Desenvolvimento   
de   
Games

|  |  |
| --- | --- |
| **DADOS DO PROJETO** | |
| **INSTITUIÇÃO DE ENSINO SUPERIOR:** | Centro Universitário Maurício de Nassau – Uninassau Graças |
| **DISCIPLINA:** | Desenvolvimento de Games |
| **CURSO(S) PROPONENTE(S):** | Ciência da Computação |
| **ÁREA TEMÁTICA:** | Desenvolvimento de Games - Equipe: B.Sabbath Studios |
| **DISCENTES RESPONSÁVEIS:**  (nome e matrícula) | Elizeu Leoncio Ferreira Junior - Matricula 01576238 Laryssa Rayanne Souza Martins - Matrícula 01612424 Matheus de Oliveira Lins Mendes Simes- Matrícula 01618966 Rafael Ferreira dos anjos - Matrícula 01579531 |
| **QUANTIDADES DE ALUNOS NO PROJETO** | 4 |

|  |
| --- |
| 1. **Jogo Marciano:** |

|  |
| --- |
| 1. **A) Jogo da velha B) Jogo da velha alternativo:** (Fizemos dois jogos, pode considerar o alternativo)   **Jogo da velha:**  **Jogo da velha alternativo:** |

|  |
| --- |
| 1. **Jogo da Forca:** 2. **Jogo do Pong:**   Interface gráfica do usuário, Aplicativo  O conteúdo gerado por IA pode estar incorreto.  Interface gráfica do usuário, PowerPoint  O conteúdo gerado por IA pode estar incorreto.  **Jogo do Pong (alternativo):**  Interface gráfica do usuário  O conteúdo gerado por IA pode estar incorreto.  Interface gráfica do usuário, Aplicativo, PowerPoint  O conteúdo gerado por IA pode estar incorreto.  **Jogo da Memória:**  Interface gráfica do usuário, Aplicativo  O conteúdo gerado por IA pode estar incorreto.  Interface gráfica do usuário, Tabela  O conteúdo gerado por IA pode estar incorreto.  **ANEXOS**  **Jogo Marciano - CÓDIGO FONTE:**  import java.util.ArrayList;  import java.util.Collections;  import java.util.Scanner;  public class Main  {  private static final int MAX\_TENTATIVAS = 5;  private static final ArrayList<Integer> recordes = new ArrayList<>();  private static int marcianoPosicao;    public static void main(String[] args) {    Scanner scanner = new Scanner(System.in);  mostrarIntroducao();  while (true) {  jogar(scanner);  System.out.print("Deseja jogar novamente? (s/n): ");  String resposta = scanner.next();  if (!resposta.equalsIgnoreCase("s")) {  break;  }  }  mostrarRecordes();  scanner.close();  }  private static void mostrarIntroducao() {  System.out.println("Bem-vindo ao Jogo do Marciano!");  System.out.println("Você deve adivinhar a posição do marciano em um tabuleiro de 1 a 10.");  System.out.println("Você tem " + MAX\_TENTATIVAS + " tentativas para acertar.");  System.out.println("Boa sorte!");  }  private static void jogar(Scanner scanner) {  marcianoPosicao = (int) (Math.random() \* 10) + 1;  int tentativas = 0;  boolean acertou = false;  while (tentativas < MAX\_TENTATIVAS) {  System.out.print("Tentativa " + (tentativas + 1) + ": Adivinhe a posição do marciano (1-10): ");  int palpite = scanner.nextInt();  tentativas++;  if (palpite == marcianoPosicao) {  System.out.println("Parabéns! Você acertou em " + tentativas + " tentativas.");  recordes.add(tentativas);  Collections.sort(recordes);  acertou = true;  break;  } else if (palpite < marcianoPosicao) {  System.out.println("O marciano está em uma posição maior.");  } else {  System.out.println("O marciano está em uma posição menor.");  }  }  if (!acertou) {  System.out.println("Você não conseguiu adivinhar. O marciano estava na posição " + marcianoPosicao + ".");  }  }  private static void mostrarRecordes() {  System.out.println("Melhores Jogadas:");  for (int i = 0; i < recordes.size(); i++) {  System.out.println((i + 1) + "º lugar: " + recordes.get(i) + " tentativas");  }    }  }  **Jogo da velha - CÓDIGO FONTE:**  int screen = 0; // 0 = Menu, 1 = Jogo PvP, 2 = Jogo PvE, 3 = Tela de Escolha de Rounds  char[][] board = new char[3][3];  boolean playerTurn = true;  boolean gameOver = false;  char winner = ' ';  int playerXWins = 0;  int playerOWins = 0;  int totalRounds = 3;  int currentRound = 1;  boolean roundSelection = false;  boolean isPvP = true; // Variável para determinar se é PvP ou PvE  void setup() {  size(400, 400);  resetBoard();  }  void draw() {  background(255);  switch(screen) {  case 0: drawMenu(); break; // Tela do menu  case 3: drawRoundSelection(); break; // Tela de seleção de rounds  case 1: case 2: drawGame(); break; // Jogo PvP ou PvE  }  }  void drawMenu() {  textSize(32);  textAlign(CENTER, CENTER);  fill(0);  text("Jogo da Velha", width / 2, 50);  textSize(24);  text("1. Jogar contra Player", width / 2, 150);  text("2. Jogar contra Máquina", width / 2, 200);  }  void drawRoundSelection() {  textSize(24);  textAlign(CENTER, CENTER);  fill(0);  text("Escolha o número de rounds:", width / 2, 100);  text("1. 3 Rounds", width / 2, 150);  text("2. 5 Rounds", width / 2, 200);  text("3. 