Due Nov 18 by 11:59pm **Points** 100 **Available** after Nov 2 at 5pm

Project 4 - Text-Based Adventure Game

Objective

The goal of this assignment is to design and implement a console controller that uses the model we built in the previous project.

According to Wikipedia in their description of an **adventure game (https://en.wikipedia.org/wiki/Adventure_game)** , "the player takes on the role of the protagonist in an interactive story driven by exploration and/or puzzle solving". In a **text-based (https://en.wikipedia.org/wiki/Text-based_game)** game, the way the game is played is through printable text data, usually through a console.

In this part of our project, we be implementing a simple text-based controller that can be used to play the game that we are working developing in our three-part MVC project. In addition, we will be adding enhancements to the dungeon to make the game more interactive and fun.

Part 1 - Refactoring the Dungeon

The first thing that you must do for this project is to refactor the dungeon to add monsters that can can be slayed by the player. This provides a way for the player to both win and lose the game.

Adding Monsters

Otyughs **(https://forgottenrealms.fandom.com/wiki/Otyugh)** are extremely smelly creatures that lead solitary lives in the deep, dark places of the world like our dungeon.

- There is always at least one Otyugh in the dungeon located at the specially designated *end* cave. The actual number is specified on the command line. There is never an Otyugh at the *start*.
- Otyugh only occupy caves and are never found in tunnels. Their caves can also contain treasure or other items.
- They can be detected by their smell. In general, the player can detect two levels of smell:
  - a less pungent smell can be detected when there is a single Otyugh 2 positions from the player's current location
  - detecting a more pungent smell either means that there is a single Otyugh 1 position from the player's current location or that there are multiple Otyughs within 2 positions from the player's current location
- They are adapted to eat whatever organic material that they can find, but love it when fresh meat happens into the cave in which they dwell. This means that a player entering a cave with an Otyugh that has not been slayed will be killed and eaten (see next section on how to slay an Otyugh).

Slaying Monsters

To give our player the ability to slay the Otyugh, they will automatically be equipped with a bow that uses crooked arrows

- Player starts with 3 crooked arrows but can find additional arrows in the dungeon with the same frequency as treasure. Arrows and treasure can be, but are not always, found together. Furthermore, arrows can be found in both caves and tunnels.
- A player that has arrows, can attempt to slay an Otyugh by specifying a direction and distance in which to shoot their crooked arrow. Distance is defined as the number of caves (but not tunnels) that an arrow travels. Arrows travel freely down tunnels (even crooked ones) but only travel in a straight line through a cave. For example,
  - a tunnel that has exits to the west and south can have an arrow enter the tunnel from the west and exit the tunnel to the south, or vice-versa (this is the reason the arrow is called a *crooked arrow*)
  - a cave that has exits to the east, south, and west will allow an arrow to enter from the east and exit to the west, or vice-versa; but an arrow that enters from the south would be stopped since there is no exit to the north
- Distances must be exact. For example, if you shoot an arrow a distance of 3 to the east and the Otyugh is at a distance of 2, you miss the Otyugh.
- It takes 2 hits to kill an Otyugh. Players has a 50% chance of escaping if the Otyugh if they enter a cave of an injured Otyugh that has been hit by a single crooked arrow.

What to do

Modify your design of your dungeon model to include monsters and the ability of the player to slay the monsters so that it meets the requirements described above. Use interfaces/classes that capture the similarities and accurately represents the relevant data following good OO principles.

Part 2 - Design a Controller

The second think to do is to design a controller that uses the refactored dungeon model to implement a text-based adventure game. You can call it anything you like. For example, here's a excerpt from my implementation (this is only one possible way for the controller to work, there are many others that would be acceptable):

```
You are in a cave
Doors lead to the N

Move, Pickup, or Shoot (M-P-S)? M
Where to? N

You are in a cave
Doors lead to the N, E, S

Move, Pickup, or Shoot (M-P-S)? M
Where to? N

You are in a cave
You find 2 rubies here
Doors lead to the N, E, S

Move, Pickup, or Shoot (M-P-S)? P
What? ruby
You pick up a ruby

You are in a cave
You find 1 ruby here
Doors lead to the N, E, S

Move, Pickup, or Shoot (M-P-S)? M
Where to? N

You are in a tunnel
that continues to the E, S

Move, Pickup, or Shoot (M-P-S)? M
Where to? E

You are in a tunnel
that continues to the S, W

Move, Pickup, or Shoot (M-P-S)? M
Where to? W

You smell something terrible nearby
You are in a cave
You find 1 arrow here
Doors lead to the N, E, S, W

Move, Pickup, or Shoot (M-P-S)? P
What? arrow
You pick up an arrow

You smell something terrible nearby
You are in a cave
Doors lead to the N, E, S, W
```

```
Move, Pickup, or Shoot (M-P-S)? S
No. of caves (1-5)? 1
Where to? N
You shoot an arrow into the darkness

You smell something terrible nearby
You are in a cave
Doors lead to the N, E, S, W

Move, Pickup, or Shoot (M-P-S)? S
No. of caves (1-5)? 1
Where to? E
You shoot an arrow into the darkness

You smell something terrible nearby
You are in a cave
Doors lead to the N, E, S, W

Move, Pickup, or Shoot (M-P-S)? S
No. of caves (1-5)? 1
Where to? S
You shoot an arrow into the darkness

You smell something terrible nearby
You are in a cave
Doors lead to the N, E, S, W

Move, Pickup, or Shoot (M-P-S)? S
No. of caves (1-5)? 2
Where to? N
You hear a great howl in the distance
You are out of arrows, explore to find more

You smell something terrible nearby
You are in a cave
Doors lead to the N, E, S, W

Move, Pickup, or Shoot (M-P-S)? M
Where to? E

You smell something terrible nearby
You are in a cave
Doors lead to the E, S, W

Move, Pickup, or Shoot (M-P-S)? M
Where to? E

Chomp, chomp, chomp, you are eaten by an Otyugh!
Better luck next time
```

