**UIT University**

Spring 2024

23FA-CS

CSC103(P) “*Object Oriented Programming*”

Ms. Shiza Hassan

Project Report:

Turn-Based Card Game

Table of Contents

[Group Members 3](#_Toc168725733)

[1. Introduction 3](#_Toc168725734)

[2. Objectives 3](#_Toc168725735)

[3. Implementation 3](#_Toc168725736)

[3.1 Code 3](#_Toc168725737)

[3.1.1 Health Bar Class 3](#_Toc168725738)

[3.1.2 Main Menu Class 5](#_Toc168725739)

[3.1.3 Game Class 8](#_Toc168725740)

[3.2. User Interface (UI) 21](#_Toc168725741)

[3.3. Game Mechanics 21](#_Toc168725742)

[3.3.1. Health Management 21](#_Toc168725743)

[3.3.2. Actions 21](#_Toc168725744)

[3.3.3. Element Interaction 21](#_Toc168725745)

[3.4. Multimedia Integration 22](#_Toc168725746)

[3.5. Game Flow 22](#_Toc168725747)

[4. Testing 22](#_Toc168725748)

[4.1. Functionality Testing 22](#_Toc168725749)

[4.2. User Experience Testing 22](#_Toc168725750)

[5. Challenges 22](#_Toc168725751)

[6. Conclusion 23](#_Toc168725752)

[7. Future Work 23](#_Toc168725753)

# Group Members

* Taha Nadeem 23FA-003-CS
* Muhammad Musaddiq 23FA-002-CS
* Maaz Saleem 23FA-013-CS
* Sahir Ul Hassan 23FA-021-CS

# 1. Introduction

This project involves the development of a turn-based card game using Java and the Swing GUI toolkit. The game features a player-versus-CPU format where both parties take turns to perform actions such as attacking, healing, or yielding. The game integrates multimedia elements such as theme music and uses images to represent different card elements.

# 2. Objectives

The main objectives of the project are:

1. To develop a GUI-based interactive card game.
2. To implement turn-based mechanics with various player actions.
3. To use multimedia components for an enhanced user experience.
4. To manage game states, health bars, and rounds dynamically.

# 3. Implementation

## 3.1 Code

### 3.1.1 Health Bar Class

**package main;**

**import java.awt.Color;**

**import java.awt.Dimension;**

**import java.awt.Graphics;**

**import javax.swing.JComponent;**

**/\*\***

**\***

**\* @author sahir**

**\*/**

**public class HealthBar extends JComponent {**

**private int health = 100;**

**//Setter method for health since it's a private field.**

**public void setHealth(int health) {**

**this.health = health;**

**repaint(); // A method from the Component class in the AWT Graphics library. Makes a call to rerender the healthbar, every time the health changes.**

**}**

**@Override**

**protected void paintComponent(Graphics g) { //overriding the Superclass's method to provide custom code for drawing.**

**super.paintComponent(g); //calling super's method to ensure all necessary paintwork done by the parent class is preserved.**

**g.setColor(Color.RED);**

**g.fillRect(0, 0, health \* 2, 30); // Bar level - multipied by two for larger scaling.**

**g.setColor(Color.BLACK);**

**g.drawRect(1, 1, 198, 28); // Bar border**

**}**

**@Override**

**public Dimension getPreferredSize() { // Dimension is a class in AWT package that determines the size to be allocated for each component. We have overridden it.**

