Trabalho Prático

Redes de Computadores I

Relatório pertinente ao Trabalho Prático, da disciplina de Redes de Computadores I, com objetivo de documentar e esclarecer o processo de desenvolvimento de um Jogo da Memória Online, a partir do uso da biblioteca Socket.

Universidade Federal Fluminense (UFF)
Instituto de Computação (IC)
Professora: Flavia Coimbra Delicato

Alunos:

- ➢ Amanda Barros Melo
- ➢ Luiz Marcelo Pereira Torre
- ➢ Thiago Prado Azevedo Andrade

Sumário

Documentação

A linguagem escolhida para a implementação do trabalho é Python, todavia o código foi modificado para Python 3. O projeto possui três arquivos de códigos, o primeiro é o JogoDaMemoria.py, e os outros dois são responsáveis pelo funcionamento do Cliente e do Servidor, ambos possuem classes e métodos para assegurar a organização dos códigos.

O Servidor é implementado da seguinte forma:

```
7   class Server:
8
9       HOST = "localhost"  # IP local
10      PORT = 1331         # porta escolhida para conexão
11      socket = None       # socket do servidor, sem valor atribuído
12      clients = []        # lista de clientes conectados
13      playersConnected = False
14      messageBuffer = []
15
16      def start(n_players: int):
17          ''' Função para inicializar o servidor. '''
18          Server.socket = Socket(socket.AF_INET, socket.SOCK_STREAM)
19
20          try:
21              Server.socket.bind((Server.HOST, Server.PORT))
22          except socket.error as e:
23              print(e)
24              sys.exit()
25          Server.socket.listen()
26          start_new_thread(Server.start_t,(n_players,))
```

A partir da classe Server, temos os atributos relacionados a criação da conexão (HOST, PORT, socket), e os atributos para o funcionamento de múltiplas conexões (clients, playersConnected, messageBuffer). As funções da classe são:

- ➔ Start, como visto na imagem acima, possui como objetivo criar o socket, e estabelecer a conexão inicial com tratamento de erro, depois inicia a thread do servidor Start_t.
- ➔ Start_t, é uma thread responsável por esperar as N conexões definidas pela variável n_players, e dar início ao jogo após todos os clientes se conectarem.

- ➔ ResetSeverInfo, esse método retorna os valores dos atributos: clients, playersConnected, messageBuffer, ao valor inicialmente atribuído pela classe **Server**.
- ➔ Client_t, é uma thread desenvolvida para receber as mensagens dos clientes, essa é iniciada após as conexões estarem estabelecidas na Start_t. Esse método averigua se o cliente permanece conectado ao servidor ou não.
- ➔ Send, é um método que realiza a conversão da string para bytes e envia essa mensagem para um cliente específico, caso ele exista.
- ➔ Send_others, seu objetivo é dado uma conexão com um certo índice, uma mensagem é enviada a todos os clientes, exceto esse específico.
- ➔ Send_all, como o método anterior, enviamos uma mensagem a todos os clientes, dessa vez sem exceções.

O Cliente é implementado da seguinte forma:

```
7   class Client():
8
9       def __init__(self):
10          ''' Inicia o Cliente '''
11          self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12          self.server = "localhost"
13          self.port = 1331
14          self.addr = (self.server, self.port)
15          self.id = self.connect()
16          self.last_response = ""
17
18      def connect(self):
19          ''' Realiza a conexão com o servidor '''
20          try:
21              self.client.connect(self.addr)
22              return self.client.recv(1024).decode().split(":")[1] #id
23          except:
24              print("Failed to connect.")
25              pass
```

A partir da classe Client, temos o init com os atributos da classe: client (cria o socket do cliente), server (dita o IP do servidor), port (indica a porta a ser utilizada), addr (endereço, une o IP e a porta), id (a conexão ocorre e retorna o id desse cliente), last_response (string responsável por armazenar a última mensagem). As funções da classe são:

- ➔ Connect, como visto na imagem acima, possui como objetivo estabelecer a conexão inicial com tratamento de erro, depois retorna o id do cliente.

