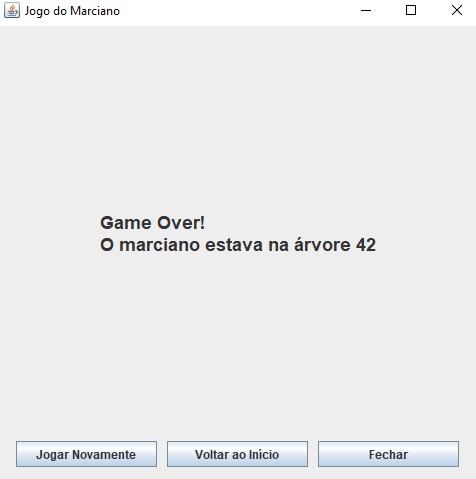
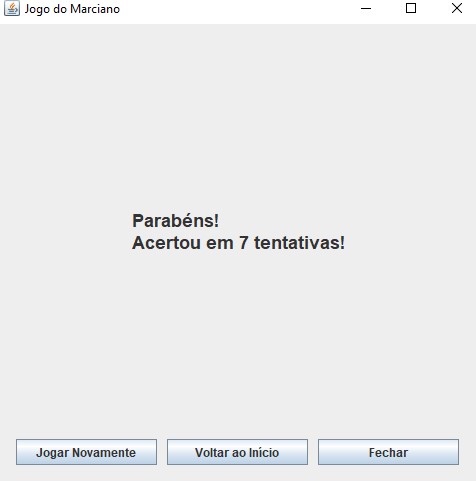
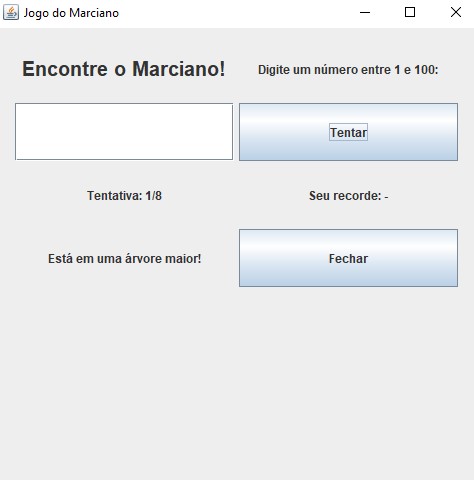
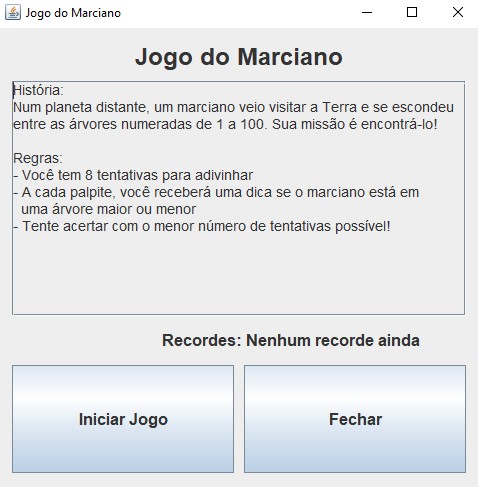
Game Studio: Tigre Produções

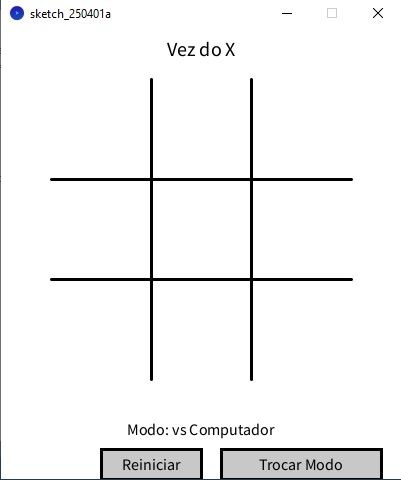
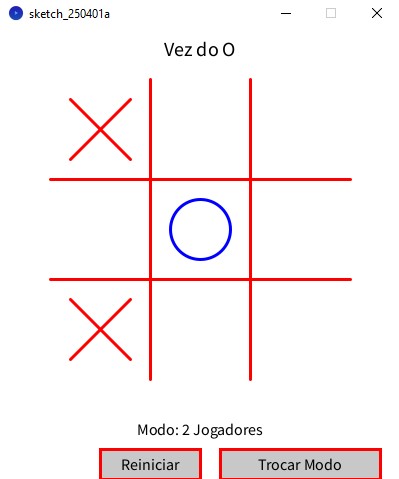
Eliel Lucas Trajano Neto – 01606048

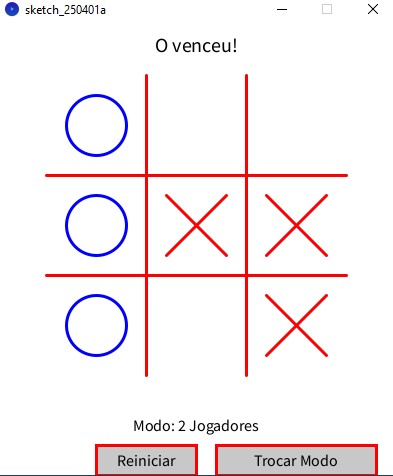
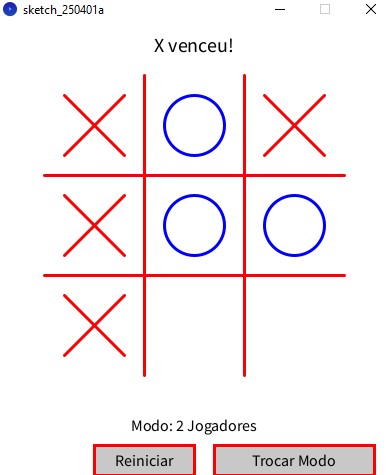
Rodrigo Andrade Cavalcanti Muniz – 01606059

1. Jogo do Marciano

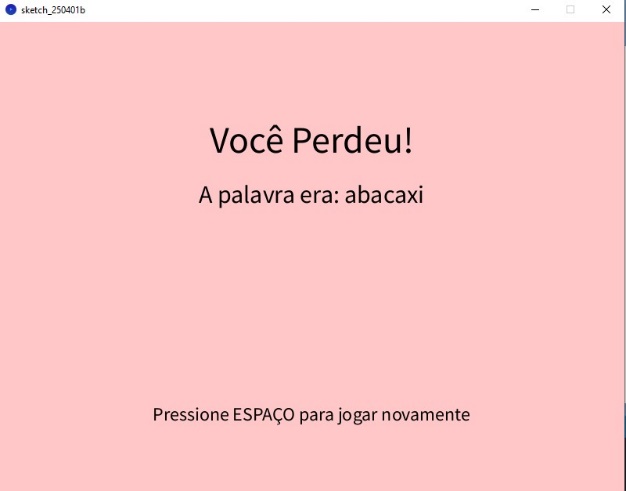
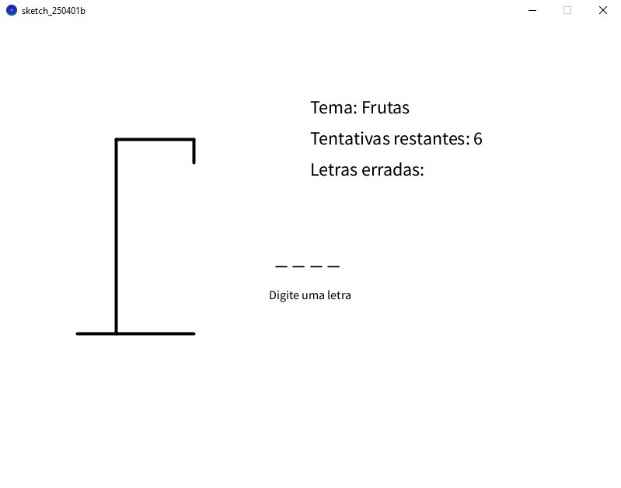
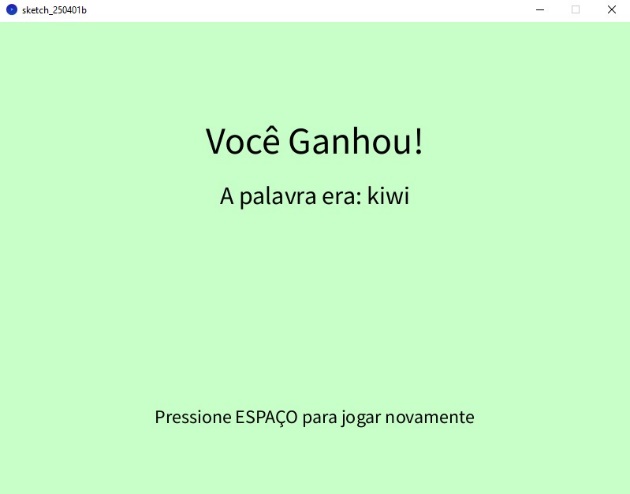


