

Relatório do Laboratório 1:

File Transfer Protocol (FTP)

Isabelle Ferreira de Oliveira
CES-35 - Engenharia da Computação 2020
Instituto Tecnológico de Aeronáutica (ITA)
São José dos Campos, Brasil
isabelle.ferreira3000@gmail.com

Resumo—O trabalho tem como objetivo a implementação de um servidor de transferência de arquivos básico inspirado na interface do File Transfer Protocol (FTP), com os requisitos conforme apresentados no roteiro do laboratório.

Index Terms—FTP, Redes de computadores, Cliente-Servidor

I. IMPLEMENTAÇÃO

A linguagem de programação utilizada foi Python 3, com o auxílio das bibliotecas: *socket* para comunicação entre os processos, *thread* para a criação das threads para cada cliente conectado ao servidor, e *shutil* e *os* para os comandos de manipulação de diretórios e arquivos.

O código do Servidor pode ser simplificada e explicado como a seguir. Ele se trata, inicialmente, de um loop infinito esperando tentativas de conexões advindas de clientes. Cada tentativa de conexão inicia uma thread que lida com cada sessão em particular. Ao chegar essa tentativa de conexão, o servidor começa, então, o processo de autenticação do cliente. Se o cliente fornecer corretamente seu nome de usuário e senha, inicia-se, por fim, outro loop infinito, dessa vez a espera de comandos digitados pelos usuários nos processos clientes. Caso o cliente falhe em sua autenticação, a conexão é encerrada. Cada comando que chega é devidamente lido de acordo com suas peculiaridades.

Já o código do Cliente, de forma superficial, trata-se de um loop infinito esperando comandos advindos do usuário. Caso esteja conectado ao servidor, cada linha de comando é enviado para o servidor e ambos os processos passam a lidar com os resultados esperados. Caso esteja desconectado, o cliente é incapaz de realizar qualquer comando, exceto pelo de conectar-se ao servidor (comando *open <server>*). Ao tentar se conectar ao servidor, o usuário é requisitado de digitar o nome de usuário e senha, para ocorrer a autenticação por parte do servidor.

Como cada comando é implementado, além de outras funcionalidades auxiliares foram melhores abordados nas subseções a seguir.

A. Funções auxiliares

Cada mensagem trocada pelo cliente e servidor é recebida através da função a seguir. Essa função recebe a mensagem em pacotes de tamanho 16, concatenando-os para formar a mensagem completa. O fim da mensagem é identificado a partir de um *carriage return* e *line feed*.

```
# Server and Client code
def receive_message(conn):
    message_received = ""
    while True:
        data_received = conn.recv(16)
        message_received = message_received +
            data_received.decode("utf-8")
        if message_received.endswith("\r\n"):
            break
    message_received =
        message_received.split("\r\n")[0]
    return message_received
```

A autenticação realizada no servidor foi apresentada a seguir. Ela se resume a comparação dos dados fornecidos pelo cliente aos armazenados em um mapa com nomes de usuários e senhas. Esse mapa é construído a partir da leitura de um arquivo de credenciais, e seu código também é apresentado a seguir.

```
# Server code
def check_authentication(conn):
    conn.sendall(bytes("user:\r\n", 'utf-8'))
    username = receive_message(conn)

    conn.sendall(bytes("password:\r\n",
        'utf-8'))
    password = receive_message(conn)

    authenticate = False
    if username in credentials:
        if credentials[username] == password:
            authenticate = True

    return authenticate, username
```

```
# Server code
def create_credentials():
    credentials_file = open("credentials.txt",
        "r")
    lines = credentials_file.readlines()
    for line in lines:
        username = line.split(" ")[0]
        password = line.split(" ")[1]
        password = password.split("\n")[0]
        credentials[username] = password
```

As linhas de comando são digitadas pelo usuário no código

do cliente, e essas linhas também são enviadas ao servidor. O parseamento dos comandos para serem lidados posteriormente são feitos através da função *split* do Python, e teve seu código apresentado a seguir.

```
# Server and Client code
command = command_line.split(" ")[0]
args = command_line.split(" ")[1:]
```

