

DG - RELATÓRIO DE ENTREGA - Trabalhos 1, 2 e 3 (Unidade 1)

- O representante do grupo deve enviar via "Chat" do TEAMS num link de acesso a um drive virtual com o material dos trabalhos (SEM COMPACTAR):

ARQUIVO 1: Relatório Técnico (Documento texto Word ou PDF) com:

- Nome dos integrantes do grupo em ordem alfabética;
- Colocar nesse relatório: "Prints" das telas dos Jogos (colocar quantas telas forem necessárias para registrar cada jogo), agrupados por jogo;
- Colocar no relatório, a cópia do código fonte de cada jogo.

Colocar um trabalho em cada pasta com todos os arquivos necessários para executar o jogo.

Nome do Game Studio: JET BEAVERS STUDIO

Alessandro Mathews Cardoso Dornelas Dos Santos - 01614625

Eduardo Rafael Silva Santos - 01602387

Gabryella Tainá Melo Silva - 01612684

Luis Gabriel da Silva Araújo - 01614692

Victhor Emanuel Soares Brito - 01615125

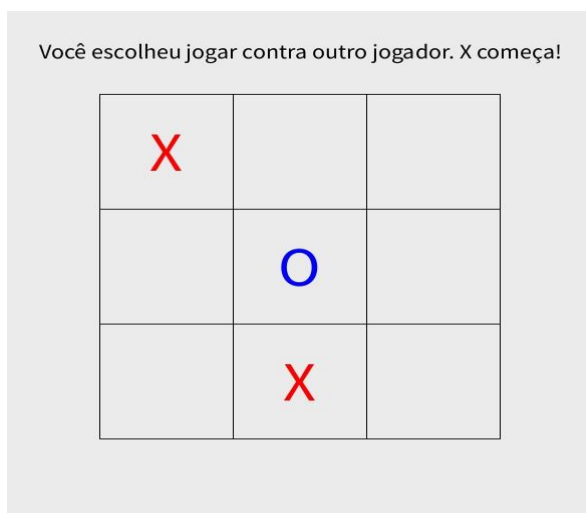
Wagner Vinícius Cassimiro da Silva – 01615748

Primeiro jogo: Jogo do Marciano, em Java.

```
=== O RESGATE DA PRINCESA CÔSMICA ===  
Há muito tempo, no sistema estelar de Auroria, a paz reinava sob o comando da Princesa Zayra...  
Até que um dia, o temível Lorde Xarkon atacou! Ele sequestrou a princesa e a escondeu em uma de suas 100 naves.  
Seu objetivo: descobrir em qual nave a princesa está antes que seja tarde demais!  
  
A missão começou! Você tem 10 tentativas para encontrar a nave certa.  
Digite o número da nave (1 a 100):
```

```
A missão começou! Você tem 10 tentativas para encontrar a nave certa.  
Digite o número da nave (1 a 100): 100  
A nave correta está em um número menor!  
Digite o número da nave (1 a 100): 50  
A nave correta está em um número menor!  
Digite o número da nave (1 a 100): 30  
A nave correta está em um número maior!  
Digite o número da nave (1 a 100): 43  
A nave correta está em um número menor!  
Digite o número da nave (1 a 100): 37  
A nave correta está em um número menor!  
Digite o número da nave (1 a 100): 32  
Parabéns, Capitão! Você encontrou a Princesa Zayra em 6 tentativas!  
  
* NOVO RECORDE! Você fez o resgate mais rápido até agora! *  
  
Deseja tentar novamente? (s/n):
```

Segundo jogo: Jogo Da Velha



Terceiro Jogo: Jogo Da Forca:

Categoria: Cidades

Tentativas restantes: 6

Letras erradas: []



Categoria: Países

Tentativas restantes: 6

Letras erradas: []



A u s t r a l i a

Você venceu!

Categoria: Animais

Tentativas restantes: 0

Letras erradas: [D, F, S, W, P, M]



