VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF SCIENCE

——————— o0o ————————

PROJECT REPORT: MATCHING GAME

SUBJECT: PROGRAMMING TECHNIQUES

Instructors	:	Nguyen Thanh Phuong
		Bui Huy Thong
		Nguyen Ngoc Thao
Class	:	22CLC10
Students	:	22127286 - Nguyen Thanh Nam
		22127426 - Dinh Duy Triet

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:

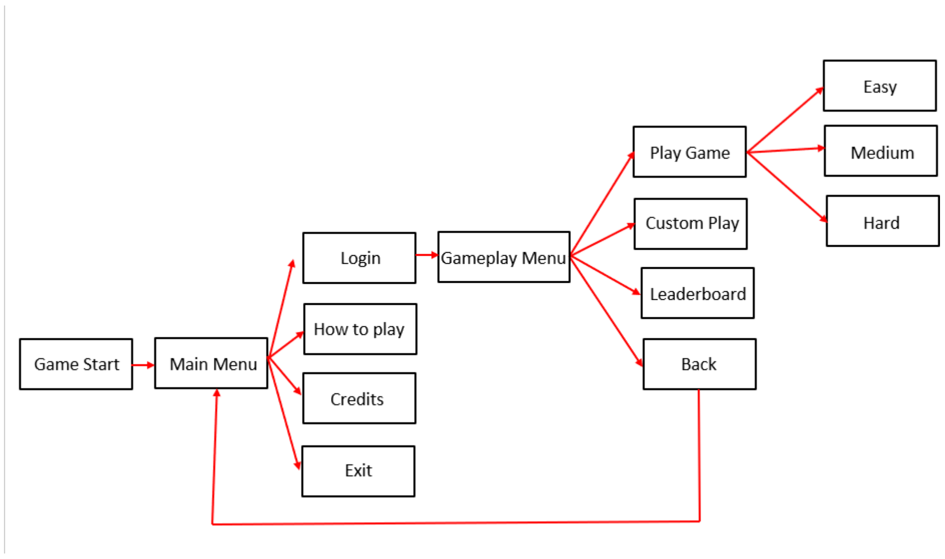


Image 1.1.1: Game Steps

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:** Information about project team members.
• **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: A 4x4 board, no shuffling after each play.
- Medium: A 8x8 board, shuffling after each play.
- Hard: A 10x10 board, shifting all blocks to the left.

• **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.

<u>*GAME TUTORIALS:</u>

- 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 player will use the WASD keys to move up, down, left, or right, and the Enter key to select a cell.
- 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.
- If the matching pattern is valid, user will be awarded 20 points. If it’s invalid, user will be deducted 5 points and lose 1 life. The player will also be deducted 5 points for each hint used.
- The game will end if the player clears all the cells on the board or runs out of 5 lives.

Chapter 2

FUNCTION

- How we organize files in the project.

- Explain how the functions work.

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

<u>1/ Menu.cpp:</u>

- 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.

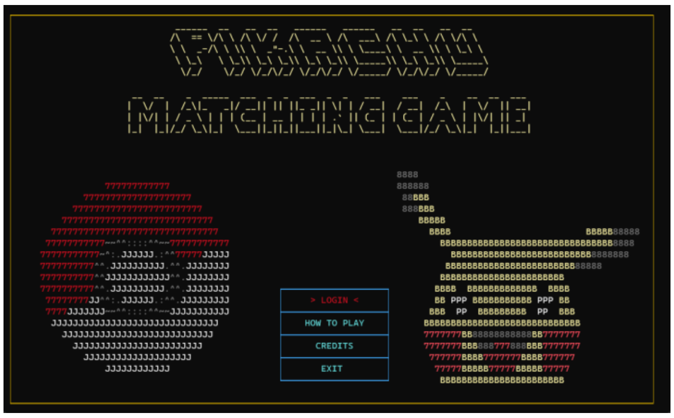


Image 2.1.1: Game starting

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

• void printMenu(int selection, int x, int y, int w, int h, int textColor, int buttonColor, int backgroundColor);
→ 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);
→ 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/ Setup.cpp:

- We have implemented functions to set up details and utilities on the game interfaces. Here are some key functions that we use:

• void button(int x, int y, int w, int h, int color, int buttonColor, int backgroundColor);
→ The ”button” function creates a graphical button on the console screen with the provided parameters for position, size, text color, button color, and background color.
- It first sets the console text color and background color using the ”consoleColor” function, and then loops through the coordinates within the button’s area (from x + 1 to x + w - 1 for x-coordinate, and from y + 1 to y + h - 1 for y-coordinate) and clears the content by printing a space character.
- Next, the function sets the cursor position to the top-left corner of the button’s area (at x + 1, y + 1) and sets the text color using the ”TextColor” function, and prints the text parameter.
- The function then sets the console color and text color again, and draws the outline

of the button using characters such as '-' (char(196)), '—' (char(179)), '+' (char(218), char(191), char(192), char(217)), which form a rectangular shape around the button's area.

