**Centro Universitário Maurício de Nassau**

**Game Studio: Força da Amizade**

Márcio da Costa Ferreira Junior - 01596976

Pedro Henrique de Souza Cavalcanti - 01595774

Thaynã Queiroz Mota - 01582087

Yuri Gabriel Nogueira De Lima - 01600961

**Trabalhos 1, 2, 3, 4 e 5**

**(Unidade 1)**

|  |
| --- |
| Trabalho apresentado ao curso de Ciências da Computação para obtenção de nota da disciplina de Desenvolvimento de Games. |
| Orientador: Leopoldo Rodrigues |

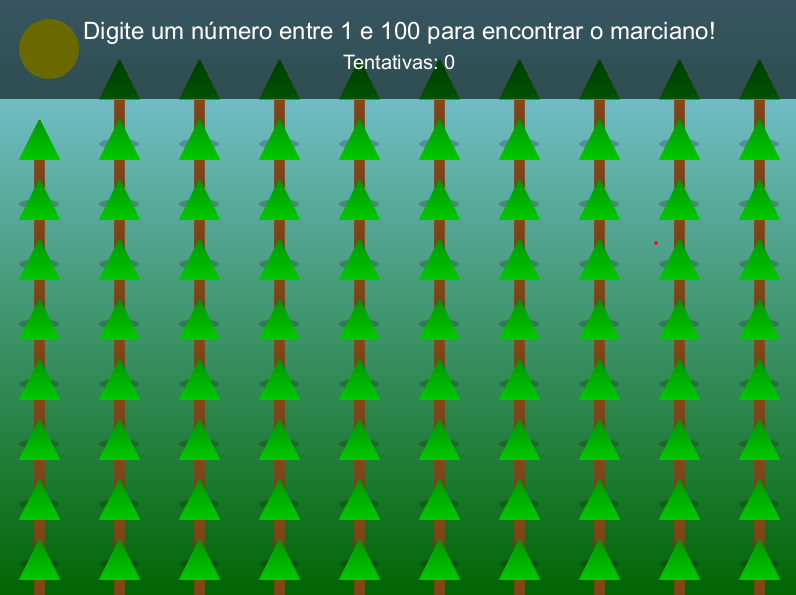
**Recife**

**2025**

**Jogo 1: Jogo do Marciano**

**Capturas de Tela**

Tela inicial do Jogo do Marciano Tela inicial com as 100 árvores numeradas!



Dica "maior" no Jogo do Marciano Feedback quando o número digitado é menor que a posição do marciano!

Uma imagem contendo Forma

O conteúdo gerado por IA pode estar incorreto.

Dica "menor" no Jogo do Marciano Feedback quando o número digitado é maior que a posição do marciano!

Uma imagem contendo Forma

O conteúdo gerado por IA pode estar incorreto.

Marciano encontrado Tela mostrando o marciano encontrado após acertar o número!

Uma imagem contendo Forma

O conteúdo gerado por IA pode estar incorreto.

**Jogo 2: Jogo da Velha (Tic-tac-toe)**

**Capturas de Tela**

Menu inicial do Jogo da VelhaTela do menu com opções de 1 vs 1 e Jogar contra a Máquina!

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Partida em andamento / Tabuleiro com jogadas em andamento!

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Vitória do X Tela mostrando a vitória do jogador X!

Forma

O conteúdo gerado por IA pode estar incorreto.

Vitória do O Tela mostrando a vitória do jogador O!

Forma

O conteúdo gerado por IA pode estar incorreto.

Empate Tela mostrando um empate!

Forma

O conteúdo gerado por IA pode estar incorreto.

**Jogo 3: Jogo da Forca**

**Capturas de Tela**

Menu de níveis do Jogo da Forca Tela de seleção de nível (Fácil, Médio, Difícil)!

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Jogo da Forca – Início Tela inicial com a palavra oculta e forca vazia!

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Jogo em andamento Tela com algumas letras descobertas e boneco parcialmente desenhado!

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Vitória no Jogo da Forca Tela de vitória com a palavra completa!

Uma imagem contendo Diagrama

O conteúdo gerado por IA pode estar incorreto.

Derrota no Jogo da Forca Tela de derrota com a palavra revelada e boneco completo!

Diagrama, Esquemático

O conteúdo gerado por IA pode estar incorreto.

**Tela inicial do jogo 4 (pong)**

Uma imagem contendo Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Escolher o nível de dificuldade:

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Pontuação mostrando:

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Computador venceu : Uma imagem contendo Texto

O conteúdo gerado por IA pode estar incorreto.

Tela inicial do jogo 5 (jogo da memoria ): Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Escolher o nível : Interface gráfica do usuário, Aplicativo, Site

O conteúdo gerado por IA pode estar incorreto.

Tela com timeout para memorizar : Forma

O conteúdo gerado por IA pode estar incorreto.

Tela com as cartas baixadas

Tabela

O conteúdo gerado por IA pode estar incorreto.

Anexos:

Código fonte do jogo 1: ( // Variáveis globais

int arvoreMarciano; // Árvore onde o marciano está escondido

int tentativaAtual; // Número atual que o jogador está tentando

int numeroTentativas = 0; // Contador de tentativas

String mensagem = "Digite um número entre 1 e 100 para encontrar o marciano!";

boolean jogoTerminado = false;

PFont fonte;

color corFundo = color(0, 100, 0);

color corArvore = color(0, 200, 0);

color corTronco = color(139, 69, 19);

color corDestaque = color(255, 255, 0);

color corTexto = color(255);

void setup() {

size(800, 600);

iniciarJogo();

fonte = createFont("Arial", 24);

textFont(fonte);

}

void iniciarJogo() {

// Gera um número aleatório entre 1 e 100

arvoreMarciano = int(random(1, 101));

tentativaAtual = 0;

numeroTentativas = 0;

mensagem = "Digite um número entre 1 e 100 para encontrar o marciano!";

jogoTerminado = false;

}

void draw() {

// Desenha o céu com gradiente

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

float inter = map(i, 0, height, 0, 1);

color c = lerpColor(color(135, 206, 235), color(0, 100, 0), inter);

stroke(c);

line(0, i, width, i);

}

// Desenha o sol

fill(255, 255, 0);

noStroke();

ellipse(50, 50, 60, 60);

// Desenha as árvores

desenharArvores();

// Desenha o marciano se o jogo estiver terminado

if (jogoTerminado) {

desenharMarciano(arvoreMarciano);

}

// Interface do usuário

desenharInterface();

}

void desenharInterface() {

// Painel de informações

fill(0, 0, 0, 150);

noStroke();

rect(0, 0, width, 100);

// Mensagem principal

fill(corTexto);

textAlign(CENTER);

textSize(24);

text(mensagem, width/2, 40);

// Contador de tentativas

textSize(20);

text("Tentativas: " + numeroTentativas, width/2, 70);

// Mostra o número atual sendo digitado

if (tentativaAtual > 0) {

textSize(30);

text(tentativaAtual, width/2, 100);

}

// Mostra instrução para reiniciar

if (jogoTerminado) {

textSize(16);

text("Pressione R para jogar novamente", width/2, 120);

}

}

void desenharArvores() {

for (int i = 1; i <= 100; i++) {

float x = (i % 10) \* 80 + 40;

float y = (i / 10) \* 60 + 100;

// Desenha sombra da árvore

fill(0, 0, 0, 50);

noStroke();

ellipse(x, y + 45, 40, 10);

// Desenha o tronco com gradiente

for (int j = 0; j < 40; j++) {

float inter = map(j, 0, 40, 0, 1);

color c = lerpColor(color(139, 69, 19), color(101, 67, 33), inter);

stroke(c);

line(x - 5, y + j, x + 5, y + j);

}

// Desenha a copa da árvore com gradiente

for (int j = 0; j < 40; j++) {

float inter = map(j, 0, 40, 0, 1);

color c = lerpColor(color(0, 200, 0), color(0, 150, 0), inter);

stroke(c);

line(x - 20 + j/2, y - j, x + 20 - j/2, y - j);

}

// Número da árvore

fill(255);

textSize(12);

textAlign(CENTER);

text(i, x, y + 60);

// Se o jogo estiver terminado e esta for a árvore correta, destaca-a

if (jogoTerminado && i == arvoreMarciano) {

stroke(corDestaque);

strokeWeight(3);

noFill();

ellipse(x, y - 20, 40, 40);

// Efeito de brilho

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

stroke(corDestaque, 100 - j \* 30);

strokeWeight(3 - j);

ellipse(x, y - 20, 40 + j \* 10, 40 + j \* 10);

}

}

}

}

void desenharMarciano(int arvore) {

float x = (arvore % 10) \* 80 + 40;

float y = (arvore / 10) \* 60 + 60;

// Efeito de brilho ao redor do marciano

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

fill(0, 255, 0, 50 - i \* 15);

noStroke();

ellipse(x, y, 40 + i \* 10, 40 + i \* 10);

}

// Corpo do marciano

fill(0, 255, 0);

noStroke();

ellipse(x, y, 30, 30);

// Olhos

fill(255);

ellipse(x - 5, y - 5, 5, 5);

ellipse(x + 5, y - 5, 5, 5);

// Pupilas

fill(0);

ellipse(x - 5, y - 5, 2, 2);

ellipse(x + 5, y - 5, 2, 2);

// Antenas

stroke(0, 255, 0);

strokeWeight(2);

line(x - 8, y - 15, x - 8, y - 8);

line(x + 8, y - 15, x + 8, y - 8);

// Bolinhas nas antenas

fill(0, 255, 0);

ellipse(x - 8, y - 15, 4, 4);

ellipse(x + 8, y - 15, 4, 4);

// Efeito de brilho nas antenas

for (int i = 0; i < 2; i++) {

stroke(0, 255, 0, 100 - i \* 50);

strokeWeight(1);

line(x - 8, y - 15 - i \* 2, x - 8, y - 8);

line(x + 8, y - 15 - i \* 2, x + 8, y - 8);

}

}

void keyPressed() {

if (key == 'r' || key == 'R') {

iniciarJogo();

return;

}

if (jogoTerminado) return;

if (key >= '0' && key <= '9') {

String numero = "";

if (tentativaAtual > 0) {

numero = str(tentativaAtual) + key;

} else {

numero = str(key);

}

tentativaAtual = int(numero);

if (tentativaAtual > 100) {

tentativaAtual = 100;

