| Student name | Student ID |
|---|---|
| Farzad Hayatbakhsh | 15424500 |
| Daniel Cha | 21797924 |

# Monster Fighter Report

## Application Structure and Design Decisions

### Monster and Item subclasses

The different types of monsters/items were implemented using subclasses instead of just having one Monster/Item class as we found that the types of monsters/items utilized the same functions. The monsters only differ in monster abilities (if any) and items only differ in usage (e.g. level up potion leveling up monster). Having them in subclasses prevents hard coding and makes it easier to balance a specific monster/item by changing variables in their subclass.

### Inventories

There are three inventory classes in the game: BattleInventory, ItemInventory, and MonsterInventory. Each inventory extends the ArrayList class and adds some functionality on top. The three inventories contain some common functionalities but also some unique ones that apply only to that inventory. This will be addressed further in the page two.

### Purchasable interface

Purchasable is an interface that is implemented by the Item and Monster classes. The purchasable interface holds the methods common between the two such as getName(), getCost(), etc. These methods are used by our custom GUI templates that display information about a purchasable. This means that the templates be used for both items and monsters.

### Exceptions

There were a few cases in which a custom exception would benefit the game and thus we added 6 exceptions. The new exceptions helped us to execute specific actions that were related to the exception, e.g. display an error message when the player has insufficient funds. There were 2 exceptions that were similar, InvalidTargetException and InvalidValueException, you could argue that an invalid target exception could be caught by the invalid value exception as well, but we wanted to separate them to allow us to be clearer with what exception is being thrown.

### GUI Design

The GUI consists of a GraphicalUserInterface class that runs the Swing application and manages the screens. There is a Screen superclass extending JPanel that sets up a basic panel template. Each screen then extends that Screen class and implements the content of the panel. We created templates for certain label and buttons that were used frequently to improve code reusability.

### Singleton classes

The GameEnvironment and the GraphicalUserInterface class were designed as singletons because there is only one instance of each in the game at a time. This makes it easier to access these instances from anywhere without having to pass them around in arguments.

## Unit Test Coverage

We got a high unit test coverage for the Items package (94.1%), the Monsters package (94.3%), and the Main package (60.3%). The coverage of the Main package was slightly lower than the other packages because it contains the CommandLine class (0%) which is not testable, and the Battle (41.7%) class which contains some randomly occurring events, so it was not easily testable. The GUI package was not unit testable due to having no functionality besides Swing code. However, we did not put off testing the GUI package and CommandLine class. We user-tested the GUI package and CommandLine class through playing the game and experimenting with many alternative flows.

## Retrospective

### What went well

The result is a smooth and bug free game. The UI looks clean given our limited experience with Java prior and the amount of time that we spent on the project.

### What did not go well

Our initial planning was not comprehensive enough. We changed our design many times during the development. This means our progress was slower because we had to make changes to previously written code.

We prioritized our efforts on the essentials to get the game working before adding any additional functionality. The game is simple; however, it lacks some features: music, sprites, animations, and difficulty balancing.

We struggled with coming up with a good design for the inventory classes. They are all extensions of ArrayList, but each with their differences so we had to make them separate classes. The overlap between the three inventories could have avoided the repetition with a better design.

### Improvements

This project taught us the importance of planning and documentation. If we were to do this project again, we would spend more time on designing the program structure and be more specific with what exactly each class does. This would have saved us a lot of time wasted down the road in making design changes and redoing code. We were lazy in our commenting of classes and methods and often had to go back and think about what a piece of code was meant to be doing.

Another improvement that we can make is to be consistent with unit testing as we write code rather than leaving it all to be done at the end.

We learned to place a higher emphasis on planning, documentation, and testing in future projects.

## Thoughts and feedback

Overall, we believe that this project improved our skills on Java programming and working with GUI’s. It helped us understand the effectiveness of inheritance, interfaces, and thorough testing. Although some aspects of the game could’ve been better, we still believe that the produced outcome was a success.

## Hours spent

Daniel spent a total of 130 hours and Farzad spent 150 hours on the project.

## Percent contribution

We agreed that there was a 50:50 contribution to the project overall.