1 Assignment

This week we are extending last weeks assignment to make a more complex game. We are adding monsters to the game. Monsters prevent players from leaving the room they are in until they have been defeated in a duel. Players will also be able to level up which will improve stats and restore health.

In completing this assignment you will be working in the same github repository as you did last weekend. You can of course start as soon as you wish even if you have not had your code review. This means that to present without being late you may have to present an older commit. If you need help with this your moderator will be able to help you with it.

One of the key goals in this assignment is to help you to understand the experience of extending code that was written for one purpose to a new or larger purpose that was not explicitly planned for in the original design.

2 JSON Schema

The JSON schema has been modified. Make sure to note these modifications carefully and make the corresponding changes in your code.

```
Direction := {
  directionName     : String
  room              : String
}

Monster := {
  name              : String
  attack            : Double
  defense           : Double
  health            : Double
}

Item := {
  name              : String
  damage            : Double
}

Room := {
  name              : String
  description       : String
  directions        : Direction []
  items             : Item []
  monstersInRoom    : String []  // Names of monster to look up in the monster array
}
```

```
Player := {
  name            : String
  items           : Item []
  attack          : Double
  defense         : Double
  health          : Double
  level           : Integer
}

Layout := {
  startingRoom    : String
  endingRoom      : String
  rooms           : Room []
  player          : Player
  monsters        : Monster []
}
```

Instead of providing a sample JSON file for you to develop with, you should write your own JSON file that follows the format above.

3 Interface (Output)

You will need to print some additional things for each room.

1. The current room's `description` printed verbatim
2. If the room is the `startingRoom` print `Your journey begins here`
3. If the room is the `endingRoom` print `You have reached your final destination`
4. A list of the items in the current room.
5. A list of monsters in the current room.
6. If all of the monsters in the room have been defeated, print a list of the directions that the player can move.

4 Interface (Input)

After you write the message to the console, you should read a line of input from the user.

1. If that line is either the words `quit` or `exit` then you should exit the program.
2. If that line is the word `playerinfo` then you should print the player level, attack, defense and health.
3. If that line is `duel aMonster` and `aMonster` is one of the monsters in the room, then you should duel (see next section) the monster. If `aMonster` is not one of the monsters in the room, then you should print `I can't duel aMonster`.
4. If that line starts with `go aDirection` and `aDirection` is one of the directions they can travel from that room, you should move in that direction and repeat the process. If there are still monsters in the room, you should print `There are still monsters here, I can't move`.

If `aDirection` doesn't match any of the directions possible, you should print `I can't go aDirection`.

5. If that line is the word `list` then you should print a list of the items you are carrying.
6. If that line starts with `take anItem` and `anItem` is one of the items in that room, you should pickup that item. If there are still monsters in the room, you should print `There are still monsters here, I can't take that`. If `anItem` is not one of the items in that room, you should print `I can't take anItem`.
7. If that line starts with `drop anItem` and `anItem` is one of the items that you are carrying, you should drop the item in the current room. If `anItem` is not one of the items in that room, you should print out `I can't drop anItem`.

If the user inputs something that isn't one of the commands above, you should print out `I don't understand` followed by what the user typed in surrounded in single quotes.

Your code should ignore case when handling commands, but when echoing back a user's text should use the original capitalization.

5 Duels

Duels happen between two entities, a player and the monster. When in a duel rather then the normal commands the player has the following commands.

1. `attack`
2. `attack with anItem`
3. `disengage`
4. `status`
5. `list` - as outside of a duel
6. `playerinfo` - as outside of a duel
7. `quit` or `exit` - as outside a duel

How to handle each command that differs from normal is described below.

attack

When the user commands the player to attack, the player should do damage to the opponent. Damage should be deducted from health immediately after an attack and should be calculated as:

$$\text{Damage} = \texttt{Player.attack} - \texttt{Monster.defense}$$

If the player uses `with anItem` then the damage should be as:

$$\text{Damage} = \texttt{Player.attack} + \texttt{anItem.damage} - \texttt{Monster.defense}$$

If the monster is no longer alive (does not have a positive health value) the the player has won the duel otherwise the monster should return damage to the player using the following calculation:

$$Damage = Monster\_attack - Player\_defense$$

If after this the player no longer has a positive health value they have lost and you should print a message saying that they have died and then quit the game.

disegage

When the user commands the player to `disengage`, the player should attempt exit the duel and return to the normal interface. The player will still be in the same room and the health of both the player and monster should still be reduced by any damage that they took in the duel.

status

When the user requests status to be displayed, both the player's health and opponent's health should be displayed. Health should be displayed using ASCII or Unicode characters in a progress bar format. An simplistic example where the player's health is 20% and the monster's health is 50% would be :

```
Player:  ####________________
Monster: ##########__________
```

Winning a Duel

When a player successfully defeats a monster, the monster should be removed from the room and the player should gain experience. Experience gained should be calculated with the following:

$$E = (\frac{Monster\_attack + Monster\_defense}{2} + Monster\_health) * 20$$

When a player earns enough experience, it should level up. Level 1 and level 2 require 25 and 50 experience, respectively, to level up. Experience required for all levels above that should be calculated using the following:

$$Exp(level) = (Exp(level - 1) + Exp(level - 2)) * 1.5$$

When the player levels up, it should increase all attributes with the following scales:

```
Attack = Attack * 1.5
Defense = Defense * 1.5
Health = Health * 1.3
```

The player should also regain all health points back.

Throughout all phases of a duel, you should print information about what is going on. How often and how you do this is up to you, but it should be **sufficient** for the user to understand what is happening.

6 Requirements

The requirements of this assignment are listed below.

1. At run time download and parse the rooms JSON from the web into objects.
2. Correctly display the description / direction options for the current room.
3. Handle the user-directed navigation between rooms.
4. Handle duels between the player and monsters.
5. Handle player level and update attributes appropriately.
6. Allow users to specify an alternate JSON file on the command line to get the room descriptions. This can be either a file as last week or a URL. Gracefully detect and handle the case where they've provided a bad input.

7 Optional Features

This time around, the optional features are up to you. There are lots of ways that you can expand and enhance the game that we have specified so far. Be creative. Once again, the more effort that you put into optional features, the more extra credit it is worth.

Design and Style: As always, use the best design and coding style that you are familiar with. In particular, for this assignment we want you to pay particular attention to your `object decomposition`, that is how you break down the functionality of the assignment into classes and functions (static or non-static) of those classes. Be sure that code related to a particular object is in that class!

Process: You should continue to work on your code so as to implement small bits of functionality, testing them, getting them to work, and committing that piece of work. We expect to see a series of commits in your github with good explanatory commit messages.