

VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

THE MATCHING GAME REPORT

Project: The Matching Game

Course: Programming Techniques

Class: 22CLC03

Student:

Hoang Bao Khanh (22127183)
Tran Huy Khanh (22127191)

Lecturers:

M.S. Bui Huy Thong
PhD. Nguyen Ngoc Thao



Contents

| | | |
|----------|---|-----------|
| 1 | Special thanks | 3 |
| 2 | Project Overview | 4 |
| 3 | Standard Features | 5 |
| 3.1 | Game starting | 5 |
| 3.2 | Matching | 6 |
| 3.2.1 | I matching | 6 |
| 3.2.2 | L matching | 7 |
| 3.2.3 | Z matching | 8 |
| 3.2.4 | U matching | 8 |
| 3.3 | Game finish verify | 10 |
| 4 | Technical Requirements | 12 |
| 4.1 | Board Definition | 12 |
| 4.2 | Account Definition | 12 |
| 4.3 | Binary save file | 13 |
| 5 | Advanced Features | 14 |
| 5.1 | Color Effects | 14 |
| 5.2 | Sound Effects | 14 |
| 5.3 | Visual Effects | 14 |
| 5.4 | Background | 16 |
| 5.5 | Leaderboard | 17 |
| 5.6 | Move Suggestion | 18 |
| 6 | Extra Advanced Features | 19 |
| 6.1 | Algorithms using 2D-Pointer array | 19 |
| 6.2 | Algorithms using Singly Linkedlist | 19 |
| 6.3 | Comparisons the 2D-Pointer Array and LinkedList | 20 |
| 7 | Other features | 24 |
| 7.1 | Sign up and login | 24 |
| 7.2 | Shuffle | 24 |
| 7.3 | Custom mode | 25 |

| | | |
|----------|--|-----------|
| 8 | Game Tutorials | 26 |
| 8.1 | Log in and Sign up | 26 |
| 8.1.1 | Log in | 26 |
| 8.1.2 | Sign up | 27 |
| 8.1.3 | Play as Guest | 28 |
| 8.2 | Menu Screen | 29 |
| 8.2.1 | New Game | 29 |
| 8.2.2 | Help | 30 |
| 8.2.3 | Leaderboard | 30 |
| 8.2.4 | Exit | 31 |
| 8.3 | Explain the necessity of all file .h and .cpp in the project | 32 |
| 9 | References | 34 |

1 Special thanks

Initially, we would like to thank to our teachers, Mr. Bui Huy Thong and Mrs. Nguyen Ngoc Thao for spending time reading my report. Besides, we are particularly grateful to teachers for answering our questions and guiding us to start and finish this report. We are also thankful for your patience that you have shown during the project.

During three weeks doing The Matching Game project, I and my teammate, Tran Huy Khanh both learn not only more knowledge, but also how to be responsible for our tasks and how to do in teamwork efficiently.

Finally, we are honored to have the opportunity to be guided by you in this course. Thank you again and we wish all the best come to you.

2 Project Overview

The Matching Game, commonly known as Pikachu Puzzle Game, includes a board of multiple cell. Each of cells presents a figure. The player have to find and match 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 of pairs are found. The Image 1 below illustrates generally the Pikachu Puzzle Game.



Image 1: The Pikachu Puzzle Game¹

¹[Google.com](https://www.google.com)

3 Standard Features

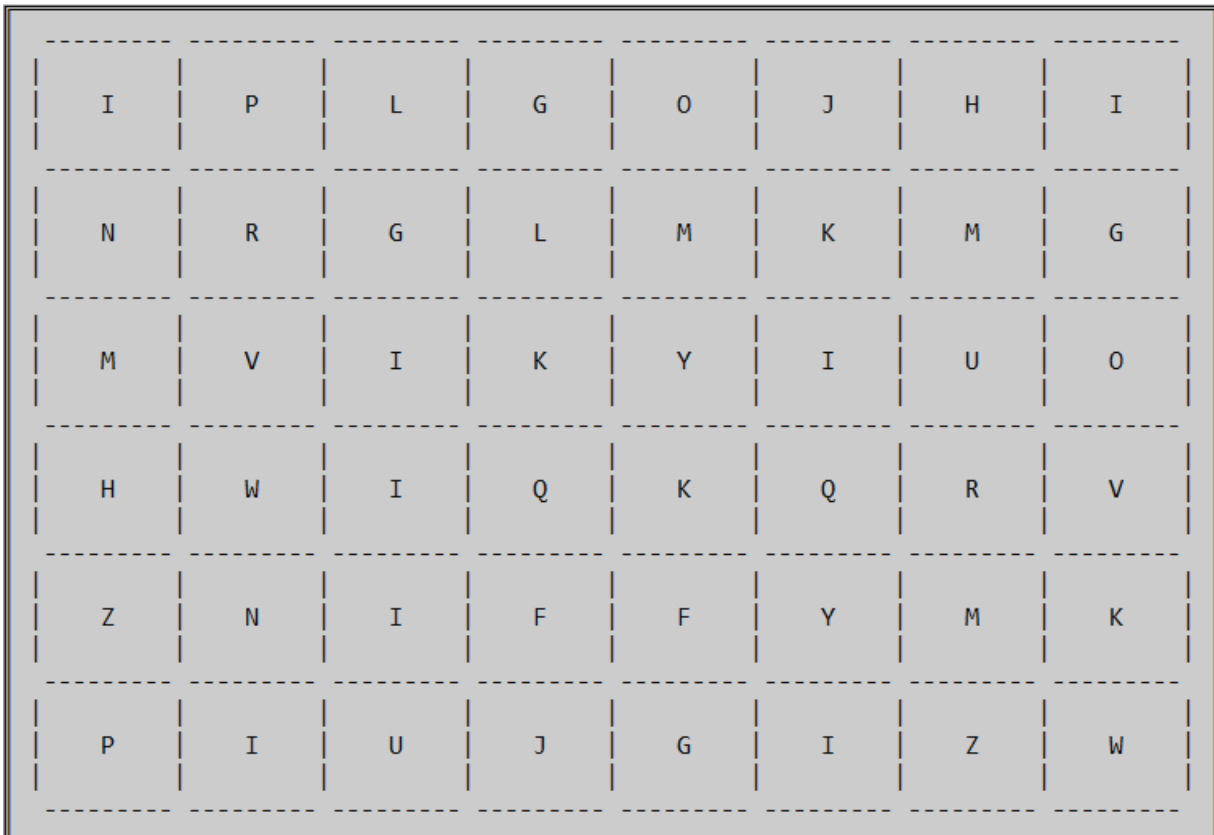
Standard Features are the basic and essential features of the project. The project must have all of this to make the game playable. In this statement, there are six main features: Game starting, I matching, L matching, U matching, Z matching and Game finish verification.

3.1 Game starting

In the Game Starting feature, we create a board with the 6x8 size, which means that there are 8 cells in the width of the board and 6 cells in the height. Therefore, there are 48 cells in total on the board.

In each cell, we use the English Alphabet as the figure. The number of occurrences for each character is always even, and the player needs to find and match a pair of the same character in four particular patterns: I, L, U, and Z matching.

Regarding a board design, we use white color for the background and black color for illustrating figures.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| I | P | L | G | O | J | H | I |
| N | R | G | L | M | K | M | G |
| M | V | I | K | Y | I | U | O |
| H | W | I | Q | K | Q | R | V |
| Z | N | I | F | F | Y | M | K |
| P | I | U | J | G | I | Z | W |

Image 2: The 6x8 board with 48 figures

3.2 Matching

When two cells are chosen, if they have the same figure and connect each other in four main patterns, the Matching feature is used to describe clearly how they can be matched.

For the matching feature, we use three main stages: check, draw, and delete. In our program, we initialize the Check function for each case of the I, L, Z, and U matching pattern in turn. Then, after identifying two cells matched, we draw a line between them to show how it connects to each other. For a few seconds, two cells and the line are deleted.

In this feature, we discuss particularly the check algorithms for each case.

3.2.1 I matching

A pair of cells connected with I matching means that the connection line between two cells can be in the form of a vertical or horizontal of the I character.

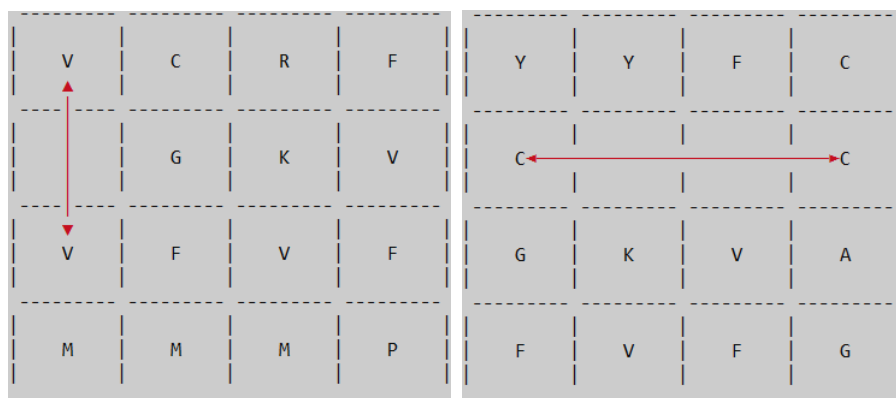


Image 4: The shapes of I matching: vertical and horizontal respectively

In the process of identifying the I matching feature, firstly, the "checkI" function is used to check two cells by creating the checkLineX function to check the empty cells between a common column pair of cells and the checkLineY function for a common row pair of cells. And then the I matching pair of cells are connected by the red line as the Image 4.

In my source code, we declare "checkI" function by pair type in C++. The purpose of using pair is that we want to return this function to a pair of x and y values. The function will return (-2, y) if a pair of cells have the same row and returns (x, -2) if a pair of cells have the same column. This pair function returns (-2, -2) if two cells are not connected with I matching.

The reason we use -2 but not -1 will be mentioned in chechU below.

3.2.2 L matching

In the L matching feature, there are four ways to match a pair of cells with an L shape.