}

} else if (key == ENTER) {

numeroTentativas++;

if (tentativaAtual == arvoreMarciano) {

mensagem = "Parabéns! Você encontrou o marciano na árvore " + arvoreMarciano + "!";

jogoTerminado = true;

} else if (tentativaAtual < arvoreMarciano) {

mensagem = "O marciano está em uma árvore com número MAIOR que " + tentativaAtual;

} else {

mensagem = "O marciano está em uma árvore com número MENOR que " + tentativaAtual;

}

tentativaAtual = 0;

} else if (key == BACKSPACE) {

tentativaAtual = 0;

mensagem = "Digite um número entre 1 e 100 para encontrar o marciano!";

}

} )

Código fonte do jogo 2: (// Variáveis globais

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

boolean vezDoX = true; // true = vez do X, false = vez do O

boolean jogoTerminado = false;

String mensagem = "Selecione o modo de jogo";

PFont fonte;

color corX = color(255, 50, 50); // Vermelho

color corO = color(50, 50, 255); // Azul

color corLinha = color(255, 255, 255); // Branco

color corFundo = color(30, 30, 30); // Cinza escuro

boolean modoComputador = false; // true = modo contra computador, false = modo 2 jogadores

boolean menuAtivo = true; // true = menu de seleção, false = jogo em andamento

void setup() {

size(600, 700);

fonte = createFont("Arial", 32);

textFont(fonte);

iniciarJogo();

}

void iniciarJogo() {

// Limpa o tabuleiro

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

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

tabuleiro[i][j] = 0;

}

}

vezDoX = true;

jogoTerminado = false;

menuAtivo = true;

mensagem = "Selecione o modo de jogo";

}

void draw() {

// Desenha o fundo com gradiente

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

float inter = map(i, 0, height, 0, 1);

color c = lerpColor(color(50, 50, 50), color(30, 30, 30), inter);

stroke(c);

line(0, i, width, i);

}

// Desenha o painel de informações

fill(0, 0, 0, 150);

noStroke();

rect(0, 0, width, 100);

// Desenha a mensagem

fill(255);

textAlign(CENTER);

textSize(32);

text(mensagem, width/2, 60);

if (menuAtivo) {

desenharMenu();

} else {

// Desenha o tabuleiro

desenharTabuleiro();

// Desenha o botão de reiniciar

desenharBotaoReiniciar();

// Se for modo computador e for a vez do O, faz a jogada

if (modoComputador && !vezDoX && !jogoTerminado) {

fazerJogadaComputador();

}

}

}

void desenharMenu() {

float x = width/2 - 150;

float y = 200;

float largura = 300;

float altura = 80;

// Botão Modo 2 Jogadores

fill(0); // Fundo preto

stroke(255);

strokeWeight(2);

rect(x, y, largura, altura, 10);

fill(255); // Texto branco

textAlign(CENTER);

textSize(24);

text("1 vs 1", width/2, y + 50);

// Botão Modo Computador

y += 100;

fill(0); // Fundo preto

rect(x, y, largura, altura, 10);

fill(255); // Texto branco

text("Jogar contra a Máquina", width/2, y + 50);

}

void desenharTabuleiro() {

float tamanho = 150; // Tamanho de cada célula

float inicioX = (width - tamanho \* 3) / 2;

float inicioY = 150;

// Desenha as linhas do tabuleiro

stroke(corLinha);

strokeWeight(4);

// Linhas horizontais

for (int i = 1; i < 3; i++) {

line(inicioX, inicioY + i \* tamanho, inicioX + 3 \* tamanho, inicioY + i \* tamanho);

}

// Linhas verticais

for (int i = 1; i < 3; i++) {

line(inicioX + i \* tamanho, inicioY, inicioX + i \* tamanho, inicioY + 3 \* tamanho);

}

// Desenha os X's e O's

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

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

float x = inicioX + j \* tamanho;

float y = inicioY + i \* tamanho;

if (tabuleiro[i][j] == 1) {

desenharX(x, y, tamanho);

} else if (tabuleiro[i][j] == 2) {

desenharO(x, y, tamanho);

}

}

}

}

void desenharX(float x, float y, float tamanho) {

stroke(corX);

strokeWeight(8);

float margem = tamanho \* 0.2;

line(x + margem, y + margem, x + tamanho - margem, y + tamanho - margem);

line(x + margem, y + tamanho - margem, x + tamanho - margem, y + margem);

}

void desenharO(float x, float y, float tamanho) {

stroke(corO);

strokeWeight(8);

noFill();

float margem = tamanho \* 0.2;

ellipse(x + tamanho/2, y + tamanho/2, tamanho - 2\*margem, tamanho - 2\*margem);

}

void desenharBotaoReiniciar() {

float x = width/2 - 100;

float y = height - 100;

float largura = 200;

float altura = 50;

// Desenha o botão

fill(0); // Fundo preto

stroke(255);

strokeWeight(2);

rect(x, y, largura, altura, 10);

// Desenha o texto

fill(255); // Texto branco

textAlign(CENTER);

textSize(24);

text("Reiniciar Jogo", width/2, y + 32);

}

void mousePressed() {

if (menuAtivo) {

float x = width/2 - 150;

float y = 200;

float largura = 300;

float altura = 80;

// Verifica clique no botão de 2 jogadores

if (mouseX >= x && mouseX <= x + largura &&

mouseY >= y && mouseY <= y + altura) {

modoComputador = false;

menuAtivo = false;

mensagem = "Vez do jogador X";

}

// Verifica clique no botão de modo computador

y += 100;

if (mouseX >= x && mouseX <= x + largura &&

mouseY >= y && mouseY <= y + altura) {

modoComputador = true;

menuAtivo = false;

mensagem = "Vez do jogador X";

}

return;

}

if (jogoTerminado) {

// Verifica se clicou no botão de reiniciar

float x = width/2 - 100;

float y = height - 100;

if (mouseX >= x && mouseX <= x + 200 && mouseY >= y && mouseY <= y + 50) {

iniciarJogo();

}

return;

}

// Se for modo computador e for a vez do O, não permite jogada

if (modoComputador && !vezDoX) {

return;

}

// Calcula a posição do clique no tabuleiro

float tamanho = 150;

float inicioX = (width - tamanho \* 3) / 2;

float inicioY = 150;

int coluna = int((mouseX - inicioX) / tamanho);

int linha = int((mouseY - inicioY) / tamanho);

// Verifica se o clique foi dentro do tabuleiro

if (linha >= 0 && linha < 3 && coluna >= 0 && coluna < 3) {

// Verifica se a posição está vazia

if (tabuleiro[linha][coluna] == 0) {

fazerJogada(linha, coluna);

}

}

}

void fazerJogada(int linha, int coluna) {

// Faz a jogada

tabuleiro[linha][coluna] = vezDoX ? 1 : 2;

// Verifica se o jogo terminou

if (verificarVitoria()) {

jogoTerminado = true;

mensagem = (vezDoX ? "X" : "O") + " venceu!";

} else if (verificarEmpate()) {

jogoTerminado = true;

mensagem = "Empate!";

} else {

// Passa a vez

vezDoX = !vezDoX;

mensagem = "Vez do jogador " + (vezDoX ? "X" : "O");

}

}

void fazerJogadaComputador() {

// Procura uma jogada vencedora

int[] jogada = encontrarJogadaVencedora(2);

if (jogada != null) {

fazerJogada(jogada[0], jogada[1]);

return;

}

// Bloqueia jogada vencedora do oponente

jogada = encontrarJogadaVencedora(1);

if (jogada != null) {

fazerJogada(jogada[0], jogada[1]);

return;

}

// Tenta jogar no centro

if (tabuleiro[1][1] == 0) {

fazerJogada(1, 1);

return;

}

// Joga em uma posição aleatória

ArrayList<int[]> posicoesVazias = new ArrayList<int[]>();

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

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

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

posicoesVazias.add(new int[]{i, j});

}

}

}

if (!posicoesVazias.isEmpty()) {

int[] posicao = posicoesVazias.get(int(random(posicoesVazias.size())));

fazerJogada(posicao[0], posicao[1]);

}

}

int[] encontrarJogadaVencedora(int jogador) {

// Verifica linhas

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

int[] jogada = verificarLinha(i, jogador);

if (jogada != null) return jogada;

}

// Verifica colunas

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

int[] jogada = verificarColuna(j, jogador);

if (jogada != null) return jogada;

}

// Verifica diagonais

int[] jogada = verificarDiagonal(jogador);

if (jogada != null) return jogada;

return null;

}

int[] verificarLinha(int linha, int jogador) {

int vazios = 0;

int[] posicaoVazia = null;

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

if (tabuleiro[linha][j] == jogador) {

vazios++;

} else if (tabuleiro[linha][j] == 0) {

posicaoVazia = new int[]{linha, j};

}

}

if (vazios == 2 && posicaoVazia != null) {

return posicaoVazia;

}

return null;

}

int[] verificarColuna(int coluna, int jogador) {

int vazios = 0;

int[] posicaoVazia = null;

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

if (tabuleiro[i][coluna] == jogador) {

vazios++;

} else if (tabuleiro[i][coluna] == 0) {

posicaoVazia = new int[]{i, coluna};

}

}

if (vazios == 2 && posicaoVazia != null) {

return posicaoVazia;

}

return null;

}

int[] verificarDiagonal(int jogador) {

// Diagonal principal

int vazios = 0;

int[] posicaoVazia = null;

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

if (tabuleiro[i][i] == jogador) {

vazios++;

} else if (tabuleiro[i][i] == 0) {

posicaoVazia = new int[]{i, i};

}

}

if (vazios == 2 && posicaoVazia != null) {

return posicaoVazia;

}

// Diagonal secundária

vazios = 0;

posicaoVazia = null;

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

if (tabuleiro[i][2-i] == jogador) {

vazios++;

} else if (tabuleiro[i][2-i] == 0) {

posicaoVazia = new int[]{i, 2-i};

}

}

if (vazios == 2 && posicaoVazia != null) {

return posicaoVazia;

}

return null;

}

boolean verificarVitoria() {

// Verifica linhas

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

if (tabuleiro[i][0] != 0 &&

tabuleiro[i][0] == tabuleiro[i][1] &&

tabuleiro[i][1] == tabuleiro[i][2]) {

return true;

}

}

// Verifica colunas

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

if (tabuleiro[0][j] != 0 &&

tabuleiro[0][j] == tabuleiro[1][j] &&

tabuleiro[1][j] == tabuleiro[2][j]) {

return true;

