**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.

January 16th, 2024

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.

Next I decided to create a helper function within the Game files that would load a txt file and read line by line, with line progression bound to the users ‘enter’ key to progress through exposition. This is because I planned on having some longer exposition/dialogue moments for special events that progress the story.

I called the function printTextFile() and it was fairly straightforward as the previous assignments covered similar materials but I also used code from here for the press enter to continue functionality:  
  
[c++ - Press Enter to Continue - Stack Overflow](https://stackoverflow.com/questions/903221/press-enter-to-continue)

After that was completed I made an intro.txt that would be opened and read after the startGame function was initiated.

February 7, 2025

I decided I needed to account for first visits to locations if I wanted to handle exposition events. There would also be some events that are initiated through actions so this kills two birds with one stone. I went back to the locations files and added another variable+getter+setters to the files for bool firstVisit and a string events (which would contain a txt file name) that would inject the file string into the printTextFile() method.

I also added an if statement to my Game::move method to check if firstVisit is true and events != “NULL” if so then it would load the event text file and set firstEvent to false.

This ended up not working as I was trying to access private member variables from Locations by using

currentLocation->firstVisit == true && currentLocation->events != "NULL"

in my Game.cpp file. I added Game as a friend class within the Locations.hpp but it still wasn’t working well.

Next I changed to using a getter function for the event variable of the Locations class but now I was running into a different problem. Using the getter method for getEvents would return the proper string variable but when injected into the printTextFile() Game class method it would trigger the “Failed to open “ <filename> cout statement. Yet the file name was the string name for event. I tried a few different variations using the “” quotations with and without as well as ‘’ single apostrophise but I so far cannot get it to work…

I tried to debug this issue for a couple of hours and found out that there is something unusual going on when the getter methods pull from the vector objects (specifically for Locations in this scenario) I’m not sure if it’s a buffer issue putting in left over white space or what… Honestly stumped as to what is going on. I found this unusual behaviour when I was testing functionality in the Game class move method:

    // iterates over each object in the locations vector uses & to reference each object to avoid copying the object for efficiency

    for (auto &location : locations)

    {

        // compares the id of the current location to the id of the next location from the switch statement

        if (location.getId() == nextLocationId)

        {

            currentLocation = &location; // sets the current location to the location element in the locations vector - if it matches location.getId()

            std::string eventString = currentLocation->getEvents();

            std::cout << "event file name:" << "'" << eventString << "'" << std::endl;

            printTextFile(eventString);

            // if the current location has not been visited before and has events, print the events

            if (currentLocation->getFirstVisit() == true && currentLocation->getEvents() != "NULL") {

                printTextFile(eventString);

                currentLocation->setFirstVisit(false);

                std::cout << "Moved to: " << currentLocation->getName() << std::endl;

                std::cout << "Description: " << currentLocation->getDescription() << std::endl;

                return;

            } else {

                currentLocation->setFirstVisit(false);

                std::cout << "Moved to: " << currentLocation->getName() << std::endl;

                std::cout << "Description: " << currentLocation->getDescription() << std::endl;

                return;

            }

        }

    }

    std::cerr << "Cannot move in that direction." << std::endl;

This caused the terminal to print strangely for this line:

std::cout << "event file name:" << "'" << eventString << "'" << std::endl;

It would output to terminal as:d  
  
'vent file name: 'fallingEvent.txt

Something strange was happening. So I decided to just try and remove all whitespace from the eventString. I used code from:  
  
[Removing white space - C++ Forum](https://cplusplus.com/forum/beginner/9557/)  
  
from the user Gumbercules

I made a Game helper function that looked like this:  
  
std::string Game::removeAllWhitespace(const std::string& input) {

std::string result = input;

result.erase(std::remove\_if(result.begin(), result.end(), isspace), result.end());

return result;

}

Then I tested it within the move function, and it worked! I haven’t got a clue why this extra white space is being created but I solved the issue.

Nevermind, I found the cause about an hour later after investigating.

It has to do with the way the text extraction works in the loadLocations method within the Game class.

