

UNIVERSITY OF SCIENCE - VNUHCM

INFORMATION TECHNOLOGY FACULTY

COURSE NAME

ADVANCED PROGRAMMING TECHNIQUE AND PRACTICE

---

# The Matching Game

---

*Group:*

Nguyễn Tấn Lộc – 23127406  
Lê Thanh Phong – 23127452  
23CLC08

*Lecturer:*

Bùi Huy Thông  
Nguyễn Trần Duy Minh



Ho Chi Minh City, April 11th 2024

# Contents

<b>1</b>	<b>Tutorial</b>	<b>2</b>
1.1	Set up	2
1.2	How to play	2
1.2.1	Menu	2
1.2.2	Login	6
1.2.3	Gameplay	7
<b>2</b>	<b>Code Explanation</b>	<b>17</b>
2.1	Game Board	17
2.2	Matching Function	18
2.2.1	Check Row Algorithm	18
2.2.2	I Matching	19
2.2.3	L Matching	20
2.2.4	Z Matching	23
2.2.5	U Matching	23
2.3	Image Illustration	26
2.4	Data Structure Comparison	28
2.4.1	2-D Array Pointer	28
2.4.2	Linked List	29
2.5	Advanced Feature	29
2.5.1	Color and Sound Effect	29
2.5.2	Visual Effect	30
2.5.3	Background	31
2.5.4	Move Suggestion	32
2.5.5	Leaderboard	33
2.5.6	Game Account	36
2.6	Extra Advanced Features	36
<b>3</b>	<b>File Structures</b>	<b>37</b>
3.1	Header Files	37
3.2	Source Files	39
3.3	Resource Files	39
	<b>Reference</b>	<b>40</b>

# 1 Tutorial

## 1.1 Set up

First, you have to make sure to download all files in our folder. From there, you can just run the file Pikachu.exe in Pikachu folder to play the game.

## 1.2 How to play

### 1.2.1 Menu

First, you will see the START box and the theme song will be played. Please open in full screen for the best experience and press any key on keyboard to continue.

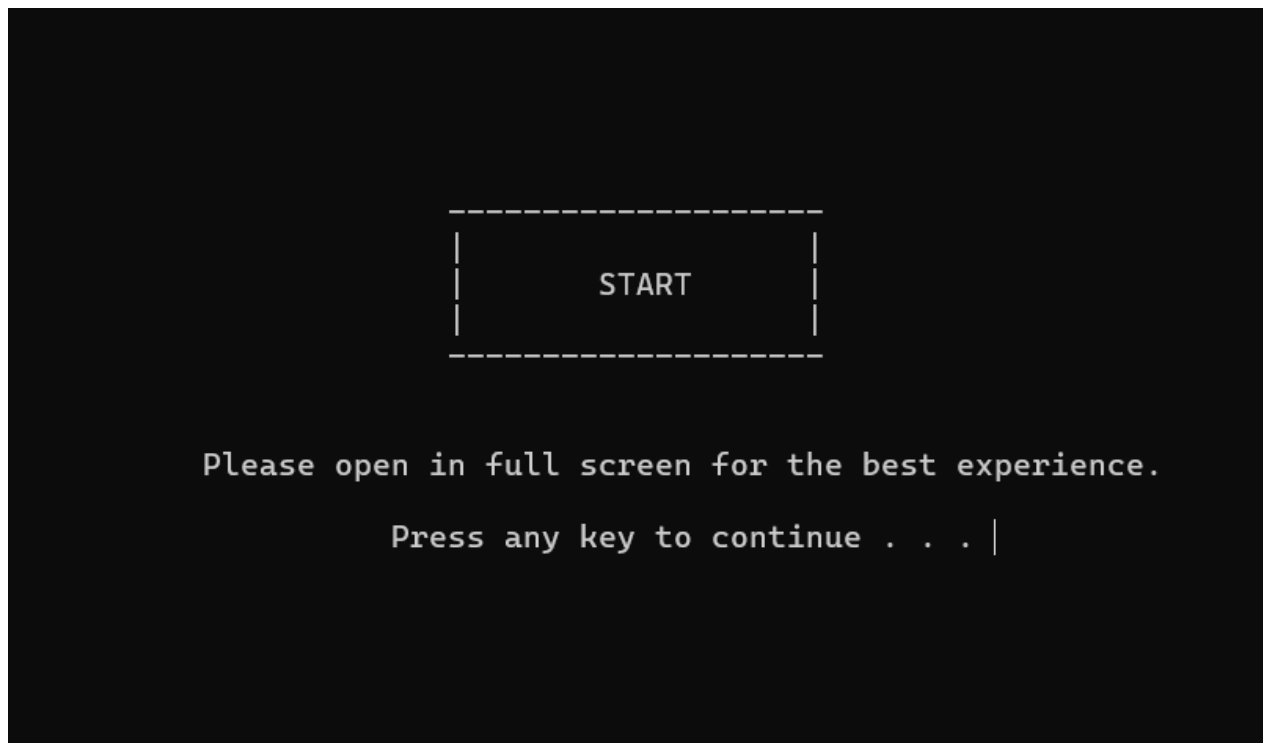


Figure 1: Start

In the main menu, you will see 5 buttons: Play, Leaderboard, Help, About and Exit. Press W or Up Arrow to move up and S or Down Arrow to move down. Press Enter to choose the current option.



Figure 2: Main menu

- Play: to choose a level to play.
  - There are 3 levels: Easy, Normal and Hard.
  - Easy level: when you successfully delete 2 cells, other cells won't slide.
  - Normal level: other cells will slide toward to the center if the next row or the next column near the center is empty.
  - In hard level, other cells will slide toward to the center after 2 cells have been deleted.
  - Similar to choose option in main menu. Using W or Up Arrow to move up and S or Down Arrow to move down. Press Enter to select the option.
  - When you move up or move down to select the option, a sound effect will be played.
  - Press Esc on keyboard to return the main menu

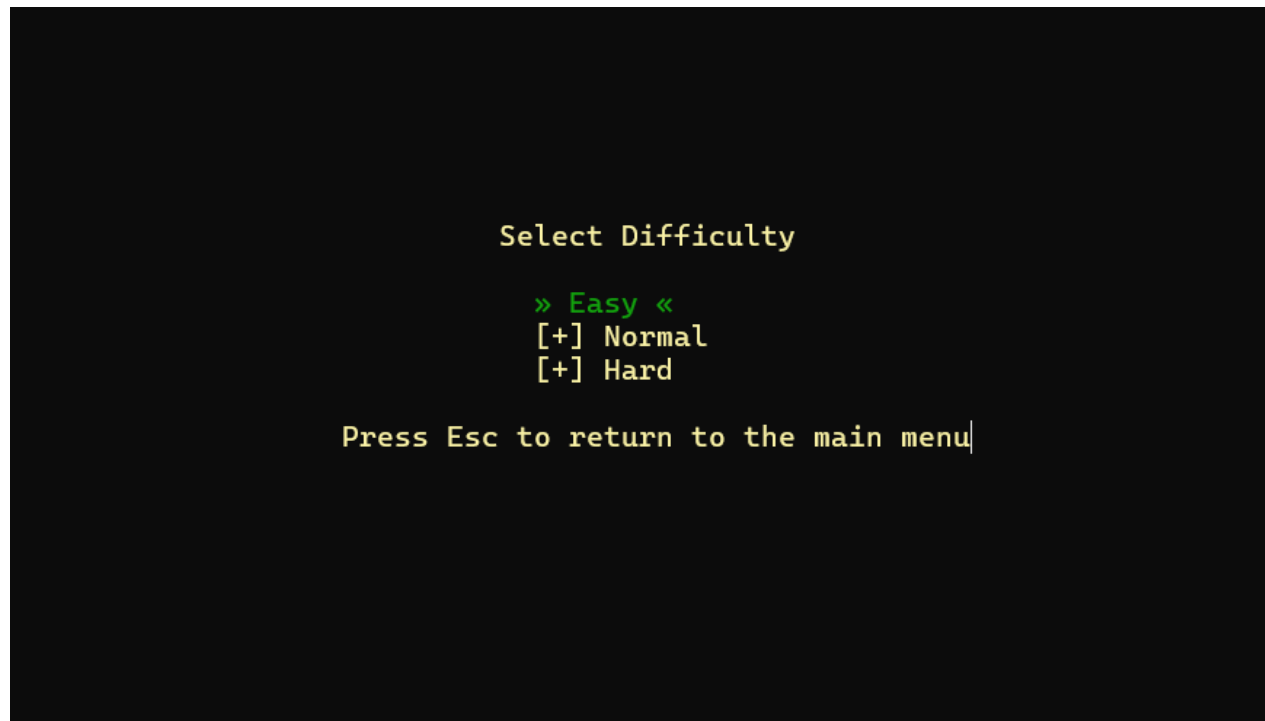



Figure 3: List of levels

Leaderboard: to display the list of top 10 highest score players

There are 3 information: Name, Level and Score. You can press Esc on the keyboard to return to the main menu.

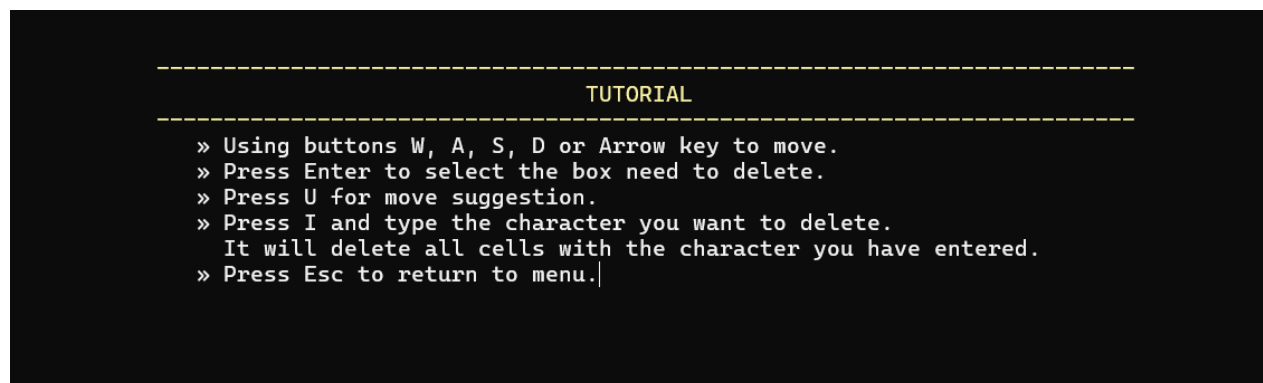


NAME	LEVEL	SCORE
loc	3	540
fds	1	130
k	1	110
p	2	90
abc	1	80
tam	3	50
phong	1	40
jkl	1	40
p	1	10
a	1	0

Press Esc to return to the main menu|

Figure 4: Leaderboard

Help: display game tutorial. Tutorial will show all keys you need to know to play all levels. You can press Esc on the keyboard to return to the main menu.



TUTORIAL
» Using buttons W, A, S, D or Arrow key to move.
» Press Enter to select the box need to delete.
» Press U for move suggestion.
» Press I and type the character you want to delete.
It will delete all cells with the character you have entered.
» Press Esc to return to menu

Figure 5: Game Tutorial

About: to display introduction of the project. You can press Esc on the keyboard to return to the main menu.

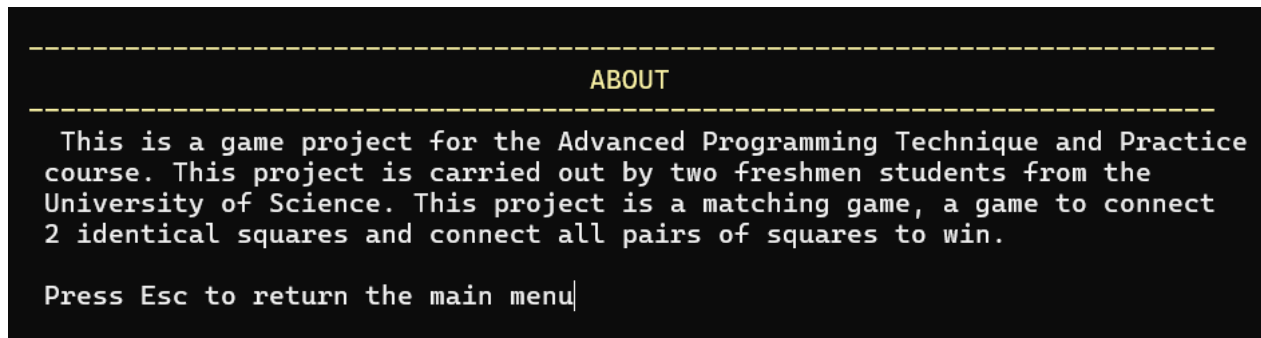


Figure 6: Game Tutorial

Exit: to end the game.



