TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

CÔNG NGHỆ THÔNG TIN

PROJECT REPORT

THE MATCHING GAME

Subject: Programming Techniques

CLASS: 23CLC08

Sinh viên thực hiện:	Lê Thị Minh Thư 23127488
	Nguyễn Ngọc Mai Xuân 23127284
Giảng viên hướng dẫn:	GV. Bùi Huy Thông
	GV. Nguyễn Trần Duy Minh

TP HCM, 11 - 4 - 2024

Contents

1 Structure of game file

1.1 Structure of game

• Play game
• Help
• LeaderBoard
• Exit

1.2 Structure of file

1.2.1 Header files

• BoardAndCell.h
• Check.h
• Console.h
• Menu.h
• MoveCursor.h

1.2.2 Resource files

Audio files include:
• background2.wav
• effect.wav
• enter.wav
• error.wav
• lose.wav
• move.wav
• placed.wav
• win.wav

1.2.3 Source files

• BoardAndCell.cpp: Perform operations with tables and cells
• Check.cpp: Check and draw check lines I, L, U, Z
• Console.cpp: includes graphics, mouse pointer, play sound and screen handles
• Menu.cpp: includes functions that draw and print to the screen to create visual effects
• MoveCursor.cpp: Move the mouse pointer with the key in the Pikachu panel, and return the result of the key selection

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 4 options: + Play game: choose to play the game with a created account
+ Help: choose to guide the user on how to play
+ LeaderBoard: view the leaderboard of scoring players
+ Exit: exit the game
- The player uses the arrow keys to navigate between options and press Enter to select

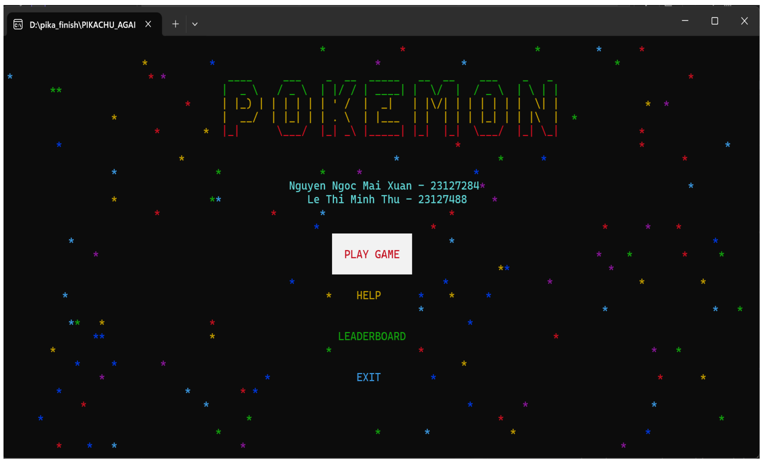
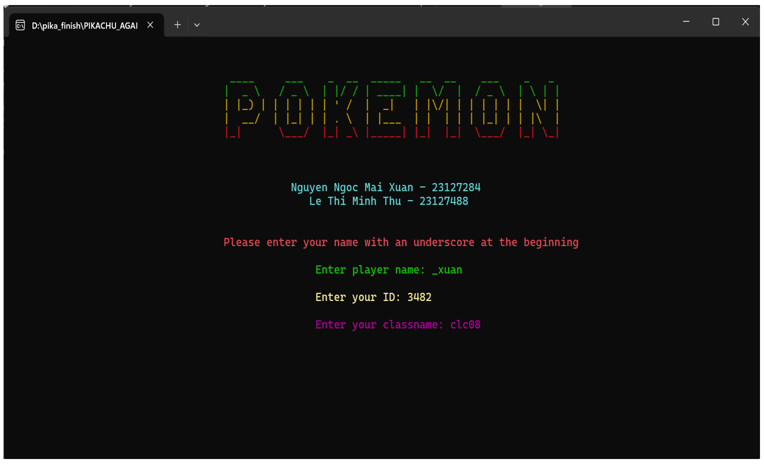

D:\pika_finish\PIKACHU_AGAI
POKEMON
Nguyen Ngoc Mai Xuan - 23127284
Le Thi Minh Thu - 23127488
PLAY GAME
HELP
LEADERBOARD
EXIT

Figure 1: Main menu

2.2 Login screen

When the player selects Play game, the system will prompt the player to enter their name, ID, classname.
Note: The player's name when entered must have an underscore at the beginning

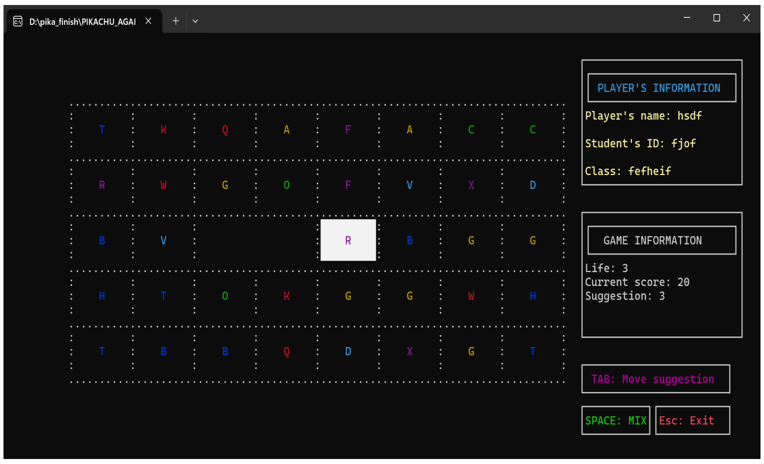


```
Nguyen Ngoc Mai Xuan - 23127284
Le Thi Minh Thu - 23127488

Please enter your name with an underscore at the beginning

Enter player name: _xuan

Enter your ID: 3482

Enter your classname: clc08
```

Figure 2: Login screen

2.3 Game play

After selecting the "play game" mode, the game will be displayed on the screen. Players will use the arrow keys to move and the Enter key to select two suitable squares. Each exact matches will earn points, and each incorrect match will be deducted points. Player can press TAB to get move suggestion, press SPACE to mix and press ESC to exit.



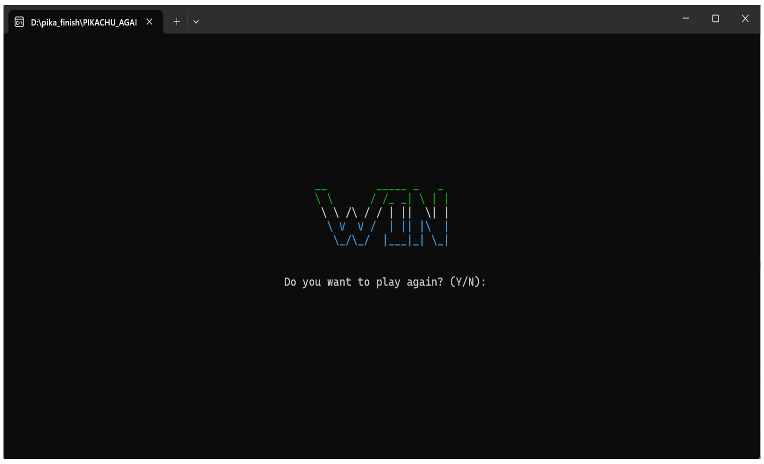
```
D:\pika_finish\PIKACHU_AGAI

  T   K   Q   A   F   A   C   C      PLAYER'S INFORMATION
                                     Player's name: hsdf
  R   W   G   O   F   V   X   D      Student's ID: fjof
                                     Class: fefheif
  B   V           R   B   G   G
                                     GAME INFORMATION
  H   T   O   K   G   G   W   H      Life: 3
                                     Current score: 20
  T   B   B   Q   D   X   G   T      Suggestion: 3

                                     TAB: Move suggestion
                                     SPACE: MIX  Esc: Exit
```

Figure 3: Play game

2.4 Game finishing

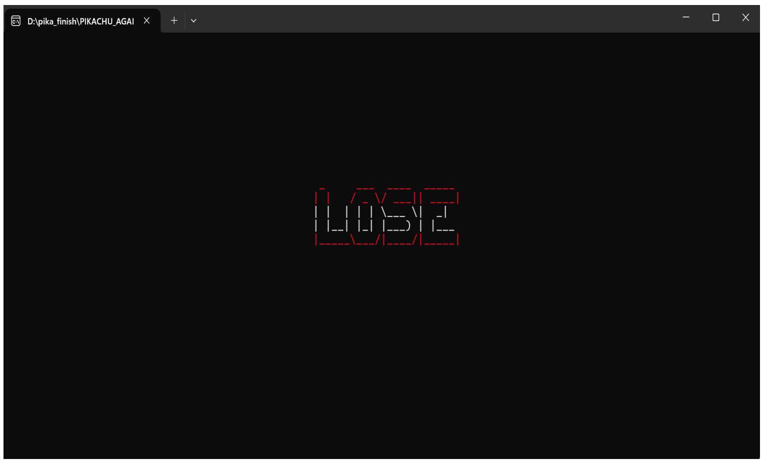
Conditions for players to win and lose:

• Victory: All pairs in the table are connected (the board has 20 pairs with the same characters), but it is not always possible to connect them all because the player's first moves select 2 cells and delete 2 cells, gradually creating a path for the remaining cells, if the player deletes unwisely, it will block the path of the remaining cells, resulting in winning the game but a low score



```
D:\pika_finish\PIKACHU_AGAI

__        _____ _   _
\ \      / /_ _| \ | |
 \ \ /\ / / | ||  \| |
  \ V  V /  | || |\  |
   \_/\_/  |___|_| \_|

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

Figure 4: Victory screen



```
 _     ___  ____  _____
| |   / _ \/ ___|| ____|
| |  | | | \___ \|  _|
| |__| |_| |___) | |___
|_____\___/|____/|_____|
```

Figure 5: Defeat screen

- Defeat: Deducted all 3 lives.

2.5 Use case diagram

Use case diagram is used to provide a basic summary of how the system works that users can directly interact with.

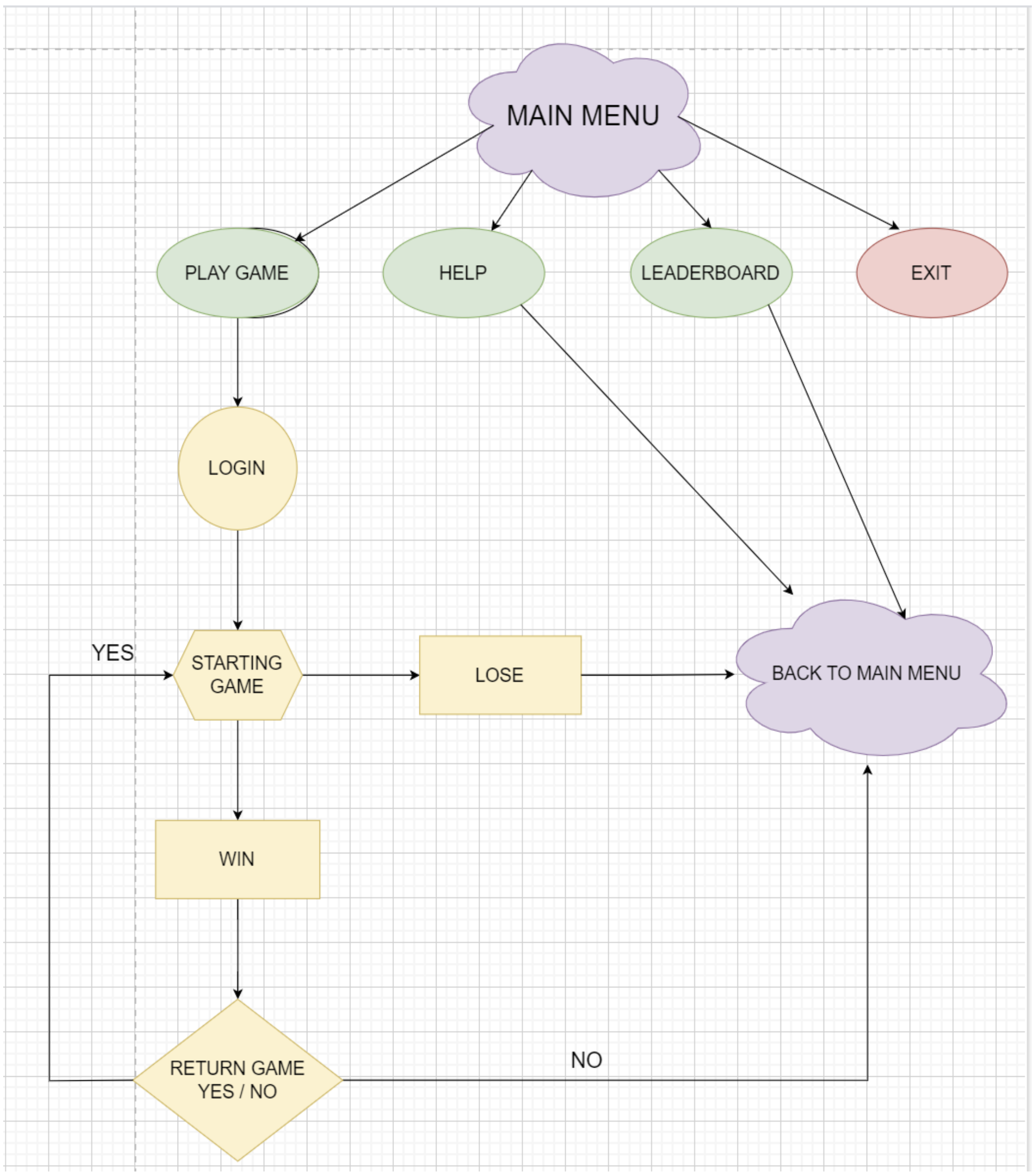
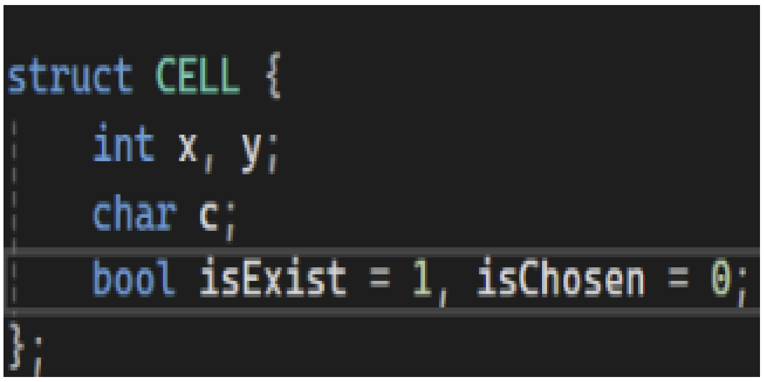


Figure 6: Flowchart

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

3.1 Create random board

- Create a table with a specified size of 8x5. In particular, 8 is the general width corresponding to the number of cells of a horizontal row, 5 is the height corresponding to the number of cells of a vertical row.
- Each cell in the table is defined with the following structure:



```
struct CELL {
    int x, y;
    char c;
    bool isExist = 1, isChosen = 0;
};
```

Figure 7: Illustrated

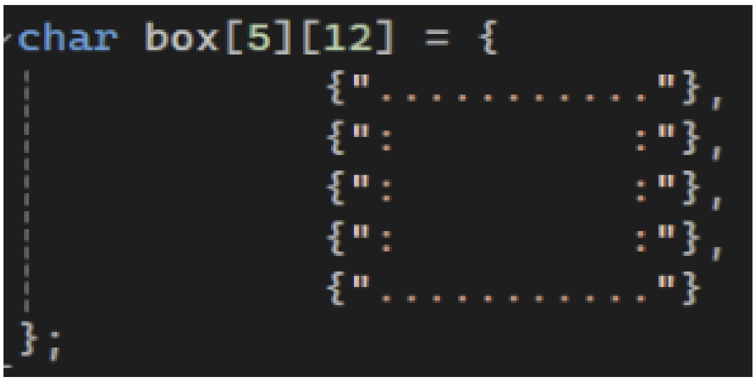
1. Include:
   (a) The integer variable x,y is the coordinate that defines the position of the cell, with x corresponding to the vertical row, y corresponding to the horizontal row of the cell.
   (b) The letter 'c' is the letter of that cell.
   (c) Variable the bool type " isExist " to check if this cell still exists, or has been deleted, the bool type variable " isChosen " to check if this cell is being selected by the player.

- We will allocate 1-dimensional dynamic arrays to the CELL structure type (defined as above). The number of allocated memory areas is equal to the number of cells in a Pikachu table of 8x5 (WIDTH x HEIGHT).

- The 40 cells in the 8x5 table are defined as follows:

1. Location: At each cell, x coordinates and y coordinates correspond to the vertical and horizontal row indexes in the table, respectively. We use 2 loops for nested, the first loop for y to run from 0 to (HEIGHT) the second loop for x runs from 0 to (WIDTH – 1). At each loop, assign cell[n].x and cell[n].y (cell has the data type is the CELL structure) for x and y.
2. Character: Since there are 40 cells, there will be 20 pairs with the same letter. We let the for loop run from 0 to (20 -1). At each iteration, perform:
   (a) Step 1: Randomize a random character from 'A' to 'Z'.
   (b) Step 2: Assign the ith cell and the th cell (40 – i) to the same random character above to make sure that there are always 2 cells with the same character.
   (c) Step 3: After 40 table cells have been initialized assigned specified characters, use the Mix-Board function with the function of " mixing " table cells at random.

- We define the DrawCell function in BoardAndCell.h to draw a 5x12 rectangle (which is an array of char characters[5][12]) around that cell, in the middle of the cell (in row 2 column 5) containing the capital character corresponding to that cell.