9 Rounds", width / 2, 250);  }  void drawGame() {  drawBoard();  drawScoreboard(); // Exibe o placar  if (gameOver) drawWinner();  }  void drawBoard() {  stroke(0);  for (int i = 1; i <= 2; i++) {  line(i \* width / 3, 0, i \* width / 3, height);  line(0, i \* height / 3, width, i \* height / 3);  }  textSize(32);  textAlign(CENTER, CENTER);  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j] != ' ') {  text(board[i][j], j \* width / 3 + width / 6, i \* height / 3 + height / 6);  }  }  }  textSize(16);  text("Rodada: " + currentRound + "/" + totalRounds, width / 6, height - 10);  }  void drawScoreboard() {  textSize(16);  fill(0);  text("Placar - X: " + playerXWins + " | O: " + playerOWins, width \* 5 / 6, height - 10);  }  void drawWinner() {  textSize(24);  textAlign(CENTER, CENTER);  fill(0);  if (currentRound < totalRounds) {  text("Vencedor: " + (winner == 'D' ? "Empate" : winner), width / 2, height - 70);  text("Clique para continuar", width / 2, height - 240);  } else {  text("Campeão Final: " + (winner == 'D' ? "Empate!" : winner), width / 2, height - 70);  text("Placar final - X: " + playerXWins + " | O: " + playerOWins, width / 2, height - 40);  text("Clique para ir ao Menu", width / 2, height - 240);  }  }  void mousePressed() {  if (screen == 0) handleMenuSelection();  else if (screen == 3) handleRoundSelection();  else if (gameOver) nextRound();  else handlePlayerMove();  }  void handleMenuSelection() {  if (mouseY > 130 && mouseY < 170) {  screen = 3; // Vai para a seleção de rounds  isPvP = true; // Jogo contra Player  } else if (mouseY > 180 && mouseY < 220) {  screen = 3; // Vai para a seleção de rounds  isPvP = false; // Jogo contra Máquina  }  }  void handleRoundSelection() {  if (mouseY > 130 && mouseY < 170) totalRounds = 3;  else if (mouseY > 180 && mouseY < 220) totalRounds = 5;  else if (mouseY > 230 && mouseY < 270) totalRounds = 9;  roundSelection = false;  resetBoard();  screen = 1; // Inicia o Jogo PvP ou PvE dependendo de isPvP  }  void handlePlayerMove() {  int col = mouseX / (width / 3);  int row = mouseY / (height / 3);  if (row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == ' ') {  board[row][col] = playerTurn ? 'X' : 'O';  playerTurn = !playerTurn;  checkWinner();  if (!isPvP && !playerTurn && !gameOver) {  aiMove(); // Faz o movimento do bot se for PvE  playerTurn = true;  checkWinner();  }  }  }  void aiMove() {  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j] == ' ') {  board[i][j] = 'O';  if (checkWinningMove('O')) return;  board[i][j] = ' '; // Desfaz a jogada  board[i][j] = 'X';  if (checkWinningMove('X')) {  board[i][j] = 'O';  return;  }  board[i][j] = ' '; // Desfaz a jogada  }  }  }  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j] == ' ') {  board[i][j] = 'O';  return;  }  }  }  }  boolean checkWinningMove(char player) {  for (int i = 0; i < 3; i++) {  if (board[i][0] == player && board[i][1] == player && board[i][2] == player) return true;  if (board[0][i] == player && board[1][i] == player && board[2][i] == player) return true;  }  if (board[0][0] == player && board[1][1] == player && board[2][2] == player) return true;  if (board[0][2] == player && board[1][1] == player && board[2][0] == player) return true;  return false;  }  void resetBoard() {  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  board[i][j] = ' ';  }  }  playerTurn = true;  gameOver = false;  winner = ' ';  }  void nextRound() {  if (currentRound < totalRounds) {  currentRound++;  resetBoard();  } else {  announceFinalWinner();  resetGame();  }  }  void resetGame() {  // Resetar a pontuação dos jogadores e variáveis relacionadas ao jogo  playerXWins = 0;  playerOWins = 0;  currentRound = 1;  totalRounds = 3; // Ou qualquer número que você definir como padrão  screen = 0; // Vai para o menu  roundSelection = false;  resetBoard(); // Reseta o tabuleiro  }  void declareWinner(char w) {  winner = w;  gameOver = true;  if (winner == 'X') playerXWins++;  if (winner == 'O') playerOWins++;  }  void announceFinalWinner() {  // Definir o vencedor final com base nas pontuações  if (playerXWins > playerOWins) {  winner = 'X';  } else if (playerOWins > playerXWins) {  winner = 'O';  } else {  winner = 'D'; // Empate  }  }  void checkWinner() {  for (int i = 0; i < 3; i++) {  if (board[i][0] != ' ' && board[i][0] == board[i][1] && board[i][1] == board[i][2]) {  declareWinner(board[i][0]);  return;  }  if (board[0][i] != ' ' && board[0][i] == board[1][i] && board[1][i] == board[2][i]) {  declareWinner(board[0][i]);  return;  }  }  if (board[0][0] != ' ' && board[0][0] == board[1][1] && board[1][1] == board[2][2]) {  declareWinner(board[0][0]);  return;  }  if (board[0][2] != ' ' && board[0][2] == board[1][1] && board[1][1] == board[2][0]) {  declareWinner(board[0][2]);  return;  }  boolean draw = true;  for (char[] row : board) {  for (char cell : row) {  if (cell == ' ') {  draw = false;  break;  }  }  if (!draw) break;  }  if (draw) {  winner = 'D';  gameOver = true;  }  }  **Jogo da velha alternativo - CÓDIGO FONTE:**  // 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 mensagens e botões)  resetGame();  }  void draw() {  background(255);  drawBoard();  drawStatus();  drawButtons();    if (gameOver) {  drawGameOver();  }  }  void drawBoard() {  // Desenha as linhas do tabuleiro  strokeWeight(4);  line(cellSize, 0, cellSize, boardSize);  line(cellSize \* 2, 0, cellSize \* 2, boardSize);  line(0, cellSize, boardSize, cellSize);  line(0, cellSize \* 2, boardSize, cellSize \* 2);    // Desenha os X e O nas células  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j] == 1) { // X  drawX(i, j);  } else if (board[i][j] == 2) { // O  drawO(i, j);  }  }  }  }  void drawX(int i, int j) {  stroke(255, 0, 0); // Vermelho para X  line(i \* cellSize + 20, j \* cellSize + 20, (i + 1) \* cellSize - 20, (j + 1) \* cellSize - 20);  line((i + 1) \* cellSize - 20, j \* cellSize + 20, i \* cellSize + 20, (j + 1) \* cellSize - 20);  }  void drawO(int i, int j) {  stroke(0, 0, 255); // Azul para O  noFill();  ellipse(i \* cellSize + cellSize/2, j \* cellSize + cellSize/2, cellSize - 40, cellSize - 40);  }  void drawStatus() {  fill(0);  textSize(20);    if (!gameOver) {  if (vsComputer && currentPlayer == 2) {  