_ a _ _ o _ _ o

Fim de Jogo! A palavra era: Cachorro

Reiniciar Jogo

PONG

1 JOGADOR

2 JOGADORES

Use as teclas CIMA/BAIXO para selecionar

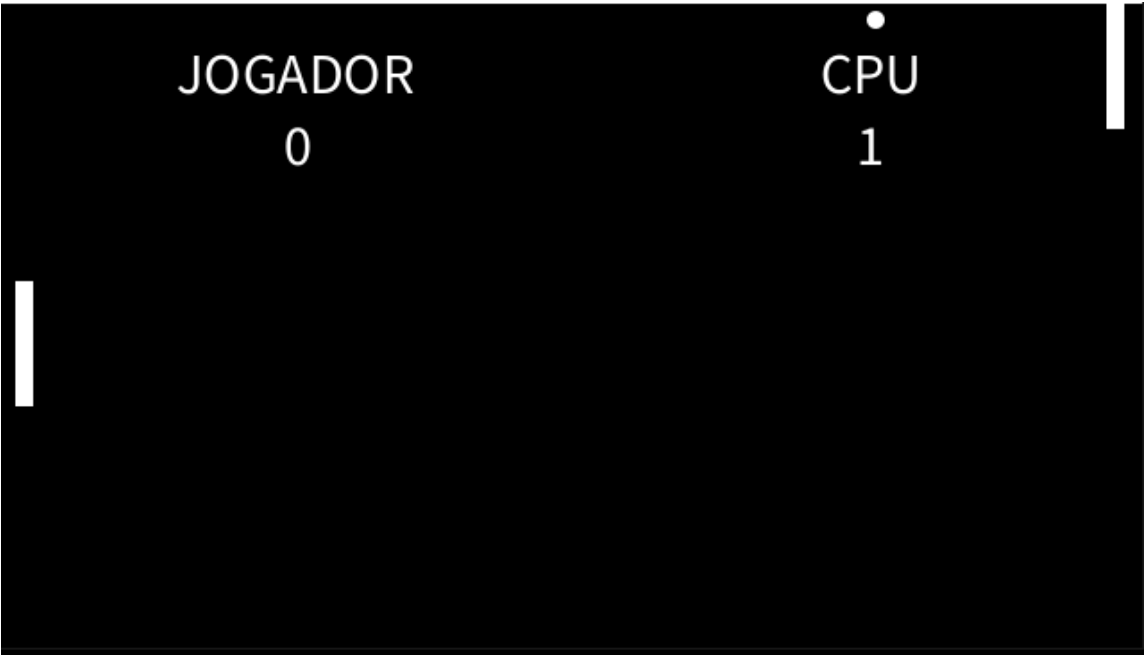
Pressione ENTER para começar

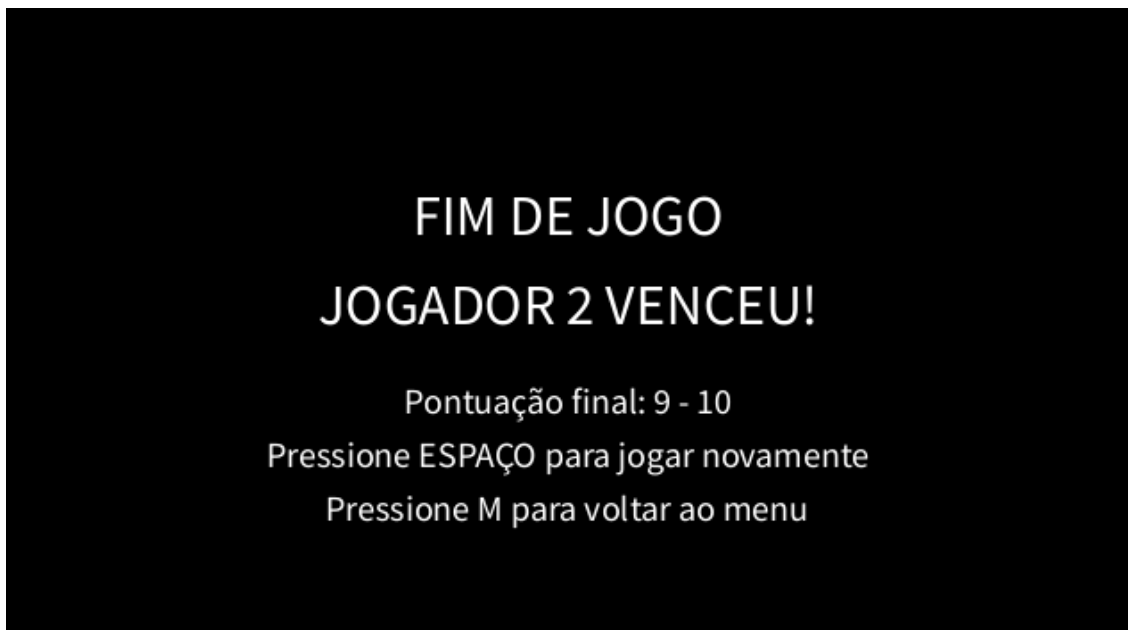
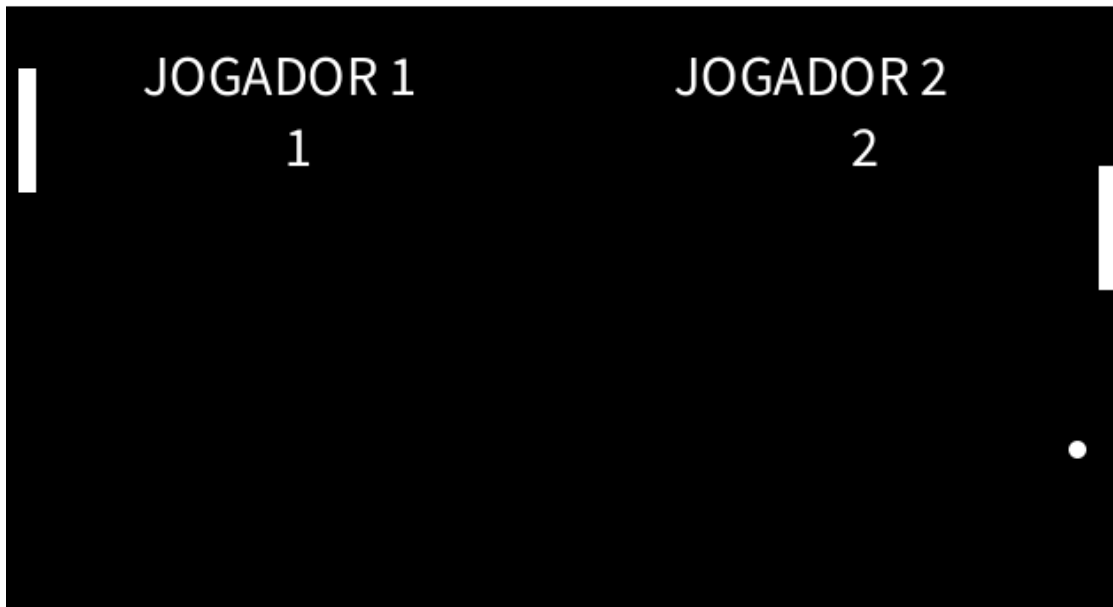
JOGADOR