```
char box[5][12] = {
                {"...........",
                {":         :"},
                {":         :"},
                {":         :"},
                {"..........."}
};
```

Figure 8: Illustrated

- We build the DrawCell function as follows:
  1. Use the gotoxy(int x, int y) function to move to the coordinates (x, y) on the console screen.
  2. DrawCell (CELL cell, int b-color = BRIGHT-WHITE):
     (a) The cell variable is passed to the function to get the position (cell.x, cell.y) of that cell stored in the table.
     (b) The variable b-color to pass in the background color is highlighted on the inside of the cell.
        i. Step 1: Check if the cell in place (cell.x, cell.y) exists (isExist), return if does not exist.
        ii. Step 2: Draw the outer border for each: (Loop i from 0 to 4, then move to the coordinates (X, Y) on the console screen: X = (cell.x + 1) * 10; Y = (cell.y + 1)*4 + i; then print to box[i])
     (c) Draw background colors and characters. Check if the cell is checked (isChosen)
        i. If selected, do :
           A. Draw a background color inside a cell (with the default color b-color=BRIGHT-WHITE)
           B. Navigate to the coordinates (X, Y) (coordinates between cells) on the console screen. Print the corresponding character of that cell. X = (cell.x + 1)*10 + 5; Y = (cell.y + 1)*4 + 2
        ii. If not selected, do :
           Navigate to the coordinates (X, Y) (coordinates between cells) on the console screen. Print the corresponding character of that cell. X = (cell.x + 1)*10 + 5; Y = (cell.y + 1)*4 + 2
     (d) Draw a Pikachu board with rectangular shapes with complete successive cells:
        Given the loop from i = 0 -> WIDTH*HEIGHT - 1), then pass the DrawCell function to draw each cell until the end.

3.2 Matching

Two cells on the Pikachu table are identical if 1 of the 5 lines connecting Next, I, L, Z, U is satisfied.

3.2.1 Next Line:

When two selected cells are consecutive and have the same character together, the Next link is satisfied

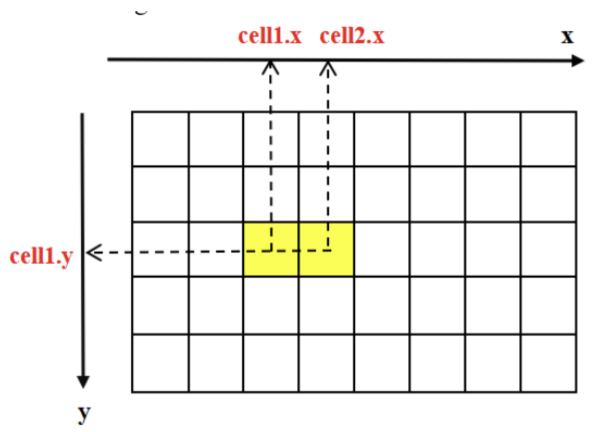
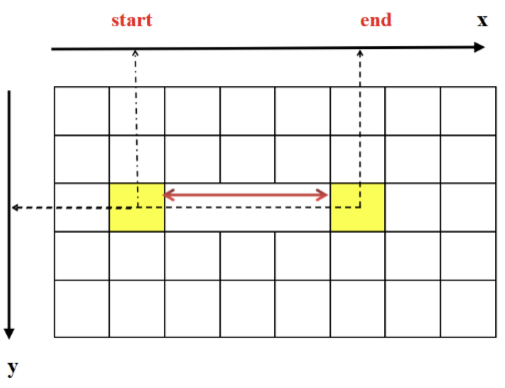


Figure 9: Illustrated

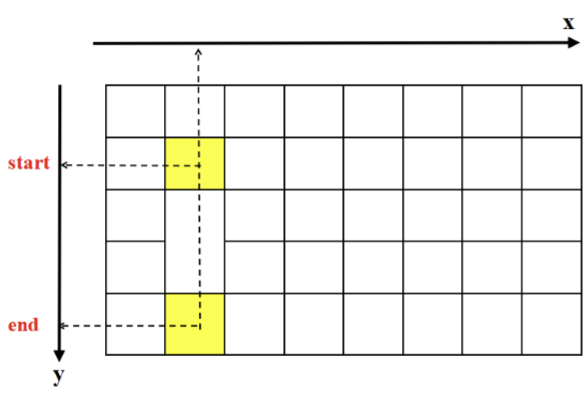
- Build the NextCheck function
  1. Step 1: Check if the characters of cells cell1 and cell2 are the same. Otherwise, return false
  2. Step 2: Check if the two cells under consideration are two consecutive cells. If so, return true. On the contrary, return false.

3.2.2 I-line:

- The bool function Icheck (CELL* cell, CELL cell1, CELL cell2) checks the line to satisfy the I-line: cell is Pikachu table, Cell1 and Cell2 are two cells passed in to consider the path.



Case 1



Case 2

1. Step 1: Check if two cells are on the same row (same y coordinates) or on the same column (same x coordinates).If not, return false, If yes:
   (a) Case 1: Two cells are on the same horizontal row (same y coordinates in Pikachu table)
      i. Step 1: Assign start = min (cell1.x, cell2.x) , end = max (cell1.x, cell2.x).
      ii. Step 2: Check if there is a cell (isExist) on the line going from cell1 to cell2. If so, return false: Looping i from start + 1 to end - 1, then Check cell status [pos], with pos = cell1.y*WIDTH + i , corresponding to the position of that cell in the Pikachu table.
   (b) Case 2: Two cells are on the same vertical row (with the same x coordinates in the Pikachu table)
      i. Step 1: Assign start = min (cell1.y, cell2.y) , end = max (cell1.y, cell2.y).
      ii. Step 2: Check if there are any cells (isExist) on the line going from cell1 to cell2. If so, return false: Looping i from start + 1 to end - 1, then check the cell status [pos], with pos = i*WIDTH + cell1.x , which corresponds to the position of that cell in the Pikachu table

In the three-link test function L, Z, and U, we use the LineCheck function (CELL* cell, CELL cell1, CELL cell2) as a subfunction, which checks whether the straight joints are satisfied to join another line to create a path.

3.2.3 Line L

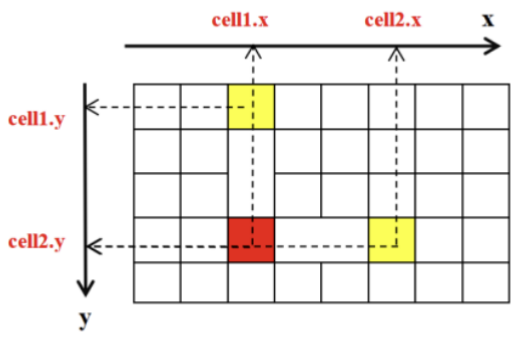


Figure 10: L case

Consider cell CELL cell3 with coordinates (cell1.x, cell2.y):
To check whether cell1 and cell2 can be connected along an L-line, we need to use the LineCheck function to check two small lines including the line between cell cell1 and cell 3 and the line between cell cell2 and cell 3. If one of the 2 lines is not satisfied, it indicates that the join L is not satisfied

The bool function Lcheck (CELL* cell, CELL cell1, CELL cell2) checks the seam satisfying the L-seam.

- cell is Pikachu table
- Cell1 and Cell2 are two cells passed in to consider the path

Cells cell1 and cell2 can be connected along an L-line if, at the intersection of their cross-cell (corresponding to the blue or red cell in the table), cross-cell does not exist (isExist=0). Besides, there must exist a straight line going from cross-cell to cell1 and cross-cell to cell2.

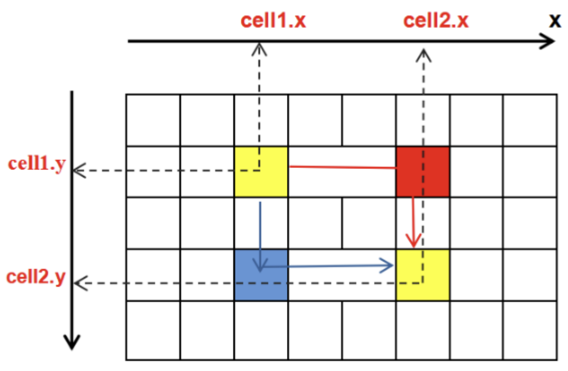


Figure 11: Illustrated

Build the Lcheck function as follows:

- Step 1: Assign cell cross-cell with coordinates (x,y) = (cell1.x, cell2.y) (corresponding to the red box on the picture) or equal to (cell2.x, cell1.y) (corresponding to the blue box on the picture)
- Step 2: Check the LineCheck connection between cell1 and cross-cell, between cell2 and cross-cell. If there is an unsatisfactory path, return false. On the contrary, return true.

3.2.4 Line Z