Figure 7: Game ending

### 1.2.2 Login

Every time playing a new level, you have to input your name to play and the game will save your name, your score and your current level you're playing.

After that, your information will be added to the list and sorted all players' score in descending order.



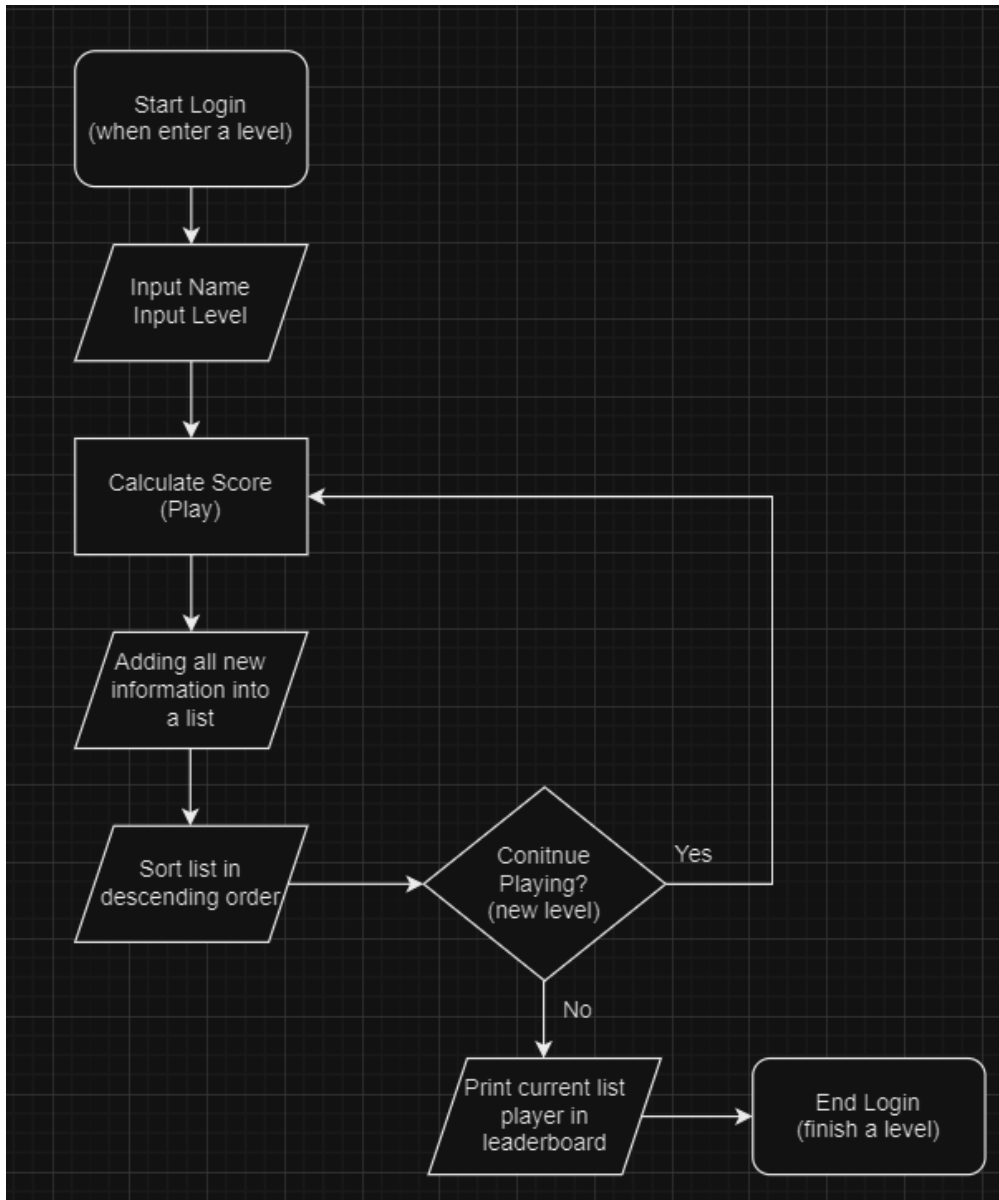


Figure 8: How to save player's information

### 1.2.3 Gameplay

After choosing a level, you will input your information to save your score in that level. The name must not be more than 20 characters or empty.

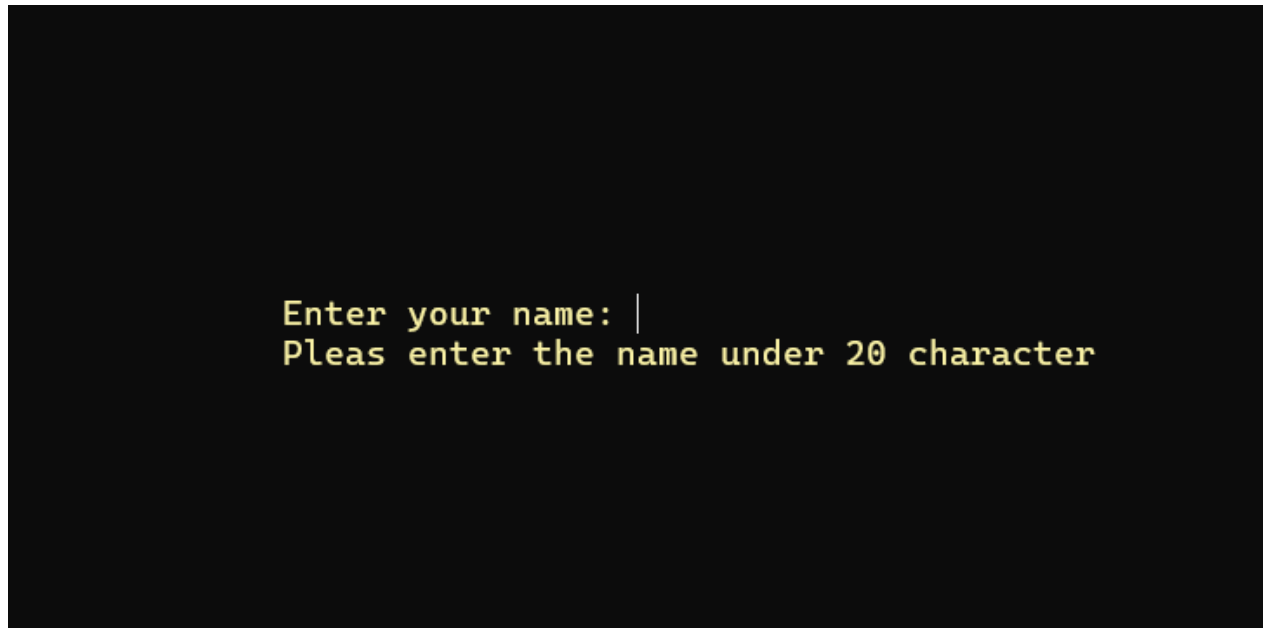


Figure 9: Login

After entering your name, you will start the level.

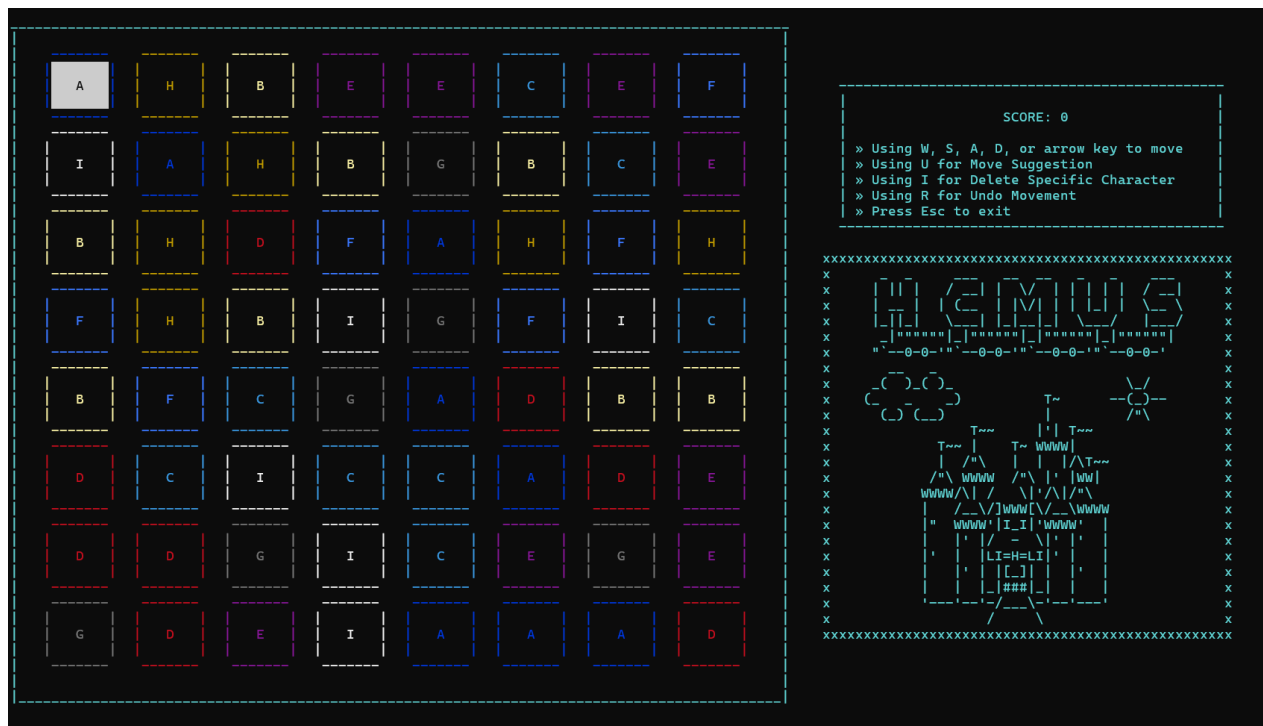


Figure 10: Playing

You will see the game board on the left is the main game session. On the right, you will see the box has your scoreboard and below that there are instructions to play the game. We also decorate the level with a picture using ASCII Art for the background.

Press W, S, A, D on the left or Arrow key on the right of keyboard to move. Press Enter to select the cell need to delete. For example, we want to delete the E pair cells. Move to the cell E and press Enter. After that, your score will increase.