0

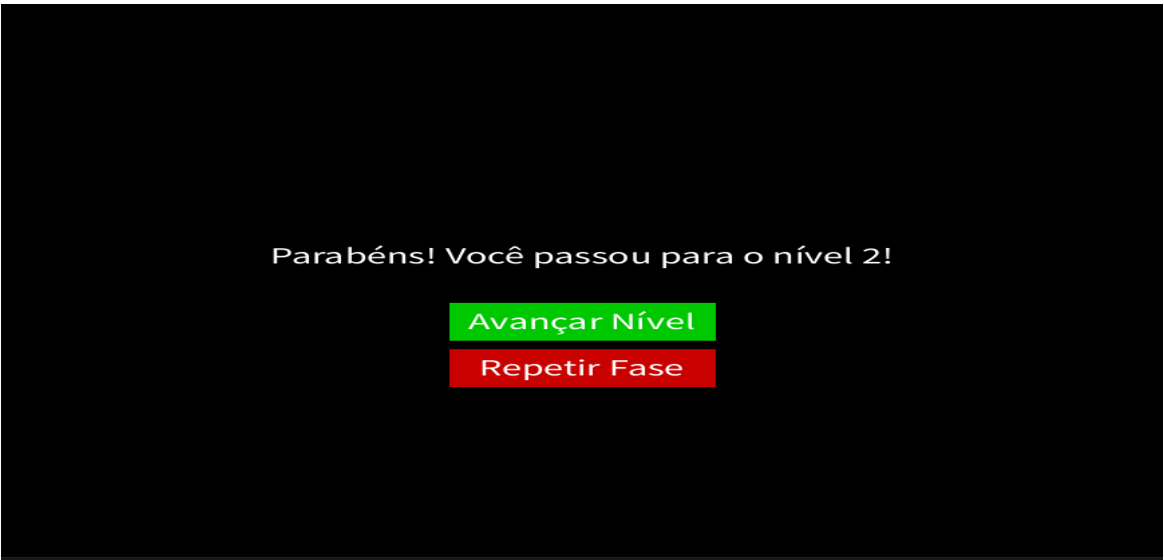
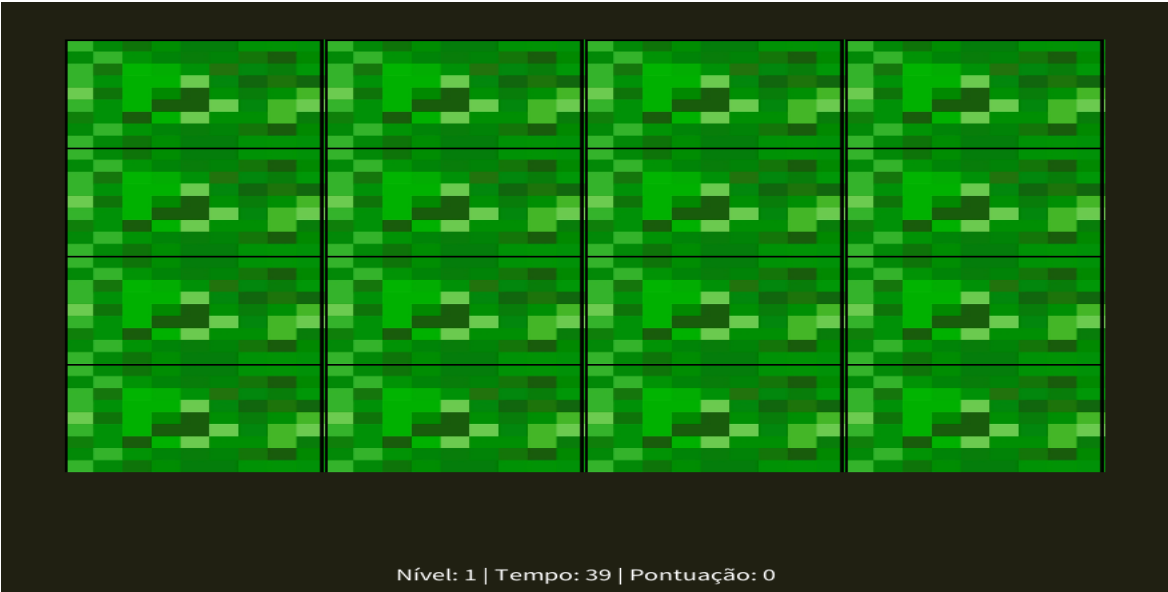
CPU

1

The game area is a black rectangle. On the left side, there is a white vertical line representing a paddle. On the right side, there is a white vertical line representing another paddle. In the center of the rectangle, there is a small white dot representing the ball.



Quinto Jogo: Jogo Da Memória



Parabéns! Você passou para o nível 3!

Avançar Nível

Repetir Fase

Cópia dos códigos fontes: Código

fonte do Jogo Do Marciano:

```
package Jogo_marciano;

import java.util.Random; import
java.util.Scanner;

public class Game {    public static void
main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    Random random = new Random();    int
    melhorTentativa = Integer.MAX_VALUE;

    boolean jogarNovamente;

    // Introdução com a história
```

```
System.out.println("=== O RESGATE DA PRINCESA CÓSMICA ===");
```

```
System.out.println("Há muito tempo, no sistema estelar de Auroria, a paz reinava  
sob o comando da Princesa Zayra...");
```

```
System.out.println("Até que um dia, o temível Lorde Xarkon atacou! Ele sequestrou  
a princesa e a escondeu em uma de suas 100 naves.");
```

```
System.out.println("Seu objetivo: descobrir em qual nave a princesa está antes que  
seja tarde demais!");
```

```
do {          int marcianoPosicao =  
random.nextInt(100) + 1;
```

```
    int tentativa;          int  
tentativasMaximas = 10;          int  
tentativas = 0;          boolean  
acertou = false;
```

```
System.out.println("\nA missão começou! Você tem " + tentativasMaximas + "  
tentativas para encontrar a nave certa.");
```

```
while (!acertou && tentativas < tentativasMaximas) {  
System.out.print("Digite o número da nave (1 a 100): ");          while  
(!scanner.hasNextInt()) {  
    System.out.print("Sério? Isso nem parece um número! Tenta de novo, mas  
    agora entre 1 e 100: ");  
    scanner.next();  
}  
tentativa = scanner.nextInt();  
  
if (tentativa < 1 || tentativa > 100) {  
    System.out.println("Número fora do intervalo! Digite um valor entre 1 e
```



```

100.");

        continue;
    }

    tentativas++;

    if (tentativa == marcianoPosicao) {

        System.out.println("Parabéns, Capitão! Você encontrou a Princesa Zayra em
" + tentativas + " tentativas!");

        acertou = true;

        if (tentativas < melhorTentativa) {

            melhorTentativa = tentativas;

            System.out.println("\n\u2B50 NOVO RECORDE! Você fez o resgate
            mais rápido até agora! \u2B50");

        }

        } else if (tentativa < marcianoPosicao) {

            System.out.println("A nave correta está em um número maior!");

        } else {

            System.out.println("A nave correta está em um número menor!");

        }

        if (tentativas == tentativasMaximas && !acertou) {

            System.out.println("\nMissão falhou! Lorde Xarkon escapou e a Princesa
            Zayra estava na nave número " + marcianoPosicao + ".");

        }

    }
}

```

```

        System.out.print("\nDeseja tentar novamente? (s/n): ");

jogarNovamente = scanner.next().equalsIgnoreCase("s");

        } while (jogarNovamente);

        System.out.println("\nObrigado por jogar! Seu melhor recorde foi " +
(melhorTentativa == Integer.MAX_VALUE ? "-" : melhorTentativa) + " tentativas.");

        scanner.close();

    }

}

```

|Código fonte do Jogo Da Velha:

```

int[][] board = new int[3][3]; boolean
gameStarted = false; boolean
playingWithComputer = false;
boolean playerX = true; boolean
gameOver = false; int winner = 0;
String gameMessage = "Escolha uma opção"; boolean
canGoBackToMenu = false;

int boardSize = 400; int
cellSize = boardSize / 3;

void setup() {
    size(600, 600);

    textAlign(CENTER, CENTER);

    textSize(32);

    drawMenu();