Somehow the lines for sorting and importing strings from the txt files are inputting whitespace in the string that doesn’t appear during cout statements. I decided to place the removeAllWhitespace() helper function in the line to fix this issue of invisible whitespace:  
  
Went from:

    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();

            }

        }

        // 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));

        }

        if (line.find("items: ") == 0)

        {

and so on…

To this format:  
  
  
  
        if (line.find("events: ") == 0)

        {

            location.setEvents(removeAllWhitespace(line.substr(8)));

        }

And this fixed the issue. I did this only to the events variable. I’ll need to keep an eye on it in the future though as it may cause issues down the line.

When I work on this project at work I have to use a browser compiler to test the code. I was having an issue with the firstVisit variable within the Locations class always being considered false. It must have just been a bug with the browser compiler as when I compile on my home PC it works just fine. Anyway, I made it so that the currentLocation that the user is in gets its firstVisit variable set to false before switching to the next cell. This way events or scripts will not trigger again.

The move function is now in a working state to deal with events (thankfully) only took a full day of debugging.

*// iterates over each object in the locations vector uses & to reference each object to avoid copying the object for efficiency*

    for (auto &location : locations)

    {

*// compares the id of the current location to the id of the next location from the switch statement*

        if (location.getId() == nextLocationId)

        {

*// Changes old current location to false - so it doesn't trigger first visit events/scripts*

            currentLocation->setFirstVisit(false);

            currentLocation = &location; *// sets the current location to the location element in the locations vector - if it matches location.getId()*

*//bool firstVisit = currentLocation->getFirstVisit(); //DEBUGGING CODE*

*//std::cout << "First visit: " << (firstVisit ? "true" : "false") << std::endl; //DEBUGGING CODE*

*// if the current location has not been visited before and has events, print the events*

            if (currentLocation->getFirstVisit() == true && currentLocation->getEvents() != "NULL") { *//DEBUG THIS THE true IS NOT WORKING SOMEWHERE BETWEEN LOCATIONS.TXT AND HERE*

                printTextFile(currentLocation->getEvents());

                std::cout << "Moved to: " << currentLocation->getName() << std::endl;

                std::cout << "Description: " << currentLocation->getDescription() << std::endl;

                return;

            } else {

                std::cout << "Moved to: " << currentLocation->getName() << std::endl;

                std::cout << "Description: " << currentLocation->getDescription() << std::endl;

                return;

            }

        }

    }

    std::cerr << "Cannot move in that direction." << std::endl;

}

February 8, 2025

I remember learning a while back during a C programming course I took on boot.dev that C-style strings have a null terminator character at the end of them, \0, to designate their end. Apparently, std::string also has this null terminator. So that probably explains why I have been having so much trouble loading a string from a vector into the std::ifstream over the past couple of days.

Anyway, I started working on the items class some more. I had to figure out a way to get items to appear properly in the Location cells. I debated changing the current format for item types from a Boolean check within the Items class to making an enum class within Actions.hpp then having the Items vector hold a value for that enum. I might come back to that later. First I made an enum class for ItemEffect though as some consumable items will have effects like shrinking and enlarging.

Next I decided to change the Locations class files to account for the items variable being a vector object of the items class rather than a string. I also modified the way the Game class loadLocations handles loading in the Items.

My first issue within the loadLocations code for parsing items was that instead of comparing the item id found in the Items.txt then comparing with the Game class items vector I was creating new item vector objects then adding (push\_back) it to the specific location’s cell items vector. Problem with this is, it would be creating an Item class vector object with no other information other than an ID – so that was a funny mix up.

Then my next issue is I thought I fixed it and was not comparing the item ID found in the locations txt to the loaded items vector for the Game class but it just wouldn’t recognize the IDs. So I thought this was another whitespace issue like with the loading of the txt file string for events; but it wasn’t. No, instead I just messed up on the loadItems function and none of the item objects were actually being loaded into the item vector.

This was because I forgot this crucial piece of code at the end of loadLocations:  
  
*// Add the last item if it has a non-empty id*

    if (!item.getId().empty())

    {

        items.push\_back(item);

    }

    file.close();

Oh, also the if statements for the while loop were all out of proper tab indexing to be considered part of the while loop (and I was missing a closing } for the while loop, oops!)

I then went back and test that every location cell was working properly and was spawning in the properly assigned items.

At this point I figured it would be a good idea to work on the inventory system to store items. I’ll probably end up working on the TAKE action simultaneously since the two go hand in hand.