Consider cell cell3 has coordinates (x0, cell1.y), cell4 has coordinates (x0, cell2.y); To check whether cell1 and cell2 can be connected in a Z-line, we need to use the LineCheck function to check 3 small lines including the line between cell cell1 and cell 3, the line between cell cell3 and cell 4, the line between cell cell4 and cell 2. If one of the 3 lines is not satisfied, it indicates that the Z join is not satisfied

The Zcheck bool function (CELL* cell, CELL cell1, CELL cell2) checks the line to satisfy the Z-line.

- cell is Pikachu table
- Cell1 and Cell2 are two cells passed in to consider the path

Cells cell1 and cell2 can join the letter Z together if their lines fall into one of the following two cases:

- Case 1: Between cell1.x and cell2.x, there exists a point x0 so that from cell cell3 with coordinates (x0, cell1.y) (in non-existent state) to cell cell4 with coordinates (x0, cell2.y) (in non-existent state), there exists a path (LineCheck = 1).
  Then, if there exists a path starting from cell1 to cell3 (x0, cell1.y) and there exists a path starting from cell4 (x0, cell2.y) to cell2, two cells cell1 and cell2 can be connected in a Z-line.

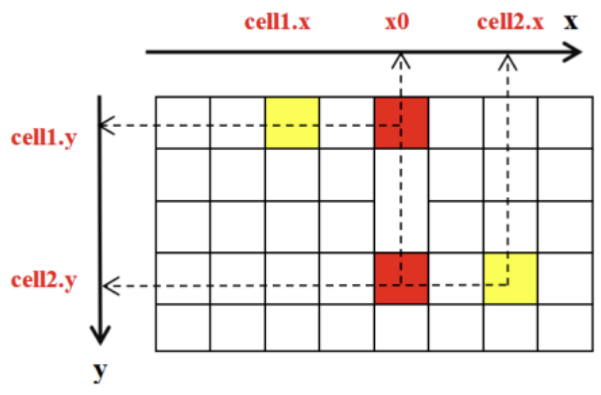


Figure 12: Z horizontal

- Case 2: Between cell1.y and cell2.y, there exists a y0 point so that from cell cell3 (cell1.x, y0) (in non-existent state) to cell cell4 (cell2.x, y0) (in non-existent state), there exists a path (LineCheck = 1).
  Then, if there exists a path starting from cell1 to cell3 (cell1.x, y0) and there exists a path starting from cell4 (cell2.x, y0) to cell2, the two cells cell1 and cell2 can be connected by a Z-line.

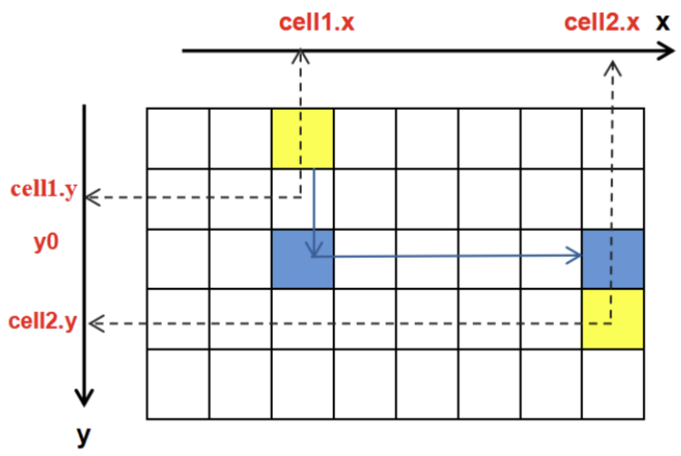


Figure 13: Z vertical

Build the Zcheck function as follows:

- Step 1:
  1. Assign start-x = min(cell1.x, cell2.x), end-x = max(cell1.x, cell2.x)
  2. Assign star-y = min(cell1.y, cell2.y), end-y = max(cell1.y, cell2.y)
- Step 2:
  1. Case 1:
     (a) Looping x from start-(x + 1) to end-(x - 1)
     (b) At each loop, check LineCheck for cell3 and cell4
     (c) If there is x0 to the LineCheck of cell3 and cell4 correctly, return false. On the contrary, implement:
        i. LineCheck of cell1 and cell3
        ii. LineCheck of cell2 and cell4

        If these two conditions are true, return true. On the contrary, return false.
  2. Case 2:
     (a) Looping y from start-(y + 1) to end-(y - 1)
     (b) At each loop, check LineCheck for cell3 and cell4
     (c) If there is y0 to the LineCheck of cell3 and cell4 true, return false. On the contrary, implement:

i. LineCheck of cell1 and cell3

ii. LineCheck of cell2 and cell4

If these two conditions are true, return true. On the contrary, return false.

If in case 1 and in case 2 neither find a satisfactory path, return false.

3.2.5 Line U

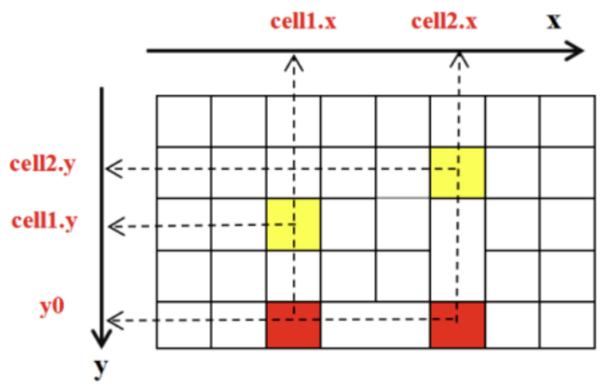


Figure 14: U case

Consider cell cell3 has coordinates (cell1.x, y0) and cell cell4 has coordinates (cell2.x, y0).
To check whether cell1 and cell2 can be connected in a U-line, we need to use the LineCheck function to check the connection between cell3 and cell4 (the junction of the U-leg), Then, check two small lines including the seam between cell1 and cell3, between cell cell4 and cell cell2. If during the test, there is an unsatisfactory line, it indicates that the U junction is not satisfied.

The bool function Ucheck (CELL* cell, CELL cell1, CELL cell2) checks the line to satisfy the U-line.

- cell is Pikachu table
- Cell1 and Cell2 are two cells passed in to consider the path

Two cells cell1 and cell2 can connect the letter U together if their seams belong to one of the following 3 cases.

- Case 1: Two cells are on one of the four sides of the Pikachu table.

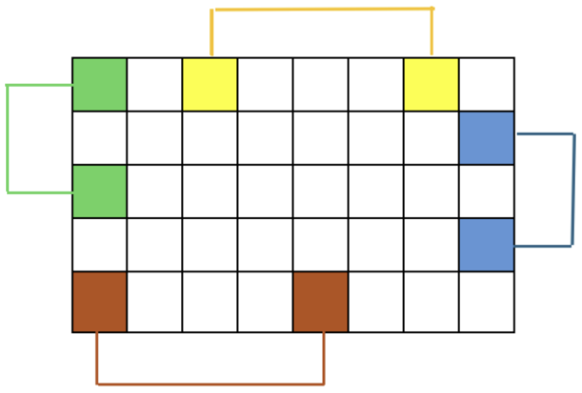


Figure 15: Illustrated

- Case 2: There exist 3 small lines (the line connecting the sides of the letter U)
  1. Consider cell cell3 with coordinates (x, y) = (cell1.x, y0) and cell cell4 with coordinates (x, y) = (cell2.x, y0) or consider cell3 with coordinates (x, y) = (x0, cell1.y) and cell4 with coordinates (x0, y) = (x0, cell2.y)
  2. If there exists a line between cell3 and cell4. Then, if the line between cell1 and cell3, the connection between cell4 and cell2 exists, then there exists a line U between cell1 and cell2.

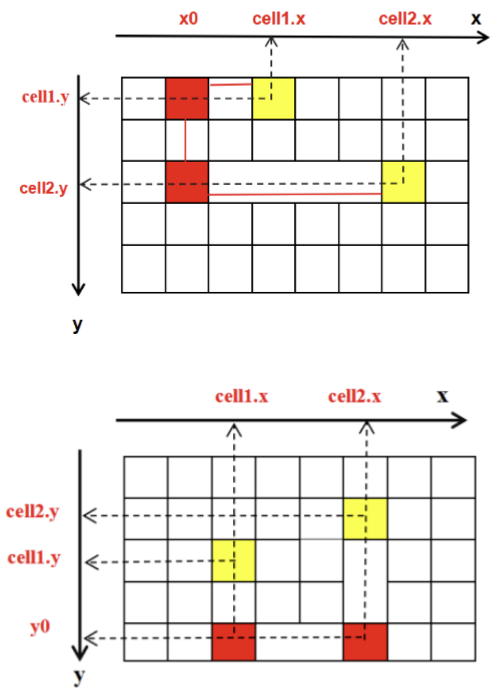


Figure 16: Illustrated

- Case 3: There are only 2 lines (the line connecting the two sides of the U, there is no line connecting the legs of the letter

1. Consider cell cell3 with coordinates (x, y) = (cell1.x, y0) and cell cell4 with coordinates (x, y) = (cell2.x, y0) or consider cell3 with coordinates (x, y) = (x0, cell1.y) and cell4 with coordinates (x0, y) = (x0, cell2.y)
2. If there does not exist a line between cell3 and cell4. Then, if the line between cell1 and cell3, between cell4 and cell2 exists and: The coordinates of x0 are equal to WIDTH - 1 or x0 = 0 (if the letter U is considered vertically) or the coordinates of y0 are equal to HEIGHT - 1 or y0 = 0 (if we are considering the letter U horizontally), then there exists a line U between cell1 and cell2.

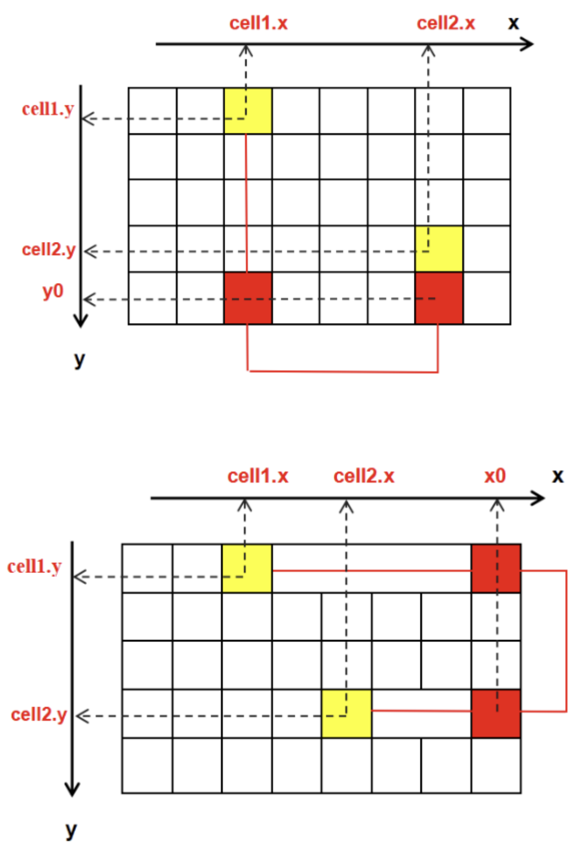


Figure 17: Illustrated

Build the Ucheck function:
We consider cases 1 to 3 listed above respectively so that we can obtain results and review functions quickly.

- Case 1:

1. Step 1: Check if the two selected cells coincide,. If so, return false (To eliminate the case of selecting the cell twice. Then the selected cell will always have x coordinates and y coordinates in cell 2 are the same as x coordinates and y coordinates in cell 1)
2. Step 2: Check whether the coordinates of cell1 and cell2 on the x-axis or on the y-axis are the same (cell1.x = cell2.x or cell1.y = cell2.y). If their x