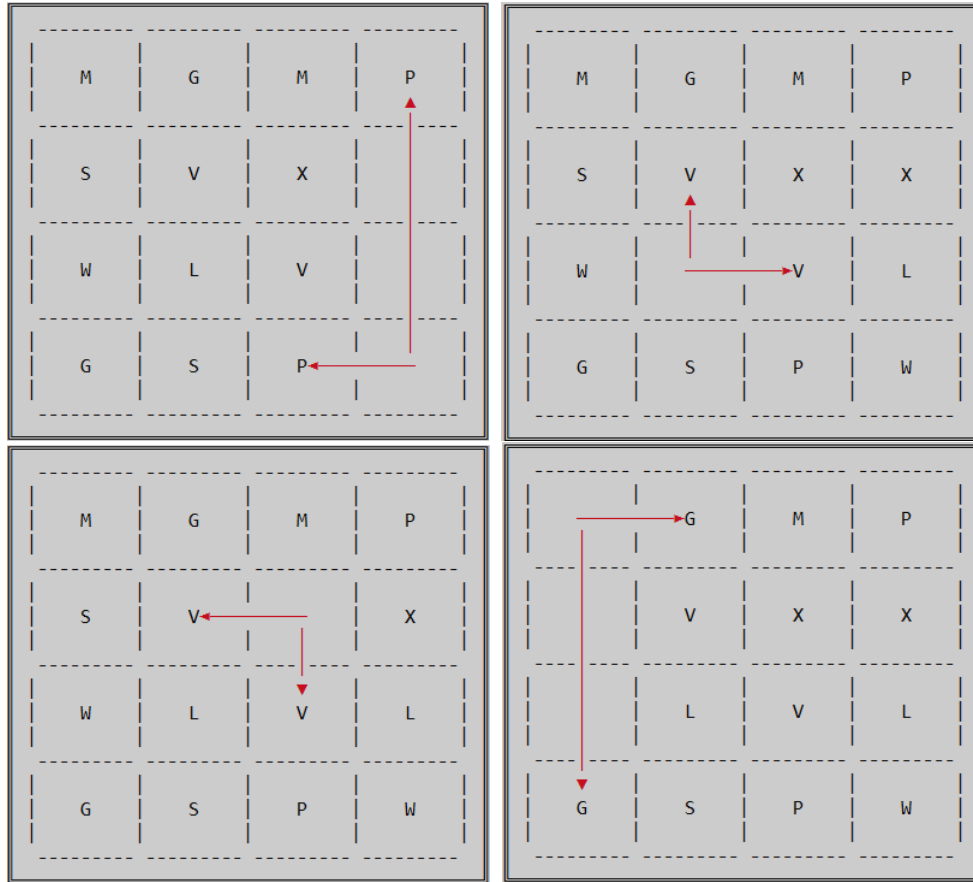


Image 6: Illustration for four types of L matching

With regard to the checkL algorithms, we find which cell has the minimum and maximum value of x ($\min X$ and $\max X$), and minimum and maximum value of y ($\min Y$ and $\max Y$). Thanks to identifying cells like that, we guarantee that the program is able to check all the cases of L matching. Then, we use the four If functions to check each case and return the appropriate pair of values. One more thing to be noted is the output of checkL function includes 5 cases, 1 is the row x of ($\min X$ or $\max X$) a pair $\langle x, -2 \rangle$ will be returned if connectable, 2 is column x of ($\min X$ or $\max X$) and pair $\langle -2, y \rangle$ will be returned if connectable, 3 and 4 is the same with $\min Y$ and $\max Y$. 5 indicates that the pair can't be matched and a pair $\langle -2, -2 \rangle$ will return.

3.2.3 Z matching

Similar to I and L matching, the Pair function is also used to check Z matching. There are four ways to connect two main cells in the Z feature.

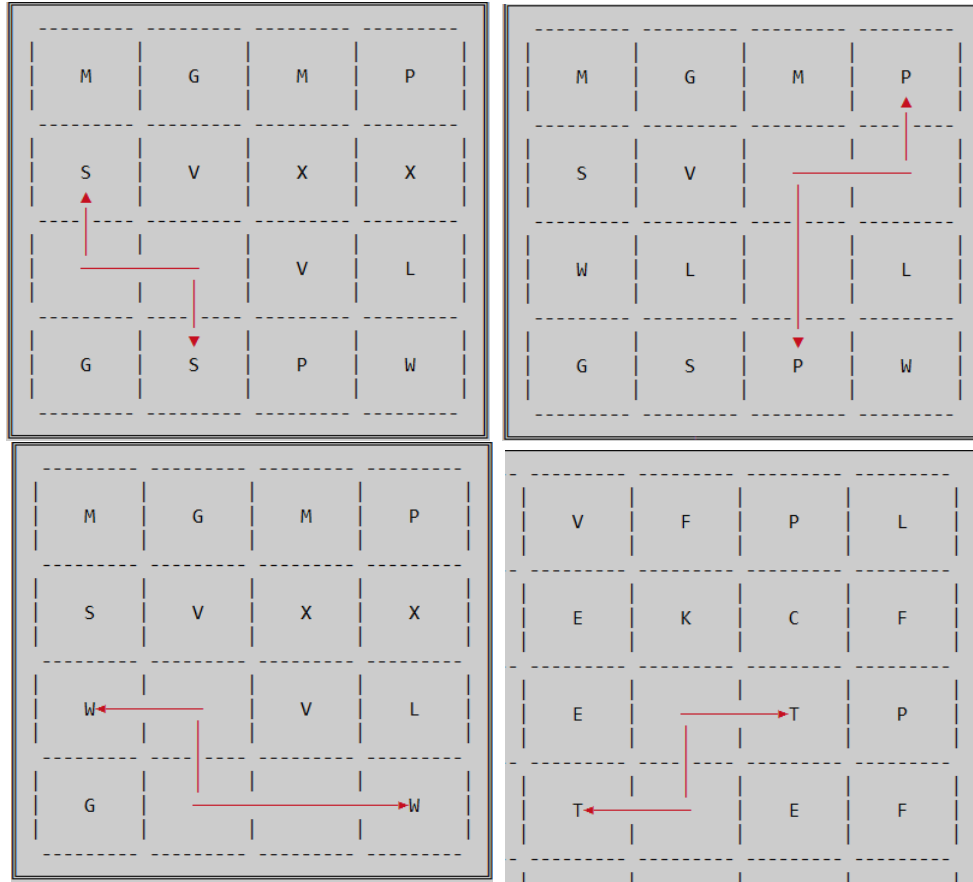


Image 7: Four types of Z matching

Initially, we find out which cell has the minimum and maximum of x and y values to make sure that we check all of the cases, and then check for each feature of Z. Continuously, we use For loop from the minimum value to the maximum value of x and y to check the empty cells between a pair of cells. Considering the output, although there is 4 way to connect them, the output is only 2 like the Imatching if we exclude the unmatchable cases. It will be $\langle x, -2 \rangle$ if a row x between 2 cells is found connectable or $\langle -2, y \rangle$ if a column y is found.

3.2.4 U matching

The U-matching case is not similar to the three cases mentioned above. Let's assume that the pairs of cells being considered are 2 out of 4 vertices of a rectangle. We can easily see that the connections in the I, L, and Z cases are all inside the

rectangle. However, in the U case, the paths have to go outside the rectangle, therefore, we have created two additional check functions, `checkMoreLineX`, and `checkMoreLineY`, to verify the paths outside the rectangle. 1 more thing is that the u match can reach even outside of the board.

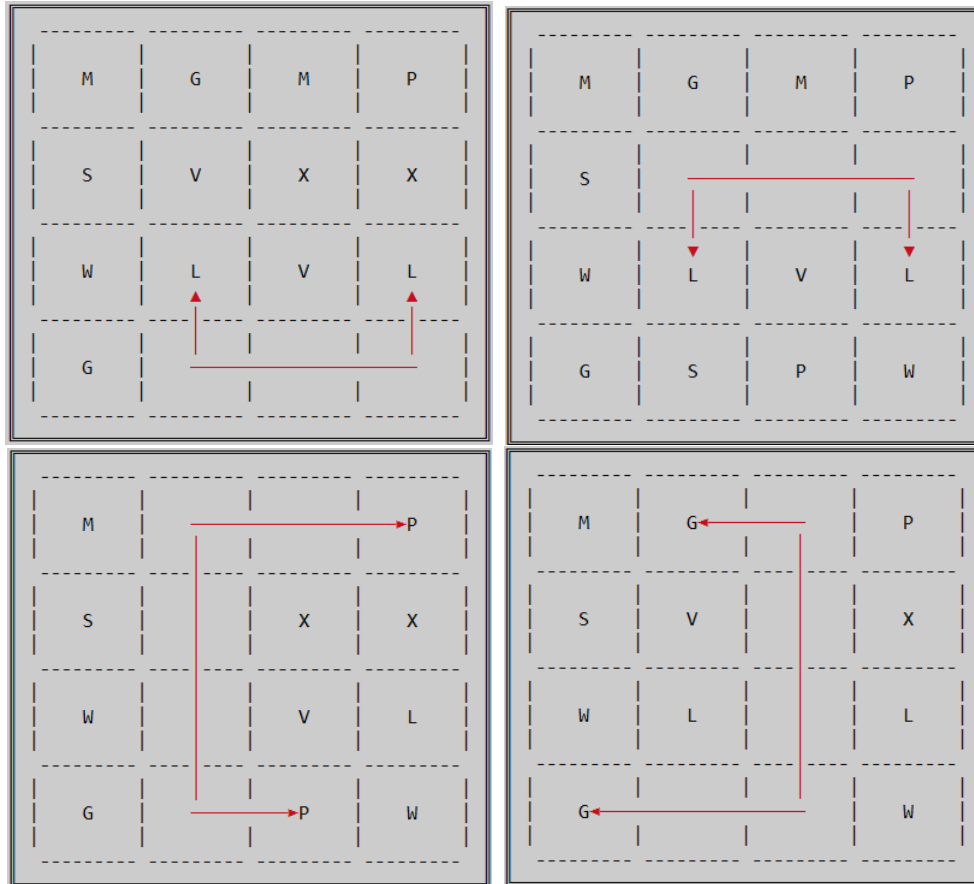


Image 8: Four types of U matching

To create the `checkU` function, first, we create an extended board including a ring of '0' and all information from the original board at the position $(i + 1, j + 1)$ so the extended board has 2 more rows and columns. Then we check outside of the rectangle until it still '0' because '0' indicate an empty cell and because of the ring, a line outside of the original board can be found. The output, in this case, includes 4 cases excluding the unmatchable case. if we use the `minX`, `maxX`, `minY`, and `maxY` again like the `Lmatching`, `minX` is the cell above `maxX`, while `minY` is in the left side of `maxY`, the output will be $\langle \text{minX} - i - 1, -2 \rangle$, $\langle \text{maxX} + i - 1, -2 \rangle$, $\langle -2, \text{minY} - j - 1 \rangle$ or $\langle -2, \text{maxY} + j - 1 \rangle$ with i and j are the count variable we let it run outside of the rectangle. Here you can see why it is not -1 but -2 as the default value. Yes, the `checkU` function can return $x = -1$ or $x = \text{height}$ and $y = -1$ or $y = \text{width}$ to indicate the matching can be done

outside of the board. This later will help us draw and erase the line outside of the board.

