

PROJECT REPORT: MATCHING GAME

K21CLC06

21127009 - Trần Minh Anh

21127021 - Trương Văn Chí

Instructor:

Nguyễn Thanh Phương

Bùi Huy Thông

Nguyễn Ngọc Thảo

Table of contents:

Table of contents	1
List of images	2
List of tables	3
1 Tutorial	4
1.1 Game state:	5
1.2 Game utilities:	6
1.2.1 Play	6
1.2.2 Custom play	8
1.2.3 Leaderboard	8
1.2.4 Exit	9
2 Functional	10
2.1 File structure	11
2.2 Functions	13
2.3 Data structures	23
3 Comparison	25
4 Advanced Features	26
4.1 Color effects	27
4.2 Visual effects	28
4.3 Leaderboard	30
4.4 Move suggestion	31
5 References	32

List of images:

1.1.1 Game state	5
1.1.2 Main menu	5
1.2.3 Choosing a mode	6
1.2.4 Highscore informing box	6
1.2.5 Enter name	7
1.2.6 Leaderboard	7
1.2.7 Enter rows and columns	8
1.2.8 Leaderboard	8
1.2.9 Exit screen	9
2.2.1 How the game works	13
4.1.1 Main menu	27
4.1.2 Matching check	27
4.2.3 Main menu	28
4.2.4 Mode	28
4.2.5 Gameplay	29
4.2.6 Exit screen	29
4.3.7 Leaderboard	30
4.4.8 Move suggestion	31

List of tables:

3.1	Pointer vs Linked List	25
-----	----------------------------------	----

Chapter 1

Tutorial

1. Game state
 - States of the game and what will be displayed on the screen.
2. Game utilities
 - What user can see and do with the game.

1.1 Game state:

This chart simplified how this program works.

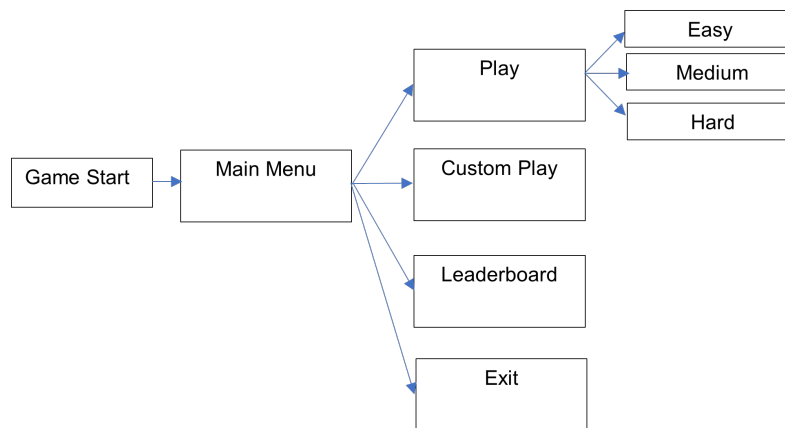


Image 1.1.1 Game state

This is the user's interface of the main menu.

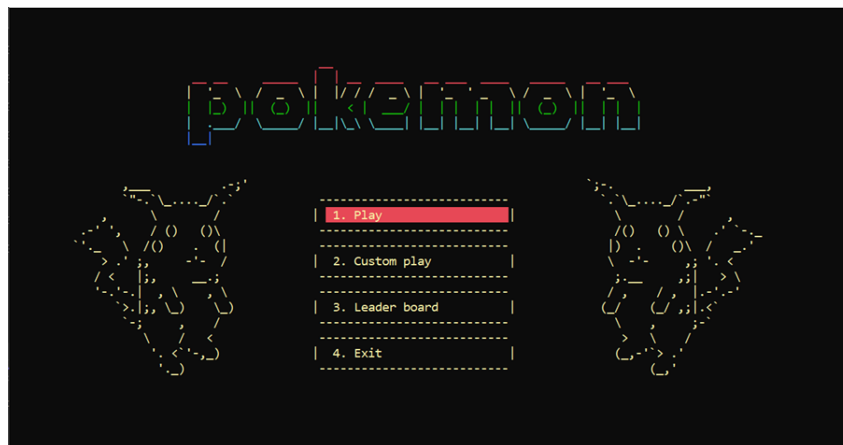


Image 1.1.2 Main menu

1.2 Game utilities:

1.2.1 Play

- Choosing difficulties, there are 3 modes: Easy, Medium, Hard.

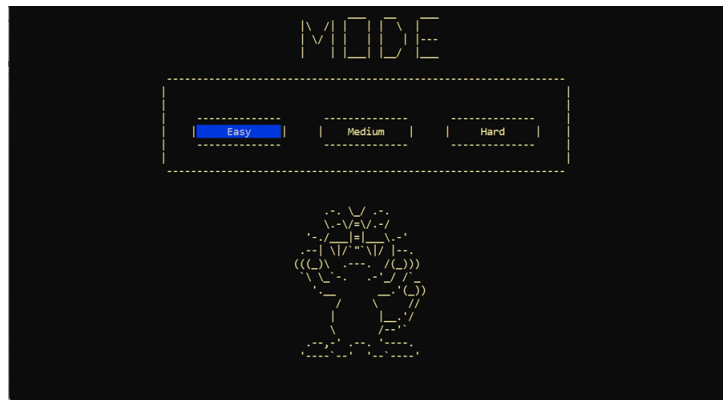


Image 1.2.3 Choosing a mode

- After playing, if the user got a high score (in each mode), there will be a board informing that you can be on the leaderboard.

