Relatório do EP1 de Redes de Computadores

O EP consiste na criação de uma aplicação web, no nosso caso, um jogo. Por meio da biblioteca Pygame, elaboramos um jogo nos moldes do arcade “Space Invaders” em Python com o diferencial da conexão com um servidor web utilizando o protocolo UDP por meio da biblioteca Flask do Python.

1. Utilização do Protocolo UDP

- Configuração do Socket: definição do endereço de IP e da porta UDP, assim, criando o socket;

```
10    # Configurações do socket UDP
11    UDP_IP = "127.0.0.1"
12    UDP_PORT = 5000
13
14    # Crie um socket UDP
15    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
16
17    # Configura o socket para permitir reutilização do endereço/porta
18    udp_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, __value: 1)
19
20    # Vincula o socket ao endereço IP e porta
21    udp_socket.bind((UDP_IP, UDP_PORT))
```

- Captação dos dados: função *“listen_for_udp_messages()”* executa em uma thread separada e carrega os dados para serem enviados à página do servidor, atualizando a pontuação;

```
1 usage
def listen_for_udp_messages():
    while True:
        data, addr = udp_socket.recvfrom(1024)  # buffer size is 1024 bytes
        data = json.loads(data.decode('utf-8'))
        print(f"Dados recebidos: {data} de {addr}")

        # Atualiza os dados de pontuação
        player = data['player']
        scores[player] = data
        print(f"Pontuação atualizada: {scores}")


# Inicie uma nova thread para ouvir mensagens UDP
threading.Thread(target=listen_for_udp_messages).start()
```

- Atualização de placar no site: o método *"update_score()"* atualiza os dados pontuação, inimigos mortos, tiros disparados, nível atual, chefes mortos e vidas restantes para que *"update_server_score(player, score, enemies_killed, shots_fired, current_level, bosses_killed, player_lives)"* possa enviá-losao servidor Flask por meio de requisições POST pela rota *"http://127.0.0.1:5000/update_score"*. Os dados são enviados em formato JSON - em estrutura de dicionário - e contêm informações sobre o jogador e seu desempenho. Caso a requisição seja bem-sucedida, o servidor retorna o código 200 para a função;

```
def update_server_score(player, score, enemies_killed, shots_fired, current_level, bosses_killed, player_lives):
    url = "http://127.0.0.1:5000/update_score"
    data = {

        "player": player,
        "score": score,
        "enemies_killed": enemies_killed,
        "shots_fired": shots_fired,
        "current_level": current_level,
        "bosses_killed": bosses_killed,
        "player_lives": player_lives,
    }
    response = requests.post(url, json=data)

    if response.status_code == 200:
        print("Pontuação atualizada no servidor com sucesso.")
    else:
        print(f"Falha ao atualizar pontuação. Status code: {response.status_code}")
```

- Requisição de acesso ao servidor Flask (retirada do terminal do PyCharm):

```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 521-656-958
127.0.0.1 - - [01/Dec/2023 11:40:59] "GET / HTTP/1.1" 200 -
```

- Sintaxe da Mensagem:
    - Exemplo de mensagem enviada:

{

“player”: “JB”

“score”: 120,

“enemies_killed”: 12,

“shots_fired”: 32,

“current_level”: 1,

“bosses_killed”: 0,

“player_lives”: 80

}

    - Exemplo do terminal PyCharm (retorno das funções *“update_score()”* e *“update_server_score(player, score, enemies_killed, shots_fired, current_level, bosses_killed, player_lives)”*):

```
Pontuação atualizada no servidor com sucesso.
120 12 32 1 0 80
Pontuação atualizada no servidor com sucesso.
130 13 32 1 0 80
Pontuação atualizada no servidor com sucesso.
130 13 33 1 0 80
Pontuação atualizada no servidor com sucesso.
130 13 33 1 0 80
Pontuação atualizada no servidor com sucesso.
140 14 33 1 0 80
```