• void createScreen();
→ A function called "createScreen" creates a screen layout with a border and a back button. The function appears to call three other functions: "clearScreen", "drawBorder", and "drawBackButton".
• **"clearScreen":** This function clears the contents of the screen to prepare for drawing the new screen layout.
• **"drawBorder":** This function draws a border around the screen, creating a visual boundary for the screen layout, based on the "drawRectangle" function.
→ **"drawRectangle":** Here is the detailed algorithm of this function:

- The function takes four integer parameters as input: "left" and "top" representing the starting position (coordinates) of the top-left corner of the rectangle on the screen, and "width" and "height" representing the width and height of the rectangle, respectively.
- The function uses the "moveCursor" function to set the cursor position on the screen to the top-left corner of the rectangle.
- The function uses the "putchar" function to output the ASCII character '218', which represents the top-left corner of the rectangle.
- The function then uses a loop to output the ASCII character '196', which represents the horizontal line of the rectangle, for "width" number of times, effectively drawing the top edge of the rectangle.
- The function outputs the ASCII character '191', which represents the top-right corner of the rectangle, to complete the top edge of the rectangle.
- Next, the function uses another loop to draw the vertical lines of the rectangle. It outputs the ASCII character '179', which represents the vertical line of the rectangle, at the left and right positions of the rectangle for "height" number of times, effectively drawing the left and right edges of the rectangle.
- The function then outputs the ASCII character '192', which represents the bottom-left corner of the rectangle, at the bottom-left corner of the rectangle, using the "moveCursor" function.
- Another loop is used to draw the bottom edge of the rectangle by outputting the ASCII character '196', which represents the horizontal line of the rectangle, for "width" number of times.
→ Finally, the function outputs the ASCII character '217', which represents the bottom-right corner of the rectangle, at the bottom-right corner of the rectangle, completing the drawing of the filled rectangle.

• void art at pos(string fileName, int textColor, int backgroundColor, int x, int y);

- The task of this function is to create artistic letters from an existing text file.
→ This function will be explained in detail in the "Visual effects" section.

* In addition, we also utilize several other functions defined in the "windows.h" library, such as: SetWindowSize, ShowScrollbar, showCursor, moveCursor, disableMouseInput, TextColor,...

3/ User.cpp:

• long long unsigned int fileSize(string file);
→ Function takes a file name as input, opens the file in read mode, and returns the size of the file in bytes.

• void pushRecord(Players p);
→ This function is used to push a player record to a binary file ”players.bin” while maintaining a sorted order based on player points and time:

- In addition to that, there are two other sub-fuctions:
    • bool comparePlayers(const Players &p1, const Players &p2);
→ This function is used to compare the results of two players based on their points and time. It takes in two objects of type Players as input, compares their points and time fields, and returns true if the first player has higher points or if their points are equal but their time is lower than the second player, and false otherwise.

    • void printLeaderboard();
→ This function is used to read data from the binary file ”players.bin” and display the leaderboard on ”Leaderboard” interface.

4/ Graphic.cpp:

• void read file at pos(string fileName, int textColor, int backgroundColor, int x, int y);
→ The function ”read_file_at_pos” reads the contents of a file specified by ”fileName” parameter at a specific position on the console window. It uses the ”textColor” and ”backgroundColor” parameters to set the text color and background color for displaying the contents. The ”x” and ”y” parameters specify the coordinates of the starting position for displaying the contents.
→ The function opens the file using an input file stream (”ifstream”) and reads its contents line by line using ”getline” function. For each line, it moves the cursor to the specified position using ”moveCursor” function, sets the console color using ”consoleColor” function, sets the text color using ”TextColor” function, and then prints the line of text to the console using ”cout” statement. After printing all the lines, the function closes the file using ”file.close()” statement.

* During the game development process, we have used many images to make the game interface lively and visually appealing. These images are implemented using separate functions, and their algorithms will be explained in the ”ADVANCED FEATURES” chapter.

5/ Check.cpp:

- The functions defined here are used to check the validity of matching patterns in the gameplay. They will be explained in detail about the algorithms in the following chapter.

6/ Board.cpp:

- The file contains function to generate the game board for the user as well as the algorithm to randomly rearrange the order of cells in the board. They will be explained in detail about the algorithms in the following chapter.

7/ Main.cpp:

- This is the main part of the program. It will run the entire program.