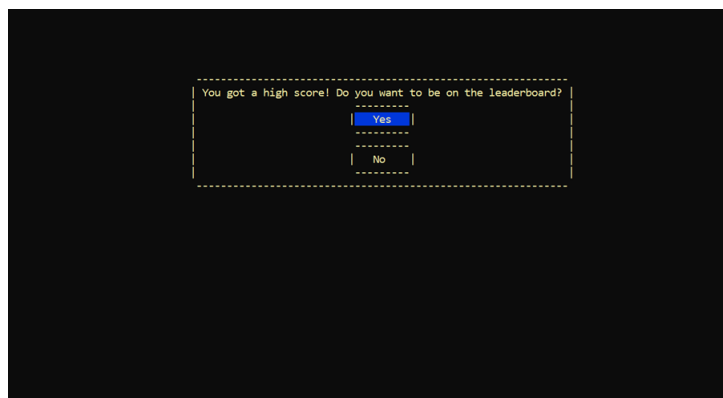


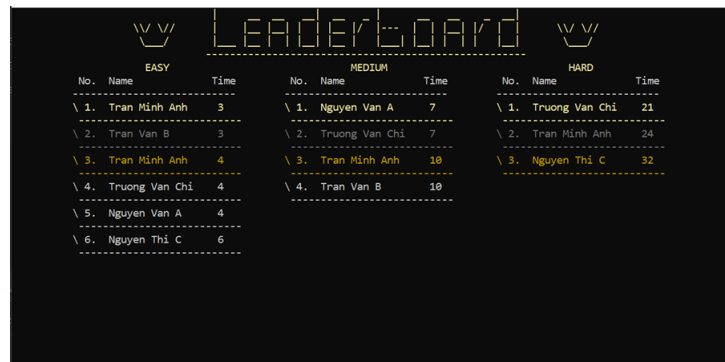
Image 1.2.4 Highscore informing box



Image 1.2.5 Enter name

- If the user presses Yes, the screen will move to a section where the user can enter his/her name.

- After entering the name, a leaderboard will be displayed and the user's name will be on the board (depending on what mode the users chose, their name will be on that mode's leaderboard).



EASY			MEDIUM			HARD		
No.	Name	Time	No.	Name	Time	No.	Name	Time
\ 1.	Tran Minh Anh	3	\ 1.	Nguyen Van A	7	\ 1.	Truong Van Chi	21
\ 2.	Tran Van B	3	\ 2.	Truong Van Chi	7	\ 2.	Tran Minh Anh	24
\ 3.	Tran Minh Anh	4	\ 3.	Tran Minh Anh	10	\ 3.	Nguyen Thi C	32
\ 4.	Truong Van Chi	4	\ 4.	Tran Van B	10			
\ 5.	Nguyen Van A	4						
\ 6.	Nguyen Thi C	6						

Image 1.2.6 Leaderboard

- If the user presses No, the program will bring them back to Main menu.

1.2.2 Custom play

- This mode will let the user create their own board. The number of rows and columns are entered by the user.

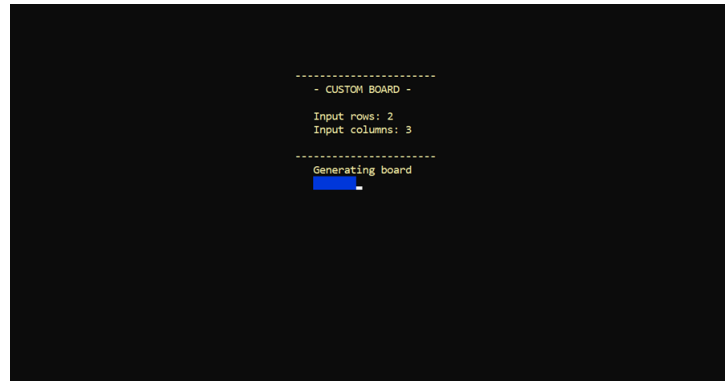


Image 1.2.7 Enter rows and columns

1.2.3 Leaderboard

- This option will show the leaderboard.

EASY			MEDIUM			HARD		
No.	Name	Time	No.	Name	Time	No.	Name	Time
\ 1.	Tran Minh Anh	3	\ 1.	Nguyen Van A	7	\ 1.	Truong Van Chi	21
\ 2.	Tran Van B	3	\ 2.	Truong Van Chi	7	\ 2.	Tran Minh Anh	24
\ 3.	Tran Minh Anh	4	\ 3.	Tran Minh Anh	10	\ 3.	Nguyen Thi C	32
\ 4.	Truong Van Chi	4	\ 4.	Tran Van B	10			
\ 5.	Nguyen Van A	4						
\ 6.	Nguyen Thi C	6						

Image 1.2.8 Leaderboard

1.2.4 Exit

- This option will show the exit screen and exit the game.

