VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION AND TECHNOLOGY

PROJECT REPORT

MATCHING GAME

23CLC10

INSTRUCTOR
BUI HUY THONG
NGUYEN NGOC THAO

-o0o-
STUDENT
23127517 NGUYEN NAM VIET
23127427 VU HOANG MINH

HO CHI MINH CITY, APRIL 16TH 2024

INTRODUCTION

In this project, we aim to create a simplified version of the beloved Matching Game, commonly known as the Pikachu Puzzle Game. The game features a board consisting of multiple cells, each displaying a unique character. The objective is for the player to locate and match pairs of cells that contain the same character, connecting them in a specific pattern. Once a valid match is made, the matched cells will disappear. The game continues until all matching pairs have been found.

For this project, we will develop a revamped version of the Matching Game, replacing the traditional figures with captivating characters. This will add a delightful twist to the gameplay experience

PREFACE

First and foremost, we extend our heartfelt appreciation to our esteemed instructors, Bui Huy Thong and Nguyen Ngoc Thao. Their unwavering guidance and support have been invaluable throughout this project. Their teaching and insightful suggestions have played a pivotal role in shaping the direction and enhancing the overall quality of our product.

Additionally, we would like to express our deep gratitude to all those who have contributed to the successful completion of this endeavor. Your valuable contributions and unwavering support have been instrumental in bringing this project to fruition. Thank you for your dedication and commitment.

SUMMARY

This report focuses on the development of the Pikachu Game using C++. The game is inspired by the popular matching game featuring Pikachu. The objective of the game is to find pairs of Pokémon and match them. Two identical Pokémon can be matched if there is a clear path connecting them in the shape of an I, L, U, or Z. It is important to ensure that no obstacles obstruct the path between the matching Pokémon. The game is considered complete when all Pokémon have been successfully matched.

MEMBER INFORMATION

Our team has two members and we are both students from the Information Technology at 23CLC10 in the UNIVERSITY OF SCIENCE.

Team Member 1:

- `Name:` Vũ Hoàng Minh
- `ID:` 23127427
- `Gmail:` vhminh23@clc.fitus.edu.vn
- `Role in Project:` Project Manager, Programmer, Report Writer, Designer, Idea Generator, Researcher, Member

Team Member 2:

- `Name:` Nguyễn Nam Việt
- `ID:` 23127517
- `Gmail:` nnviet23@clc.fitus.edu.vn
- `Role in Project:` Programmer, Report Writer, Designer, Idea Generator, Researcher, Member

In our project journey, we encountered various challenges such as idea generation, finding a starting point, mastering complex library functions, tackling new problems, and exploring advanced algorithms, libraries, and processing methods. However, we overcame these challenges by carefully selecting capable team members who were well-equipped to handle them.

Moreover, throughout the project, we acquired valuable skills and experiences. We developed effective collaboration and communication skills, which proved essential for working harmoniously as a team. Additionally, we honed our problem-solving abilities and learned how to efficiently manage our time.

These experiences have not only helped us overcome obstacles but have also enriched our professional growth.

Please let us know if there is any additional information you require for the project submission through gmail or directly.

CONTENTS

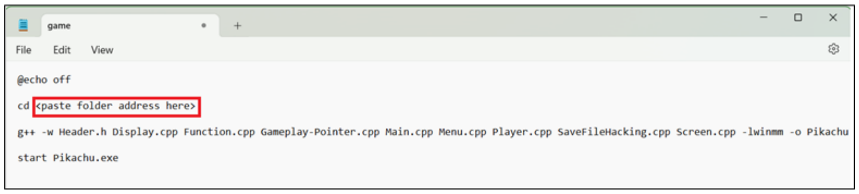
Mục lục

1 GAME TUTORIAL

1.1 Running game

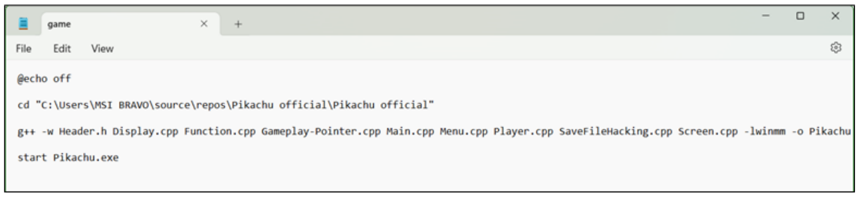
Player can follow the instruction below to start the game (the tutorial for Windows 11):

- `Step 1:` Right click at the file ‘Pikachu.bat’ ⇒ Show more option ⇒ Edit.
- `Step 2:` Copy the address of the folder named ’Pikachu official’ and paste into the tag <paste folder address here>



```
@echo off
cd <paste folder address here>
g++ -w Header.h Display.cpp Function.cpp Gameplay-Pointer.cpp Main.cpp Menu.cpp Player.cpp SaveFileHacking.cpp Screen.cpp -lwinmm -o Pikachu
start Pikachu.exe
```

