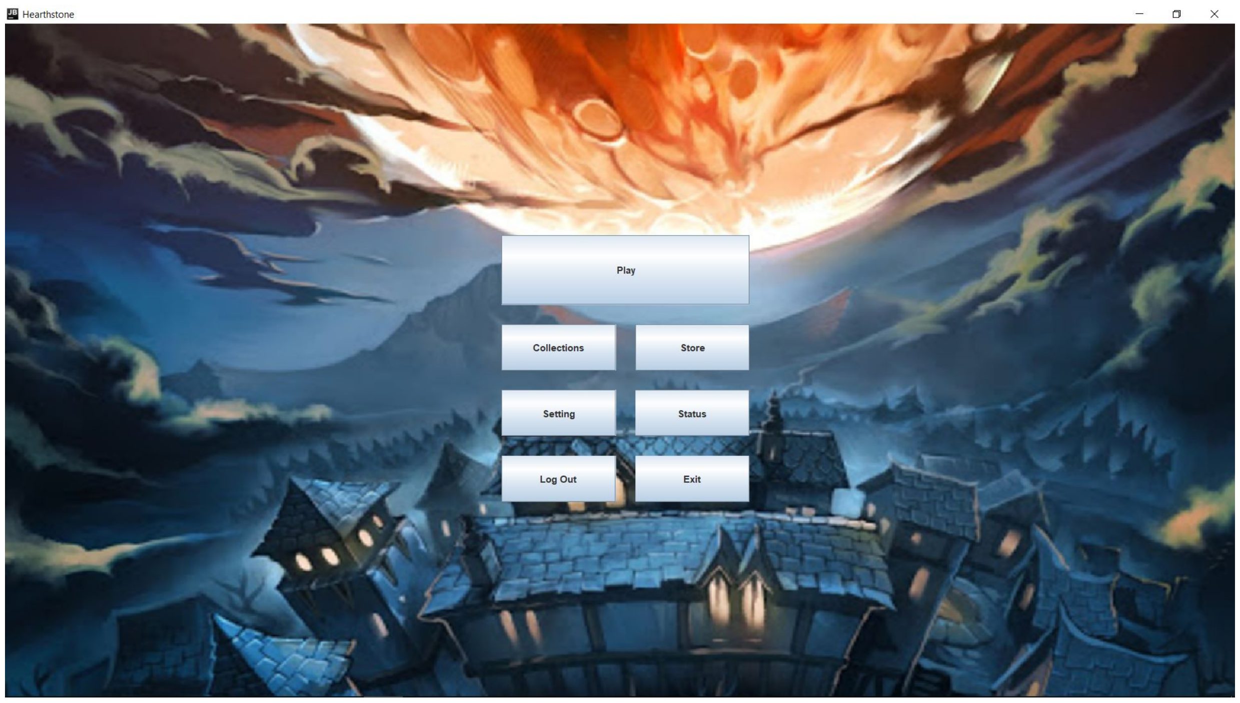


Figure 2: The main menu panel

4.3 `CollectionsPanel`

This panel is the graphical implementation of the games collections. It consists of some smaller panels, some responsible for holding different button, and one responsible for showing cards. Each card is actually a `JButton` with a background, so one can easily click on them to directly buy or sell a card. On the panel on the right of collections, a `CardLayout` layout manager has been used to have two cards, one for a list of decks and another to show the cards present in each deck. These two are shown in figures 3 and 4 respectively.

4.4 `StorePanel`

This panel, as depicted in figure 5, is very similar to the collections panel and the implementation is quite the same. Actually the collections panel almost contains an independent store within itself, so one can easily design a store panel after having one for the collections.

4.5 `StatusPanel`

This panel has also inherited most of its graphical features from the collections panel (by inheritance, we mean the similarity in appearance not the popular concept of programming). It includes two separate panels, one for showing the statistics (the larger panel on the left), and another to show a list of decks or the cards inside a deck. This is shown in figures 6 and 7.

4.6 `PlayPanel`

This is the panel on which the actual game takes place. We have used the absolute positioning system on this panel, since we need to organize the objects neatly on the background picture.