coordinates are equal and equal to x0 (x0 = 0 or x0 = WIDTH - 1), or if their y coordinates are equal and y0 (y0 = 0 or y0 = HEIGHT - 1), return true.

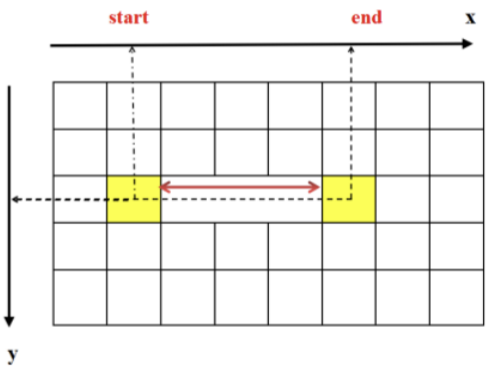
- Case 2
  1. Step 1:
     (a) Assign start-x = min(cell1.x, cell2.x), end-x = max(cell1.x, cell2.x)
     (b) Assign start-y = min(cell1.y, cell2.y), end-y = max(cell1.y, cell2.y)
  2. Step 2:
     (a) Case 2.1: Standing U Consideration
        i. Looping x from 0 to start-(x - 1) and from end-(x + 1)to WIDTH - 1 (x is not considered in paragraphs start-x to end-x)
        ii. At each loop, check if at point x = x0. If a path exists between cell3 and cell4, do it.
           A. Check the path from cell1 to cell3
           B. Check the path from cell3 to cell2

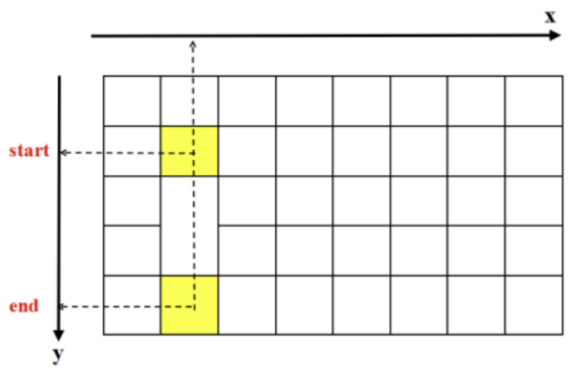
           If two lines being tested exist, return true
     (b) Case 2.2: Horizontal U Consideration
        i. Looping y from 0 to start-(y - 1) and end-(y + 1) to HEIGH - 1 (y is not considered in paragraphs start-y to end-y)
        ii. At each loop, check if at point y = y0. If a path exists between cell3 and cell4, do it.
           A. Check the path from cell1 to cell3
           B. Check the path from cell3 to cell2

           If two lines being tested exist, return true
- Case 3: When no line exists between cell3 and cell4
  1. Step 1: Check x0 and y0. If x0 = 0 or x0 = WIDTH - 1 (in vertical U-direction) or y0 = 0 or y0 = HEIGHT - 1 (in horizontal U-direction)
  2. Step 2:
     (a) Check the path from cell1 to cell3
     (b) Check the path from cell3 to cell2

     If two lines being tested exist, return true
     If the line between cell1 and cell2 does not match one of the above three cases, return false.

3.2.6 Build a LineCheck function





- Step 1: Check if two cells are on the same row (same y coordinates) or on the same column (same x coordinates).
  1. If not, return false
  2. If yes then:
     (a) Case 1: Two cells are on the same horizontal row (same y coordinates in Pikachu table)
        i. Step 1: Assign start = min (cell1.y, cell2.y) , end = max (cell1.y, cell2.y).
        ii. Step 2: Check if there are any cells (isExist) on the seam going from cell1 to cell2. If there are more than 1 cell, return false:
        Looping i from start to end, then check the cell status [pos], with pos = i*WIDTH + cell1,x , corresponding to the position of that cell in the Pikachu table.
     (b) Case 2: Two cells are on the same vertical row (with the same x coordinates in the Pikachu table)
        i. Step 1: Assign start = min (cell1.y, cell2.y) , end = max (cell1.y, cell2.y).
        ii. Step 2: Check if there are any cells (isExist) on the seam going from cell1 to cell2. If there is more than 1 cell, return false: Looping i from start to end, then check the cell status [pos]. For pos = i*WIDTH + cell1,x , which corresponds to the position of that cell in the Pikachu table.
- Step 2:
  1. If, after checking on the seam between cell1 and cell2, no cells still exist, return true.
  2. If checking on the seam between cell1 and cell2 shows that a cell exists, check that the cell matches the position of cell cell1 or cell 2. If so, return true. Otherwise, return false.

3.3 Move cursor in source files

In the "Move cursor" section, there will be 2 functions "DisplayResultChosen" - the function that prints the states when considering a certain pair and the "Movement" function - The function has the purpose of controlling the movement of the cursor on a table of squares, performing actions such as selecting cells, checking and processing when two cells are selected, moving the cursor left, right, up, down on the board, mixing the board, and displaying hints to the player.

- In "DisplayResultChosen" function:

• Step 1: Check the result and take the corresponding action. If the matching result is wrong, mark the 1st and 2nd selection boxes in red, then return white to those 2 boxes, while reducing the life by 1 and the player is deducted 15 points. If the matching results are correct, mark the first and 2nd cells in yellow, then pause for about 0.5 seconds and then delete those 2 cells and update the status of the first and 2nd cells and add 20 points to the player.

• Step 2: Reset the state of the 1st cell and the 2nd cell not selected.

• Step 3: Display player information on the gaming screen and introduce support function keys.

- In "Movement" function:

• Step 1: Read a character from the keyboard.

• Step 2: If the character is ESC, the function assigns a value of 2 to the status variable to exit the game and play an error sound.

• Step 3: If the character is ENTER, the function saves the position of the cursor to the select array, calculates the absolute position of the selected cell in the cell array, and sets a flag for that cell. Then the function checks whether the two selected cells are the same, if any, displays the incorrect result, if not, the function checks whether the two cells can be joined together and displays the corresponding result.

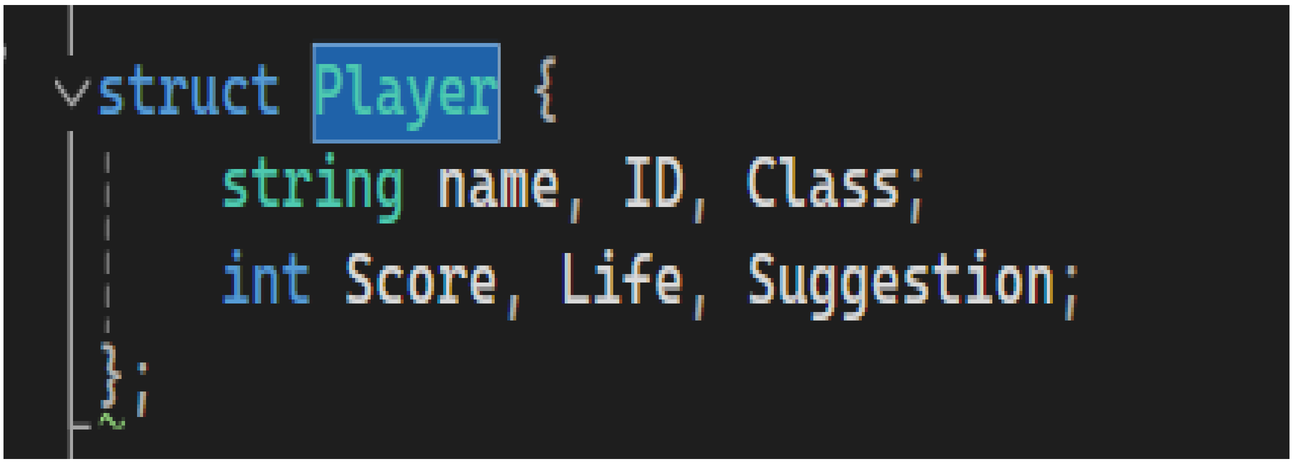
• Step 4:If two cells are selected, the function resets the related variables, and repositions the cursor.

• Step 5:If the character is not ENTER or ESC, the function checks for other characters, including moving the cursor left, right, up, down, mixing tables, and displaying hints.

4 Advanced Features

4.1 Game account and saved stages for each account

First, create a struct to store player information and necessary data such as name, score, ID and classname when the player exits the game before completing it.

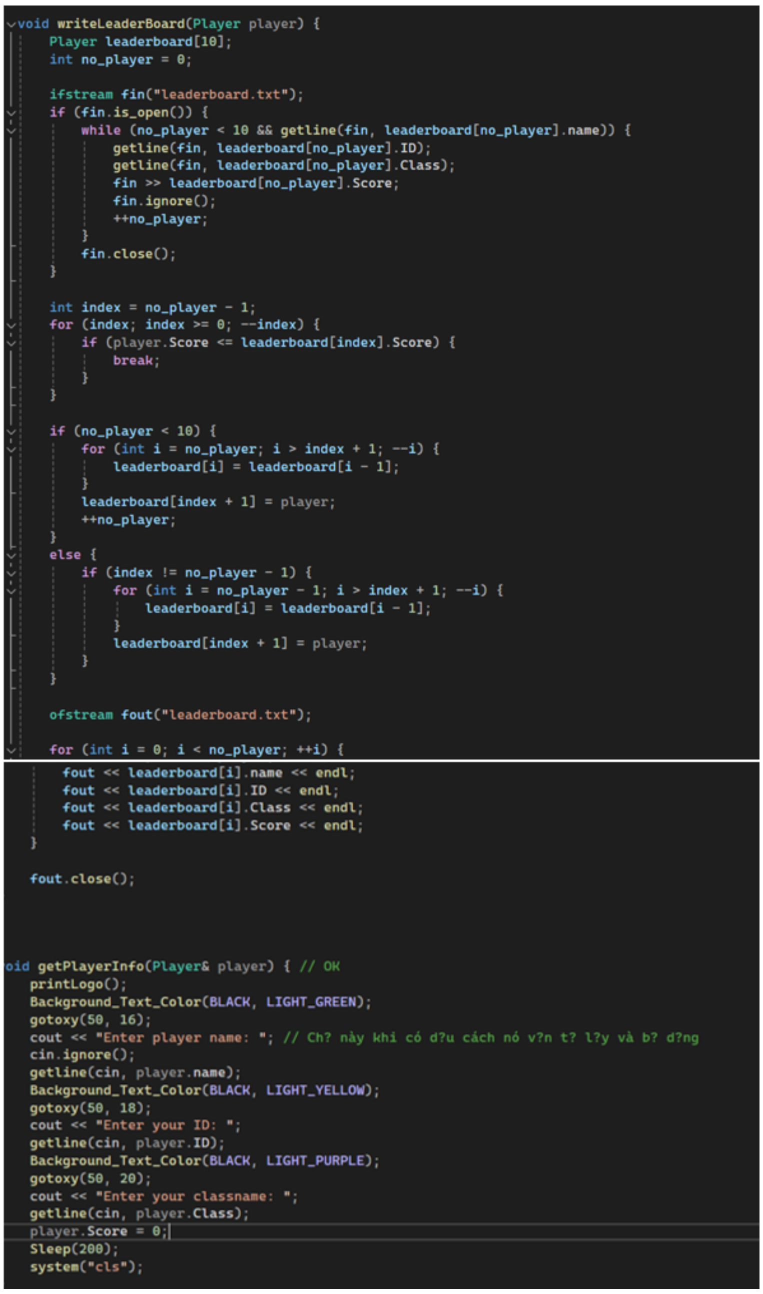