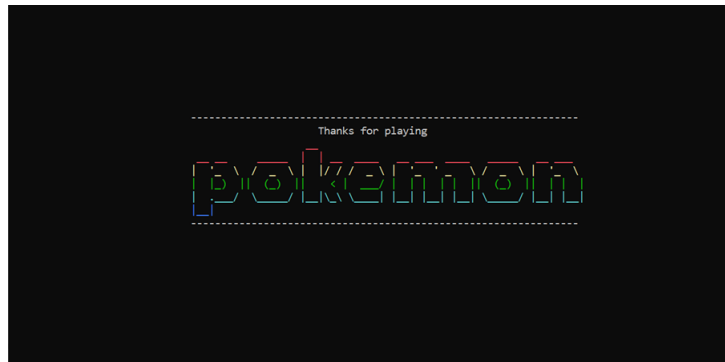


Image 1.2.9 Exit screen

Chapter 2

Functional

1. File structure
 - How we organize the files.
2. Functions
 - Explain how the functions work.
3. Data structures
 - What data structures we used
 - Some other functions provided by C++.

2.1 File structure

- We divided into files depending on the function of that file. For instance, drawConsole.h includes all the functions used to render boards, set color for text and background,...
- There are 6 main files, 1 .cpp file and 5 header files.

Pikachu.cpp

- > This is the source code of the program.
- > Libraries and Header files included: Game.h.

drawConsole.h

- > This header file includes all the functions used to manipulate the user interface, such as generate the board, generate the main menu, print the Ascii art,...
- > Libraries and Header files included: Window.h, iostream, fstream.

Game.h

- > This header file includes all the functions used to control the game, such as using AWSD to move, choosing the difficulties, playing the game,...
- > Libraries and Header files included: conio.h, ctime, string, drawConsole.h, gameFunction.h, leaderBoard.h.

gameFunction.h

- > This header file includes all the functions used to check the matching condition of the game, such as checking if the path is valid, suggesting a move, randomize the game if there is no valid path left,...
- > Libraries and Header files included: synchapi.h, Queue.h.

leaderBoard.h

- > This header file includes all the functions used to manipulate the leaderboard, such as adding data to leaderboard's file, printing the leaderboard,...
- > Libraries and Header files included: fstream, string.

Queue.h

→ This header file includes all the functions used to create queues using linked list.

→ Libraries and Header files included: fstream, string.

- The program also has some .txt files:

1) leaderBoardEasy.txt

→ save the data of the game in easy mode (user's name, play time).

2) leaderBoardMedium.txt

→ save the data of the game in medium mode (user's name, play time).

3) leaderBoardHard.txt

→ save the data of the game in hard mode (user's name, play time).

4) pikachu1.txt, pikachu2.txt, pikachu3.txt, pikachu4.txt

→ Pokemon Ascii Art.

2.2 Functions

- This chart is an overview of how this program works.

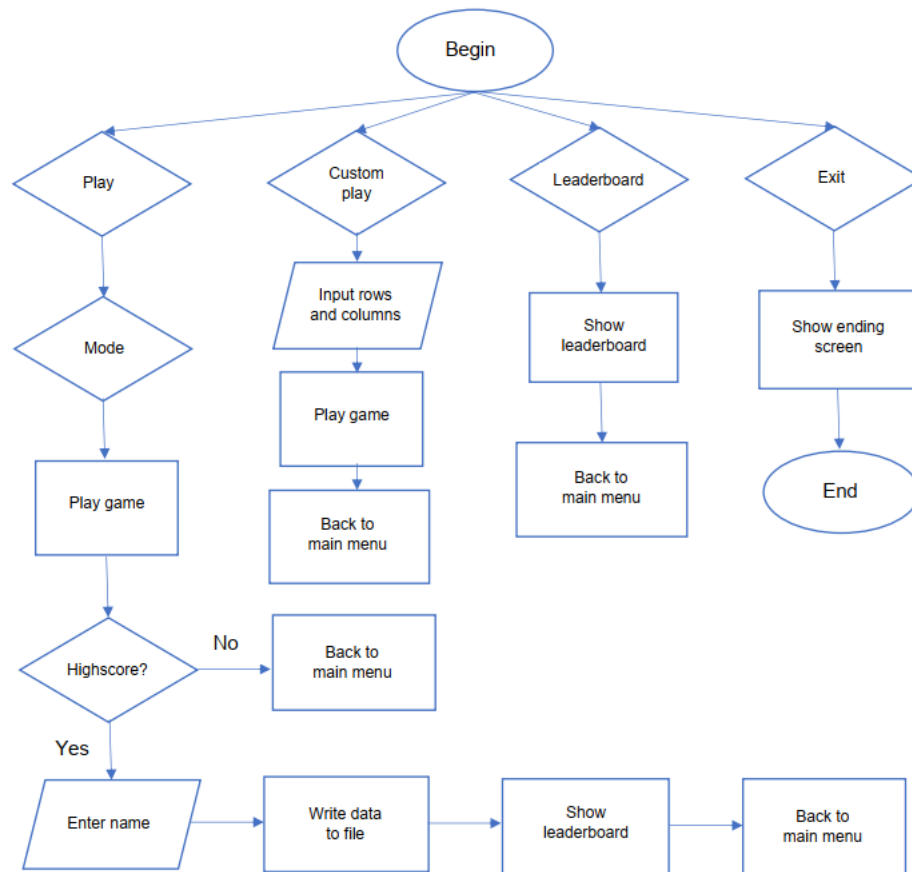


Image 2.2.1 How the game works

Pikachu.cpp

```
int getUserEnterPos(int& x)
```

→ Used to move up or down to choose one of the options in the main menu.

