

UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES
Redes de Computadores

ANDREAS HUKUHARA CHRISTE - 13673110
NICOLAS PEREIRA RISSO VIEIRA - 13672262
PEDRO PALAZZI DE SOUZA - 13748012

Relatório EP1

São Paulo

2023

Proposta

Um Game de Pedra, Papel e Tesoura desenvolvido em Python utilizando o modelo Cliente-Servidor. Um jogador conecta-se ao servidor, seleciona um 'Nick' e espera a próxima pessoa se conectar. Quando ambos estiverem prontos, uma interface com as opções aparece e novamente o jogo aguarda até que ambos tenham selecionado sua jogada para continuar para o resultado.

Quando um jogador selecionar sua opção, uma mensagem é enviada, notificando o outro jogador de que seu oponente já escolheu. Além disso, durante o jogo ou antes mesmo da partida, ambos podem se comunicar com um chat pelo terminal.

Nesta arquitetura, estruturamos um hub para aguardar que ambos os jogadores estejam prontos para a partida. O jogo também notifica desconexões e aceita mais de um participante em seu chat. Existem cores para diferenciar mensagens de terceiros da sua própria.

Protocolo e mensagens

A conexão TCP (Transmission Control Protocol) é um protocolo de comunicação confiável e orientado à conexão, amplamente utilizado na arquitetura cliente-servidor para garantir a entrega ordenada e sem erros de dados entre dispositivos em uma rede. No contexto de um jogo como o Pedra, Papel e Tesoura, essa conexão é fundamental para estabelecer uma comunicação estável e precisa entre o cliente (jogador) e o servidor que hospeda o jogo. A aplicação do TCP permite que o cliente envie seus movimentos (seleção de pedra, papel ou tesoura) para o servidor de forma confiável, enquanto o servidor processa essas ações e as comunica de volta ao cliente, mantendo a sincronização e a consistência do jogo. O uso do TCP nesse jogo específico assegura que as interações dos jogadores sejam transmitidas corretamente e que ambos os lados permaneçam atualizados sobre o estado atual da partida, proporcionando uma experiência de jogo coesa e interativa.

Abaixo estão descritas e explicadas todas as mensagens que são feitas entre cliente e servidor juntamente com em qual linha do código são encontradas:

1. **Conectar com o servidor do chat** (rockPaperScissors/chat/client.py, linha 64):

O socket é conectado ao servidor.

2. **Enviar Mensagem para o servidor** (rockPaperScissors/chat/client.py, linha 95):

Após digitar algo no terminal, o código envia a mensagem para o servidor.

3. **Enviar Mensagem para o cliente** (rockPaperScissors/(chat/server.py, linha 108):

Após receber a mensagem de um usuário, a repassa para todos os outros conectados ao servidor.

4. **Verificar se o ip informado pelo cliente existe** (rockPaperScissors/client.py, linha 40):

O jogo tentará se conectar brevemente ao servidor, apenas para verificar se o usuário passou os dados corretos para efetivar a conexão.

5. **Procurando sala** (rockPaperScissors/client.py, linha 126):
Envia mensagem para o servidor avisando que está procurando uma sala para jogar.
6. **Avisa que já jogou** (rockPaperScissors/client.py, linha 136):
Comunica o servidor que está esperando o jogo reiniciar, já que ambos os jogadores já jogaram.
7. **Enviar jogada** (rockPaperScissors/client.py, linha 173 ou 176):
Envia ao servidor uma string “Rock” ou “Paper” ou “Scissor” para determinar sua jogada.
8. **Envia a partida pro cliente** (rockPaperScissors/server.py, linha 36):
Mensagem confirmando que achou a sala e conectando o player a ela.
9. **Envia mensagem com resultado** (rockPaperScissors/server.py, linha 54):
Envia mensagem com a escolha do oponente. O próprio cliente contabiliza o resultado.
10. **Conexão perdida** (rockPaperScissors/server.py, linha 62):
Ao deletar a sala o socket envia uma mensagem de que a conexão foi perdida para o cliente, assim, voltando para o menu principal.

