INSIDE

PROJECT REPORT

Subject:	Fundamentals of Computer Programming
Class:	BESE-14B
Instructor:	Miss Momina Moetesum
Due Date:	25th of December, 2023
IDE Used:	Visual Studio Community 2022

Group Members:

Rafay Ahmad (CMS ID #478383)

Danish Munib (CMS ID #461050)

Table of Contents:

1. Introduction 3
   1.1. Project Overview 3
2. Game Concept 4
   2.1. Storyline 4
   2.2. Gameplay Mechanics 5
3. Design 6
   3.1. Architecture 6
   3.2. User Interface 13
   3.3. Flow Charts 19
4. Technical Requirements 22
   4.1. Programming Language 22
   4.2. Libraries & Tools 23
   4.3. Platform Compatibility 23
5. Code 24
   5.1. Organization 24
   5.2. Key Snippets 25
6. Challenges Faced 29
   6.1. Technical Challenges 29
   6.2. Design Challenges 30
   6.3. Level Design Iterations 32
7. Learning Objectives 33
   7.1. Programming Skills 33
   7.2. Problem Solving 34
   7.3. Collaboration & Communication 34
8. Conclusion 36
   8.1. Summary 36
   8.2. Future Enhancements 37
9. Acknowledgements 38
10. References 39

1. Introduction

Welcome to the immersive world of "Inside," a captivating text adventure game that invites players to embark on a journey of mystery and discovery.

With this project, our team aims to create an engaging and interactive narrative-driven experience, where players navigate a richly detailed virtual environment through the power of their words.

1.1. Project Overview

Project Overview:

Inside" is a text-based adventure game that places players at the heart of a compelling and enigmatic storyline.

Set in a world filled with secrets, challenges, and unexpected twists, the game unfolds as players make decisions, solve puzzles, and shape the outcome through their choices.

The narrative experience has been carefully crafted to offer a unique and memorable gaming experience, combining both suspense and exploration.

Purpose and Objectives:

The primary purpose of "Inside" is to provide players with a captivating and intellectually stimulating gaming experience.

Through intricate storytelling and well-designed challenges, we aim to engage players both emotionally and intellectually.

Our project focuses on developing a dynamic and branching narrative structure, ensuring that player decisions have a meaningful impact on the unfolding storyline.

Our objectives include creating a user-friendly interface, albeit inside the console environment, whilst also designing compelling story arcs, and implementing a robust decision-making system to enhance replay ability.

Target Audience:

"Inside" is designed for adventure game enthusiasts, story-driven gamers, and anyone who appreciates the art of interactive storytelling. The target audience includes individuals who enjoy exploring virtual worlds, solving puzzles, and making impactful decisions that influence the course of the narrative. The game's accessible text-based format allows players to immerse themselves in the storyline without the need for complex graphics, making it suitable for a wide range of devices and platforms.

2. Game Concept

The following sub-sections delve deeper into the general game concept that we had in mind during the development of our game:

2.1. Storyline

Inside the enigmatic world of "Inside", you step into the shoes of Nell, a resilient female protagonist seeking to unravel the mysteries of her past.

Synopsis:

The narrative unfolds with Nell standing on a cliffside, haunted by the shadows of forgotten memories, providing players with an immersive and intriguing starting point.

As Nell, your primary objective is to navigate through the cryptic landscape and confront the looming silhouette of a mysterious house that beckons in the distance. Armed with little knowledge of the game world, your journey within the house unfolds as a series of intricate puzzles and challenges, each unlocking fragments of Nell's forgotten memories.

The house itself is ever evolving, a labyrinth of secrets and surprises that reflect the depths of Nell's subconscious.

Your exploration serves as a key to unveiling the truths buried within her past, shedding light on the circumstances that led her to the cliffside.

Every puzzle solved and memory unlocked brings you closer to understanding the enigma that surrounds Nell.

Characters:

Nell:

Role: The central figure and main protagonist.

Background: Haunted by her mysterious past, Nell stands on the precipice of forgotten memories.

Journey: As players guide Nell through the enigmatic house, they unravel the layers of her past while making choices that shape her destiny.

Emotional Arc: Nell's character development hinges on the player's decisions, and her resilience is tested as she seeks solace in the shadows of her own history.

Adrian:

Role: Nell's husband.

Significance: Adrian's importance in the storyline deepens as players unlock memories of Nell.

Mystery: The enigmatic nature of Adrian's role unfolds through the narrative, revealing connections that bind him to Nell's haunted past.

Impact: Understanding Adrian's significance becomes a key element in navigating the complexities of Nell's story.

Mark:

Role: Nell's brother.

Presence: While not prominently featured, Mark still acts as a means for exposition for the player.

2.2. Gameplay Mechanics

Inside employs an intuitive command-based system that allows players to navigate the world and interact with its intricacies.

Fundamental commands such as **"move," "look," "interact,"** or **"check"** empower players to explore the mysterious house and delve into the narrative.

These actions serve as the foundation for a gripping gameplay experience, combined with various other commands for ease of accessibility like *showing the map, saving, loading, describing the environments as well as checking the inventory.*

Some examples:

Move: Navigate through the ever-evolving house and its diverse environments.

Look: Examine your surroundings, revealing hidden clues and nuances

Interact/Check: Investigate in-game objects, solving puzzles and unlocking memories tied to Nell's past.

Inventory: Displays the current items that the player holds.

The narrative takes a psychological twist as Nell faces different phobias, each manifesting within her in a dynamic combat system crafted from the ground up.

Your strategic choices in facing these fears not only shape the character's development but also impact the unfolding story.

Nell's evolution, both emotionally and psychologically, is intricately linked to your decisions, creating a deeply personalized gameplay experience.

Attack, Defend, and Focus: You can strategically choose between attacking foes, defending against their assaults, or focusing to charge up powerful attacks. Each decision affects the flow of combat.

Slider-Based Real-Time Combat: Engage in combat using a slider-based system, where precision and timing influence the success of attacks, defences, and special moves.

Key Story Beats: Throughout your playthrough, the importance of unlocking key story beats becomes apparent, as these revelations are vital to navigating the final encounter of the game.

Choice-Based Culmination: Reaching the pinnacle of the game with a choice-based final encounter that tests your understanding of the story elements and challenges your interactions with the game.

Divergent Endings: Your decisions throughout the game culminate in diverse outcomes, leading to either positive or negative endings based on the choices made during the final encounter.

Your actions, decisions, and the mysteries you uncover directly influence the narrative's outcome, ensuring a unique and highly replay able gaming experience.

As a result, "Inside" transforms traditional text-based gameplay into an immersive experience, where exploration, combat, and decision-making seamlessly weave together, providing players with a truly interactive and emotionally resonant adventure.