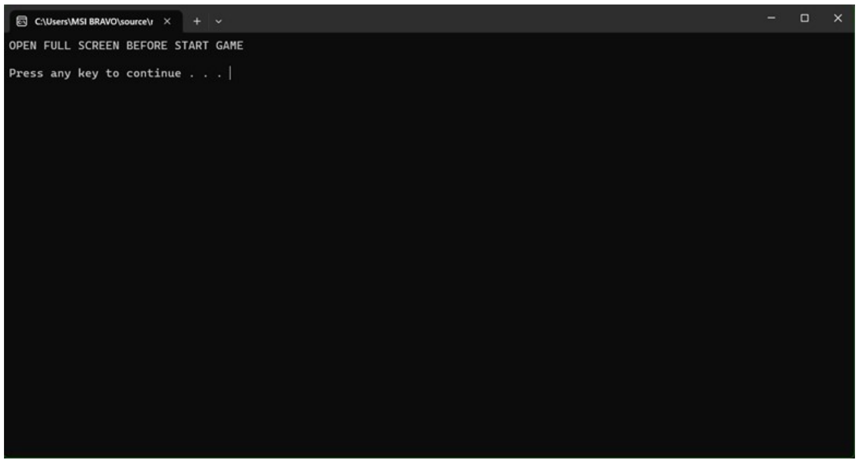
Figure 1.1: *Pasting folder address tutorial*



```
@echo off
cd "C:\Users\MSI BRAVO\source\repos\Pikachu official\Pikachu official"
g++ -w Header.h Display.cpp Function.cpp Gameplay-Pointer.cpp Main.cpp Menu.cpp Player.cpp SaveFileHacking.cpp Screen.cpp -lwinmm -o Pikachu
start Pikachu.exe
```

Figure 1.2: *Example of a legally folder address pasting*

- `Step 3:` Run file ‘Pikachu.bat’, the game start successfully when the computer displays this screen



```
OPEN FULL SCREEN BEFORE START GAME
Press any key to continue . . .
```

Figure 1.3: *Game starting screen*

To visualize more clearly:
https://drive.google.com/file/d/13TGbQflwu82coJw1urlZ4Fw6sfI5YW0K/view?usp=sharing

1.2 Game mechanic

1.2.1 Control

- `Up arrow`: move up
- `Down arrow`: move down
- `Left arrow`: move left
- `Right arrow`: move right
- `Space`: Select
- `F`: Shuffle (use in-game)
- `H`: Hint (use in-game)
- `S`: Save game (use in game)
- `ESC`: Quit the game, back to the menu (use in-game)

1.2.2 Game features

Shuffle: In the event that no moves are available, the game board automatically shuffles the positions of the Pokémon to generate new potential matches. Additionally, players have the option to activate this feature during the game.

Hint: Matching pairs of Pokémon are highlighted with a vibrant green color, providing players with a suggested move to consider.

Save Game: Players can save their current game board if they wish to exit the game, allowing them to resume their progress from where they left off. The saved board can be recreated when the player chooses to continue their previous game.

1.2.3 Gameplay

The objective of this game is to locate and match pairs of Pokémon. Two identical Pokémon can be legally matched if they form an I, L, U, or Z line, without any obstacles in between. The game concludes when no Pokémon are remaining on the board. The default game board size is 6x8.

There are three available modes in the game:

- `Easy Mode`: After a successful match, the cells containing the matched Pokémon will disappear, leaving empty spaces in their original positions.
- `Hard Mode`: After a successful match, the neighboring cells will slide down into the empty space created by the match.
- `Custom Mode`: Players have the freedom to customize the game board according to their preferences.

1.2.4 Point

In the game, players earn points based on the type of matching they achieve:

- `For an "I" matching`: 1 point.
- `For an "L" matching`: 3 points.
- `For a "U" or "Z" matching`: 5 points.

Upon completing a single game in both easy and hard modes, the program will save the player's points and finishing time to compare with their previous best record. The best record will then be compared with other players, and the top 5 players with the best records will be showcased on the game leaderboard.

The leaderboard is divided into two sections, one for easy mode and another for hard mode. Each section displays the best time and best points record achieved by players.

2 FILE INSTRUCTIONS AND LIBRARY

2.1 Library

The code starts with some preprocessor directives `#pragma comment(lib, "winmm.lib")` and `#pragma warning(disable : 4996)`.

The `#pragma comment(lib, "winmm.lib")` directive is used to specify a library to be linked with the program. In this case, it is linking the `winmm.lib` library, which provides access to functions for multimedia programming in

The `#pragma warning(disable : 4996)` directive disables a specific compiler warning with code 4996. This warning is related to the use of deprecated functions, such as `strcpy`, `strncpy`, `strcat`, and `strncat`. Disabling this warning means that the code can use these functions without generating a warning.