1. Jogo da velha

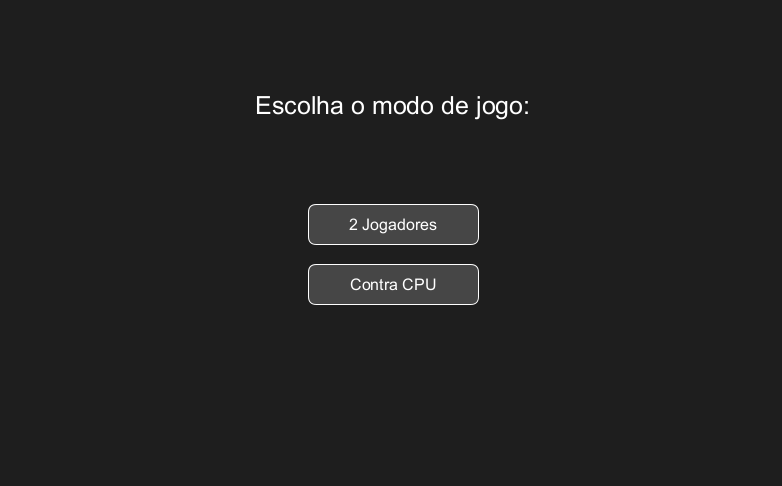


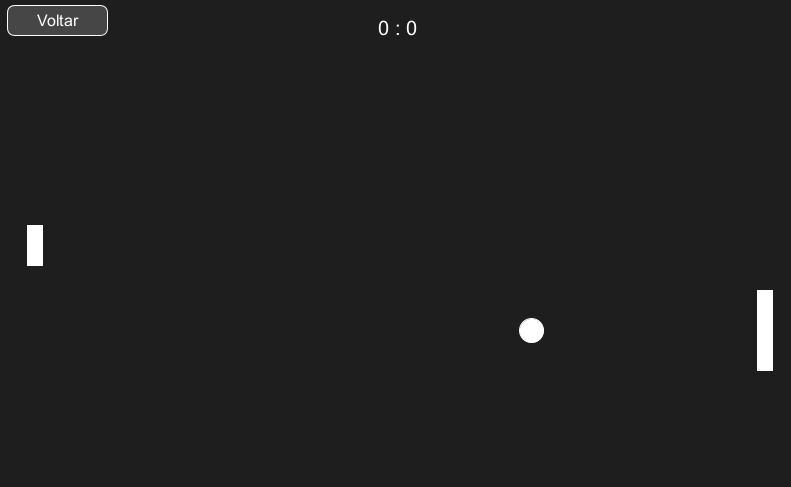
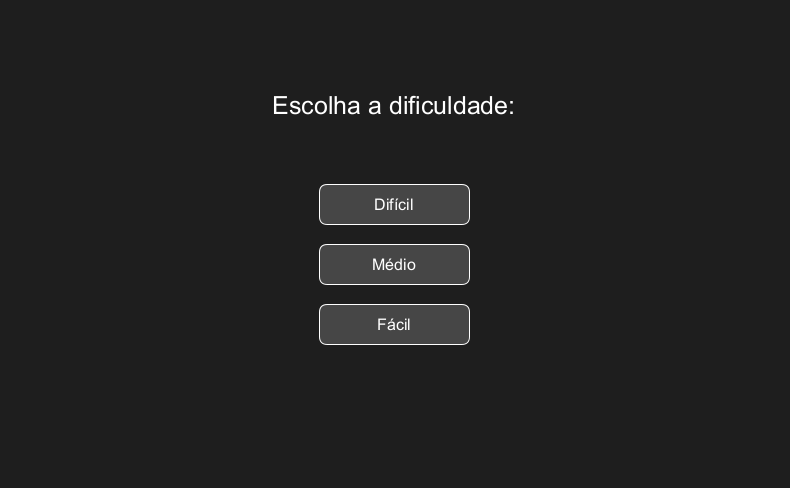