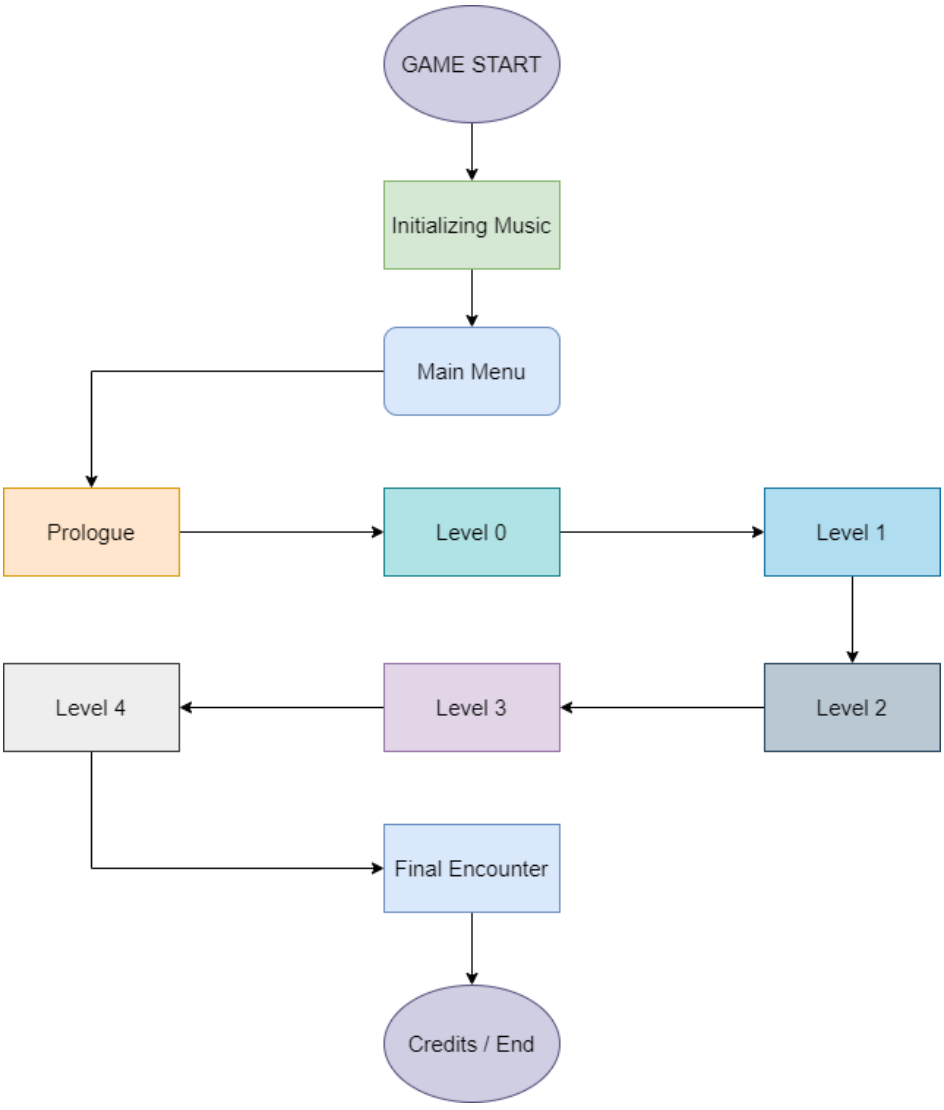
3. Design

The following section highlights the overall design and structure of our game, it's user interface design.

Various Flow Charts are included to illustrate the design of our game in a digestible manner.

3.1. Architecture

Here's a general flow structure for our game:



At the start, the game initializes all the in-game music, then continues to execute the main menu function. Depending on the user's choices, different functions can be executed by the main menu function.

However, for simplicity's sake, here we have only outlined the generalized structure of what a typical playthrough will look like, as executed by the game engine.

NOTE: Flow Charts for individual functions are present in the Flow Chart section. (See Page 19 onwards)

Functions:

Text.cpp: *displayText()* and all its variations to handle text printing.

Events:

- Loops through all the characters in the provided string and prints them with a delay.
- Keeps polling for user pressing the [Right Shift] key.
- Skips printing delay if [Right Shift] key is pressed.

Variations:

Other variations include displayTexty(), displayTextr(), displayTextg(), displayTexti(), displayTextgrey().

These variations call the main displayText() function but change the formatting using C escape sequences.

Affiliated Functions:

- *pause():* Pauses and waits for user pressing [Backspace] to continue.

Menu.cpp: *menu()* function to handle all the main menu events.

Events:

- Printing the welcome screen.
- Getting user input.
- Executing commands matching the user input.
- If incorrect command, printing error and retaking input.

Affiliated Functions:

- *help():* Prints a list of commands and instructions for playing the game.

Movement.cpp: *inputID()* & *computeInput()* functions to handle most of the player inputs.

Events:

- They both work in conjunction with each other.
- inputID() compares input to preset inputs and returns the ID of matching pair.
- computeInput() uses a switch to jump to the ID of matching pair and executes said action.
- Prints and error statement if the command is not identified.

Affiliated Functions:

- *lowercase():* Converts user input to lowercase.
- *displayPos():* Displays current location of the player inside the game world.
- *displayDirections():* Displays available movement directions to the player.
- *check():* Used when user inputs "check" or "interact". Displays unique text for checking any item / location.
- *updateMap():* Updates the map to reveal more information on the in-game map.
- *clearMap():* Clears the map when the user starts a new level.
- *printMap():* Prints the map if the user inputs "show map" or "map".

***Sound.cpp*:** *initMusic()* function to initialize all in-game music.

Events:

- Loads all music from files in the game directory (Inside Music Folder)
- Sets various music objects on loop.

Inventory.cpp: *displayInventory()* to display current inventory items.

Events:

- Checks for items that the player has and prints their name and descriptions.

Credits.cpp: *credits_good()* & *credits_bad()* to display good / bad credits sequences.

Events:

- Plays credit music.
- Prints the credits.
- Exits once completed. (Stops Music as well)

Memories.cpp: *memory_1(), memory_2(), memory_3(), memory_4(), prologue().*

Events:

- Plays memory music.
- Prints the memory text.
- Stops music and ends function once complete.

Affiliated Functions:

- *mem_1e():* Dialogue after ending level 1. (Changes based on your choice in-game)
- *mem_2e():* Dialogue after ending level 2. (Changes based on your choice in-game)

Enemy.cpp: *enemy(), enemyAttack(), attack(), defend()*

Events:

- Sets player health according to in-game buff items.
- Prints enemy encounter.
- Gets User Input.
- Calls attack(), defend() if user inputs “attack” or “defend”.
- Increases player damage to 2x if user inputs “focus”. (Skips their turn)
- Calls enemyAttack(), if user doesn’t manage to defend properly.
- Checks for player / enemy death.
- Repeats.

Affiliated Functions:

- *boss():* Prints final encounter, gives choices to player, computes said choices and branches accordingly until an ending is reached.

Levels.h: *level_0(), level_1(), level_2(), level_3(), level_4(), level_2pre(), level_3pre()*

Events:

- Starts level music.
- Begins Level Loop.
- Prints dialogue / text for current location.
- Updates map, displays available directions and current position.
- Gets user input.
- Compares input to unique events set for each level.
- If none match, sends input to be computed by computeInput().
- If pre-requisites for ending level are met, stop music and end level loop.

Affiliated Functions:

- *checkPos():* Unique hints and enemy encounters for each level.

Note: *The typical level format in our game will be touched upon later.* *(See Page 19)*

Others.cpp: *save(), load() for saving and loading.*

Events:

- Opens (or creates) *save.txt* in root directory. (Prints error if it fails. Returns error code)
- save() stores the values of in-game variables inside *save.txt*. *(As integers)*
- load() reads these values and updates the in-game variables.

Note: Boolean Values are stored as 1s and 0s, while integers are stored as they are.

The line the variable is stored in indicates which variable it is. (Using enums for this purpose)

Data Structures:

Maps.cpp:

Variables:

- *maps[][][]:* 3D integer array to store map data for each level.
- *tempMap[][]:* 2D integer array to temporarily store map data for a level for printing.
- *mapSize[]:* 1D integer array to store the size of each level (for the printMap function)

Describe.cpp & Directions.cpp:

Variables:

- *describeText[][]:* 2D string array to store point secondary descriptions for each level.
- *points[][]:* 2D string array to store location names each level.
- *checkT[][]:* 2D string array to store "check" descriptions for each location.
- *directions[][]:* 2D array of structure "lookText" to store 4 strings (North, South, East, West) and a location ID. Used to store descriptions for each direction when using "look".

Info.cpp:

Variables:

- *info[][]:* 2D array of a structure "text" to store 5 strings and a location ID.

Used for storing primary descriptions of each level (when player first visits a new location).

Allows up to 5 strings to be stored for the sake of accessibility during development.

Others.cpp:

Variables:

-	*locationFlags[][]:*	2D Boolean array to store if player has been to a certain location.
-	*inputQ[]:*	String array to store questions when user is prompted for input.
-	*moveFail[]:*	String array to store text to print when movement attempt fails.
-	*choiceQ:*	Integer stored randomly to be used when printing randomly.
-	*level:*	Integer to store current level.
-	*enemy_1a:*	Bool to store if enemy_1a is alive. (Level 1 enemy)
-	*enemy_2a, b, c:*	Bool to store if enemy_2a, b or c is alive. (Level 2 enemies)
-	*enemy_3a:*	Bool to store if enemy_3a is alive. (Level 3 enemy)
-	*locked:*	Bool to store if door in Level 2 is locked.
-	*puzzle:*	Bool to store if puzzle in Level 3 has been solved.

Note: *inputQ[]* and *moveFail[]* are printed at random using a random value stored in *choiceQ*.

Inventory.cpp:

Variables:

-	*hasKey:*	Does player have the key (from Level 0).
-	*hasAxe:*	Does player have the axe (found in Level 2).
-	*hasBook:*	Does player have the book (for puzzle in Level 1).
-	*hasSketch:*	Does player have the sketchbook (Unlocks Memory 1 in Level 1).
-	*hasBPrint:*	Does player have the blueprint (Unlocks Memory 2 in Level 2).
-	*hasReport:*	Does player have the medical reports (Unlocks Memory 3 in Level 3).

