REPORT
PIKACHU MATCHING GAME

July 3, 2024

Course name: Programming technique

Student name:
Dat, Pham Thanh - 23127170
Luat, Nguyen Tien - 23127221
Advisors:
Thong, Bui Huy
Minh, Nguyen Tran Duy

Contents

1 Introduction

The Matching Game (commonly known as Pikachu Puzzle Game) includes a board of multiple cells, each of which presents a figure. The player finds and matches a pair of cells that contain the same figure and connect each other in some particular pattern. A legal match will make the two cells disappear. The game ends when all matching pairs are found. Figure 1 shows some snapshots from the Pikachu Puzzle Game.

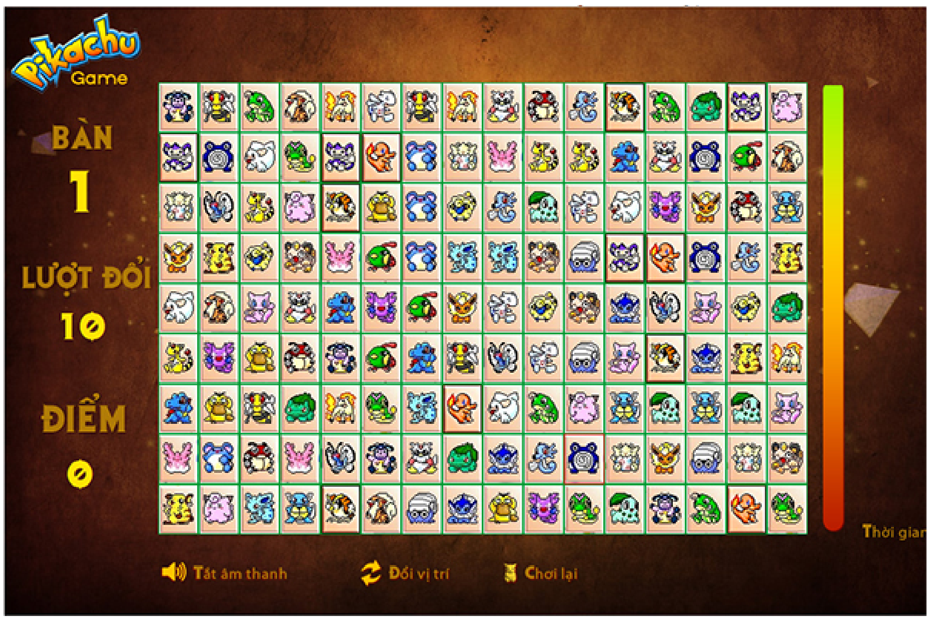


Figure 1: Pikachu matching game

In this project, we will develop a simplified version of this Matching Game by remaking the game with characters (instead of figures)

2 A tutorial of how the game works

2.1 Main screen

- First, the game will display the main screen of Pikachu game, where you will have 6 options:

- **Normal**: choose to play the game with normal mode and access the normal mode account.
- **Hard**: choose to play the game with hard mode and access the hard mode account.
- **Ranking**: view the leaderboard of the 10 highest scoring players of each modes.

- **PvP**: choose to play the game with PvP mode
- **Hidden**: choose to play the game with hidden mode
- **Exit**: exit the game.

- The player uses the arrow keys to navigate between options and press Enter to select.

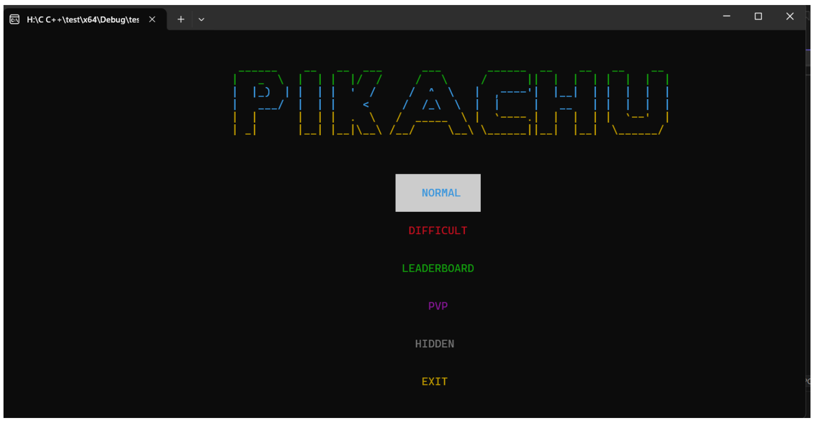
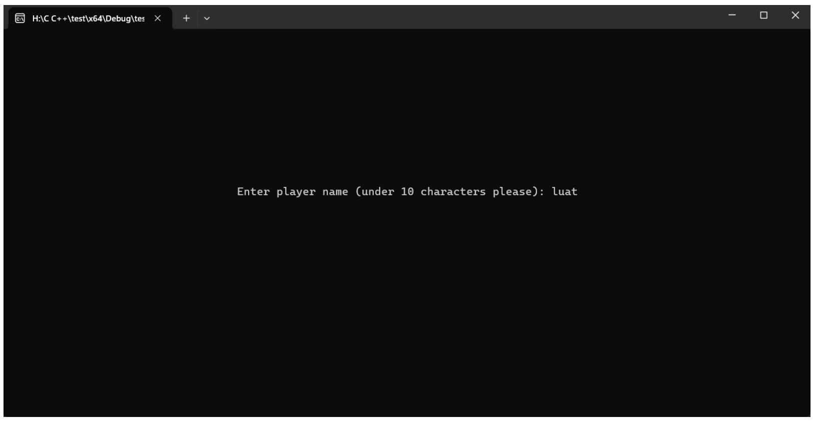


Figure 2: Main menu

2.2 Login screen

- When players select the normal mode or hard mode,the system will prompt the player to enter their name.



```
Enter player name (under 10 characters please): luat
```

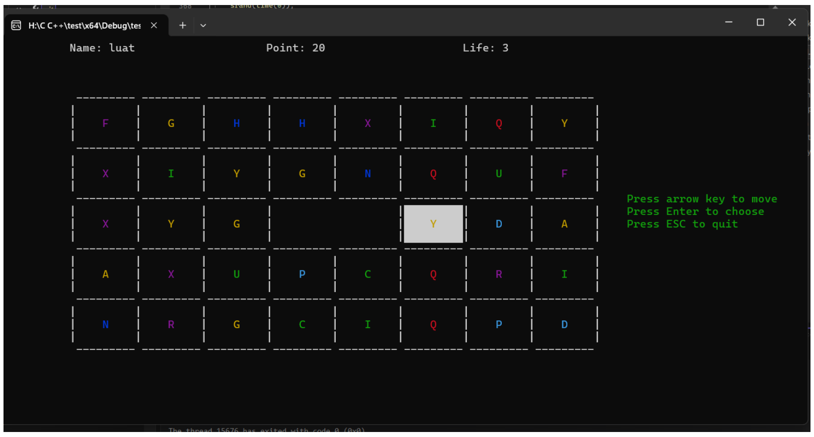
Figure 3: Login screen

- In case the account name already exists in the game's data of normal mode or hard mode or hidden mode, the system will overwrite the current account record if the current score is higher than the previous score and update the rank in the leader board.

- If the account name is new, a new account will be generated. If the new account have score in top 10, the system will update the leaderboard with the new account score.

2.3 Gameplay

- The player will use arrow keys to move and the Enter key to select two matching squares. Players have 3 life point, each correct match will earn points, and each incorrect match will deduct a life point.



```
H:\C C++\test\x64\Debug\tes
        Name: luat                 Point: 20                 Life: 3


     --------- --------- --------- --------- --------- --------- --------- ---------
    |         |         |         |         |         |         |         |         |
    |    F    |    G    |    H    |    H    |    X    |    I    |    Q    |    Y    |
    |         |         |         |         |         |         |         |         |
     --------- --------- --------- --------- --------- --------- --------- ---------
    |         |         |         |         |         |         |         |         |
    |    X    |    I    |    Y    |    G    |    N    |    Q    |    U    |    F    |
    |         |         |         |         |         |         |         |         |
     --------- --------- --------- --------- --------- --------- --------- ---------     Press arrow key to move
    |         |         |         |                   |         |         |         |    Press Enter to choose
    |    X    |    Y    |    G    |                   |    Y    |    D    |    A    |    Press ESC to quit
    |         |         |         |                   |         |         |         |
     --------- --------- --------- --------- --------- --------- --------- ---------
    |         |         |         |         |         |         |         |         |
    |    A    |    X    |    U    |    P    |    C    |    Q    |    R    |    I    |
    |         |         |         |         |         |         |         |         |
     --------- --------- --------- --------- --------- --------- --------- ---------
    |         |         |         |         |         |         |         |         |
    |    N    |    R    |    G    |    C    |    I    |    Q    |    P    |    D    |
    |         |         |         |         |         |         |         |         |
     --------- --------- --------- --------- --------- --------- --------- ---------
```

Figure 4: Gameplay

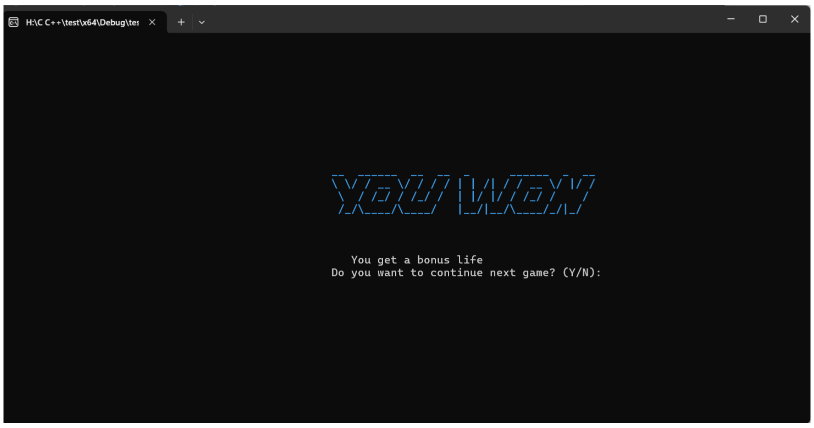
2.4 Game finish

- Players will win the game if the board is completely clear.

- After winning the game, the system will show out **you won** at the end and ask players whether they want to play more or not. If players choose to continue playing, the game mode start again, players will earn a bonus life point (normal mode), and if they don't, the system will update the **leaderboard** and return the players to the main screen.

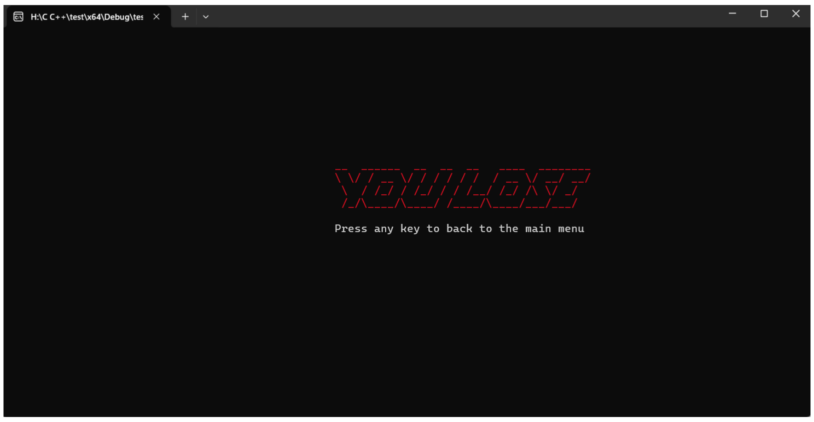
- If players lose 3 life point, players will lose the game, the system will show out **you lose** and also return to the game main screen.

- If there are no more valid pairs left, players also lose the game.



```
    You get a bonus life
Do you want to continue next game? (Y/N):
```

Figure 5: Win screen



```
Press any key to back to the main menu
```

Figure 6: Lose screen

3 Game console setting

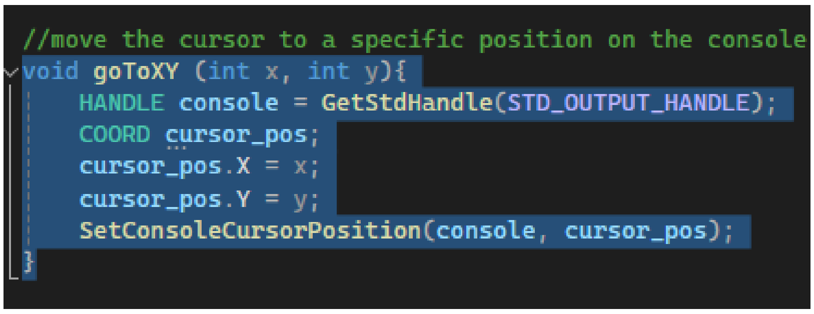
- These functions are utility functions used to manipulate the console window in C++.

3.1 goToXY

Function: goToXY (int x, int y)

- This function moves the cursor to a specific position on the console window.

