**Documentação programa cliente-servidor utilizando rest api**

Vinícius Ortiz Fontes matricula num: 2018091861
Arthur Nunes Cascardo matricula num: 2018013968

**Percepções gerais:**

Para a implementação do que foi pedido no descritivo do trabalho, optamos por usar Python, essa linguagem possui um nível um pouco maior que C, que facilita o desenvolvimento de alguns pontos. Além de ser uma linguagem que possuímos mais familiaridade.

No começo do trabalho, para ser possível trabalhar da melhor forma em dupla, criamos um repositório no github, onde cada um de nós criou branches e desenvolveu parte do código. Com esse repositório pronto, já sendo iniciado com o conteúdo do ultimos trabalho, começamos o desenvolvimento efetivo do programa relativo a rest api. O github foi usado também para versionamento do código, assim tivemos mais tranquilidade para trabalhar sem o receio de perder alguma etapa funcional.

Para conseguir organizar o trabalho melhor e ter direcionamentos para cada um, dividimos o trabalho em pequenas entregas: Criação arquivo servidor, comunicação cliente servidor usando sockets, criação da api no servidor, consumir dados da api usando socket pelo lado do cliente e finalmente implementação das análises solicitadas. Essas etapas serão descritas a seguir:

- **Criação arquivo servidor**

Nesse ponto, foi criado um arquivo server.py com o objetivo de estruturar nele uma configuração de sockets que fosse responsável pelo servidor. Dessa forma, esse arquivo seria encarregado por receber mensagens e se comunicar com o cliente, para essa primeira implementação simples, foi usado o seguinte código:

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(("localhost", 80))
server_socket.listen(5)
while True:
    (client_socket, address) = server_socket.accept()
    data = client_socket.recv(1024)
    print(data.decode("ascii"))
    message = "Server: " + input()
    client_socket.send(bytearray(message, "ascii"))
```

- **Comunicação cliente servidor usando sockets**

Para comunicar com o servidor criado no primeiro tópico, é necessário que o arquivo client.py faça a conexão com o socket implementado pelo servidor. Essa conexão precisa de alguns parâmetros como o host e a porta de conexão, com esses parâmetros passados conseguimos conectar e ler o input do teclado para enviar mensagens para o servidor. Parte do código implementado foi:

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(tuple(ip_port))
message = "Client: " + input()
client_socket.send(bytearray(message, "ascii"))
data = client_socket.recv(1024)
print(data.decode("ascii"))
```

- **Criação da api**

Para a criação da api REST, nós optamos por utilizar a lib Flask disponível no python, conforme o sugerido pelo professor. Nossa escolha se baseou principalmente no fato dessa biblioteca ser bastante intuitiva e abstrair de alguns detalhes a implementação de uma API, dessa forma o processo de criação é mais eficiente.

Após estudar a documentação dessa lib, fizemos a implementação de cada uma das rotas necessárias para a futura implementação das análises, as rotas criadas foram:

***@app.route("/api/game/<id>")***

```
@app.route("/api/game/<id>")
def game(id):
    stats = "No games found in DataSet"
    for games in games_played:
        if games["id"] == int(id):
            stats = games
            break
        else:
            stats = "No game found for id: " + str(id)
    return {"game_id": id, "game_stats": stats}
```

Essa rota, consulta informações de um jogo específico quando é fornecido para ela o id do jogo desejado, aqui está um exemplo de chamada e retorno dessa api:

http://localhost:5000/api/game/3


```
{"game_id":"3","game_stats":{"auth":"2016026191:111:7609d636ff02b901aad0bcae4b65c3f9b505d816a6215eb772f6
9f2d5fddf159+2015435632:111:577a175cd7c375fa798d42f01151b33911c9ddba716c7fb0923f8a0f990f81db+33779cff20a
ca4725a07e82ba8cc1d2fcce74b1295e3cb89580b19e2d2013325","cannons":[[8,1],[3,0],[6,2],[2,3],[2,2],[4,2],[5
,2],[4,0]],"description":"Game
completed!","id":3,"score":{"escaped_ships":381,"getcannons_received":1,"getturn_received":1089,"last_tu
rn":272,"remaining_life_on_escaped_ships":668,"servers_authenticated":[1,2,3,4],"ship_moves":4839,"shot_
received":1403,"sunk_ships":647,"tstamp_auth_completion":1640281937.7747936,"tstamp_auth_start":16402819
34.2148864,"tstamp_completion":1640283731.0174139,"valid_shots":1403},"status":0,"type":"gameover"}}
```