}

}

// Verifica diagonais

if (tabuleiro[0][0] != 0 &&

tabuleiro[0][0] == tabuleiro[1][1] &&

tabuleiro[1][1] == tabuleiro[2][2]) {

return true;

}

if (tabuleiro[0][2] != 0 &&

tabuleiro[0][2] == tabuleiro[1][1] &&

tabuleiro[1][1] == tabuleiro[2][0]) {

return true;

}

return false;

}

boolean verificarEmpate() {

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

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

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

return false;

}

}

}

return true;

})

Código fonte do jogo 3: (// Variáveis globais

String[][] palavras = {

// Nível Fácil (palavras curtas e comuns)

{

"CASA", "BOLA", "GATO", "CACHORRO", "LIVRO",

"MESA", "CADEIRA", "JANELA", "PORTA", "ÁRVORE",

"FLOR", "SOL", "LUA", "MAR", "RIO"

},

// Nível Médio (palavras médias e um pouco mais complexas)

{

"COMPUTADOR", "INTERNET", "PROGRAMACAO", "ALGORITMO", "DESENVOLVIMENTO",

"APLICACAO", "SOFTWARE", "HARDWARE", "SISTEMA", "REDE",

"DADOS", "INFORMATICA", "TECNOLOGIA", "PROCESSAMENTO", "INFORMACAO"

},

// Nível Difícil (palavras longas e mais complexas)

{

"DESENVOLVIMENTO", "PROGRAMACAO", "APLICACAO", "COMPUTADOR", "ALGORITMO",

"INTERFACE", "SOFTWARE", "HARDWARE", "SISTEMA", "REDE",

"DADOS", "INFORMATICA", "TECNOLOGIA", "PROCESSAMENTO", "INFORMACAO"

}

};

String palavra; // Palavra a ser adivinhada

String palavraAtual = ""; // Palavra com as letras descobertas

ArrayList<Character> letrasUsadas = new ArrayList<Character>();

ArrayList<Character> letrasErradas = new ArrayList<Character>();

int vidas = 6; // Número de vidas

boolean jogoTerminado = false;

String mensagem = "Escolha um nível para começar!";

PFont fonte;

color corForca = color(139, 69, 19); // Marrom

color corLetra = color(255, 255, 255); // Branco

color corLetraErrada = color(255, 0, 0); // Vermelho

color corLetraCerta = color(0, 255, 0); // Verde

int nivelAtual = -1; // -1 significa que nenhum nível foi escolhido

void setup() {

size(800, 600);

fonte = createFont("Arial", 32);

textFont(fonte);

}

void iniciarJogo() {

// Escolhe uma palavra aleatória do nível atual

palavra = palavras[nivelAtual][int(random(palavras[nivelAtual].length))];

// Inicializa a palavra atual com underscores

palavraAtual = "";

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

palavraAtual += "\_";

}

// Limpa as listas de letras

letrasUsadas.clear();

letrasErradas.clear();

// Reseta as vidas e o estado do jogo

vidas = 6;

jogoTerminado = false;

mensagem = "Digite uma letra para começar!";

}

void draw() {

// Desenha o fundo com gradiente

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

float inter = map(i, 0, height, 0, 1);

color c = lerpColor(color(50, 50, 50), color(30, 30, 30), inter);

stroke(c);

line(0, i, width, i);

}

// Desenha o painel de informações com efeito de sombra

fill(0, 0, 0, 200);

noStroke();

rect(0, 0, width, 100);

// Desenha a mensagem com sombra

fill(0, 0, 0, 100);

textAlign(CENTER);

textSize(24);

text(mensagem, width/2 + 2, 42);

fill(255);

text(mensagem, width/2, 40);

// Desenha as vidas restantes com sombra

fill(0, 0, 0, 100);

text("Vidas: " + vidas, width/2 + 2, 72);

fill(255);

text("Vidas: " + vidas, width/2, 70);

// Desenha o botão de engrenagem (sempre visível durante o jogo)

if (nivelAtual != -1) {

desenharBotaoEngrenagem();

}

// Se nenhum nível foi escolhido, desenha os botões de nível

if (nivelAtual == -1) {

desenharBotoesNivel();

} else {

// Desenha a forca

desenharForca();

// Desenha a palavra

desenharPalavra();

// Desenha as letras usadas

desenharLetrasUsadas();

// Se o jogo terminou, desenha o botão de reiniciar

if (jogoTerminado) {

desenharBotaoReiniciar();

}

}

}

void desenharBotoesNivel() {

float y = height/2 - 100;

float largura = 200;

float altura = 50;

float espaco = 20;

// Botão Fácil

float x = width/2 - largura/2;

if (mouseX >= x && mouseX <= x + largura && mouseY >= y && mouseY <= y + altura) {

fill(50);

} else {

fill(0);

}

stroke(255);

strokeWeight(2);

rect(x, y, largura, altura, 10);

// Botão Médio

if (mouseX >= x && mouseX <= x + largura && mouseY >= y + altura + espaco && mouseY <= y + altura \* 2 + espaco) {

fill(50);

} else {

fill(0);

}

rect(x, y + altura + espaco, largura, altura, 10);

// Botão Difícil

if (mouseX >= x && mouseX <= x + largura && mouseY >= y + (altura + espaco) \* 2 && mouseY <= y + (altura + espaco) \* 3) {

fill(50);

} else {

fill(0);

}

rect(x, y + (altura + espaco) \* 2, largura, altura, 10);

// Textos dos botões

fill(255);

textAlign(CENTER);

textSize(24);

text("Nível Fácil", width/2, y + 32);

text("Nível Médio", width/2, y + altura + espaco + 32);

text("Nível Difícil", width/2, y + (altura + espaco) \* 2 + 32);

}

void desenharForca() {

float x = 150; // Movido mais para a esquerda

float y = 400;

stroke(corForca);

strokeWeight(4);

// Base da forca

line(x - 50, y, x + 50, y);

// Poste vertical

line(x, y, x, y - 300);

// Topo da forca

line(x, y - 300, x + 150, y - 300);

// Corda

line(x + 150, y - 300, x + 150, y - 250);

// Desenha o boneco se houver erros

if (letrasErradas.size() > 0) {

// Cabeça

noFill();

stroke(corLetra);

strokeWeight(3);

ellipse(x + 150, y - 220, 40, 40);

if (letrasErradas.size() > 1) {

// Corpo

line(x + 150, y - 200, x + 150, y - 100);

}

if (letrasErradas.size() > 2) {

// Braço esquerdo

line(x + 150, y - 180, x + 100, y - 140);

}

if (letrasErradas.size() > 3) {

// Braço direito

line(x + 150, y - 180, x + 200, y - 140);

}

if (letrasErradas.size() > 4) {

// Perna esquerda

line(x + 150, y - 100, x + 120, y - 40);

}

if (letrasErradas.size() > 5) {

// Perna direita

line(x + 150, y - 100, x + 180, y - 40);

}

}

}

void desenharPalavra() {

float x = width/2 - (palavra.length() \* 20);

float y = 200;

textAlign(LEFT);

textSize(40);

// Adiciona sombra ao texto

fill(0, 0, 0, 100);

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

text("\_", x + i \* 40 + 2, y + 2);

}

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

char letra = palavraAtual.charAt(i);

if (letra == '\_') {

fill(255);

text("\_", x + i \* 40, y);

} else {

fill(corLetraCerta);

text(letra, x + i \* 40, y);

}

}

}

void desenharLetrasUsadas() {

float x = 50;

float y = 500;

textAlign(LEFT);

textSize(20);

// Letras erradas com fundo

fill(0, 0, 0, 150);

noStroke();

rect(x - 10, y - 25, 300, 30, 10);

fill(corLetraErrada);

text("Letras erradas: ", x, y);

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

text(letrasErradas.get(i), x + 150 + i \* 20, y);

}

// Letras usadas com fundo

fill(0, 0, 0, 150);

noStroke();

rect(x - 10, y + 5, 300, 30, 10);

fill(255);

text("Letras usadas: ", x, y + 30);

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

text(letrasUsadas.get(i), x + 150 + i \* 20, y + 30);

}

}

void desenharBotaoEngrenagem() {

float engrenagemX = width - 60;

float engrenagemY = 30;

float engrenagemTamanho = 40;

// Efeito de hover na engrenagem

if (mouseX >= engrenagemX - engrenagemTamanho/2 &&

mouseX <= engrenagemX + engrenagemTamanho/2 &&

mouseY >= engrenagemY - engrenagemTamanho/2 &&

mouseY <= engrenagemY + engrenagemTamanho/2) {

fill(50);

} else {

fill(0);

}

stroke(255);

strokeWeight(2);

ellipse(engrenagemX, engrenagemY, engrenagemTamanho, engrenagemTamanho);

// Desenha os dentes da engrenagem

for (int i = 0; i < 8; i++) {

float angulo = i \* PI / 4;

float x1 = engrenagemX + cos(angulo) \* engrenagemTamanho/2;

float y1 = engrenagemY + sin(angulo) \* engrenagemTamanho/2;

float x2 = engrenagemX + cos(angulo) \* (engrenagemTamanho/2 + 10);

float y2 = engrenagemY + sin(angulo) \* (engrenagemTamanho/2 + 10);

line(x1, y1, x2, y2);

}

// Desenha o centro da engrenagem

fill(0);

ellipse(engrenagemX, engrenagemY, engrenagemTamanho/3, engrenagemTamanho/3);

}

void desenharBotaoReiniciar() {

float x = width/2 - 100;

float y = height - 100;

float largura = 200;

float altura = 50;

// Efeito de hover

if (mouseX >= x && mouseX <= x + largura && mouseY >= y && mouseY <= y + altura) {

fill(50);

} else {

fill(0);

}

stroke(255);

strokeWeight(2);

rect(x, y, largura, altura, 10);

// Desenha o texto com sombra

fill(0, 0, 0, 100);

textAlign(CENTER);

textSize(24);

text("Jogar Novamente", width/2 + 2, y + 32);

fill(255);

text("Jogar Novamente", width/2, y + 30);

}

void mousePressed() {

if (nivelAtual == -1) {

float y = height/2 - 100;

float largura = 200;

float altura = 50;

float espaco = 20;

float x = width/2 - largura/2;

// Verifica clique no botão Fácil

if (mouseX >= x && mouseX <= x + largura && mouseY >= y && mouseY <= y + altura) {

nivelAtual = 0;

iniciarJogo();

