VIETNAM NATIONAL UNIVERSITY,
HO CHI MINH

HO CHI MINH UNIVERSITY OF SCIENCE
Faculty of Information Technology

Report Programming Techniques
Matching Game

Do Dang Nhat Tien
23127269
23CLC10
ddntien23@clc.fitus.edu.vn

Vu Tien Luc
23127222
23CLC10
vtluc23@clc.fitus.edu.vn

March 8, 2025

Contents

List of Figures

1 Introduction to the game

This study examines the game mechanic known as the Matching Game (commercially branded as the Pikachu Puzzle Game). The game features a board with multiple cells, each displaying a distinct visual element (figure). Players must locate and connect matching pairs of figures that conform to a specific spatial pattern. Correctly identified pairs are removed from the board. The game concludes once all possible matching pairs have been found.

This project focuses on developing a simplified variant of the Matching Game. This new version will utilize characters as the primary visual elements, replacing the figures used in the original game. The game offers content across two distinct difficulty tracks: a standard track encompassing levels 1 through 3, and an advanced track comprising levels 4 and 5.

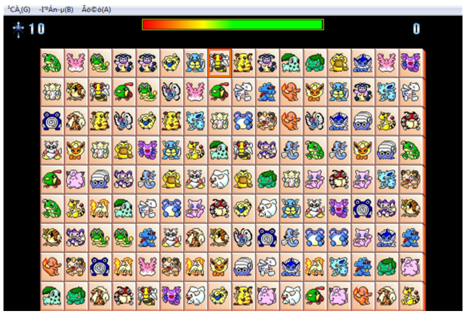


Figure 1: Pikachu Game

2 Tutorial

2.1 Requirements

This game was developed using **Code::Blocks**, an *Integrated Development Environment (IDE)* known for its user-friendly interface and suitability for beginners like interns. The **GNU C++ Compiler** was chosen for its compatibility with **Code::Blocks** and its adherence to a recent compiler version. This ensures the game's executability on a wider range of computers, including older models. Consequently, users will require **Code::Blocks** installed on their machines to run the game.If not, they can download it from the official website *(https://www.codeblocks.org/)*. In addition, players should **turn off Telex Vietnamese** for convenience in using the keys

The game's codebase utilizes the **C++11** programming language standard (compiled with the **-std=c++11** flag). To ensure optimal performance and compatibility, we strongly recommend running the game on a **64-bit Windows operating system**.

2.2 Instruction

- Step 1: Open the source file included in the project submission.

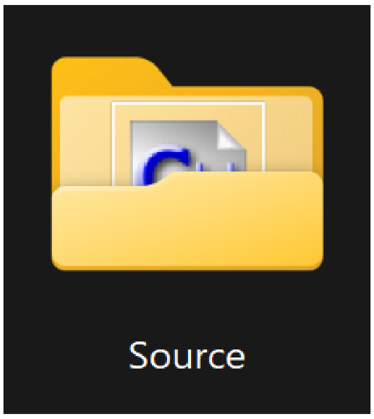
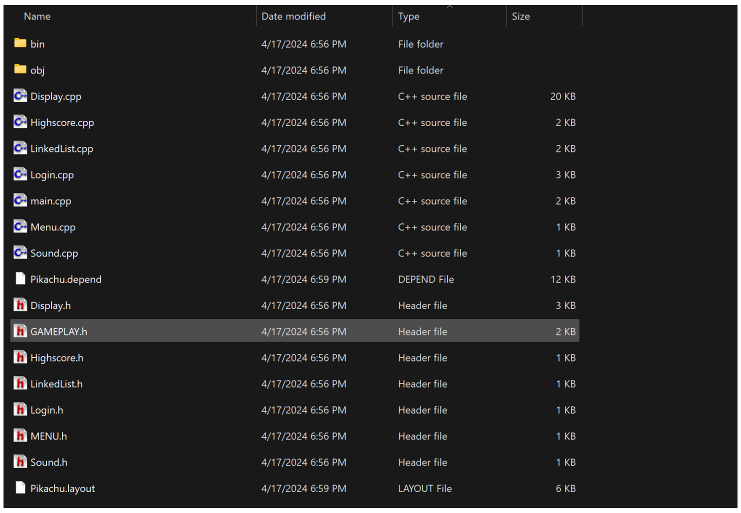


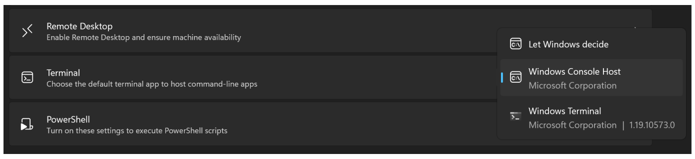
Figure 2: Source file



Name	Date modified	Type	Size
bin	4/17/2024 6:56 PM	File folder	
obj	4/17/2024 6:56 PM	File folder	
Display.cpp	4/17/2024 6:56 PM	C++ source file	20 KB
Highscore.cpp	4/17/2024 6:56 PM	C++ source file	2 KB
LinkedList.cpp	4/17/2024 6:56 PM	C++ source file	2 KB
Login.cpp	4/17/2024 6:56 PM	C++ source file	3 KB
main.cpp	4/17/2024 6:56 PM	C++ source file	2 KB
Menu.cpp	4/17/2024 6:56 PM	C++ source file	1 KB
Sound.cpp	4/17/2024 6:56 PM	C++ source file	1 KB
Pikachu.depend	4/17/2024 6:59 PM	DEPEND File	12 KB
Display.h	4/17/2024 6:56 PM	Header file	3 KB
GAMEPLAY.h	4/17/2024 6:56 PM	Header file	2 KB
Highscore.h	4/17/2024 6:56 PM	Header file	1 KB
LinkedList.h	4/17/2024 6:56 PM	Header file	1 KB
Login.h	4/17/2024 6:56 PM	Header file	1 KB
MENU.h	4/17/2024 6:56 PM	Header file	1 KB
Sound.h	4/17/2024 6:56 PM	Header file	1 KB
Pikachu.layout	4/17/2024 6:59 PM	LAYOUT File	6 KB

Figure 3: Items Inside Folder

- Step 2: Open settings, find Temporal Settings in the search bar and click on it. Then, in the Terminal section, select Windows Console Host for convenience when enlarging the screen.



Remote Desktop
Enable Remote Desktop and ensure machine availability

Terminal
Choose the default terminal app to host command-line apps

PowerShell
Turn on these settings to execute PowerShell scripts

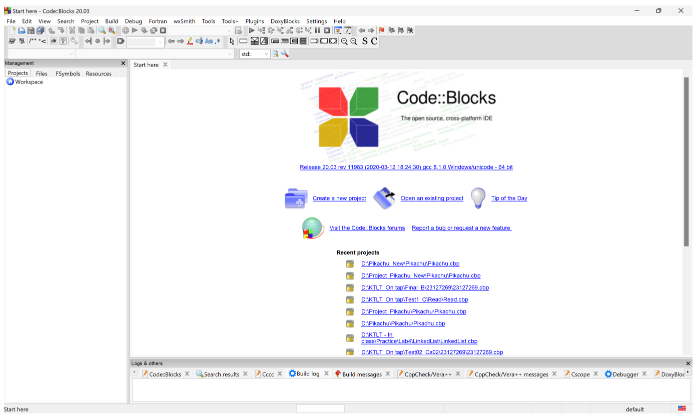
Let Windows decide

Windows Console Host
Microsoft Corporation

Windows Terminal
Microsoft Corporation | 1.19.10573.0

Figure 4: Terminal Setting

- Step 3: Launch Code::Blocks to proceed.


