Sumário

Menu.py **3**
Introdução 3
Desenhar Texto 4
Conexão ao Banco de Dados 4
Atualização de Pontuações 4
Menu Principal 5
Manipulação de Erros 6
Conexão SSH 7
Função Jogar 8
Tela de Tabelas de Classificação 11
Tela de Login 12
Tela de Registro 14
Validação de Usuário 17
Login do Usuário 17
Ponto de Entrada Principal 17
Game_Offline.py **19**
Introdução 19
Configuração inicial e criação da janela do jogo 19
Loop do jogo e manipulação de eventos 19
Condições de fim de jogo e lógica de reinício 20
Movimento da cobra e detecção de colisão 21
Geração de comida e crescimento da cobra 21
Início do jogo 22
Client.py **23**
Introdução 23
Configurações de tela e jogo 23
Inicialização do cliente e comunicação com o servidor 24
Loop do jogo e manipulação de eventos 25
Desenho de elementos do jogo e atualização da tela 26
Servidor **28**
Introdução 28
Inicialização dos estados do servidor e dos jogadores 28
Manipulação de conexões dos jogadores 28

Gerenciamento do estado do jogo e interações dos jogadores 29
Atualização do banco de dados com os resultados do jogo 30
Finalização e fechamento do servidor 31
Modelo de Dados 33
Modelo Conceitual 33
Modelo Lógico 33
Relacionamentos 34

Menu.py

Introdução

Este documento guiará você pela implementação da funcionalidade de Menu Principal para o jogo Snake. Banco de dados criado na **AWS** e usando uma VM no **Google Cloud**.

A funcionalidade inclui:

- Inicialização da tela do jogo e configuração do menu principal.
- Manipulação de interações do usuário, como login, registro e jogabilidade.
- Conexão a um banco de dados para buscar e atualizar dados do usuário.
- Exibição de tabelas de classificação e gerenciamento de sessões de jogo.

Variáveis Globais e Inicialização do Pygame

Este código inicializa a biblioteca *Pygame*, cria uma janela do jogo com uma legenda, configura cores e uma imagem de fundo, e define fontes. Também declara uma variável global para armazenar o apelido do usuário logado e especifica o número de entradas a serem exibidas por página.

```
import pygame
import sys
import mysql.connector
import re
import tkinter as tk
from tkinter import messagebox
import paramiko
from client import inicializa_client, game_loop

from game_offline import rodar_jogo

import config
pygame.init()
# Inicializar Pygame
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Snake Game")
# Cores
white = config.branco
black = config.preto
gray = (200, 200, 200)
active_color = pygame.Color('dodgerblue2')
inactive_color = pygame.Color('lightskyblue3')
bg = pygame.image.load("BG.jpg")
# Fonte
font = pygame.font.Font("Unlock-Regular.ttf", 32)
fontPlay = pygame.font.Font("Unlock-Regular.ttf", 50)

# Variável Global para Armazenar Nickname do Usuário Logado
```

```
logged_in_user = None
# Número de entradas por página
entries_per_page = 10
```

Desenhar Texto

Este trecho de código define uma função *draw_text* que recebe uma superfície(tela), texto, posição e uma cor opcional como argumentos. Renderiza o texto usando uma cor especificada e o exibe na tela na posição fornecida.

```
def draw_text(surface, text, pos, color=black):
    text_surface = font.render(text, True, color)
    surface.blit(text_surface, pos)
```

Conexão ao Banco de Dados

Definimos uma função para conectar ao banco de dados MySQL criado na AWS. Esta função lida com erros de conexão e sai do programa se uma conexão não puder ser estabelecida.

```
def connect_to_db():
    try:
        connection = mysql.connector.connect(
            host="snakegame-python-estacio.cfaemws4kovz.us-east-
1.rds.amazonaws.com",
            port=3306,
            user="admin",
            password="Teste123",
            database="SnakePythonDB"
        )
        return connection
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        sys.exit(1)
```

Atualização de Pontuações

Implementamos uma função para atualizar a pontuação do usuário no banco de dados. Esta função conecta ao banco de dados, executa uma query de atualização e lida com possíveis erros. Esta função é executada após a partida online e não é usada se o jogo estiver sendo jogado no modo offline.

```
def update_score(new_score):
    global pont
    if new_score > pont:
        try:
            pont = new_score
            connection = connect_to_db()
            cursor = connection.cursor()
            query = "UPDATE PLAYER SET nmaior_pontc = %s WHERE
cd_player = %s"
```

```
            cursor.execute(query,(new_score, cd_player))
            connection.commit()
            cursor.close()
            connection.close()
        except mysql.connector.Error as err:
            show_popup_error("Erro", f"Falha ao buscar dados:
{translate_error(err)}")
```

Menu Principal

Definimos a função do menu principal, que cria botões para diferentes ações como jogar, fazer login, registrar-se, visualizar tabelas de classificação e sair do jogo. Alguns botões não aparecem quando o usuário está logado.

```
def main_menu():
    # Dimensões dos botões
    button_width = 150
    button_height = 50
    # Criar botões
    play_button = pygame.Rect(300, 200, button_width +50,
button_height+ 50)
    login_button = pygame.Rect(0, 0, button_width, button_height)
    register_button = pygame.Rect(650, 0, button_width,
button_height)
    credits_button = pygame.Rect(350, 550, button_width,
button_height)
    leaderboards_button = pygame.Rect(0, 550, button_width+90,
button_height)
    quit_button = pygame.Rect(650, 550, button_width, button_height)
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if play_button.collidepoint(event.pos):
                    play_game()

                if login_button.collidepoint(event.pos) and not
logged_in_user:
                    login_screen()

                if register_button.collidepoint(event.pos)and not
logged_in_user:
                    register_screen()

                #if credits_button.collidepoint(event.pos):
                #    credits_screen()

                if leaderboards_button.collidepoint(event.pos):
                    leaderboards_screen()
```

```
                if quit_button.collidepoint(event.pos):
                    running = False
                    pygame.quit()
                    sys.exit()
        screen.fill(white)
        screen.blit(bg, (0 ,0))
        pygame.draw.rect(screen, black, play_button, 2)

        if logged_in_user is None:
            pygame.draw.rect(screen, black, login_button, 2)
            pygame.draw.rect(screen, black, register_button, 2)

        #pygame.draw.rect(screen, black, credits_button, 2)
        pygame.draw.rect(screen, black, leaderboards_button, 2)
        pygame.draw.rect(screen, black, quit_button, 2)
        draw_text2(screen, "Play", (play_button.x + 40,
play_button.y + 20))

        if logged_in_user is None:
            draw_text(screen, "Login", (login_button.x + 15,
login_button.y + 10))
            draw_text(screen, "Register", (register_button.x + 5,
register_button.y + 10))

        #draw_text(screen, "Credits", (credits_button.x + 15,
credits_button.y + 10))
        draw_text(screen, "Leaderboards", (leaderboards_button.x +
5, leaderboards_button.y + 10))
        draw_text(screen, "Quit", (quit_button.x + 40, quit_button.y
+ 10))

        # Exibir nickname do usuário logado
        if logged_in_user:
            draw_text(screen, f"{logged_in_user}", (10, 10))
            draw_text(screen, f"{pont}", (650, 10))
        pygame.display.flip()
```

