Zork Final Project

Developers: Mayson Koliba, Sean Gallagher, Raeshawn Bart 12/9/18

Project Description

For our final project we decided to program the game of Zork. Our goal was to take the ideas and concepts from Zork and modify it to include our own unique features. Throughout our game, the player (hero) will come across many different situations and perils as they try the complete their quest. They will be able to collect numerous different items, potions, and spells to help them along their way. The player will have to manage their items accordingly in order to help them progress throughout the story. The game is all about strategy, and thinking ahead. If the players health drops to zero, they will lose the game, if they manage to find and beat the final boss they will win the game.

Original Design

There was a lot of planned functionality that never made it in to our final game. We realized while programming that our initial goals may have been too ambitious for the scope of this project, but we were still able to implement a complex and fun game.

Before it was cleaned up, there were many more prototypes of public member functions in the creature class, such as for melee attack accuracy, dodge chances, spell resistances, as well as for four main stats that would be used to scale other stats of each creature. Mana would scale with Intelligence, melee damage would scale with strength, dodge with dexterity, etc. There were also plans for there to be equipment the player could equip which would modify these scores, such as an enchanted sword that increases their strength by a certain amount.

The exploration aspect of the game was also going to be further fleshed out. There were supposed to be 3 dungeons which the player would be able to look at a map of while they were progressing in them. There were going to be story items in two dungeons which allowed for the opening of the third, final dungeon. Similarly, there were plans for a town to be implemented which the player could buy and sell items, talk to townspeople who would give the player quests which would give rewards, the ability to save their game, however all of these functions were only going to be added if we were able to get to the other functionality.

There were also going to be several different types of monsters that could be fought, each with different stats, weapons, spells, and tables of items that they could drop. The type of enemy that spawned in an encounter would also have been related to the environment that the player is in. For example, a bear would spawn in a cave, not in a dungeon.

Program Features List

1. Generating random items and potions.
2. Allowing user to access found items during combat.
3. Allowing user to learn and cast spells to help them during combat.
4. Generating different locations and allowing the user to check inside them.
5. Using the rand() function to vary attack damage and the rate at which items and enemies are encountered.
6. Allowing user to make decisions regarding the path they want to take and items they want to use.
7. Incorporated a storyline with an incentive for the player to reach.
8. Dynamic text and story elements to help the user feel immersed.
9. Fighting bosses and gaining new abilities after each encounter.

Testing Methodologies

Some of the testing methodologies that we implemented to make sure our game was running correctly was having multiple people try to play our game and let us know if they ran into any hiccups. While programming, we also ran into many issues ourself and were able to compensate for this to make sure the user would not make the same mistakes. For instance, if the user types “ewplre” instead of “explore” the game will tell them to “Enter a valid command”. This helps the code to run smoothly, and ensures logical commands are entered by the user.

Usage Guidelines

Most of how the program works is explained within the program itself. The player is either in exploration mode, or combat mode. Most commands that we tested should output “Enter a valid command” in exploration mode, like 0, extremely high and low numbers, as well as integers and symbols. When selecting which inventory item, potion, or spell to use, make sure to only enter the number of the item you want to use (integer). If anything else is entered (such as a word or a char) the program may enter an infinite loop. The user is intended to only enter an integer anyways, but just in case you accidently press the wrong key. After using a potion or inventory item, they are deleted from your repertoire. The user gains the ability to cast more spells as the game progresses, but they must have enough “mana” to do so. Casting a spell will decreases the players mana by a certain amount. The user can find potions to refill their “mana” and allow them to cast spells again. All items are intended to be used during combat.

Lessons Learned

Mayson Koliba

I learned a lot of coding lessons and tricks while working on this project. One of the big lessons that I learned is that it is best to make most, if not all, of your class variables private instead of public. At first we had it set up where most of our variables were public, but this ended up leading to many issues along the way. For instance, when the variables were public it was hard to update them inside functions and have the variable transfer over its value throughout the game. We would make changes to the variable in one function but when we accessed the variable from another function it would change its value. Once we changed the variables to private and used accessor functions to get the values where we needed to, this allowed us to access and modify the values more reliably.