Structs Used:

-	*items:*	Stores strings *itemName* and *itemDesc*. *(For inventory)*
-	*lookText:*	Stores strings “north, south, east, west” and an int (For look command)
-	*text:*	Stores 5 strings and a location ID (For primary location descriptions)

Note: Location IDs are used most for our convenience. There’s no real point for using a struct for them otherwise as all this text can still be printed using ordinary string arrays (Like we have used in Describe.cpp & Directions.cpp)

Enums Used:

-	*inputs:*	Used in computeInput() as the switch cases.
-	*values:*	Variable names. Used in load() function as the switch cases.

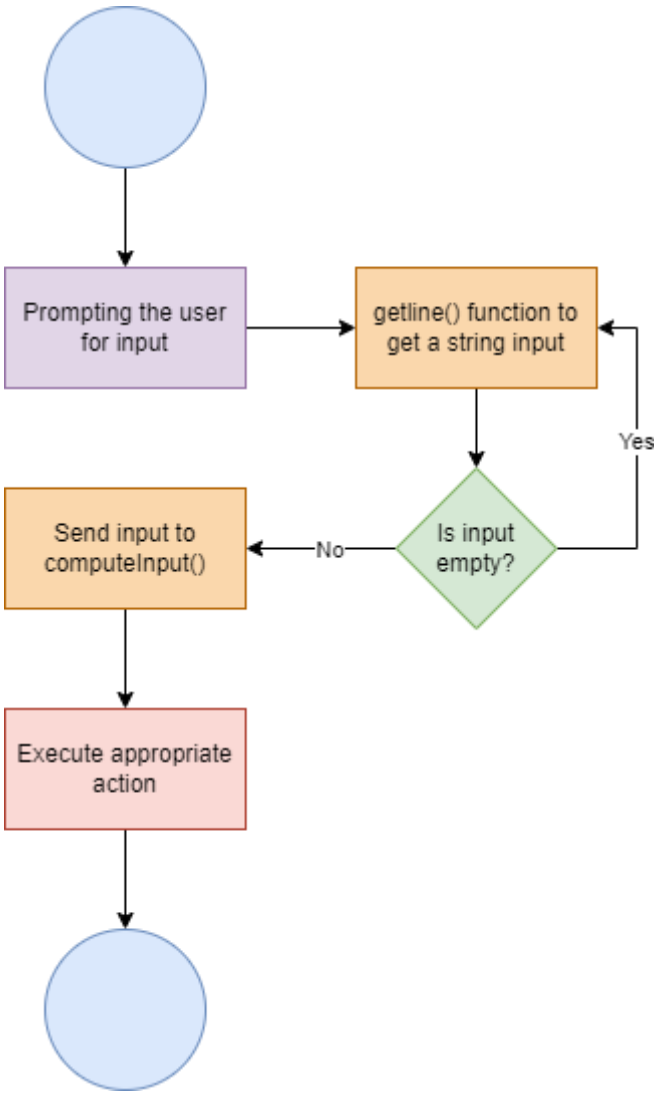
3.2. User Interface

The user interface of "Inside" uses a type-writer style text output system that loops through each character and prints a letter step by step, with each print action followed by a delay (that can be configured as a function parameter).

Each time the text is printed, a sound is also played to mimic a typical Role-Playing Game (RPG) text printing system.

User Input:

Here's how input is take from the user, processed, and the result executed.



Although, the basic structure of taking input, processing it, and executing the appropriate action remains largely similar throughout the game, there are certain sections (like the levels) where the input is first compared to set events within the level code before being sent to ***computeInput()*.**

Examples of User Input:

```
Welcome to INSIDE

INSIDE is best experienced in Fullscreen with Sound.
Press [F11] to enter Fullscreen Mode.

You may use the following commands to navigate the world:

-  look [direction]
-  move [direction]
-  interact / check
-  inventory

For additional commands, type "info".
You can fast forward the text using [Right Shift]

You may exit the game anytime by typing "exit".

To Begin, type "start":
```

Here's the first instance of when the game asks for input. Let's try typing **"info"** to see the output the game processes:

```
Movement Commands:

- "move"  [direction]
- [direction]

- "look"  [direction]

[direction] can be "north", "south", "east", "west".

General Commands:

- "check / interact":   Checks or interacts with objects in the game world.
- "items / inventory":  Displays your inventory.
- "help / info":        Prints out help...Oh you already used this command...
- "save / load":        Saves or loads your progress.
- "describe":           Describes your environment once more.
- "map":                Shows a map of the game world.
- "pick":               Picks up a key object.
- "place":              Places a key object.
- "open":               Opens a door.
- "exit":               Exits the game.

Combat Commands:

- "attack":     Deals a normal attack.
- "defend":     Defends against opponent's attack with a low chance of healing you.
- "focus":      Charges your next attack to deal 2x damage to the opponent.
- "stats":      Shows you the health of your opponent and yourself.

You can use [Right Shift] to fast forward text.

In key moments, [Backspace] can be used to continue the dialogue...
```

Here we can see that the game correctly identifies **"info"** as a valid command and executes the **help()** function. Now let's try typing a random command that the game may not recognize:

```
To Begin, type "start":  cool game
I'm afraid I do not recognize that command...

To Begin, type "start":
```

Again, the game correctly identifies **"cool game"** as a command that it does not recognizes and prints an error message, followed by prompting the player to enter another command.

Let's start the game. Here in Level 0, we get our first free choice inside the game world:

```
Waking up on a cliffside, you feel a cool breeze and the distant sound of crashing waves.
There's a mysterious house ahead, standing against the fading daylight.
The air is filled with anticipation.
Available Directions:  North
Current Location:  Cliffside
What do you want to do?
```

Here, we only have one direction we can move in. Let's input **"move north"**:

```
A mysterious house looms ahead. It is surrounded by an ominous aura.
Available Directions:  South  East  West
Current Location:  Front of house
What do you want to do?
```

Great! The game understood our command and we moved. Let's input **"show map"**:

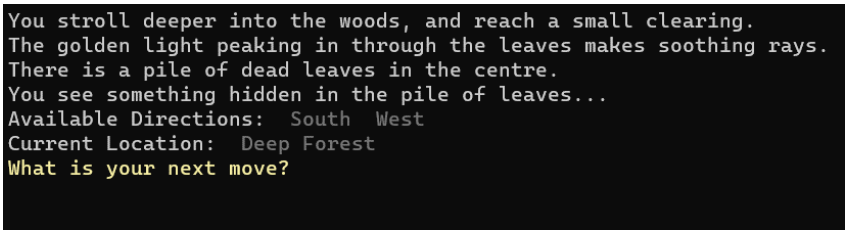
```
    X X X X X
    X X X X X
        X
    X X   X X

You are here:  X

Available Directions:  South  East  West
Current Location:  Front of house
How do you proceed?
```

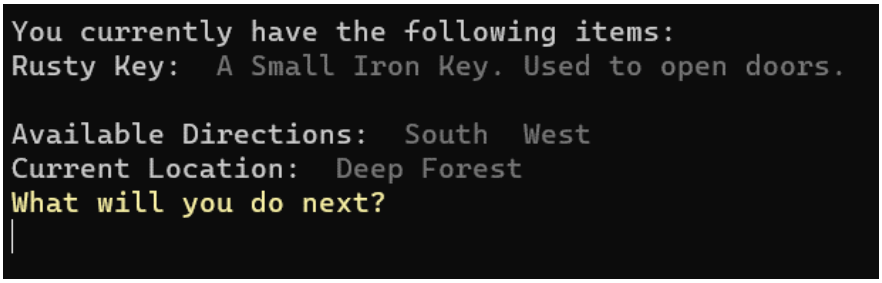
There we go. The map was printed correctly. It seems empty, but that's because we haven't explored that area yet. Let's do that and show the changes:

More of the map seems to have been revealed now. While exploring, I reached a point with some bushes:



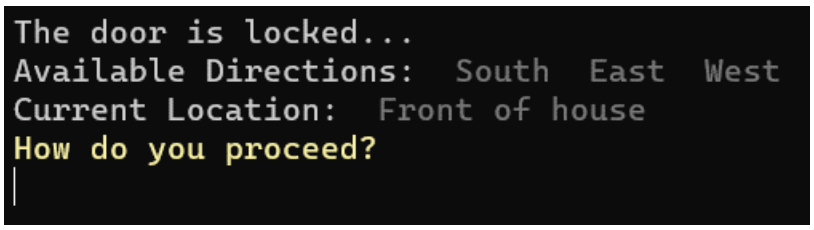
Let's use **"check"** to check the bushes:

Great! We should not have the key in our inventory. Let's use **"show inventory"**:

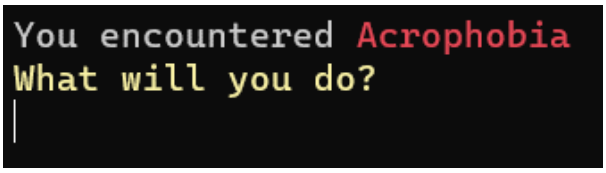


There we go. We now have the key. We can probably enter the house now.

