INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE FEDERAL DE ALAGOAS - UFAL

LÍVIA ANDRESSA DA SILVA SANTOS
MARIA APARECIDA DA SILVA NASCIMENTO

Jogo da Velha

Link do repositório do projeto: Jogo-da-velha

MACEIÓ
2023

Introdução

O projeto em questão é um jogo da velha multiplayer desenvolvido em Python, aproveitando a tecnologia de sockets para criar uma conexão cliente-servidor usando o protocolo TCP. A aplicação oferece uma experiência interativa em tempo real, permitindo que jogadores em diferentes computadores compitam no clássico jogo da velha. Além da funcionalidade multiplayer, o jogo conta com uma interface gráfica intuitiva atualizada em tempo real por meio de threads, proporcionando uma experiência de usuário fluida e envolvente. Este relatório tem como objetivo mostrar os principais pontos do projeto.

Como executar o projeto

Para executar o projeto localmente, você é necessário ter o Python instalado em sua máquina e algumas dependências. Para fazer isto, os seguintes comandos devem ser feitos:

1. **python3 -m venv env**
   cria um ambiente virtual na pasta do projeto
2. **. env/bin/activate**
   Ativa o ambiente virtual criado
3. **pip install -r requirements.txt**
   Instala as dependências

Após a instalação ser concluída, é necessário iniciar o servidor da aplicação com os seguintes comandos:

1. **python server.py**
   Inicia o servidor (é necessário estar com o ambiente virtual ativo)

Por padrão, o servidor estará rodando na porta 5000. Para mudar isso basta alterando o arquivo server.py trocando para a porta desejada. Depois de iniciar o servidor o cliente pode ser iniciado usando o comando:

1. **python main.py**
   lembre-se de ativar o ambiente virtual no outro dispositivo (ou terminal) também

Agora, com o cliente em execução, ao clicar em "play", o jogo estabelecerá automaticamente uma conexão com o servidor. O cliente aguardará a conexão do oponente, iniciando o jogo somente quando ambos os jogadores estiverem conectados. O jogo será encerrado instantaneamente se ocorrer desconexão por parte de qualquer um dos jogadores.

Principais funcionalidades

Um jogador pode se conectar ao servidor, e esperar o oponente se conectar;

```
self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    self.client.connect(('localhost', 5000))
    response = self.client.recv(1024).decode()

    if response == 'Server is full':
        print(response)
        sys.exit()

    self.player, self.moveType = response.split(';')
    print(f'You are player {self.player}')
except Exception as e:
    if e == KeyboardInterrupt:
        print('Shutting down...')
    else:
        print('Server is offline')
    sys.exit()

if self.player == 'P1':
    self.worker = SecondPlayerWorker(client=self.client)
    self.worker.found_second_player.connect(self.startGame)
    self.worker.found_second_player.connect(self.worker.deleteLater)
    self.worker.start()
else:
    res = self.client.recv(1024).decode()
    print(res)
    if res == 'Starting game...':
        self.startGame()
    else:
        sys.exit()
```

O servidor pode receber uma jogada e enviar aos demais jogadores;

```
def threaded_client(connection, addr):
    while True:
        try:
            data = connection.recv(1024)
            if not data:
                connection.sendall(str.encode('Server is shutting down'))
                break
            reply = f'{data.decode()}'
            for player in players:
                player.sendall(str.encode(reply))
        except:
            break
```

O cliente pode colocar uma jogada no tabuleiro (a própria ou a do oponente)

```
def setPlay(self, i, j, moveType):
    self.currentBtn = self.findChild(QPushButton, f'E{i}{j}')
    if not self.currentBtn or (not self.game_worker.yourTurn and moveType == self.moveType):
        return

    if moveType == self.moveType:
        self.client.send(f'{i};{j};{moveType}'.encode())

    self.fadeIn(self.currentBtn)

    # save moves-----------------------

    if moveType == "X":
        self.game_worker.xMoves.append(f'{i}{j}')
    else:
        self.game_worker.oMoves.append(f'{i}{j}')

    self.currentBtn.setIcon(QIcon(self.xmap if moveType == 'X' else self.omap))
```

Sobre o projeto

No processo de desenvolvimento, o jogo foi construído utilizando Python, com a biblioteca **sockets** para a parte do servidor e a biblioteca **_thread** para implementar o multithreading. Na parte do cliente, foi adotada a biblioteca PyQt5 para a interface gráfica, e para o multithreading, foi utilizado o componente QThread, integrado à própria biblioteca de interface gráfica. Essa abordagem permitiu criar threads dedicadas para aguardar as respostas do servidor de forma eficiente.

No lado do servidor, o protocolo TCP foi implementado na camada de transporte, utilizando multithreading para facilitar a conexão entre os dois jogadores. Um mecanismo de verificação foi

incorporado para evitar a entrada de um terceiro jogador, desconectando-o automaticamente do servidor com a mensagem "O servidor já está cheio".

A lógica do jogo foi distribuída entre os clientes, cada um responsável por verificar o estado do jogo, determinando se o jogo terminou (indicando vitória de um dos jogadores ou empate). Em cada jogada, o cliente envia sua posição e o símbolo jogado ('X' ou 'O') para o servidor. Nesse modelo, o servidor atua principalmente como um intermediário das jogadas, transmitindo as informações relevantes para o oponente.

Vale ressaltar que essa abordagem, embora funcional, pode não ser a mais escalável a longo prazo. Em futuras implementações, considerar o desacoplamento da lógica do jogo do cliente, transferindo essa responsabilidade para o servidor.

Futuras Implementações:

- Protocolo Mais Organizado: Uma melhoria crucial seria a criação de um protocolo de comunicação mais estruturado e definido. Isso garantiria uma transmissão de mensagens mais organizada, especialmente em relação às informações dos jogadores e ao estado do jogo. Um protocolo bem definido facilitaria a interpretação e processamento das mensagens, tornando a comunicação mais eficiente e menos propensa a erros.
- Refatoração do Servidor como Controlador: Uma modificação fundamental seria transferir a lógica do jogo para o servidor. Atualmente, a lógica está distribuída entre os clientes, o que pode levar a discrepâncias nos resultados entre os lados. Refatorar o código para que o servidor atue como o controlador da lógica do jogo garantiria consistência nos resultados e evitaria inconsistências entre os jogadores.
- Implementação do Botão "Restart": Adicionar um botão de "restart" à interface do jogo seria uma melhoria de usabilidade significativa. Atualmente, os jogadores precisam encerrar a conexão e criar uma nova para re-jogar com os mesmos participantes, o que pode ser inconveniente. Um botão "restart" permitiria reiniciar o jogo com os mesmos jogadores de forma rápida e simples, melhorando a experiência do usuário.
- Refatoração do Envio de Jogadas no Servidor: No servidor, reestruturar o código para que ele envie apenas as jogadas do cliente ao seu oponente, e exclusivamente ao seu oponente, seria uma otimização importante. Isso reduziria o tráfego de rede e melhoraria a eficiência, garantindo que as informações do jogo sejam transmitidas apenas para quem realmente precisa delas, evitando excesso de dados na rede.

Referências

[1] - Acervo Lima. Sockets em Python. Disponível em: <https://acervolima.com/sockets-python/>. Acesso em: 27 de setembro de 2023

[2] - Python Software Foundation. Python Documentation. Disponível em: <https://www.python.org/doc/>. 27 de setembro de 2023.

[3] - Progressivo. Como Criar Jogo da Velha em Python. Disponível em: <https://www.pythonprogressivo.net/2018/10/Como-Criar-Jogo-Velha-Python.html>. 29 de setembro de 2023.