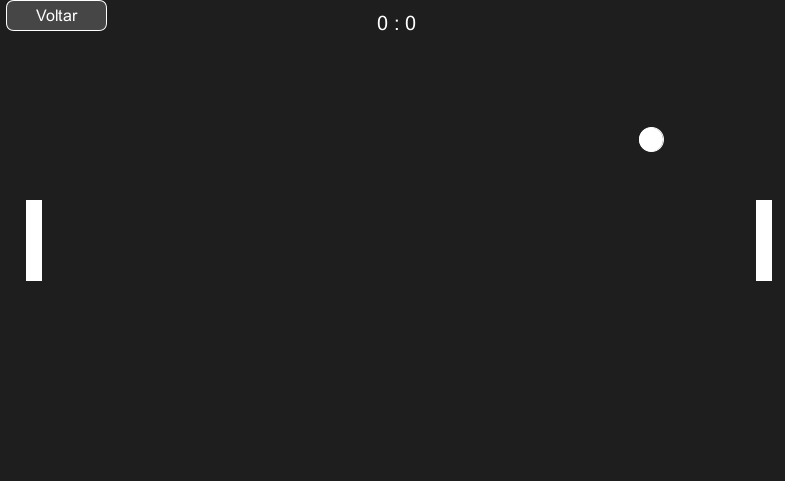
1. Jogo da forca



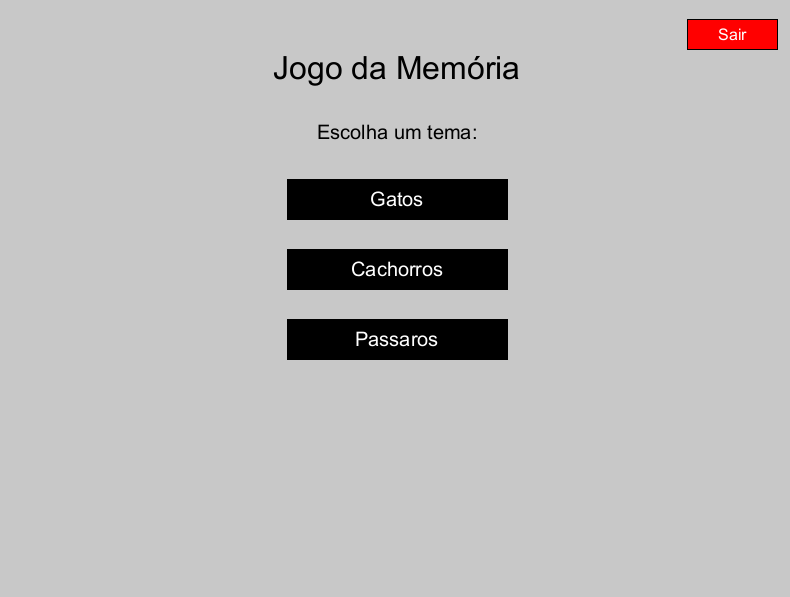
1. Jogo do Pong

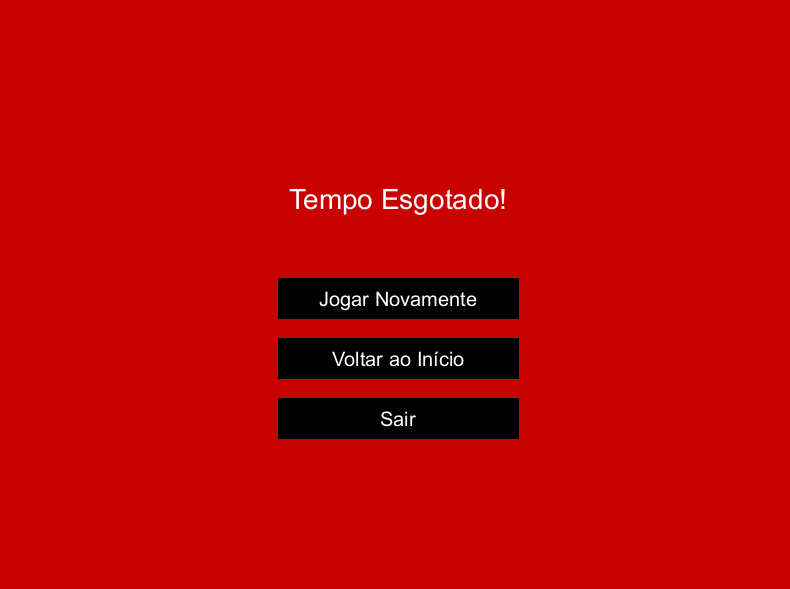
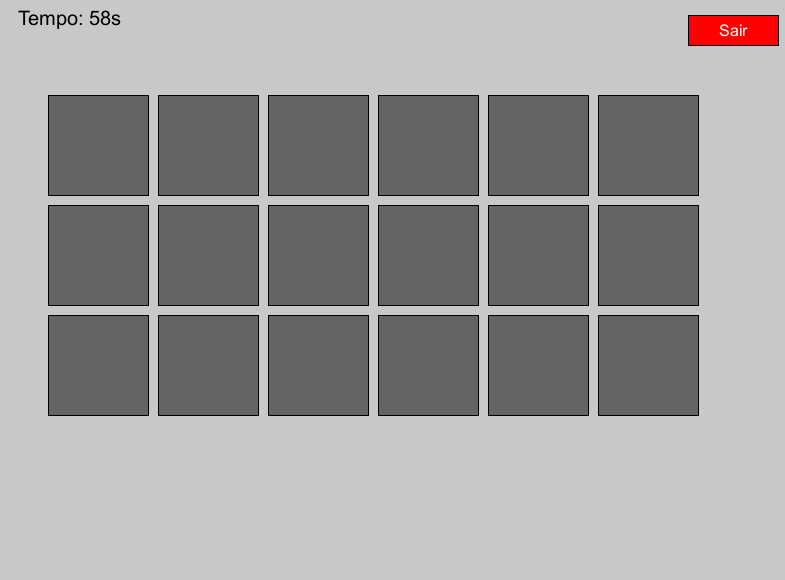


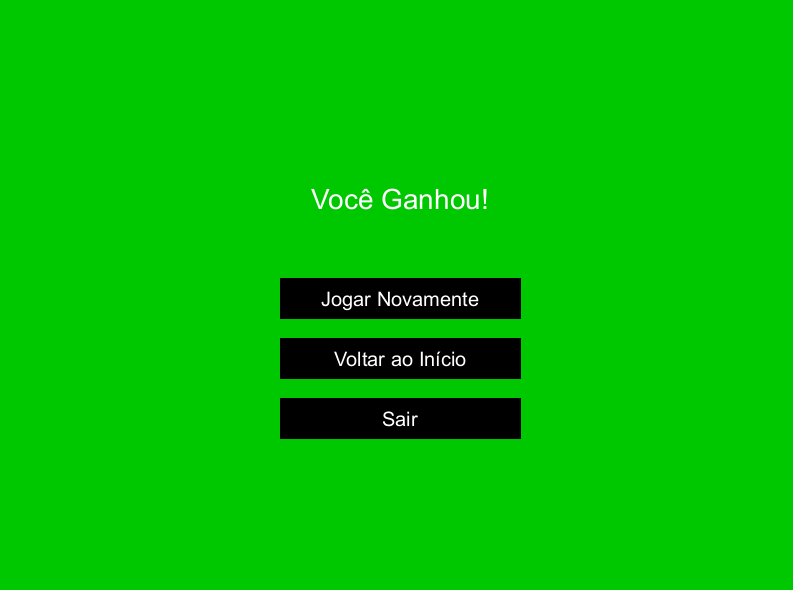
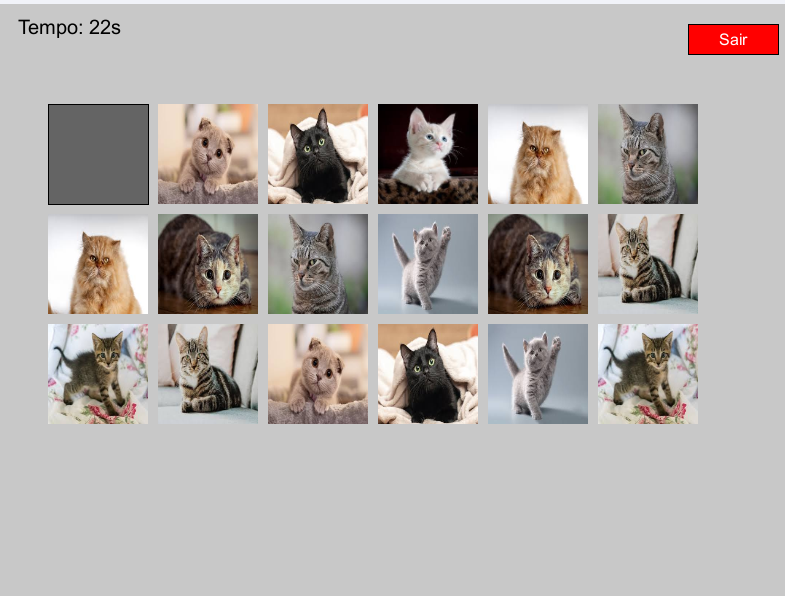




