# PAINNASSAU GAME STUDIO

Integrantes:

-Camile Marcele Araújo-01592555,

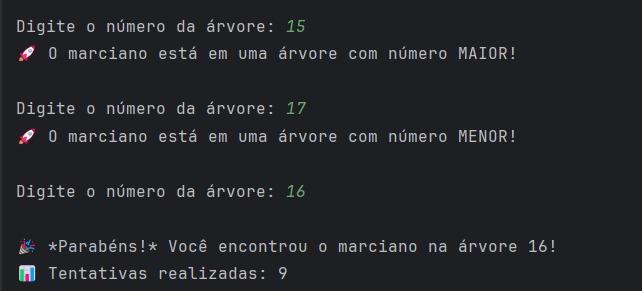
-Márcio Maycom da Silva oliveira-01607574,

-Pedro Miranda Marinho- 01607408,

-Rafaella Guedes Barbosa - 01588710

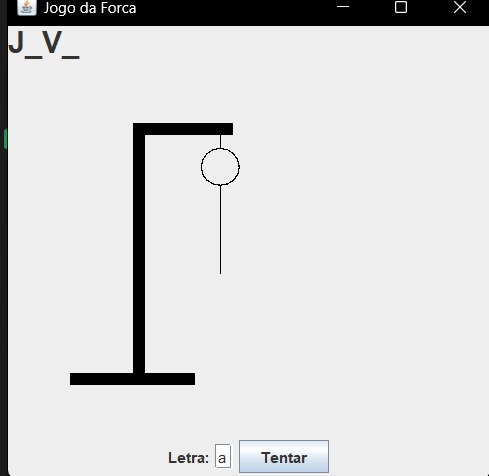
-Ricardo César Ottoni-01596659

Jogo do Marciano

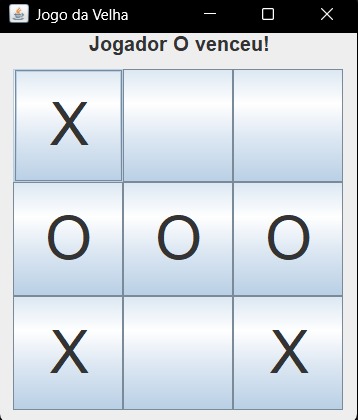
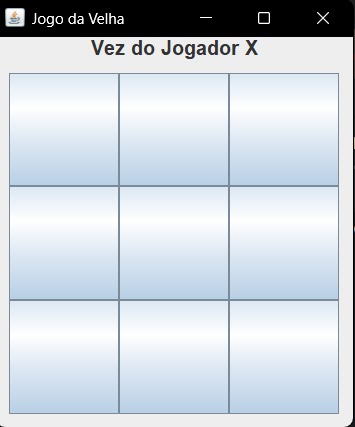


Jogo da Forca

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Jogo da Velha



Jogo Pong

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Uma imagem contendo Forma

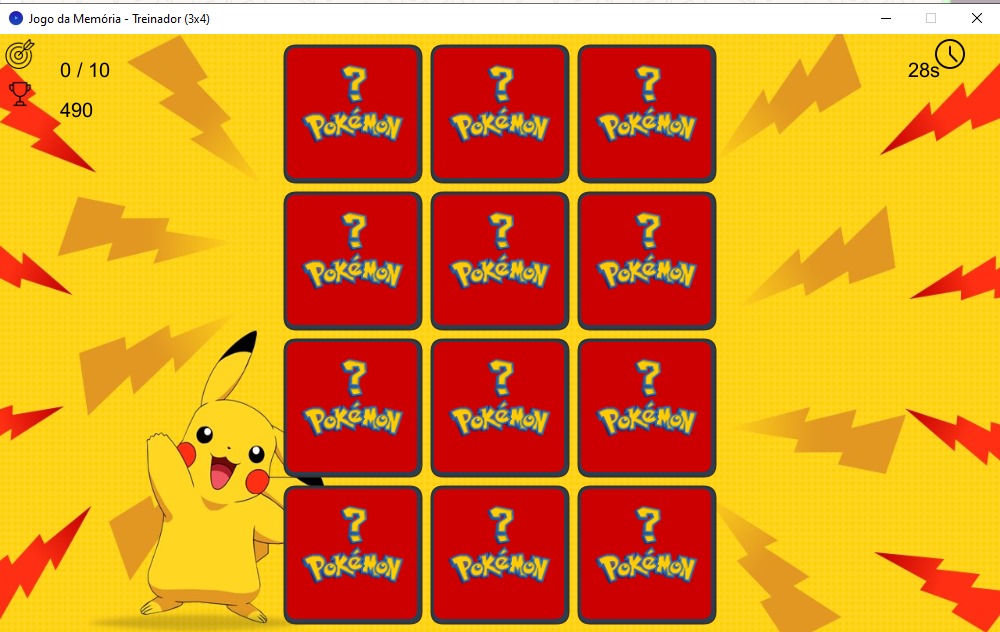
O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

Jogo da Memória







**- Código fonte -**

**JogoMarciano\Marciano.java**

|  |  |
| --- | --- |
| 1 | **package** JogoMarciano; |
| 2 |  |
| 3 | **import** java.util.Random; |
| 4 | **import** java.util.Scanner; |
| 5 |  |
| 6 | **public** **class** **Marciano** { |
| 7 | **public** **static** **void** **main**(String[] args) { |
| 8 | // Configuração inicial |
| 9 | Random random = **new** **Random**(); |
| 10 | Scanner scanner = **new** **Scanner**(System.in); |
| 11 |  |
| 12 | int numeroMarciano = random.nextInt(100) + 1; // Número aleatório entre 1 e 100 |
| 13 | int tentativas = 0; |
| 14 | int palpite; |
| 15 |  |
| 16 | System.out.println("🌌 \*Jogo do JogoMarciano.Marciano\* 🌌"); |
| 17 | System.out.println("Um marciano está escondido em uma árvore numerada de 1 a 100. Encontre-o!"); |
| 18 |  |
| 19 | // Loop principal do jogo |
| 20 | **do** { |
| 21 | System.out.print("\nDigite o número da árvore: "); |
| 22 | palpite = scanner.nextInt(); |
| 23 | tentativas++; |
| 24 |  |
| 25 | **if** (palpite < numeroMarciano) { |
| 26 | System.out.println("🚀 O marciano está em uma árvore com número MAIOR!"); |
| 27 | } **else** **if** (palpite > numeroMarciano) { |
| 28 | System.out.println("🚀 O marciano está em uma árvore com número MENOR!"); |
| 29 | } |
| 30 | } **while** (palpite != numeroMarciano); |
| 31 |  |
| 32 | // Mensagem de vitória |
| 33 | System.out.println("\n🎉 \*Parabéns!\* Você encontrou o marciano na árvore " + numeroMarciano + "!"); |
| 34 | System.out.println("📊 Tentativas realizadas: " + tentativas); |
| 35 |  |
| 36 | scanner.close(); |
| 37 | } |
| 38 | } |