text("Vez do Computador (O)", 10, boardSize + 30);  } else {  text("Vez do Jogador " + (currentPlayer == 1 ? "X" : "O"), 10, boardSize + 30);  }  }  }  void drawButtons() {  // Botão para reiniciar o jogo  fill(200);  rect(10, boardSize + 60, 180, 40, 5);  fill(0);  text("Reiniciar Jogo", 40, boardSize + 85);    // Botão para alternar modo de jogo  fill(200);  rect(210, boardSize + 60, 180, 40, 5);  fill(0);  text(vsComputer ? "Jogar vs Humano" : "Jogar vs Computador", 220, boardSize + 85);  }  void drawGameOver() {  fill(0, 150);  rect(50, 100, 200, 100, 10);  fill(255);  textSize(24);    if (winner == 3) {  text("Empate!", 120, 150);  } else {  text("Jogador " + (winner == 1 ? "X" : "O") + " venceu!", 80, 150);  }    textSize(16);  text("Clique para continuar", 90, 180);  }  void mousePressed() {  if (gameOver) {  resetGame();  return;  }    // Verifica clique nos botões  if (mouseY > boardSize + 60 && mouseY < boardSize + 100) {  if (mouseX > 10 && mouseX < 190) {  resetGame(); // Botão Reiniciar  } else if (mouseX > 210 && mouseX < 390) {  vsComputer = !vsComputer; // Botão Alternar Modo  resetGame();  }  return;  }    // Verifica clique no tabuleiro (só se for a vez do jogador humano)  if (!gameOver && mouseX < boardSize && mouseY < boardSize &&  (!vsComputer || (vsComputer && currentPlayer == 1))) {  int i = mouseX / cellSize;  int j = mouseY / cellSize;    if (board[i][j] == 0) { // Célula vazia  board[i][j] = currentPlayer;  checkGameOver();    if (!gameOver && vsComputer) {  currentPlayer = 2;  // Adiciona um pequeno atraso para a jogada do computador  delay(500);  computerMove();  checkGameOver();  }  }  }  }  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  if (checkWinner() == 2) {  return; // Computador vence  }  board[i][j] = 0; // Desfaz  }  }  }    // Depois verifica se precisa bloquear o jogador  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j] == 0) {  board[i][j] = 1; // X  if (checkWinner() == 1) {  board[i][j] = 2; // Bloqueia  return;  }  board[i][j] = 0; // Desfaz  }  }  }    // Tenta jogar no centro se estiver vazio  if (board[1][1] == 0) {  board[1][1] = 2;  return;  }    // Joga em um canto vazio aleatório  int[] corners = {0, 2};  for (int i : corners) {  for (int j : corners) {  if (board[i][j] == 0) {  if (random(1) > 0.5) { // Aleatoriedade para não ser previsível  board[i][j] = 2;  return;  }  }  }  }    // Joga em qualquer posição vazia  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j] == 0) {  board[i][j] = 2;  return;  }  }  }  }  void checkGameOver() {  winner = checkWinner();    if (winner != 0) {  gameOver = true;  } else if (isBoardFull()) {  gameOver = true;  winner = 3; // Empate  } else {  currentPlayer = currentPlayer == 1 ? 2 : 1; // Alterna jogador  }  }  int checkWinner() {  // Verifica linhas  for (int j = 0; j < 3; j++) {  if (board[0][j] != 0 && board[0][j] == board[1][j] && board[1][j] == board[2][j]) {  return board[0][j];  }  }    // Verifica colunas  for (int i = 0; i < 3; i++) {  if (board[i][0] != 0 && board[i][0] == board[i][1] && board[i][1] == board[i][2]) {  return board[i][0];  }  }    // Verifica diagonais  if (board[0][0] != 0 && board[0][0] == board[1][1] && board[1][1] == board[2][2]) {  return board[0][0];  }  if (board[2][0] != 0 && board[2][0] == board[1][1] && board[1][1] == board[0][2]) {  return board[2][0];  }    return 0; // Sem vencedor  }  boolean isBoardFull() {  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 3; j++) {  if (board[i][j] == 0) {  return false;  }  }  }  return true;  }  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;  }  **Jogo da Forca - CÓDIGO FONTE:**  // Jogo da Forca em Processing  // Autor: [Seu Nome]  // Data: [Data]  // Variáveis do jogo  String[] categorias = {"Frutas", "Cidades", "Animais", "Países"};  String[][] palavras = {  {"BANANA", "MORANGO", "ABACAXI", "LARANJA", "UVA", "MELANCIA", "KIWI"},  {"SÃO PAULO", "RIO DE JANEIRO", "BELO HORIZONTE", "SALVADOR", "CURITIBA", "PORTO ALEGRE"},  {"ELEFANTE", "GIRAFA", "TIGRE", "LEÃO", "ZEBRA", "RINOCERONTE"},  {"BRASIL", "ARGENTINA", "CANADÁ", "JAPÃO", "AUSTRÁLIA", "ALEMANHA"}  };  int categoriaSelecionada = 0;  String palavraSecreta = "";  char[] letrasDescobertas;  ArrayList<Character> letrasErradas = new ArrayList<Character>();  ArrayList<Character> letrasTentadas = new ArrayList<Character>();  int tentativasRestantes = 6;  boolean jogoAtivo = false;  boolean vitoria = false;  boolean derrota = false;  // Configuração inicial  void setup() {  size(800, 600);  textAlign(CENTER, CENTER);  textSize(24);  selecionarNovaPalavra(0); // Começa com a primeira categoria  }  // Loop principal  void draw() {  background(240);    if (!jogoAtivo) {  telaInicial();  } else if (vitoria) {  telaVitoria();  } else if (derrota) {  telaDerrota();  } else {  desenharForca();  desenharPalavra();  desenharLetrasErradas();  verificarFimDoJogo();  }  }  // Tela inicial para selecionar categoria  void telaInicial() {  fill(0);  textSize(32);  text("JOGO DA FORCA", width/2, 80);  textSize(24);  text("Selecione uma categoria:", width/2, 150);    // Desenhar botões de categoria  for (int i = 0; i < categorias.length; i++) {  fill(200);  rect(width/2 - 150, 200 + i\*70, 300, 50, 10);  fill(0);  text(categorias[i], width/2, 225 + i\*70);  }  }  // Tela de vitória  void telaVitoria() {  desenharForca();  desenharPalavra();    fill(0, 180, 0, 200);  rect(width/4, height/3, width/2, height/3, 20);  fill(255);  textSize(32);  