**return new Dimension(200, 30); // Fixed size for the health bar**

**}**

**}**

### 3.1.2 Main Menu Class

**package main;**

**import javax.swing.\*;**

**import java.awt.\*;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**public class MainMenu extends JFrame {**

**public MainMenu() {**

**// Set the title of the window**

**setTitle("Game Window");**

**// Set the default close operation**

**setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**// Set the layout manager**

**setLayout(new GridBagLayout()); // Use GridBagLayout to center the buttons**

**// Set the background color**

**getContentPane().setBackground(Color.lightGray);**

**// Create the heading label**

**JLabel heading = new JLabel("Triple Element Showdown!");**

**heading.setFont(new Font("Arial", Font.BOLD, 27));**

**heading.setForeground(Color.GREEN);**

**// Create the "Play" button**

**JButton playButton = new JButton("Play");**

**playButton.setBackground(Color.GREEN);**

**playButton.setForeground(Color.BLACK);**

**playButton.setFont(new Font("Arial", Font.BOLD, 24));**

**playButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**// Execute the game code**

**new Game();**

**}**

**});**

**// Create the "Quit" button**

**JButton quitButton = new JButton("Quit");**

**quitButton.setBackground(Color.BLUE);**

**quitButton.setForeground(Color.BLACK);**

**quitButton.setFont(new Font("Arial", Font.BOLD, 24));**

**quitButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**// Exit the application**

**System.exit(0);**

**}**

**});**

**// Create panel to hold the buttons**

**JPanel buttonPanel = new JPanel();**

**buttonPanel.setBackground(Color.LIGHT\_GRAY); // Match the panel background with the frame**

**buttonPanel.add(playButton);**

**buttonPanel.add(quitButton);**

**// Create panel to hold the heading**

**JPanel headingPanel = new JPanel();**

**headingPanel.setBackground(Color.LIGHT\_GRAY); // Match the panel background with the frame**

**headingPanel.add(heading);**

**// Set bounds for the heading panel**

**GridBagConstraints gbcHeading = new GridBagConstraints();**

**gbcHeading.gridx = 0;**

**gbcHeading.gridy = 100;**

**gbcHeading.insets = new Insets(20, 0, 20, 0); //heading position**

**gbcHeading.anchor = GridBagConstraints.NORTH;**

**add(headingPanel, gbcHeading);**

**// Set bounds for the button panel**

**GridBagConstraints gbcButtons = new GridBagConstraints();**

**gbcButtons.gridx = 0;**

**gbcButtons.gridy = 1;**

**gbcButtons.insets = new Insets(20, 0, 20, 0); // button placement and all**

**gbcButtons.anchor = GridBagConstraints.CENTER;**

**add(buttonPanel, gbcButtons);**

**setExtendedState(JFrame.MAXIMIZED\_BOTH);**

**setUndecorated(true); // Remove title bar for full screen effect**

**setVisible(true);**

**}**

**public static void main(String[] args) {**

**MainMenu gameWindow = new MainMenu();**

**}**

**}**

### 3.1.3 Game Class

**package main;**

**import javax.swing.\*;**

**import java.awt.\*;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.util.Random;**

**import javax.sound.sampled.\*;**

**import java.io.File;**

**public class Game extends JFrame implements ActionListener {**

**private JButton attack\_button, heal\_button, yield\_button;**

**private JButton fire\_button, water\_button, snow\_button;**

**private JLabel label\_hp\_player, label\_hp\_cpu, label\_message, label\_round, label\_final\_message;**

**private HealthBar playerHealthBar, cpuHealthBar;**

**private JLabel playerCard, cpuCard;**

**private Clip themeClip;**

**private int playerHealth = 100;**

**private int cpuHealth = 100;**

**private int round = 1;**

**private int player\_win\_round = 0;**

**private int cpu\_win\_round = 0;**

**private String playerElement, cpuElement;**

**Game() {**

**playThemeSong();**

**initUI();**

**}**

**private void playThemeSong() {**

**try {**

**File themeFile = new File("ThemeSong.wav");**

**AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(themeFile);**

**themeClip = AudioSystem.getClip();**

**themeClip.open(audioInputStream);**

**themeClip.loop(Clip.LOOP\_CONTINUOUSLY);**

**} catch (java.io.IOException | UnsupportedAudioFileException | LineUnavailableException e) {**

**e.printStackTrace();**

**}**

**}**

**private void initUI() {**

**// Object creation - Buttons and labels**

**attack\_button = new JButton("Attack");**

**heal\_button = new JButton("Heal");**

**yield\_button = new JButton("Yield");**

**fire\_button = new JButton("Fire");**

**water\_button = new JButton("Water");**

**snow\_button = new JButton("Snow");**

**label\_hp\_player = new JLabel("Your health: " + playerHealth + "%");**

**label\_hp\_cpu = new JLabel("CPU health: " + cpuHealth + "%");**

**label\_message = new JLabel("");**

**label\_message.setHorizontalAlignment(SwingConstants.CENTER);**

**label\_round = new JLabel();**

**label\_round.setHorizontalAlignment(SwingConstants.CENTER);**

**label\_round.setText("Round: " + round);**

**label\_final\_message = new JLabel();**

**label\_final\_message.setHorizontalAlignment(SwingConstants.CENTER);**

**label\_final\_message.setFont(new Font("Serif", Font.PLAIN, 32));**

**// Health bars**

**playerHealthBar = new HealthBar();**

**cpuHealthBar = new HealthBar();**

**// Card labels**

**playerCard = new JLabel(new ImageIcon());**

**cpuCard = new JLabel(new ImageIcon());**

**// Set layout and bounds**

**setLayout(new BorderLayout());**

**// Game control panel**

**JPanel controlPanel = new JPanel();**

**controlPanel.setLayout(new GridLayout(1, 3));**

**controlPanel.add(attack\_button);**

**controlPanel.add(heal\_button);**

**controlPanel.add(yield\_button);**

**// Element selection panel**

**JPanel elementPanel = new JPanel();**

**elementPanel.setLayout(new GridLayout(1, 3));**

**elementPanel.add(fire\_button);**

**elementPanel.add(water\_button);**

**elementPanel.add(snow\_button);**

**// Health and message panel**

**JPanel healthPanel = new JPanel();**

**healthPanel.setLayout(new GridLayout(2, 1));**

**JPanel healthLabelsPanel = new JPanel(new GridLayout(2, 1));**

**healthLabelsPanel.add(label\_hp\_player);**

**healthLabelsPanel.add(label\_hp\_cpu);**

**healthPanel.add(healthLabelsPanel);**

**JPanel healthBarsPanel = new JPanel(new GridLayout(2, 1));**

**healthBarsPanel.add(playerHealthBar);**

**healthBarsPanel.add(cpuHealthBar);**

**healthPanel.add(healthBarsPanel);**

**// Round and final message panel**

**JPanel messagePanel = new JPanel(new GridLayout(3, 1));**

**messagePanel.add(label\_round);**

**messagePanel.add(label\_message);**

**messagePanel.add(label\_final\_message);**

**// Adding panels to the frame**

**JPanel topPanel = new JPanel(new BorderLayout());**

**topPanel.add(controlPanel, BorderLayout.NORTH);**

**topPanel.add(elementPanel, BorderLayout.SOUTH);**

**add(topPanel, BorderLayout.NORTH);**

**// Center panel**

**JPanel centerPanel = new JPanel(new GridLayout(1, 2));**

**// Player panel**

**JPanel playerPanel = new JPanel(new BorderLayout());**

**playerPanel.add(playerCard, BorderLayout.CENTER);**

**JPanel playerHealthPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));**

**playerHealthPanel.add(label\_hp\_player);**

**playerHealthPanel.add(playerHealthBar);**

**playerPanel.add(playerHealthPanel, BorderLayout.SOUTH);**

**// CPU panel**

**JPanel cpuPanel = new JPanel(new BorderLayout());**

**cpuPanel.add(cpuCard, BorderLayout.CENTER);**

**JPanel cpuHealthPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));**

**cpuHealthPanel.add(label\_hp\_cpu);**

**cpuHealthPanel.add(cpuHealthBar);**

**cpuPanel.add(cpuHealthPanel, BorderLayout.SOUTH);**

**centerPanel.add(playerPanel);**

**centerPanel.add(cpuPanel);**

**add(centerPanel, BorderLayout.CENTER);**

**add(messagePanel, BorderLayout.SOUTH);**

**setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**setResizable(false);**

**setTitle("Turn-Based Card Game");**

**setSize(1920, 1080); // Set Window/Frame size**

**setVisible(true);**

**// Adding action listeners**

**attack\_button.addActionListener(this);**

**heal\_button.addActionListener(this);**

**yield\_button.addActionListener(this);**

**fire\_button.addActionListener(this);**

**water\_button.addActionListener(this);**

**snow\_button.addActionListener(this);**

**// Initialize health bars**

**updateHealthBars();**

**}**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**JButton button = (JButton) e.getSource();**

**Random random = new Random();**

**String[] elements = {"fire", "water", "snow"};**

**if (button == attack\_button || button == heal\_button || button == yield\_button) {**

**if (playerElement == null) {**

**JOptionPane.showMessageDialog(this, "Please select an element first!", "Warning", JOptionPane.WARNING\_MESSAGE);**

**return;**

**}**

**cpuElement = elements[random.nextInt(elements.length)];**

**handleGameActions(button);**

**updateCpuCardImage();**

**updateLabels();**

**updateHealthBars();**

**} else {**

**handleElementSelection(button);**

**}**

**}**

**private void handleGameActions(JButton button) {**

**if (button == attack\_button) {**

**handleAttackAction();**

**} else if (button == heal\_button) {**

**handleHealAction();**

**} else if (button == yield\_button) {**

**handleYieldAction();**

**}**

**}**

**private void updateCpuCardImage() {**

**switch (cpuElement) {**

**case "fire":**

**cpuCard.setIcon(new ImageIcon("Fire\_pokemon\_1.jpg"));**