}

// Verifica clique no botão Médio

else if (mouseX >= x && mouseX <= x + largura && mouseY >= y + altura + espaco && mouseY <= y + altura \* 2 + espaco) {

nivelAtual = 1;

iniciarJogo();

}

// Verifica clique no botão Difícil

else if (mouseX >= x && mouseX <= x + largura && mouseY >= y + (altura + espaco) \* 2 && mouseY <= y + (altura + espaco) \* 3) {

nivelAtual = 2;

iniciarJogo();

}

} else {

// Verifica clique no botão de engrenagem (sempre visível durante o jogo)

float engrenagemX = width - 60;

float engrenagemY = 30;

float engrenagemTamanho = 40;

if (mouseX >= engrenagemX - engrenagemTamanho/2 &&

mouseX <= engrenagemX + engrenagemTamanho/2 &&

mouseY >= engrenagemY - engrenagemTamanho/2 &&

mouseY <= engrenagemY + engrenagemTamanho/2) {

nivelAtual = -1; // Volta para a seleção de nível

mensagem = "Escolha um nível para começar!";

}

// Verifica clique no botão Jogar Novamente (apenas quando o jogo terminou)

else if (jogoTerminado) {

float x = width/2 - 100;

float y = height - 100;

if (mouseX >= x && mouseX <= x + 200 && mouseY >= y && mouseY <= y + 50) {

iniciarJogo();

}

}

}

}

void keyPressed() {

if (jogoTerminado) {

return;

}

// Converte a tecla pressionada para maiúscula

char letra = Character.toUpperCase(key);

// Verifica se a tecla pressionada é uma letra

if (letra >= 'A' && letra <= 'Z') {

// Verifica se a letra já foi usada

if (letrasUsadas.contains(letra)) {

mensagem = "Esta letra já foi usada!";

return;

}

// Adiciona a letra à lista de letras usadas

letrasUsadas.add(letra);

// Verifica se a letra está na palavra

boolean acertou = false;

String novaPalavra = "";

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

if (palavra.charAt(i) == letra) {

novaPalavra += letra;

acertou = true;

} else {

novaPalavra += palavraAtual.charAt(i);

}

}

if (acertou) {

palavraAtual = novaPalavra;

mensagem = "Acertou! Continue tentando!";

// Verifica se o jogador venceu

if (palavraAtual.equals(palavra)) {

jogoTerminado = true;

mensagem = "Parabéns! Você venceu!";

}

} else {

letrasErradas.add(letra);

vidas--;

mensagem = "Errou! Tente novamente!";

// Verifica se o jogador perdeu

if (vidas <= 0) {

jogoTerminado = true;

mensagem = "Game Over! A palavra era: " + palavra;

}

}

} else {

// Se não for uma letra, mostra mensagem de ajuda

mensagem = "Digite uma letra (A-Z) para jogar!";

}

})

Código fonte do jogo 4: (// Variáveis globais

float paddleHeight = 150;

float paddleWidth = 20;

float ballSize = 20;

float ballSpeedX = 4;

float ballSpeedY = 4;

float maxBallSpeed = 12;

float speedIncrease = 0.1;

// Constantes do jogo

final int SCORE\_LIMIT = 8;

final int TIME\_ATTACK\_DURATION = 120; // 2 minutos em segundos

final int SURVIVAL\_SPEED\_INCREASE\_INTERVAL = 30; // Aumenta a velocidade a cada 30 segundos

// Posições dos elementos

float leftPaddleY;

float rightPaddleY;

float ballX;

float ballY;

// Velocidades das raquetes

float leftPaddleSpeed = 0;

float rightPaddleSpeed = 0;

float paddleSpeed = 12;

float paddleAcceleration = 0.8;

float paddleFriction = 0.8;

// IA da raquete direita

float aiReactionTime = 1.5;

float aiPrediction = 0.4;

float aiError = 0.5;

float aiDelay = 0.3;

float aiTargetY = 0;

float aiLastUpdate = 0;

// Níveis de dificuldade

int currentLevel = 1;

int maxLevel = 5;

float[] levelReactionTimes = {2.0, 1.5, 1.2, 0.8, 0.5}; // Tempos de reação para cada nível

float[] levelPredictions = {0.3, 0.4, 0.5, 0.7, 0.9}; // Previsão para cada nível

// Pontuação

int leftScore = 0;

int rightScore = 0;

// Estados do jogo

boolean gameStarted = false;

boolean gameOver = false;

String winner = "";

boolean settingsOpen = false;

int gameMode = 0; // 0: Clássico, 1: Contra o Tempo, 2: Sobrevivência

int timeLeft = TIME\_ATTACK\_DURATION;

int survivalTime = 0;

int lastSpeedIncrease = 0;

// Efeitos visuais

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

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

color[] colors = {#FF0000, #00FF00, #0000FF, #FFFF00, #FF00FF, #00FFFF};

color leftPaddleColor = #FFFFFF;

color rightPaddleColor = #FFFFFF;

// Configurações da engrenagem

float gearX = 750;

float gearY = 50;

float gearSize = 30;

float gearRotation = 0;

// Variáveis de Power-Ups

class PowerUp {

float x, y;

int type;

float duration;

PowerUp(float x, float y, int type) {

this.x = x;

this.y = y;

this.type = type;

this.duration = 10; // 10 segundos de duração

}

void draw() {

noStroke();

switch(type) {

case 0: // Aumentar raquete

fill(0, 255, 0, 200);

break;

case 1: // Bola lenta

fill(0, 0, 255, 200);

break;

case 2: // Escudo

fill(255, 255, 0, 200);

break;

case 3: // Congelar IA

fill(255, 0, 0, 200);

break;

}

ellipse(x, y, 30, 30);

}

}

ArrayList<PowerUp> powerUps = new ArrayList<PowerUp>();

float powerUpSpawnTimer = 0;

float powerUpSpawnInterval = 15; // Spawn a cada 15 segundos

// Variáveis de efeito de power-up

boolean hasPaddleSizeBoost = false;

float paddleSizeBoostTimer = 0;

boolean hasSlowBall = false;

float slowBallTimer = 0;

boolean hasShield = false;

float shieldTimer = 0;

boolean freezeAI = false;

float freezeAITimer = 0;

void setup() {

size(800, 600);

// Inicializa as posições

leftPaddleY = height/2 - paddleHeight/2;

rightPaddleY = height/2 - paddleHeight/2;

ballX = width/2;

ballY = height/2;

// Define a velocidade inicial da bola

resetBall();

}

void draw() {

background(0);

// Desenha a linha central pontilhada

stroke(255, 100);

for (float y = 0; y < height; y += 30) {

line(width/2, y, width/2, y + 15);

}

// Atualiza as posições das raquetes

if (gameStarted && !gameOver && !settingsOpen) {

// Atualiza velocidade da raquete esquerda

if (keyPressed) {

if (key == 'w') {

leftPaddleSpeed -= paddleAcceleration;

}

if (key == 's') {

leftPaddleSpeed += paddleAcceleration;

}

}

// Verifica se a IA está congelada

if (!freezeAI) {

// IA da raquete direita com atraso e erro

float currentTime = millis() / 1000.0;

if (currentTime - aiLastUpdate > aiDelay) {

aiTargetY = predictBallPosition();

aiLastUpdate = currentTime;

}

float currentY = rightPaddleY + paddleHeight/2;

float error = random(-aiError, aiError) \* paddleHeight;

if (abs(aiTargetY + error - currentY) > 5) {

if (aiTargetY + error > currentY) {

rightPaddleSpeed = paddleSpeed \* 0.6;

} else {

rightPaddleSpeed = -paddleSpeed \* 0.6;

}

} else {

rightPaddleSpeed \*= paddleFriction;

}

}

// Aplica fricção

leftPaddleSpeed \*= paddleFriction;

rightPaddleSpeed \*= paddleFriction;

// Limita a velocidade máxima

leftPaddleSpeed = constrain(leftPaddleSpeed, -paddleSpeed, paddleSpeed);

rightPaddleSpeed = constrain(rightPaddleSpeed, -paddleSpeed, paddleSpeed);

// Atualiza posições

leftPaddleY += leftPaddleSpeed;

rightPaddleY += rightPaddleSpeed;

// Limita as posições das raquetes

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

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

// Atualiza o modo de jogo

updateGameMode();

}

// Desenha as raquetes com cores personalizadas

noStroke();

fill(leftPaddleColor);

rect(0, leftPaddleY, paddleWidth, paddleHeight);

fill(rightPaddleColor);

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

// Desenha o rastro da bola

drawBallTrail();

// Desenha a bola

fill(255);

ellipse(ballX, ballY, ballSize, ballSize);

// Desenha a pontuação e informações do modo de jogo

textSize(32);

textAlign(CENTER);

fill(255);

text(leftScore, width/4, 50);

text(rightScore, 3\*width/4, 50);

drawGameModeInfo();

// Desenha a engrenagem

drawGear();

// Desenha as partículas

for (int i = particles.size() - 1; i >= 0; i--) {

PVector p = particles.get(i);

fill(colors[i % colors.length], 150);

ellipse(p.x, p.y, 5, 5);

p.x += random(-2, 2);

p.y += random(-2, 2);

p.z -= 0.1;

if (p.z <= 0) particles.remove(i);

}

// Atualiza power-ups

updatePowerUps();

// Desenha power-ups

drawPowerUps();

if (gameStarted && !gameOver) {

if (!settingsOpen) {

// Atualiza a posição da bola

ballX += ballSpeedX;

ballY += ballSpeedY;

// Verifica colisão com as bordas superior e inferior

if (ballY < ballSize/2 || ballY > height - ballSize/2) {

ballSpeedY \*= -1;

createParticles(ballX, ballY);

}

// Verifica colisão com as raquetes

if (ballX < paddleWidth + ballSize/2 && ballY > leftPaddleY && ballY < leftPaddleY + paddleHeight) {

float hitPoint = (ballY - leftPaddleY) / paddleHeight;

float angle = map(hitPoint, 0, 1, -PI/4, PI/4);

float speed = sqrt(ballSpeedX \* ballSpeedX + ballSpeedY \* ballSpeedY);

speed = min(speed \* 1.1, maxBallSpeed);

ballSpeedX = speed \* cos(angle);

ballSpeedY = speed \* sin(angle);

createParticles(ballX, ballY);