Manipulação de Erros

Implementamos funções para traduzir erros do MySQL em mensagens amigáveis ao usuário e mostrar mensagens pop-up para erros e informações.

```
def translate_error(err):
    if err.errno == mysql.connector.errorcode.ER_DUP_ENTRY:
        return "E-mail já cadastrado."
    elif err.errno == mysql.connector.errorcode.ER_BAD_DB_ERROR:
        return "O banco de dados especificado não existe."
    elif err.errno ==
mysql.connector.errorcode.ER_ACCESS_DENIED_ERROR:
        return "Credenciais de acesso ao banco de dados inválidas."
    else:
        return "Ocorreu um erro desconhecido."
```

Este trecho de código define duas funções *show_popup* e *show_popup_error* que exibem janelas pop-up com um título e mensagem especificados. As janelas são criadas usando a biblioteca *tkinter* e o módulo *messagebox*. A janela raiz é escondida e as janelas pop-up são configuradas para estarem sempre no topo.

```
def show_popup(title, message):
    root = tk.Tk()
    root.withdraw()  # Esconder a janela principal
    root.attributes('-topmost', True)
    messagebox.showinfo(title, message)

def show_popup_error(title, message):
    root = tk.Tk()
    root.withdraw()  # Esconder a janela principal
    root.attributes('-topmost', True)
    messagebox.showerror(title, message)
```

Conexão SSH

Este trecho de código conecta a uma VM no Google Cloud via SSH usando o endereço IP, nome de usuário e chave privada fornecidos. Ele cria um cliente SSH, define a política de chave de host ausente para aceitar automaticamente a chave do servidor e tenta estabelecer a conexão SSH. Se o servidor não estiver em execução, executa um comando para iniciar um servidor Python. Em seguida, conecta ao banco de dados, recupera o código do último jogo, incrementa em 1, insere um novo jogo na tabela PARTIDAS, efetua *commit* (efetiva as alterações feitas) no banco de dados e fecha o cursor e a conexão. Finalmente, retorna o código que foi incluído na tabela PARTIDAS. Se houver erros durante o processo, exibe a mensagem de erro.

O comando `'nohup python3 ./server.py > server.log 2>&1 &'` utilizado para executar um script Python em segundo plano, de forma que ele continue rodando mesmo que a sessão seja encerrada.

```
def connect_vm_ssh():
    # Definindo as informações da conexão
    ip_address = '35.212.236.121'
    username = 'chuaum141'
    private_key_path = 'keyVM-open'  # Substitua pelo caminho
correto da sua chave privada
    # Cria um cliente SSH
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  #
Aceita automaticamente a chave do servidor
    try:
        # Carrega a chave privada
        private_key =
paramiko.RSAKey.from_private_key_file(private_key_path)
        ssh.connect(ip_address, username=username, pkey=private_key)
        if not check_server_running(ssh):

            ssh.exec_command('nohup python3 ./server.py > server.log
2>&1 &')
        try:
            # Conectar ao banco de dados
```

```
            connection = connect_to_db()
            cursor = connection.cursor()
            # Buscar o código da última partida
            query = "SELECT MAX(cd_partida) FROM PARTIDAS"
            cursor.execute(query)
            result = cursor.fetchone()
            cd_partida = 0
            if result[0]:
                cd_partida = result[0]
            cd_partida += 1
            # Inserir nova partida na tabela PARTIDAS
            query = "INSERT INTO PARTIDAS (cd_partida, dt_inic,
dt_fim, ctpo_sit, cd_vencedor) VALUES (%s, CURRENT_TIMESTAMP, NULL,
3, NULL)"
            cursor.execute(query, (cd_partida,))
            # Confirmar as alterações no banco de dados
            connection.commit()
            # Fechar o cursor e a conexão
            cursor.close()
            connection.close()
            return cd_partida
        except mysql.connector.Error as err:
            # Tratar mensagens de erro
            error_message = translate_error(err)

            # Exibir popup de erro
            show_popup_error("Erro", f"Falha ao registrar:
{error_message}")
    except paramiko.AuthenticationException:
        print("Falha na autenticação, verifique suas credenciais")
    except paramiko.SSHException as sshException:
        print(f"Erro ao se conectar ao servidor SSH:
{sshException}")
    except Exception as e:
        print(f"Ocorreu um erro: {e}")
    finally:
        # Fecha a conexão SSH
        ssh.close()
```

Este trecho de código verifica se um servidor está em execução executando o comando `'pgrep -f server.py'` em uma conexão SSH. Em seguida, lê a saída do comando e retorna **True** se a saída não estiver vazia, indicando que o servidor está em execução, e **False** caso contrário.

```
def check_server_running(ssh):
    stdin, stdout, stderr = ssh.exec_command('pgrep -f server.py')
    #stdin, stdout, stderr = ssh.exec_command('ls')
    print(stdout.read().decode())
    if not stdout.read().decode():
        return False
    return True
```

Função Jogar

Implementamos a função **`play_game()`**, que lida com modos de jogo offline e online. Também atualiza a pontuação do usuário e exibe a tela de vencedor após o fim da partida online. O botão play_on só é habilitado se o usuário estiver logado.

Quando clicado no botão play_on, a função `connect_vm_ssh()` é chamada para estabelecer uma conexão com o servidor e inicializar o jogo por ***/client.py***.

```
def play_game():
    back_button = pygame.Rect(650, 550, 120, 50)
    play_off_button = pygame.Rect(300, 250, 220, 50)
    play_on_button = pygame.Rect(300, 350, 220, 50)
    player1_name = logged_in_user

    player2_name = ""
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if back_button.collidepoint(event.pos):
                    return
                if play_off_button.collidepoint(event.pos):
                    rodar_jogo()

                if play_on_button.collidepoint(event.pos) and
logged_in_user:

                    partida = connect_vm_ssh()

                    data = inicializa_client(player1_name,
player2_name, partida, cd_player)

                    if data is not None:
                        winner, players = data

                        if players is not None and "player1" in
players and "id_jogador" in players["player1"]:

                            if int(players["player1"]["id_jogador"])
== cd_player:


update_score(int(players["player1"]["score"]))
                            else:
update_score(int(players["player2"]["score"]))

                            winner_screen(winner)

                        else:
                            show_popup("Server", "Servidor
Encerrado!")
        screen.fill(white)

        screen.blit(bg, (0, 0))
```

```
        pygame.draw.rect(screen, black, play_off_button, 2)
        draw_text(screen, "Play Offline", (play_off_button.x + 20,
play_off_button.y + 10))

        if logged_in_user:
            pygame.draw.rect(screen, black, play_on_button, 2)
            draw_text(screen, "Play Online", (play_on_button.x + 20,
play_on_button.y + 10))

        pygame.draw.rect(screen, black, back_button, 2)
        draw_text(screen, "Back", (back_button.x + 20, back_button.y
+ 10))

        # Exibir nickname do usuário logado
        if logged_in_user:
            draw_text(screen, f"{logged_in_user}", (10, 10))
            draw_text(screen, f"{pont}", (650, 10))

        pygame.display.flip()
```

