VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE

——————— o0o ———————

PROJECT REPORT: MATCHING GAME

SUBJECT: PROGRAMMING TECHNIQUES

Students : 22127286 - Nguyen Thanh Nam
22127426 - Dinh Duy Triet

Class : 22CLC10

Instructors : Nguyen Thanh Phuong
Nguyen Ngoc Thao
Bui Huy Thong

Ho Chi Minh - 2023

Table of Contents

Chapter 1

TUTORIALS

- Provide a brief overview of the steps in this game.
- The in-game utilities that users can use throughout their experience.

1.1 Game Steps

- This is a diagram that simulates all the steps of the game:

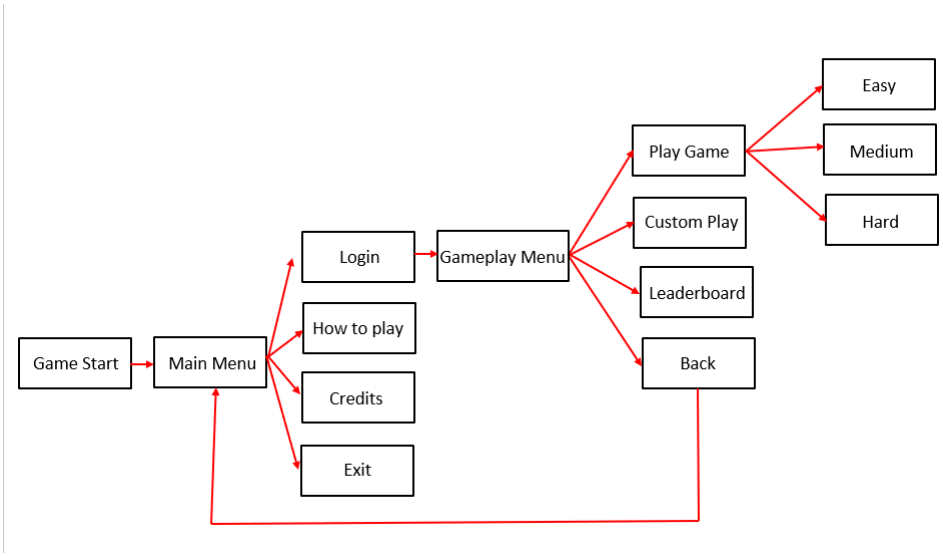

Image 1.1.1: Game Steps

1.2 Game Walkthroughs

- When the user starts the game, at the main menu, they will have 4 options:
  - **Login:** Enter gameplay.
  - **How to play:** How to move, game rules and scoring methods in the game.
  - **Credits:** How to move, game rules and scoring methods in the game.
  - **Exit:** Exit the game.

- Once the user selects **Login**, they will be redirected to the login interface and prompted to enter their email address in the correct format to proceed.
- Next, the user will be taken to the gameplay interface. Here, they will also be provided with 4 options:
  - **Play Game:** User can choose 1 of 3 available modes:
    - Easy: ...
    - Medium: ...
    - Hard: ...
  - **Custom Play:** Users can set up their own game board by entering the size of rows and columns, and choosing whether they want their board to be automatically shuffled after each move or not.
  - **Leaderboard:** Users can view the results of the top 5 players with the highest scores achieved so far.
  - **Back:** Return to the main menu.

***GAME RULES:**

- Both ”Play Game” and ”Custom” gameplay are based on the same classic pattern.
- Gameplay consists of a board containing multiple cells representing characters or letters.
- The objective of the player is to select two cells with matching characters or letters that can be connected by 1 of 4 patterns: I, L, U, and Z.
- Validly connected patterns will make the two cells disappear.
- The game will end if the player clears all the cells on the board or runs out of 5 lives.