**break;**

**case "water":**

**cpuCard.setIcon(new ImageIcon("Water\_pokemon\_1.jpg"));**

**break;**

**case "snow":**

**cpuCard.setIcon(new ImageIcon("Snow\_pokemon\_1.jpg"));**

**break;**

**}**

**}**

**private void handleElementSelection(JButton button) {**

**if (button == fire\_button) {**

**playerElement = "fire";**

**playerCard.setIcon(new ImageIcon("Fire\_pokemon\_1.jpg"));**

**} else if (button == water\_button) {**

**playerElement = "water";**

**playerCard.setIcon(new ImageIcon("Water\_pokemon\_1.jpg"));**

**} else if (button == snow\_button) {**

**playerElement = "snow";**

**playerCard.setIcon(new ImageIcon("Snow\_pokemon\_1.jpg"));**

**}**

**label\_message.setText("You selected: " + playerElement);**

**}**

**private void handleAttackAction() {**

**String[] action = {"Fight", "Heal"};**

**Random random = new Random();**

**String cpu\_action = action[random.nextInt(action.length)];**

**if ("Fight".equals(cpu\_action)) {**

**performFightAction();**

**} else if ("Heal".equals(cpu\_action)) {**

**performCpuHealAction();**

**}**

**checkEndOfRound();**

**}**

**private void handleHealAction() {**

**String[] action = {"Fight", "Heal"};**

**Random random = new Random();**

**String cpu\_action = action[random.nextInt(action.length)];**

**if ("Fight".equals(cpu\_action)) {**

**performCpuFightAction();**

**} else if ("Heal".equals(cpu\_action)) {**

**performHealAction();**

**}**

**checkEndOfRound();**

**}**

**private void handleYieldAction() {**

**label\_message.setText("You yielded! CPU wins the game!");**

**playerHealth = 0;**

**updateLabels();**

**updateHealthBars();**

**checkEndOfRound();**

**checkWinner();**

**}**

**private void performFightAction() {**

**if (playerElement.equals(cpuElement)) {**

**label\_message.setText("Same elements...partial damage was done to both pokemons...");**

**playerHealth -= 7;**

**cpuHealth -= 7;**

**} else if (isPlayerElementStrongAgainst()) {**

**label\_message.setText("Your attack was very effective! Foe's attack was weak...");**

**playerHealth -= 7;**

**cpuHealth -= 20;**

**} else {**

**label\_message.setText("Your Foe's attack was very effective! Your attack was weak...");**

**playerHealth -= 20;**

**cpuHealth -= 7;**

**}**

**}**

**private void performCpuFightAction() {**

**if (playerElement.equals(cpuElement)) {**

**label\_message.setText("Same elements...both pokemons partially healed");**

**playerHealth = Math.min(100, playerHealth + 3);**

**cpuHealth = Math.min(100, cpuHealth + 3);**

**} else if (isPlayerElementStrongAgainst()) {**

**label\_message.setText("Healing successful! Pokemon healed!");**

**playerHealth = Math.min(100, playerHealth + 20);**

**} else {**

**label\_message.setText("Healing failed. Health is partially lost...");**

**playerHealth -= 7;**

**}**

**}**

**private void performCpuHealAction() {**

**if (playerElement.equals(cpuElement)) {**

**label\_message.setText("Same elements...both pokemons partially healed");**

**playerHealth = Math.min(100, playerHealth + 3);**

**cpuHealth = Math.min(100, cpuHealth + 3);**

**} else if (isPlayerElementStrongAgainst()) {**

**label\_message.setText("CPU healed partially. Health is restored.");**

**cpuHealth = Math.min(100, cpuHealth + 20);**

**} else {**

**label\_message.setText("CPU healing failed. Health is partially lost.");**

**cpuHealth -= 7;**

**}**

**}**

**private void performHealAction() {**

**if (playerElement.equals(cpuElement)) {**

**label\_message.setText("Same elements...both pokemons partially healed");**

**playerHealth = Math.min(100, playerHealth + 3);**

**cpuHealth = Math.min(100, cpuHealth + 3);**

**} else if (isPlayerElementStrongAgainst()) {**

**label\_message.setText("Player healing successful! Pokemon healed!");**

**playerHealth = Math.min(100, playerHealth + 20);**

**} else {**

**label\_message.setText("Player healing failed. Health is partially lost...");**

**playerHealth -= 7;**

**}**

**}**

**private void checkEndOfRound() {**

**if (playerHealth <= 0 || cpuHealth <= 0) {**

**round++;**

**if (playerHealth > cpuHealth) {**

**player\_win\_round++;**

**label\_message.setText("You won this round!");**

**} else {**

**cpu\_win\_round++;**

**label\_message.setText("CPU won this round!");**

**}**

**if (round <= 4) {**

**playerHealth = 100;**

**cpuHealth = 100;**

**label\_round.setText("Round: " + round);**

**playerElement = null;**

**}**

**updateLabels();**

**updateHealthBars();**

**checkWinner();**

**}**

**}**

**private void checkWinner() {**

**if (round > 4) {**

**disableActionButtons();**

**if (player\_win\_round > cpu\_win\_round) {**

**label\_final\_message.setText("You won the game!");**

**} else