1. Get the character entered by the user.
2. Compare it to the given cases: w, W, s, S (if it is w or W → move upward, if it is s or S → move downward).
3. Press Enter to return the value of x, which is the Y coordinate.

```
void mainMenu()
```

→ Used to choose one of the options in the main menu.

1. There will be a base Y coordinate $j = 17$, then its value will be changed after calling `getUserEnterPos(j)`.
2. Compare the value of j to the cases (if $j == 17$ → Play game with modes chosen by the player, if $j == 20$ → Play custom game with the number of rows and columns are entered by the user, if $j == 23$ → Show the leaderboard, if $j == 26$ → Show the ending screen).

drawConsole.h

```
void ScreenSize(int x, int y)
```

→ Used to adjust the size of the screen.

```
void gotoXY(int x, int y)
```

→ Used to move the cursor to the desired position.

→ It will get the coordinate through the parameters of the function and then move the cursor to that coordinate of the screen.

```
void setColor(int color)
```

→ Used to change the color of the text or background.

```
void renderConsole(int x, int y, int k, int color)
```

→ Used to generate the playing board.

1. If $k \neq -1$, means this character hasn't been matched yet, a cell will be printed and the value of k will be put in the middle of the cell.
2. If $k == -1$, means character has already been matched, a set of whitespaces will be printed.

```
void renderPosMenu(int y, int color)
```

-> Used to generate the main menu.

-> It will get the coordinate through the parameters of the function and then the user can move up or down to choose an option by using W and S keys.

```
void renderChoiceMenu(int y, int color)
```

-> Used to generate the choice menu when the user got a highscore.

-> It will get the coordinate through the parameters of the function and then the user can move up or down to choose an option by using W and S keys.

```
void renderDifficulty(int y, int color)
```

-> Used to generate the mode board when the user chose Play option at the main menu.

-> It will get the coordinate through the parameters of the function and then the user can move left or right to choose an option by using A and D keys.

```
void Text_Animation(char a[100])
```

-> Used to create a text animation.

```
void MainMenu()
```

-> Used to print the user interface of the main menu.

```
void Pikachu(), void Mode_Image(), void Play_Image()
```

-> Used to print the Pokemon Ascii arts.

Game.h

```
int aKey(int** Pika, int n, int m, int x, int y)
```

→ Used to move the cursor when the user press a or A.

```
int sKey(int** Pika, int n, int m, int x, int y)
```

→ Used to move the cursor when the user press s or S.

```
int dKey(int** Pika, int n, int m, int x, int y)
```

→ Used to move the cursor when the user press d or D.

```
int wKey(int** Pika, int n, int m, int x, int y)
```

→ Used to move the cursor when the user press w or W.

```
Point getUserEnterPoint(int** Pika, int n, int m, int& i, int& j, Point s)
```

→ Used to get the user's entered key and then the cursor will be move up or down, left or right depends on what key the user entered. A coordinate of a point will be returned after the user pressed Enter.

1. Get the character entered by the user.
2. Compare it to the given cases: w, W, s, S, a, A, d, D (if it is w or W → move up, if it is s or S → move down, if it is a or A → move left, if it is d or D → move right). If it is h or H, moveSuggetion function will be called and it will show a hint.
3. After pressing Enter, the coordinate of the point the cursor standing at will be returned.

```
int getUserChoice(int& x)
```

→ Used to move up or down to choose one of the options in the highscore menu.

1. Get the character entered by the user.
2. Compare it to the cases: w, W, s, S (if it is w or W → move upward, if it is s or S → move downward).
3. While pos != 13, the choice menu will be rendered again.

4. Press Enter to return the value of x which is the Y coordinate and the option at that coordinate will be chosen.

```
int getDifficulty(int& x)
```

→ Used to move left or right to choose one of the options in the mode menu.

1. Get the character entered by the user.
2. Compare it to the cases: a, A, d, D (if it is a or A → move to the left, if it is d or D → move to the right).
3. While pos != 13, the mode menu will be rendered again.
4. Press Enter to return the value of x which is the X coordinate and the option at that coordinate will be chosen.

```
void GameMode(int &m, int &n, int &difficulty)
```

→ Used to print the game mode user interface and the parameters of the function will be assigned a constant value set by the programmer (the value depends on the mode chosen by the user).

```
void CustomPlay(int& m, int& n)
```

→ Used to print the custom game user interface and get the number of rows and columns entered by the user.

```
void playGame(int** Pika, int n, int m, int choice, int difficulty)
```

→ Used to play the game.

