

INSTITUTO FEDERAL DO ESPÍRITO SANTO  
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA – PPCOMP

**BRUNA RUPP RUELA**

**RELATÓRIO DE RESULTADOS - IMPLEMENTAÇÃO DE APRENDIZADO POR  
REFORÇO PARA OS PROBLEMAS MOUNTAIN CAR E CAR RACING**

Serra  
2024

BRUNA RUPP RUELA

**RELATÓRIO DE RESULTADOS - IMPLEMENTAÇÃO DE APRENDIZADO POR  
REFORÇO PARA OS PROBLEMAS MOUNTAIN CAR E CAR RACING**

Trabalho apresentado ao Programa de Pós-Graduação  
em Computação Aplicada – PPCOMP do Instituto  
Federal do Espírito Santo, como requisito para apro-  
vação da Disciplina de Inteligencia Artificial.

Orientador: Prof. Dr. Sergio Nery Simões

Serra  
2024

## LISTA DE FIGURAS

Figura 1 – Aprendizado por Reforço . . . . .	4
Figura 2 – ambiente do Mountain Car . . . . .	5
Figura 3 – Ambiente do Car Racing . . . . .	6
Figura 4 – Loop do Agente no Ambiente . . . . .	7
Figura 5 – Usando Stable Baseline3 para treinar, salvar, carregar e inferir uma ação de uma politica . . . . .	8
Figura 6 – Trecho de código 1 - Jupyter Notebook . . . . .	10
Figura 7 – Trecho de código 2 - Jupyter Notebook . . . . .	11
Figura 8 – Trecho de código 3 - Jupyter Notebook . . . . .	11
Figura 9 – Treinamento Modelo - Mountain Car . . . . .	12
Figura 10 – Trecho de código 1 - Car Racing . . . . .	13
Figura 11 – Trecho de código 2 - Mountain Car . . . . .	14

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO . . . . .</b>	<b>3</b>
1.1	O PROBLEMA . . . . .	4
1.1.1	Problema 1 - Aplicação de Aprendizado por Reforço no ambiente Mountain Car . . . . .	4
1.1.2	Problema 2 - Aplicação de Aprendizado por Reforço no ambiente Car Racing . . . . .	5
1.2	OBJETIVO GERAL . . . . .	6
1.3	A PROPOSTA . . . . .	6
1.3.1	Gymnasium . . . . .	7
1.3.2	Stable Baselines . . . . .	8
1.4	MATERIAIS E MÉTODOS . . . . .	8
1.5	IMPLEMENTAÇÃO DO PROBLEMA MOUNTAIN CAR . . . . .	9
1.5.1	Hiperparâmetros e Configuração do Modelo . . . . .	9
1.5.2	Processo de Treinamento . . . . .	10
1.5.3	Avaliação . . . . .	11
1.5.4	Gravação de Vídeo . . . . .	12
1.6	IMPLEMENTAÇÃO CAR RACING . . . . .	12
1.6.1	Hiperparâmetros e Configuração do Modelo . . . . .	12
1.6.2	Processo de Treinamento . . . . .	13
1.6.3	Avaliação . . . . .	13
1.6.4	Gravação de Vídeo . . . . .	14
1.7	DISCUSSÕES . . . . .	14
1.8	CONCLUSÃO . . . . .	15
	REFERÊNCIAS . . . . .	16

## 1 INTRODUÇÃO

O Aprendizado por Reforço (Reinforcement Learning - RL) é uma área de estudo no campo da inteligência artificial (IA) e do aprendizado de máquina que se concentra em como os agentes podem aprender a tomar decisões em um ambiente, de modo a maximizar alguma noção de recompensa cumulativa. Diferente de outros métodos de aprendizado de máquina, o aprendizado por reforço não depende de um conjunto de dados pré-existent; em vez disso, os agentes aprendem com a interação direta com o ambiente, recebendo feedback na forma de recompensas ou penalidades (SUTTON; BARTO, 2018).

