

**SOFTWARE RE-ENGINEERING**

**ASSIGNMENT 3**

**Course Instructor:**

Ma’am Shahila Saif

**Name:**

* Kisa Fatima (21i-1138)
* Maham Munir (21i-1231)

**Section:**

SE-P

1. [**Tool Selection 3**](#_lfp5iq36o0ub)
2. [**Software Selection 3**](#_1nxmhk1d3aax)
3. [**Statistical Analysis: 4**](#_fyptujhnsk6v)
4. [**Control Flow Graph 18**](#_tunp29je4rm)
5. [**Memory Heap 20**](#_w2arqbo1vv22)
6. [**Code Extraction 21**](#_xem42j9ujuh5)
7. [**Program Tree 25**](#_x9m0hdcdavjc)
8. [**Symbol Tree 25**](#_j6v5xauqii3a)
9. [**Findings on Software Architecture and Functionalities 26**](#_3013bp58txzy)

[Software Architecture 26](#_yiw5woiql7hh)

[Functionalities 27](#_d5h2ekmdodnr)

## **Tool Selection**

Ghidra is an advanced open-source reverse engineering framework that has been developed by the NSA. Its power feature and user-friendly interface make it highly suitable. Here are the main reasons why it has been chosen:

**Decompilation**

* It converts assembly language to C-like representation to facilitate the better understanding of the code.
* Interactive decompiler can rename variables and provide comments for better clarity.

**Disassembly**

* Machine code to assembly language is efficiently translated.

**Advanced Code Analysis**

* Offers control flow graph visualization, dataflow analysis, and cross references.
* Functions as an enabler to analyze function call chains, code dependencies and possible flaws.

**Scripting and Extensibility**

* Automated analysis with scriptability of Java and Python.
* Ready-made scripts or contributed ones from the users are provided to make most of this feature.

Ghidra was selected because it supports both novice and expert reverse engineers with its powerful features, versatility, and ease of use. It is perfect for cross-platform executable analysis since it can be integrated with a variety of processors and operating systems.

## **Software Selection**

We have chosen the reverse engineering project in C++ implementation of **Snake and Ladder.** The game is a simple yet console-based game, ideal to explore reverse engineering techniques without being too complex.

**Familiarity and Accessibility**

* The widely recognized game that would easily explain its functionality before proceeding with the code analysis.
* It is a compact, console-based implementation, and this makes it quite feasible while reverse engineering.

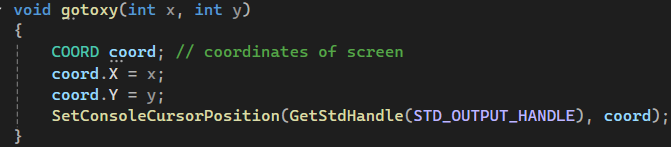
**Sufficient Complexity:**

* This includes some of the key programming concepts such as
  + Random number generation,
  + Game logic.
  + Control flow for player movement.
* These elements make it complex enough to demonstrate meaningful insights from reverse engineering.

**Relevance to Reverse Engineering**

* The project allows us to explore reverse engineering techniques, such as:
  + Analyzing game logic and flow.
  + Extracting and modifying rules, board layouts, or mechanics.
  + Understanding how C++ handles input/output operations.

## **Statistical Analysis:**

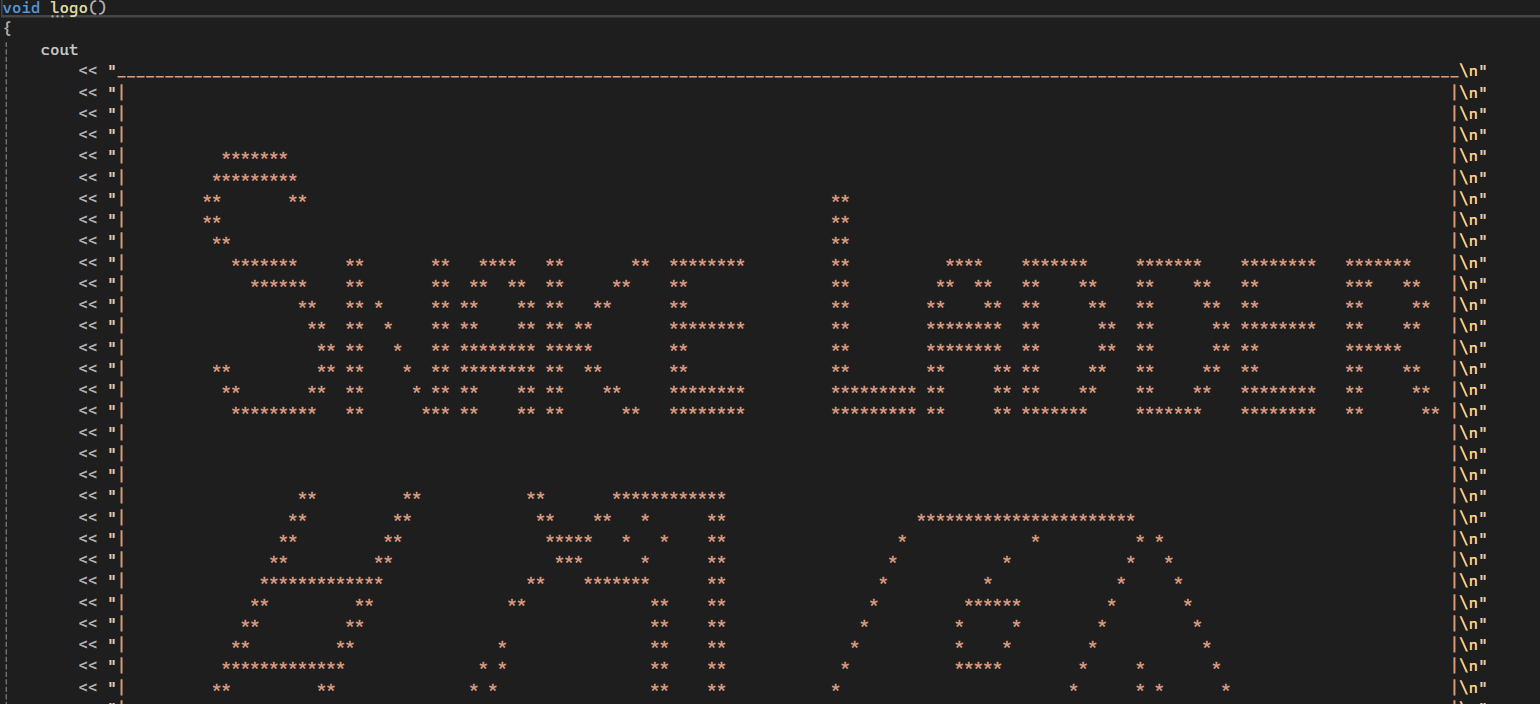


**Parameters**:

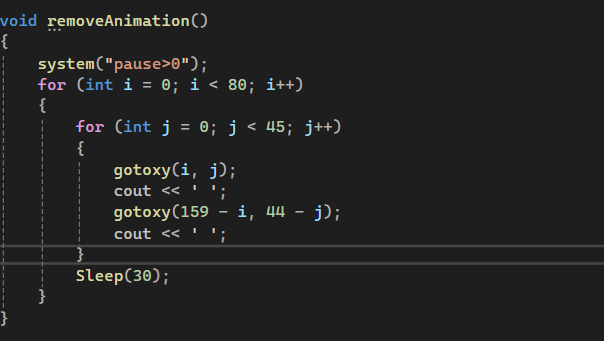
* int x: The x-coordinate (column) where the cursor should move.
* int y: The y-coordinate (row) where the cursor should move.

