

**NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY**

School of Electrical Engineering and Computer Sciences

# **CS110 Fundamentals of Computer Programming**

**Semester Project Report**

**SUBMITTED TO: Ms. Momina Moestesm**

**SUBMITTED BY: Anas Rehman , 464092**

**Muhammad Ahmad Raza , 466747**

**CLASS + SECTION: BESE-14A (2k23)**

**Date of Submission: 24th December, 2023**

**INTRODUCTION:**

**Project Overview**

**Brief Discription of the Game:**

"Agent 47" is an immersive text-based game developed in C++. Players take on the role of a daring protagonist navigating through a dynamic map, striving to reach a predetermined score while avoiding the relentless pursuit of the mafia. Navigate the city map, accumulating points to achieve the required score before being caught by the pursuing mafia. There are three levels in the game easy, medium and hard. The required score depend on the difficulty level. Upon reaching the score for the level, a final showdown ensues, testing the player's luck and strategic choices in a battle against the mafia. The game combines strategy, chance, and decision-making as players face off against the criminal underworld.

**Purpose and Objective of the Game:**

The purpose of "Agent 47" is to provide players with an engaging and thrilling text-based gaming experience that combines strategic decision-making, chance, and a dynamic narrative. The game aims to immerse players in a suspenseful adventure within a city filled with challenges and the looming threat of a pursuing mafia. Through a mix of exploration, resource management, and a final showdown, players will be challenged to outsmart the criminal underworld and emerge victorious.

The primary objective of the game is for the player-controlled protagonist to navigate the city map, accumulating the required score while avoiding capture by the relentless mafia. The journey is fraught with obstacles, random events, and the constant threat of being caught. Once the protagonist reaches the target score, the objective shifts to defeating the mafia in a final showdown. The player must strategically choose actions (attack, defend, or heal) during the battle, relying on random outcomes to determine the success of each move. The ultimate goal is to deplete the mafia's power before the protagonist's power reaches zero, securing victory and proving mastery over the city's criminal underworld. Success requires a combination of careful planning, adaptability, and a bit of luck, making each play through a unique and exciting challenge.

**Target Audience:**

The target audience for "Agent 47" is likely individuals who enjoy text-based games, strategy, decision-making, and a suspenseful narrative. Here's a breakdown of the potential target audience:

* Players who appreciate games that rely on storytelling, character interactions, and decision-making rather than graphical elements.
* Individuals who enjoy games that require strategic planning, resource management, and critical thinking to navigate challenges and overcome opponents.
* Players who enjoy games where their choices impact the outcome and direction of the story, creating a sense of agency and personalization.
* Those who appreciate a well-crafted narrative with suspenseful elements, unexpected twists, and a dynamic storyline.

**GAME CONCEPT:**

**Storyline**

**Game's Plot and Setting:**

“Agent 47” unfolds in the gritty and mysterious setting of a sprawling metropolis dominated by an intricate network of crime syndicates. Players step into the shoes of a daring protagonist with a troubled past, striving for redemption and justice. The protagonist must accumulate a predetermined score by undertaking the required tasks, solving missions and outsmarting rival factions, all while avoiding the ever-watchful eye of pursuing mafia. Tensions escalate as the protagonist edges closer to the elusive target score, setting the stage for a final showdown with the powerful mafia that controls the city.

The game is set in a sprawling, densely populated metropolis with a diverse range of environments. Within this city, players navigate through a labyrinthine map, encountering diverse locations such as seedy bars, high-stakes casinos, abandoned warehouses, and covert hideouts. Each area presents unique challenges, opportunities, and characters that contribute to the overarching narrative. The visual imagery is conveyed through vivid text descriptions, allowing players to immerse themselves in the gritty details of the city's underworld. As the protagonist inches closer to the required score, the tension escalates, leading to a climactic final showdown with the mafia in a secret location. The setting is designed to evoke a sense of urgency and suspense, providing players with a gripping and memorable gaming experience in the heart of a city consumed by crime and corruption.

**Main Character and Player Role**

**Main Characters:**

1. **Agent 47:**
   * ***Attributes:*** A mysterious and skilled individual with a troubled past, known for their stealth, strategy, and combat abilities. Motivated by a desire for redemption, justice, and the exposure of the city's criminal conspiracies.
2. **The Pursuing Mafia Leader:**
   * ***Attributes****:* A cunning and ruthless leader of the city's most powerful mafia organization. Known for his strategic mind and relentless pursuit of anyone who threatens the mafia's interests. The mastermind behind the criminal web that ensnares the city.

**Player's Role in the Adventure:**

The role of the player includes:

* Exploration and Navigation
* Decision-Making and Dialogue Choices
* **Strategic Planning for the Final Showdown**
* Adaptability to Random Events

**Gameplay Mechanics**

**Interaction between the player and the Game:**

The interaction between the player and "Agent 47" is characterized by immersive decision-making, strategic planning, and dynamic engagement with the game's narrative elements. Here's a breakdown of the key aspects of player interaction:

* Text-Based Decision-Making
* Navigation and Exploration
* Mission Undertaking and Challenges
* Strategic Planning for the Final Showdown
* Adaptation to Random Events
* Score Tracking and Progression

The interaction between the player and the game is designed to be dynamic, engaging, and responsive to the player's choices. The text-based format allows for a rich narrative experience, encouraging players to invest in the protagonist's journey and experience the consequences of their decisions in a suspenseful and challenging urban adventure. Top of Form

**Key Decisions**

**Navigation Choices:**

Decide where the protagonist goes on the dynamic city map, choices impact the accumulation of points, encounters with the mafia.

**Level Undertaking:**

Choose which level to undertake, each with its own set of challenges and difficulty.

**Strategic Planning for the Final Showdown:**

Prepare for the ultimate confrontation with the mafia by strategically choosing the protagonist's actions in the final showdown. Decide whether to attack, defend, or heal during the battle, with outcomes determined by random events and the current state of the protagonist's and mafia's power.

**Challenges:**

**Mafia Pursuit:**

Navigate the city while avoiding the relentless pursuit of the mafia. The challenge is to accumulate the required score before being caught. The mafia's movements are unpredictable, and players must carefully plan their route to maximize points while minimizing the risk of encounters.

**Random Events:**

Encounter unexpected challenges and opportunities through random events that can alter the course of the game.

**Resource Mangement:**

Manage the protagonist's health. The balance between resource use and conservation is crucial for survival and success.

**Potential Outcomes:**

**Victory:**

Successfully reach the required score and defeat the mafia in the final showdown.

**Defeat:**

Fail to accumulate the required score before being caught by the mafia, resulting in the protagonist's capture or demise.

**DESIGN:**

**Architecture**

**Overall Structure of the Game:**

**Game Initialization and Introduction (main function):**

* The game starts with an introduction, providing instructions and background information about the game.
* Players can choose the difficulty level (easy, normal, or hard) to determine the scoring requirements for encountering the mafia.
* The game utilizes a simple console-based interface.

**Loading Screen (loading function):**

* A loading screen is display to create anticipation before the game starts.
* The loading function uses console graphics to simulate a loading bar.