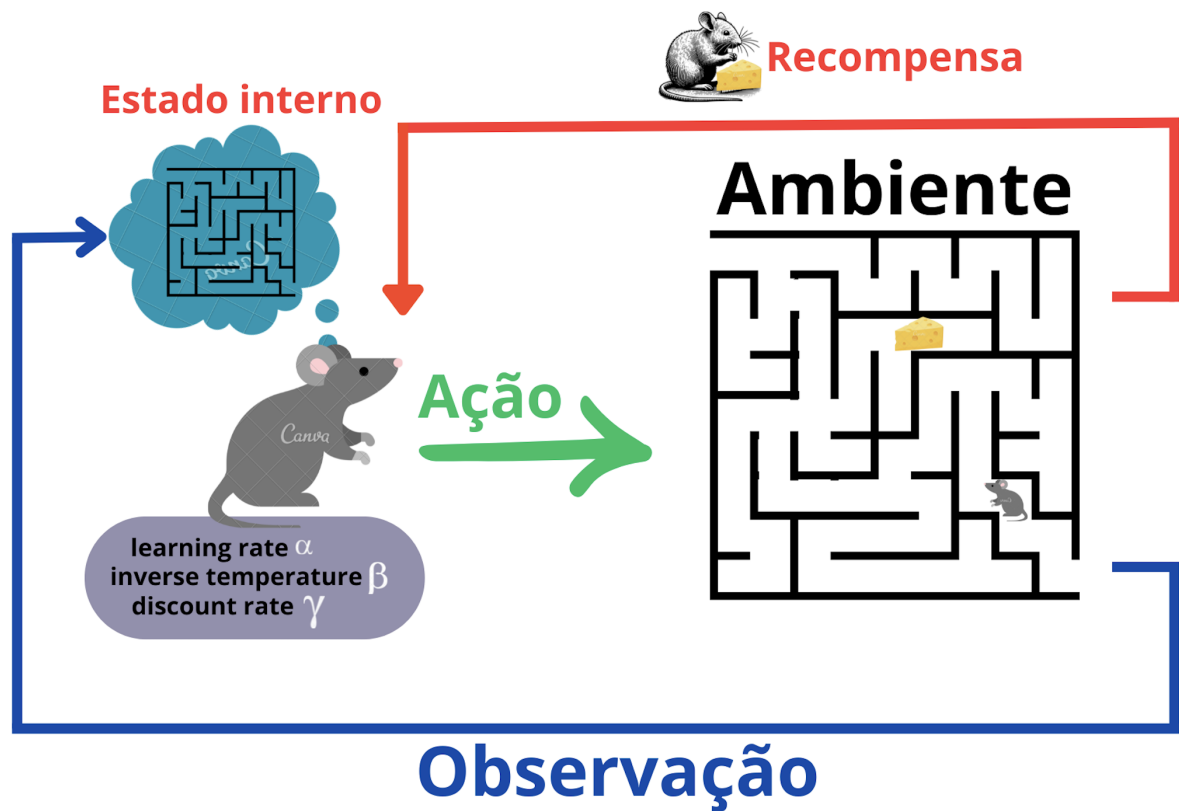
Essa abordagem é inspirada na forma como os seres humanos e animais aprendem novas habilidades e comportamentos, através de tentativa e erro e do ajuste contínuo de suas ações com base nas consequências que experimentam. O RL é particularmente útil em situações onde a solução ideal não é conhecida e deve ser descoberta através da experiência. Aplicações dessa técnica podem ser encontradas em diversas áreas, como jogos, robótica, finanças, controle de tráfego, e sistemas de recomendação.

Um agente de aprendizado por reforço toma ações em um ambiente para alcançar um objetivo, aprendendo uma política, que é um mapa das situações observadas para as ações a serem tomadas. Ao longo do tempo, o agente utiliza um processo de exploração, balanceando entre tentar novas ações para descobrir suas recompensas potenciais e escolher ações conhecidas que já proporcionaram boas recompensas no passado. As principais ferramentas matemáticas utilizadas no RL incluem processos de decisão de Markov (MDPs) e algoritmos como Q-Learning, SARSA, e métodos baseados em política, como o método REINFORCE.

O rato na Figura 1 representa o Agente. O labirinto o ambiente e o queijo a recompensa. Os caminhos do labirinto são análogos aos Estados no aprendizado por reforço. Então, a rato tem que decidir a quais caminhos percorrer (ou seja, calcular o valor de cada caminho). Este será o trabalho da nossa Função de Valor. Então, toda vez que ele anda de um caminho para outro, ele ganha uma Recompensa e os métodos que ele usa para farejar o queijo dentro do tempo são a nossa Política.

Isso é feito pelo nosso sistema de dopamina no cérebro, que cuida do comportamento motivado por recompensa. A maioria dos tipos de recompensas aumenta o nível de dopamina no cérebro. Por exemplo, quando você come chocolate, isso aumenta a atividade neuronal da dopamina. Então você se sente recompensado e satisfeito, daí o efeito. O aprendizado por reforço é um campo dinâmico e promissor que continua a evoluir, com avanços contínuos na teoria e em suas aplicações práticas, impulsionando o desenvolvimento de sistemas mais inteligentes e adaptativos.