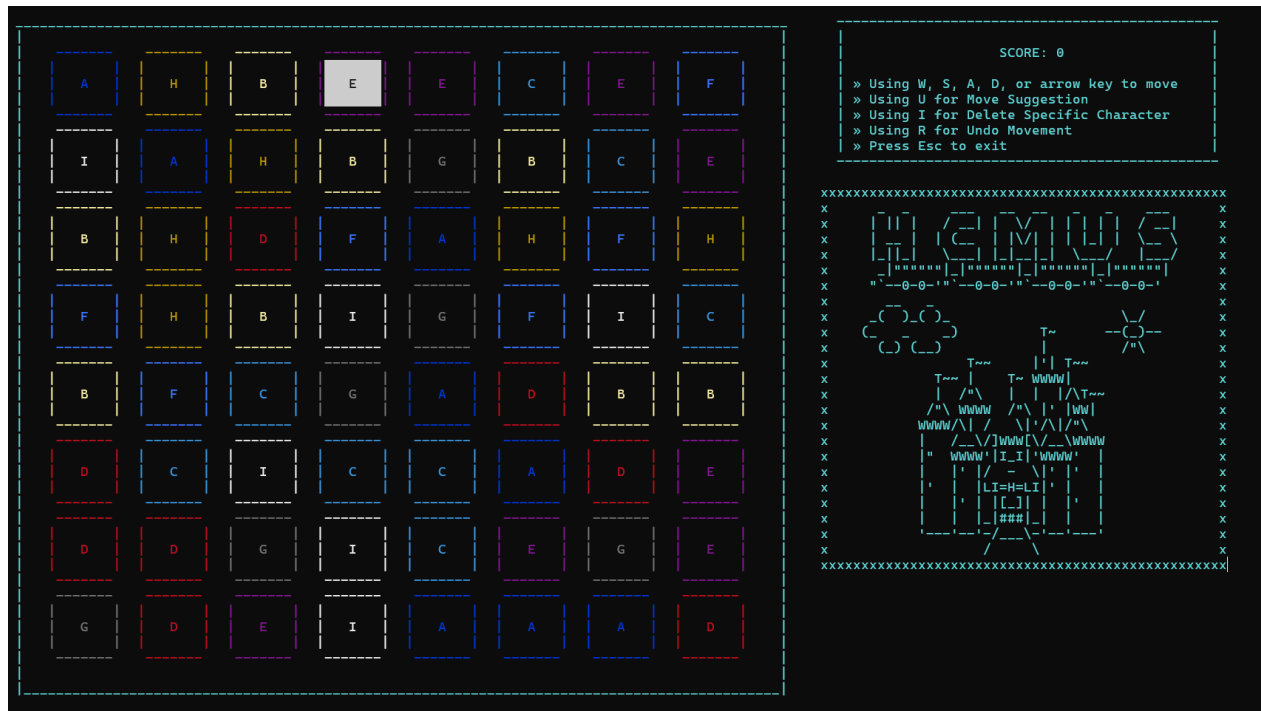


Figure 11: Moving to the first E cell

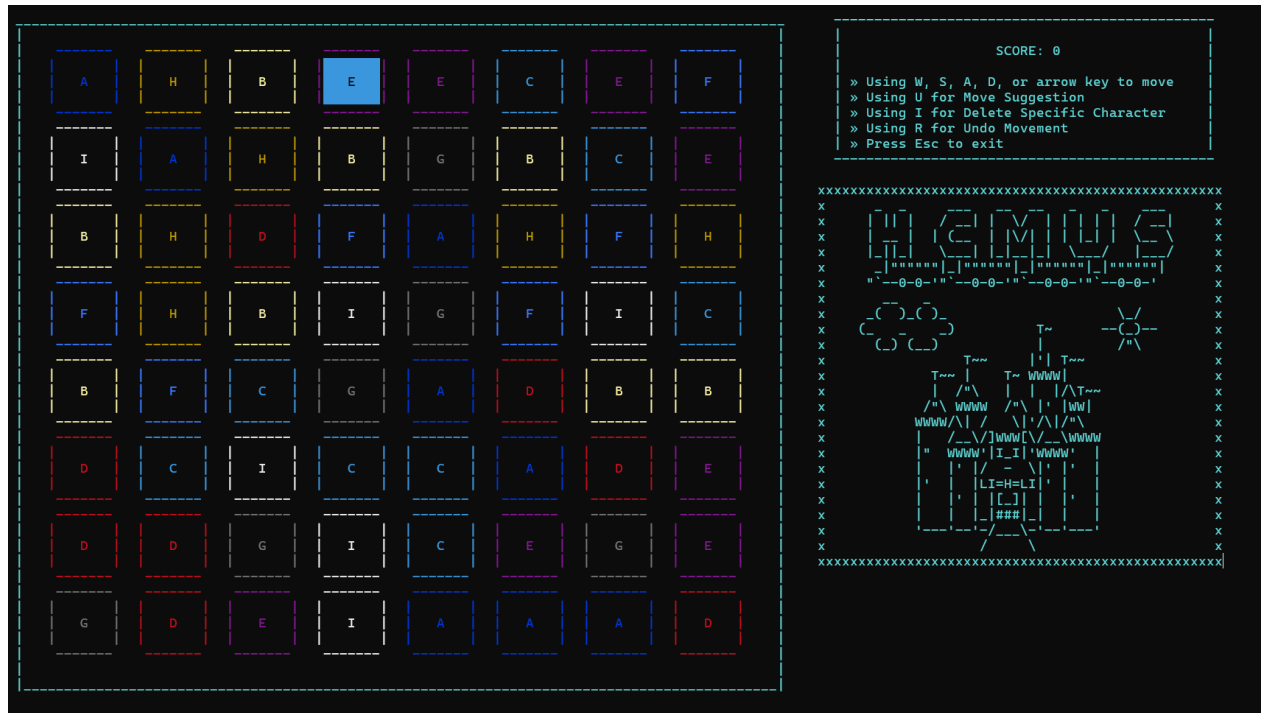


Figure 12: After pressing Enter to choose 1

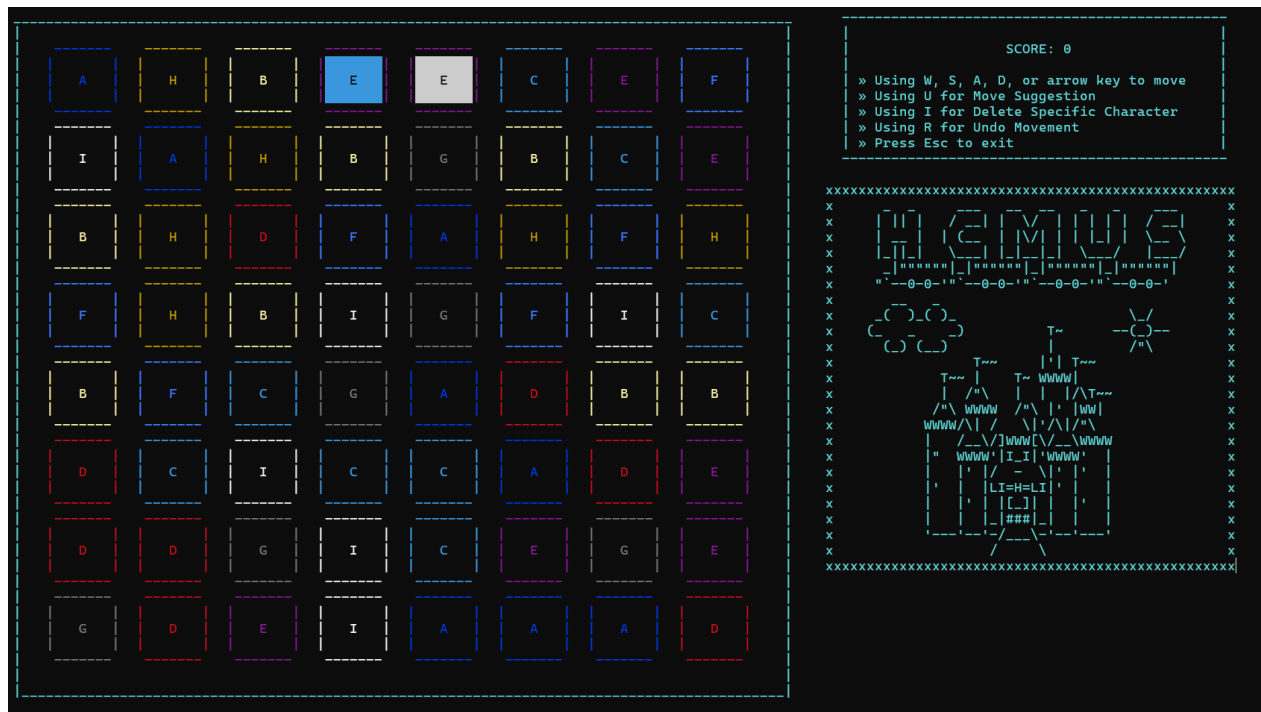
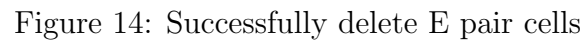


Figure 13: Moving to the second E cell



The screenshot shows a terminal window titled "The Game". The main area is a 10x10 grid of letters, each enclosed in a dashed border of a different color. The letters are as follows:

A	H	B		C	E	F			
I	A	H	B	G	B	C	E		
B	H	D	F	A	H	F	H		
F	H	B	I	G	F	I	C		
B	F	C	G	A	D	B	B		
D	C	I	C	C	A	D	E		
D	D	G	I	C	E	G	E		
G	D	E	I	A	A	A	D		

Below the grid, there is a list of controls:

- » Using W, S, A, D, or arrow key to move
- » Using U for Move Suggestion
- » Using I for Delete Specific Character
- » Using R for Undo Movement
- » Press Esc to exit

At the bottom of the terminal window, there is a large, stylized ASCII art logo for "THE GAME" with a complex, abstract design below it.

Figure 15: Before Undo

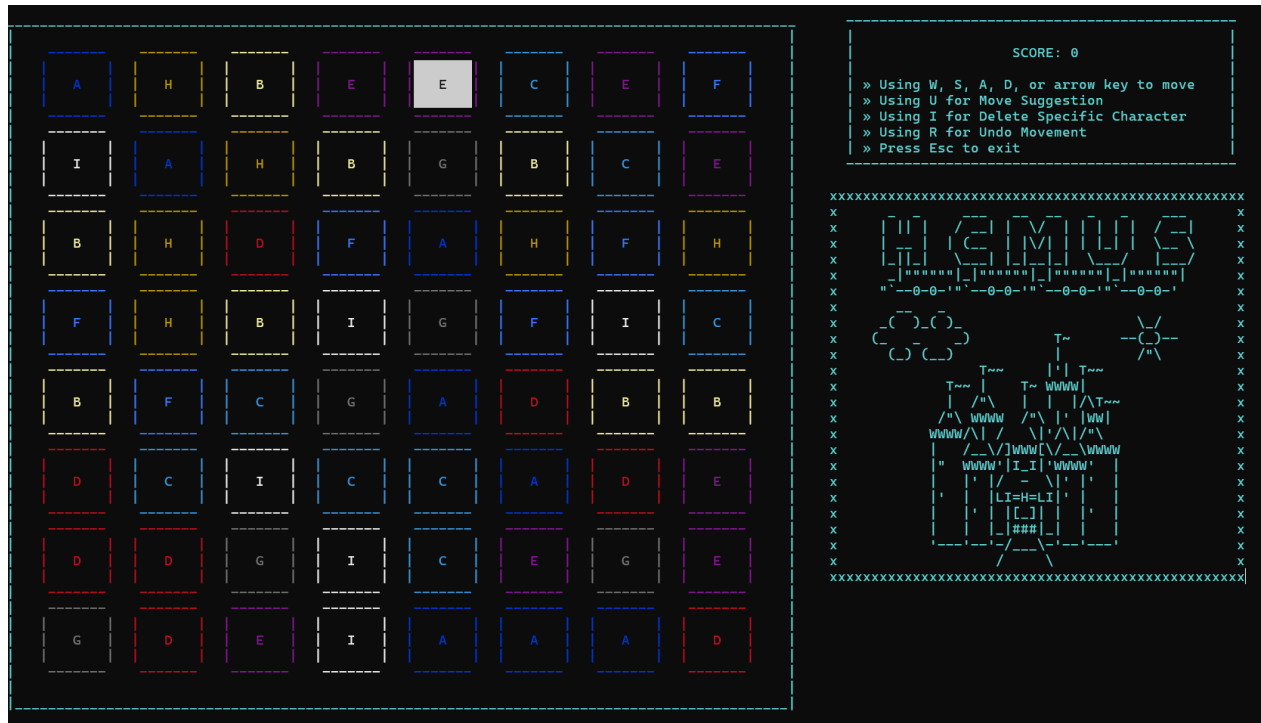


Figure 16: After Undo

Press I to delete all character similar to the specified character you enter. For example, we want to delete all I cells. After using that, your score will decrease.

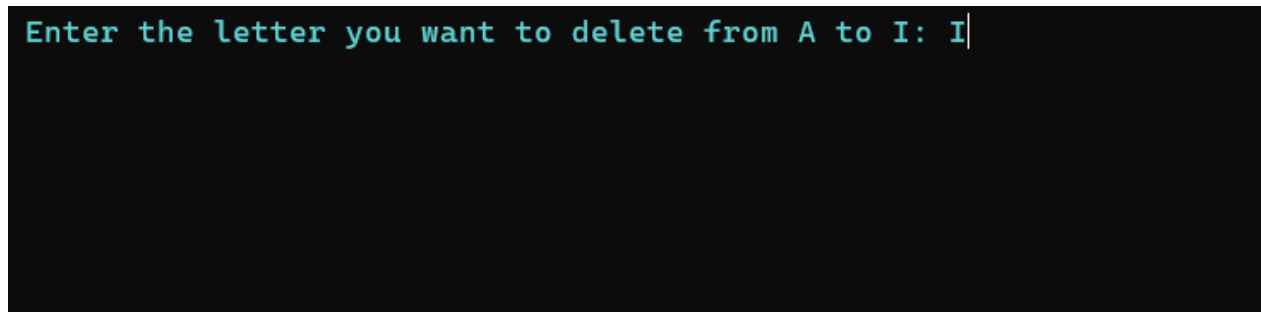
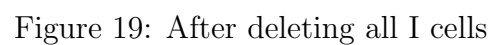
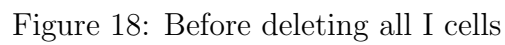


Figure 17: Input I to delete



You can also undo that deletion, just press R.