**Steps**:

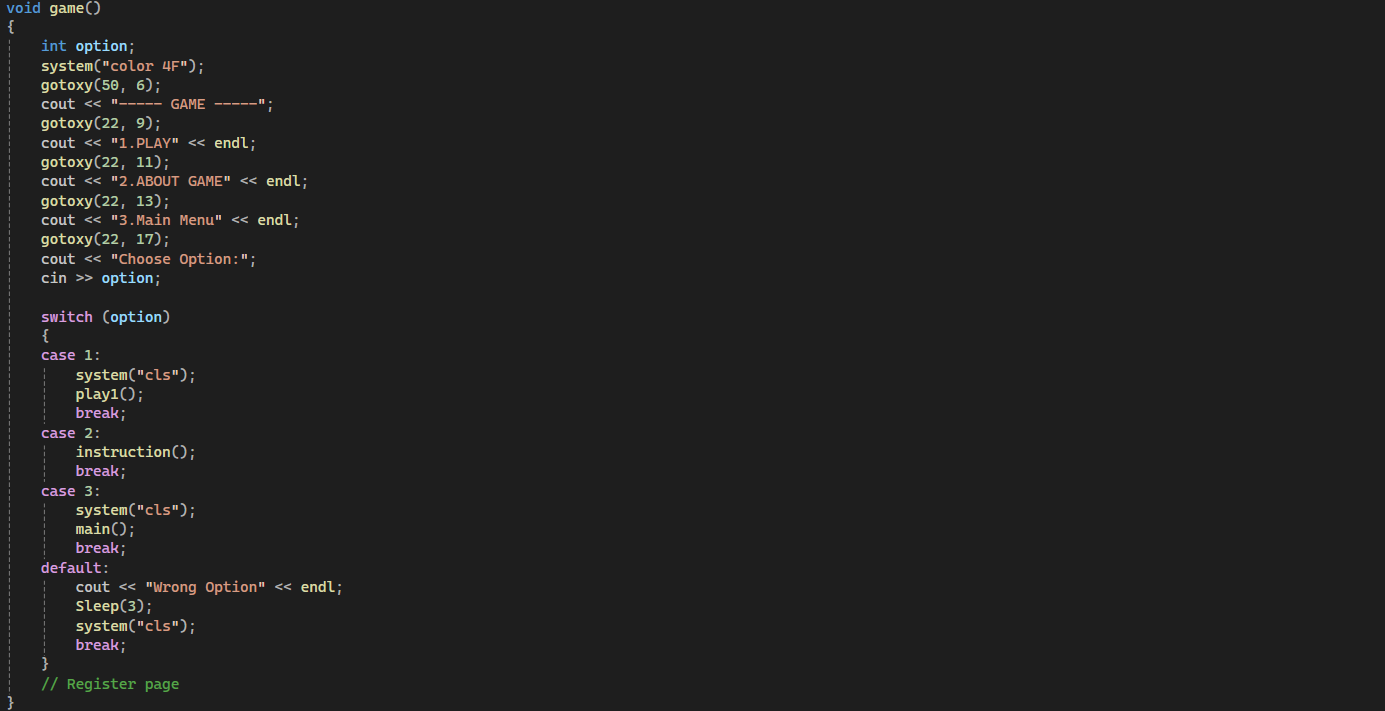
1. **COORD** : A COORD structure is created to store the x and y coordinates of the cursor.
2. **Set x and y**: The coord.X is set to x, and coord.Y is set to y.
3. **SetConsoleCursorPosition**: The SetConsoleCursorPosition function is called to position the cursor at the given coordinates in the console window.



The function just displays the logo of the gam at the start



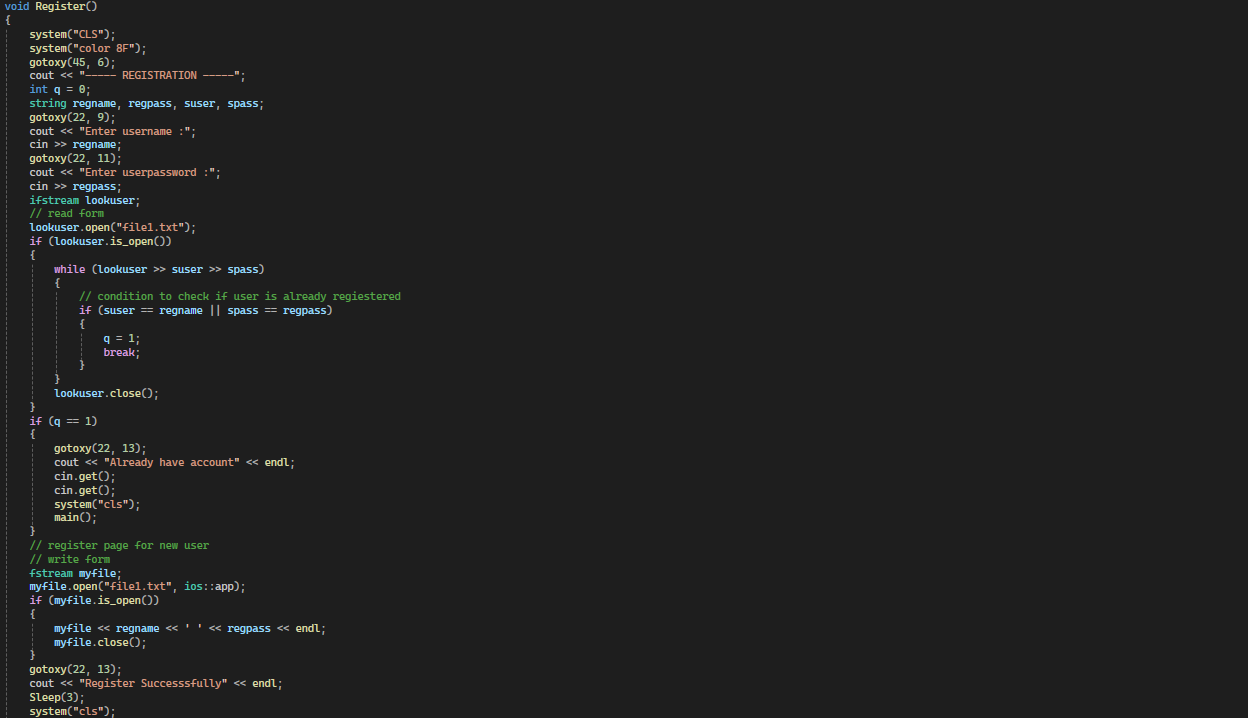
This function creates a "clearing" animation by gradually overwriting the console screen with spaces. The two loops work together to erase the screen from both top-left and bottom-right corners toward the centre. The Sleep (30) ensures that the clearing effect happens slowly enough to be visible as an animation.



The game () function displays a console menu with three options: **Play**, **About Game**, and **Main Menu**. It takes user input and performs actions based on the selection:

1. **Option 1 (Play)**: Clears the screen and calls play1() to start the game.
2. **Option 2 (About Game)**: Displays game instructions by calling instruction ().
3. **Option 3 (Main Menu)**: Clears the screen and calls main() to return to the main menu.
4. **Invalid Option**: Displays an error message, waits for 3 milliseconds, and clears the screen before re-displaying the menu.

The function uses gotoxy() to position text and system("cls") to clear the console.



The Register () function handles user registration:

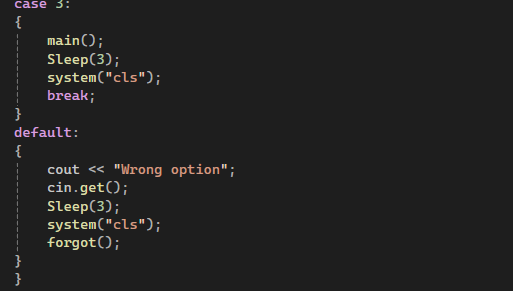
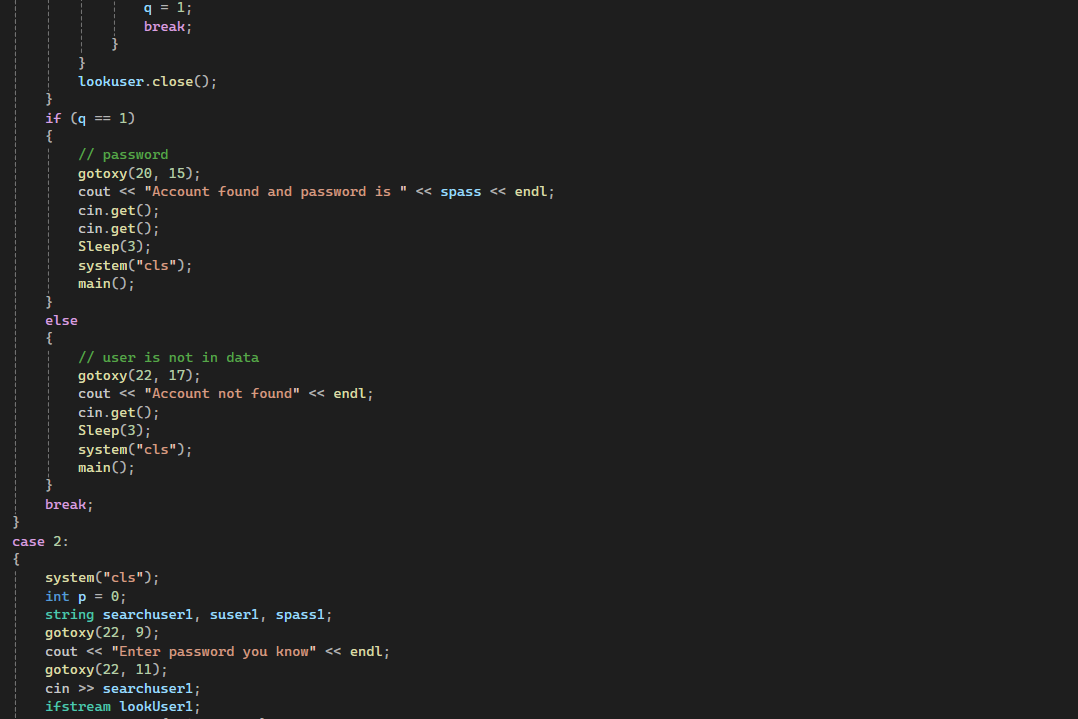
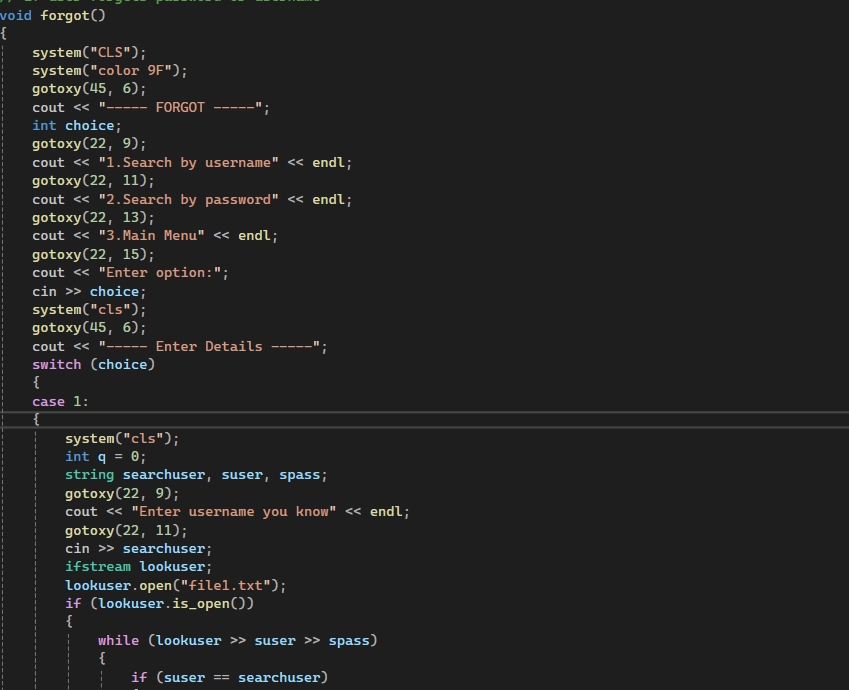
1. **User Input**: Prompts the user to enter a username and password.
2. **Check Existing User**: Reads the file file1.txt to check if the username or password already exists.
   * If the user is found, it displays a message and returns to the main menu.
3. **Register New User**: If the username is not taken, it appends the new username and password to file1.txt.
4. **Success Message**: Displays a success message, waits for 3 seconds, and then navigates to the game menu.