1. The board will be rendered on the screen.
2. Before starting the game, time_t data type is used to count the time
3. While not all cells are matched (end game condition):
 - First, the game will check if there are any valid paths left, if not the program will rerandomize and render the board again, else the program continues.
 - Get the user's first point and second point using getUserEntryPoint function.
 - The program will check if the two points have the same character and if there is a valid path connecting that two points. If all conditions above are

satisfied, those points' cells will be deleted, the value of the point will be -1 (which means whitespace).

4. If the user got a highscore (checked using `checkTopPlayer`), there will be a box informing that the user got a highscore. If the user chooses Yes, he/she can enter his/her name, the name and the playtime will be added to a file and later printed to the screen in the leaderboard. If the user chooses No, he/she will be brought back to the main menu. (Only applied on Play mode).

```
void game(int choice, int &difficulty)
```

→ Used to create a gameplay.

1. Check the if the user chose Play or Custom play, then the number of rows and column given by the function will be assigned to variable m and variable n respectively.
2. Randomize the matrix of characters.
3. `playGame` function is called to start the game.

```
void Ending()
```

→ Used to print the ending screen.

gameFunction.h

```
bool isInsideArr(Point point, int n, int m)
```

→ Used to check if the given point is in the playing board.

```
bool getPathValue(Point s, Point t, int** Pika, int n, int m)
```

→ Used to get the path connecting two points.

→ We used BFS for this function.

1. Initialize a two-dimensional array `subArr`: `subArr[i][j]` holds a number indicating how many times the path changes direction when going from point s to point `[i][j]`. At first, `subArr[i][j] = 4` in all cases of i and j.
2. Initialize a queue with a starting point s.
3. Queue checking. With each starting point u, we check 4 points surrounding it (vertically and horizontally) and call each point when examining as point v. If from the trace of u to v there isn't any direction change, the value of v will be

equal to the value of u , else the value of v will be equal to the value of u plus 1. Comparing if the value of v is greater or equal to $\text{subArr}[v.x][v.y]$, v will be added to a queue with the trace of v which is equal to u , else v won't be added to a queue.

4. After examining the queue, we check if $\text{subArr}[t.x][t.y]$ is lesser than 4, that means the number of times the direction changed when going from s to t is lesser than 4, return true, else return false.

- The benefit of using BFS: With only one function using BFS, we can check all 4 cases of matching. Whereas using arrays, we have to create 4 functions. The complexity of BFS is $O(4*m*n)$ while that of using arrays is $O(m*n)^2$.

```
void moveSuggestion(int** Pika, int n, int m, Point enterPoint)
```

→ Used to give a hint.

1. Use 4 loops to find a matching pair.
2. When the matching pair is found, the program will check if there is a valid path connecting that pair using `getPathValue` function.
3. If `getPathValue` returns true, the program will set the color of those cells' background to dark blue as a signal to a hint.

```
bool checkEndGame(int** Pika, int n, int m)
```

→ Used to check if all the cells are matched and there is no cell left.

→ Use 2 loops to check if all the matrix's elements' values are all -1, if they are return true, else return false.

```
bool checkNotPath(int** Pika, int n, int m)
```

→ Used to check if there is a valid path.

1. Use 4 loops to find a matching pair.
2. When the matching pair is found, the program will check if there is a valid path connecting that pair using `getPathValue` function.
3. If there is no valid path left, `checkNotPath` function will return true, else return false.

```
void randomGame(int** Pika, int n, int m)
```

→ Used to rerandomize the game (this function is used when there is no valid path left).

leaderBoard.h

```
struct LeaderBoardType
```

→ This type will hold the data of the player (time, name).

```
void getDataLeaderBoard(LeaderBoardType* leaderBoardArr, int &count,  
int difficulty)
```

→ Used to get the data from files.

→ Depending on what the difficulty is, the data from that difficulty's leaderboard file will be read and assigned to leaderBoardArr.

```
void writeDataLeaderBoard(LeaderBoardType* leaderBoardArr, int  
&count, int difficulty)
```

→ Used to write the data to files.

→ Depending on what the difficulty is, the data will be written to that difficulty's leaderboard file.

```
void printLeaderBoard()
```

→ Used to print the data from all 3 leaderboard files to the screen.

```
void addLeaderBoard(LeaderBoardType user, int difficulty)
```

→ Used to add data of new highscorers to files.

1. Get the data from the leaderboard file.
2. There will be a insertion sort to find the rank of the player.
3. Write the new highscorer's data to the file.
4. Print the leaderboard.

```
bool checkTopPlayer(double totalTime, int difficulty)
```

→ Used to check if the player got a highscore.

Queue.h*** Queue of Points:**

`struct Point`

→ Holds the coordinate of a point.

`struct Path`

→ Holds the coordinates, value of the path and the trace.

`struct Node`

→ Holds the data of a node in a linked list.

`struct list`

→ Holds the data of the head and the tail in a linked list.

`Node* createNode(Path value)`

→ Used to create a new node in a linked list.

`struct Queue`

→ Holds the data of a queue.

`bool Queue::empty()`

→ Used to check if the queue is empty.