Start here - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Files FSymbols Resources
Workspace
Start here
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messages Cscope Debugger DoxyBlocks
default

Code::Blocks

The open source, cross-platform IDE

Release 20.03 rev 11983 (2020-03-12 18:24:30) gcc 8.1.0 Windows/unicode - 64 bit

Create a new project Open an existing project Tip of the Day

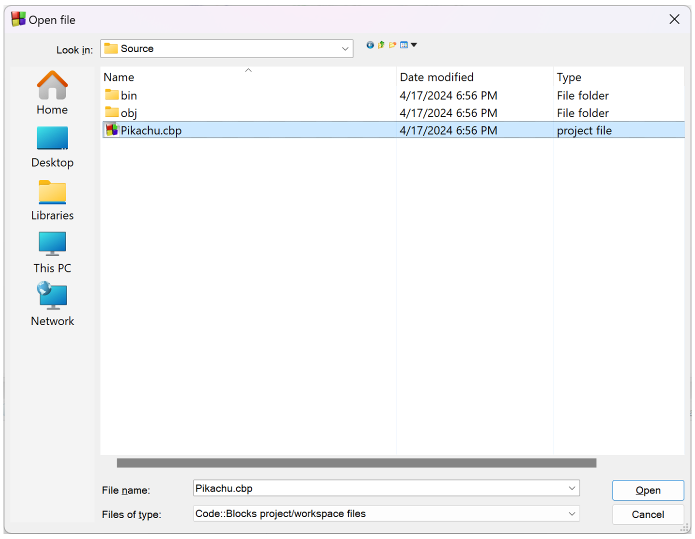
Visit the Code::Blocks forums Report a bug or request a new feature

Recent projects

D:\Pikachu_New\Pikachu\Pikachu.cbp
D:\Project_Pikachu_New\Pikachu\Pikachu.cbp
D:\KTLT_On tap\Final_B\23127269\23127269.cbp
D:\KTLT_On tap\Test1_C\Read\Read.cbp
D:\Project_Pikachu\Pikachu\Pikachu.cbp
D:\Pikachu\Pikachu\Pikachu.cbp
D:\KTLT - In class\Practice\Lab4\LinkedList\LinkedList.cbp
D:\KTLT_On tap\Test02_Ca02\23127269\23127269.cbp

Figure 5: Code::Blocks Interface

- Step 4: Navigate to the menu and select "Open an existing project." Then, locate the folder containing the extracted game files using the file browser window, identify the file with the *".cbp"* extension and select it. After that, click the "Open" button to initiate the project.

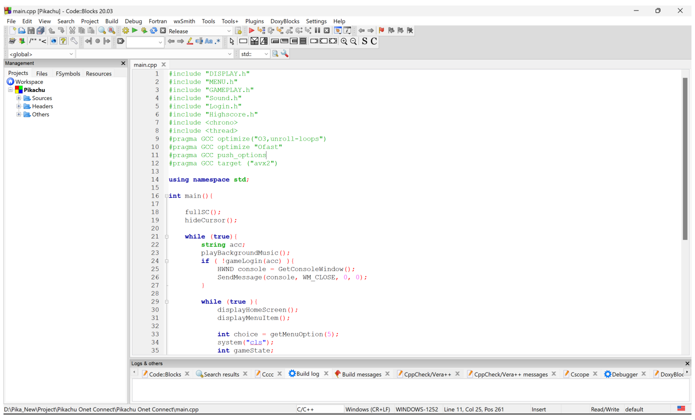


Name	Date modified	Type
bin	4/17/2024 6:56 PM	File folder
obj	4/17/2024 6:56 PM	File folder
Pikachu.cbp	4/17/2024 6:56 PM	project file

File name: Pikachu.cbp

Files of type: Code::Blocks project/workspace files

Figure 6: Choosing .cbp File And Initiate Project


main.cpp [Pikachu] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Release
Management
Projects Files FSymbols Resources
Workspace
Pikachu
Sources
Headers
Others
main.cpp
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messages Cscope Debugger DoxyBlocks

```
#include "DISPLAY.h"
#include "MENU.h"
#include "GAMEPLAY.h"
#include "Sound.h"
#include "Login.h"
#include "Highscore.h"
#include <chrono>
#include <thread>
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC optimize "Ofast"
#pragma GCC push_options
#pragma GCC target ("avx2")

using namespace std;

int main(){

    fullSC();
    hideCursor();

    while (true){
        string acc;
        playBackgroundMusic();
        if ( !gameLogin(acc) ){
            HWND console = GetConsoleWindow();
            SendMessage(console, WM_CLOSE, 0, 0);
        }

        while (true ){
            displayHomeScreen();
            displayMenuItem();

            int choice = getMenuOption(5);
            system("cls");
            int gameState;
```

D:\Pika_New\Project\Pikachu Onet Connect\Pikachu Onet Connect\main.cpp C/C++ Windows (CR+LF) WINDOWS-1252 Line 11, Col 25, Pos 261 Insert Read/Write default

Figure 7: Project Opened

- Step 5: Before initiating compilation and launching the game, change *Debug* to *Release* and click the **Rebuild** button (two blue arrows which form a cycle). After that, locate and click the **Build and Run** button on the toolbar. This button is typically identified by an icon depicting a gear symbol interlocked with a play symbol.

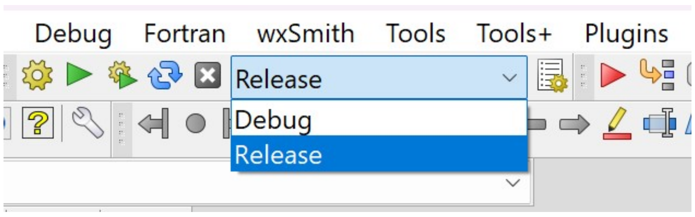


Figure 8: Changing Debug To Release

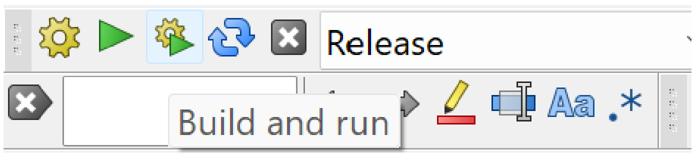


Figure 9: Build And Run Button

- Step 6: On the menu screen, there will be 3 options for players: Log In, Sign Up or Exit Game. Select Sign Up to create a new account, or select Log In if players already have an account.

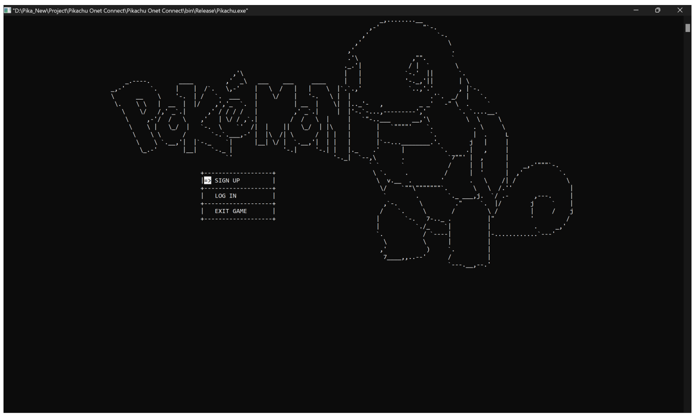
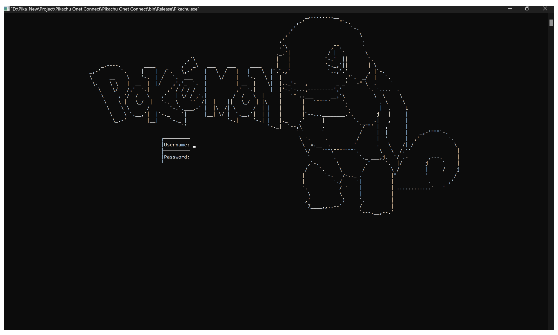

"D:\Pika_New\Project\Pikachu Onet Connect\Pikachu Onet Connect\bin\Release\Pikachu.exe"
SIGN UP
LOG IN
EXIT GAME

Figure 10: Home Screen



```
"D:\Pika_New\Project\Pikachu Onet Connect\Pikachu Onet Connect\bin\Release\Pikachu.exe"
Username:
Password:
```

Figure 11: Sign Up/Log In Screen

- Step 7:
  - Upon successful completion of step 5, the game will present with the main menu. This menu offers five distinct options: New Game, Load Game, Tutorial, Leaderboard, and Log Out.