**- Código fonte -**

**JogoForca\ForcaGUI.java**

|  |  |
| --- | --- |
| 1 | **package** JogoForca; |
| 2 |  |
| 3 | **import** javax.swing.\*; |
| 4 | **import** java.awt.\*; |
| 5 | **import** java.util.Arrays; |
| 6 |  |
| 7 | **public** **class** **ForcaGUI** **extends** **JFrame** { |
| 8 | **private** String palavraSecreta = "JAVA"; |
| 9 | **private** char[] letrasDescobertas; |
| 10 | **private** int tentativasRestantes = 6; |
| 11 | **private** StringBuilder letrasErradas = **new** **StringBuilder**(); |
| 12 |  |
| 13 | **private** JPanel enforcadoPanel; |
| 14 | **private** JLabel palavraLabel; |
| 15 | **private** JTextField letraField; |
| 16 | **private** JButton tentarButton; |
| 17 |  |
| 18 | **public** **ForcaGUI**() { |
| 19 | setTitle("Jogo da Forca"); |
| 20 | setSize(400, 400); |
| 21 | setDefaultCloseOpera­tion(JFrame.EXIT\_ON\_CLOSE); |
| 22 | setLayout(**new** **BorderLayout**()); |
| 23 |  |
| 24 | letrasDescobertas = **new** **char**[palavraSecreta.length()]; |
| 25 | Arrays.fill(letrasDescobertas, '\_'); |
| 26 |  |
| 27 | // Painel do desenho do enforcado |
| 28 | enforcadoPanel = **new** **JPanel**() { |
| 29 | @Override |
| 30 | **protected** **void** **paintComponent**(Graphics g) { |
| 31 | super.paintComponent(g); |
| 32 | desenharForca(g); |
| 33 | } |
| 34 | }; |
| 35 | enforcadoPanel.setPreferredSize(**new** **Dimension**(200, 200)); |
| 36 |  |
| 37 | // Painel de controle |
| 38 | JPanel controlePanel = **new** **JPanel**(); |
| 39 | palavraLabel = **new** **JLabel**(String.valueOf(letrasDescobertas)); |
| 40 | palavraLabel.setFont(**new** **Font**("Arial", Font.BOLD, 24)); |
| 41 |  |
| 42 | letraField = **new** **JTextField**(1); |
| 43 | tentarButton = **new** **JButton**("Tentar"); |
| 44 |  |
| 45 | tentarButton.addActionListener(e -> processarTentativa()); |
| 46 |  |
| 47 | controlePanel.add(**new** **JLabel**("Letra:")); |
| 48 | controlePanel.add(letraField); |
| 49 | controlePanel.add(tentarButton); |
| 50 |  |
| 51 | add(enforcadoPanel, BorderLayout.CENTER); |
| 52 | add(palavraLabel, BorderLayout.NORTH); |
| 53 | add(controlePanel, BorderLayout.SOUTH); |
| 54 |  |
| 55 | setVisible(true); |
| 56 | } |
| 57 |  |
| 58 | **private** **void** **desenharForca**(Graphics g) { |
| 59 | g.setColor(Color.BLACK); |
| 60 | // Base |
| 61 | g.fillRect(50, 250, 100, 10); |
| 62 | // Poste vertical |
| 63 | g.fillRect(100, 50, 10, 200); |
| 64 | // Braço horizontal |
| 65 | g.fillRect(100, 50, 80, 10); |
| 66 | // Corda |
| 67 | g.drawLine(170, 50, 170, 70); |
| 68 |  |
| 69 | **if** (tentativasRestantes < 6) { // Cabeça |
| 70 | g.drawOval(155, 70, 30, 30); |
| 71 | } |
| 72 | **if** (tentativasRestantes < 5) { // Corpo |
| 73 | g.drawLine(170, 100, 170, 170); |
| 74 | } |
| 75 | **if** (tentativasRestantes < 4) { // Braço esquerdo |
| 76 | g.drawLine(170, 120, 140, 140); |
| 77 | } |
| 78 | **if** (tentativasRestantes < 3) { // Braço direito |
| 79 | g.drawLine(170, 120, 200, 140); |
| 80 | } |
| 81 | **if** (tentativasRestantes < 2) { // Perna esquerda |
| 82 | g.drawLine(170, 170, 140, 200); |
| 83 | } |
| 84 | **if** (tentativasRestantes < 1) { // Perna direita |
| 85 | g.drawLine(170, 170, 200, 200); |
| 86 | } |
| 87 | } |
| 88 |  |
| 89 | **private** **void** **processarTentativa**() { |
| 90 | String input = letraField.getText().toUpperCase(); |
| 91 | letraField.setText(""); |
| 92 |  |
| 93 | **if** (input.length() != 1 || !Character.isLetter(input.charAt(0))) { |
| 94 | JOptionPane.showMessageDialog(this, "Digite uma única letra!"); |
| 95 | **return**; |
| 96 | } |
| 97 |  |
| 98 | char letra = input.charAt(0); |
| 99 | boolean acertou = false; |
| 100 |  |
| 101 | **for** (int i = 0; i < palavraSecreta.length(); i++) { |
| 102 | **if** (palavraSecreta.charAt(i) == letra) { |
| 103 | letrasDescobertas[i] = letra; |
| 104 | acertou = true; |
| 105 | } |
| 106 | } |
| 107 |  |
| 108 | **if** (!acertou) { |
| 109 | tentativasRestantes--; |
| 110 | letrasErradas.append(letra).append(" "); |
| 111 | } |
| 112 |  |
| 113 | palavraLabel.setText(String.valueOf(letrasDescobertas)); |
| 114 | enforcadoPanel.repaint(); |
| 115 |  |
| 116 | verificarFimJogo(); |
| 117 | } |
| 118 |  |
| 119 | **private** **void** **verificarFimJogo**() { |
| 120 | **if** (String.valueOf(letrasDescobertas).equals(palavraSecreta)) { |
| 121 | JOptionPane.showMessageDialog(this, "Parabéns! Você ganhou!"); |
| 122 | System.exit(0); |
| 123 | } **else** **if** (tentativasRestantes == 0) { |
| 124 | JOptionPane.showMessageDialog(this, "Game Over! A palavra era: " + palavraSecreta); |
| 125 | System.exit(0); |
| 126 | } |
| 127 | } |
| 128 |  |
| 129 | **public** **static** **void** **main**(String[] args) { |
| 130 | **new** **ForcaGUI**(); |
| 131 | } |
| 132 | } |