- It takes two integer parameters, x and y.



```
//move the cursor to a specific position on the console
void goToXY (int x, int y){
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD cursor_pos;
    cursor_pos.X = x;
    cursor_pos.Y = y;
    SetConsoleCursorPosition(console, cursor_pos);
}
```

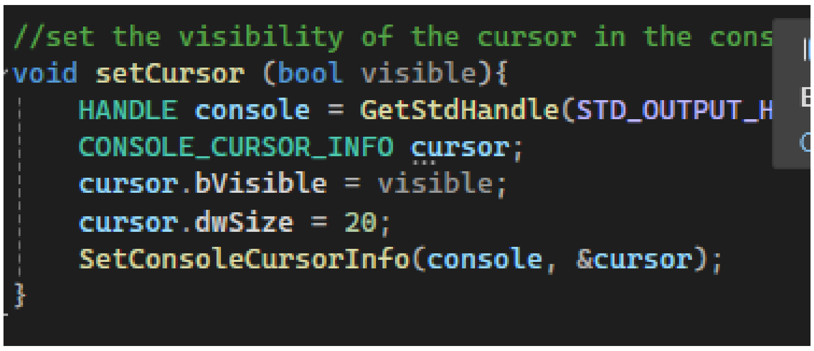
Figure 7: goToXY

1. It retrieves a handle to the standard output (the console window) using the GetStdHandle function from the Windows API.
2. Then it declares a COORD structure named cursor_pos, which is used to hold the coordinates of the cursor position on the console window. COORD is a structure provided by the Windows API that contains two members: X (horizontal position) and Y (vertical position).
3. After that it assigns 2 parameters x and y to cursor_pos.X and cursor_.Y
4. SetConsoleCursorPosition function from the Windows API to set the position of the cursor in the console window.

3.2 setCursor

Function: setCursor (bool visible)

- This function sets the visibility of the cursor in the console window.



```
//set the visibility of the cursor in the cons
void setCursor (bool visible){
    HANDLE console = GetStdHandle(STD_OUTPUT_H
    CONSOLE_CURSOR_INFO cursor;
    cursor.bVisible = visible;
    cursor.dwSize = 20;
    SetConsoleCursorInfo(console, &cursor);
}
```

Figure 8: setCursor

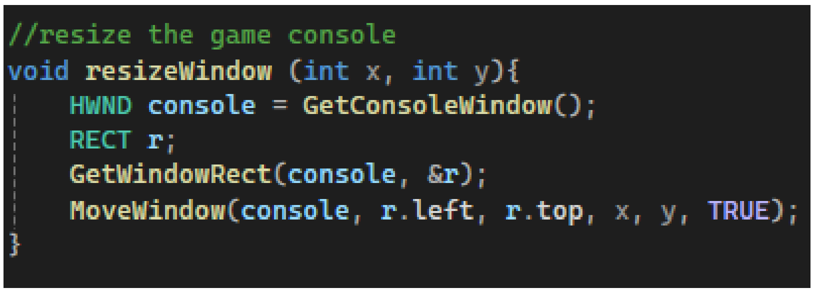
- It takes a boolean parameter visible

1. It retrieves a handle to the standard output (the console window) using the GetStdHandle function from the Windows API, similar to the previous function.
2. Then it declares a CONSOLE_CURSOR_INFO structure named cursor, which is used to hold information about the console cursor. CONSOLE_CURSOR_INFO is

a structure provided by the Windows API that contains two members: bVisible (a boolean indicating whether the cursor is visible) and dwSize (an integer representing the size of the cursor).
3. It assigns visible to bVisible of the cursor and set dwSize to 20. If visible is true, the cursor will be visible; if false, the cursor will be hidden.
4. After that it calls the SetConsoleCursorInfo function from the Windows API to set the cursor information for the console window.

3.3 resizeWindow

Function: resizeWindow (int x, int y)



```
//resize the game console
void resizeWindow (int x, int y){
    HWND console = GetConsoleWindow();
    RECT r;
    GetWindowRect(console, &r);
    MoveWindow(console, r.left, r.top, x, y, TRUE);
}
```

Figure 9: resizeWindow

- This function is used to resize the console window.

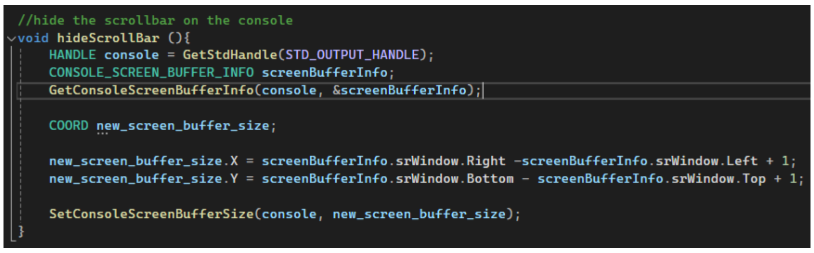
- It takes two integer parameters, x and y, as the length and width of the window.

1. It retrieves a handle to the console window using the GetConsoleWindow function. This handle (console) is of type HWND, which stands for ”handle to a window” in the Windows API.
2. Then it declares a structure of type RECT named r. RECT is a structure provided by the Windows API that represents a rectangle. It’s used here to store information about the dimensions of the console window.
3. The GetWindowRect function is used to obtain the dimensions of a specified window, and it takes two parameters: the handle to the window (console) and a pointer to a RECT structure (r) to receive the dimensions.
4. It resizes and repositions the console window using the MoveWindow function.

3.4 hideScrollBar

Function: void hideScollBar ()

- This function is responsible for hiding the scroll bar in the console window.



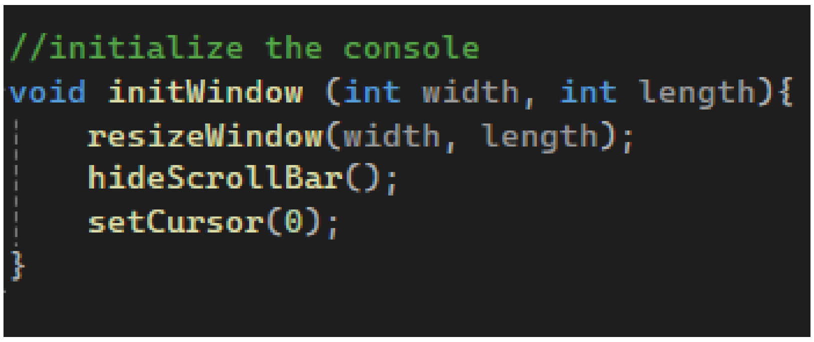
```
//hide the scrollbar on the console
void hideScrollBar (){
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO screenBufferInfo;
    GetConsoleScreenBufferInfo(console, &screenBufferInfo);

    COORD new_screen_buffer_size;

    new_screen_buffer_size.X = screenBufferInfo.srWindow.Right -screenBufferInfo.srWindow.Left + 1;
    new_screen_buffer_size.Y = screenBufferInfo.srWindow.Bottom - screenBufferInfo.srWindow.Top + 1;

    SetConsoleScreenBufferSize(console, new_screen_buffer_size);
}
```

Figure 10: hideScrollbar

1. It retrieves a handle to the standard output device (the console) using the GetStdHandle function.
2. Then it declares a variable screenBufferInfo of type CONSOLE_SCREEN_BUFFER_INFO, which is a structure provided by the Windows API to hold information about the console screen buffer.
3. The GetConsoleScreenBufferInfo function takes two parameters: the handle to the console (console) and a pointer to a CONSOLE_SCREEN_BUFFER_INFO structure (screenBufferInfo) to receive the information.
4. Create a variable with new coordinate and assign the new width and height of the console screen buffer.
5. Sets the size of the console screen buffer to the newly calculated size using the SetConsoleScreenBufferSize function.

3.5 initWindow

Function: initWindow(int width, int height)
- This function is used to initialize the console window for the game.



```
//initialize the console
void initWindow (int width, int length){
    resizeWindow(width, length);
    hideScrollBar();
    setCursor(0);
}
```

Figure 11: InitWindow

- It takes two integer parameters, the width and the height of the window.

1. First, it esizes the console window to the specified dimensions using the resizeWindow function with two parameters.
2. Then, it calls the hideScrollBar function to hide the scroll bar in the console window. It ensures that the console window appears clean without any scroll bars.
3. After that, it calls the setCursor function with 0 to set the visibility of the cursor in the console window into invisibility.

4 Structure declaration

4.1 Struct Position

- This structure represents a position on the game board, specified by its x and y coordinates.

- Members:
1. x: an integer that represents the x-coordinate of the position.
2. y: an integer that represents the y-coordinate of the position.

4.2 Struct Player

- This structure represents the account of a player in the game.

- Members:
1. name: A string representing the name of the player.
2, point: An integer representing the points scored by the player.
3, life: An integer representing the remaining life of the player.

4.3 Struct Cell_1 (array of pointers)

- This structure represents a cell in a 2D array used for the game board. It's used when the game board is implemented as an array of pointers.

- Members:
1. i: An integer representing the row index of the cell.
2. j: An integer representing the column index of the cell.
3. c: A character representing the content of the cell.
4. valid: A boolean indicating whether the cell is valid or not.
5. selected: A boolean indicating whether the cell is selected or not.

- Functions:
1. drawBox(int): Draws the cell with a specified color.
2. deleteBox(): Deletes the content of the cell.
3. drawHiddenBox(int): Draws the cell with hidden character (use for hidden mode)

4.4 Struct Cell_2 (linked list)

- This structure represents a cell in a linked list used for the game board. It's used when the game board is implemented as a linked list.

- Members:
1. i: An integer representing the row index of the cell.
2, j: An integer representing the column index of the cell.
3. c: A character representing the content of the cell.
4. selected: A boolean indicating whether the cell is selected or not.
5. next: A pointer to the next cell in the linked list.

- Functions:
1. drawBox(int): Draws the cell with a specified color.
2. deleteBox(): Deletes the content of the cell.

5 An description of how to complete all the requirements in Standard Mode

5.1 Game starting

5.1.1 Initialize the board

Function: initBoard(Cell_1 board)**

- This function initializes the game board by dynamically allocating memory for a 2D array of Cell_1 objects and assigning characters to each box on the board.

- It begins by dynamically allocating memory for the game board using a nested loop and allocates memory for each row of the board and assigns the i and j coordinates to a element in the board representing a box on the board.

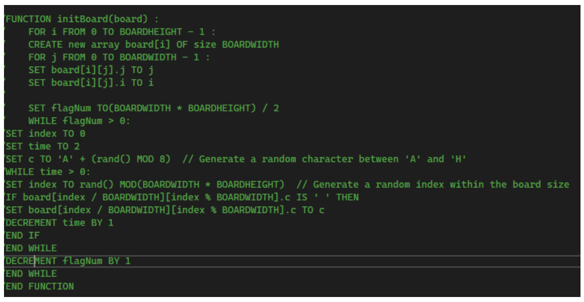
- Then it proceeds to assign characters to each box on the board. It does so by iterating through a loop until a certain number of unique characters have been assigned to the board.

- "flagNum" variable represents the total number of characters that need to be assigned to the board, which is half the total number of boxes (since each character appears twice).

- In the loop, a character is generated by using rand() function and assign it twice in randomly box of the board if the the box is empty.

- This process continues until the required number of characters (flagNum) have been assigned to the board.

- Here the pseudo code for it:



```
FUNCTION initBoard(board) :
    FOR i FROM 0 TO BOARDHEIGHT - 1 :
    CREATE new array board[i] OF size BOARDWIDTH
    FOR j FROM 0 TO BOARDWIDTH - 1 :
    SET board[i][j].j TO j
    SET board[i][j].i TO i

    SET flagNum TO(BOARDWIDTH * BOARDHEIGHT) / 2
    WHILE flagNum > 0:
SET index TO 0
SET time TO 2
SET c TO 'A' + (rand() MOD 8)  // Generate a random character between 'A' and 'H'
WHILE time > 0:
SET index TO rand() MOD(BOARDWIDTH * BOARDHEIGHT)  // Generate a random index within the board size
IF board[index / BOARDWIDTH][index % BOARDWIDTH].c IS ' ' THEN
SET board[index / BOARDWIDTH][index % BOARDWIDTH].c TO c
DECREMENT time BY 1
END IF
END WHILE
DECREMENT flagNum BY 1
END WHILE
END FUNCTION
```

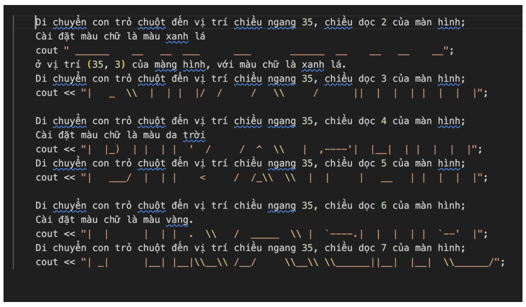
Figure 12: initialize board

Note: initBoard function for the Hard mode is similar to this function

5.1.2 Create pikachu text

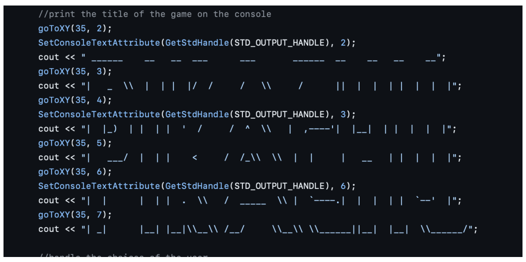
- Create a text title "PIKACHU" using the characters "-", '—', '¿' '¡', '/'.