3.3 Game finish verify

In terms of the game finishing, we create the algorithms to calculate the number of cells left and the valid pairs left, and it updates subsequently while the player is playing the game.

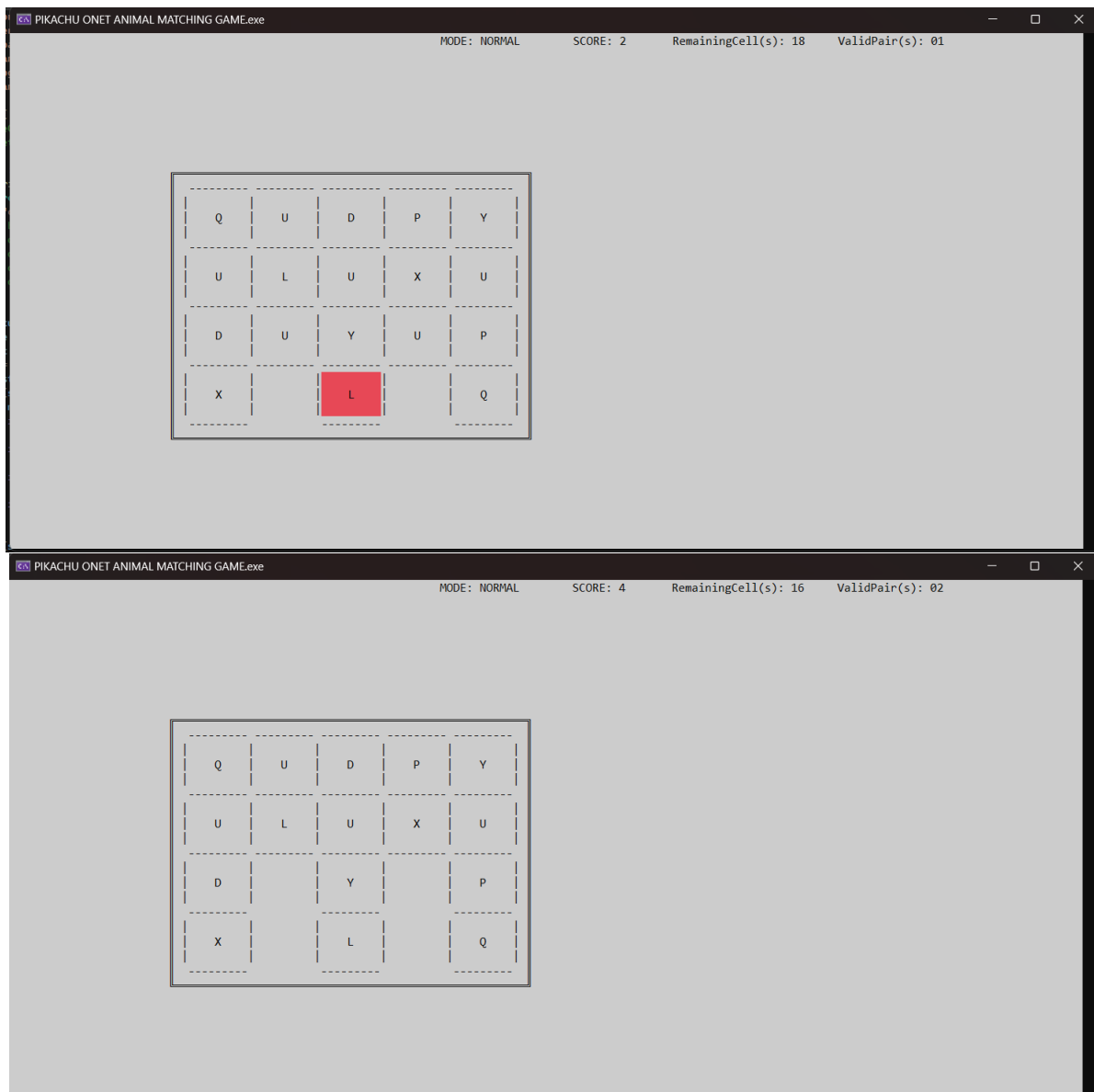


Image 9: The remaining cells and valid cells are updated after each turn

Regarding the algorithm for counting the remaining cells, we use a For loop to count the number of features in each cell. After removing the connected cells, the

count variable will be reduced by 2 after each move, and the game will end when the number of remaining cells equals 0.

As for the algorithm for counting the valid cells, first, we scan the board to find a valid cell, then we use the check2cells function to check that one valid cell with the remaining cells if a match can be made the validpairs variable increase by 1. Then we find the next valid cells and check with the remaining cell excluding the already check one. The way we use to scan the board is to use division and modulus, we let i run from 0 to height times width, i divided by width and i mod width to locate the cell position in a 2d array. Then we let j equal to i plus 1 run to the end of the 2d array by using the same method. Then we compare them and count validpairs. A valid pair must be connectable and identical.

4 Technical Requirements

4.1 Board Definition

In terms of the algorithms to design a board, firstly, we create a structure for cells in a board. In the cell structure, we define `x`, `y` as the positions of cells in the 2D Array; `x_console` and `y_console` as the positions of cells in console screen. Moreover, there are "pokemon" variable to store the feature of cell, draw cell and delete cell function.

The board structure is constructed by using four main distance variables:

- top: the distance of the cursor from the top edge of console screen after pressing new line character
- left: the distance of the cursor from the left edge of console screen after pressing space bar
- height: the height of the board (number of row)
- width: the width of a board (number of column)

Besides, draw and erase functions are also included in this structure. We draw a cell and then multiply it according to the given width and height. In order to set up the character inside, we use random algorithm from A to Z and fill in each cell with one character afterward.

4.2 Account Definition

To identify the player, we construct Account structure to store the player information. Initially, we create Num variable to save the account number, we assume that '0' is a guest account. Additionally, there are string Username and string Password to store the Log in information. Username is also used as player name, so identical Username is not available. These information will be stored in a text file and will be opened again when the user log in.

Moreover, struct Account has more members to store the information when the player finish game. The 'time' variable is used to calculate how long the player finish game as second, 'score' variable to store the point the player get, and 'mode' to identify which game mode the player have played. These information will be stored in another text file to display on the leaderboard if the player performance is good. There is variable 'isFinished' to mark if the player complete the game, this variable will only be used in future patch.

4.3 Binary save file

The Binary save file is used to write and read the information of players. After the player finishes the game, the information of players will be written to the binary file called "leaderboard.bin". To be more specific, because our struct uses the username as a string class object, so first, we convert it to `char*` with the `c-str` function and store 20 bytes in the binary file, one note is the username won't exceed 15 chars so 16 is enough though. There is also 1 other fact that we don't store the struct object but the struct variable separately because of the username.

5 Advanced Features

5.1 Color Effects

Regarding color, we rely on the color table in the windows.h library in C++. The two main colors we have chosen for our program are white (the dominant color for the background) and black (the dominant color to describe features such as board lines, filling characters, cell borders, lines, as well as notes). In addition, we also use other colors to highlight other features.

5.2 Sound Effects

Regarding the sound, we use 2 main tracks for the game. These tracks are in the .wav file format, and we obtained them from the website talkingwav.com. In our program, we identify them as MENU_SOUND (or alternatively MOVE_SOUND) and WIN_SOUND.

For the MENU_SOUND (MOVE_SOUND) tracks, we set up the sound when the player moves the arrow keys on the keyboard. As soon as the movement is made, the program will play this type of track. This sound is used when moving the arrows in the game menu interface or while playing in the board game.

As for the WIN_SOUND track, we use it as soon as the program is initialized and when the Animation feature is displayed. In addition, we also use this track after the player has completed the game and immediately when the results screen appears. The ERROR_SOUND is used where the player gives a wrong matching.

5.3 Visual Effects

Regarding Visual Effects, we have designed effects for our program such as distinguishing the selected cells from the remaining cells. If the cursor points to the position of the selected cell, the entire cell will be colored light red compared to the remaining cells.