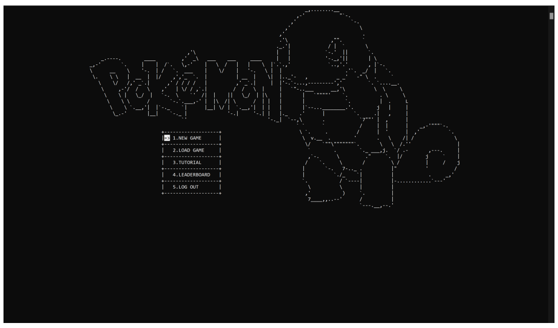
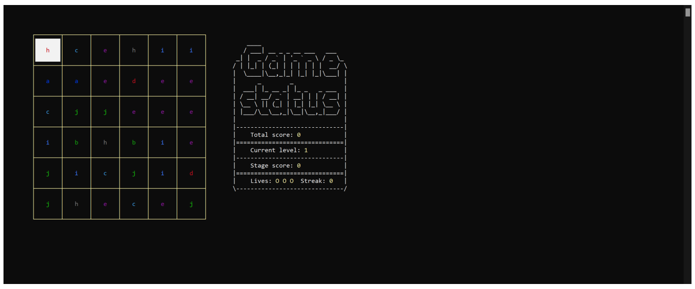


Figure 12: Main Menu Screen

    * New Game: Start a new game.
    * Load Game: Display the game saved by players.

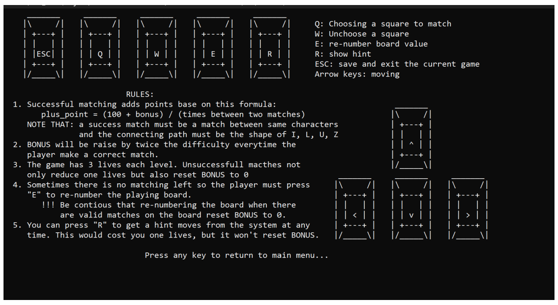


```
h c e h i i
a a e d e e
c j j e e e
i b h b i e
j i c j i d
j h e c e j

Total score: 0
Current level: 1
Stage score: 0
Lives: 0 0 0  Streak: 0
```

Figure 13: Playing Screen

    * Tutorial: Instruct keys, functions as well as requirements for each key when playing games.



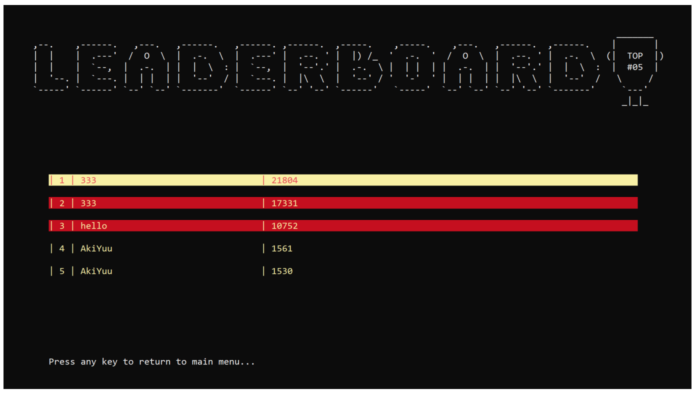
```
Q: Choosing a square to match
W: Unchoose a square
E: re-number board value
R: show hint
ESC: save and exit the current game
Arrow keys: moving

                  RULES:
1. Successful matching adds points base on this formula:
      plus_point = (100 + bonus) / (times between two matches)
   NOTE THAT: a success match must be a match between same characters
              and the connecting path must be the shape of I, L, U, Z
2. BONUS will be raise by twice the difficulty everytime the
   player make a correct match.
3. The game has 3 lives each level. Unsuccessfull macthes not
   only reduce one lives but also reset BONUS to 0
4. Sometimes there is no matching left so the player must press
   "E" to re-number the playing board.
      !!! Be contious that re-numbering the board when there
          are valid matches on the board reset BONUS to 0.
5. You can press "R" to get a hint moves from the system at any
   time. This would cost you one lives, but it won't reset BONUS.

                    Press any key to return to main menu...
```

Figure 14: Tutorial Screen

  - ∗ Leaderboard: Display the top 5 players who has the highest total score.



```
| 1 | 333                          | 21804
| 2 | 333                          | 17331
| 3 | hello                        | 10752
| 4 | AkiYuu                       | 1561
| 5 | AkiYuu                       | 1530

Press any key to return to main menu...
```

Figure 15: Leaderboard Screen

  - ∗ Log Out: Log out of player's account and return to home screen.

  - Select New Game or Load Game and use the keys in the Tutorial to play the game.

- Step 8: When finished playing, select Log Out and Exit Game to exit the game.

Game's tutorial video: https://youtu.be/EES5XeOzoLg

3 Features

3.1 Game starting:

3.1.1 Full Screen

Upon game initialization, the **fullSC**[4] function will be executed. This function serves the purpose of maximizing the dimensions of the console window, thereby enhancing the user's gameplay experience by providing a more expansive viewing area.

3.1.2 Cursor Setting

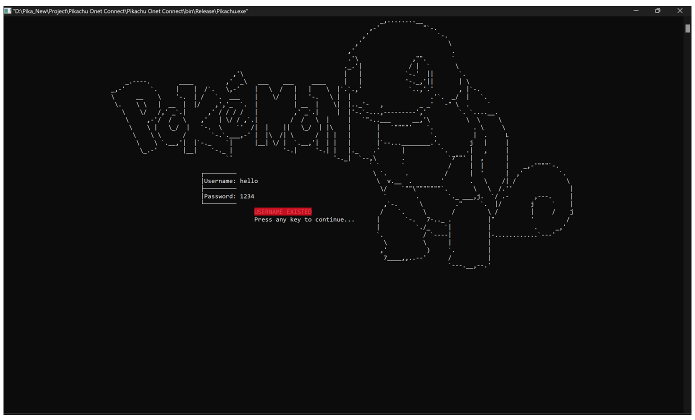
To streamline player interaction within the game environment, the **setCursorPosition**[5] function will be employed upon game entry.

This function ensures that the cursor is precisely positioned at designated locations on the screen, thereby facilitating intuitive gameplay and minimizing the need for excessive cursor movement by the player.

3.1.3 Log In and Sign Up

After maximizing the dimensions of the console window, players will be presented with three distinct choices by **displayRegAndLogBlock** function: establish a new account, utilize an existing account, or terminate the application.

- **Sign up:**
  - When the player selects the sign up button, the system will call **displayNameAndPassBlock** function to display the block of *username and password.*
  - After that, the system will call **getUserInfo** function, which enable them to create an account and password.
  - The system calls the **saveAccount** function to verify the uniqueness of the chosen username. Following the completion of user credential verification, the **displaySignUpResult** function will be invoked to present the outcome of the registration process to the user.
    - In the event that the username already exists within the system, the function returns a negative acknowledgement *(false)*. Consequently, the system generate a notification informing the player of the username conflict *("Username Existed")* and prompt them to establish a new account.

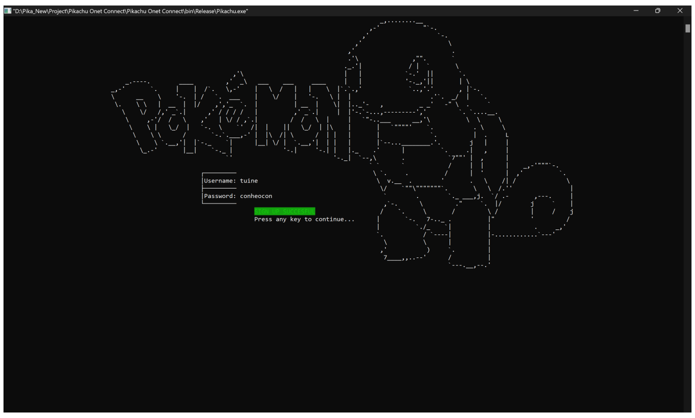


```
"D:\Pika_New\Project\Pikachu Onet Connect\Pikachu Onet Connect\bin\Release\Pikachu.exe"

Username: hello
Password: 1234
          USERNAME EXISTED
          Press any key to continue...
```

Figure 16: Username Existed

    - Conversely, if the username is deemed unique, the system will informing a successful sign-up notification (*"Sign Up Successful"* permanently store the player's credentials within a designated file named **"AccountList.txt"**. This file serves as a secure repository for authentication purposes during subsequent login attempts, and the function returns a positive acknowledgement *(true)*.

