**ASSIGNMENT 2**

Name: Lucas Monico

Athabasca ID: 3572445

Course: COMP 206

Date: 13 November 2024

**ASSIGNMENT 4:**

The first thing I did for this project was read the Wikipedia page for ‘Colossal Cave Adventure’ as well as watch this YouTube video:

<https://www.youtube.com/watch?v=sfGrPM5Bxeo>

While I’m not quite old enough (31) to have first-hand experience with text-based adventure games during their heyday, though I’m not entirely ignorant about their existence as I’ve always been interested in video game design and history. I also read many Choose Your Own Adventure novels as a kid, which had a similar premise.

My next step was to read Lewis Carroll's original novel, as I had never read it before and had only seen a couple of different movie adaptations of it. I'm not going to lie; the book was fairly odd, There was some fun wordplay in it, though.

After I read the book, I started making separate text files with all the characters, notable locations, items, and actions. I also drew some quick sketches of the locations for the game. I started piecing the game components together like filling out the descriptions for the different characters and locations as well as mapping out all the connection points between locations.

I knew this game was not going to have combat as Alice is a child, it will mostly be exploration and dialogue driven. Trying to stay true to the original while still giving the player freedom for moments of emergent gameplay. Or at least attempt to. This is supposed to be a relatively simplistic assignment I don’t want to make to mistake of increasing the scope to unrealistic levels for my ability.

I also made a simple block diagram of how to map will look. I understand that several of the rooms will only have one way in and out (share an entrance+exit) which is technically against the assignment guidelines, but for story purposes it seemed appropriate in certain cases and also kind of seemed like an arbitrary rule that doesn’t make much sense.

It is now January 16th. I took a long break as some personal things came up and I was procrastinating as I was having an issue with paralysis from analysis. I have not written any code since mid November so I’m feeling rather out of practice. I decided to watch a few tutorials videos that have a similar premise to this project:

[C++ Code-along Ep 8: Demo RPG Game | Core Stat Abstractions & Overloading Operators](https://www.youtube.com/watch?v=0BkNyp8wJkU&list=PLalVdRk2RC6pqOVxRNj5Uui7FN4r-WorM&index=8)

This one was particularly similar to this project so I watched both episodes, though I found her code process to be rather chaotic and not well explained:

[Let's make a text adventure in C++ (part 2/2) - Open Source Gamedev](https://www.youtube.com/watch?v=ikA4yiurkug)

I know I need to compartmentalize this project into smaller segments to prevent myself from being overwhelmed with the scale. I’ll start sorting the different classes and their objectives (Locations, Characters, Items, Actions, Inventory, etc.)

LOCATIONS:

* Should contain a description of the surroundings and pathways for travel
* Should contain hints in the description for items and characters that can be interacted with
* Some locations will have a temporary state that will change upon the next visit (either due to a one time event or because characters leave that area)
* Can travel North, South, East, or West for navigation (Some travel will be forced if an event is triggered – e.g. Tears event in the doorway hall at the beginning of the game)

CHARACTERS:

* There will be 23 characters in the game. I’m not sure yet If I will make them all intractable through actions yet or if they will be side decoration for dialogue (e.g. the ocean bank segment where Alice meets the Mouse & gang)
* Characters are: Alice, Sister, White Rabbit, Queen of Hearts, King of Hearts, Cheshire Cat, Duchess, Caterpillar, Mad Hatter, March Hare, Dormouse, Mock Turtle, Knave, Mouse, Dodo, Lory, Cook, Pigeon, Two, Five, Seven, Bill the Lizard, Frog Footman, Gryphon, and Pig Baby, \*enormous\* puppy
* Some characters will have special interactions with the ACTION keys and special interactions with used items in Alice’s inventory
* I am unsure how many liberties I can take with the dialogue from the original Alice in Wonderland book – due to it being public domain is it considered plagiarism to copy dialogue word for word? Or can I give credits at the beginning of the game to Lewis Carrol to acknowledge the borrowed work? I will have to modify the text and take creative liberties to avoid any possible issues.

ITEMS:

* Items are intractable objects found in the environs of the current location the player character is oriented.
* Some items can be added to Alice’s inventory using the TAKE action – and some will not be allowed to be TAKE(n)
* Some items could possibly be given directly to Alice from another non player character and therefore will never be interacted with or TAKE(n) from the environment.
* Some items in the game will be: Orange Marmalade, tiny golden key, curtain (cannot be added to inventory), tonic bottle (shrinking effect when consumed), little glass box (cannot be added to inventory), small cake (enlarging effect when consumed), white kid gloves, hand fan, box of comfits (in inventory at start of game), thimble (in inventory at start of game), small bottle (in W Rabbit’s house – enlarging effect), little cake/pebble (shrinking effect), stick, Mushroom piece (shrinking), Mushroom piece (neck stretching), Pig baby, tea, bread, croquet mallet (flamingo), tart, squeaky pencil, etc.