Figura 1 – Aprendizado por Reforço



Fonte: Elaborado pela autora

## 1.1 O PROBLEMA

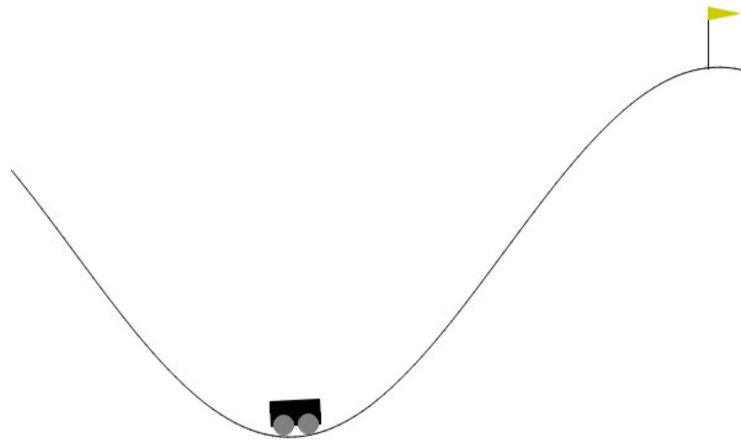
### 1.1.1 Problema 1 - Aplicação de Aprendizado por Reforço no ambiente Mountain Car

O problema “Mountain Car” é um ambiente de aprendizado por reforço onde um carro deve superar a força da gravidade para alcançar o topo de uma montanha. O objetivo do agente é fazer com que o carro suba a montanha dentro de um limite de tempo, usando uma quantidade limitada de força disponível.

O ambiente é representado por uma pista de corrida bidimensional, que consiste em uma colina com uma forma peculiar. A colina possui um vale estreito e achatado, onde o carro começa sua jornada, e duas montanhas de cada lado do vale. O carro é inicialmente colocado no vale e deve acelerar ou frear para ganhar impulso e superar a gravidade, tentando atingir o topo de uma das montanhas.

A física do problema é definida por um conjunto de regras e leis que governam o movimento do carro. O carro é modelado como um objeto com massa e está sujeito a duas forças principais: a força gravitacional, que puxa o carro para baixo, e a força do motor, que pode acelerar ou frear o carro. No entanto, o carro possui uma limitação: sua força de motor

Figura 2 – ambiente do Mountain Car



Fonte: AVA - Especificação do trabalho

não é suficiente para diretamente vencer a gravidade e subir a montanha íngreme. Para superar esse desafio, o agente precisa usar estratégias inteligentes. Por exemplo, o agente pode acelerar o carro quando ele está descendo uma montanha para ganhar velocidade e momentum, o que pode ajudar a empurrá-lo para cima da próxima montanha. No entanto, o agente também precisa ter cuidado para não acelerar demais e perder o controle do carro ou ultrapassar a montanha.

O ambiente fornece observações ao agente, que incluem a posição do carro e sua velocidade atual. Além disso, o ambiente define regras de terminação, como um limite máximo de tempo ou uma posição específica que o carro deve atingir para considerar a tarefa concluída com sucesso. O agente recebe recompensas positivas quando o carro se move em direção ao topo da montanha e recompensas negativas quando o carro se move para trás ou quando o tempo limite é atingido.

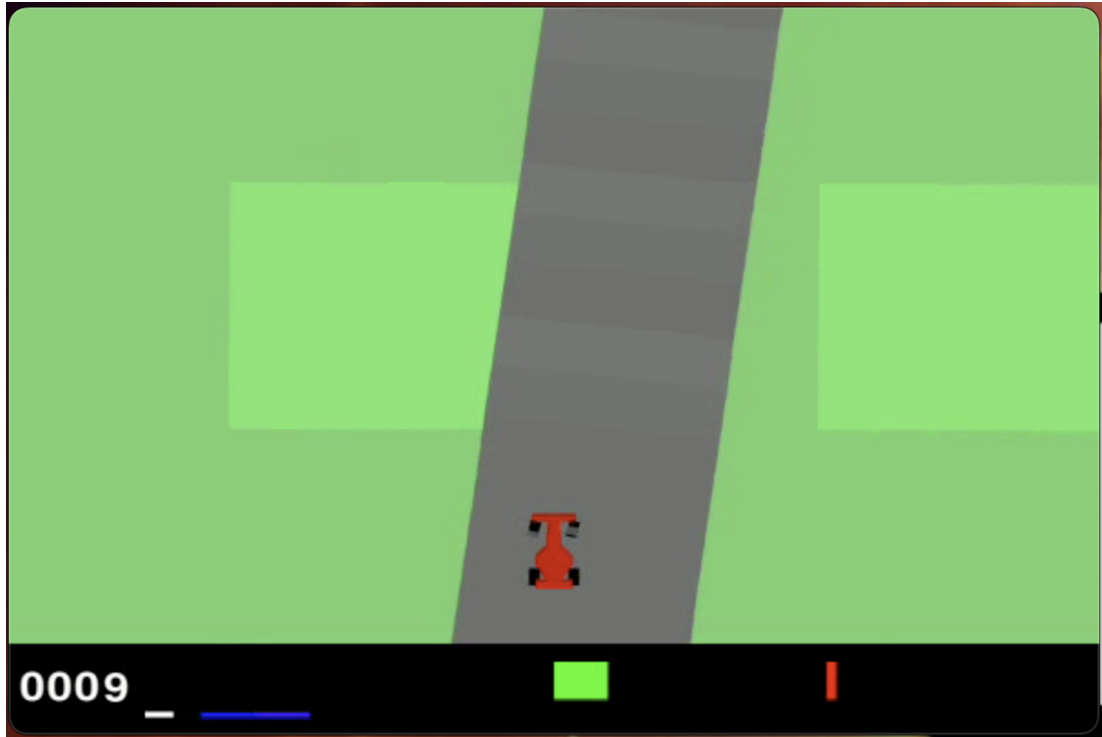
Resolver o problema “Mountain Car” requer uma estratégia inteligente para equilibrar o uso adequado da força do motor, o gerenciamento de momentum e a exploração do ambiente. Os algoritmos de aprendizado por reforço são utilizados para treinar um agente capaz de aprender essas estratégias através da interação com o ambiente, permitindo que o carro supere os desafios físicos e alcance o topo da montanha dentro do limite de tempo estabelecido. Além disso, é necessário descobrir a política apropriada para solucionar o problema, já que nem todas podem ser aplicadas.