```
struct Player {
    string name, ID, Class;
    int Score, Life, Suggestion;
};
```

Figure 18: Struct savegame

How to update and write player data to a leaderboard in a game.
The "getPlayerInfo" function is responsible for obtaining player information, including name, ID, and class name. Each piece of information is input from the keyboard using the getline(cin, player.name) function and stored in the corresponding variable of the Player object.

The "writeLeaderBoard" function is used to update and write player information to leaderboard.txt file. Initially, it reads information from the leaderboard.txt file and saves to the leaderboard array up to 10 players. It then compares the scores of new players with the scores of the players in the existing leaderboard to determine the proper position to insert new players into the leaderboard. If the leaderboard is not full, new players will be inserted into the appropriate position after comparing with the scores of existing players. If the leaderboard is full, new players will only be inserted into the leaderboard if their score is higher than at least one player in the leaderboard. Finally, the player's information after being updated will be written to the leaderboard.txt file.

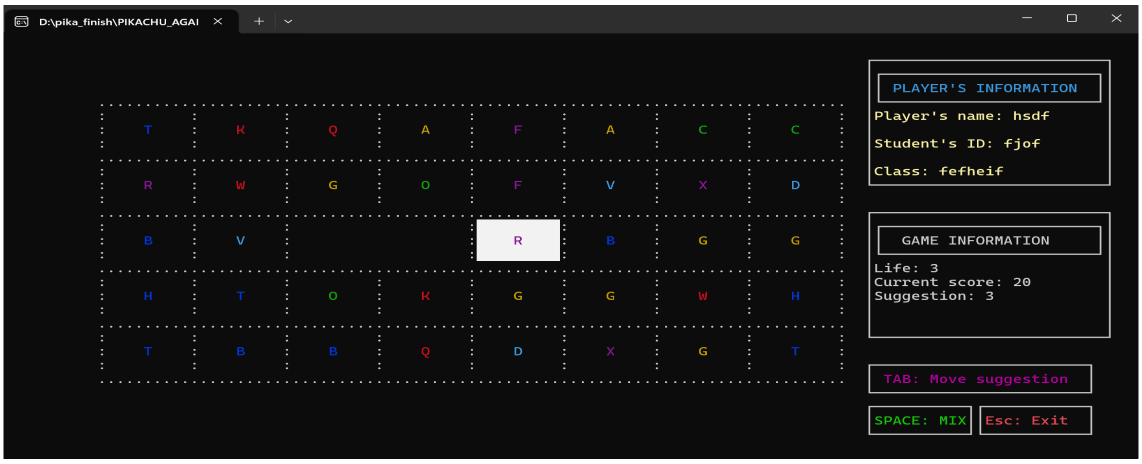


