Assignment 4 Notebook

Jenna McDonnell

In this assignment, one of the biggest struggles for me was finding the best way to store and output text. I initially was typing out all of the text in the actual code, but I realized this made my code much longer than I wanted it to be. So, I decided to make a text file for each of the different locations and separate the paragraphs by "/n". This way, I would later be able to split the text into paragraphs, putting each paragraph into a spot in an array, and I could output whichever paragraph was needed for the situation.

I thought this text file method would make things a lot easier and while it did make my code look smaller, getting the program to read the file the way I intended proved to be challenging. My first test to see if this text reading worked, the program only output the last line of text in the file. After fixing some problems with the for and while loops I had set up for my BufferedReader, I eventually had the program printing out all of the text, but it wasn't being properly put into the array, so I still wasn't able to separate it the way I wanted to.

I was finally able to separate the text properly by using a StringBuilder to create one long String of text from the text file and using the split() method to separate the paragraphs. Now I could print the paragraphs separately and at the appropriate times, but the lines weren't separated the way I want. So, I had to figure out how to include line breaks where I had line breaks in the text file. I was able to do this by appending "\n" to the StringBuilder after each line I appended [1]. The text was finally printing out in the exact format I desired and at the appropriate times.

My next problem was that the program wasn't accepting all of the answers I designed it to accept. To keep things simple, I gave the user a few options to choose from after each paragraph of text. When I did my research on text-based adventure games, I found the ambiguity of acceptable answers made the game move slower. It's nice to have lots of options but at the same time, if you are stuck in the same spot trying to figure out what you are supposed to do for a long time, it can be difficult to progress and stay interested. For that reason, I designed my game like a choose your own adventure, where there are still multiple paths to take in the story, but each step of the way is narrowed down to a few options.

With the assignment requiring that I accept different kinds of answers, I wanted to take three different acceptable forms of each option. For example, if the option was: a) climb down tree, the program should accept "a", "climb down tree", or "down" as answers. I had created a method testForMultipleAnswers() that was made to take the answer, the intended output, and those three forms of an answer as the parameters. It would then test to see if the answer was one of these forms and convert it to the exact words (in this case, "climb down tree") and it would return the answer to continue in the story. When I tested the program however, it wasn't accepting the answers I wanted.

I realized that my problem was the way I was comparing the answer to the accepted answers. I had forgotten that when comparing two Strings, you can't use == as I believe that will compare the value of the characters in the String. To test if two Strings contain the same characters, I needed to use the String.equals() method [2]. When I substituted this for ==, my program accepted multiple answers the way I intended it to.

I thought the testForMultipleAnswers() method was a good idea because I wouldn't have to write out the whole code for each answer, so it saved a lot of space. However, there were a few issues with this. For one thing, this method required five parameters as input when it was called. I know it's not a good idea to have more than one or two parameters for a method because this makes it confusing to use the method, but I found that the space it was saving made it worth the extra parameters, so I left the method in. The other issue I had was that in my switch statement, the default method was called if the answer provided was not an accepted answer. The statement asked the user to answer with one of the options listed. However, when it takes in the new answer, it doesn't know which acceptable answers to test from because there is no way of telling which part of the story the user just came from. Therefore, if the first answer was not accepted and the user gets to the default statement, the answer they enter must be word for word with the options provided.

The String.equals() method also fixed another problem of mine: I couldn't get my while loop to end the way I wanted it to. I had a while loop around the switch statement for each location and the last answer for the user to end the first location was "adventure". For this reason, I set the condition of the loop to while (answer != "adventure") but this wasn't ending my loop when I typed "adventure". I realized I needed to use String.equals("adventure") and because the loop was meant to end when this condition became true, I needed to say while(answer.equals("adventure") == false) to keep the loop running until the user entered "adventure".