It then includes various header files:

- `iostream`: It provides input/output stream functionality.
- `Windows.h`: It provides Windows-specific functions and types.
- `conio.h`: It provides functions for console input/output.
- `string.h`: It provides string manipulation functions.
- `stdlib.h`: It provides general utility functions.
- `vector`: It provides a container class for dynamic arrays.
- `fstream`: It provides file input/output functionality.
- `mmsystem.h`: It provides multimedia functions and types.
- `cmath`: It provides mathematical functions.
- `cstring`: It provides C-style string functions.
- `ctime`: It provides functions for working with date and time.
- `string`: It provides string manipulation functions.

These header files are included to provide the necessary functions and types required for the implementation of the game.

2.2 Header File: Header.h

Header.h is an essential file in the project, it provides necessary constants, data structures, and function declarations for the project.

- Constants: define color code, keyboard input ASCII code, and screen size.
- Important data structure:
  - Board: includes the number of rows and columns of the game board and its height and width.
  - Pokemon: includes the coordinate of the Pokemon/cell on the virtual board (use for algorithm) and on the window output console (use for game UI).
  - Player and InforMatches: stores player information such as username, matches played, best point, best time, etc.

2.3 Source Files

2.2.1 Display.cpp

Display.cpp is a source code file containing all functions related to game UI. More specifically, these functions use ASCII characters to create a border, game box, matching line and background.

2.2.2 Function.cpp

Function.cpp includes functions used for console output manipulation such as:

- **void gotoxy(int x, int y)** $\Rightarrow$ Moving the console cursor to the specified position on the screen.
- **void TextColor(int x)** $\Rightarrow$ Change the color of the console text.
- **void ShowConsoleCursor(bool showFlag)** $\Rightarrow$ Show or hide the console cursor (0: hide cursor, 1: show cursor).

2.2.3 Gameplay-Pointer.cpp

Gameplay-Pointer.cpp is a critical file of the game program as it contains the entire code for the gameplay. There are a variety of functions in this file created to perform specific tasks, and the important functions are listed below:

- **void createBoard(int**& board, Board gameBoard)** $\Rightarrow$ Allocate the memory for the game board.
- **void createPoint(int**& board, Board gameBoard)** $\Rightarrow$ Create the pokemons.
- **bool matchingI(int** board, Pokemon p1, Pokemon p2)** $\Rightarrow$ Check I matching.
- **bool matchingL(int** board, Pokemon p1, Pokemon p2, Pokemon& corner)** $\Rightarrow$ Check L matching.
- **bool matchingUZ(int** board, Pokemon& p1, Pokemon& p2, Pokemon& corner1, Pokemon& corner2, Board gameBoard)** $\Rightarrow$ Check U and Z matching.
- **bool checkMatching(int** board, Pokemon p1, Pokemon p2, int& point, Board gameBoard, bool display)** $\Rightarrow$ Check if two selected pokemon are matched or not by checking I, U, L, and Z patterns at once.

- **void deletePoint(int** board, Pokemon p1, Pokemon p2, int mode)** ⇒ Delete two pokemons if they are matched.
- **bool checkContinue(int** board, Pokemon& p1, Pokemon& p2, Board gameBoard)** ⇒ Check if there are any legal matches left on the board.
- **void Gameplay(int**& board, int mode, Board& gameBoard, vector<Player*> list, int playerPos)** ⇒ The main gameplay.
- **void GameplayCustom(int**& board, int mode, Board& gameBoard, vector<Player*> list, int playerPos, bool& hackFile)** ⇒ The gameplay for custom mode.
- **void deleteBoard(int**& board, Board gameBoard)** ⇒ Delete the board after playing.

The function 'Gameplay' is the main function in this file, because it is responsible for the entire gameplay process. The function 'GameplayCustom' is the copy of 'Gameplay' but eliminates the save game, point record and time record as custom mode does not include these features.

2.2.4 Menu.cpp

Menu.cpp contains code that allows the player to interact with the menu screen, for instance, moving between options and leading the player to the screen based on their selection.

2.2.5 Player.cpp

Functions in the file "Player.cpp" are used to read from and write to the binary file 'Player.bin'. When the program starts, the 'Player.bin' file is read to obtain information about the players, and when the program ends, the updated information is written back to the 'Player.bin' file to store the data. Additionally, some functions support sorting players' best points and best time records to provide information for the leaderboard.