B. Navegação e listagem de diretórios

```
# Server code
if comm == "cd":
    curr_path = curr_session.current_directory

    try:
        os.chdir(curr_path + "/" + dirname)
        curr_path = os.getcwd()
        curr_session.current_directory =
            curr_path
        conn.sendall(bytes("ok\r\n", 'utf-8'))

    except FileNotFoundError:
        conn.sendall(bytes("Error: directory
            does not exists in server\r\n",
                'utf-8'))
```

```
# Server code
elif comm == "ls":
    if len(args) != 0:
        dirname = args[0]

    try:
        path = curr_session.current_directory
            + "/" + dirname
        fileslist = os.listdir(path)
        conn.sendall(bytes(str(fileslist) +
            "\r\n", 'utf-8'))

    except FileNotFoundError:
        conn.sendall(bytes("Error: directory
            does not exists in server\r\n",
                'utf-8'))

    else:
        path = curr_session.current_directory
        fileslist = os.listdir(path)
        conn.sendall(bytes(str(fileslist) +
            "\r\n", 'utf-8'))
```

```
# Server code
elif comm == "pwd":
    path = curr_session.current_directory
    conn.sendall(bytes(path + "\r\n", 'utf-8'))
```

C. Manipulação de diretórios

```
# Server code
elif comm == "mkdir":
    try:
        path = curr_session.current_directory
        os.mkdir(path + "/" + dirname)
        conn.sendall(bytes("ok\r\n", 'utf-8'))
```

```
except OSError:
    conn.sendall(bytes("Error: directory
        already exists in server\r\n",
            'utf-8'))
```

```
# Server code
elif comm == "rmdir":
    try:
        path = curr_session.current_directory
        shutil.rmtree(path + "/" + dirname)
        conn.sendall(bytes("ok\r\n", 'utf-8'))
```

```
except OSError:
    conn.sendall(bytes("Error: directory
        does not exists in server\r\n",
            'utf-8'))
```

D. Manipulação de arquivos

```
# Server code
elif comm == "get":
    if check_if_file_exists(conn, filename):
        # if file not exists in local
        # or if can overwrite
        feedback = receive_message(conn)

        if feedback == "can get":
            path = curr_session.current_directory
            file = open(path + "/" + filename,
                "rb")
            aux = file.read(1024)
            while aux:
                conn.send(aux)
                aux = file.read(1024)
            conn.sendall(bytes("\r\n", 'utf-8'))
```

```
# Client code
elif command == "get":
    # if file exists in server
    feedback = receive_message(sock)

    if feedback == "file already exists":
        # if file exists in local
        already_exists =
            check_if_file_already_exists(filename)
```

```
can_get = True
if already_exists:
    print("File already exists. Do you
        want to overwrite local file?
        [Y/N]")
    while True:
        answer = input()
        if answer.upper() == "Y":
            break
        elif answer.upper() == "N":
            can_get = False
            break
    else:
        print("Invalid answer. Please,
            answer with Y or N.")
```

```

if can_get:
    sock.sendall(bytes("can get\r\n",
        'utf-8'))
    f = open(str(filename), 'wb')
    aux = sock.recv(1024)
    while aux:
        f.write(aux)
        aux = sock.recv(1024)
        if aux.endswith(bytes("\r\n",
            'utf-8')):
            break
    print("File downloaded!")
else:
    sock.sendall(bytes("can not get\r\n",
        'utf-8'))

else:
    print("Error: file does not exists in
        server")

```

```

# Server code
elif comm == "put":
    # if file exists in local
    feedback = receive_message(conn)

    if feedback == "files exists in local":
        filename = filename.split("/")[-1]

        # if file exists in server
        check_if_file_already_exists(conn,
            filename)

    can_continue = receive_message(conn)
    if can_continue == "Y":
        path = curr_session.current_directory
        f = open(path + "/" + filename, 'wb')
        aux = conn.recv(1024)
        while aux:
            f.write(aux)
            aux = conn.recv(1024)
            if aux.endswith(bytes("\r\n",
                'utf-8')):
                break
    else:
        conn.sendall(bytes("ok\r\n", 'utf-8'))