I also changed the order of the loadGameData to load the Items, Characters, Inventory BEFORE the locations as it would cause issues with load order (items, etc. would not load in as their vector objects would not be created before being reference by the loadLocations function).

February 9, 2025

Let’s get started with the Inventory system and Take command.

I started with the framework for the Inventory in the Inventory.hpp by declaring the variables and functions. First I made a map variable that would store a string and an int. This map would be called items.

Next I declared the functions addItem and printInventory. Fairly straightforward the addItem function would take an Item class vector object then use it’s getter method to increment it’s count into the Inventory map.

The printInventory function goes through a loop of the inventory’s key and values then prints them to the console:

I used code that I found on stackoverflow for the loop through a map:  
  
<https://stackoverflow.com/questions/26281979/how-do-you-loop-through-a-stdmap>

After that I move to the Game.hpp and declared a new function for the takeCommand. As I wanted the user input to be diverted from the switch case in the userInput method to the takeCommand; similar to how it works for the MOVE command.

I also made sure to add an inventory class object within the private section of the Game.hpp

Then I quickly added the inventory class printInventory() function to the switch case action for INVENTORY.

After that I began working on the takeCommand function that would filter the command word

At first I had it react similarly to the playerDirectionalInput function but then that wasn’t versatile enough to handle multiple word items (such as tiny gold key) as that only takes the second index of the user commands from the userInput function. I also realized that I would have to convert the player’s input for the name of the item vector to translate to the item’s ID as there was odd behaviour of a user input string not being recognized when compared to the getName getter function for the Items class.

I decided to make a map within the takeCommand function that would try to account for all the user input variations of the ingame items; then assign them to their respective item ID. This map function would operate very similarly to the one present in userInput for the actionsMap.

After it the user input gets sorted through the itemInputMap it gets processed through a for loop that goes through the items vector objects that belong to the current Locations cell that the user is present in. If the user input string is the same as the getID of a Item present in the location cell AND the item has the Boolean flag set to true for canTake, then the user adds that item to their inventory. If not it prints and error message.

Also back in the userInput function for the TAKE action I made a for loop that checks if the commands vector size is over 1 then uses size\_t to cycle through the commands vector starting at index 1 to prevent adding the action command to the string. The loop also adds a whitespace in between the commands found; then it injects the final string into the takeCommand function. This was done to account for items with multiple words.

That’s the end of my work for today. The next thing I need to work on is Items being removed from the location cell and/or the inventory if they are taken, used, or consumed. I also need to think about how I’m going to handle users performing actions on items that are in their inventory. Will they need to open inventory first then perform another command such as:

>Inventory

Inventory:

- CAKE: 1

>Consume Cake

Cake has been consumed

Or will they be able to just always perform actions on items in their inventory as if they were still part of the current cell. Meaning no requirement to prompt with the inventory command if they already know they have that item in their inventory…

Something to think about.

February 10, 2025

I had to change the way the for loop worked in the takeCommand method as I was trying to modify the items vector within the locations cell. This also required some tweaking of the referencing & and const values for the Locations header and cpp files for the getItems functions. Anyway, I created a variable that would hold the referenced vector of the current location’s items vector then I iterated over it in a for loop using auto. It would iterate by 1 through the vector until it reached the end.

If the current item Id matches the user’s input string and the canTake Boolean is true, then the item is added to the player’s inventory and removed from the locations items vector.

It now seems to be working…

I still need to implement and test:

USE action

CONSUME action

TALK action

INSPECT action (currently just been using it for temporary debugging cout statements)

Characters class files – and loadCharacters function within Game.cpp

Scripted events for main story beats (like Doorway hall crying event)

Update dialogue and location descriptions to describe the valid directions, present characters and items.

Further documentation and test cases – I have no idea how I’m currently going to go about this for a project this scale

Next I moved onto the consume command, which would allow the player to consume or eat/drink items that would give a ‘buff’ of sort. Since this game has no combat, it’s less a stat buff and more a Boolean flag that will be true or false for certain conditions; like enlarged, shrunk, satiated, etc. These will mostly just be used as a check to get through certain locked doors.

As soon as I started working on it I realized I’m probably going to need that itemInputMap map that I made in the takeCommand to convert an item’s name into it’s ID. So I moved that map into the Game constructor and forward declared the map in the Game header file.