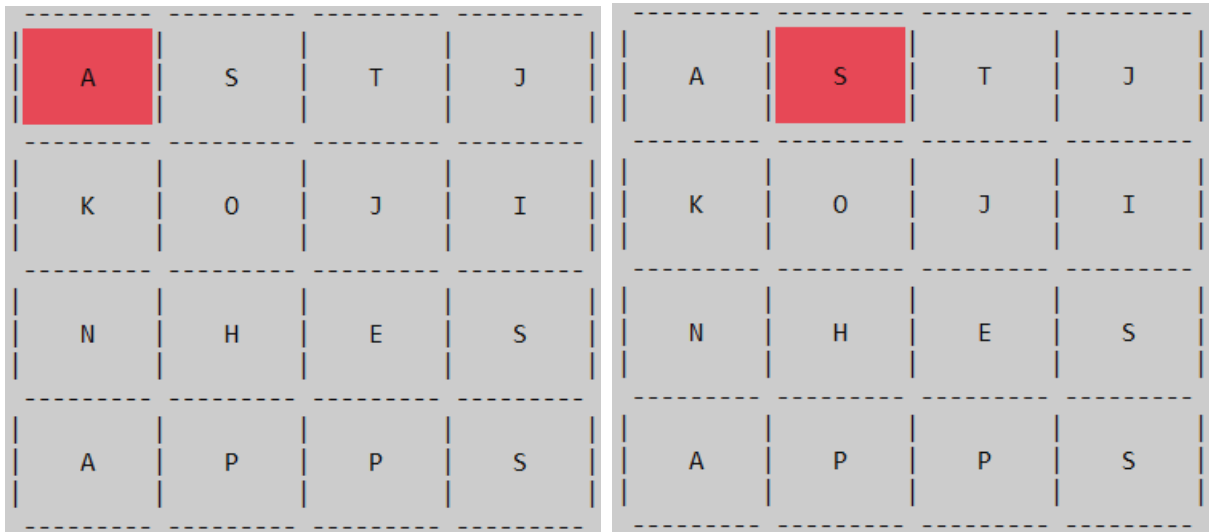


Image 10: The selected cells are colored red

In addition, in terms of the locked cell, it will be colored yellow as in the image below. And after matching a pair of cells, these cells are colored green, and then the red line will be drawn to illustrate how it can be matched and then two cells disappear later.

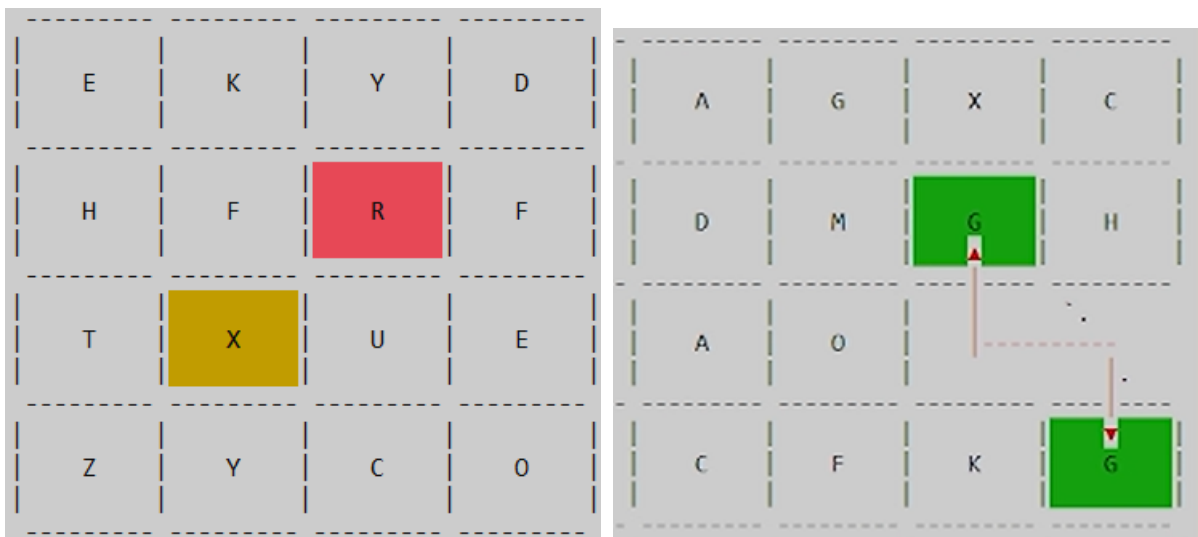


Image 11: The locked cells and a matched pair of cells are colored

Moreover, when the player presses the "A" button on the keyboard for requesting a move suggestion, a pair of cells will appear with a purple color and then they disappear after that.

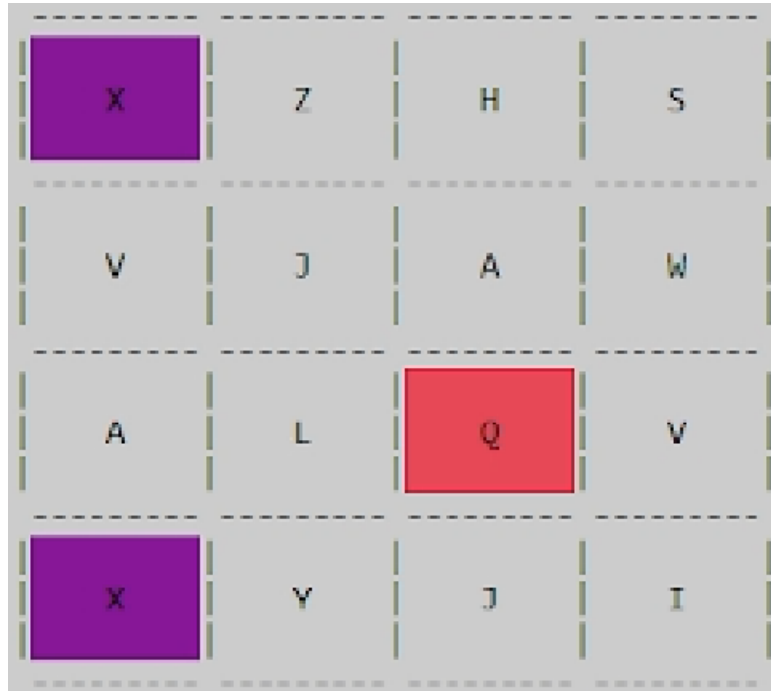


Image 12: A pair of cells when the player uses Move Suggestion

5.4 Background

As for the background, first thing is that we take it from asciart.eu, and then we modify it so it suits the board size. To be more specific, the formula we use when we modify the background is board height times 4 plus 1 and board width times 10 plus 1. There is that number because every cell has the size of 11 spaces width-wise and 5 new line characters height-wise. And the way we display the board in the console is to delete the cell including the borderline of each cell every time a correct matching is made, and then we draw the current board again which means if the adjacent cell of the deleted cell is still here that border is drawn again. And because the border of each cell overlaps the other, there is that formula. The second thing about displaying the background is that for every cell being deleted, the corresponding part of the background will be displayed so this avoids the problem which is to display the full background in every loop and overlapping it with the board. But there is 1 last problem every time the loop is made the background at the deleted cell will be redrawn which includes the border of the adjacent valid cell. Then that valid cell is redrawn by the board so there is a lag at the borderline of every loop.

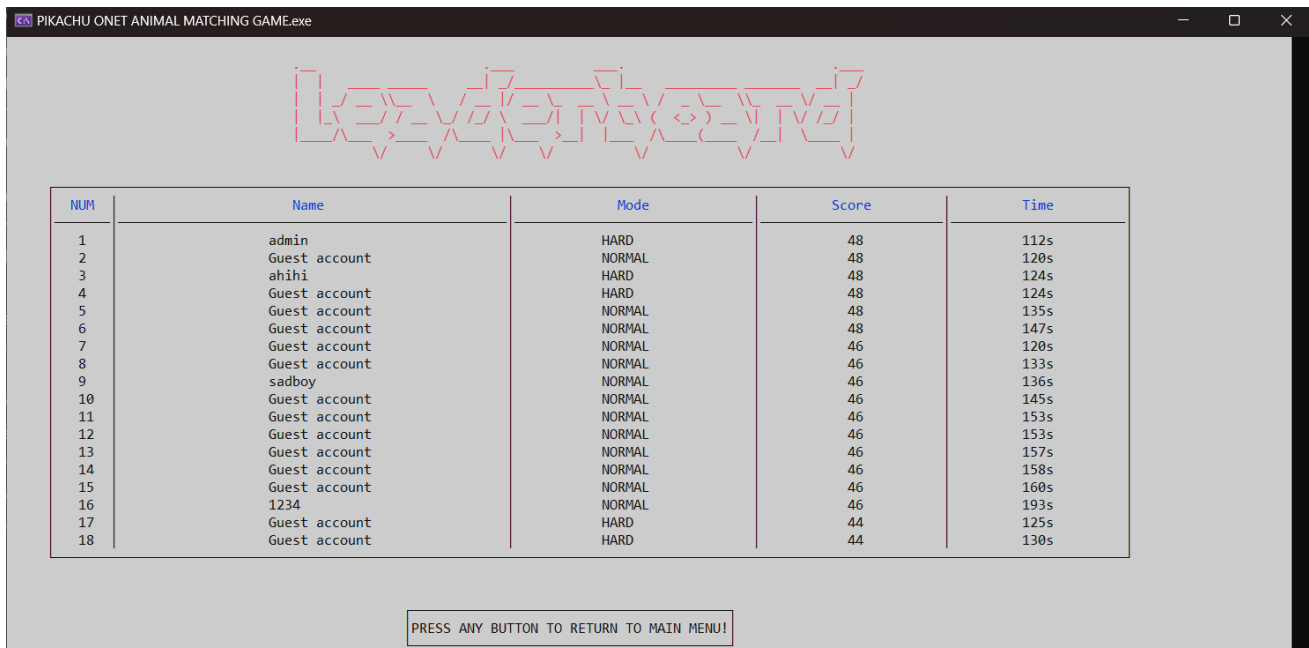
5.5 Leaderboard

A Leaderboard is a list that includes information on players with the highest scores. The list includes information such as the number of players in the leaderboard (Num), the players' name (Name), the players' scores (Score), and the time taken to complete the game (Time).

In terms of the algorithm, to create a leaderboard, we will use a leaderboard.bin to store all the information of players after they complete the game. Then, from the leaderboard.bin, we will read this information into an array with the data type Account called listAccountRecord.

From the array, we will sort it in descending order based on the score, and if the scores of the players are equal, the program will sort them according to the least time taken to complete the game. The 18 best records will be displayed on the leaderboard.