2.2.6 Screen.cpp

The Screen.cpp file consists of six functions used to display six specific screens. These screens not only present the interface but also allow players to interact with them. The functions are as follows:

- **void IntroScreen()** ⇒ Intro screen: displays the game opening animation.
- **void loginScreen(vector <Player*>& list, int& playerPos)** ⇒ Login screen: allows players to sign up or sign in.
- **void helpScreen()** ⇒ Help screen: provides instructions for players.
- **void leaderboardScreen(vecto <Player*> list)** ⇒ Leaderboard screen: shows the top 5 players with the highest points and fastest times.
- **void customScreen(int& mode, Board& gameBoard)** ⇒ Custom screen: allows players to input the board size.
- **void hackScreen(vector <Player*>& list, int playerPos, Board& gameBoard, int**& board, bool& hackFile)** ⇒ Hack screen: used for the hack save file feature, allowing players to input their hack file.

2.2.7 Main.cpp

Main.cpp is the main file that runs the whole program from start to end.

2.2.8 Screen Info.txt

Screen Info.txt is file text which contains remarkable information about the size and the corner's coordinate of a board or a picture. This information can be used by programmers to save time when designing game UI by using the existing numbers for calculations.

2.2.9 Player.bin

Player.bin is a binary file that stores players data. It is used to check available username in log in phrase and give information of top players for leaderboard.

2.4 Other file

2.3.1 File.WAV

There are four files included: `INTRO.WAV`, `MATCHING.WAV`, `MISTAKE.WAV`, and `SELECT.WAV`. These files are used to provide sound effects in the game.

2.3.2 Pikachu.exe

This is an executable file that can be launched by double-clicking on it. It allows the game program to run without the need for compiling it within an integrated development environment (IDE) like Visual Studio.

2.3.3 Pikachu.bat

`Pikachu.bat` is a batch file that contains the necessary commands to compile and run the game using `g++` and the command prompt. After players edit the batch file as instructed in the "1.1 Running game" section, the game can be launched automatically by double-clicking the `Pikachu.bat` file.

3 STANDARD FEATURES

3.1 Game starting

...

3.2 Matching algorithm

Assuming two selected cells (named *cell 1* and *cell 2*) have the index in dynamic 2D array are (*i1*, *j1*) and (*i2*, *j2*). There are four types of matching must be checked: matching I, matching L, matching U and matching Z.

3.2.1 Matching I

There are two cases for I matching.

Case 1: Two cells in same row ($i1 = i2$)

> If $j1 > j2$ (*cell 1* is to the right *cell 2*), swap the index of *cell 1* and *cell 2*. When two cells in same row, it is not important to check left to right or right to left (in code file, it will be checked from left to right) because the result will be same.
>
> To check if two cells can be connected: Using a while loop check from the cell which index ($i1$, $j1 + 1$) to ($i2$, $j2$ - 1). While it is checking, if it catches a cell whose value is different from 0 (not an empty cell), it is immediately return "false". If a loop complete without finding any cell other 0, it means *cell 1* and *cell 2* can connected, and return "true".

Case 2: Two cells in same column ($j1 = j2$)

> Similar to case 1, if $i1 > i2$ (*cell 1* under *cell 2*), swap the index of *cell 1* and *cell 2*. Following this, two cells are checked by a while loop

from index (*i1 + 1*, *j1*) to (*i2 - 1*, *j2*), if any cells do not contain value 0 is found from a loop, it returns "false", otherwise it returns "true".

3.2.2 Matching L

There are four ways to connect a pair of cells by L line, illustrated by the picture below.

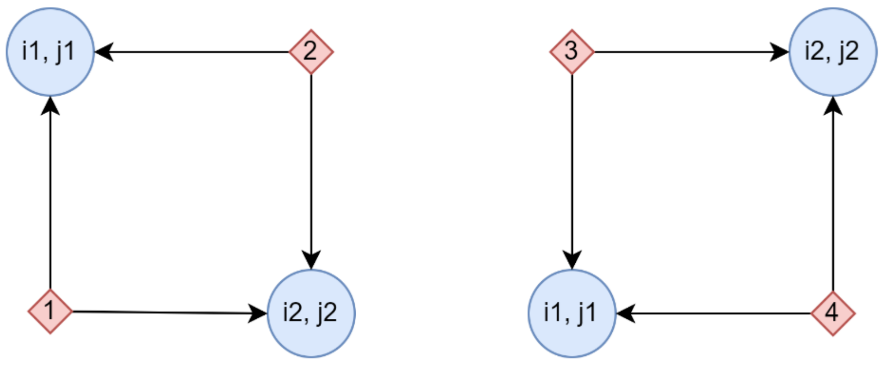


Figure 3.2.2.1: *4 case of L matching*

However, there are only two cases to check two cells based on the similarity of the corner cell's coordinate. In first case, the index of the corner cell is (*i1*, *j2*) (same row with *cell 1* and same column with *cell 2*). The second case, the index of the corner cell is (*i2*, *j1*).

The program checks for a legal match in both situations by checking matching I from *cell 1* to the corner cell and from *cell 2* to the corner cell. The matching is considered valid only when both conditions return "true".