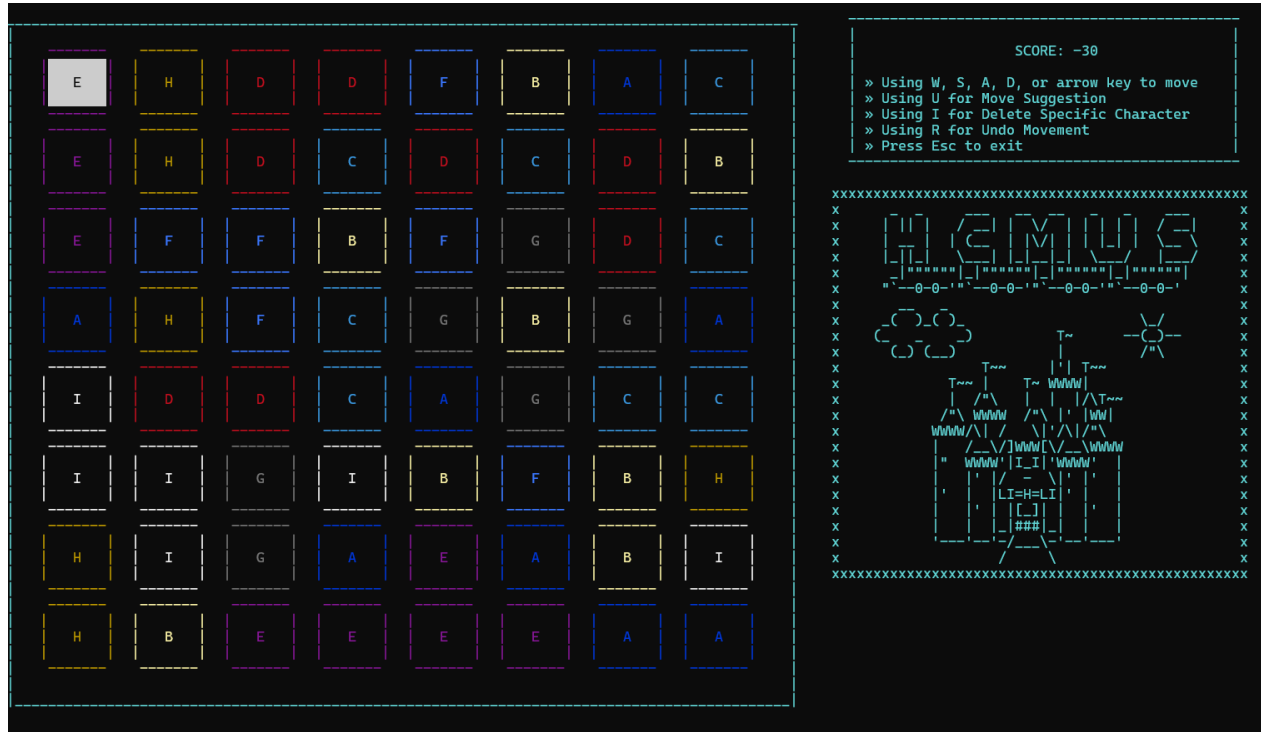
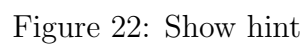


Figure 20: Undo deletion of all I cells

Next, you won't find any valid way, you can press U for hints.





You press Esc to exit the game. The game will ask you if you want to continue playing that level or not.

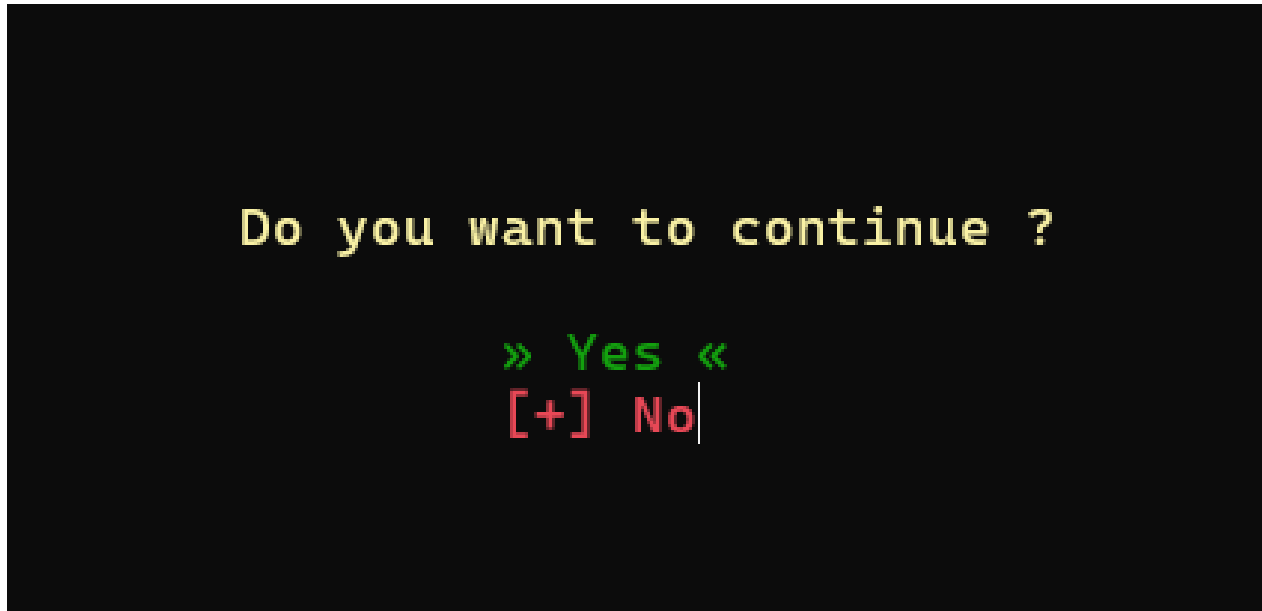


Figure 23: Continue to playing or not

When you matching all pair cells, you win the game and the game will show winner congratulation.



Figure 24: Winner

Finally, you can press any key on keyboard to continue and the game will ask you if you want to continue playing that level or not. You can return to the leaderboard to see if your score is in that list or not.

We also have a demonstration video for [How To Play](#).

## 2 Code Explanation

### 2.1 Game Board

We are creating a game screen consisting of an 8x8 array with 9 characters in English alphabet from A to I appearing with a ratio of 5 occurrences of the number 8 and 4 occurrences of the number 6, and the total game screen has 64 characters corresponding to 32 pairs.

With that quantity, we have created a two-dimensional pointer array to generate the game screen with an initial size of 10x10 arrays to facilitate subsequent connection algorithms.

---

#### Algorithm 1 Generate Board

---

```

1: function GENBOARD
2:   Initialize random number generator with current time
3:   Declare a two-dimensional pointer 'a'
4:   Allocate memory for a two-dimensional array 'a' with 10 rows
5:   for  $t \leftarrow 0$  to 9 do
6:     Allocate memory for each row with 10 columns
7:   for  $i \leftarrow 0$  to 9 do
8:     for  $j \leftarrow 0$  to 9 do
9:       Assign a whitespace ( ' ' ) to each cell
10:  Initialize an array 'rd' with 9 elements, each element initially set to 8
11:  Declare variable 'random'
12:  Generate a random number and assign it to 'random'
13:  for  $i \leftarrow 1$  to 8 do
14:    for  $j \leftarrow 1$  to 8 do
15:      while the quantity of the element at position 'random' in the array 'rd' is 0
16:        do
17:          Generate a new random number and assign it to 'random'
18:          Assign the value from the character 'A' plus 'random' to the cell at row 'i' and
19:          column 'j' of the board
20:          Decrease the value of the element at 'random' index in the array 'rd' by 1
21:          Generate a new random number and assign it to 'random'
22:  Return the pointer 'a' (representing the generated board)

```

---

Afterward, we created a one-dimensional array to sequentially assign 5 occurrences of the number 8 and 4 occurrences of the number 6 as the number of random occurrences of letters from A to I, and assigned them to the main array. When the value in the one-dimensional array becomes 0, the corresponding letter in the cell will no longer appear. Finally, we returned a two-dimensional pointer array for the gameplay part.

In addition to that, we also created a three-dimensional pointer array to store the steps that the player has taken in order to implement the function of allowing the player to go back to previously taken steps.

---

**Algorithm 2** Create a 3-D array to save all movement

---

```

1: function CREATEREDOBOARD
2:   Declare a three-dimensional pointer 'a'
3:   Allocate memory for a three-dimensional array 'a' with 32 planes
4:   for  $p \leftarrow 0$  to 31 do
5:     Allocate memory for each row with 10 rows
6:     for  $r \leftarrow 0$  to 9 do
7:       Allocate memory for each column with 10 columns
8:       for  $c \leftarrow 0$  to 9 do
9:         Assign a whitespace ( ' ') to each cell
10:  Return 'a' (representing the created 3D board)

```

---

## 2.2 Matching Function

### 2.2.1 Check Row Algorithm

The algorithm to check the space between 2 points along a row is illustrated in the following flowchart:

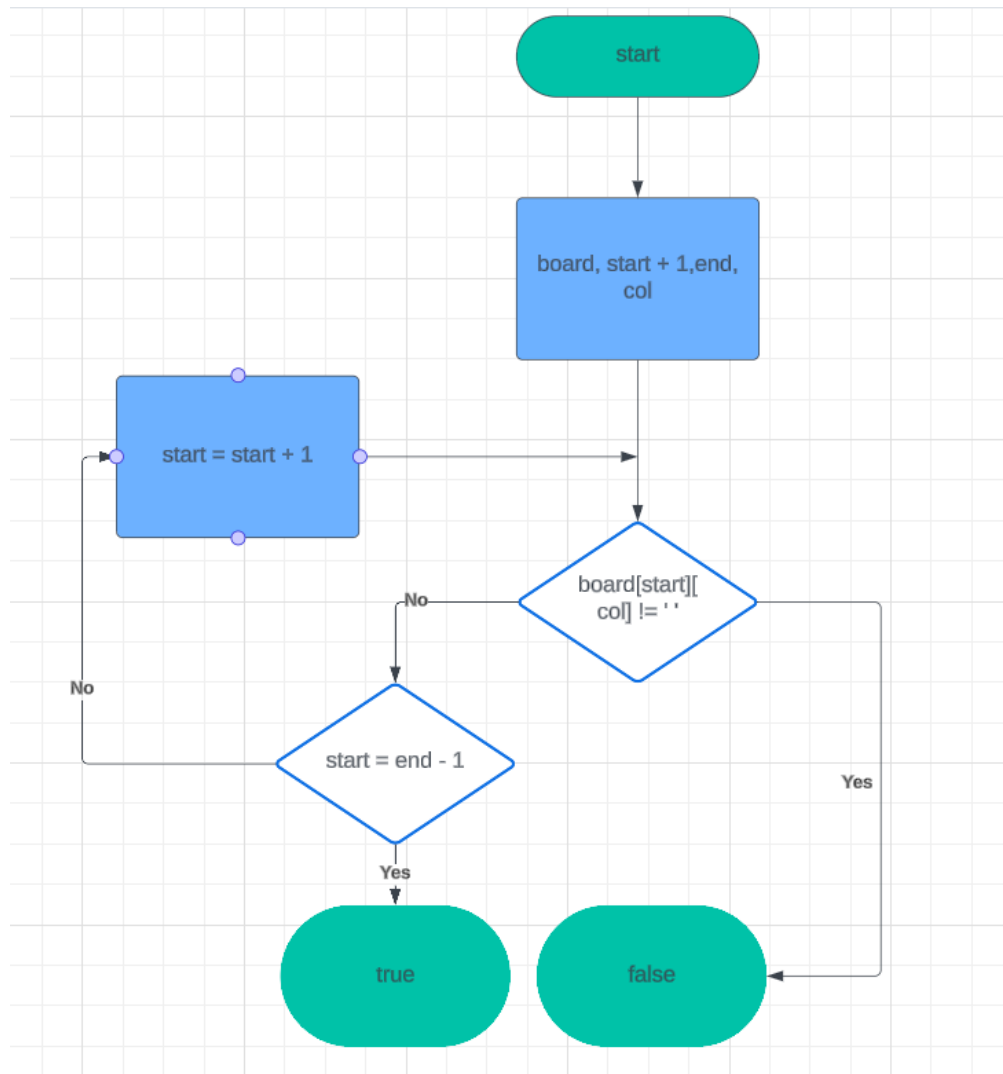


Figure 25: Check row function

### 2.2.2 I Matching

Combining the two functions above, we can check the possible cases for connecting two points along the straight line of the letter 'I':