- Use the gotoXY function to move the mouse cursor to the specified position and the SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), color code) function to change the character color.



```
Di chuyển con trỏ chuột đến vị trí chiều ngang 35, chiều dọc 2 của màn hình;
Cài đặt màu chữ là màu xanh lá
cout " ______   __   __  ___      ___       ______  __    __   __    __";
ở vị trí (35, 3) của màng hình, với màu chữ là xanh lá.
Di chuyển con trỏ chuột đến vị trí chiều ngang 35, chiều dọc 3 của màn hình;
cout << "|   _  \\  |  | |  |/  /     /   \\     /      ||  |  |  | |  |  |  |";

Di chuyển con trỏ chuột đến vị trí chiều ngang 35, chiều dọc 4 của màn hình;
Cài đặt màu chữ là màu da trời
cout << "|  |_)  | |  | |  '  /     /  ^  \\   |  ,----'|  |__|  | |  |  |  |";
Di chuyển con trỏ chuột đến vị trí chiều ngang 35, chiều dọc 5 của màn hình;
cout << "|   ___/  |  | |    <     /  /_\\  \\  |  |     |   __   | |  |  |  |";

Di chuyển con trỏ chuột đến vị trí chiều ngang 35, chiều dọc 6 của màn hình;
Cài đặt màu chữ là màu vàng.
cout << "|  |      |  | |  .  \\   /  _____  \\ |  `----.|  |  |  | |  `--'  |";
Di chuyển con trỏ chuột đến vị trí chiều ngang 35, chiều dọc 7 của màn hình;
cout << "| _|      |__| |__|\\__\\ /__/     \\__\\ \\______||__|  |__|  \\______/";
```

Figure 13: Pseudo code



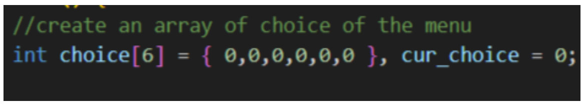
```
//print the title of the game on the console
goToXY(35, 2);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 2);
cout << " ______   __   __  ___      ___       ______  __    __   __    __";
goToXY(35, 3);
cout << "|   _  \\  |  | |  |/  /     /   \\     /      ||  |  |  | |  |  |  |";
goToXY(35, 4);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 3);
cout << "|  |_)  | |  | |  '  /     /  ^  \\   |  ,----'|  |__|  | |  |  |  |";
goToXY(35, 5);
cout << "|   ___/  |  | |    <     /  /_\\  \\  |  |     |   __   | |  |  |  |";
goToXY(35, 6);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 6);
cout << "|  |      |  | |  .  \\   /  _____  \\ |  `----.|  |  |  | |  `--'  |";
goToXY(35, 7);
cout << "| _|      |__| |__|\\__\\ /__/     \\__\\ \\______||__|  |__|  \\______/";
```

Figure 14: Code

5.1.3 Choice array for menu

- The game start screen will have 6 options: "DIFFICUTLT", "NORMAL", "LEADERBOARD", "PVP", "HIDDEN" and "EXIT".

- Create an array of choice of the menu.



```
//create an array of choice of the menu
int choice[6] = { 0,0,0,0,0,0 }, cur_choice = 0;
```

Figure 15: Array of choice

- Cur_choice = 1 is DIFFICULT, Cur_choice = 2 is NORMAL , Cur_choice = 3 is LEADERBOARD, Cur_choice = 4 is PVP, Cur_choice = 5 is HIDDEN and Cur_choice = 6 is EXIT.

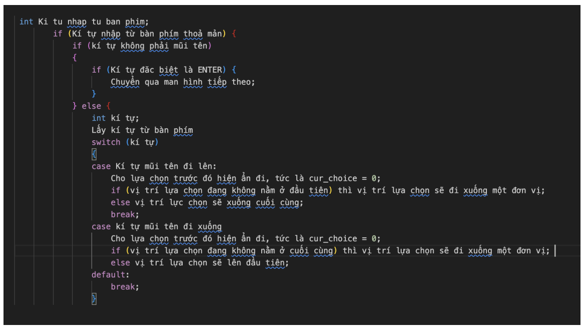
- If choice[cur_choice] = 1, the mouse pointer is at the corresponding selection position, and change color at that location.

5.1.4 Check keyboard input

- We will create a loop until the player presses ENTER for his choice. Each time you move up or down, the loop starts again.

- Players will use the up and down arrows to move to the option they want. Each move will change the "Cur_choice" value, which will change to correspond to the choices.

And here is the code for the program:



```
int Ki tu nhap tu ban phim;
        if (Kí tự nhập từ bàn phím thoả mản) {
            if (kí tự không phải mũi tên)
            {
                if (Kí tự đăc biệt là ENTER) {
                    Chuyển qua man hình tiếp theo;
                }
            } else {
                int kí tự;
                Lấy kí tự từ bàn phím
                switch (kí tự)
                {
                case Kí tự mũi tên đi lên:
                    Cho lựa chọn trước đó hiện ẩn đi, tức là cur_choice = 0;
                    if (vị trí lựa chọn đang không nằm ở đầu tiên) thì vị trí lựa chọn sẽ đi xuống một đơn vị;
                    else vị trí lực chọn sẽ xuống cuối cùng;
                    break;
                case kí tự mũi tên đi xuống
                    Cho lựa chọn trước đó hiện ẩn đi, tức là cur_choice = 0;
                    if (vị trí lựa chọn đang không nằm ở cuối cùng) thì vị trí lựa chọn sẽ đi xuống một đơn vị;
                    else vị trí lựa chọn sẽ lên đầu tiên;
                default:
                    break;
                }
```

Figure 16: Pseudo code for check keyboard

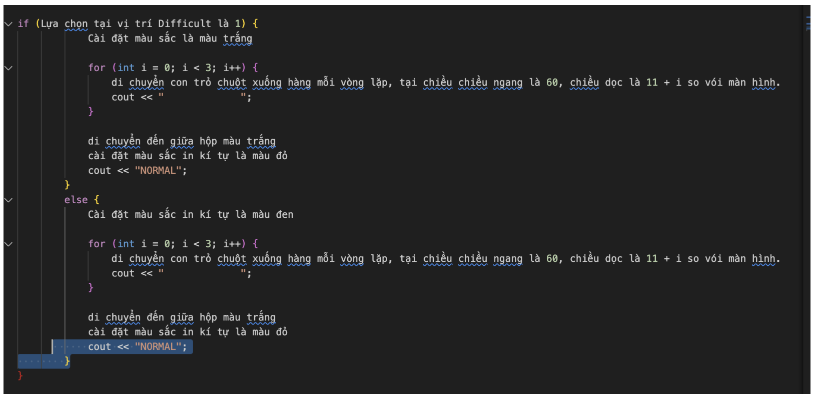
- When the player presses Enter, the value of Cur_choice will be returned. From the returned value, the screen will display corresponding to the choice.

5.1.5 Handle the choices of the user

- When the selection position moves to a cell, the box around that position will be white, other cells will be black.

- For example: If cur_choice = 0 and choice[cur_choice] = 1, the selection position is located in DIFFICULT. The box around that position is white, the word DIFFICULT is red.

- At the start of the loop, choice[for choice] = 1, meaning the color of the position in the current choice will change.



```
if (Lựa chọn tại vị trí Difficult là 1) {
        Cài đặt màu sắc là màu trắng

        for (int i = 0; i < 3; i++) {
            di chuyển con trỏ chuột xuống hàng mỗi vòng lặp, tại chiều chiều ngang là 60, chiều dọc là 11 + i so với màn hình.
            cout << "                ";
        }

        di chuyển đến giữa hộp màu trắng
        cài đặt màu sắc in kí tự là màu đỏ
        cout << "NORMAL";
    }
    else {
        Cài đặt màu sắc in kí tự là màu đen

        for (int i = 0; i < 3; i++) {
            di chuyển con trỏ chuột xuống hàng mỗi vòng lặp, tại chiều chiều ngang là 60, chiều dọc là 11 + i so với màn hình.
            cout << "                ";
        }

        di chuyển đến giữa hộp màu trắng
        cài đặt màu sắc in kí tự là màu đỏ
        cout << "NORMAL";
    }
}
```

Figure 17: Pseudo code for located in normal

- The logic is similar to other choices.

6 Matching

6.1 Matching for pointer (Normal mode)

6.1.1 Line check

Function: bool linecheck(Cell_1 board, int point1x, int point1y, int point2x, int point2y)**

- This function is utilized to ensure that there is no more than one cell between two positions. If there are more than 1 cell, it will return 0. It is employed in the I, U, Z, and L matching functions.

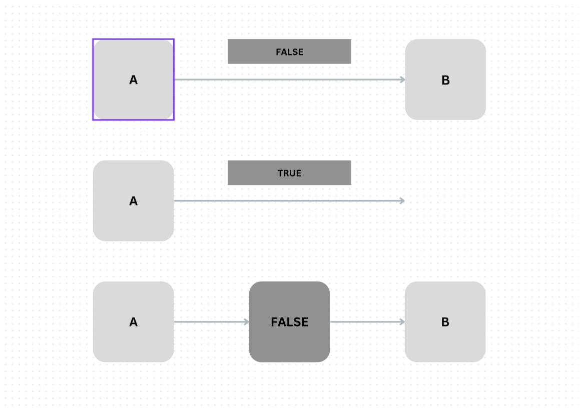
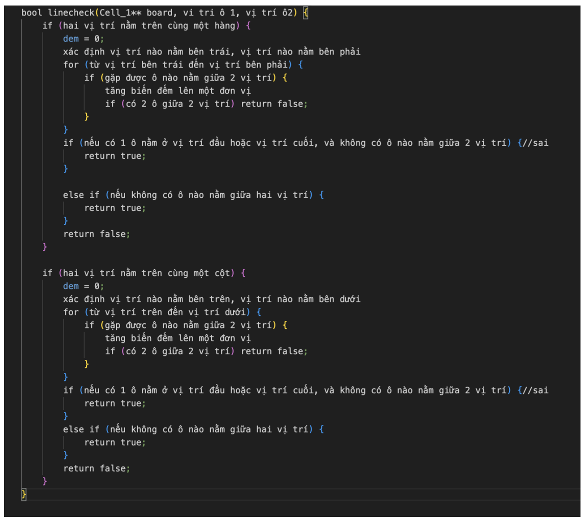


Figure 18: Check line



```
bool linecheck(Cell_1** board, vi tri ô 1, vị trí ô2) {
    if (hai vị trí nằm trên cùng một hàng) {
        dem = 0;
        xác định vị trí nào nằm bên trái, vị trí nào nằm bên phải
        for (từ vị trí bên trái đến vị trí bên phải) {
            if (gặp được ô nào nằm giữa 2 vị trí) {
                tăng biến đếm lên một đơn vị
                if (có 2 ô giữa 2 vị trí) return false;
            }
        }
        if (nếu có 1 ô nằm ở vị trí đầu hoặc vị trí cuối, và không có ô nào nằm giữa 2 vị trí) {//sai
            return true;
        }

        else if (nếu không có ô nào nằm giữa hai vị trí) {
            return true;
        }
        return false;
    }

    if (hai vị trí nằm trên cùng một cột) {
        dem = 0;
        xác định vị trí nào nằm bên trên, vị trí nào nằm bên dưới
        for (từ vị trí trên đến vị trí dưới) {
            if (gặp được ô nào nằm giữa 2 vị trí) {
                tăng biến đếm lên một đơn vị
                if (có 2 ô giữa 2 vị trí) return false;
            }
        }
        if (nếu có 1 ô nằm ở vị trí đầu hoặc vị trí cuối, và không có ô nào nằm giữa 2 vị trí) {//sai
            return true;
        }
        else if (nếu không có ô nào nằm giữa hai vị trí) {
            return true;
        }
        return false;
    }
}
```

Figure 19: Pseudo code for checkline

6.1.2 Icheck

- This function determines if there exists a horizontal or vertical I shape between two cells.

Function: bool Icheck(Cell_1 board, int point1x, int point1y, int point2x, int point2y)**

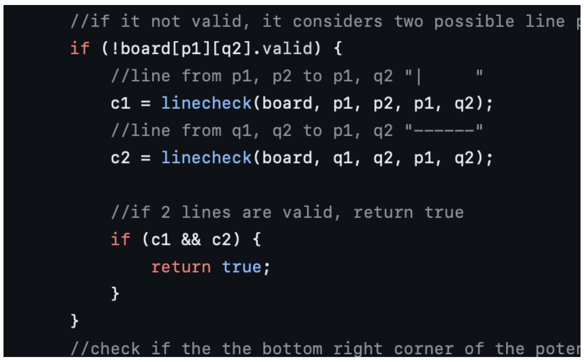
- Set x, y.
- When two cells align horizontally, the 'x' value corresponds to the column position of the left cell, while the 'y' value represents the column position of the right cell.
- Similarly, if two cells align vertically, the 'x' value denotes the row position of

the cell above, and the 'y' value denotes the row position of the cell below.