INVENTORY:

* Will be a storage container that can be accessed at anytime during gameplay.
* Will hold all items that have been given by NPCs or taken from the environment
* Will be able to do several actions with them (Throw, Inspect, Use)

ACTIONS:

* Commands input by the user to perform tasks such as: Talk, Inspect, Take, Move/Go, Throw (item only), Use (item only)

Considering how much word play and dialogue is involved in this game, I wanted to make it exploration and dialogue driven. Meaning I didn’t want to have fail states like death (so no combat). Now I would have to get a bit more creative with my writing abilities as a novel is obviously very linear in it’s storyline.

I then began creating the appropriate hpp and cpp files for the components of the game:

Main.cpp – where the game class object will be created and ran

Game.hpp and .cpp – where the main game logic and game class are declared and defined

Locations.hpp and .cpp – where the Location class logic is declared and defined

Characters.hpp and .cpp – where the Character class logic is declared and defined

Items.hpp and .cpp – where the Item class logic is declared and defined

Inventory.hpp and .cpp – where the Inventory class logic is declared and defined

Actions.hpp and .cpp – where the Action class logic is declared and defined

I used a few other people’s projects as reference as I was having a tough time deciding how to proceed with this project. I referenced:

[TextAdventure · master · Rachel Wil Sha Singh / Simple CPP Text Adventure · GitLab](https://gitlab.com/RachelWilShaSingh/simple-cpp-text-adventure/-/tree/master/TextAdventure?ref_type=heads)

[Zelda-Game/Zelda at master · harismuneer/Zelda-Game · GitHub](https://github.com/harismuneer/Zelda-Game/tree/master/Zelda)

I started with the Game.hpp and .cpp files. Leaving the Game constructor and destructor as default as I couldn’t think of anything for them just yet. The main.cpp will create a Game class object then run the member function of Game.startGame() which will output a few intro lines for the game and load the game data by calling loadGameData(); Then that will call the individual loading functions for the text files of Characters, Locations, Items, etc. Those individual functions have similar but not quite the same function of loading and parsing a text file then inputting the data into vectors to organize.

The first load function I worked on was the Locations. After spending some time formatting the Locations.txt file, making them all uniform so they could be loaded in order I started making the function within the Game class called loadLocations(). Using code from earlier assignments to open and read from a text file I started creating the framework:

void Game::loadLocations()

{

    std::ifstream file("Locations.txt");

    if (!file.is\_open()) {

        std::cerr << "Failed to open Locations.txt" << std::endl;

        return;

    }

    std::string line;

    Location location;

    while (std::getline(file, line)) {

        if (line.empty()) continue;

        // Check if the 'line' in the txt file starts with the '#' character

        if (line[0] == '#') {

            // Check if the current location object has a non-empty id

            if (!location.getId().empty()) {

                // Add the current location object to the locations vector

                locations.push\_back(location);

                // Reset the location object for the next set of data

                location = Location();

            }

        }

I knew I was going to use vectors for the text files so before I created this in the Game.cpp I declared the Game member functions and variables in the Game.hpp. I made all the ones that I thought I would need then commented them out for ease of testing functionality as I slowly built the game up. Starting with the locations and movement seemed like the logical path to take.

In the Game class private access specifier, I started declaring variables I thought or knew I was going to need. A pointer to the player’s current location. An instance of vector<Location> locations (so a vector instance of the Location class I will be making with the label locations). I also created a frame for items, characters, actions but I commented them out as I wasn’t ready to use them yet.

In the public access specifier I declared the specific methods for Locations like loadLocations(), move, userInput, playerDirectionalInput. These weren’t all done at the same time as I was jumping back and forth between files and modified a lot before writing this segment of the journal.

I then declared all the variables for Locations within the Locations.hpp then declared getters and setters. I decided to use the **this** pointer function to grab the pointer for the called object and access its member functions/variables. So for

void Location::setName(const std::string& name) { this->name = name; }

If I invoked this in Game.cpp for the getters and setters, it would look like:

 currentLocation->setName("Test Name");

        std::cout << "Current Location: " << currentLocation->getName() << std::endl;

For the Locations class I decided to have an:

* id
* Name
* Description
* items (available in location cell)
* characters (currently present in cell)
* path \*direction\* (would call the name if a valid path was in that direction)
* \*direction\* description (a default description would output if it was an invalid direction)
* A Boolean if that pathway direction was locked
* A key item used to unlock a pathway – it would be assigned to an item class object
* Possibly adding an int count or Boolean that checks if is the first time the player visits a Location (I plan on setting up specific events that happen on the first visit to an area)

After this I went back to the Game.cpp and finished the loadLocations member function for Game.

    std::string line;

    Location location;

    while (std::getline(file, line)) {

        if (line.empty()) continue;

        // Check if the 'line' in the txt file starts with the '#' character

        if (line[0] == '#') {

            // Check if the current location object has a non-empty id

            if (!location.getId().empty()) {

                // Add the current location object to the locations vector

                locations.push\_back(location);

                // Reset the location object for the next set of data

                location = Location();

            }

        }

From where we left off the function opens the file and creates an object of the Location class called location. It starts a new instance within this location vector if the line starts with a # to designate a new location. Then it goes through a series of if statements to categorize the location cell.

