

# Relatório da Atividade 1:

## Logical Clock

Isabelle Ferreira de Oliveira  
CES-27 - Engenharia da Computação 2020  
Instituto Tecnológico de Aeronáutica (ITA)  
São José dos Campos, Brasil  
isabelle.ferreira3000@gmail.com

**Resumo**—Esse relatório documenta a implementação da simulação de processos rodando e trocando seus relógios lógicos entre si (Logical Clock definido por Lamport). Esses relógios foram tanto escalares quanto vetoriais.

**Index Terms**—Relógio lógico, Relógio lógico escalar, Relógio lógico vetorial, algoritmo de Lamport

### I. IMPLEMENTAÇÃO

#### A. Tarefa 1: Relógio Lógico Escalar

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].

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))
```

---

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 II (Resultados e Conclusões), e a linha de retorno da função.

#### B. Tarefa 2: Relógio Lógico Vetorial

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  $\epsilon$ -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  $\epsilon$ , 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

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.

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 1. Sumário do modelo implementado em Keras.

## II. RESULTADOS E CONCLUSÕES

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

Já a Figura 2 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.

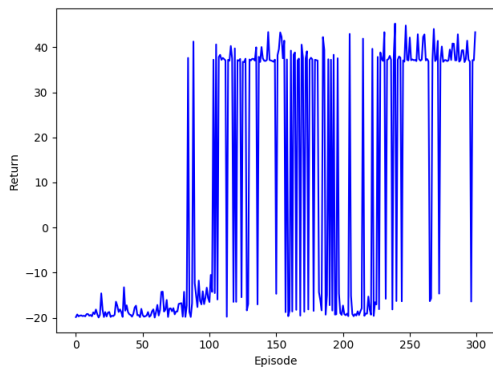


Figura 2. Recompensa acumulativa com o passar dos episódios, no treinamento do modelo para 300 episódios.

Já a aplicação do modelo implementado no ambiente do Mountain Car gerou as Figuras de 3 a 5.

A partir da Figura 3, 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 4.

Por fim, acerca da Figura 5, 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

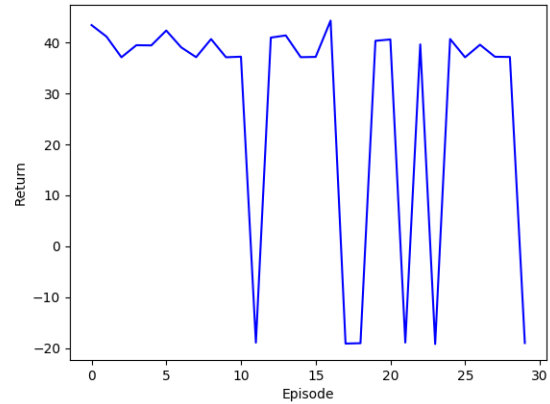


Figura 3. Representação em cores da tabela de action-value calculada, para algoritmo de Sarsa.

```
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: 27.79701181215842
```

Figura 4. 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.

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 im-

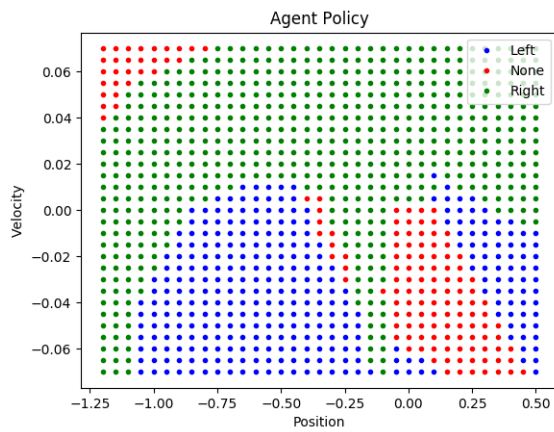


Figura 5. Representação em cores da tabela de greedy-policy calculada, para algoritmo de Sarsa.

plementada 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.