Lee Rice  
Final Project Reflection  
CS162  
11/18/2017

**Design Description**

My project will be a Canyon Escape text adventure. The player will traverse a canyon collecting items that will eventually allow them to climb out of the canyon. The limiting factor will be the player’s hydration. The player starts with a finite amount of time before they pass out from dehydration. They’ll pick up water bottles throughout the game. I’m considering adding a random element where some sources of water are good, and some sources of water are bad.

To implement the different areas, I’ll create the derived spaces using constructors in the menu class and link them together as they’re created. I’m thinking of simply using a vector for the player’s objects and implementing a separate container that holds an amount of water the player can fill up at different water sources. At this point, I’m not sure if the water container is really needed since I don’t want the game to be excessively long.

I am a fan of the Choose Your Own Adventure style books and will be modeling the game after them. The difficulty is how change the dialog in a Space that the player has already been in. I plan on doing this by using an array of integers that vary based on player’s actions.

Input validation will make use of the same input validation functions I’ve used throughout the class. In order to custom tailor the output to the various menus, I think I’ll create different similar input validation functions based on the amount of choices available rather than create a single mega function. This will make the code easier to read, and simpler to customize the output.

**Reflection**

First and foremost, I ended up tossing out the entire canyon escape idea. I finished constructing the map and realized I didn’t find the overall structure of it to be that interesting, so I scrapped it and came up with a Space station escape style game. The player is a chef on the space station when suddenly alarm bells indicating a loss of oxygen in the station go off. The player must then find their space suit. The limiting factor is oxygen. The player will collect oxygen canisters to refill their suit, but it is entirely possible to run out of air and perish.

I also did away with the random events aspect. This issue was that the player would occasionally have a series of bad events which made the game unfinishable. My concern was that the grading TA would get stuck with a couple different runs of bad events and not see the game finish out.

I quickly ran into an issue with my idea of how link the spaces together. It seems obvious now, but here’s what I tried first:

Menu::Menu()

{

startArea = new WalledCanyon(nullptr, waterDeadEnd, nullptr, nullptr);

waterDeadEnd = new DeadEnd(startArea, nullptr, nullptr, nullptr);

startArea = new WalledCanyon(nullptr, waterDeadEnd, nullptr, nullptr);

}

Of course, that didn’t work. Going to waterDeadEnd, then back to startArea worked, and it seems like the player should be able to go back to waterDeadEnd. However, the player is actually in the first startArea object. That object was created before waterDeadEnd was created, so waterDeadEnd in the first object just points to a garbage address. This is interesting because now two spaces exist with the same name. It’s interesting, but not very useful.

Instead I ended up resorting to a double linked list using four functions called addNorth(), addSouth(), addEast(), and addWest(). These functions take an incoming derived space class as an argument and link the spaces to create an overall map. This occurs when the menu class is first called. When constructing my map this way, I originally tried to create the map by moving a node pointer in various directions to find when next pointer would be null, then placing the incoming node there. The problem was this method was too difficult to keep track of throughout the different rooms. Instead, and because I only had eight rooms total, I just did one off if statements as needed in the add direction functions.

The next large issue I ran into was how with the overall function of the player inventory. I was having a difficult time allowing the various derived functions to access the inventory vector. The problem was the derived classes were not being called with objects, and a single vector couldn’t be created in the base class. Therefore, I could have either restructured my entire project or change the classes pass the vector around as references. Considering the amount of time it would have taken to plan, and implement an entire new project, I decided to pass the vectors as references between the classes. While this might not be the cleanest way, it works without issue and the resource usage is negligible. A learning experience for next time though.

The problem I ran into with the above is that by passing arguments to derived Space classes, not all of my functions in the base Space class could be pure. I wrestled with this issue for a while, but according to an instructor endorsed post on Piazza, not all functions in the Space class need to be pure and virtual. My interpretation is that we are to implement a pure virtual function as a way of showing that we understand how to implement them. I don’t have a reason to use a pure virtual function in my program, but I coded one in to fulfil the assignment requirements. The post regarding having not every virtual function be pure can be found at the following link: <https://piazza.com/class/j6zp57dd1bh3uy?cid=403>

For the player’s inventory, I decided to allow them to choose a ships operations manual or a crowbar. The manual allows for a more thoughtful problem-solving approach, while the crowbar provides brute force solutions. Initially, I was tracking the various items through a vector for the player’s inventory. However, this became redundant because I was also using an array to track the state of the world. The array keeps track of what the player has done so the correct dialog options are shown. Therefore, I cut the inventory down to only one item. I am still using a container to handle the inventory, per the assignment’s requirements, however only one item is needed because the player’s interactions with the world are being recorded in an array. I made this choice because I thought changing the dialog was more interesting for a player than having them enter a set of key presses to use various items. Instead, now their actions change the environment.