3.2.3 Matching U Z

There are many cases of matching U and Z; however, both U matching and Z matching use the same algorithm to check matching. The main idea of the algorithm is finding an empty line connect the column or the row of two selected cells. Based on this common point, we can merge U matching and Z matching for common consideration.

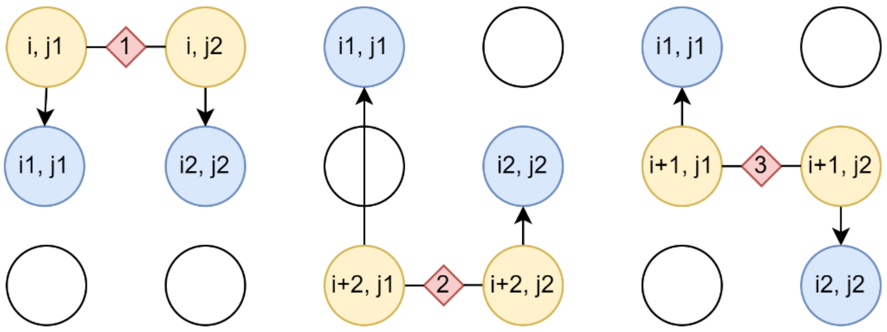


Figure 3.2.3.1: *First case of U, Z matching*

In this case, the connecting line is horizontal, a for loop is executed with two corners cells have the coordinate (i, j1) and (i, j2) corresponding to the same column as *cell 1* and *cell 2* from the upper border to the lower border. For each case of connecting line in for loop, three I matchings are considered (*cell 1* to *corner1*), (*corner1* to *corner2*) and (*corner2* to *cell 2*). If all 3 I matchings are satisfied, then the pair can be connected, and the loop is broken with a return value of 'true'. If the loop can not find a matching pair, it means that either the matching condition is not satisfied or *cell 1* and *cell 2* is in another U or Z matching case. In this situation, the program continues to check the next case

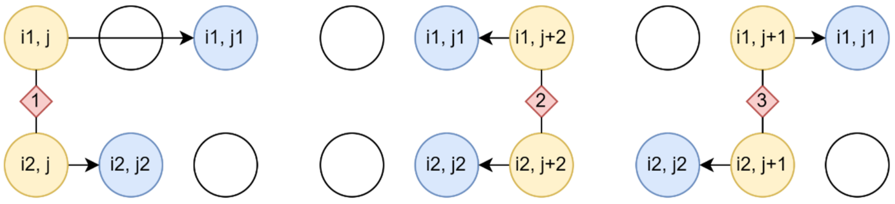


Figure 3.2.3.2: *Second case of U, Z matching*

In the second case, the connecting line is vertical, the program will run as same as above, only change a loop, the index of two corners cells are (*i1*, *j*) and (*i2*, *j*), run from the left to the right and check three I matchings for each case in loop

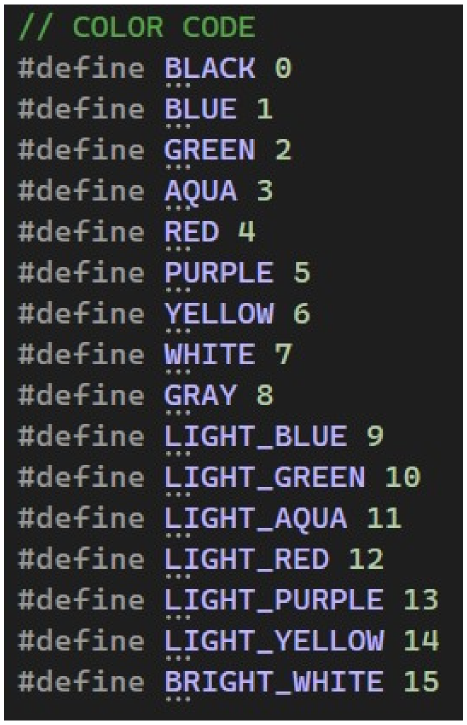
3.3 Game finish verify

To determine if the game has reached its conclusion, a variable called 'pokemon' is defined, which represents the total number of Pokémon on the game board. This 'pokemon' variable is calculated by multiplying the number of rows and columns of the game board. Each time the player successfully removes a pair of Pokémon, the value of the 'pokemon' variable decreases by 2 units. This process continues until the value of 'pokemon' reaches 0, indicating that all pairs of Pokémon have been eliminated from the game. At this point, the player has completed the game's objective, and the game comes to an end.