Let's discuss more the way time play is counted in our program. We use the clock() function in library c-time to count time. First, when the game begins we get the start time with the clock() function then when the player presses the ESC button or 'I' button for more information we get the end time, and a time-use variable is used to count to the time with the addition. Then right before the player come back to the game the start time will be gotten again. The game is finished is when the end time will be taken for the last time and no more start time, and again time-use is added and we have the final playtime.



| NUM | Name | Mode | Score | Time |
|-----|---------------|--------|-------|------|
| 1 | admin | HARD | 48 | 112s |
| 2 | Guest account | NORMAL | 48 | 120s |
| 3 | ahihi | HARD | 48 | 124s |
| 4 | Guest account | HARD | 48 | 124s |
| 5 | Guest account | NORMAL | 48 | 135s |
| 6 | Guest account | NORMAL | 48 | 147s |
| 7 | Guest account | NORMAL | 46 | 120s |
| 8 | Guest account | NORMAL | 46 | 133s |
| 9 | sadboy | NORMAL | 46 | 136s |
| 10 | Guest account | NORMAL | 46 | 145s |
| 11 | Guest account | NORMAL | 46 | 153s |
| 12 | Guest account | NORMAL | 46 | 153s |
| 13 | Guest account | NORMAL | 46 | 157s |
| 14 | Guest account | NORMAL | 46 | 158s |
| 15 | Guest account | NORMAL | 46 | 160s |
| 16 | 1234 | NORMAL | 46 | 193s |
| 17 | Guest account | HARD | 44 | 125s |
| 18 | Guest account | HARD | 44 | 130s |

PRESS ANY BUTTON TO RETURN TO MAIN MENU!

Image 13: The Leaderboard interface in our program

5.6 Move Suggestion

As for the move suggestion, first, we use the same method as mentioned in the Game finish verification part to locate the first valid pair. Then we color it purple and let the program sleep for 500 milliseconds then color it white again. This action lessens the effort to paint the suggested pair with every move the player makes if we just let it purple until it is deleted. And because it is the first valid pair so even if the player uses the assist key right again they got the same results and that suit the game aesthetic.

6 Extra Advanced Features

In the Extra Advanced Features section, we will analyze the Stage Difficulty Increase. Stage Difficulty Increase is a high difficulty level where after the player connects and deletes a pair of cells, all the cells to the right of the deleted cells will slide and stack up on the left side.

In this section, we have implemented this feature using both 2D-Pointer array and LinkedList. In this report, we will analyze both methods and compare their running time and complexity requirements.

6.1 Algorithms using 2D-Pointer array

- Step 1: Determine which of the selected cells is behind the other (i.e., has a greater y-coordinate), and call it maxCell and the other minCell.
- Step 2: Use a while loop to iterate through the row of maxCell starting at (the column of maxCell + 1), and in each iteration, assign the "pokemon" starting at (the column of maxCell) to the pokemon at (the column of maxCell + 1). The loop stops when it encounters a cell with a pokemon of '0' or reaches the end of the row.
- Step 3: After completing the loop above, assign the pokemon of the ending cell (the last cell with a non-'0' pokemon in the same row as maxCell) to '0'.
- Step 4: Repeat steps 2 and 3 for minCell.

Analysis: By choosing and starting from the cell to the right of the other cell, we can cover cases where the two selected cells are on the same row and may be adjacent to each other. Using the while loop helps to optimize the algorithm's processing time.

6.2 Algorithms using Singly Linkedlist

Regarding the nodes in a linked list, each node has a char variable containing the pokemon, an integer variable containing the row x, an integer for the column y, and a Node* pNext pointing to the next node. To be more specific about assigning data to this linked list $x = i / \text{WIDTH}$ and $y = i \% \text{WIDTH}$ when iterating from i equals 0 to i less than $\text{HEIGHT} * \text{WIDTH}$.

Additionally, we use another function called getNode which takes in pHead(a pointer to head of the linked list) and pos (the position), and then returns the

address of the node at the pos position.

The algorithm for one iteration is as follows:

- Step 1: Use the getNode function to get the addresses of the two selected nodes.
- Step 2: Assign the pokemon of the two nodes to '1'.
- Step 3: Use the getNode function to get the addresses of the two head nodes of the selected nodes.
- Step 4: Traverse the two rows with selected nodes starting from the head of each row, and traverse each row once.
- Step 5: When encountering a node with a pokemon of '1', assign the pokemon of that node to the pokemon of the next node if the pokemon is not '0' and the next node is in the same row.
- Step 6: Continue assigning the pokemon of the following nodes to the pokemon of the next node until the end of the row.
- Step 7: Assign the pokemon of the last node in the row to '0'.

Analysis:

Because the linked list is single, there is no way to iterate backwards from the end of the row to the selected node. So that, we need to iterate twice for two rows which have the selected nodes (once for each row), and iterate from the beginning to the end of the row. Note that these two rows may actually be the same row.

By setting the pokemon of the selected node to '1' before, we can distinguish it from '0' to save the number of iterations needed. Additionally, in case the two selected nodes are in the same row, we can still find the node that contains the pokemon of '1' by iterating from the beginning to the end of the row.

We do not use node deletion because it is difficult to synchronize with a two-dimensional array. After performing operations on the linked list, we need to copy it back to the 2D-Pointer array to display it on the console that move the pointer around the 2D array.

6.3 Comparisons the 2D-Pointer Array and LinkedList

1. The worst-case complexity when choosing a correct pair of cells once

- 2D-Pointer Array:

We use one comparison to find the maximum and minimum cell, two while loops iterating until the end of the two rows (of the max and min cells), so we have $2 * (WIDTH - 1)$ iterations in total, and within each loop, there are 2 checks: $i < WIDTH - 1$ and the next cell in the same row and column has a pokemon $!= '0'$, so we have a total of $4 * (WIDTH - 1)$ comparisons. In total, there are $4 * WIDTH - 3$ comparisons in the worst case for one selection.

- Linked list:

Because in any selection, we need to loop twice from the beginning of the row to the end of the row to find the node with the pokemon value of '1', so the worst-case scenario is selecting two nodes at the end of two different rows. At this moment, the number of times each row is looped is $WIDTH - 1$, two rows are $2 * (WIDTH - 1)$.

Each loop requires 4 comparison operators. Three operators are: checking the next node is different from NULL and checking the row x of the next node is equal to the row x of the current node; or checking the column y is in the end of row. So there are $8 * (WIDTH - 1)$ worst cases.

2. The best-case complexity

- 2D-Pointer Array:

The best-case complexity for selecting two cells at the ends is when the number of comparisons is $1 + 2 * 2 = 5$.

- Linked List:

The best-case complexity in linked list is $4 * (WIDTH - 1)$ because we can remove the comparison of the value y of the node and $WIDTH - 1$, and check the pokemon equals to '1' or not.

3. The worst-case complexity for all game

- 2D-Pointer Array:

The complexity to complete the entire game in the worst-case scenario (where the player always selects the first two cells of different rows throughout the game. For example, the Board has $HEIGHT \times WIDTH$ cells, number of times selected correctly is $N = (HEIGHT + WIDTH) / 2$, is as follows:

- N comparisons are needed to find the maximum and minimum cells.
- In the worst-case scenario of selecting the first two cells of different rows, the number of comparisons in the first round is $4 * (WIDTH - 1)$, and in subsequent rounds, assuming the player still selects the first two cells of those rows, the number of comparisons is $4 * (WIDTH - 2)$ until the end of those two rows, which results in $4 * (WIDTH - 1)!$ comparisons in total.
- Assuming HEIGHT is even, there are $HEIGHT / 2$ pairs of rows, and each pair requires $4 * (WIDTH - 1)!$ comparisons. Therefore, to compare the entire board, we need $2 * HEIGHT * (WIDTH - 1)!$ comparisons in total.
- The total number of comparisons in the worst-case scenario for one game is $N + 2 * HEIGHT(WIDTH-1)!$.
- Linked List:

At first, we need $8 * (WIDTH - 1)$ comparison operations. Next time, suppose that we choose two last nodes in two rows, we need $8 * (WIDTH - 2) + 2$ operations. And next time, we need $8 * (WIDTH - 3) + 4$. Subsequently, for finishing two rows, we need $8 * (WIDTH - 1)! + (WIDTH - 1) * (WIDTH - 2)$ operations. And suppose that we have $HEIGHT / 2$ pairs of rows, so we need $(HEIGHT / 2) * (8 * (WIDTH - 1)! + (WIDTH - 1) * (WIDTH - 2))$ operations in total.

