package com.dataflair.Javagenerics;  
  
import java.awt.\*;  
import java.awt.event.\*;  
import javax.swing.\*;  
import javax.swing.Timer;  
import java.util.Random;  
  
public class MazeSolverGame extends JFrame {  
 public static final int *ROWS* = 10; // Number of rows in the maze  
 public static final int *COLS* = 10; // Number of columns in the maze  
 public static final int *CELL\_SIZE* = 60; // Size of each cell in the maze  
 public static final int *WALL* = 1; // Wall representation  
 public static final int *PATH* = 0; // Path representation  
 public static final int *PLAYER* = 2; // Player representation  
 public static final int *EXIT* = 3; // Exit representation  
  
 int[][] maze; // Maze array  
 private int playerRow, playerCol; // Player's current position  
 private int moveLimit = 50; // Maximum moves allowed  
 private int moves = 0; // Counter for moves made  
  
 private JPanel mazePanel; // Panel to draw the maze  
 private JLabel scoreLabel; // Label to display score and time  
 private Timer timer; // Timer for the game  
 private int timeLimit = 60; // Time limit for the game  
 private int timeRemaining; // Remaining time  
  
 private JPanel coverPanel; // Cover page panel  
  
 private Image startImage;  
 private Image endImage;  
  
 public MazeSolverGame() {  
 setTitle("Maze Solver Game");  
 setSize(*COLS* \* *CELL\_SIZE* + 50, *ROWS* \* *CELL\_SIZE* + 150);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLocationRelativeTo(null);  
 setLayout(new BorderLayout());  
  
 // Load the cartoon images  
 startImage = new ImageIcon("C:\\Users\\Dell\\Downloads\\tom\_start.png").getImage();  
 endImage = new ImageIcon("C:\\Users\\Dell\\Downloads\\jerry\_end.png").getImage();  
  
 createCoverPage();  
 }  
  
 private void createCoverPage() {  
 coverPanel = new JPanel() {  
 private <T> void drawMazeObject(Graphics g, T object, int row, int col) {  
 if (object instanceof ImageIcon) {  
 // If the object is an image (like start or end), draw the image  
 g.drawImage(((ImageIcon) object).getImage(), col \* *CELL\_SIZE* + 10, row \* *CELL\_SIZE* + 10,  
 *CELL\_SIZE* - 20, *CELL\_SIZE* - 20, null);  
 }  
 // You can add more conditions here for other types (like a text or color)  
 }  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 // Ensure the background image is drawn correctly  
 ImageIcon background = new ImageIcon("C:\\Users\\Dell\\Downloads\\maze\_bg.jpg"); // Use the correct path for your image  
 g.drawImage(background.getImage(), 0, 0, getWidth(), getHeight(), this);  
 }  
 };  
 coverPanel.setLayout(new BorderLayout());  
  
 // Title Label with enhanced styling  
 JLabel titleLabel = new JLabel("MAZE SOLVER", JLabel.*CENTER*);  
 titleLabel.setFont(new Font("Arial", Font.*BOLD*, 50)); // Large bold font for title  
 titleLabel.setForeground(Color.*WHITE*); // Title in white  
 titleLabel.setOpaque(false); // Make the label transparent  
 titleLabel.setBorder(BorderFactory.*createEmptyBorder*(20, 0, 20, 0)); // Add padding for title  
 coverPanel.add(titleLabel, BorderLayout.*NORTH*);  
  
 // Instructions with HTML styling  
 JLabel instructionLabel = new JLabel("<html><div style='text-align: center; color: white;'>"  
 + "Use <b>UP</b>, <b>DOWN</b>, <b>LEFT</b>, <b>RIGHT</b> keys to navigate the maze.<br>"  
 + "Find your way from <b>START</b> to <b>END</b>!</div></html>", JLabel.*CENTER*);  
 instructionLabel.setFont(new Font("Arial", Font.*PLAIN*, 20)); // Smaller font for instructions  
 coverPanel.add(instructionLabel, BorderLayout.*CENTER*);  
  
 // Start button with improved hover effect  
 JButton startButton = new JButton("Start Game");  
 startButton.setFont(new Font("Arial", Font.*BOLD*, 24)); // Larger text on the button  
 startButton.setForeground(Color.*WHITE*); // Text color for the button  
 startButton.setBackground(Color.*DARK\_GRAY*); // Button color  
 startButton.setFocusPainted(false); // Remove focus effect  
 startButton.setBorder(BorderFactory.*createEmptyBorder*(10, 20, 10, 20)); // Padding for the button  
 startButton.setCursor(new Cursor(Cursor.*HAND\_CURSOR*)); // Change cursor on hover  
  
 // Button hover effects  
 startButton.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseEntered(MouseEvent e) {  
 startButton.setBackground(Color.*LIGHT\_GRAY*); // Change background color when hovered  
 }  
  