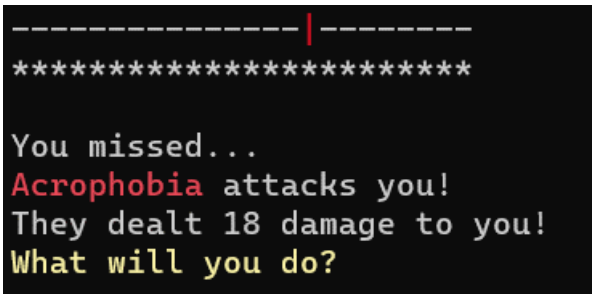
You might be wondering, what happens if you try to enter the house without the key? Let's start a new game and try it by typing **"enter"** when in front of the house:



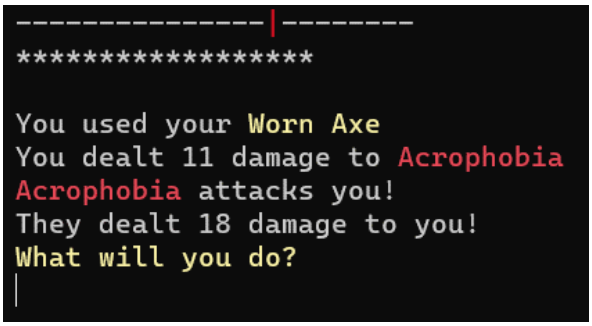
Quite anti-climactic. But it does prevent you from progressing without the key! Let's take a look at the combat system now:



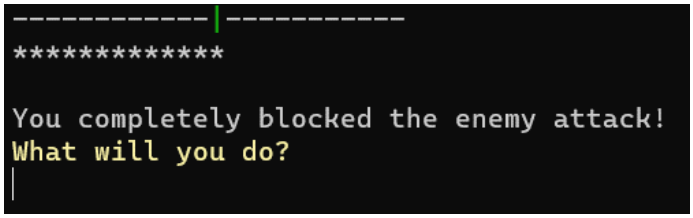
It seems we encountered **Acrophobia**. Let's **"attack"**:



Whoa! What just happened? It seems I didn't press [Right Shift] in time. Let's do that the next time we **"attack"**:



It seems I was slightly off. Doesn't matter though, let's try **"defend":**



I managed to evade their attack! Moreover, it seems I also had gained some health by defending. Let's **"focus"** now and deal a hefty hit:

```
You decided to use Focus. Your damage for next round increased.
```

```
---------------|--------
******************

You used your Worn Axe
You dealt 26 damage to Acrophobia
Acrophobia attacks you!
They dealt 17 damage to you!
What will you do?
```

That's a heavy hit! Let's check our **"stats"**:

```
Your health: 58 HP.
Acrophobia health: 58 HP.
What will you do?
```

Going quite close...But before we end it, let's try the **"info"** command during a battle scenario:

```
You can do the following:
- Attack:       Deals a normal attack.
- Defend:       Defends against opponent's attack with a low chance of healing you.
- Focus:        Charges your next attack to deal 2x damage to the opponent.
- Stats:        Shows you the health of your opponent and yourself.
- Info: You just did that, didn't you?
What will you do?
```

Ah that explains a lot. Alright, let's finish the battle:

```
---------------|--------
*****************

You land a critical hit!
You used your Worn Axe
You dealt 40 damage to Acrophobia
You won!
```

Oh! A critical hit! What a nice way to end the battle.

These are just a few examples of how the input system works in our game. There's much more to the game than just these commands. A full list will be available in the **readme** file.

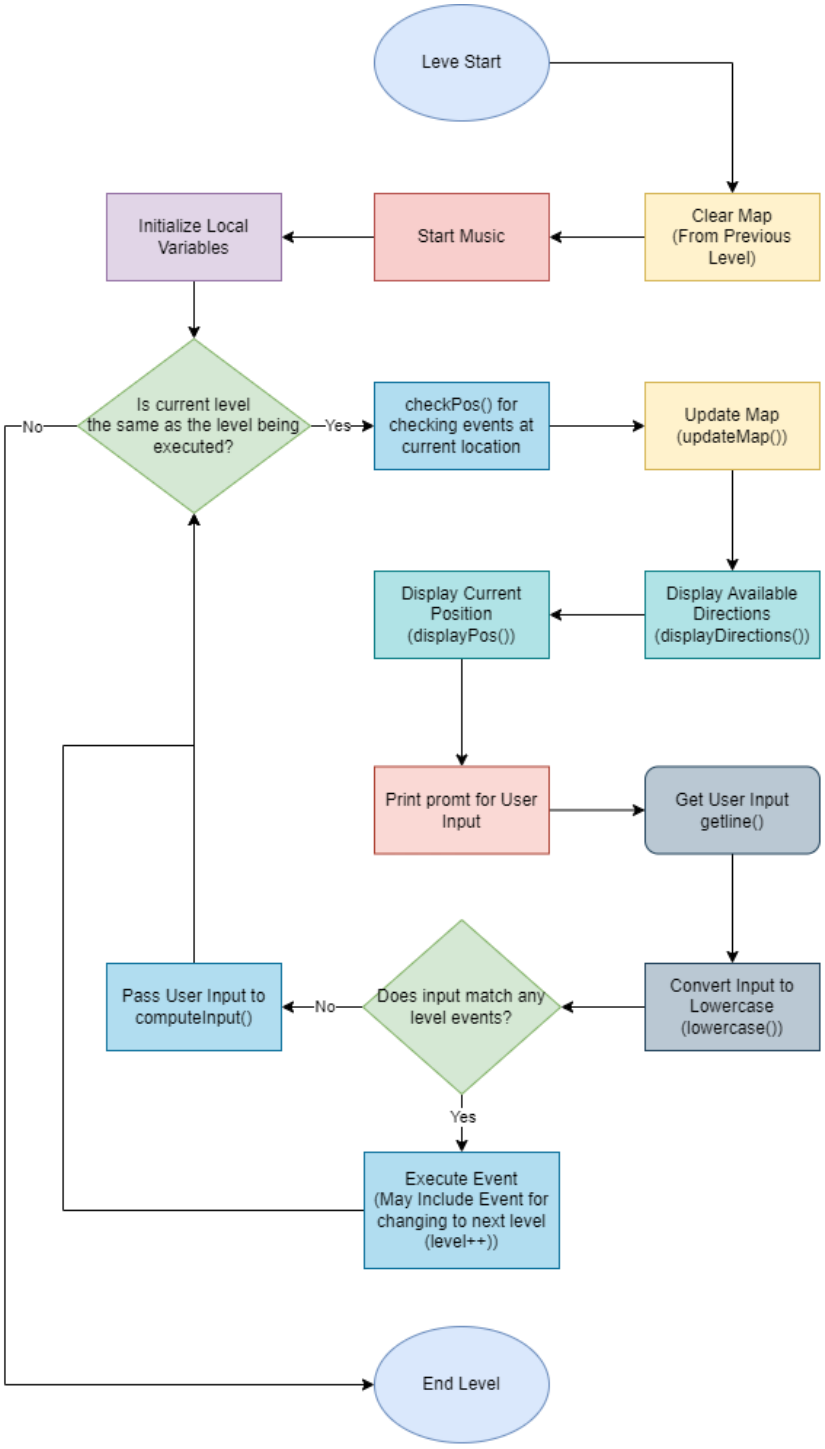
3.3. Flow Charts

In this section, we have added multiple flow charts to show the flow of the different parts of our game.

