PROJECT REPORT (Updated Version)

"MEMORY MATCHING GAME USING SFML"

FALL 2024

CSE208L OBJECT ORIENTED PROGRAMMING LAB

SUBMITTED BY :

SYED MUHAMMAD AHSAN JAN (23PWCSE2309)   MIAN

SAEED AHMAD (23PWCSE2319)

MUHAMMAD HUSSAIN (23PWCSE2272)

SECTION : A

SUBMITTED TO :

Engr. Sumayyea Salahuddin

(January 27, 2025)

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar
Memory Matching Game using SFML

Objective

The objective of this lab was to develop a memory matching game using the Simple and Fast Multimedia Library (SFML) in C++. The game challenges players to match cards based on their colors, testing their memory and cognitive skills. The game progresses through multiple levels, with increased difficulty based on the number of mistakes and time limitations.

Tools and Technologies

- SFML (Simple and Fast Multimedia Library): A multimedia library used for creating games and interactive applications.

- C++: Programming language used for developing the game logic.

- Font: The "arial.ttf" font was used to display text in the game.

- C++ Standard Library: Utilized for random number generation, vector manipulation, and time handling.

About Game (Levels , Cards Grid and Color Randomization)

1. Levels and Progression

    - The game progresses through multiple levels, starting from level 1.

    - Completing a level involves matching all pairs of cards correctly.

    - Levels become more challenging due to reduced card display times and increased hardship levels.

    - If the player reaches level 10 successfully, a special message, "YOUR MEMORY IS A MIRACLE," is displayed, marking the end of the game.

2. Color Assignment and Randomization

    - Each level's grid contains paired cards with random colors.

    - The initializeCards function creates a list of random colors for card pairs, ensures there are two cards of each color, and then shuffles the color list to randomize card placement.

    - Random colors are generated using rand(), creating variability in card colors every game. 3.

      Hardship Levels

    - The hardship increases after every three levels.

    - Hardship is primarily reflected in the reduced time cards remain face-up for the player to memorize their positions.

- At the beginning of the game, cards are visible for 5 seconds. This time decreases incrementally with advancing levels.

4. Total Levels and Game Over

   - The maximum level is 10. Completing it successfully ends the game with a congratulatory message.

   - The game ends prematurely if the player accumulates 10 mistakes or their score falls below -20.

5. Cards Grid

   - The game uses a 4x4 grid with 16 cards, calculated based on the GRID_SIZE constant (set to 4).

   - Each card is 80x80 pixels in size.

   - The cards are initially placed face down and shuffled randomly.

   - Cards are created using SFML RectangleShape objects, with their positions calculated according to their index in the grid.

   - The grid's total width and height depend on the GRID_SIZE and card dimensions.

   - The card grid remains the same throughout the game, with the layout being consistent in every level.

6. Comments

   - The game displays dynamic comments based on player performance.

   - Positive comments like "You're on fire!" and "Keep it up!" are shown when the player performs well.

   - Negative comments such as "Oops, focus harder!" are shown when the player makes mistakes.

   - The comment displayed is determined by the player's score and mistakes.

   - Comments are displayed in cyan-colored text below the gameplay information (score, level, etc.).

   - Comments are intended to motivate and provide feedback, adding a fun and interactive element to the game.

Game Overview

1. Game Mechanics:

   o   The game consists of a grid of cards, initially faced down.

   o   Players click on cards to flip them over, trying to find matching pairs. o   After a certain

       time, the cards are flipped back to their original state.

   o   If the player makes a correct match, the score increases; if the player makes a mistake, the
       score decreases.

   o   The game progresses through levels with increasing difficulty, which is determined by the
       number of mistakes and the time left.

2. Hardship Levels:

   o   As the player advances through levels, the hardship level increases, which affects the
       duration the cards are displayed before being flipped back over. Initially, the cards are
       displayed for 5 seconds, but the time decreases to 4, 3, and so on with each new level.

3. Game Over Condition:

   o   The game ends if the player runs out of time, makes too many mistakes, or reaches level
       10.

   o   If the player completes level 10, the game displays a message stating "YOUR MEMORY
       IS A MIRACLE" and ends.

4. User Interface:

   o   The game provides feedback such as the score, level, mistakes, and other relevant
       information on the screen.

   o   Encouraging or humorous comments are displayed based on the player's score and
       mistakes.