Sobre o código

O código consiste em um jogo de Pedra Papel Tesoura feito em Python com a ajuda da biblioteca pygame, responsável pela parte gráfica. O projeto possui a seguinte estrutura, tendo todo o código responsável por fazer o jogo funcionar dentro da pasta rockPaperScissors:

```
/rockPapersScissors
>/chat
>>client.py
>>colors.py
>>server.py
>/classes
>>button.py
>>inputbox.py
>>text.py
>client.py
>game.py
>network.py
>server.py
HowToCompile.txt
README.md
```

A seguir será feita uma breve explicação de cada parte do código, o foco das explicações está em generalizar o que cada script faz para proporcionar um entendimento melhor da lógica utilizada na programação desse jogo, partes que não são focos do trabalho, como a lógica por trás da funcionalidade do game e a implementação gráfica, serão abordadas de maneira simples e direta..

1. **chat/client.py:** É onde o cliente se conecta, recebe e envia mensagens com o servidor do chat.
2. **chat/colors.py:** É uma classe feita para facilitar o manuseamento de cores em python.
3. **chat/server.py:** É onde o servidor do chat realiza sua lógica (para distribuir corretamente entre os outros jogadores os dados recebidos) e mensagens.
4. **classes/button.py:** Classe que possui um botão para ser usado no jogo.
5. **classes/inputbox.py:** Classe que possui o inputbox utilizado pelo jogo.
6. **classes/text.py:** Classes que possui os textos utilizados pelo jogo (referente à parte visual).
7. **client.py:** O jogo do usuário em si, possui 3 interfaces. A do menu para passar o seu username, o ip do servidor e a porta, a de procurar sala em que espera a conexão com um lobby para jogar e a tela do jogo em si, em que joga pedra papel tesoura. Também cria uma thread para executar o cliente do chat enquanto todo o código do jogo roda, a fim de não pausar o game para ler o chat..
8. **game.py:** Classe feita para realizar a lógica do jogo. Existe para organizar o projeto.
9. **network.py:** Classe feita para realizar as mensagens de socket do jogo. Existe para organizar o projeto. Basicamente resume o código, no lugar de digitar “client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)” pode-se utilizar comandos mais diretos e simples, como: “client_socket = Network(server=ip, port=porta)”.
10. **server.py:** Servidor que hospeda o jogo em si, também cria uma thread para executar o servidor do chat. No início de sua execução pode-se fornecer o ip que deseja utilizar para hospedar o servidor tão como qual porta será utilizada.

Como executar

1. O Pedra Papel Tesoura (PPT) precisa de pacotes extras, instale-os:
pip install pygame
2. Vá até o diretório do PPT:
cd rockPapersScissors
3. Para rodar o PPT é necessário primeiro iniciar o servidor (isso já na pasta do rockPapersScissors):
python server.py
4. Depois, em outros 2 terminais, execute os clients:
python client.py

Testes

Para melhor visualização dos procedimentos e mensagens utilizadas no exercício programa, observe as imagens abaixo:

A figura 1 identifica o processo de execução do servidor sendo necessário inicialmente identificar o ip do Servidor que será hospedado e o número da porta em que os serviços irão trafegar, após isso o servidor aguarda conexão de clientes.

Repare que o servidor dá feedback tanto para sua conexão com o Game (destacada em verde), quanto para sua conexão com o Chat (destacada em magenta), isso foi feito para facilitar a visualização e interpretação dos dados que estão entrando e saindo do servidor em si.

```
PS C:\Users\DELL\Documents\USP\Semestre-4\redes\ep1-redes\EP1_Redes\rockPaperScissors> python server.py
Digite o seu ip para hospedar o Servidor: localhost
Digite a porta do Servidor: 6969
[GAME]> Waiting for a connection, Server Started
[GAME]> Conectado à: ('127.0.0.1', 61442)
[GAME]> Criando novo jogo...
[GAME]> Conexão Perdida
[GAME]> Fechando Jogo 0
[CHAT]> Servidor ouvindo conexão em localhost:6968...
█
```

Figura 1: Terminal rodando servidor ilustrando 1 cliente conectado

A figura 2 identifica o processo de execução do cliente, sendo necessário inicialmente identificar o nome do usuário, o ip do servidor e o número da porta em que os serviços irão trafegar(mesma porta do servidor), após isso o cliente aguarda conexão a resposta do servidor por um adversário.