```

```

# Client code
elif command == "put":
    # if file exists in local
    file_exists_in_local =
        check_if_file_exists(filename)

    if file_exists_in_local:
        sock.sendall(bytes("files exists in
            local\r\n", 'utf-8'))

        # if file exists in server
        feedback = receive_message(sock)
        can_receive = False

    if feedback == "file already exists":
        print("File already exists. Do you
            want to overwrite remote file?
            [Y/N]")

```

```

while True:
    answer = input()
    if answer.upper() == "Y":
        sock.sendall(bytes("Y\r\n",
            'utf-8'))
        can_receive = True
        break
    elif answer.upper() == "N":
        sock.sendall(bytes("N\r\n",
            'utf-8'))
        break
    else:
        print("Invalid answer. Please,
            answer with Y or N.")

else:
    can_receive = True
    sock.sendall(bytes("Y\r\n", 'utf-8'))

if can_receive:
    file = open(filename, "rb")
    aux = file.read(1024)
    while aux:
        sock.send(aux)
        aux = file.read(1024)
        sock.sendall(bytes("\r\n", 'utf-8'))
        print("File sent!")

else:
    sock.sendall(bytes("files does not exists
        in local\r\n", 'utf-8'))

```

```

# Server code
elif comm == "delete":
    try:
        path = curr_session.current_directory
        os.remove(path + "/" + filename)
        conn.sendall(bytes("ok\r\n", 'utf-8'))

    except OSError:
        conn.sendall(bytes("Error: file does not
            exists in server\r\n", 'utf-8'))

```

E. Gerenciamento de conexões

```

# Server code
if command == "close" or command == "quit":
    sessions.pop(conn)
    conn.close()
    break

```

II. DIAGRAMAS DE SEQUÊNCIA

III. FUNCIONAMENTO

A. Implementação da Definição da Rede Neural

Essa primeira etapa se tratou da implementação do método `build_model()` da classe `DQNAgent` de `dqn_agent.py`, script fornecido no código base do laboratório. Nesse método, era preciso construir uma rede em Keras de acordo com as especificações apresentadas na Tabela 3 do roteiro do laboratório [1].

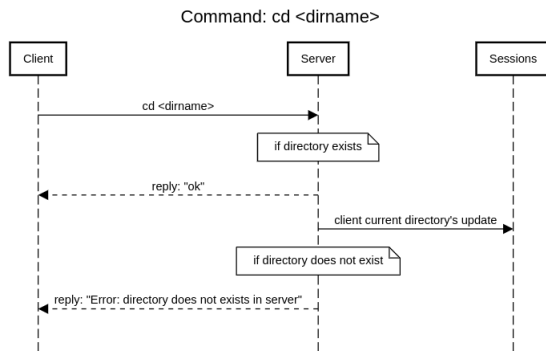


Figura 1. Sumário do modelo implementado em Keras.

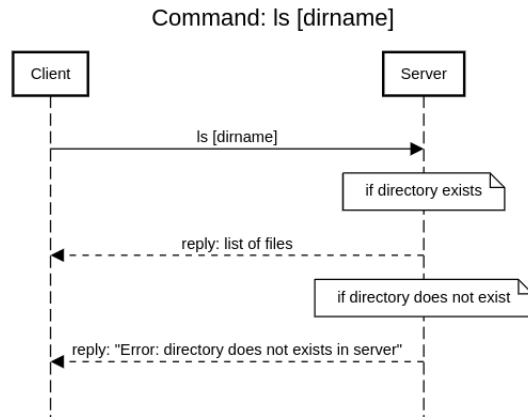


Figura 2. Sumário do modelo implementado em Keras.

Essa implementação foi feita de forma bastante análoga à maneira do laboratório 8 [2], ou seja, seguindo o apresentado no pseudo-código em Python a seguir.

```

# Adds the first layer
model.add(layers.Dense(num_neurons,
    activation=activations.some_function,
    input_dim=state_size))

# Adds another layer (not first)
model.add(layers.Dense(num_neurons,
    activation=activations.some_function))
  
```

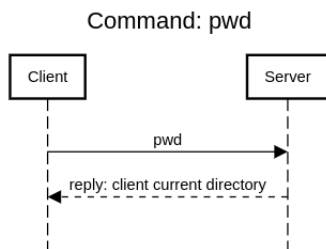


Figura 3. Sumário do modelo implementado em Keras.