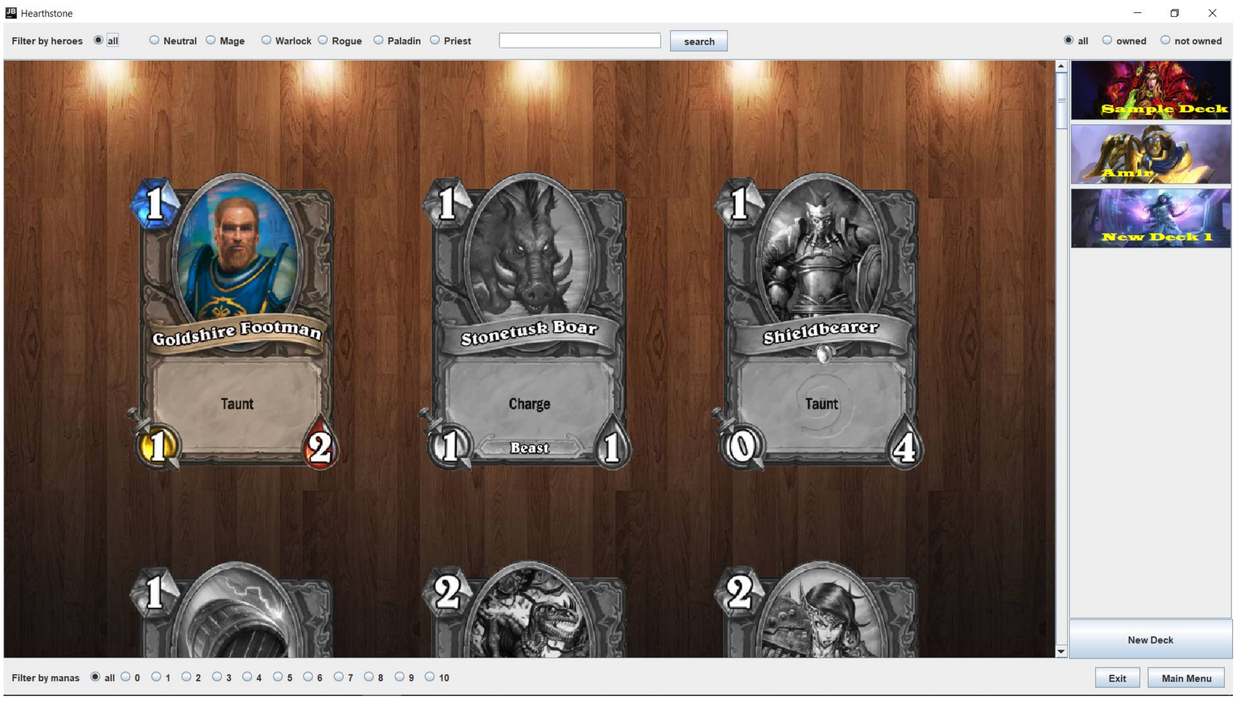