### 1.1.2 Problema 2 - Aplicação de Aprendizado por Reforço no ambiente Car Racing

Similar ao problema 1, mas com um o ambiente é 2D, o objetivo desta implementação é aplicar técnicas de aprendizado por reforço para treinar um agente capaz de resolver o ambiente “Car Racing” disponível na biblioteca GYM (Oleg Klimov, 2023). Esse ambiente

simula uma pista de corrida onde um carro deve aprender a navegar pela pista e completar voltas no menor tempo possível, evitando colisões com obstáculos.

Figura 3 – Ambiente do Car Racing



Fonte: Do modelo da autora

## 1.2 OBJETIVO GERAL

Este trabalho tem como objetivo abordar o tema do aprendizado por reforço aplicado ao treinamento de agentes para resolver desafios nos ambientes “Mountain Car” e “Car Racing” disponíveis na biblioteca GYMNASIUM (que contém diversos ambientes didáticos). Por meio da utilização da biblioteca Stable Baselines, exploraremos a aplicação de diversas políticas para solucionar problemas por aprendizado por reforço em dois ambientes diferentes.

## 1.3 A PROPOSTA

A proposta desse trabalho é descrever o processo de implementação e análise de cada problema utilizando a biblioteca *Stable Baselines* e as APIs do projeto *Gymnasium*. O treinamento deve ser executado por um número suficiente de episódios para permitir que o agente aprenda a superar os desafios do ambiente. Será utilizado nos treinamentos os número de iterações de 50K, 100K, 200K e 400K.

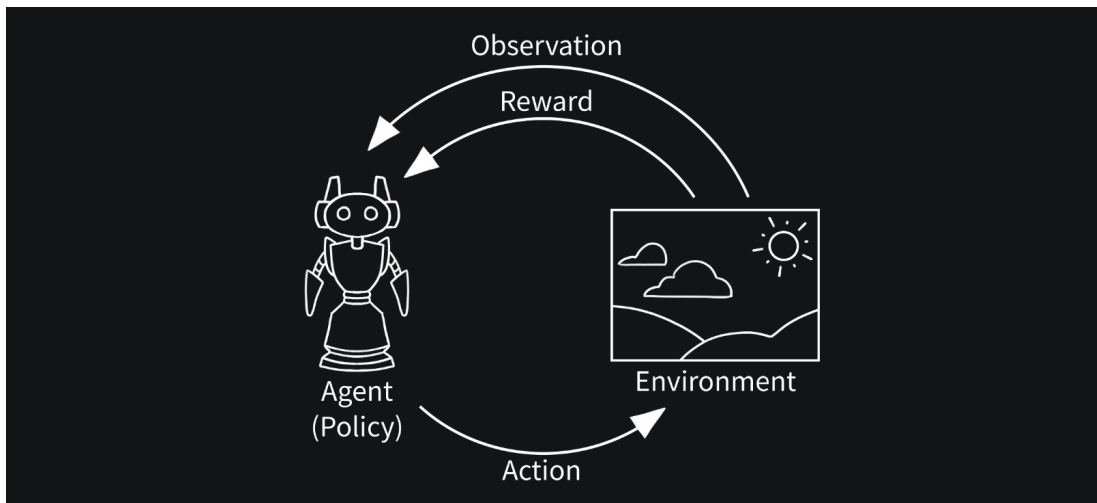


### 1.3.1 Gymnasium

Gymnasium é um projeto que fornece uma API para todos os ambientes de aprendizado por reforço de agente único e inclui implementações de ambientes comuns: cartpole, pendulum, mountain-car, mujoco, atari e muito mais (MOORE, 1990).