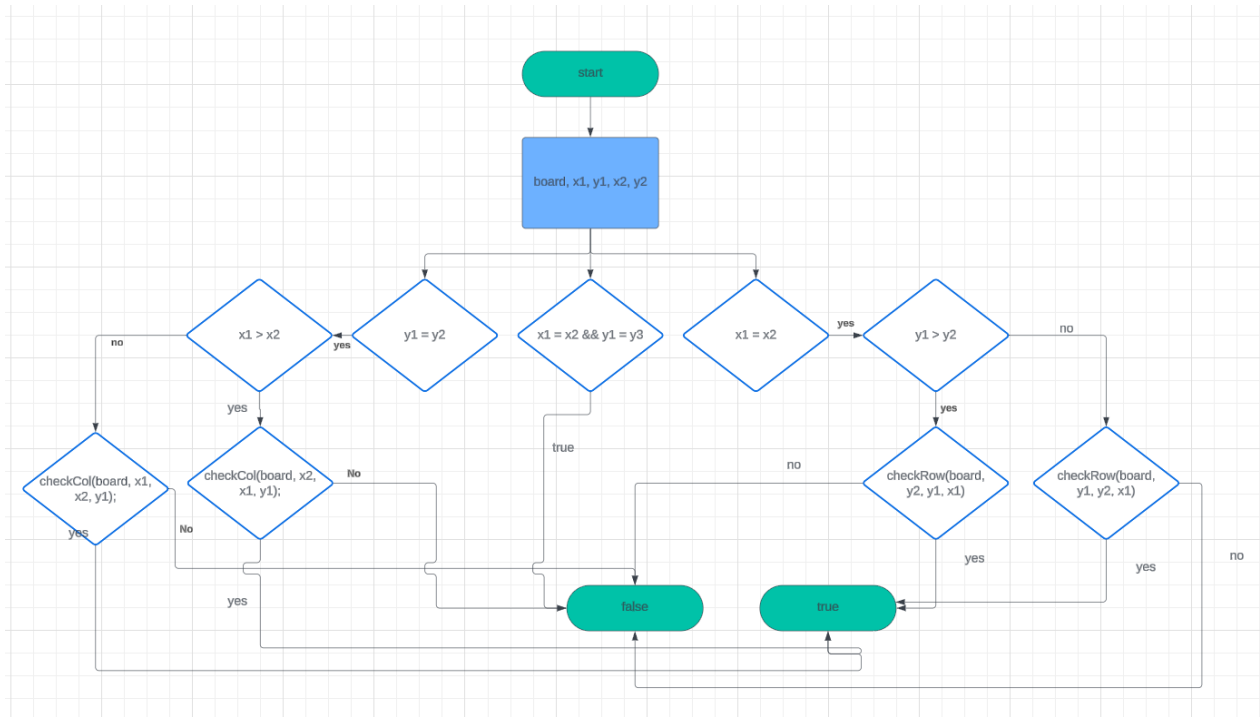


Figure 26: CheckI function

### 2.2.3 L Matching

In the L Matching algorithm, we have built the checkL function based on the results of the checkI function. This function is divided into three cases.

1. Case 1: This case eliminates situations where the shape of the path is not an L shape.

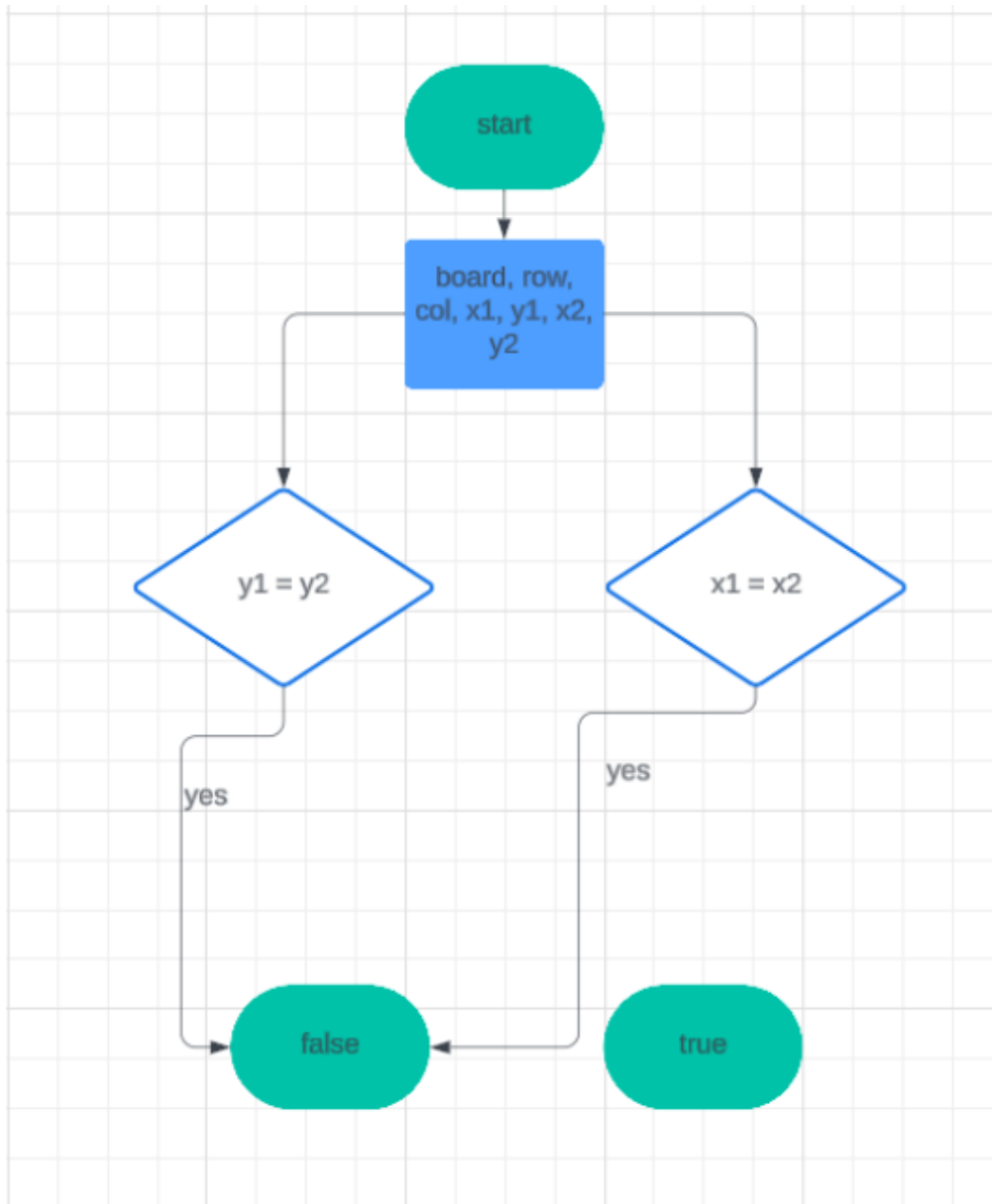
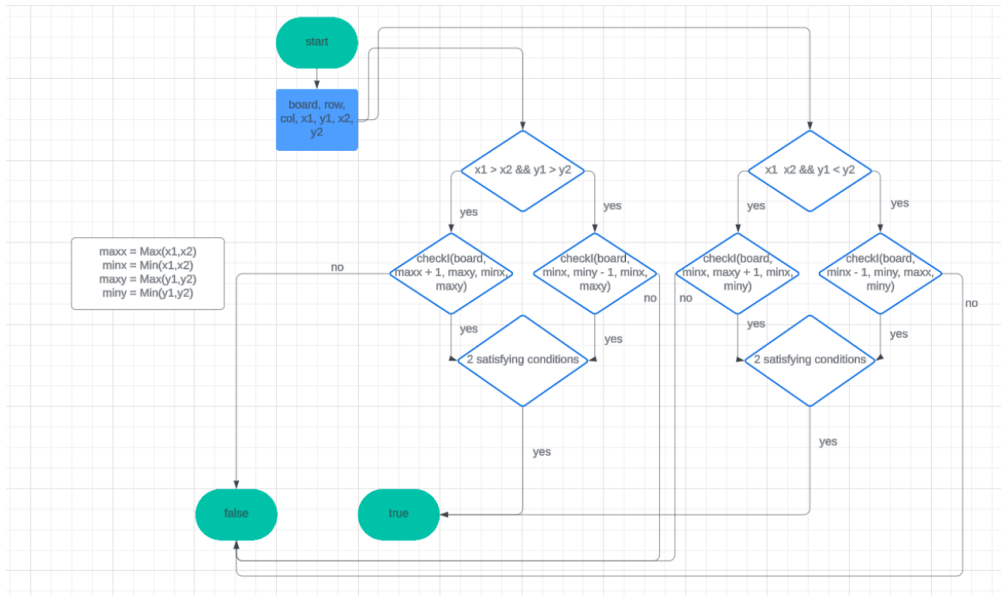
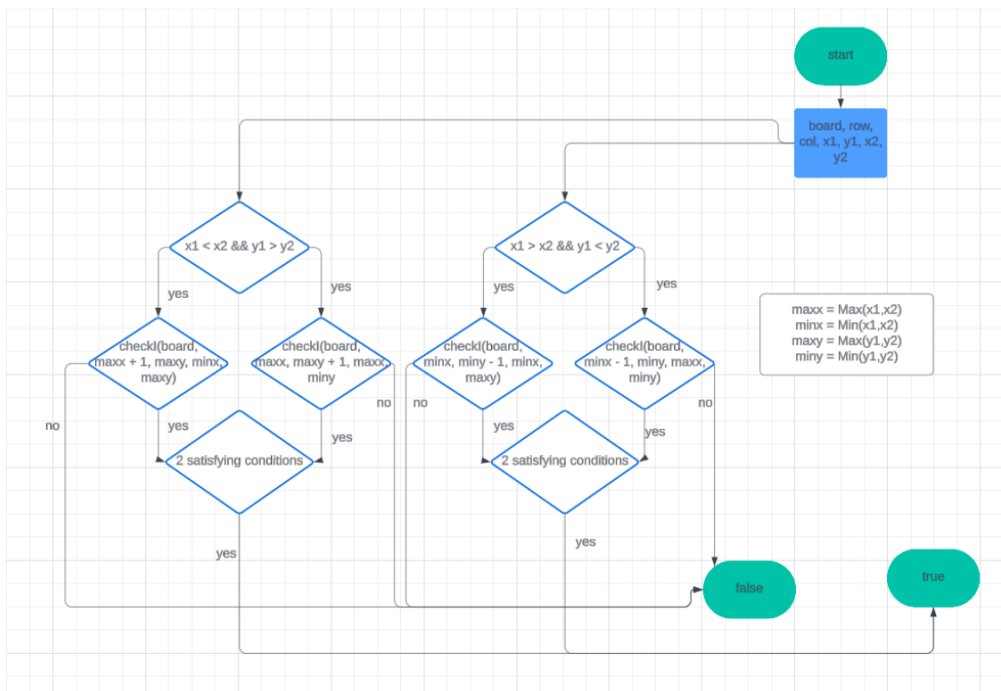


Figure 27: Case 1 in the checkL function

2. Case 2: This case checks the position of the two points, one at the top right corner and the other at the bottom left corner.



3. Case 3: This case checks the position of the two points, both at the top right and bottom left corners.





### 2.2.4 Z Matching

The Z Matching part utilizes the checkI and checkL functions to verify the connection between two points forming the shape of the letter 'Z'.