\*\*Nevermind – moving the map into the constructor caused the takecommand to not recognize items anymore. I also tried moving the map into the loadGameData function and had the same issue. However, when I moved the map to the private sector of the Game header file; then it worked. So I guess that’s where it will be located. I also had a weird bug where picking up the small cake item caused the cout message claiming the player picked up the red paint bucket to appear. However, when I checked which item despawned in the location and the item in the player inventory it was the proper cake item. This bug did not persist after the recompiling the executable so I have no clue what happened there. I’m also using GitHub’s codespace for my IDE today as I can’t natively compile C++ code on my work laptop. I’m assuming that is what probably caused the strange behaviour.

A few days ago I made an enum class in the actions header file for consumeEffect. I guess my plan is to refactor code that I used from the takeCommand (specifically the for loop that uses auto) and then if it matches the item ID and the bool for canConsume is true then it goes through another switch case statement for the different consume effects. Then it would cause a Boolean to switch to true for the player… Do I make player it’s own class within the characters files? Not sure yet.

When I refactored the code from takeCommand I wanted to modify that variable that copies the currentLocations items vector to instead copy and compare the items vector for the player’s inventory – I made a decision to only allow the user to USE and Consume items that are in their inventory. It would be too much work to allow for both the inventory and the currentLocations items. I’m already way out of scope on this project I need to cut some corners.

I decided not to use the Enum class for the consume effect as it was causing me to go more than three layers deep in a loop abstraction. Instead I think I’m going to have a playerEffect variable in the Game files that is Normal by default and then changed to whatever the consumeEffect is. Since the shrinking and enlarging are part of major story beats I’ll probably manually reset to Normal during specific events.

I finalized on this for now:

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

{

    std::string inputString = input;

    // Iterates through the itemInputMap to find a matching value for the user input keyword

    auto it = itemInputMap.find(inputString);

    if (it != itemInputMap.end())

    {

        inputString = it->second;

    }

    else

    {

        std::cerr << "Item not found." << std::endl;

        return;

    }

    // Follows a similar format to the takeCommand function but compares with the inventory items vector instead of the current location's items vector

    auto &inventoryItems = inventory.getItems();

    for (auto itemCycle = inventoryItems.begin(); itemCycle != inventoryItems.end(); ++itemCycle)

    {

        if (itemCycle->getId() == inputString && itemCycle->getIsConsumable() == true)

        {

            playerEffect = itemCycle->getConsumeEffect();

            std::cout << "Effect: " << playerEffect << std::endl;

            inventoryItems.erase(itemCycle); // Removes the item from the inventory

            std::cout << "You have consumed: " << itemCycle->getName() << std::endl;

            return;

        }

        else if (itemCycle->getId() == inputString && itemCycle->getIsConsumable() == false)

        {

            std::cout << "You cannot consume: " << itemCycle->getName() << std::endl;

            return;

        }

    }

    std::cerr << "Item not found in the inventory." << std::endl;

I was overcomplicating the first if statement to check if the consume effect was not “NULL” but then I realized that if the Boolean flag for isConsumable = true then there will always be an associated player effect.

Now that the consumables have been sorted out properly (I think) it’s time to move onto the use effect. This will be interesting as there will be a couple of systems talking to each other here.

That doorway hall location will require the playerEffect = “SHRINK” and also the player will need to use the tiny gold key item on the south doorway that leads to the Royal Gardens location. However, I am also going to try and do some scripting involved with the first visit to this location as I do not want the player to be able to visit the Royal gardens initially. There will be a series of actions that will happen:

* User opens door with key
* Needs to consume the tonic to shrink
* Now the door is locked again and the key is on the table – cannot reach the table due to SHRINK effect
* Finds a small box – opens the box – there’s a cake inside.
* Consume cake to have ENLARGED playerEffect
* Alice starts to panic and cry – White Rabbit spawns into the room
* Player (Alice) TALKs to the White rabbit – who scurries away dropping the hand fan and white gloves items
* Player cries more and is asked to use the hand fan
* Player shrinks from hand fan then a txt file event is loaded to exposition the tears flooding the room
* Introduces the mouse and other characters
* Character is scripted to move to the Beach Bank location
* firstVisit Boolean is set to false for the Doorway Hall and next time player visits they can properly get to the Royal Gardens

My next step is to work on the USE action. I’m going to also restrict it to only items within the inventory.