Este trecho de código define uma função **winner_screen** que recebe um argumento winner retornado por **/server.py** para **/client.py**. Esta é uma função que exibe o vencedor do jogo.

```
def winner_screen(winner):
    back_button = pygame.Rect(650, 550, 120, 50)
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if back_button.collidepoint(event.pos):
                    return
        screen.fill(white)
        screen.blit(bg, (0, 0))
        if winner == "":
            draw_text(screen, "Empate!", (400, 300))

        else:
            draw_text(screen, f"{winner} Venceu!", (200, 300))

        pygame.draw.rect(screen, black, back_button, 2)
        draw_text(screen, "Back", (back_button.x + 20, back_button.y
+ 10))

        # Exibir nickname do usuário logado
        if logged_in_user:
            draw_text(screen, f"{logged_in_user}", (10, 10))
            draw_text(screen, f"{pont}", (650, 10))

        pygame.display.flip()
```

Tela de Tabelas de Classificação

Definimos a função **`leaderboards_screen()`**, que exibe os dados dos principais jogadores e permite a navegação através das páginas de entradas de tabelas de classificação.

```
def leaderboards_screen():
    leaderboards_data = fetch_leaderboards_data()
    back_button = pygame.Rect(650, 550, 120, 50)
    next_button = pygame.Rect(680, 500, 80, 40)
    prev_button = pygame.Rect(20, 500, 80, 40)
    page = 0
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if back_button.collidepoint(event.pos):
                    return
                if next_button.collidepoint(event.pos):
                    if (page + 1) * entries_per_page <
len(leaderboards_data):

                        page += 1
                if prev_button.collidepoint(event.pos):
                    if page > 0:
                        page -= 1
        screen.fill(white)

        screen.blit(bg, (0, 0))

        # Desenhar botões
        pygame.draw.rect(screen, black, back_button, 2)

        draw_text(screen, "Back", (back_button.x + 20, back_button.y
+ 10))

        pygame.draw.rect(screen, black, next_button, 2)

        draw_text(screen, "Next", (next_button.x + 10, next_button.y
+ 5))

        pygame.draw.rect(screen, black, prev_button, 2)

        draw_text(screen, "Prev", (prev_button.x + 10, prev_button.y
+ 5))

        # Desenhar cabeçalhos da tabela
        draw_text(screen, "Nickname", (150, 50))

        draw_text(screen, "Maior Pontuação", (500, 50))

        # Desenhar dados da tabela
```

```
        start_index = page * entries_per_page

        end_index = start_index + entries_per_page

        for i, (nickname, score) in
enumerate(leaderboards_data[start_index:end_index]):
            y_pos = 100 + i * 40
            order_number = start_index + i + 1
            draw_text(screen, str(order_number), (50, y_pos))  #
Exibir ordem do registro

            draw_text(screen, nickname, (150, y_pos))

            draw_text(screen, str(score), (500, y_pos))

        # Exibir nickname do usuário logado
        if logged_in_user:

            draw_text(screen, f"{logged_in_user}", (10, 10))

            draw_text(screen, f"{pont}", (650, 10))

        pygame.display.flip()
```

Criamos uma função para buscar os dados dos 100 melhores jogadores do banco de dados. Esta função conecta ao banco de dados, executa uma query de seleção e retorna o resultado.

```
def fetch_leaderboards_data():
    try:
        connection = connect_to_db()
        cursor = connection.cursor()
        query = "SELECT nickname, nmaior_pontc FROM PLAYER ORDER BY
nmaior_pontc DESC LIMIT 100"
        cursor.execute(query)
        result = cursor.fetchall()
        cursor.close()
        connection.close()
        return result
    except mysql.connector.Error as err:
        show_popup_error("Erro", f"Falha ao buscar dados:
{translate_error(err)}")
        return []
```

Tela de Login

Criamos a função `login_screen()`, que lida com a entrada do usuário para email e senha, e realiza a validação do login.

```
def login_screen():
    email_input = pygame.Rect(300, 200, 400, 32)
    password_input = pygame.Rect(300, 250, 400, 32)
    login_button = pygame.Rect(350, 300, 120, 50)  # Botão de Login
```

```
    back_button = pygame.Rect(650, 550, 120, 50)
    email_text = ''
    password_text = ''
    active_input = None
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if email_input.collidepoint(event.pos):
                    active_input = 'email'
                elif password_input.collidepoint(event.pos):
                    active_input = 'password'
                elif login_button.collidepoint(event.pos):
                    if email_text == '' or password_text == '':
                        show_popup_error("Erro", "Preencha todos os
campos.")
                    else:
                        if login_user(email_text, password_text):

                            show_popup("Sucesso", "Login realizado
com sucesso!")
                            return
                        else:
                            show_popup_error("Erro", "E-mail ou
senha inválidos.")
                elif back_button.collidepoint(event.pos):
                    return
                else:
                    active_input = None
            if event.type == pygame.KEYDOWN and active_input:
                if event.key == pygame.K_BACKSPACE:
                    if active_input == 'email':
                        email_text = email_text[:-1]
                    elif active_input == 'password':
                        password_text = password_text[:-1]
                else:
                    if active_input == 'email':
                        email_text += event.unicode
                    elif active_input == 'password':
                        password_text += event.unicode
        screen.fill(white)

        screen.blit(bg, (0, 0))

        draw_text(screen, "Login Screen", (300, 150))

        # Desenha caixa de e-mail
        draw_text(screen, "Email:", (190, email_input.y + 5))

        pygame.draw.rect(screen, active_color if active_input ==
'email' else inactive_color, email_input, 2)

        draw_text(screen, email_text, (email_input.x,
email_input.y))
```

```
        # Desenha caixa de senha
        draw_text(screen, "Senha:", (180, password_input.y + 5))

        pygame.draw.rect(screen, active_color if active_input ==
'password' else inactive_color, password_input, 2)

        draw_text(screen, "*" * len(password_text),
(password_input.x + 5, password_input.y + 5), color=black)

        # Desenha botão de login
        pygame.draw.rect(screen, black, login_button, 2)

        draw_text(screen, "Login", (login_button.x + 15,
login_button.y + 10))

        pygame.draw.rect(screen, black, back_button, 2)

        draw_text(screen, "Back", (back_button.x + 20, back_button.y
+ 10))

        pygame.display.flip()
```