}

if (ballX > width - paddleWidth - ballSize/2 && ballY > rightPaddleY && ballY < rightPaddleY + paddleHeight) {

float hitPoint = (ballY - rightPaddleY) / paddleHeight;

float angle = map(hitPoint, 0, 1, -PI/4, PI/4);

float speed = sqrt(ballSpeedX \* ballSpeedX + ballSpeedY \* ballSpeedY);

speed = min(speed \* 1.1, maxBallSpeed);

ballSpeedX = -speed \* cos(angle);

ballSpeedY = speed \* sin(angle);

createParticles(ballX, ballY);

}

// Verifica se a bola passou das raquetes

if (ballX < 0) {

rightScore++;

resetBall();

}

if (ballX > width) {

leftScore++;

resetBall();

}

// Verifica vitória

if (gameMode == 0 && (leftScore >= SCORE\_LIMIT || rightScore >= SCORE\_LIMIT)) {

gameOver = true;

winner = leftScore >= SCORE\_LIMIT ? "Jogador Esquerdo" : "Computador";

}

} else {

drawSettingsMenu();

}

} else if (!gameStarted) {

// Menu principal

textSize(48);

textAlign(CENTER);

fill(255);

text("PONG ARCADE", width/2, height/2 - 50);

textSize(24);

text("Modo Atual: " + (gameMode == 0 ? "Clássico" : gameMode == 1 ? "Contra o Tempo" : "Sobrevivência"), width/2, height/2);

text("Pressione ESPAÇO para começar", width/2, height/2 + 50);

text("Pressione C para configurações", width/2, height/2 + 80);

} else if (gameOver) {

// Tela de game over

textSize(48);

textAlign(CENTER);

fill(255);

text("GAME OVER", width/2, height/2 - 50);

textSize(24);

text(winner + " venceu!", width/2, height/2 + 20);

text("Pontuação: " + leftScore + " - " + rightScore, width/2, height/2 + 50);

text("Pressione ESPAÇO para jogar novamente", width/2, height/2 + 90);

}

}

void drawGear() {

pushMatrix();

translate(gearX, gearY);

rotate(gearRotation);

// Desenha a engrenagem

stroke(255, 200);

strokeWeight(2);

fill(255, 100);

// Círculo central

ellipse(0, 0, gearSize, gearSize);

// Dentes da engrenagem

for (int i = 0; i < 8; i++) {

float angle = i \* PI / 4;

float x = cos(angle) \* gearSize/2;

float y = sin(angle) \* gearSize/2;

rect(x - 5, y - 10, 10, 20);

}

popMatrix();

// Atualiza a rotação

gearRotation += 0.02;

}

void drawSettingsMenu() {

// Fundo semi-transparente

fill(0, 200);

rect(0, 0, width, height);

// Título

textSize(48);

textAlign(CENTER);

fill(255);

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

// Níveis

textSize(24);

for (int i = 1; i <= maxLevel; i++) {

float x = width/2 - 100 + (i-1) \* 50;

float y = height/2;

// Botão do nível

if (i == currentLevel) {

fill(255, 200);

ellipse(x, y, 40, 40);

fill(0);

} else {

fill(255, 100);

ellipse(x, y, 40, 40);

fill(255);

}

text(i, x, y + 8);

}

// Instruções

textSize(20);

fill(255, 200);

text("Clique em um número para mudar o nível", width/2, height/2 + 50);

text("Pressione ESC para voltar", width/2, height/2 + 80);

}

void mousePressed() {

if (gameStarted && !gameOver) {

// Verifica clique na engrenagem

float d = dist(mouseX, mouseY, gearX, gearY);

if (d < gearSize/2) {

settingsOpen = !settingsOpen;

}

// Verifica clique nos níveis no menu de configurações

if (settingsOpen) {

for (int i = 1; i <= maxLevel; i++) {

float x = width/2 - 100 + (i-1) \* 50;

float y = height/2;

float d2 = dist(mouseX, mouseY, x, y);

if (d2 < 20) {

currentLevel = i;

}

}

}

}

}

float predictBallPosition() {

// Calcula onde a bola estará quando chegar à raquete direita

float timeToReach = (width - paddleWidth - ballX) / ballSpeedX;

float predictedY = ballY + ballSpeedY \* timeToReach \* aiPrediction;

// Faz a bola quicar nas bordas superior e inferior

while (predictedY < 0 || predictedY > height) {

if (predictedY < 0) {

predictedY = -predictedY;

}

if (predictedY > height) {

predictedY = 2 \* height - predictedY;

}

}

// Adiciona um pequeno erro aleatório na previsão

predictedY += random(-paddleHeight/4, paddleHeight/4);

return predictedY;

}

void resetBall() {

ballX = width/2;

ballY = height/2;

// Reduz a velocidade inicial

float speed = random(4, 6);

float angle = random(-PI/6, PI/6);

ballSpeedX = speed \* cos(angle);

ballSpeedY = speed \* sin(angle);

}

void createParticles(float x, float y) {

for (int i = 0; i < 10; i++) {

particles.add(new PVector(x, y, 1));

}

}

void keyPressed() {

if (key == ' ') {

if (gameOver) {

resetGame();

} else if (!gameStarted) {

gameStarted = true;

} else if (settingsOpen) {

settingsOpen = false;

}

}

if (key == '1' || key == '2' || key == '3') {

gameMode = int(key) - 49; // Converte char para int (1->0, 2->1, 3->2)

}

if (key == 'q') {

leftPaddleColor = color(random(255), random(255), random(255));

}

if (key == 'e') {

rightPaddleColor = color(random(255), random(255), random(255));

}

if (key == 'c') {

settingsOpen = !settingsOpen;

}

}

void updateGameMode() {

if (gameMode == 1) { // Contra o Tempo

if (frameCount % 60 == 0) { // Atualiza a cada segundo

timeLeft--;

if (timeLeft <= 0) {

gameOver = true;

winner = leftScore > rightScore ? "Jogador Esquerdo" : "Computador";

}

}

} else if (gameMode == 2) { // Sobrevivência

survivalTime++;

if (survivalTime - lastSpeedIncrease >= SURVIVAL\_SPEED\_INCREASE\_INTERVAL \* 60) {

ballSpeedX \*= 1.1;

ballSpeedY \*= 1.1;

lastSpeedIncrease = survivalTime;

}

}

}

void drawBallTrail() {

// Adiciona a posição atual da bola ao rastro

ballTrail.add(new PVector(ballX, ballY));

// Limita o tamanho do rastro

if (ballTrail.size() > 20) {

ballTrail.remove(0);

}

// Desenha o rastro

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

float alpha = map(i, 0, ballTrail.size(), 50, 255);

fill(255, alpha);

noStroke();

ellipse(ballTrail.get(i).x, ballTrail.get(i).y, ballSize \* 0.8, ballSize \* 0.8);

}

}

void drawGameModeInfo() {

textSize(16);

textAlign(LEFT);

fill(255);

if (gameMode == 1) { // Contra o Tempo

int minutes = timeLeft / 60;

int seconds = timeLeft % 60;

text("Tempo: " + minutes + ":" + (seconds < 10 ? "0" : "") + seconds, 10, 30);

} else if (gameMode == 2) { // Sobrevivência

text("Velocidade: " + nf(sqrt(ballSpeedX \* ballSpeedX + ballSpeedY \* ballSpeedY), 1, 1), 10, 30);

}

}

void resetGame() {

leftScore = 0;

rightScore = 0;

gameOver = false;

winner = "";

timeLeft = TIME\_ATTACK\_DURATION;

survivalTime = 0;

lastSpeedIncrease = 0;

ballSpeedX = 4;

ballSpeedY = 4;

ballTrail.clear();

resetBall();

}

void updatePowerUps() {

// Spawn de power-ups

powerUpSpawnTimer += 1.0/60; // Assume 60 FPS

if (powerUpSpawnTimer >= powerUpSpawnInterval) {

// Chance de spawnar um power-up

if (random(1) < 0.7) {

float x = random(width/4, 3\*width/4);

float y = random(height/4, 3\*height/4);

powerUps.add(new PowerUp(x, y, int(random(4))));

}

powerUpSpawnTimer = 0;

}

// Verificar colisão com power-ups

for (int i = powerUps.size() - 1; i >= 0; i--) {

PowerUp pu = powerUps.get(i);

// Verifica colisão com jogador

if (dist(ballX, ballY, pu.x, pu.y) < (ballSize + 30)/2) {

activatePowerUp(pu.type);

powerUps.remove(i);

}

}

// Gerenciar duração dos power-ups

if (hasPaddleSizeBoost) {

paddleSizeBoostTimer -= 1.0/60;

if (paddleSizeBoostTimer <= 0) {

hasPaddleSizeBoost = false;

paddleHeight = 150; // Volta ao tamanho original

}

}

if (hasSlowBall) {

slowBallTimer -= 1.0/60;

if (slowBallTimer <= 0) {

hasSlowBall = false;

maxBallSpeed = 12; // Volta à velocidade original

}

}

if (hasShield) {

shieldTimer -= 1.0/60;

if (shieldTimer <= 0) {

hasShield = false;

}

}

if (freezeAI) {

freezeAITimer -= 1.0/60;

if (freezeAITimer <= 0) {

freezeAI = false;

}

}

}

void activatePowerUp(int type) {

switch(type) {

case 0: // Aumentar raquete

hasPaddleSizeBoost = true;

paddleSizeBoostTimer = 10;

paddleHeight = 250; // Aumenta significativamente

break;

case 1: // Bola lenta

hasSlowBall = true;

slowBallTimer = 10;

maxBallSpeed = 6; // Reduz muito a velocidade

break;

case 2: // Escudo

hasShield = true;

shieldTimer = 10;

break;

case 3: // Congelar IA

freezeAI = true;

freezeAITimer = 10;

break;

}

}

void drawPowerUps() {

// Desenha power-ups ativos

for (PowerUp pu : powerUps) {

pu.draw();

}

// Desenha efeitos de power-ups

if (hasShield) {

noFill();

stroke(255, 255, 0, 100);

strokeWeight(5);

ellipse(width/2, height/2, width, height);

}

}

void checkCollisions() {

// Verifica colisão com as raquetes

if (ballX < paddleWidth + ballSize/2 && ballY > leftPaddleY && ballY < leftPaddleY + paddleHeight) {

// Lógica de colisão existente

// ...