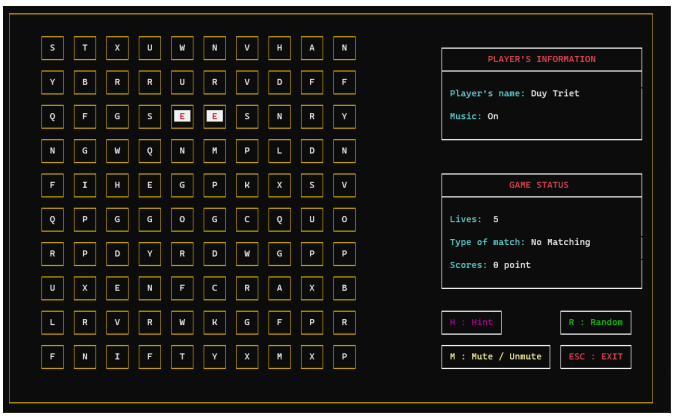

Image 1.2.1: Gameplay Interface

Chapter 2

FUNCTION

- How we organize files in the project.
- Explain how the functions work.
- List all the libraries and built-in functions we used in our project.

2.1 File Structure

The source code of the project is divided in total into 8 .cpp files and 8 header files.
Each file includes functions to perform tasks as its given filename

1/ Menu.cpp:

- This file contains functions to generate complete interfaces and link them together with game files in this project.
- We have implemented the following function: void MainMenu();
→ This is the main function that will handle the entire main menu interface. It includes functions for printing animated text, displaying images, and processing the options selected by the user in this interface.

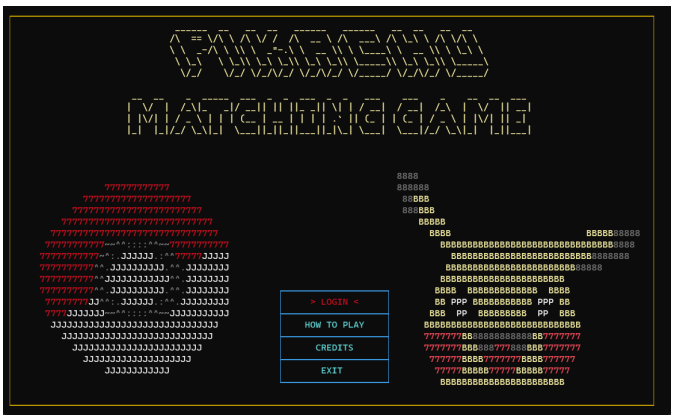

Image 2.1.1: Game starting

* Below, we will provide a more detailed explanation of the structure of this function:

void **printMenu**(selection, x, y, w, h, textColor, buttonColor, backgroundColor, text);
→ This is a sub-function in "MainMenu" function that defined some other functions for printing animated text and coloring images. These functions will be explained in detail in "Color effects" and "Visual effect" section later on.

- The next step is to implement the method for highlighting the position of the option when the user uses arrow keys to navigate through the menu.
→ We use a loop to get the user's input character and an integer variable called "selection" initialized with a value of 1. When the user presses the 'W' key (move up), the "selection" variable will be incremented by 1, and it will be decremented by 1 if the user presses the 'S' key (move down). Since the main menu only has 4 options, when the "selection" variable is 0, it will be reassigned to 4, and when it is 5, it will be set back to 1.

- The "selectionMenu" is a subfunction within the "printMenu" function that handles the information about the position of the selected option by the user:
*void selectionMenu(int selection, int x, int y, int w, int h, int textColor, int buttonColor, int backgroundColor, string text);
→ The task of the "selectionMenu" function is to change the text color of the first option to red when the "selection" variable has a value of 1, while keeping the text color of the other options as cyan. Similarly, it will change the text color of the second, third, or fourth

option to red when the ”selection” variable has a value of 2, 3, or 4, respectively.
→ This simulates a cursor moving along the menu options based on the directional keys input by the user.

- Based on the value of the selection variable, the switch-case statement will be used to determine which option the user has selected when they press the ”Enter” key. Each case in the switch statement contains the corresponding commands to execute the selected option of the program.
- Two other sub-functions void LoginMenu(); (to generate menu interface after successful login) and void GamePlayMenu(); (menu for selecting the gameplay mode) also rely on a similar algorithm as in the MainMenu function to execute.

2/ User.cpp:

1/ Main.cpp:
- This is the main part of the program.
- Header file included: **Menu.h**.