Object-Oriented Programming (OOP) Connection

The game design incorporates several key Object-Oriented Programming (OOP) concepts to structure and
organize the code effectively. Here's a breakdown of how these principles are used in the game:

1. Encapsulation:

   o   The Card class encapsulates all the properties and behaviors related to a card. It stores the
       card's shape, color, flipped state, and its corresponding border. The behavior of flipping
       the card and drawing it to the screen is also encapsulated in the class through methods
       like flip() and draw(). This allows the Card class to maintain its internal state while
       exposing only the necessary functionality to other parts of the game.

Example:

```cpp
cpp                                                    Copy   Edit

class Card {
public:
    sf::RectangleShape shape, border;
    sf::Color originalColor;
    bool flipped;

    // Constructor, flip method, and draw method to manage card's behavior
};
```

2. Abstraction:

☐ The game code abstracts complex card behaviors into simple operations like flip() and draw().
   For instance, when the player clicks on a card, the game doesn't need to worry about how the
   card's internal shape is rendered or how it flips—those details are handled by the Card class
   itself. The abstraction simplifies the gameplay logic and allows the developer to focus on higher-
   level features.

Example:

```cpp
cpp                                                    Copy   Edit

card.flip();  // Flips the card without needing to know its internal workings
```

Code Explanation

The code is divided into several sections, each focusing on a specific aspect of the game.

1. Card Class:

   o The Card class is used to represent each individual card in the grid. It includes properties
     like the card's shape, color, and whether it has been flipped.

   o The flip method is used to change the card's state from face-down to face-up and vice
     versa.

2. Game Initialization:

   o The initializeCards function generates a shuffled set of cards with random colors,
     ensuring that there are pairs of matching colors.

   o This function sets up the initial grid layout of cards based on the predefined grid size.

3. Display Functions:

o The displayMessage function shows messages like "Game Over" or "Well Done" to the player.

o The displayCardSequence function shows the sequence of cards for a specified amount of time before flipping them back over.

4. Game Loop:

o The game runs within a loop that checks for user input (mouse clicks) to flip the cards.

o If two cards are flipped, the game checks if they match, updating the score and making the appropriate changes to the game state.

o The game tracks the player's progress through levels and displays relevant information such as score, time remaining, and the number of mistakes.

