Integrante 1: **Hélio Rodrigues Feitosa Neto - 2019056890**
Integrante 2: **Vando Carlos Diniz Reis - 2019057195**
Integrante 3: **Igor Cleto Silva de Araujo - 2018014727**
Integrante 4: **Mateus Daniel Simonetti - 2019057004**
Data de entrega: **31/01/2022**

Relatório Bridge Buff

1. Introdução

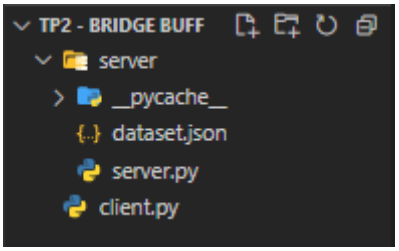
Bridge Buff é um sistema de consulta para jogos executado no servidor do TP-1. Ele é composto por um par cliente-servidor no qual o servidor atende às requisições do cliente e o cliente faz requisições HTTP para o servidor através de uma API REST.

2. Objetivo

Este relatório tem como objetivo explicar como foi feito o desenvolvimento do programa Bridge Buff, mostrando os problemas que apareceram ao longo da execução, assim como algumas especificações do sistema.

3. Desenvolvimento

Antes de iniciar o desenvolvimento do programa, foi criado o repositório **privado** no github de nome TP2 - Bridge Buff, assim como a sua estrutura de diretórios que pode ser visualizada na imagem abaixo:



A pasta principal nomeada TP2 - Bridge Buff é o local que contém todo o código do projeto, sendo ele dividido no arquivo client.py, que é o processo responsável por fazer todas as requisições para o servidor e analisar os dados recebidos, e a pasta server que contém o arquivo dataset.json, onde fica armazenada a base de dados dos jogos executados no TP1 - Bridge Defense, e o arquivo server.py que contém todo o código do servidor.

3.1. sever.py

Iniciamos o desenvolvimento do projeto pelo server.py, que é responsável por carregar as informações dos jogos e responder requisições de clientes através dos endpoints. O server.py cria 3 endpoints, o primeiro que é /api/game/{id} é responsável por recuperar informações do jogo a partir do id passado, o segundo cuja rota é /api/rank/sunk?start={index}&end={index} é responsável por recuperar os identificadores dos jogos com maior número de navios afundados e o último cuja rota é /api/rank/escaped?start={index}&end={index} é responsável por recuperar os identificadores dos jogos com menor número de navios escapados.

Com o Flask, foi possível criar uma interface visual para o servidor

3.1.2 Flask

Antes de começar a programar o servidor de fato, pesquisamos por Web frameworks que facilitam a implementação de endpoints e escolhemos o Flask, que foi recomendado no documento da prática, por acharmos o mais simples de ser usado, servir para os propósitos da prática e ser compatível com as tecnologias utilizadas.

Com o Flask é possível criar um aplicativo e interagir com suas dependências através de rotas.Instanciamos um aplicativo Flask, conforme o código abaixo:

É possível ver mais informações sobre apps em Flask em:

https://flask.palletsprojects.com/en/2.0.x/quickstart/

Logo após esta etapa temos que realizar a leitura da base de dados fornecida pelo professor na qual contém os logs desejados para enviar ao lado do cliente.Utilizamos a biblioteca json do código Python para realizar o processo desejado.

3.1.3 /api/game/{id}

Após instalarmos e importarmos o flask para o projeto, iniciamos a primeira rota proposta, a /api/game/{id}. Com a base de dados já importada, criamos a rota

com app.route cujos parâmetros são a rota e o tipo de requisição que neste caso é GET. Em seguida criamos a função get_id que recebe um id e retorna um json com o id do jogo e o game_stats.

Para isso, o método avalia se a base de dados possui algum id compatível com o digitado pelo usuário. Se sim, ela busca as informações desse jogo para construir o corpo da resposta. Caso não encontre, uma exceção é disparada, informando que o jogo não foi encontrado

```
@app.route('/api/game/<id>', methods=['GET'])
def get_id(id):
    try:
        for game in dataset:
            if 'id' in game.keys() and game['id'] == int(id):
                jsonReturned = {
                'game_id': game['id'],
                'game_stats': game
                }
                return json.dumps(jsonReturned)
    except:
        return json.dumps({'msg':'Game not found'})
```