2/ Setup.cpp:
- In this file, we install functions to set up the interface of the game such as: drawBorder, button (create buttons with content inside), box (draw cells for the game board), createScreen, clearScreen...
- Besides, there are also other supporting utilities in setting up the user experience: SetWindowSize, SetScreenBufferSize, DisableCtrButton, ShowScrollbar, showCursor, moveCursor, disableMouseInput, consoleColor, TextColor...
- Header file included: **Setup.h**.

3/ User.cpp:
- The functions for reading, saving, and processing user's information are defined in this file: verify (check the correctness of the email entered by the user.), comparePlayers, pushRecord, printLeaderboard.
- Header file included: **User.h**.

4/ Graphic.cpp:
- This file contains definitions of functions for reading artistic letters from text files and custom functions for handling images that appear in the game: art_at_pos, read_file_at_pos, pokemon_ball, login_icon...
- Header file included: **Graphic.h**.

5/ Check.cpp:
- Here, we define functions to check matching pattern motifs of I, L, Z, U, function to check whether the user wins or not and function to suggest moves to the user: check_I, check_L, check_Z, check_U, checkGameWin, moveSuggestion,...
- Header file included: **Check.h**.

6/ Board.cpp:
- This file includes all the functions that generate the game board, notify the users when they make a right move or not, and also suggest a move: Board::drawBox, createBoard, retreatBoard, trueMatch, wrongMatch, suggestMatch,...

- Header file included: **Board.h**.

7/ Game.cpp:

- The functions in the this file, which are synthesized from the functions in the Check.cpp and Board.cpp files, are used to set up a complete interface for gameplay. In addition, they also provide an interface for users to input their names and customize the game board: gameLoop, playerInformation, normalForm, customForm, normalGame, customGame.
- Header file included: **Game.h**.

2.2 Library

- Data structures used in this program are arrays (used to hold the value of the cells), linked list and queue (used to find the path to two cells).

1/ Library: `<Window.h>`

- HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
- HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
- HWND hWnd = GetConsoleWindow();
→ Used to handle the cursor position on the screen.
→ If not using this, it will be hard to control the position of the cursor.

- void SetWindowSize(int width, int height);
- void SetScreenBufferSize(SHORT width, SHORT height);
→ Used to change the size of the console.
→ If not using this, there will be a bug when the board is too big compares to the screen.

- void moveCursor(int posX, int posY);
→ Used to put cursor to a desired position.
→ If not using this, we can't set the cursor to the position that we want.

- SetConsoleTextAttribute(hConsoleOutput, wAttributes);
→ Used to print text and background in color.
→ If not using this, there will be no color effect.

2/ Library: `<conio.h>`

- _getch()
→ Used to get a character entered by the user.
→ If not using this, the user will not be able to control the cursor using a key-board.

3/ Library: `<ctime>`

- clock_t
→ Used to calculate the time.
→ If not using this, we can't count the time of a gameplay.

- srand()
→ Used to initialize random number generator.
→ If not using this, we can't randomize the board.

Chapter 3

FEATURES

3.1 STANDARD FEATURES

3.1.1 Game starting

3.1.2 I, L, U, Z matching

- When the user selects two cells correctly in 1 of the 4 connection patterns (I, L, U, or Z), the system will display the selected connection pattern on the screen.

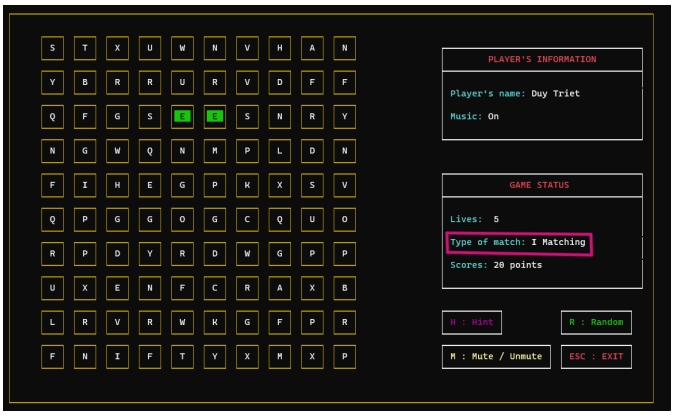

Image 3.1.2: Display matching pattern

3.1.3 Game finish verify

- When the user finishes a game level, the result will be displayed on the screen.