Flowchart
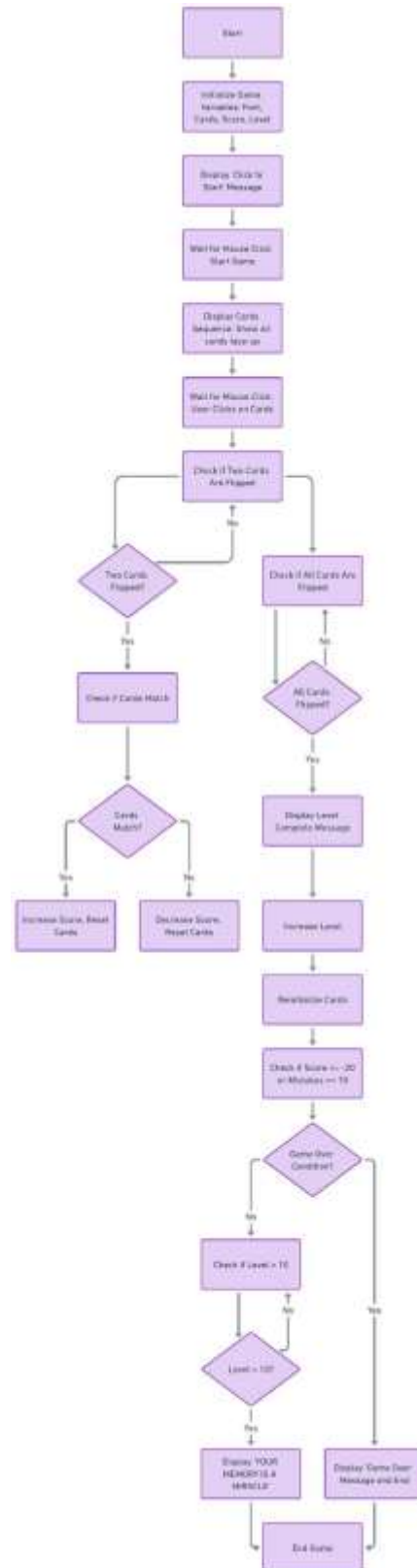
Code Listing

```cpp
#include <SFML/Graphics.hpp>
#include <vector>
#include <ctime>
#include <cstdlib>
#include <iostream>
#include <sstream>
#include <random>

const int GRID_SIZE = 4;
const int CARD_WIDTH = 80;
const int CARD_HEIGHT = 80;
int WINDOW_WIDTH = GRID_SIZE * CARD_WIDTH + 200;
int WINDOW_HEIGHT = GRID_SIZE * CARD_HEIGHT;

class Card {
public:
    sf::RectangleShape shape, border;
    sf::Color originalColor;
    bool flipped;

    Card(int x, int y, sf::Color color) {
        shape.setSize(sf::Vector2f(CARD_WIDTH, CARD_HEIGHT));
        shape.setPosition(x, y);
        shape.setFillColor(sf::Color(169, 169, 169));

        border.setSize(sf::Vector2f(CARD_WIDTH, CARD_HEIGHT));
        border.setPosition(x, y);
        border.setFillColor(sf::Color::Transparent);
        border.setOutlineThickness(4);
        border.setOutlineColor(sf::Color::Black);

        flipped = false;
        originalColor = color;
    }

    void flip() {
        flipped = !flipped;
        shape.setFillColor(flipped ? originalColor : sf::Color(169, 169, 169));
    }

    void draw(sf::RenderWindow &window) {
        window.draw(shape);
        window.draw(border);
    }
};

void initializeCards(std::vector<Card> &cards, std::vector<sf::Color> &colors) {
    cards.clear();
    int numPairs = (GRID_SIZE * GRID_SIZE) / 2;
    colors.clear();

    for (int i = 0; i < numPairs; ++i) {
        colors.push_back(sf::Color(rand() % 256, rand() % 256, rand() % 256));
    }
    colors.insert(colors.end(), colors.begin(), colors.end());
    std::random_shuffle(colors.begin(), colors.end());

    int index = 0;
    for (int i = 0; i < GRID_SIZE; ++i) {
        for (int j = 0; j < GRID_SIZE; ++j) {
            cards.emplace_back(j * CARD_WIDTH, i * CARD_HEIGHT, colors[index]);
            index++;
        }
    }
}
```

```cpp
void displayMessage(sf::RenderWindow &window, const std::string &message, sf::Font &font) {
    sf::Text text(message, font, 30);
    text.setFillColor(sf::Color::White);
    text.setPosition(WINDOW_WIDTH / 2 - text.getLocalBounds().width / 2, WINDOW_HEIGHT / 2 - 30);
    window.clear();
    window.draw(text);
    window.display();
}

void displayCardSequence(sf::RenderWindow &window, std::vector<Card> &cards, int viewTime) {
    for (auto &card : cards) {
        card.flip();
    }
    window.clear(sf::Color::Black);
    for (auto &card : cards) {
        card.draw(window);
    }
    window.display();
    sf::sleep(sf::seconds(viewTime));
    for (auto &card : cards) {
        card.flip();
    }
}

void displayLevelComplete(sf::RenderWindow &window, sf::Font &font) {
    displayMessage(window, "Well done!", font);
    sf::sleep(sf::seconds(2));
    displayMessage(window, "Prepare for next level!", font);
    sf::sleep(sf::seconds(2));
}

void displayGameOver(sf::RenderWindow &window, sf::Font &font) {
    displayMessage(window, "Game Over! Try again!", font);
    sf::sleep(sf::seconds(2));
}


std::string getRandomComment(int score, int mistakes) {
    std::vector<std::string> positive = {"You're on fire!", "Amazing memory!", "Keep it up!"};
    std::vector<std::string> negative = {"Oops, focus harder!", "Don't give up!", "You can do this!"};
    return mistakes > score / 10 ? negative[rand() % negative.size()] : positive[rand() % positive.size()];
}

int main() {
    int level = 1, score = 0, mistakes = 0, hardshipLevel = 1;
    int cardShowTime = 5;