```

```
}
```

```
void draw() {
```

```
background(235);
```

```
if (!gameStarted) {
```

```
drawMenu(); }
```

```
else {
```

```
drawBoard();
```

```
displayMessage();
```

```
if (gameOver) {
```

```
drawBackButton();
```

```
 }
```

```
 }
```

```
}
```

```
void drawMenu() {
```

```
background(235);
```

```
fill(0); textSize(50); text("Jogo da Velha", width / 2,  
height / 4 - 30); fill(70, 130, 180); noStroke(); rect(width  
/ 4, height / 2 - 40, width / 2, 50, 20); rect(width / 4, height /  
2 + 50, width / 2, 50, 20); fill(255); textSize(20);  
text("Jogar contra o computador", width / 2, height / 2 - 15);  
text("Jogar contra outro jogador", width / 2, height / 2 + 73);  
}
```

```

void drawBoard() { int startX =
(width - boardSize) / 2; int startY =
(height - boardSize) / 2; for (int i =
0; i < 3; i++) { for (int j = 0; j < 3;
j++) { float x = startX + i *
cellSize; float y = startY + j *
cellSize; if (board[i][j] == 1) {
fill(255, 0, 0); textSize(64);
text("X", x + cellSize / 2, y +
cellSize / 2); } else if (board[i][j]
== -1) { fill(0, 0, 255);
textSize(64); text("O", x +
cellSize / 2, y + cellSize / 2);
} stroke(0); noFill();
rect(x, y, cellSize, cellSize);
}
}
}

```

```

void displayMessage() {
fill(0); textSize(24); text(gameMessage,
width / 2, height - 550); }

```

```

void drawBackButton() {

```

```

fill(34, 139, 34); noStroke(); rect(width /
4, height - 90, width / 2, 40, 20); fill(255);

textAlign(CENTER, CENTER);

textSize(20); text("Voltar ao Menu", width /
2, height - 70);

}

```

```

void mousePressed() { if (!gameStarted) { if (mouseX > width /
4 && mouseX < width / 4 + width / 2 && mouseY > height / 2
- 30 && mouseY < height / 2 + 10) { playingWithComputer =
true; gameStarted = true; gameMessage = "Você jogará
contra o computador. X começa!";

initializeBoard();

}

```

```

if (mouseX > width / 4 && mouseX < width / 4 + width / 2 &&
mouseY > height / 2 + 40 && mouseY < height / 2 + 80) {

playingWithComputer = false; gameStarted = true;

```

```

gameMessage = "Você escolheu jogar contra outro jogador. X começa!";

initializeBoard();

}

```

```

} else if (gameOver && canGoBackToMenu) { if (mouseX >
width / 4 && mouseX < width / 4 + width / 2 && mouseY >
height - 90 && mouseY < height - 50) { gameStarted = false;

gameOver = false; canGoBackToMenu = false;

gameMessage = "Escolha uma opção";

```

```

        initializeBoard();

redraw();

    }

    } else {    int i = floor((mouseX - (width - boardSize) / 2) /
cellSize);    int j = floor((mouseY - (height - boardSize) / 2) /
cellSize);

        if (board[i][j] == 0) {    board[i][j] = playerX ? 1 : -1;
playerX = !playerX;    checkWinner();    if (!gameOver
&& playingWithComputer && !playerX) {
computerMove();    checkWinner();    playerX =
!playerX;

        }

redraw();

    }

}

}

}

void checkWinner() {

    for (int i = 0; i < 3; i++) {    if (abs(board[i][0] +
board[i][1] + board[i][2]) == 3) {    gameOver = true;
winner = board[i][0];

        }

        if (abs(board[0][i] + board[1][i] + board[2][i]) == 3) {
gameOver = true;    winner = board[0][i];

```

```
}  
}
```

```
if (abs(board[0][0] + board[1][1] + board[2][2]) == 3) {  
gameOver = true;    winner = board[0][0];  
  
}  
  
if (abs(board[0][2] + board[1][1] + board[2][0]) == 3) {  
gameOver = true;    winner = board[0][2];  
  
}
```

```
boolean full = true; for  
(int i = 0; i < 3; i++) {    for  
(int j = 0; j < 3; j++) {    if  
(board[i][j] == 0) {  
full = false;  
  
    }  
  
    }  
  
}  
  
if (full && !gameOver) {  
gameOver = true;    winner  
= 2;  
  
}
```

```
if (gameOver) {
```

```

        gameMessage = winner == 1 ? "Jogador X venceu!" : (winner == -1 ? "Jogador O
venceu!" : "Empate!");    canGoBackToMenu = true;

        redraw();

    }

}

```

```

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

board[i][j] = 0;

    }

}    playerX = true;

gameOver = false;    winner =
0;    canGoBackToMenu =
false;    redraw();

}

```

```

void computerMove() {

    int i, j;    do {        i =
int(random(3));        j =
int(random(3));    } while
(board[i][j] != 0);

board[i][j] = -1;

}

```


Código fonte Jogo Da Forca:

```
import java.util.HashSet;
```

```
String[][] categorias = {
```

```
    {"Frutas", "Banana", "Maca", "Laranja", "Morango", "Uva", "Abacaxi", "Kiwi",  
    "Melancia", "Cabeluda"},
```

```
    {"Animais", "Cachorro", "Gato", "Elefante", "Tigre", "Leao", "Girafa", "Zebra",  
    "Rato", "Macaco"},
```

```
    {"Cidades", "Sao Paulo", "Rio de Janeiro", "Salvador", "Recife", "Brasilia", "Porto  
Alegre", "Curitiba", "Fortaleza", "Manaus"},
```

```
    {"Veiculos", "Carro", "Moto", "Bicicleta", "Onibus", "Aviao", "Navio", "Trem",  
    "Helicoptero", "Caminhao"},
```

```
    {"Esportes", "Futebol", "Basquete", "Volei", "Natacao", "Handebol", "Golfe", "Boxe",  
    "Rugby", "Xadrez", "Corrida"},
```

```
    {"Comidas", "Pizza", "Hamburguer", "Macarrao", "Sushi", "Feijoada", "Lasanha",  
    "Salada", "Churrasco", "Arroz", "Bife"},
```

```
    {"Países", "Brasil", "Argentina", "Franca", "Italia", "Espanha", "Alemanha", "Estados  
Unidos", "Japao", "Canada", "Australia"}
```

```
};
```

```
String categoriaEscolhida;
```

```
String palavraEscolhida;
```

```
String palavraOculta;
```

```
String dica; int
```

```
tentativas = 6;
```

```
ArrayList<Character> letrasErradas;
```

```
HashSet<Character> letrasTentadas;
```

```
boolean jogoAtivo; boolean
```

```
jogadorVenceu; int tempoReinicio =
```

```
0;
```

```
void setup() {
```

```
size(800, 600);
```

```
frameRate(60);
```

```
iniciarJogo();
```

```
}
```

```
void draw() {
```

```
background(245);
```

```
    textSize(24);
```

```
    fill(50);
```

```
    textAlign(LEFT);
```

```
    text(dica, 30, 50);
```

```
    mostrarPalavraOculta();
```

```
    textSize(20); fill(200, 0, 0); text("Tentativas
```

```
restantes: " + tentativas, 30, 120);
```

```
    fill(0);
```

```
    text("Letras erradas: " + letrasErradas.toString().toUpperCase(), 30, 150);
```

```

desenharForca();

if (!jogoAtivo) {

    fill(0);

    textSize(28);

    textAlign(CENTER, CENTER);

    if (jogadorVenceu) {    fill(0, 200, 0);

    text("Você venceu!", width / 2, height / 2 + 200);

    if (millis() - tempoReinicio > 3000) {

    iniciarJogo();

        }

        } else {    fill(200, 0, 0);    text("Fim de Jogo! A palavra era: " + palavraEscolhida,

width / 2, height / 2 + 200);    desenharBotaoReiniciar();

        }

    }

}

void keyPressed() {  if (jogoAtivo && (key >= 'a' && key <= 'z' || key >=

'A' && key <= 'Z')) {    char letra = Character.toLowerCase(key);

    if (letrasTentadas.contains(letra)) {

return;

        }

        letrasTentadas.add(letra);