Another lesson that I learned is that using "cin >>" and "\n" with the "getline()" function can lead to problems. When using "cin >>" statements and "\n" this would leave blank spaces in the buffer and the "getline()" function would pick these up. This lead to a whole bunch of errors while coding and took some time to figure out what was going wrong. In order to prevent the "getline()" function from picking up these blank values I had to use "cin.ignore()" after "cin >>" statements, and use "endl" instead of "\n".

Another lesson that I learned was that once you figure out how to make one part of the code work, this often helps you with many other parts of the project as well. For instance, was I figured out how to add,store,use, and delete inventory items, the same process could then be applied to the potions in the game. Once I figured out how to generate one random location, I could apply this same method for generating other locations as well.

Raeshawn Bart

While working on the final project I learned that there is a wide number of ways to program and when programming with other every person has their own way of coding and when you put your minds together you tend to get a solution that not just one person by themselves would of came to the conclusion. I also learned that when it comes to group programming that when making functions and classes it tends to make things better by using correct language or leave notes so that your other group members know what you had added and where the current progress is and when they need to add something into the program it tends to makes things easier when you can trace where and how the codes was added so when you go to add your own things it tends to make adding so much easier. I also learned when it comes to making classes and functions for game programs and other similar programs that it is better to add sever different classes and function based on the operation that will be taking place in it for example we made the inventory a separate section this way when it came to editing the code I could see where the items was sent and made it easier for me to edit the code same goes for the monsters. When it came to the weapons you can see how the damage dice was and it was easier to change the damage of items. I also found the sending items found easier being that they was just added to the inventory and when it came to using a vector instead of an array it made changing the size of the vector code so much more helpful so all we had to do when adding an item to the inventory was use push back this item if found and then when the person wants to display their inventory we can use find or search inventory for item or display all items in the inventory. I also learned when it came to a fighting system it tends to make things better when the player themselves can choose what to do next and with the different classes this change was easy to put into the program giving the player so much more to choose from and also when it comes to team programing when everyone is working on it tends to help everyone out when you have the code formatted nicely so that everyone can follow the formatting and be able to understand it this made it easier to change and edit without losing place. So in this situation when it came to team programing I learned that it is nice to have other to program with you because of styles of each other can help overall solve the main problem your focusing on and that when it came to classes I learned that it's best to make one for one intended purpose and then if want to add another function just do so and not add it to the same class of a different function of the game. So this is what I overall learned from this project and this will help my future structure coding style and collaboration when it comes to programming

Sean Gallagher

Working with a group on a project taught me a lot about what is required from you when working on code which is edited by multiple people. One major thing that I learned was the importance of adding comments. Before now, I always felt like adding comments wasn’t very helpful most of the time because it’s usually just me working on a project and I know what everything does. When you’re working closely with 2 other people though, that changes a lot. Thankfully, the code was relatively simple to understand for the genre of game that we chose, but there are still some cases where a comment is necessary to understand what is happening. I also feel like I became a lot more familiar with implementing and using classes in a large project. Almost the entirety of our code is written within function definitions for the public member functions of our creature class.

I also learned that it is not always possible to implement every little detail that you want to into a project with other people. There were a lot of different features that I had originally planned to implement that never made it in, such as a town which you can go to to buy and sell items. There was a lot of functionality that you could have told was going to be implemented by the looking at the creature class definition before Mayson cleaned it up.

Another concept that we didn’t use in most of the other labs was interdependencies between source and header files. In our project, we had a header file which only contained the prototypes for the functions, which were then defined in the gameLogic source file. In smaller projects like our labs, it doesn’t make as much sense to to this, but in a large project like this it is vital that there are dependencies so we don’t end up with a single large block of code.