- Semântica e Regras: cada chave no dicionário representa um campo específico que será atualizado no servidor. Por exemplo, *"score"* representa a pontuação do jogador, *"enemies_killed"* o número de inimigos mortos, etc. O servidor Flask espera receber esses campos e atualiza os dados do jogador no servidor.
- Resposta do Servidor: *"update_server_score(player, score, enemies_killed, shots_fired, current_level, bosses_killed, player_lives)"* verifica o código de status da resposta do servidor. Se o código for 200, assume-se que a atualização foi bem-sucedida. Caso contrário, é exibida uma mensagem indicando falha na atualização. Abaixo, segue um histórico de requisições ao servidor após uma partida:

```
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1499, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
127.0.0.1 - - [02/Dec/2023 10:52:33] "POST /update_score HTTP/1.1" 200 -
127.0.0.1 - - [02/Dec/2023 10:52:33] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1499, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
127.0.0.1 - - [02/Dec/2023 10:52:33] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1500, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1500, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
127.0.0.1 - - [02/Dec/2023 10:52:33] "POST /update_score HTTP/1.1" 200 -
127.0.0.1 - - [02/Dec/2023 10:52:33] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1500, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1500, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
127.0.0.1 - - [02/Dec/2023 10:52:33] "POST /update_score HTTP/1.1" 200 -
127.0.0.1 - - [02/Dec/2023 10:52:34] "POST /update_score HTTP/1.1" 200 -
127.0.0.1 - - [02/Dec/2023 10:52:34] "POST /update_score HTTP/1.1" 200 -
127.0.0.1 - - [02/Dec/2023 10:52:34] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1501, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1501, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1501, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
127.0.0.1 - - [02/Dec/2023 10:52:34] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1501, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
127.0.0.1 - - [02/Dec/2023 10:52:34] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1502, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
127.0.0.1 - - [02/Dec/2023 10:52:34] "POST /update_score HTTP/1.1" 200 -
127.0.0.1 - - [02/Dec/2023 10:52:34] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1502, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 1}}
127.0.0.1 - - [02/Dec/2023 10:52:34] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 3270, 'enemies_killed': 326, 'shots_fired': 1502, 'current_level': 21, 'bosses_killed': 1, 'player_lives': 0}}
```

Em suma, a comunicação entre o jogo e o servidor é realizada por meio de requisições HTTP, utilizando o método POST para enviar dados do jogo para o servidor. O servidor Flask, por sua vez, possui uma rota específica (*"/update_score"*) para receber esses dados e atualizar as pontuações dos jogadores.

2. Descrição do código

main.py: executa o programa completo. Ambos os arquivos do servidor e do jogo são executados em threads criadas separadamente

- **def run_flask()**: executa o comando do sistema operacional para rodar app.py.
- **def run_game()**: executa o comando do sistema operacional para rodar Game.py.

app.py: faz a inicialização e envia as mensagens para o servidor

- **def listen_for_udp_messages()**: escuta mensagens UDP e atualiza os dados de pontuação. É executada em uma thread separada.
- **def update_score()**: função que pega os dados e transforma-os em uma mensagem .JSON para o servidor pela rota *"http://127.0.0.1:5000/update_score"*.
- **def index()**: função que faz a renderização da página HTML com os novos dados.

Game.py: roda o jogo, atualiza as informações em dicionário para app.py

- **Dados para o servidor**: um dicionário é criado para os dados, nele estão:

- *current_score*: a pontuação do jogador, incrementada de 10 em 10 a cada inimigo morto: *current_score* = (*enemies_killed* + *bosses_killed*)*10.
- *player_lives*: a quantidade de vidas do jogador, inicializada com 80 e decaindo de 1 em 1 conforme balas inimigas atingem o jogador. Quando chega a 0, o jogo acaba.
- *enemies_killed:* quantos inimigos foram mortos.
- *bosses_killed*: quantos chefes foram mortos.
- quantos tiros foram dados (*shots_fired*) e
- o nível até o qual o jogador chegou (*current_level*).

