Department of Electronic Engineering

Royal Holloway, University of London

EE1010: Programming in C++

NAME: Aymen Mtibaa, year 1, 07/01/24

Game Description:

"TRY TO ESCAPE" challenges players with navigating mazes of varying difficulty (easy, average, hard) with 1 to 3 players. Find the exit marked 'O,' collect 'B' bonuses for extra moves and 25 points. Players move once per turn unless a bonus is obtained. Successfully escaping awards 50 points, and correct moves earn 10 points each.

Controls and Gameplay:

Player 1(P): w(up) / s(down) / a(left) / d(right)

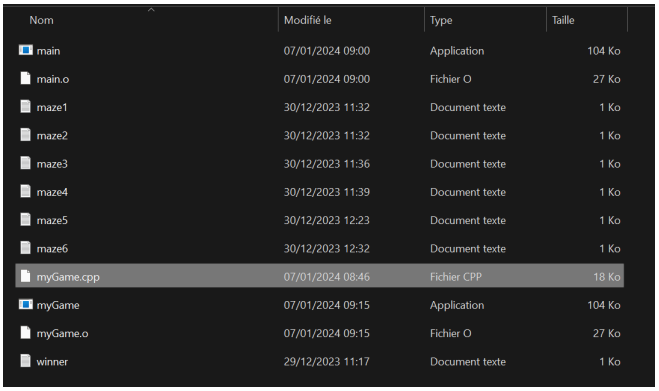
Player 2(Q): i(up) / k(down)/ j(left) / l(right)

Player 3(R): g(up) / b(down)/ v(left) / n(right)

1-User manual:

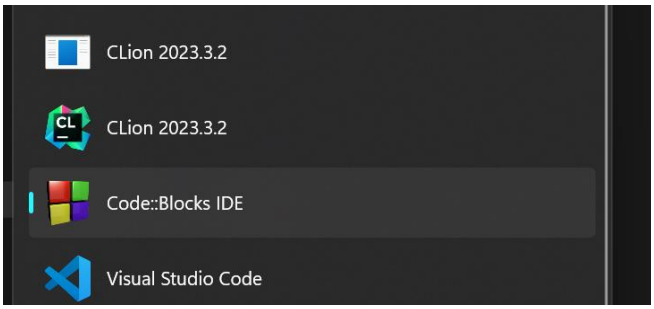
The program was developed on Windows 11 using [Code::Blocks]. Open a command prompt or terminal in the directory containing the game's .cpp file. Compile the cpp file and then run the compiled executable. Make sure all maze files (maze1, maze2, maze3, maze4, maze5, maze6, winner) are in the same folder. Additionally, confirm that Mingw-w64 is installed on your system.

1/Opening the Zip File and Clicking on the CPP File

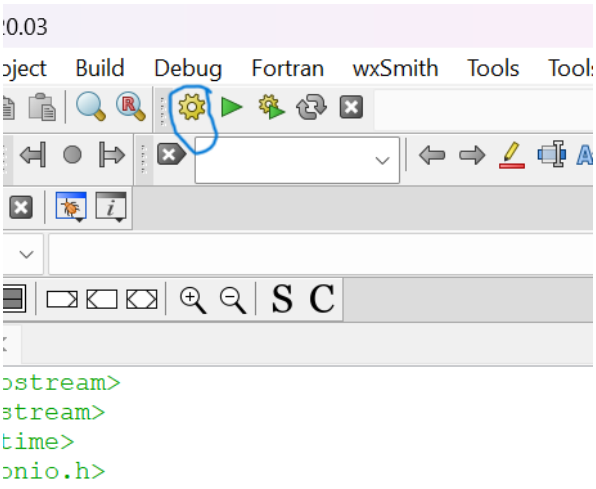


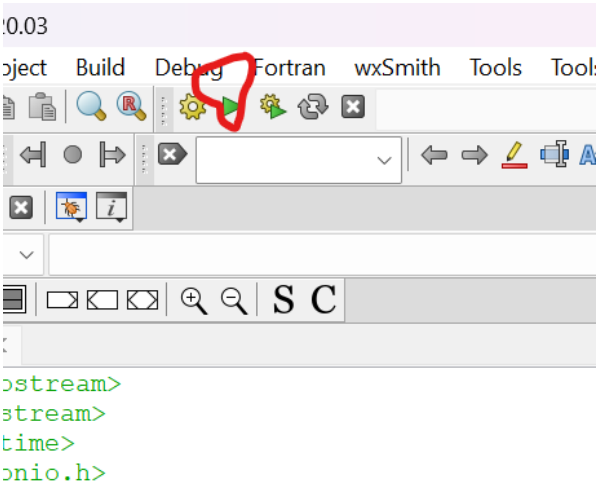
2/For optimal performance, it is recommended

to choose Code::Blocks as the compiler.