1. Case 1: This case checks two points corresponding to positions at the top left and bottom right.

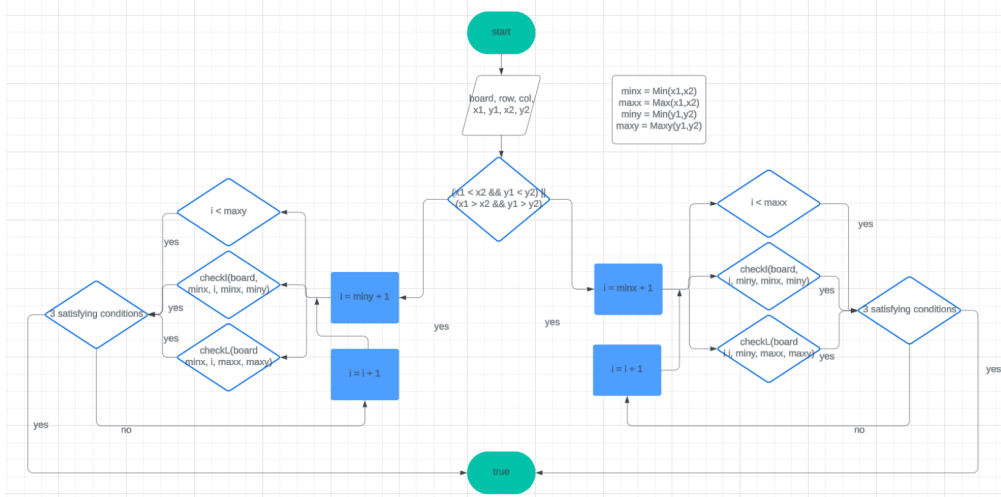


Figure 30: Case 1 in the checkZ function

2. Case 2: This case handles two points positioned at the top right and bottom left corners.

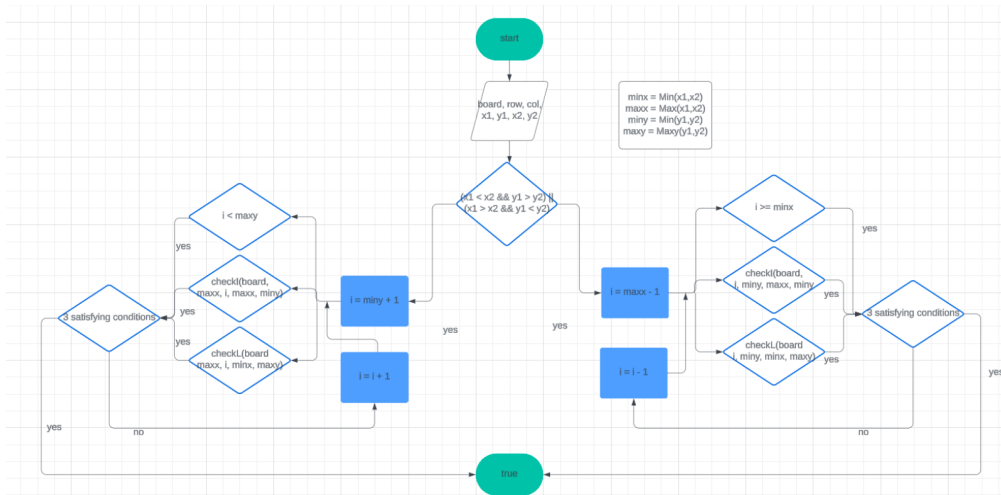


Figure 31: Case 2 in the checkZ function

### 2.2.5 U Matching

U Matching involves a combination of the checkI and checkL functions and has five separate cases.

1. Case 1: This case addresses situations where two points are located on the same border within the board.

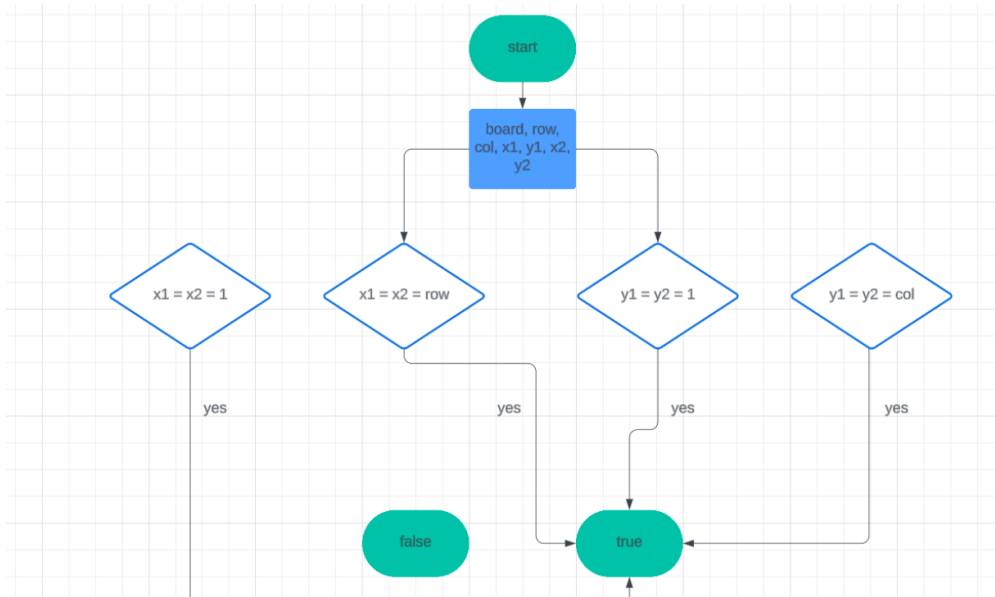


Figure 32: Case 1 in the checkU function

2. Case 2: This case addresses situations where the x-coordinates of the two points are the same.

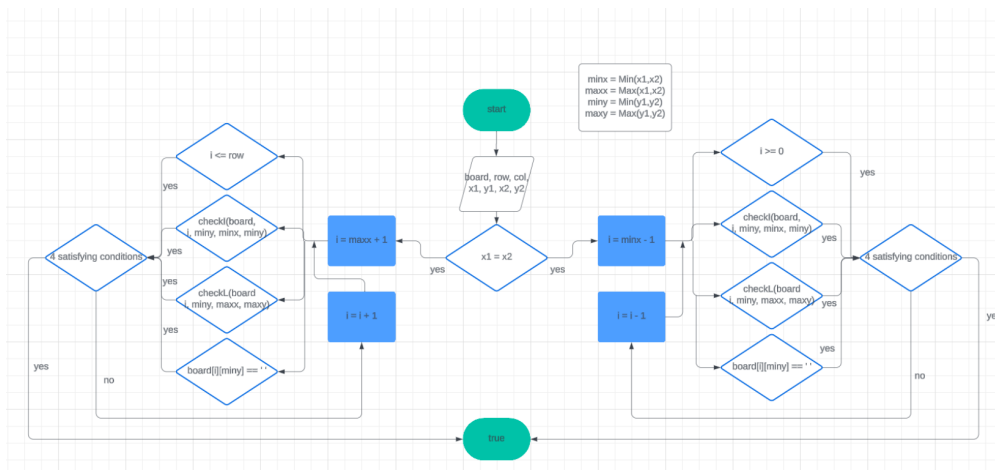


Figure 33: Case 2 in the checkU function

3. Case 3: This case addresses situations where the y-coordinates of the two points are the same.

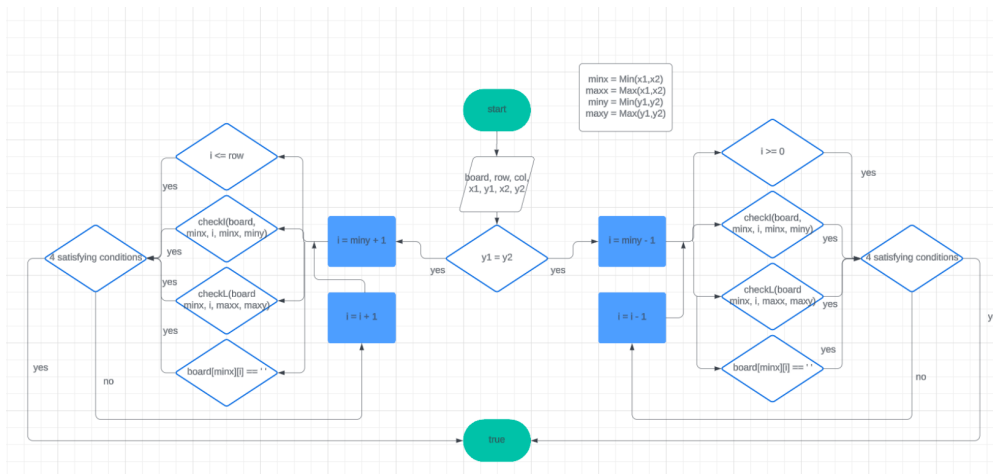


Figure 34: Case 3 in the checkU function

4. Case 4: This case addresses situations where one point is at the top left and the other is at the bottom right corner.

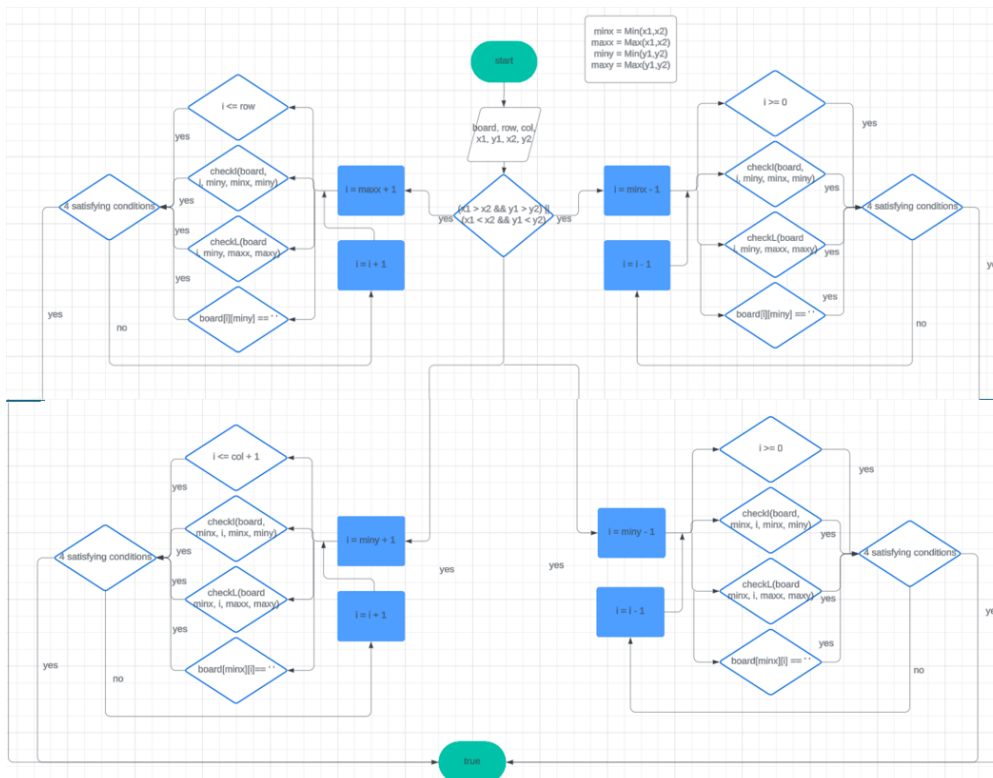
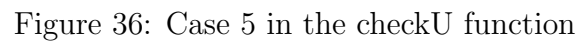


Figure 35: Case 4 in the checkU function

5. Case 5: This case addresses situations where one point is at the top right and the other is at the bottom left corner.