2.2 Program Executing Instruction

- We offer two options for executing the program:

Option 1: To compile and link the program via command line, type the following command:
'g++ -std=c++11 Main.cpp Graphic.cpp Menu.cpp Setup.cpp User.cpp Board.cpp Check.cpp Game.cpp -o main.exe -lwinmm'.

Option 2: To run the program, use the *'run.bat'* file.

Chapter 3

FEATURES

3.1 STANDARD FEATURES

3.1.1 Game starting

- This is how we create the game board and ensure that the occurrences of each character on the board are even.

In file: Board.cpp

Function: void createBox();

- We create a 2D array of characters to generate a square box on the game board, utilizing ASCII characters.

Function: Board::drawBox(Board **&board);

- In the "struct Board", we have created a method to change the status of a block, which includes the states of unselected block, selected block, and terminated block.

Function: Board **randomize(Board **&board, int row, int col);

- This function demonstrates how we shuffle the characters. First, we create a 1D array from a vector:

```
Create a vector of Board objects called vec
For i = 0 to _row − 1
    For j = 0 to _col − 1
        Add board[i][j] to vec
```

- Next, we utilize the std::random shuffle() function from the algorithm library to shuffle the characters randomly in the 1D array:

```
random_shuffle(vec.begin(), vec.end())
```

- After shuffling the characters in the 1D array, we push them back onto the game board:

```
index = 0
For i = 0 to _row − 1
    For j = 0 to _col − 1
        Set board[i][j] to vec[index]
        Increment index
```

Function: Board **createBoard(Board **&board, int row, int col);

- First, we declare a dynamically allocated 2D array in C++ using the new operator.
- To ensure that the occurrences of a specific character on the game board are even, we utilize the following algorithm, which is divided into two cases:

Case 1: When the value of the column is odd, we assign the first character in the 2D array of characters to be equal to the last character in the 2D array of characters, and continue with this pattern.

```
For i = 0 to _row / 2
    For j = 0 to _col − 1
        board[i][j].ch = 65 + rand() % 26
        board[_row − i − 1][_col − j − 1].ch = board[i][j].ch
    End For
End For
```

Case 2: When the value of the column is even, we assign the first character in the 2D array of characters to be equal to the last character in the 2D array of characters, both of them are in the same row and continue with this pattern.

```
For i = 0 to _row − 1
    For j = 0 to _col / 2
        board[i][j].ch = 65 + rand() % 26
        board[i][_col − j − 1].ch = board[i][j].ch
    End For
End For
```

- After applying the algorithm to ensure even occurrences of a specific character on the game board, we call the ”randomize(board, _row, _col);” function to shuffle all the characters in different positions. Then, we set all blocks to unselected blocks.

3.1.2 I Matching

- The main idea is to divide the matching pattern into two cases:

Case 1: When two points have the same row: Check if there is any barrier between 2 points:

In file: Check.cpp

Function: bool checkLineX(Board **board, int y1, int y2, int x);

- Here is the pseudocode code:

```
min_val = min(y1, y2)
max_val = max(y1, y2)
For y = min_val + 1 to max_val − 1
    If board[x][y].ch != ' ' Then
        Return false
    End If
End For
Return true
```

Case 2: When two points have the same column: Check if there is any barrier between 2 points:

In file: Check.cpp
Function: bool checkLineY(Board **board, int x1, int x2, int y);
- Here is the pseudocode code:

```
min_val = min(x1, x2)
max_val = max(x1, x2)
For x = min_val + 1 to max_val − 1
    If board[x][y].ch != ' ' Then
        Return false
    EndIf
End For
Return true
```

- The function: bool check I(Board **board, int x1, int y1, int x2, int y2); is created by merging the two functions mentioned above to check for I Matching.

3.1.3 L Matching

- The main idea is to find the "Intersection" of two points (Point A and Point B) on the board, denoted as Point C. Then, we use "checkLineX" and "checkLineY" functions to check the two paths from Point A to Point C and from Point C to Point B, respectively.
In file: Check.cpp
Function: bool check L(Board **board, int x1, int y1, int x2, int y2);
- Here is the pseudocode:

```
If board[x2][y1].ch == ' '
    AND checkLineX(board, y1, y2, x2)
    AND check Line Y(board, x1, x2, y1) Then
        Return true
Else If board[x1][y2].ch == ' '
    AND checkLineX(board, y1, y2, x1)
    AND check Line Y(board, x1, x2, y2) Then
        Return true
Else
    Return false
End If
```

3.1.4 Z Matching

- The main idea is to check three lines that form a Z-shaped pattern, called line A, line B, and line C. This can be divided into two cases:
In file: Check.cpp
Case 1:
Function: bool checkRectX(Board **board, int x1, int y1, int x2, int y2);
- The function checks if line A and C are parallel and if they are horizontal lines.
- It uses a loop to iterate over columns and find possible moves. In the loop, it checks three lines.
- It uses the checkLineX function to check for possible moves in horizontal line A and

line C. Additionally, it uses the checkLineY function to check the vertical line B. If all three lines create a valid path, the boolean statement returns true.