6.1.3 Lcheck

- This function determine if there exists a horizontal or vertical L shape between 2 cells.
Function: bool Lcheck(Cell_1 board, int point1x, int point1y, int point2x, int point2y)**

- If the two boxes are aligned either horizontally or vertically (point1x == point2x or point1y = point2y) return false after checking their positions.
- Set c1, c2 as variables to verify whether the columns and rows of the two boxes meet the respective conditions.
- If the box positioned at the L-shaped corner of the two boxes under consideration does not exist, then check if there is a box between the two rows and columns. If both lines meet the condition, return true; otherwise, return false.



```
//if it not valid, it considers two possible line p
if (!board[p1][q2].valid) {
    //line from p1, p2 to p1, q2 "|      "
    c1 = linecheck(board, p1, p2, p1, q2);
    //line from q1, q2 to p1, q2 "------"
    c2 = linecheck(board, q1, q2, p1, q2);

    //if 2 lines are valid, return true
    if (c1 && c2) {
        return true;
    }
}
//check if the the bottom right corner of the poter
```

Figure 20: Check Twoline

- If both rows and columns are satisfied, the Icheck function returns true

6.1.4 Zcheck

- This function determine if there exists a horizontal or vertical Z shape between 2 cells.

Funtion: bool Zcheck(Cell_1 board, int point1x, int point1y, int point2x, int point2y)**

- If the two boxes are aligned either horizontally or vertically (point1x == point2x or point1y = point2y) return false after checking their positions.

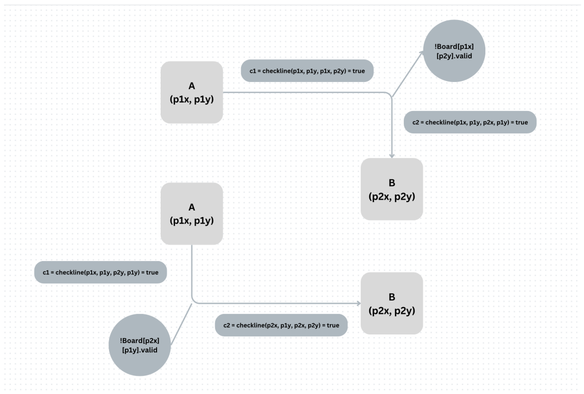

A
(p1x, p1y)
c1 = checkline(p1x, p1y, p1x, p2y) = true
!Board[p1x]
[p2y].valid
c2 = checkline(p1x, p1y, p2x, p1y) = true
B
(p2x, p2y)
A
(p1x, p1y)
c1 = checkline(p1x, p1y, p2y, p1y) = true
B
(p2x, p2y)
c2 = checkline(p2x, p1y, p2x, p2y) = true
!Board[p2x]
[p1y].valid

Figure 21: checkL

- Set c1, c2, c3. With c1 and c2, it is to check whether the line at both ends of the letter Z has more than 1 box or not, and c3 is to check the body of the letter Z.(bool c1, c2, c3)
- There are 2 cases: First is to check in a **vertical Z shape**, second is to check in a **horizontal Z shape**.
- With vertical Z shape:

  Find the left and right column positions of the two boxes under consideration.

  Let the loop run from left to right, each loop checks line **c3 is a Z body**, contains less than one box or not.

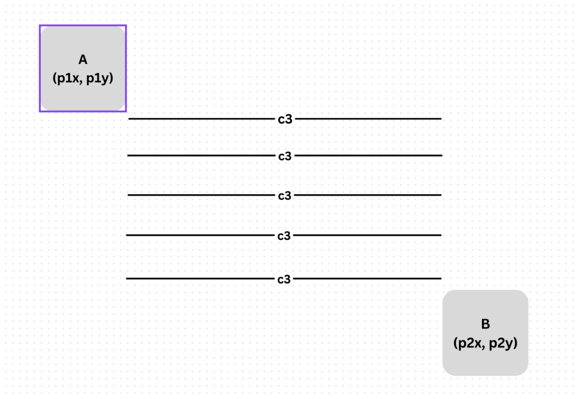
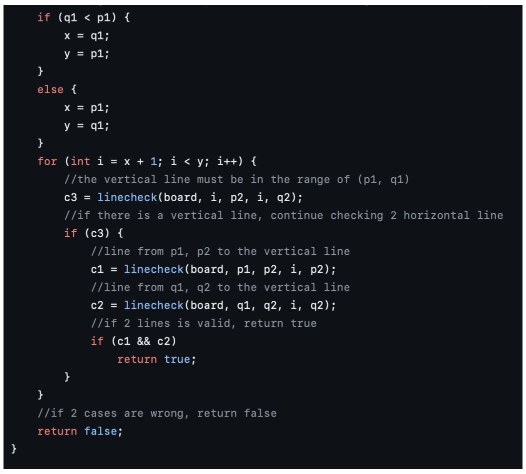


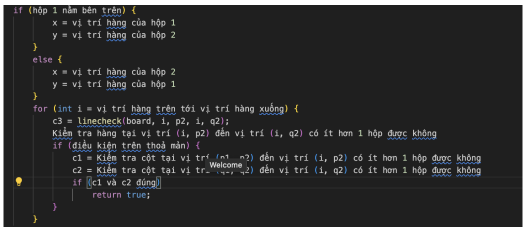
Figure 22: Checkline of rows

If so, check c1, c2, the two ends of the letter Z, which are two rows that contain a single box or not. If all are satisfied, it returns true, otherwise the Zcheck function returns false.



```
    if (q1 < p1) {
        x = q1;
        y = p1;
    }
    else {
        x = p1;
        y = q1;
    }
    for (int i = x + 1; i < y; i++) {
        //the vertical line must be in the range of (p1, q1)
        c3 = linecheck(board, i, p2, i, q2);
        //if there is a vertical line, continue checking 2 horizontal line
        if (c3) {
            //line from p1, p2 to the vertical line
            c1 = linecheck(board, p1, p2, i, p2);
            //line from q1, q2 to the vertical line
            c2 = linecheck(board, q1, q2, i, q2);
            //if 2 lines is valid, return true
            if (c1 && c2)
                return true;
        }
    }
    //if 2 cases are wrong, return false
    return false;
}
```

Figure 23: Code for vertical Zcheck



```
if (hộp 1 nằm bên trên) {
        x = vị trí hàng của hộp 1
        y = vị trí hàng của hộp 2
    }
    else {
        x = vị trí hàng của hộp 2
        y = vị trí hàng của hộp 1
    }
    for (int i = vị trí hàng trên tới vị trí hàng xuống) {
        c3 = linecheck(board, i, p2, i, q2);
        Kiểm tra hàng tại vị trí (i, p2) đến vị trí (i, q2) có ít hơn 1 hộp được không
        if (điều kiện trên thoả mản) {
            c1 = Kiểm tra cột tại vị trí (p1, p2) đến vị trí (i, p2) có ít hơn 1 hộp được không
            c2 = Kiểm tra cột tại vị trí (q1, q2) đến vị trí (i, q2) có ít hơn 1 hộp được không
            if (c1 và c2 đúng)
                return true;
        }
    }
```

Figure 24: Pseudo code for vertical Zcheck

- With **horizontal Z** shape:

  Find the above and below positions of the two boxes under consideration

  Let the loop run from above to below, each loop checks line c3 is a Z body, contains a single box or not.

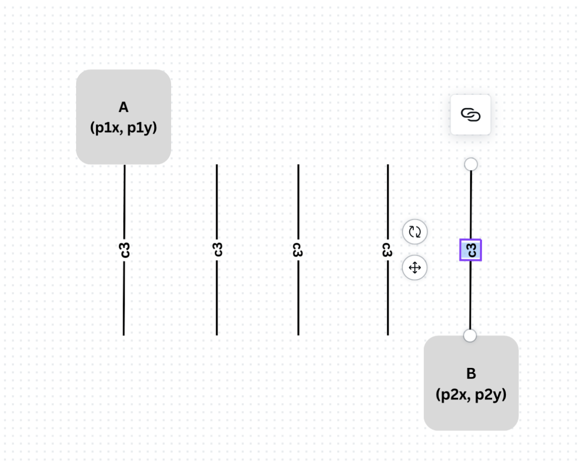
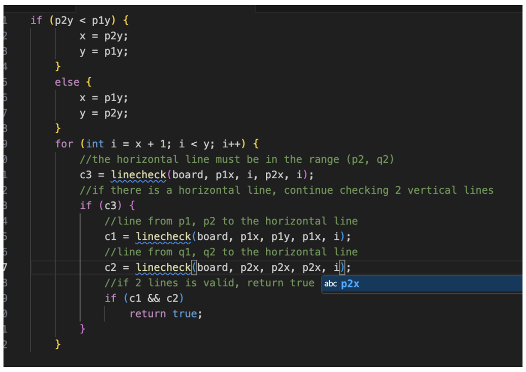


Figure 25: Checkline of cows

  If so, check c1, c2, the two ends of the letter Z, which are two cows that contain a single box or not. If all are satisfied, it returns true, otherwise the Zcheck function returns false.

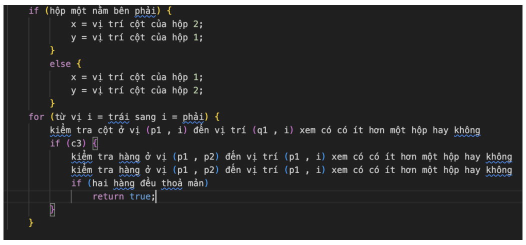
6.1.5 Ucheck

- This function determine if there exists a horizontal or vertical Z shape between 2 cells.



```
if (p2y < p1y) {
        x = p2y;
        y = p1y;
    }
    else {
        x = p1y;
        y = p2y;
    }
    for (int i = x + 1; i < y; i++) {
        //the horizontal line must be in the range (p2, q2)
        c3 = linecheck(board, p1x, i, p2x, i);
        //if there is a horizontal line, continue checking 2 vertical lines
        if (c3) {
            //line from p1, p2 to the horizontal line
            c1 = linecheck(board, p1x, p1y, p1x, i);
            //line from q1, q2 to the horizontal line
            c2 = linecheck(board, p2x, p2x, p2x, i);
            //if 2 lines is valid, return true
            if (c1 && c2)
                return true;
        }
    }
```

Figure 26: Code for horizontal Zcheck



```
if (hộp một nằm bên phải) {
        x = vị trí cột của hộp 2;
        y = vị trí cột của hộp 1;
    }
    else {
        x = vị trí cột của hộp 1;
        y = vị trí cột của hộp 2;
    }
for (từ vị i = trái sang i = phải) {
    kiểm tra cột ở vị (p1 , i) đến vị trí (q1 , i) xem có có ít hơn một hộp hay không
    if (c3) {
        kiểm tra hàng ở vị (p1 , p2) đến vị trí (p1 , i) xem có có ít hơn một hộp hay không
        kiểm tra hàng ở vị (p1 , p2) đến vị trí (p1 , i) xem có có ít hơn một hộp hay không
        if (hai hàng đều thoả mản)
            return true;
    }
}
```

Figure 27: Pseudo code for horizontal Zcheck

Funtion: bool Ucheck(Cell_1 board, int p1, int p2, int q1, int q2)**

- check if the 2 chosen boxes is at the border of the board, it is a U shape.

  (p1 == q1) && (p1 == 0 || p1 == BOARDHEIGHT - 1) 2 chosen boxes is top or bot at border of the board.

  ((p2 == q2) && (p2 == 0 || q2 == BOARDWIDTH - 1)) 2 chosen boxes is right or left at border of the board.

- Set c1, c2 and c3. c1 and c2 to check if the **U-shaped body** has less than one box. c3 checks whether the **U-shaped bottom** line has less than one box.

- There are 2 cases: First is to check in a **vertical U shape**, second is to check in a **horizontal U shape**.

- With **Vertical U** shape:

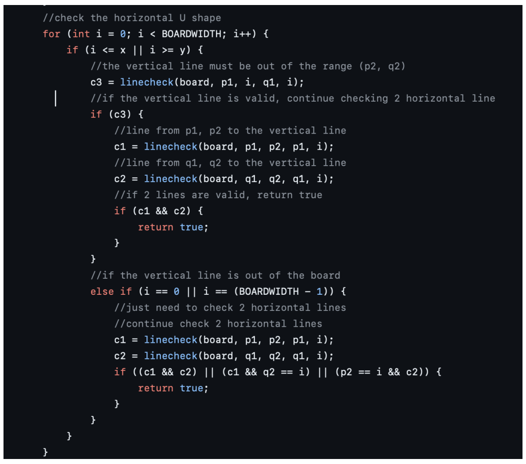
  Let the loop run from top to bottom of the board (from i = 0 to i = BOARDHEIGHT - 1). Each loop will check whether lines c1, c2, c3, contain less than one box.

  The **horizontal line c3** must be out of the range **(p1, q1)**.

  Check line horizontal line c3 contains less than one box c3 = **linecheck(board, i, p2, i, q2)** from the column position containing the first box to the column position containing the second box in row i **(i, p2)** to **(i, q2)**.