February 11th, 2025

I figured out how to stop that bug when the user takes an item in a location then it prints the wrong item name to the terminal. I simply swapped the order of the cout statement that prints the current itemCycle item’s name and the locationItems.erase(itemCycle) so that it prints the statement first before deleting the item from the location’s item vector.

I also had an issue in the inventory files. Due to switching to a vector of the Items class to store the items (which was necessary), my current solution for adding an int count of the items was outputting the total integer count of items in the player’s inventory rather than the int count for individual items. Uncommenting the original map solution I used (before moving to an Item class vector) and slightly tweaking it to keep count of the individual items fixed the issue. Now they both work together.

Within the userInput method of the Game files I modified the USE command, which basically mirrors exactly how take and consume work. Then I created the useCommand() method within Game.cpp. I also declared it in the header file. It also mostly mirrors the take and consume with it’s formatting except now when it’s looping through the items vector for the players inventory (as the player will only be able to use an item in their inventory) It will go through another layer of if statements for the different item effects. For example:

           if (effect == "SHRINK")

            {

                playerEffect = "SHRINK";

                std::cout << "You are now shrunk" << std::endl;

            }

The problem is the effects of using an item varies so greatly with the item. One of the items (stick) will initiate a txt file to load for a story beat (dogStickEvent.txt) where Alice throws a stick while shrunk to distract an excited dog. One of the items is the tiny gold key that will open the Royal Gardens location. For the key I had to create yet another method within the game files called handleUnlockEffect() which looks like this:  
  
  
void Game::handleUnlockEffect(const std::string& locationId)

{

    for (auto& location : locations)

    {

        if (location.getId() == locationId)

        {

            location.setSouthIsLocked(false);

            std::cout << "Unlocked location: " << location.getName() << std::endl;

            return;

        }

    }

    std::cerr << "Location to unlock not found." << std::endl;

}

Thankfully I can keep it half baked as there is only one key item in the game that is used to unlock an area. So I only have to account for setSouthIsLocked (into the Royal Gardens) otherwise I’d have to make a switch or if statement to check what directional path the key item unlocks.

There was one item; the small box. That would function by the player taking it into their inventory then using it, which would spawn a cake into their inventory. I tried to get this part to work for several hours, made a function called addInventoryItemById() that would take a user input string then loop through the game’s items vector looking for a match Id then add the item to the players inventory using the Inventory class function additem().

There was an issue where opening the box would not spawn the cake item, nor would it remove the box from the players inventory. However, when the player used the item again it would then properly remove the box but then spawn 2 cakes into the players inventory.

Turns out the problem was because I had the inventoryItems.erase(itemCycle) outside of the if statement. This was the problem code:  
  
  
auto &inventoryItems = inventory.getItems();

    for (auto itemCycle = inventoryItems.begin(); itemCycle != inventoryItems.end(); ++itemCycle)

    {

        if (itemCycle->getId() == inputString && itemCycle->getIsUsable() == true)

        {

            std::string effect = itemCycle->getUseEffect();

            if (effect == "SHRINK")

            {

                playerEffect = "SHRINK";

                std::cout << "You are now shrunk" << std::endl;

            }

            else if (effect == "ENLARGE")

            {

                playerEffect = "ENLARGE";

                std::cout << "You are now enlarged" << std::endl;

            }

            else if (effect == "UNLOCK")

            {

                std::cout << "You have used: " << usedItemName << std::endl;

                handleUnlockEffect(itemCycle->getKeyLocationId());

            }

            else if (effect == "WEAR")

            {

                std::cout << "You are now wearing the " << usedItemName << std::endl;

            }

            else if (effect == "THROW")

            {

                printTextFile("dogStickEvent.txt");

                //currentLocation->setLocation()

            }

            else if (effect == "SPAWN\_CAKE")

            {

                addInventoryItemById("CAKE");

                std::cout << "There was a small cake inside the box! It looks delicious." << std::endl;

            }

            else

            {

                std::cerr << "Unknown effect: " << effect << std::endl;

            }

std::string usedItemName = itemCycle->getName();

            inventoryItems.erase(itemCycle); // Removes the item from the inventory

            return;

        }

        else if (itemCycle->getId() == inputString && itemCycle->getIsUsable() == false)