}

if (ballX > width - paddleWidth - ballSize/2 && ballY > rightPaddleY && ballY < rightPaddleY + paddleHeight) {

// Se a IA tiver escudo, não marca ponto

if (!hasShield) {

// Lógica de colisão existente

// ...

}

}

})

Código fonte do jogo 5: (// Jogo da Memória com Imagens - Versão Completa

// Desenvolvido com clean code e boas práticas

// Enums globais

enum GameState {

MENU, PREVIEW, PLAYING, GAME\_OVER, SETTINGS

}

enum Difficulty {

EASY, MEDIUM, HARD

}

class MemoryGame {

// Configurações do jogo

// Para cada dificuldade: [FÁCIL, MÉDIO, DIFÍCIL]

final int[][] GRID\_CONFIG = {

{3, 2}, // Fácil: 3x2

{4, 4}, // Médio: 4x4

{6, 4} // Difícil: 6x4

};

final int[] PREVIEW\_TIME = {300, 180, 120}; // 5s, 3s, 2s a 60 FPS

final int[] MAX\_ATTEMPTS = {20, 15, 10}; // Tentativas por dificuldade

// Tamanho das cartas adaptável por dificuldade

final int[] CARD\_SIZES = {120, 120, 90}; // Tamanho para cada dificuldade

int CARD\_SIZE; // Será definido com base na dificuldade

final int CARD\_MARGIN = 10;

// Configuração de rede - definir como false para usar apenas imagens geradas

boolean online = false;

// Animação e efeitos

final int CARD\_FLIP\_FRAMES = 8; // Duração da animação de virar carta

// Atributos do jogo

private int[] cards;

private boolean[] revealed;

private boolean[] matched;

private float[] cardFlipProgress; // 0 = fechada, 1 = aberta

private int GRID\_ROWS;

private int GRID\_COLS;

// Estados do jogo

private GameState currentState;

private Difficulty currentDifficulty = Difficulty.MEDIUM;

private int selectedCard1 = -1;

private int selectedCard2 = -1;

private int matchedPairs = 0;

private int attempts = 0;

private int maxAttempts;

// Sistema de tempo e pontuação

private int previewTimer;

private int gameTimer;

private int score;

private int[] highScores = {0, 0, 0}; // Recordes para cada dificuldade

// Recursos visuais

private PImage[] cardImages;

private PImage cardBack;

private PFont gameFont;

MemoryGame() {

// Carregar recursos

try {

gameFont = createFont("Arial", 16, true);

} catch (Exception e) {

println("Erro ao carregar fonte: " + e.getMessage());

// Usar fonte padrão se necessário

}

// Configurar tamanho inicial da grade

GRID\_ROWS = GRID\_CONFIG[currentDifficulty.ordinal()][0];

GRID\_COLS = GRID\_CONFIG[currentDifficulty.ordinal()][1];

maxAttempts = MAX\_ATTEMPTS[currentDifficulty.ordinal()];

CARD\_SIZE = CARD\_SIZES[currentDifficulty.ordinal()];

// Criar imagens do verso das cartas

cardBack = createCardBackImage();

// Iniciar no menu principal

currentState = GameState.MENU;

// Inicializar pontuações

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

highScores[i] = 0;

}

}

private void loadImages() {

println("Criando imagens para o jogo...");

println("Tamanho das cartas: " + CARD\_SIZE + "x" + CARD\_SIZE);

// Recriar a imagem do verso da carta com o tamanho correto

cardBack = createCardBackImage();

// Calcular número de pares necessários

int pairsCount = (GRID\_ROWS \* GRID\_COLS) / 2;

cardImages = new PImage[pairsCount];

// Usar apenas imagens geradas localmente

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

// Gerar imagem com formas e cores diferentes

cardImages[i] = createCardImage(i);

println("Imagem " + (i+1) + " de " + pairsCount + " criada");

}

println("Carregamento de imagens concluído!");

}

private PImage createCardBackImage() {

// Criar uma imagem para o verso da carta

PImage img = createImage(CARD\_SIZE, CARD\_SIZE, ARGB);

img.loadPixels();

// Preencher com um padrão

for (int y = 0; y < CARD\_SIZE; y++) {

for (int x = 0; x < CARD\_SIZE; x++) {

// Criar um padrão simples

if ((x / 10 + y / 10) % 2 == 0) {

img.pixels[y \* CARD\_SIZE + x] = color(80, 80, 90);

} else {

img.pixels[y \* CARD\_SIZE + x] = color(60, 60, 70);

}

// Borda da carta

if (x < 3 || y < 3 || x >= CARD\_SIZE - 3 || y >= CARD\_SIZE - 3) {

img.pixels[y \* CARD\_SIZE + x] = color(40, 40, 45);

}

}

}

img.updatePixels();

return img;

}

private PImage createCardImage(int index) {

// Criar uma imagem para a carta

PImage img = createImage(CARD\_SIZE, CARD\_SIZE, ARGB);

img.loadPixels();

// Gerar cores mais vibrantes para cada carta

color bgColor = color(240, 240, 240);

color[] colors = {

color(255, 87, 51), // Laranja vibrante

color(29, 233, 182), // Turquesa

color(255, 45, 85), // Rosa choque

color(88, 86, 214), // Roxo

color(255, 204, 0), // Amarelo

color(0, 122, 255), // Azul

color(76, 217, 100), // Verde

color(255, 149, 0), // Laranja

color(94, 53, 177), // Roxo escuro

color(255, 59, 48) // Vermelho

};

color mainColor = colors[index % colors.length];

color secondaryColor = colors[(index + 5) % colors.length];

// Preencher fundo

for (int y = 0; y < CARD\_SIZE; y++) {

for (int x = 0; x < CARD\_SIZE; x++) {

img.pixels[y \* CARD\_SIZE + x] = bgColor;

// Borda da carta

if (x < 3 || y < 3 || x >= CARD\_SIZE - 3 || y >= CARD\_SIZE - 3) {

img.pixels[y \* CARD\_SIZE + x] = color(180, 180, 180);

}

}

}

// Desenhar forma baseada no índice

switch (index % 10) {

case 0: // Círculo

drawCircle(img, mainColor);

break;

case 1: // Quadrado

drawSquare(img, mainColor);

break;

case 2: // Triângulo

drawTriangle(img, mainColor);

break;

case 3: // Estrela

drawStar(img, mainColor);

break;

case 4: // Coração

drawHeart(img, mainColor);

break;

case 5: // Losango

drawDiamond(img, mainColor);

break;

case 6: // Cruz

drawCross(img, mainColor);

break;

case 7: // Lua

drawMoon(img, mainColor);

break;

case 8: // Espiral

drawSpiral(img, mainColor, secondaryColor);

break;

case 9: // Hexágono

drawHexagon(img, mainColor);

break;

}

img.updatePixels();

return img;

}

// Funções para desenhar formas nas imagens das cartas

private void drawCircle(PImage img, color c) {

int center = CARD\_SIZE / 2;

int radius = CARD\_SIZE / 3;

for (int y = 0; y < CARD\_SIZE; y++) {

for (int x = 0; x < CARD\_SIZE; x++) {

float dist = sqrt(pow(x - center, 2) + pow(y - center, 2));

if (dist < radius) {

img.pixels[y \* CARD\_SIZE + x] = c;

}

}

}

}

private void drawSquare(PImage img, color c) {

int margin = CARD\_SIZE / 4;

for (int y = margin; y < CARD\_SIZE - margin; y++) {

for (int x = margin; x < CARD\_SIZE - margin; x++) {

img.pixels[y \* CARD\_SIZE + x] = c;

}

}

}

private void drawTriangle(PImage img, color c) {

int center = CARD\_SIZE / 2;

int height = CARD\_SIZE / 2;

for (int y = center; y < center + height; y++) {

int width = (y - center) \* 2;

for (int x = center - width / 2; x < center + width / 2; x++) {

if (x >= 0 && x < CARD\_SIZE && y >= 0 && y < CARD\_SIZE) {

img.pixels[y \* CARD\_SIZE + x] = c;

}

}

}

}

private void drawStar(PImage img, color c) {

int center = CARD\_SIZE / 2;

int radius = CARD\_SIZE / 3;

for (int angle = 0; angle < 360; angle += 5) {

float rad = radians(angle);

int r = (angle % 36 < 18) ? radius : radius / 2;

int x = center + int(cos(rad) \* r);

int y = center + int(sin(rad) \* r);

if (x >= 0 && x < CARD\_SIZE && y >= 0 && y < CARD\_SIZE) {

img.pixels[y \* CARD\_SIZE + x] = c;

// Preencher um pouco em volta para tornar mais espesso

for (int dx = -1; dx <= 1; dx++) {

for (int dy = -1; dy <= 1; dy++) {

int nx = x + dx;

int ny = y + dy;

if (nx >= 0 && nx < CARD\_SIZE && ny >= 0 && ny < CARD\_SIZE) {

img.pixels[ny \* CARD\_SIZE + nx] = c;

}

}

}

}

}

}

private void drawHeart(PImage img, color c) {

int center = CARD\_SIZE / 2;

int size = CARD\_SIZE / 3;

for (int y = 0; y < CARD\_SIZE; y++) {

for (int x = 0; x < CARD\_SIZE; x++) {

// Fórmula para desenhar um coração

float dx = abs(x - center) / float(size);

float dy = (y - center) / float(size);

if (pow(dx, 2) + pow(dy - 0.5 \* sqrt(abs(dx)), 2) < 1) {

img.pixels[y \* CARD\_SIZE + x] = c;

}

}

}

}

private void drawDiamond(PImage img, color c) {

int center = CARD\_SIZE / 2;

int size = CARD\_SIZE / 3;

for (int y = 0; y < CARD\_SIZE; y++) {

for (int x = 0; x < CARD\_SIZE; x++) {

if (abs(x - center) + abs(y - center) < size) {

img.pixels[y \* CARD\_SIZE + x] = c;

}

}

}

}

private void drawCross(PImage img, color c) {

int center = CARD\_SIZE / 2;

int size = CARD\_SIZE / 5;

for (int y = 0; y < CARD\_SIZE; y++) {

for (int x = 0; x < CARD\_SIZE; x++) {

if ((abs(x - center) < size) || (abs(y - center) < size)) {

img.pixels[y \* CARD\_SIZE + x] = c;