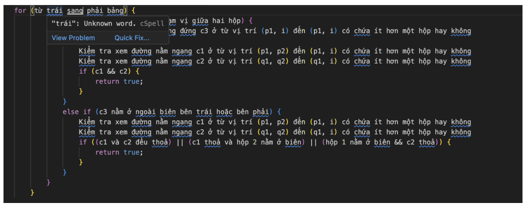
  If true, Check two vertical line contains less than one box **c1 = linecheck(board, p1, p2, i, p2)** and **c2 = linecheck(board, q1, q2, i, q2)** from the row containing the box to the row i and column containing the box. If true, return Ucheck true.

If the horizontal line is out of the board (i == 0 || i == (BOARDHEIGHT - 1)), just check whether the two vertical lines contain less than one box. If both lines are correct, or one of the lines is correct, and there is a box at border of the board, return Ucheck true



```
//check the horizontal U shape
for (int i = 0; i < BOARDWIDTH; i++) {
    if (i <= x || i >= y) {
        //the vertical line must be out of the range (p2, q2)
        c3 = linecheck(board, p1, i, q1, i);
        //if the vertical line is valid, continue checking 2 horizontal line
        if (c3) {
            //line from p1, p2 to the vertical line
            c1 = linecheck(board, p1, p2, p1, i);
            //line from q1, q2 to the vertical line
            c2 = linecheck(board, q1, q2, q1, i);
            //if 2 lines are valid, return true
            if (c1 && c2) {
                return true;
            }
        }
        //if the vertical line is out of the board
        else if (i == 0 || i == (BOARDWIDTH - 1)) {
            //just need to check 2 horizontal lines
            //continue check 2 horizontal lines
            c1 = linecheck(board, p1, p2, p1, i);
            c2 = linecheck(board, q1, q2, q1, i);
            if ((c1 && c2) || (c1 && q2 == i) || (p2 == i && c2)) {
                return true;
            }
        }
    }
}
```

Figure 28: Code for vertical Ucheck



```
for (từ trái sang phải bằng) {
                        am vị giữa hai hộp) {
                        ng đứng c3 ở từ vị trí (p1, i) đến (p1, i) có chứa ít hơn một hộp hay không

            Kiểm tra xem đường nằm ngang c1 ở từ vị trí (p1, p2) đến (p1, i) có chứa ít hơn một hộp hay không
            Kiểm tra xem đường nằm ngang c2 ở từ vị trí (q1, q2) đến (q1, i) có chứa ít hơn một hộp hay không
            if (c1 && c2) {
                return true;
            }
        }
        else if (c3 nằm ở ngoài biên bên trái hoặc bên phải) {
            Kiểm tra xem đường nằm ngang c1 ở từ vị trí (p1, p2) đến (p1, i) có chứa ít hơn một hộp hay không
            Kiểm tra xem đường nằm ngang c2 ở từ vị trí (q1, q2) đến (q1, i) có chứa ít hơn một hộp hay không
            if ((c1 và c2 đều thoả) || (c1 thoả và hộp 2 nằm ở biên) || (hộp 1 nằm ở biên && c2 thoả)) {
                return true;
            }
        }
    }
}
```

Figure 29: Pseudo code for vertical Ucheck

- With **horizontal U** shape:

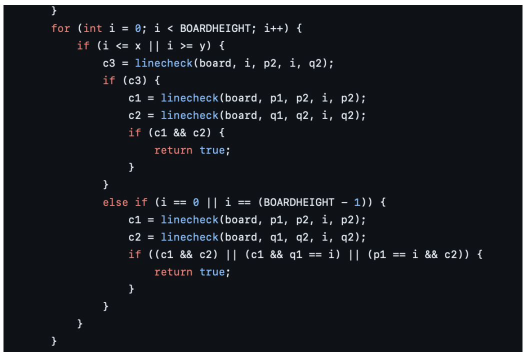
  Let the loop run from right to left of the board for (from i = 0 to i = BOARDWIDTH - 1). Each loop will check whether lines c1, c2, c3, contain less than one box.

  The vertical line c3 must be out of the **range (p2, q2)**.

  Check line vertical line c3 contains less than one box **c3 = linecheck(board, p1, i, q1, i)** from the row position containing the first box to the row position containing the second box in column i **(p1, i)** to **(p1, i)**.

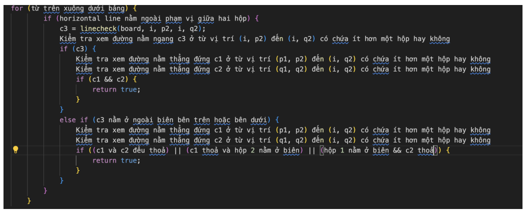
  If true, Check two horizontal line contains less than one box c1 = linecheck(board, p1, p2, p1, i) and **c2 = linecheck(board, q1, q2, q1, i)** from the column containing the box to the column i and row containing the box. If true, return Ucheck true.

  If the vertical line is out of the board (i == 0 —— i == (BOARDWIDTH - 1)), just check whether the two horizontal lines contain less than one box. If both lines are correct, or one of the lines is correct, and there is a box at border of the board, return Ucheck true



```
    }
    for (int i = 0; i < BOARDHEIGHT; i++) {
        if (i <= x || i >= y) {
            c3 = linecheck(board, i, p2, i, q2);
            if (c3) {
                c1 = linecheck(board, p1, p2, i, p2);
                c2 = linecheck(board, q1, q2, i, q2);
                if (c1 && c2) {
                    return true;
                }
            }
            else if (i == 0 || i == (BOARDHEIGHT - 1)) {
                c1 = linecheck(board, p1, p2, i, p2);
                c2 = linecheck(board, q1, q2, i, q2);
                if ((c1 && c2) || (c1 && q1 == i) || (p1 == i && c2)) {
                    return true;
                }
            }
        }
    }
```

Figure 30: Code for horizontal Ucheck



```
for (từ trên xuống dưới bảng) {
        if (horizontal line nằm ngoài phạm vi giữa hai hộp) {
            c3 = linecheck(board, i, p2, i, q2);
            Kiểm tra xem đường nằm ngang c3 ở từ vị trí (i, p2) đến (i, q2) có chứa ít hơn một hộp hay không
            if (c3) {
                Kiểm tra xem đường nằm thẳng đứng c1 ở từ vị trí (p1, p2) đến (i, q2) có chứa ít hơn một hộp hay không
                Kiểm tra xem đường nằm thẳng đứng c2 ở từ vị trí (q1, q2) đến (i, q2) có chứa ít hơn một hộp hay không
                if (c1 && c2) {
                    return true;
                }
            }
            else if (c3 nằm ở ngoài biên bên trên hoặc bên dưới) {
                Kiểm tra xem đường nằm thẳng đứng c1 ở từ vị trí (p1, p2) đến (i, q2) có chứa ít hơn một hộp hay không
                Kiểm tra xem đường nằm thẳng đứng c2 ở từ vị trí (q1, q2) đến (i, q2) có chứa ít hơn một hộp hay không
                if ((c1 và c2 đều thoả) || (c1 thoả và hộp 2 nằm ở biên) || (hộp 1 nằm ở biên && c2 thoả)) {
                    return true;
                }
            }
        }
    }
```

Figure 31: Pseudo code for horizontal Ucheck

6.1.6 allcheck

Funtion: bool allcheck(Cell_1 board, int p1, int p2, int q1, int q2)**

- This function combines all the checks, checking to see if the positions of the two boxes entered have the correct check. If one of the checks is correct, return true, otherwise false

6.2 Matching for linked list (Hard mode)

6.2.1 Find the Node

- Each Cell is a node in the **Linkedlist**. Each Cell contains x, y positions. When a Cell is NULL, it means the Cell does not exist.

- Where arr is the pointer variable of the Cell2 structure pointer. Pointer arr is dynamically allocated the length of the board, each pointer arr[i] is dynamically allocated the width of the board.

- To shorten the Matching functions, this is the function to search for x, y positions in Cells and return Cell.

Funtion: Cell_2* findTheNode(Cell_2 arr, int y, int x)**

- Check if the desired location is outside the table. When it not

$$y < 0 \quad \| \quad y > 4 \quad \| \quad x < 0 \quad \| \quad x > 7$$

.

- We browse all nodes linked to pointer arr[y](Cell2 temp = arr[y], temp = temp-¿next) to find the Cell with column position x. If find, return temp.

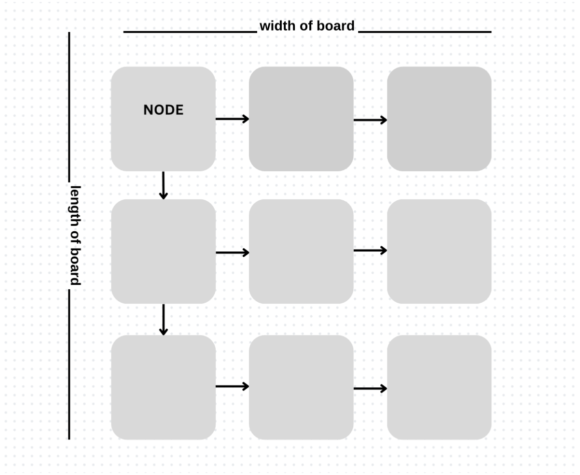


Figure 32: Linked list for board

6.2.2 Icheck

- This is the function used to determine whether straight line I is valid or not.

- If the two cells at both ends have the same characters and there are no cells between the two boxes, return true.

- If at either end there is a box that does not exist or both boxes do not exist and between those two positions there is not any cell, return true, in the remaining cases return false.

- Similar to pointer. When Icheck, we also divide it into two different cases: the letter I is horizontal and the letter I is vertical.

Function: bool Icheck(Cell_2 arr, int y1, int x1, int y2, int x2)**

- With the letter I **horizontal** we determine the Cell located above (ma = max(y1, y2)) and determine the Cell located below (mi = min(y1, y2)).

  Set **tempHead** = **findTheNode**(arr, mi, x1) and **temp** = **tempHead**, meaning **temp** and **tempHead** is holding the address of the Cell below.

  If the cell below the letter I does not exist. Let a loop run until temp is not NULL. Each loop gives the value mi++ and updates **temp** = **findTheNode**(arr, mi, x1). If the loop runs until mi is equal to ma, that is, there is no cell between the two positions (y1, x1) and (y2, x2), then return true, but if mi is smaller than ma, and **temp** != NULL, then return false.

  If the cell below the letter I exist. Set i = 0. Increment mi by one unit, update **temp** = **(**findTheNode)(arr, mi + i, x1). Create a loop until **temp** != NULL, which means a certain cell above the lash line exists. There, if **mi + i >ma**, that is, there is no box between the two positions (y1, x1) and (y2, x2), and at position (y2, x2) there is no cell, then return true.

If **mi + i == ma**, that is, at position (y2, x2) there exists a cell, then we check the characters of the first and last two positions. If they are the same, return true, otherwise return false.

If **mi + i <ma**, **temp** = **findTheNode**(arr, mi + i, x1), **temp** != NULL, meaning there exists another Cell between two Cells, then return false.

- Similar to the vertical letter I, just change the y value to x, and vice versa.

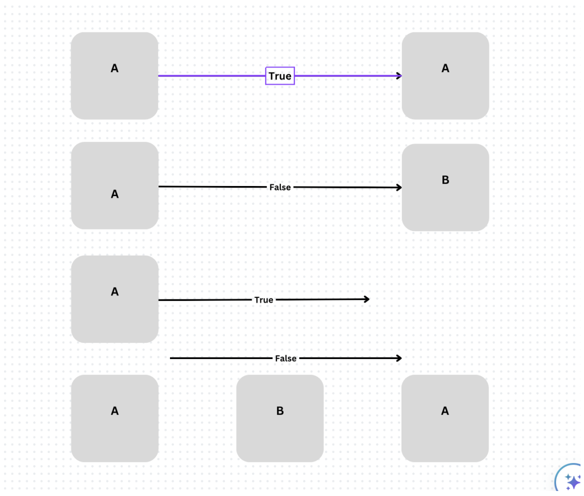


Figure 33: Icheck for Linked list

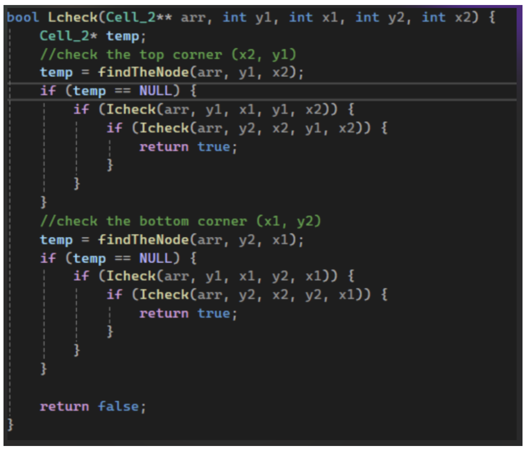
6.2.3 Lcheck

- This function determines if there exists a horizontal or vertical L shape between two cells by Linked list.