4. Run time

We use the calculating time function `clock()` in `ctime` library in C++.
The run time for one iteration in the worst-case is:

- 2D-Pointer array: approximately 0,002s
- Linked List: approximately 0s

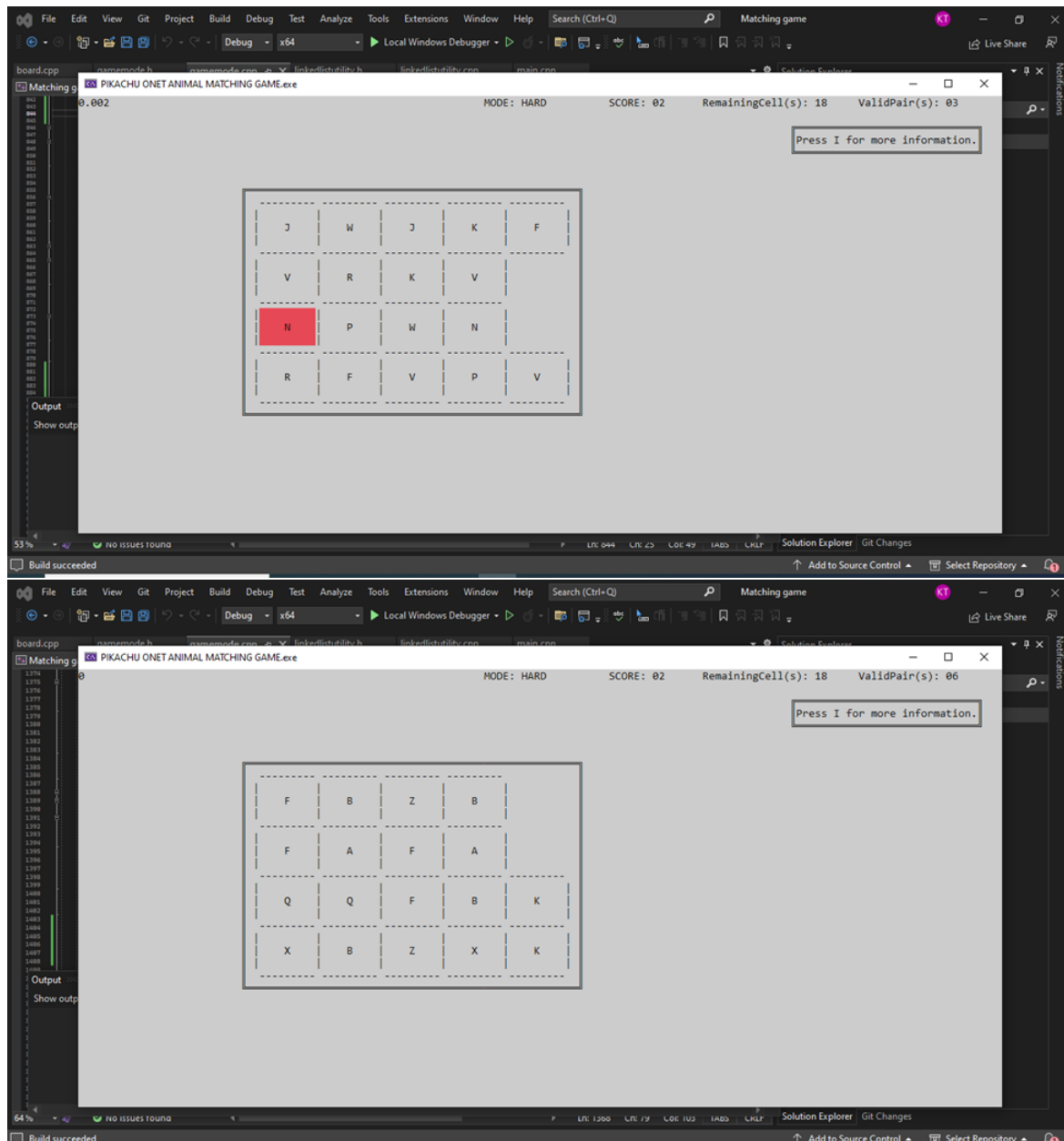


Image 14: The run time is in the up left corner of the console

7 Other features

7.1 Sign up and login

- Sign up: In this feature, the player is asked to enter a username, a password, and that password again to check the spelling. The username and password will be stored in the account struct object as a string class object. Later the information in this account will be stored in a file text called "account.txt". The storing format for the account in account.txt is Num/Username/Password with Num is that account number (default: '0' for the guest account). Something that can happen in the process of signing up is the user input an already existing username, in this case, the program gives you a notification and you are allowed to input the name again. One more thing is username and password don't exceed 15 characters and once again you will receive a notification if you input something more than 15 characters and you can input again. The last thing is if you repeat the password wrong you have to input your password again.
- Login: In this feature, the player is asked to input the username and password. The program then opens the account.txt to search for your username and password. If there is a match, you will successfully log in, else it will give you a notification about the wrong login information and you can retype your username and password. There is a way to return to sign up if you happen to misclick the login first before signing up.
- There is also a feature which is called "play as a guest" that allows you to play the game without an account. Your game information after you finish the game will also be stored in leaderboard.bin but with Num equals to '0' and the username is "Guest account".

7.2 Shuffle

- This feature shuffles the board randomly which means the position of cells is rearranged. This can happen in 2 ways, one is when you are out of moves which means the remaining cell is more than 0 and the valid pair is 0, the board will be shuffled automatically, and you won't lose any points in this case. The second case is if you press the 'S' button the board will also shuffle but your point decreases by 2.
- To build this feature we use vectors because vectors are very convenient. First, to shuffle the board, you will have to know the current status of the board or to be more specific all the cells that are not empty (cell Pokemon is not '0') and because

we don't know this number beforehand a static list is a no. But how about a dynamic list, well it is true that it is possible to use a dynamic list but the first you must do is to scan all the board data to count the number of the not empty cell and then use this number to create a dynamic list. But by using a vector you can skip this step with the pushback function. The second reason why vector is better than dynamic array is it is more flexible, its size can decrease in run time which is required in the algorithm we use to randomly reallocate the cell and vector also have many supporting functions which makes it super convenient just like string class.

- About the algorithm, First, I add all the not-empty cell pokemon in vector A and create an empty vector B. And let k be the size of said vector A. I let it loop from 0 to $k - 1$, in every loop I get 'n' a random number from 0 to the current size of vector A minus 1. Then I access this n position in vector A with $A[n]$. Then the Pokemon at $A[n]$ is pushed back to vector B, and I delete the Pokemon at $A[n]$ with erase function which causes the size of vector A to decrease by 1 but not k . This loop continues until vector A is empty and vector B is filled with k pokemon randomly. Then read the information in vector B into the board back orderly just like when we read the data from the board to vector A.

7.3 Custom mode

- This feature gives you 3 choices to go with the board size with is 3x4, 4x5, or 5x6. In this mode, the board is smaller than the standard mode 6x8 which will help you enjoy the game faster but because the way to count point is not that different from the standard mode the total point you can get is smaller and it may not be enough for your name to appear in the leader board. One more special thing is every custom mode has its own background so go find it out.

8 Game Tutorials

Initially, when you start the program, the animation will appear like the image below.

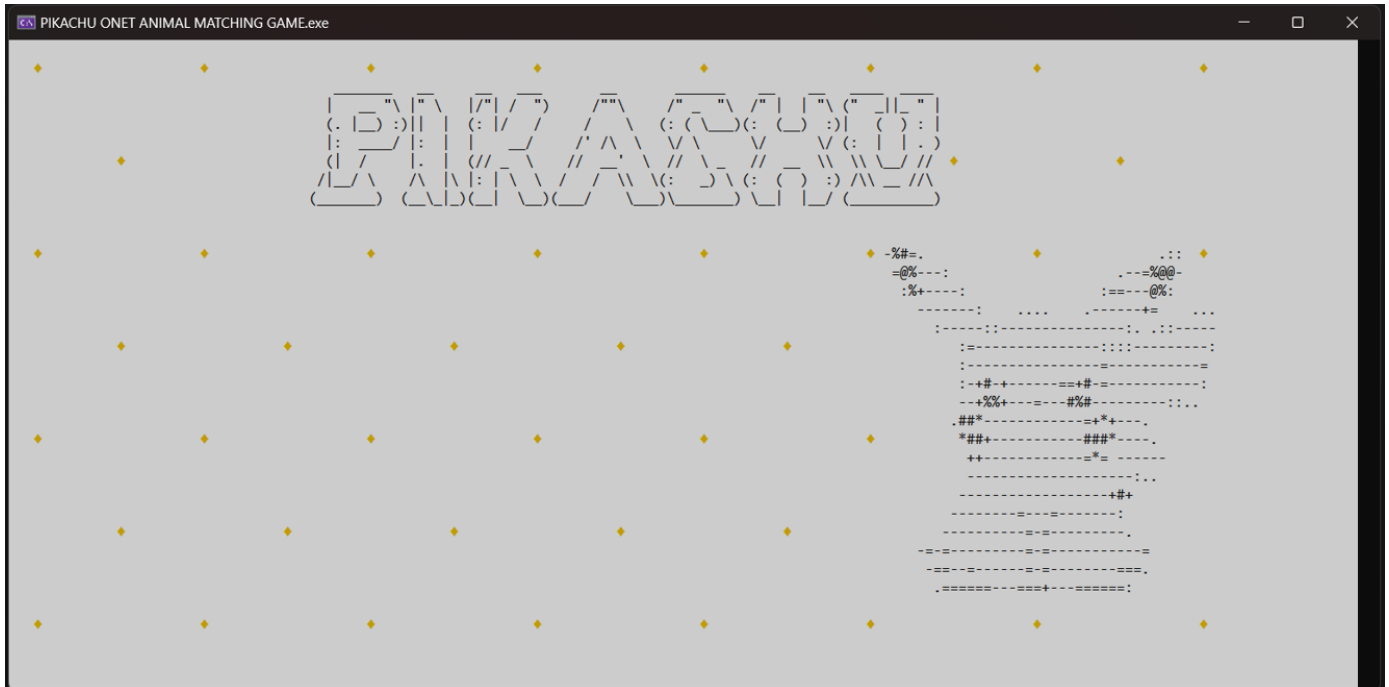


Image 15: Animation in our program

After that, the new screen will appear and there are many functions such as log in account, sign up account or you can play game as guess.

8.1 Log in and Sign up

8.1.1 Log in

With regard to Log in function, this is a place where users will enter their previously registered accounts.