}

}

}

}

private void drawMoon(PImage img, color c) {

int center = CARD\_SIZE / 2;

int radiusOuter = CARD\_SIZE / 3;

int radiusInner = CARD\_SIZE / 5;

int offset = CARD\_SIZE / 10;

for (int y = 0; y < CARD\_SIZE; y++) {

for (int x = 0; x < CARD\_SIZE; x++) {

float distOuter = sqrt(pow(x - center, 2) + pow(y - center, 2));

float distInner = sqrt(pow(x - (center + offset), 2) + pow(y - center, 2));

if (distOuter < radiusOuter && distInner > radiusInner) {

img.pixels[y \* CARD\_SIZE + x] = c;

}

}

}

}

// Novos desenhos para as cartas

private void drawSpiral(PImage img, color c1, color c2) {

int center = CARD\_SIZE / 2;

float maxRadius = CARD\_SIZE \* 0.4;

for (float angle = 0; angle < 15 \* PI; angle += 0.1) {

float radius = maxRadius \* angle / (15 \* PI);

int x = center + int(cos(angle) \* radius);

int y = center + int(sin(angle) \* radius);

if (x >= 0 && x < CARD\_SIZE && y >= 0 && y < CARD\_SIZE) {

// Alternar cores

color c = (int(angle / 0.5) % 2 == 0) ? c1 : c2;

// Desenhar ponto mais espesso

for (int dx = -2; dx <= 2; dx++) {

for (int dy = -2; dy <= 2; dy++) {

int nx = x + dx;

int ny = y + dy;

if (nx >= 0 && nx < CARD\_SIZE && ny >= 0 && ny < CARD\_SIZE) {

img.pixels[ny \* CARD\_SIZE + nx] = c;

}

}

}

}

}

}

private void drawHexagon(PImage img, color c) {

int center = CARD\_SIZE / 2;

int radius = CARD\_SIZE / 3;

for (int i = 0; i < 6; i++) {

float angle1 = PI / 3 \* i;

float angle2 = PI / 3 \* (i + 1);

int x1 = center + int(cos(angle1) \* radius);

int y1 = center + int(sin(angle1) \* radius);

int x2 = center + int(cos(angle2) \* radius);

int y2 = center + int(sin(angle2) \* radius);

// Desenhar linha

drawLine(img, x1, y1, x2, y2, c, 3);

// Preencher hexágono

fillHexagon(img, center, center, radius, c);

}

}

private void drawLine(PImage img, int x1, int y1, int x2, int y2, color c, int thickness) {

// Algoritmo de Bresenham para desenhar linha

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int sx = (x1 < x2) ? 1 : -1;

int sy = (y1 < y2) ? 1 : -1;

int err = dx - dy;

while (true) {

// Desenhar ponto espesso

for (int tx = -thickness/2; tx <= thickness/2; tx++) {

for (int ty = -thickness/2; ty <= thickness/2; ty++) {

int nx = x1 + tx;

int ny = y1 + ty;

if (nx >= 0 && nx < CARD\_SIZE && ny >= 0 && ny < CARD\_SIZE) {

img.pixels[ny \* CARD\_SIZE + nx] = c;

}

}

}

if (x1 == x2 && y1 == y2) break;

int e2 = 2 \* err;

if (e2 > -dy) { err -= dy; x1 += sx; }

if (e2 < dx) { err += dx; y1 += sy; }

}

}

private void fillHexagon(PImage img, int centerX, int centerY, int radius, color c) {

// Preenchimento simples de hexágono

for (int y = centerY - radius; y <= centerY + radius; y++) {

for (int x = centerX - radius; x <= centerX + radius; x++) {

if (x >= 0 && x < CARD\_SIZE && y >= 0 && y < CARD\_SIZE) {

// Usar distância Manhattan modificada para hexágono

float dx = abs(x - centerX);

float dy = abs(y - centerY);

if (dx \* 0.5 + dy \* 0.866 < radius \* 0.75) {

img.pixels[y \* CARD\_SIZE + x] = c;

}

}

}

}

}

void initializeGame() {

// Configurar tamanho da grade baseado na dificuldade

GRID\_ROWS = GRID\_CONFIG[currentDifficulty.ordinal()][0];

GRID\_COLS = GRID\_CONFIG[currentDifficulty.ordinal()][1];

maxAttempts = MAX\_ATTEMPTS[currentDifficulty.ordinal()];

CARD\_SIZE = CARD\_SIZES[currentDifficulty.ordinal()];

// Carregar imagens de acordo com o número de pares

loadImages();

// Inicialização dos arrays

int totalCards = GRID\_ROWS \* GRID\_COLS;

cards = new int[totalCards];

revealed = new boolean[totalCards];

matched = new boolean[totalCards];

cardFlipProgress = new float[totalCards];

// Gerar pares de cartas

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

cards[i] = i / 2;

cardFlipProgress[i] = 0; // Todas as cartas começam fechadas

}

// Embaralhar cartas

for (int i = totalCards - 1; i > 0; i--) {

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

int temp = cards[i];

cards[i] = cards[j];

cards[j] = temp;

}

// Mostrar todas as cartas no início

previewTimer = PREVIEW\_TIME[currentDifficulty.ordinal()];

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

revealed[i] = true;

cardFlipProgress[i] = 1; // Cartas totalmente abertas durante o preview

}

// Resetar valores do jogo

currentState = GameState.PREVIEW;

selectedCard1 = -1;

selectedCard2 = -1;

matchedPairs = 0;

attempts = 0;

gameTimer = 0;

score = 0;

}

void update() {

if (currentState == GameState.PREVIEW) {

previewTimer--;

if (previewTimer <= 0) {

// Esconder todas as cartas após o tempo de preview

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

revealed[i] = false;

// Iniciar animação de virar todas as cartas

thread("animateAllCardsFlip");

}

currentState = GameState.PLAYING;

}

} else if (currentState == GameState.PLAYING) {

// Atualizar timer do jogo

gameTimer++;

// Atualizar animações das cartas

updateCardAnimations();

}

}

void updateCardAnimations() {

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

// Animar cartas sendo reveladas

if (revealed[i] && cardFlipProgress[i] < 1) {

cardFlipProgress[i] += 1.0 / CARD\_FLIP\_FRAMES;

if (cardFlipProgress[i] > 1) cardFlipProgress[i] = 1;

}

// Animar cartas sendo escondidas

else if (!revealed[i] && cardFlipProgress[i] > 0) {

cardFlipProgress[i] -= 1.0 / CARD\_FLIP\_FRAMES;

if (cardFlipProgress[i] < 0) cardFlipProgress[i] = 0;

}

}

}

// Função executada em thread separada para animar todas as cartas juntas

void animateAllCardsFlip() {

for (int f = 0; f < CARD\_FLIP\_FRAMES; f++) {

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

cardFlipProgress[i] = max(0, 1 - (float)f / CARD\_FLIP\_FRAMES);

}

delay(1000 / 60); // Aproximadamente 60 FPS

}

}

void display() {

switch(currentState) {

case MENU:

displayMenu();

break;

case SETTINGS:

displaySettings();

break;

case PREVIEW:

case PLAYING:

displayGameBoard();

break;

case GAME\_OVER:

displayGameOver();

break;

}

}

private void displayMenu() {

background(240);

textFont(gameFont);

textAlign(CENTER, CENTER);

// Título

textSize(48);

fill(0, 122, 255);

text("JOGO DA MEMÓRIA", width/2, height/4);

// Botões

int buttonY = height/2;

int buttonSpacing = 100;

// Botão Iniciar

displayButton("Iniciar Jogo", width/2, buttonY, 250, 60);

// Botão Configurações

displayButton("Configurações", width/2, buttonY + buttonSpacing, 250, 60);

// Mostrar dificuldade atual

textSize(20);

fill(100);

text("Dificuldade atual: " + getDifficultyName(currentDifficulty), width/2, height - 80);

// Mostrar recorde

text("Recorde: " + highScores[currentDifficulty.ordinal()], width/2, height - 50);

}

private void displaySettings() {

background(240);

textFont(gameFont);

textAlign(CENTER, CENTER);

// Título

textSize(36);

fill(0, 122, 255);

text("CONFIGURAÇÕES", width/2, height/4);

// Opções de dificuldade

int buttonY = height/2 - 50;

int buttonSpacing = 80;

// Botão de Dificuldade Fácil

if (currentDifficulty == Difficulty.EASY) fill(0, 180, 0, 100);

else fill(0, 122, 255);

rect(width/2 - 125, buttonY, 250, 60, 10);

fill(255);

textSize(24);

text("Fácil", width/2, buttonY + 30);

// Botão de Dificuldade Média

if (currentDifficulty == Difficulty.MEDIUM) fill(0, 180, 0, 100);

else fill(0, 122, 255);

rect(width/2 - 125, buttonY + buttonSpacing, 250, 60, 10);

fill(255);

text("Médio", width/2, buttonY + buttonSpacing + 30);

// Botão de Dificuldade Difícil

if (currentDifficulty == Difficulty.HARD) fill(0, 180, 0, 100);

else fill(0, 122, 255);

rect(width/2 - 125, buttonY + 2 \* buttonSpacing, 250, 60, 10);

fill(255);

text("Difícil", width/2, buttonY + 2 \* buttonSpacing + 30);

// Botão Voltar

fill(150, 150, 150);

rect(width/2 - 125, buttonY + 3 \* buttonSpacing, 250, 60, 10);

fill(255);

text("Voltar", width/2, buttonY + 3 \* buttonSpacing + 30);

}

private void displayGameBoard() {

background(240);

textFont(gameFont);

// Calcular posição inicial para centralizar o tabuleiro

float startX = (width - (GRID\_COLS \* (CARD\_SIZE + CARD\_MARGIN))) / 2;

float startY = (height - (GRID\_ROWS \* (CARD\_SIZE + CARD\_MARGIN))) / 2;

// Desenhar cartas com animação

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

int row = i / GRID\_COLS;

int col = i % GRID\_COLS;

float x = startX + col \* (CARD\_SIZE + CARD\_MARGIN);

float y = startY + row \* (CARD\_SIZE + CARD\_MARGIN);

// Efeito de escala para animação de flip

float scaleFactor = abs(cardFlipProgress[i] - 0.5) \* 2;

pushMatrix();

translate(x + CARD\_SIZE/2, y + CARD\_SIZE/2);

scale(scaleFactor, 1);

if (cardFlipProgress[i] > 0.5) {

// Carta virada para cima

if (matched[i]) {

// Cartas correspondidas (semi-transparentes)

tint(255, 100);

image(cardImages[cards[i]], -CARD\_SIZE/2, -CARD\_SIZE/2);

noTint();