**Map Display and Movement (ShowMap, gotoxy, Entity, walk, target):**

* The game map is display using a 2D array of characters.
* The player (Agent 47) and the mafia (M) are represent on the map.
* The Entity structure handles the position and movement of characters.
* The gotoxy function is use to position the cursor on the console.

**Pathfinding Algorithm (BFSArray, AddArray, FindPath):**

* The game uses a basic pathfinding algorithm (Breadth-First Search) to calculate the shortest path from the mafia to the player.
* The algorithm helps the mafia move strategically toward the player.

**Menu and Difficulty Selection (menu function):**

* Players can choose the difficulty level (easy, normal, or hard) before the game starts.
* The menu function handles user input for difficulty selection.

**Game Loop (while loop in main):**

* The game runs in a loop, allowing the player to move through the map, accumulate points, and encounter the mafia.
* The player can move using arrow keys and must avoid being caught by the mafia.
* The loop also checks for user input to exit the game.

**Encounter Function (encounter function):**

* When the player accumulates enough points, an encounter with the mafia is trigger.
* The encounter function prints a story and prepares the player for the battle.

**Battle Simulation (simulateBattle function, response function, checker function):**

* The game enters a battle mode where the player can choose to attack, defend, or heal using random functions.
* The simulateBattle function initiates the battle and interacts with the player's choices.
* The response function simulates the effects of the player's actions.
* The checker function ensures that health values do not go below zero.

**Exit Function (exit function):**

The exit function is called when the player chooses to exit the game, displaying a closing message.

This summarizes the overall structure of the game structure. The game involves map navigation, encounters, and a turn-based battle system, providing an interactive and challenging experience for the player.

**Structures:**

**Entity Structure:**

* Represents an entity's position on the map.
* Tracks the x and y coordinates of the entity.

**walk Structure:**

* Used in the BFS algorithm for pathfinding.
* Stores information about a position in the map during the search.

**target Structure:**

* Represents a target position.
* Used in the BFS algorithm to store the path.

**Functions:**

**main Function:**

* Entry point of the program.
* Handles the game initialization, introduction, and difficulty selection.

**loading Function:**

* Simulates a loading screen using console graphics.

**ShowMap Function:**

* Displays the game map on the console.

**gotoxy Function:**

* Positions the cursor on the console.

**FindPath Function:**

* Implements Breadth-First Search (BFS) algorithm for pathfinding.

**menu Function:**

* Manages the game menu, allowing the player to choose the difficulty level.

**simulateBattle Function:**

* Initiates the battle mode, where the player can choose to attack, defend, or heal.

**response Function:**

* Simulates the effects of the player's chosen action in the battle.

**checker Function:**

* Checks if health values go below zero, signaling the end of the game.

**encounter Function:**

* Triggers an encounter with the mafia, displaying a story.

**exit Function:**

* Displays an exit message when the player chooses to exit the game.

**Relationships:**

**BFS Algorithm and Movement:**

* The BFS algorithm (FindPath) is used to calculate the shortest path from the mafia to the player.
* The path is stored in the walk\_queue vector.
* The player's movement is influenced by the calculated path.

**Game Loop and Actions:**

* The game runs in a loop **(while(running))** to allow the player to move through the map and accumulate points.
* Actions such as movement, encounters, and battles are determined within this loop.

**Encounter and Battle:**

* When the player accumulates enough points, an encounter with the mafia is triggered (**encounter** function).
* This leads to the initiation of a turn-based battle (**simulateBattle** function).
* The player's choices during the battle are handled by the **response** function.

**Health Management:**

* The **checker** function ensures that health values do not go below zero during the battle.

**Menu and Game Initialization:**

* The **menu** function handles the selection of the game difficulty level during the game initialization phase.

**Loading and Visuals:**

* The **loading** function provides a visual loading screen to enhance the player experience.

**User Interaction with the Game:**

**Main Menu Interaction:**

* The game starts with a main menu where the player is prompted to choose an option - either to start the game or exit.
* The user enters their choice using the keyboard ('S' to start, 'E' to exit).

while (true) {

cout << "S --> Start" << endl;

cout << "E --> Exit" << endl << endl;

cout << "Select your option : ";

cin >> enter;

**Game Mode Selection:**

* After selecting to start the game, the player is prompted to choose a difficulty level (Easy, Normal, or Hard).
* The user enters their choice using the keyboard ('E', 'N', or 'H').

cout << "H -> Hard\nN -> Normal\nE -> Easy\nQ -> Quit\n\nSelect game mode : ";

cin >> diffi;

**Gameplay Interaction:**

* During gameplay, the player moves the protagonist using arrow keys (UP, DOWN, LEFT, RIGHT).
* The protagonist interacts with dots on the map to gain points, which are incremented when the protagonist moves over a dot ('.').  
  The player can press the ESC key to quit the game.

if (GetAsyncKeyState(VK\_UP))

if (GetAsyncKeyState(VK\_DOWN))

if (GetAsyncKeyState(VK\_LEFT))

if (GetAsyncKeyState(VK\_RIGHT))

if (GetAsyncKeyState(VK\_ESCAPE))

**Encounter and Battle Interaction:**

* When the player accumulates enough points, an encounter with the mafia is triggered.
* The player is then prompted to choose actions during the battle: Attack ('A' or 'a'), Defend ('D' or 'd'), or Heal ('H' or 'h').
* The user enters their choice using the keyboard.

cout << "A --> Attack" << endl;

cout << "D --> Defend" << endl;

cout << "H --> Heal" << endl << endl << endl;

cout << "Your current hp is : " << hp << endl;

cout << "Current hp of Mafia is : " << mhp << endl;

cout << "Click your response : ";

cin >> choice;

**Battle Outcome Interaction:**

* The outcome of the battle is display, showing the damage dealt to the mafia and the player, as well as any healing.
* The player is inform about their current and remaining health.

cout << "You give a damage to mafia of : " << tempmhp - mhp << endl;

cout << "You got damage of : " << temphp - hp << endl;

**Game Over Interaction:**

* If the player's health or the mafia's health drops below zero during the battle, the game ends.
* A game over message is display, indicating whether the player won or lost.

cout << "You Lose and your score is : " << pts << endl;

**Exit Interaction:**

* If the player chooses to exit the game, an exit message is displayed.

void exit() {

// Display exit message

}

**Sample Dialogues and Interactions:**

**Main Menu Interaction:**

+=======+

| Agent 47 |

+=======+

Hey... Agent, you are in a maze city.

You are a secret agent, and you have to kill MAFIA.