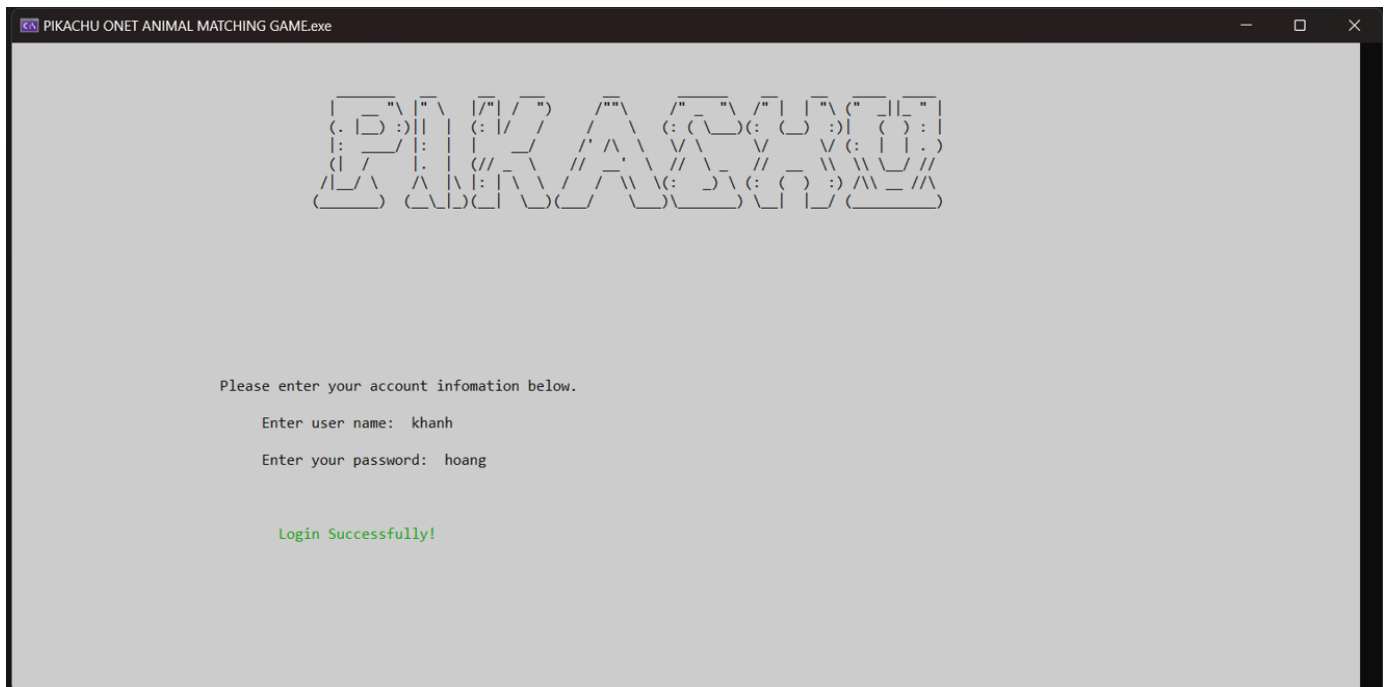


Image 16: Log in Screen

8.1.2 Sign up

Otherwise, in the Sign up option, the new users can create their account here and the new account will be stored in our program. To creating a new account, the users need to enter their user name, password and then repeat their password. If they repeat the wrong password, users need to enter again until the password is recognized valid.

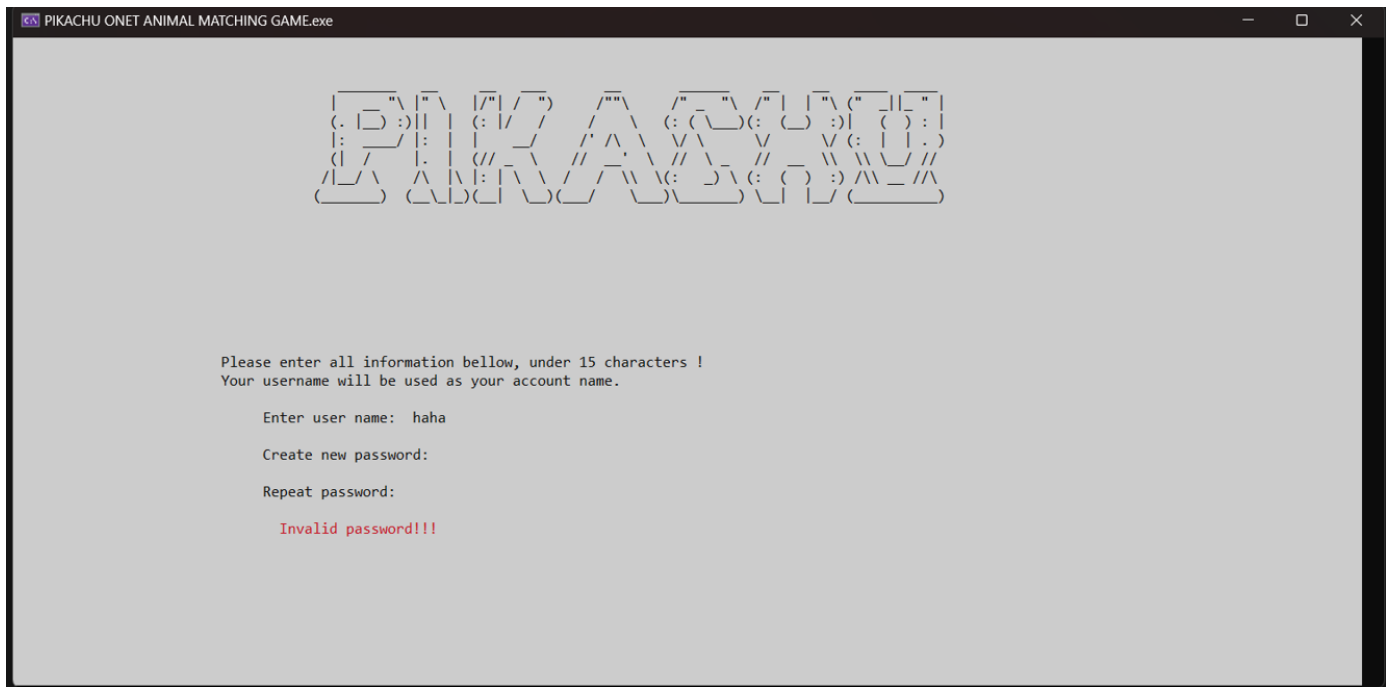


Image 17: The program announced like this if the users enter wrong password

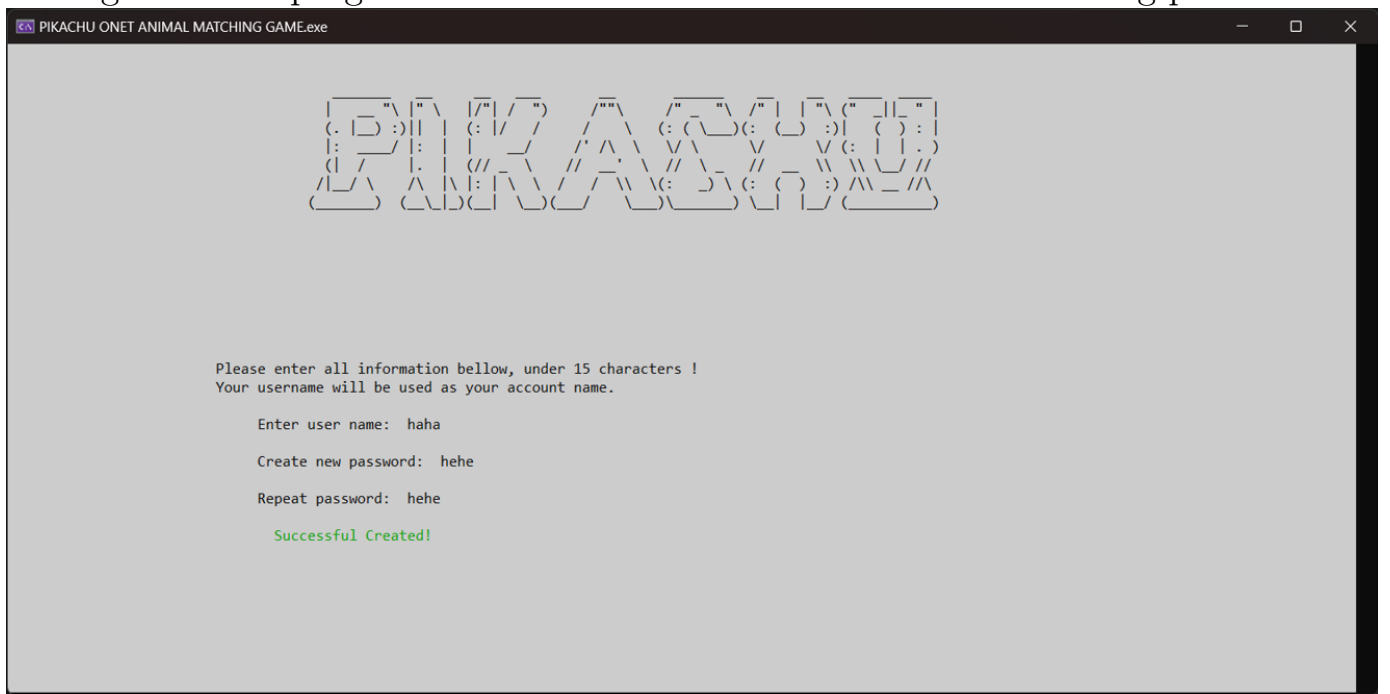


Image 18: Successfully sign up

8.1.3 Play as Guest

If the users do not want to create account, they can also play the game as the Guest, and their account name used in the game called Guest account.

8.2 Menu Screen

After finishing log in and sign up function, the Menu screen will appear with four main options for you.

8.2.1 New Game

First, there is a New Game option. After pressing it, there are four other options for players.

- **Normal:**

Normal option is a choice for users who want to play game in the normal mode. It means that there is no Stage Difficult Increase.

- **Hard:**

Hard option is a choice for users who want to play in the hard mode. This stage absolutely has the Stage Difficult Increase.

- **Custom:**

Custom mode is also a choice for playing game. When the player choose Custom, there are 3 options about the board size: 3x4, 4x5, 5x6. However, if the players want to go back, there is also Back option for them.



Image 19: The Custom interface

- **Back:**

There is also a Back button for users who want to go Back the Menu screen.

8.2.2 Help

The second option in Menu screen is Help. The Help is a tool that helps players learn more about the gameplay and other features in a game. In addition, the rules of game are also displayed in this function, along with the information about us and the game program.

