IC/UFAL - Instituto de Computação

Erick Keven da Silva Alves
Sandro Gomes Paulino

Projeto de Redes de Computadores

Maceió-AL
2024

Introdução

Este é um projeto desenvolvido como parte da disciplina de Redes de Computadores. Consiste em uma implementação do jogo da velha utilizando a biblioteca socket e threads em Python. O jogo da velha é um clássico passatempo para dois jogadores, onde cada um deles marca alternadamente espaços em uma grade de 3x3. O objetivo é conseguir uma linha, coluna ou diagonal com três símbolos iguais (X ou O). Neste projeto, o jogo da velha foi implementado como uma aplicação cliente-servidor. O servidor gerencia as conexões entre os jogadores e coordena o progresso do jogo, enquanto os clientes (os jogadores) se conectam ao servidor para jogar entre si. Este projeto visa fornecer uma experiência prática na implementação de conceitos de redes de computadores, como comunicação entre processos através de sockets e o uso de threads para permitir a interação simultânea de 2 jogadores.

Como executar o Jogo:

No Windows

- Certifique-se de ter o Python instalado em sua máquina. Você pode baixá-lo em python.org.

No Linux

- Para instalar o Python3 no Linux, execute os seguintes comandos no terminal:
  sudo apt update
  sudo apt install python3

Executando o Servidor e o Cliente

1. Clonar ou Baixar:
   Para clonar:
   git clone https://github.com/sandrogomes7/jogo-da-velha.git
   Se quiser baixar apenas, clique em CODE e clique em *DOWNLOAD ZIP*
2. Executar o Servidor: No terminal, execute o seguinte comando para iniciar o servidor:
   python3 ./servidor.py *IP do Computador*

3. Executar o Cliente: No terminal, execute o seguinte comando para iniciar o cliente:
   python3 ./cliente.py *IP do Servidor*

Principais funcionalidades:

OBS: Aqui apenas estão partes do código, para vê-lo completo, acesse o GitHub no final do PDF

Indo para o código do servidor, começando da main

```python
# Main
# Cria o socket do servidor com IPv4 e TCP AQUI
socketServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Permite que o servidor reutilize a porta AQUI
socketServer.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

socketServer.bind((HOST, PORTA))

print("Servidor iniciado!\n")
print("IP: " + HOST + "\n")

while True:
    socketServer.listen(3)
    print("Esperando conexões/jogada
                                    (variable) socketServer: socket
    (conexaoCliente, (IP, PORTA)) = socketServer.accept()

    # Verifica se o servidor já está cheio
    if jogadores[JOGADOR1] and jogadores[JOGADOR2]:
        print("Um intruso tentou conectar no servidor! Conexão negada!\n")
        pacote = {"casas": "{}", "mensagem": "Jogo permitido apenas para 2 jogadores!", "status": "INTRUSO"}
        conexaoCliente.send(json.dumps(pacote).encode())
        conexaoCliente.close()
        continue
    else:
        novaThread = ClienteThread(IP, PORTA, conexaoCliente)
        novaThread.start()
```

Essa parte do código inicializa o servidor, configurando um socket TCP/IP para ouvir conexões em um endereço IP e porta específicos. Após a inicialização, o servidor entra em um loop infinito, esperando por conexões de clientes. Quando uma conexão é aceita, verifica se já há dois jogadores conectados. Se o servidor já estiver cheio, recusa a conexão do cliente e avisa ao cliente que um intruso tentou conectar no servidor. Caso contrário, cria uma nova thread para lidar com as interações do cliente aceito.

```python
# Classe para criar threads
class ClienteThread(threading.Thread):
    # Inicializa a thread
    def __init__(self, ip, porta, clienteSocket):
        threading.Thread.__init__(self)
        self.ip = ip
        self.porta = porta
        self.conexaoCliente = clienteSocket
        print("Nova thread iniciada: " + ip)

        # Escolhe aleatoriamente o jogador que irá começar
        global primeiroJogador
        jogador_atual = random.choice([JOGADOR1, JOGADOR2])

        if jogadores[jogador_atual]:
            jogador_atual = JOGADOR2 if jogador_atual == JOGADOR1 else JOGADOR1

        if (jogador_atual == JOGADOR1 and primeiroJogador == ""):
            primeiroJogador = JOGADOR1
        elif (jogador_atual == JOGADOR2 and primeiroJogador == ""):
            primeiroJogador = JOGADOR2

        # Atualiza o dicionário com as informações do jogador atual
        print(f"Conexão do {jogador_atual} estabelecida.")
        jogadores[jogador_atual].update({"conexao": {}})
        jogadores[jogador_atual]["conexao"]["ip"] = self.ip
        jogadores[jogador_atual]["conexao"]["porta"] = self.porta
        jogadores[jogador_atual]["conexao"]["socket"] = self.conexaoCliente
        jogadores[jogador_atual]["nome"] = jogador_atual
        jogadores[jogador_atual]["status"] = OBT_NOME
```