Tela de Registro

Implementamos a função register_screen, que lida com a entrada do usuário para detalhes de registro e realiza a validação antes de registrar um novo usuário.

```
def register_screen():
    label_offset_y = 5  # Distância vertical entre a label e o campo
de entrada
    email_input = pygame.Rect(300, 150, 400, 32)
    nickname_input = pygame.Rect(300, 200, 400, 32)
    password_input = pygame.Rect(300, 250, 400, 32)
    confirm_password_input = pygame.Rect(300, 300, 400, 32)
    register_button = pygame.Rect(340, 350, 160, 50)  # Botão
Registrar
    back_button = pygame.Rect(650, 550, 120, 50)
    email_text = ''
    nickname_text = ''
    password_text = ''
    confirm_password_text = ''
    active_input = None
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if email_input.collidepoint(event.pos):
                    active_input = 'email'
                elif nickname_input.collidepoint(event.pos):
                    active_input = 'nickname'
```

```
                elif password_input.collidepoint(event.pos):
                    active_input = 'password'
                elif confirm_password_input.collidepoint(event.pos):
                    active_input = 'confirm_password'
                elif register_button.collidepoint(event.pos):
                    if email_text == '' or nickname_text == '' or
password_text == '' or confirm_password_text == '':
                        show_popup_error("Erro", "Preencha todos os
campos.")
                    else:
                        if validate_email(email_text):

                            if validate_field(nickname_text, 15):

                                if validate_field(password_text,
20):

                                    if password_text ==
confirm_password_text:
                                        if
register_user(nickname_text, email_text, password_text):

                                            return
                                    else:
                                        show_popup_error("Erro", "As
senhas não coincidem.")
                                else:
                                    show_popup_error("Erro", "Senha
inválida. Máximo de 20 caracteres e sem acentos.")
                            else:
                                show_popup_error("Erro", "Nickname
inválido. Máximo de 15 caracteres e sem acentos.")
                        else:
                            show_popup_error("Erro", "E-mail
inválido.")
                elif back_button.collidepoint(event.pos):
                    return
                else:
                    active_input = None
            if event.type == pygame.KEYDOWN and event.key !=
pygame.K_RETURN and active_input:
                if event.key == pygame.K_BACKSPACE:
                    if active_input == 'email':
                        email_text = email_text[:-1]
                    elif active_input == 'nickname':
                        nickname_text = nickname_text[:-1]
                    elif active_input == 'password':
                        password_text = password_text[:-1]
                    elif active_input == 'confirm_password':
                        confirm_password_text =
confirm_password_text[:-1]
                else:
                    if active_input == 'email':
                        email_text += event.unicode
                    elif active_input == 'nickname':
                        nickname_text += event.unicode
                    elif active_input == 'password':
```

```
                        password_text += event.unicode
                    elif active_input == 'confirm_password':
                        confirm_password_text += event.unicode
        screen.fill(white)

        screen.blit(bg, (0, 0))

        draw_text(screen, "Register Screen", (300, 100))

        # Desenha rótulos e caixas de entrada
        draw_text(screen, "Email:", (190, email_input.y +
label_offset_y))

        pygame.draw.rect(screen, active_color if active_input ==
'email' else inactive_color, email_input, 2)

        draw_text(screen, email_text, (email_input.x,
email_input.y))

        draw_text(screen, "Nickname:", (115, nickname_input.y +
label_offset_y))

        pygame.draw.rect(screen, active_color if active_input ==
'nickname' else inactive_color, nickname_input, 2)

        draw_text(screen, nickname_text, (nickname_input.x,
nickname_input.y))

        draw_text(screen, "Senha:", (180, password_input.y +
label_offset_y))

        pygame.draw.rect(screen, active_color if active_input ==
'password' else inactive_color, password_input, 2)

        draw_text(screen, "*" * len(password_text),
(password_input.x + 5, password_input.y + 5), color=black)

        draw_text(screen, "Conf. a Senha:", (60,
confirm_password_input.y + label_offset_y))

        pygame.draw.rect(screen, active_color if active_input ==
'confirm_password' else inactive_color,

                         confirm_password_input, 2)
        draw_text(screen, "*" * len(confirm_password_text),

                  (confirm_password_input.x + 5,
confirm_password_input.y + 5), color=black)

        # Desenha botão Registrar
        pygame.draw.rect(screen, black, register_button, 2)

        draw_text(screen, "Register", (register_button.x + 10,
register_button.y + 10))

        pygame.draw.rect(screen, black, back_button, 2)
```

```
        draw_text(screen, "Back", (back_button.x + 20, back_button.y
+ 10))

        pygame.display.flip()
```

Validação de Usuário

Definimos funções para validar email e outros campos de entrada.

```
def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None
def validate_field(field_text, max_length):
    pattern = fr'^[a-zA-Z0-9]{{1,{max_length}}}$'
    return re.match(pattern, field_text) is not None
```

Login do Usuário

Implementamos a função **login_user**, que valida as credenciais do usuário contra o banco de dados e define variáveis globais para o usuário logado.

```
def login_user(email, password):
    try:
        connection = connect_to_db()
        cursor = connection.cursor()
        # Buscar usuário na tabela Player
        query = "SELECT cd_player, nickname, nmaior_pontc FROM
PLAYER WHERE email_player = %s AND password = %s"
        cursor.execute(query, (email, password))
        result = cursor.fetchone()
        cursor.close()
        connection.close()
        if result:
            global logged_in_user, cd_player, pont

            logged_in_user = result[1]

            cd_player = result[0]
            pont = result[2]
            return True
        else:
            return False
    except mysql.connector.Error as err:
        # Tratar mensagens de erro
        error_message = translate_error(err)

        show_popup_error("Erro", f"Falha ao fazer login:
{error_message}")
        return
```

Ponto de Entrada Principal

Finalmente, chamamos a função main_menu se o script for executado diretamente.

```
if __name__ == "__main__":
    main_menu()
```

Isso conclui o walkthrough da implementação da funcionalidade de menu. Cada seção é projetada para lidar com aspectos específicos do jogo, desde a interação do usuário até o gerenciamento do banco de dados.

Game_Offline.py

Introdução

Este documento irá guiá-lo através da implementação da funcionalidade "Modo Offline" para um jogo Snake.

Essa funcionalidade permite que o jogo seja jogado offline com funcionalidades básicas como movimento, detecção de colisão e condições de fim de jogo.

Vamos cobrir:

- Configuração inicial e criação da janela do jogo.
- Loop do jogo e manipulação de eventos.
- Movimento da cobra e detecção de colisão.
- Geração de comida e crescimento da cobra.
- Condições de fim de jogo e lógica de reinício.

Configuração inicial e criação da janela do jogo

Nós inicializamos o jogo e configuramos a janela de exibição com dimensões específicas.

```
import pygame
import random
pygame.init()
pygame.display.set_caption('Snake')
largura, altura = 800, 600
tela = pygame.display.set_mode((largura, altura))

relogio = pygame.time.Clock()
```

Definimos cores e constantes do jogo para uso posterior.

```
preto = (0, 0, 0)
branco = (255, 255, 255)
verde = (0, 255, 0)
vermelho = (255, 0, 0)
tamanho_quadrado = 10
velocidade_cobra = 10
```

Loop do jogo e manipulação de eventos

Este trecho de código desenha uma cobra na tela usando a função `pygame.draw.rect`. Ele itera sobre cada parte da cobra na `lista_cobra` e desenha um retângulo com um tamanho de `tamanho_quadrado` nas coordenadas especificadas na superfície tela.

```
def desenhar_cobra(tamanho_quadrado, lista_cobra):
    for parte in lista_cobra:
        pygame.draw.rect(tela, verde, [parte[0], parte[1],
tamanho_quadrado, tamanho_quadrado])
```

Este trecho de código define uma função `rodar_jogo()` que define duas variáveis booleanas `fim_jogo` e `fim_de_jogo` como False. E ele define a posição e velocidade iniciais da cobra, gera coordenadas aleatórias para a comida e cria uma lista vazia para armazenar o corpo da cobra. A variável `comprimento_cobra` é usada para manter o controle do comprimento da cobra.

```
def rodar_jogo():
    fim_jogo = False
    fim_de_jogo = False
    x_cobra = largura // 2
    y_cobra = altura // 2
    x_cobra_velocidade = 0
    y_cobra_velocidade = 0
    x_comida = round(random.randrange(0, largura - tamanho_quadrado)
/ 10.0) * 10.0
    y_comida = round(random.randrange(0, altura - tamanho_quadrado)
/ 10.0) * 10.0

    lista_cobra = []
    comprimento_cobra = 1
```