// Check if the line starts with a specific keyword and set the appropriate value in the location object

        if (line.find("id: ") == 0) {       // If the line starts with "id: ", extract the rest of the line

            location.setId(line.substr(4)); // Set the id of the location object to the rest of the line starting at the 4th character index

        }

        if (line.find("name: ") == 0) {     // Same as above just with different keywords

            location.setName(line.substr(6));

        }

        if (line.find("description: ") == 0) {

            location.setDescription(line.substr(13));

and so on for the various categories of a Location object.

I then began creating the userInput member function of the Game class.

void Game::userInput(const std::string &input)

The const is so that the method cannot modify the input object and the &input is so it uses the original input object that is passed in – for efficiency (not that it matters much with a program of this size)

It’s now February 6th 2025. I made a bunch of code for the userInput Game member function. The basic outline is that main will create a Game object, run the startGame function then ask for a user input that injects that into the userInput Game function. From there it converts it to lower case, separates the individual words using the whitespace as a marker and creates a string vector. I also have a map within userInput that maps the individual action keywords, this pulls from the Action.hpp where there is an enum class for the action keywords (like move, inventory, inspect, etc.). I’m debating if I should move this action map somewhere outside of the userInput function (I don’t want to introduce more bugs as I’m already having a bad time getting this to all work).

After the user input is separated into a vector It then gets injected into an auto variable to find a match with the users first command word with a word associated in the actionMap. If there is a match it then gets injected into a switch case and takes the value associated with that actionMap key command word. From there it will get diverted to the relevant member function for that command. Since I started with MOVE lets start with that one.

It checks if the commands vector has more than one object (more than one word). If it does it injects the second word of the MOVE command to a Game class member function called playerDirectionalInput().

The playerDirectionalInput function/method then uses if/else statements to find the appropriate fork for directions (north, south, east, west, or other). If it is a valid direction then it calls the Game class function move(Direction direction)

This function uses a switch statement to compare the user supplied ‘direction’ input, converts it to the appropriate Direction enum class (located in Actions.hpp) and then assigns the location id to a string variable using the Locations class get methods.

Then a for loop using auto compares all the locations vector objects id with the user input assigned location id. If there’s a match then it moves the player’s current location to the requested location.

This took several days to debug and get in a semi working state. I feel like I’m a bit over my head at the moment. It is getting difficult to keep track of the chain of functions that intertwine with each other. I may have bitten off more than I can chew with this project. I am trying to have descriptive comments within the files as the abstraction can be hard to remember. Especially when I’m sometimes not working on this project for weeks at a time.

I next made sure the quit action worked. I had some redundancy going on where the quit was technically handled by an if statement in the main.cpp. The user input would go through it’s switch statement for quit and call a return statement which then went back to main.cpp if statement quit that would break; the user input loop. I instead got rid of the main.cpp if statement and changed the return to quit(0); in the QUIT switch case for the userInput method.

I proceeded to work on the HELP action. To prevent feature creep (as I’m already in over my head) I decided to scrap the idea that you could use HELP in combination with other words for contextual hints. The HELP command will now purely print out the menu commands and how to use them.

This was easy enough to make. I declared the method in Game.hpp using ‘void printHelp() const’ as it doesn’t take any input and will not be returning a value. It simply prints text to the terminal. Then I typed out the method in Game.cpp which was just a few lines of cout statements.

Then modified the switch statement in the userInput method to deal with the HELP action. I also noticed it was just completely missing while I did this. Right now I have placeholders for all the actions to simply output a cout statement and then break; but I somehow forgot to make one for the HELP action.

I then decided to start working on the items class. I have already created the formatting on the items.txt so I basically need to copy my homework with the Locations class with how it loads in objects. I am getting a bit concerned with how items and locations will interact as I currently have the Locations class designate present items as a string. Will I need to convert that to the Item class object? Or an int counter of that class object if there’s more than one? I’m not sure yet.

I decided to make some changes while declaring the Item class variables. I figured I would need some Booleans for if it can be taken (into inventory), is consumable, is usable, is a key item (to unlock a location pathway). Then I needed some strings for the keyLocationId (Location ID that is unlocked if the item is a key item). Also the consume effect and use effect. I decided to make the take effect an int (as it made sense at the time) I assume I’ll be able to just +1 to the item’s count into the player’s inventory. Then I typed up the getters and setters in the header file.