text("PARABÉNS!", width/2, height/2 - 30);  textSize(24);  text("Você acertou a palavra:", width/2, height/2 + 10);  text(palavraSecreta, width/2, height/2 + 50);    // Botão para jogar novamente  fill(50, 150, 255);  rect(width/2 - 100, height/2 + 100, 200, 50, 10);  fill(255);  text("Jogar Novamente", width/2, height/2 + 125);  }  // Tela de derrota  void telaDerrota() {  desenharForca();  desenharPalavra();    fill(180, 0, 0, 200);  rect(width/4, height/3, width/2, height/3, 20);  fill(255);  textSize(32);  text("FIM DE JOGO", width/2, height/2 - 30);  textSize(24);  text("A palavra era:", width/2, height/2 + 10);  text(palavraSecreta, width/2, height/2 + 50);    // Botão para jogar novamente  fill(50, 150, 255);  rect(width/2 - 100, height/2 + 100, 200, 50, 10);  fill(255);  text("Jogar Novamente", width/2, height/2 + 125);  }  // Desenha a forca e o boneco  void desenharForca() {  stroke(0);  strokeWeight(4);    // Base  line(100, 500, 300, 500);  // Poste vertical  line(150, 500, 150, 100);  // Topo  line(150, 100, 300, 100);  // Corda  line(300, 100, 300, 150);    // Desenhar o boneco baseado nas tentativas restantes  if (tentativasRestantes < 6) { // Cabeça  ellipse(300, 180, 60, 60);  }  if (tentativasRestantes < 5) { // Corpo  line(300, 210, 300, 350);  }  if (tentativasRestantes < 4) { // Braço esquerdo  line(300, 250, 250, 300);  }  if (tentativasRestantes < 3) { // Braço direito  line(300, 250, 350, 300);  }  if (tentativasRestantes < 2) { // Perna esquerda  line(300, 350, 250, 420);  }  if (tentativasRestantes < 1) { // Perna direita  line(300, 350, 350, 420);  }    // Mostrar tentativas restantes  fill(0);  textSize(20);  text("Tentativas restantes: " + tentativasRestantes, 650, 100);  }  // Desenha a palavra com traços e letras descobertas  void desenharPalavra() {  textSize(36);  fill(0);    for (int i = 0; i < letrasDescobertas.length; i++) {  float x = 400 + i \* 40;  float y = 400;    // Desenhar traço  line(x - 15, y + 10, x + 15, y + 10);    // Desenhar letra se descoberta  if (letrasDescobertas[i] != '\_') {  text(letrasDescobertas[i], x, y);  }  }  }  // Desenha as letras erradas  void desenharLetrasErradas() {  textSize(20);  fill(180, 0, 0);  text("Letras erradas:", 650, 150);    for (int i = 0; i < letrasErradas.size(); i++) {  text(letrasErradas.get(i), 650 + i \* 20, 180);  }  }  // Verifica se o jogo terminou (vitória ou derrota)  void verificarFimDoJogo() {  // Verificar vitória  boolean completa = true;  for (char c : letrasDescobertas) {  if (c == '\_') {  completa = false;  break;  }  }    if (completa) {  vitoria = true;  jogoAtivo = false;  return;  }    // Verificar derrota  if (tentativasRestantes <= 0) {  derrota = true;  jogoAtivo = false;  // Revelar palavra  for (int i = 0; i < palavraSecreta.length(); i++) {  letrasDescobertas[i] = palavraSecreta.charAt(i);  }  }  }  // Seleciona uma nova palavra da categoria especificada  void selecionarNovaPalavra(int categoria) {  categoriaSelecionada = categoria;  int indice = (int)random(palavras[categoria].length);  palavraSecreta = palavras[categoria][indice];    // Inicializar array de letras descobertas  letrasDescobertas = new char[palavraSecreta.length()];  for (int i = 0; i < letrasDescobertas.length; i++) {  if (palavraSecreta.charAt(i) == ' ') {  letrasDescobertas[i] = ' ';  } else {  letrasDescobertas[i] = '\_';  }  }    // Reiniciar variáveis do jogo  letrasErradas.clear();  letrasTentadas.clear();  tentativasRestantes = 6;  jogoAtivo = true;  vitoria = false;  derrota = false;  }  // Reinicia o jogo com a mesma categoria  void reiniciarJogo() {  selecionarNovaPalavra(categoriaSelecionada);  }  // Trata pressionamento de teclas  void keyPressed() {  if (jogoAtivo && !vitoria && !derrota) {  if (key >= 'A' && key <= 'Z') {  char letra = key;  processarLetra(letra);  } else if (key >= 'a' && key <= 'z') {  char letra = Character.toUpperCase(key);  processarLetra(letra);  }  }  }  // Processa uma letra tentada pelo jogador  void processarLetra(char letra) {  // Verificar se a letra já foi tentada  if (letrasTentadas.contains(letra)) {  return;  }    letrasTentadas.add(letra);  boolean acertou = false;    // Verificar se a letra está na palavra  for (int i = 0; i < palavraSecreta.length(); i++) {  if (palavraSecreta.charAt(i) == letra) {  letrasDescobertas[i] = letra;  acertou = true;  }  }    if (!acertou) {  letrasErradas.add(letra);  tentativasRestantes--;  }  }  // Trata cliques do mouse  void mousePressed() {  if (!jogoAtivo) {  // Verificar cliques na tela inicial (seleção de categoria)  for (int i = 0; i < categorias.length; i++) {  if (mouseX > width/2 - 150 && mouseX < width/2 + 150 &&  mouseY > 200 + i\*70 && mouseY < 250 + i\*70) {  selecionarNovaPalavra(i);  return;  }  }  } else if (vitoria || derrota) {  // Verificar clique no botão "Jogar Novamente"  if (mouseX > width/2 - 100 && mouseX < width/2 + 100 &&  mouseY > height/2 + 100 && mouseY < height/2 + 150) {  reiniciarJogo();  }  }  }  **Jogo do Pong - CÓDIGO FONTE:**  // Variáveis do jogo  int larguraTela = 800;  int alturaTela = 600;  int meioTelaX = larguraTela / 2;  int meioTelaY = alturaTela / 2;  // Variáveis das paletas  int larguraPaleta = 15;  int alturaPaletaFacil = 150;  int alturaPaletaMedio = 100;  int alturaPaletaDificil = 60;  int alturaPaletaJogador = alturaPaletaFacil; // Começa no fácil  int alturaPaletaCPU = alturaPaletaFacil;  int posicaoPaletaJogador = alturaTela / 2;  int posicaoPaletaCPU = alturaTela / 2;  int velocidadePaletaJogador = 8;  int velocidadePaletaCPU = 5;  // Variáveis da bola  int tamanhoBola = 20;  float posicaoBolaX = meioTelaX;  float posicaoBolaY = meioTelaY;  float velocidadeBolaX = 5;  float velocidadeBolaY = 5;  // Placar  int placarJogador = 0;  int placarCPU = 0;  int placarMaximo = 5; // Limite para vencer  boolean jogoTerminou = false;  String vencedor = "";  // Estados do jogo  final int TELA\_INICIO = 0;  final int JOGANDO = 1;  final int TELA\_VENCEDOR = 2;  int estadoJogo = TELA\_INICIO;  // Dificuldade  final int FACIL = 0;  final int MEDIO = 1;  final int DIFICIL = 2;  int dificuldade = FACIL;  void setup() {  size(800, 600);  noStroke();  rectMode(CENTER);  ellipseMode(CENTER);  textAlign(CENTER, CENTER);  textSize(32);  }  void draw() {  background(0);    if (estadoJogo == TELA\_INICIO) {  telaInicio();  } else if (estadoJogo == JOGANDO) {  jogar();  } else if (estadoJogo == TELA\_VENCEDOR) {  telaVencedor();  }  }  void telaInicio() {  fill(255);  text("PONG MODIFICADO", meioTelaX, 100);  text("Selecione a dificuldade:", meioTelaX, 200);    // Botão Fácil  fill(dificuldade == FACIL ? color(0, 255, 0) : 255);  rect(meioTelaX - 150, 300, 200, 60);  fill(0);  text("Fácil", meioTelaX - 150, 300);    // Botão Médio  fill(dificuldade == MEDIO ? color(255, 255, 0) : 255);  rect(meioTelaX, 300, 200, 60);  fill(0);  text("Médio", meioTelaX, 300);    // Botão Difícil  fill(dificuldade == DIFICIL ? color(255, 0, 0) : 255);  rect(meioTelaX + 150, 300, 200, 60);  fill(0);  text("Difícil", meioTelaX + 150, 300);    // Botão Iniciar  fill(0, 0, 255);  rect(meioTelaX, 400, 200, 60);  fill(255);  text("Iniciar Jogo", meioTelaX, 400);  }  void mousePressed() {  if (estadoJogo == TELA\_INICIO) {  // Verifica clique nos botões de dificuldade  if (mouseX > meioTelaX - 250 && mouseX < meioTelaX - 50 && mouseY > 270 && mouseY < 330) {  dificuldade = FACIL;  } else if (mouseX > meioTelaX - 100 && mouseX < meioTelaX + 100 && mouseY > 270 && mouseY < 330) {  dificuldade = MEDIO;  } else if (mouseX > meioTelaX + 50 && mouseX < meioTelaX + 250 && mouseY > 270 && mouseY < 330) {  dificuldade = DIFICIL;  }  // Verifica clique no botão iniciar  else if (mouseX > meioTelaX - 100 && mouseX < meioTelaX + 100 && mouseY > 370 && mouseY < 430) {  iniciarJogo();  }  } else if (estadoJogo == TELA\_VENCEDOR) {  if (mouseX > meioTelaX - 100 && mouseX < meioTelaX + 100 && mouseY > 370 && mouseY < 430) {  reiniciarJogo();  }  }  }  void iniciarJogo() {  // Ajusta as configurações baseado na dificuldade  switch(dificuldade) {  case FACIL:  alturaPaletaJogador = alturaPaletaFacil;  alturaPaletaCPU = alturaPaletaFacil;  velocidadeBolaX = 4;  velocidadeBolaY = 4;  velocidadePaletaCPU = 4;  break;  case MEDIO:  alturaPaletaJogador = alturaPaletaMedio;  alturaPaletaCPU = alturaPaletaMedio;  velocidadeBolaX = 6;  velocidadeBolaY = 6;  velocidadePaletaCPU = 6;  break;  case DIFICIL:  alturaPaletaJogador = alturaPaletaDificil;  alturaPaletaCPU = alturaPaletaDificil;  velocidadeBolaX = 8;  velocidadeBolaY = 8;  velocidadePaletaCPU = 8;  break;  }    estadoJogo = JOGANDO;  placarJogador = 0;  placarCPU = 0;  jogoTerminou = false;  posicaoBolaX = meioTelaX;  posicaoBolaY = meioTelaY;  }  void reiniciarJogo() {  estadoJogo = TELA\_INICIO;  }  void telaVencedor() {  fill(255);  text("FIM DE JOGO", meioTelaX, 150);  text(vencedor + " VENCEU!", meioTelaX, 250);    // Botão Reiniciar  fill(0, 0, 255);  rect(meioTelaX, 400, 200, 60);  fill(255);  text("Jogar Novamente", meioTelaX, 400);  }  void jogar() {  // Desenha elementos do jogo  desenharPaletas();  desenharBola();  desenharPlacar();  desenharLinhaCentral();    // Movimentação  moverPaletaJogador();  moverPaletaCPU();  moverBola();    // Verifica colisões  verificarColisoes();    // Verifica se alguém ganhou  verificarVencedor();  }  void desenharPaletas() {  // Paleta do jogador (esquerda)  rect(larguraPaleta/2, posicaoPaletaJogador, larguraPaleta, alturaPaletaJogador);    // Paleta da CPU (direita)  rect(larguraTela - larguraPaleta/2, posicaoPaletaCPU, larguraPaleta, alturaPaletaCPU);  }  void desenharBola() {  ellipse(posicaoBolaX, posicaoBolaY, tamanhoBola, tamanhoBola);  }  void desenharPlacar() {  text(placarJogador + " " + placarCPU, meioTelaX, 50);  }  void desenharLinhaCentral() {  stroke(255);  for (int i = 0; i < alturaTela; i += 40) {  line(meioTelaX, i, meioTelaX, i + 20);  }  noStroke();  }  void moverPaletaJogador() {  // Movimento com as teclas W e S  if (keyPressed) {  if (key == 'w' || key == 'W') {  posicaoPaletaJogador -= velocidadePaletaJogador;  }  if (key == 's' || key == 'S') {  posicaoPaletaJogador += velocidadePaletaJogador;  }  }    // Limitar paleta dentro da tela  posicaoPaletaJogador = constrain(posicaoPaletaJogador, alturaPaletaJogador/2, alturaTela - alturaPaletaJogador/2);  }  void moverPaletaCPU() {  // IA simples - seguir a bola  if (posicaoPaletaCPU + alturaPaletaCPU/4 < posicaoBolaY) {  posicaoPaletaCPU += velocidadePaletaCPU;  } else if (posicaoPaletaCPU - alturaPaletaCPU/4 > posicaoBolaY) {  posicaoPaletaCPU -= velocidadePaletaCPU;  }    // Limitar paleta dentro da tela  posicaoPaletaCPU = constrain(posicaoPaletaCPU, alturaPaletaCPU/2, alturaTela - alturaPaletaCPU/2);  }  void moverBola() {  posicaoBolaX += velocidadeBolaX;  posicaoBolaY += velocidadeBolaY;    // Rebater nas bordas superior e inferior  if (posicaoBolaY <= tamanhoBola/2 || posicaoBolaY >= alturaTela - tamanhoBola/2) {  