Braeden Kuether
August 16, 2018

Final Project Reflection Document

Design -

This project was by far the most difficult to design. The fact that the board had to be made out of Space pointers made creating a board significantly more difficult than in other projects, where you would simply create a 2D array. In this case, I had to come up with a system for linking together each individual space object to the adjacent objects so that the player could move from one space to another. On top of that, the Space class is an abstract class which I don't know very much about and have to do more research on to understand the proper way to implement it. The one thing I'm very much looking forward to is the freedom with the assignment. I've decided to make my game Skyrim themed, where the player must defeat Alduin to win. Along the way, the player and pick up items, open chests for items, or fight monsters who drop items. I plan on incorporating an Item struct, which holds the name of the item as well as the strength of the item, that way the player can gain strength by picking up items, and I can add some flavor to the game by incorporating real Skyrim items. On top of that, I plan on implementing a difficulty setting that is called when the chest items are constructed, that way some chests are much more difficult to open then others, just like in skyrim. Additionally, I do a similar thing with monsters where you never know when you step on a monster space if you'll get an easy opponent, like a bandit, or a hard one, like a giant. Lastly, all of the battles between player and monster will be determined by using the rand function between 1 and the player's strength, and same with the monster. That way, there is some variance in how likely you are to beat an enemy, and if you get bad luck with a giant or dragon early on you could still high-roll to beat them.

Specifics -

To link all of the spaces on the board, I will have to start by creating a white space that the player will start on. Then, using the *this* keyword, I will create new space objects as one of the derived classes (such as an item) and use a set_right or set_top function to linked the pieces (set_bottom and set_left as well). Additionally, I'm going to have to link every individual piece to the next, so I will begin by linking the in a snake pattern (horizontally), and then do the opposite vertically. In terms of how I will randomly set each space object to a specific item or monster, in the class constructor I will call a function that randomly picks from a longer list of possibilities, that way you essentially never play the same game twice. Lastly, to keep track of the player's items, I will use a vector, and pass that vector into functions where I alter the player's items.

Test Tables -

Function	Test Value	Expected Outcome	Observed Outcome
input_validation()	a	This is expected to reprompt because it is not an integer.	Function reprompts.
input_validation()	1	This should return true.	Returns true, allows user to pick up item.
input _validation()	1.0	This is expected to reprompt because the check function will find the period, which is not a 0-9 integer.	Function reprompts.
input_validation	20	This should reprompt the user.	Function reprompts.
add_inventory(Item)	Not full inventory	This should simply add the item to the vector.	Adds item to the vector.
add_inventory(Item)	Strength = -1	The function will simply return, because items designated with a -1 strength are not meant to be picked up by design.	Does not pick up item, returns.
add_inventory(Item)	Full inventory	Should ask user to drop an item if they wish to pick up the new one	Tells user inventory is full and asks that they replace an Item in the inventory.
fight(player_strength)	Player strength = 110 Monster = 35	This should return a winner, which will almost certainly be the player.	Player wins, monster dropped loot.
fight(player_strength)	Player strength = 80 Monster = 100	This should favor the monster, given how the randomness is minimized by the use of an average of 2 rand functions.	Player still won, monster dropped loot.
open_chest(vector<Item>)	Inventory not full	This should attempt to open the chest, then ask the user if they want the item.	Chest opens and asks user to add item.
open_chest(vector<Item>)	Inventory full	This should attempt to open the chest, then ask the user if they want the item and which item to drop.	Chest opens and asks user to add item, then asks which to drop.
open_chest(vector<Item>)	Inventory not full	This should attempt to open	The chest does not open and

		the chest, then ask the user if they want the item.	the turn is over.
print_inventory()	Inventory full	Should list all six items with corresponding number.	Lists all items.
print_inventory()	Inventory empty	Should print nothing.	Prints one item with random strength and weird symbols. Upon review, the function works correctly after fixing an issue in my declaration.

Reflection -

This assignment turned out to be very long and hard to do well. The way that the pointer system worked made the board and set up very difficult. That being said, it was very fun to get a chance to work on a project where I decided the theme and how to build it and had a lot of liberty. I found that in the end, my game was actually really fun to play and had a lot of dimensions. You could strategize by farming up item and chest spaces before every facing a monster, that way you had a low chance of losing, or you could go for the better loot early on by fighting monsters and avoiding item spaces. Additionally, I was able to give my game a lot of flavor in all the different items I added, so the player can associate strengths and weaknesses with items if they've played Skyrim in real life (not to mention the fact that it made me want to play Skyrim). All in all, I ran into a lot of problems! In the compiling of my project, for some reason the compiler kept giving me weird errors that didn't make sense. It wasn't until I realized 45 minutes later that I had one function in my .hpp that I didn't define in my .cpp. Other smaller errors like typos, running into some functions that didn't behave properly were common. For instance, I ran into a problem where if the player didn't want to pick up an item it would still return a random item of random strength. I had to instead create a temporary item object with negative one strength and return that, than filter out items with negative one strength in my calculate_strength function. Additionally, I ran into a ton of errors while trying to build my board, but I was eventually able to get it done. All in all, the assignment was a lot of work and a lot of fun, and was a great way to leave off given the complexity and freedom.