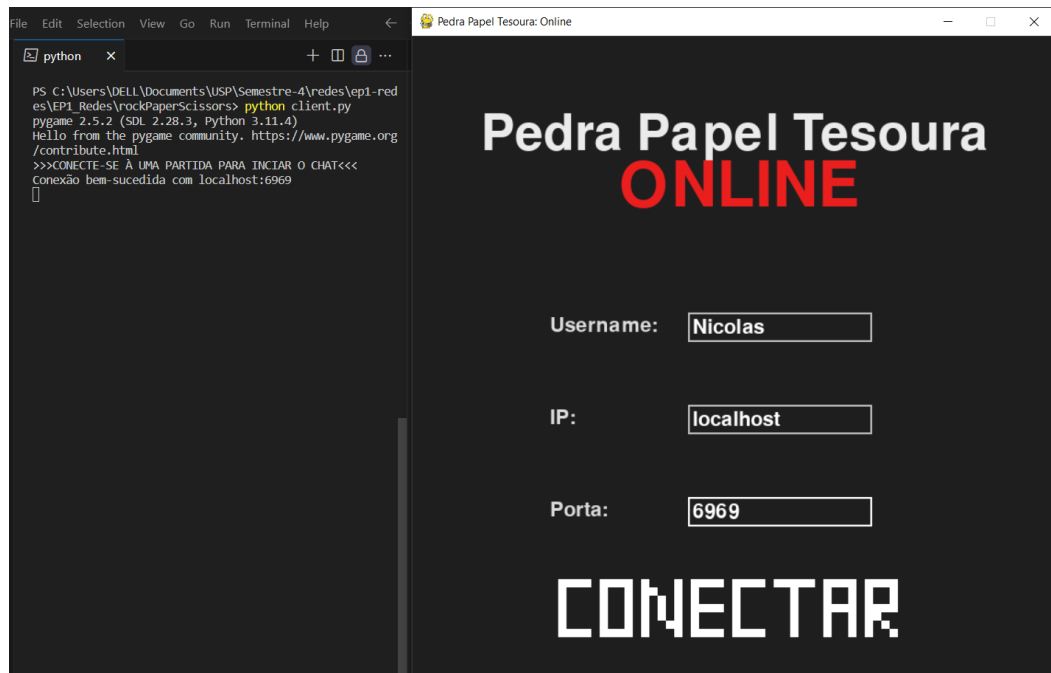


figura 2.1: Terminal rodando um cliente

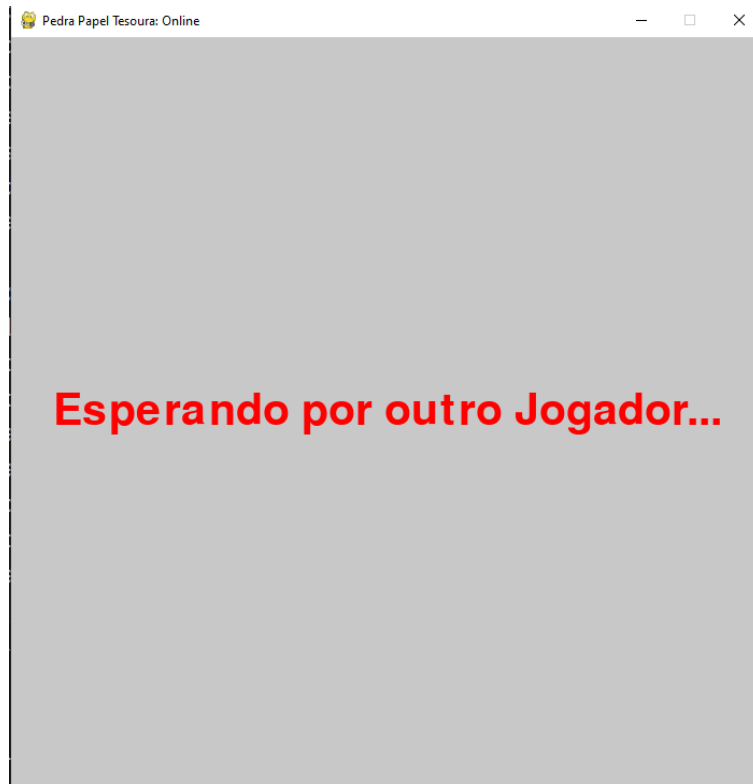


figura 2.2: Cliente esperando encontrar um adversário.