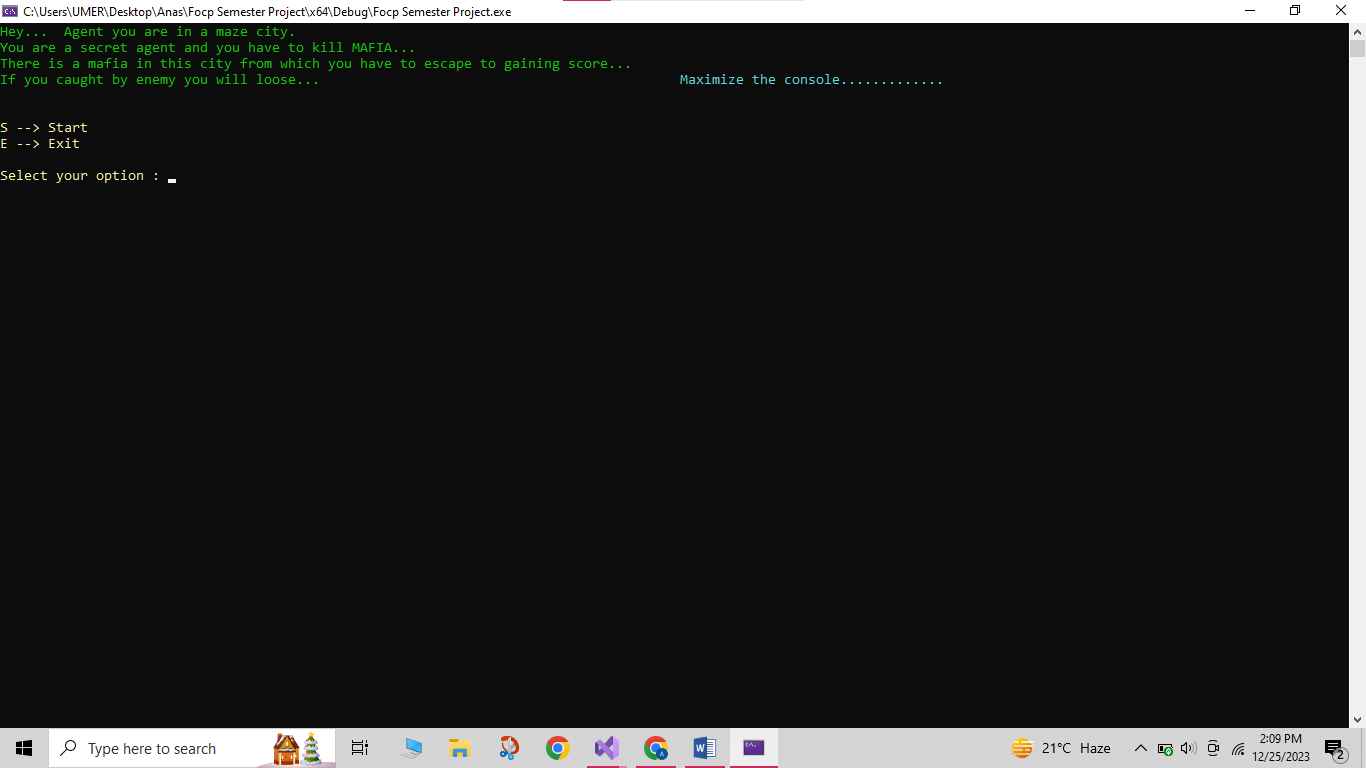
There is a mafia in this city from which you have to escape to gain a score.

If you get caught by the enemy, you will lose.

S --> Start

E --> Exit

Select your option: S

****

**Game Mode Selection:**

Instructions:

1. Arrow Keys to move.

2. Eat the dots produced by the enemy to gain points

3. Don't get caught by the Enemy

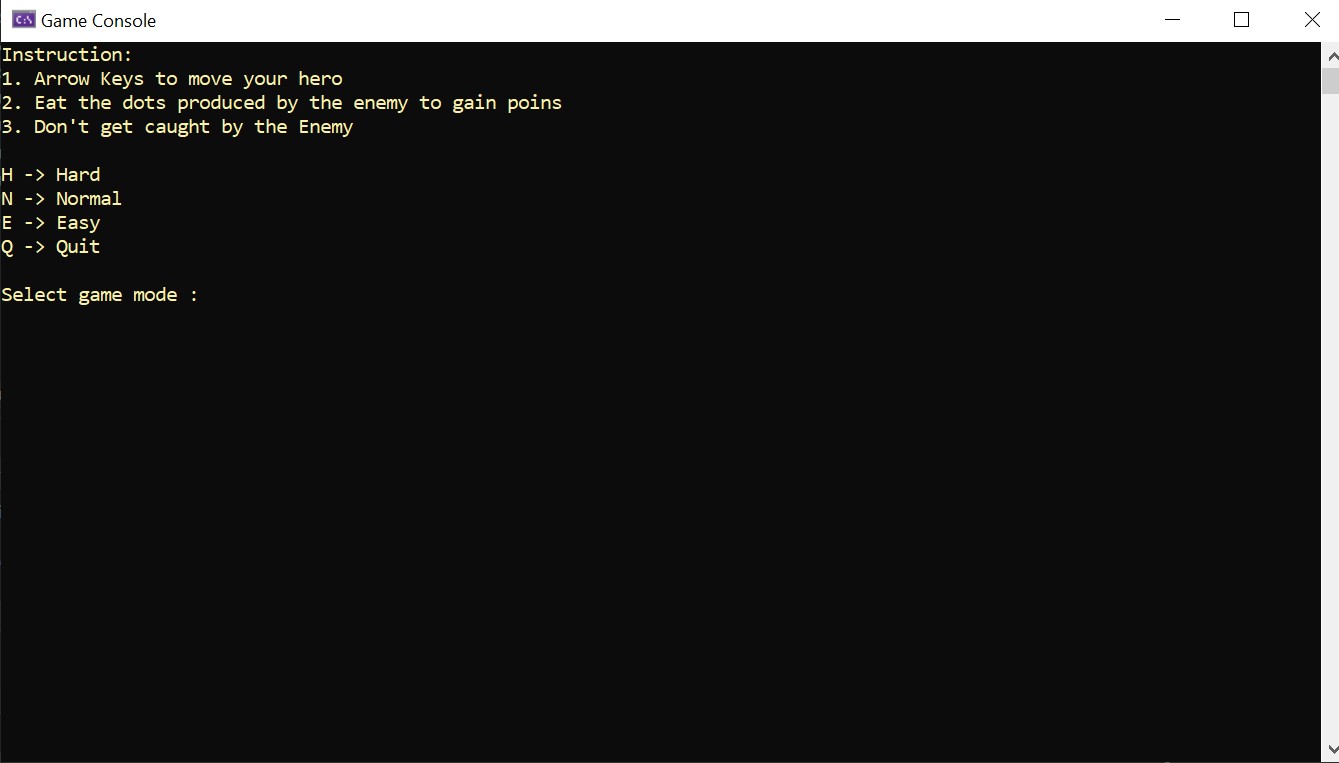
H -> Hard

N -> Normal

E -> Easy

Q -> Quit

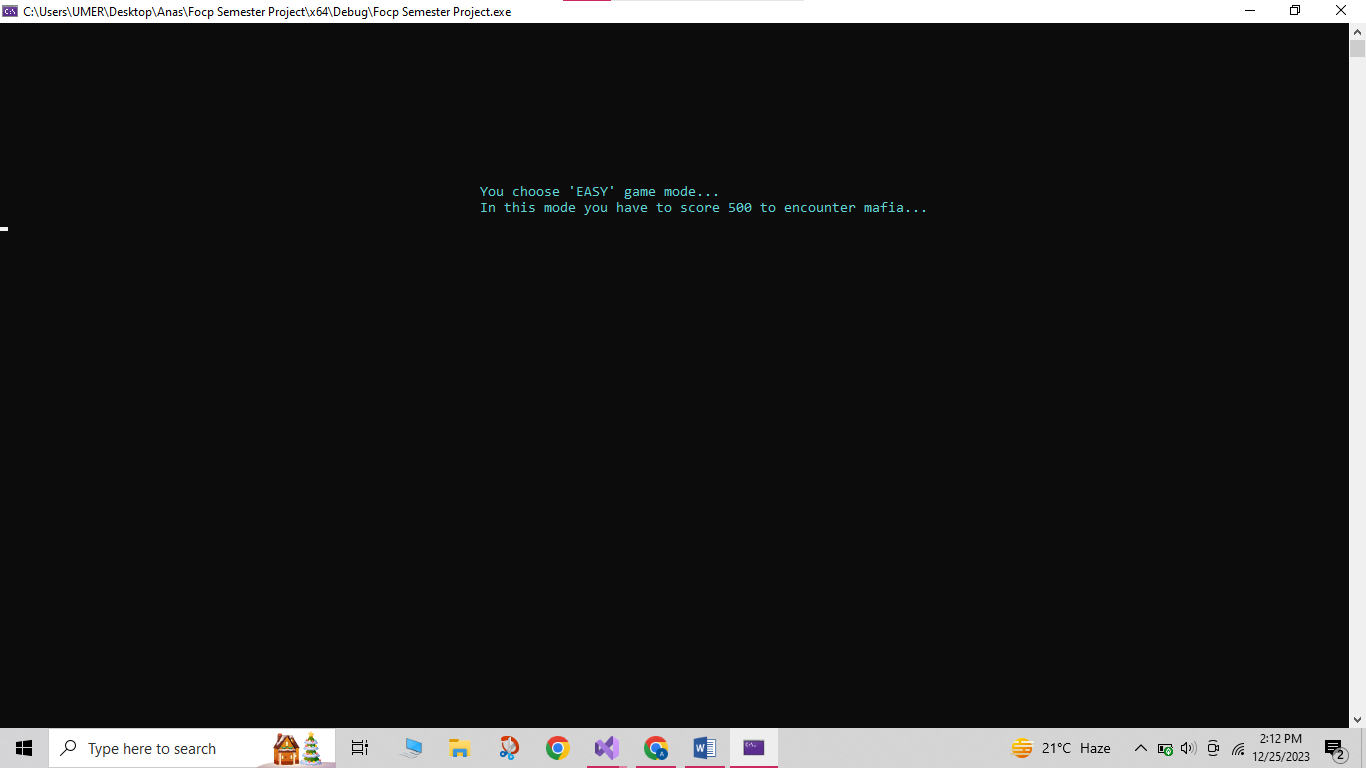
Select game mode: N



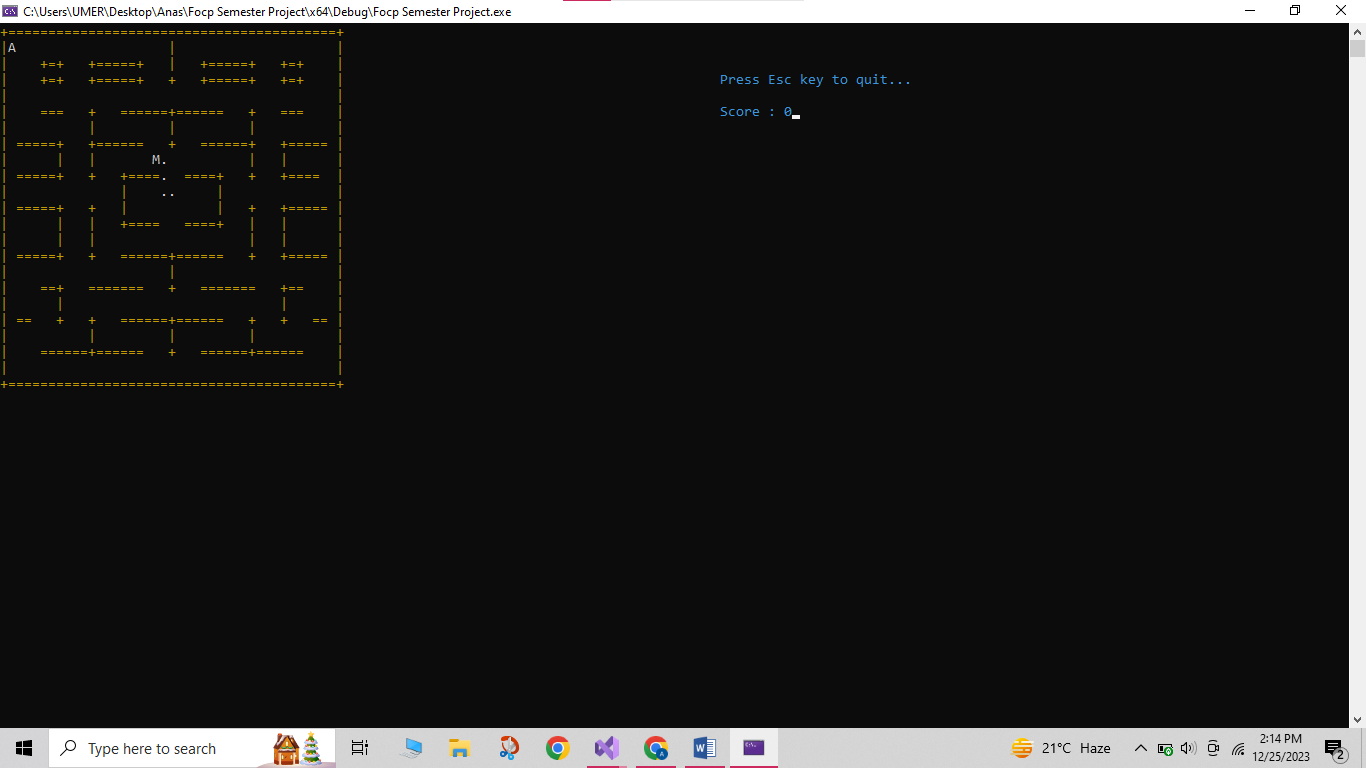
Loading....

Press Esc key to quit...

Score: 0



**Gameplay Interaction:**



**Encounter and Battle Interaction:**

You encountered MAFIA!

Instructions:

Enter the Character as your response.

You can heal only when your health is less than 50.

When you heal, the mafia will also get healed...

Choose your ACTION...