General Level Structure:

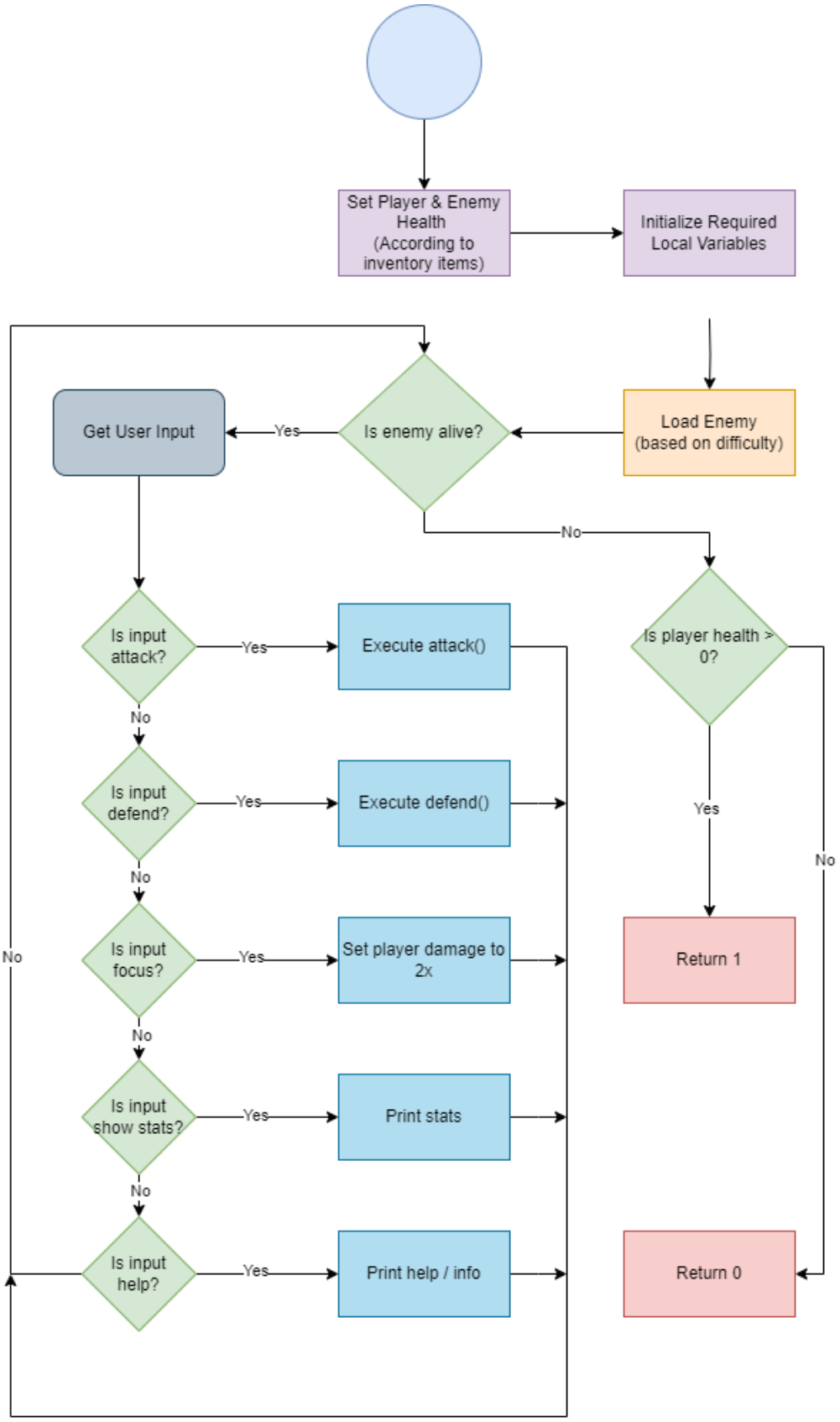
This is the general level structure in our game.

Although, each level has different events described inside the level code, they still follow this pattern for execution:



Enemy Encounter:

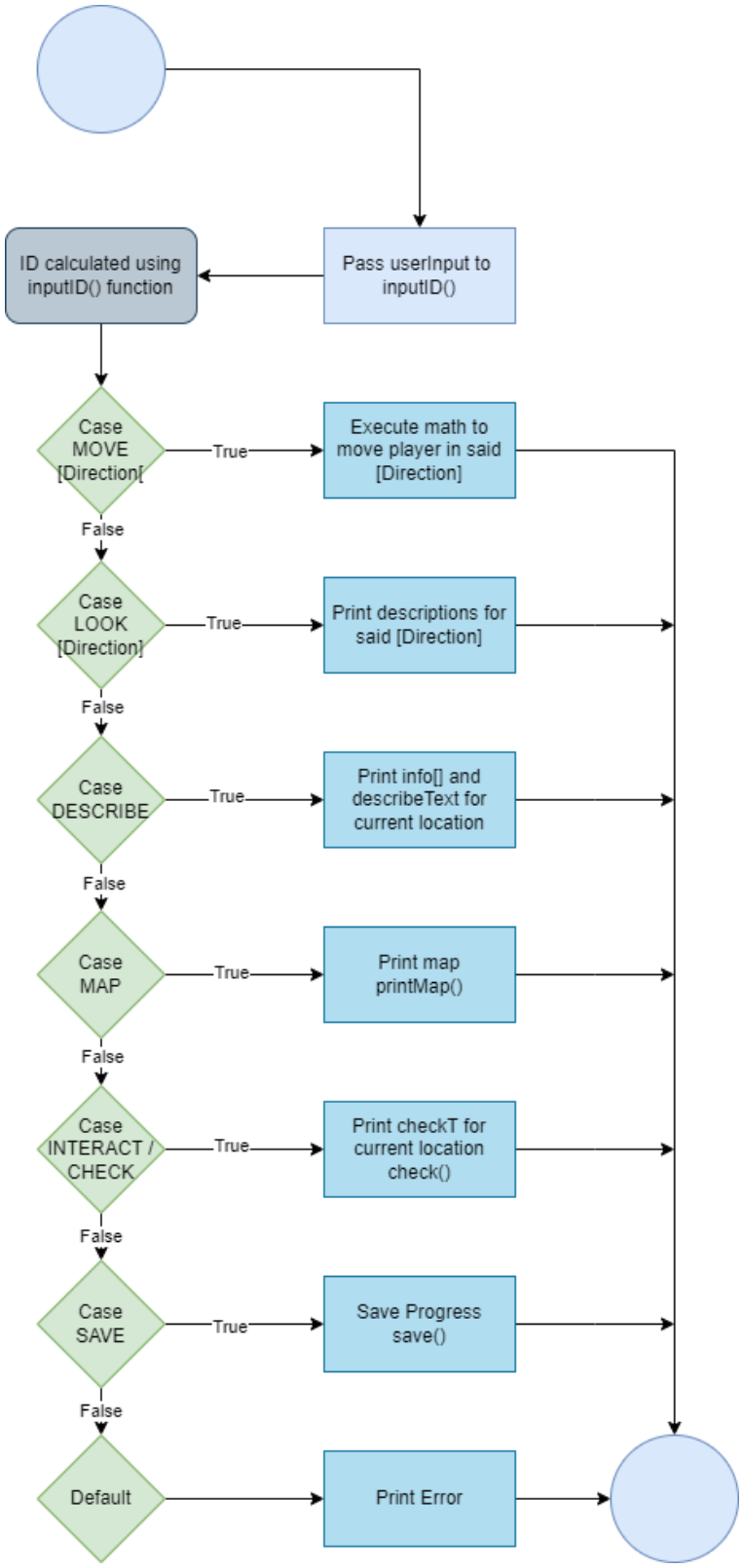
The enemy encounters follow the following flow structure:



computeInput():

Here's the flow structure of the computeInput() function.

Although there are many more cases that are checked, this flowchart contains most of the commonly used ones.



4. Technical Requirements

This section covers the technical requirements of our game.

4.1. Programming Language

In the development of "Inside," the programming language chosen is C++, leveraging specific features that contribute to the project's success.

The decision to use C++ is based on its unique capabilities and advantages in game development scenarios.

Structs for Data Bundling:

In “Inside”, structs are used for bundling multiple data types together into one single structure. Examples of structs being used include the **“text”** and **“lookText”** data type that allowed us to hold multiple strings in one data type for each location.

While normal strings could have been used for a similar effect, using structs was easier and more convenient for us as the developers as it allowed us to tie in location IDs to each individual element in the arrays, allowing easier useability.

Furthermore, the use of structs for storing inventory item descriptions was another use case where, while possible to use strings, was generally going to be far more convoluted to do so.

Enums for Symbolic Representation:

Enums provide a symbolic representation of integer values, enhancing code readability. “Inside” also uses enums to represent different inputs received from the inputID() function (which basically returns an ID based on the specific match with the input).