5. Jogo da Memória







1. Jogo do marciano
2. import javax.swing.*\**;
3. import java.awt.*\**;
4. import java.util.Random;
5. import java.util.ArrayList;
6. import java.util.Collections;
7. public class JogoMarcianoFinal extends JFrame {
8. private final *int* LIMITE\_TENTATIVAS = 8;
9. private ArrayList<Integer> recordes = **new** ArrayList<>();
11. // Componentes da tela inicial
12. private JPanel initialPanel;
13. private JButton iniciarButton, fecharButton1;
14. private JTextArea historiaArea;
15. private JLabel recordesLabel;
17. // Componentes do jogo
18. private JPanel gamePanel;
19. private JLabel tentativaLabel;
20. private JTextField entradaField;
21. private JLabel mensagemLabel;
22. private JLabel recordeAtualLabel;
23. private JButton fecharButton2;
25. // Componentes do fim de jogo
26. private JPanel endPanel;
27. private JButton fecharButton3;
29. // Variáveis do jogo
30. private *int* arvore;
31. private *int* tentativas;
32. public JogoMarcianoFinal() {
33. initComponents();
34. mostrarTelaInicial();
35. }
36. private *void* initComponents() {
37. setTitle("Jogo do Marciano");
38. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
39. setSize(500, 500);
40. setLocationRelativeTo(null);
41. setLayout(**new** CardLayout());
42. // Tela inicial
43. initialPanel = **new** JPanel(**new** BorderLayout(10, 10));
44. initialPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));
46. JLabel tituloInicial = **new** JLabel("Jogo do Marciano", *SwingConstants*.CENTER);
47. tituloInicial.setFont(**new** Font("Arial", Font.BOLD, 24));
49. // Área de história com rolagem
50. historiaArea = **new** JTextArea(
51. "História:\n" +
52. "Num planeta distante, um marciano veio visitar a Terra e se escondeu\n" +
53. "entre as árvores numeradas de 1 a 100. Sua missão é encontrá-lo!\n\n" +
54. "Regras:\n" +
55. "- Você tem " + LIMITE\_TENTATIVAS + " tentativas para adivinhar\n" +
56. "- A cada palpite, você receberá uma dica se o marciano está em\n" +
57. "  uma árvore maior ou menor\n" +
58. "- Tente acertar com o menor número de tentativas possível!"
59. );
60. historiaArea.setFont(**new** Font("Arial", Font.PLAIN, 14));
61. historiaArea.setEditable(false);
62. historiaArea.setLineWrap(true);
63. historiaArea.setWrapStyleWord(true);
64. historiaArea.setBackground(getBackground());
66. recordesLabel = **new** JLabel("Recordes: Nenhum recorde ainda", *SwingConstants*.CENTER);
67. recordesLabel.setFont(**new** Font("Arial", Font.BOLD, 16));
69. // Painel de botões da tela inicial
70. JPanel botoesInicioPanel = **new** JPanel(**new** GridLayout(1, 2, 10, 10));
71. iniciarButton = **new** JButton("Iniciar Jogo");
72. iniciarButton.setFont(**new** Font("Arial", Font.BOLD, 16));
73. iniciarButton.addActionListener(*e* *->* iniciarJogo());
75. fecharButton1 = **new** JButton("Fechar");
76. fecharButton1.setFont(**new** Font("Arial", Font.BOLD, 16));
77. fecharButton1.addActionListener(*e* *->* System.exit(0));
79. botoesInicioPanel.add(iniciarButton);
80. botoesInicioPanel.add(fecharButton1);
82. JPanel centerPanel = **new** JPanel();
83. centerPanel.setLayout(**new** BoxLayout(centerPanel, BoxLayout.Y\_AXIS));
84. centerPanel.add(**new** JScrollPane(historiaArea));
85. centerPanel.add(Box.createRigidArea(**new** Dimension(0, 15)));
86. centerPanel.add(recordesLabel);
87. centerPanel.add(Box.createRigidArea(**new** Dimension(0, 15)));
88. centerPanel.add(botoesInicioPanel);
90. initialPanel.add(tituloInicial, BorderLayout.NORTH);
91. initialPanel.add(centerPanel, BorderLayout.CENTER);
93. // Tela do jogo
94. gamePanel = **new** JPanel();
95. gamePanel.setLayout(**new** GridLayout(7, 1, 5, 5)); // Aumentei para 7 linhas
96. gamePanel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
98. JLabel tituloJogo = **new** JLabel("Encontre o Marciano!", *SwingConstants*.CENTER);
99. tituloJogo.setFont(**new** Font("Arial", Font.BOLD, 20));
101. JLabel instrucaoLabel = **new** JLabel("Digite um número entre 1 e 100:", *SwingConstants*.CENTER);
103. entradaField = **new** JTextField();
104. entradaField.setHorizontalAlignment(JTextField.CENTER);
106. JButton tentarButton = **new** JButton("Tentar");
107. tentarButton.addActionListener(*e* *->* verificarPalpite());
109. tentativaLabel = **new** JLabel("Tentativa: 0/" + LIMITE\_TENTATIVAS, *SwingConstants*.CENTER);
110. recordeAtualLabel = **new** JLabel("Seu recorde: -", *SwingConstants*.CENTER);
111. mensagemLabel = **new** JLabel("", *SwingConstants*.CENTER);
113. fecharButton2 = **new** JButton("Fechar");
114. fecharButton2.addActionListener(*e* *->* System.exit(0));
116. gamePanel.add(tituloJogo);
117. gamePanel.add(instrucaoLabel);
118. gamePanel.add(entradaField);
119. gamePanel.add(tentarButton);
120. gamePanel.add(tentativaLabel);
121. gamePanel.add(recordeAtualLabel);
122. gamePanel.add(mensagemLabel);
123. gamePanel.add(fecharButton2);
125. // Tela de fim de jogo
126. endPanel = **new** JPanel(**new** BorderLayout(10, 20));
127. endPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
129. JLabel fimLabel = **new** JLabel("", *SwingConstants*.CENTER);
130. fimLabel.setFont(**new** Font("Arial", Font.BOLD, 18));
131. endPanel.add(fimLabel, BorderLayout.CENTER);
133. JPanel botoesFimPanel = **new** JPanel(**new** GridLayout(1, 3, 10, 10));
134. JButton reiniciarButton = **new** JButton("Jogar Novamente");
135. reiniciarButton.addActionListener(*e* *->* iniciarJogo());
137. JButton voltarButton = **new** JButton("Voltar ao Início");
138. voltarButton.addActionListener(*e* *->* mostrarTelaInicial());
140. fecharButton3 = **new** JButton("Fechar");
141. fecharButton3.addActionListener(*e* *->* System.exit(0));
143. botoesFimPanel.add(reiniciarButton);
144. botoesFimPanel.add(voltarButton);
145. botoesFimPanel.add(fecharButton3);
146. endPanel.add(botoesFimPanel, BorderLayout.SOUTH);
148. // Adiciona todos os painéis
149. add(initialPanel, "inicio");
150. add(gamePanel, "jogo");
151. add(endPanel, "fim");
152. }
153. private *void* mostrarTelaInicial() {
154. atualizarRecordesLabel();
155. ((CardLayout) getContentPane().getLayout()).show(getContentPane(), "inicio");
156. }
157. private *void* iniciarJogo() {
158. Random random = **new** Random();
159. arvore = random.nextInt(100) + 1;
160. tentativas = 0;
162. tentativaLabel.setText("Tentativa: 0/" + LIMITE\_TENTATIVAS);
163. mensagemLabel.setText("");
164. entradaField.setText("");
166. if (!recordes.isEmpty()) {
167. recordeAtualLabel.setText("Seu recorde: " + Collections.min(recordes) + " tentativas");
168. } else {
169. recordeAtualLabel.setText("Seu recorde: -");
170. }
172. ((CardLayout) getContentPane().getLayout()).show(getContentPane(), "jogo");
173. entradaField.requestFocus();
174. }
175. private *void* verificarPalpite() {
176. try {
177. *int* palpite = Integer.parseInt(entradaField.getText());
179. if (palpite < 1 || palpite > 100) {
180. mensagemLabel.setText("Digite um número entre 1 e 100!");
181. return;
182. }
184. tentativas++;
185. tentativaLabel.setText("Tentativa: " + tentativas + "/" + LIMITE\_TENTATIVAS);
187. if (palpite == arvore) {
188. finalizarJogo(true);
189. return;
190. }
192. if (tentativas >= LIMITE\_TENTATIVAS) {
193. finalizarJogo(false);
194. return;
195. }
197. if (palpite > arvore) {
198. mensagemLabel.setText("Está em uma árvore menor!");
199. } else {
200. mensagemLabel.setText("Está em uma árvore maior!");
201. }
203. entradaField.setText("");
204. } catch (NumberFormatException *ex*) {
205. mensagemLabel.setText("Digite um número válido!");
206. }
207. }
208. private *void* finalizarJogo(*boolean* *vitoria*) {
209. JLabel fimLabel = (JLabel) endPanel.getComponent(0);
211. if (*vitoria*) {
212. fimLabel.setText("<html>Parabéns!<br>Acertou em " + tentativas + " tentativas!</html>");
213. recordes.add(tentativas);
214. Collections.sort(recordes);
215. } else {
216. fimLabel.setText("<html>Game Over!<br>O marciano estava na árvore " + arvore + "</html>");
217. }
219. ((CardLayout) getContentPane().getLayout()).show(getContentPane(), "fim");
220. }
221. private *void* atualizarRecordesLabel() {
222. if (recordes.isEmpty()) {
223. recordesLabel.setText("Recordes: Nenhum recorde ainda");
224. } else {
225. StringBuilder sb = **new** StringBuilder("<html>Recordes:<br>");
226. *int* max = Math.min(5, recordes.size());
228. for (*int* i = 0; i < max; i++) {
229. sb.append(i+1).append(". ").append(recordes.get(i)).append(" tentativas<br>");
230. }
232. sb.append("</html>");
233. recordesLabel.setText(sb.toString());
234. }
235. }
236. public static *void* main(String[] *args*) {
237. SwingUtilities.invokeLater(() *->* {
238. JogoMarcianoFinal jogo = **new** JogoMarcianoFinal();
239. jogo.setVisible(true);
240. });
241. }
242. }