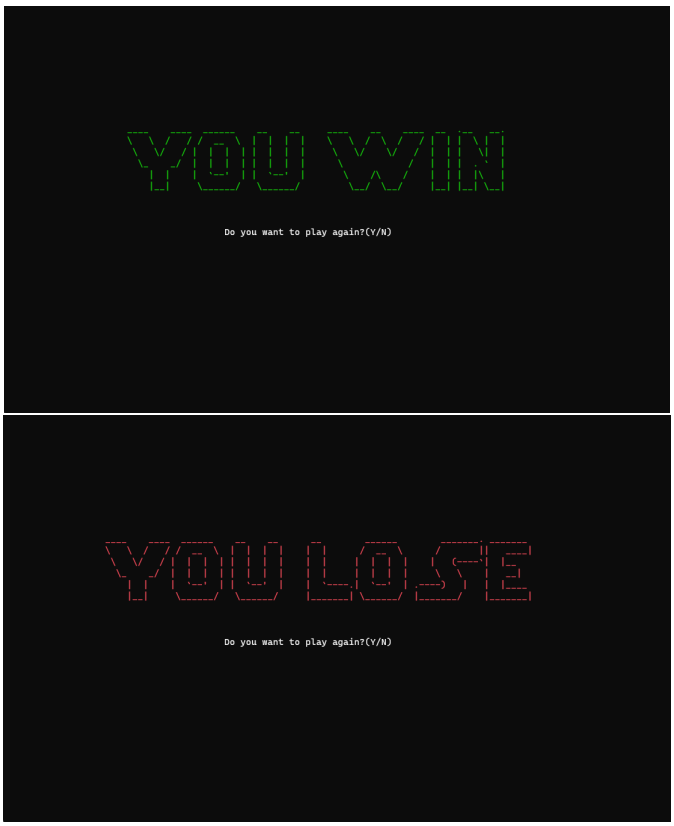

Image 3.1.3: Game finish verify

3.2 ADVANCED FEATURES

3.2.1 Color effects

- Interface: Throughout all stages of the game, we have added various color effects to make it more visually appealing and enhance the overall user experience.