        {

            std::cout << "You cannot use: " << itemCycle->getName() << std::endl;

            return;

        }

    }

    std::cerr << "Item not found in the inventory." << std::endl;

}

And it started to work properly when I changed it to this:  
  
  
    auto &inventoryItems = inventory.getItems();

    for (auto itemCycle = inventoryItems.begin(); itemCycle != inventoryItems.end(); ++itemCycle)

    {

        if (itemCycle->getId() == inputString && itemCycle->getIsUsable() == true)

        {

            std::string effect = itemCycle->getUseEffect();

            if (effect == "SHRINK")

            {

                std::string usedItemName = itemCycle->getName();

                inventoryItems.erase(itemCycle); // Removes the item from the inventory

                playerEffect = "SHRINK";

                std::cout << "You are now shrunk" << std::endl;

            }

            else if (effect == "ENLARGE")

            {

                std::string usedItemName = itemCycle->getName();

                inventoryItems.erase(itemCycle); // Removes the item from the inventory

                playerEffect = "ENLARGE";

                std::cout << "You are now enlarged" << std::endl;

            }

            else if (effect == "UNLOCK")

            {

                std::string usedItemName = itemCycle->getName();

                inventoryItems.erase(itemCycle); // Removes the item from the inventory

                std::cout << "You have used: " << usedItemName << std::endl;

                handleUnlockEffect(itemCycle->getKeyLocationId());

            }

            else if (effect == "WEAR")

            {

                std::string usedItemName = itemCycle->getName();

                inventoryItems.erase(itemCycle); // Removes the item from the inventory

                std::cout << "You are now wearing the " << usedItemName << std::endl;

            }

            else if (effect == "THROW")

            {

                std::string usedItemName = itemCycle->getName();

                inventoryItems.erase(itemCycle); // Removes the item from the inventory

                printTextFile("dogStickEvent.txt");

                //currentLocation->setLocation()

            }

            else if (effect == "SPAWN\_CAKE")

            {

                std::string usedItemName = itemCycle->getName();

                inventoryItems.erase(itemCycle); // Removes the item from the inventory

                addInventoryItemById("CAKE");

                std::cout << "There was a small cake inside the box! It looks delicious." << std::endl;

            }

            else

            {

                std::string usedItemName = itemCycle->getName();

                inventoryItems.erase(itemCycle); // Removes the item from the inventory

                std::cerr << "Unknown effect: " << effect << std::endl;

            }

            return;

        }

        else if (itemCycle->getId() == inputString && itemCycle->getIsUsable() == false)

        {

            std::cout << "You cannot use: " << itemCycle->getName() << std::endl;

            return;

        }

    }

    std::cerr << "Item not found in the inventory." << std::endl;

}

I’m glad it finally got sorted but this project is a mess. I’m having trouble keeping track of all the different pathways and interactions. If things are convoluted at this scale, I can only imagine how bad it is for professional game developers.

February 12th, 2025

The main structure for the USE, TAKE, and CONSUME actions are done; though I will have to go back and do some functionality tweaking to fit the story beats.

I’m finally going to start implementing the characters and the Game method to load them into the game through a txt file. Thankfully most of the work is done for me already it’s mostly a copy and paste job from loadItems() and loadLocations()

I tweaked the format inside Characters.txt to include a talk option for all the locations in game. Just for consistency sake I’m not sure how a vector will behave if some characters have talk option rows that aren’t present for other characters so for consistency I just included all talk\_AREA for all characters and set to NULL if the character isn’t present in that area.

I modified the Locations.cpp and header file to account for characters being a vector rather than a string now (same formatting I did for Items)

Then I did the time consuming task of creating the header and cpp file for Characters. I’m regretting making a row for a talk option for every location cell now. Lots of getters and setters.

I briefly tested the code and found a few bugs (of course) typos, accidentally calling the getCharacters() method from the Characters class rather than the Locations class, and some other minor things. I noticed the characters weren’t loading into the locations even after fixing all the errors and it turns out I forgot to remove the comment that disabled the loadCharacters() method within loadGameData(). Just generally silly errors.