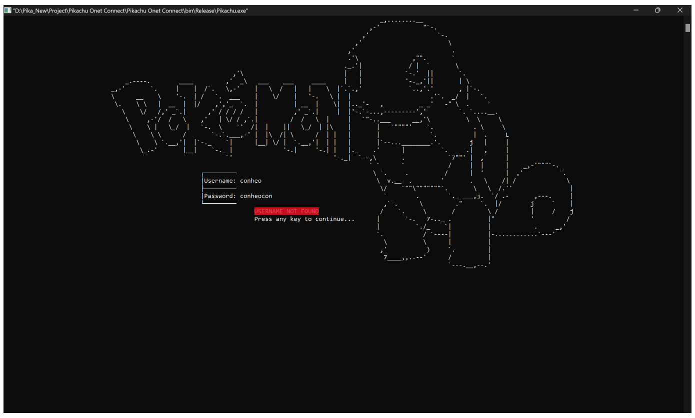


```
"D:\Pika_New\Project\Pikachu Onet Connect\Pikachu Onet Connect\bin\Release\Pikachu.exe"

Username: tuine
Password: conheocon
        SIGN UP SUCCESFUL
        Press any key to continue...
```

Figure 17: Sign Up Successful

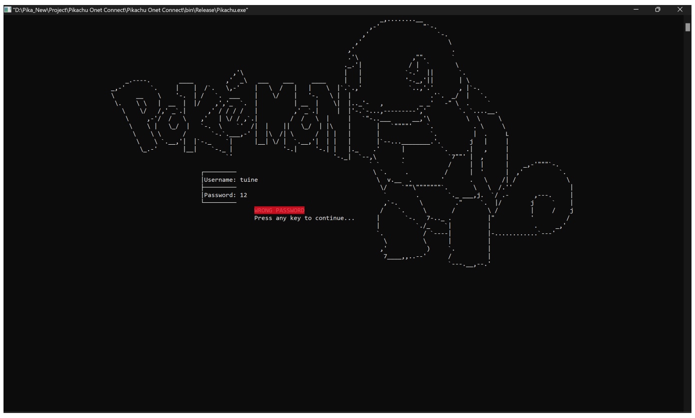
- **Log in:**
  - Players possessing a pre-existing account will proceed by selecting the designated login button.
  - Subsequently, the system will invoke the **getUserInfo** function, prompting the player to furnish their account credentials.
  - Following this, the **accountVerification** function is then called upon to verify the accuracy of the provided account information. Subsequently, the **displayLogInResult** function will be employed to communicate the login attempt's success or failure to the player.
    - If the player provide incorrect username credentials or an invalid password, the function will return a negative acknowledgement *(false)*. Afterwards, the system will generate a notification informing the player of the authentication error *(”Username Not Found” or ”Wrong Password”)*, and prompting them to re-enter their credentials.



```
"D:\Pika_New\Project\Pikachu Onet Connect\Pikachu Onet Connect\bin\Release\Pikachu.exe"

Username: conheo
Password: conheocon
        USERNAME NOT FOUND
        Press any key to continue...
```

Figure 18: Username Not Found

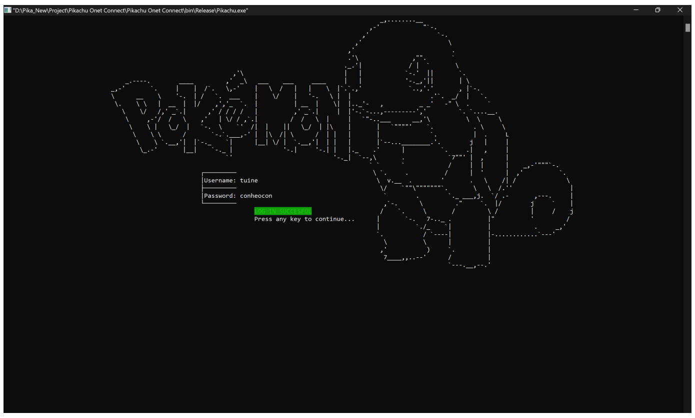


```
"D:\Pika_New\Project\Pikachu Onet Connect\Pikachu Onet Connect\bin\Release\Pikachu.exe"

Username: tuine
Password: 12
        WRONG PASSWORD
        Press any key to continue...
```

Figure 19: Wrong Password

* If the player's credentials are successfully validated, the function will return a positive acknowledgement *(true)*. The system will provide the player with a confirmation message ("Log In Successful") and subsequently grant access to the game screen.



```
"D:\Pika_New\Project\Pikachu Onet Connect\Pikachu Onet Connect\bin\Release\Pikachu.exe"

Username: tuine
Password: conheocon
        LOG IN SUCCESFUL
        Press any key to continue...
```

Figure 20: Log In Successful

- **Exit game:** The player's selection of the exit option will prompt the program to terminate its execution and subsequently close the application window.

3.1.4 Game's Main Menu

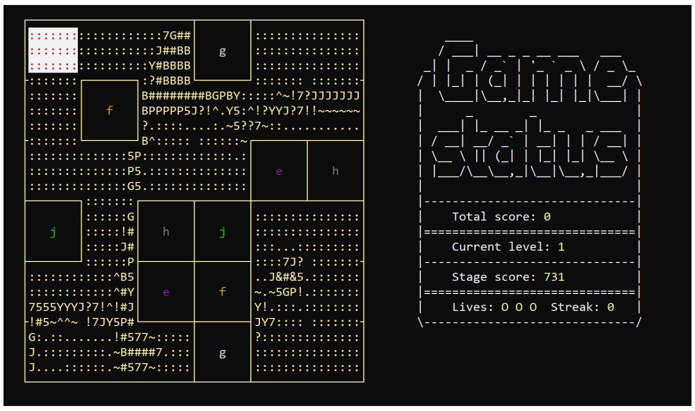
- Following successful login or registration, the player will be presented with the game's main menu.
- This menu offers a selection of five distinct options, catering to various player preferences: New Game, Load Game, Tutorial, Leaderboard and Log Out.

- New Game: Upon player selection of *New Game*, the **playGame** function is invoked. This function orchestrates the creation of a fresh game session, initializing the game screen and setting the starting level to 1.
- Load Game:In the scenario where a player has previously saved game progress, selecting *Load Game* will present three distinct options:
  * Unfinished level: In the first scenario, if the player previously exited the game during an incomplete level, selecting *Load Game* will trigger the **loadGame** function. This function will reload the partially completed level, allowing the player to resume their progress and attempt to finish it.
  * Level Completion: In the second scenario, if the player has successfully completed a level and subsequently saved their progress, selecting *Load Game* will prompt the **loadGame** function to generate the next level in the sequence. This allows the player to seamlessly progress through the game without repeating completed levels.
  * No Saved Game Found: If the **loadGame** function is unable to locate any saved game data, it will signal this by returning a *"false"* value. The player will then be returned to the main menu, presented with the opportunity to select a different option.

```
NO SAVED GAME FOUND
```

Figure 21: No Saved Games Found Notification

- Tutorial: Selecting the *Tutorial* option triggers the **displayTutorial** function. This function presents on-screen guidance for the player,



```
Total score: 0
Current level: 1
Stage score: 731
Lives: O O O  Streak: 0
```

Figure 22: Loading Saved Game

outlining the controls and gameplay mechanics through clear instructions.