3/Compiling and Running the CPP File





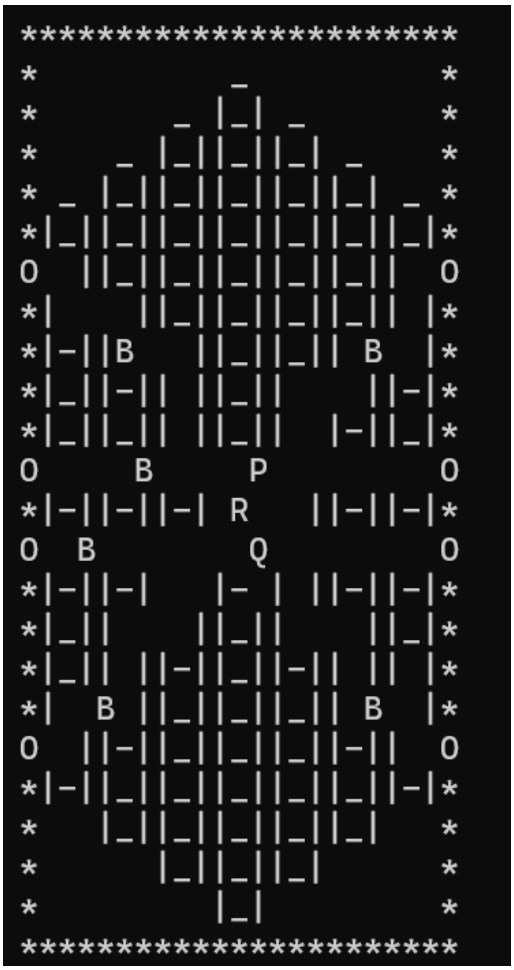
4/If all steps are executed correctly, a terminal will appear with a brief game description. You'll be prompted to input the number of players and select the difficulty level.

```
**********************************************************Title: TRY TO ESCAPE**********************************************************
Welcome to my game!
-Immerse yourself in a captivating 2D maze .The key to your escape lies in navigating your way out by uncovering paths marked with 'O'.
-Consuming a bonus marked with the letter'B' grants you an extra movement
-Additionally, every bonus consumed contributes to your score, with 25 points awarded for each bonus successfully eaten.
-If you successfully escape the maze, 50 points will be added to your score, 10 points are awarded for every correct move.
*************************************************************************************************************************************
Please specify number of players (1, 2, or 3):
```

5/Upon entering player count and difficulty,

a random maze will be generated.

Enjoy the game experience!



6/Upon winning, a large "Winner" message will display, featuring the victorious player on top.

Scores for each player will be presented.

The program will suggest a replay, and pressing 'Y' initiates another game.

7/If choosing not to play again and pressing 'N', a prominent "Bye" message will be displayed

2-Code development:

In the programme, a variety of functionalities was integrated to elevate the gaming experience. From the basic features list, I implemented a compelling storyline with multiple difficulty levels, a cooperative mode featuring keyboard button assistance and bonuses, and a turn-tracking data structure that compiles a game summary, showcasing the score of each player at the end.

Furthermore, I explored advanced features, enabling a solo player to engage with the computer in a distinctive manner. Recognizing the challenge of having the computer navigate the maze without an AI library, I offered players the option to experience the game individually, attempting to escape the maze on their own.

The game's flexibility shines through in supporting 1 to 3 players, providing diverse gaming scenarios. The visual representation of the 2D board, stored in a two-dimensional array, was achieved by directly manipulating the data. I employed 'ifstream' to visualize the maze matrix and to read the winner file, displaying the "Winner" message at the game's conclusion.

One particularly ingenious segment of the code is the 'display_maze' function. Although concise, this function employs a nested loop (with 'i' as the x-axis and 'j' as the y-axis) to efficiently handle the matrix. This implementation, while seemingly straightforward, played a pivotal role in unlocking the potential for additional features. It's worth noting that this achievement was especially gratifying considering my initial plan for a simple text game.

A major turning point occurred when introduced to the power of 'getch()' and the capabilities of the 'conio.h' library by a classmate. Inspired by these tools, I dedicated a month to self-learning through YouTube tutorials videos. While time constraints prevented further additions, this experience underscored the project's evolution from a basic text game to a dynamic 2D game.

-code snippet:

While it may be impossible to explain a small part of the code without including the entire codebase, as each function is utilized in the main program and other functions, for brevity and not crossing the maximum words allowed, I'll focus on the most essential parts of the features.

1/Basic features:(multiple level of difficulty)

```
do{//ask the user about the difficulty of the game
      cout<<"Select the game Difficulty: 1 for easy , 2 for average ,3 for hard :";
   cin>>diff;
  }while(!(diff>=1 && diff<=3));//keep asking the difficulty until the user puts a number between 1 and 3
   if (diff==1){
      level=rand()%2+1;//if diff=1 we choose either maze 1 or 2
   }
   else if (diff==2){
      level=rand()%2+3;//if diff=2 we choose maze 3 or 4
   }
```

```
    else {
        level=rand()%2+5; //if diff=3 we chose maze 5 or 6
    }
    show_maze(("maze"+to_string(level)+".txt").c_str(), maze);//converting the random number
//generated to a string and add the word maze in front and .txt in the end so it matches the file name
```

2/Advanced features(the screen represents a 2D board, kept in a two-dimensional array, which data is directly manipulated by the program):

```
//function to display the maze
void display_maze(char maze[max_data_size][max_data_size]) {
    for (int i = 0; i < max_data_size; ++i) {// starting a loop function to display the x-axis as i
        for (int j = 0; j < max_data_size; ++j) {//starting a for function to display y-axis as j
            cout << maze[i][j] ;//we display each line and collumn
        }
        cout <<endl;//we return to line, so the next message doesn't show next to the maze
    }
}
```