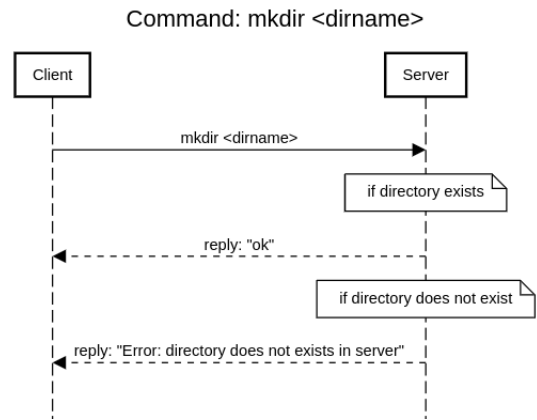


Figura 4. Sumário do modelo implementado em Keras.

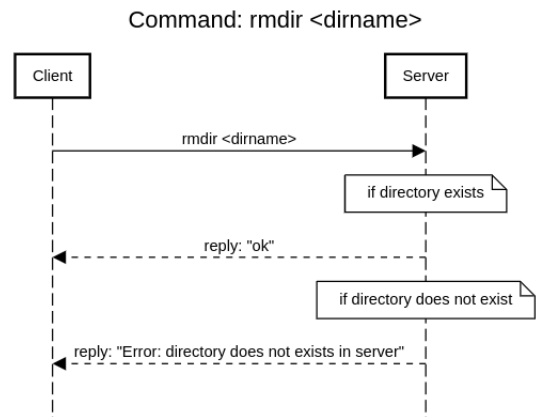


Figura 5. Sumário do modelo implementado em Keras.

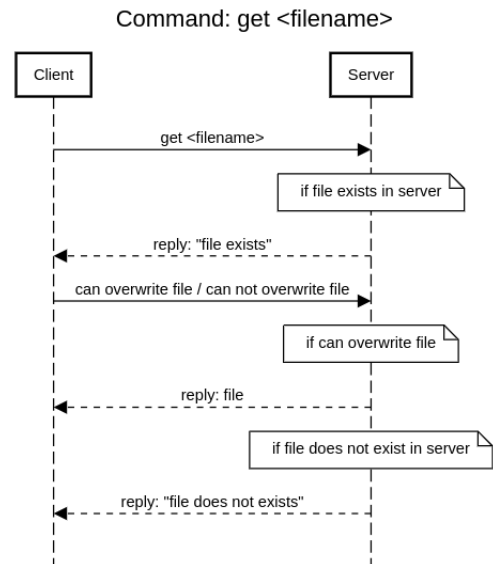


Figura 6. Sumário do modelo implementado em Keras.

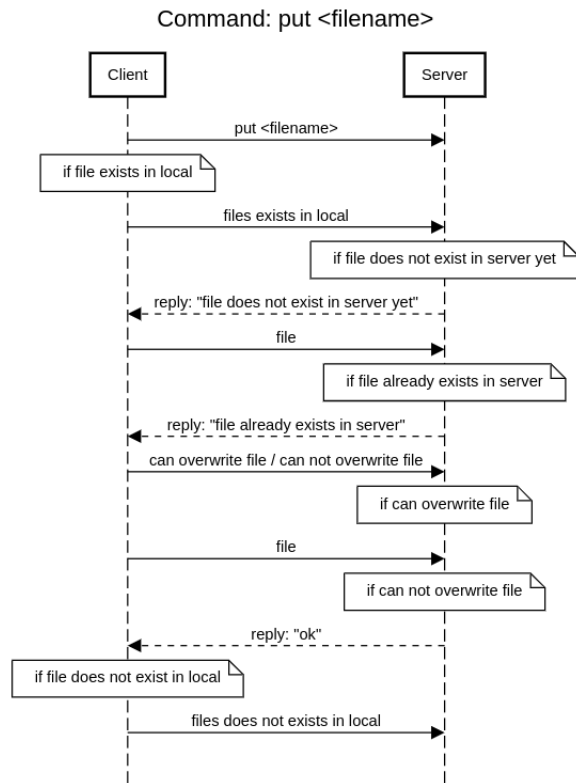


Figura 7. Sumário do modelo implementado em Keras.