A --> Attack

D --> Defend

H --> Heal

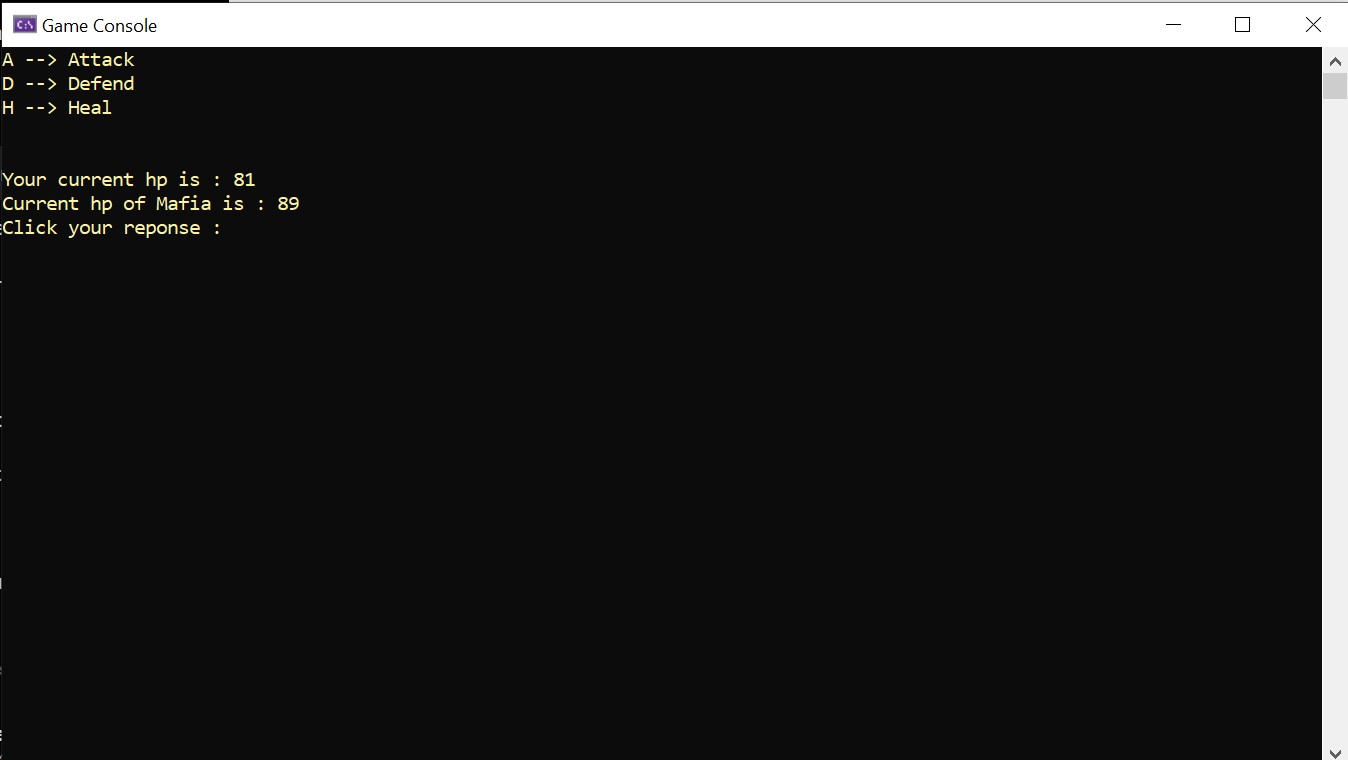
Your current hp is: 58

Current hp of Mafia is: 25

Click your response: A

You give damage to the mafia of: 18

You got damage of: 5



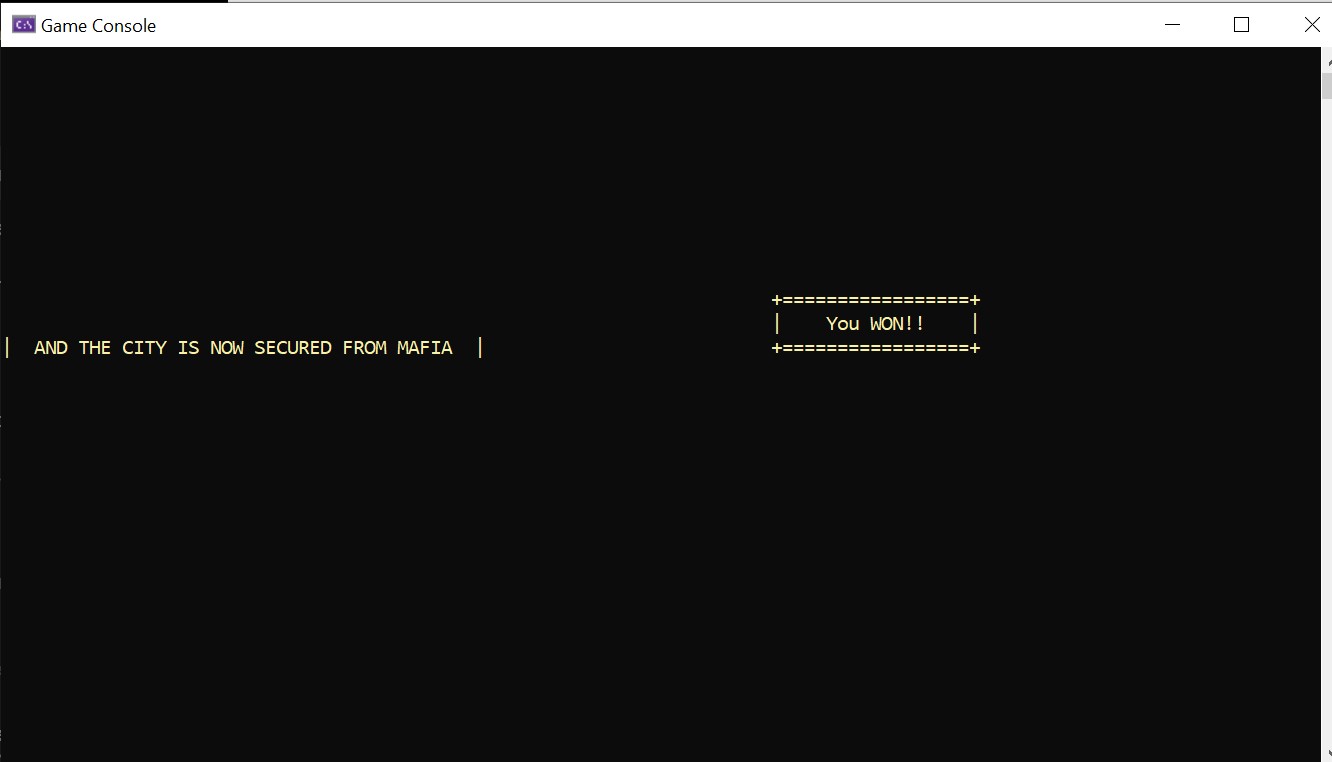
**Battle Outcome Interaction:**

You WON!!