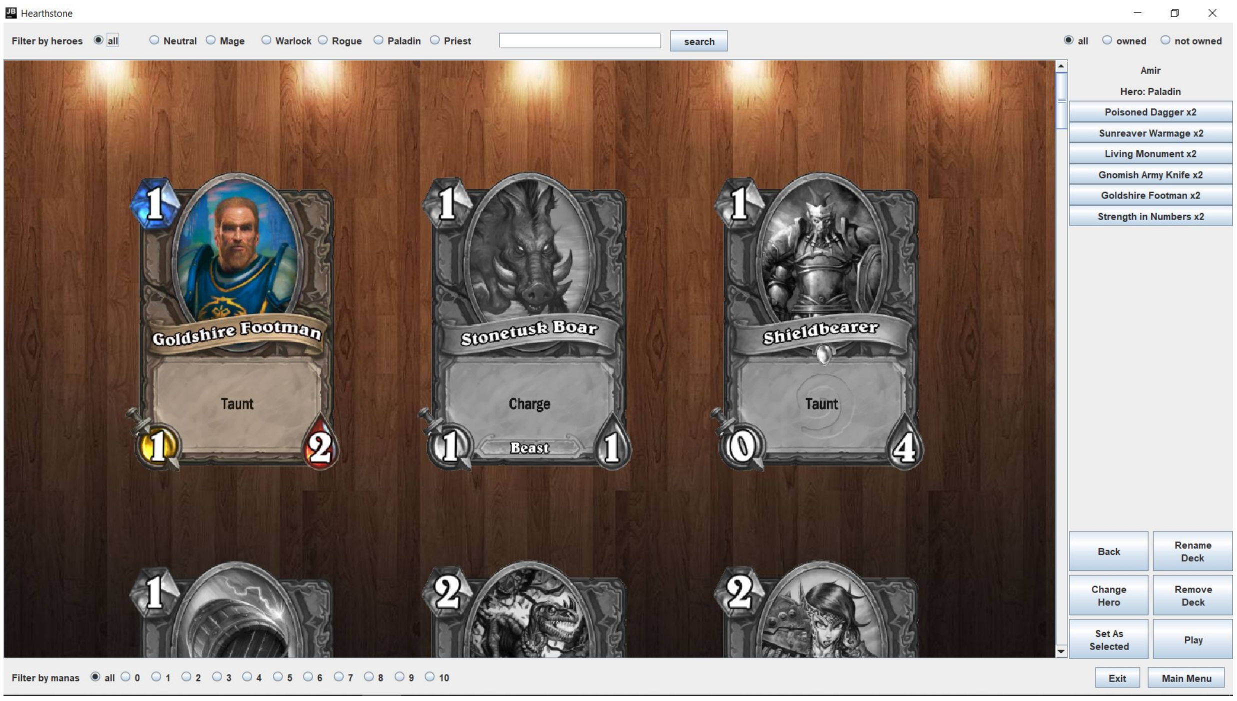