- ➔ Send, é um método que realiza a conversão da string para bytes e envia essa mensagem para o servidor, com tratamento de erro.
- ➔ Receive_messages, é o método mais complexo das duas classes, pois mantém a conexão procurando por mensagens do servidor, decodifica a mensagem recebida, processa as mensagens, caso mais de uma tenha sido enviada simultaneamente. Além disso, o console do cliente permanece limpo, sem conseguir rever o tabuleiro anteriormente aberto, pois é executado o comando nessa função. Em seguida, as mensagens são tratadas, caso sejam ações, erros, informes especiais (ex. o placar), e se o jogo chegou ao fim, ou então são impressas no terminal da forma como foram entregues.
- ➔ Close, seu objetivo é fechar a conexão do cliente.

As bibliotecas importadas para ambos os códigos são: socket (responsável pela comunicação), _thread (utilizado no servidor), sys (para fechar o servidor em caso de erro), os (com o objetivo de realizar a manutenção no terminal do cliente).

Modo de Uso

Para rodar o cliente e o servidor é necessário primeiro rodar o código principal do jogo (JogoDaMemoria.py), e então duas perguntas são feitas:

1. Tamanho do tabuleiro.
2. Número de jogadores.

Após esse momento, o servidor será ligado automaticamente. Depois, para conectar o cliente é apenas necessário rodar o programa do cliente (cliente.py). Cada vez que o cliente for executado em um terminal diferente será um novo jogador conectado na partida até que todos estejam conectados.

Decisões e melhorias

A respeito das melhorias, nosso grupo optou por separar as mensagens caso essas fossem enviadas em conjunto, e durante o Jogo da Memória, no método LeCoordenada, implementamos o recebimento das mensagens do cliente e já lidamos com o Buffer mantendo ele limpo. Como visto a seguir:

```
207  def leCoordenada(dim, vez):
208      Server.send(vez, "> \nEspecifique uma peca: ")
209
210      #Esperando mensagem do servidor
211      while len(Server.messageBuffer[vez]) == 0: pass
212
213      inp = Server.messageBuffer[vez].pop(0)
214
215      try:
216          i = int(inp.split(' ')[0])
217          j = int(inp.split(' ')[1])
```

Com o estudo que tivemos, esse tráfego de informação do cliente para o servidor pode ser implementado de forma mais simplificada, mas optamos por esse método por ser o mais seguro para receber a mensagem.

Política de Colaboração

Para a realização do trabalho foi decidido pelo grupo que a melhor forma de colaboração seria participarmos de maneira síncrona, remotamente. Portanto, o nosso grupo utilizou um aplicativo de comunicação online, Discord, onde um dos integrantes compartilhava a tela e os demais do grupo decidiam juntos a melhor forma de solucionar cada problema.

Dessa forma, realizamos um revezamento no desenvolvimento, e na pesquisa, pois enquanto um escrevia o código, os outros pesquisavam sobre os métodos e funções ofertados pelas bibliotecas. Em resumo, construímos todo o trabalho por meio dessas reuniões marcadas online.

Em relação ao relatório, decidimos criar um “Documentos do Google”, onde os integrantes poderiam escrever ao mesmo tempo. Separamos as seções entre nós e estabelecemos a política de revisar os textos uns dos outros.

Considerações finais

Os integrantes nunca tiveram contato com a biblioteca Socket anteriormente, por isso o projeto se mostrou mais desafiador do que o esperado. Porém, a experiência de desenvolvimento se mostrou rica, interessante, e definitivamente nos auxiliou a compreender os conceitos dados em sala de aula. Além disso, realizar testes na aplicação nos acrescentou bastante em conhecimento, tanto de programação, quanto de comunicação em redes.

A classe Servidor teve sua construção marcada principalmente pela tentativa e erro, e também por conhecimentos adquiridos em outras disciplinas anteriormente cursadas. Em contrapartida, a classe Cliente, seu desenvolvimento se deu através de mais pesquisas, principalmente, pois deparamos com erros distintos, tais como mensagens concatenadas.

Em resumo, agradecemos pela oportunidade de trabalhar nesse projeto, e esperamos que esteja de acordo com o esperado.