```

```

        if (palavraEscolhida.toLowerCase().contains(String.valueOf(letra))) {
            atualizarPalavraOcultada(letra);    if (palavraOcultada.equalsIgnoreCase(palavraEscolhida))
        {
            jogoAtivo = false;    jogadorVenceu = true;    tempoReinicio = millis();

        }

        } else {

            letrasErradas.add(Character.toUpperCase(letra));

            tentativas--;    if (tentativas <= 0) {        jogoAtivo
            = false;    jogadorVenceu = false;

                }

            }

        }

    }
}

```

```

void mousePressed() {    if (!jogoAtivo &&
!jogadorVenceu &&    mouseX > width - 200 &&
mouseX < width - 50 &&    mouseY > height - 60 &&
mouseY < height - 20) {    iniciarJogo();

    }

}

```

```

void iniciarJogo() {    int categoriaIndex =
int(random(categorias.length));    String[] categoria =
categorias[categoriaIndex];    categoriaEscolhida =
categoria[0];    dica = "Categoria: " +
categoriaEscolhida;

```

```
int palavraIndex = int(random(1, categoria.length));  
palavraEscolhida = categoria[palavraIndex]; palavraOculta  
= "";
```

```
for (int i = 0; i < palavraEscolhida.length(); i++) {  
if (palavraEscolhida.charAt(i) == ' ') {  
palavraOculta += " ";  
} else {  
palavraOculta += "_";  
}  
}
```

```
letrasErradas = new ArrayList<Character>();  
letrasTentadas = new HashSet<Character>();  
tentativas = 6; jogoAtivo = true;  
jogadorVenceu = false;  
}
```

```
void mostrarPalavraOculta() {  
textSize(48);  
fill(0); float x = width / 2 - (textWidth(palavraOculta.replace(" ",  
"_")) / 2);
```

```
for (int i = 0; i < palavraOculta.length(); i++) {  
char c = palavraOculta.charAt(i);
```

```

    if (c == ' ') {

        // Não mostra nada para espaços

    } else if (c == '_') {        text("_", x + i
* 30, height / 2 + 30);

    } else {        text(c, x + i * 30,
height / 2 + 30);

    }

}

}

}

```

```

void atualizarPalavraOculta(char letra) {

    StringBuilder sb = new StringBuilder(palavraOculta);  for (int i =
0; i < palavraEscolhida.length(); i++) {    if
(Character.toLowerCase(palavraEscolhida.charAt(i)) == letra) {
sb.setCharAt(i, palavraEscolhida.charAt(i));

    }

    }  palavraOculta =
sb.toString(); }

```

```

void desenharForca() {

    float baseX = width / 9;

    float baseY = height / 1.7;

    float largura = 8;

```

```

stroke(0); strokeWeight(largura); line(baseX, baseY
+ 200, baseX, baseY - 120); line(baseX, baseY - 120,
baseX + 80, baseY - 120); line(baseX + 80, baseY -
120, baseX + 80, baseY - 40);

if (tentativas <= 5) ellipse(baseX + 80, baseY - 40, 30, 30);

if (tentativas <= 4) line(baseX + 80, baseY - 20, baseX + 80, baseY + 40);

if (tentativas <= 3) line(baseX + 80, baseY - 15, baseX + 60, baseY + 20);
if (tentativas <= 2) line(baseX + 80, baseY - 15, baseX + 100, baseY + 20);

if (tentativas <= 1) line(baseX + 80, baseY + 40, baseX + 60, baseY + 80);
if (tentativas <= 0) line(baseX + 80, baseY + 40, baseX + 100, baseY + 80); }

void desenharBotaoReiniciar() {

    fill(255, 120, 120); noStroke();

    rect(width - 200, height - 60, 150, 40, 40);

    fill(255);

    textSize(18);

    textAlign(CENTER, CENTER);

    text("Reiniciar Jogo", width - 125, height - 40);

}