It uses gotoxy () to position text and system("cls") to clear the screen.



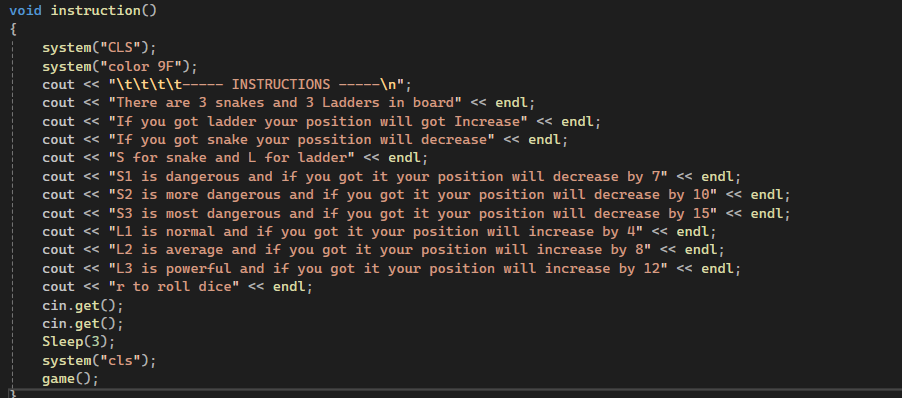
The login () function handles user login:

1. **User Input**: Prompts the user to enter their username and password.
2. **Authentication**: Reads file1.txt to verify the credentials.
   * If the username and password match an entry, the user is welcomed and redirected to the game menu.
   * If no match is found, an error message is displayed, and the user is returned to the main menu.
3. **Screen Management**: Clears the screen (system("cls")) between transitions and uses gotoxy () to position text.

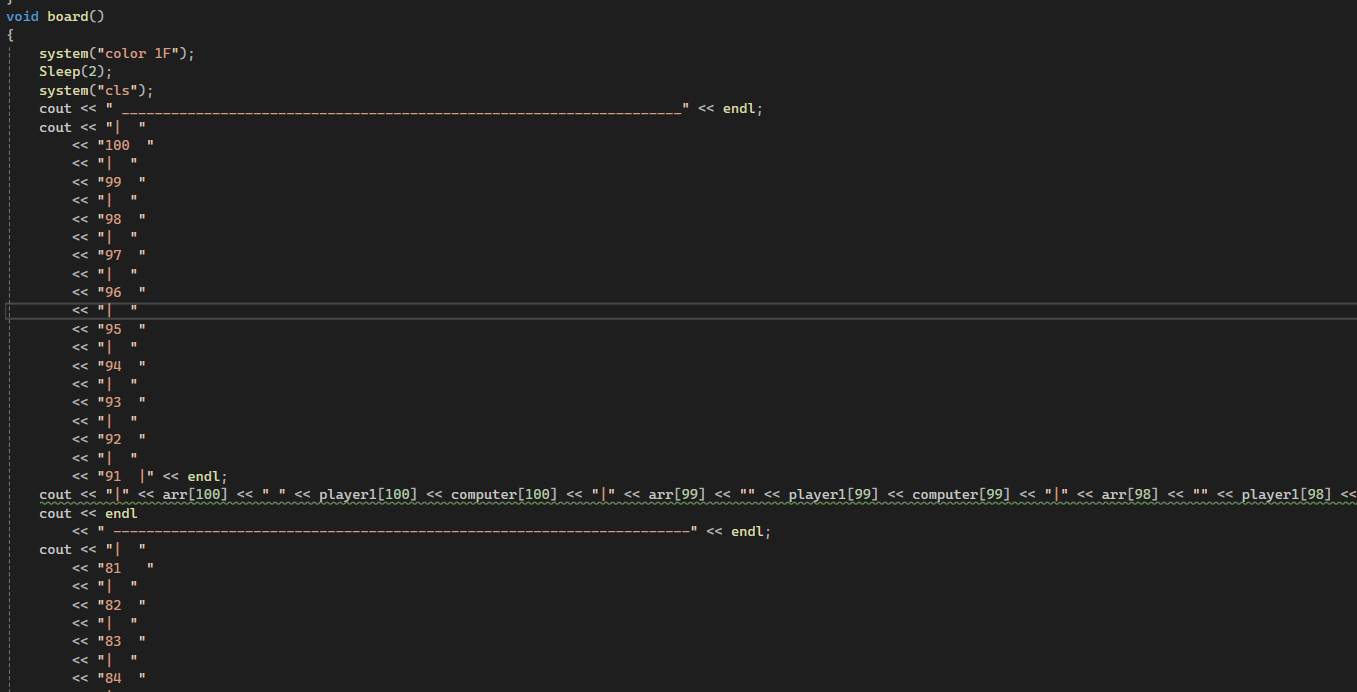


The forgot() function helps users recover their account details :

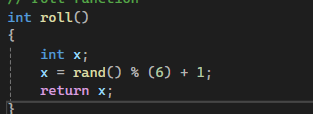
1. **Menu Options**: The user is presented with three choices:
   * **Option 1**: Search for the account by username.
   * **Option 2**: Search for the account by password.
   * **Option 3**: Return to the main menu.
2. **Functionality**:
   * **Option 1 (Search by Username)**:
     + Prompts the user to enter a username.
     + Searches file1.txt for the username.
     + If found, display the corresponding password.
     + If not found, informs the user and returns to the main menu.
   * **Option 2 (Search by Password)**:
     + Prompts the user to enter a password.
     + Searches file1.txt for the password.
     + If found, displays the corresponding username.
     + If not found, informs the user and returns to the main menu.
   * **Option 3 (Main Menu)**: Redirects the user to the main menu.
3. **Error Handling**:
   * If an invalid menu option is entered, an error message is displayed, and the menu is reloaded.