A API contém quatro funções principais: make, reset, step e render. No núcleo do Gymnasium está Env, uma classe python de alto nível que representa um processo de decisão de Markov (MDP) da teoria de aprendizado por reforço. Dentro do gymnasium, os ambientes (MDPs) são implementados como Envclasses, junto com Wrappers, que fornecem utilitários úteis e podem alterar os resultados passados ao usuário. A Figura 4 ilustra de forma simplificada como Gymnasium implementa o aprendizado por reforço.

Figura 4 – Loop do Agente no Ambiente



Fonte: Towers et al. (2023)

Para utilizar o projeto Gym, são declaradas as especificações dos ambientes que são essenciais para o correto funcionamento dos algoritmos de aprendizado por reforço *Mountain Car* e o *Car Racing* conforme especificado na Tabela 1.

Informações	MountainCar-v0	CarRacing-v2
Action Space	Discrete(3)	Box([-1. 0. 0.], 1.0, (3,), float32)
Observation Space	Box([-1.2 -0.07], [0.6 0.07], (2,), float32)	Box(0, 255, (96, 96, 3), uint8)
Import	<code>gymnasium.make("MountainCar-v0")</code>	<code>gymnasium.make("CarRacing-v2")</code>

Tabela 1 – Especificações do ambiente do *Mountain Car* e *Car Racing*

Uma diferença importante entre esses ambientes é a natureza do espaço de ações e observações. No *Mountain Car*, o espaço de ações é discreto, com apenas três ações

possíveis, e o espaço de observações é um vetor contínuo de duas dimensões representando a posição e a velocidade do carro. Em contraste, o *Car Racing* possui um espaço de ações contínuo, permitindo maior precisão no controle, e um espaço de observações muito mais complexo, que é uma imagem em cores de 96x96 pixels. Esta diferença reflete a complexidade e a natureza visualmente rica do ambiente de corrida em comparação com o ambiente mais simples de controle do carro de montanha.

### 1.3.2 Stable Baselines

*Stable-Baselines3* é uma biblioteca poderosa e versátil para a implementação de algoritmos de Aprendizado por Reforço (RL). Construída sobre o *PyTorch*, ela oferece uma coleção abrangente de algoritmos de RL, como A2C, PPO, DDPG, TD3 e SAC, de forma modular e fácil de usar. *Stable-Baselines3* se destaca por sua documentação clara, integração com a OpenAI Gym e suporte ativo da comunidade, o que facilita o desenvolvimento, o teste e a implementação de agentes de RL em diferentes ambientes (Stable-Baselines3 Team, 2024).

Usar a *Stable-Baselines3* para resolver o problema do Mountain Car permite explorar o poder dos algoritmos de RL em um ambiente desafiador e bem conhecido, proporcionando uma sólida base para aplicações mais complexas no futuro. A Figura 5 mostra que para treinar agentes usando a biblioteca basta apenas alguns linhas de código, onde agente também pode ser consultado para ações.

Figura 5 – Usando Stable Baseline3 para treinar, salvar, carregar e inferir uma ação de uma política

```
import gym
from stable_baselines3 import SAC
# Train an agent using Soft Actor-Critic on Pendulum-v0
env = gym.make("Pendulum-v0")
model = SAC("MlpPolicy", env).learn(total_timesteps=20000)
# Save the model
model.save("sac_pendulum")
# Load the trained model
model = SAC.load("sac_pendulum")
# Start a new episode
obs = env.reset()
# What action to take in state `obs`?
action, _ = model.predict(obs, deterministic=True)
```

Fonte: (RAFFIN et al., 2021)

## 1.4 MATERIAIS E MÉTODOS

Para a implementação dos algoritmos de RL para o problema do Mountain Car e Car Racing foi considerada as especificações da documentação do Gymnasium para o ambiente e a

biblioteca Stable baseline. Foi utilizado o Jupiter Notebook para processamento do modelo com um computador Macbook air com 8GB de memória Ram e chip de processamento Apple M2. O resultado e código completo pode ser conferido nesse link do Github.