Possible positions of the two points:

- Figure 37: Top left and bottom right positions

- Page 26

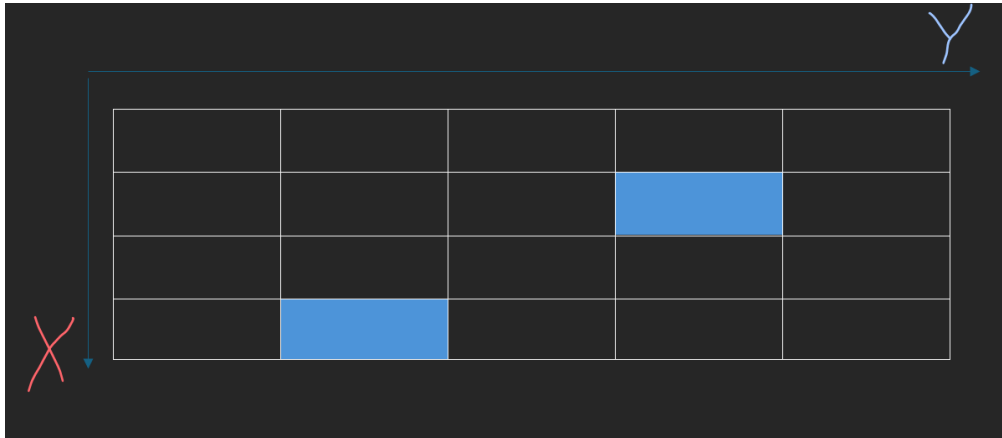


Figure 38: Top right and bottom left positions

3. The x-coordinate of the two points is the same

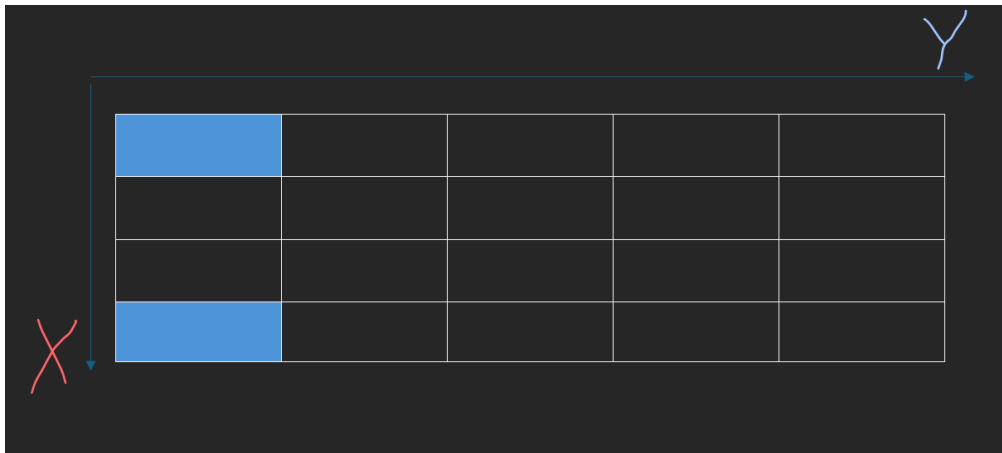


Figure 39: The x-coordinate of the two points is the same

4. The y-coordinate of the two points is the same

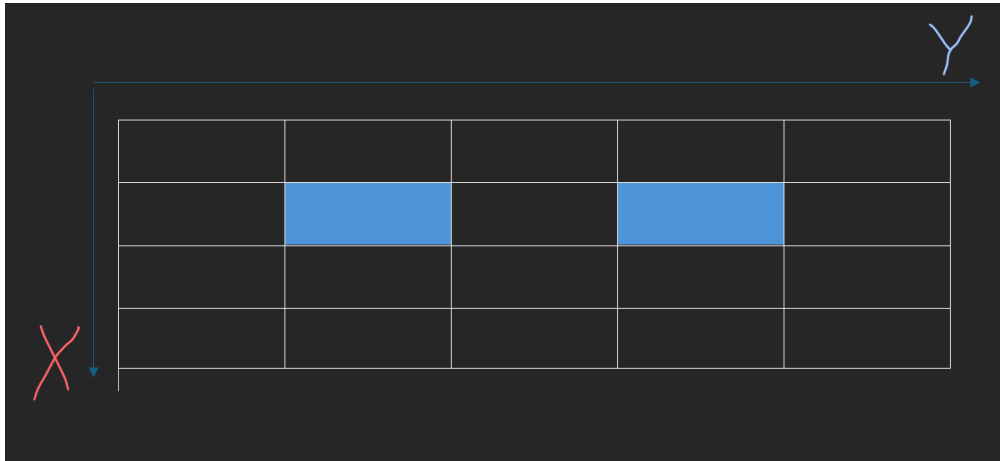


Figure 40: The y-coordinate of the two points is the same

5. The coordinates sequentially lie on two borders

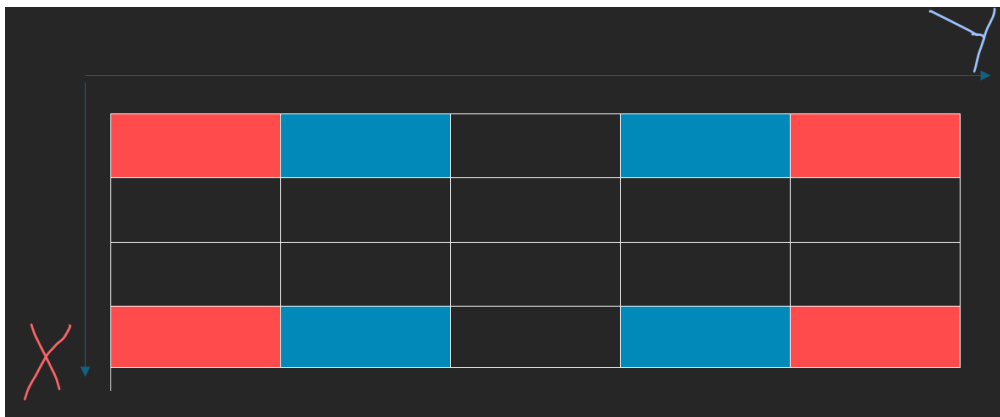


Figure 41: The coordinates lie on two borders

## 2.4 Data Structure Comparison

### 2.4.1 2-D Array Pointer

#### Advantages:

1. **Simple and easy to understand:** Using pointers to access cells in the game board is a simple and easy-to-understand approach.
2. **High performance:** Accessing cells in the board via pointers can achieve high performance because there is no need to traverse complex data structures like linked lists.
3. **Easy implementation of Matching algorithms:** Pointers make it easier to implement matching algorithms due to direct access to memory locations.

**Disadvantages:**

1. **Costly in terms of Matching checks:** Requires two nested loops each time, which can be costly in terms of performance.

**2.4.2 Linked List****Advantages:**

1. **Flexibility in adding/removing elements:** Linked lists provide high flexibility when adding or removing elements. Adding or removing an element does not affect other elements in the list.
2. **Cost-saving with only one loop needed to traverse all elements in the array:** Unlike arrays, linked lists only require one loop to traverse all elements, which can save computational cost.

**Disadvantages:**

1. **Requires traversing the entire list:** In some cases, when operations such as deleting elements are required, having to traverse the entire list can lead to inefficient performance, especially with large game boards.
2. **Costly in terms of storage space because additional nodes are needed to link between elements:** Linked lists require additional memory overhead to store pointers linking between elements, which can lead to higher storage requirements compared to arrays.
3. **High complexity in Matching check algorithms:** Implementing matching check algorithms with linked lists can be more complex due to the need for traversing and manipulating pointers.

**2.5 Advanced Feature****2.5.1 Color and Sound Effect**

In setting.cpp, we have a function to set color for text.

---

**Algorithm 3** Set text color using the specified number

---

```
function TEXTCOLOR(number):
    color ← Get handle to console output
    Set text attribute to the specified number
```

---

And here is some function to play sound effect and song.

Playing theme song for the menu.

---

**Algorithm 4** Play theme song

---

**function** `PLAYSOUND`

Playing file MarioThemeSong.wav

Playing theme song concurrently with other function when in the main menu

    Theme song will be replayed when it's finished

---

Playing sound effect when moving up or down or select the level in level list.

---

**Algorithm 5** Play sound effect

---

**function** `PLAYSOUNDEFFECT`

Playing file SoundEffect.wav

    Playing sound effect concurrently with other function when in the level list

---

Paying sound to congratulate player after player has cleared all cells in the board.

---

**Algorithm 6** Play sound to congratulate player

---

**function** `PLAYSOUNDWINNER`

Playing file WinnerSong.wav

    Play sound parallel

---

### 2.5.2 Visual Effect

We have implemented a visual effects system to highlight the current cursor position and the points selected by the player during the game. The system uses variables  $px$  and  $py$  to represent the cursor position, and  $x1$ ,  $y1$ ,  $x2$ ,  $y2$  to represent the positions of the selected points.

---

**Algorithm 7** Check conditions and print based on colors (in MakeBoard function)

---

```

1: if ( $i == x1$  and  $j == y1$ ) or ( $i == x2$  and  $j == y2$ ) then
2:   cout << " "
3:   MakeColor(a[i][j]);
4:   cout << "|";
5:   textColor(0x30);
6:   cout << " ";
7:   MakeColor(a[i][j]);
8:   cout << "|";
9: else if ( $i == px$  and  $j == py$ ) then
10:  cout << " "
11:  MakeColor(a[i][j]);
12:  cout << "|";
13:  textColor(0x70);
14:  cout << " ";
15:  MakeColor(a[i][j]);
16:  cout << "|";

```

---



The visual effects are applied as follows:

- If an element in the array is directly under the cursor  $(px, py)$ , it is filled with white color.
- If an element in the array is one of the selected points  $(x1, y1)$  or  $(x2, y2)$ , it is filled with green color.

Below is the pseudocode snippet illustrating the implementation of these visual effects in the `MakeBoard` function of the `Board.cpp` file:

### 2.5.3 Background

We use a function to set the cursor's position on console to draw background at specified places.

---

**Algorithm 8** Move cursor to the specified position  $(x, y)$

---

**function** `GOTOXY`( $x, y$ )

    Declare a handle to the console output

    Create a `COORD` structure with the specified position  $(x, y)$

    Get the handle to the console output

    Set the cursor's position to the specified coordinate

---

And the background we draw in each level.

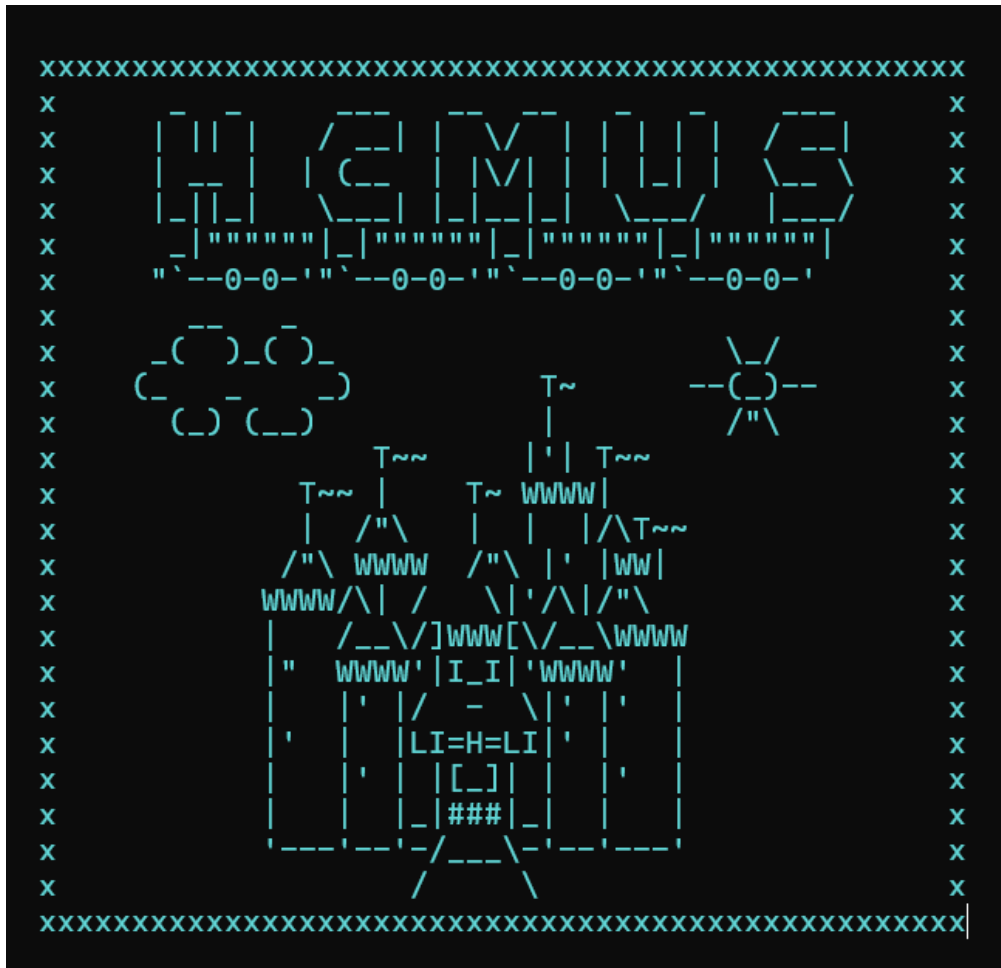


Figure 42: Background

#### 2.5.4 Move Suggestion

With the aim of making the game more accessible for everyone to experience, we have created a function to guide the next move for the player in cases where there is no immediate solution available.