2. Jogo da velha

// Variáveis globais

int boardSize = 300; // Tamanho do tabuleiro

int cellSize = boardSize / 3; // Tamanho de cada célula

int[][] board = new int[3][3]; // 0 = vazio, 1 = X, 2 = O

int currentPlayer = 1; // Começa com X

boolean gameOver = false;

int winner = 0; // 0 = sem vencedor, 1 = X, 2 = O, 3 = empate

boolean vsComputer = false; // Modo de jogo (true = vs computador, false = vs jogador)

void setup() {

size(400, 450); // Largura x Altura (incluindo espaço para botões)

resetGame();

// Configuração do texto

textSize(24);

textAlign(CENTER, CENTER);

}

void draw() {

background(255);

// Desenha o tabuleiro

drawBoard();

// Desenha as marcações (X e O)

drawMarks();

// Desenha o status do jogo

drawStatus();

// Desenha os botões

drawButtons();

// Verifica se é a vez do computador no modo vs computador

if (vsComputer && currentPlayer == 2 && !gameOver) {

delay(500); // Pequeno atraso para parecer mais natural

computerMove();

}

}

void drawBoard() {

strokeWeight(3);

line(50, 150, 350, 150); // Linha horizontal superior

line(50, 250, 350, 250); // Linha horizontal inferior

line(150, 50, 150, 350); // Linha vertical esquerda

line(250, 50, 250, 350); // Linha vertical direita

}

void drawMarks() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

float x = 50 + j \* cellSize + cellSize/2;

float y = 50 + i \* cellSize + cellSize/2;

if (board[i][j] == 1) { // X

stroke(255, 0, 0);

line(x - 30, y - 30, x + 30, y + 30);

line(x + 30, y - 30, x - 30, y + 30);

} else if (board[i][j] == 2) { // O

stroke(0, 0, 255);

noFill();

ellipse(x, y, 60, 60);

}

}

}

}

void drawStatus() {

fill(0);

textSize(20);

if (gameOver) {

if (winner == 3) {

text("Empate!", width/2, 20);

} else if (winner == 1) {

text("X venceu!", width/2, 20);

} else if (winner == 2) {

text("O venceu!", width/2, 20);

}

} else {

if (currentPlayer == 1) {

text("Vez do X", width/2, 20);

} else {

text("Vez do O", width/2, 20);

}

}

// Mostra o modo de jogo

textSize(16);

if (vsComputer) {

text("Modo: vs Computador", width/2, 400);

} else {

text("Modo: 2 Jogadores", width/2, 400);

}

}

void drawButtons() {

// Botão de reiniciar

fill(200);

rect(100, 420, 100, 30);

fill(0);

textSize(16);

text("Reiniciar", 150, 435);

// Botão de trocar modo

fill(200);

rect(220, 420, 160, 30);

fill(0);

text("Trocar Modo", 300, 435);

}

void mousePressed() {

// Verifica clique nos botões

if (mouseX >= 100 && mouseX <= 200 && mouseY >= 420 && mouseY <= 450) {

resetGame();

return;

}

if (mouseX >= 220 && mouseX <= 380 && mouseY >= 420 && mouseY <= 450) {

vsComputer = !vsComputer;

resetGame();

return;

}

// Se o jogo acabou, não processa cliques no tabuleiro

if (gameOver) return;

// Verifica se o clique foi dentro do tabuleiro

if (mouseX >= 50 && mouseX <= 350 && mouseY >= 50 && mouseY <= 350) {

int row = (mouseY - 50) / cellSize;

int col = (mouseX - 50) / cellSize;

// Verifica se a célula está vazia

if (board[row][col] == 0) {

board[row][col] = currentPlayer;

// Verifica se houve vencedor

checkWinner();

// Troca o jogador se o jogo não acabou

if (!gameOver) {

currentPlayer = (currentPlayer == 1) ? 2 : 1;

}

}

}

}

void checkWinner() {

// Verifica linhas

for (int i = 0; i < 3; i++) {

if (board[i][0] != 0 && board[i][0] == board[i][1] && board[i][0] == board[i][2]) {

gameOver = true;

winner = board[i][0];

return;

}

}

// Verifica colunas

for (int j = 0; j < 3; j++) {

if (board[0][j] != 0 && board[0][j] == board[1][j] && board[0][j] == board[2][j]) {

gameOver = true;

winner = board[0][j];

return;

}

}

// Verifica diagonais

if (board[0][0] != 0 && board[0][0] == board[1][1] && board[0][0] == board[2][2]) {

gameOver = true;

winner = board[0][0];

return;

}

if (board[0][2] != 0 && board[0][2] == board[1][1] && board[0][2] == board[2][0]) {

gameOver = true;

winner = board[0][2];

return;

}

// Verifica empate

boolean isTie = true;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == 0) {

isTie = false;

break;

}

}

if (!isTie) break;

}

if (isTie) {

gameOver = true;

winner = 3; // Empate

}

}

void computerMove() {

// Primeiro, verifica se pode vencer na próxima jogada

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == 0) {

board[i][j] = 2; // O é o computador

checkWinner();

if (gameOver && winner == 2) {

return; // Computador vence

} else {

board[i][j] = 0; // Desfaz a jogada

gameOver = false;

winner = 0;

}

}

}

}

// Depois, verifica se precisa bloquear o jogador humano

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == 0) {

board[i][j] = 1; // X é o jogador humano

checkWinner();

if (gameOver && winner == 1) {

board[i][j] = 2; // Bloqueia

gameOver = false;

winner = 0;

currentPlayer = 1; // Passa a vez para o jogador humano

return;

} else {

board[i][j] = 0; // Desfaz a jogada

gameOver = false;

winner = 0;

}

}

}

}

// Se não houver jogadas críticas, faz uma jogada aleatória

ArrayList<PVector> emptyCells = new ArrayList<PVector>();

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == 0) {

emptyCells.add(new PVector(i, j));

}

}

}

if (emptyCells.size() > 0) {

int randomIndex = (int)random(emptyCells.size());

PVector move = emptyCells.get(randomIndex);

board[(int)move.x][(int)move.y] = 2;

checkWinner();

currentPlayer = 1; // Passa a vez para o jogador humano

}

}

void resetGame() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

board[i][j] = 0;

}

}

currentPlayer = 1;

gameOver = false;

winner = 0;