ClienteThread, que é uma subclasse de threading ela é responsável por lidar com as interações de um cliente específico em uma thread separada. No método __init__, a inicialização da thread é configurada. Os parâmetros de entrada incluem o IP, a porta do cliente, e o socket de conexão com o cliente (clienteSocket). Durante a inicialização, é escolhido aleatoriamente qual jogador será atribuído ao cliente atual. Isso é feito para distribuir igualmente as oportunidades de iniciar o jogo entre os jogadores.

```python
    # Função para executar a thread e receber as requisições do cliente
    def run(self):
        global primeiroJogador
        print("Conexão de : " + self.ip)
        mensagem = "Bem-vindo ao servidor.\n\n"

        # Servidor envia primeira requisição para o cliente
        if primeiroJogador == JOGADOR1:
            if not jogadores[JOGADOR2] and jogadores[JOGADOR1]["nome"] == JOGADOR1:
                enviarRequisicao(mensagem, JOGADOR1, OBT_NOME)

            if jogadores[JOGADOR2] and jogadores[JOGADOR2]["nome"] == JOGADOR2:
                enviarRequisicao(mensagem, JOGADOR2, OBT_NOME)
        else:
            if not jogadores[JOGADOR1] and jogadores[JOGADOR2]["nome"] == JOGADOR2:
                enviarRequisicao(mensagem, JOGADOR2, OBT_NOME)

            if jogadores[JOGADOR1] and jogadores[JOGADOR1]["nome"] == JOGADOR1:
                enviarRequisicao(mensagem, JOGADOR1, OBT_NOME)
        try:
            while True:
                pacote = self.conexaoCliente.recv(2048) # Recebe as requisições do cliente
```

Uma parte do Run, este método é responsável por lidar com a comunicação entre o servidor e um cliente específico, processando as jogadas do cliente, atualizando o estado do jogo seja para definir o nome,aguardar sua vez do jogo,atualizar jogadas e gerenciando as desconexões dos clientes.

Indo agora para a parte do código do cliente temos:

No código, a conexão com o servidor é estabelecida usando a função socket.connect() e um socket é criado e conectado ao endereço do servidor especificado pelo host e a porta.

```
# Conexão com o servidor
clienteSocket = socket.socket()
clienteSocket.connect((HOST, PORTA))
```

Essa funcionalidade é implementada na parte do código que recebe pacotes do servidor, decodifica-os e atualiza o tabuleiro do jogo cliente recebe um pacote de dados do servidor, decodifica o JSON contido no pacote e atualiza o tabuleiro do jogo com as informações recebidas.

```
# Recebendo todas as respostas do servidor
while True:
    pacote = clienteSocket.recv(2048) # Recebendo pacote do servidor

    if not pacote:
        print("Desconectado do servidor")
        break

    pacote = json.loads(pacote.decode()) # Decodificando pacote
    jogo.casas = pacote["casas"]         # Atualizando tabuleiro
```

Indo para a última parte do codigo que é a ClasseJogo:

```
# Mostra o tabuleiro
def mostrarTab(self):
    print("\n\t╔═══════════════════════════════════════╗")
    print("\t║                                       ║")
    print("\t║              Jogo da Velha            ║")
    print("\t║                                       ║")
    print("\t║                                       ║")
    print("\t║                ┌───┬───┬───┐          ║")
    print("\t║                │ %s │ %s │ %s │          ║" % (self.casas[0], self.casas[1], self.casas[2]))
    print("\t║                ├───┼───┼───┤          ║")
    print("\t║                │ %s │ %s │ %s │          ║" % (self.casas[3], self.casas[4], self.casas[5]))
    print("\t║                ├───┼───┼───┤          ║")
    print("\t║                │ %s │ %s │ %s │          ║" % (self.casas[6], self.casas[7], self.casas[8]))
    print("\t║                └───┴───┴───┘          ║")
    print("\t║                                       ║")
    print("\t║                                       ║")
    print("\t╚═══════════════════════════════════════╝")
    print()
```

Nessa parte do código temos uma função logo acima de limpar a tela do console, utilizando o módulo os para detectar o sistema operacional.
Temos também o mostrarTab que exibi o tabuleiro atual do jogo na tela do console e que criou uma aparência do tabuleiro do jogo da velha.
Abaixo temos as partes do código para verificar o Vencedor, onde ele verifica todas as possibilidades de vitória(linhas,colunas e diagonais) e ele retorna o vencedor.
Temos também a parte para verificar se o jogo deu velha que ele verifica se todas as casas do tabuleiro estão preenchidas e nenhum jogador venceu e por fim temos a função de resetar, onde é utilizado para redefinir o tabuleiro do jogo.

Principais melhorias:

1. Integrar um banco de dados para armazenar informações sobre jogadores, incluindo seus nomes, pontuações e histórico de vitórias, permitindo que os jogadores acompanhem seu progresso e comparem seus resultados com outros.
2. Aprimorar a experiência do usuário implementando uma interface gráfica mais amigável e interativa permitindo que os jogadores personalizem a aparência do jogo, escolhendo entre diferentes temas, estilos de tabuleiro,proporcionando uma experiência visualmente única e cativante
3. Implementar uma IA mais sofisticada que ofereça diferentes níveis de dificuldade para os jogadores enfrentarem, desde iniciantes até especialistas, proporcionando uma experiência desafiadora e divertida para todos os níveis de habilidade.

Dificuldades encontradas:

Dificuldade em implementar as funcionalidades do jogo da velha devido à inexperiência em Python, especialmente na parte de enviar e receber as jogadas usando sockets.

GITHUB: https://github.com/sandrogomes7/jogo-da-velha