} else {

// Cartas reveladas

image(cardImages[cards[i]], -CARD\_SIZE/2, -CARD\_SIZE/2);

}

} else {

// Carta virada para baixo

image(cardBack, -CARD\_SIZE/2, -CARD\_SIZE/2);

}

popMatrix();

}

// Mostrar informações do jogo

displayGameInfo();

// Mostrar contador de preview

if (currentState == GameState.PREVIEW) {

textAlign(CENTER, CENTER);

textSize(36);

fill(0);

text("Memorize! " + (previewTimer / 60 + 1), width/2, 50);

}

}

private void displayGameInfo() {

textSize(20);

fill(0);

textAlign(LEFT);

// Mostrar dificuldade

text("Dificuldade: " + getDifficultyName(currentDifficulty), 20, 30);

// Mostrar tentativas

text("Tentativas: " + attempts + "/" + maxAttempts, 20, 60);

// Mostrar pares encontrados

text("Pares: " + matchedPairs + "/" + (cards.length / 2), 20, 90);

// Mostrar tempo

int seconds = gameTimer / 60;

text("Tempo: " + nf(seconds / 60, 2) + ":" + nf(seconds % 60, 2), 20, 120);

// Mostrar pontuação atual

textAlign(RIGHT);

text("Pontuação: " + calculateScore(), width - 20, 30);

}

private void displayGameOver() {

background(240);

textFont(gameFont);

textAlign(CENTER, CENTER);

// Calcular pontuação final

int finalScore = calculateScore();

// Verificar se é um novo recorde

boolean isNewHighScore = finalScore > highScores[currentDifficulty.ordinal()];

if (isNewHighScore) {

highScores[currentDifficulty.ordinal()] = finalScore;

}

// Título

textSize(36);

if (matchedPairs == cards.length / 2) {

fill(0, 122, 255);

text("PARABÉNS! VOCÊ VENCEU!", width/2, height/4 - 30);

} else {

fill(255, 45, 85);

text("FIM DE JOGO", width/2, height/4 - 30);

}

// Mostrar estatísticas

textSize(24);

fill(0);

int statsY = height/2 - 80;

int lineHeight = 35;

text("Pares encontrados: " + matchedPairs + "/" + (cards.length / 2), width/2, statsY);

text("Tentativas utilizadas: " + attempts, width/2, statsY + lineHeight);

int seconds = gameTimer / 60;

text("Tempo: " + nf(seconds / 60, 2) + ":" + nf(seconds % 60, 2), width/2, statsY + 2 \* lineHeight);

// Mostrar pontuação

fill(0, 180, 0);

textSize(30);

text("Pontuação: " + finalScore, width/2, statsY + 3 \* lineHeight);

// Mostrar novo recorde

if (isNewHighScore) {

fill(255, 200, 0);

textSize(28);

text("NOVO RECORDE!", width/2, statsY + 4 \* lineHeight);

}

// Botões

int buttonY = height - 140;

// Botão de Nova Partida

displayButton("Nova Partida", width/2, buttonY, 250, 60);

// Botão de Menu Principal

displayButton("Menu Principal", width/2, buttonY + 80, 250, 60);

}

// Helper para exibir botões padronizados

private void displayButton(String text, float x, float y, float w, float h) {

// Verificar se o mouse está sobre o botão

boolean hover = mouseX > x - w/2 && mouseX < x + w/2 &&

mouseY > y - h/2 && mouseY < y + h/2;

// Cor do botão

if (hover) fill(0, 150, 255);

else fill(0, 122, 255);

// Desenhar botão

rect(x - w/2, y - h/2, w, h, 10);

// Texto do botão

fill(255);

textSize(24);

textAlign(CENTER, CENTER);

text(text, x, y);

}

// Calcular pontuação baseada em tempo, tentativas e dificuldade

private int calculateScore() {

if (currentState != GameState.GAME\_OVER && matchedPairs < cards.length / 2) {

// Pontuação durante o jogo

return matchedPairs \* 100;

} else {

// Pontuação final

int timeBonus = max(0, 3000 - (gameTimer / 60) \* 10); // Menos tempo = mais pontos

int attemptBonus = max(0, maxAttempts - attempts) \* 50; // Menos tentativas = mais pontos

int difficultyMultiplier = currentDifficulty.ordinal() + 1; // Fácil=1, Médio=2, Difícil=3

return (matchedPairs \* 100 + timeBonus + attemptBonus) \* difficultyMultiplier;

}

}

// Converter enum de dificuldade para texto

private String getDifficultyName(Difficulty diff) {

switch(diff) {

case EASY: return "Fácil";

case MEDIUM: return "Médio";

case HARD: return "Difícil";

default: return "";

}

}

void handleMousePressed() {

switch(currentState) {

case MENU:

handleMenuClick();

break;

case SETTINGS:

handleSettingsClick();

break;

case PLAYING:

handleGameClick();

break;

case GAME\_OVER:

handleGameOverClick();

break;

// Ignorar cliques durante o preview

case PREVIEW:

break;

}

}

private void handleMenuClick() {

// Botão Iniciar

if (isButtonClicked("Iniciar Jogo", width/2, height/2, 250, 60)) {

initializeGame();

}

// Botão Configurações

if (isButtonClicked("Configurações", width/2, height/2 + 100, 250, 60)) {

currentState = GameState.SETTINGS;

}

}

private void handleSettingsClick() {

int buttonY = height/2 - 50;

int buttonSpacing = 80;

// Botão Fácil

if (mouseX > width/2 - 125 && mouseX < width/2 + 125 &&

mouseY > buttonY - 30 && mouseY < buttonY + 30) {

currentDifficulty = Difficulty.EASY;

CARD\_SIZE = CARD\_SIZES[currentDifficulty.ordinal()];

// Recriar a carta de fundo para o novo tamanho

cardBack = createCardBackImage();

}

// Botão Médio

if (mouseX > width/2 - 125 && mouseX < width/2 + 125 &&

mouseY > buttonY + buttonSpacing - 30 && mouseY < buttonY + buttonSpacing + 30) {

currentDifficulty = Difficulty.MEDIUM;

CARD\_SIZE = CARD\_SIZES[currentDifficulty.ordinal()];

// Recriar a carta de fundo para o novo tamanho

cardBack = createCardBackImage();

}

// Botão Difícil

if (mouseX > width/2 - 125 && mouseX < width/2 + 125 &&

mouseY > buttonY + 2 \* buttonSpacing - 30 && mouseY < buttonY + 2 \* buttonSpacing + 30) {

currentDifficulty = Difficulty.HARD;

CARD\_SIZE = CARD\_SIZES[currentDifficulty.ordinal()];

// Recriar a carta de fundo para o novo tamanho

cardBack = createCardBackImage();

}

// Botão Voltar

if (mouseX > width/2 - 125 && mouseX < width/2 + 125 &&

mouseY > buttonY + 3 \* buttonSpacing - 30 && mouseY < buttonY + 3 \* buttonSpacing + 30) {

currentState = GameState.MENU;

}

}

private void handleGameClick() {

// Ignorar cliques se já houver duas cartas selecionadas ou animações em andamento

if (selectedCard1 != -1 && selectedCard2 != -1) return;

// Calcular posição inicial para centralizar o tabuleiro

float startX = (width - (GRID\_COLS \* (CARD\_SIZE + CARD\_MARGIN))) / 2;

float startY = (height - (GRID\_ROWS \* (CARD\_SIZE + CARD\_MARGIN))) / 2;

// Verificar clique nas cartas

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

int row = i / GRID\_COLS;

int col = i % GRID\_COLS;

float x = startX + col \* (CARD\_SIZE + CARD\_MARGIN);

float y = startY + row \* (CARD\_SIZE + CARD\_MARGIN);

if (mouseX > x && mouseX < x + CARD\_SIZE &&

mouseY > y && mouseY < y + CARD\_SIZE &&

!revealed[i] && !matched[i]) {

// Ignorar cliques durante animações

boolean animationInProgress = false;

for (int j = 0; j < cards.length; j++) {

if (cardFlipProgress[j] > 0 && cardFlipProgress[j] < 1) {

animationInProgress = true;

break;

}

}

if (!animationInProgress) {

revealCard(i);

}

break;

}

}

}

private void revealCard(int index) {

revealed[index] = true;

if (selectedCard1 == -1) {

// Primeira carta selecionada

selectedCard1 = index;

} else {

// Segunda carta selecionada

selectedCard2 = index;

attempts++;

// Verificar correspondência após um curto delay

thread("checkMatch");

}

}

// Esta função será executada em uma thread separada

void checkMatch() {

// Deixar a animação de flip terminar

delay(CARD\_FLIP\_FRAMES \* 16);

if (cards[selectedCard1] == cards[selectedCard2]) {

// Par encontrado

matched[selectedCard1] = true;

matched[selectedCard2] = true;

matchedPairs++;

// Verificar condição de vitória

if (matchedPairs == cards.length / 2) {

currentState = GameState.GAME\_OVER;

}

} else {

// Par não corresponde

revealed[selectedCard1] = false;

revealed[selectedCard2] = false;

}

// Resetar seleção

selectedCard1 = -1;

selectedCard2 = -1;

// Verificar condição de derrota

if (attempts >= maxAttempts) {

currentState = GameState.GAME\_OVER;

}

}

private void handleGameOverClick() {

int buttonY = height - 140;

// Botão Nova Partida

if (isButtonClicked("Nova Partida", width/2, buttonY, 250, 60)) {

initializeGame();

}

// Botão Menu Principal

if (isButtonClicked("Menu Principal", width/2, buttonY + 80, 250, 60)) {

currentState = GameState.MENU;

}

}

// Helper para verificar clique em botão

private boolean isButtonClicked(String text, float x, float y, float w, float h) {

return mouseX > x - w/2 && mouseX < x + w/2 &&

mouseY > y - h/2 && mouseY < y + h/2;

}

}

// Variável global do jogo

MemoryGame game;

void setup() {

size(800, 600);

smooth();

// Inicializar jogo

game = new MemoryGame();

}

void draw() {

game.update();

game.display();

}

void mousePressed() {

game.handleMousePressed();

}

// Funções de thread usadas pelo jogo

void animateAllCardsFlip() {

if (game != null) {

game.animateAllCardsFlip();

}

}

void checkMatch() {

if (game != null) {

game.checkMatch();

}

})