4 ADVANCED FEATURE

4.1 Color effect

The 'TextColor' function, located in the 'Function.cpp' file, is responsible for changing the color of the text in the console output window. This function takes an integer parameter, 'x', which represents the color code in C++. To utilize this function, it is necessary to include the 'Windows.h' library in our C++ program.



```
// COLOR CODE
#define BLACK 0
#define BLUE 1
#define GREEN 2
#define AQUA 3
#define RED 4
#define PURPLE 5
#define YELLOW 6
#define WHITE 7
#define GRAY 8
#define LIGHT_BLUE 9
#define LIGHT_GREEN 10
#define LIGHT_AQUA 11
#define LIGHT_RED 12
#define LIGHT_PURPLE 13
#define LIGHT_YELLOW 14
#define BRIGHT_WHITE 15
```

Figure 4.1: *Color code defined in Header*

The source code of the function is referenced from: https://tuicocach.com/viet-ham-thaydoi-mau-chu-trong-man-hinh-console-c-c/

4.2 Sound effect

To play a sound in the game, the **PlaySound** function is utilized with the following syntax:

PlaySound(TEXT("fileName.wav"), NULL, SND_ASYNC);

The first parameter represents the name of the sound file, which must be in the WAV format and saved in the same folder as the source code files.

The second parameter is set to **NULL** as per the instruction: "This parameter must be **NULL** unless **SND_RESOURCE** is specified in the last parameter."

The final parameter, **SND_ASYNC**, is the flag used to play the sound. **SND_ASYNC** allows the program to continue executing the following commands while the sound is being played.

Alternatively, using **SND_SYNC** would cause the program to wait until the sound has finished playing before executing the subsequent commands. However, this may result in a slight delay when continuously receiving input from the player via the keyboard.

4.3 Visual effect

The function 'matchingLineType' takes two parameters: 'p', which is a Pokemon object, and 'type', which is an integer. The 'p' parameter is used to determine the position of the cell and display the appropriate pattern based on the given type. The available line types are illustrated in Figure 4.2, showcasing six different matching line patterns.

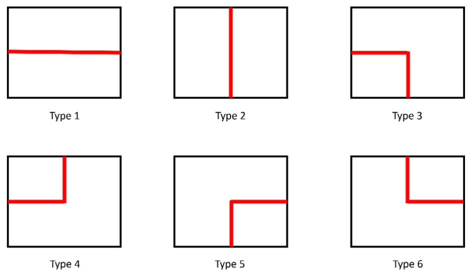


Figure 4.2: *Six matching line types*

Additionally, we have the 'displayMatchingLine' function, which is responsible for drawing a line between two given cells. The process of displaying these matching lines is depicted in Figure 4.3.

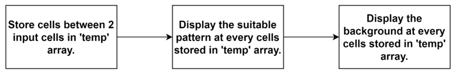

Store cells between 2 input cells in 'temp' array.
Display the suitable pattern at every cells stored in 'temp' array.
Display the background at every cells stored in 'temp' array.

Figure 4.3: *4.3 Process of display matching line*

This function utilizes two types of patterns: type 1 for drawing horizontal lines and type 2 for drawing vertical lines. It also includes two special parameters: an array called 'temp' and a boolean variable named 'display'. The 'temp' array is used to store the cells between the two input cells, represented by Pokemon objects. The 'display' variable determines whether to display the matching line or not. This approach allows us to store the cell values in the 'temp' array without immediately displaying the matching line. After the line is displayed, the 'temp' array is cleared to prevent any memory leaks. This functionality is particularly useful for displaying matching lines L, U, and Z.

The 'displayMatchingLine' function serves as the foundation for displaying matching lines for the L, U, and Z shapes, and it also ensures that the corners are properly displayed to complete these functions. The selection of types 3, 4, 5, and 6 depends on the current cell's location in relation to the two connecting cells.

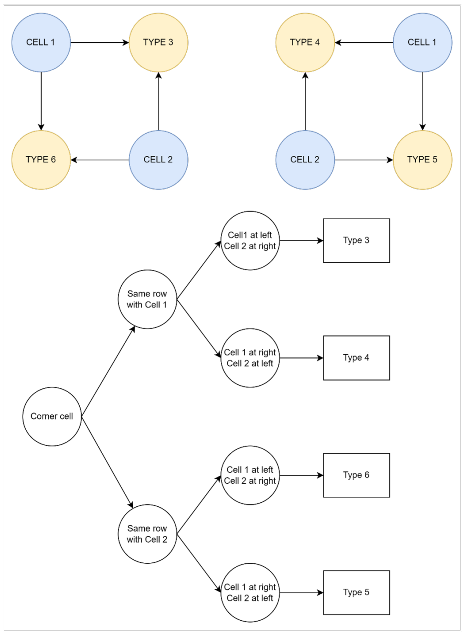


Figure 4.4: *Case diagram of displaying corner cell*

Although matching Z has two corners, it uses the same logic as matching L to display the corners

4.4 Background

To enhance the player's experience, we can improve the display of the background in the game. Instead of showing the entire game board with empty cells, we can first display the background and then fill in the remaining cells. However, this approach may cause a small flicker and negatively impact the player's experience.