    sf::RenderWindow window(sf::VideoMode(WINDOW_WIDTH, WINDOW_HEIGHT), "Memory Matching Game");
    sf::Font font;
    if (!font.loadFromFile("C:\\Windows\\Fonts\\arial.ttf")) {
        std::cerr << "Failed to load font!\n";
        return -1;
    }

    std::vector<Card> cards;
    std::vector<sf::Color> colors;

    initializeCards(cards, colors);
    bool firstCardFlipped = false, secondCardFlipped = false;
    int firstCardIndex = -1, secondCardIndex = -1;
    bool gameStarted = false;

    displayMessage(window, "Click to Start!", font);
    while (window.isOpen() && !gameStarted) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) window.close();
            if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button == sf::Mouse::Left) {
                gameStarted = true;
            }
        }
    }
```

```cpp
std::string currentComment = "Get ready!";  // Initial comment
while (window.isOpen()) {
    int viewTime = std::max(3, 6 - hardshipLevel);

    displayCardSequence(window, cards, viewTime);

    sf::Clock levelClock;

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) window.close();

            if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button == sf::Mouse::Left) {
                sf::Vector2i mousePos = sf::Mouse::getPosition(window);
                int x = mousePos.x / CARD_WIDTH;
                int y = mousePos.y / CARD_HEIGHT;

                if (x < GRID_SIZE && y < GRID_SIZE) {
                    int cardIndex = y * GRID_SIZE + x;

                    if (!cards[cardIndex].flipped && gameStarted) {
                        cards[cardIndex].flip();
                        if (!firstCardFlipped) {
                            firstCardFlipped = true;
                            firstCardIndex = cardIndex;
                        } else if (!secondCardFlipped) {
                            secondCardFlipped = true;
                            secondCardIndex = cardIndex;
                        }
                    }
                }
            }
        }

        if (firstCardFlipped && secondCardFlipped) {
            if (cards[firstCardIndex].originalColor == cards[secondCardIndex].originalColor) {
                score += 10 * level;
                currentComment = getRandomComment(score, mistakes);  // Update comment after a correct match
                firstCardFlipped = secondCardFlipped = false;

                bool allFlipped = true;
                for (auto &card : cards) {
                    if (!card.flipped) {
                        allFlipped = false;
                        break;
                    }
                }

                if (allFlipped) {
                    displayLevelComplete(window, font);
                    level++;
                    if (level % 3 == 0) hardshipLevel++;

                    initializeCards(cards, colors);
                    levelClock.restart();
                    break;
                }
            } else {
                sf::sleep(sf::seconds(0.5));
                cards[firstCardIndex].flip();
                cards[secondCardIndex].flip();
                score -= 2 * level;
                mistakes++;
                currentComment = getRandomComment(score, mistakes);  // Update comment after a mistake
                firstCardFlipped = secondCardFlipped = false;
```

```cpp
                if (score <= -20 || mistakes >= 10) {
                    displayGameOver(window, font);
                    return 0;
                }