- First we check the top corner position of the letter L. If it does not exist, we use the Icheck function to check for the first cell position (y1, x1) and the top corner position (y1, x2) and use the Icheck function, check for second cell position (y2, x2) and top corner position (y1, x2). If both are correct, return true, meaning there is no cell between each two boxes with the top corner position.
- Similar to the bottom corner, we also use the Icheck function similar to the top corner

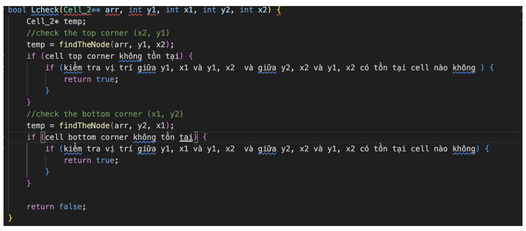
6.2.4 UandZcheck

- Similar to the U and z check function at the pointer, instead of using the linecheck function, the U and Z check at the Linked List use the Icheck function to check whether each line of the letters U and Z is valid or not.
- The UandZcheck function is also divided into two cases. One is the letters U and Z



```
bool Lcheck(Cell_2** arr, int y1, int x1, int y2, int x2) {
    Cell_2* temp;
    //check the top corner (x2, y1)
    temp = findTheNode(arr, y1, x2);
    if (temp == NULL) {
        if (Icheck(arr, y1, x1, y1, x2)) {
            if (Icheck(arr, y2, x2, y1, x2)) {
                return true;
            }
        }
    }
    //check the bottom corner (x1, y2)
    temp = findTheNode(arr, y2, x1);
    if (temp == NULL) {
        if (Icheck(arr, y1, x1, y2, x1)) {
            if (Icheck(arr, y2, x2, y2, x1)) {
                return true;
            }
        }
    }

    return false;
}
```

Figure 34: Code for Lcheck for Linked list



```
bool Lcheck(Cell_2** arr, int y1, int x1, int y2, int x2) {
    Cell_2* temp;
    //check the top corner (x2, y1)
    temp = findTheNode(arr, y1, x2);
    if (cell top corner không tồn tại) {
        if (kiểm tra vị trí giữa y1, x1 và y1, x2  và giữa y2, x2 và y1, x2 có tồn tại cell nào không ) {
            return true;
        }
    }
    //check the bottom corner (x1, y2)
    temp = findTheNode(arr, y2, x1);
    if (cell bottom corner không tồn tai) {
        if (kiểm tra vị trí giữa y1, x1 và y1, x2  và giữa y2, x2 và y1, x2 có tồn tại cell nào không) {
            return true;
        }
    }

    return false;
}
```

Figure 35: Pseudo code for Lcheck by Linked list

horizontally, the other is the letters U and Z vertically

6.3 Game finish verify

6.3.1 Ways to end the game

- The maximum score a player can receive is 370 points (if players use all the suggestion move) or 400 points (if players don't use any suggestion move).
- Whether exiting the game, running out of lives, or completing the game, the players will still have their achievements saved in the Leaderboard.

Ways to end the game:
- When players press ESC to exit the game.
- When you have completed the game board.
- When your life in the game reaches 0.
- When there are no more two valid cells with the same characters but the board is still not completed.

There is only one way to win the game, which is to complete all the boxes , otherwise the player will lose.

6.3.2 checkValidPair for Pointer

- This is function to check whether there exist valid pairs of cells with the same character in the board.

- **Function: bool checkValidPairs(Cell_1** board)**

- Let the loop run from 'A' to 'B'. For each loop, if there exist two boxes with

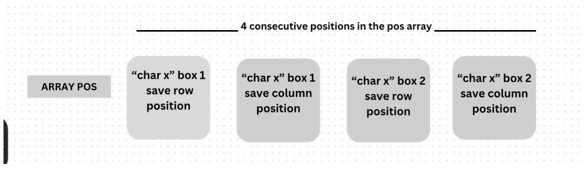

4 consecutive positions in the pos array
ARRAY POS
"char x" box 1 save row position
"char x" box 1 save column position
"char x" box 2 save row position
"char x" box 2 save column position

Figure 36: Array pos

the same characters and are valid in the board, store them in the pos array.

- Use the allcheck function mentioned in section 5.1.6 to check whether the two boxes stored in the pos array match each other. If so, return true, but if the loop runs out and no two boxes match, return false

6.3.3 checkValidPair for Linked List

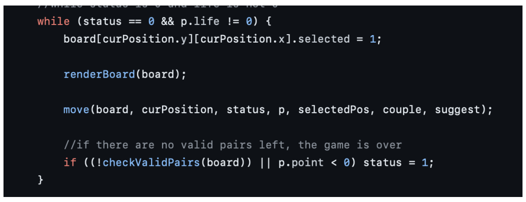
Function: bool checkValidPairs(Cell_2 arr)**

– Similar to pointer, we browse two nodes of all nodes in the board game in turn. If any two nodes have the same character and satisfy all I, L, U, Z checks (allcheck), it returns true, and if no two cells satisfy, it returns false.

6.3.4 Explanation code fishing game

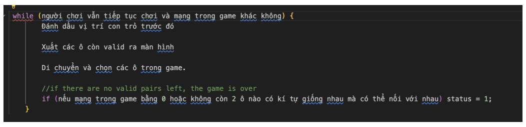
- First we will set a status variable. If the status is 0 then the player can continue playing, if the status is 1, the player loses, or has completed all matching of all boxes, if the status is 2 the player presses ESC to exit the game.

- If the status is 0 and the player still has lives, the game will continue



```
while (status == 0 && p.life != 0) {
    board[curPosition.y][curPosition.x].selected = 1;

    renderBoard(board);

    move(board, curPosition, status, p, selectedPos, couple, suggest);

    //if there are no valid pairs left, the game is over
    if ((!checkValidPairs(board)) || p.point < 0) status = 1;
}
```

Figure 37: Code for status 0



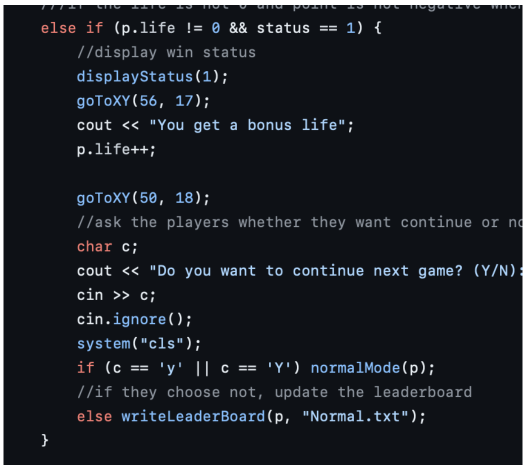
```
while (người chơi vẫn tiếp tục chơi và mạng trong game khác không) {
    Đánh dấu vị trí con trỏ trước đó

    Xuất các ô còn valid ra màn hình

    Di chuyển và chọn các ô trong game.

    //if there are no valid pairs left, the game is over
    if (nếu mạng trong game bằng 0 hoặc không còn 2 ô nào có kí tự giống nhau mà có thể nối với nhau) status = 1;
}
```

Figure 38: Pseudo code for status

- If status is equal to 2, since the player exits, the player's exit achievements will still be in the leaderboard

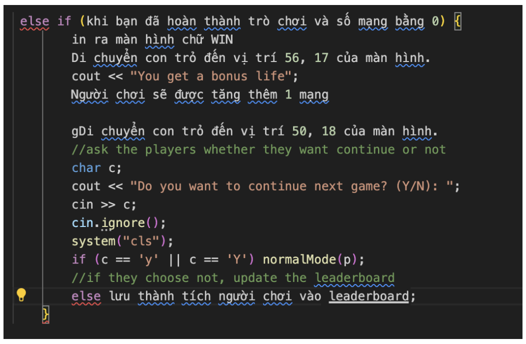
- If the player has 0 live, or the boxes still exists in the board (which means the player's point is under 370), but there are no two boxes with the same character that can be matched, the system prints "YOU LOSE" and saves the player's achievements at that time to the leaderboard.

- If the above two conditions are not met, and player's life in the game is not zero, meaning the player has completed the game, the system prints "YOU WON" to the screen and saves the player's achievement to the leaderboard.



```
else if (p.life != 0 && status == 1) {
    //display win status
    displayStatus(1);
    goToXY(56, 17);
    cout << "You get a bonus life";
    p.life++;

    goToXY(50, 18);
    //ask the players whether they want continue or no
    char c;
    cout << "Do you want to continue next game? (Y/N):
    cin >> c;
    cin.ignore();
    system("cls");
    if (c == 'y' || c == 'Y') normalMode(p);
    //if they choose not, update the leaderboard
    else writeLeaderBoard(p, "Normal.txt");
}
```

Figure 39: Code for status 1 and life != 0



```
else if (khi bạn đã hoàn thành trò chơi và số mạng bằng 0) {
    in ra màn hình chữ WIN
    Di chuyển con trỏ đến vị trí 56, 17 của màn hình.
    cout << "You get a bonus life";
    Người chơi sẽ được tăng thêm 1 mạng

    gDi chuyển con trỏ đến vị trí 50, 18 của màn hình.
    //ask the players whether they want continue or not
    char c;
    cout << "Do you want to continue next game? (Y/N): ";
    cin >> c;
    cin.ignore();
    system("cls");
    if (c == 'y' || c == 'Y') normalMode(p);
    //if they choose not, update the leaderboard
    else lưu thành tích người chơi vào leaderboard;
}
```

Figure 40: Pseudo code for status 1 and life != 0

7 Advance Features

7.1 Color effect

Function:
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), color)

- This function has two parameters. One for the text and one for the text color that will be printed to the console.

- The text can be gotten by using GetStdHandle(STD_OUTPUT_HANDLE) function. GetStdHandle() is a Microsoft Windows API function that retrieves a handle to the standard output device associated with the current process. In this case, STD_OUTPUT_HANDLE specifies the standard output device, which is typically the console window where text is displayed.

- The color parameter is an integer value specifying the color attributes to set

- We use this function to set the color of text printed to the console.

7.2 Sound effect

Function:
PlaySound(TEXT(path), NULL, SND_FILENAME — SND_ASYNC)

- PlaySound function in Windows API is used to play a sound specified by a filename or resource identifier.

- TEXT is typically used to define string literals that are compatible with both Unicode and ANSI. The first parameter is a pointer to a string that specifies the sound filename.

- The third parameter is a set of flags that specify how the sound should be played. SND_FILENAME indicates that the first parameter is a filename, and SND_ASYNC specifies that the sound should be played asynchronously, meaning that the function returns immediately, and the sound plays in the background.

All audio files:
- Enter sound: enter sound
- Error sound: error sound
- Move sound: move sound
- Win sound: win sound
- Lose sound: lose sound
- Background: background sound. All sound in the source code.

Note: The path of the sound file must be the actual path in your device, so you must rewrite the path in the source code to get the sound.

7.3 Background

Function: void getNormaldBg(char bg[][41])

- This function reads in the contents of a text file and store the background of the normal mode into an array of characters.

- The steps of this function:
1. Open the file of normal background
2. If the file can be opened, loop through the file to get the file and stores the character into the array.
3. After looping, close the file.
4. If the file cannot open, fill the array with space character.

Function: void displayNormalBg(char bg[][41], int x, int y)

- When two blocks are matching, the program will draw background content corresponding to those emptied cells.

- The following steps for the function:
1. Change the color of the background.
2. Use the coordinate (x, y) to find the location of the emptied cell and go to there.
3. Draw the part of the background on that location.
4. Change the color back to white.

Note: Note: The path of the background files must be the actual path in your device, so you must rewrite the path in the source code to get the background.

Note: functions "getHardBg(char bg[][51])" and "displayHardBg(char bg[][51], int x, int y)" of the hard mode work the same as the normal mode.

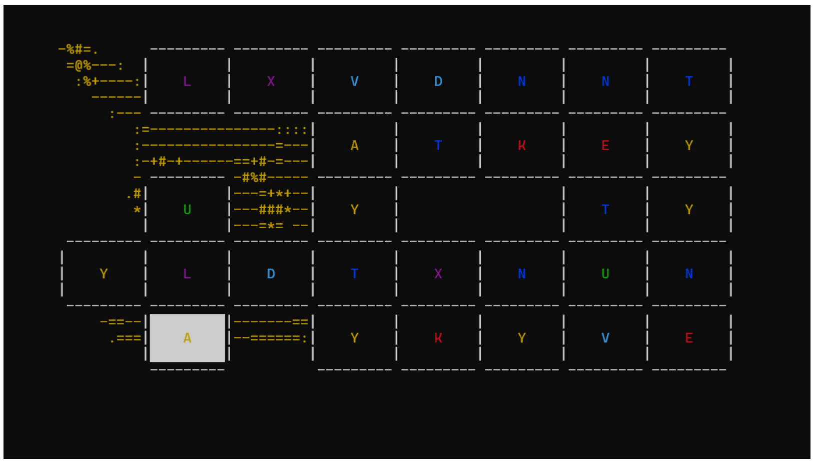


Figure 41: Background reveal

7.4 Visual effect

7.4.1 For Pointer

Function void Cell_1::drawBox (int color)
- This is the function used to draw cells in the board game. If the cell to be drawn is not yet matched, this function will perform the drawing, but if it is matched, it will return. When a cell is **selected**, the function will draw a cell with **a white border**, otherwise it will be black.
- We will have a loop used to draw a predefined two-dimensional array as the image of a game cell.