Case 2:
Function: bool checkRectY(Board **board, int x1, int y1, int x2, int y2);
- The function checks if lines A and C are parallel and if they are vertical lines.
- It uses a loop to iterate over rows and find possible moves. In the loop, it checks three lines.
- It uses the checkLineY function to check for possible moves in vertical lines A and C. Additionally, it uses the checkLineX function to check the horizontal line B. If all three lines create a valid path, the boolean statement returns true.

Function: bool check Z(Board **board, int x1, int y1, int x2, int y2); is created by merging the two functions mentioned above to check for Z Matching.

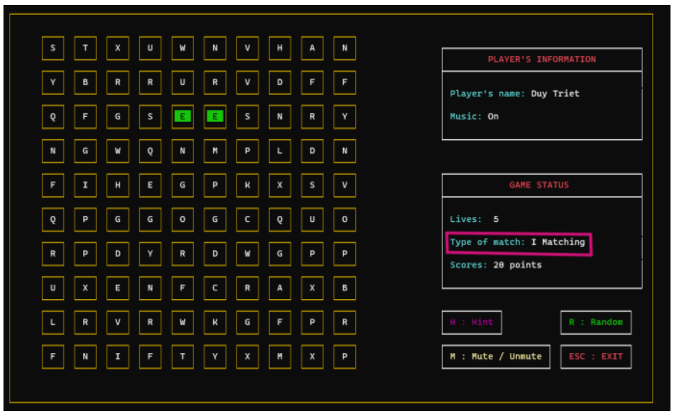
3.1.5 U Matching

- The main idea is to check three lines that form a U-shaped pattern, called line A, line B, and line C. This can be divided into two cases:
In file: Check.cpp
Case 1:
Function: bool checkMoreLineX(Board **board, int row, int col, int x1, int y1, int x2, int y2);
- The function checks if lines A and C are parallel and if they are horizontal lines.
- It uses a loop that runs from -1 to the value of the column to find possible moves. Inside the loop, it checks three lines.
- It uses the checkLineX function to check for possible moves in the horizontal lines A and C. Additionally, if line B is at column value or -1, then line B is considered possible. Otherwise, it uses the checkLineY function to check the vertical line B. If all three lines create a valid path, the boolean statement returns true.

Case 2:
Function: bool checkMoreLineY(Board **board, int row, int col, int x1, int y1, int x2, int y2);
- The function checks if lines A and C are parallel and if they are vertical lines.
- It uses a loop that runs from -1 to the value of the row to find possible moves. Inside loop, it checks three lines.
- It uses the checkLineY function to check for possible moves in the vertical lines A and C. Additionally, if line B is at row value or -1, then line B is considered possible. Otherwise, it uses the checkLineX function to check the horizontal line B. If all three lines create a valid path, the boolean statement returns true.

Function: bool check U(Board **board, int row, int col, int x1, int y1, int x2, int y2); is created by merging the two functions mentioned above to check for U 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.



```
S T X U W N V H A N
Y B R R U R V D F F
Q F G S E E S N R Y
N G W Q N M P L D N
F I H E G P K X S V
Q P G G O G C Q U O
R P D Y R D W G P P
U X E N F C R A X B
L R V R W K G F P R
F N I F T Y X M X P
```

PLAYER'S INFORMATION

Player's name: Duy Triet

Music: On

GAME STATUS

Lives: 5

Type of match: I Matching

Scores: 20 points

Image 3.1.5: Display matching pattern

3.1.6 Game finish verify

- If there are no cells left, the player will win the game. However, if the player's lives reach 0, the game will end immediately and the player will lose.
- Here is how we have implemented this feature:

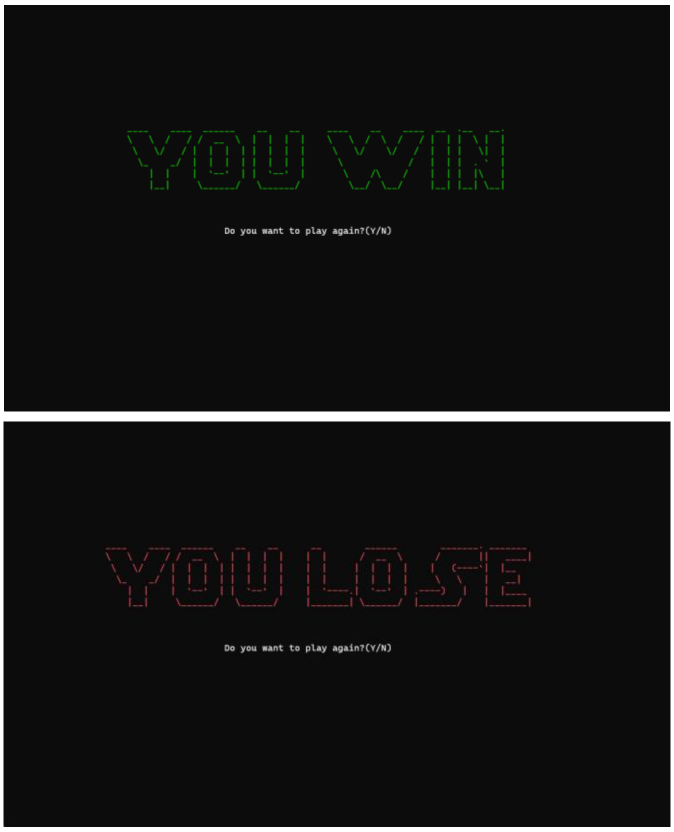
In file: Check.cpp

Function: bool checkGameWin(Board **board, int row, int col);

- The function uses 2 loops to check if there are any cells left. If cells are found, the boolean value returns false, and the game continues. However, if no cells are left, the boolean value returns true, indicating that the player wins the game.
- Here is the pseudocode code:

```
For i = 0 to row − 1
    For j = 0 to col − 1
        If board[i][j].ch != ' ' Then
            Return false
        End If
    End For
End For
Return true
```

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



```
Do you want to play again?(Y/N)
```

```
Do you want to play again?(Y/N)
```

Image 3.1.6: Game finish verify

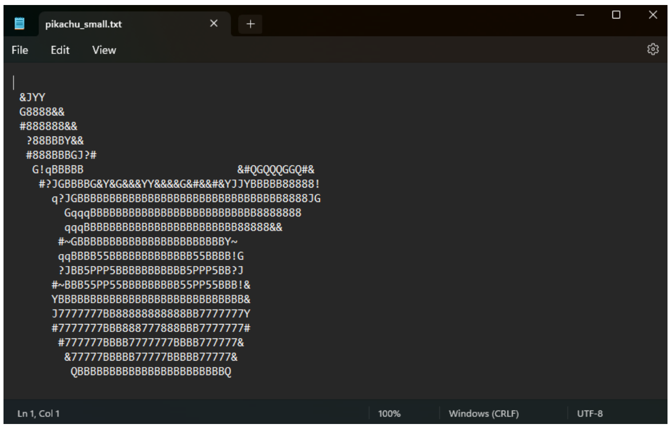
3.2 ADVANCED FEATURES

3.2.1 Color effects

* To set the color for objects on the interface, we primarily rely on the function void TextColor(WORD color);. This is a C++ function that sets the text color in the console window using Windows API functions. It takes a WORD parameter color which specifies the desired color for the text:

- It first retrieves the handle to the console output buffer using **GetStdHandle(STD_OUTPUT_HANDLE)** function and stores it in **hConsoleOutput** variable.
- It then retrieves the current attributes of the console screen buffer using GetConsoleScreenBufferInfo function and stores it in screen_buffer_info variable.
- Next, it extracts the current attributes of the text by masking out the color bits using bitwise AND (&) operation with 0x000f and stores it in color variable.
- It also masks out the color bits from the current attributes of the console screen buffer by bitwise AND operation with 0xfff0 and stores it in wAttributes variable.
- It then sets the text color by bitwise OR (—) operation between color and wAttributes and passes the result to SetConsoleTextAttribute function to update the text color in the console window.

* In addition, the images are also drawn using functions that we defined. Each image has its own dedicated function for drawing. Here is an example of how the algorithms of these functions work:
- First, we use a tool to convert images into text files.



```
&JYY
G8888&&
#888888&&
 ?88BBBY&&
 #888BBBGJ?#
  G!qBBBBB                         &#QGQQQGGQ#&
   #?JGBBBBG&Y&G&&&YY&&&&G&#&&#&YJJYBBBBB88888!
     q?JGBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB8888JG
       GqqqBBBBBBBBBBBBBBBBBBBBBBBBBBB8888888
       qqqBBBBBBBBBBBBBBBBBBBBBBBBB88888&&
      #~GBBBBBBBBBBBBBBBBBBBBBBBBY~
      qqBBBB55BBBBBBBBBBBBBB55BBBB!G
      ?JBB5PPP5BBBBBBBBBBB5PPP5BB?J
     #~BBB55PP55BBBBBBBBBB55PP55BBB!&
     YBBBBBBBBBBBBBBBBBBBBBBBBBBBBB&
     J7777777BB88888888888BB7777777Y
     #7777777BBB888777888BBB7777777#
      #777777BBBB7777777BBBB777777&
       &77777BBBBB77777BBBBB77777&
        QBBBBBBBBBBBBBBBBBBBBBBBQ
```