The next issue I had was less of a technical issue and more organizational. My methods in the Locations class were still way too long, so I wanted to break it up into smaller methods. I decided that the Control class would be a good place to put those methods, so I made Locations extend Control and started moving code over to the Control class. I was confused at first because I thought extending the Control class would allow me to use any methods from that class in the Locations class. However, it wasn't letting me do this at compile time because Command Prompt was telling me that Locations couldn't find those methods. After playing around with creating a Control object in the Locations class to be able to use those methods, I realized I had accidentally tried to extend Controls instead of Control. Once I fixed this, I didn't need to create a Control object to use those methods in the Locations class.

Another issue I came across was figuring out how to load a new text file after the first one had been loaded and I had finished using it. I realized that if I was going to use the same variables for each text file I read, I needed to clear these variables before loading the next file. I cleared out the String variable holding the previous file, the array holding the lines of the previous file, and the bufferedReader for the previous file. Before, the program didn't crash but after the first scene, it would just display the same scene over again, instead of displaying the next scene in the game.

I ended up taking a break from this project for three months so I could focus on my fall classes and I now feel I have a lot more to reflect on. In that time, I learned C++, and learning a different object-oriented language allowed me to better understand object-oriented programming. When I started reading through the code that I had already written, I realized that although I had broken my code up into different classes, I hadn't really made the program any more efficient or clean. At best, the program was slightly more organized. I think the point I was missing before is that my objects and functions should be somewhat generic so they can be used for multiple scenarios. Instead, I had made the functions very specific to the situation I was using them for at the time and ended up making a lot more functions than needed. If I were to do it over again, I would make fewer functions and make them simpler so there are easier to use in a broad range of situations. I am now beginning to understand that this is one of the main reasons to use object-oriented programming, to reuse code and make the code simpler to read and to use.

Another thing I learned from C++ is why multiple inheritance can be useful. Being able to simply # include a bunch of classes at the beginning of the program requiring the functions from the classes, makes it a lot easier to keep track of inheritance in my opinion. In Java, I have mainly used extensions for inheritance and extending does not allow for multiple inheritance. However, I know that using interfaces in Java is a way to get around the multiple inheritance issue, and while I didn't understand where or why I would use this when I read it, after doing some object-oriented programming in C++, I definitely see a use for it. I now see it as having a basic outline for a function and being able to adjust that function to whatever situation you are using it in. This is again how object-oriented programming allows you to reuse code and make it simpler.

I have also found that in making my functions and my text output very specific to each situation of the game, it really limited the game's flexibility because for every specific situation I wrote, I had to account for many possible answers the user could give, and instead of making an easy function to check for certain words in the input and checking them against acceptable inputs, I ended up making a function with five parameters that still didn't accept as many answers as I would have liked it to. I know it is best to have few parameters for each function but at the time I wrote this program, I didn’t know how to write a better function to check for multiple answers. If I had more time in between my university’s semesters to make the program better, I would probably make a function that checks each word of the input, and if the word only appears in one of the accepted answers, I would assume that this is the answer the user was trying to input. I would likely have to use a stringTokenizer for this, to separate the words in the input given by the user, as I don’t believe there is an easy way to do this in Java like there is in C++ where you can just use cin for each word.

One more thing I would’ve done better if I were to redo this project would be to name my variables more carefully. I started off the project with specific names that were unique, but as I got further into it, I found that I was using the same variable names in different classes and this made things slightly more confusing and disorganized. Looking back on it, I would’ve made the variables more specific or just passed the variables into functions in each class so that I didn’t need to make the same variable in each class.

The research I did for this project was focused on “Colossal Cave Adventure”, since that is what the assignment description used as an example [3]. The main points I gathered from studying this game was that the user had multiple options and there were multiple locations and items to interact with. The creators of that game did a better job of giving multiple choices than I did, as they didn’t even give options for the user to choose from. The game just asks the user a question and the user can answer however they want. While some answers will get a more generic response asking for the user to give a different response, there was more than one accepted answer at each stage of the game.