Condições de fim de jogo e lógica de reinício

Este trecho de código exibe uma tela de fim de jogo com opções para o jogador jogar novamente ou retornar ao menu. Ele preenche a tela com uma cor preta, renderiza e exibe mensagens de texto usando diferentes fontes, e atualiza a exibição.

```
while not fim_jogo:
    while fim_de_jogo:
        tela.fill(preto)

        fonte = pygame.font.SysFont(None, 100)

        mensagem = fonte.render("Fim de jogo", True, branco)
        tela.blit(mensagem, [largura // 2 - 90, altura // 3])

        fonte = pygame.font.SysFont(None, 50)

        mensagem = fonte.render("Pressione C para Jogar Novamente",
True, branco)
        tela.blit(mensagem, [largura // 6, altura // 3 + 200])

        mensagem = fonte.render("Pressione ESCAPE para Voltar ao
Menu", True, branco)
        tela.blit(mensagem, [largura // 6, altura // 3 + 300])

        pygame.display.update()
```

Lidamos com as entradas do usuário durante o estado de fim de jogo para reiniciar o jogo ou retornar ao menu.

```
for evento in pygame.event.get():
    if evento.type == pygame.QUIT:
        fim_jogo = True
        fim_de_jogo = False
    if evento.type == pygame.KEYDOWN:
        if evento.key == pygame.K_q:
```

```
            fim_jogo = True
            fim_de_jogo = False
        if evento.key == pygame.K_c:
            fim_de_jogo = False
            rodar_jogo()

        elif evento.key == pygame.K_ESCAPE:
            return
```

Movimento da cobra e detecção de colisão

Lidamos com as entradas do usuário para controlar a direção do movimento da cobra.

```
for evento in pygame.event.get():
    if evento.type == pygame.QUIT:
        fim_jogo = True
    if evento.type == pygame.KEYDOWN:
        if evento.key == pygame.K_LEFT or evento.key == pygame.K_a:
            x_cobra_velocidade = -tamanho_quadrado

            y_cobra_velocidade = 0
        elif evento.key == pygame.K_RIGHT or evento.key ==
pygame.K_d:
            x_cobra_velocidade = tamanho_quadrado

            y_cobra_velocidade = 0
        elif evento.key == pygame.K_UP or evento.key == pygame.K_w:
            y_cobra_velocidade = -tamanho_quadrado

            x_cobra_velocidade = 0
        elif evento.key == pygame.K_DOWN or evento.key ==
pygame.K_s:
            y_cobra_velocidade = tamanho_quadrado

            x_cobra_velocidade = 0
        elif evento.key == pygame.K_ESCAPE:
            return
```

Atualizamos a posição da cobra com base na velocidade atual e verificamos colisões com os limites do jogo.

```
x_cobra += x_cobra_velocidade
y_cobra += y_cobra_velocidade
if x_cobra >= largura or x_cobra < 0 or y_cobra >= altura or y_cobra
< 0:
    fim_de_jogo = True
tela.fill(preto)
```

Geração de comida e crescimento da cobra

Desenhamos a comida na tela e gerenciamos o crescimento da cobra quando ela come a comida.

```
pygame.draw.rect(tela, vermelho, [x_comida, y_comida,
tamanho_quadrado, tamanho_quadrado])

cabeca_cobra = [x_cobra, y_cobra]
```

```
lista_cobra.append(cabeca_cobra)
if len(lista_cobra) > comprimento_cobra:
    del lista_cobra[0]
```

Verificamos colisões entre a cabeça da cobra e seu corpo, e atualizamos a exibição.

```
for parte in lista_cobra[:-1]:
    if parte == cabeca_cobra:
        fim_de_jogo = True
desenhar_cobra(tamanho_quadrado, lista_cobra)

pygame.display.update()
```

Este trecho de código verifica se as variáveis `x_cobra` e `y_cobra` são iguais a `x_comida` e `y_comida` respectivamente. Se forem iguais, atualiza as variáveis `x_comida` e `y_comida` com novos valores aleatórios. Também aumenta a variável comprimento_cobra em 1. O código então espera por um certo tempo determinado pela variável velocidade_cobra usando a função `relogio.tick`. Finalmente, após o loop, ele fecha a janela do Pygame usando pygame.quit e retorna ao menu principal.

```
    if x_cobra == x_comida and y_cobra == y_comida:
        x_comida = round(random.randrange(0, largura -
tamanho_quadrado) / 10.0) * 10.0

        y_comida = round(random.randrange(0, altura -
tamanho_quadrado) / 10.0) * 10.0

        comprimento_cobra += 1
    relogio.tick(velocidade_cobra)

pygame.quit()
```

Início do jogo

Este trecho de código executa a função `rodar_jogo` se o módulo atual for o módulo principal.

```
if __name__ == "__main__":
    rodar_jogo()
```

Isso conclui a implementação da funcionalidade "Modo Offline". O jogo agora está pronto para ser jogado offline com funcionalidades básicas.

Client.py

Introdução

Este documento irá guiá-lo através da implementação da funcionalidade "Modo Online”.

A funcionalidade permite jogar multiplayer online usando Pygame e comunicação por socket.

Vamos cobrir:

- Configurações de tela e jogo.
- Inicialização do cliente e comunicação com o servidor.
- Loop do jogo e manipulação de eventos.
- Desenho de elementos do jogo e atualização da tela.

Configurações de tela e jogo

Começamos configurando a tela e definindo constantes do jogo. Isso é essencial para inicializar a janela do jogo e definir as direções para o movimento da cobra.

Este código inicializa a biblioteca Pygame, importa módulos necessários e configura a janela do jogo. Define constantes para diferentes direções no jogo e inicia o jogo com a direção STOP.

```
import pygame
import socket
import ast
import tkinter as tk
from tkinter import messagebox
import config  # Importa as configurações do arquivo config.py
pygame.init()
# Configurações da tela
screen = pygame.display.set_mode((config.width, config.height))
pygame.display.set_caption("Snake Game Online")
# Direções
UP = "UP"
DOWN = "DOWN"
LEFT = "LEFT"
RIGHT = "RIGHT"
STOP = "STOP"
```

Primeiro, procedemos para configurar o relógio do jogo e inicializar as variáveis de estado do jogador e do jogo, que serão atualizadas de acordo com a comunicação do servidor.

```
# Configurações do jogo
clock = pygame.time.Clock()
# Inicializa as cobras e comida (será atualizado do servidor)
player_snake = []
other_snake = []
food_pos = [0, 0]
# Função para mostrar a pontuação (opcional)
font_style = pygame.font.Font(None, 20)
waiting_font = pygame.font.Font(None, 50)
```