+============+

| You WON!! |

+============+

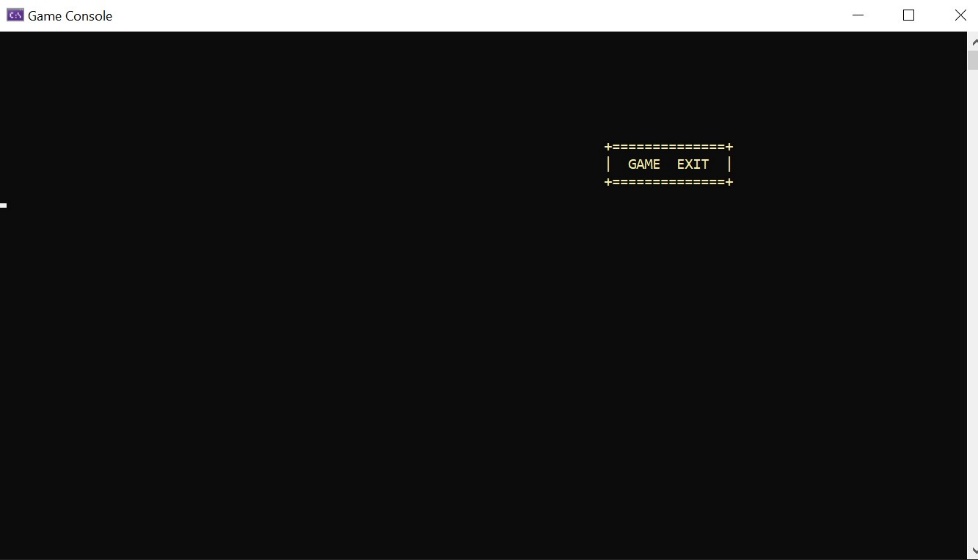


**Exit Interaction:**

+===========+

| GAME EXIT |

+===========+



**Flowchart:**

START

Prompt the user to start or exit.

**GAMEPLAY LOOP**

Display the game map and player's position.

**GAME MODE SELECTION**

Prompt the user to select the game difficulty level (Easy, Normal, Hard).

GAME INITIALIAZATION

Initialize player and mafia attributes (health, attack power, etc.).

**MAFIA ENCOUNTER**

Initiate the battle simulation.

BATTLE SIMULATION

Display battle options (Attack, Defend, Heal).

**CHECK GAME OVER CONDITIONS**

Display victory or defeat message.

**EXIT OR RESTART**

Allow the user to exit the game or restart.

END

**Decision Points and Possible Decisions**

In the text-based game, decision points and possible branches in the story occur at various stages, affecting the gameplay and outcome. Here are some key decision points and branches:

**Game Mode Selection:**

* **Decision Point:** User chooses the difficulty level (Easy, Normal, Hard).
* **Branches:**

The selected difficulty level determines the target score to encounter the mafia and the overall difficulty of the game.

**Gameplay Loop:**

* **Decision Point:** Player moves through the map and encounters events.
* **Branches:**

If the player collects dots, their score increases.

If the player encounters the mafia, it triggers the battle simulation.

**Battle Simulation:**

* **Decision Point:** Player chooses an action (Attack, Defend, Heal) during the battle.
* **Branches:**

Outcome of the chosen action affects player and mafia health.

Different outcomes based on random factors, influencing the progression of the battle.

**Check Game over Conditions:**

* **Decision Point**: Check if player health or mafia health reaches zero.
* **Branches:**

If player health reaches zero, the player loses, and the game ends.

If mafia health reaches zero, the player wins, and the game proceeds to the victory outcome.

**Exit or Restart:**

* **Decision Point:** After a game over or victory, the player decides whether to exit or restart.
* **Branches:**

If the player chooses to exit, the game ends.

If the player chooses to restart, the game returns to the gameplay loop.

**Encounter Conditions:**

* **Decision Point**: Player reaches the required score to encounter the mafia.
* **Branches:**

The game triggers an encounter with the mafia, leading to the battle simulation.

**Technical Requirements:**

**Programming Language**

**Specific Features of C++ :**

**Console Manipulation:**

Our project utilizes Windows-specific console manipulation functions, such as **SetConsoleCursorPosition** and **GetAsyncKeyState**, to interact with the console window.

**Random Number Generation:**

The **rand()** function is used for generating random numbers. This is employed in various parts of the code, such as determining the outcome of actions in the battle simulation.

**Structures:**

The code uses C++ structures (**struct**) to define entities like the player, walk, and target. Structures allow grouping related variables together.

**Vectors:**

The **std::vector** container is used to store instances of the **walk** and **target** structures. Vectors dynamically resize, providing flexibility in handling varying numbers of elements.

**Memory Operations:**

The **memcpy** function is used for memory copying, specifically to copy the game map to a temporary map for path finding purposes.

**Constants and Macros:**

Constants and macros are defined using **#define**, such as defining colors for console output (**RESET**, **YELLOW**).

**Console Text Formatting:**

Escape sequences (**\033[0m**, **\033[33m**) are used for console text formatting, providing colored output.

**Console Clearing:**

The **system("cls")** function is used to clear the console screen, creating a smoother transition between different game states.

**Justification for their Selection:**

**Console Manipulation (Windows API):**

* **Justification:** The project targets a Windows environment, so using Windows-specific console manipulation functions (**SetConsoleCursorPosition**, **GetAsyncKeyState**) provides precise control over the console window and input handling.

**Random Number Generation with rand():**

* **Justification:** **rand()** is a simple and widely used function for generating pseudorandom numbers. While it may not be the most sophisticated random number generator, it is suitable for this game's purposes and offers a good balance between simplicity and randomness.

**Structures and Vectors:**

* **Justification:** Structures provide a convenient way to group related variables together, improving code organization. Vectors dynamically resize, making them suitable for storing variable-sized collections of entities like **walk** and **target**.

**Memory Operations with memcpy:**

* **Justification**: **memcpy** is used to copy the game map to a temporary map for pathfinding. This allows the program to perform pathfinding operations on a temporary map without modifying the original game map, ensuring data integrity.

**Constants and Macros:**

* **Justification:** Constants and macros improve code readability and maintainability. For example, defining color codes as macros (RESET, YELLOW) makes it easier to manage and modify the appearance of console output.

**Console Clearing with system("cls"):**

* **Justification:** Clearing the console screen using **system("cls")** provides a clean and visually appealing transition between different game states. While alternatives exist, this method is straightforward and effective for a console-based game.

**Libraries and Tools**

**External Libraries:**

The provided C++ code uses the following external libraries and tools**:**

* **Windows API**
* **Console Escape Sequences (e.g., "\033[0m", "\033[33m")**
* **Windows-specific Console Commands (e.g., system("cls"))**

**Contribution:**

The external tools and libraries used in the C++ code contribute to the project in the following ways:

**Windows API:**

**Contribution:** The Windows API is crucial for console manipulation, allowing precise control over the console window. Functions like SetConsoleCursorPosition enable the placement of characters at specific coordinates, enhancing the visual representation of the game. Additionally, GetAsyncKeyState is employed for handling user input, making the game responsive to keyboard commands.

**Console Escape Sequences:**

**Contribution**: Escape sequences such as "\033[0m" and "\033[33m" contribute to text formatting in the console. They provide a way to change text color, enhancing the visual appeal of the game. This makes it easier to distinguish between different elements in the console output, improving the overall user experience.

**Windows-specific Console Commands (e.g., system("cls")):**

**Contribution**: The use of system("cls") contributes to the project by clearing the console screen. This command creates a cleaner and more organized presentation, preventing the accumulation of unnecessary information on the console. Clearing the screen at specific points, such as between different game states, ensures a visually smooth and comprehensible user interface.