if (cpu\_win\_round > player\_win\_round) {**

**label\_final\_message.setText("CPU won the game!");**

**} else {**

**label\_final\_message.setText("The game ended in a draw!");**

**}**

**}**

**}**

**private void disableActionButtons() {**

**attack\_button.setEnabled(false);**

**heal\_button.setEnabled(false);**

**yield\_button.setEnabled(false);**

**}**

**private boolean isPlayerElementStrongAgainst() {**

**return (playerElement.equals("water") && cpuElement.equals("fire"))**

**|| (playerElement.equals("snow") && cpuElement.equals("water"))**

**|| (playerElement.equals("fire") && cpuElement.equals("snow"));**

**}**

**private void updateLabels() {**

**label\_hp\_player.setText("Your health: " + playerHealth + "%");**

**label\_hp\_cpu.setText("CPU health: " + cpuHealth + "%");**

**}**

**private void updateHealthBars() {**

**playerHealthBar.setHealth(playerHealth);**

**cpuHealthBar.setHealth(cpuHealth);**

**}**

**public static void main(String[] args) {**

**Game game = new Game();**

**}**

**}**

## 3.2. User Interface (UI)

The UI is implemented using Java Swing and is structured into various panels and components:

* **Control Panel**: Contains action buttons for Attack, Heal, and Yield.
* **Element Selection Panel**: Contains buttons for selecting elements (Fire, Water, Snow).
* **Health and Message Panel**: Displays health information for both player and CPU, and messages regarding game actions and outcomes.
* **Round and Final Message Panel**: Shows the current round and final game outcome.
* **Player and CPU Panels**: Display selected element images and health bars.

## 3.3. Game Mechanics

### 3.3.1. Health Management

* **Player and CPU Health**: Both start with 100% health. Health updates dynamically based on actions performed.
* **Health Bars**: Graphical representation of health percentages using custom HealthBar components.

### 3.3.2. Actions

* **Attack**: Compares player and CPU elements to determine damage dealt. Includes logic for effectiveness based on element types.
* **Heal**: Allows partial or full health recovery based on element match-ups.
* **Yield**: Ends the game prematurely, declaring the CPU as the winner.

### 3.3.3. Element Interaction

* **Elements**: Fire, Water, Snow. Each element has strengths and weaknesses:
  + Fire > Snow
  + Water > Fire
  + Snow > Water

## 3.4. Multimedia Integration

* **Theme Song**: A continuous loop of theme music (ThemeSong.wav) is played using Java Sound API.
* **Images**: Element-specific images are displayed for player and CPU selections.

## 3.5. Game Flow

1. **Element Selection**: Players must select an element before performing any action.
2. **Action Execution**: Based on selected actions, health is adjusted and messages are displayed.
3. **Round Management**: The game progresses through rounds, resetting health after each round until a maximum of four rounds.
4. **Endgame**: Determines the winner based on round victories and displays a final message.

# 4. Testing

## 4.1. Functionality Testing

* **Buttons**: Ensured all buttons trigger the correct actions.
* **Health Bars**: Verified accurate updates to health bars after each action.
* **Game Logic**: Tested various scenarios to ensure element interactions and health calculations are correct.
* **Multimedia**: Confirmed proper loading and looping of the theme song and display of element images.

## 4.2. User Experience Testing

* **Usability**: Ensured the interface is intuitive and responsive.
* **Messages and Feedback**: Verified that the game provides clear feedback through labels and messages.

# 5. Challenges

* **Audio Integration**: Handling exceptions related to audio playback and ensuring seamless looping.
* **Dynamic Updates**: Ensuring health bars and labels update correctly in real-time.
* **Element Logic**: Implementing and debugging the interaction logic between different elements.

# 6. Conclusion

The turn-based card game successfully meets the project objectives, providing an engaging user experience through a well-designed GUI, dynamic game mechanics, and multimedia integration. This project demonstrates the effective use of Java Swing for GUI applications and Java Sound API for audio management, and it lays a solid foundation for further enhancements, such as adding more elements, multiplayer support, and advanced animations.

# 7. Future Work

Potential enhancements include:

* **Additional Elements**: Introducing more elements for increased complexity.
* **Multiplayer Mode**: Allowing two players to compete.
* **Animations**: Adding animations for attacks and health changes.
* **AI Improvements**: Enhancing CPU decision-making algorithms for a more challenging gameplay experience.