Na figura 3 é possível identificar que a partida foi encontrada, logo há dois clientes conectados ao servidor.

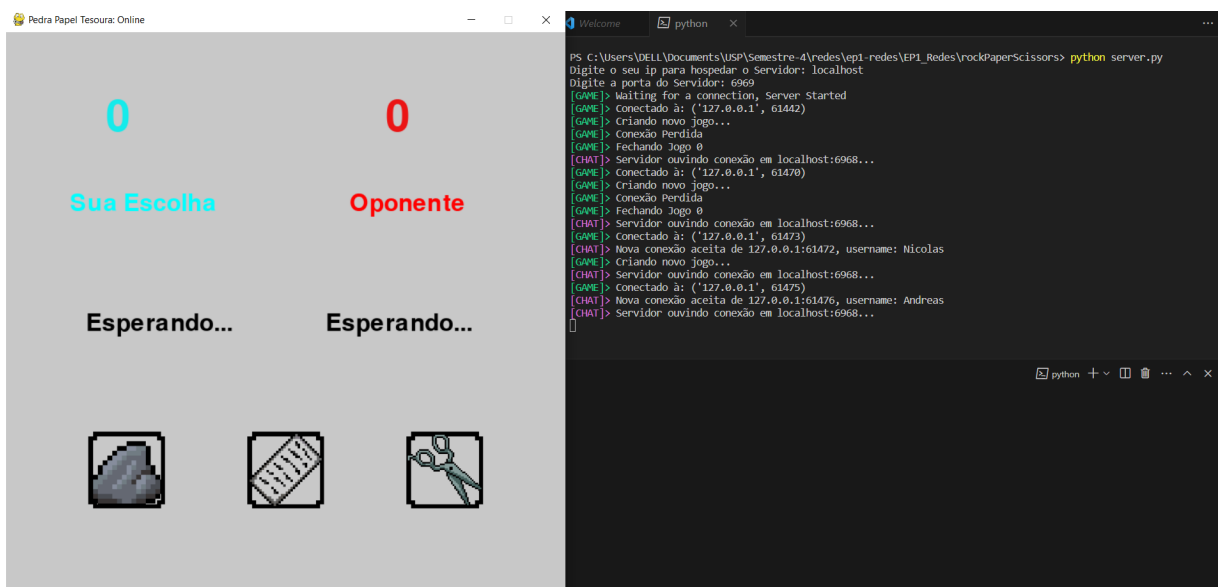


figura 3: partida encontrada

Na figura 4 é possível observar a partida propriamente dita, no qual o jogador a esquerda escolheu paper e o jogador a esquerda ainda não realizou a jogada



figura 4: jogando

Já na figura 5, temos a identificação do vencedor e o incremento no contador de vitórias ou derrotas de cada jogador.



figura 5: Identificação do vencedor

Os players que estão jogando também possuem acesso ao chat, em que a comunicação é realizada diretamente pelo terminal, como na figura 6 a seguir, vale ressaltar que o chat somente é habilitado quando o usuário se conecta ao servidor.