At the time I originally started writing my game, I didn't have much experience in programming, so I thought the easiest way to accept multiple answers would be to give the user choices to choose from and just accept different forms of their answer. After taking a break to do a full-time fall semester at a University, I've learned it may be easier to check each word of the user's answer for similar words between the answer and the accepted answers. Then I could just see which accepted answer lines up more with the user's answer and I wouldn't have to guess different ways the user might enter a certain answer.

As for the multiple locations, I probably didn't design that the best way either. For locations, you can only progress forward in my game. Once you leave a room, you can't go back. This means that if there was an item or something to pick up in one of the locations, like the Cheshire Cat's name tag, you can't go back to get it once you've reached the Cheshire Cat. I still made the game so that you can't progress forward without necessary items like Alice's weapon or the key to open the door in the hallway, but it might have been nice to be able to go back to find extra hearts or something before fighting the Queen.

The items themselves weren't so much of a struggle for me as maintaining inventory was. I spent a lot of time fixing my open bag functions and they still aren't ideal, but they function well enough for the user to finish the game successfully. At first, I was just using a set size array for the backpack and incrementing a count to keep track of the number of items in the bag. However, after learning C++ and using vectors, I realized that ArrayList is similar to vectors and would actually be better for the job. Then I didn't need the count for number of items because I could just use the size function for ArrayList. ArrayList had a lot of already created functions and it was just easy to use that than using an array and creating my own functions.

One of the design decisions I made for my game was to change the way I organized my classes. I kept the game, locations, inventory, and control, but I ended up making classes for enemies and weapons instead of some of the other classes listed in requirements. The enemy and weapons classes function as actions, characters and items as they hold fight actions, character variations, information for enemy characters, and weapon design. It just made most sense for me at the time to organize it like this.

If I were to do the project again, I would probably make the separate actions and items classes and make methods in those classes that are easy to use at any point in the game. One feature I probably would've included in the items class would be Boolean variables to determine whether the user had already interacted with certain items already. Then I wouldn't have to worry about the same item being added to inventory multiple times or being used more than once where it wouldn't make sense to do so. For the actions class, I would probably make more generic actions like "take", "fight", and "move" which would probably allow the user more freedom in choosing their answers and make it easier for me to accept multiple answers.

For testing this game, a lot of the testing I did as I was making it. Something I learned from the classes in my fall semester is that if you constantly test new methods and additions to the program throughout the process, it makes it a lot easier to fix errors and progress through the programming process. Before, I would just type out the majority of the program and test it when I was done or close to done writing it. Then I would have a lot of errors to fix and it would be overwhelming and I'd be stuck fixing the program for a longer amount of time than the time it took me to actually write the program. Coming back to the game after the fall semester, I used this method of testing pretty much every time I added a new method and this really helped me with the testing process. There was still a lot of debugging to do from when I had written the first part of the program in the summer, and I didn't get all of those issues fixed but I made sure it ran smoothly enough to get through the whole game.

For the testing procedure I wrote in the documentation of the code, I just wrote a pretty basic set of answers for the user to input that should ensure they win the game. The only way it shouldn't work is if the player runs out of hearts before finishing the game, which is somewhat luck based but usually the player should make it through the game easily if they follow these instructions. I didn't want to write anymore than I already did in the testing plan space because it was already cluttered enough, so that's why I only did that one case. Then for the bad cases, I decided to show an example of a variation of an answer that should still be accepted, as well as an answer that is not accepted which demonstrates the default answer. All parts of the testing procedure worked as expected but there were still some bugs to be worked out in different parts of the game that were not necessary in completing the game.

References

[1] lllep, *How to append a newline to StringBuilder,* January 2013. Accessed on January 16, 2021. [Online]. Available: https://stackoverflow.com/questions/14534767/how-to-append-a-newline-to-stringbuilder

[2] Code_r, *Compare two Strings in Java,* March 2020. Accessed on January 16, 2021. [Online]. Available: https://www.geeksforgeeks.org/compare-two-strings-in-java/#:~:text=Using%20String.,same%20then%20it%20returns%20true.

[3] W. Crowther, *Colossal Cave Adventure* [Text Adventure Game]. 1976.