Protocolo de um jogo de corrida online em UDP - Documentação de código

Willyan P. Bueno , Vitor A. B. Von Gilsa , Gabriel R. Verruck , Sofia M. Port

Instituto Federal de Educação, Ciência e Tecnologia - Campus Concórdia
Distrito Fragosos - s/n Km 8, Concórdia - SC, 89700-000 - Brasil

willyanpaproski123@gmail.com, vitoraipumirim@gmail.com,
verruckgabriel@gmail.com, sofia.manduca.port06092004@gmail.com

1. Resumo

Dada a proposta de desenvolver um sistema que utilize dos protocolos das camadas de aplicação e transporte da rede, este documento tem como objetivo mostrar como foi o processo de elaboração da atividade. Será apresentado as ferramentas utilizadas, métodos, e explicação da funcionalidade.

2. Introdução

Para a realização deste projeto, tivemos como escolha 7 propostas para serem feitas com a linguagem de programação Python:

1. Chat Seguro, utilizando criptografia simétrica;
2. Compartilhamento de Arquivos;
3. Controle de Ar Condicionado via sistema;
4. Streaming de Áudio;
5. Jogo Online de Tiro em Primeira Pessoa;
6. Jogo RPG Online;
7. Jogo Online de Corrida Multiplayer.

Em concordância com o grupo, selecionamos o Jogo Online de Corrida Multiplayer para desenvolver.

3. Ferramentas

Para realizar a proposta escolhida pesquisamos e selecionamos algumas bibliotecas do Python para utilizar no desenvolvimento:

1. **Threading:** Esta biblioteca torna possível a utilização de threads em nosso código. Segundo Developerworks (2012) Threads são fluxos de programas que executam em paralelo dentro de uma aplicação, isto é, uma ramificação de uma parte da aplicação que é executada de forma independente e escalonada independentemente do fluxo inicial da aplicação.
2. **Socket:** De acordo com Machado (2018), Sockets são usados para enviar dados através da rede. Para que isso seja feito, devemos estabelecer um servidor, e os clientes, que vão trocar mensagens através dos protocolos das camadas de aplicação e transporte, sendo eles TCP e UDP.
3. **Time:** Segundo Python (), a biblioteca Time é nativa da linguagem python, e provê várias funções relacionadas a tempo.

4. Estrutura da aplicação

Para construir a aplicação, dividimos ela em servidor e cliente, onde juntos fazem todo o sistema ser executado:

4.1. Servidor

O lado do servidor precisa sempre estar funcionando para os clientes se conectarem e jogarem o jogo de corrida, que foi feito de acordo com as regras de um DragRacing.

Primeiramente, dentro de uma condição onde o código tenta criar uma variável que basicamente define o próprio servidor, e junto da função socket, é definido para ele utilizar somente endereços do tipo v4 e protocolo UDP para troca de dados. Após esta definição, o servidor vai estabelecer sua conexão no IP da máquina onde será hospedado, na porta 10101, e por fim é mostrado no terminal de execução, a mensagem ”Servidor conectado”. Depois, temos um loop onde o servidor sempre estará recebendo mensagens dos clientes, e as transformando de bytes para o padrão de caracteres utf-8. Tendo as mensagens do cliente decodificadas, é estabelecido uma condição onde é verificado se o cliente enviou um ”y”ou um ”Y”, e caso isso for verdadeiro, o servidor vai mandar uma mensagem de confirmação para que o jogo de início no lado do cliente. Caso estas etapas não ocorram, será mostrado no terminal de execução a mensagem ”Não foi possível conectar-se ao servidor”. Veja no código à seguir:

```
1
import socket
3
try:
5
    udpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7   udpSocket.bind(('10.0.23.107', 10101))

9   print('Servidor conectado')

11  while(True):

13      bytesClient = udpSocket.recvfrom(2048)
        message = bytesClient[0]
15      address = bytesClient[1]

17      m = message.decode('utf-8')

19      if(m == 'y' or m == 'Y'):

21          contagemMsg = 'ok'
            contagem = str.encode(contagemMsg)
23          udpSocket.sendto(contagem, address)

25          print(f'Confirmacao do jogador concedida')

27      print(m)

29 except:

31  print('Nao foi possivel conectar-se ao servidor')
```

4.2. Cliente

O lado do servidor precisa estar conectado ao servidor para funcionar, e ser possível a execução de um jogo de corrida DragRacing.