}

3.Jogo da forca

// Variáveis globais

int boardSize = 300; // Tamanho do tabuleiro

int cellSize = boardSize / 3; // Tamanho de cada célula

int[][] board = new int[3][3]; // 0 = vazio, 1 = X, 2 = O

int currentPlayer = 1; // Começa com X

boolean gameOver = false;

int winner = 0; // 0 = sem vencedor, 1 = X, 2 = O, 3 = empate

boolean vsComputer = false; // Modo de jogo (true = vs computador, false = vs jogador)

void setup() {

size(400, 450); // Largura x Altura (incluindo espaço para botões)

resetGame();

// Configuração do texto

textSize(24);

textAlign(CENTER, CENTER);

}

void draw() {

background(255);

// Desenha o tabuleiro

drawBoard();

// Desenha as marcações (X e O)

drawMarks();

// Desenha o status do jogo

drawStatus();

// Desenha os botões

drawButtons();

// Verifica se é a vez do computador no modo vs computador

if (vsComputer && currentPlayer == 2 && !gameOver) {

delay(500); // Pequeno atraso para parecer mais natural

computerMove();

}

}

void drawBoard() {

strokeWeight(3);

line(50, 150, 350, 150); // Linha horizontal superior

line(50, 250, 350, 250); // Linha horizontal inferior

line(150, 50, 150, 350); // Linha vertical esquerda

line(250, 50, 250, 350); // Linha vertical direita

}

void drawMarks() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

float x = 50 + j \* cellSize + cellSize/2;

float y = 50 + i \* cellSize + cellSize/2;

if (board[i][j] == 1) { // X

stroke(255, 0, 0);

line(x - 30, y - 30, x + 30, y + 30);

line(x + 30, y - 30, x - 30, y + 30);

} else if (board[i][j] == 2) { // O

stroke(0, 0, 255);

noFill();

ellipse(x, y, 60, 60);

}

}

}

}

void drawStatus() {

fill(0);

textSize(20);

if (gameOver) {

if (winner == 3) {

text("Empate!", width/2, 20);

} else if (winner == 1) {

text("X venceu!", width/2, 20);

} else if (winner == 2) {

text("O venceu!", width/2, 20);

}

} else {

if (currentPlayer == 1) {

text("Vez do X", width/2, 20);

} else {

text("Vez do O", width/2, 20);

}

}

// Mostra o modo de jogo

textSize(16);

if (vsComputer) {

text("Modo: vs Computador", width/2, 400);

} else {

text("Modo: 2 Jogadores", width/2, 400);

}

}

void drawButtons() {

// Botão de reiniciar

fill(200);

rect(100, 420, 100, 30);

fill(0);

textSize(16);

text("Reiniciar", 150, 435);

// Botão de trocar modo

fill(200);

rect(220, 420, 160, 30);

fill(0);

text("Trocar Modo", 300, 435);

}

void mousePressed() {

// Verifica clique nos botões

if (mouseX >= 100 && mouseX <= 200 && mouseY >= 420 && mouseY <= 450) {

resetGame();

return;

}

if (mouseX >= 220 && mouseX <= 380 && mouseY >= 420 && mouseY <= 450) {

vsComputer = !vsComputer;

resetGame();

return;

}

// Se o jogo acabou, não processa cliques no tabuleiro

if (gameOver) return;

// Verifica se o clique foi dentro do tabuleiro

if (mouseX >= 50 && mouseX <= 350 && mouseY >= 50 && mouseY <= 350) {

int row = (mouseY - 50) / cellSize;

int col = (mouseX - 50) / cellSize;

// Verifica se a célula está vazia

if (board[row][col] == 0) {

board[row][col] = currentPlayer;

// Verifica se houve vencedor

checkWinner();

// Troca o jogador se o jogo não acabou

if (!gameOver) {

currentPlayer = (currentPlayer == 1) ? 2 : 1;

}

}

}

}

void checkWinner() {

// Verifica linhas

for (int i = 0; i < 3; i++) {

if (board[i][0] != 0 && board[i][0] == board[i][1] && board[i][0] == board[i][2]) {

gameOver = true;

winner = board[i][0];

return;

}

}

// Verifica colunas

for (int j = 0; j < 3; j++) {

if (board[0][j] != 0 && board[0][j] == board[1][j] && board[0][j] == board[2][j]) {

gameOver = true;

winner = board[0][j];

return;

}

}

// Verifica diagonais

if (board[0][0] != 0 && board[0][0] == board[1][1] && board[0][0] == board[2][2]) {

gameOver = true;

winner = board[0][0];

return;

}

if (board[0][2] != 0 && board[0][2] == board[1][1] && board[0][2] == board[2][0]) {

gameOver = true;

winner = board[0][2];

return;

}

// Verifica empate

boolean isTie = true;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == 0) {

isTie = false;

break;

}

}

if (!isTie) break;

}

if (isTie) {

gameOver = true;

winner = 3; // Empate

}

}

void computerMove() {

// Primeiro, verifica se pode vencer na próxima jogada

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == 0) {

board[i][j] = 2; // O é o computador

checkWinner();

if (gameOver && winner == 2) {

return; // Computador vence

} else {

board[i][j] = 0; // Desfaz a jogada

gameOver = false;

winner = 0;

}

}

}

}

// Depois, verifica se precisa bloquear o jogador humano

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == 0) {

board[i][j] = 1; // X é o jogador humano

checkWinner();

if (gameOver && winner == 1) {

board[i][j] = 2; // Bloqueia

gameOver = false;

winner = 0;

currentPlayer = 1; // Passa a vez para o jogador humano

return;

} else {

board[i][j] = 0; // Desfaz a jogada

gameOver = false;

winner = 0;

}

}

}

}

// Se não houver jogadas críticas, faz uma jogada aleatória

ArrayList<PVector> emptyCells = new ArrayList<PVector>();

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (board[i][j] == 0) {

emptyCells.add(new PVector(i, j));

}

}

}

if (emptyCells.size() > 0) {

int randomIndex = (int)random(emptyCells.size());

PVector move = emptyCells.get(randomIndex);

board[(int)move.x][(int)move.y] = 2;

checkWinner();

currentPlayer = 1; // Passa a vez para o jogador humano

}

}

void resetGame() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

board[i][j] = 0;

}

}

currentPlayer = 1;

gameOver = false;

winner = 0;

}

4. Jogo do Pong

// Estados do jogo

final int MENU = 0;

final int SELECAO = 1;

final int DIFICULDADE = 2;

final int JOGO = 3;

final int FIM = 4;

int estadoAtual = MENU;

PFont fonte;

// Botões

Botao botaoJogar, botaoSair, botaoVsPlayer, botaoVsCPU, botaoVoltar;

Botao botaoDificil, botaoMedio, botaoFacil; // Botões de dificuldade

boolean modoVsCPU = false;

// Variáveis do jogo

int pontosEsq = 0, pontosDir = 0;

Paddle paddleEsq, paddleDir;

Ball bola;

float dificuldade = 0.07; // Definindo a dificuldade inicial

float tamanhoPaddleEsq = 80; // Tamanho inicial da paleta do jogador 1

float tamanhoPaddleDir = 80; // Tamanho inicial da paleta do jogador 2

float velocidadeBola = 5; // Velocidade inicial da bola

void setup() {

size(800, 500);

fonte = createFont("Arial", 20);

textFont(fonte);

botaoJogar = new Botao("Jogar", width/2 - 75, height/2 - 40, 150, 40);

botaoSair = new Botao("Sair", width/2 - 75, height/2 + 20, 150, 40);

botaoVsPlayer = new Botao("2 Jogadores", width/2 - 85, height/2 - 40, 170, 40);

botaoVsCPU = new Botao("Contra CPU", width/2 - 85, height/2 + 20, 170, 40);

botaoVoltar = new Botao("Voltar", 10, 10, 100, 30);