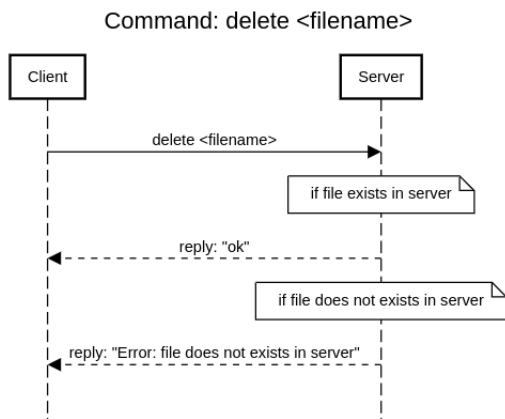


Figura 8. Sumário do modelo implementado em Keras.

Vale ressaltar que, para atender os critérios requisitados, some_function do pseudo-código acima se tratou de *relu* para as duas primeiras camadas, e de *linear* para terceira camada. Além disso, num_neurons foram 24, 24 e action_size para as primeira, segunda e terceira camada, respectivamente.

Fora isso, bastou-se descomentar as linhas de criação de uma pilha linear de camadas, as linhas compilação do modelo e impressão do summary do modelo, apresentado futuramente na seção IV (Resultados e Conclusões), e a linha de retorno da função.

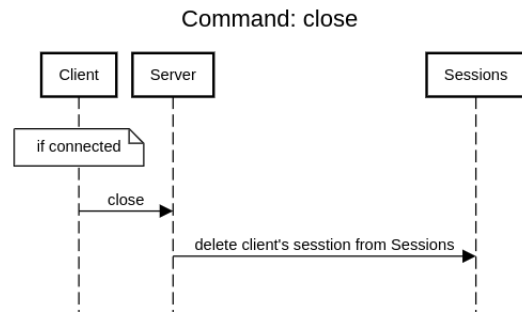


Figura 9. Sumário do modelo implementado em Keras.

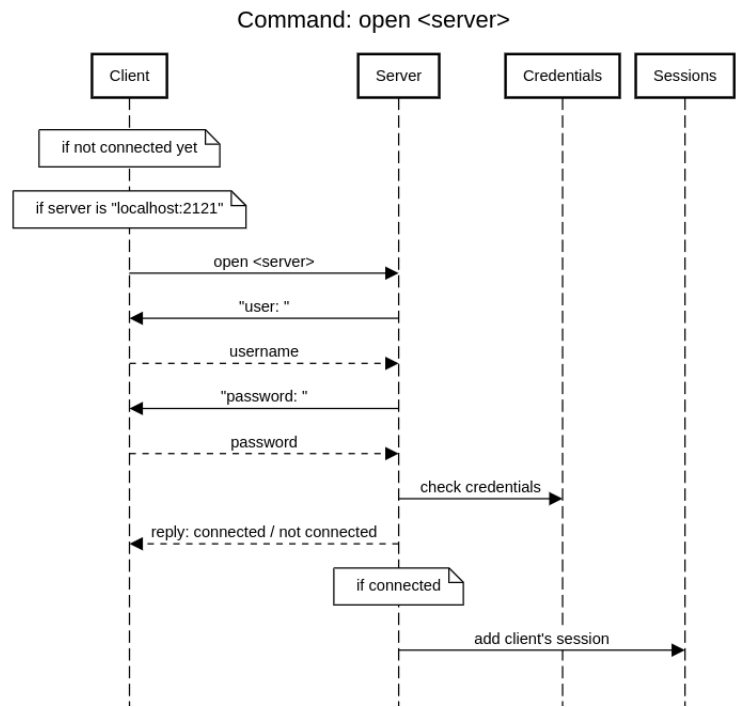


Figura 10. Sumário do modelo implementado em Keras.