- **Classes de Sprites:** os sprites do jogo são inicializados (*def _ _init_ _(<parâmetros>)*) e atualizados (*def update(self)*) em suas classes de acordo com os parâmetros estabelecidos. Temos as seguintes classes:
  - Bullet: a bala disparada pelo jogador. Ela causa 30 pontos de dano em cada inimigo atingido (reduz a vida em 30 pontos).
  - Player: jogador representado pela nave. O movimento é dado pelas teclas W (cima), A (esquerda), S (baixo) e D (direita), enquanto os tiros são dados pela tecla K.
  - EnemyBullet: bala disparada pelos inimigos. Ela causa 1 dano no jogador.
  - Enemy: um inimigo com movimento lateral e disparo vertical. Possui 20 pontos de vida.
  - Boss: chefe com movimento lateral e disparo diagonal. Possui 2000 pontos de vida.

    - BossBullet: bala disparada pelo chefe. O Player perde 1 de vida a cada EnemyBullet que o atinge.
- **def is_boss_level()**: verifica se o nível atual é um nível de chefe.
- **def new_enemies()**: adiciona inimigos ao grupo de sprites com base no nível atual.
- **def update_server_score()**: envia dados de pontuação para um servidor Flask usando a biblioteca requests.
- **Loop Principal:** o loop principal executa enquanto o jogador tiver vidas restantes. Ele também verifica eventos do Pygame (por exemplo, se o usuário fechou a janela) e atualiza os sprites (verifica colisões e desenha na tela) e outras informações do jogo na tela para enfim atualizar a pontuação no servidor Flask.

3. Compilação e Execução

Para executar o código, somente abrir o app main.exe deveria ser suficiente. Porém, caso não funcione, é possível utilizar uma IDE como PyCharm, por exemplo; uma vez instalada, deve-se executar o arquivo main.py (juntando os comandos para rodar o servidor e o jogo em threads separadas). Ainda, é possível instalar Python e utilizar os seguintes comandos:

pip install pygame

pip install flask

pip install socket

pip install random

pip install requests

pip install threading

pip install json

para instalar as bibliotecas. Em seguida, para executar os códigos, utilizamos os comandos na seguinte ordem no terminal:

python app.py

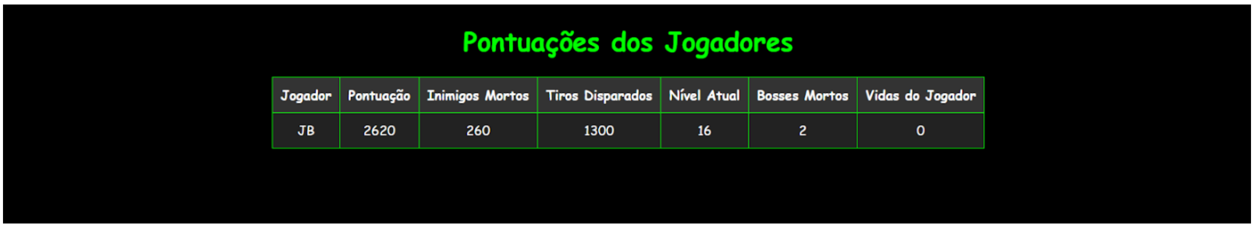
python Game.py

4. Testes

Realizamos um teste que foi gravado e enviado com os documentos. Ele está disponível em: https://drive.google.com/file/d/1H2nsRZXzFwdOlua9DTshI-xDdSTlv33Z/view?usp=sharing

.

