Universidade Federal de Alagoas - UFAL

Instituto de Computação - IC

Relatório do projeto de Redes de Computadores

Iasmin Cristina Borba
Jhordan Borges Almeida de Lacerda
João Lucas Oliveira Costa
José Alexandre da Silva França

Maceió - 2023

Repositório do projeto: GitHub

Descrição

O projeto é um jogo da velha multiplayer feito em **Python**. Graças ao uso de **sockets**, o jogo pode ser jogado em diferentes computadores, um deles atuará como um servidor que outro cliente pode acessar para jogar. A conexão é feita utilizando a arquitetura **cliente-servidor**, utilizando sockets com o protocolo **TCP**. A aplicação conta com uma interface gráfica que é vista em ambos os computadores e atualizada em tempo real, usando o conceito de **threads**.

O jogo da velha é um jogo simples que pode ser jogado em qualquer lugar, precisando apenas de um papel e caneta. O tabuleiro do jogo consiste em um quadrado dividido em 9 partes (casas). É necessário dois jogadores, que são representados cada um por um símbolo **'X'** ou '**O'**.

O jogo é dividido em turnos, cada turno é jogado por um participante que marca seu símbolo em uma das casas não marcadas do tabuleiro. O jogo acaba quando não houver mais espaços (casas) disponíveis para marcar (**EMPATE**) ou quando um dos jogadores conseguir marcar 3 de seus símbolos em uma linha reta (**VITÓRIA**), essa linha pode ser na *horizontal*, *vertical* ou *diagonal*.

Como executar

Para executar este projeto localmente, você precisará ter o Python instalado em sua máquina. Com o Python instalado, você precisará instalar as dependências. Para fazer isso, execute os seguintes comandos:

```
# Criando um ambiente virtual na pasta do projeto
python3 -m venv env # ou python -m venv env, dependendo
  do seu sistema operacional

# Ativando o ambiente virtual
. env/bin/activate # ou env/Scripts/activate, dependendo
  do seu sistema operacional

# Instalando as dependências
pip install -r requirements.txt
```

Com as dependências instaladas, você precisará iniciar o servidor da aplicação. Para fazer isso, execute o seguinte comando:

```
# Iniciando o servidor
python server.py # lembre-se de possuir o ambiente
virtual ativo
```

Por padrão, o servidor estará rodando na porta ***5000***. Se você quiser mudar isso, você pode fazer isso alterando o arquivo ***server.py***. Depois de iniciar o servidor, você pode iniciar o cliente. Para fazer isso, execute o seguinte comando:

```
  # Iniciando a interface gráfica
  python main.py # lembre-se de ativar o ambiente virtual
    no outro dispositivo (ou terminal) também
```

Agora com o cliente rodando, ao pressionar ***"play"***, o jogo se conectará ao servidor, e irá esperar pelo oponente se conectar também, começando o jogo quando os dois estiverem conectados, e se encerrando se qualquer um dos dois se desconectar.

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':
        sys.exit()
    self.player, self.moveType = response.split(';')
    print(f'You are player {self.player}')
    if self.player == 'P1':
        print('Waiting for second player...')
        self.keep_running = True
        while self.keep_running:
            res = self.client.recv(1024).decode()
            print(res)
            if res == 'Starting game...':

                ...
```

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

```
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):
    if moveType == self.moveType:
        self.client.send(str(i).encode() + str(j).encode()
    + moveType.encode())


    if moveType == "X":
        self.game_worker.xMoves.append(str(i)+str(j))
    else:
        self.game_worker.oMoves.append(str(i)+str(j))


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

    ...
```

- O cliente pode mostrar o resultado do jogo.

```
self.winMoves =[
            ["00", "11", "22"], # diagonal 1
            ["20", "11", "02"], # diagonal 2
            ["00", "01", "02"], # horizontal 0
            ["10", "11", "12"], # horizontal 1
            ["20", "21", "22"], # horizontal 2
            ["00", "10", "20"], # vertical 0
            ["01", "11", "21"], # vertical 1
            ["02", "12", "22"]  # vertical 2
        ]

def checkGameStatus(self):
    for move in self.winMoves:
        if all(item in self.gw.oMoves for item in move):
            self.endGame("O")
            return
        elif all(item in self.gw.xMoves for item in move):
            self.endGame("X")
            return
    if len(self.gw.oMoves) + len(self.gw.xMoves) >= 9:
        self.endGame("draw")
    ...
```

Desenvolvimento

O jogo foi feito com Python, na parte do servidor usando a biblioteca ***sockets*** e para o multithreading, a biblioteca ***_thread***. Já na parte do cliente, foi utilizada a biblioteca ***PyQt5*** para a interface gráfica, e para o multithreading, foi utilizado um componente da própria biblioteca de interface gráfica, ***QThread***, que cuidou da criação de uma thread para esperar as respostas do servidor.

Na parte do servidor, foi implementado o protocolo **TCP** na camada de transporte, usando o multithread para a conciliação da conexão dos dois jogadores, verificando se houvesse um terceiro, o mesmo seria desconectado do servidor com uma mensagem de "O servidor já está cheio".

A forma como está implementada, cada cliente possui sua lógica de verificação do jogo, como se ele já está acabado (significando que um dos jogadores tenha vencido, ou houve um empate), e enviando, a cada jogada, sua posição e qual símbolo foi jogado ('**X**' ou '**O**').

Portanto, o servidor acabou servindo apenas de mediador das jogadas, enviando as informações da jogada para o oponente. (Como será dito nas "**Futuras implementações**", esse modo pode não ser o melhor para uma escalabilidade da aplicação, uma vez que a lógica poderia ter sido desacoplada do cliente, e ter sido um trabalho para o servidor cuidar.)

Futuras implementações

- Criar um protocolo para o envio das mensagens mais organizado e definido (principalmente de informações sobre o jogador e o estado do jogo);
- Refatorar o código para com que o servidor seja o controlador da lógica do jogo, e não o próprio cliente (isso poderia evitar erros onde um dos lados tivesse um resultado diferente do outro);
- Implementar um botão de "**restart**", pois atualmente é necessário encerrar a conexão e criá-la novamente para re-jogar com os mesmos participantes;
- Refatorar o código do servidor, para que ele envie apenas as jogadas do cliente ao seu oponente, e somente ao seu oponente.

Dificuldades

- Elaborar um protocolo efetivo a fim de tornar a comunicação cliente-servidor escalável para futuras aprimorações do jogo;
- Conciliar as threads, principalmente da interface gráfica, para que a aplicação continue rodando ao esperar pela jogada do oponente.