3.1.4 /api/rank/sunk?start={index}&end={index}

Feita a primeira rota, repetimos o procedimento para a segunda usando o app.route com o método GET e fizemos a função para recuperar os identificadores dos jogos com maior número de navios afundados. A função pega os parâmetros da requisição correspondente ao índice inicial e ao índice final e usa o sort para ordenar a base de dados passando o número de návios afundados como chave. A maior dificuldade do server foi ordenar os navios na ordem proposta visto que tivemos dificuldade para entender a lógica de exception do .sort e passar a key (função customizados). Após mexer bastante no código e refazer a função algumas vezes conseguimos recuperar o json na ordem correta de forma que essa rota GET agora estava funcional.

```python
@app.route('/api/rank/sunk', methods=['GET'])
def get_sunk_ships():
    start_index = int(request.args.get('start'))
    end_index = int(request.args.get('end'))
    dataset.sort(key=sunk_ships_number)
    ranking = dataset[start_index-1:end_index]
    idRank = map(get_id_by_game, ranking)
    jsonReturn = { "ranking": "sunk",
    "start": start_index,
    "end": end_index,
    "game_ids": list(idRank)
    }

    return json.dumps(jsonReturn)

def sunk_ships_number(k):
    try:
        return k['score']['sunk_ships'] * -1
    except:
        return -1
```

3.1.5 /api/rank/escaped?start={index}&end={index}

Iniciamos agora a última rota /api/rank/escaped?start={index}&end={index} que teve uma implementação bastante similar à rota anterior. Assim como a última, pegamos os parâmetros da requisição correspondentes ao índice inicial e ao índice final e usamos o sort para ordenar a base de dados passando, desta vez, o número de návios que escaparam como chave e retornamos um json com os jogos com menor número de navios escapados. Como resolvemos o problema do sort na rota anterior, não tivemos muita dificuldade na implementação deste endpoint.

```python
@app.route('/api/rank/escaped', methods=['GET'])
def get_escaped_ships():
    start_index = int(request.args.get('start'))
    end_index = int(request.args.get('end'))
    dataset.sort(key=escaped_ships_number)
    ranking = dataset[start_index-1:end_index]
    idRank = map(get_id_by_game, ranking)
    jsonReturn = { "ranking": "escaped",
    "start": start_index,
    "end": end_index,
    "game_ids": list(idRank)
    }

    return json.dumps(jsonReturn)

def escaped_ships_number(k):
    try:
        return k['score']['escaped_ships']
    except:
        return -1

app.run(debug=True)
```

3.2. client.py

Após criar o server.py, iniciamos a implementação do client.py, que é responsável por fazer todas as requisições para o servidor e fazer duas análises a partir dos dados recebidos. A análise 1 retorna os SAGs com melhor desempenho, ou seja, os SAGs que tiveram mais ocorrências no top 100 do ranking de navios afundados em ordem decrescente e a análise 2 retorna os melhores posicionamentos de canhões, ou seja, os SAGs que tiveram menor número de navios escapados e calcular a média desse número para cada posicionamento normalizado de canhões.

3.2.1 Analisa_Jogo

Iniciamos o client.py fazendo a conexão com o servidor, passando o IP do servidor e a porta indicada. Ao passarmos as informações com o client.connect conseguimos estabelecer uma conexão que retornava status 200, indicando que a conexão foi bem sucedida. Em seguida criamos a função Analisa_Jogo (função responsável por pegar um jogo específico) que recebe um type e um id. Essa função envia para o servidor com o client.send o endpoint /api/game/{id} e enquanto o buffer não estiver vazio recebe as informações do servidor. Essa função também trata a resposta e salva algumas informações úteis adicionando elas em uma lista. A primeira dificuldade que tivemos na implementação foi como enviar endpoints para o servidor, e só após pesquisar bastante encontramos o formato correto de envio como é mostrado a seguir:

```
entrada = f"GET /api/game/{id} HTTP/1.1\r\nHost: {IP}\r\n\r\n".encode('utf-8')
```

Caso o ‘type’ passado na função seja igual a ‘sunk’, a função trata de coletar dados a respeito dos SAGS e nos números de navios afundados de um jogo. Caso o ‘type’ seja igual a ‘escaped’, a função coleta dados a respeito da disposição de canhões e do número de navios escapados de um jogo.

```
#SALVANDO AS INFORMAÇÕES UTEIS
if type == 'sunk':
    SAGS.append(resposta['game_stats']['auth'])
    SUNK.append(resposta['game_stats']['score']['sunk_ships'])
elif type == 'escaped':
    CANNONS.append(resposta['game_stats']['cannons'])
    ESCAPED.append(resposta['game_stats']['score']['escaped_ships'])
```

3.2.2 Analisa_Conjunto

Após fazer a função Analisa_Jogo, criamos a função Analisa_Conjunto (função que requisita os 100 melhores de um tipo) que recebe uma string ‘type’.

Essa função, assim como a anterior, envia para o servidor com o client.send um endpoint, dessa vez o /api/rank/{type}?start={1}&end={100} e também recebe as informações do servidor enquanto o buffer não estiver vazio. A função retorna todos os game_ids que conseguimos na resposta do servidor e armazena em uma variável que chama game_ids.