```

Quarto Jogo: Jogo Pong

```
//Variáveis para o posicionamento da bola

int ballX, ballY;

int ballSize = 10;

int ballSpeedX = 3; // Velocidade inicial

int ballSpeedY = 3; // Velocidade inicial

float difficultMultiplier = 1.0; // Multiplicador de dificuldade


// Controle de tempo para aumentar a dificuldade

int lastDifficultyIncrease = 0;

int difficultyIncreaseInterval = 10000; // A cada 10 segundos

float difficultyIncreaseAmount = 0.1; // Aumenta 10% da velocidade cada vez

float maxDifficultyMultiplier = 3.0; // Limite máximo do multiplicador (3x a
velocidade inicial)


//Variáveis para o posicionamento das palhetas do jogador

int playerPaddleX, playerPaddleY;

int playerWidth = 10;

int playerInitialHeight = 70; // Altura inicial da paleta do jogador

int playerHeight = playerInitialHeight; // Altura atual da paleta

int playerSpeedY = 0;

int minPlayerHeight = 35; // Altura mínima da paleta do jogador


//Variáveis para o posicionamento das palhetas do adversário

int cpuPaddleX, cpuPaddleY;

int cpuWidth = 10;

int cpuInitialHeight = 70; // Altura inicial da paleta do CPU
```



```
int cpuHeight = cpuInitialHeight; // Altura atual da paleta

int cpuSpeedY = 0;

int minCpuHeight = 35; // Altura mínima da paleta do CPU


//Variáveis para marcar a pontuação

int scorePlayer = 0;

int scoreCPU = 0;

int winningScore = 10; // Pontuação máxima para vencer

boolean gameOver = false;

String winner = "";


// Variáveis para o menu e modo de jogo

boolean gameStarted = false;

boolean playerVsCPU = true; // true = player vs CPU, false = player vs player

int player2SpeedY = 0; // Velocidade da segunda paleta no modo multiplayer


//Função de inicialização

void setup()

{

    //Determinar o tamanho e a cor de fundo da tela

    size(640, 360);

    background(0);


    //Atribui posição inicial da bola

    ballX = width / 2;
```

```
ballY = height / 2;
```

```
//Atribui posição inicial da palheta do jogador
```

```
playerPaddleX = 10;
```

```
playerPaddleY = height / 2 - playerHeight / 2;
```

```
//Atribuir posição inicial da palheta do adversário
```

```
cpuPaddleX = width - 20;
```

```
cpuPaddleY = height/2 - cpuHeight / 2;
```

```
//Determina o tamanho da fonte do texto
```

```
textSize(32);
```

```
// Inicializa o tempo para o sistema de dificuldade
```

```
lastDifficultyIncrease = millis();
```

```
}
```

```
//Função de repetição
```

```
void draw()
```

```
{
```

```
//Atualiza a cor de fundo toda vez que a tela for desenhada
```

```
background(0);
```

```
// Verifica se o jogo começou
```

```
if (!gameStarted) {
```

```
displayMenu();

return; // Sai da função draw se estiver no menu
}


// Verifica se o jogo acabou
if (gameOver) {

displayGameOver();

return; // Sai da função draw se o jogo tiver acabado
}


// Atualiza o sistema de dificuldade (somente no modo CPU)
if (playerVsCPU) {

updateDifficulty();
}


//Chamadas de função para execução do jogo

score();

ballMovement();


if (playerVsCPU) {

cpuPaddle(); // IA controla a raquete direita
} else {

player2Paddle(); // Jogador 2 controla a raquete direita
}
```

```
playerPaddle();  
  
}  
  
// Função para exibir o menu de seleção  
void displayMenu() {  
  
    background(0);  
  
    textAlign(CENTER);  
  
    fill(255);  
  
    text("PONG", width/2, height/2 - 80);  
  
  
    // Destaca a opção selecionada  
    if (playerVsCPU) {  
  
        fill(255, 255, 0); // Amarelo para a opção selecionada  
        text("1 JOGADOR", width/2, height/2);  
  
        fill(255); // Branco para opção não selecionada  
        text("2 JOGADORES", width/2, height/2 + 50);  
    } else {  
  
        fill(255); // Branco para opção não selecionada  
        text("1 JOGADOR", width/2, height/2);  
  
        fill(255, 255, 0); // Amarelo para a opção selecionada  
        text("2 JOGADORES", width/2, height/2 + 50);  
    }  
  
    fill(255);  

```

```
    textSize(20);

    text("Use as teclas CIMA/BAIXO para selecionar", width/2, height/2 + 100);

    text("Pressione ENTER para começar", width/2, height/2 + 130);


    // Restaura o tamanho do texto e alinhamento

    textSize(32);

    textAlign(LEFT);

}


// Função para mostrar a tela de fim de jogo

void displayGameOver() {

    background(0);

    textAlign(CENTER);

    fill(255);

    text("FIM DE JOGO", width/2, height/2 - 50);

    text(winner + " VENCEU!", width/2, height/2);

    textSize(20);

    text("Pontuação final: " + scorePlayer + " - " + scoreCPU, width/2, height/2 + 50);

    text("Pressione ESPAÇO para jogar novamente", width/2, height/2 + 80);

    text("Pressione M para voltar ao menu", width/2, height/2 + 110);


    // Restaura o tamanho do texto

    textSize(32);

    textAlign(LEFT);
```

```
}
```

```
// Função para atualizar a dificuldade com base no tempo de jogo
```

```
void updateDifficulty() {
```

```
    // Verifica se é hora de aumentar a dificuldade
```

```
    if (millis() - lastDifficultyIncrease > difficultyIncreaseInterval) {
```

```
        // Aumenta o multiplicador de dificuldade
```

```
        if (difficultMultiplier < maxDifficultyMultiplier) {
```

```
            difficultMultiplier += difficultyIncreaseAmount;
```

```
        // Limita o multiplicador ao máximo definido
```

```
        if (difficultMultiplier > maxDifficultyMultiplier) {
```

```
            difficultMultiplier = maxDifficultyMultiplier;
```

```
        }
```

```
    // Reduz o tamanho da paleta do jogador
```

```
    int heightReduction = 3; // Redução de 3 pixels por nível
```

```
    playerHeight = playerHeight - heightReduction;
```

```
    // Limita a altura mínima da paleta do jogador
```

```
    if (playerHeight < minPlayerHeight) {
```

```
        playerHeight = minPlayerHeight;
```

```
    }
```

```
    // Reduz o tamanho da paleta do CPU também (para equilibrar)
```

```

int cpuHeightReduction = 6; // Redução maior para o CPU, já que começa maior

cpuHeight = cpuHeight - cpuHeightReduction;


// Limita a altura mínima da paleta do CPU

if (cpuHeight < minCpuHeight) {

    cpuHeight = minCpuHeight;

}


// Ajusta a posição Y para manter a paleta centralizada após a redução

playerPaddleY = playerPaddleY + (heightReduction / 2);

cpuPaddleY = cpuPaddleY + (cpuHeightReduction / 2);

}


// Atualiza o tempo do último aumento

lastDifficultyIncrease = millis();

}

}


//Função da movimentação da bola

void ballMovement() {

    // Calcula a velocidade atual com base na dificuldade (só aplica no modo CPU)

    float currentSpeedX = playerVsCPU ? ballSpeedX * difficultMultiplier : ballSpeedX;

    float currentSpeedY = playerVsCPU ? ballSpeedY * difficultMultiplier : ballSpeedY;


    //Atualização da posição da bola com a velocidade ajustada