**- Código fonte -**

**JogoDaVelha\JogoDaVelhaGUI.java**

|  |  |
| --- | --- |
| 1 | **package** JogoDaVelha; |
| 2 |  |
| 3 | **import** javax.swing.\*; |
| 4 | **import** java.awt.\*; |
| 5 | **import** java.awt.event.ActionEvent; |
| 6 | **import** java.awt.event.ActionListener; |
| 7 |  |
| 8 | **public** **class** **JogoDaVelhaGUI** **extends** **JFrame** { |
| 9 | **private** JButton[][] botoes = **new** **JButton**[3][3]; |
| 10 | **private** char jogadorAtual = 'X'; |
| 11 | **private** boolean jogoAtivo = true; |
| 12 | **private** JLabel statusLabel; |
| 13 |  |
| 14 | **public** **JogoDaVelhaGUI**() { |
| 15 | setTitle("Jogo da Velha"); |
| 16 | setDefaultCloseOpera­tion(JFrame.EXIT\_ON\_CLOSE); |
| 17 | setSize(300, 350); |
| 18 | setLayout(**new** **BorderLayout**()); |
| 19 |  |
| 20 | // Painel de status |
| 21 | statusLabel = **new** **JLabel**("Vez do Jogador " + jogadorAtual, SwingConstants.CENTER); |
| 22 | statusLabel.setFont(**new** **Font**("Arial", Font.BOLD, 16)); |
| 23 | add(statusLabel, BorderLayout.NORTH); |
| 24 |  |
| 25 | // Painel do tabuleiro |
| 26 | JPanel tabuleiroPanel = **new** **JPanel**(**new** **GridLayout**(3, 3)); |
| 27 | tabuleiroPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); |
| 28 |  |
| 29 | // Cria os botões do tabuleiro |
| 30 | **for** (int linha = 0; linha < 3; linha++) { |
| 31 | **for** (int coluna = 0; coluna < 3; coluna++) { |
| 32 | JButton botao = **new** **JButton**(); |
| 33 | botao.setFont(**new** **Font**("Arial", Font.PLAIN, 48)); |
| 34 | botao.setFocusPainted(false); |
| 35 | botao.addActionListener(**new** **BotaoClickListener**(linha, coluna)); |
| 36 | botoes[linha][coluna] = botao; |
| 37 | tabuleiroPanel.add(botao); |
| 38 | } |
| 39 | } |
| 40 |  |
| 41 | add(tabuleiroPanel, BorderLayout.CENTER); |
| 42 | setVisible(true); |
| 43 | } |
| 44 |  |
| 45 | **private** **class** **BotaoClickListener** **implements** **ActionListener** { |
| 46 | **private** int linha; |
| 47 | **private** int coluna; |
| 48 |  |
| 49 | **public** **BotaoClickListener**(int linha, int coluna) { |
| 50 | this.linha = linha; |
| 51 | this.coluna = coluna; |
| 52 | } |
| 53 |  |
| 54 | @Override |
| 55 | **public** **void** **actionPerformed**(ActionEvent e) { |
| 56 | JButton botaoClicado = (JButton) e.getSource(); |
| 57 |  |
| 58 | **if** (botaoClicado.getText().isEmpty() && jogoAtivo) { |
| 59 | botaoClicado.setText(String.valueOf(jogadorAtual)); |
| 60 |  |
| 61 | **if** (verificarVencedor()) { |
| 62 | statusLabel.setText("Jogador " + jogadorAtual + " venceu!"); |
| 63 | jogoAtivo = false; |
| 64 | } **else** **if** (verificarEmpate()) { |
| 65 | statusLabel.setText("Empate!"); |
| 66 | jogoAtivo = false; |
| 67 | } **else** { |
| 68 | trocarJogador(); |
| 69 | statusLabel.setText("Vez do Jogador " + jogadorAtual); |
| 70 | } |
| 71 | } |
| 72 | } |
| 73 | } |
| 74 |  |
| 75 | **private** **void** **trocarJogador**() { |
| 76 | jogadorAtual = (jogadorAtual == 'X') ? 'O' : 'X'; |
| 77 | } |
| 78 |  |
| 79 | **private** boolean **verificarVencedor**() { |
| 80 | // Verifica linhas e colunas |
| 81 | **for** (int i = 0; i < 3; i++) { |
| 82 | **if** (checarCombinacao(botoes[i][0], botoes[i][1], botoes[i][2]) || |
| 83 | checarCombinacao(botoes[0][i], botoes[1][i], botoes[2][i])) { |
| 84 | **return** true; |
| 85 | } |
| 86 | } |
| 87 |  |
| 88 | // Verifica diagonais |
| 89 | **return** checarCombinacao(botoes[0][0], botoes[1][1], botoes[2][2]) || |
| 90 | checarCombinacao(botoes[0][2], botoes[1][1], botoes[2][0]); |
| 91 | } |
| 92 |  |
| 93 | **private** boolean **checarCombinacao**(JButton b1, JButton b2, JButton b3) { |
| 94 | **return** !b1.getText().isEmpty() && |
| 95 | b1.getText().equals(b2.getText()) && |
| 96 | b2.getText().equals(b3.getText()); |
| 97 | } |
| 98 |  |
| 99 | **private** boolean **verificarEmpate**() { |
| 100 | **for** (int i = 0; i < 3; i++) { |
| 101 | **for** (int j = 0; j < 3; j++) { |
| 102 | **if** (botoes[i][j].getText().isEmpty()) { |
| 103 | **return** false; |
| 104 | } |
| 105 | } |
| 106 | } |
| 107 | **return** true; |
| 108 | } |
| 109 |  |
| 110 | **public** **static** **void** **main**(String[] args) { |
| 111 | SwingUtilities.invokeLater(() -> **new** **JogoDaVelhaGUI**()); |
| 112 | } |
| 113 | } |

**- Código fonte -**

**Games\pongzin.pde**

// ============================

// 1. Variáveis Globais

// ============================

int player1Score = 0; // Pontuação do jogador 1

int player2Score = 0; // Pontuação do jogador 2

int maxScore = 5; // Pontuação máxima para vencer

float paddleWidth; // Largura das palhetas

float paddleHeight; // Altura das palhetas

float p1Y, p2Y; // Posição vertical das palhetas

float p1Speed = 6, p2Speed = 6; // Velocidade das palhetas

float ballX, ballY; // Posição da bola

float ballSpeedX, ballSpeedY; // Velocidade da bola

float ballSize = 20; // Tamanho da bola

String gameState = "menu"; // Estado atual: "menu", "jogando", "pause", "vitoria"

boolean versusBot = true; // Modo contra BOT ou dois jogadores

String winner = ""; // Nome do vencedor

String dificuldade = "Normal"; // Nível de dificuldade: "Fácil", "Normal", "Difícil"

float ballSpeedMin, ballSpeedMax, botSpeed; // Configurações baseadas na dificuldade

boolean trainingMode = false; // Modo treino (não conta pontos)

// Botões

Button btnContinuar, btnReiniciar, btnMenu;

Button btnContraBot, btnContraPlayer, btnTreino;

Button btnDificuldade;

Button btnMenuVitoria;

// ============================

// 2. Função setup()

// ============================

void setup() {

size(1250, 720); // Define o tamanho da tela

frameRate(60); // Define a taxa de quadros por segundo

setupButtons(); // Configura os botões

inicializarJogo(); // Inicializa o jogo

}

// ============================

// 3. Função setupButtons()

// ============================

void setupButtons() {

// Cria os botões para o menu, pausa e tela de vitória

btnContinuar = new Button("Continuar", width/2 - 75, 220, 150, 40);

btnReiniciar = new Button("Reiniciar", width/2 - 75, 280, 150, 40);

btnMenu = new Button("Menu", width/2 - 75, 340, 150, 40);

btnContraBot = new Button("Jogar contra BOT", width/2 - 130, 240, 260, 60);

btnContraPlayer = new Button("2 Jogadores", width/2 - 130, 320, 260, 60);

btnTreino = new Button("Modo Treino", width/2 - 130, 400, 260, 60);

btnDificuldade = new Button("Dificuldade: " + dificuldade, width/2 - 130, 480, 260, 60);

btnMenuVitoria = new Button("Voltar ao Menu", width/2 - 100, height/2 + 40, 200, 40);

}

// ============================

// 4. Função configurarDificuldade()

// ============================

void configurarDificuldade() {

// Ajusta as configurações de acordo com a dificuldade selecionada

if (dificuldade.equals("Fácil")) {

paddleHeight = 150;

ballSpeedMin = 3;

ballSpeedMax = 4;

botSpeed = 2;

} else if (dificuldade.equals("Normal")) {

paddleHeight = 100;

ballSpeedMin = 4;

ballSpeedMax = 6;

botSpeed = 4;

} else if (dificuldade.equals("Difícil")) {

paddleHeight = 60;

ballSpeedMin = 6;

ballSpeedMax = 8;

botSpeed = 6;

}

}

// ============================

// 5. Função inicializarJogo()

// ============================

void inicializarJogo() {

configurarDificuldade(); // Define as configurações baseadas na dificuldade

paddleWidth = 20; // Define a largura das palhetas

p1Y = height/2 - paddleHeight/2; // Centraliza as palhetas

p2Y = height/2 - paddleHeight/2;

ballX = width/2; // Centraliza a bola

ballY = height/2;

ballSpeedX = random(1) > 0.5 ? random(ballSpeedMin, ballSpeedMax) : -random(ballSpeedMin, ballSpeedMax); // Velocidade inicial da bola

ballSpeedY = random(-3, 3);

player1Score = 0; // Zera as pontuações

player2Score = 0;

winner = ""; // Limpa o vencedor

}

// ============================

// 6. Função draw()

// ============================

void draw() {

background(20, 20, 50); // Fundo escuro

// Controla o estado atual do jogo

if (gameState.equals("menu")) {

desenharMenu();

} else if (gameState.equals("jogando")) {

atualizarJogo();

desenharJogo();

} else if (gameState.equals("pause")) {

desenharJogo();

desenharPausa();

} else if (gameState.equals("vitoria")) {

desenharVitoria();

}

}

// ============================

// 7. Funções de Desenho

// ============================

void desenharMenu() {

// Exibe o menu inicial com opções de modo de jogo e dificuldade

fill(255);

textAlign(CENTER);

textSize(64);

text("PONG", width/2, 150);

textSize(24);

text("Escolha um modo de jogo e a dificuldade", width/2, 200);

btnContraBot.label = "Jogar contra BOT";

btnContraPlayer.label = "2 Jogadores";

btnTreino.label = trainingMode ? "Desativar Treino" : "Ativar Treino";

btnDificuldade.label = "Dificuldade: " + dificuldade;

btnContraBot.display();

btnContraPlayer.display();

btnTreino.display();

btnDificuldade.display();