`void Queue::push(Path value)`

→ Used to insert a new node at the end of the queue.

`Path Queue::front()`

→ Used to get the key value of the node in the front of the queue.

`void Queue::pop()`

→ Used to delete a node.

*** Queue of Integers:**

```
struct NodeInt
struct QueueInt
bool QueueInt::empty()
void QueueInt::push(int value)
int QueueInt::front()
void QueueInt::pop()
```

→ These structures and functions are the same as the above except instead of a queue of points, these make a queue of integers.

2.3 Data structures

- Data structures used in this program are arrays (used to hold the value of the cells), linked list and queue (used to find the path to two cells).

* Library: <Window.h>

HANDLE hStdin;

HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);

COORD CursorPosition;

-> Used to handle the cursor position on the screen.

-> If not using this, it will be hard to control the position of the cursor.

SetConsoleScreenBufferSize(console, CursorPosition);

SetConsoleWindowInfo(console, TRUE, Rect);

-> Used to change the size of the console.

-> If not using this, there will be a bug when the board is too big compares to the screen.

SetConsoleCursorPosition(console, CursorPosition);

-> Used to put cursor to a desired position.

-> If not using this, we can't set the cursor to the position that we want.

SetConsoleTextAttribute(console, color);

-> Used to add color to text and background.

-> If not using this, there will be no color effect.

* Library: <conio.h>

_getch()

-> Used to get a character entered by the user.

-> If not using this, the user will not be able to control the cursor using a keyboard.

* Library: `<ctime>`

`time_t`

→ Used to calculate the time.

→ If not using this, we can't count the time of a gameplay.

`rand()`

→ Used to get a random number.

→ If not using this, we can't randomize the board.

Chapter 3

Comparison

Pointer vs Linked List

Function	Pointer	Linked list
Generate board	Can access to the desired element directly → Because elements in arrays can be randomly accessed.	Need a for-loop to access to the desired element → Because elements in linked list can only be accessed sequentially.
Check matching	Using arrays to check matching. Needed 4 functions for 4 cases of matching. Complexity: $O(m*n)^2$.	Using BFS to check matching. Only needed 1 function for all 4 cases of matching. Complexity: $O(4*m*n)$.

Table 3.1: Pointer vs Linked List

* Conclusion:

- Pointers are better when random accessing to the elements is required.
- Linked lists are better when lots of element insertions or deletions are required, like using queue or stack.

Chapter 4

Advanced Features

1. Color effects
2. Visual effects
3. Leaderboard
4. Move suggestion

4.1 Color effects

Main menu

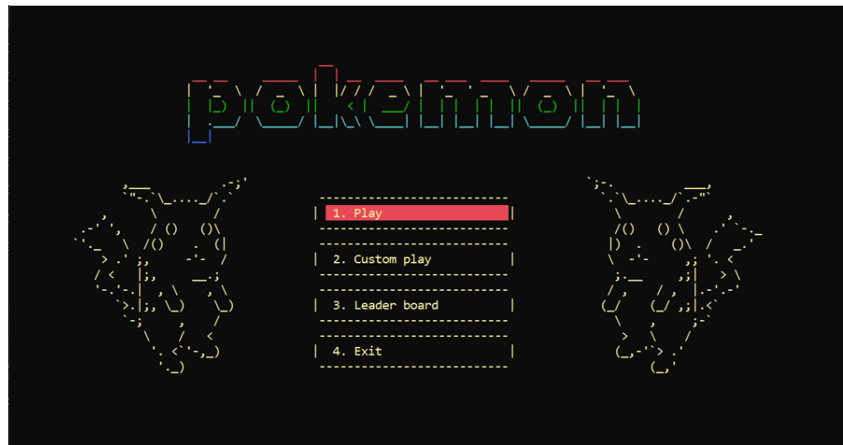


Image 4.1.1 Main menu

Gameplay

* Color will appear in these 3 cases:

- Selecting 2 cells → Blue
- If 2 cells have the same value and a valid path → Green
- If 2 cells don't have the same value or a valid path → Red