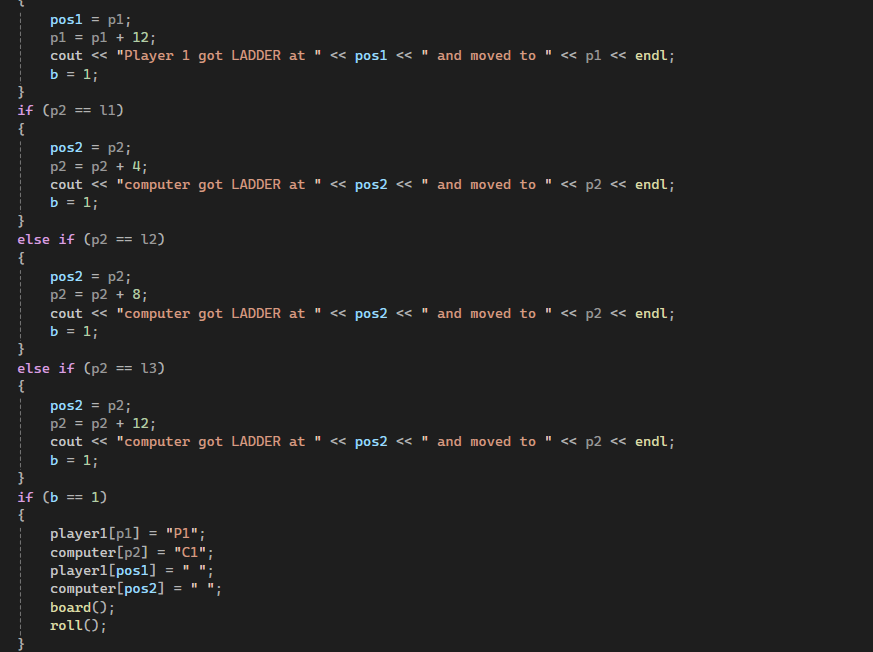
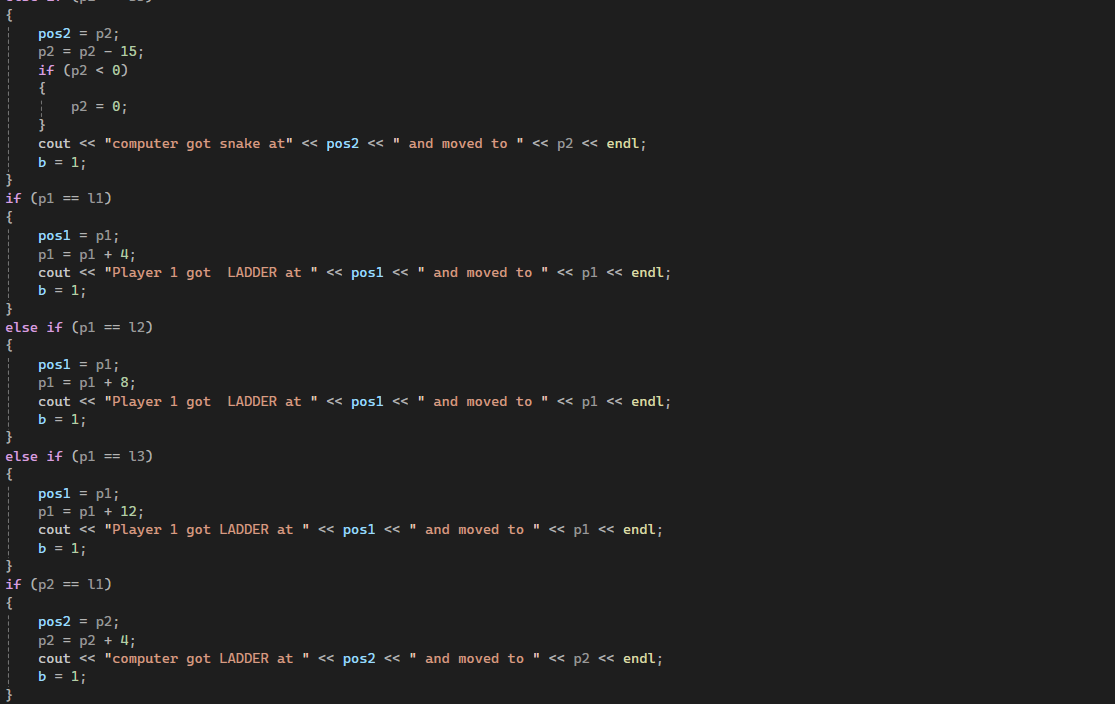
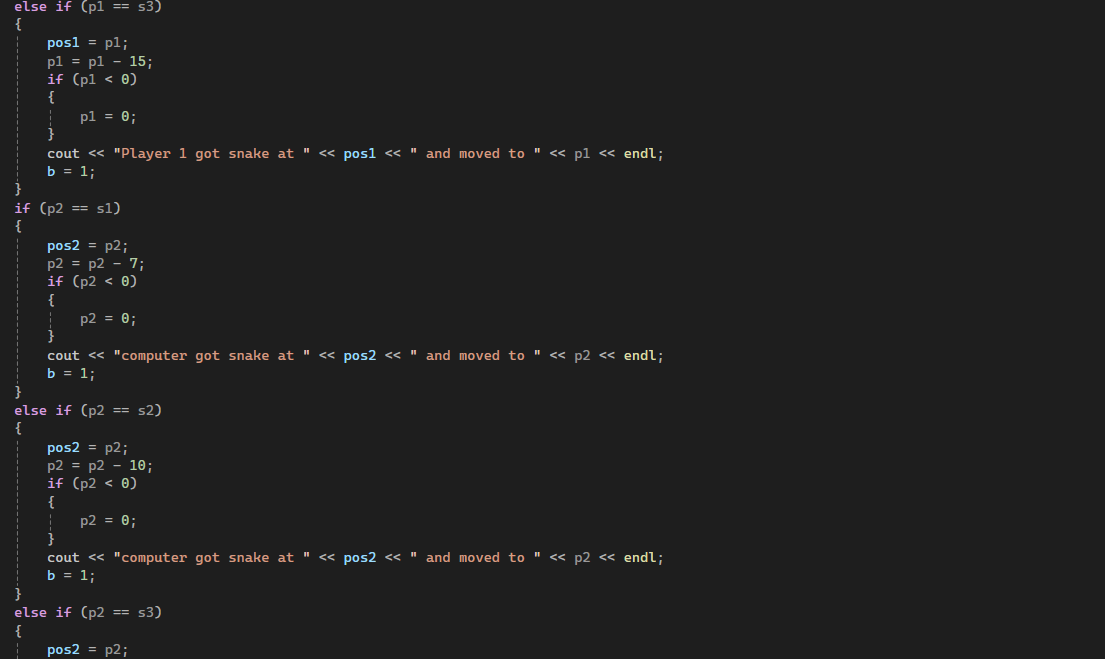
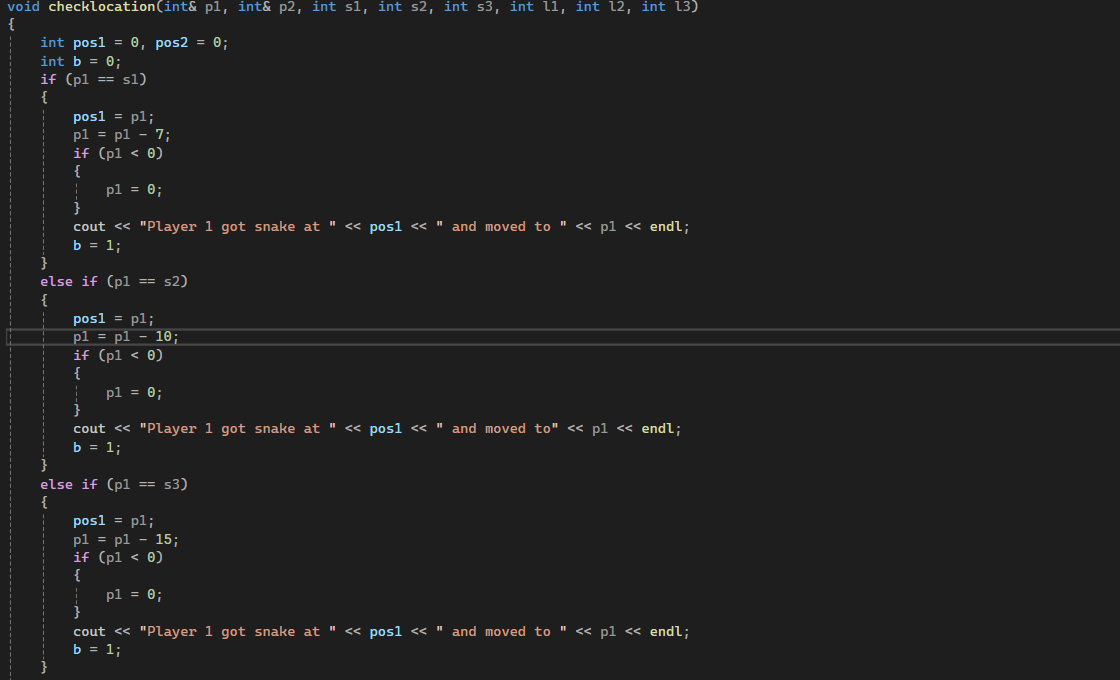
The function displays the instructions regarding the game play or rules