}

void desenharPausa() {

// Exibe a tela de pausa com opções para continuar, reiniciar ou voltar ao menu

fill(0, 180);

rect(0, 0, width, height);

fill(255);

textAlign(CENTER);

textSize(36);

text("PAUSA", width/2, 150);

btnContinuar.display();

btnReiniciar.display();

btnMenu.display();

}

void desenharVitoria() {

// Exibe a tela de vitória com o nome do vencedor e opção para voltar ao menu

fill(255);

textAlign(CENTER);

textSize(36);

text("VENCEDOR: " + winner, width/2, height/2 - 40);

textSize(20);

text("Pressione 'M' ou clique para voltar ao menu", width/2, height/2);

btnMenuVitoria.display();

}

void desenharJogo() {

// Desenha a linha central, palhetas, bola e placar

stroke(255);

strokeWeight(2);

for (int i = 0; i < height; i += 20) line(width/2, i, width/2, i+10);

noStroke();

fill(255);

rect(0, p1Y, paddleWidth, paddleHeight);

rect(width - paddleWidth, p2Y, paddleWidth, paddleHeight);

ellipse(ballX, ballY, ballSize, ballSize);

textSize(36);

textAlign(CENTER);

text(player1Score + " x " + player2Score, width/2, 50);

textSize(16);

text("Pressione ESC para pausar", width/2, height - 10);

}

// ============================

// 8. Função atualizarJogo()

// ============================

void atualizarJogo() {

// Movimenta as palhetas dos jogadores

if (keyPressed) {

if (key == 'w') p1Y -= p1Speed;

if (key == 's') p1Y += p1Speed;

if (!versusBot) {

if (keyCode == UP) p2Y -= p2Speed;

if (keyCode == DOWN) p2Y += p2Speed;

}

}

// Limita as palhetas dentro da tela

p1Y = constrain(p1Y, 0, height - paddleHeight);

p2Y = constrain(p2Y, 0, height - paddleHeight);

// Movimento do BOT

if (versusBot) {

float erro = map(abs(ballSpeedX), ballSpeedMin, ballSpeedMax, 50, 10); // Mais rápido = menos erro

float alvo = ballY + random(-erro, erro);

float centroBot = p2Y + paddleHeight / 2;

if (centroBot < alvo - 5) {

p2Y += botSpeed;

} else if (centroBot > alvo + 5) {

p2Y -= botSpeed;

}

}

// Movimenta a bola

ballX += ballSpeedX;

ballY += ballSpeedY;

// Colisão com as bordas superior e inferior

if (ballY < 0 || ballY > height) ballSpeedY \*= -1;

// Colisão com as palhetas

if (ballX < paddleWidth && ballY > p1Y && ballY < p1Y + paddleHeight) ballSpeedX \*= -1;

if (ballX > width - paddleWidth && ballY > p2Y && ballY < p2Y + paddleHeight) ballSpeedX \*= -1;

// Verifica se a bola saiu da tela

if (ballX < 0) {

if (!trainingMode) player2Score++;

resetarBola();

}

if (ballX > width) {

if (!trainingMode) player1Score++;

resetarBola();

}

// Verifica se alguém venceu

if (player1Score >= maxScore) {

gameState = "vitoria";

winner = "Jogador 1";

} else if (player2Score >= maxScore) {

winner = versusBot ? "BOT" : "Jogador 2";

gameState = "vitoria";

}

}

// ============================

// 9. Função resetarBola()

// ============================

void resetarBola() {

// Reseta a posição e velocidade da bola

ballX = width/2;

ballY = height/2;

ballSpeedX = random(1) > 0.5 ? random(ballSpeedMin, ballSpeedMax) : -random(ballSpeedMin, ballSpeedMax);

ballSpeedY = random(-3, 3);

}

// ============================

// 10. Funções de Entrada

// ============================

void keyPressed() {

// Lida com eventos de teclado

if (key == ESC) {

key = 0;

if (gameState.equals("jogando")) {

gameState = "pause";

} else if (gameState.equals("pause")) {

gameState = "jogando";

}

return;

}

if (gameState.equals("vitoria")) {

if (key == 'm' || key == 'M') {

gameState = "menu";

}

}

}

void mousePressed() {

// Lida com cliques nos botões

if (gameState.equals("pause")) {

if (btnContinuar.isMouseOver()) {

gameState = "jogando";

} else if (btnReiniciar.isMouseOver()) {

inicializarJogo();

gameState = "jogando";

} else if (btnMenu.isMouseOver()) {

gameState = "menu";

}

} else if (gameState.equals("menu")) {

if (btnContraBot.isMouseOver()) {

versusBot = true;

inicializarJogo();

gameState = "jogando";

} else if (btnContraPlayer.isMouseOver()) {

versusBot = false;

inicializarJogo();

gameState = "jogando";

} else if (btnTreino.isMouseOver()) {

trainingMode = !trainingMode;

} else if (btnDificuldade.isMouseOver()) {

if (dificuldade.equals("Fácil")) dificuldade = "Normal";

else if (dificuldade.equals("Normal")) dificuldade = "Difícil";

else dificuldade = "Fácil";

}

} else if (gameState.equals("vitoria")) {

if (btnMenuVitoria.isMouseOver()) {

gameState = "menu";

}

}

}

// ============================

// 11. Classe Button

// ============================

class Button {

String label;

float x, y, w, h;

Button(String label, float x, float y, float w, float h) {

this.label = label;

this.x = x;

this.y = y;

this.w = w;

this.h = h;

}

void display() {

// Desenha o botão com destaque se o mouse estiver sobre ele

fill(isMouseOver() ? color(255, 215, 0) : color(255));

stroke(50);

strokeWeight(2);

rect(x, y, w, h, 12);

fill(0);

textAlign(CENTER, CENTER);

textSize(20);

text(label, x + w/2, y + h/2);

}

boolean isMouseOver() {

// Verifica se o mouse está sobre o botão

return mouseX >= x && mouseX <= x + w && mouseY >= y && mouseY <= y + h;

}

}

**- Código fonte -**

**Games\MemoryGame\MemoryGame.pde**

import java.util.ArrayList;

import java.util.Collections;

// ESTADOS DO JOGO

final int MENU = 0, PLAYING = 1, LEVEL\_COMPLETE = 2, GAME\_OVER = 3, GAME\_COMPLETE = 4, PAUSED = 5;

int gameState = MENU;

// MODOS DE JOGO

final int MODE\_PROGRESSIVO = 0, MODE\_INDIVIDUAL = 1;

int gameMode = MODE\_PROGRESSIVO;

// CONFIGURAÇÕES DOS NÍVEIS

final int[][] LEVELS = { {2, 2}, {3, 4}, {4, 4} };

final String[] LEVEL\_NAMES = {"Iniciante (2x2)", "Treinador (3x4)", "Mestre (4x5)"};

final int[] MAX\_ATTEMPTS = { 3, 10, 15 };

final int[] LEVEL\_TIME\_LIMITS = { 15000, 30000, 75000 };

final color BG\_COLOR = #F0F0F0;

final color CARD\_COLOR = #2C3E50;

final color MATCHED\_COLOR = #2ECC71;

final color REVEALED\_COLOR = #E74C3C;

// ESTADO DO JOGO DA MEMÓRIA

ArrayList<Card> cards = new ArrayList<>();

Card firstCard, secondCard;

PImage cardBackRounded;

Theme pokemonTheme;

PImage cardBackImg;

PImage levelbackgroundImg;

PImage backgroundImg;

PImage backgroundImg2;

boolean lockInput = false;

int currentLevel = 0;

int score = 0;

int levelStartTime;

int levelEndTime = -1;

int revealStartTime;

int attempts = 0;

boolean levelBonusApplied = false;

int pauseStartTime = 0;

int totalPausedTime = 0;

// INFORMAÇÕES DO JOGADOR

String playerName = "";

boolean nameEntered = false;