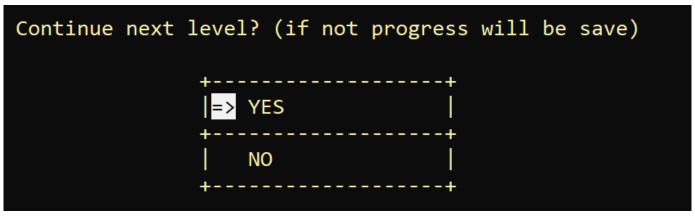
  - Leaderboard: Analogous to the behavior of the **displayTutorial** function, selecting *Leaderboard* invokes the **displayLeaderboard** function. This function serves to display a leaderboard showcasing the top five players with the most impressive cumulative scores.
  - Log Out: Choosing *Log Out* from the main menu initiates the user's session termination. This action triggers the **displayHomeScreen** function, which refreshes the game's interface, presenting the user with the initial home screen.

3.1.5 Save Game

The game leverages the **saveGameData** function to persistently store relevant game data. This function is invoked whenever the game needs to be saved. There are two primary scenarios that trigger game saves:

- Unfinished level: In the event a player terminates a level prematurely (by pressing the *ESC* key) without achieving completion, the system nonetheless retains the current state of the level for subsequent gameplay sessions.

- Level Completion: Upon completion of each level, the **displayYesNoQuestion** function prompts the player to indicate their desire to proceed to the subsequent level by presenting a yes or no question. The player's selection, captured by the **getYesNoQuestion** function, determines the next course of action. If the player chooses *No* (equivalent to a *false value*), the **saveGameData** function records the successful completion of the current level. Subsequently, the system generates the next level for the player's future gameplay session.



```
Continue next level? (if not progress will be save)

                       +--------------------+
                       |=> YES              |
                       +--------------------+
                       |   NO               |
                       +--------------------+
```

Figure 23: Yes/No Question

3.2 Matching Features

The game offers four distinct match types to match two elements having the same value: I, L, U, and Z. To facilitate the Matching functions within the game, a data conversion will be implemented, transforming elements from their character type representation to an integer type for processing and vice versa.

3.2.1 I Matching

The I Matching function is a *Boolean function* that utilizes pointer variables or linked lists to determine whether a specified connection, adjacent along a vertical or horizontal path, analogous to the shape of a capital 'I', exists between two elements within a grid structure.

- Step 1: First, the function prioritizes efficiency by checking whether the selected elements reside within the same row or column. If neither of these conditions is met, the function concludes that the elements are not connected in the desired I-shape and returns a value indicating this disconnection (often represented by *"false"* in programming languages).
- Step 2: In the event that elements are located within the same row or column, the function evaluates each intervening element along that row or column.
  - If any element in between has a value (meaning it's not empty), the I-shape connection fails, and the function returns false.
  - In the scenario where all intermediate elements within the same row possess a value of zero, as determined by the function's examination, the function returns a positive acknowledgement (*true*). This signifies the possibility of establishing a valid I-shaped connection between the two elements.

3.2.2 L Matching

Similar to the I Matching function, the L Matching function is a Boolean function that uses pointer variables or linked lists to check for an 'L' shaped connection between two molecules in a two-dimensional array. To facilitate differentiation, we will denote the element occupying the lower row index as A and the element in the higher one as B.

- Step 1: First, The I Matching function will be used to check whether element A can be matched with an element that has the same row index as A and the same column index as B (temporarily called element C). Concurrently, the verification process determines whether the value is equal to zero or not.
- Step 2:
  - The failure of either condition to be met will result in the termination of the evaluation process within the function and the return of a false value.
  - If both conditions hold, the I Matching function is recalled to check the connection between B and C.
    - If the I Matching function finds a connection between B and C, the L Matching function immediately returns *true*, indicating a L-shape connection between A and B.
    - Otherwise, the entire operation concludes by returning *a false value.* This indicates that A and B are not connected in an L-shape.

3.2.3 U and Z Matching

Similar to the two functions above, the U And Z Matching function is also used to check whether an element with a lower row index (called A) and an element with a higher row index (called B) exists an U-shaped or Z-shaped connection. n light of the significant similarity between the U Matching and Z Matching algorithms, their algorithm will be merged into a singular function.

- Step 1: First, an iterative process will be conducted to examine all elements within the row containing element A. The objective is to identify element C, which possesses a value of zero and satisfies the connectivity criteria of forming an I-shaped line with element A.
- Step 2:
  - Upon successfully locating element C, the function will employ the I Matching function twice to ascertain the connectivity between C and both elements B and D (where D resides in the same row as B and the same column as C).
    - If both B and C can connect to D, the function will return a value of *true*, signifying the possibility of connecting elements A and B via a U or Z-shaped configuration.
    - Otherwise, the iterative process will persist in examining elements within the row until it reaches a point where no elements C and D can be identified that fulfill the established criteria.
- Step 3: In the event that element C is not located within the row containing element A, the search will be redirected to examine elements within the same column as A. This process will mirror step 1, implying a repeated application of the initial search criteria.
- Step 4:
  - If both B and C can connect to D, the function will return a value of *true*, signifying the possibility of connecting elements A and B via a U or Z-shaped configuration.
  - Otherwise, the iterative process will persist in examining elements within the row until it reaches a point where no elements C and D can be identified that fulfill the established criteria.

3.3 Game Finishing Verify

Two cases for verification is: Level Completion and Game Completion

3.3.1 Level Completion

- The determination of level completion is implemented within two distinct functions. The **basicStage** function handles this logic for *standard levels (levels 1-3)*, while the **hardStage** function is responsible for evaluating *advanced level* completion *(levels 4-5)*.

- With the ability to change the game board (a feature detailed later in this document), the level concludes only when all cells have been successfully connected.

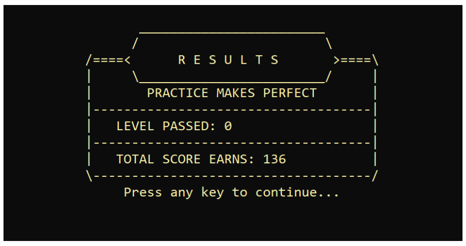
3.3.2 Game Completion

- The **playGame** function is responsible for assessing the overall game state, including determining game over conditions.
- The game terminates in one of three distinct outcomes contingent upon player performance: Lost, Victory and Perfect Victory.

3.3.3 Result

The presentation of results will entail three distinct scenarios:

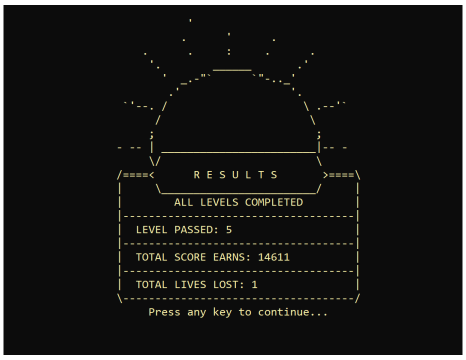
- **Lost:** Upon depletion of the player's lives, signifying a loss condition, the game will transition to a results screen. The displayed information will encompass a motivational message (*"Practice makes perfect"*), the number of levels successfully completed by the player, and their total accumulated points.



```
       ________________________
      /                        \
/====<      R E S U L T S       >====\
|     \________________________/     |
|       PRACTICE MAKES PERFECT       |
|------------------------------------|
|   LEVEL PASSED: 0                  |
|------------------------------------|
|   TOTAL SCORE EARNS: 136           |
\------------------------------------/
     Press any key to continue...
```