To address this issue, we can divide the background into parts that match the size of each cell, which is 5x10. This way, we can display the exact part of the background corresponding to each cell without any flickering.

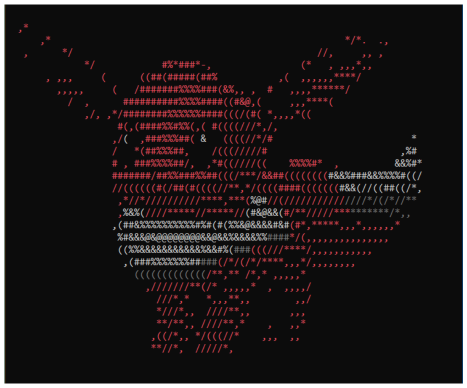


Figure 4.5: *: Background divisio*

To implement this, we can use the function 'background_1_upgrade' which takes a Pokemon data type as input and displays the appropriate part of the background. The picture used for the background is created by the talented designer Ydeval and can be downloaded from the website https://cults3d.com/en.

To convert the downloaded picture into ASCII characters, we can utilize the tool available at https://manytools.org/hackertools/convert-images-to-ascii-art/. This will allow us to represent the background using ASCII art, adding a unique visual element to the game.

4.5 Leaderboard

To sort the players' points in descending order and their times in ascending order, a selection sort algorithm is utilized. However, there is an issue with sorting the player times. When a player has not played any games, the program assigns a value of 0 to their best point and best time. Consequently, after sorting, the players with a time of 0 appear at the top of the leaderboard. To address this, a variable named 'skip' is introduced to keep track of the number of occurrences of 0 in the list. This allows the loop to start from 'skip' and display the top players on the leaderboard correctly.

4.6 Move suggestion

To provide a move suggestion, the 'checkContinue' function in the 'Gameplay.cpp' file is utilized. This function ensures that the game can continue and identifies any pairs of cells that are available to be matched. These pairs are then stored in the variables 'p1' and 'p2' of the 'Pokemon' data type. Finally, the highlighted cells that were stored previously are used to offer the player a move suggestion.

5 EXTRA ADVANCED FEATURE

5.1 Stage difficult increase

In the challenging hard mode, when a pair of cells is matched, the positions of the deleted cells will shift, causing the cells above them to slide down and fill the resulting gaps. This mechanism is depicted in Figure 5.1, showcasing the deletion of cells in hard mode.

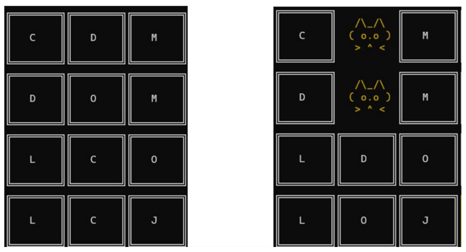


Figure 5: *Delete cells in hard mode*

To implement the shifting of the remaining cells after the removal of a pair, a simple loop is utilized. This loop iterates through all the cells positioned on the locations previously occupied by the two removed cells. Each cell is then moved down by one row, effectively filling the gap created by the deletion.

Furthermore, if there are two cells positioned in the same column, and the first cell is below the second cell, an additional check is performed. In this case, the second cell is also moved down by one row to prevent any overlapping and ensure the proper alignment of the remaining cells. This prevents any visual inconsistencies that may arise from cells overlapping each other.

5.2 Pointer/Linkedlist Comparison

When utilizing a LinkedList to initialize a game board, it is crucial to handle node construction carefully to prevent memory leaks. Unlike arrays, LinkedList nodes do not possess coordinate indices. Therefore, when creating a struct, additional variables must be added to store the coordinates, which increases memory usage.

Compared to using pointers, LinkedLists do not allow direct access to specific elements. Instead, sequential traversal using a loop is required to access elements from the beginning to the end. However, this limitation can be overcome by implementing a function, such as 'findPos', which can locate the x and y coordinates of a given Pokémon and return its address. By combining the 'findPos' function with pointer operations, we can efficiently address element access issues in LinkedLists.

In summary, LinkedLists are more complex than pointers, requiring sequential element access and resulting in increased time consumption.

6 ADDING FEATURE

6.1 Shuffle

The shuffle feature plays a crucial role when the game runs out of available matches or when the player selects the 'F' button to shuffle the game board. The algorithm behind this feature is relatively straightforward: The first idea is shuffling all cells, but it needs to use four "for" loops, so it is not optimized. Therefore, the code is using the "count" variable and use one "while" loop (it can shuffle almost cells). In this loop has two cells and the index of these will be randomized. The condition of "while" loop is "count" larger or equal (row x column - 1) (because the number of cells in custom mode can be odd). However, there are some cases lead to several error in the game board when shuffle:

- If the value in cell 1 is equal to 0 (representing an empty cell).
- If the value in cell 2 is equal to 0 (also representing an empty cell).
- If the value in cell 1 is equal to the value in cell 2.
- If cell 1 and cell 2 have the same coordinates.