Winning

- The player wins by reaching the *end* location.
- The player loses by being eaten by an Otyugh.

What to do

The *controller* should implement a text-based adventure game as described above. Your controller should be able to:

- Use the input from the user to
  - navigate the player through the dungeon.
  - pick up treasure and/or arrows if they are found in the same location as the player
  - shoot an arrow in a given direction
- Output clues about the nearby caves and other relevant aspects of the current game state without dumping the dungeon map to the screen.

To launch your controller, include a *driver* that handles the **command-line arguments (https://www.baeldung.com/java-command-line-arguments)** needed to specify the properties of our game (size of the dungeon, its interconnectivity, whether it is wrapping or not, the percentage of caves that have treasure) as well as the difficulty (the number of Otyugh). The driver should further specify that the input will be from `System.in` and the output will be to `System.out`.

What to prepare

A single PDF containing a UML class diagram that captures all aspects of your design including a testing plan. You should submit both a preliminary design as well as a final design with your implementation submission. You do not need to submit your preliminary design, though you may want to gather feedback on it from instructors during office hours.

Part 3 - Implementation

Implement the interfaces/classes that you have specified in Parts 1 and 2. Be sure that you use abstraction whenever applicable and minimize code repetition.

Hint: Now that we have an interactive game, you can use an actual random number in the executable JAR file that you are providing. Though you will want to continue to use the deterministic behavior for your test cases.

Documentation

We expect your code to be well-commented using well-formed English sentences. The expectations are:

- Each interface and class contains a comment above it explaining specifically what it represents. This should be in plain language, understandable by anybody wishing to use it. Comment above a class should be specific: it should not merely state that it is an implementation of a particular interface.
- Each public method should have information about what this method accomplishes (purpose), the nature and explanation of any arguments, return values and exceptions thrown by it and whether it changes the calling object in any way (contract).

- If a class implements a method declared in an interface that it implements, **and** the comments in the interface describe this implementation completely and accurately, there is no need to replicate that documentation in the class.
- All comments should be in `Javadoc`-style.

Create a JAR file of your program

In order to make your application easier to run, you are required to create and submit an executable `JAR` file that can be executed on the command-line using Java 11:

- Directions for doing this in Eclipse can be found at **this link (https://www.codejava.net/ides/eclipse/how-to-create-jar-file-in-eclipse)** .
- Directions for doing this in IntelliJ can be found at **this link (https://www.jetbrains.com/help/idea/packaging-a-module-into-a-jar-file.html)** .

What to submit

Log on to the **Handins submission server (https://handins.ccs.neu.edu/)** and upload a ZIP file of your assignment. Your ZIP file should contain three folders: src/, test/ and res/ (even if empty).

- All your code should be in src/.
- All your tests should be test/.
- Your design documents (original and revised) should be in res/.
- Submit a correct `JAR` file in the res/ folder. We should be able to run your program from this jar file using Java 11 from a command line using the command `java -jar NameOfYourFile.jar command_line_arguments`
- In the res/ folder, also submit at least two example runs of your driver program in a ***simple text file*** that can be used to verify that your model meets all of the above specifications. Your runs should include:
  - One run that shows the player navigating through the dungeon
  - One run that shows the player picking up treasure
  - One run that shows the player picking up arrows
  - One run that shows the player being eaten by a Otyugh
  - One run that shows the player killing a Otyugh
  - One run that shows the player winning the game by reaching the *end*
- Submit a README.md file that documents your program **following these guidelines**.
- Your zip file should also be free of IDE configuration files, compiled .class files, and other non-essential files that are automatically generated. The META-INF/MAINFEST.MF file is required for your JAR file to be executable and should be included.

Criteria for grading

Your projects will be assessed in three different ways:

1. Completeness, correctness, design, and testing will be assessed through the completion of a *self-evaluation* similar to the ones you have been doing for lab assignments. In the self-evaluation, you will be asked to answer a series of questions about your code, provide pointers (tags) into your code, and provide explanations of your answers and/or implementation. The pointers into your code will allow us to confirm your answer since they will show us where to look in your implementation. Self-evaluations are worth ~50% of the project grade.
2. Style and whether you are following good programming principles will be assessed on the Handins server via your submission. Style is assessed automatically by Handins and good programming principles will be assessed manually. To see details of what will be manually assessed, refer to the **Manual Grading Checklist**. The submission and the manual grading of said submission is worth ~20% of the project grade.
3. Finally, you are asked to defend your design and/or implementation choices in a meeting with your instructors for the last ~30% of the project grade including submitting a reflection of your meeting experience.