Este código define duas funções, `show_popup_error` e `show_popup`, que exibem janelas pop-up com um título e mensagem especificados. As janelas são criadas usando a biblioteca `tkinter` e o módulo `messagebox`. O mesmo presente no Menu.py.

```
def show_popup_error(title, message):
    root = tk.Tk()
    root.withdraw()  # Esconder a janela principal
    root.attributes('-topmost', True)
    messagebox.showerror(title, message)
def show_popup(title, message):
    root = tk.Tk()
    root.withdraw()  # Esconder a janela principal
    root.attributes('-topmost', True)
    messagebox.showinfo(title, message)
```

Inicialização do cliente e comunicação com o servidor

Este trecho de código inicializa um socket cliente, conecta-o a um servidor no endereço IP 35.212.236.121 (IP da VM no Google Cloud) e porta 9999, envia uma mensagem com `player1_name`, `partida` e `cd_player`, recebe uma resposta do servidor, exibe uma mensagem de espera na tela e então inicia um loop de jogo. Se houver um vencedor no loop do jogo, ele é retornado. Se houver uma exceção durante o processo, um pop-up de erro é exibido com a mensagem de erro.

```
def inicializa_client(player1_name, player2_name, partida,
cd_player):
    try:
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect(("35.212.236.121", 9999))
        #client.connect(("192.168.15.2", 9999))

client.sendall(f"{player1_name},{player2_name},{partida},{cd_player}
".encode('utf-8'))
        data = client.recv(2048).decode('utf-8')
        if data:
            received_player1_name, received_player2_name =
data.split(',')
            print(f"[Client] Recebido do servidor:
{received_player1_name}, {received_player2_name}")
        waiting_message = waiting_font.render("Aguardando conexão
Player 2", True, config.branco)
        screen.fill(config.preto)
        screen.blit(waiting_message, (
            (config.width - waiting_message.get_width()) // 2,
            (config.height - waiting_message.get_height()) // 2
        ))
        pygame.display.update()
        winner = game_loop(client)
        if winner is not None:
            return winner
    except Exception as e:
        show_popup_error('Error',f"Erro ao conectar ao servidor:
{e}")
```

Loop do jogo e manipulação de eventos

Este trecho de código é um loop de jogo que lida com a entrada do usuário e comunicação com um servidor. Ele usa a biblioteca Pygame para ouvir eventos do teclado e atualizar a variável de direção de acordo. Em seguida, envia a direção para o servidor e recebe o estado do jogo em retorno. O estado do jogo é decodificado de uma string de dados recebida usando `ast.literal_eval()`. A cobra do jogador, pontuação e a cobra e pontuação de outros jogadores são extraídas do estado do jogo e armazenadas em variáveis. A posição da comida e o nome do jogador também são extraídos do estado do jogo.

```
def game_loop(client):
    global player_snake, other_snake, food_pos
    winner = None
    game_state = None
    running = True
    direction = STOP
    player1_score = 0
    player2_score = 0
    player1_name = ''
    player2_name = ''
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                break
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_w or event.key ==
pygame.K_UP:
                    direction = UP
                elif event.key == pygame.K_s or event.key ==
pygame.K_DOWN:
                    direction = DOWN
                elif event.key == pygame.K_a or event.key ==
pygame.K_LEFT:
                    direction = LEFT
                elif event.key == pygame.K_d or event.key ==
pygame.K_RIGHT:
                    direction = RIGHT
                elif event.key == pygame.K_ESCAPE:
                    running = False
                    return
        try:
            # Envia a direção para o servidor
            client.sendall(direction.encode('utf-8'))
            # Recebe o estado do jogo do servidor
            data = client.recv(2048).decode('utf-8')
            if len(data) == 0:
                print("Nenhum dado recebido do servidor")
                break
            try:
                game_state = ast.literal_eval(data)
                print("Estado do jogo recebido do servidor:",
game_state)
            except SyntaxError as e:
                print("Erro ao decodificar o estado do jogo:", e)
```

```
                break
            if game_state:
                player_snake = game_state["player1"]["snake"]
                player1_score = game_state["player1"]["score"]
                other_snake = game_state["player2"]["snake"]
                player2_score = game_state["player2"]["score"]
                food_pos = game_state["food_pos"]
                player1_name = game_state["player1"]["name"]
                player2_name = game_state["player2"]["name"]
                if game_state["player2"]["venceu"]:
                   winner = game_state["player2"]["name"]
                elif game_state["player1"]["venceu"]:
                    winner = game_state["player1"]["name"]
                else:
                    winner = ""
        except Exception as e:
            print(f"Erro ao comunicar com o servidor: {e}")
            break
        # Desenha tudo na tela
        screen.fill(config.preto)
        for block in player_snake:
            pygame.draw.rect(screen, config.verde,
pygame.Rect(block[0], block[1], config.snake_block,
config.snake_block))
        for block in other_snake:
            pygame.draw.rect(screen, config.azul,
pygame.Rect(block[0], block[1], config.snake_block,
config.snake_block))
        pygame.draw.rect(screen, config.vermelho,
pygame.Rect(food_pos[0], food_pos[1], config.snake_block,
config.snake_block))
        # Mostrar as pontuações
        show_score(player1_name, player1_score, 10, 10)
        show_score(player2_name, player2_score, config.width - 150,
10)  # Ajuste a posição conforme necessário
        # Atualiza a tela
        pygame.display.update()
        clock.tick(config.snake_speed)
    client.close()  # Fecha a conexão com o servidor
    print("Cliente desconectado")
    return winner, game_state
```

Este trecho de código renderiza um valor de texto na tela, exibindo os valores player_name e score usando a fonte font_style. O texto renderizado é então colocado na tela nas coordenadas (x, y).

```
def show_score(player_name, score, x, y):
    value = font_style.render(f"{player_name}: {score}", True,
config.branco)
    screen.blit(value, [x, y])
```

Desenho de elementos do jogo e atualização da tela

Atualizamos o estado do jogo com os dados recebidos e verificamos um vencedor. Isso garante que a lógica do jogo esteja consistente com o servidor.

```
if game_state:
    player_snake = game_state["player1"]["snake"]
```

```
    player1_score = game_state["player1"]["score"]
    other_snake = game_state["player2"]["snake"]
    player2_score = game_state["player2"]["score"]
    food_pos = game_state["food_pos"]
    player1_name = game_state["player1"]["name"]
    player2_name = game_state["player2"]["name"]
    if game_state["player2"]["venceu"]:
       winner = game_state["player2"]["name"]
    elif game_state["player1"]["venceu"]:
        winner = game_state["player1"]["name"]
    else:
        winner = ""
```

Em seguida, desenhamos as cobras, comida e pontuações na tela. Isso visualiza o estado do jogo para os jogadores.

```
for block in player_snake:
    pygame.draw.rect(screen, config.verde, pygame.Rect(block[0],
block[1], config.snake_block, config.snake_block))
for block in other_snake:
    pygame.draw.rect(screen, config.azul, pygame.Rect(block[0],
block[1], config.snake_block, config.snake_block))
pygame.draw.rect(screen, config.vermelho, pygame.Rect(food_pos[0],
food_pos[1], config.snake_block, config.snake_block))
# Mostrar as pontuações
show_score(player1_name, player1_score, 10, 10)
show_score(player2_name, player2_score, config.width - 150, 10)  #
Ajuste a posição conforme necessário
```

Finalmente, atualizamos a exibição e controlamos a velocidade do jogo usando o relógio. O socket do cliente é fechado quando o jogo termina.

```
    # Atualiza a tela
    pygame.display.update()
    clock.tick(config.snake_speed)
client.close()  # Fecha a conexão com o servidor
print("Cliente desconectado")
return winner, game_state
```

Este documento cobriu as principais decisões de design e detalhes de implementação para o jogo de cobra multiplayer online. A funcionalidade aproveita o Pygame para renderização e comunicação por socket para sincronização do estado do jogo entre os jogadores.