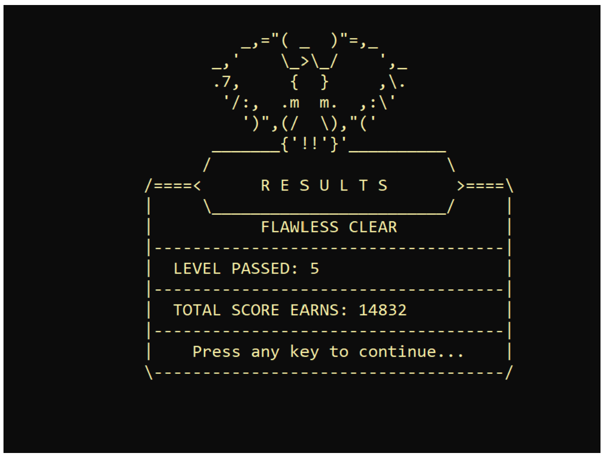
Figure 24: Lost Result

- **Victory:** In the event the player successfully navigates all five rounds (achieving a victory condition), but sustains some life loss during gameplay, the game will transition to a results screen. This screen will be visually distinguished by a sun frame and will display the following information: a congratulatory message (*"All levels completed"*), the number of rounds completed, the player's total score, and the total number of lives lost.
- **Perfect Victory:** Upon achieving a flawless victory by successfully navigating all five rounds without incurring any lives loss, the game will transition to a results screen distinguished by a phoenix frame. This screen will display a message of exceptional achievement (*"Flawless clear"*), alongside the number of rounds completed, the player's total accumulated points, and a celebratory acknowledgment of their perfect performance (*"No lives lost"*).



```
                  '
                .    '    .
          .     .    :    .     .
           '.      ______      .'
             '  _.-"`      `"-.._'
              .'                '.
      `'--. /                      \ .--'`
           /                        \
          ;                          ;
    - -- |____________________________|-- -
         \/                            \
    /====<         R E S U L T S        >====\
    |     \____________________________/      |
    |           ALL LEVELS COMPLETED          |
    |-----------------------------------------|
    |  LEVEL PASSED: 5                        |
    |-----------------------------------------|
    |  TOTAL SCORE EARNS: 14611               |
    |-----------------------------------------|
    |  TOTAL LIVES LOST: 1                    |
    \-----------------------------------------/
          Press any key to continue...
```

Figure 25: Victory Result



```
             _,="( _  )"=,_
          _,'    \_>\_/    ',_
          .7,     {  }     ,\.
           '/:,  .m  m.  ,:\'
             ')",(/  \),"('
          ______{'!!'}'__________
         /                      \
    /====<     R E S U L T S      >====\
    |     \______________________/     |
    |           FLAWLESS CLEAR         |
    |----------------------------------|
    |  LEVEL PASSED: 5                 |
    |----------------------------------|
    |  TOTAL SCORE EARNS: 14832        |
    |----------------------------------|
    |    Press any key to continue...  |
    \----------------------------------/
```

Figure 26: Perfect Victory Result

3.4 Color Effects[1]

- Upon initialization of the game, the visual components encompassing the game frame, and the game status frame will be presented in yellow.
- In the following analysis, correctly identified pairs of cells will be distinguished with green highlighting. Conversely, incorrectly identified pairs will be highlighted in red for easy recognition.
- To do this, the **GetStdHandle** function and the **SetConsoleTextAttribute** function will be used to print characters and other elements in the desired color.

3.5 Sound Effects

To incorporate various auditory elements into the game, such as background music, sound effects for player actions (moving), and audio cues for success or failure (right or wrong sounds), **PlaySound**[2] function will be employed. This function will load and play sound effects in *WAV* format.

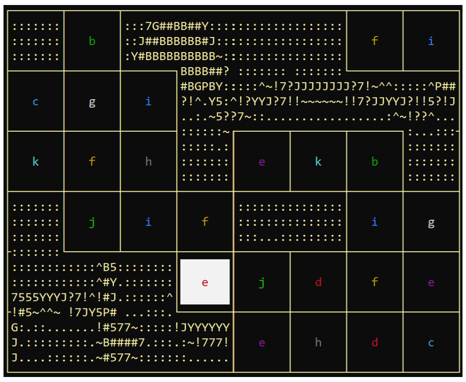
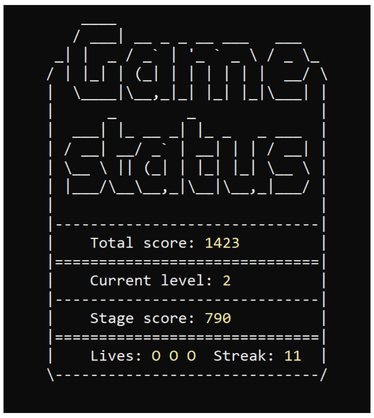


Figure 27: Color Effects

3.6 Visual Effects

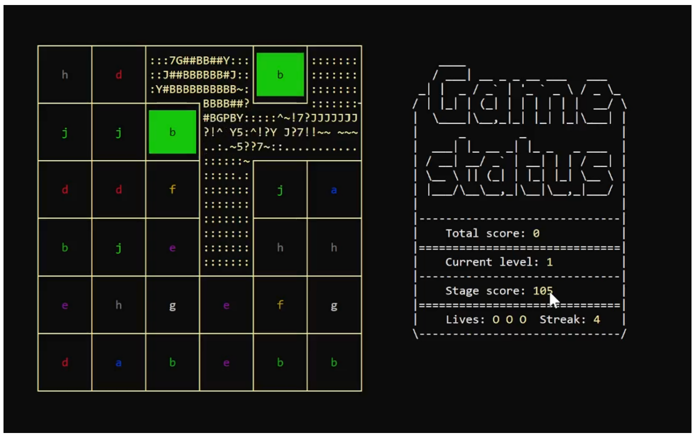
- **Game Status:** The dedicated functions is employed to dynamically generate a visual representation of the player's game status within each level.



```
Total score: 1423
Current level: 2
Stage score: 790
Lives: O O O   Streak: 11
```

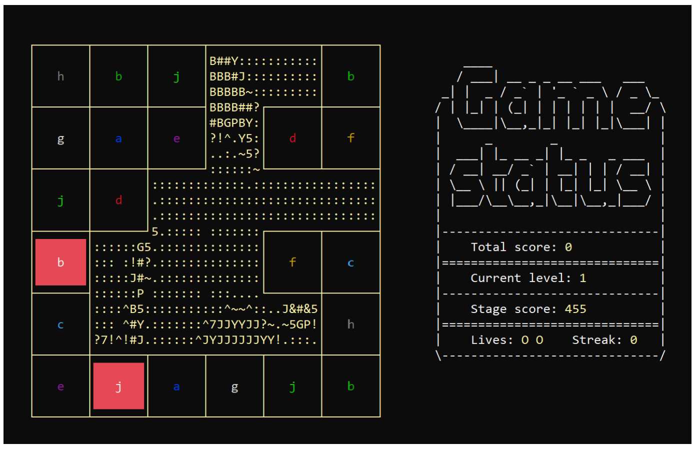
Figure 28: Game Status

- **Console Cursor Hiding:** Each individual console is equipped with its own dedicated cursor. To achieve a cleaner visual presentation and eliminate potential distractions, the default cursor associated with the console is concealed through the utilization of the **hideCursor**[3] function.
- **Game Cursor:** A white cursor will visually highlight the cell it traverses within the game. The selected cell will undergo a color change to gray, and the corresponding character displayed within the cell will transition from yellow to white.
- **Matches status notification:** upon finished choosing 2 cells to match, the algorithm will verify whether it is a valid match or not. Results of this validation will be noticed to the player via coloring the two selected cells: green stands for a valid match and red means the opposite.



```
Total score: 0
Current level: 1
Stage score: 105
Lives: 0 0 0  Streak: 4
```

Figure 29: Two selected cells turns green when the match is valid

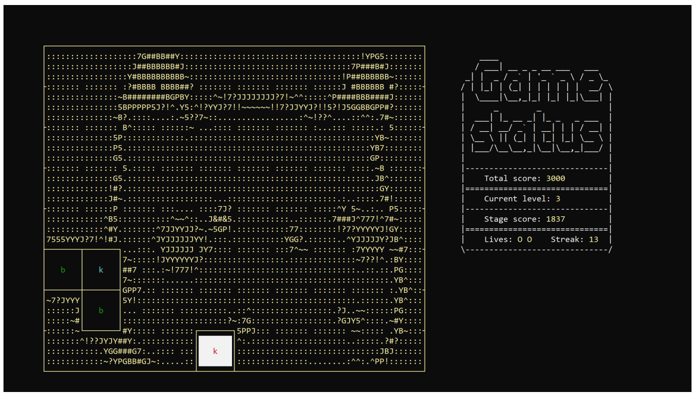


```
Total score: 0
Current level: 1
Stage score: 455
Lives: 0 0    Streak: 0
```

Figure 30: Two selected cells turns red when the match is invalid

3.7 Background

- To achieve the progressive background reveal, we will leverage the **printField** function.
- Due to the implementation of type casting between integer and character data types, a two-dimensional integer array is dynamically allocated to efficiently store the background information underlying the game frame.
- The game employs a progressive reveal mechanic for the background displayed behind the game frame.
  - Each successful connection established between two matching letters results in the unveiling of a corresponding portion of the background image.
  - The complete revelation of the entire background is contingent upon the successful connection of all letter pairs.



```
Total score: 3000
Current level: 3
Stage score: 1837
Lives: O O    Streak: 13
```

Figure 31: Background

3.8 Leaderboard (Figure 15)

- **Format:** The leaderboard prominently displays the top five players who have achieved the highest scores within the game. This leaderboard provides detailed information for each player, including their rank position, player name, and their total score.