While using actual values instead of enums was possible, this was another case of data organization that made things easier for code readability and manageability.

Pointers:

In "Inside," pointers have been used to manage dynamic data, such as managing current level, current location on the map, directly editing map data in real time and more.

While it is possible to manage all of these tasks without pointers by declaring global variables, it is commonly bad practice to use global variables where unnecessary.

4.2. Libraries & Tools

In “Inside”, we used the **SFML (Simple and Fast Multimedia Library) Audio Module** to enhance the gaming experience.

We simply cannot understate the importance of Audio in our narrative presentation.

It allowed us to integrate background music, sound effects, and atmospheric sounds, contributing to the immersive and dynamic nature of our game.

SFML simplifies audio management, enabling seamless incorporation of auditory elements that complement the narrative and gameplay.

4.3. Platform Compatibility

Compatibility: "Inside" has been thoroughly tested and confirmed to run seamlessly on Windows operating systems including Windows 7, 8, 8.1, 10, and 11.

Optimization: The game is optimized to leverage Windows-specific features and functionalities, ensuring a smooth gaming experience for users on Windows platforms.

While "Inside" is currently optimized for Windows, we acknowledge the importance of platform diversity and will consider expanding compatibility based on user feedback.

Note: “Inside” requires the Visual C++ 2015-2022 Redistributable Package (x86) to work

5. Code

This section is focused towards providing an overview of our codebase, code organization, while providing key snippets of our code to highlight critical functionalities.

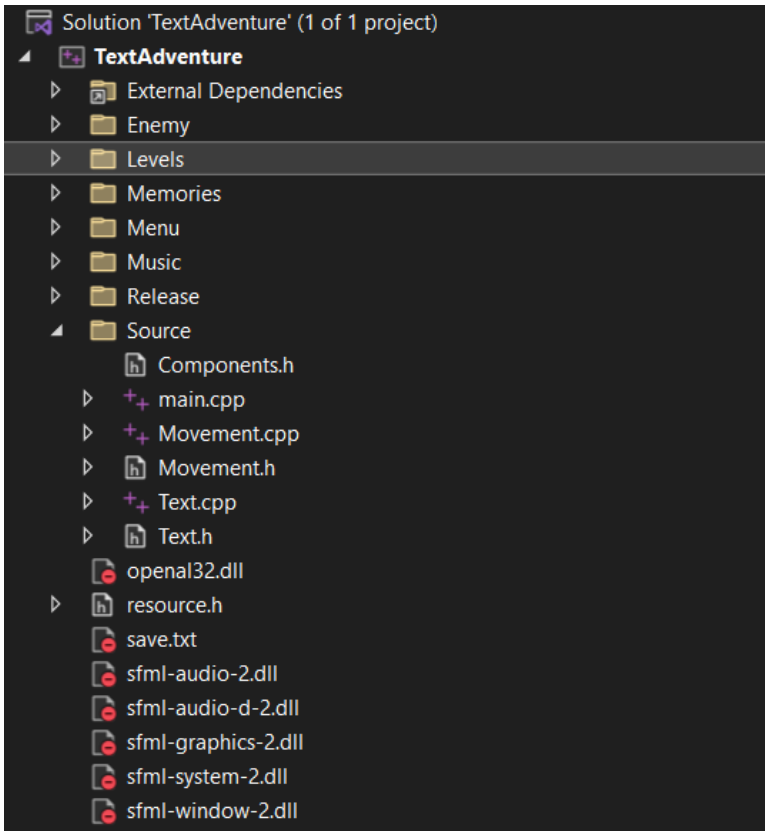
5.1. Code Organization

The project directory structure for "Inside" is organized to enhance code maintainability and readability.

Modularity: Each major functionality (game logic, levels, combat system) is encapsulated in separate modules, reducing interdependence, and easing maintenance.

Header Files: The use of header files in the promotes a clear separation of interface and implementation, aiding code readability and providing a convenient overview of available functionalities.

Below is an overview of our game directory:



Enemy/: Contains source code files for the Enemy, and Final Boss fight.

Levels/: Contains source code files for the levels as well as for other level data.

Memories/: Contains source code files for all the memories.

Menu/: Contains source code files for the Main Menu.

Music/: Contains audio files and source file for sound. (music, sound effects, etc.)

Source/: Contains source code files for the movement and text system.

5.2. Key Code Snippets

This subsection showcases excerpts from the code that highlight critical functionalities.

main():

```
int main()
{
	bool isRunning = true;

	//Initializing all the music
	initMusic();  system("cls");

	//Menu
	menu(&isRunning);


	//Game Loop
	while (isRunning)
	{
		//Cases for individual levels
		switch (level)
		{
		case 0:
			prologue();
			level_0(&level);
			break;
		case 1:
			level_1(&level);
			mem_1e();
			break;
		case 2:
			level_2(&level);
			mem_2e();
			break;
		case 3:
			level_3(&level);
			memory_4();
			break;
		case 4:
			level_4(&level);
			break;
		default:
			displayTextr("An error occured and the program had to exit.
Please try again.\n", 40);
			exit(1);
		}
	}

	return 0;
}
```

Comments:

The main game loop uses follows a mainly linear structure, by first initializing the music, followed by loading the menu, which then leads into a single switch-based system to load the levels.

This structure enhances readability and supports potential future expansions.

displayText():

```
// Function to print text. (Backbone of our game)
void displayText(std::string string, int speed, int volume)
{
	dialogue.setBuffer(buffer);
	dialogue.setVolume((float)volume);

	for (int k = 0; k < string.size(); ++k) {
		std::cout << string[k];
		if (string[k] == ',' || string[k] == '.')	Sleep(100);
		if (!sf::Keyboard::isKeyPressed(sf::Keyboard::Scan::RShift))
		{
			dialogue.play();	Sleep(speed);
		}
	}
}
```

Comments:

The display text function utilizes a for loop to loop through each individual character in a string and prints it with a delay in-between.

We used this system for printing mainly to give that RPG-like feel to our game but also to make the text more digestible and easier to read.

updateMap(), printMap() & clearMap():

```
//Updating the map to reveal more of the map
void updateMap(int x, int y){
	for (int k = 0; k < 5; k++)
	{
		for (int l = 0; l < 5; l++)
		{
			tempMap[y - 2 + k][x - 2 + l] = 1;
		}
	}
}

//Clearing the map (Used at the start of each level)
void clearMap(){
	for (int i = 0; i < 20; i++)
		for (int j = 0; j < 20; j++)
			tempMap[i][j] = 0;
}
```

```
//Printing the map.
void printMap(int map[][20][20], int location, int size[], int level){
	for (int i = 0; i < size[level]; i++)
	{
		for (int j = 0; j < size[level]; j++)
		{
			if (map[level][i][j] != location)
			{
				if ((level == 1 && j == 0))continue;
				else if (map[level][i][j] == 0 && tempMap[i][j] == 1)
	std::cout << "\033[1;90m" << "X" << "\033[1;0m";
				else    std::cout << "\033[1;90m" << " " << "\033[1;0m";
			}

			else
				std::cout << "\033[1;31m" << "X" << "\033[1;0m";
			std::cout << " ";
		}
		if ((level == 1 && i >= 15))
			break;
		std::cout << std::endl;
	}

	std::cout << std::endl;
	displayText("You are here:  ", 50); displayTextr("X\n\n", 50);
}
```

Comments:

The map functions use nested loops to iterate over the rows and columns of the map at the specified level.

printMap() checks if the current cell's value in the map is equal to the player's location. If true, it prints a red "X" to represent the player's position; otherwise, it prints a space.

The function uses the tempMap variable to check if it should print a location. This variable is updated as the player unlocks more of the map for printing via the updateMap() function and is cleared using the clearMap() function.

Movement Part of computeInput():

```
//Movement
case NORTH:
	if (map[level][*y - 1][*x] <= 0)
		displayText(moveFail[choiceQ], 50);
	else
	{
		*y = *y - 1;
		while (map[level][*y][*x] == 1)
		{
			updateMap(*x, *y);
			*y = *y - 1;
		}
		*isChoice = false;
	}
	break;
```

Comments:

This segment of the computeInput is for the case where the user inputs “move north”.

The function checks if the move to said location is possible or not (determined by the location ID being greater than 0) and acts accordingly by updating the x, y coordinates or by printing an error.