The function makes a game board

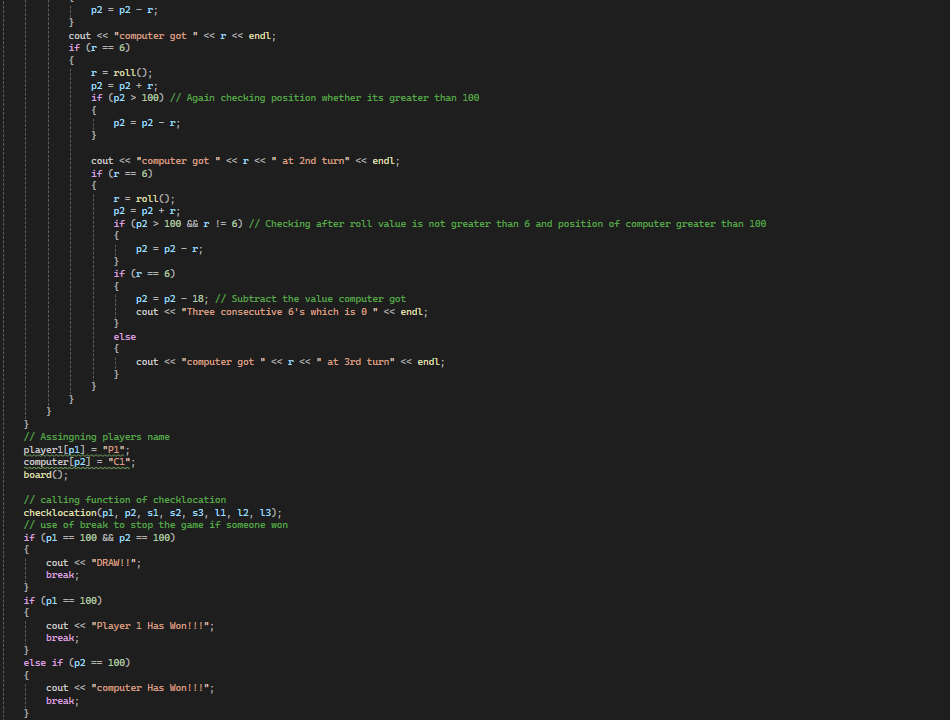
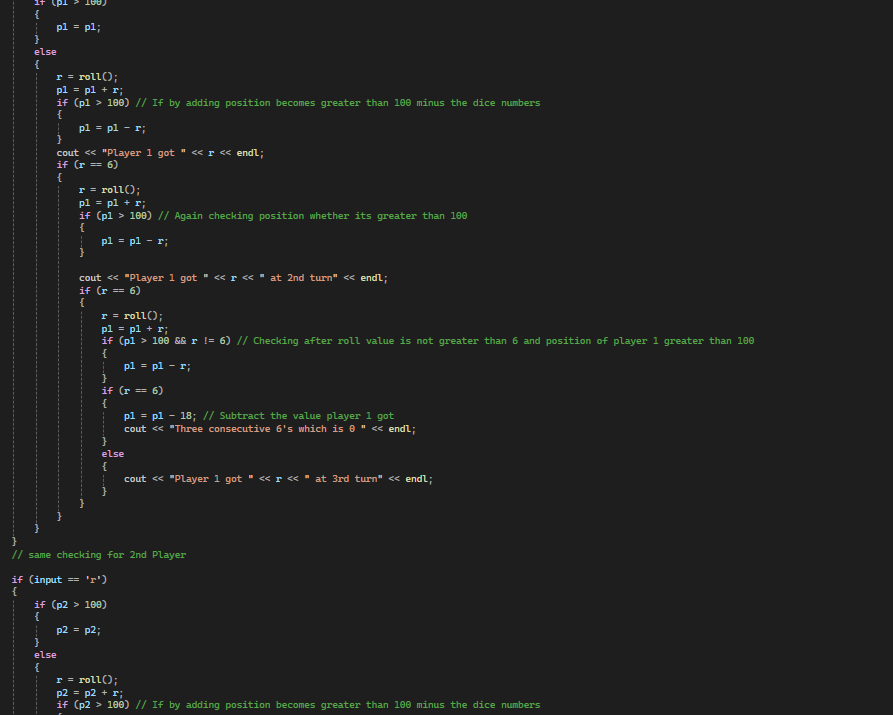
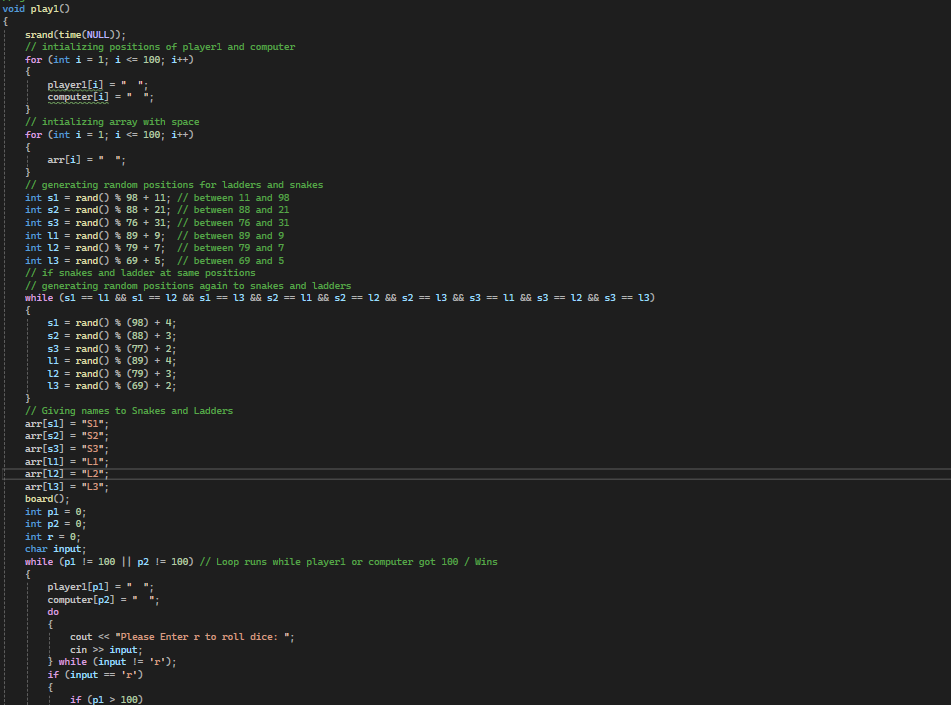


The function roll dice randomly between numbers 1 and 6



The checklocation function in the **Snakes and Ladders** game adjusts player positions based on snakes and ladders:

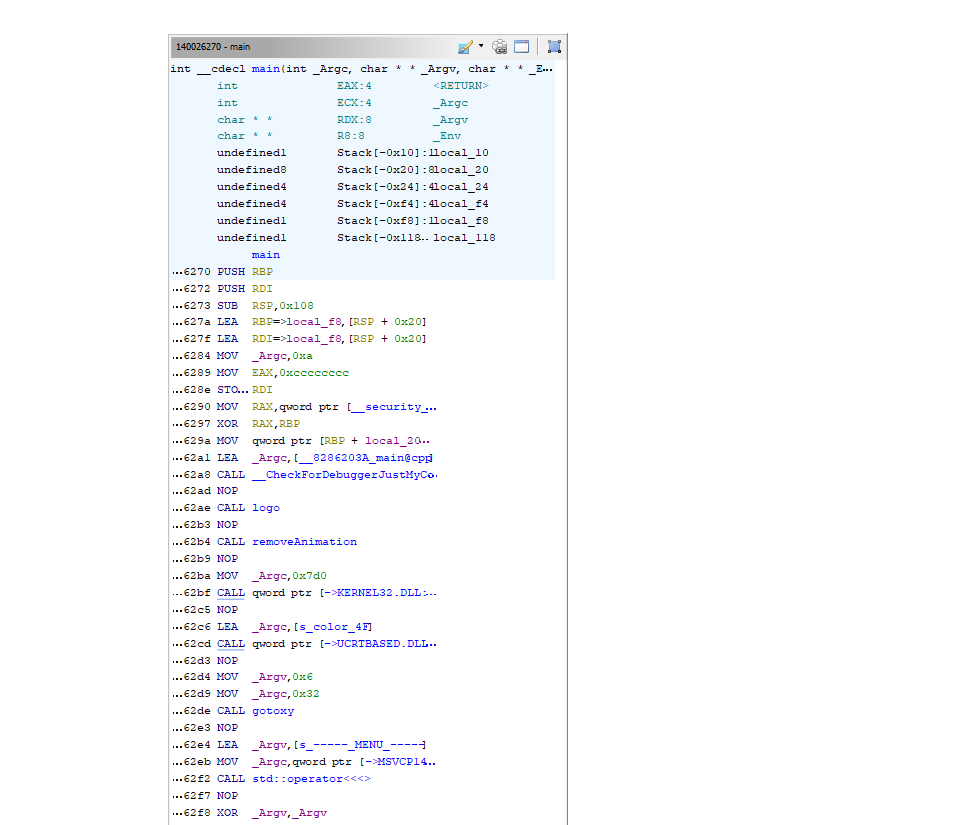
1. **Snake Check**:
   * If a player's position matches a snake (s1, s2, s3), they move back (e.g., -7, -10, -15 steps). Ensures positions don’t go below 0.
   * Displays a message about the snake encounter.
2. **Ladder Check**:
   * If a player's position matches a ladder (l1, l2, l3), they move forward (e.g., +4, +8, +12 steps).
   * Displays a message about the ladder encounter.
3. **Board Update**:
   * Updates the board with new positions for Player 1 (P1) and the computer (C1), clearing old positions.
4. **Continue Game**:
   * Calls board () to redraw the board and roll() for the next turn if any movement occurred.