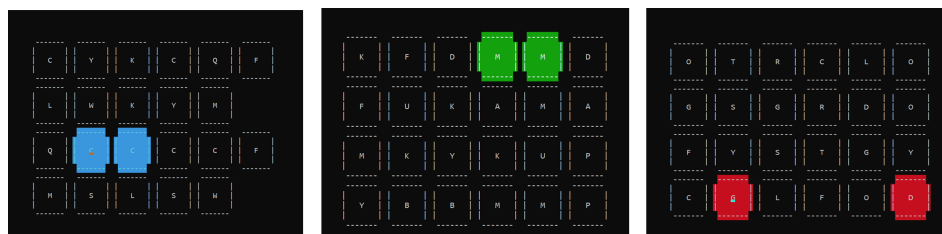


Image 4.1.2 Matching check

4.2 Visual effects

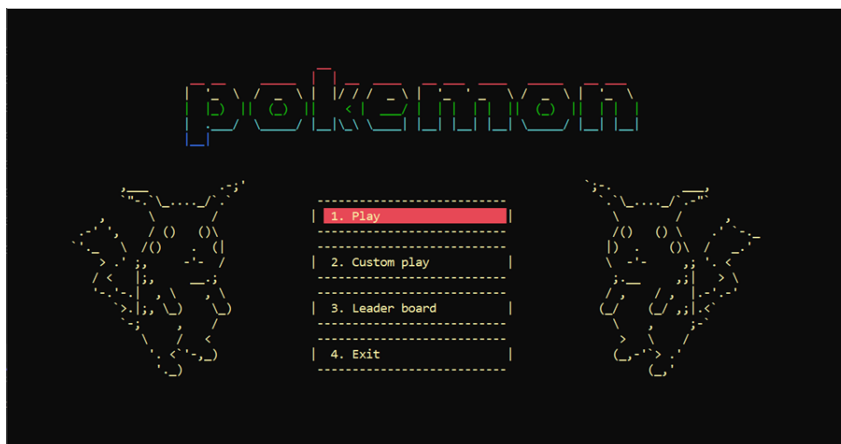


Image 4.2.3 Main menu



Image 4.2.4 Mode

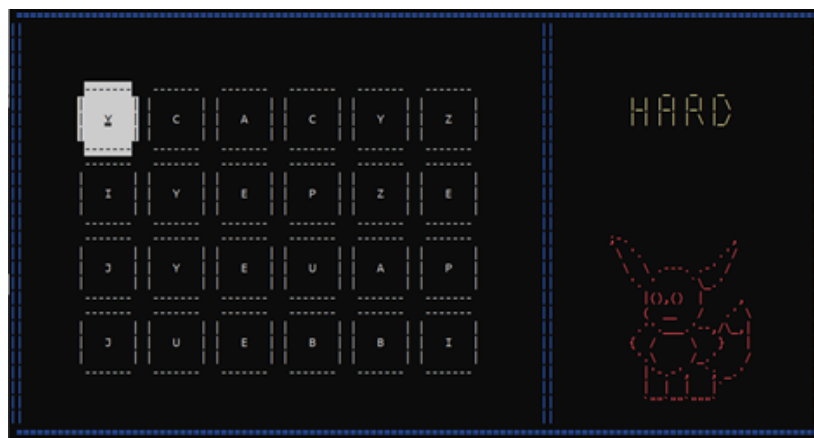


Image 4.2.5 Gameplay

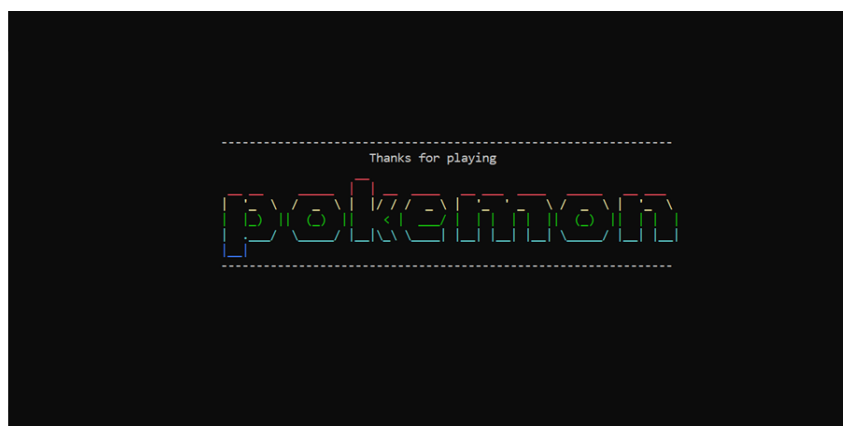
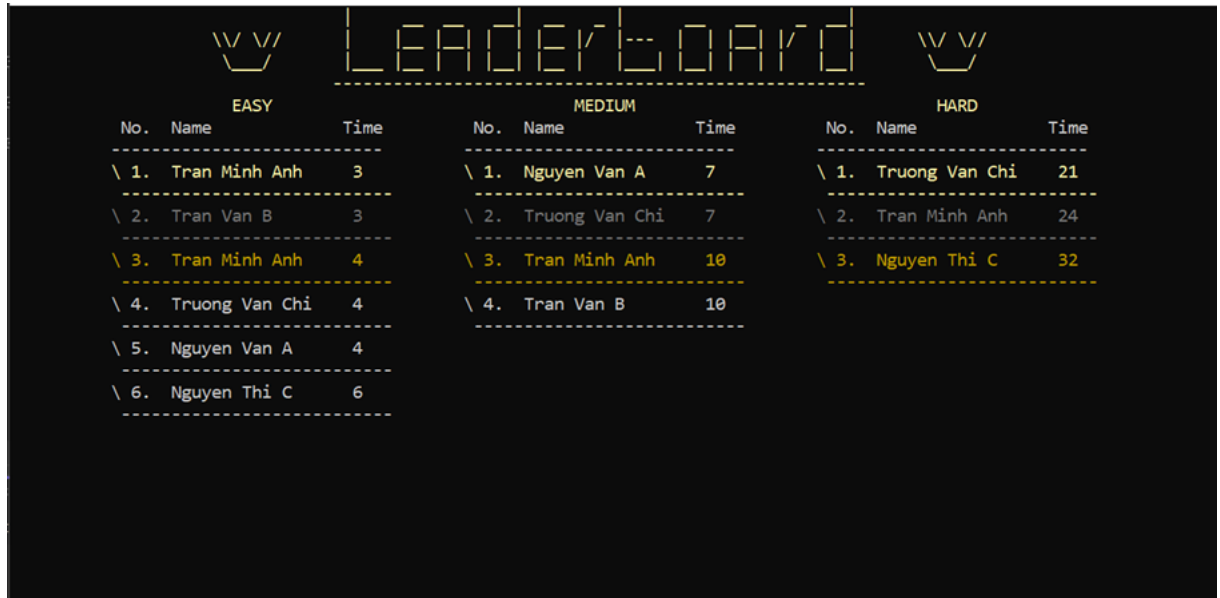