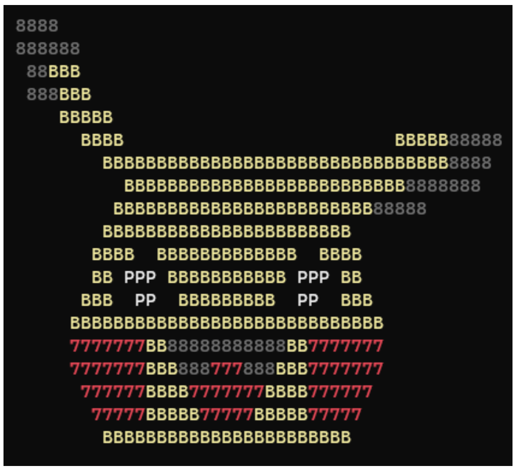
Image 3.2.1a: Convert image into text file

- Then, we implement functions to print this text file with desired colors:

• void pikachu small(int x, int y);

→ The "pokemon_ball" function is an example of how we draw images in our game. It reads an ASCII art file called "pokemonball.txt" from the "ascii_art" directory. The function uses the characters in the file to draw the image on the console screen.
- If the character '7' is encountered, it sets the background color to black, text color to red, and then prints the character at the current cursor position with the specified colors.
- If the character 'J' is encountered, it sets the background color to black, text color to

light white, and then prints the character at the current cursor position with the specified colors.
- If a newline character is encountered, it moves the cursor to the next line and resets the X position. Otherwise, it sets the background color to black, text color to gray, and prints the character at the current cursor position with the specified colors.
→ This way, the function reads the ASCII art file and interprets the characters to draw the image with the desired colors at the specified position on the console screen.



```
8888
888888
 88BBB
 888BBB
    BBBBB
      BBBB                             BBBBB88888
        BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB8888
          BBBBBBBBBBBBBBBBBBBBBBBBBBB8888888
         BBBBBBBBBBBBBBBBBBBBBBBBB88888
        BBBBBBBBBBBBBBBBBBBBBBBB
       BBBB  BBBBBBBBBBBBB  BBBB
       BB PPP BBBBBBBBBBB PPP BB
      BBB  PP  BBBBBBBBB  PP  BBB
     BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
     7777777BB88888888888BB7777777
     7777777BBB888777888BBB7777777
      777777BBBB7777777BBBB777777
       77777BBBBB77777BBBBB77777
        BBBBBBBBBBBBBBBBBBBBBBBB
```

Image 3.2.1b: Image when printed to the screen

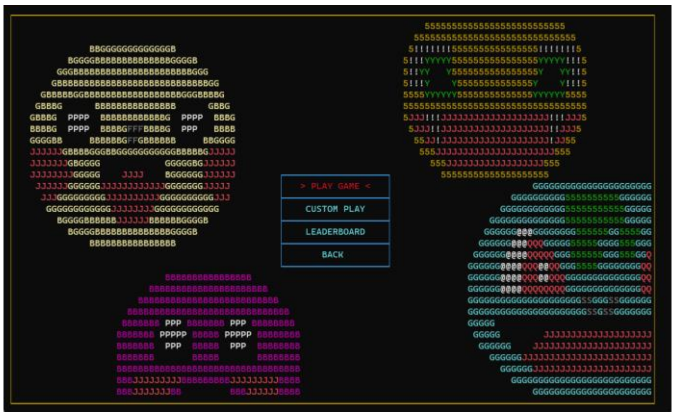
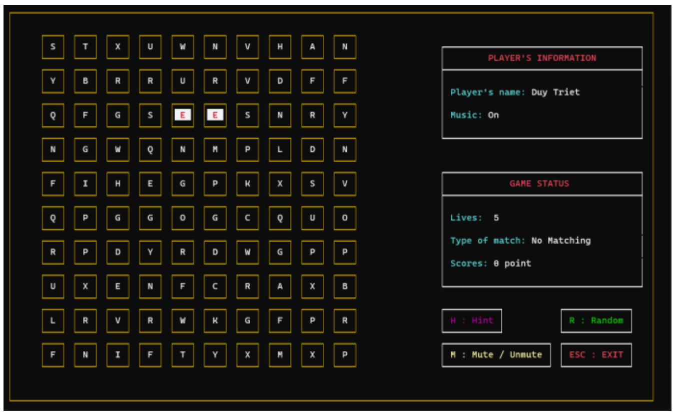


Image 3.2.1c: Applying images to the menu interface

3.2.2 Visual effects

Below is an example of how we apply visual effects in gaming experience:

- When the user uses the keys to move and select two arbitrary cells ⟶ 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.