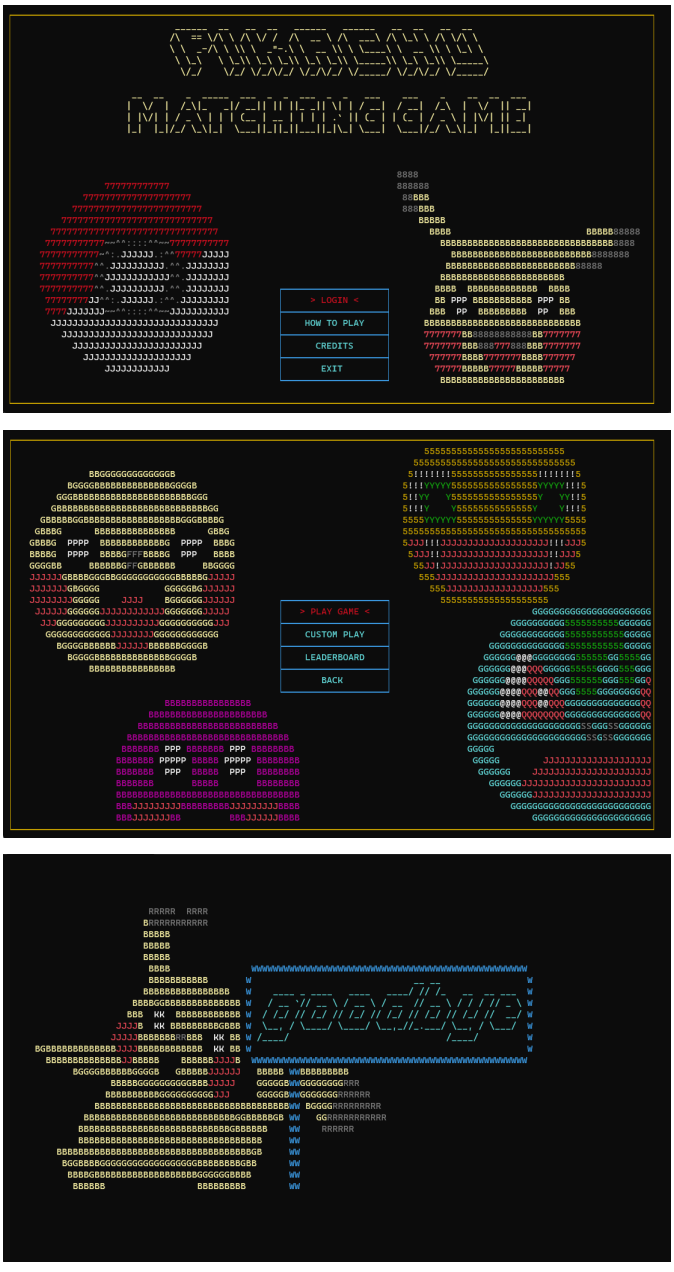

Image 3.2.1a: Color effects in interface

- Gameplay: Color will appear in these 4 cases:

. Selecting 2 cells → Light white.
. If 2 cells have the same value and a valid path → Green.
. If 2 cells don't have the same value or a valid path → Red.