Figure 3: The collections panel and the list of decks



Amir
Hero: Paladin
Poisoned Dagger x2
Sunreaver Warmage x2
Living Monument x2
Gnomish Army Knife x2
Goldshire Footman x2
Strength in Numbers x2

Figure 4: The collections panel and the cards of a deck

Also for each card we have used a panel, so we can recognize clicks, mouse enters, and mouse exits; and moreover, it is an easy element to work with. The result is depicted in figure 8.

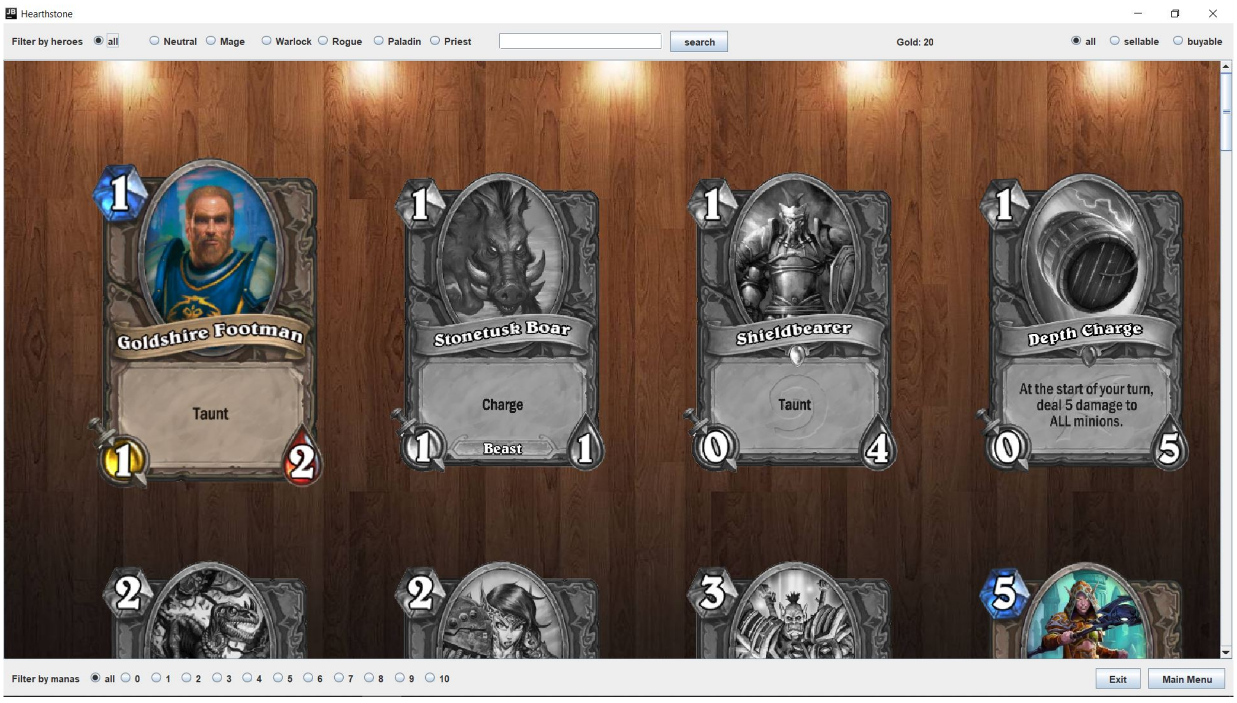


Figure 5: The store panel

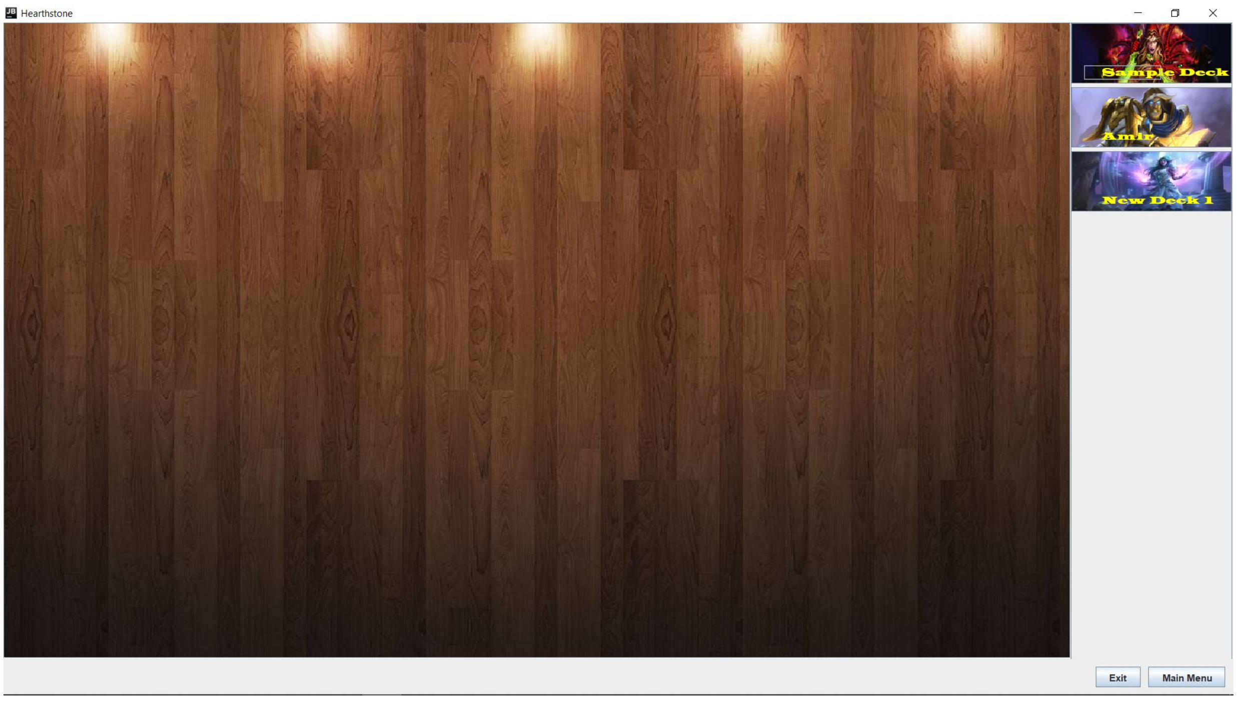
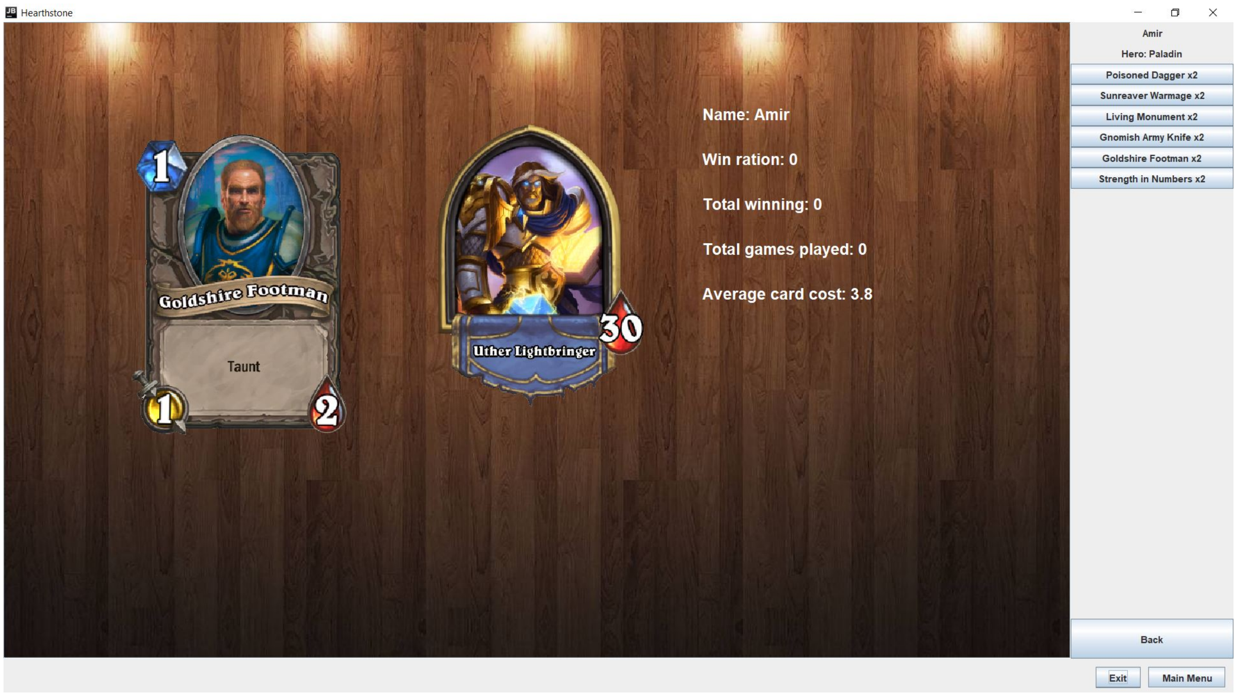