## 1.5 IMPLEMENTAÇÃO DO PROBLEMA MOUNTAIN CAR

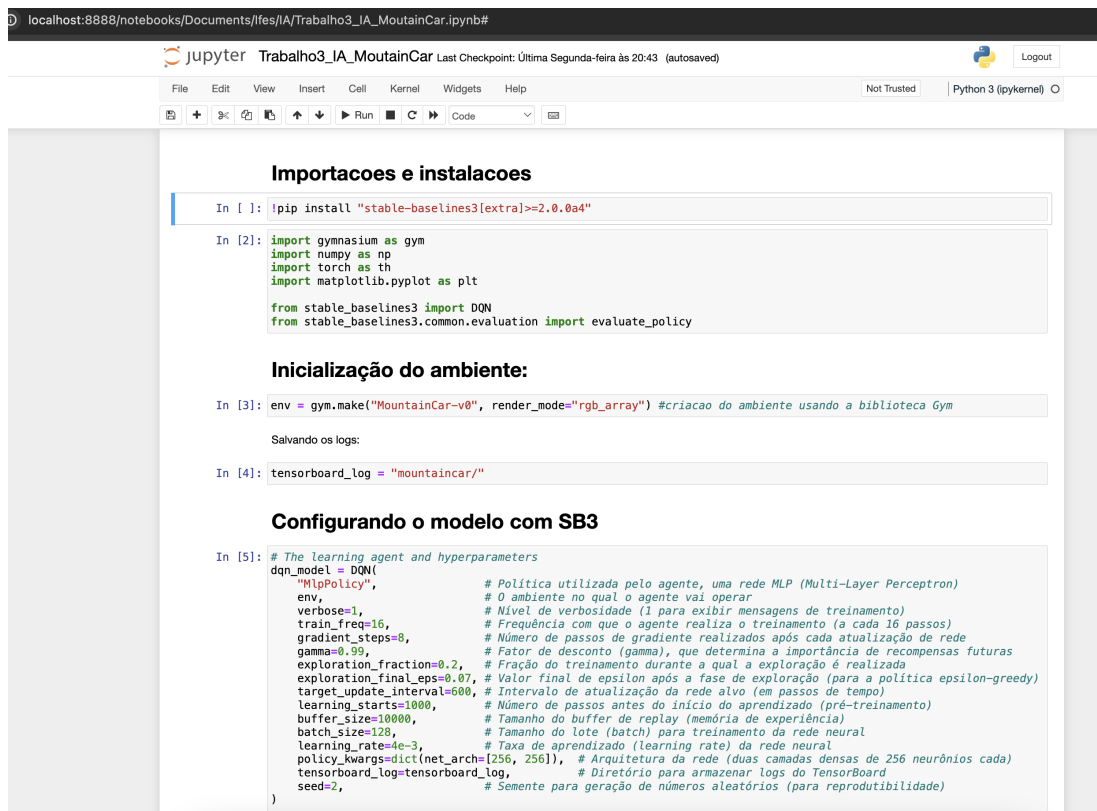
O problema do Mountain Car é um clássico desafio de controle em aprendizado por reforço, onde um carro deve ser conduzido para o topo de uma montanha, mas o motor do carro não é forte o suficiente para alcançar o topo diretamente. O carro deve aprender a oscilar para trás e para frente para ganhar impulso suficiente para superar a colina.

### 1.5.1 Hiperparâmetros e Configuração do Modelo

O DQN é uma abordagem baseada em redes neurais para aprendizado por reforço que utiliza uma rede neural profunda para aproximar a função de valor  $Q$ , que estima a qualidade de uma ação em um determinado estado. A rede neural recebe como entrada o estado do ambiente e produz uma estimativa do valor  $Q$  para cada ação possível.

Na Figura 6 a Política MLP (Multi-Layer Perceptron) é utilizada para mapear estados a ações através de uma rede neural. A Frequência de Treinamento (train freq=16) atualiza a rede neural a cada 16 passos. Os Passos de Gradiente (gradient steps=8) Realiza 8 atualizações da rede neural após cada ciclo de treinamento. O Fator de Desconto (gamma=0.99) desconta as recompensas futuras para dar mais importância às recompensas imediatas. A Exploração (exploration fraction=0.2) considera 20% do tempo dedicado à exploração de novas ações. O Buffer de Replay (buffer size=10000) armazena as experiências passadas para amostras aleatórias durante o treinamento. O Lote de Treinamento (batch size=128) é o tamanho do lote de experiências utilizado para treinar a rede neural. A Taxa de Aprendizado (learning rate=4e-3) controla a velocidade de atualização dos pesos da rede neural e por fim a arquitetura da Rede (net arch=[256, 256]) determina duas camadas ocultas com 256 neurônios cada.

Figura 6 – Trecho de código 1 - Jupyter Notebook



The screenshot shows a Jupyter Notebook titled 'Trabalho3\_IA\_MountainCar'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and saving. The code is organized into sections with bold headings.

