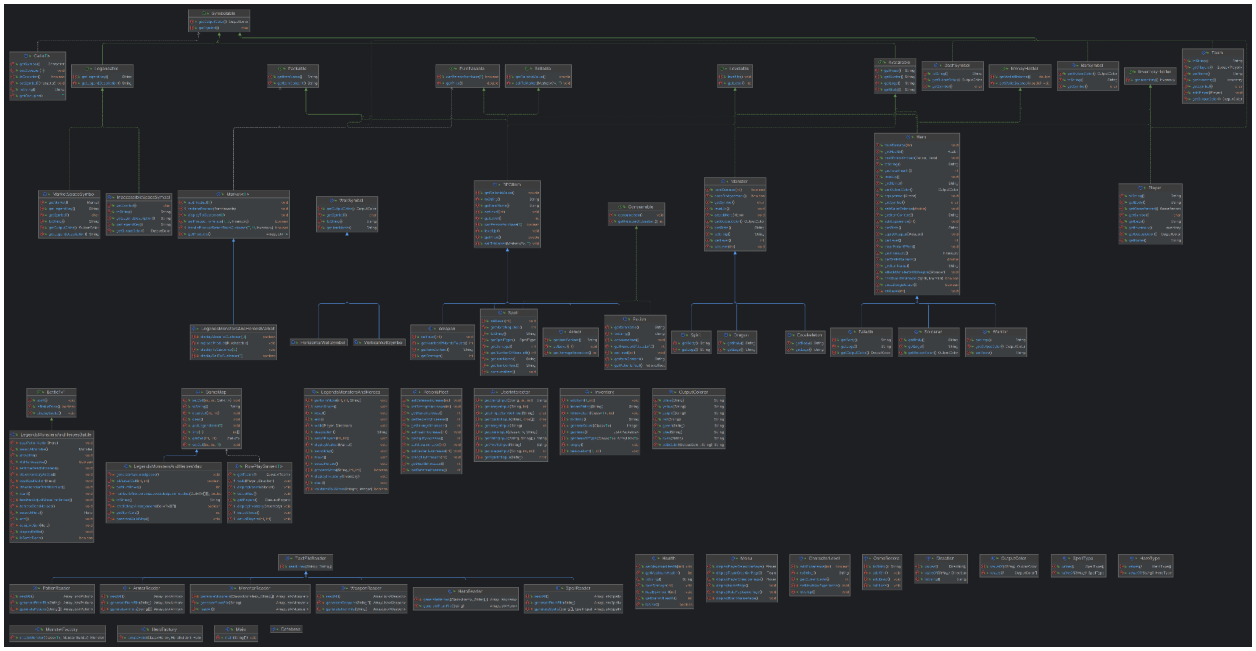
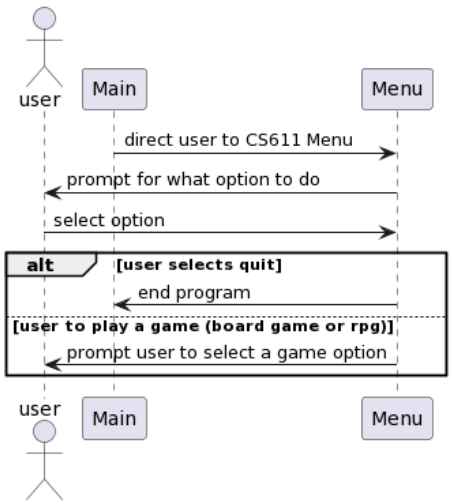
CS611 Assignment 4 Design Document

This document covers the high-level and compactly written object oriented design of Reshab Chhabra's CS611 Assignment 4. *Note: classes could in fact be made to abstract more content, but as we apply bias for action, we will add them once they are fully needed (SOLID).*

Project Architecture



UML Diagram for all my classes. This is also located in the Gradescope submission as "`Assignment4UMLDiagram.png`" in case this is too small



To begin, the code is initialized with the behavior interaction between Main.java and Menu.java. Then, once the user has decided on a game to play, the menu will `start()` that specific game.

This design structure to initialize a game has been used since Assignment 1 and did not need to be adjusted.

Overall Organization Pattern: Key Design Choices

`RolePlayGame`: as a start to creating the game, the game will extend the abstract class RolePlayGame which respectively shares common features with what a role play game has. This abstract class would then implement a `Game` interface.

`HeroBuilder` and `MonsterBuilder`: These are helper static classes that allow for a neat generation of a hero. The inspiration behind this is as there are many parameters to define a hero, and not all are necessarily required, it is better to `build()` a hero using chained calls e.g. `HeroBuilder.health(1050).level(2).build();`. This is similarly done with creating monsters too.

Factory classes e.g. `HeroFactory`, `MonsterFactory`: Using good practice, these factory patterns are used to create a hero or monster of a specific type, using the builder to generate the object.

`TextFileReader` and SubReaders: As the instructions in the assignment are to generate heroes, monsters, and rpg items, these all have respective classes such as HeroReader or SpellReader.

`GameMap`: Rather than using a Board that was used for the first 3 assignments, a map has been created. The key difference here is that the map has a `legend` to display items in the map's significance. The map itself is a 2D array of `Cell<T>` objects which are `Symbolable`, as symbolable objects can be seen in a map, like `MarketSpaceSymbol`. For Legends Monsters and Heroes, a game-specific class `LegendsMonstersAndHeroesMap` was made.

`Battle`: to represent a battle, the interface `Battle1v1` has been made which is used in `LegendsMonstersAndHeroesBattle` class.

`Market`: an abstract class for a generic market `Market<T>` was made, where the market sells items of type `T`. For the game Legends Monsters and Heroes, the market `LegendsMonstersAndHeroesMarket` has been made branching out of it.

Strategy Pattern with interfaces: To ensure full flexibility, multiple interfaces have been defined, including `Avatarable`, `Consumable`, `Sellable`, `Purchasable`, and `Levelable`. This idea was inspired from when I used this in Assignment 3: Quoridor with `Packable`. `InventoryHolder` and `MoneyHolder` are similar strategy pattern-based patterns.