***@app.route("/api/rank/sunk")***

```
def sunk():
    args = request.args.to_dict()
    start = int(args["start"])
    end = int(args["end"])
    podium_sunk = {"ranking": "sunk"}
    if start > end:
        return "End id must be greater than start id"
    else:
        podium_sunk["start"] = start
        podium_sunk["end"] = end
        scoreboard = getScore(games_played, 'sunk_ships')
        scoreboard_sorted = sorted(scoreboard, key=lambda i: i['sunk_ships'], reverse=True)
        podium_sunk['game_ids'] = getIdByScore(scoreboard_sorted, podium_sunk["start"], podium_sunk["end"])
        return podium_sunk
```

Essa rota é responsável por retornar os ids dos jogos mais bem posicionados no ranking de sunks, devem ser passados para ela o range que deseja consultar por meio de querystring, aqui está um exemplo de chamada e retorno dessa rota:

http://localhost:5000/api/rank/sunk?start=1&end=10


```
{"end":10,"game_ids":[2005,2001,1334,1949,72,1423,1766,1491,1481,1285],"ranking":"sunk","start":1}
```

***@app.route("/api/rank/escaped")***

```
def escaped():
    args = request.args.to_dict()
    start = int(args["start"])
    end = int(args["end"])
    podium_escaped = {"ranking": "escaped"}
    if start > end:
        return "End id must be greater than start id"
    else:
        podium_escaped["start"] = start
        podium_escaped["end"] = end
        scoreboard = getScore(games_played, 'escaped_ships')
        scoreboard_sorted = sorted(scoreboard, key=lambda i: i['escaped_ships'], reverse=False)
        podium_escaped['game_ids'] = getIdByScore(scoreboard_sorted, podium_escaped["start"], podium_escaped["end"])
        print(podium_escaped)
        return podium_escaped
```

Essa rota é responsável por retornar os ids dos jogos com menor número de navios escapados, devem ser passados para ela o range que deseja consultar por meio de querystring, aqui está um exemplo de chamada e retorno dessa rota:

http://localhost:5000/api/rank/escaped?start=1&end=10

{"end":10,"game_ids":[2005,2001,1334,1949,72,1423,1766,1491,1481,1285],"ranking":"sunk","start":1}

Além de criar essas rotas conforme foi exigido na descrição do trabalho, foram criadas também algumas funções auxiliares para manipulação do dataset enviado pelo professor, foram elas:

```
def getScore(dataset, key):
    scores = []
    for games in dataset:
        if key in games['score']:
            scores.insert(0, games['score'])
    return scores
```

Nessa primeira o dataset é varrido e todos os scores são armazenados em uma variável chamada scores.

```
def getIdByScore(dataset, start, end):
    ids = []
    length = (end - start) + 1
    if length > len(dataset):
        return "Too many games!"
    for scores in dataset:
        for games in games_played:
            if scores == games['score']:
                ids.insert(0, games['id'])
        if len(ids) == length:
            return ids
```

A segunda, varre todo o dataset fornecido e linka o id encontrado com o seu respectivo jogo, de acordo com o score.

É importante ressaltar que todos os valores retornados pelas API's também como os valores retornados pelas funções, tem como base o dataset fornecido pelo professor.

Para o bom funcionamento do flask, foram incluídas as seguintes linhas de código no início

```
import json
from flask import request, Flask

app = Flask(__name__)
```

No final do arquivo foram colocadas as linhas responsáveis pelo arquivo e por rodar o flask

```
fp = open("dataset.json", 'r')
games_played = json.load(fp)
app.run()
#eof
```

- **Consumir dados da api usando socket pelo lado do cliente**

Nessa etapa do trabalho, foi necessário adaptar a lógica do lado do cliente, de maneira que fossem passados para a função connect do socket os parâmetros necessários para a conexão com o servidor implementado pelo Flask. Por padrão o flask cria o servidor na porta 5000 do localhost.

Para fazer a requisição para o flask, precisávamos que alguns parâmetros fossem passados por ela, como qual o método da requisição, qual o host e qual efetivamente era o caminho a ser consultado pela requisição. Após algumas pesquisas e testes, chegamos a seguinte formatação de requisição para o endpoint da api:

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((ip_port[0], ip_port[1]))
request = f"GET {req} HTTP/1.1\r\nHost: {ip_port[0]}:{ip_port[1]}\r\n\r\n".encode()
response = ""
client.sendall(request)
```

Após acertar essa formatação da requisição, tivemos um segundo ponto que foi a recepção desses dados, para conseguir receber toda a resposta da api, tivemos que implementar um laço while junto com a função recv() do Socket:

```
while True:
    recv = client.recv(1024)
    if recv == b'':
        break
    response += recv.decode()
```

Assim fazemos a recepção de todos os dados enviados pela api, porém tínhamos um problema que era a resposta do endpoint vinha concatenada com o cabeçalho da requisição, e para solucionar esse problema usamos a função find com o objetivo de localizar somente o objeto json devolvido pela api `json_obj = json.loads(response[response.find("{"):])`

- **Implementação das análises solicitadas**

Após ter conseguido consumir a resposta da api implementada, era necessário criar dentro do client.py uma lógica responsável por formatar e analisar os dados recebidos conforme solicitado. O client.py começa interpretando os parâmetros passados para o programa e divide isso em um array, de acordo com o que cada um significa, e assim ele direciona em seguida, qual das análises será executada. Isso tudo é feito por uma função auxiliar criada:

```
def parse_input(arg):
    games: dict     # Remover warning da IDE
    ip_port = arg.pop(0).split(":")
    ip_port[1] = int(ip_port[1])
    analysis = arg.pop(0)
    if analysis == '1':
        games = httpRequest(ip_port, analysis, None)
        games_info = getTop100GamesInfo(ip_port, games["game_ids"])
        return analise_1(games_info)
    elif analysis == '2':
        games = httpRequest(ip_port, analysis, None)
        games_info = getTop100GamesInfo(ip_port, games["game_ids"])
        return analise_2(games_info)
    else:
        eprint("Wrong argument for 'analysis' parameter")
```

Além dessa função auxiliar, foram criadas algumas outras que são usadas nas análises, são elas:

**eprint()**

```
def eprint(args):
    print(args, file=sys.stderr)
```

Essa função é responsável por definir a saída usando stderr

**normalizeCannons()**

```
def normalizeCannons(cannons):
    normalized_string = ""
    normalized = [0] * 8
    for cannon in cannons:
        normalized = insertPlusOne(normalized, cannon[0])
    for num in normalized:
        normalized_string += str(num)
    return normalized_string
```

Essa função recebe o parâmetro cannons e é responsável por normalizar a posição deles.

**insertPlusOne()**

```
def insertPlusOne(lis, index):
    lis[index - 1] += 1
    return lis
```

Essa função recebe como parâmetro, uma lista e um index, e tem como objetivo adicionar 1 a posição passada por “index”.

**getTop100GamesInfo()**

```
def getTop100GamesInfo(ip_port, games_ids):
    analysis = '0'
    games_info = []
    for ids in games_ids:
        games_info.insert(0, httpRequest(ip_port, analysis, ids))
    games_info.reverse()
    return games_info
```

Essa função recebe como parâmetro, ids dos jogos e informações de conexão do socket, ela é responsável por fazer a requisição para a api e trazer informações sobre os top 100 jogos.

**httpRequest()**

```
def httpRequest(ip_port, analysis, game_id):
    if analysis == '0':
        req = "/api/game/" + str(game_id)
    elif analysis == '1':
        req = "/api/rank/sunk?start=1&end=100"
    elif analysis == '2':
        req = "/api/rank/escaped?start=1&end=100"
    else:
        req = ""
        eprint("Wrong argument for analysis")
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((ip_port[0], ip_port[1]))
    request = f"GET {req} HTTP/1.1\r\nHost: {ip_port[0]}:{ip_port[1]}\r\n\r\n".encode()
    response = ""
    client.sendall(request)
    while True:
        recv = client.recv(1024)
        if recv == b'':
            break
        response += recv.decode()
    json_obj = json.loads(response[response.find("{"):])
    return json_obj
```

Essa função recebe como parâmetros, as informações de conexão com o socket, a análise desejada e o id do jogo desejado. Com isso, ela é responsável por direcionar quais paths de api serão enviados na requisição, além de receber e tratar a resposta da api.

Usando essas funções auxiliares criadas, desenvolvemos as análises pedidas no descritivo do trabalho, as funções ficaram da seguinte forma:

**analise1()**