3.2.3 Immortals

A função Immortals é responsável por realizar a análise 1. Ela calcula quantas ocorrências cada SAG teve utilizando o count e listas auxiliares para garantir que o sag não seja contado mais de uma vez e não apareça repetido na lista. Em seguida, recuperamos a lista de SAGs ordenados com a função '.sort()' utilizando as ocorrências e o reverse=True devido a lista ser do maior para o menor número de ocorrências. A função também calcula a média e adiciona a média de cada SAG na lista de SAGs ordenados.

3.2.4 Top_Meta

Esta função é responsável por gerar um número para cada disposição de canhões e ordená-las de acordo com a menor média de navios escapados. Primeiramente, a função normaliza todas as disposições de canhões armazenadas na lista 'CANNONS' em números de 8 dígitos e os colocam na lista 'metas'. Feito isso, o algoritmo conta o número de ocorrências e o número de navios escapados por cada meta. Com esses dados é possível calcular a média de navios escapados por cada meta. Feito isso, adicionamos as informações que nos importam (número, média) no dicionário 'metas_ordenados' e o ordenamos da menor média até a maior.

3.2.4 Análises

Após criarmos todas as funções, executamos as análises pedidas no documento de aula. Iniciamos pela análise 1 que é executada quando o usuário digita a tecla 1 como argumento ao rodar o cliente. Essa análise chama a função 'Analisa_Conjunto(type)' passando a string 'sunk' como parâmetro para conseguir pegar os 100 game_ids que mais afundaram navios. Em seguida, para cada id no game_id ela chama a função 'Analisa_Jogo(type, id)' passando a string 'sunk' e o id como os parâmetros para salvar os SAGS e dados sobre navios afundados que serão usados na função Immortals.

Ao tentarmos testar esse procedimento, encontramos outra dificuldade de implementação, ao passarmos o IP do servidor e a porta indicada, nós estabelecemos uma conexão que retornava status 200, porém, ao tentarmos chamar a próxima função, o cliente alegava que uma conexão estabelecida era anulada no computador host.

```
ConnectionAbortedError: [WinError 10053] Uma conexão estabelecida foi anulada pelo software no computador host
```

Após mexermos bastante com o código, a única solução encontrada pelo grupo foi encerrar a conexão manualmente a cada função de requisição com um ‘client.close()’ e abrir uma nova conexão com o ‘client.connect()’. Como já tínhamos um for que chamava a função analisaJogo() 100 vezes, apenas colocamos essas duas linhas de código nesse looping.

Por último, a função Immortals é executada e coloca seu resultado em uma variável chamada sags_ordenados no formato desejado e imprime na tela o CSV desejado. A análise 2 funciona de forma similar a análise 1 mudando apenas o comando inserido pelo usuário no terminal. Na hora de rodar o cliente, o parâmetro type passa a ser ‘escaped’ nas funções Analisa_Jogo e Analisa_Conjunto, e chamamos a função Top_Meta() no final.