// RANKING DOS JOGADORES (SOMENTE NA SESSÃO)

class RankingEntry {

String name;

int score;

RankingEntry(String name, int score) {

this.name = name;

this.score = score;

}

}

ArrayList<RankingEntry> ranking = new ArrayList<RankingEntry>();

// CONFIGURAÇÕES VISUAIS

final int CARD\_MARGIN = 10;

final int CARD\_ROUNDING = 12;

// ÍCONES PARA A INTERFACE

PImage timeIcon, attemptsIcon, scoreIcon;

HashMap<String, Theme> themes = new HashMap<String, Theme>();

String currentThemeName = "Pokemon";

boolean selectingTheme = false;

void settings() {

size(1000, 600);

}

void setup() {

timeIcon = loadImage("timeIcon.png");

attemptsIcon = loadImage("attemptsIcon.png");

scoreIcon = loadImage("scoreIcon.png");

String[] pokePaths = {

"themes/pokemon/zoroark.png",

"themes/pokemon/charmeleon.png",

"themes/pokemon/clefable.png",

"themes/pokemon/pikachu.png",

"themes/pokemon/haunter.png",

"themes/pokemon/persian.png",

"themes/pokemon/vulpix.png",

"themes/pokemon/empoleon.png",

"themes/pokemon/snorlax.png",

"themes/pokemon/gengar.png"

};

themes.put("Pokemon", new Theme(

"Pokemon",

pokePaths,

"themes/pokemon/cardback.jpg",

"themes/pokemon/background.png",

"themes/pokemon/background22.jpg",

"themes/pokemon/levelbackground.jpg"

));

String[] animePaths = {

"themes/hxh/gon.png",

"themes/hxh/killua.png",

"themes/hxh/hisoka.png",

"themes/hxh/shizuku.png",

"themes/hxh/feitan.png",

"themes/hxh/chrollo.png",

"themes/hxh/kurapika.png",

"themes/hxh/pitou.png",

"themes/hxh/netero.png",

"themes/hxh/leorio.png"

};

themes.put("HXH", new Theme(

"HXH",

animePaths,

"themes/hxh/cardback.jpg",

"themes/hxh/background3.jpg",

"themes/hxh/background222.jpg",

"themes/hxh/levelbackground.png"

));

rectMode(CORNER);

textFont(createFont("Arial", 20));

surface.setTitle("Jogo da Memória - Menu");

applyTheme("Pokemon");

loadAssets();

}

void applyTheme(String name) {

Theme t = themes.get(name);

if (t != null) {

currentThemeName = name;

backgroundImg = t.menuBackground;

cardBackImg = t.cardBack;

levelbackgroundImg = t.levelCompleteBackground;

backgroundImg2 = t.playingBackground;

t.menuBackground.resize(width, height);

t.playingBackground.resize(width, height);

t.levelCompleteBackground.resize(width, height);

}

cardBackRounded = createRoundedImage(cardBackImg, 10);

}

void loadAssets() {

Theme currentTheme = themes.get(currentThemeName);

if (currentTheme == null) return;

backgroundImg = currentTheme.menuBackground;

backgroundImg.resize(width, height);

levelbackgroundImg = currentTheme.levelCompleteBackground;

levelbackgroundImg.resize(width, height);

backgroundImg2 = currentTheme.playingBackground;

backgroundImg2.resize(width, height);

cardBackImg = currentTheme.cardBack;

if (backgroundImg == null) {

println("Erro ao carregar backgroundImg");

}

if (cardBackImg == null) {

println("Erro ao carregar cardBackImg");

}

}

void resetToMenu() {

if (cards != null) cards.clear();

firstCard = null;

secondCard = null;

lockInput = false;

attempts = 0;

score = 0;

levelStartTime = 0;

levelEndTime = -1;

totalPausedTime = 0;

levelBonusApplied = false;

currentLevel = 0;

}

void draw() {

if (gameState == MENU) {

if (backgroundImg != null) {

image(backgroundImg, 0, 0, width, height);

} else {

background(255);

}

} else {

background(255);

}

switch (gameState) {

case MENU:

drawMenu();

break;

case PLAYING:

drawPlaying();

break;

case PAUSED:

drawPlaying();

drawPauseMenu();

break;

case LEVEL\_COMPLETE:

drawLevelComplete();

break;

case GAME\_OVER:

drawGameOver();

break;

case GAME\_COMPLETE:

drawGameComplete();

break;

}

if (lockInput && millis() - revealStartTime > 500) {

if (firstCard != null) firstCard.hide();

if (secondCard != null) secondCard.hide();

firstCard = secondCard = null;

lockInput = false;

}

}

void drawPauseMenu() {

fill(0, 150);

rect(0, 0, width, height);

int modalWidth = 300, modalHeight = 250;

int modalX = width / 2 - modalWidth / 2;

int modalY = height / 2 - modalHeight / 2;

fill(255);

rect(modalX, modalY, modalWidth, modalHeight, 10);

fill(0);

textSize(24);

textAlign(CENTER, CENTER);

text("Pausado", width / 2, modalY + 40);

int btnWidth = 200, btnHeight = 40;

int btnX = width / 2;

int btnY = modalY + 80;

fill(#3498DB);

rectMode(CENTER);

rect(btnX, btnY, btnWidth, btnHeight, 10);

fill(255);

textSize(20);

text("Continuar", btnX, btnY);

btnY += btnHeight + 20;

fill(#3498DB);

rect(btnX, btnY, btnWidth, btnHeight, 10);

fill(255);

text("Menu", btnX, btnY);

btnY += btnHeight + 20;

fill(#3498DB);

rect(btnX, btnY, btnWidth, btnHeight, 10);

fill(255);

text("Sair", btnX, btnY);

rectMode(CORNER);

}

//////////////////////////////

// MENU INICIAL

//////////////////////////////

void drawMenu() {

background(backgroundImg);

// Título

noStroke();

fill(0);

textSize(36);

textAlign(CENTER, CENTER);

text("Jogo da Memória", width/2, height/8);

int themeBtnWidth = 150;

int themeBtnHeight = 40;

int themeBtnX = width - themeBtnWidth - 20;

int themeBtnY = 20;

fill(#ED0239);

rect(themeBtnX, themeBtnY, themeBtnWidth, themeBtnHeight, 10);

fill(255);

textSize(16);

textAlign(CENTER, CENTER);

text("Tema: " + currentThemeName, themeBtnX + themeBtnWidth/2, themeBtnY + themeBtnHeight/2);

// Caixa de entrada para o nome

textSize(24);

textAlign(CENTER, CENTER);

fill(0);

text("Player:", width/2, height/8 + 50);

fill(#EBB09E);

int inputWidth = 300, inputHeight = 40;

rect(width/2 - inputWidth/2, height/8 + 70, inputWidth, inputHeight, 10);

fill(0);

textSize(20);

text(playerName, width/2, height/8 + 70 + inputHeight/2);

if (selectingTheme) {

int tY = themeBtnY + themeBtnHeight + 10;

for (String themeName : themes.keySet()) {

fill(currentThemeName.equals(themeName) ? #ED0239 : #FF7E73);

rect(themeBtnX, tY, themeBtnWidth, themeBtnHeight, 10);

fill(255);

text(themeName, themeBtnX + themeBtnWidth/2, tY + themeBtnHeight/2);

tY += themeBtnHeight + 5;

}

}

// Selecione o modo

textSize(24);

fill(0);

text("Selecione o modo", width/2, height/8 + 140);

// Botões dos modos (horizontal)

int modeBtnWidth = 180, modeBtnHeight = 50;

int modeBtnY = height/8 + 170;

int modeBtnX1 = width/2 - modeBtnWidth - 10;

int modeBtnX2 = width/2 + 10;

fill((gameMode == MODE\_PROGRESSIVO) ? #ED0239 : #FF7E73);

rect(modeBtnX1, modeBtnY, modeBtnWidth, modeBtnHeight, 10);

fill(255);

textSize(18);

text("Modo progressivo", modeBtnX1 + modeBtnWidth/2, modeBtnY + modeBtnHeight/2);

fill((gameMode == MODE\_INDIVIDUAL) ? #ED0239 : #FF7E73);

rect(modeBtnX2, modeBtnY, modeBtnWidth, modeBtnHeight, 10);

fill(255);

text("Modo individual", modeBtnX2 + modeBtnWidth/2, modeBtnY + modeBtnHeight/2);