```
def analise_1(games_info):
    f = open("output.csv", "w")
    auth_ids = []
    idsAndSunkShips = []
    games_out = {}
    games_out_list = []
    for games in games_info:  # pega todos os sags
        auth_ids.append(games['game_stats']['auth'])  # array com os sag de todos os jogadores do top 100
        idsAndSunkShips.append(games['game_stats']['auth'])
        idsAndSunkShips.append(games['game_stats']['score']['sunk_ships'])
    for x in auth_ids:  # ve numero de repeticoes dos sags
        aux2 = 0
        aux3 = 0
        numOfGames = 0
        sumOfSunk = 0
        while aux2 < 100:
            if x == auth_ids[aux2]:
                numOfGames = numOfGames + 1
            aux2 = aux2 + 1
        while aux3 < 200:
            aux3 = aux3 + 1
            if x == idsAndSunkShips[aux3-1]:
                sumOfSunk = sumOfSunk + idsAndSunkShips[aux3]
            aux3 = aux3 + 1
        avrgSunk = sumOfSunk/numOfGames
        games_out['SAG'] = x
        games_out['num_of_games'] = numOfGames
        games_out['avrg_sunk'] = avrgSunk
        games_out_list.insert(0, games_out)
        games_out = {}
    games_out_list = sorted(games_out_list, key=lambda y: y['avrg_sunk'], reverse=True)
    for game in games_out_list:
        print(game['SAG'], ",", game['num_of_games'], ",", round(game['avrg_sunk'], 2), file=f)
    f.close()
    return games_out_list
```

Essa função recebe como parâmetro uma variável games_info e a partir dela faz requisições a api buscando os valores necessários para construir a análise, cada um dos valores finais necessários é armazenado em um dict. Uma vez que a coleta de dados é concluída esse dict é registrado em um arquivo .csv nomeado de output.csv. É importante ressaltar que cada vez que o programa é rodado o arquivo output.csv será sobrescrito, caso exista.

Analisando o resultado obtido após a análise 1, podemos identificar que nenhum dos alunos conseguiu atingir o proposto de 800 navios afundados. Além disso, concluímos também que mais de 40% dos top100 jogos pertencem a uma dupla de alunos.

**analise2()**

```
def analise_2(game_info):
    f = open("output.csv", "w")
    cannon_pos = {}
    normalized_cannons = []
    aux_list = []
    median_escaped = {}
    for game in game_info:  # Percorre as informações dos top 100 jogos e normaliza o posicionamento dos canhões
        normalized_cannons.insert(0, normalizeCannons(
            game["game_stats"]["cannons"]))
        try:
            cannon_pos[normalized_cannons[0]].append(game["game_stats"]["score"]["escaped_ships"])
        except KeyError:
            aux_list.insert(0, game["game_stats"]["score"]["escaped_ships"])
            cannon_pos[normalized_cannons[0]] = aux_list  # Lista com todos os valores de navio escapados associados a um posicionamento normalizado
        aux_list = []
    normalized_cannons = list(dict.fromkeys(normalized_cannons))  # Dicionario ordenado cuja chave é a normalização dos canhões e o valor associado à chave, o numero de navios escapados
    for normals in normalized_cannons:
        median_escaped[normals] = float(statistics.median(cannon_pos[normals]))  # Tira a média dos navios escapados
    for key in median_escaped:
        print(key, ",", median_escaped[key], file=f)
    f.close()
    return median_escaped
```

Essa função, assim como a análise 1, recebe uma variável games_info que contém parâmetros necessários para a chamada de cada caminho de api, essas chamadas são feitas e tem seus resultados tratados utilizando das funções auxiliares previamente descritas. Os dados coletados são armazenados em um dict que é impresso em um arquivo output.csv.

Analisando o resultado obtido após a análise 2, podemos identificar que os jogos com 2 ou mais canhões em suas carreiras, ou canhões em 3 ou mais carreiras consecutivas foram mais bem sucedidos

**Execução do programa**

Para a compilação e bom funcionamento do programa, é necessário ter instalado o python. Uma vez com o python instalado, deve ser rodado o prompt de comando no diretório onde se encontram os arquivos, em seguida o arquivo server.py deve ser executado da seguinte maneira:

*python server.py*

Ainda com o servidor sendo executado, agora devemos abrir uma nova janela do prompt de comando no mesmo diretório e executar o cliente passando a análise desejada, assim como o host e a porta de conexão, como mostrado no exemplo:

*python client.py ./127.0.0.1:5000 1*

É importante ressaltar que o flask cria o endpoint no host local de seu computador e na porta 5000, sendo assim o valor passado como host e porta deve ser 127.0.0.1:5000 ou localhost:5000. O parâmetro em vermelho no exemplo é o número da análise, e seus valores possíveis são 1 ou 2.

Após a execução do client.py o arquivo output.txt será criado no mesmo diretório em que foi executado o programa, caso já exista, ele terá seu conteúdo sobrescrito.

**Conclusão:**

Consideramos que o programa implementado foi um sucesso, visto que conseguimos contemplar todos os pontos descritos na tarefa. Foi um trabalho de muito aprendizado visto que, apesar de ter um pouco de costume com o python, nenhum de nós havia montado uma api. Foi interessante também ter que repensar a lógica de requisições por código visto que não podíamos usar bibliotecas como a request.