**Platform Compatibility**

**Platform:**

It is designed to be executed on Windows platforms. The Windows API functions used in the code, such as **SetConsoleCursorPosition** and **GetAsyncKeyState**, are specific to the Windows operating system. The game is intended to be executed on Windows platforms, and adjustments would be needed for it to run on other operating systems.

**Requirements:**

**Windows Operating System:**

The code is designed to run on the Windows operating system. It utilizes Windows-specific console manipulation functions and commands, such as **SetConsoleCursorPosition** and **system("cls")**.

**Windows Console Environment:**

The game is intended to be executed in a Windows console environment, such as the Windows Command Prompt or PowerShell. It relies on features and commands that are specific to the Windows console.

**Keyboard Input:**

The game relies on keyboard input for user interaction. Arrow keys (UP, DOWN, LEFT, RIGHT) are used for controlling the protagonist's movement.

**CODE:**

**Code Organization**

**Directory Structure:**

**Root Directory:**

* **src:**

main.cpp

main.sln

* **include:**

iostream

windows.h

vector

**File Types and Counts:**

* .cpp files: 1
* .sln files: 1

**Build time:**

7.143 seconds

**Additional Insights:**

* Single Source File: The entire game code is contained within a single .cpp file, suggesting a relatively compact codebase.
* Standard Libraries: The code utilizes standard C++ libraries (iostream, windows.h, vector) for input/output, console window management, and data storage.

**Code Modules:**

**Key Modules (Functions) and Their Responsibilities:**

* **menu():** Manages core game logic, including difficulty selection, game loop, scoring, and battle initiation.
* **simulateBattle():** Handles battle mechanics, presenting attack/defend/heal options, calculating damage and healing, and determining win/loss conditions.
* **response():** Calculates damage and healing amounts based on player actions and difficulty.
* **checker():** Checks for win/loss outcomes based on HP values.
* **encounter():** Displays the message when the protagonist encounters the mafia.
* **exit():** Exits the game with a visual prompt.
* **FindPath():** Manages enemy pathfinding logic.
* **ShowMap():** Displays the game map.
* **loading():** Shows a loading screen animation.
* **gotoxy():** Positions the cursor on the console.

**Keys Code Snippets**

**Critical Functionalities:**

**Initialization and Introduction:**

Setting up the console and displaying an introduction.

**Menu Function:**

Displaying a menu and getting the user's choice.

**Game Loop and User Input:**

Handling user input during the game loop.

**Encounter and Battle Simulation:**

Initiating an encounter and simulating a battle.

**Response and Game Outcome:**

Handling user responses and determining game outcomes.

These excerpts capture key functionalities such as initialization, menu navigation, user input handling, encounter initiation, and battle simulation.

**Implementation Choices:**

**Console Interface:**

We used a console-based interface for simplicity and focus on game logic.

**Random Gameplay:**

We added unpredictability with random functions for attacks, defense, and healing.

**Character Encoding:**

We represented game elements with characters due to limited graphical assets.

**Enemy Movement:**

Implemented breadth-first search for efficient enemy movement in maze-like environments.

**Menu-Driven Interaction:**

Enabled user choices through a menu-driven system for simplicity.

**Encounter and Battle:**

Separated encounter and battle phases for a structured gameplay flow.

**Dynamic Difficulty:**

Included dynamic difficulty levels for varied player experiences.

**USE CASES:**

**Player Interactions**

**Player’s Actions:**

**Exploration Phase:**

* Player navigates a maze-like city using arrow keys.
* Collects dots to gain points.

**Encounter with Mafia:**

* Player encounters the Mafia after reaching a specific score.
* Enters the battle phase with the Mafia.

**Battle Phase:**

* Player chooses actions (Attack, Defend, Heal) using text commands.
* Randomized outcomes for each action affect player and Mafia health.

**Decision Points:**

* Player decides whether to attack aggressively, defend strategically, or heal tactically.
* Choices impact the outcome of the battle.

**Scoring System:**

* Player needs to achieve a certain score to trigger the Mafia encounter.
* Different difficulty levels require varying scores for encounters.

**Dynamic Difficulty:**

* Difficulty levels (Easy, Normal, Hard) influence the required score and battle parameters.

**Healing Mechanism:**

* Player can choose to heal but risks the Mafia healing as well.
* Timing of healing decisions is crucial.

**Game Responses:**

**Attack (A/a):**

* **Outcome**: Reduces Mafia and player health.
* **Response:** Displays damage dealt to Mafia and damage received by the player.

**Defend (D/d):**

* **Outcome**: Reduces Mafia damage**.**
* **Response:** Informs the player that they are defending, reducing the impact of the Mafia's attack.

**Heal (H/h):**

* **Outcome:** Restores player health.
* **Response:** Shows the amount of health restored to the player and may indicate Mafia healing in response.

**Invalid Option:**

* **Outcome:** Displays an error message.
* **Response:** Informs the player that the selected option is invalid and prompts for a valid choice.

**Game Over (Caught by Mafia):**

* **Outcome:** Ends the game.
* **Response:** Notifies the player that they were caught by the Mafia during exploration, displaying the final score.

**Battle Victory (Mafia's health reaches zero):**

* **Outcome:** Ends the battle phase.
* **Response**: Declares victory, displays the final score, and may provide a narrative continuation.

**Battle Defeat (Player's health reaches zero):**

**Outcome:** Ends the battle phase.

**Response:** Declares defeat, displays the final score, and may provide a narrative conclusion.

**Edge Cases**

**Edge Cases in User Inputs:**

**Invalid Input Handling:**

* **Edge Case:** Non-numeric input.

**Boundary Checking:**

* **Edge Case:** Health below zero or exceeding limits.

**Memory Allocation Errors:**

* **Edge Case:** Memory allocation failure.

**Concurrency Issues:**

* **Edge Case:** Simultaneous input causing issues.

**Empty Map or Missing Data:**

* **Edge Case:** Map loading issues.

**Handling of Edge Cases:**

**Invalid Input Handling:**

* **Handling:** Implemented input validation**.**

**Boundary Checking:**

* **Handling:** Implemented checks to ensure valid health values.

**Memory Allocation Errors:**

* **Handling:** Included error-checking and graceful exit.

**Concurrency Issues:**

* **Handling:** Implemented synchronization mechanisms**.**

**Empty Map or Missing Data:**

* **Handling:** Checked and handled missing or corrupted map data.

**CHALLENGES FACED:**

**Technical Challenges**

**Technical Obstacles:**

Technical obstacles encountered during development are following:

**Algorithm Complexity:**

* It is a difficult task for us to implement the algorithms effectively and efficiently.

**User Input Handling:**

* Managing user input robustly to handle unexpected or invalid inputs without causing errors or unexpected behavior.

**Random Number Generation:**

* Generating random numbers effectively for various game elements, such as combat outcomes and enemy movements.

**Map Design:**

* Creating interesting and challenging maps that contribute to the overall game experience.

**User Experience:**

* Designing the user interface and overall user experience to be intuitive and enjoyable.