Servidor

Introdução

Este documento irá guiá-lo através da implementação da lógica do servidor para um jogo de cobra multiplayer.

A funcionalidade envolve:

- Inicialização dos estados do servidor e dos jogadores.
- Manipulação de conexões dos jogadores.
- Gerenciamento do estado do jogo e interações dos jogadores.
- Atualização do banco de dados com os resultados do jogo.

Inicialização dos estados do servidor e dos jogadores

Começamos definindo o estado inicial do servidor e dos jogadores. Isso inclui rastrear jogadores conectados, conexões dos jogadores e a posição inicial da comida.

```
players_connected = [False, False]
player_connections = {}
server_running = True
food_pos = [random.randrange(1, (width // snake_block)) *
snake_block,
            random.randrange(1, (height // snake_block)) *
snake_block]
```

Em seguida, definimos o estado inicial para cada jogador, incluindo a posição da cobra, direção, pontuação e outros atributos relevantes.

```
players = {
    "player1": {"snake": [[100, 100]], "direction": "STOP", "score":
0, "name": "","id_jogador": 0, "venceu": False},
    "player2": {"snake": [[200, 200]], "direction": "STOP", "score":
0, "name": "", "id_jogador": 0, "venceu": False}
}
vencedor = False
```

Manipulação de conexões dos jogadores

A função `main()` configura o servidor para escutar conexões recebidas. Ela vincula o servidor a um IP e porta e começa a escutar até 2 conexões.

```
def main():
    global players_connected, player_connections, server_running
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(("0.0.0.0", 9999))
    server.listen(2)
    server_ip, server_port = server.getsockname()
    print(f"[Main] Servidor iniciado no IP {server_ip} na porta
{server_port}, aguardando conexões...", flush=True)
```

Em seguida, verificamos as conexões dos jogadores. Se menos de 2 jogadores se conectarem em um minuto, o servidor é desligado.

```
player_count = 0
```

```
threads = []
def check_connections():
    global server_running
    time.sleep(60)
    if sum(players_connected) < 2:
        print("[Main] Não há 2 conexões em 1 minuto. Encerrando o
servidor...", flush=True)
        server_running = False
        for sock in player_connections.values():
            sock.close()
```

Quando um jogador se conecta, atribuímos um slot de jogador e iniciamos uma nova thread para manipular sua conexão.

```
connection_checker = threading.Thread(target=check_connections)
connection_checker.start()
try:
    while server_running:
        server.settimeout(1)
        try:
            client_socket, addr = server.accept()
            player_count += 1
            player = f"player{player_count}"
            print(f"[Main] Conexão aceita de {addr}. Atribuído a
{player}.", flush=True)
```

Atualizamos a lista de jogadores conectados e iniciamos uma thread para lidar com a comunicação com o cliente.

```
players_connected[player_count - 1] = True
player_connections[player] = client_socket
client_handler = threading.Thread(target=handle_client,
args=(client_socket, player))
threads.append(client_handler)
client_handler.start()
```

Assim que ambos os jogadores estão conectados, o loop do jogo continua a rodar enquanto o servidor estiver ativo.

```
        if player_count == 2:
            break
    except socket.timeout:
        continue
while server_running and all(players_connected):
    time.sleep(1)
```

Gerenciamento do estado do jogo e interações dos jogadores

Manipulamos o movimento da cobra e verificamos colisões entre os jogadores. Se uma colisão for detectada, determinamos o vencedor com base nas pontuações e paramos o servidor.

```
snake.insert(0, new_head)
snake.pop()
player1_snake = players["player1"]["snake"]
player2_snake = players["player2"]["snake"]
if check_collision_players(player1_snake, player2_snake):
    print("[Update Snake] Colisão detectada entre player1 e
player2", flush=True)
    if players["player1"]["score"] > players["player2"]["score"]:
        players["player1"]["venceu"] = True
```

```
    elif players["player2"]["score"] > players["player1"]["score"]:
        players["player2"]["venceu"] = True
    vencedor = True
    server_running = False
    return
```

Essas duas funções são as que verificam se um jogador saiu dos limites dateka. Se isso acontecer, o outro jogador é declarado vencedor, e o servidor para.

```
if is_out_of_bounds(player1_snake):
    print("[Update Snake] Player1 saiu da tela", flush=True)
    vencedor = True
    players["player2"]["venceu"]  = True
    server_running = False
    return
if is_out_of_bounds(player2_snake):
    print("[Update Snake] Player2 saiu da tela", flush=True)
    vencedor = True
    players["player1"]["venceu"]  = True
    server_running = False
    return
```

Atualização do banco de dados com os resultados do jogo

Quando o segundo jogador se conecta, inserimos um novo registro na tabela `PARTIC_PARTIDA` com o código `partida` recebido do primeiro jogador conectado.

```
def insere_partic_partida(partida, cd_player1, cd_player2):
    try:
        global cd_participa
        # Conectar ao banco de dados
        connection = connect_to_db()
        cursor = connection.cursor()
        # Buscar o código da última partida
        query = "SELECT MAX(cd_participa) FROM PARTIC_PARTIDA"
        cursor.execute(query)
        result = cursor.fetchone()
        cd_participa = 0
        if result[0]:
            cd_participa = result[0]
        cd_participa += 1
        # Inserir nova partida na tabela PARTIC_PARTIDA
        query = "INSERT INTO PARTIC_PARTIDA(cd_participa
,fk_PARTIDA_cd_partida, fk_cd_player1, fk_cd_player2) value (%s, %s,
%s, %s)"
        cursor.execute(query, (cd_participa, partida, cd_player1,
cd_player2))
        # Confirmar as alterações no banco de dados
        connection.commit()
        # Fechar o cursor e a conexão
        cursor.close()
        connection.close()
    except mysql.connector.Error as err:
        print(err)
```

Após o jogo encerra, atualizamos o status do jogo, o vencedor e a pontuação final de cada jogador no banco de dados.

```
def update_partic_partida(partida, sit_partida, id_player):
    try:
        # Conectar ao banco de dados
        connection = connect_to_db()
        cursor = connection.cursor()
        # Inserir nova partida na tabela PARTIC_PARTIDA
        query = ("UPDATE PARTIC_PARTIDA SET nponts_player1 = %s,
nponts_player2 = %s "
                 "WHERE cd_participa = %s")
        cursor.execute(query, (int(players["player1"]["score"]),
int(players["player2"]["score"]), cd_participa,))
        # Confirmar as alterações no banco de dados
        connection.commit()
        query = "UPDATE PARTIDAS SET dt_fim = CURRENT_TIMESTAMP,
cd_vencedor = %s, ctpo_sit = %s WHERE cd_partida = %s"
        if id_player == 0:
            id_player = None
        cursor.execute(query, (id_player,sit_partida,partida,))
        # Confirmar as alterações no banco de dados
        connection.commit()
        # Fechar o cursor e a conexão
        cursor.close()
        connection.close()
    except mysql.connector.Error as err:
        print(err)
```