// Botões de dificuldade corrigidos

botaoDificil = new Botao("Difícil", width/2 - 75, height/2 - 60, 150, 40); // "Fácil" será o mais difícil

botaoMedio = new Botao("Médio", width/2 - 75, height/2, 150, 40);

botaoFacil = new Botao("Fácil", width/2 - 75, height/2 + 60, 150, 40); // "Difícil" será o mais fácil

iniciarJogo();

}

void draw() {

background(30);

switch (estadoAtual) {

case MENU:

mostrarMenu();

break;

case SELECAO:

mostrarSelecao();

break;

case DIFICULDADE:

mostrarDificuldade(); // Tela de seleção de dificuldade

break;

case JOGO:

jogar();

break;

case FIM:

mostrarFim();

break;

}

}

void mousePressed() {

if (estadoAtual == MENU) {

if (botaoJogar.clicado(mouseX, mouseY)) {

estadoAtual = SELECAO;

} else if (botaoSair.clicado(mouseX, mouseY)) {

exit();

}

} else if (estadoAtual == SELECAO) {

if (botaoVsPlayer.clicado(mouseX, mouseY)) {

// Modo "2 Jogadores" usa a dificuldade média automaticamente

modoVsCPU = false;

dificuldade = 0.10; // Configuração de dificuldade média

tamanhoPaddleEsq = 80; // Paleta normal

tamanhoPaddleDir = 80; // Paleta normal

velocidadeBola = 5; // Velocidade normal da bola

iniciarJogo();

estadoAtual = JOGO;

} else if (botaoVsCPU.clicado(mouseX, mouseY)) {

modoVsCPU = true;

estadoAtual = DIFICULDADE; // Vai para a tela de seleção de dificuldade

}

} else if (estadoAtual == DIFICULDADE) {

if (botaoDificil.clicado(mouseX, mouseY)) {

dificuldade = 0.90; // "Fácil" será o modo mais difícil

tamanhoPaddleEsq = 40; // Reduz a paleta pela metade no modo difícil

tamanhoPaddleDir = 80; // A paleta da máquina permanece a mesma

velocidadeBola = 10; // Aumenta a velocidade da bola no modo difícil

iniciarJogo();

estadoAtual = JOGO;

} else if (botaoMedio.clicado(mouseX, mouseY)) {

dificuldade = 0.10;

tamanhoPaddleEsq = 80; // Paleta normal

tamanhoPaddleDir = 80; // Paleta normal

velocidadeBola = 5; // Velocidade normal da bola

iniciarJogo();

estadoAtual = JOGO;

} else if (botaoFacil.clicado(mouseX, mouseY)) {

dificuldade = 0.03; // "Difícil" será o modo mais fácil

tamanhoPaddleEsq = 80; // Paleta normal

tamanhoPaddleDir = 80; // Paleta normal

velocidadeBola = 2; // Velocidade normal da bola

iniciarJogo();

estadoAtual = JOGO;

}

} else if (estadoAtual == JOGO) {

if (botaoVoltar.clicado(mouseX, mouseY)) {

estadoAtual = MENU;

}

} else if (estadoAtual == FIM) {

if (botaoVoltar.clicado(mouseX, mouseY)) {

estadoAtual = MENU;

}

}

}

void keyPressed() {

if (estadoAtual == JOGO) {

if (key == 'w') paddleEsq.mover(-1);

if (key == 's') paddleEsq.mover(1);

if (!modoVsCPU) {

if (keyCode == UP) paddleDir.mover(-1);

if (keyCode == DOWN) paddleDir.mover(1);

}

}

}

void keyReleased() {

if (estadoAtual == JOGO) {

paddleEsq.parar();

paddleDir.parar();

}

}

void mostrarMenu() {

fill(255);

textAlign(CENTER);

textSize(30);

text("PONG", width/2, 120);

botaoJogar.desenhar();

botaoSair.desenhar();

}

void mostrarSelecao() {

fill(255);

textAlign(CENTER);

textSize(25);

text("Escolha o modo de jogo:", width/2, 120);

botaoVsPlayer.desenhar();

botaoVsCPU.desenhar();

}

void mostrarDificuldade() { // Tela de seleção de dificuldade

fill(255);

textAlign(CENTER);

textSize(25);

text("Escolha a dificuldade:", width/2, 120);

botaoDificil.desenhar(); // "Fácil" é o mais difícil

botaoMedio.desenhar(); // Média dificuldade

botaoFacil.desenhar(); // "Difícil" é o mais fácil

}

void mostrarFim() {

fill(255);

textAlign(CENTER);

textSize(28);

String vencedor = pontosEsq == 7 ? "Jogador 1 venceu!" : "Jogador 2 venceu!";

if (modoVsCPU && pontosDir == 7) vencedor = "CPU venceu!";

text(vencedor, width/2, height/2 - 20);

botaoVoltar.desenhar();

}

void iniciarJogo() {

pontosEsq = 0;

pontosDir = 0;

paddleEsq = new Paddle(30, tamanhoPaddleEsq);

paddleDir = new Paddle(width - 40, tamanhoPaddleDir);

bola = new Ball();

}

void jogar() {

paddleEsq.atualizar();

paddleDir.atualizar();

bola.atualizar();

if (modoVsCPU) {

paddleDir.seguirBola(bola);

}

paddleEsq.desenhar();

paddleDir.desenhar();

bola.desenhar();

fill(255);

textSize(20);

textAlign(CENTER);

text(pontosEsq + " : " + pontosDir, width/2, 40);

botaoVoltar.desenhar();

if (pontosEsq >= 7 || pontosDir >= 7) {

estadoAtual = FIM;

}

}

class Botao {

String texto;

float x, y, w, h;

Botao(String texto, float x, float y, float w, float h) {

this.texto = texto;

this.x = x;

this.y = y;

this.w = w;

this.h = h;

}

void desenhar() {

fill(70);

stroke(255);

rect(x, y, w, h, 8);

fill(255);

textAlign(CENTER, CENTER);

textSize(16);

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

}

boolean clicado(float mx, float my) {

return mx > x && mx < x + w && my > y && my < y + h;

}

}

class Paddle {

float x, y, w, h;

float velocidade = 5;

float dy = 0;

Paddle(float x, float h) {

this.x = x;

this.h = h;

this.w = 15;

this.y = height/2 - h/2;

}

void atualizar() {

y += dy;

y = constrain(y, 0, height - h);

}

void desenhar() {

fill(255);

rect(x, y, w, h);

}

void mover(int dir) {

dy = dir \* velocidade;

}

void parar() {

dy = 0;

}

void seguirBola(Ball b) {

float centro = y + h/2;

float erro = b.y - centro;

dy = erro \* dificuldade;

float maxVelocidade = 4;

dy = constrain(dy, -maxVelocidade, maxVelocidade);

}

}

class Ball {

float x, y, r = 12;

float vx, vy;

float velocidade = 5;

Ball() {

resetar();

}

void resetar() {

x = width/2;

y = height/2;

vx = random(1) > 0.5 ? velocidade : -velocidade;

vy = random(-3, 3);

}

void atualizar() {

x += vx;

y += vy;

if (y < 0 || y > height) vy \*= -1;

if (colidiu(paddleEsq) || colidiu(paddleDir)) {

vx \*= -1.05;

vy = random(-4, 4);

}

if (x < 0) {

pontosDir++;

resetar();

}

if (x > width) {

pontosEsq++;

resetar();

}

}

void desenhar() {

fill(255);

ellipse(x, y, r\*2, r\*2);

}

boolean colidiu(Paddle p) {

return x - r < p.x + p.w && x + r > p.x && y > p.y && y < p.y + p.h;

}

}

5. Jogo da Memória

// === Jogo da Memória com Animais ===

final int TOTAL\_PARES = 9;

final int TOTAL\_CARTAS = TOTAL\_PARES \* 2;

final int COLUNAS = 6;

final int LINHAS = 3;

final int LARGURA\_CARTA = 100;

final int ALTURA\_CARTA = 100;