```
void writeLeaderBoard(Player player) {
    Player leaderboard[10];
    int no_player = 0;

    ifstream fin("leaderboard.txt");
    if (fin.is_open()) {
        while (no_player < 10 && getline(fin, leaderboard[no_player].name)) {
            getline(fin, leaderboard[no_player].ID);
            getline(fin, leaderboard[no_player].Class);
            fin >> leaderboard[no_player].Score;
            fin.ignore();
            ++no_player;
        }
        fin.close();
    }

    int index = no_player - 1;
    for (index; index >= 0; --index) {
        if (player.Score <= leaderboard[index].Score) {
            break;
        }
    }

    if (no_player < 10) {
        for (int i = no_player; i > index + 1; --i) {
            leaderboard[i] = leaderboard[i - 1];
        }
        leaderboard[index + 1] = player;
        ++no_player;
    }
    else {
        if (index != no_player - 1) {
            for (int i = no_player - 1; i > index + 1; --i) {
                leaderboard[i] = leaderboard[i - 1];
            }
            leaderboard[index + 1] = player;
        }
    }

    ofstream fout("leaderboard.txt");

    for (int i = 0; i < no_player; ++i) {
      fout << leaderboard[i].name << endl;
      fout << leaderboard[i].ID << endl;
      fout << leaderboard[i].Class << endl;
      fout << leaderboard[i].Score << endl;
   }

   fout.close();



oid getPlayerInfo(Player& player) { // OK
   printLogo();
   Background_Text_Color(BLACK, LIGHT_GREEN);
   gotoxy(50, 16);
   cout << "Enter player name: "; // Ch? này khi có d?u cách nó v?n t? l?y và b? d?ng
   cin.ignore();
   getline(cin, player.name);
   Background_Text_Color(BLACK, LIGHT_YELLOW);
   gotoxy(50, 18);
   cout << "Enter your ID: ";
   getline(cin, player.ID);
   Background_Text_Color(BLACK, LIGHT_PURPLE);
   gotoxy(50, 20);
   cout << "Enter your classname: ";
   getline(cin, player.Class);
   player.Score = 0;
   Sleep(200);
   system("cls");
```

Figure 19: Binary savegame

4.2 Color effect

We can use the SetConsoleTextAttribute(GetStdHandle(STDOUTPUTHANDLE), a) function with an integer a to determine the desired color and combine it with the gotoxy(int x, int y) function to print characters with the desired color.

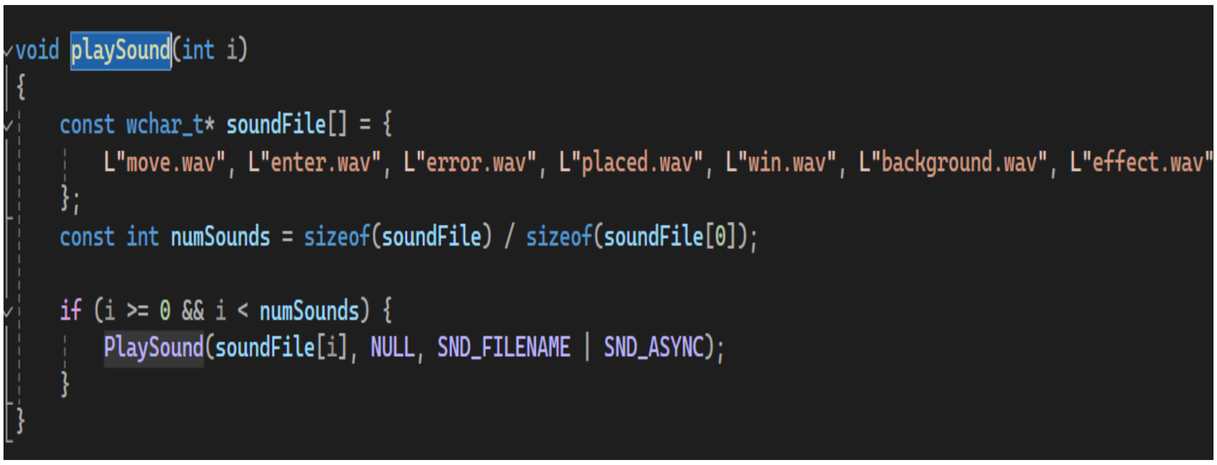


```
D:\pika_finish\PIKACHU_AGAI

  T   K   Q   A   F   A   C   C        PLAYER'S INFORMATION
                                       Player's name: hsdf
  R   W   G   O   F   V   X   D        Student's ID: fjof
                                       Class: fefheif
  B   V           R   B   G   G
                                       GAME INFORMATION
  H   T   O   K   G   G   W   H        Life: 3
                                       Current score: 20
  T   B   B   Q   D   X   G   T        Suggestion: 3

                                       TAB: Move suggestion
                                       SPACE: MIX   Esc: Exit
```

Figure 20: Color effect

4.3 Sound effect

• We use PlaySound to add the sound in ".wav" file so when we move in the gameplay, it will include the sound.
• We also use mciSendString to add the sound when the matching in the gameplay is correct or incorrect, add a background music that plays continuously as the user interacts with the system.



```
void playSound(int i)
{
    const wchar_t* soundFile[] = {
        L"move.wav", L"enter.wav", L"error.wav", L"placed.wav", L"win.wav", L"background.wav", L"effect.wav"
    };
    const int numSounds = sizeof(soundFile) / sizeof(soundFile[0]);

    if (i >= 0 && i < numSounds) {
        PlaySound(soundFile[i], NULL, SND_FILENAME | SND_ASYNC);
    }
}
```

Figure 21: Sound effect

4.4 Visual effect

Interface design: Use pre-built functions to draw rectangular frames and display user information.

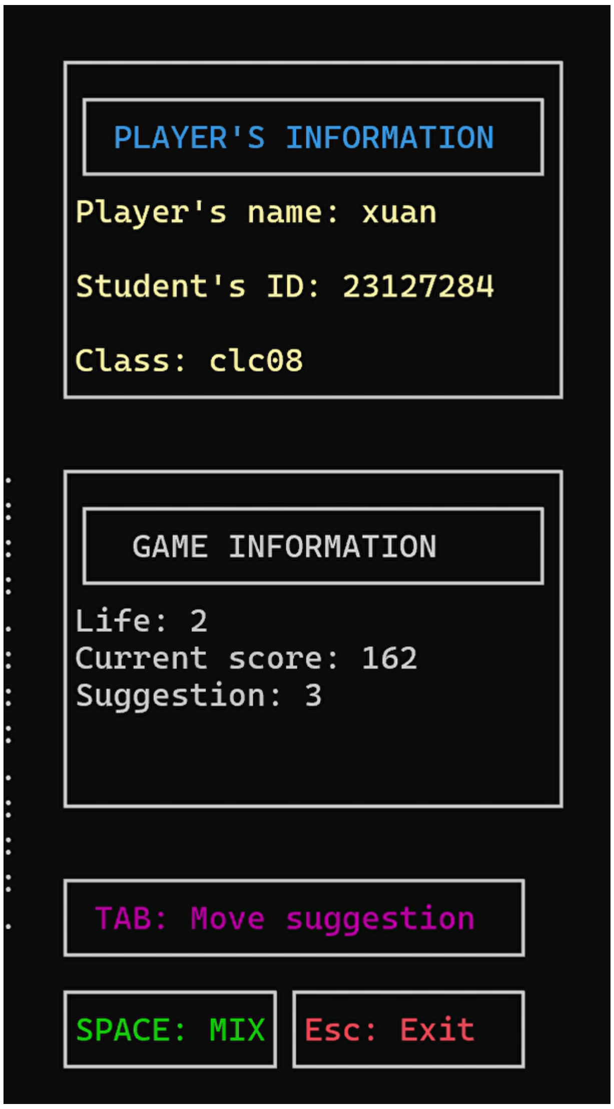
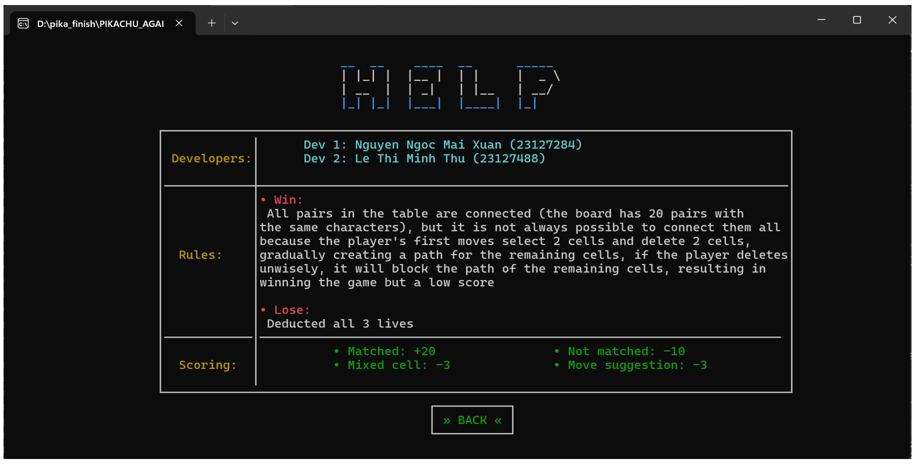