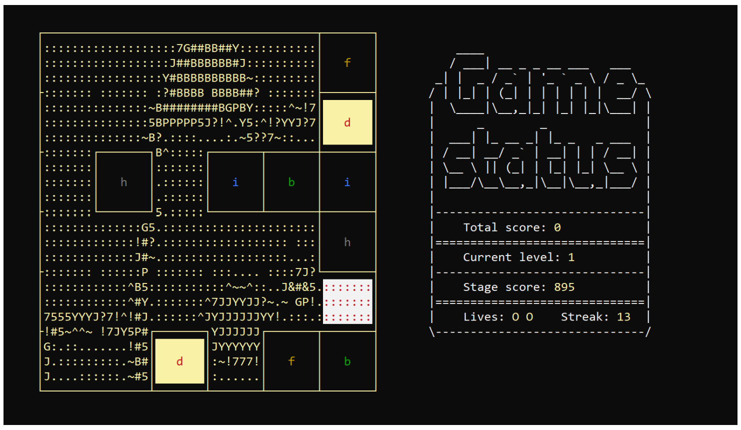
- Both informations will be persistently stored within a *text file*. This approach facilitates the convenient updating of the ranking information whenever necessary.

- To visually distinguish the top performers, the ranking information for the three players with the highest scores will be presented using a color-coded scheme. The player occupying the top rank (*1st place*) will have their information displayed in gold, while the information for the players in the subsequent ranks (*2nd and 3rd*) will be colored red. This color scheme will be implemented by leveraging the functionalities established within the **Color Effects** section.

3.9 Move Suggestion

- Upon player selection of a hint (by pressing "E" button on keyboard), the game run the corresponding code snipppets inside game stage function, **basicStage()** and **hardStage()**, to give suggestion for the next move by highlighting the appropriate cells or print a notification "No valid matching, please reset" when there is no more valid match. In addition, both highlighted color and text color are yellow in background, and the text is red.

- Opting to utilize a hint during gameplay will incur a penalty in the form of *one life lost*. It is important to note that this penalty does not extend to a deduction of bonus points. However, to prevent an excessive reliance on hints, the system is designed to restrict hint availability once the player's remaining lives are reduced to a single unit.



```
Game Status

Total score: 0
Current level: 1
Stage score: 895
Lives: O O    Streak: 13
```

Figure 32: Game suggestion

3.10 Playing Board Changing

- In the scenario where a player reaches a point of strategic impasse, having exhausted all possible moves within the current game board configuration, they will have option to initiate a board refresh.

- The newly generated board will maintain the same character count. In essence, the number of characters on the new board will directly correspond to the number of characters remaining on the old board after all possible matches have been made.

- Similar to Move Suggestion, changing tables is also included in the **basicStage** and **hardStage** functions.

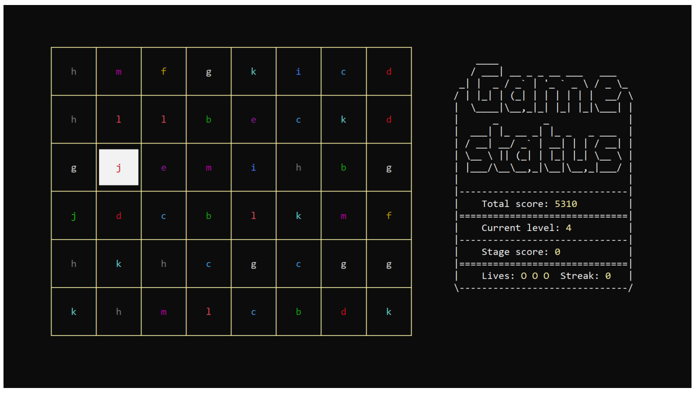
- It is important to emphasize that electing to change the game board while valid moves still exist on the current board incurs a penalty. This penalty entails the forfeiture of any accumulated bonus points and the resetting of the player's streak value to *zero*.

3.11 Level's Difficulty Increase

In the original game, difficulty of each stage is determined by 2 attriubute, eiter the size of the board or the element diversity. In our game, as level increases, both of these element increases. Furthermore, to build more interest, when reached a certain game level, sliding mode will be activated and it will slide all element of a row to right whenever there is a square space.

4 Comparison Between Pointer And LinkedList

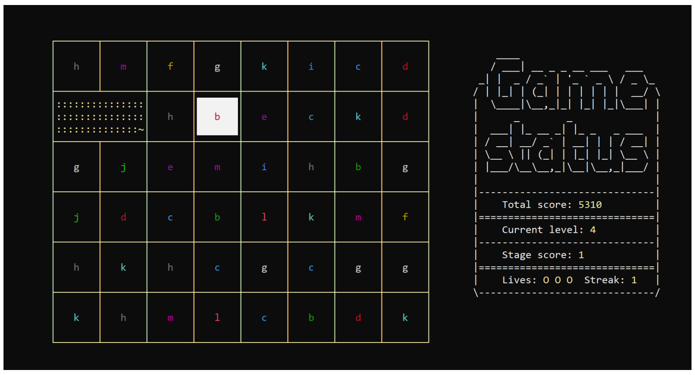
Overall, both pointers and linked lists have their advantages and drawbacks, which vary depending on the specific circumstances.



```
h  m  f  g  k  i  c  d
h  l  l  b  e  c  k  d
g  j  e  m  i  h  b  g
j  d  c  b  l  k  m  f
h  k  h  c  g  c  g  g
k  h  m  l  c  b  d  k
```

```
|------------------------------|
|   Total score: 5310          |
|==============================|
|   Current level: 4           |
|------------------------------|
|   Stage score: 0             |
|==============================|
|   Lives: O O O  Streak: 0    |
\------------------------------/
```

Figure 33: Before Sliding



```
h  m  f  g  k  i  c  d
      h  b  e  c  k  d
g  j  e  m  i  h  b  g
j  d  c  b  l  k  m  f
h  k  h  c  g  c  g  g
k  h  m  l  c  b  d  k
```

```
|------------------------------|
|   Total score: 5310          |
|==============================|
|   Current level: 4           |
|------------------------------|
|   Stage score: 1             |
|==============================|
|   Lives: O O O  Streak: 1    |
\------------------------------/
```

Figure 34: After Sliding

4.1 Pointers

An obvious advantage of pointers is that dynamically allocated arrays support random access to their values. This speeds up the process of retrieving a value at a specific address when checking for valid matches, reducing the time complexity to constant time. On the other hands, when it comes to inserting and removing an element from an array, this method reveals its weakness.

4.1.1 Pros

- Dynamically Allocated Arrays using pointers store elements contiguously. This allows for faster calculation of addresses, resulting in quicker access to elements for retrieval or updates (also known as 'Random Access')
- Pointers proves its effectiveness in certain levels of game in this project when there are not any value deletion or insert to the game board.

4.1.2 Cons

- As mentioned above, using pointers (as in dinamically allocated array) may leads to inconvenienced when inserting and removing. This is because we have to move each element on the array one by one to its new position after the remove or insertion of another element; precisely, this process took linear time to finish.