**Importacoes e instalacoes**

```
In [1]: !pip install "stable-baselines3[extra]>=2.0.0a4"
```

```
In [2]: import gymnasium as gym
import numpy as np
import torch as th
import matplotlib.pyplot as plt

from stable_baselines3 import DQN
from stable_baselines3.common.evaluation import evaluate_policy
```

**Inicialização do ambiente:**

```
In [3]: env = gym.make("MountainCar-v0", render_mode="rgb_array") #criacao do ambiente usando a biblioteca Gym
```

Salvando os logs:

```
In [4]: tensorboard_log = "mountaincar/"
```

**Configurando o modelo com SB3**

```
In [5]: # The learning agent and hyperparameters
dqn_model = DQN(
    "MlpPolicy",          # Política utilizada pelo agente, uma rede MLP (Multi-Layer Perceptron)
    env,                  # O ambiente no qual o agente vai operar
    verbose=1,            # Nível de verbosidade (1 para exibir mensagens de treinamento)
    train_freq=16,        # Frequência com que o agente realiza o treinamento (a cada 16 passos)
    gradient_steps=8,     # Número de passos de gradiente realizados após cada atualização de rede
    gamma=0.99,           # Fator de desconto (gamma), que determina a importância de recompensas futuras
    exploration_fraction=0.2, # Fração do treinamento durante a qual a exploração é realizada
    exploration_final_eps=0.07, # Valor final de epsilon após a fase de exploração (para a política epsilon-greedy)
    target_update_interval=600, # Intervalo de atualização da rede alvo (em passos de tempo)
    learning_starts=1000,  # Número de passos antes do início do aprendizado (pré-treinamento)
    buffer_size=10000,     # Tamanho do buffer de replay (memória de experiência)
    batch_size=128,        # Tamanho do lote (batch) para treinamento da rede neural
    learning_rate=4e-3,    # Taxa de aprendizado (learning rate) da rede neural
    policy_kwargs=dict(net_arch=[256, 256]), # Arquitetura da rede (duas camadas densas de 256 neurônios cada)
    tensorboard_log=tensorboard_log, # Diretório para armazenar logs do TensorBoard
    seed=2,                # Semente para geração de números aleatórios (para reprodutibilidade)
)
```

Fonte: Elaborado pela Autora

## 1.5.2 Processo de Treinamento

No treinamento inicial, o agente é treinado por 120.000 passos de tempo, durante os quais ele interage com o ambiente, armazena experiências no buffer de replay e atualiza a rede neural com base em amostras do buffer. Pode ser observado na Figura 7 a saída das informações de progresso sendo exibidas a cada 10 passos. Essa saída também permite visualizar se o agente está melhorando ao longo do tempo e ajusta o treinamento conforme necessário.

Figura 7 – Trecho de código 2 - Jupyter Notebook

```

>print(f"mean_reward:{mean_reward:.2f} +/- {std_reward:.2f}")

mean_reward:-200.00 +/- 0.00

In [15]: # Optional: Monitor training in tensorboard
# %load_ext tensorboard
# %tensorboard --logdir $tensorboard_log

Treinamento do agente DQN por 120.000 passos de tempo,
é carregado na saída as informações de progresso sendo exibidas a cada 10 passos. Permite visualizar se o agente está melhorando ao longo do tempo e
ajusta o treinamento conforme necessário.

In [7]: dqn_model.learn(int(1.2e5), log_interval=10) # int(1.2e5) Converte o valor 1.2e5 (120.000) para um número inteiro



| ep/             | rew_mean | exp_rate | time/ |
|-----------------|----------|----------|-------|
| episodes        | 10       |          |       |
| fps             | 3592     |          |       |
| time_elapsed    | 0        |          |       |
| total_timesteps | 2000     |          |       |
| train/          |          |          |       |
| learning_rate   | 0.004    |          |       |
| loss            | 1.97e-05 |          |       |
| n_updates       | 496      |          |       |



| rollout/         | ep_len_mean | ep_reward_mean | exploration_rate | time/ |
|------------------|-------------|----------------|------------------|-------|
| ep_len_mean      | 200         |                |                  |       |
| ep_reward_mean   | -200        |                |                  |       |
| exploration_rate | 0.845       |                |                  |       |
| time/            |             |                |                  |       |
| episodes         | 20          |                |                  |       |



In [8]: mean_reward, std_reward = evaluate_policy(dqn_model, dqn_model.get_env(), deterministic=True, n_eval_episodes=20)

print(f"mean_reward:{mean_reward:.2f} +/- {std_reward:.2f}")

mean_reward:-98.55 +/- 9.04

```

Fonte: Elaborado pela Autora

### 1.5.3 Avaliação

Periodicamente, o modelo é avaliado utilizando uma política determinística para verificar o desempenho do agente, calculando a recompensa média e o desvio padrão sobre vários episódios. Para 120.000 passos o modelo calculou a recompensa média de -200 e desvio padrão 0 (mean reward:-200.00 +/- 0.00). Recompensa Média é um indicador de quão bem o agente está performando no ambiente. Uma recompensa média mais alta sugere que o agente está aprendendo a resolver o problema de maneira eficaz. O Desvio Padrão é a medida da variabilidade nas recompensas obtidas. Um desvio padrão baixo como gerado no primeiro treino indica que o agente está se comportando de maneira consistente.