PLAYER'S INFORMATION

Player's name: Duy Triet

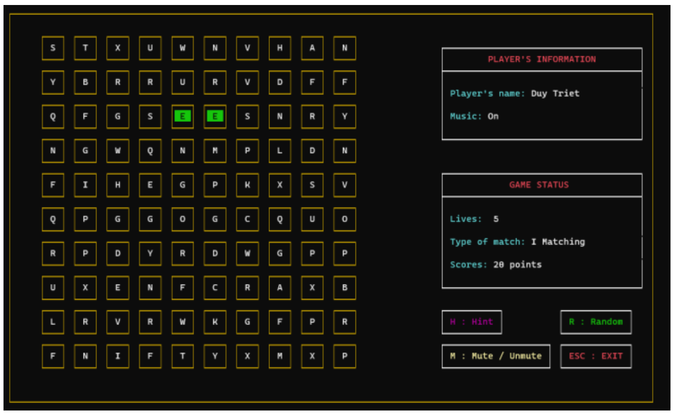
Music: On

GAME STATUS

Lives: 5

Type of match: No Matching

Scores: 0 point



PLAYER'S INFORMATION

Player's name: Duy Triet

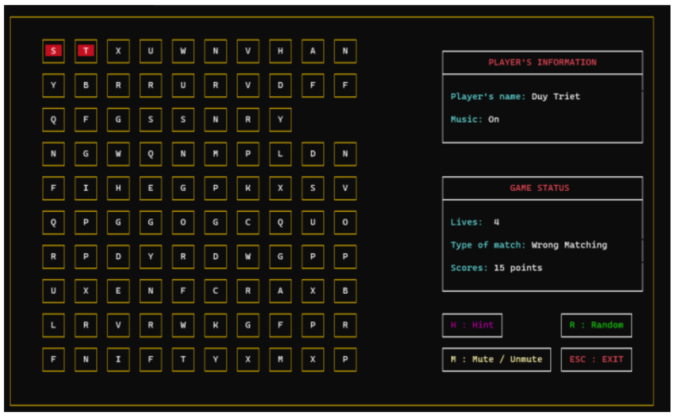
Music: On

GAME STATUS

Lives: 5

Type of match: I Matching

Scores: 20 points



PLAYER'S INFORMATION

Player's name: Duy Triet

Music: On

GAME STATUS

Lives: 4

Type of match: Wrong Matching

Scores: 15 points

Image 3.2.2: Visual effects in gameplay

3.2.3 Sound effects

- We mainly use this code to play music:
PlaySound(TEXT(”path to music”), 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 have implemented 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 a shorter time will be considered.

- Here is the pseudocode:

```
If p1.point != p2.point Then
    Return p1.point > p2.point
Else
    Return p1.time < p2.time
End If
```

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. If the file does not exist, the binary file will be automatically created.

Function: void pushRecord(Players p);

- This function is used to record high scores and write them to a 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.
- This is how we sort the players:

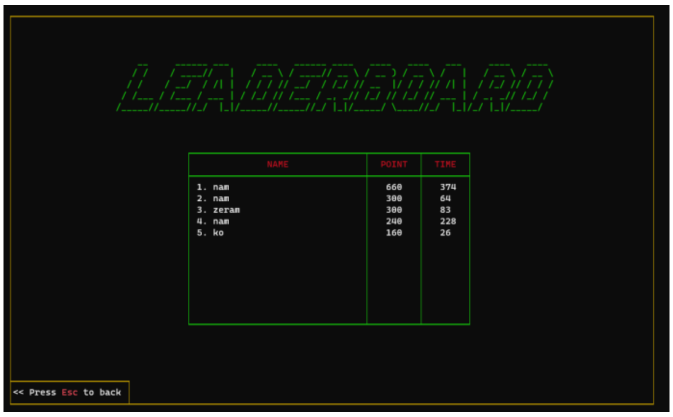
```
sort(players_read, players_read + curSize, comparePlayers);
```

→ 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.
- This is how we sort the players:

```
sort(players_read, players_read + 5, comparePlayers);
```

Function: void printLeaderboard();

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



```
                    NAME                     POINT   TIME
 1. nam                                       660     374
 2. nam                                       300     64
 3. zeram                                     300     83
 4. nam                                       240     228
 5. ko                                        160     26

<< Press Esc to back
```