4.2 Linked List

Linked list, on the other hand, are constructed completly different. It consists of a series of nodes, which are connected by pointers. As a result of this, linked list has a completely different strength and weakness compare to pointers.

4.2.1 Pros

- As been created from pointers that links nodes together, removal and insertion on linked list is more straightforward. The procedure of removing and adding new element to the list only involves adjusting pointers, without the need to shift other elements. This makes removing and inserting costs constants time. This advantages lay the foundation for our sliding features in harder stages of the game.

4.2.2 Cons

- Due to its node-linking construction, linked list do not support Random Access. This worsen the time consumption of a valid match checking process.

5 Project Format Explanation

5.1 Libraries

5.1.1 `<chrono>`

`<chrono>` directive allows you to use the **C++11 standard library for time-related operations**, which provides facilities to work with **time duration** and **time points**. To be specific, in this project, time duration is used to set the delaying time for some game notifications, and time points are used to calculation between two matches (for scoring purposes).

5.1.2 `<thread>`

`<thread>` gives access to the **C++11 threading library**.Specifically, it provides functions for managing threads, such as creating threads, joining them, and controlling their execution. `<thread>` is commonly used to **introduce a delay or pause in the program execution**, this is also the reason we use it in this project.

5.1.3 `<ratio>`

`<ratio>` header in C++ provides a set of standard ratio types. These types allow precise representation of fractional relationships using integers. Including `<ratio>` is considered a best practice when working with time-related code. It ensures that your **time calculations are precise, maintainable, and**

consistent. While the code itself does not directly use ratios, it leverages the ratio library functionality indirectly through the `<chrono>` library. In other words, while removing ratio will not necessarily break the code, it is advisable to keep it for accurate time computations and to adhere to best practices.

5.1.4 `<stdlib.h>`

`<stdlib.h>` header provide several general-purpose functions that include **generate a random numbers**. This is crucial in this project since we need random numbers to set up the game board.

5.1.5 `<ctime>`

`<ctime>` allow access to functions and types related to date and time manipulation. Specifically, it includes the **time()** function, which returns the current time as the number of seconds elapsed since a fixed reference point. When you seed the random number generator using the current time (as shown in your code snippet with srand(time(NULL))), it **ensures that each program execution starts with a different seed value**.

5.1.6 `<Windows.h>`

`<Windows.h>` is a Windows-specific header file for the C and C++ programming languages. It contains declarations for all of the functions in the Windows API, which allows you to interact with various aspects of the Windows operating system. In this project functions like **GetConsoleWindow(), ShowWindow(), SetConsoleTextAttribute(), GetStdHandle(), and SetConsoleCursorPosition()** are part of the Windows API. Those afordmentioned functions are used, in combinations, to help displaying the game including background, menu, board, visual effect and also notifications.

5.1.7 `<conio.h>`

The inclusion of `<conio.h>` headers in the project code serves a specific purpose related to console input and output operations. Specifically, function **_getch()** is used in the code for reading a single keyboard inputs from the player without echoing it (i.e., without displaying it on the screen).

5.1.8 `fstream`

The `<fstream>` header provides the necessary tools for **working with file streams**. It defines classes like **ifstream** (for reading from files), **ofstream** (for writing to files). In the project, file streams are used in tasks like **loading** game state, background, game board designs, game tutorial, ascii pictures from files to display on screen, and **saving** accounts information, game state, highscore by writing it to files in some predetermined format.

5.1.9 `<iostream>`

The `<iostream>` header is an essential part of the C++ Standard Library.It provides functionality for input and output operations using streams. Specifically, it defines the standard input stream (std::cin) and the standard output stream (std::cout) that allows us to **read input** from the user and **display output** to the console.

5.1.10 `<string>`

`<string>` header provides essential functionality for working with strings in C++. It defines the string class, which allows you to **create, manipulate, and manage character strings** efficiently. Many aspect of the code requires using string, for instance, inpuing and saving account names, or loading menu title.

5.1.11 `<mmsystem.h>`

The `<mmsystem.h>` serves a specific purpose in C++ programs when dealing with multimedia and sound-related functionality. It is is a Windows-specific header that provides access to the Multimedia API. It allows you to work with multimedia features such as audio, MIDI, and other sound-related operations. In the project the **PlaySound()** is used to play various game sound effect and background music.

5.2 Header Files (.h)

5.2.1 `<Display.h>`

This header file contains function to handle task related to printing/displaying to the console screen. Starting with the fundamentals functions including setting the printing cursor position and changing console printing color. Building up from there, other effects, visual features are created: from displaying the game outer look, to printing menus, questions, notifications, changing color of cells, etc.

5.2.2 `<Login.h>`

The header files consists of functions that handle account access. This is where the sign up and log in procedure at the beginning of the game came from. `<Login.h>` can take log in (or sign up) data, which are username and password, and verifies it. If a new account is created, its log in data is saved for future log in.

5.2.3 `<Sound.h>`

`<Sound.h>` header contains functions to play the sound for specific events/notifications in the game. For example, the sound effect of a cursor moving, of a correct/wrong match or the home screen background music.

5.3 `<Highscore.h>`

Code that corresponding to updating and manipulating ranking order the in the leaderboard is located in this header file.

5.3.1 `<Menu.h>`

`<Menu.h>` contains function that handle taking the menu choices and returning them for other task to response properly.

5.3.2 `<LinkedList.h>`

A specify header that implement operations on Linked Lists that are used in the harder stages of the game.

5.3.3 `<Gameplay.h>`

`<Gameplay.h>` header is where the game logic located. There are functions to help runs the game, for example: function loadGame() to load the game depends on whether its a new level or read the saved from a file; getKey(), which take a legal input from the player; function that check a legal moves checkMatching(); the game logic of game stage depends on the difficulty are set in 2 functions basicStage() and hardStage(), while playGame() ensures score save and the game difficulty gets progressively harder every stage.

5.4 Source Code Files (.cpp)

5.4.1 main.cpp

This primary source file encapsulates all essential libraries and header files required for game functionality. Additionally, it houses the program's entry point, the main function, responsible for invoking the various functions that orchestrate the game's execution.

5.4.2 Other Source Files

Other source files each possess a corresponding header file that shares the same name. These header files primarily serve as declarations for the functions utilized within the game. The implementation details for these functions, encompassing the specific code that governs their behavior, are housed within the corresponding source file.

6 Resources and reference

Resources

- Background picture - DeviantArt. https://www.deviantart.com/8-bitpixelpony/art/Pikachu-354930937
- Ascii art - ASCII Art Archive. https://www.asciiart.eu/video-games/pokemon
- Ascii text generator - ASCII Art Archive. https://www.asciiart.eu/text-to-ascii-art
- Home screen background music - Pixabay. https://pixabay.com/music/video-games-music-for-arcade-style-game-146875/

- Wrong match sound - Pixabay. https://pixabay.com/sound-effects/fart-83471/
- Correct match sound - Pixabay. https://pixabay.com/sound-effects/news-ting-6832/
- Move sound - Pixabay. https://pixabay.com/sound-effects/book-foley-finger-slide-3-189800/

References

[1] Colorizing text in the console with C++ — stackoverflow.com. https://stackoverflow.com/a/4053879. [Accessed 16-04-2024].

[2] PlaySound in C++ Console application? — stackoverflow.com. https://stackoverflow.com/questions/9961949/playsound-in-c-console-application. [Accessed 16-04-2024].

[3] Remove blinking underscore on console / cmd prompt — stackoverflow.com. https://stackoverflow.com/a/18028927. [Accessed 16-04-2024].

[4] Setting Console to Maximized in Dev C++ — stackoverflow.com. https://stackoverflow.com/questions/6606884/setting-console-to-maximized-in-dev-c/6620462#6620462. [Accessed 16-04-2024].

[5] Setting the Cursor Position in a Win32 Console Application — stackoverflow.com. https://stackoverflow.com/a/2732327. [Accessed 16-04-2024].