We chose this technique as it is a simple yet effective way to traverse the world map.

save() & load():

```
int save(){
	std::ofstream outf{ "save.txt" };

	// If we couldn't open the input file stream for reading
	if (!outf){
		displayTextr("Could not save...\n", 50);
		return 1;
	}

	// Save current value for each (valuable) variable into the save file
	outf << std::to_string(level) << "\n";

	if (hasKey)
		outf << "1\n";
	else
		outf << "0\n";

	// More variables that are stored, but skipped for this demonstration
	outf.close();
	return 0;
}

int load(){
	int line = 0;
	std::ifstream inf{ "save.txt" };

	if (!inf){
		displayTextr("Uh oh...a save file was not found...\n\n", 50);
		return 1;
	}

	while (inf){
	// Read stuff from the file and update the in-game variables accordingly
		std::string strInput;
		inf >> strInput;
		line++;

		switch (line){
		case LEVEL:
			level = stoi(strInput);
			break;

		default:
			break;
		}
	}
	inf.close();
	return 0;
}
```

6. Challenges Faced

In the course of developing this program, our team encountered various challenges inherent in the software development process.

These challenges manifested in diverse forms, encompassing both technical intricacies and design considerations.

6.1. Technical Challenges

These were some of the major technical challenges we faced during the development:

User Interface (UI) Design:

Challenge: Creating an intuitive and aesthetically pleasing text-based UI that effectively communicates game information.

Resolution: Iteratively refined the UI design, gathered user feedback, and conducted usability testing to enhance the overall user experience.

Integration of Audio Module (SFML):

Challenge: Integrating the audio component from the SFML library seamlessly into the project.

Resolution: Followed SFML documentation, incorporated necessary dependencies, and conducted extensive testing to ensure the proper functioning of audio elements within the game.

Map & Text Storage:

Challenge: Translating our in-house map designs into the game engine and storing level descriptions, dialogue and more, to allow us to easily store our writing inside our code.

Resolution: Implemented a 3D array-based movement system that dynamically calculates player position. Storing level descriptions and dialogue in a user defined data type to allow easier implementation of said text.

Dynamic Map Printing System:

Challenge: Keeping track of player movement and dynamically printing only those parts of the map that the player has traversed.

Resolution: Added additional arrays specifically to store data regarding where the player has been and incorporating it into the map printing system.

Maintainability & Readability:

Challenge: During development, it is easy to lose track of readability, as we had experienced ourselves. It was especially difficult to understand code written by the other members of our group.

Resolution: Increased communication within group members and proper implementation of general coding practices. Adding comments was also a huge help.

Quality Assurance:

Challenge: It was especially difficult to keep track of bugs and glitches that popped up during the development of "Inside".

Resolution: Making a list of bugs encountered during playthroughs. Hiring QA Testers from other groups. This also helped us incorporate player feedback more effectively.

6.2. Design Challenges

The development of "Inside" was far more design focused than originally expected.

Our team faced multitude of challenges while developing a compelling narrative, gameplay and User Interface for the final product to be as enjoyable as it is today.

Narrative Cohesion:

Challenge: Ensuring that our narrative is engaging while seamlessly integrating player choices into the equation was a truly monumental task.

Resolution: We conducted extensive storyboarding while iteratively refining the narrative branches based on playtesting and player feedback. We also had to emphasize key story beats to maintain coherence while allowing for diverse player paths.

Player Agency and Endings:

Challenge: Balancing player agency in decision-making while maintaining a structured narrative leading to meaningful endings.

Resolution: Designed decision points with impactful consequences, ensuring they contributed to character development and overall story arcs. Iteratively adjusted the branching paths to maintain coherence in the narrative outcomes.

Visual Representation in a Text-Based Game:

Challenge: Effectively conveying the game's environment and key elements without traditional graphics is a task that truly makes you understand the importance of in-game graphics and the work that goes into designing such an experience.

Resolution: Implemented color-coded text to enhance visual representation while constantly testing various ways to express environmental fidelity through text alone. Conducted playtesting to refine visual cues and ensure clarity in conveying the game world to the player.

User Interface (UI) Clarity:

Challenge: Designing a text-based UI that communicates essential information without overwhelming the player.

Resolution: Iteratively refined UI elements, considering typing speed, colour, and text formatting. Incorporated directional hints and contextual prompts to guide players without disrupting immersion.

Pacing and Engagement:

Challenge: Maintaining a balance between exploration, puzzle-solving, and combat encounters to sustain player engagement. This task is especially important for a game with the only means of player engagement being text.

Resolution: Conducted playtesting to analyse player preferences while simultaneously adjusting the pacing accordingly. Iteratively tweaked the placement and complexity of puzzles and combat scenarios to ensure a satisfying and immersive gameplay experience.

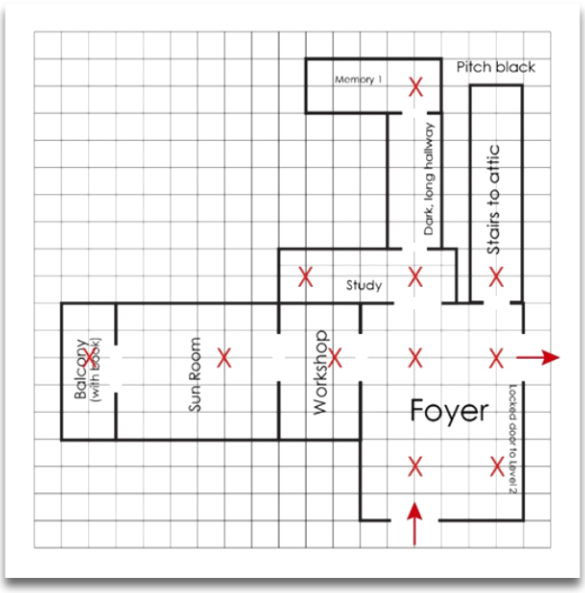
Level Design:

Challenge: Ensuring that a level is designed with player preferences in mind. It is essential that the player does not feel underwhelmed with lack of engagement, while also making sure that the level essentially guides the player subconsciously.

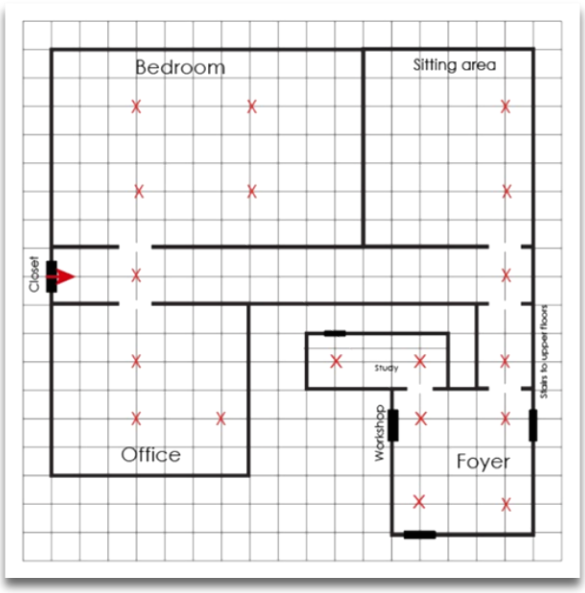
Resolution: Went through multiple iterations of level designs and incorporated player feedback into dialling the amount of content within each level.

6.3. Level Design Iterations

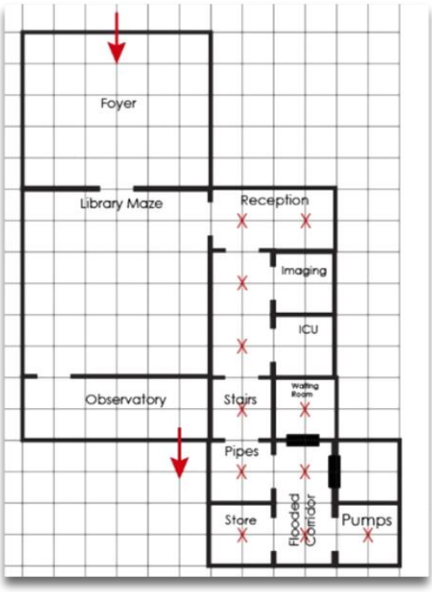
Level 1 Map (Illustration)



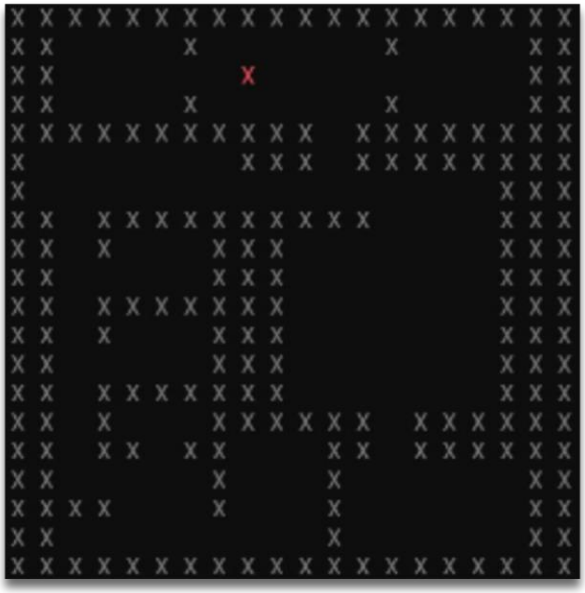
Level 2 Map (Illustration)



Old Level 3 (Illustration)



Current Level 3 (In-Game)



For Level 3, we had to completely change the orientation of the map in order to properly fit into our 20x20 grid-based system. Moreover, because it was our most ambitious level design, we designed more rooms to fully utilize all the space we had available.