And with that goal in mind, we have devised an algorithm to run suggestions taken from the first character, and from there, iterate through the remaining elements. If any element meets the condition, it can return those two elements. We will provide a clearer explanation through the pseudocode below:

---

**Algorithm 9** moveSuggestion which provides a hint for the next move in the game.

---

```

1: function MOVE_SUGGESTION (a, row, col, px, py, x1, y1, x2, y2)
2:   for i  $\leftarrow$  1 to 8 do
3:     for j  $\leftarrow$  1 to 8 do
4:       for k  $\leftarrow$  j + 1 to 8 do
5:         if a[i][j] = a[i][k] and a[i][j]  $\neq$  ' ' then
6:           if checkI(a, row, col, px, py, i, j, i, k) or checkZ(a, row, col, px, py, i, j,
           i, k) or checkL(a, row, col, px, py, i, j, i, k) or checkU(a, row, col, px, py, i, j, i, k) then
7:             x1  $\leftarrow$  i, y1  $\leftarrow$  j, x2  $\leftarrow$  i, y2  $\leftarrow$  k
8:             MakeBoard(a, row, col, px, py, x1, y1, x2, y2)
9:             return
10:        if i < 9 then
11:          for m  $\leftarrow$  i + 1 to 8 do
12:            for n  $\leftarrow$  1 to 8 do
13:              if a[i][j] = a[m][n] and a[i][j]  $\neq$  ' ' then
14:                if checkI(a, row, col, px, py, i, j, m, n) or checkZ(a, row, col, px,
                py, i, j, m, n) or checkL(a, row, col, px, py, i, j, m, n) or checkU(a, row, col, px, py, i, j,
                m, n) then
15:                  x1  $\leftarrow$  i, y1  $\leftarrow$  j, x2  $\leftarrow$  m, y2  $\leftarrow$  n
16:                  MakeBoard(a, row, col, px, py, x1, y1, x2, y2)
17:                  return

```

---

### 2.5.5 Leaderboard

First, when you select the option Leaderboard in the main menu, the game will read the file leaderboard.txt and print out only 10 highest score players.

---

**Algorithm 10** Read leaderboard data from file

---

```

function READLEADERBOARD
    Open file "Text/leaderboard.txt" for reading
    if file not opened successfully then
        Output "Cannot open leaderboard file"
        return
    Initialize numPlayers to 0
    Initialize temp string
    while ((read line from file into temp) and (numPlayers < Max_Players)) do
        if temp is empty then
            Continue to next iteration of loop
        Find position of '|' in temp
        if | not found then
            Continue to next iteration of loop
        Extract player's name from temp up to '|'
        Erase extracted part from temp
        Find position of '|' in temp
        if | not found then
            Continue to next iteration of loop
        Extract player's score from temp up to '|'
        Erase extracted part from temp
        Convert extracted score to integer
        Extract player's level from temp
        Increment numPlayers
    Close the file

```

---



---

**Algorithm 11** Print leaderboard to console

---

```

function PRINTLEADERBOARD
    for  $i \leftarrow 0$  to numPlayers do
        if  $i \geq 10$  then
            break
        goToXY(56, 8 +  $i$ )
        Output leaderboard[ $i$ ].name
        goToXY(83, 8 +  $i$ )
        Output leaderboard[ $i$ ].level
        goToXY(108, 8 +  $i$ )
        Output leaderboard[ $i$ ].score

```

---

When you finished the level, we will also using update function to add new player to the leaderboard and sort all players' score in descending order.

If the player is not in the file leaderboard.txt. It will add to the list. If not, we will find the same player playing the same level to change the score.

Here is the sort for leaderboard.

---

**Algorithm 12** Sort leaderboard by score

---

```

function SORTLEADERBOARD
  for  $i \leftarrow \text{numPlayers} - 1$  downto 1 do
    for  $j \leftarrow 0$  to  $i$  do
      if  $\text{leaderboard}[j].\text{score} < \text{leaderboard}[j + 1].\text{score}$  then
        Swap  $\text{leaderboard}[j]$  with  $\text{leaderboard}[j + 1]$ 

```

---

Function to input player's information when player start playing a level.

---

**Algorithm 13** Input player information

---

```

function INPUTPLAYERINFO(selectedLevel)
  goToXY(70, 21)
  Output "Please enter the name under 20 characters"
  goToXY(70, 20)
  Output "Enter your name: "
  Read input into newPlayer.name
  while length of newPlayer.name > 20 or newPlayer.name = "" do
    Clear screen
    goToXY(70, 22)
    Output "Invalid length"
    goToXY(70, 21)
    Output "Please enter the name under 20 characters"
    goToXY(70, 20)
    Output "Enter your name: "
    Read input into newPlayer.name
  Set newPlayer.level to selectedLevel
  Set newPlayer.score to 0

```

---

Update the leaderboard after input information.

---

**Algorithm 14** Update leaderboard with new player information

---

```

function UPDATELEADERBOARD(newPlayerInfo)
  // Check if new player's name is similar to any player's name in the leaderboard
  foundName  $\leftarrow$  false
  for  $i \leftarrow 0$  to numPlayers do
    if ( $\text{leaderboard}[i].\text{name} = \text{newPlayerInfo.name}$ ) and ( $\text{leaderboard}[i].\text{level} =$ 
    newPlayerInfo.level) then
      foundName  $\leftarrow$  true
      if  $\text{newPlayerInfo.score} > \text{leaderboard}[i].\text{score}$  then
         $\text{leaderboard}[i].\text{score} \leftarrow \text{newPlayerInfo.score}$ 
      break
  if not foundName and numPlayers < Max_Players then
     $\text{leaderboard}[\text{numPlayers}++] \leftarrow \text{newPlayerInfo}$ 
  sortLeaderboard()

```

---

Finally, we rewrite the file leaderboard.txt with the new version of leaderboard list.

---

**Algorithm 15** Write leaderboard data to file
 

---

```

function WRITELEADERBOARD
  readLeaderboard()
  updateLeaderboard(newPlayer)
  Open file "Text/leaderboard.txt" for writing
  if file not opened successfully then
    Output "Cannot open file leaderboard.txt for writing"
    return
  for  $i \leftarrow 0$  to numPlayers  $- 1$  do
    Write leaderboard[ $i$ ].name, "|", leaderboard[ $i$ ].score, "|", leaderboard[ $i$ ].level to file
    if  $i < \text{numPlayers} - 1$  then
      Write newline to file
  Close the file

```

---

### 2.5.6 Game Account

Player can enter name to login, the game will save your information in a file leaderboard.txt. Function login will be started whenever you choose a level to play. After your victory or exiting that level, your name, your score and the level's number will be save in leaderboard.txt. (You can see in details in [Input player's information](#))

## 2.6 Extra Advanced Features

In the "Extra Features" section, we have built three new features based on player experience to enhance the gameplay.

1. **Undo:** Undo can be understood as a feature used to save the steps that the player has taken during the game. To implement this Undo feature, we have created a three-dimensional pointer to store the player's moves. From there, the player can revert back to the moves they have made.
  - The player can execute this Undo command by pressing the "r" or "R" key to revert to the previous move.
  - The operation of this Undo function is to store all the states on the game board from the beginning of the player's gameplay. When the player presses 'r' or 'R', the three-dimensional pointer will assign the previous board back to the current board, and then print out that board. This process is performed according to the number of times the player presses the Undo function.
2. **Delete by Letter:** In this feature, we allow players to delete cells with letters corresponding to the letter they want to delete.

- Players can activate this feature by pressing the 'i' or 'I' key, and then the player needs to enter the letter they want to delete.
  - The operation of this feature is that when the player enters the letter they want to delete, we iterate through all the elements in the array and then search for that corresponding letter in the board and delete it. After that, we return the board with all elements corresponding to that letter deleted.
3. **Shrink Empty Cells:** The final feature is the function to shrink empty cells towards the center. This feature will be integrated into our hard mode gameplay to increase the difficulty of the game further.
- The operation of this feature will involve 2 functions working together corresponding to columns and rows. Initially, both functions need to check if there are any empty cells, and from there, they can move the elements from the outer edges towards the center.
  - There is a warning in this gameplay section that it may lead to unwinnable game scenarios, specifically where elements cannot be connected. In such situations, if players wish to win, they must resort to using our features.

## 3 File Structures

### 3.1 Header Files

1. Setting.h: To declare functions for decorating
  - textColor: Set color for text
  - goToXY: Set cursor's position
2. Menu.h: To declare functions in menu and set up levels
  - (a) Const
    - List: maximum members in the list: Menu list, Level list and Continue list
    - Color: color code for text
    - Basic key: key to interact in the game
    - Maximum line in leaderboard.txt
  - (b) Function showMenu: to start the game from the main menu
  - (c) Main function in the main menu
    - showLevel (Play button): show a list of level
    - showLeaderboard (Leaderboard button): show list of leaderboard
    - showTutorial (Help button): show game tutorial
    - showIntroduction (About button): show some information of project
  - (d) Function to play music

- playSound: play theme song for the main menu
  - playWinnerSong: play song for winner player
  - playSoundEffect: play sound effect when selecting levels
- (e) Functions in leaderboard (You can see in details in [Leaderboard](#)).
- (f) Functions to show levels
- easyLevel: show Easy level
  - normalLevel: show Normal level
  - hardLevel: show Hard level
3. Check.h: To declare functions to set rules for matching
- (a) Check rows and columns for matching
- checkCol: check the elements between the 2 selected cells in column.
  - checkRow: check the element between the 2 selected cells in row.
  - checkEmptyCol: check the column is empty or not.
  - checkEmptyRow: check the row is empty or not.
- (b) Check matching (You can see in details in [Matching Function](#))
- checkI: check for I Matching.
  - checkL: check for L Matching.
  - checkU: check for U Matching.
  - checkZ: check for Z Matching.
- (c) Delete
- deleteLineRow
  - deleteLineCol
  - deleteCell: delete cell is valid matched.
  - DeleteElementFocus: delete all cells that have specified character that player input when using feature Delete By Letter.
  - deleteI
  - deallocate
- For Normal level and Hard level
- DelEmptyRowDown: delete empty row and move down.
  - DelEmptyRowUp: delete empty row and move up.
  - DelEmptyColLeft: Delete empty column and move left.
  - DelEmptyColRight: Delete empty column and move right.
- (d) Slide cells after matching
- SortEmptyRow: Move row up and down to the center.
  - SortEmptyCol: Move column left and right to the center.



4. Board.h: To declare functions for draw game board and movement (You can see in details in [Game Board](#))
  - (a) Display board
    - MakeColor: set color for a specified character
    - MakeBoard: draw board, scoreboard and background.
    - genBoard: generate values for each cell in the board
  - (b) Undo board
    - CreateUndoBoard: create a board to store movements.
    - GenUndoBoard
    - GenUndoBoardToMainBoard: reverse the last movements back.
    - deallocateUndo
  - (c) Movement for each level
    - MoveBoardForEasy: movement for Easy level.
    - MoveBoardForNormal: movement for Normal level.
    - MoveBoardForHard: movement for Hard level

### 3.2 Source Files

1. Setting.cpp: Define all functions are declared in Setting.h.
2. Menu.cpp: Define all functions are declared in Menu.h.
3. Check.cpp: Define all functions are declared in Check.h.
4. Board.cpp: Define all functions are declared in Board.h.
5. Main.cpp: Run the game.

### 3.3 Resource Files

1. leaderboard.txt: Save players' information in leaderboard.
2. MarioThemeSong.wav: The song played in the main menu.
3. SoundEffect.wav: The sound played in list of level.
4. WinnerSong.wav: The song played in winner session.

## References

- [1] Loius2602. File medium.txt, from line 1 - 25. [Online]. Available: <https://github.com/Louis2602/Pikachu-Game/blob/master/Pikachu/images/medium.txt>
  - [2] T. T. Nguyen. From line 16 - 30. [Online]. Available: <http://codepad.org/VyqVTZtS>
  - [3] Loius2602. File board.cpp, from line 7 - 150, 174 - 218. [Online]. Available: <https://github.com/thuanphat611/Matching-Game/blob/master/Source/board.cpp>
- (1) (2) (3)