// Se o modo Individual estiver selecionado, exibe as opções de dificuldade verticalmente

int iniciarY;

if (gameMode == MODE\_INDIVIDUAL) {

textSize(24);

fill(0);

text("Selecione a dificuldade:", width/2, modeBtnY + modeBtnHeight + 40);

int diffBtnWidth = 200, diffBtnHeight = 40;

int diffBtnX = width/2 - diffBtnWidth/2;

int diffBtnY = modeBtnY + modeBtnHeight + 70;

fill((currentLevel == 0) ? #ED0239 : #FF7E73);

rect(diffBtnX, diffBtnY, diffBtnWidth, diffBtnHeight, 10);

fill(255);

textSize(18);

text("Fácil", width/2, diffBtnY + diffBtnHeight/2);

diffBtnY += diffBtnHeight + 10;

fill((currentLevel == 1) ? #ED0239 : #FF7E73);

rect(diffBtnX, diffBtnY, diffBtnWidth, diffBtnHeight, 10);

fill(255);

text("Médio", width/2, diffBtnY + diffBtnHeight/2);

diffBtnY += diffBtnHeight + 10;

fill((currentLevel == 2) ? #ED0239 : #FF7E73);

rect(diffBtnX, diffBtnY, diffBtnWidth, diffBtnHeight, 10);

fill(255);

text("Difícil", width/2, diffBtnY + diffBtnHeight/2);

iniciarY = diffBtnY + diffBtnHeight + 40;

} else {

iniciarY = height - 100;

currentLevel = 0;

}

fill(#F0691F);

rectMode(CENTER);

rect(width/2, iniciarY, 150, 50, 10);

fill(255);

textSize(20);

text("Iniciar", width/2, iniciarY);

rectMode(CORNER);

}

//////////////////////////////

// JOGO EM ANDAMENTO

//////////////////////////////

void drawPlaying() {

background(backgroundImg2);

// Desenha as cartas do grid

for (Card card : cards) {

card.display();

}

// ==== Indicadores: Tentativas, Score, Tempo ====

int elapsedTime = (millis() - levelStartTime) - totalPausedTime;

int timeLeft = (gameState == PAUSED)

? LEVEL\_TIME\_LIMITS[currentLevel] - ((pauseStartTime - levelStartTime) - totalPausedTime)

: max(LEVEL\_TIME\_LIMITS[currentLevel] - elapsedTime, 0);

String timeText = nf(timeLeft / 1000) + "s";

if (timeLeft == 0) {

gameState = GAME\_OVER;

}

// UI layout

int uiMargin = 20;

int iconSize = 30;

int spacing = 10;

int textYOffset = iconSize / 2;

fill(0);

textSize(20);

textAlign(LEFT, CENTER);

// Tentativas

int y = uiMargin;

if (attemptsIcon != null) image(attemptsIcon, uiMargin, y, iconSize, iconSize);

text(attempts + " / " + MAX\_ATTEMPTS[currentLevel], uiMargin + iconSize + spacing, y + textYOffset);

// Score

y += iconSize + spacing;

if (scoreIcon != null) image(scoreIcon, uiMargin, y, iconSize, iconSize);

text(str(score), uiMargin + iconSize + spacing, y + textYOffset);

// Tempo (canto superior direito)

int timeX = width - uiMargin - iconSize;

if (timeIcon != null) image(timeIcon, timeX, uiMargin, iconSize, iconSize);

textAlign(RIGHT, CENTER);

text(timeText, timeX - spacing, uiMargin + textYOffset);

// ==== Verificações de estado ====

// Se todas as cartas foram combinadas, congela o timer e muda para tela de nível concluído

if (allCardsMatched()) {

if (levelEndTime == -1) {

levelEndTime = millis();

}

gameState = LEVEL\_COMPLETE;

}

// Tempo esgotado

if (timeLeft == 0 && gameState == PLAYING) {

gameState = GAME\_OVER;

}

// Tentativas esgotadas

if (attempts > MAX\_ATTEMPTS[currentLevel]) {

gameState = GAME\_OVER;

}

}

//////////////////////////////

// TELA DE NÍVEL CONCLUÍDO

//////////////////////////////

void drawLevelComplete() {

background(levelbackgroundImg);

fill(0, 200);

rect(0, 0, width, height);

fill(255);

textSize(36);

textAlign(CENTER, CENTER);

text("Parabéns, " + playerName + "!", width/2, height/3);

int levelTime = min(levelEndTime - levelStartTime - totalPausedTime, LEVEL\_TIME\_LIMITS[currentLevel]);

// Aplica bônus apenas uma vez

if (!levelBonusApplied) {

int totalPairs = cards.size()/2;

int extraAttempts = attempts - 0; // Aqui somente erros contam

int bonus = max((totalPairs \* 100) - int((levelTime/1000.0)\*5) - (extraAttempts\*20), 0);

score += bonus;

levelBonusApplied = true;

}

textSize(24);

text("Erros: " + attempts, width/2, height/2);

text("Tempo: " + nf(levelTime/1000) + "s", width/2, height/2 + 40);

text("Pontuação: " + score, width/2, height/2 + 80);

// Botão para avançar

fill(#3498DB);

rectMode(CENTER);

rect(width/2, height - 100, 200, 50, 10);

fill(255);

textSize(20);

text((gameMode == MODE\_PROGRESSIVO) ?

((currentLevel < LEVELS.length-1) ? "Próximo Nível" : "Finalizar Jogo") :

"Voltar ao Menu", width/2, height - 100);

rectMode(CORNER);

}

//////////////////////////////

// TELA DE GAME OVER (excesso de tentativas)

//////////////////////////////

void drawGameOver() {

fill(0, 200);

rect(0, 0, width, height);

fill(255, 100, 100);

textSize(36);

textAlign(CENTER, CENTER);

text("Game Over!", width/2, height/3);

textSize(24);

text("Tentativas: " + attempts, width/2, height/3 + 40);

// Se for modo progressivo, exibe dois botões: "Jogar Novamente" e "Voltar ao Menu"

if (gameMode == MODE\_PROGRESSIVO) {

int btnWidth = 200, btnHeight = 50;

int btnY = height - 100;

int btnY2 = height - 180;

int btnX = width/2 + 10;

fill(#3498DB);

rectMode(CENTER);

rect(btnX, btnY2, btnWidth, btnHeight, 10);

rect(btnX, btnY, btnWidth, btnHeight, 10);

fill(255);

textSize(20);

text("Jogar Novamente", btnX, btnY2);

text("Voltar ao Menu", btnX, btnY);

rectMode(CORNER);

} else {

// No modo Individual, apenas "Voltar ao Menu"

fill(#3498DB);

rectMode(CENTER);

rect(width/2, height - 100, 200, 50, 10);

fill(255);

textSize(20);

text("Voltar ao Menu", width/2, height - 100);

rectMode(CORNER);

}

}

//////////////////////////////

// TELA FINAL (após os níveis em modo progressivo ou se individual e finalizar)

//////////////////////////////

void drawGameComplete() {

fill(0, 200);

rect(0, 0, width, height);

fill(255);

textSize(36);

textAlign(CENTER, CENTER);

text("Parabéns, " + playerName + "!", width/2, height/4);

textSize(28);

text("Pontuação Final: " + score, width/2, height/4 + 40);

// Mostra o ranking dos jogadores (ordenado por pontuação decrescente)

textSize(24);

text("Ranking:", width/2, height/2);

for (int i = 0; i < ranking.size(); i++) {

RankingEntry re = ranking.get(i);

text((i+1) + ". " + re.name + " - " + re.score, width/2, height/2 + 30 + i\*30);