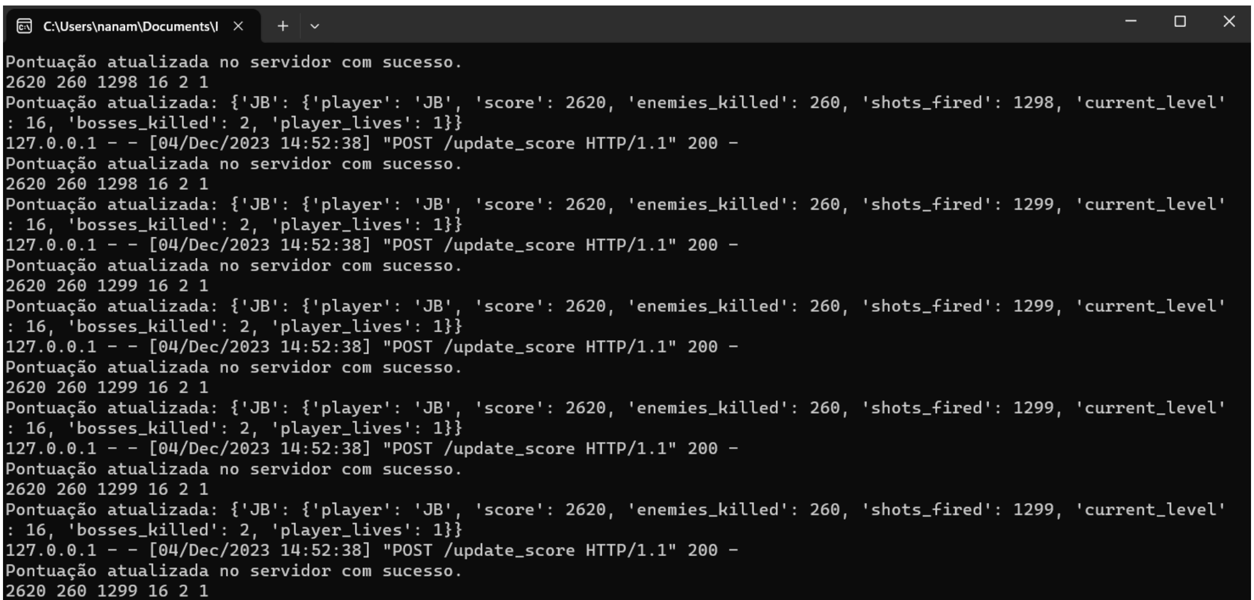
Atualização da página após término do jogo:



Pontuações dos Jogadores

Jogador	Pontuação	Inimigos Mortos	Tiros Disparados	Nível Atual	Bosses Mortos	Vidas do Jogador
JB	2620	260	1300	16	2	0

Terminal após o jogo (última mensagem é a de encerramento):




```
Pontuação atualizada no servidor com sucesso.
2620 260 1298 16 2 1
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 2620, 'enemies_killed': 260, 'shots_fired': 1298, 'current_level'
: 16, 'bosses_killed': 2, 'player_lives': 1}}
127.0.0.1 - - [04/Dec/2023 14:52:38] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada no servidor com sucesso.
2620 260 1298 16 2 1
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 2620, 'enemies_killed': 260, 'shots_fired': 1299, 'current_level'
: 16, 'bosses_killed': 2, 'player_lives': 1}}
127.0.0.1 - - [04/Dec/2023 14:52:38] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada no servidor com sucesso.
2620 260 1299 16 2 1
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 2620, 'enemies_killed': 260, 'shots_fired': 1299, 'current_level'
: 16, 'bosses_killed': 2, 'player_lives': 1}}
127.0.0.1 - - [04/Dec/2023 14:52:38] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada no servidor com sucesso.
2620 260 1299 16 2 1
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 2620, 'enemies_killed': 260, 'shots_fired': 1299, 'current_level'
: 16, 'bosses_killed': 2, 'player_lives': 1}}
127.0.0.1 - - [04/Dec/2023 14:52:38] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada no servidor com sucesso.
2620 260 1299 16 2 1
Pontuação atualizada: {'JB': {'player': 'JB', 'score': 2620, 'enemies_killed': 260, 'shots_fired': 1299, 'current_level'
: 16, 'bosses_killed': 2, 'player_lives': 1}}
127.0.0.1 - - [04/Dec/2023 14:52:38] "POST /update_score HTTP/1.1" 200 -
Pontuação atualizada no servidor com sucesso.
2620 260 1299 16 2 1
```

Além disso, durante a gameplay, se o usuário atualizar a página múltiplas vezes, perceberá que os placares vão mudando, o que comprova a recepção da imagem.