Image 4.2.6 Exit screen

4.3 Leaderboard

- When a user got a highscore, he/she will be asked if he/she wants to be on the leaderboard. If the user chooses Yes, the program will ask for his/her name. After that, the name and the playtime of that user will be compared to those of other users and finally added to the leaderboard in the right place.
- To know how it works, you can read about it in the function section above.



EASY			MEDIUM			HARD		
No.	Name	Time	No.	Name	Time	No.	Name	Time
\ 1.	Tran Minh Anh	3	\ 1.	Nguyen Van A	7	\ 1.	Truong Van Chi	21
\ 2.	Tran Van B	3	\ 2.	Truong Van Chi	7	\ 2.	Tran Minh Anh	24
\ 3.	Tran Minh Anh	4	\ 3.	Tran Minh Anh	10	\ 3.	Nguyen Thi C	32
\ 4.	Truong Van Chi	4	\ 4.	Tran Van B	10			
\ 5.	Nguyen Van A	4						
\ 6.	Nguyen Thi C	6						

Image 4.3.7 Leaderboard

4.4 Move suggestion

- When the moveSuggestion function is called (by pressing h or H while playing), the hint will be shown. There will be 2 cells which the color of those cells' background is set to dark blue as a signal to a hint.
- To know how it works, you can read about it in the function section above.

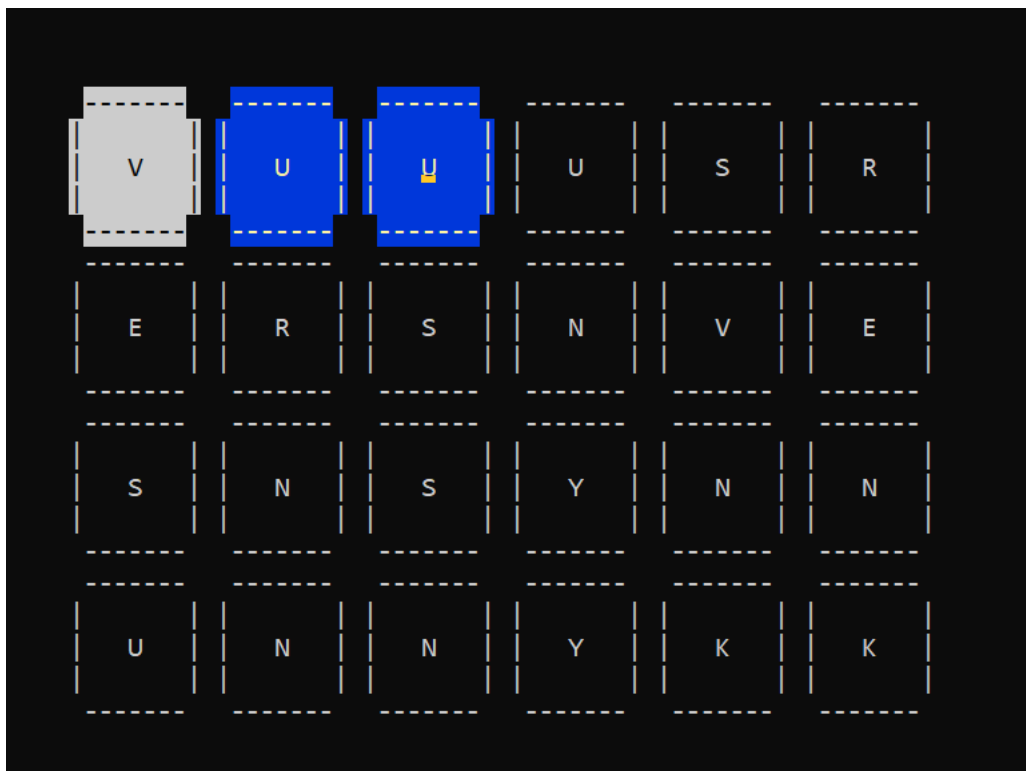


Image 4.4.8 Move suggestion

Chapter 5

References

1. <https://codelearn.io/sharing/windowsh-va-ham-dinh-dang-console-p1>
2. <https://zfull.net/thu-thuat-it/doi-mau-chu-va-mau-nen-don-gian-trong-c/>
3. <https://www.cplusplus.com/reference/>
4. Tony Gaddis - "Starting Out with C++ from Control Structures to Objects"