}

// Botão para finalizar o jogo

fill(#3498DB);

rectMode(CENTER);

rect(width/2, height - 100, 200, 50, 10);

fill(255);

textSize(20);

text("Finalizar", width/2, height - 100);

rectMode(CORNER);

}

//////////////////////////////

// INÍCIO DE UM NOVO NÍVEL

//////////////////////////////

void startNewLevel(int level) {

int spacing = 10;

int maxGridWidth = width - 2 \* CARD\_MARGIN;

int maxGridHeight = height - 2 \* CARD\_MARGIN;

levelStartTime = millis();

totalPausedTime = 0;

cards.clear();

currentLevel = level;

surface.setTitle("Jogo da Memória - " + LEVEL\_NAMES[level]);

int cols = LEVELS[level][0];

int rows = LEVELS[level][1];

int totalPairs = (cols \* rows) / 2;

Theme currentTheme = themes.get(currentThemeName);

if (currentTheme.cardImages.size() < totalPairs) {

println("Tema não possui imagens suficientes para esse nível.");

resetToMenu();

gameState = MENU;

return;

}

ArrayList<PImage> selectedImages = currentTheme.getShuffledPairs(totalPairs);

int cardSizeW = (maxGridWidth - spacing \* (cols - 1)) / cols;

int cardSizeH = (maxGridHeight - spacing \* (rows - 1)) / rows;

int cardSize = min(cardSizeW, cardSizeH);

int gridWidth = cols \* cardSize + spacing \* (cols - 1);

int gridHeight = rows \* cardSize + spacing \* (rows - 1);

int startX = (width - gridWidth) / 2;

int startY = (height - gridHeight) / 2;

for (int r = 0; r < rows; r++) {

for (int c = 0; c < cols; c++) {

int x = startX + c \* (cardSize + spacing);

int y = startY + r \* (cardSize + spacing);

// Pegando diretamente da lista embaralhada

PImage img = selectedImages.remove(0);

cards.add(new Card(x, y, cardSize, img));

}

}

// Redimensiona a imagem de fundo do verso uma única vez

cardBackImg.resize(cardSize, cardSize);

cardBackRounded = createRoundedImage(cardBackImg, 10);

firstCard = secondCard = null;

lockInput = false;

attempts = 0;

levelBonusApplied = false;

levelEndTime = -1;

}

//////////////////////////////

// MANIPULAÇÃO DE CLIQUES

//////////////////////////////

void mousePressed() {

if (gameState == MENU) {

int inputWidth = 300, inputHeight = 40;

int inputX = width/2 - inputWidth/2;

int inputY = height/8 + 70;

if (mouseX > inputX && mouseX < inputX + inputWidth &&

mouseY > inputY && mouseY < inputY + inputHeight) {

nameEntered = true;

} else {

nameEntered = false;

}

// Clique no botão principal de tema

int themeBtnWidth = 150;

int themeBtnHeight = 40;

int themeBtnX = width - themeBtnWidth - 20;

int themeBtnY = 20;

if (mouseX > themeBtnX && mouseX < themeBtnX + themeBtnWidth &&

mouseY > themeBtnY && mouseY < themeBtnY + themeBtnHeight) {

selectingTheme = !selectingTheme; // abre ou fecha dropdown

return;

}

// Se o dropdown está aberto, checa se clicou em algum tema

if (selectingTheme) {

int tY = themeBtnY + themeBtnHeight + 10;

for (String themeName : themes.keySet()) {

if (mouseX > themeBtnX && mouseX < themeBtnX + themeBtnWidth &&

mouseY > tY && mouseY < tY + themeBtnHeight) {

applyTheme(themeName);

loadAssets(); // recarrega imagens

selectingTheme = false;

return;

}

tY += themeBtnHeight + 5;

}

}

// Botões de modo

int modeBtnWidth = 180, modeBtnHeight = 50;

int modeBtnY = height/8 + 170;

int modeBtnX1 = width/2 - modeBtnWidth - 10;

int modeBtnX2 = width/2 + 10;

if (mouseX > modeBtnX1 && mouseX < modeBtnX1 + modeBtnWidth &&

mouseY > modeBtnY && mouseY < modeBtnY + modeBtnHeight) {

gameMode = MODE\_PROGRESSIVO;

}

if (mouseX > modeBtnX2 && mouseX < modeBtnX2 + modeBtnWidth &&

mouseY > modeBtnY && mouseY < modeBtnY + modeBtnHeight) {

gameMode = MODE\_INDIVIDUAL;

}

// Se modo Individual, verifica botões de dificuldade

if (gameMode == MODE\_INDIVIDUAL) {

int diffBtnWidth = 200, diffBtnHeight = 40;

int diffBtnX = width/2 - diffBtnWidth/2;

int diffBtnY = modeBtnY + modeBtnHeight + 70;

if (mouseX > diffBtnX && mouseX < diffBtnX + diffBtnWidth &&

mouseY > diffBtnY && mouseY < diffBtnY + diffBtnHeight) {

currentLevel = 0;

}

diffBtnY += diffBtnHeight + 10;

if (mouseX > diffBtnX && mouseX < diffBtnX + diffBtnWidth &&

mouseY > diffBtnY && mouseY < diffBtnY + diffBtnHeight) {

currentLevel = 1;

}

diffBtnY += diffBtnHeight + 10;

if (mouseX > diffBtnX && mouseX < diffBtnX + diffBtnWidth &&

mouseY > diffBtnY && mouseY < diffBtnY + diffBtnHeight) {

currentLevel = 2;

}

int iniciarY = diffBtnY + diffBtnHeight + 40;

if (mouseX > width/2 - 75 && mouseX < width/2 + 75 &&

mouseY > iniciarY - 25 && mouseY < iniciarY + 25 && playerName.length() > 0) {

score = 0;

startNewLevel(currentLevel);

gameState = PLAYING;

}

} else {

int iniciarY = height - 100;

if (mouseX > width/2 - 75 && mouseX < width/2 + 75 &&

mouseY > iniciarY - 25 && mouseY < iniciarY + 25 && playerName.length() > 0) {

score = 0;

currentLevel = 0;

startNewLevel(currentLevel);

gameState = PLAYING;

}

}

return;

}

// Se estiver no modo de jogo

if (gameState == PLAYING) {

if (allCardsMatched()) {

checkNextButtonClick();

return;

}

if (lockInput) return;

for (Card card : cards) {

if (card.isMouseOver()) {

handleCardClick(card);

break;

}

}

return;

}

// Se o jogo está pausado, trata os cliques no modal de pausa

if (gameState == PAUSED) {

// Janela modal centralizada

int modalWidth = 300, modalHeight = 250;

int modalX = width/2 - modalWidth/2;

int modalY = height/2 - modalHeight/2;

// Botões da janela

int btnWidth = 200, btnHeight = 40;

int btnX = width/2;

int btnY\_cont = modalY + 80; // "Continuar"

int btnY\_menu = btnY\_cont + btnHeight + 20; // "Menu"

int btnY\_exit = btnY\_menu + btnHeight + 20; // "Sair"

if (mouseY > btnY\_cont - btnHeight/2 && mouseY < btnY\_cont + btnHeight/2) {

int pauseDuration = millis() - pauseStartTime;

totalPausedTime += pauseDuration; // Acumula o tempo pausado

gameState = PLAYING;

}

if (mouseX > btnX - btnWidth/2 && mouseX < btnX + btnWidth/2) {

if (mouseY > btnY\_cont - btnHeight/2 && mouseY < btnY\_cont + btnHeight/2) {

// Retoma o jogo

gameState = PLAYING;

} else if (mouseY > btnY\_menu - btnHeight/2 && mouseY < btnY\_menu + btnHeight/2) {

resetToMenu();

gameState = MENU;

} else if (mouseY > btnY\_exit - btnHeight/2 && mouseY < btnY\_exit + btnHeight/2) {

exit();

}

}

return;

}

// Se estiver na tela de nível completo ou game over, trata os botões

if (gameState == LEVEL\_COMPLETE) {

checkNextButtonClick();

return;