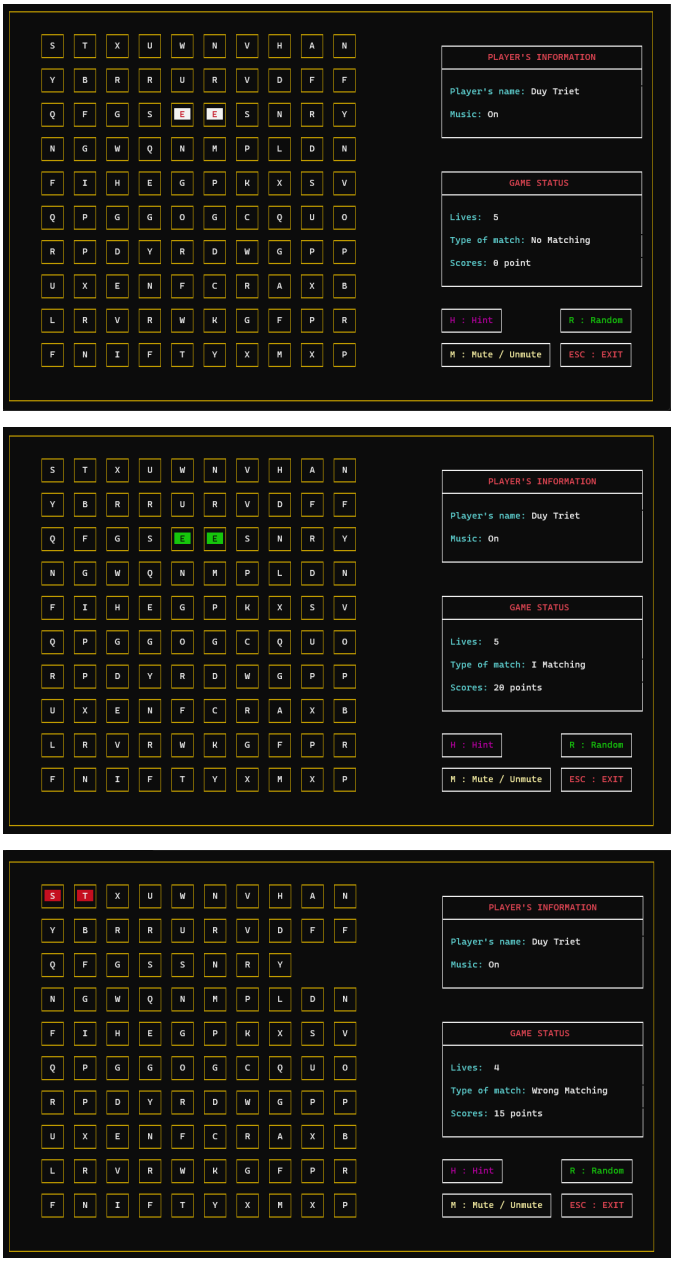

Image 3.2.1b: Color effects in gameplay

3.2.2 Visual effects

Image 3.2.2a: Main menu

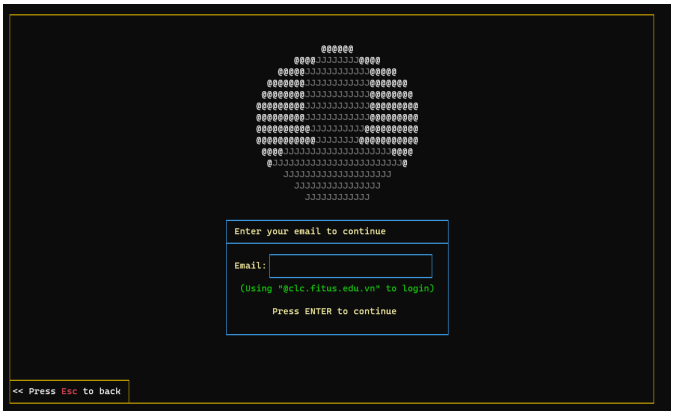

Image 3.2.2b: Login interface

Image 3.2.2c: Login Menu

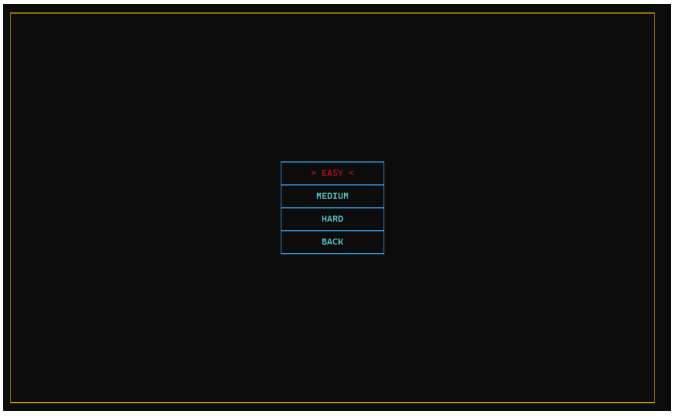

Image 3.2.2d: Gameplay modes

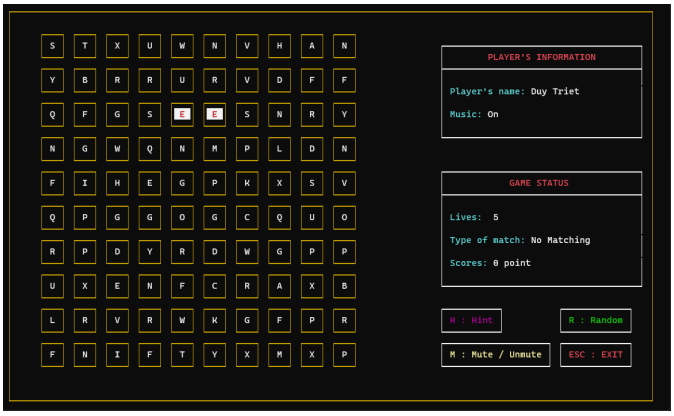

Image 3.2.2e: Gameplay

Image 3.2.2f: Exit

3.2.3 Sound effects

We mainly use this code to play music: PlaySound(TEXT("path"), NULL, SND_ASYNC);
→ To make the code run on Windows, we include the "-lwinmm" flag in the execution command.

3.2.4 Leaderboard

- The top 5 players, who have achieved the highest scores and completed their games in the shortest time, will have their information registered on the Leaderboard.
- Here is how we implement this feature:

In file: User.cpp

Function: bool comparePlayers(const Players &p1, const Players &p2);

- This function compares two players based on their points. If one player has higher points than the other, they will be swapped so that the player with higher points appears in front. If the points are equal, the player with shorter time will be considered.

Function: long long unsigned int fileSize(string file);

- This function uses fseek() and ftell() to find the total size of the binary file in bytes, so that we can calculate the number of players in the binary file by dividing the size of the file by the size of the struct Players.

Function: void pushRecord(Players p);

- This function is use to record high scores in write to binary file. There are 2 cases:

→ There are fewer than 5 players in the binary file. We will open the binary file in append and binary mode to write a new player to the end of the file. Next, we will reopen the binary file in binary mode and use std::sort() from the algorithm library to sort the players. Finally, we will open the binary file again in binary mode and write to the file once more.