- Resultado da Análise 1:

```
PS D:\Github\REDES-UFMG\TP2 - Bridge Buff> python .\client.py 127.0.0.1:5000 1
2019057160:1:9a826d21c407cda11ae1e2ecd623ff2c0fef7b71bcae83c7b632bbeb6a73474e+2019057020:2:b97ebda29201ae3c507e1efdca11d6269a73ff9bad981dabe2
e6ade97f7edc34+3c9ce75b3c2f0577498a7b99f69384900d94a846ec66f8d72de932e6ffb30e02, 42,  736.5238095238095

2019103910:1:8766567b5d1f70b7470d3729a20569f5561a6a5104e0efb152730b897aeddd52+2016060780:2:295e7dbd1b677ae86631054d5013e4f7fbda30980ba123b32b
f837ad2fe48a88+c416fcc735ecfe5eaed4e0abd083486afd2a8832fb937bd3d7788b178f92c28d, 15,  741.0666666666667

2016026191:111:7609d636ff02b901aad0bcae4b65c3f9b505d816a6215eb772f69f2d5fddf159+2015435632:111:577a175cd7c375fa798d42f01151b33911c9ddba716c7f
b0923f8a0f990f81db+33779cff20aca4725a07e82ba8cc1d2fcce74b1295e3cb89580b19e2d2013325, 8,  731.0

2017001800:2:146f91a1c3dc759f6f6857140fd007b4964031403 61dc9534fbac127ebc53d27+2013030015:2:f9d80f689032ef381dcc6f351903154737bfdf5ed2927f3318
7e9afc24ccf564+e7e25e6bfeeed424e48ad41661bae9b6a67e767436c13390703c1886467a2d4c, 7,  742.5714285714286

2019087450:1:e5c84a02008a3c52448dff503c7e5d0e9b115844cae5fb91ce641e63ad20aedc+11670cf3f34e65386e90079b2c0b3ea72eb88c35017cc7b07b0e4f328a618fa
c, 7,  734.4285714285714

2017001800:1:e8f87871e110a08d210e5f48d930c260e37bb1b5127b47a12546ad1dcf4d4553+2013030015:1:6f9e34cf5f58d0d7172df5f2a481ef1ca9e54c6ba0aa89c6c1
27e1e736efffc9+b24bf43891a5c2728fc4824535b24de24c98a02273c98d292720f93af80191a8, 6,  730.3333333333334

2019056741:2:2a7be3ceb4b1885ba079bf1579b558d3f41c153b0dbddf4033eb8fa46f8ded8c+2019056938:9:3280312e1d2a194affef4cbaf24da44b191725ef1ee3af5155c29692b39a291f+42f80fd7be8708f3179c
678f248183c10b92c891a579a2c70b66ea4cfdde446f, 4,  752.0

2019056784:1:30a70a620e3c82e9817a4606cb1f7146aea203f65e714ef80442aba080c9e29f+2019057152:2:13f5d3cfb6ef3de103241a43406d782d211fd9d831b5f278fcb572f8b5a9c389+952af6e78c9560b9230e
86046b2cb6e18f038b0b12e6bb1271a8ac031e368558, 3,  740.0

2017001800:3:6028543b20a28c53aff86066cff9be02d1d79d44a3e0cf0d6c766bf263d0a0b7+2013030015:3:a3d67bfcfe24b36b11627d2dda2c7af7138eae6b3d0f341279ae2701f18fe4f9+b2d2e6fb1fb4948d973a
d6dd59b6c24dc0f239b7dc9c4204c8fa904b37c00a31, 3,  733.0

2019057195:12142021:713956ac462e3cc9736660c44697d3b6d91ffbe60ee2911114890582c2435f72+2019056890:12142021:d4ae8849f0d2f8ccf163b12a3fcf45908b61c8f2239f3806fe6292f3428a37ce+393318
3216bb827a7cdca38687047dd1a191952b1afb1a01bcfd92ade29ae224, 2,  753.5

2017001800:4:aff4666db9eeee2765b169d15e9492d85a1aba052925f92584352b4b9fe22ba5+2013030015:4:677cfa3984354393da572642c014e5e82def0315bf51228734de457b6e4e447e+22cf352e8b49ecfb5374
5a421fe5a7273548956f5edfdd98ceb962f28a095464, 2,  744.0

2018014697:41221:c3201c42c381ff041eea2dfd400684f817db117c625dbb583f0e238581d1f758+bb074a173223346c050a643d447ca26567e1ebca7e511eadd66e1e4454ad1440, 1,  723.0
```

- Resultado da Análise 2:

```
PS D:\Github\REDES-UFMG\TP2 - Bridge Buff> python .\client.py 127.0.0.1:5000 2
01200000, 273.25

10110000, 278.6666666666667

20200000, 289.92857142857144

02010000, 290.0

12100000, 290.6764705882353

21010000, 292.75

23000000, 293.1333333333333

31100000, 294.0769230769231

04000000, 296.0

40010000, 299.0
```

Quanto às dificuldades com o server, houve uma situação em que os ids presentes na lista sunk e escaped não estavam ordenados: valores muito baixos estavam na primeira posição. O problema era que a api game/<id> retornava jogos diferentes do solicitado, o que gerou confusão quanto à ordem correta da lista e, no final, a lista estava correta, apenas o game/<id> não estava

4. Como rodar

Para iniciar o programa, é necessário ter o flask instalado, e no diretório do server digitar o seguinte comando no terminal:

python server.py

Após digitar o comando, o programa estará rodando o servidor na sua máquina e agora é necessário rodar o cliente com o seguinte comando:

python client.py {IP}:{PORT} {COMANDO}

Exemplo:

```
\TP2 - Bridge Buff> python .\client.py 127.0.0.1:5000 1
```

Assim, o programa estará rodando a função desejada modificando o COMANDO, caso seja 1, será feita a análise 1 e caso seja 2, será feita a análise 2.

5. Conclusão

Apesar das muitas dificuldades que tivemos ao longo do desenvolvimento, como a manipulação de respostas, como enviar um endpoint e como conseguir a resposta de rotas GET, que atrapalharam bastante o desenvolvimento, esse trabalho agregou bastante para fixar os conceitos aprendidos na disciplina de redes de computadores sobre programação em redes e como lidar com os problemas referentes a isso, além dos conhecimentos adquiridos sobre Web frameworks que conseguimos ao pesquisar soluções para implementar os endpoints.