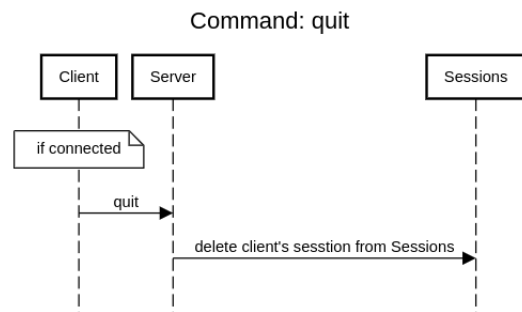


Figura 11. Sumário do modelo implementado em Keras.


```
python3 Client.py
python3 Server.py 47x32
--isabelle@isabelle-Inspiron-5448 ~/Graduacao/Lourenco-cecilia-CES-35/lab2/Servidor 'master'
$ python3 Server.py
starting up on localhost port 2121
Connection attempt from ('127.0.0.1', 46354)
[bell] connected!
[bell] disconnected!

python3 Client.py
python3 Client.py 47x32
--isabelle@isabelle-Inspiron-5448 ~/Graduacao/Lourenco-cecilia-CES-35/lab2/Cliente 'master'
$ python3 Client.py
>>> open localhost:2121
user: bell
password: 123
Connected!
>>> close
>>>
```

Figura 20. Sumário do modelo implementado em Keras.

```
python3 Client.py
python3 Server.py 47x32
--isabelle@isabelle-Inspiron-5448 ~/Graduacao/Lourenco-cecilia-CES-35/lab2/Servidor 'master'
$ python3 Server.py
starting up on localhost port 2121
Connection attempt from ('127.0.0.1', 46318)
[bell] connected!

python3 Client.py
python3 Client.py 47x32
--isabelle@isabelle-Inspiron-5448 ~/Graduacao/Lourenco-cecilia-CES-35/lab2/Cliente 'master'
$ python3 Client.py
>>> open localhost:2121
user: bell
password: 123
Connected!
>>>
```

Figura 21. Sumário do modelo implementado em Keras.

B. Escolha de Ação usando Rede Neural

Já essa etapa se tratou da implementação do método `act()` também da classe `DQNAgent` de `dqn_agent.py`. Nesse método, era escolhido e retornado uma ação de acordo com a política ϵ -greedy.

Essa implementação foi feita de forma bastante análoga à maneira do laboratório 12 [3]. Assim, gerou-se um número aleatório entre 0 e 1 e, caso esse valor aleatório seja menor que ϵ , então uma ação aleatória é escolhida; caso contrário, é escolhida a ação gulosa, através do retorno do índice do máximo elemento do array `model.predict(state)[0]`.

C. Reward Engineering

Nesse momento, foi implementado o método `reward_engineering_mountain_car()` de `utils.py`, script também fornecido no código base do laboratório. Nesse método, eram calculadas e retornadas as recompensas intermediárias "artificiais", chamadas `reward engineering`, a fim de tornar o treino mais rápido no ambiente do Mountain Car.

Essa implementação foi feita conforme as equações apresentadas na seção 4.3 do roteiro do laboratório [1], ou seja, assim como apresentado no pseudo-código em Python a seguir.

```
reward = reward + (position - start) *
              (position - start) + velocity * velocity

aux = 0
if next_position >= 0.5:
    aux = 1
```

```
isabelle@isabelle-Inspiron-5448: ~/Graduacao/Lourenco-cecilia-CES-35/lab2/Cliente
$ python3 Server.py 47x32
starting up on localhost port 2121
Connection attempt from ('127.0.0.1', 46340)
[bell] connected!
[bell] disconnected!

isabelle@isabelle-Inspiron-5448: ~/Graduacao/Lourenco-cecilia-CES-35/lab2/Cliente
$ python3 Client.py
>>> open localhost:2121
user: bell
password: 123
Connected!
>>> quit
```

Figura 22. Sumário do modelo implementado em Keras.

```
reward += 50 * aux
```

Os valores de `position`, `start`, `velocity` e `next_position` também eram fornecidos no roteiro [1], e bastava substituí-los no pseudo-código acima.

D. Treinamento usando DQN

Bastava treinar o modelo implementado, executando o script `train_dqn.py`, também do código base, e observar os resultados e os gráficos obtidos.