 @Override  
 public void mouseExited(MouseEvent e) {  
 startButton.setBackground(Color.*DARK\_GRAY*); // Revert back to dark gray when not hovered  
 }  
 });  
  
 // Action on button click  
 startButton.addActionListener(e -> {  
 remove(coverPanel); // Remove cover page  
 initializeGame(); // Start game logic  
 revalidate(); // Revalidate layout  
 repaint(); // Repaint screen  
 });  
  
 // Add button to a JPanel to add padding and layout management  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setOpaque(false); // Make button panel background transparent  
 buttonPanel.add(startButton);  
 coverPanel.add(buttonPanel, BorderLayout.*SOUTH*);  
  
 add(coverPanel); // Add the coverPanel to the frame  
 }  
  
 // Shadow effect for the title  
 class ShadowEffect extends javax.swing.plaf.LabelUI {  
 @Override  
 public void paint(Graphics g, JComponent c) {  
 JLabel label = (JLabel) c;  
 Graphics2D g2 = (Graphics2D) g.create();  
 g2.setRenderingHint(RenderingHints.*KEY\_TEXT\_ANTIALIASING*, RenderingHints.*VALUE\_TEXT\_ANTIALIAS\_ON*);  
 g2.setColor(Color.*BLACK*);  
 g2.drawString(label.getText(), label.getInsets().left + 4, label.getHeight() - 4); // Shadow  
 g2.setColor(label.getForeground());  
 g2.drawString(label.getText(), label.getInsets().left, label.getHeight() - 8); // Main text  
 g2.dispose();  
 }  
 }  
  
 private void initializeGame() {  
 // Create the centeredPanel and override its paintComponent method  
 JPanel centeredPanel = new JPanel() {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 // Load the background image for the maze panel  
 ImageIcon background = new ImageIcon("C:\\Users\\Dell\\Downloads\\maze\_bg.jpg"); // Provide the correct path to your image  
 Image img = background.getImage();  
 g.drawImage(img, 0, 0, getWidth(), getHeight(), this); // Draw the image to fill the entire panel  
 }  
 };  
  
 // Set the layout and preferred size of the panel  
 centeredPanel.setLayout(new GridBagLayout());  
 centeredPanel.setPreferredSize(new Dimension(*COLS* \* *CELL\_SIZE*, *ROWS* \* *CELL\_SIZE*));  
  
 mazePanel = new JPanel() {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 for (int row = 0; row < *ROWS*; row++) {  
 for (int col = 0; col < *COLS*; col++) {  
 Color cellColor;  
  
 if (maze[row][col] == *WALL*) {  
 cellColor = new Color(50, 50, 50);  
 } else if (maze[row][col] == *PATH*) {  
 cellColor = new Color(240, 240, 240);  
 } else {  
 cellColor = Color.*WHITE*;  
 }  
  
 Graphics2D g2d = (Graphics2D) g;  
 g2d.setColor(cellColor);  
 g2d.fillRoundRect(col \* *CELL\_SIZE*, row \* *CELL\_SIZE*, *CELL\_SIZE*, *CELL\_SIZE*, 15, 15);  
 g.setColor(Color.*BLACK*);  
 g.drawRoundRect(col \* *CELL\_SIZE*, row \* *CELL\_SIZE*, *CELL\_SIZE*, *CELL\_SIZE*, 15, 15);  
  
 // Draw images  
 if (maze[row][col] == *PLAYER*) {  
 int imageSize = *CELL\_SIZE* - 20;  
 g.drawImage(startImage, col \* *CELL\_SIZE* + 10, row \* *CELL\_SIZE* + 10, imageSize, imageSize, null);  
 } else if (maze[row][col] == *EXIT*) {  
 int imageSize = *CELL\_SIZE* - 20;  
 g.drawImage(endImage, col \* *CELL\_SIZE* + 10, row \* *CELL\_SIZE* + 10, imageSize, imageSize, null);  
 }  
 }  
 }  
 }  
 };  
  
 mazePanel.setPreferredSize(new Dimension(*COLS* \* *CELL\_SIZE*, *ROWS* \* *CELL\_SIZE*));  
  
 // Add the mazePanel to the centeredPanel  
 centeredPanel.add(mazePanel);  
 add(centeredPanel, BorderLayout.*CENTER*); // Add the centeredPanel to the main frame  
  
 scoreLabel = new JLabel("Moves: 0 / " + moveLimit + " Time: " + timeLimit + "s");  
 scoreLabel.setFont(new Font("Arial", Font.*BOLD*, 18));  
 scoreLabel.setForeground(Color.*BLUE*);  
  
 JButton restartButton = new JButton("Restart");  
 restartButton.setFont(new Font("Arial", Font.*BOLD*, 16));  
 restartButton.addActionListener(e -> resetGame());  
  