            }
        }

        window.clear(sf::Color::Black);
        for (auto &card : cards) card.draw(window);

        sf::Text scoreText("Score: " + std::to_string(score), font, 20);
        scoreText.setPosition(WINDOW_WIDTH - 180, 20);
        window.draw(scoreText);

        sf::Text levelText("Level: " + std::to_string(level), font, 20);
        levelText.setPosition(WINDOW_WIDTH - 180, 50);
        window.draw(levelText);

        sf::Text hardshipText("Hardship Level: " + std::to_string(hardshipLevel), font, 20);
        hardshipText.setPosition(WINDOW_WIDTH - 180, 80);
        window.draw(hardshipText);

        sf::Text mistakeText("Mistakes: " + std::to_string(mistakes), font, 20);
        mistakeText.setPosition(WINDOW_WIDTH - 180, 110);
        window.draw(mistakeText);

        int remainingTime = 105 - levelClock.getElapsedTime().asSeconds();
        std::stringstream timeStream;
        timeStream << "Time: " << remainingTime << "s";
        sf::Text timerText(timeStream.str(), font, 20);
        timerText.setPosition(WINDOW_WIDTH - 180, 140);
        window.draw(timerText);

        // Add a two-line gap before the "Cheers:" heading
        sf::Text cheersHeading("Cheers:", font, 20);
        cheersHeading.setPosition(WINDOW_WIDTH - 180, 170 + 40); // Adjusted gap (two-line gap)
        cheersHeading.setFillColor(sf::Color::White); // white for heading
        window.draw(cheersHeading);

        // Display the comment (below "Cheers:" heading)
        sf::Text funnyComment(currentComment, font, 18);
        funnyComment.setFillColor(sf::Color::Cyan);
        funnyComment.setPosition(WINDOW_WIDTH - 180, 200 + 40); // Adjusted positioning
        window.draw(funnyComment);

        window.display();

        if (remainingTime <= 0) {
            displayMessage(window, "Time's Up! Game Over!", font);
            sf::sleep(sf::seconds(2));
            return 0;
        }

        if (level > 10) {
            displayMessage(window, "YOUR MEMORY IS A MIRACLE", font);
            sf::sleep(sf::seconds(2));
            return 0;
        }
    }
}

return 0;
}
```
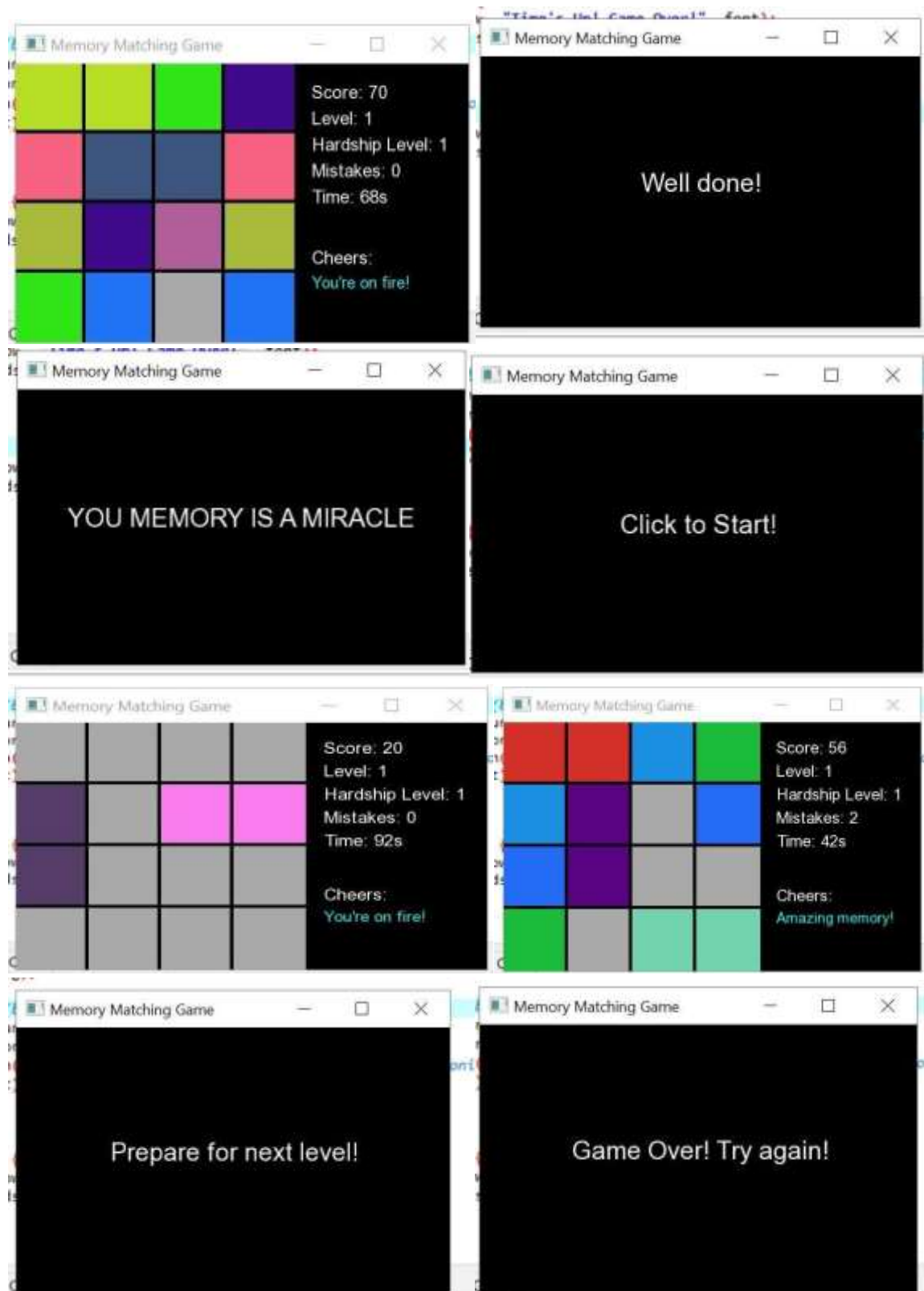
Output Screenshots (Taken randomly during gameplay)

Testing and Results

1. Functionality Testing:

- o The game logic was tested to ensure that the cards flip correctly, matches are detected, and the score updates appropriately.

- o The game was also tested for the display of messages such as "Game Over," "Level Complete," and other feedback messages.

2. Performance Testing:

- o The game was tested to ensure smooth performance without any lag or crashes. It performed well even with a higher number of cards or increased difficulty.

3. Usability Testing:

- o The user interface was tested for clarity, and the instructions were easy to follow. Players found the game intuitive and engaging.

Conclusion

The memory matching game was successfully developed using SFML and C++. The game meets the requirements by offering engaging gameplay, a clear user interface, and challenging levels. The random card pairing, coupled with dynamic difficulty levels, ensures that the game remains interesting as players progress. The inclusion of feedback messages, such as random comments based on performance, adds a fun and motivational element to the game. The use of ObjectOriented Programming (OOP) principles like Encapsulation and Abstraction has helped organize the game code and make it easier to maintain and extend.