To prevent these issues, a while loop is implemented. If the randomly selected cells violate any of the aforementioned conditions, the shuffle process is repeated until valid cells are chosen. This ensures that the shuffle feature operates smoothly and avoids any unintended errors in the game board.

6.2 Save game

To provide a seamless user experience, the game includes a feature that allows the player to save their progress if they choose to quit the game midway. To ensure that there are no memory leaks and the saved game process is executed correctly, the saving process follows a step-by-step approach as illustrated in the diagram below:

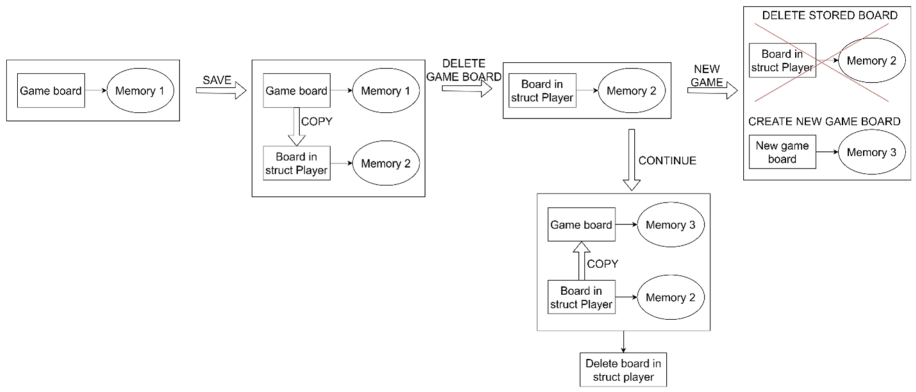


Figure 6.1: *The process of saving game*

By following this step-by-step process, the game board is saved securely, allowing the player to resume their game from where they left off without any data loss.

7 CONCLUSION

In conclusion, this report serves as a comprehensive tutorial, offering insights into the mechanics of the game, in-depth descriptions of its features, and a breakdown of the functions and algorithms employed throughout the project. The game itself stands as a prime example of harnessing the power of programming languages to craft captivating and enjoyable gaming experiences.

8 REFERENCES

1. Nguyen To Son (2017). Building the gotoXY, whereX, whereY functions in Dev C++. Available at: `https://hoctincungthukhoa.com/index.php/l-p-trinh-c-c/64-xay-d-ng-ham-gotoxy-wherex-wherey-trong-dev-c`. Lines 1-21. (Accessed: March, 2023).

2. Louis2602 (2022). Louis2602/pikachu-game: Pikachu game made with C++: Hcmus programming techniques project, GitHub. Available at: `https://github.com/Louis2602/Pikachu-Game`. (Accessed: March, 2023).

   Reference source: BoardView.cpp (Lines 1-1051), BoardView.h (Lines 1-55), Controller.cpp (Lines 1-140), Controller.h (Lines 1-64), Game.cpp (Lines 1-897), Game.h (Lines 1-55), Menu.cpp (Lines 1-560), Menu.h (Lines 1-42), Point.cpp (Lines 1-53), Point.h (Lines 1-27), Main.cpp (Lines 1-8).

3. Xxnithicxx (2022). Xxnithicxx/pikachu-game: Pikachu game for BHT, GitHub. Available at: `https://github.com/xxnithicxx/Pikachu-Game`. (Accessed: March, 2023).

   Reference source: Background.cpp (Lines 1-175), Source.cpp (Lines 1-1862).

4. Tan Chan (2023). Writing the function to change text color in console screen C/C++ - textcolor(). Available at: https://tuicocach.com/viet-ham-thay-doi-mau-chu-trong-man-hinh-console-c-c. (Accessed: March, 2023).

5. Nguyen Phuc Khang (2023). Npeka/ CNTT-22CLC08-HCMUS / HK2/ CSC10002_Ky thuat lap trinh/Lab/Project Pokemon. Available at: `https://github.com/Npeka/CNTT-22CLC08HCMUS/tree/main/HK2/CSC10002_Ky%20thuat%20lap%20trinh/Lab/Project%20Pokemon`. (Accessed: January, 2024).

   Reference source: display.cpp (Lines 1-1036), screen.cpp (Lines 1-464), Gameplay-Pointer.cpp (Lines 1-1228), Menu.cpp (Lines 1-157), Player.cpp (Lines 1-201), Main.cpp (Lines 1-16), Header.h (Lines 1-197).

6. Microsoft Learn. Playsound function (2016) (Windows). Available at: `https://learn.microsoft.com/en-us/previous-versions/dd743680(v=vs.85)`. (Accessed: March, 2023).