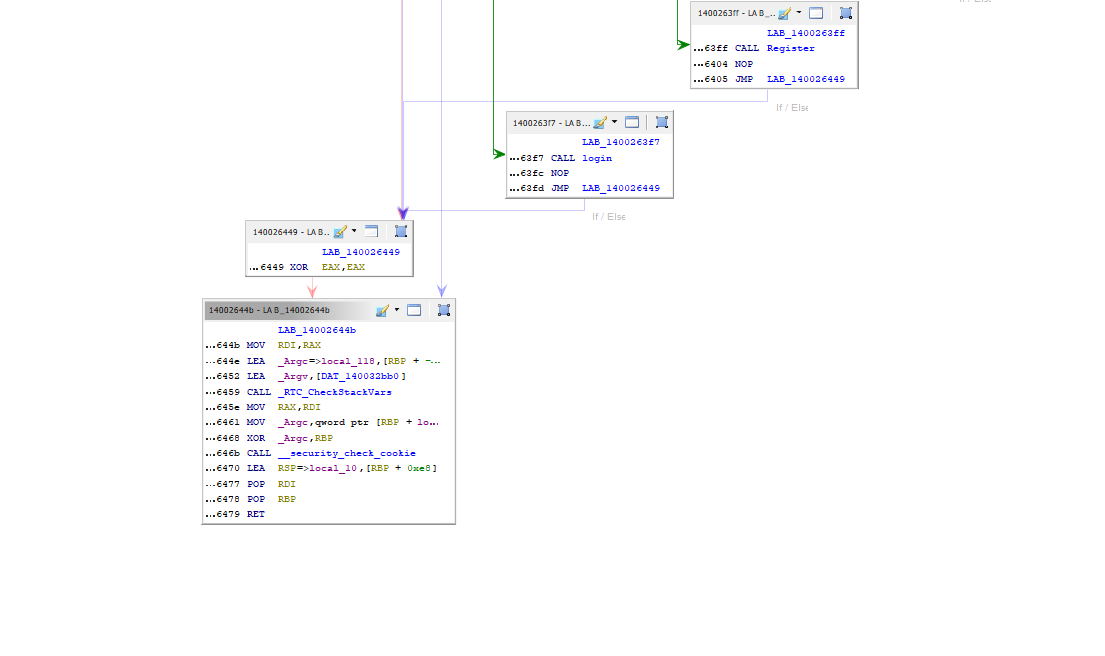
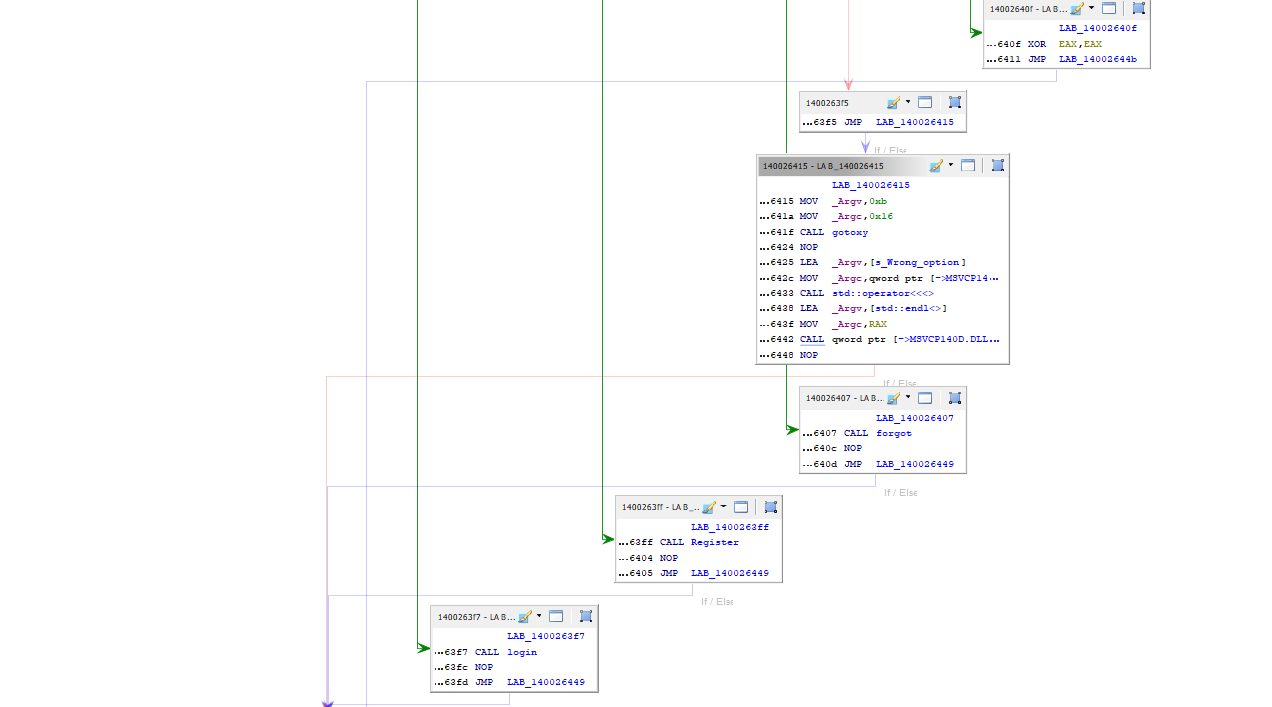
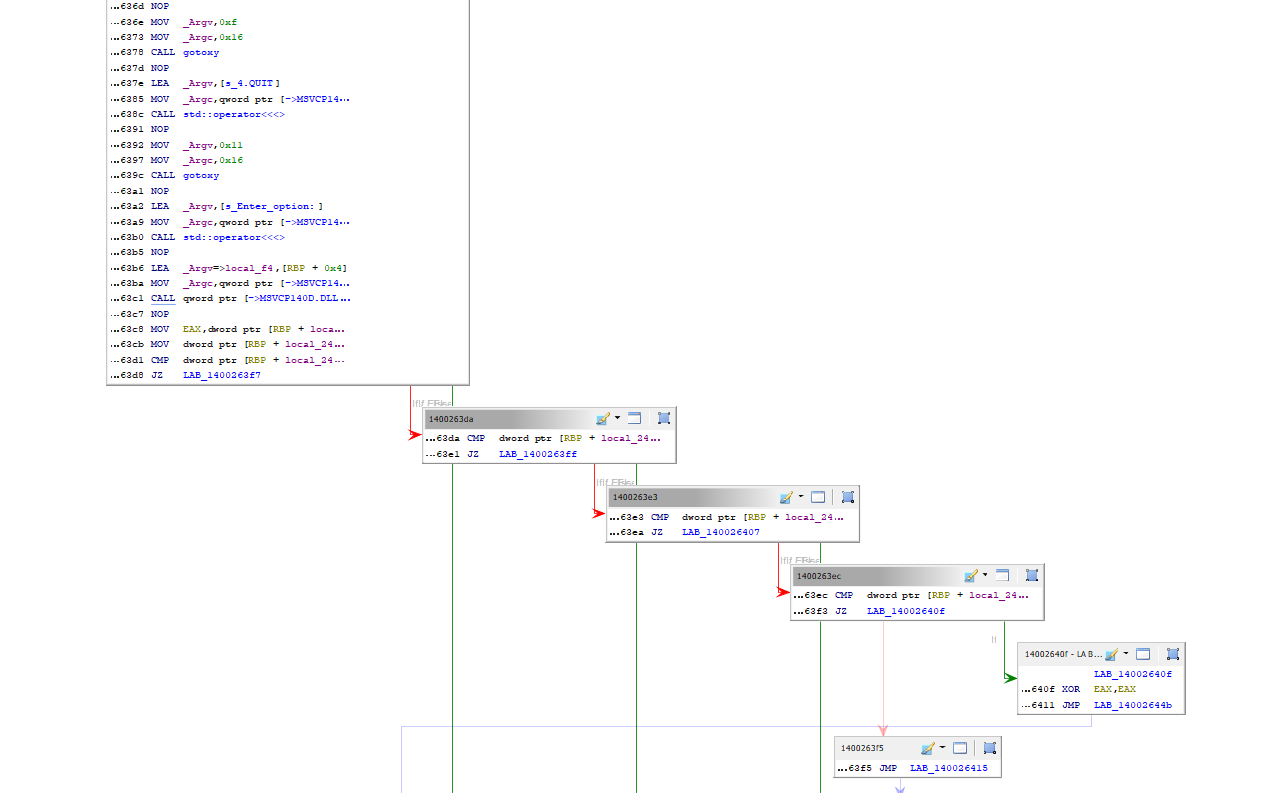


The play1 function implements the **Snakes and Ladders** game mechanics for a single player (Player 1) versus the computer:

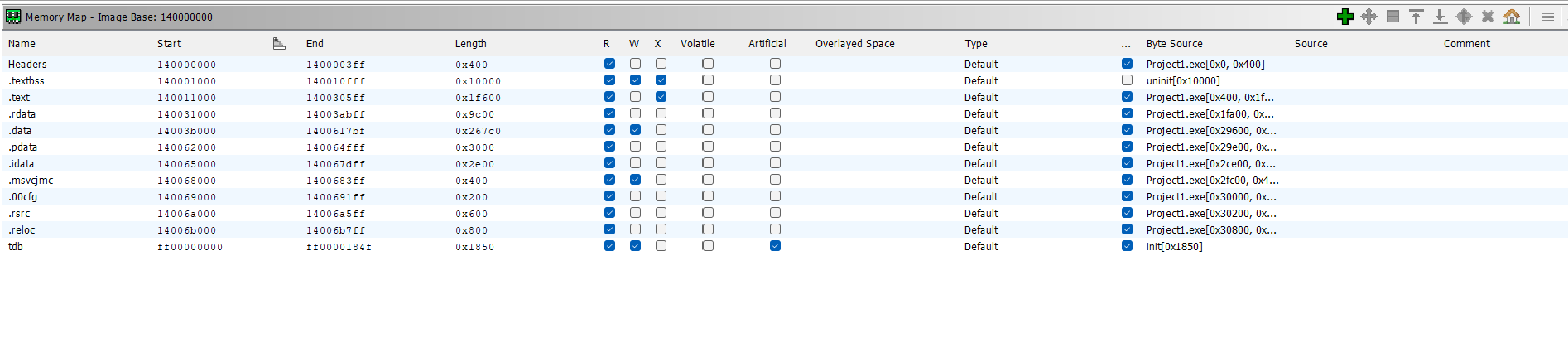
1. **Initialization**:
   * Clears and prepares game boards (player1, computer, arr).
   * Randomly generates positions for 3 snakes (s1, s2, s3) and 3 ladders (l1, l2, l3), ensuring no overlaps.
2. **Gameplay Loop**:
   * Alternates turns between Player 1 and the computer.
   * Rolls dice (roll ()), adds to the player's position, and ensures the position doesn’t exceed 100.
   * Handles special cases for rolling consecutive sixes (three sixes reset position by -18).
   * Updates board positions after each move.
3. **Check for Snakes and Ladders**:
   * Calls checklocation to adjust positions if players land on snakes or ladders.
4. **Win Conditions**:
   * Checks if either Player 1 or the computer reaches position 100. Declares the winner or a draw.
5. **Display Updates**:
   * Redraws the board after every move using the board () function.
   * Prompts the player to roll the dice by pressing 'r'.