Function void Cell_2::deleteBox()
- This is the function to delete cell2. When the cells are matched, the system will use this function to delete cell2.

Function void renderBoard(Cell_1** board)

- This is a function that, from a given two-dimensional Board pointer, draws a game board consisting of many cells. We use a loop that runs from left to right and then from top to bottom, each loop uses the drawbox() function.

7.4.2 For Linked list

Function void Cell_2::drawBox(int color)
- Similar to point, this is the board game's cell2 drawing function.

Function void Cell_2::deleteBox()
- This is the function to delete cell2. When the cells are matched, the system will use this function to delete cell2.

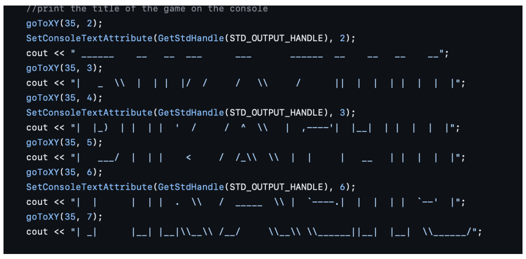
Function void renderList(Cell_2** arr)
- This is Hard Mode's board game drawing function, creating a loop through each node of the Cell2 structure list. Each Node location uses the **drawBox()** function to draw.

7.4.3 The word

ASCII art sources:

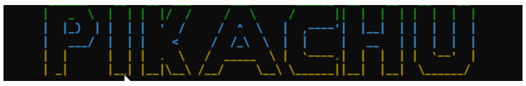
- Game name: asciiart
- Other arts: Create ASCII text banners online

- In addition, the large letters created by the characters "-", "_", "—", "/" also make the board more lively.
- For example, the word PIKACHU.



```
    //print the title of the game on the console
    goToXY(35, 2);
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 2);
    cout << " ______    __   __  ___      ___       ______  __    __   __    __";
    goToXY(35, 3);
    cout << "|   _  \\  |  | |  |/  /     /   \\     /      ||  |  |  | |  |  |  |";
    goToXY(35, 4);
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 3);
    cout << "|  |_)  | |  | |  '  /     /  ^  \\   |  ,----'|  |__|  | |  |  |  |";
    goToXY(35, 5);
    cout << "|   ___/  |  | |    <     /  /_\\  \\  |  |     |   __   | |  |  |  |";
    goToXY(35, 6);
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 6);
    cout << "|  |      |  | |  .  \\   /  _____  \\ |  `----.|  |  |  | |  `--'  |";
    goToXY(35, 7);
    cout << "| _|      |__| |__|\\__\\ /__/     \\__\\ \\______||__|  |__|  \\______/";
```

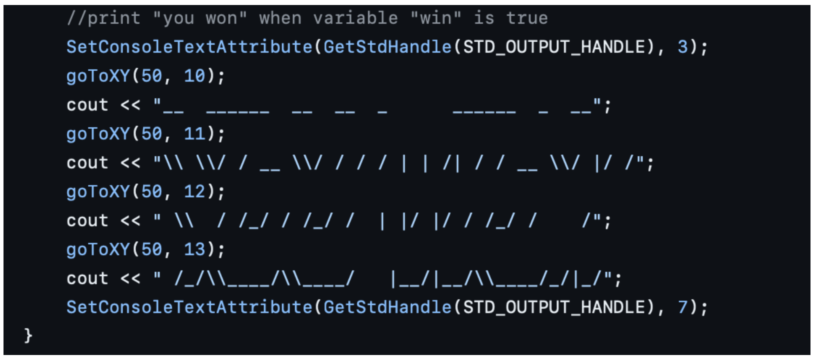
Figure 42: Code word PIKACHU



```
 ______    __   __  ___      ___       ______  __    __   __    __
|   _  \  |  | |  |/  /     /   \     /      ||  |  |  | |  |  |  |
|  |_)  | |  | |  '  /     /  ^  \   |  ,----'|  |__|  | |  |  |  |
|   ___/  |  | |    <     /  /_\  \  |  |     |   __   | |  |  |  |
|  |      |  | |  .  \   /  _____  \ |  `----.|  |  |  | |  `--'  |
| _|      |__| |__|\__\ /__/     \__\ \______||__|  |__|  \______/
```

Figure 43: word PIKACHU

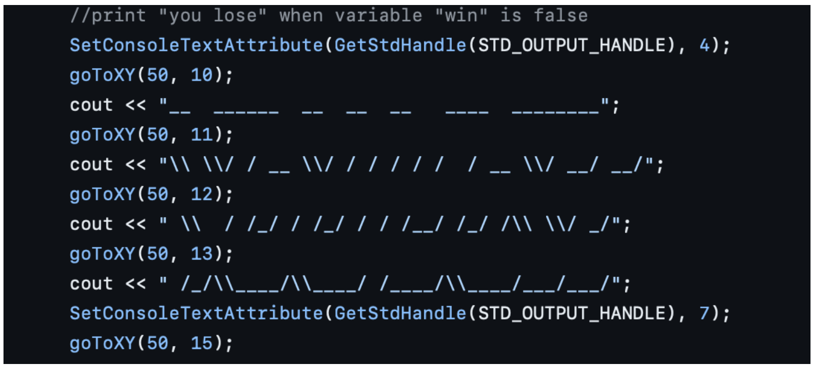
- The word WON when the player wins



```
    //print "you won" when variable "win" is true
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 3);
    goToXY(50, 10);
    cout << "__  ______  __  __  _      ______  _  __";
    goToXY(50, 11);
    cout << "\\ \\/ / __ \\/ / / / | | /| / / __ \\/ |/ /";
    goToXY(50, 12);
    cout << " \\  / /_/ / /_/ /  | |/ |/ / /_/ /    /";
    goToXY(50, 13);
    cout << " /_/\\____/\\____/   |__/|__/\\____/_/|_/";
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 7);
}
```

Figure 44: Word WIN

- The word LOSE when the player lose



```
    //print "you lose" when variable "win" is false
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 4);
    goToXY(50, 10);
    cout << "__  ______  __  __  __   ____  ________";
    goToXY(50, 11);
    cout << "\\ \\/ / __ \\/ / / / / /  / __ \\/ __/ __/";
    goToXY(50, 12);
    cout << " \\  / /_/ / /_/ / / /__/ /_/ /\\ \\/ _/";
    goToXY(50, 13);
    cout << " /_/\\____/\\____/ /____/\\____/___/___/";
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 7);
    goToXY(50, 15);
```

Figure 45: Word LOSE

7.5 Leaderboard

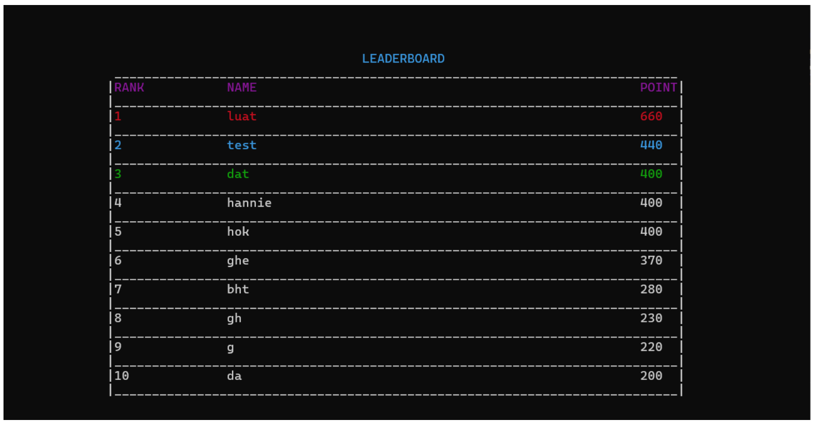
- If the players enter the Leaderboard on the menu screen, the program will ask the players whether they want to see normal or hard mode or hidden mode leaderboard.

- The leaderboard works base on following functions.

Function: void sortLeaderBoard (Player list[], int n)

-This function is used to sort the list of Player with n elements for ranking the leaderboard. The function performs the Bubble sort algorithm to sort the players information in descending order based on their best score.

Function: void printLeaderBoard(char &c)



```
                         LEADERBOARD
 ______________________________________________________________
|RANK          NAME                                       POINT|
|______________________________________________________________|
|1             luat                                       660  |
|______________________________________________________________|
|2             test                                       440  |
|______________________________________________________________|
|3             dat                                        400  |
|______________________________________________________________|
|4             hannie                                     400  |
|______________________________________________________________|
|5             hok                                        400  |
|______________________________________________________________|
|6             ghe                                        370  |
|______________________________________________________________|
|7             bht                                        280  |
|______________________________________________________________|
|8             gh                                         230  |
|______________________________________________________________|
|9             g                                          220  |
|______________________________________________________________|
|10            da                                         200  |
|______________________________________________________________|
```

Figure 46: Leaderboard

- The function will print the leaderboard on the console based on the text file that stores all information of 10 players who have highest score.

- The function follows these steps:
1. Get the character c that players enter to know which file to open. If players enter "N" or "n", the program opens the normal file, and the hard file (enter "h" or "H") and hidden file (enter "i" or "I") are similar to the normal file.
2. If players enter wrong character, the function will end.
3. Get the information in the file.
4. Print all the information to the console, change the color of the information of the 3 players with the highest scores.
5. close the file.
6. After that, the function will delay until players press any key to end.

Function: void writeLeaderBoard(Player p, string filename)

- After finishing playing, the program will write the players information into the correct file of the mode that they have played by calling this function.

- It follows these steps:
1. Open the correct filename to write the information. If the file doesn't have any information, fill it with the new information, close the file and end the function.
2. If the file is not empty, store all the information of the file into a temperature "list" array of Player type.
3. If the player information is already in the file, the program will update the score if it has a higher score than the current one.
4. If the player information is not in the file, the program will check if the list is over 9 or not. If it is not over 9, put the information into the end of the "list" and sort it. If it is, still put the information into the end of the "list" and sort it, after that the program will only take 10 players who have highest score.
5. After that, the program will rewrite the updated "list" back into the file, free the "list" and then close the file.

Note: The path of all leader files also must be the actual path in your device, so you must rewrite the path in the source code to get the files.

7.6 Move suggestion (only for the normal and hidden mode)

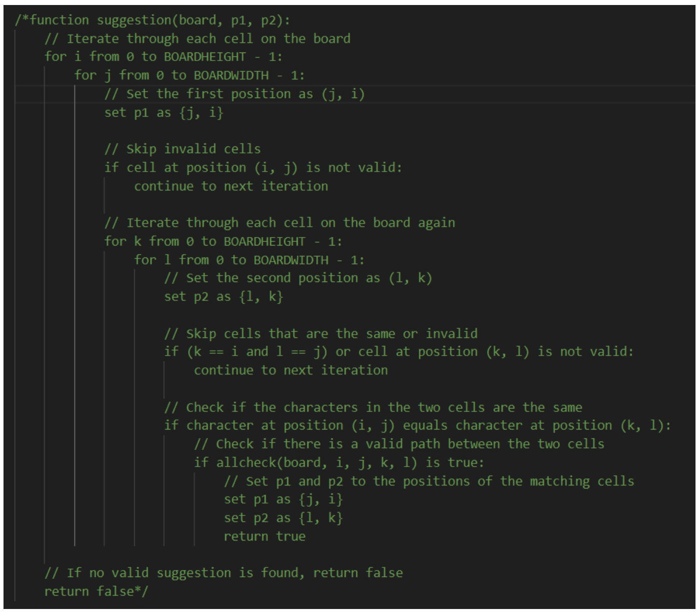
- The game will give players 3 chances to get free move. If the players has used all the chances, the players cannot use them until the next board.

- If the players use suggestion the program will delete a valid pair if it can find out, and the players will gain 10 points and lose 1 chance.

Function: bool suggestion(Cell_1 board, Position & p1, Position & p2)**

- If a matching pair of boxes is found, the function returns 1 then sets p1 and p2 to the index of those boxes. If not, it returns 0.