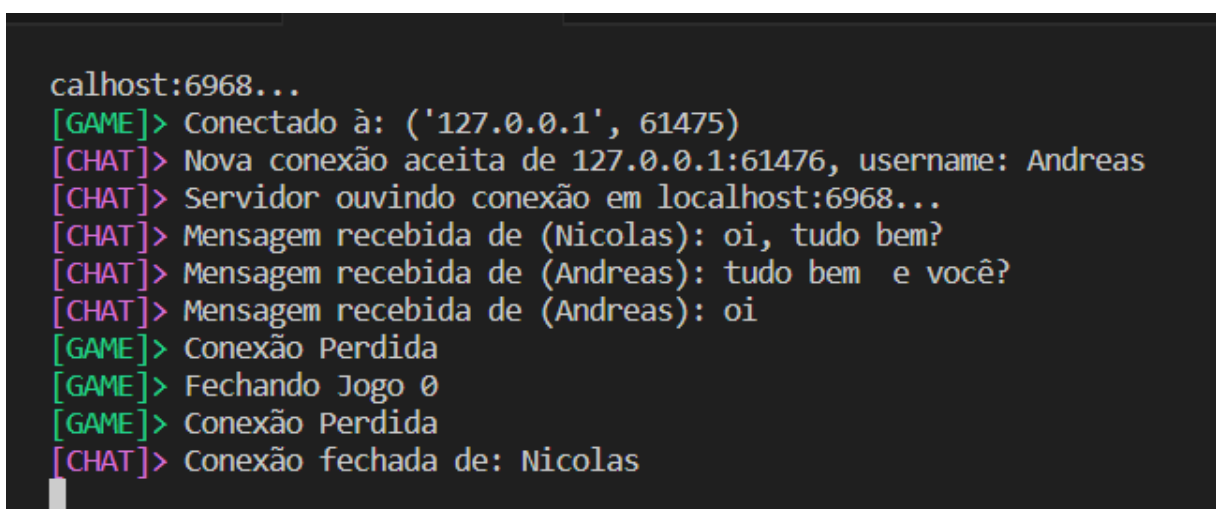


```
>>>CONECTE-SE À UMA PARTIDA PARA INICIAR O CHAT<<<
Conexão bem-sucedida com localhost:6969
Digite no terminal para enviar mensagens.
(Nicolas): oi, tudo bem?
(Andreas): tudo bem e você?
[]

PS C:\Users\DELL\Documents\USP\Semestre-4\redes\ep
1-redes\EP1_Redes\rockPaperScissors> python client
.py
pygame 2.5.2 (SDL 2.28.3, Python 3.11.4)
Hello from the pygame community. https://www.pygam
e.org/contribute.html
>>>CONECTE-SE À UMA PARTIDA PARA INICIAR O CHAT<<<
Conexão bem-sucedida com localhost:6969
Digite no terminal para enviar mensagens.
(Nicolas): oi, tudo bem?
(Andreas): tudo bem e você?
[]
```

Figura 6 - Comunicação via chat (Terminal)

Em caso de desconexão (fechar o terminal do cliente) o servidor identifica que a conexão foi perdida e, por consequência, fecha a conexão, como ilustra na figura 7.



```
calhost:6968...
[GAME]> Conectado à: ('127.0.0.1', 61475)
[CHAT]> Nova conexão aceita de 127.0.0.1:61476, username: Andreas
[CHAT]> Servidor ouvindo conexão em localhost:6968...
[CHAT]> Mensagem recebida de (Nicolas): oi, tudo bem?
[CHAT]> Mensagem recebida de (Andreas): tudo bem e você?
[CHAT]> Mensagem recebida de (Andreas): oi
[GAME]> Conexão Perdida
[GAME]> Fechando Jogo 0
[GAME]> Conexão Perdida
[CHAT]> Conexão fechada de: Nicolas
```

Figura 7 - Conexão fechada identificada pelo servidor

Bugs

Em meio a uma partida do clássico jogo Pedra, Papel e Tesoura, encerrar a janela do jogo não é o suficiente para informar ao servidor que a conexão com o jogador foi interrompida. Surpreendentemente, o servidor permanece alheio à desconexão até que o cliente finalize sua execução diretamente pelo terminal. A falta de detecção imediata da desconexão pela simples ação de fechar a janela pode ter implicações no desempenho e na experiência do usuário, destacando a importância de aprimoramentos na detecção e resposta a eventos de desconexão para garantir uma jogabilidade mais estável e confiável. Para que ocorra o reconhecimento da queda de conexão de um cliente pelo servidor é necessário que o cliente termine a execução pelo terminal.

Acredita-se que isso ocorre devido ao multithreading de execução do jogo, em que, embora a

conexão com o servidor do jogo tenha sido encerrada, a conexão com o servidor do chat se mantém alheia.

Fontes

Ficam aqui agradecimentos ao canal “Sentdex” do youtube, com tutoriais de um chat cliente-servidor em python, e ao canal “Tech with Tim” que auxiliou com seus guias de desenvolvimento de um jogo de pedra papel tesoura cliente-servidor com python e sua biblioteca pygame.

Segue abaixo links para eles:

- <https://www.youtube.com/watch?v=ytu2yV3Gn1I>
- <https://www.techwithtim.net/tutorials/python-online-game-tutorial/online-rock-paper-scissors-p-4>