Felipe Monfardini Giori França
Luiz Filipe M. Vieira
Redes de computadores
30 de abril de 2018

Trabalho prático 1 - Batalha Naval

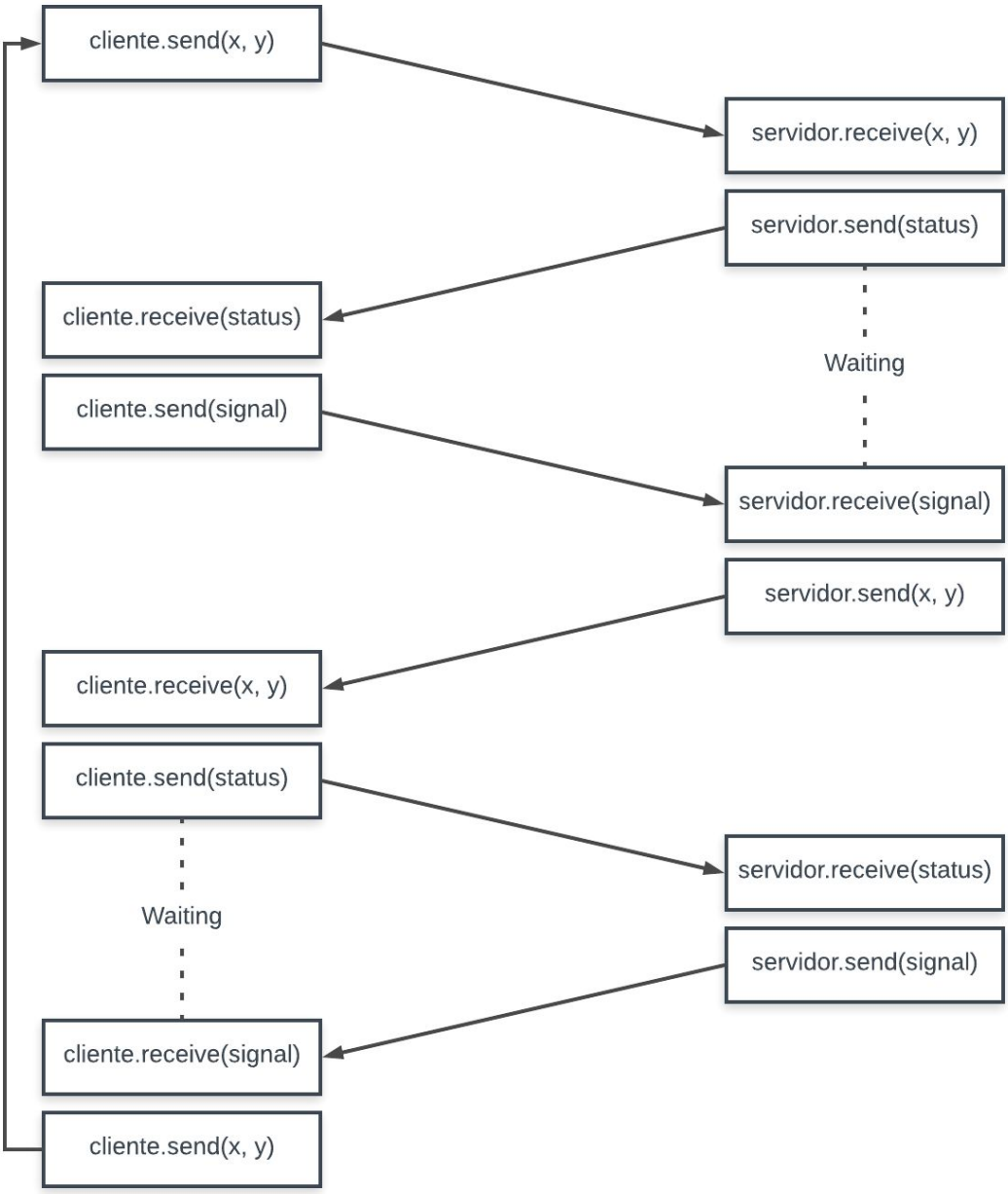
1. Introdução

O objetivo do trabalho foi implementar um jogo de batalha naval com dois jogadores: um servidor e um cliente. Os dois jogadores devem jogar comunicando-se através da rede. Na batalha naval, os dois jogadores têm um tabuleiro 10x10 onde podem distribuir sua frota de navios. Após distribuída a frota, cada jogador tem seu turno onde um deles escolhe um par x e y que são as coordenadas do seu ataque no tabuleiro adversário. O jogador que receber o ataque deve avisar se o ataque foi um sucesso ou um fracasso e logo depois fazer seu ataque. O jogo se repete dessa forma até que a frota de um dos jogadores tenha afundado.