}

//GAMEOVER

if (gameState == GAME\_OVER) {

if (gameMode == MODE\_PROGRESSIVO) {

int btnWidth = 200, btnHeight = 50;

int btnY\_jogar = height - 180; // "Jogar Novamente"

int btnY\_menu = height - 100; // "Voltar ao Menu"

int btnX = width/2; // Centralizado

if (mouseX > btnX - btnWidth/2 && mouseX < btnX + btnWidth/2 &&

mouseY > btnY\_jogar - btnHeight/2 && mouseY < btnY\_jogar + btnHeight/2) {

startNewLevel(currentLevel);

gameState = PLAYING;

} else if (mouseX > btnX - btnWidth/2 && mouseX < btnX + btnWidth/2 &&

mouseY > btnY\_menu - btnHeight/2 && mouseY < btnY\_menu + btnHeight/2) {

resetToMenu();

gameState = MENU;

}

} else { // Modo Individual

int btnY = height - 100;

if (mouseX > width/2 - 100 && mouseX < width/2 + 100 &&

mouseY > btnY - 25 && mouseY < btnY + 25) {

resetToMenu();

gameState = MENU;

}

}

return;

}

if (gameState == GAME\_COMPLETE) {

// Agora com duas opções: "Finalizar" e "Voltar ao Menu"

int btnWidth = 200, btnHeight = 50;

int btnY\_final = height - 180; // "Finalizar" botão

int btnY\_menu = height - 100; // "Voltar ao Menu"

int btnX = width/2;

if (mouseX > btnX - btnWidth/2 && mouseX < btnX + btnWidth/2) {

if (mouseY > btnY\_final - btnHeight/2 && mouseY < btnY\_final + btnHeight/2) {

exit();

} else if (mouseY > btnY\_menu - btnHeight/2 && mouseY < btnY\_menu + btnHeight/2) {

resetToMenu();

gameState = MENU;

}

}

return;

}

}

void keyPressed() {

if (key == ESC) {

key = 0;

if (gameState == PLAYING) {

gameState = PAUSED;

pauseStartTime = millis(); // marca quando começou a pausa

} else if (gameState == PAUSED) {

gameState = PLAYING;

totalPausedTime += millis() - pauseStartTime; // acumula tempo pausado

}

}

if (gameState == MENU && nameEntered) {

if (key == BACKSPACE) {

if (playerName.length() > 0)

playerName = playerName.substring(0, playerName.length()-1);

} else if (key == ENTER || key == RETURN) {

nameEntered = false;

} else {

playerName += key;

}

}

// Se estiver jogando, aperta Esc para pausar

if (gameState == PLAYING && key == ESC) {

key = 0; // Evita que o Processing feche o sketch

gameState = PAUSED;

}

// Se estiver pausado, o Esc pode também retomar o jogo (opcional)

}

void handleCardClick(Card card) {

if (card.matched || card.revealed) return;

card.reveal();

if (firstCard == null) {

firstCard = card;

} else {

secondCard = card;

if (firstCard.img == secondCard.img) {

firstCard.match();

secondCard.match();

score += 100;

firstCard = secondCard = null;

} else {

attempts++;

lockInput = true;

revealStartTime = millis();

}

}

}

void checkNextButtonClick() {

// Botão nas telas de nível completo ou game over

if (mouseX > width/2 - 100 && mouseX < width/2 + 100 &&

mouseY > height - 125 && mouseY < height - 75) {

if (gameState == LEVEL\_COMPLETE) {

if (gameMode == MODE\_PROGRESSIVO) {

if (currentLevel < LEVELS.length-1) {

startNewLevel(currentLevel + 1);

gameState = PLAYING;

} else {

// Último nível: adiciona ao ranking e finaliza o jogo

ranking.add(new RankingEntry(playerName, score));

gameState = GAME\_COMPLETE;

}

} else {

// No modo individual, após terminar um nível, volta para o menu

resetToMenu();

gameState = MENU;

}

}

}

}

boolean allCardsMatched() {

for (Card card : cards) {

if (!card.matched) return false;

}

return !cards.isEmpty();

}

PImage createRoundedImage(PImage img, float radius) {

PGraphics mask = createGraphics(img.width, img.height);

mask.beginDraw();

mask.background(0);

mask.fill(255);

mask.noStroke();

mask.rect(0, 0, img.width, img.height, radius);

mask.endDraw();

PImage result = img.copy();

result.mask(mask.get());

return result;

}

//////////////////////////////

// CLASSE CARTA

//////////////////////////////

class Card {

float x, y, size;

boolean revealed = false;

boolean matched = false;

PImage img;

Card(int x, int y, int size, PImage img) {

this.x = x;

this.y = y;

this.size = size;

this.img = img;

}

void display() {

stroke(50);

strokeWeight(2);

if (matched) {

fill(MATCHED\_COLOR);

rect(x, y, size, size, CARD\_ROUNDING);

imageMode(CENTER);

image(img, x + size/2, y + size/2, size \* 0.8, size \* 0.8);

} else if (revealed) {

fill(REVEALED\_COLOR);

rect(x, y, size, size, CARD\_ROUNDING);

imageMode(CENTER);

image(img, x + size/2, y + size/2, size \* 0.8, size \* 0.8);

} else {

fill(CARD\_COLOR);

rect(x, y, size, size, CARD\_ROUNDING);

imageMode(CENTER);

image(cardBackRounded, x + size/2, y + size/2, size \* 0.95, size \* 0.95);

}

noStroke();

}

boolean isMouseOver() {

return mouseX > x && mouseX < x + size && mouseY > y && mouseY < y + size;

}

void reveal() {

revealed = true;

}

void hide() {

revealed = false;

}

void match() {

matched = true;

}

boolean isMatch(Card other) {

return this.img == other.img;

}

}

void generateCards(int level, Theme theme) {

cards.clear();

int cols = LEVELS[level][0];

int rows = LEVELS[level][1];

int totalPairs = (cols \* rows) / 2;

ArrayList<PImage> images = theme.getShuffledPairs(totalPairs);

// Calcula tamanho das cartas e centraliza o grid

int availableWidth = width - 2 \* CARD\_MARGIN;

int availableHeight = height - 100; // Espaço para UI no topo

int cardSize = min(availableWidth / cols, availableHeight / rows);

int gridX = (availableWidth - cols \* cardSize) / 2 + CARD\_MARGIN;

int gridY = (availableHeight - rows \* cardSize) / 2 + 80; // Espaço para UI

ArrayList<Integer> indices = new ArrayList<Integer>();

for (int i = 0; i < totalPairs; i++) {

indices.add(i);

indices.add(i); // Para pares

}

Collections.shuffle(indices);

for (int row = 0; row < rows; row++) {

for (int col = 0; col < cols; col++) {

int x = gridX + col \* cardSize;

int y = gridY + row \* cardSize;

int index = indices.remove(0);

PImage cardImg = images.get(index);

cards.add(new Card(x, y, cardSize, cardImg));

}

}

}

class Theme {

String name;

ArrayList<PImage> cardImages;

PImage cardBack;

PImage menuBackground;

PImage playingBackground;

PImage levelCompleteBackground;

Theme(String name, String[] imagePaths, String cardBackPath, String menuBG, String playBG, String levelBG) {

this.name = name;

cardImages = new ArrayList<PImage>();

for (String path : imagePaths) {

cardImages.add(loadImage(path));

}

cardBack = loadImage(cardBackPath);

menuBackground = loadImage(menuBG);

playingBackground = loadImage(playBG);

levelCompleteBackground = loadImage(levelBG);

}

ArrayList<PImage> getShuffledPairs(int pairCount) {

ArrayList<PImage> available = new ArrayList<PImage>(cardImages);

Collections.shuffle(available);

if (available.size() < pairCount) {

println("Erro: Tema '" + name + "' não possui imagens suficientes para " + pairCount + " pares.");

return new ArrayList<PImage>();

}

ArrayList<PImage> selected = new ArrayList<PImage>();

for (int i = 0; i < pairCount; i++) {

PImage img = available.get(i);

selected.add(img);

selected.add(img);

}

Collections.shuffle(selected);

return selected;

}

}