## **Control Flow Graph**



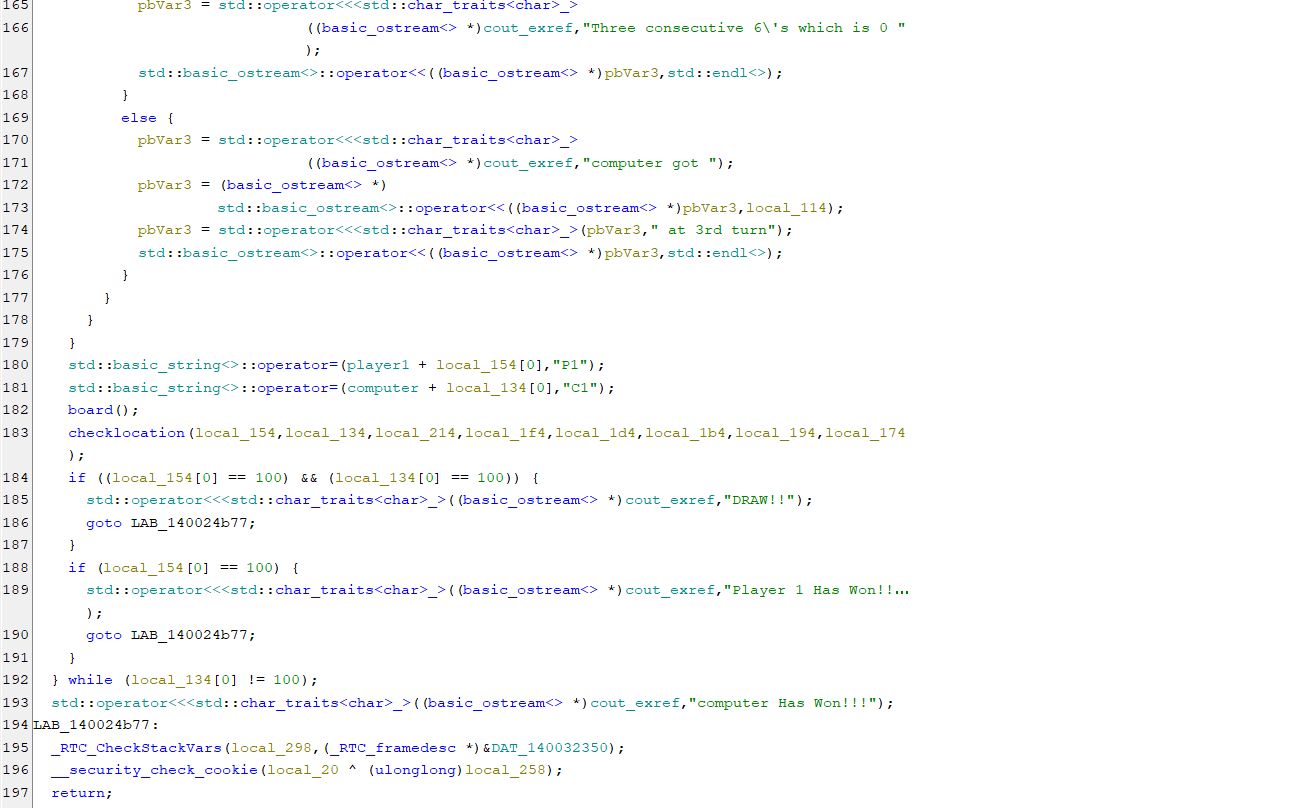
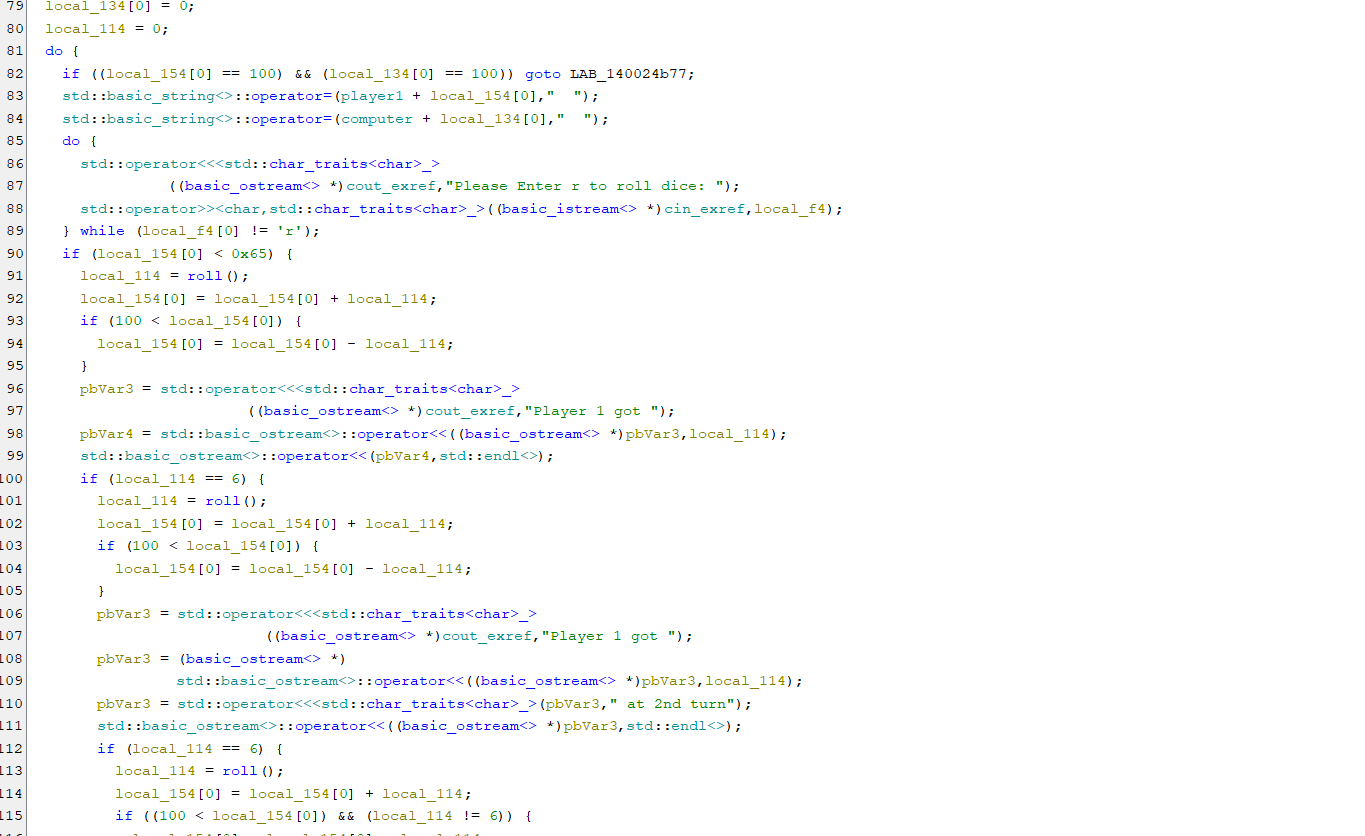
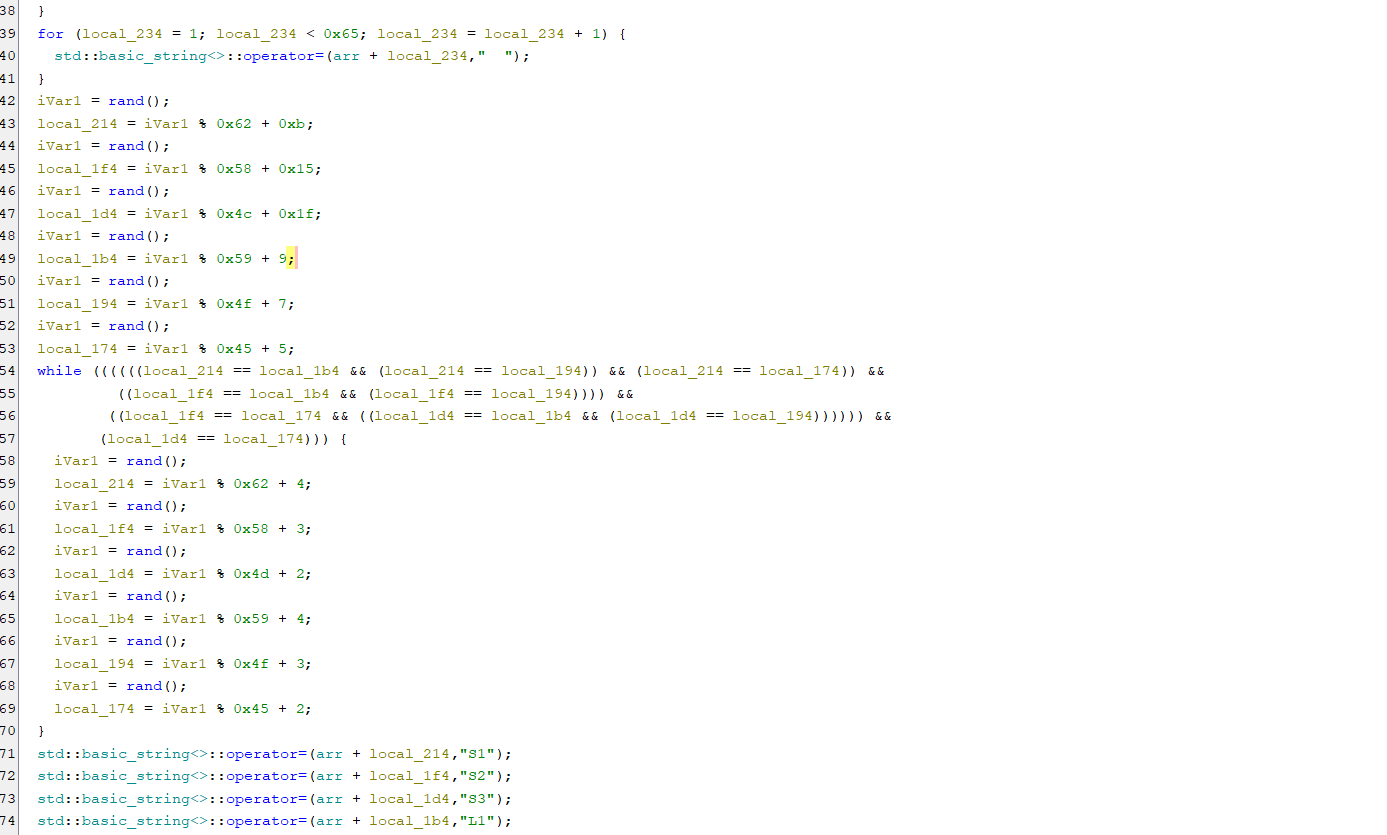
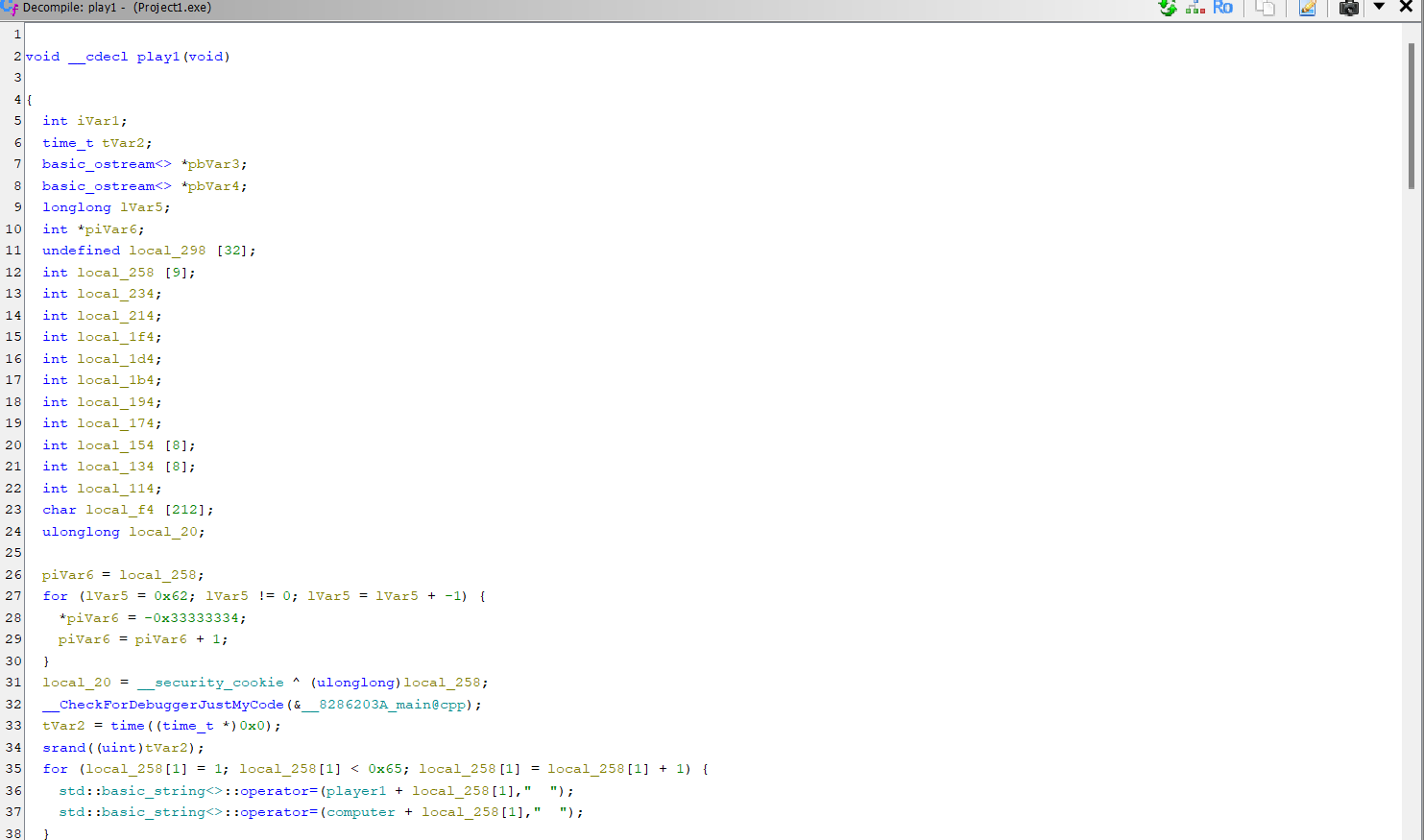