I now have the characters loading into locations properly however, while I was navigating through the locations verifying this I realized I could move into the Royal Gardens even though I did not use the key item. Then I realized I forgot to add a check to see if a directional pathway is locked when the character moves locations. I figured the best spot to put a check would be in the switch case statement in the move() method. It’s just a simple:

if (currentLocation->getNorthIsLocked() == true)

{

Pathway is locked error

Return;

}

Else

{

Carry on to new location

Break;

}

Seems to be working fine. But now after testing the key in the Doorway Hall location I noticed it’s not actually unlocking the Royal Gardens location. After looking at the useCommand() and handleUnlockEffect() I figured out the issue. At first I thought it was the placement of the inventoryItems.erase(itemCycle); line within the useCommand() for the UNLOCK effect, but after looking at the handleUnlockEffect() code I recognized the issue immediately.

It was the line location.setSouthIsLocked(false);

//used for key items that unlock a location cell

void Game::handleUnlockEffect(const std::string& locationId)

{

    for (auto& location : locations)

    {

        if (location.getId() == locationId)

        {

            location.setSouthIsLocked(false);

            std::cout << "Unlocked location: " << location.getName() << std::endl;

            return;

        }

    }

    std::cerr << "Location to unlock not found." << std::endl;

}

What this was doing was taking the location object for the Royal Gardens and unlocking the south pathway there rather than unlocking the south pathway in the player’s current location; the Doorway Hall. This also explains why the Throne room location was unlocked when I was testing the characters earlier (as that is south of the Royal Gardens and has the isLockedSouth set to true).

I changed this line to:

currentLocation->setSouthIsLocked(false);

And now it is working as intended.

Before I start working on implementing the TALK command I wanted to make a helper method within Game that will allow me to add characters to the player’s current location and for that matter I should probably make a version to remove a character from a location too.

I started by making a method within the Locations files; following the same general path as I did for adding an item to the player’s inventory. It basically just uses the push\_back() member method of a vector to add the character to the back of the characters vector within Locations.

Then I began working on the method within the Game files. I titled the new helper method as addCharacterToLocation() then used the same outline as the addInventoryItemById() method by using an auto for loop the cycle through the characters vector until it matches the given ID with the vector’s IDs then it calls the newly made addCharacter() method on that character’s vector object.

void Game::addCharacterToLocation(const std::string& characterId)

{

    for (auto& character : characters)

    {

        if (character.getId() == characterId)

        {

            currentLocation->addCharacter(character);

            return;

        }

    }

    std::cout << "Character not found!" << std::endl;

}

February 13th, 2025

Finally, I can start working on the talk command. The end is nearly here. The talk command switch case within the userInput() method copied the same format as take, use, and consume to filter the first action word then taking whatever words are left and making them into a string: in this case, a character’s name.

Then I declared the talkCommand function within the Game header file and began working on the definition for it within the cpp file.

I didn’t want to go through the trouble of creating another map to look for slight matches like I did with the item names. So I decided to do a bit of online searching (or reminding, as I’m pretty sure I’ve learned this before but I have the memory of a goldfish) to match a string with another to find partial matches. Using this link to the cpp docs I found:  
  
<https://en.cppreference.com/w/cpp/string/basic_string/find>  
  
with uses the .find() function to check if a substring (in this case the use input) exists within the characterName string if it does, even partially then it will then check the current location and return the talk dialogue for that location.

void Game::talkCommand(const std::string &input) {

    std::string inputString = input;

*// Iterate through the characters in the current location*

    for (const Character& character : currentLocation->getCharacters()) {

        std::string characterName = character.getName();

*//converts character name to lower case to match player input*

        for (int n = 0; n < characterName.length(); n++)

        {

            characterName[n] = tolower(characterName[n]);

        }

*// Checks if the player input is even sligtly a match to a character name present in currentLocation*

        if (characterName.find(inputString) != std::string::npos) {

*// Pull the appropriate talk line for the current location*

            std::string talkLine;

*//finds current location and assigns the correct contextual talk dialogue for currentLocation*

            if (currentLocation->getId() == "riverBank") {

                talkLine = character.getTalkRiverBank();

            } else if (currentLocation->getId() == "rabbitHole") {

                talkLine = character.getTalkRabbitHole();

            } else if (currentLocation->getId() == "landingHall") {

                talkLine = character.getTalkLandingHall();

            } else if (currentLocation->getId() == "doorwayHall") {

                talkLine = character.getTalkDoorwayHall();