velocidadeBolaY \*= -1;  }    // Verificar se passou pelas paletas (ponto)  if (posicaoBolaX <= 0) {  placarCPU++;  resetarBola();  }    if (posicaoBolaX >= larguraTela) {  placarJogador++;  resetarBola();  }  }  void verificarColisoes() {  // Colisão com a paleta do jogador  if (posicaoBolaX - tamanhoBola/2 <= larguraPaleta &&  posicaoBolaY >= posicaoPaletaJogador - alturaPaletaJogador/2 &&  posicaoBolaY <= posicaoPaletaJogador + alturaPaletaJogador/2) {  velocidadeBolaX = abs(velocidadeBolaX); // Garante que seja positivo (direita)  // Ajusta o ângulo baseado em onde a bola acertou a paleta  float diferencaY = posicaoBolaY - posicaoPaletaJogador;  velocidadeBolaY = diferencaY \* 0.1;  }    // Colisão com a paleta da CPU  if (posicaoBolaX + tamanhoBola/2 >= larguraTela - larguraPaleta &&  posicaoBolaY >= posicaoPaletaCPU - alturaPaletaCPU/2 &&  posicaoBolaY <= posicaoPaletaCPU + alturaPaletaCPU/2) {  velocidadeBolaX = -abs(velocidadeBolaX); // Garante que seja negativo (esquerda)  // Ajusta o ângulo baseado em onde a bola acertou a paleta  float diferencaY = posicaoBolaY - posicaoPaletaCPU;  velocidadeBolaY = diferencaY \* 0.1;  }  }  void resetarBola() {  posicaoBolaX = meioTelaX;  posicaoBolaY = meioTelaY;  // Direção aleatória no eixo Y  velocidadeBolaY = random(-3, 3);  // Mantém a direção X mas randomiza um pouco  velocidadeBolaX = (velocidadeBolaX > 0 ? 1 : -1) \* random(3, 5);    // Pequena pausa para dar tempo de se preparar  delay(500);  }  void verificarVencedor() {  if (placarJogador >= placarMaximo || placarCPU >= placarMaximo) {  jogoTerminou = true;  if (placarJogador > placarCPU) {  vencedor = "JOGADOR";  } else {  vencedor = "CPU";  }  estadoJogo = TELA\_VENCEDOR;  }  }  **Jogo do Pong Alternativo - CÓDIGO FONTE:**  // Estados: 0 = menu, 1 = jogando, 2 = vencedor  int estado = 0;  int dificuldade = 1; // 0 = fácil, 1 = médio, 2 = difícil  // Pontuação  int placarEsquerda = 0;  int placarDireita = 0;  int limitePlacar = 5;  String vencedor = "";  int tempoVitoria;  // Bola  float bolaX, bolaY;  float bolaDX, bolaDY;  float bolaTamanho = 15;  float velocidadeBase;  // Palhetas  float palhetaAltura;  float palhetaLargura = 10;  float palhetaEsquerdaY, palhetaDireitaY;  float palhetaVel = 6;  // Controles  boolean wPress, sPress, upPress, downPress;  void setup() {  size(700, 400);  textAlign(CENTER, CENTER);  textSize(24);  }  void draw() {  background(30);    if (estado == 0) {  desenhaMenu();  } else if (estado == 1) {  atualizaBola();  atualizaPalhetas();  desenhaJogo();  verificaPonto();  } else if (estado == 2) {  desenhaVencedor();  if (millis() - tempoVitoria > 4000) {  reiniciar();  }  }  }  // ---------- Menu ----------  void desenhaMenu() {  fill(255);  text("JOGO PONG", width / 2, 50);  textSize(20);  text("Selecione a dificuldade:", width / 2, 120);  text("1 - Fácil", width / 2, 160);  text("2 - Médio", width / 2, 200);  text("3 - Difícil", width / 2, 240);  textSize(24);  }  void keyPressed() {  if (estado == 0) {  if (key == '1') iniciarJogo(0);  if (key == '2') iniciarJogo(1);  if (key == '3') iniciarJogo(2);  } else if (estado == 1) {  if (key == 'w') wPress = true;  if (key == 's') sPress = true;  if (keyCode == UP) upPress = true;  if (keyCode == DOWN) downPress = true;  }  }  void keyReleased() {  if (estado == 1) {  if (key == 'w') wPress = false;  if (key == 's') sPress = false;  if (keyCode == UP) upPress = false;  if (keyCode == DOWN) downPress = false;  }  }  // ---------- Inicialização ----------  void iniciarJogo(int nivel) {  dificuldade = nivel;  if (nivel == 0) {  palhetaAltura = 100;  velocidadeBase = 4;  } else if (nivel == 1) {  palhetaAltura = 80;  velocidadeBase = 5.5;  } else {  palhetaAltura = 60;  velocidadeBase = 7;  }  bolaX = width / 2;  bolaY = height / 2;  bolaDX = random(1) > 0.5 ? velocidadeBase : -velocidadeBase;  bolaDY = random(-2, 2);  palhetaEsquerdaY = height / 2 - palhetaAltura / 2;  palhetaDireitaY = height / 2 - palhetaAltura / 2;  placarEsquerda = 0;  placarDireita = 0;  estado = 1;  }  // ---------- Jogo ----------  void atualizaPalhetas() {  if (wPress) palhetaEsquerdaY -= palhetaVel;  if (sPress) palhetaEsquerdaY += palhetaVel;  if (upPress) palhetaDireitaY -= palhetaVel;  if (downPress) palhetaDireitaY += palhetaVel;  palhetaEsquerdaY = constrain(palhetaEsquerdaY, 0, height - palhetaAltura);  palhetaDireitaY = constrain(palhetaDireitaY, 0, height - palhetaAltura);  }  void atualizaBola() {  bolaX += bolaDX;  bolaY += bolaDY;  if (bolaY <= 0 || bolaY >= height) bolaDY \*= -1;  // Colisão com palhetas  if (bolaX - bolaTamanho/2 <= palhetaLargura &&  bolaY >= palhetaEsquerdaY && bolaY <= palhetaEsquerdaY + palhetaAltura) {  bolaDX \*= -1;  bolaX = palhetaLargura + bolaTamanho/2;  }  if (bolaX + bolaTamanho/2 >= width - palhetaLargura &&  bolaY >= palhetaDireitaY && bolaY <= palhetaDireitaY + palhetaAltura) {  bolaDX \*= -1;  bolaX = width - palhetaLargura - bolaTamanho/2;  }  }  void desenhaJogo() {  // Palhetas  fill(255);  rect(0, palhetaEsquerdaY, palhetaLargura, palhetaAltura);  rect(width - palhetaLargura, palhetaDireitaY, palhetaLargura, palhetaAltura);    // Bola  ellipse(bolaX, bolaY, bolaTamanho, bolaTamanho);    // Placar  textSize(20);  text(placarEsquerda, width/4, 30);  text(placarDireita, 3\*width/4, 30);  }  void verificaPonto() {  if (bolaX < 0) {  placarDireita++;  reiniciarBola();  } else if (bolaX > width) {  placarEsquerda++;  reiniciarBola();  }  if (placarEsquerda >= limitePlacar) {  