```
PLAYER'S INFORMATION
Player's name: xuan
Student's ID: 23127284
Class: clc08

GAME INFORMATION
Life: 2
Current score: 162
Suggestion: 3

TAB: Move suggestion
SPACE: MIX   Esc: Exit
```

Figure 22: visual effect

4.5 Help screen

The "helpScreen" function is written to display a help screen for the player in the game. The main purpose of this function is to introduce the rules of the game and how to calculate points in the game, along with information about the developers of the game.
This function performs the following steps:
• Disable the blinking cursor.
• Clear the screen using the system ("cls") command.
• Draw the title of the game in a fixed position on the screen.
• Draw a frame containing help information with lines and items of specific information about the rules of the game and scoring.
• Displays the rules of the game and how points are scored in the game
• Display information about developers
• Display the "BACK" button so that the player can return to the previous screen.
• Wait for the user to press the key and process the event if the user presses the Enter

key, such as playing a sound or performing an action.
• Clear the screen and activate the cursor to blink again.

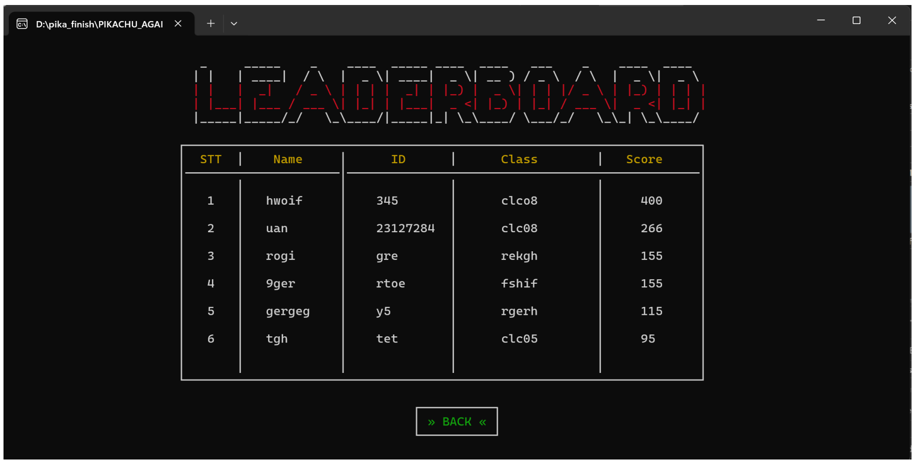


```
Developers:   Dev 1: Nguyen Ngoc Mai Xuan (23127284)
              Dev 2: Le Thi Minh Thu (23127488)

Rules:        • Win:
               All pairs in the table are connected (the board has 20 pairs with
              the same characters), but it is not always possible to connect them all
              because the player's first moves select 2 cells and delete 2 cells,
              gradually creating a path for the remaining cells, if the player deletes
              unwisely, it will block the path of the remaining cells, resulting in
              winning the game but a low score

              • Lose:
               Deducted all 3 lives

Scoring:          • Matched: +20                 • Not matched: -10
                  • Mixed cell: -3               • Move suggestion: -3
```

Figure 23: help screen

4.6 Leaderboard

• Designing the leaderboard: the leaderboard consists of information such as player name, score,and completion time of each level.
• To draw a leaderboard, we need to build functions to draw rectangular frames. Therefore, we will reuse the functions used in the "Visual Effects" section and customize some parameters
such as coordinates, length, and height of the rectangular frame to draw a leaderboard with a suitable size.
• The display information of the leaderboard will be stored in a file binary, which will be presented in the binary save file section.

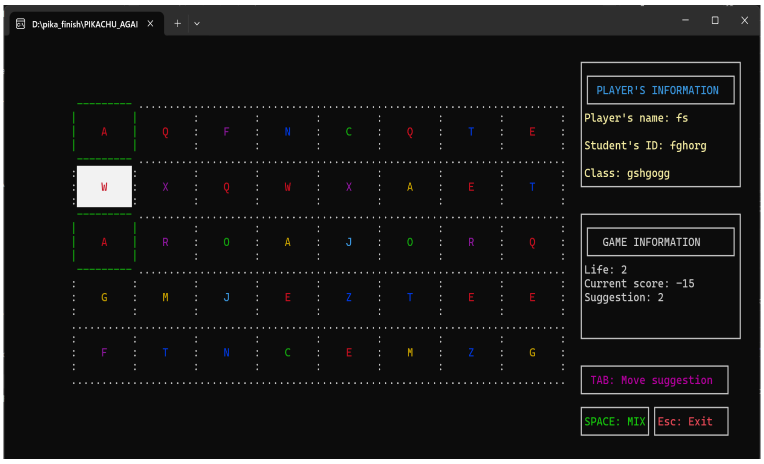


STT	Name	ID	Class	Score
1	hwoif	345	clco8	400
2	uan	23127284	clc08	266
3	rogi	gre	rekgh	155
4	9ger	rtoe	fshif	155
5	gergeg	y5	rgerh	115
6	tgh	tet	clc05	95

Figure 24: LeaderBoard

4.7 Move suggestion

Idea: Use the Check-Exist-EqualPair function. If you encounter a pair of umbrellas, satisfy the seam.
Implement: Fill in the color around the cell border to make it stand out on the Pikachu board. Sleep (1500) so that the player can observe. Then, stop the loop



```
 A   Q   F   N   C   Q   T   E
 W   X   Q   W   X   A   E   T
 A   R   O   A   J   O   R   Q
 G   M   J   E   Z   T   E   E
 F   T   N   C   E   M   Z   G

PLAYER'S INFORMATION
Player's name: fs
Student's ID: fghorg
Class: gshgogg

GAME INFORMATION
Life: 2
Current score: -15
Suggestion: 2

TAB: Move suggestion
SPACE: MIX  Esc: Exit
```

Figure 25: Move suggest

4.8 A comparison of the complexity of implementing functions with increased difficulty between using Pointers and LinkedLists.

Pointers serve as variables that hold memory addresses, enabling indirect access to variable or data structure values.Conversely, a linked list comprises nodes referencing subsequent nodes, commonly utilized for managing dynamic collections like game objects. While pointers are generally considered simpler due to their fundamental role in programming languages and their use in implementing complex data structures, they necessitate careful memory management to prevent issues like memory leaks. On the other hand, linked lists can be more intricate to implement but offer simplification for certain tasks. Ultimately, the choice between pointers and linked lists depends on the developer's preferences, with pointers being simpler to use but requiring meticulous memory management, while linked lists may involve more complexity in implementation but can streamline certain operations.

5 References

https://www.youtube.com/watch?v=I2FbZyldJuwt=1s: Code reference

https://github.com/Louis2602/Pikachu-Game: Code reference

https://www.asciiart.eu/text-to-ascii-art : convert photos to ASCII code

https://www.youtube.com/watch?v=9gWIbhOudEkt=3s : C++ graphical functions

https://codelearn.io/sharing/windowsh-va-ham-dinh-dang-console-p1 : console screen

https://www.iostream.co/article/do-hoa-tren-cua-so-dong-lenh-console-graphics-clmLe1 : Background color for text

https://nguyenvanhieu.vn/bat-su-kien-ban-phim-trong-c-cpp/ : Catch key events

https://estools.blogspot.com/2011/04/so-sanh-swich-case-va-if-else.html : The difference between switch case and if else, avoid causing errors when making keys move up and down in Pikachu panel

6 Demo video

https://youtu.be/oTozxItqMY0?si=JyEOO3WXwKsg5uZy