Neste trabalho, toda a comunicação entre os dois jogadores deve ser feita pela rede, de maneira que um não tenha acesso ao tabuleiro do outro. A comunicação foi feita utilizando a biblioteca socket do python.

2. Modelagem do problema

O jogo começa com cada jogador distribuindo sua frota em uma matriz 10x10. O jogador cliente começa primeiro e seleciona um par x, y com $x = \{a, b, c, d, e, f, g, h, i, j\}$ e $y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ e envia esse par para o servidor. O servidor irá verificar se o disparo acertou um navio ou a água e irá devolver a resposta para o cliente. Então, o servidor também irá escolher um par x, y e enviá-lo para o cliente que também fará a verificação em seu tabuleiro e devolver a resposta avisando se o disparo acertou a água ou algum navio. O fluxo da comunicação também pode ser visto no diagrama abaixo.



Um sinal foi colocado no fluxo para impedir que um dos jogadores mande um ataque sem que o outro esteja escutando.

3. Detalhes de implementação

O trabalho foi implementado usando python 3. As seguintes bibliotecas não nativas foram usadas: pandas e numpy

Pandas: é uma biblioteca de estrutura de dados que foi utilizada para representar a matriz do tabuleiro.

Numpy: é uma biblioteca para cálculos matemáticos que foi usada para gerar uma lista das posições da matriz que ainda não foi feito um ataque.

A implementação tem 4 arquivos .py: Client.py, Server.py, cliente.py e servidor.py.

Client.py: Classe do cliente com funções e mecânicas relacionadas a parte do cliente como, verificar se as coordenadas são válidas, processar ataques e mostrar tabuleiro.

Server.py: Classe do servidor com funções e mecânicas relacionadas a parte do servidor como, gerar um ataque aleatório, processar ataques e verificações de comportamento.

cliente.py: Cuida da conexão com o servidor e do fluxo de execução principal do cliente. Há três funções principais:

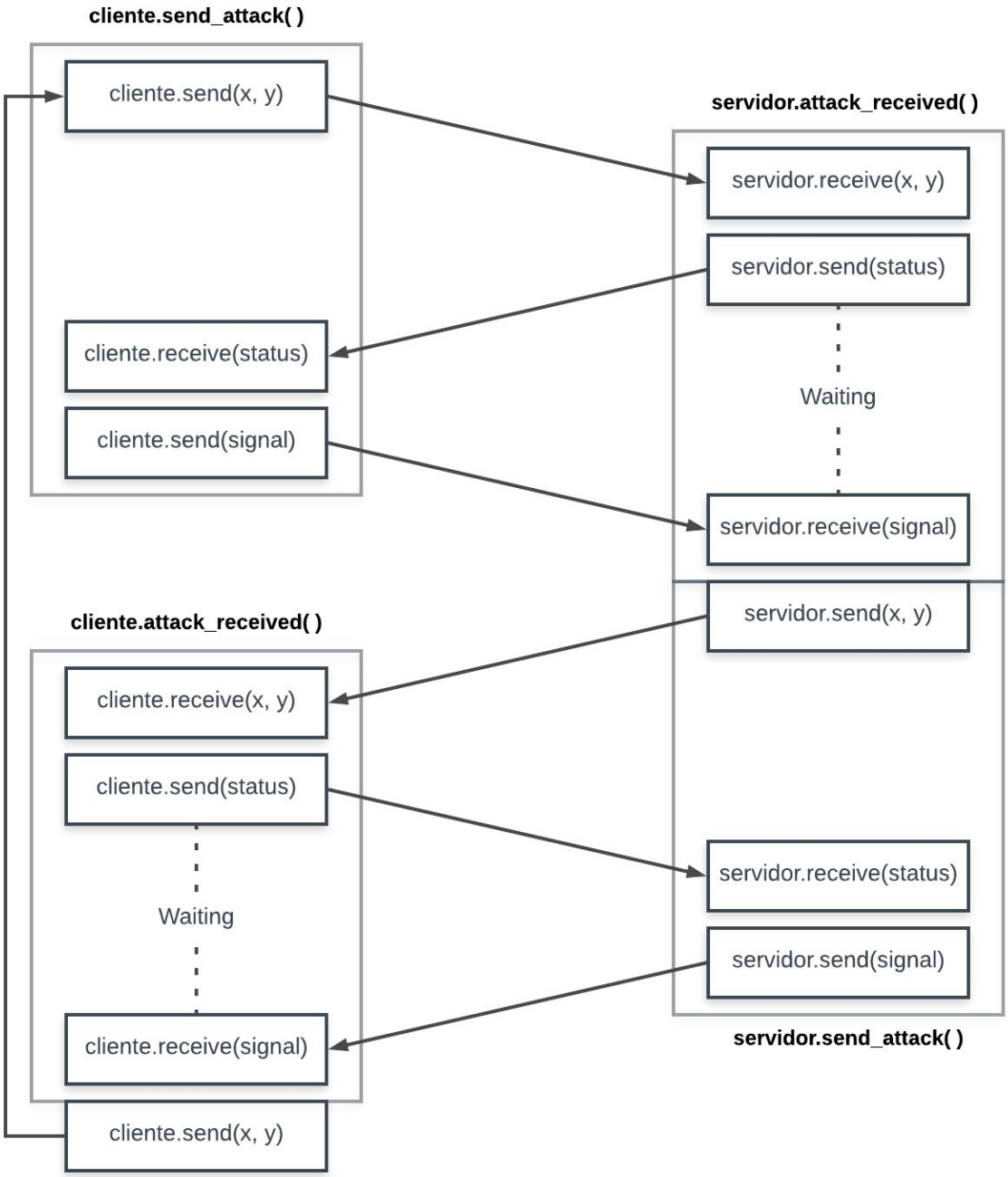
- `connection_setup()`: Verifica se o endereço é IPv4 ou IPv6 e então tenta iniciar uma conexão com o servidor. A verificação é feita através da função `socket.inet_aton(address)`, que lança uma exceção caso o endereço não seja IPv4.
- `attack_received()`: Esta função cuida de receber as coordenadas de ataque do servidor, chamar as funções que fazem o cálculo do ataque, enviar o resultado do ataque para o servidor e esperar o sinal do servidor para prosseguir.
- `send_attack()`: Responsável por receber uma coordenada do terminal, verificar se é uma entrada válida, enviar a coordenada de ataque para o servidor, esperar o resultado do ataque e enviar um sinal para que o servidor mande seu ataque. Junto com o resultado do ataque também vai viu uma verificação para saber se o servidor ainda está no jogo.

servidor.py: Cuida da conexão e do fluxo de conexão do servidor. Há três funções principais:

- `connection_setup()`: Responsável pela conexão. Os parâmetros aceitam IPv6 por default, pois um endereço IPv4 recebido pode ser mapeado para um IPv6.
- `attack_received()`: Esta função cuida de receber as coordenadas de ataque do cliente, chamar as funções que fazem o cálculo do ataque, enviar o resultado do ataque para o cliente e esperar o sinal do cliente para prosseguir.

- `send_attack()`: Enviar a coordenada de ataque para o cliente, esperar o resultado do ataque e enviar um sinal para que o cliente mande seu ataque. Junto com o resultado do ataque também vai viu uma verificação para saber se o cliente ainda está no jogo.

O diagrama do fluxo de comunicação abaixo ajuda a visualizar as tarefas que cada função é responsável.



Quando um dos jogadores perde, ele envia uma uma mensagem concatenada na confirmação do acerto do tiro adversário. O adversário recebe a mensagem e troca o valor de uma flag que termina o fluxo de execução do programa.

4. Execução

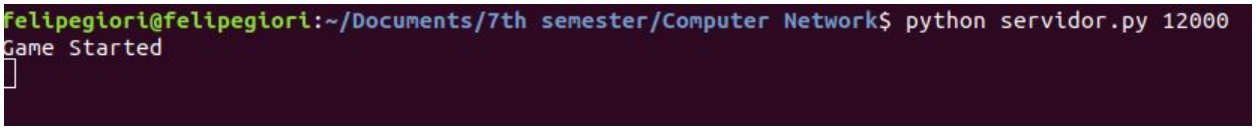
O servidor é executado da seguinte forma:

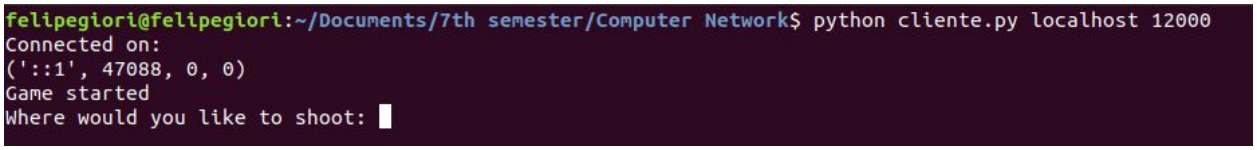
```
python3 servidor.py <port>
```

O cliente é executado da seguinte forma:

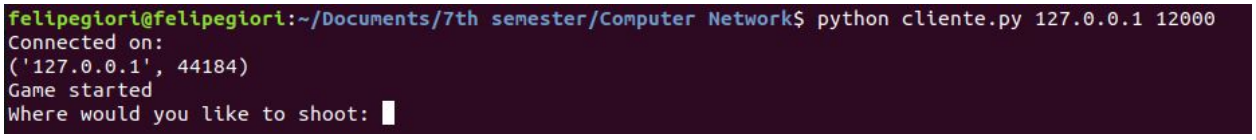
```
python3 cliente.py <address> <port>
```

IPv6





IPv4



As coordenadas são dadas da seguinte forma:

```
Where would you like do shoot: <y> <x>
```

<y> e <x> devem estar separados por um espaço simples.

Para o cliente acessar sua tabela ele deve entrar apenas com a letra p

```
Where would you like do shoot: p
```

No comando p são mostradas duas tabelas: Uma com os tiros disparados onde 0 é uma posição que ainda não disparamos, M para miss e H para hit. A outra tabela é correspondente a distribuição de sua frota onde 0 indica água, 1 é uma parte de um navio e H é a parte de um navio que foi atingida por um disparo do servidor.

```
felipegiori@felipegiori:~/Do  felipegiori@felipegiori:~/Documents/7th semester/Computer Network$ python cliente.py localhost 12000
Game Started                  Connected on:
Attack received at a, 1       ('::1', 47108, 0, 0)
They missed                   Game started
Fire at f, 4                  Where would you like to shoot: a 1
Target missed                 Fire at a, 1
Attack received at j, 6       Target missed
They missed                   Attack received at f, 4
Fire at f, 9                  They missed
Target hit                    Where would you like to shoot: j 6
Attack received at i, 8       Fire at j, 6
They missed                   Target missed
Fire at g, 9                  Attack received at f, 9
Target hit                    We've been hit
Attack received at f, 0       Where would you like to shoot: i 8
They missed                   Fire at i, 8
Fire at h, 9                  Target missed
Target hit                    Attack received at g, 9
                              We've been hit
                              Where would you like to shoot: z 10
                              Invalid input. Coordinates must be between a ... j and 0 ... 9
                              Where would you like to shoot: teste
                              This is not a valid input
                              Where would you like to shoot: 598 r 46
                              Too many values. Please type only the x and y coordinates or p for the table
                              Where would you like to shoot: f 0
                              Fire at f, 0
                              Target missed
                              Attack received at h, 9
                              We've been hit
                              Where would you like to shoot:
```

```
Where would you like to shoot: p
Shots table:
H: hit
M: miss
   0  1  2  3  4  5  6  7  8  9
0  0  M  0  0  0  0  0  0  0  0
1  0  0  0  0  0  0  0  0  0  0
2  0  0  0  0  0  0  0  0  0  0
3  0  0  0  0  0  0  0  0  0  0
4  0  0  0  0  0  0  0  0  0  0
5  M  0  0  0  0  0  0  0  0  0
6  0  0  0  0  0  0  0  0  0  0
7  0  0  0  0  0  0  0  0  0  0
8  0  0  0  0  0  0  0  0  M  0
9  0  0  0  0  0  0  M  0  0  0
Your table:
   0  1  2  3  4  5  6  7  8  9
0  0  0  0  1  1  0  0  0  1  0
1  0  0  1  1  1  0  0  0  1  0
2  1  0  0  0  0  1  1  0  1  0
3  1  0  0  0  0  0  0  0  1  0
4  1  0  0  1  1  0  0  0  1  0
5  1  0  1  0  0  0  0  0  0  H
6  0  0  1  0  0  1  1  0  0  H
7  0  0  1  0  0  0  0  0  0  H
8  0  0  0  0  0  0  0  0  0  1
9  1  1  1  0  0  0  0  0  0  0
Where would you like to shoot:
```