Similar iterations were made for the other levels, though not to the same extent.

7. Learning Objectives Achieved

The development of "Inside" significantly contributed to the enhancement of programming proficiency for the entire team. Several key areas saw notable improvement:

7.1. Programming Skills

Algorithmic Problem-Solving:

Designing and implementing a dynamic map traversal algorithm, combat systems, and decision-based narratives required intricate problem-solving. Overcoming these challenges refined the team's ability to devise efficient and effective algorithms.

Integration of External Libraries:

Integrating the audio component from the SFML library into the project enhanced skills in linking external libraries, understanding documentation, and troubleshooting integration issues. This experience will be valuable for future projects involving third-party libraries.

User Interface Design:

Crafting an intuitive and visually appealing text-based user interface honed skills in UI design. Balancing aesthetics with functionality and iteratively refining the UI based on user feedback contributed to improved design proficiency.

Testing and Quality Assurance:

Implementing thorough testing procedures at various development stages enhanced skills in identifying and rectifying bugs, ensuring code reliability, and validating the functionality of different game components.

Documentation and Communication:

Documenting code, design decisions, and project details improved communication within the team and allowed for future maintainability.

Our team developed a habit of maintaining clear, comprehensive documentation throughout the development process.

7.2. Problem Solving

Problem-solving played a central role in the development of "Inside," fostering a dynamic and adaptive approach to overcome technical and design challenges.

Several instances highlight the development and application of problem-solving skills have already been mentioned in the *Challenges Faced* section of the document. (See Page 29)

We can say with certainty that the development of "Inside" showcased the team's ability to approach challenges with creativity, research, and adaptability.

Each instance of problem-solving contributed to a more refined and robust game, highlighting the iterative nature of development and the importance of dynamic problem-solving skills in the software development process.

7.3. Collaboration & Communication

Collaboration and effective communication were pivotal aspects of the "Inside" development, contributing significantly to the project's success.

The collaborative efforts primarily focused on narrative writing, map design, level design, and quality assurance.

Narrative Writing:

Our team actively collaborated on crafting the game's narrative, ensuring a cohesive and engaging storyline. Frequent brainstorming sessions allowed for the exploration of various plotlines, character arcs, and thematic elements.

Clear and concise discussions facilitated the alignment of the narrative with the overall vision for the game.

Map & Level Design:

Map and level design involved collaborative efforts to create diverse and immersive game environments.

Both of our team members collaborated on sketching layouts, defining key locations, and establishing the overall structure of the in-game world.

We articulated design concepts, shared visual references, and iteratively refined ideas based on input from each other.

Bug Testing:

Bug testing was a specifically difficult phase of our project, where we systematically identified, reported, and resolved defects in our game logic, which occasionally left both of us in dismay.

This involved extensive playtesting to uncover issues related to gameplay mechanics, UI, and overall system stability.

General Collaborative Dynamics:

Regular Meetings: Scheduled regular team meetings facilitated updates on individual tasks, addressed challenges, and ensured both of us were aligned with our goals. This regular communication strengthened team cohesion and allowed for quick adaptation to changing requirements.

Shared Responsibilities: Tasks related to coding, design, and testing were often collaborative efforts, with work divided between both of our members contributing their specific expertise to different aspects of the project. This shared responsibility fostered a sense of ownership and accountability while also enhancing parts of the project that otherwise would've been sub optimal.

8. Conclusion

8.1. Summary

The development of "Inside" has been a comprehensive journey, encompassing various facets of software development and collaboration. Key points discussed in the report include:

Project Overview:

"Inside" is a text adventure game featuring a dynamic narrative, challenging puzzles, and a built-from-the-ground-up combat system.

The game follows the protagonist, Nell, on a journey of self-discovery through an ever-evolving house, unveiling memories and facing the past.

Project Objectives:

The primary objectives include creating an immersive text adventure experience, implementing innovative gameplay mechanics, and addressing technical and design challenges.

Target Audience:

The game is designed for players who appreciate narrative-driven experiences, challenging puzzles, and a unique combat system. The target audience spans those with a penchant for exploration and decision-making.

Technical Challenges:

UI Design, Map & Text Storage, audio integration, dynamic map printing, quality assurance and code maintainability were key technical challenges addressed during development.

Design Challenges:

Narrative cohesion, player agency and endings, visual representation in text format, UI clarity, pacing and engagement, and level design were pivotal design challenges met through iterative development.

Programming Skills Enhancement:

The development of "Inside" significantly enhanced programming proficiency, emphasizing algorithmic problem-solving, integration of external libraries, UI Design, Testing and Documentation.

Collaboration and Communication:

Collaboration was integral to narrative writing, map and level design, bug testing, and general project dynamics.

Effective communication skills played a crucial role in expressing ideas, providing feedback, and ensuring a cohesive team environment.

Conclusion:

The development of "Inside" not only resulted in a unique and engaging text adventure game but also served as a platform for enhancing technical and design skills, fostering collaboration, and refining communication abilities within a team setting.

The learning objectives set for this project were not only met but exceeded.

Team members developed proficiency in various programming aspects, honed problem-solving skills, and gained valuable experience in collaborative software development.

8.2. Future Enhancements

Expanded Storyline:

Introduce additional narrative branches and story arcs to enhance replay ability and cater to diverse player choices.

Explore branching paths based on secondary character interactions, offering deeper insights into Nell's relationships and past.

Dynamic Environments:

Implementation dynamic map generation system to introduce more varied environments, challenges, and hidden areas.

GUI Implementation:

Exploring the use of a graphical user interface, including the incorporation of simple graphical elements to complement the text-based interface.

Cross-Platform Compatibility:

Porting the game code to run on Linux and Mac OS, ensuring a seamless experience on these platforms.

Also Exploring the possibility of porting the game to additional platforms, such as mobile devices, to reach a broader audience.

Advanced Combat:

Expanding the combat system with a wider array of enemies, each influencing gameplay in distinct ways.

Introducing various strategic elements such as skill development, more weapon choices, as well as an overhauled command-based system adding depth to combat encounters.

9. Acknowledgements

The successful development of "Inside" would not have been possible without the invaluable contributions of various resources, tutorials, and individuals. We extend our sincere gratitude to:

SFML Documentation and Community:

The SFML documentation and community for comprehensive resources on integrating the audio component, troubleshooting, and optimizing audio features within the game.

Programming Tutorials and Forums:

Online programming tutorials, forums, and communities (like GeeksForGeeks, learncpp.com, Stack Overflow, etc.) that served as constant sources of learning and problem-solving throughout the development journey.

Beta Testers and User Feedback:

Beta testers and users who dedicated their time to playtesting "Inside" and providing valuable feedback. Your insights have been instrumental in refining the game.

Cicada Sirens & 1000 Eyes:

Special thanks to Cicada Sirens & 1000 Eyes for creating the atmospheric soundtrack for the video game “Signalis”. Using the soundtrack has been paramount to creating the immersive experience of "Inside."

ChatGPT:

Lastly, we thank ChatGPT for allowing us to enhance our documentation with additional information, improved formatting as well as refined vocabulary.

Note: Full Credits for the game are available inside the Credits document & in the game.

10. References

SFML Documentation:

Simple and Fast Multimedia Library (SFML) documentation. Available at:

SFML Documentation (SFML / Learn / 2.6.1 Documentation)

Signalis Original Soundtrack:

The official soundtrack for “Signalis”. Available at:

https://signalis-ost.bandcamp.com/album/signalis-original-soundtrack