 JPanel topPanel = new JPanel();  
 topPanel.add(scoreLabel);  
 topPanel.add(restartButton);  
 add(topPanel, BorderLayout.*NORTH*);  
  
 mazePanel.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyPressed(KeyEvent e) {  
 if (moves < moveLimit) {  
 switch (e.getKeyCode()) {  
 case KeyEvent.*VK\_UP* -> movePlayer(-1, 0);  
 case KeyEvent.*VK\_DOWN* -> movePlayer(1, 0);  
 case KeyEvent.*VK\_LEFT* -> movePlayer(0, -1);  
 case KeyEvent.*VK\_RIGHT* -> movePlayer(0, 1);  
 }  
 mazePanel.repaint();  
 }  
 }  
 });  
  
 mazePanel.setFocusable(true);  
 mazePanel.requestFocusInWindow();  
  
 generateMaze();  
 startTimer();  
 }  
  
  
 void generateMaze() {  
 maze = new int[*ROWS*][*COLS*];  
 Random random = new Random();  
  
 for (int row = 0; row < *ROWS*; row++) {  
 for (int col = 0; col < *COLS*; col++) {  
 maze[row][col] = (random.nextDouble() < 0.3) ? *WALL* : *PATH*;  
 }  
 }  
  
 maze[0][0] = *PLAYER*;  
 playerRow = 0;  
 playerCol = 0;  
 maze[*ROWS* - 1][*COLS* - 1] = *EXIT*;  
 }  
  
 private void movePlayer(int dRow, int dCol) {  
 int newRow = playerRow + dRow;  
 int newCol = playerCol + dCol;  
  
 // Check if the move is valid (inside bounds and not a wall)  
 if (newRow >= 0 && newRow < *ROWS* && newCol >= 0 && newCol < *COLS* && maze[newRow][newCol] != *WALL*) {  
 // Update the player's position  
 maze[playerRow][playerCol] = *PATH*; // Set the old position back to PATH  
 playerRow = newRow;  
 playerCol = newCol;  
 maze[playerRow][playerCol] = *PLAYER*; // Update to the new position  
 moves++;  
 scoreLabel.setText("Moves: " + moves + " / " + moveLimit + " Time: " + timeRemaining + "s");  
  
 // Check if the player has reached the exit  
 if (playerRow == *ROWS* - 1 && playerCol == *COLS* - 1) {  
 stopTimer();  
  
 // Change Jerry's image to a "scared" version  
 endImage = new ImageIcon("C:\\Users\\Dell\\Downloads\\jerry scared image.jpg").getImage();  
 mazePanel.repaint(); // Repaint the maze to reflect the updated image  
  
 int totalTime = timeLimit - timeRemaining;  
 int response = JOptionPane.*showConfirmDialog*(this,  
 "CONGRATULATIONS 🎉 TOM GOT HIS FOOD!\nReached the exit in " + moves + " moves and " + totalTime + " seconds!\n" +  
 "Do you want to restart?", "Game Over",  
 JOptionPane.*YES\_NO\_OPTION*);  
  
 if (response == JOptionPane.*YES\_OPTION*) {  
 resetGame();  
 } else {  
 // Exit the game  
 System.*exit*(0);  
 }  
 }  
  
 // Check if the player has exceeded the move limit  
 if (moves >= moveLimit) {  
 stopTimer();  
 JOptionPane.*showMessageDialog*(this, "TOM FAILED TO GET HIS FOOD 😟!\n You've exceeded the maximum number of moves.");  
 resetGame();  
 }  
 }  
 }  
  
  
 private void startTimer() {  
 timeRemaining = timeLimit;  
 timer = new Timer(1000, e -> {  
 timeRemaining--;  
 scoreLabel.setText("Moves: " + moves + " / " + moveLimit + " Time: " + timeRemaining + "s");  
 if (timeRemaining <= 0) {  
 stopTimer();  
 JOptionPane.*showMessageDialog*(this, "Time's up!");  
 resetGame();  
 }  
 });  
 timer.start();  
 }  
  
 private void stopTimer() {  
 if (timer != null) {  
 timer.stop();  
 }  
 }  
  
 private void resetGame() {  
 stopTimer(); // Stop the current timer  
  
 // Reset the moves counter and update the score label  
 moves = 0;  
 scoreLabel.setText("Moves: " + moves + " / " + moveLimit + " Time: " + timeLimit + "s");  
  
 // Reset the maze  
 generateMaze();  
  
 // Restart the timer  
 startTimer();  
  
 // Repaint the maze  
 mazePanel.repaint();  
  
 // Request focus to ensure the player can start moving immediately  
 mazePanel.requestFocusInWindow();  
 }  
  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> {  
 MazeSolverGame game = new MazeSolverGame();  
 game.setVisible(true);  
 });  
 }  
}