The next problem I had was with tracking the player’s oxygen usage. I tried using mutator and accessor functions in my menu class and also in my base class. Neither would allow the derived classes to access them without creating an object. I could have passed an object between the derived classes, but that seemed more complicated than just passing an integer and incrementing/decrementing it in the various classes. I am sure there’s a better way to accomplish this, but I believe it would require restructuring my program to have the menu class handle more of the calls. This lead to me having to implement an if statement at the beginning of every action change to see if the player had run out of oxygen. Again, it’s not pretty, but it does work. At this point, I don’t think there is much to be gained since the program is running correctly, but in the future, I’ll need to do a better job of planning to ensure I can better modularize the various functions.

The one issue I couldn’t solve involved a check of the inventory array. Valgrind would throw various “conditional jump or moved depends on uninitialized” values errors, but I checked multiple times that my variables were properly initialized. I would guess it has to do with me passing the vector so many times, but I wasn’t able to resolve that particular message. In the end though, the program runs and exits with no memory leaks, so it’s also possible this is a false positive from valgrind

A perfect run of the game is: 1 (start a new game), 1 (take manual), 1 (get space suit), 1 (put on space suit), 1 (head south), 2 (head into low oxygen environment), 3 (get floating oxygen bottle), 2 (go into storage room), 1 (pull breaker), 1 (back into dining room), 3 (go into exercise room), 2 (take slip of paper), 2 (punch in code), 2 (Get into escape pod), 3(Emergency launch button).

**Test Cases**Note: Due to limited variability of input (user can only enter 1, 2, or 3 throughout the program), input validation is only run through three functions, so only official three integer validation checks are needed. I did many more though throughout the testing process

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input Values** | **Driver Functions** | **Expected Outcome** | **Observed Outcome** |
| Entering a letter as input asks player to reenter input | Your choice: a | oneToThreeIntegerValidation(); | Player is asked to enter a number 1, 2, or 3 | Player was asked to enter a number 1, 2, or 3 |
| Entering a letter as input asks player to reenter input | Your choice: a | oneToTwoIntegerValidation(); | Player is asked to enter a number 1, or 2. | Player was asked to enter a number 1, or 2. |
| Entering a letter as input asks player to reenter input | Your choice: a | oneIntegerValidation(); | Player is asked to enter a number 1. | Player was asked to enter a number 1. |
| Player runs out of oxygen by going back and forth between rooms. | (after getting space suit)  Your choice: 2  Your choice: 1 | All Derived Space classes | Player is given game over message when air is equal to zero | Player was given game over message when air is equal to zero |
| Valgrind returns no memory leaks. | Choices: 1,1,1,1,2,3,2,1,1,3,2,2,3 | All functions | Valgrind confirms no memory leaks | Valgrind confirmed no memory leaks |
| Player enters oxygenless area without suit | Choices:  1,2,2 | Galley(); | Player is killed and game exits | Player is killed and game exits |
| Valgrind returns no memory leaks when player dies without suit. | Choices:  1,2,2 | Galley();  Menu class deconstructor | Valgrind confirms no memory leaks | Valgrind confirmed no memory leaks |
| Correct dialog give if player has crowbar in inventory | Choices:  1,2,1,1,3,2,1,1,3,2,2 | ControlRoom(); | Player is given dialog referring to taking the crowbar | Player was given dialog referring to taking the crowbar |
| Player can traverse all rooms, backwards and forwards, which shows list double linked correctly | Choice:  Varied as all rooms are cycled through by a player  (Infinite oxygen given for testing purposes) | Map(); //Constructed list  Menu();  Space(); | Player can able to travel through all the rooms. | Player was able to travel through all the rooms. |