Image 3.2.4: Leaderboard

3.2.5 Move suggestion

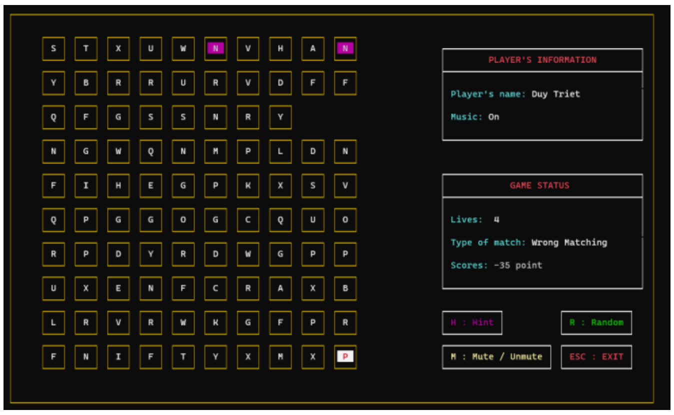
- When the ”moveSuggestion” function is called by pressing ”h” or ”H”, the hint will be shown in magenta background color in order to signal the hint.
- Here is how we have implemented this feature:

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.



```
S T X U W N V H A N          PLAYER'S INFORMATION
Y B R R U R V D F F          Player's name: Duy Triet
Q F G S S N R Y              Music: On
N G W Q N M P L D N
F I H E G P K X S V          GAME STATUS
Q P G G O G C Q U O          Lives:  4
R P D Y R D W G P P          Type of match: Wrong Matching
U X E N F C R A X B          Scores: -35 point
L R V R W K G F P R          H : Hint             R : Random
F N I F T Y X M X P          M : Mute / Unmute    ESC : EXIT
```

Image 3.2.5: Move suggestion

3.3 OTHERS FEATURES

3.3.1 How to play

The game instructions are as follows:
1/ We use a function that we defined to draw the game board.
2/ The content of the game board is read from a text file using the ”read file at pos” function, which was explained in the previous section:

- 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 player will use the WASD keys to move up, down, left, or right, and the Enter key to select a cell.
- 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.
- If the matching pattern is valid, user will be awarded 20 points. If it’s invalid, user will be deducted 5 points and lose 1 life. The player will also be deducted 5 points for each hint used.
- The game will end if the player clears all the cells on the board or runs out of 5 lives.

3.3.2 Auto shuffle if no valid pairs

- While the game is in a blocked state with no valid moves, we will automatically shuffle the board until a valid move is found.
- Here is how we have implemented this feature:

In file: Board.cpp

Function: Board **randomize(Board **&board, int row, int col); which has already shown in Game starting.

3.3.3 More Modes

- Users will be provided with 3 modes:

1/ Easy: A 4x4 board, no shuffling after each play.
2/ Medium: A 8x8 board, shuffling after each play.

- We create a mode that automatically shuffles the board after each move based on the ’randomize’ function.

3/ Hard: A 10x10 board, shifting all blocks to the left.

In file: Board.cpp

- We create a mode that automatically shifts all blocks to the left.
- Here is the pseudocode:

```
n = _col − 1
While n > 0
{
    For i = 0 to _row − 1
        For j = 0 to _col − 2
            If board[i][j].status == 2   // terminate block
                Swap(board[i][j], board[i][j+1])
            End If
        End For
    End For
    n−−
}
```

Chapter 4

PROJECT MANAGEMENT TOOLS

- Collaborative workspace: https://github.com/mareZ-noob/Project_KTLT
- Convert images into ASCII art: https://www.text-image.com/convert/ascii.html
- Text to ASCII art generator (TAAG): http://www.patorjk.com/
- Integrated development environment (IDE): Visual Studio Code.

Chapter 5

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. User.cpp
   https://www.youtube.com/watch?v=Ws0yV6WAsvA

   https://www.youtube.com/watch?v=Gb8_KGRGIFM
5. Board.cpp
   https://cplusplus.com/reference/algorithm/random_shuffle/
6. Sounds we have used:
   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/

   https://youtu.be/WhghMTAWgFE

7. Images we have used:
https://www.seekpng.com/png/detail/138-1388103_user-login-icon-login.png

https://www.pinterest.com/pin/572520171383162889/

https://www.kindpng.com/picc/m/117-1172479_charmander-pokemon-face-clipart-png-download-pokemon-charmander.png

https://i.pinimg.com/736x/0c/1d/4c/0c1d4cee2f6b582b2f265faf0b3ebf6a.jpg

https://i.pinimg.com/1200x/b1/2a/f2/b12af22ea8f29a7c9722bb06d771c323.jpg

https://i.ytimg.com/vi/OcGBNpCXYy0/maxresdefault.jpg

Chapter 6

VIDEO DEMO

- Link video: https://youtu.be/02dxjbPMKgA