## **Memory Heap**



## **Code Extraction**

Extracted Code for play function

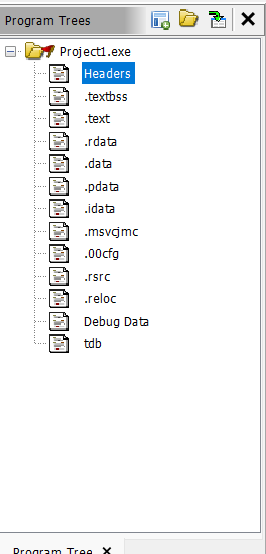


Extracted Code for game function

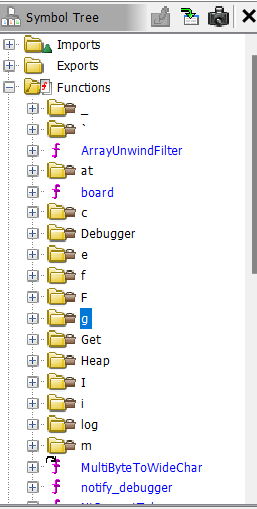
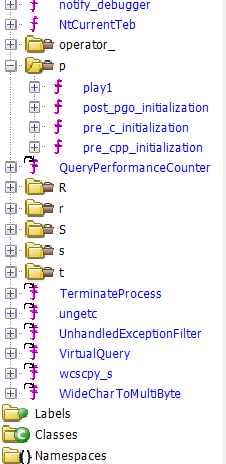




## **Program Tree**



## **Symbol Tree**

## **Findings on Software Architecture and Functionalities**

### **Software Architecture**

1. **Headers and Libraries:**
   * The code includes several libraries such as:
     + iostream: For input and output operations.
     + cstring and cstdlib: For string manipulation and standard functions.
     + ctime: For random number generation.
     + windows.h and unistd.h: For platform-specific operations like cursor positioning and sleep functionality.
2. **Function-Based Structure:**
   * The program is structured around multiple functions rather than classes, making it a procedural-oriented application.
   * Key functions identified:
     + gotoxy(int x, int y): Controls console cursor positioning for display formatting.
     + logo(): Likely displays the game's logo.
     + removeAnimation(): May handle visual effects or cleanup tasks.
     + play1(): Could be the primary game loop or logic handler.
     + board(): Displays or manages the game board.
     + roll(): Implements dice rolling functionality.
     + checklocation(...): Evaluates player positions and applies game rules like snakes and ladders.
     + instruction(): Provides game instructions.
     + game(): Likely encapsulates the main gameplay logic.
     + login(): Handles user login, potentially for player identification.
3. **No Classes:**
   * The program doesn't appear to use object-oriented programming (OOP) principles such as classes.
   * This suggests the game is implemented with a procedural paradigm.
4. **File Size and Scope:**
   * The code spans 972 lines, indicating a substantial implementation, possibly including visual effects, user interactions, and gameplay mechanics.

### **Functionalities**

1. **Game Mechanics:**
   * Implements core Snake and Ladder gameplay, including:
     + Dice rolls.
     + Player movement.
     + Interaction with board elements (snakes and ladders).
2. **User Interaction:**
   * Likely includes login functionality (login (), registration ()) and interactive instructions (instruction()).
3. **Visual Effects:**
   * gotoxy () and possibly removeAnimation () contribute to console-based visuals for enhanced user experience.
4. **Randomization:**
   * The use of ctime suggests randomness in dice rolls or other game events.
5. **Console-Based Interface:**
   * The game operates entirely within the console, leveraging windows.h for cursor control and formatting.