Figure 6: The status panel with no decks selected



Name: Amir

Win ration: 0

Total winning: 0

Total games played: 0

Average card cost: 3.8

Figure 7: The status panel with a selected deck

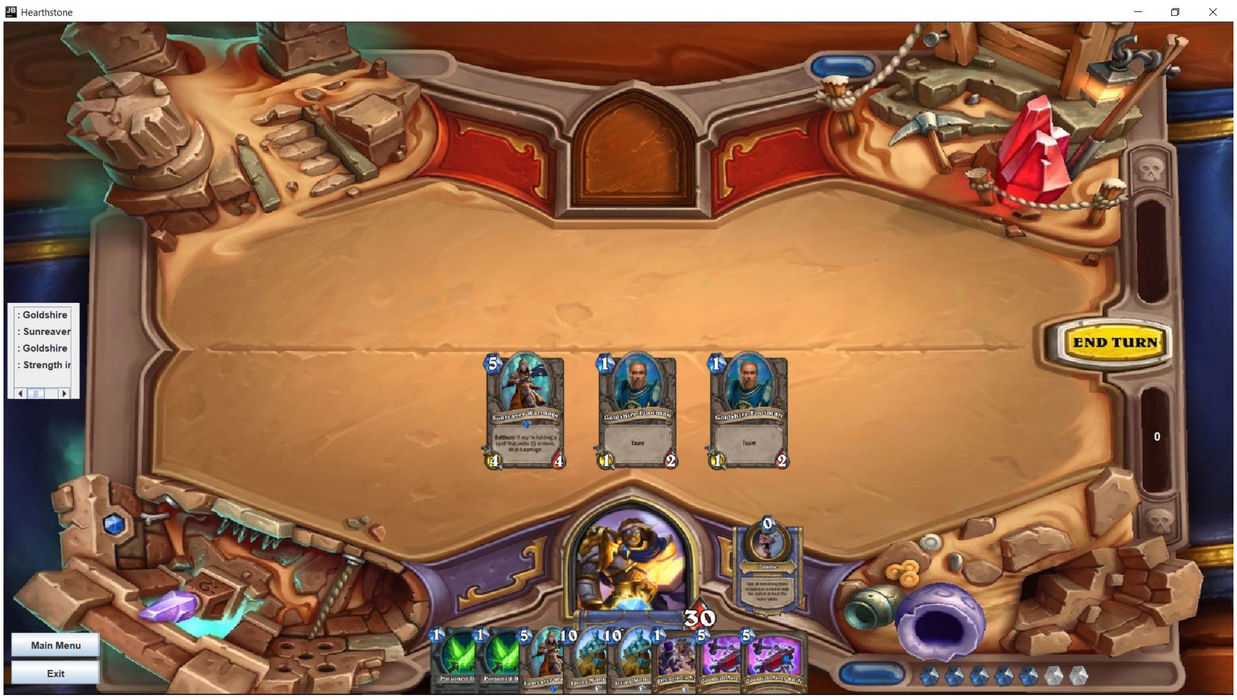


Figure 8: The battlefield of the game

5 Packages `Events` and `Listeners`

The concepts of *events* and *event–listeners* are of great importance in our program design. Using events and listeners, we can model the occurrence of an event and the jobs other modules should do after that. We have designed many event classes (all extending `EventObject`) and many listeners (all extending `EventListener`) although we could have done without them, or at least without most of them. Many of these classes have the same code within themselves, but the reason we created them separately is to make the structure of our design more easily understood. Most of these events happen when a button is pressed, and the listener (which is set by the main frame) will run a method to do the proper job. This structure results in the GUI being almost unaware of what goes on in the game logic, which was initially one of our main design interests.

6 Packages `Dialogs` and `Utils`

Apart from the main panels, we sometimes need other graphical utilities. This is why we have designed two more packages. The package `Dialogs` consists of different classes which produce a new small frame to get a response from the user (or to simply deliver a message). Needing them for simple message delivery, getting a yes–no response, getting a multiple choice response, or a text input; we have designed four dialog classes.

The package `Utils` also includes some extensions of standard classes, which could be useful in our program. The class `BackgroundedPanel` extends `JPanel` and creates a panel with a background image. The class `backgroundJViewport` extends the class `JViewport` to implement a structure to have a panel with scroll bar and background image. Finally, the class `DynamicIcon` is used to create a `JButton` which has an image icon and a text written over it.

7 Resources

In this section, we introduce all external resources we have used for this phase of the project.

7.1 The Code

Most of the code is generated by the author without external help (other than some normal internet searches). The only part to be mentioned specifically is the class `ImageLoader` from the package `Utilities`, which has been initially taken from the asteroid project provided by the course instructors.

7.2 Game Photos

This project includes many pictures for the GUI. These pictures have either been downloaded directly, or designed by the author. The first group have mostly been downloaded from `https://hearthstone.gamepedia.com/Hearthstone_Wiki`. For the cards which we needed to design, we have used `http://www.hearthcards.net/`, a website that lets you design any card with an incredibly efficient interface.