- Here the pseudo code for this function

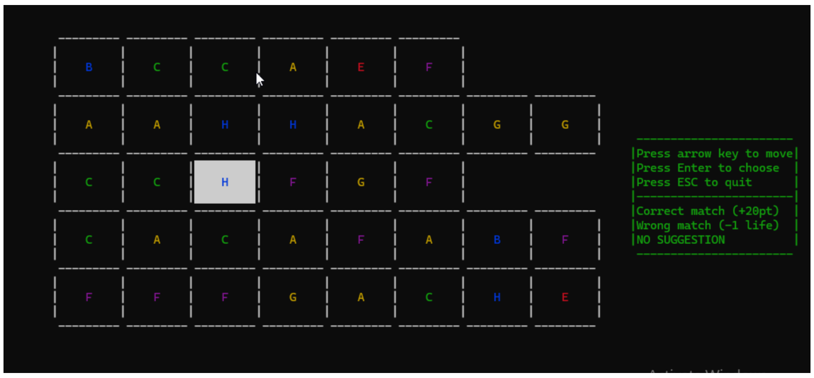


```
/*function suggestion(board, p1, p2):
    // Iterate through each cell on the board
    for i from 0 to BOARDHEIGHT - 1:
        for j from 0 to BOARDWIDTH - 1:
            // Set the first position as (j, i)
            set p1 as {j, i}

            // Skip invalid cells
            if cell at position (i, j) is not valid:
                continue to next iteration

            // Iterate through each cell on the board again
            for k from 0 to BOARDHEIGHT - 1:
                for l from 0 to BOARDWIDTH - 1:
                    // Set the second position as (l, k)
                    set p2 as {l, k}

                    // Skip cells that are the same or invalid
                    if (k == i and l == j) or cell at position (k, l) is not valid:
                        continue to next iteration

                    // Check if the characters in the two cells are the same
                    if character at position (i, j) equals character at position (k, l):
                        // Check if there is a valid path between the two cells
                        if allcheck(board, i, j, k, l) is true:
                            // Set p1 and p2 to the positions of the matching cells
                            set p1 as {j, i}
                            set p2 as {l, k}
                            return true

    // If no valid suggestion is found, return false
    return false*/
```

Figure 47: Pseudo code for suggestion move

7.7 Stage difficulty increase - Hard mode

How to play When a matching pair is found, it moves all the blocks in the same row from the right side of the deleted pair to the left.



```
| B | C | C | A | E | F |
| A | A | H | H | A | C | G | G |
| C | C | H | F | G | F |
| C | A | C | A | F | A | B | F |
| F | F | F | G | A | C | H | E |

 ------------------------
|Press arrow key to move|
|Press Enter to choose  |
|Press ESC to quit      |
|-----------------------|
|Correct match (+20pt)  |
|Wrong match (-1 life)  |
|NO SUGGESTION          |
 ------------------------
```

Figure 48: Hard mode

How to implement this mode This mode uses the Linked list data structure. Each row will be a list of many nodes. Every time 2 cells match, the node of that row will be deleted. When deleted, the lists will be shortened, and the nodes will move to the left. We will have a function called renderList to update the table.

- The **Init**, **matching**, **checkValidPair functions** and **Cell2 structure** are newly created to match the Linked List data structure explained in the above sections.

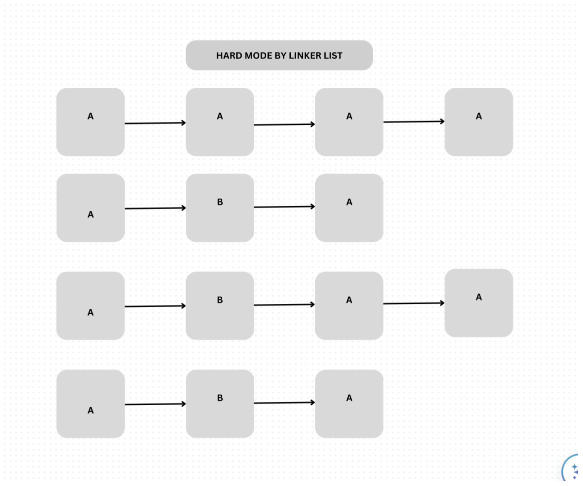
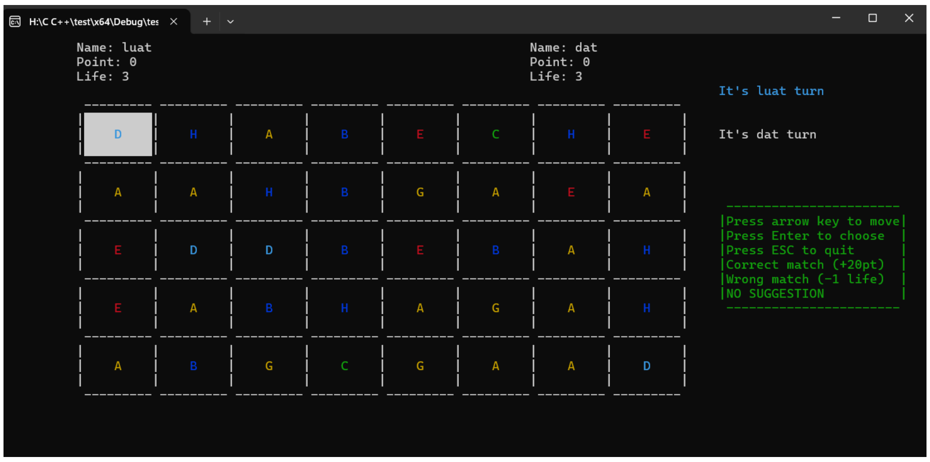


Figure 49: Linked List

7.8 PVP mode

How to play: This mode is based on the Normal mode. In this mode, 2 players will find a valid pair in turn. The rule is similar to the Normal mode, but there will no suggestion. There is only one player win the game if 2 scores are different. If 2 scores are the same, it will be draw.

How to implement this code: To do this, we use the code of Normal mode with



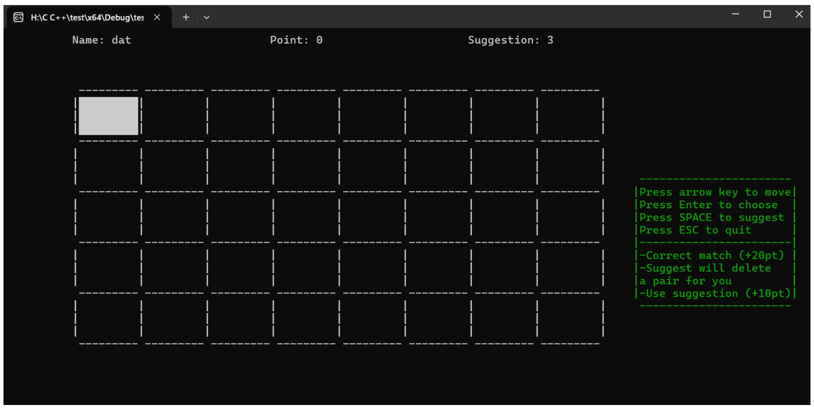
```
H:\C C++\test\x64\Debug\tes

Name: luat                                          Name: dat
Point: 0                                            Point: 0
Life: 3                                             Life: 3
                                                                          It's luat turn

|   D   |   H   |   A   |   B   |   E   |   C   |   H   |   E   |         It's dat turn

|   A   |   A   |   H   |   B   |   G   |   A   |   E   |   A   |
                                                                          ------------------------
|   E   |   D   |   D   |   B   |   E   |   B   |   A   |   H   |         |Press arrow key to move|
                                                                          |Press Enter to choose  |
|   E   |   A   |   B   |   H   |   A   |   G   |   A   |   H   |         |Press ESC to quit      |
                                                                          |Correct match (+20pt)  |
|   A   |   B   |   G   |   C   |   G   |   A   |   A   |   D   |         |Wrong match (-1 life)  |
                                                                          |NO SUGGESTION          |
                                                                          ------------------------
```

Figure 50: PvP mode

some addition. We get 2 players' information with 2 Player variable p1 and p2. There also a integer variable called cur_player to know that whose turn is coming.

7.9 Hidden mode

How to play: This is an additional feature based on the idea of memorize game. The characters and colors are hidden most of the time players play. Players only peek the whole board at the beginning of the game or after you choose a pair of boxes. The rule is similar to the Normal mode but there will no life point.



```
H:\C C++\test\x64\Debug\tes

Name: dat                    Point: 0                    Suggestion: 3

                                                                          ------------------------
                                                                          |Press arrow key to move|
                                                                          |Press Enter to choose  |
                                                                          |Press SPACE to suggest |
                                                                          |Press ESC to quit      |
                                                                          |-----------------------|
                                                                          |-Correct match (+20pt) |
                                                                          |-Suggest will delete   |
                                                                          |a pair for you         |
                                                                          |-Use suggestion (+10pt)|
                                                                          ------------------------
```

Figure 51: Hidden mode

How to implement this code: They key idea of this mode is to add another function for struct Player named "drawHiddenBox" to hide the characters. With this function, we can write a function called "renderHiddenBoard" to create a hidden board for this mode. To reveal and delete boxes, we can use other functions in struct Player.

8 Pointer and Linked list Comparison

- In game development, pointers and linked lists serve different purposes and have distinct characteristics. They have both advantages and disadvantages.

Pointers

Purpose: Pointers are used to store memory addresses, allowing direct access to data stored at those addresses. In this game, pointers can be used to efficiently access and manipulate game objects.

Advantages:
- Efficient memory usage: Pointers consume minimal memory overhead, making them suitable for storing large amounts of data efficiently.
- Direct access: Pointers provide direct access to data in memory, allowing for fast retrieval and modification of game objects.

Disadvantages:
- Manual memory management: Pointers require manual memory allocation and deallocation, which can lead to memory leaks or segmentation faults if not handled properly.
- Fixed size: Although pointers allow us to allocate the memory for game objects such as board, it still need get the fixed size in the run time of the game.

Linked List

Purpose: Since linked list is structure composed of nodes, where each node contains a data element and a reference (pointer) to the next node in the sequence, linked lists can be used to manage dynamic collections of game objects.

Advantage:
- Dynamic size: Linked lists can grow or shrink dynamically, allowing for efficient insertion and removal of elements at any position.
- Simple insertion and deletion: Inserting or deleting elements in a linked list is relatively simple and does not require shifting elements, unlike arrays.

Disadvantage:
- No random access: Traversing a linked list requires sequential access from the head node, which can be less efficient than direct access, especially for large lists.
- Memory overhead: Linked lists require extra memory to store pointers/references for linking nodes together. This overhead can be significant compared to arrays, especially for small data elements.

9 Program executing instruction

9.1 G++ version

- The program was built on from this version: g++ (MinGW.org GCC-6.3.0-1) 6.3.0

9.2 How to compile

- All source codes are located in the "Source" folder.

- Sound files in "Sound" folder, background images and leaderboard records in "Text" folder.

- The program uses PlaySound function, so **make sure that you links the actual path to the sound file in the function after downloading the sound file on your device.**

- Note: all text files about leaderboard and background must be connected to the actual path in your device to make the program run properly. So you must rewrite the path for all place has the text files.

- To run the program, you compile and run in the "main.cpp" file.

10 Project format exlaination

10.1 Header file

These files are in "Source" folder:

- Structure.h: This file contains the declaration of the below things:

  Struct Cell2, Cell1, Position and Player.

  Functions used to process cells in the screen such as drawing, deleting or hiding cells.

- Setting.h: This file contains the declaration of the below things:

  It is included in the console setting section. Used to set display effects

  The functions that are used to play sound.

- Checking.h: This file contains the declaration of the below things:

  Declare matching function between two cells (I, U, L, Z check, ...) for pointer.

  Declare a function to check for cells with the same characters and matched for pointer.

- CheckingLinkedList.h: This file contains the declaration of the below things:

  Declare matching function between two cells (I, U, L, Z check, ...) for Linked List.

  Declare a function to check for cells with the same characters and matched for Linked List.

- Utility.h: This file contains the declaration of the below things:

  Declare functions to process data from the LeaderBoard.

  Declare a function to get information from the player.

  Declare functions that display winning or losing status words.

  Declare a function to print the background of each mode

- Normal.h: This file contains the declaration of the below things:

  Declare the control function of normal mode.

  Declare board processing functions (eg init, delete, ..).

  Declare a function that displays the game screen of normal mode, and the game completion conditions of this mode.

- Hard.h: This file contains the declaration of the below things:

  Declare the control function of Hard mode.

  Declare board processing functions (eg init, delete, ..) by Linked List.

  Declare a function that displays the game screen of Hard mode, and the game completion conditions of this mode.

- Hidden.h: This file contains the declaration of the below things:

  Declare operating functions in Hidden mode similar to Normal mode.

- PvP.h: This file contains the declaration of the below things:

  Declare operating functions in PvP mode similar to Normal mode.

10.2 ".cpp" files

- These files are in “MatchingGame” folder. They contain all definitions of the functions that are declared in the Header files and a “main.cpp” contains the main() function.

10.3 Text file

- There are 2 text files named “Hcmus.txt” and “pika.txt” located in “Text” folder. They are used to display the background of the game.
- There are 3 text files named “Normal.txt”, "Hard.txt” and “Hidden.txt” located in “Text” folder. They are used to save player achievements in each mode to the Leaderboard.

10.4 Sound

- There are 6 “.wav” file located in “Sound” folder. They are used to play sound when playing the game.

11 Demo video

Here is demonstration video on how to play.

12 Reference

- PhVanMin check1.cpp, all sources , check2.cpp all sources

- PhVanMin Normal.cpp move function (from line 56 to line 323), normalMode function (from line 325 to 387)

- PhVanMin DifficultmMode.cpp move function (from line 78 to line 343), difficultMode function (from line 345 to 407)

- PhVanMin Utility.cpp displayStatus function (from line 112 to line 137), mainMenu function (from line 168 to line 319)

- PhVanMin Normal mode background

- Louis Tran Hard mode background

- Louis Tran enter sound, error sound, move sound, win sound, lose sound, background sound. All sound in the source code.

- Max O'Didily Youtube Link. How to Stop and Play Music Using C++ (Simple) (PlaySound).

- Microsoft Windows API GetStdHandle, SetConsoleTextAttribute, setConsoleCursorPosition, setConsoleCursorInfo

- Chat GPT Link