vencedor = "Jogador da Esquerda venceu!";  estado = 2;  tempoVitoria = millis();  } else if (placarDireita >= limitePlacar) {  vencedor = "Jogador da Direita venceu!";  estado = 2;  tempoVitoria = millis();  }  }  void reiniciarBola() {  bolaX = width / 2;  bolaY = height / 2;  bolaDX = random(1) > 0.5 ? velocidadeBase : -velocidadeBase;  bolaDY = random(-2, 2);  }  void desenhaVencedor() {  fill(255);  textSize(28);  text(vencedor, width / 2, height / 2);  textSize(16);  text("Reiniciando em instantes...", width / 2, height / 2 + 40);  }  void reiniciar() {  estado = 0;  vencedor = "";  }  **Jogo da Memória - CÓDIGO FONTE:**  // Jogo da Memória no Processing - Versão Corrigida  // Autor: [Seu Nome]  // Versão: 1.1  // Estados do jogo  final int MENU = 0;  final int JOGANDO = 1;  final int VITORIA = 2;  int estadoJogo = MENU;  // Configurações do jogo  int cols = 6;  int rows = 4;  int totalPares = (cols \* rows) / 2;  int tamanhoCarta = 100;  int espacamento = 10;  int margemX, margemY;  // Temas disponíveis  String[] temas = {"Frutas", "Objetos", "Animais", "Personagens"};  int temaSelecionado = 0;  PImage[][] imagensTemas;  PImage[] cartas;  PImage versoCarta;  // Variáveis do jogo  boolean[] cartasViradas;  boolean[] cartasEncontradas;  int carta1 = -1;  int carta2 = -1;  int paresEncontrados = 0;  int tentativas = 0;  boolean podeVirar = true;  int tempoUltimaJogada = 0;  int atrasoVirar = 1000; // 1 segundo  // Elementos da interface  PFont fonte;  void setup() {  size(800, 600);  fonte = createFont("Arial", 24);  textFont(fonte);    // Calcular margens para centralizar o grid  margemX = (width - (cols \* (tamanhoCarta + espacamento) - espacamento)) / 2;  margemY = (height - (rows \* (tamanhoCarta + espacamento) - espacamento)) / 2;    // Carregar imagens  carregarImagens();    // Inicializar cartas  inicializarCartas();    // Criar imagem do verso da carta  versoCarta = createImage(tamanhoCarta, tamanhoCarta, RGB);  desenharVersoCarta();  }  void desenharVersoCarta() {  // Desenhar o verso da carta  PGraphics pg = createGraphics(tamanhoCarta, tamanhoCarta);  pg.beginDraw();  pg.background(0, 100, 200);  pg.fill(255);  pg.textSize(20);  pg.textAlign(CENTER, CENTER);  pg.text("?", pg.width/2, pg.height/2);  pg.endDraw();  versoCarta = pg.get();  }  void carregarImagens() {  // Aqui você deve carregar as imagens para cada tema  // Este é um exemplo com imagens simples geradas  imagensTemas = new PImage[temas.length][totalPares];    for (int t = 0; t < temas.length; t++) {  for (int i = 0; i < totalPares; i++) {  // Criar PGraphics para desenhar a imagem  PGraphics pg = createGraphics(tamanhoCarta, tamanhoCarta);  pg.beginDraw();    // Preencher fundo com cor base  if (t == 0) { // Frutas  pg.background(100 + i \* 20, 150 + i \* 10, 50 + i \* 5);  } else if (t == 1) { // Objetos  pg.background(50 + i \* 5, 100 + i \* 20, 150 + i \* 10);  } else if (t == 2) { // Animais  pg.background(150 + i \* 10, 50 + i \* 5, 100 + i \* 20);  } else { // Personagens  pg.background(200 - i \* 5, 100 + i \* 15, 50 + i \* 25);  }    // Adicionar padrão  pg.noStroke();  for (int x = 0; x < pg.width; x += 10) {  for (int y = 0; y < pg.height; y += 10) {  if ((x + y) % 20 == 0) {  pg.fill(255, 50);  pg.rect(x, y, 10, 10);  }  }  }    // Adicionar texto para identificar  pg.fill(255);  pg.textSize(20);  pg.textAlign(CENTER, CENTER);  pg.text(temas[t].charAt(0) + "" + (i+1), pg.width/2, pg.height/2);    pg.endDraw();    imagensTemas[t][i] = pg.get();  }  }  }  void inicializarCartas() {  cartas = new PImage[totalPares \* 2];  cartasViradas = new boolean[totalPares \* 2];  cartasEncontradas = new boolean[totalPares \* 2];    // Preencher com pares de imagens  for (int i = 0; i < totalPares; i++) {  cartas[i] = imagensTemas[temaSelecionado][i];  cartas[i + totalPares] = imagensTemas[temaSelecionado][i];  }    // Embaralhar cartas  for (int i = 0; i < cartas.length; i++) {  int r = (int)random(i, cartas.length);  PImage temp = cartas[i];  cartas[i] = cartas[r];  cartas[r] = temp;  }    // Resetar estado do jogo  for (int i = 0; i < cartasViradas.length; i++) {  cartasViradas[i] = false;  cartasEncontradas[i] = false;  }    carta1 = -1;  carta2 = -1;  paresEncontrados = 0;  tentativas = 0;  }  void draw() {  background(240);    // Atualizar lógica do jogo  update();    switch(estadoJogo) {  case MENU:  desenharMenu();  break;  case JOGANDO:  desenharJogo();  break;  case VITORIA:  desenharVitoria();  break;  }  }  void desenharMenu() {  fill(0);  textAlign(CENTER, CENTER);  textSize(36);  text("Jogo da Memória", width/2, 100);    textSize(24);  text("Selecione um tema:", width/2, 180);    // Botões de tema  for (int i = 0; i < temas.length; i++) {  fill(i == temaSelecionado ? color(100, 200, 100) : color(200));  rect(width/2 - 150, 230 + i \* 60, 300, 50, 10);  fill(i == temaSelecionado ? color(255) : color(0));  text(temas[i], width/2, 255 + i \* 60);  }    // Botão iniciar  fill(50, 150, 255);  rect(width/2 - 100, 500, 200, 50, 10);  fill(255);  text("Iniciar Jogo", width/2, 525);  }  void desenharJogo() {  // Desenhar cartas  for (int i = 0; i < cols; i++) {  for (int j = 0; j < rows; j++) {  int index = i + j \* cols;  int x = margemX + i \* (tamanhoCarta + espacamento);  int y = margemY + j \* (tamanhoCarta + espacamento);    if (cartasEncontradas[index]) {  // Carta já encontrada - mostrar com brilho  fill(255, 255, 0, 100);  rect(x, y, tamanhoCarta, tamanhoCarta, 5);  image(cartas[index], x, y);  } else if (cartasViradas[index]) {  // Carta virada - mostrar imagem  image(cartas[index], x, y);  } else {  // Carta não virada - mostrar verso  image(versoCarta, x, y);  }    // Efeito de hover  if (mouseX >= x && mouseX <= x + tamanhoCarta &&  mouseY >= y && mouseY <= y + tamanhoCarta &&  !cartasEncontradas[index] &&  (carta1 != index && carta2 != index)) {  fill(255, 255, 255, 50);  rect(x, y, tamanhoCarta, tamanhoCarta, 5);  }  }  }    // Informações do jogo  fill(0);  textAlign(LEFT, TOP);  textSize(18);  text("Tema: " + temas[temaSelecionado], 20, 20);  text("Tentativas: " + tentativas, 20, 50);  text("Pares encontrados: " + paresEncontrados + "/" + totalPares, 20, 80);    // Botão voltar ao menu  fill(200, 100, 100);  rect(width - 120, 20, 100, 40, 5);  fill(255);  textAlign(CENTER, CENTER);  text("Menu", width - 70, 40);  }  void desenharVitoria() {  fill(0);  textAlign(CENTER, CENTER);  textSize(36);  text("Parabéns!", width/2, height/2 - 60);  textSize(24);  text("Você completou o jogo em " + tentativas + " tentativas.", width/2, height/2);    // Botão jogar novamente  fill(100, 200, 100);  rect(width/2 - 100, height/2 + 60, 200, 50, 10);  fill(255);  text("Jogar Novamente", width/2, height/2 + 85);    // Botão voltar ao menu  fill(50, 150, 255);  rect(width/2 - 100, height/2 + 130, 200, 50, 10);  fill(255);  text("Menu", width/2, height/2 + 155);  }  void mousePressed() {  if (estadoJogo == MENU) {  // Verificar clique nos botões de tema  for (int i = 0; i < temas.length; i++) {  if (mouseX >= width/2 - 150 && mouseX <= width/2 + 150 &&  mouseY >= 230 + i \* 60 && mouseY <= 280 + i \* 60) {  temaSelecionado = i;  }  }    // Verificar clique no botão iniciar  if (mouseX >= width/2 - 100 && mouseX <= width/2 + 100 &&  mouseY >= 500 && mouseY <= 550) {  inicializarCartas();  estadoJogo = JOGANDO;  }  }  else if (estadoJogo == JOGANDO) {  // Verificar clique no botão menu  if (mouseX >= width - 120 && mouseX <= width - 20 &&  mouseY >= 20 && mouseY <= 60) {  estadoJogo = MENU;  return;  }    // Verificar clique nas cartas  if (podeVirar) {  for (int i = 0; i < cols; i++) {  for (int j = 0; j < rows; j++) {  int index = i + j \* cols;  int x = margemX + i \* (tamanhoCarta + espacamento);  int y = margemY + j \* (tamanhoCarta + espacamento);    if (mouseX >= x && mouseX <= x + tamanhoCarta &&  mouseY >= y && mouseY <= y + tamanhoCarta &&  !cartasViradas[index] && !cartasEncontradas[index]) {    // Virar a carta  cartasViradas[index] = true;    if (carta1 == -1) {  carta1 = index;  } else if (carta2 == -1 && index != carta1) {  carta2 = index;  tentativas++;    // Verificar se formam um par  if (cartas[carta1] == cartas[carta2]) {  cartasEncontradas[carta1] = true;  cartasEncontradas[carta2] = true;  carta1 = -1;  carta2 = -1;  paresEncontrados++;    // Verificar vitória  if (paresEncontrados == totalPares) {  estadoJogo = VITORIA;  }  } else {  // Não é par, aguardar um pouco e virar de volta  podeVirar = false;  tempoUltimaJogada = millis();  }  }  }  }  }  }  }  else if (estadoJogo == VITORIA) {  // Verificar clique no botão jogar novamente  if (mouseX >= width/2 - 100 && mouseX <= width/2 + 100 &&  mouseY >= height/2 + 60 && mouseY <= height/2 + 110) {  inicializarCartas();  estadoJogo = JOGANDO;  }    // Verificar clique no botão menu  if (mouseX >= width/2 - 100 && mouseX <= width/2 + 100 &&  mouseY >= height/2 + 130 && mouseY <= height/2 + 180) {  estadoJogo = MENU;  }  }  }  void keyPressed() {  // Pressionar ESC para voltar ao menu  if (key == ESC && estadoJogo != MENU) {  estadoJogo = MENU;  key = 0; // Prevenir comportamento padrão  }  }  void mouseMoved() {  // Mudar cursor quando sobre algo clicável  if (estadoJogo == MENU) {  boolean sobreBotao = false;    // Verificar temas  for (int i = 0; i < temas.length; i++) {  if (mouseX >= width/2 - 150 && mouseX <= width/2 + 150 &&  mouseY >= 230 + i \* 60 && mouseY <= 280 + i \* 60) {  sobreBotao = true;  break;  }  }    // Verificar botão iniciar  if (!sobreBotao && mouseX >= width/2 - 100 && mouseX <= width/2 + 100 &&  mouseY >= 500 && mouseY <= 550) {  sobreBotao = true;  }    cursor(sobreBotao ? HAND : ARROW);  }  else if (estadoJogo == JOGANDO) {  boolean sobreCarta = false;    // Verificar botão menu  if (mouseX >= width - 120 && mouseX <= width - 20 &&  mouseY >= 20 && mouseY <= 60) {  cursor(HAND);  return;  }    // Verificar cartas  for (int i = 0; i < cols; i++) {  for (int j = 0; j < rows; j++) {  int index = i + j \* cols;  int x = margemX + i \* (tamanhoCarta + espacamento);  int y = margemY + j \* (tamanhoCarta + espacamento);    if (mouseX >= x && mouseX <= x + tamanhoCarta &&  mouseY >= y && mouseY <= y + tamanhoCarta &&  !cartasEncontradas[index] &&  (carta1 != index && carta2 != index)) {  sobreCarta = true;  break;  }  }  if (sobreCarta) break;  }    cursor(sobreCarta ? HAND : ARROW);  }  else if (estadoJogo == VITORIA) {  // Verificar botões  boolean sobreBotao =  (mouseX >= width/2 - 100 && mouseX <= width/2 + 100 &&  mouseY >= height/2 + 60 && mouseY <= height/2 + 110) ||  (mouseX >= width/2 - 100 && mouseX <= width/2 + 100 &&  mouseY >= height/2 + 130 && mouseY <= height/2 + 180);    cursor(sobreBotao ? HAND : ARROW);  }  }  void update() {  // Verificar se é hora de virar as cartas para trás  if (!podeVirar && millis() - tempoUltimaJogada > atrasoVirar) {  cartasViradas[carta1] = false;  cartasViradas[carta2] = false;  carta1 = -1;  carta2 = -1;  podeVirar = true;  }  } |