Finalização e fechamento do servidor

Finalmente, lidamos com o processo de desligamento do servidor. Se um vencedor for determinado, atualizamos o banco de dados de acordo. Em seguida, fechamos os threads e o socket do servidor.

```
finally:
    if vencedor:
        if players["player1"]["venceu"]:
            id_vencedor = int(players["player1"]["id_jogador"])
        elif players["player2"]["venceu"]:
            id_vencedor = int(players["player2"]["id_jogador"])
        else:
            id_vencedor = 0
        if id_vencedor > 0:
            update_partic_partida(cd_partida,1,id_vencedor)
        else:
            update_partic_partida(cd_partida, 1, 0)
    else:
        update_partic_partida(cd_partida, 3, 0)
    print("[Main] Encerrando o servidor...", flush=True)
    server_running = False
    for t in threads:
        t.join(timeout=1)
    server.close()
    print("[Main] Servidor fechado.", flush=True)
    exit()
```

O servidor é fechado, e o programa é encerrado.

```
if __name__ == "__main__":
```

```
    try:
        main()
    except KeyboardInterrupt:
        server_running = False
        print("\n[Main] Servidor interrompido pelo usuário.",
flush=True)
        for sock in player_connections.values():
            sock.close()
```

Isso conclui a implementação do servidor para o jogo de cobra multiplayer.

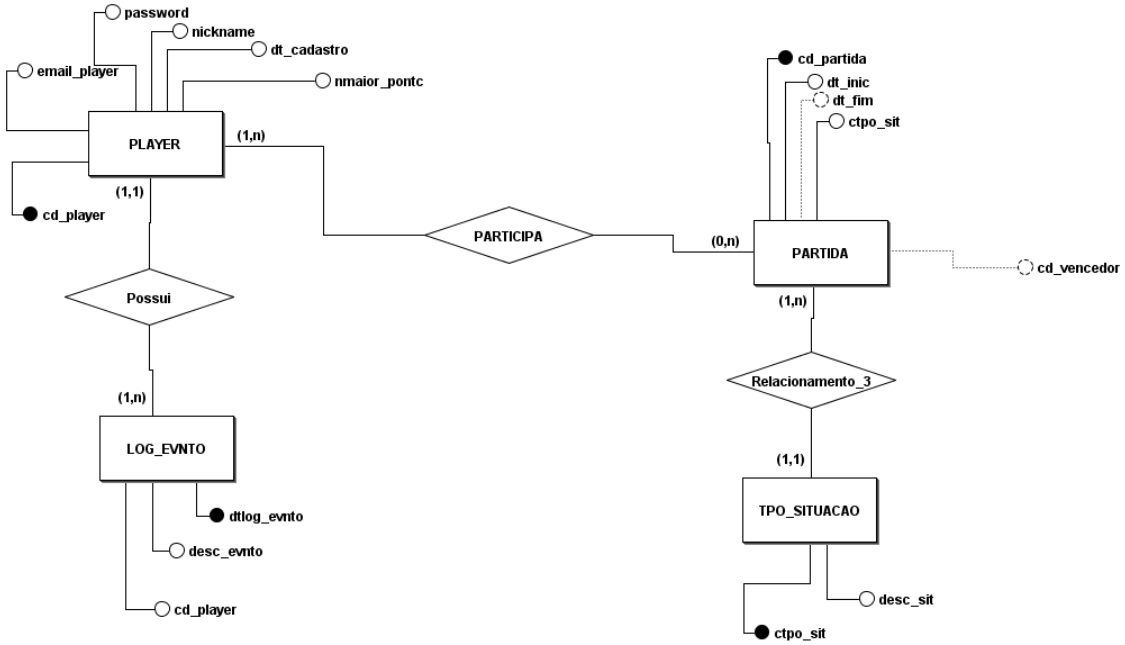
Modelo de Dados

Modelo Conceitual

Este é um modelo de dados conceitual para o sistema de gerenciamento de jogadores e partidas do jogo Snake.

1. **PLAYER**: Representa os jogadores no sistema. Os principais atributos incluem um código único do jogador, email, senha, nickname, data de cadastro e a maior pontuação do jogador.
2. **LOG_EVTNO**: Armazena os eventos de log relacionados aos jogadores. Inclui um código de evento, data do evento, descrição e um código de jogador (que referencia a tabela PLAYER).
3. **PARTIDA**: Representa as partidas do jogo. Os atributos principais são o código da partida, data de início e fim, código da situação da partida, e código do vencedor.
4. **TPO_SITUACAO**: Define os tipos de situação das partidas, como "em andamento" ou "concluída". Possui um código e uma descrição da situação.
5. **Relacionamentos**:
    - **Possui**: Liga os jogadores aos seus logs de eventos.
    - **PARTICIPA**: Liga os jogadores às partidas em que participaram.
    - **Situação**: Relaciona as partidas aos tipos de situação.

Esse modelo ajuda a entender como os dados de jogadores, eventos e partidas estão interconectados, simplificando o gerenciamento e a consulta das informações no sistema.



Modelo Lógico

1. **PLAYER**
    - **email_player:** VARCHAR(100)

- **password:** VARCHAR(20)
- **nickname:** VARCHAR(15)
- **dt_cadastro:** DATE
- **nmaior_ponte:** INTEGER
- **cd_player:** INTEGER (Primary Key)

2. **LOG_EVNT**
   - **dtlog_evnto:** TIMESTAMP (Primary Key)
   - **cd_player:** INTEGER (Foreign Key)
   - **desc_evnto:** VARCHAR(255)
3. **PARTIC_PARTIDA**
   - **fk_PARTIDA_cd_partida:** INTEGER (Foreign Key)
   - **fk_cd_player1:** INTEGER (Foreign Key)
   - **nponts_player1:** INTEGER
   - **fk_cd_player2:** INTEGER (Foreign Key)
   - **nponts_player2:** INTEGER
   - **cd_participa:** INTEGER (Primary Key)
4. **PARTIDAS**
   - **cd_partida:** INTEGER (Primary Key)
   - **dt_inic:** TIMESTAMP
   - **dt_fim:** TIMESTAMP
   - **ctpo_sit:** INTEGER (Foreign Key)
   - **cd_vencedor:** INTEGER (Foreign Key)
5. **TPO_SITUACAO**
   - **ctpo_sit:** INTEGER (Primary Key)
   - **desc_sit:** VARCHAR(255)

Relacionamentos

- **PLAYER** para **LOG_EVNT:** Um jogador pode ter muitos eventos de log (1,n).
- **PLAYER** para **PARTIC_PARTIDA:** Um jogador pode participar de muitas partidas (1,n).
- **PARTIC_PARTIDA** para **PARTIDAS:** Muitas participações pertencem a uma partida (n,1).
- **PARTIDAS** para **TPO_SITUACAO:** Muitas partidas podem ter um tipo de situação (n,1).

Este modelo é mais detalhado e específico para implementação do sistema de banco de dados relacional. Cada entidade corresponde a uma tabela, e os relacionamentos são implementados através de chaves estrangeiras, garantindo a integridade referencial e organizando como os

dados são interligados.