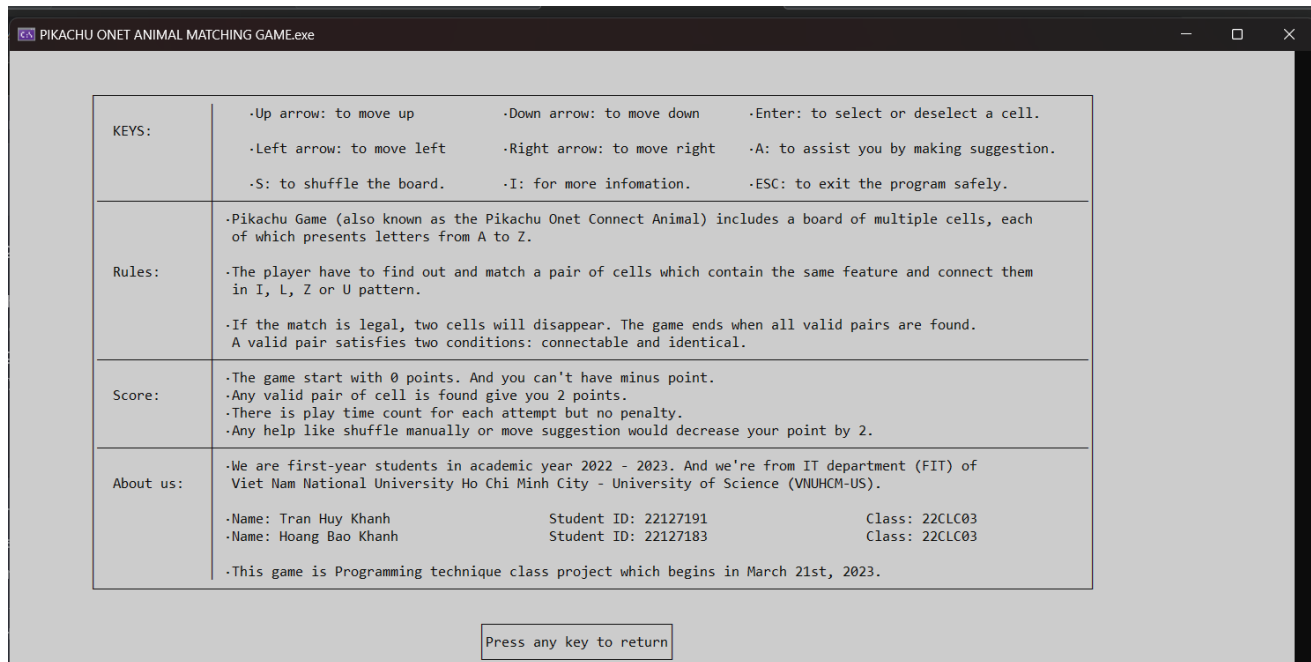


Image 20: The Help interface

8.2.3 Leaderboard

The third option is Leaderboard. This is a place that the top players who have the highest scores will be honored. In the Leaderboard, there are many players' information such as Name, Game mode, Scores and Time.

PIKACHU ONET ANIMAL MATCHING GAME.exe

Level 1

| STT | Name | Mode | Score | Time |
|-----|---------------|--------|-------|------|
| 1 | Guest account | NORMAL | 48 | 96s |
| 2 | Guest account | NORMAL | 48 | 113s |
| 3 | Guest account | HARD | 42 | 135s |
| 4 | Guest account | CUSTOM | 30 | 46s |
| 5 | Guest account | CUSTOM | 30 | 64s |
| 6 | Guest account | CUSTOM | 30 | 75s |
| 7 | Guest account | CUSTOM | 20 | 25s |
| 8 | Guest account | CUSTOM | 20 | 25s |
| 9 | Guest account | CUSTOM | 20 | 26s |
| 10 | Guest account | CUSTOM | 20 | 26s |
| 11 | Guest account | CUSTOM | 12 | 10s |
| 12 | Guest account | CUSTOM | 12 | 10s |
| 13 | Guest account | CUSTOM | 12 | 11s |
| 14 | Guest account | NORMAL | 12 | 11s |
| 15 | Guest account | HARD | 12 | 11s |
| 16 | Guest account | HARD | 12 | 13s |
| 17 | Guest account | CUSTOM | 12 | 13s |
| 18 | Guest account | CUSTOM | 12 | 13s |

PRESS ANY BUTTON TO RETURN TO MAIN MENU!

Image 21: The Leaderboard Interface

8.2.4 Exit

When the player does not want to continue playing the game, they can choose Exit to turn off the program. After choosing it, there will be a window to confirm whether the player wants to exit the program or not. If you choose "No", the program will turn to the Menu Screen, otherwise, the console will turn to the white screen and then you press Enter to close it.

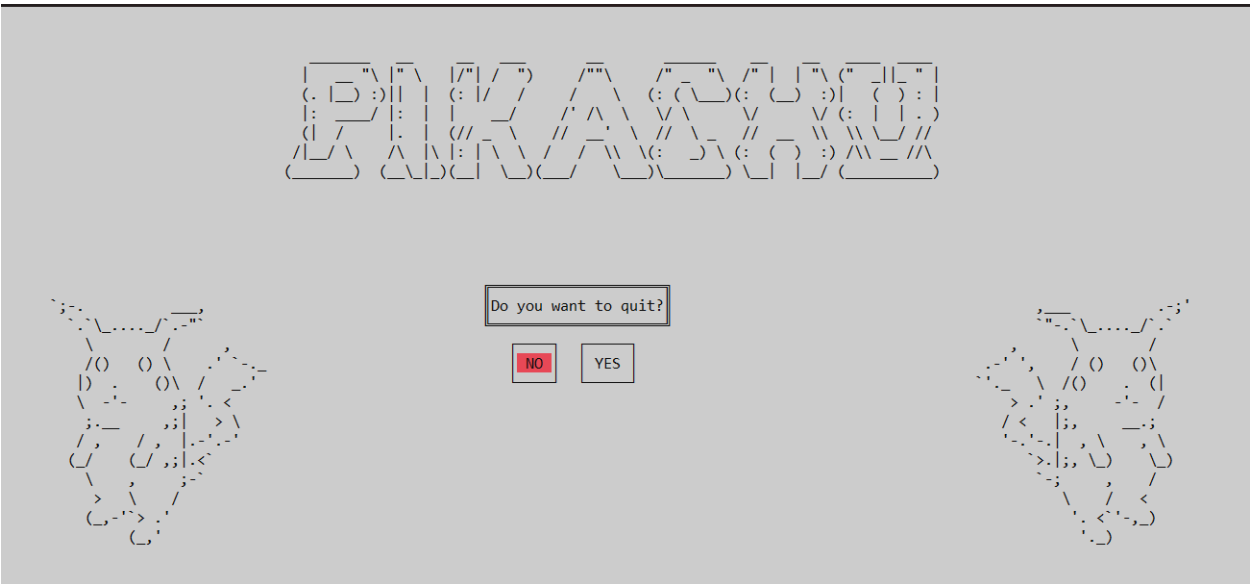


Image 22: Exit interface

8.3 Explain the necessity of all file .h and .cpp in the project

1. The console.h and console.cpp file

This is where the library windows.h is used. It mainly contains code to modify the console including color, console size, sound, disabling mouse input, disabling maximize, hiding cursor, hiding scrollbar... This is the first file to be created and will be included by all the files below.

2. The login.h and login.cpp file

This is the second file. It includes the definition of struct Account for the login screen and also for saving data after the player finished the game. There is a struct Login Screen here including all functions about sign up and login activity.

3. The menu.h and menu.cpp file

This file has the definition of the main menu which you get to after login. It contains all definitions for all kinds of screens like the main screen, the mode selecting screen, the exit screen, the win screen, and the help screen,... It also contains functions to draw and decorate the console like printing Logo, and drawing rectangles,...

4. The board.h and board.cpp file

The fourth-in-line file contains the definition of the board game. It includes board games in data in a 2d array and the information about the board which will be displayed in the console. There is a struct Cell here and a board is made of cells. The cell is a member in a 2d array and also contains information about where it is in the console. The struct Board has a constructor to initialize board data, the pattern drawing function, the background-wise functions, and the shuffle function.

5 The gameplay.h and gameplay.cpp file

The fifth file contains all the functions and variables to store information throughout the playing process. It includes the check pattern function and the final check 2 cells function which return a line to match 2 cell if that 2 cell is connectable. The variable here is the score, remaining Cells, and valid pair.

6. The linkedlistutility.h and inkedListutility.cpp file

This file exists outside of the stream of this program. This file provides support to create a hard game linked list mode.

7. The gamemode.h and gamemode.cpp file

This is the sixth file that includes all the above files. This file contains the definition of all game modes including normal, hard, and custom modes. There is also a function move suggestion here to use in every game mode.

8. The main.cpp

The file to run the program.

Note: You can see more detailed instructions through our video which has been posted on Youtube via the following link: [Game Tutorials](#)

More Importance Note:

To run the program correctly, you should run the Matching Game.sln. There is a .exe file in 64x/debug/ but it won't run well if you just double-click into it here. If you just want to run the .exe file, pass it to the Matching game/matching game folder and run it here. This happens because the .exe file in 64x/debug/ can not read the data in the matching game file. One more thing is when you run a .sln file, visual studio will automatically generate a hidden .vs file which is very heavy.

9 References

- [1] nguyenvanquan7826, "[Thuật toán Pokemon \(Pikachu\)](#)", 25/03/2014: reference algorithm for checking I, U, L, Z.
- [2] J.Delta, codelearn.io, "[Windows.h và Hàm định dạng màn hình Console P1](#)", 19/06/2020: reference setting up console
- [3] J.Delta, codelearn.io, "[Windows.h và Hàm định dạng màn hình Console P2](#)", 25/06/2020: reference setting up console
- [4] Louis2602, github.com, [Pikachu-Game](#), 22/04/2022: reference about setting up sound and screen dimensions for a console and also the way to use a structure to store function.
- [5] PhVanMin, github.com, [Pikachuuu](#), 22/04/2022:
- [6] Ascii Art Archive, asciiart.eu, [Ascii Art Archive](#): using for finding background.
- [7] patorjk.com, [Text to Ascii Art Generator](#): using for creating logo.