Para desenvolver o código do lado do cliente, foram utilizadas as bibliotecas threading, socket e time da linguagem de programação python. Para as regras do jogo, foram utilizadas as mesmas funções da versão do jogo em conexão TCP:

1. **Função marchas():** Esta função é basicamente o que define as regras do Drag Racing, e faz a interação com o usuário para ele jogar, e recebe como parâmetro o nome de usuário, e o cliente. Primeiramente é definido uma variável que armazena o inicio da corrida, e outra que armazena a marcha que o jogador inicia, sendo a marcha um. Depois é feito um loop condicional, onde ele executa enquanto a variável marcha for menor do que cinco. Dentro deste loop, é mostrado no terminal a mensagem "Acelerando", e o código espera um tempo de o valor da marcha que o jogador está mais três segundos para ir para a próxima etapa. Após este tempo de espera, é mostrado uma mensagem para o jogador pressionar a tecla Q e pressionar Enter para passar a marcha. Quando o jogador realiza as instruções, é feito uma verificação onde é averiguado se o usuário às fez corretamente, e caso o jogador tenha feito certo, é acrescentado um valor a mais na marcha e mostrado no terminal a marcha em que o jogador está. Quando a marcha alcança o valor de cinco, é executado um bloco de código, onde acrescentamos o valor "6"à variável marcha, e é definido o fim da corrida. Após isso é calculado o tempo que o usuário levou para finalizar a corrida (determinante para saber quem venceu em um Drag Racing), e mostrado no terminal juntamente do nome e o tempo que o jogador levou para alcançar o final da corrida, a mensagem "Jogador ¡nome do jogador¿ alcançou a linha de chegada em ¡tempo¿ segundos. Também, através da função send(), é enviado ao servidor o resultado da corrida. Veja no código à seguir:

```
2     def marchas(cliente, username):

4     inicioCorrida = round(float(time.time()%60), 2)

6     marcha = 1

8     while marcha < 5:

10        print('Acelerando\n')
          time.sleep(marcha + 3)
12        marchaComando = input('Pressione <Q> e <ENTER> para
              passar a marcha: ')

14        if(marchaComando == 'q' or marchaComando == 'Q'):
```

```
16              marcha+=1
                print(f"Marcha {marcha}!")
18
            else:
20
                print('Game Over!')
22              exit()

24          if(marcha == 5):

26              time.sleep(8)
                marcha = 6
28
                fimCorrida = round(float(time.time()%60), 2)
30              resultado = round((fimCorrida - inicioCorrida), 2)

32              chegada  = f'Jogador <{username}> alcancou a linha de
                    chegada em {abs(resultado)} segundos'
                cliente.send(f'{chegada}'.encode('utf-8'))
34
                print(chegada)
```

2. **Função contagemRegressiva():** Como o próprio nome sugere, esta função realiza uma contagem regressiva. É utilizada no código para fazer a contagem antes do jogo ser iniciado, sendo de cinco segundos. Para que a contagem seja realizada, é definida uma variável que recebe o valor "6", e depois é feito um loop condicional onde enquanto a variável anteriormente declarada for maior que um, é mostrado no terminal o valor desta menos 1, o código aguarda um segundo para realizar a próxima etapa, e decresce o valor da variável em um. Confira abaixo:

```
2       def contagemRegressiva():

4       contagem = 6

6       while contagem > 1:

8           print(contagem-1)
            time.sleep(1)
10          contagem-=1
```

3. **Execução do jogo:** Para o jogo ser executado, dentro de uma condição, o código tenta executar um loop, onde primeiramente é definido a conexão do cliente, juntamente da função socket, fazendo-o realizar conexões somente com endereços do tipo v4 e utilizar o protocolo UDP para troca de pacotes na rede. Depois em uma variável, passamos o endereço IP do servidor e a porta em que ele está conectado para que seja possível o envio de mensagens do cliente. Após estas configurações, o jogo toma início mostrando uma mensagem perguntando se o cliente deseja iniciar o jogo, e é verificado se o retorno do mesmo é um