            } else if (currentLocation->getId() == "beachBank") {

                talkLine = character.getTalkBeachBank();

            } else if (currentLocation->getId() == "whiteRabbitHome") {

                talkLine = character.getTalkWhiteRabbitHome();

            } else if (currentLocation->getId() == "denseWoods") {

                talkLine = character.getTalkDenseWoods();

            } else if (currentLocation->getId() == "mushroomPatch") {

                talkLine = character.getTalkMushroomPatch();

            } else if (currentLocation->getId() == "duchessHomeExt") {

                talkLine = character.getTalkDuchessHomeExt();

            } else if (currentLocation->getId() == "duchessHomeInt") {

                talkLine = character.getTalkDuchessHomeInt();

            } else if (currentLocation->getId() == "marchHareHome") {

                talkLine = character.getTalkMarchHareHome();

            } else if (currentLocation->getId() == "royalGardens") {

                talkLine = character.getTalkRoyalGardens();

            } else if (currentLocation->getId() == "croquetField") {

                talkLine = character.getTalkCroquetField();

            } else if (currentLocation->getId() == "royalBeach") {

                talkLine = character.getTalkRoyalBeach();

            } else if (currentLocation->getId() == "throne") {

                talkLine = character.getTalkThrone();

            }

*//edge case if a character is present but doesn't have a talk line - the player should never see this message in final draft.*

            if (talkLine == "NULL")

            {

                std::cout << character.getName() << " has nothing to say to you." << std::endl;

                return;

            }

*// Print the talk line*

            std::cout << character.getName() << " says: " << talkLine << std::endl;

            return;

        }

    }

    std::cerr << "Character not found in the current location." << std::endl;

}

I might refactor the USE and CONSUME actions to use this as it seems a bit more elegant in execution. That will have to be a task tomorrow, though. I still need to complete the INSPECT command as well.

February 14th, 2025

I decided against refactoring the consume and take command to function the same as the talk command with the line:

if (characterName.find(inputString) != std::string::npos)

and an attempt to remove the itemInputMap within the Game class. I tried with the consume command and it ended up introducing a bunch of unintended behaviour. So I reverted back. It works as is so no need to fix what isn’t (mostly) broken.

I did notice during my testing that the inventory had strange behaviour if the player happened to have more than one unit of the same item in their inventory. It would print the item name twice (if the amount of units was 2) and it would also not update the count as you used or consumed items (until you used all items in which case the item was completely remove from inventory). I at first I wanted to make a method within the Inventory class that would account for both the removal and decrement of the item but I just could not get it working properly. So I made the method within the Inventory class simply take an item object input and decrement the inventory count for that item ID. Then I kept the same lines in the consume and take methods and simply called the new decrementItem() method about the erase(itemCycle) lines.

To deal with the duplicate entries in the inventory when using the inventory command, I tweaked the printInventory() method within the Inventory class.

I added another map that exists only in the method but functions like the itemCounts map within the inventory class. It basically acts as a counter for unique item ids. If the item ID already has a count equal or greater than 1 it breaks from that cycle of the loop and doesn’t print the item again.

//loops through the items map and prints the item ID and count that currently exist in the inventory

void Inventory::printInventory() const {

    std::cout << "Inventory:" << std::endl;

    std::map<std::string, int> itemCount;

    for (const auto& item : items) {

        if (itemCount[item.getId()] >= 1) {

            break;

        }

        itemCount[item.getId()] += 1;

        std::cout << " - " << item.getName() << ": " << itemCounts.at(item.getId()) << std::endl;

    }

}

After that I decided to move the inspect command output and made a display command (as I was essentially using the inspect command in place of display). I made it so display can be called in game to check the status of the game, it lists off things like current location, location items, inventory items, location characters, and current player effect. I moved this functionality as the Inspect command will be part of the gameplay loop to get descriptions of items, pathways, locations and characters and shouldn’t be used for debugging purposes.

I then started fleshing out the Inspect command. I was a bit stumped at first on how to sort through the user input to check if it was a direction, location name, character name, or item name. Then I came to the conclusion that I’ve basically already done this filtering process several times over and I just need to repeat the steps with individual if statements (and for loops to cycle through the different vector types for locations, items, and characters). I’m sure there’s a more elegant way of doing this but I’m not smart enough to figure it out. So I mostly just refactored code I’ve already written.