E. Avaliação da Política

Bastava aplicar o modelo implementado no ambiente do Mountain Car, executando o script `evaluate_dqn.py`, também do código base, e observar a simulação, os resultados e os gráficos obtidos.

IV. RESULTADOS E CONCLUSÕES

O summary do modelo implementado em `make_model()` foi apresentado na Figura 23, e condiz com os requisitos pedidos na Tabela 3 do roteiro do laboratório [1].

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 24)	72
dense_2 (Dense)	(None, 24)	600
dense_3 (Dense)	(None, 3)	75
Total params: 747		
Trainable params: 747		
Non-trainable params: 0		

Figura 23. Sumário do modelo implementado em Keras.

Já a Figura 24 representa as recompensas acumulativas advindas do treinamento do modelo em 300 episódios. Esse resultado dependem diretamente da correta implementação e funcionamento dos métodos `make_model()` e `act()`.

Pode-se dizer que esse gráfico condiz com o esperado, uma vez que é possível notar inicialmente recompensas pequenas para os primeiros episódios e, mais ou menos a partir do episódio 80, tornou-se frequente recompensas com valores elevados, chegando a valores próximos de 40, indicando um aprendizado significativamente correto.

Já a aplicação do modelo implementado no ambiente do Mountain Car gerou as Figuras de 25 a 27.

A partir da Figura 25, pode-se concluir que a implementação e treino chegaram em resultados satisfatórios, uma vez que grande parte das recompensas acumuladas foi alta, próximas de 40, chegando no final de 30 episódios a uma média de 27,8, conforme apresentado na Figura 26.

Por fim, acerca da Figura 27, pode-se observar que:

- Para velocidades para direita, quase unanimemente a decisão do carro é continuar para direita. Exclui-se disso as situações de posição muito à esquerda e velocidades altas, na qual é decidido fazer nada, e de velocidades para direita muito baixas, na qual pouquíssimas vezes o

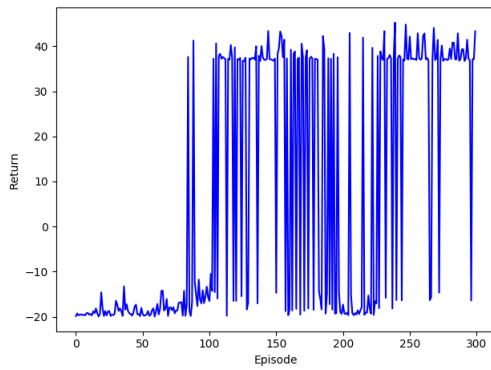


Figura 24. Recompensa acumulativa com o passar dos episódios, no treinamento do modelo para 300 episódios.

