# Assignment 4: Alice in Wonderland Personal notes.

Jackson Wiebe

3519635

The largest challenge I had was where to place my main loop. That would be the part of my code where the decision tree would be held and proceed with the user actions. I want to have this main loop be in one place, however each room was going to have different actions and results so it was tricky where to put this. I eventually settled on having a main “room” class (Location.java) that could have overridable methods in the sub class that I would create for each room.

The Location.java class would act as a main loop as I decided. It would do most of the heavy lifting for this program and call the sub classes and actions as needed. Many child classes were build off this class where I used super methods to “add” objects to the room and assign identity to it. With this framework I could keep most of the generic functions within the parent class and override any small functions. 2 special cases for this were the start and end rooms. Since I wanted the user to make a choice right away in the first room the Yes and No methods were overridden to do those special actions. I also added some graphics in the first RiverBank class that seemed to add character to the game and a sense of falling down the rabbit hole. These rooms are held in a master Game thread that would call the enter() method of each room until it returned a new room to enter.

Action class was tricky for me to create. As I understand it in the assignment. We were to get these action words from a Text file however that did not make sense to me. These action words were going to be essentially static variables, so I created an ENUM *ActionType.java* with all the options and built from that. In this manner my actions were still statically defined and the fixed set of actions could always be executed. After deciding on the action words I built am Action.Java class that would handle the specific action desired. These classes would be treated as objects that contain the action and any optional arguments given by the user. The constructor for this class was to be given a string gathered with the use of the TEXTIO library as provided. From there a large If ELSE loop decodes the action. The IF Else loop seems rather ugly to me however given all the string comparison required I failed to see another way to simplify this code. After decoding the action object can be passed around and some getters and setters used to determine the action.

Item class was pretty straight forward for me, I followed the same formula as before. I created an Enum containing all my possible items I wanted to include in the game. I then created a master class for Item.java which would contain each items identity and a set of generic getters and setters for the name, description, action, and action text. In the constructor for this master class I used static methods created in the Control class to get the name, description and generic action from that items respective text file. So for generic child classes like Mallet, I would only need to call a super method that would assign its identity. However for items like the ItemWatch.java I wanted to include some extra ability and such that specific item overrides the Action class and includes some functionality that also prints out the day of the month as would be respective of the rabbits watch in the game.

Item class follows the same methodology as before, creating an enum of all desire items, creating a master class and sub classes of each item that could override some of the methods as desired. In addition to creating these classes we also needed an inventory system so Inventory.java was created to handle this. The inventory classes would contain a linked list of items that could be added, subtracted, and searched for specific items. I would need to apply this inventory class to the main character Alice and to the storage room for her to put items. Since the Garden room was chosen as my safe room I overrode many of the functions of that room in respect of the inventory for that room. Since takeItem() was override for this class, the Garden room contains no items of its own for the user to take outside of what would be stored in this chest system. One Issue I would like more time to look at is the instancing of each object. Since multiple objects of the same type are not given unique identification it could get tricky in the future if the game was expanded to include multiple uses for each item.

About half the items also include extra actions that override the default. For example the Mushroom item will take the to a minigame where you must get allice to the correct size to proceed. The Watch class includes special code that will get the day of the month from the Java VM system and print that to the terminal. Instead of a fixed number it more dynamic integration. Some items like drinking the Tea will end the game and a special action was written to immediately exit the game and print a exit code.

The character system was the hardest type for me. I had intended to create a full conversation system for each none playable character however given the scope of this challenge I instead decided to give each NPC a single query which the user would need to answer to solve their unique puzzle. This conversation system works by creating conversation nodes which gather the appropriate text information and wait for a specific String response to give approval and continue. I also did not want to make the game too hard so if multiple attempts were done the conversation node would spit out a hint after 3 tries. Thankfully this game did not need to include any combat system however I did consider adding these unique traits of health, life and death status and such. Since the game characters were more generic than the items or room I did not need to create unique subclasses for each character and instead opted to create the characters from the master Character.java class. Given their name the class would load the appropriate text file, display name, and conversation node.

I knew I wanted to include some graphics in my game so I use an online generator to create the title screen ASCII art and I wrote a custom loop in RoomRiverBank.tumblingGraphics() that would create a series of graphics to simulate following down a hole. These graphics make use of the java Thread.sleep() functions which is not the best way as it locks up the thread however for this use case I felt it was the easiest solution. Having a multithreaded class where the graphics can be printed on their own would be better or having a non-blocking code loop could also be a solution. I also wanted to clear the screen between each room for clarity however there is no simple java method so I wrote a function in the Control class that would print 50 empty lines to simulate this functionality.

The class of Alice.java was specially created. As the main character she was going to have some unique traits and I did not want to build this off a parent class to keep maximum flexibility. That class includes an inventory object that can be accessed at any time and some special methods to keep track of Alice’s size as that would be needed to fit through certain doorways.

All in I found this project quite daunting and a good challenge of my skills. If I had more time I would like to flesh out the conversation system better and add more minigames for the different rooms.