PImage[][] temas = new PImage[3][TOTAL\_PARES]; // 0: gatos, 1: cachorros, 2: passaros

String[] nomesTemas = {"Gatos", "Cachorros", "Passaros"};

int temaSelecionado = -1;

int[] embaralhamento = new int[TOTAL\_CARTAS];

boolean[] cartaVirada = new boolean[TOTAL\_CARTAS];

boolean[] cartaFixa = new boolean[TOTAL\_CARTAS];

int primeiraCarta = -1;

int segundaCarta = -1;

boolean aguardando = false;

int tempoVirada;

PFont fonte;

String estado = "inicio";

int tempoInicio;

int tempoLimite = 60;

void setup() {

size(800, 600);

fonte = createFont("Arial", 20, true);

for (int t = 0; t < 3; t++) {

for (int i = 0; i < TOTAL\_PARES; i++) {

String nomeBase = "";

if (t == 0) nomeBase = "gato";

if (t == 1) nomeBase = "cachorro";

if (t == 2) nomeBase = "passaro";

temas[t][i] = loadImage(nomeBase + i + ".jpg");

temas[t][i].resize(LARGURA\_CARTA, ALTURA\_CARTA);

}

}

embaralharCartas();

}

void draw() {

background(200);

if (estado.equals("inicio")) {

telaInicio();

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

telaJogo();

mostrarTempo();

verificarVitoria();

} else if (estado.equals("fim")) {

telaFinal();

}

if (aguardando) {

if (embaralhamento[primeiraCarta] == embaralhamento[segundaCarta]) {

cartaFixa[primeiraCarta] = true;

cartaFixa[segundaCarta] = true;

primeiraCarta = -1;

segundaCarta = -1;

aguardando = false;

} else if (millis() - tempoVirada > 1000) {

cartaVirada[primeiraCarta] = false;

cartaVirada[segundaCarta] = false;

primeiraCarta = -1;

segundaCarta = -1;

aguardando = false;

}

}

}

void telaInicio() {

textFont(fonte);

textAlign(CENTER);

fill(0);

textSize(32);

text("Jogo da Memória", width / 2, 80);

textSize(20);

text("Escolha um tema:", width / 2, 140);

for (int i = 0; i < nomesTemas.length; i++) {

int x = width / 2 - 110;

int y = 180 + i \* 70;

fill(temaSelecionado == i ? color(100, 200, 100) : 0);

rect(x, y, 220, 40);

fill(255);

text(nomesTemas[i], width / 2, y + 27);

}

if (temaSelecionado != -1) {

fill(0);

rect(width / 2 - 100, 450, 200, 40);

fill(255);

text("Iniciar Jogo", width / 2, 477);

}

fill(255, 0, 0);

rect(width - 110, 20, 90, 30);

fill(255);

textSize(16);

textAlign(CENTER, CENTER);

text("Sair", width - 65, 35);

}

void telaJogo() {

for (int i = 0; i < TOTAL\_CARTAS; i++) {

int x = (i % COLUNAS) \* (LARGURA\_CARTA + 10) + 50;

int y = (i / COLUNAS) \* (ALTURA\_CARTA + 10) + 100;

if (cartaFixa[i] || cartaVirada[i]) {

image(temas[temaSelecionado][embaralhamento[i]], x, y);

} else {

fill(100);

rect(x, y, LARGURA\_CARTA, ALTURA\_CARTA);

}

}

fill(255, 0, 0);

rect(width - 110, 20, 90, 30);

fill(255);

textSize(16);

textAlign(CENTER, CENTER);

text("Sair", width - 65, 35);

}

void telaFinal() {

background(estadoVitoria() ? color(0, 200, 0) : color(200, 0, 0));

fill(255);

textFont(fonte);

textSize(28);

textAlign(CENTER, CENTER);

text(estadoVitoria() ? "Você Ganhou!" : "Tempo Esgotado!", width / 2, height / 2 - 100);

textSize(20);

fill(0);

rect(width / 2 - 120, height / 2 - 20, 240, 40);

fill(255);

text("Jogar Novamente", width / 2, height / 2);

fill(0);

rect(width / 2 - 120, height / 2 + 40, 240, 40);

fill(255);

text("Voltar ao Início", width / 2, height / 2 + 60);

fill(0);

rect(width / 2 - 120, height / 2 + 100, 240, 40);

fill(255);

text("Sair", width / 2, height / 2 + 120);

}

void mousePressed() {

if (estado.equals("inicio")) {

for (int i = 0; i < nomesTemas.length; i++) {

int y = 180 + i \* 70;

if (mouseX > width / 2 - 110 && mouseX < width / 2 + 110 && mouseY > y && mouseY < y + 40) {

temaSelecionado = i;

}

}

if (temaSelecionado != -1 && mouseX > width / 2 - 100 && mouseX < width / 2 + 100 && mouseY > 450 && mouseY < 490) {

reiniciarJogo(); // <- IMPORTANTE!

estado = "jogando";

}

if (mouseX > width - 110 && mouseX < width - 20 && mouseY > 20 && mouseY < 50) {

exit();

}

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

if (mouseX > width - 110 && mouseX < width - 20 && mouseY > 20 && mouseY < 50) {

exit();

}

if (!aguardando) {

for (int i = 0; i < TOTAL\_CARTAS; i++) {

int x = (i % COLUNAS) \* (LARGURA\_CARTA + 10) + 50;

int y = (i / COLUNAS) \* (ALTURA\_CARTA + 10) + 100;

if (!cartaFixa[i] && !cartaVirada[i] && mouseX > x && mouseX < x + LARGURA\_CARTA && mouseY > y && mouseY < y + ALTURA\_CARTA) {

cartaVirada[i] = true;

if (primeiraCarta == -1) {

primeiraCarta = i;

} else if (segundaCarta == -1 && i != primeiraCarta) {

segundaCarta = i;

aguardando = true;

tempoVirada = millis();

}

break;

}

}

}

} else if (estado.equals("fim")) {

if (mouseX > width / 2 - 120 && mouseX < width / 2 + 120) {

if (mouseY > height / 2 - 20 && mouseY < height / 2 + 20) {

reiniciarJogo();

estado = "jogando";

} else if (mouseY > height / 2 + 40 && mouseY < height / 2 + 80) {

estado = "inicio";

temaSelecionado = -1;

} else if (mouseY > height / 2 + 100 && mouseY < height / 2 + 140) {

exit();

}

}

}

}

void mostrarTempo() {

int tempoRestante = tempoLimite - (millis() - tempoInicio) / 1000;

fill(0);

textSize(20);

textAlign(LEFT);

text("Tempo: " + tempoRestante + "s", 20, 30);

if (tempoRestante <= 0) estado = "fim";

}

void verificarVitoria() {

for (boolean fixa : cartaFixa) {

if (!fixa) return;

}

estado = "fim";

}

boolean estadoVitoria() {

for (boolean fixa : cartaFixa) {

if (!fixa) return false;

}

return true;

}

void embaralharCartas() {

int[] pares = new int[TOTAL\_CARTAS];

for (int i = 0; i < TOTAL\_PARES; i++) {

pares[i \* 2] = i;

pares[i \* 2 + 1] = i;

}

for (int i = 0; i < TOTAL\_CARTAS; i++) {

int r = int(random(i, TOTAL\_CARTAS));

int temp = pares[i];

pares[i] = pares[r];

pares[r] = temp;

}

for (int i = 0; i < TOTAL\_CARTAS; i++) {

embaralhamento[i] = pares[i];

cartaVirada[i] = false;

cartaFixa[i] = false;

}

}

void reiniciarJogo() {

embaralharCartas();

primeiraCarta = -1;

segundaCarta = -1;

aguardando = false;

tempoInicio = millis();

}