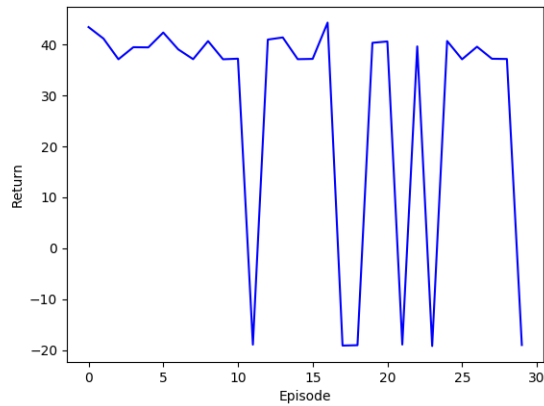


Figura 25. Representação em cores da tabela de action-value calculada, para algoritmo de Sarsa.

```
episode: 1/30, time: 107, score: 43.4619, epsilon: 0.0
episode: 2/30, time: 94, score: 41.2016, epsilon: 0.0
episode: 3/30, time: 159, score: 37.1377, epsilon: 0.0
episode: 4/30, time: 85, score: 39.5076, epsilon: 0.0
episode: 5/30, time: 84, score: 39.4702, epsilon: 0.0
episode: 6/30, time: 101, score: 42.3907, epsilon: 0.0
episode: 7/30, time: 83, score: 39.0879, epsilon: 0.0
episode: 8/30, time: 160, score: 37.1565, epsilon: 0.0
episode: 9/30, time: 91, score: 40.7103, epsilon: 0.0
episode: 10/30, time: 159, score: 37.138, epsilon: 0.0
episode: 11/30, time: 167, score: 37.2522, epsilon: 0.0
episode: 12/30, time: 200, score: -18.9488, epsilon: 0.0
episode: 13/30, time: 92, score: 41.0069, epsilon: 0.0
episode: 14/30, time: 95, score: 41.4258, epsilon: 0.0
episode: 15/30, time: 160, score: 37.1562, epsilon: 0.0
episode: 16/30, time: 163, score: 37.2112, epsilon: 0.0
episode: 17/30, time: 113, score: 44.3414, epsilon: 0.0
episode: 18/30, time: 200, score: -19.131, epsilon: 0.0
episode: 19/30, time: 200, score: -19.0591, epsilon: 0.0
episode: 20/30, time: 89, score: 40.3713, epsilon: 0.0
episode: 21/30, time: 90, score: 40.6353, epsilon: 0.0
episode: 22/30, time: 200, score: -18.9305, epsilon: 0.0
episode: 23/30, time: 86, score: 39.6682, epsilon: 0.0
episode: 24/30, time: 200, score: -19.2182, epsilon: 0.0
episode: 25/30, time: 91, score: 40.7233, epsilon: 0.0
episode: 26/30, time: 160, score: 37.1333, epsilon: 0.0
episode: 27/30, time: 85, score: 39.6013, epsilon: 0.0
episode: 28/30, time: 165, score: 37.2369, epsilon: 0.0
episode: 29/30, time: 161, score: 37.1958, epsilon: 0.0
episode: 30/30, time: 200, score: -19.0234, epsilon: 0.0
Mean return: 27.79701181215042
```

Figura 26. Recompensa acumulada em função das iterações, para algoritmo de Sarsa.

carro decide ir para esquerda, talvez já se enquadrando nas intenções descritas no próximo item.

- Para velocidades para esquerda, as decisões do carro diferem bastante da posição na qual ele se encontra. Para posições mais a esquerda, o carro decide continuar indo para esquerda, talvez para pegar impulso da subida e, quando por fim chegar em posições mais a esquerda (consequentemente mais altas) possíveis, decidir ir com velocidade para direita. Já para posições relativamente próximas da posição objetivo, aparecem também decisões de não fazer nada, indicando que o carro irá mais para esquerda e cairá na situação anteriormente descrita, na qual ele decidirá continuar indo para esquerda e pegará o impulso da elevação.

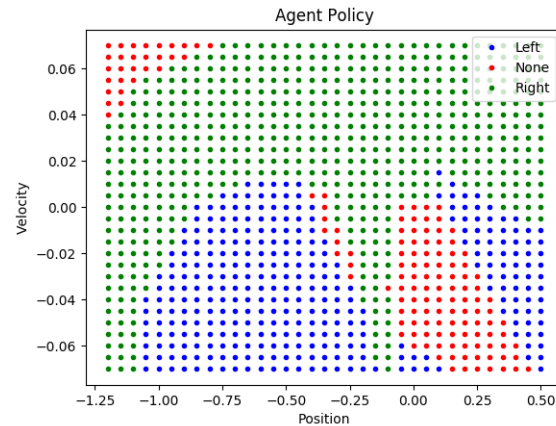


Figura 27. Representação em cores da tabela de greedy-policy calculada, para algoritmo de Sarsa.

Como as decisões aprendidas e tomadas pelo carro fizeram sentido e puderam ser interpretadas satisfatoriamente, pode-se dizer que a proposta do laboratório foi corretamente implementada e se mostrou satisfatória em resolver o problema proposto.

REFERÊNCIAS

- [1] M. Maximo, “Roteiro: Laboratório 12 - Deep Q-Learning”. Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.
- [2] M. Maximo, “Roteiro: Laboratório 8 - Imitation Learning com Keras”. Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.
- [3] M. Maximo, “Roteiro: Laboratório 12 - Aprendizado por Reforço Livre de Modelo”. Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.