```

```
ballX = ballX + int(currentSpeedX);
```

```
ballY = ballY + int(currentSpeedY);
```

```
//Verificação da colisão da bola com as extremidades laterais
```

```
if (ballX > width) {
```

```
    ballX = width / 2;
```

```
    ballY = height / 2;
```

```
    ballSpeedX = ballSpeedX * -1;
```

```
    scorePlayer += 1;
```

```
    checkWin();
```

```
}
```

```
if (ballX < 0) {
```

```
    ballX = width / 2;
```

```
    ballY = height / 2;
```

```
    ballSpeedX = ballSpeedX * -1;
```

```
    scoreCPU += 1;
```

```
    checkWin();
```

```
}
```

```
//Verificação da colisão da bola com as extremidades superior e inferior da tela
```

```
if (ballY > height) {
```

```
    ballY = height - ballSize / 2;
```

```
    ballSpeedY = ballSpeedY * -1;
```

```
}
```

```
if (ballY < 0) {
```



```

    ballY = ballSize / 2;

    ballSpeedY = ballSpeedY * -1;
}

//Verificação da colisão da bola com a palheta do jogador
if ((ballY + ballSize / 2 >= playerPaddleY && ballY - ballSize / 2 <= playerPaddleY +
playerHeight) &&
    (ballX + ballSize / 2 >= playerPaddleX && ballX - ballSize / 2 <= playerPaddleX +
playerWidth)) {
    ballSpeedX = ballSpeedX * -1;

    ballX = ballSize / 2 + playerPaddleX + playerWidth;
}

//Verificação da colisão da bola com a palheta do adversário
if ((ballY + ballSize / 2 >= cpuPaddleY && ballY - ballSize / 2 <= cpuPaddleY +
cpuHeight) &&
    (ballX + ballSize / 2 >= cpuPaddleX && ballX - ballSize / 2 <= cpuPaddleX +
cpuWidth)) {
    ballSpeedX = ballSpeedX * -1;

    ballX = cpuPaddleX - ballSize / 2;
}

noStroke();

fill(255); // Cor branca para a bola
ellipse(ballX, ballY, ballSize, ballSize);

```

```
}
```

```
// Função para verificar se alguém venceu
```

```
void checkWin() {
```

```
    if (scorePlayer >= winningScore) {
```

```
        gameOver = true;
```

```
        winner = playerVsCPU ? "JOGADOR" : "JOGADOR 1";
```

```
    } else if (scoreCPU >= winningScore) {
```

```
        gameOver = true;
```

```
        winner = playerVsCPU ? "CPU" : "JOGADOR 2";
```

```
    }
```

```
}
```

```
//Função da palheta do adversário
```

```
void cpuPaddle() {
```

```
    //Atualizando a posição da palheta do adversário
```

```
    cpuPaddleY = cpuPaddleY + cpuSpeedY;
```

```
//Criação do comportamento da palheta inimiga
```

```
if (ballX > width / 2) {
```

```
    if (ballY - ballSize > cpuPaddleY + cpuHeight / 2) {
```

```
        cpuSpeedY = 5;
```

```
    } else if (ballY + ballSize < cpuPaddleY + cpuHeight / 2) {
```

```
        cpuSpeedY = -5;
```

```
    } else {
```

```
    cpuSpeedY = 0;

}

} else {

    cpuSpeedY = 0;

}
```

```
//Limitação dos movimentos da palheta dentro do espaço da tela
```

```
if (cpuPaddleY + cpuHeight > height) {

    cpuPaddleY = height - cpuHeight;

}

if (cpuPaddleY < 0) {

    cpuPaddleY = 0;

}
```

```
fill(255); // Cor branca para a paleta
```

```
rect(cpuPaddleX, cpuPaddleY, cpuWidth, cpuHeight);

}
```

```
// Função da palheta do jogador 2 (para o modo multiplayer)
```

```
void player2Paddle() {
```

```
    // Atualizando a posição da palheta do jogador 2
```

```
    cpuPaddleY = cpuPaddleY + player2SpeedY;
```

```
// Limitação dos movimentos da palheta dentro do espaço da tela
```

```
if (cpuPaddleY + cpuHeight > height) {
```

```
    cpuPaddleY = height - cpuHeight;  
}
```

```
if (cpuPaddleY < 0) {  
    cpuPaddleY = 0;  
}
```

```
fill(255); // Cor branca para a paleta
```

```
rect(cpuPaddleX, cpuPaddleY, cpuWidth, cpuHeight);  
}
```

```
//Função da palheta do jogador
```

```
void playerPaddle() {
```

```
    //Atualizando a posição da palheta do jogador
```

```
    playerPaddleY = playerPaddleY + playerSpeedY;
```

```
//Limitação dos movimentos da palheta dentro do espaço da tela
```

```
if (playerPaddleY + playerHeight > height) {
```

```
    playerPaddleY = height - playerHeight;
```

```
}
```

```
if (playerPaddleY < 0) {
```

```
    playerPaddleY = 0;
```

```
}
```

```
fill(255); // Cor branca para a paleta
```

```
rect(playerPaddleX, playerPaddleY, playerWidth, playerHeight);
```

```
}
```

```
// Exibe o placar
```

```
void score() {
```

```
    fill(255); // Cor branca para o texto
```

```
    // Ajusta o texto dependendo do modo de jogo
```

```
    if (playerVsCPU) {
```

```
        text("JOGADOR", 100, 50);
```

```
        text("CPU", 460, 50);
```

```
    } else {
```

```
        text("JOGADOR 1", 80, 50);
```

```
        text("JOGADOR 2", 380, 50);
```

```
    }
```

```
    text(scorePlayer, 160, 90);
```

```
    text(scoreCPU, 480, 90);
```

```
}
```

```
//Verificação do pressionamento dos botões
```

```
void keyPressed() {
```

```
    // Controles menu
```

```
    if (!gameStarted) {
```

```
        if (keyCode == UP || keyCode == DOWN) {
```

```
            playerVsCPU = !playerVsCPU; // Alterna entre os modos
```

```
}

if (keyCode == ENTER) {

    gameStarted = true;

    resetGame();

}

return;

}


// Controles durante o jogo

if (key == 's' || key == 'S') {

    playerSpeedY = 5;

}

if (key == 'w' || key == 'W') {

    playerSpeedY = -5;

}


// Controles do jogador 2 (somente no modo multiplayer)

if (!playerVsCPU) {

    if (keyCode == DOWN) {

        player2SpeedY = 5;

    }

    if (keyCode == UP) {

        player2SpeedY = -5;

    }

}
```

```
// Reiniciar o jogo quando pressionar espaço na tela de fim de jogo
```

```
if (key == ' ' && gameOver) {
```

```
    resetGame();
```

```
}
```

```
// Voltar ao menu quando pressionar M na tela de fim de jogo
```

```
if ((key == 'm' || key == 'M') && gameOver) {
```

```
    gameStarted = false;
```

```
    gameOver = false;
```

```
}
```

```
}
```

```
// Função para reiniciar o jogo
```

```
void resetGame() {
```

```
    // Reinicia pontuações
```

```
    scorePlayer = 0;
```

```
    scoreCPU = 0;
```

```
    // Reinicia as alturas das paletas
```

```
    playerHeight = playerInitialHeight;
```

```
    cpuHeight = cpuInitialHeight;
```

```
    // Reinicia posições
```

```
    ballX = width / 2;
```

```
ballY = height / 2;

playerPaddleY = height / 2 - playerHeight / 2;

cpuPaddleY = height / 2 - cpuHeight / 2;


// Reinicia velocidades

playerSpeedY = 0;

cpuSpeedY = 0;

player2SpeedY = 0;


// Reinicia dificuldade

difficultMultiplier = 1.0;


// Reinicia o tempo para o sistema de dificuldade

lastDifficultyIncrease = millis();


// Reinicia o estado do jogo

gameOver = false;

winner = "";

}


//Verificação do soltar dos botões

void keyReleased() {

    // Controles do jogador 1

    if (key == 's' || key == 'S' || key == 'w' || key == 'W') {

        playerSpeedY = 0;
```



```
}
```

```
// Controles do jogador 2 (somente no modo multiplayer)
```

```
if (!playerVsCPU) {
```

```
    if (keyCode == DOWN || keyCode == UP) {
```

```
        player2SpeedY = 0;
```

```
    }
```

```
}
```

```
}
```