”y”ou um ”Y”, e caso isso seja verdadeiro, essa mensagem é enviada ao servidor, que retorna uma confirmação, fazendo com que o jogo comece a ser executado chamando as funções contagemRegressiva() e marchas(). Caso o retorno do cliente seja diferente, a sua aplicação é encerrada, e a conexão com o servidor perdida. E caso nenhuma destas etapas executem, é mostrado a mensagem ”Game over!”no terminal do cliente. Confira no código abaixo:

```
1
    try:
3
    while True:
5
        udpScoket = socket.socket(socket.AF_INET, socket.
            SOCK_DGRAM)
7       portServer = ('10.0.23.107', 10101)

9       msgStart = input('Pronto?(Pressione a tecla <Y> para
            confirmar) ')
        if(msgStart == 'y' or msgStart == 'Y'):
11
            msgStartSend = str.encode(msgStart)
13          udpScoket.sendto(msgStartSend, portServer)
        else:
15
            exit()
17

19      username = input('Digite seu Username: ')

21      msgServer = udpScoket.recvfrom(2048)
        msg = msgServer[0].decode()
23
        if(msg == 'ok'):
25
            print('Verificacao do servidor realizada, o jogo pode
                iniciar')
27          contagemRegressiva()
            print('Marcha 1!')
29          marchas(username)
            exit()
31
    except:
33
        print('Game Over!')
```

Contudo, o código para o lado do cliente executar o jogo fica da seguinte forma:

```
2   import socket
    import time
4
    def contagemRegressiva():
```

```
6
        contagem = 6
8
        while contagem > 1:
10          print(contagem-1)
            time.sleep(1)
12          contagem-=1
14  def marchas(username):
16      inicioCorrida = round(float(time.time()%60), 2)
18      marcha = 1
20      while marcha < 5:
22          print('Acelerando\n')
            time.sleep(marcha + 3)
24          marchaComando = input('Pressione <Q> e <ENTER> para
                passar a marcha: ')
26          if(marchaComando == 'q' or marchaComando == 'Q'):
28              marcha+=1
                print(f"Marcha {marcha}!")
30
            if(marcha == 5):
32
                time.sleep(8)
34              marcha = 6
36              fimCorrida = round(float(time.time()%60), 2)
                resultado = round((fimCorrida - inicioCorrida),
                    2)
38
                chegada  = f'Jogador <{username}> alcancou a
                    linha de chegada em {abs(resultado)} segundos
                    '
40              chegadaSend = str.encode(chegada)
                udpScoket.sendto(chegadaSend, portServer)
42
                print(chegada)
44
46  try:
48      while True:
            udpScoket = socket.socket(socket.AF_INET, socket.
                SOCK_DGRAM)
50          portServer = ('10.0.23.107', 10101)
52          msgStart = input('Pronto?(Pressione a tecla <Y> para
                confirmar) ')
            if(msgStart == 'y' or msgStart == 'Y'):
54
                msgStartSend = str.encode(msgStart)
```

```
56                  udpScoket.sendto(msgStartSend, portServer)
            else:
58
                    exit()
60
62          username = input('Digite seu Username: ')

64          msgServer = udpScoket.recvfrom(2048)
            msg = msgServer[0].decode()
66
            if(msg == 'ok'):
68
                    print('Verificacao do servidor realizada, o jogo
                        pode iniciar')
70                  contagemRegressiva()
                    print('Marcha 1!')
72                  marchas(username)
                    exit()
74
    except:
76
        print('Game Over!')
```

Referências

DEVELOPERWORKS, B. **Threads em Python**. 2012. Url https://imasters.com.br/back-end/threads-em-python. Acessado em: 2023/05/21.

MACHADO, A. **Socket em Python**. 2018. Url https://blog.4linux.com.br/socket-em-python/. Acessado em: 2023/05/21.

PYTHON, O. **time - Acesso ao horário e conversões**. Url https://docs.python.org/pt-br/3.9/library/time.html. Acessado em: 2023/05/21.