→ There are 5 players in the binary file. We will open the binary file to read the data of the 5 players. After that, we will check if the new player has a better result than the fifth player in the binary file. If yes, we will replace the fifth player with the new one. Finally, we will sort the 5 players and write them back to the binary file.

Function: void printLeaderboard();

- This function is used to print the top 5 players by reading the datas from a binary file into a struct.

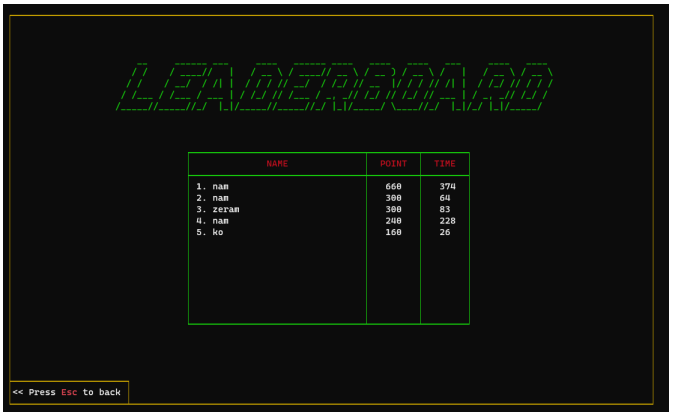

Image 3.2.4: Leaderboard

3.2.5 Move suggestion

- When the ”moveSuggestion” function is called by pressing ”h” or ”H”, the hint will be shown in magenta background color in order to signal to the hint.
- Here is how we implement this feature:

In file: Check.cpp

Function: bool moveSuggestion(Board **board, int _row, int _col, Point &p1, Point &p2);

- We will utilize 4 loops to identify valid pairs based on the boolean check provided by the function bool check_All(Board **board, int _row, int _col, int x1, int y1, int x2, int y2);. If the boolean check returns true, then there are valid pairs, which can be accessed by calling the 4 points p1.x, p1.y, p2.x, p2.y.

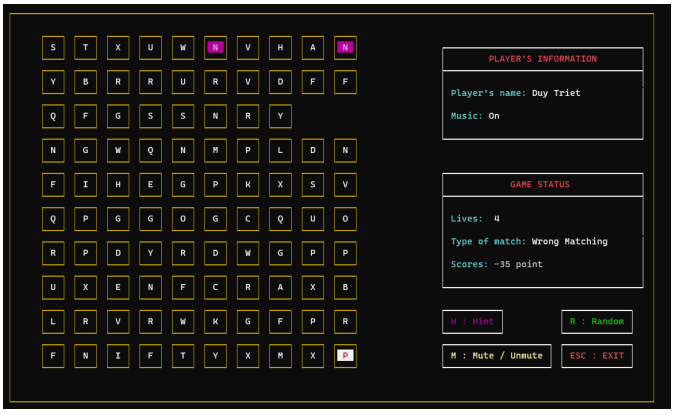

Image 3.2.5: Move suggestion

3.3 OTHERS FEATURES

3.3.1 How to play

3.3.2 Auto shuffle if no valid pairs

Chapter 4

REFERENCES

All the resources that we have consulted for the project.

1. Setup.cpp
   https://learn.microsoft.com/en-us/cpp/?view=msvc-170
   https://stackoverflow.com/questions/2347770/how-do-you-clear-the-console-screen-in-c
   https://cplusplus.com/forum/beginner/
   https://codelearn.io/sharing/windowsh-va-ham-dinh-dang-console-p1
   https://codelearn.io/sharing/windowsh-ham-dinh-dang-noi-dung-console
2. Menu.cpp
   https://cplusplus.com/forum/general/55170/
   https://www.youtube.com/watch?v=UjQIwlr_DqI
3. Check.cpp
   https://cachhoc.net/2014/03/25/thuat-toan-game-pokemon-pikachu/
4. Sounds we have use:
   https://www.youtube.com/watch?v=RUCn4w-S2KM
   https://www.youtube.com/watch?v=jAIlKqL3nHo
   https://mixkit.co/free-sound-effects/tap/
   https://www.followchain.org/cristiano-ronaldo-siuuu-sound-effect/
   https://mixkit.co/free-sound-effects/game/