Quinto jogo: Jogo Da Memória

```
PImage[] cartas;
```

```
PImage fundo, verso;
```

```
int[] tabuleiro;
```

```
boolean[] reveladas;
```

```
int cartaVirada1 = -1, cartaVirada2 = -1;
```

```
boolean podeVirar = true;
```

```
int paresEncontrados = 0;
```

```
int nivel = 1;
```

```
int tempoRestante;
```

```
int pontuacao = 0;
```

```
int colunas, linhas, totalCartas;
```

```
int cartaLargura, cartaAltura;
```

```
int margemX, margemY;
```

```
boolean transicaoNivel = false;
```

```
void setup() {  
  
    size(880, 720); // Tamanho fixo da tela  
  
    fundo = loadImage("fundo.png");  
  
    fundo.resize(width, height);  
  
    verso = loadImage("verso.png");  
  
    iniciarJogo();  
  
}
```

```
void iniciarJogo() {  
  
    definirDificuldade();  
  
    cartas = new PImage[totalCartas / 2];  
  
    tabuleiro = new int[totalCartas];  
  
    reveladas = new boolean[totalCartas];  
  
    cartaLargura = (width - 100) / colunas;  
  
    cartaAltura = (height - 200) / linhas;  
  
    margemX = (width - (colunas * cartaLargura)) / 2;  
  
    margemY = 50;  
  
  
    for (int i = 0; i < totalCartas / 2; i++) {  
  
        cartas[i] = loadImage("carta" + i + ".png");  
  
    }  
  
  
  
    int[] valores = new int[totalCartas];  
  
    for (int i = 0; i < totalCartas / 2; i++) {  
  
        valores[i * 2] = i;
```

```

        valores[i * 2 + 1] = i;
    }

    valores = embaralhar(valores);

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

        tabuleiro[i] = valores[i];

        reveladas[i] = false;

    }

    cartaVirada1 = -1;

    cartaVirada2 = -1;

    podeVirar = true;

    paresEncontrados = 0;

    tempoRestante = 60;

    transicaoNivel = false;

    loop();

}

void definirDificuldade() {

    if (nivel == 1) { colunas = 4; linhas = 4; }

    else if (nivel == 2) { colunas = 6; linhas = 4; }

    else { colunas = 8; linhas = 4; }

    totalCartas = colunas * linhas;

}

int[] embaralhar(int[] array) {

    for (int i = array.length - 1; i > 0; i--) {

```

```

    int j = (int) random(i + 1);

    int temp = array[i];

    array[i] = array[j];

    array[j] = temp;

}

return array;

}

void draw() {

    if (transicaoNivel) {

        telaTransicao();

        return;

    }

    background(fundo);

    fill(255);

    textSize(20);

    textAlign(CENTER);

    text("Nível: " + nivel + " | Tempo: " + tempoRestante + " | Pontuação: " + pontuacao,
width / 2, height - 20);

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

        int x = (i % colunas) * cartaLargura + margemX;

        int y = (i / colunas) * cartaAltura + margemY;

        if (reveladas[i]) {

            image(cartas[tabuleiro[i]], x, y, cartaLargura, cartaAltura);

```

```

    } else {

        image(verso, x, y, cartaLargura, cartaAltura);

    }

}

if (frameCount % 60 == 0 && tempoRestante > 0) {

    tempoRestante--;

}

if (tempoRestante == 0) {

    nivel = 1;

    iniciarJogo();

}

}

void mousePressed() {

    if (transicaoNivel) {

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

            if (mouseY > height / 2 + 30 && mouseY < height / 2 + 80) {

                nivel++;

                if (nivel > 3) nivel = 1;

                iniciarJogo();

            } else if (mouseY > height / 2 + 90 && mouseY < height / 2 + 140) {

                iniciarJogo();

            }

        }

    }

    return;
}

```

```
}
```

```
if (!podeVirar) return;
```

```
if (cartaVirada1 != -1 && cartaVirada2 != -1) verificarPar();
```

```
int coluna = (mouseX - margemX) / cartaLargura;
```

```
int linha = (mouseY - margemY) / cartaAltura;
```

```
int index = linha * colunas + coluna;
```

```
if (index < totalCartas && !reveladas[index]) {
```

```
    if (cartaVirada1 == -1) cartaVirada1 = index;
```

```
    else if (cartaVirada2 == -1) cartaVirada2 = index;
```

```
    reveladas[index] = true;
```

```
}
```

```
}
```

```
void verificarPar() {
```

```
    if (tabuleiro[cartaVirada1] != tabuleiro[cartaVirada2]) {
```

```
        reveladas[cartaVirada1] = false;
```

```
        reveladas[cartaVirada2] = false;
```

```
    } else {
```

```
        paresEncontrados++;
```

```
        pontuacao += 10 * nivel;
```

```
    }
```

```
    cartaVirada1 = -1;
```

```
    cartaVirada2 = -1;
```

```
    podeVirar = true;
```

```
if (paresEncontrados == totalCartas / 2) {  
    transicaoNivel = true;  
}  
}  
  
void telaTransicao() {  
    background(0, 0, 0, 150);  
    fill(255);  
    textSize(30);  
    textAlign(CENTER);  
    text("Parabéns! Você passou para o nível " + (nivel + 1) + "!", width / 2, height / 2 -  
20);  
    fill(0, 200, 0);  
    rect(width / 2 - 100, height / 2 + 30, 200, 50);  
    fill(255);  
    text("Avançar Nível", width / 2, height / 2 + 65);  
    fill(200, 0, 0);  
    rect(width / 2 - 100, height / 2 + 90, 200, 50);  
    fill(255);  
    text("Repetir Fase", width / 2, height / 2 + 125);  
}
```