**Addressing of Challenges:**

**Algorithm Complexity:**

* Tried to break complex algorithms into modular and manageable components.
* Optimized critical sections of code for performance.

**User Input Handling:**

* Used functions or methods to encapsulate input handling logic.

**Random Number Generation:**

* Seeded the random number generator appropriately.

**Map Design:**

* Created diverse and interesting maps that challenge players.
* Ensured a balance between difficulty and enjoyment.

**User Experience:**

Designed a clear and intuitive user interface.

**Design Challenges**

**Challenges related to Game’s Narrative:**

**Player Choices and Consequences:**

* Implemented a system that allows players to make meaningful choices that impact the game's progression.
* Balanced the consequences of player choices to maintain a fair and engaging experience**.**

**Pacing and Flow:**

* Managed the pacing of the narrative to maintain player interest.
* Ensured a smooth flow between different story events and gameplay segments.

**Integration of Story and Gameplay:**

* Seamlessly integrated narrative elements with gameplay mechanics to create a cohesive and immersive experience.
* Avoided situations where the narrative feels disconnected from the player's actions.

**Design Iterations:**

* Storyline Enhancements
* Randomization Logic
* User Interface (UI)
* Difficulty Levels
* Player Choices

**LEARNING OBJECTIVE ACHIEVED:**

**Programming Skills**

**Project Enhanced Programming Proficiency:**

This project enhanced our programming proficiency skills in several ways:

* C++ Language Proficiency
* Algorithmic Thinking
* Game Development Concepts
* Randomization and Decision Making
* Error Handling
* User Interface Design
* Using External Libraries

**Problem Solving**

**Problem-Solving Skills:**

Problem-solving skills were developed and applied throughout the development of the text-based game in C++.

**Algorithmic Design:**

* Developed algorithms for character movement, enemy AI, and randomization, requiring a systematic approach to problem-solving.

**Game Mechanics:**

* Resolved challenges related to game mechanics, such as scoring, encounters, and battles.

**Error Debugging:**

* Identified and fixed errors and bugs in the code through systematic debugging, enhancing problem-solving abilities.

**User Interface Challenges:**

* Solved challenges related to console manipulation and user interface design, ensuring a clear and user-friendly display.

**Randomization Logic:**

* Implemented effective randomization logic for actions like attacks and healing, requiring problem-solving skills to balance gameplay.

**Difficulty Levels:**

* Designed different difficulty levels with appropriate scoring requirements

**Implementation of Creative Solutions:**

* Dynamic Map Display
* Randomized Battle Outcomes
* Enemy Pathfinding with BFS
* User-Friendly Interface
* Difficulty Levels
* Realistic Healing Mechanism

**Collaboration and Communication**

**Collaborative Aspect:**

Collaboration played a significant role in various aspects of the project, enhancing its overall development and quality.

**Division of Tasks:**

* We identified interests of each other and took the tasks accordingly. One member did the report and the other one did code and programing.

**Regular Communication:**

* We maintained open and regular communication throughout the project which helped us a lot in developing the project.

**Problem-Solving Together:**

* Encountered and addressed challenges collaboratively.

**Quality Assurance:**

* Conducted joint testing and quality assurance activities.

**Motivation and Support:**

* Provided mutual motivation and support. Encouragement during challenging phases and celebrating small victories together fostered a positive and productive working environment.

**Learning Opportunities:**

* Embraced the project as a learning opportunity.

**Communication Skills during Project:**

We communicated well in our team. Regular updates and clear discussions helped us understand each other. This made working together easy and helped us solve problems quickly. Good communication kept our team strong and productive during the project. This facilitated seamless task coordination, efficient problem-solving, and mutual feedback. Strong communication skills fostered a cohesive and productive team dynamic throughout the development process.

**CONCLUSION:**

**Summary**

**Key Points in Report:**

Our report covers various essential aspects of the project:

1. **Introduction:**
   * Provides an initial overview of the project.
2. **Project Overview:**
   * Summarizes the key elements and goals of the project.
3. **Game Concept:**
   * Describes the main idea and theme of the game.
4. **Storyline:**
   * Outlines the narrative or plot of the game.
5. **Gameplay Mechanics:**
   * Explains how the game functions, including player actions and interactions.
6. **Design:**
   * Covers the overall design considerations for the game.
7. **Architecture:**
   * Discusses the structural layout of the code and program.
8. **User Interface:**
   * Describes the visual and interactive elements of the game.
9. **Flowchart:**
   * Provides a visual representation of the main flow of the game.
10. **Technical Requirements:**
    * Lists the necessary specifications for running the game.
11. **Programming Language:**
    * Specifies the language used for coding.
12. **Libraries and Tools:**
    * Identifies external resources utilized in the project.
13. **Platform Compatibility:**
    * States where the game can be executed.
14. **Code:**
    * Highlights sections of the actual code.
15. **Player Interactions:**
    * Explains how users engage with the game.
16. **Challenges Faced:**
    * Discusses difficulties encountered during development.
17. **Technical Challenges:**
    * Describes specific technical issues encountered.
18. **Design Challenges:**
    * Discusses challenges related to the game's design.
19. **Learning Objectives Achieved:**
    * Reflects on the educational aspects of the project.
20. **Programming Skills:**
    * Discusses improvements in coding proficiency.
21. **Problem Solving:**
    * Highlights instances where problem-solving skills were applied.
22. **Collaboration and Communication:**
    * Emphasizes teamwork and effective communication.

**Significance of Project in Learning Objectives:**

This project significantly contributed to achieving learning objectives by providing hands-on experience in game development, enhancing programming skills, and fostering problem-solving abilities. The complexity of coding, design challenges, and collaborative efforts enriched the learning process, ensuring a holistic understanding of software development. The project serves as a practical application of theoretical concepts, reinforcing key programming principles and team collaboration skills.

**Future Enhancements**

**Potential Improvements:**

* Expanded Storyline
* Graphics Upgrade
* Multiplayer Mode
* Dynamic Environments

**Learning and Exploration:**

1. **Graphics Programming**: Dive deeper into graphics libraries to implement advanced visual effects.
2. **Networking in Games**: Learn about multiplayer networking and real-time communication for online gameplay.
3. **Game Design Principles**: Study game design principles to optimize user experience and engagement.
4. **Optimization Techniques**: Explore methods to optimize code for better performance in resource-intensive games.

**Acknowledgements**

**Online Tutorials:**

Various online tutorials and forums provided valuable insights into game development concepts and C++ programming**.**

**Collaboration with Teammate:**

Effective collaboration with my teammate facilitated brainstorming, problem-solving, and project completion and quick problem solving.

**Online Coding Resources:**

Online coding resources provided guidance throughout the project.

**References**

1. <https://www.geeksforgeeks.org/c-plus-plus/>

2. <https://www.youtube.com/playlist?list=PLxgZQoSe9cg1drBnejUaDD9GEJBGQ5hMt>

3. <https://www.youtube.com/playlist?list=PLxgZQoSe9cg0df_GxVjz3DD_Gck5tMXAd>

4. <https://www.youtube.com/watch?v=wjqSZy4pMT4>