Figura 8 – Trecho de código 3 - Jupyter Notebook

```

Avaliação da política:
A função evaluate_policy avalia o desempenho do modelo DQN treinado.

In [6]: mean_reward, std_reward = evaluate_policy(
    dqn_model,          # O modelo DQN que foi treinado e será avaliado
    dqn_model.get_env(), # O ambiente no qual o modelo será avaliado, obtido a partir do modelo treinado
    deterministic=True,  # Se verdadeiro, o agente usará ações determinísticas (sem explorar) durante a avaliação
    n_eval_episodes=20,  # Número de episódios de avaliação para calcular a média e o desvio padrão da recompensa
)

#Resultado: A função evaluate_policy retorna a recompensa média (mean_reward)
#e o desvio padrão da recompensa (std_reward) após a avaliação do agente em 20 episódios.

print(f"mean_reward:{mean_reward:.2f} +/- {std_reward:.2f}")

mean_reward:-200.00 +/- 0.00

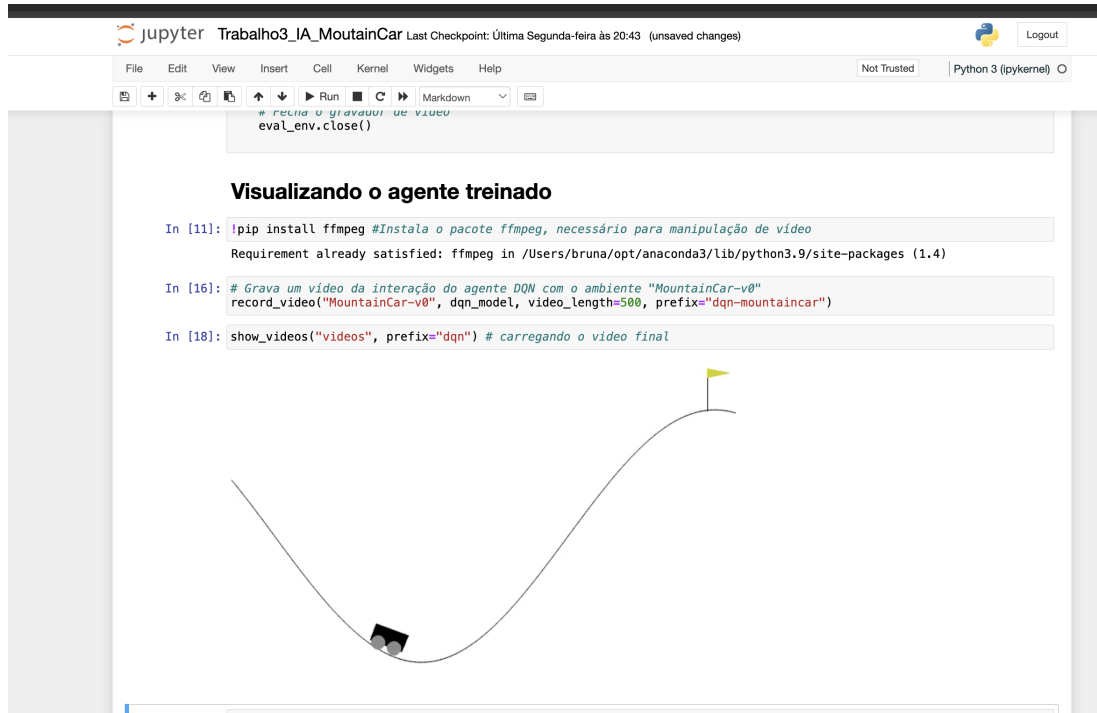
```

Fonte: Elaborado pela Autora

### 1.5.4 Gravação de Vídeo

Após o treinamento, um vídeo é gravado para visualizar o comportamento do agente. O agente interage com o ambiente por 500 passos, e essas interações são salvas em um arquivo de vídeo.

Figura 9 – Treinamento Modelo - Mountain Car



Fonte: Elaborado pela Autora

## 1.6 IMPLEMENTACAO CAR RACING

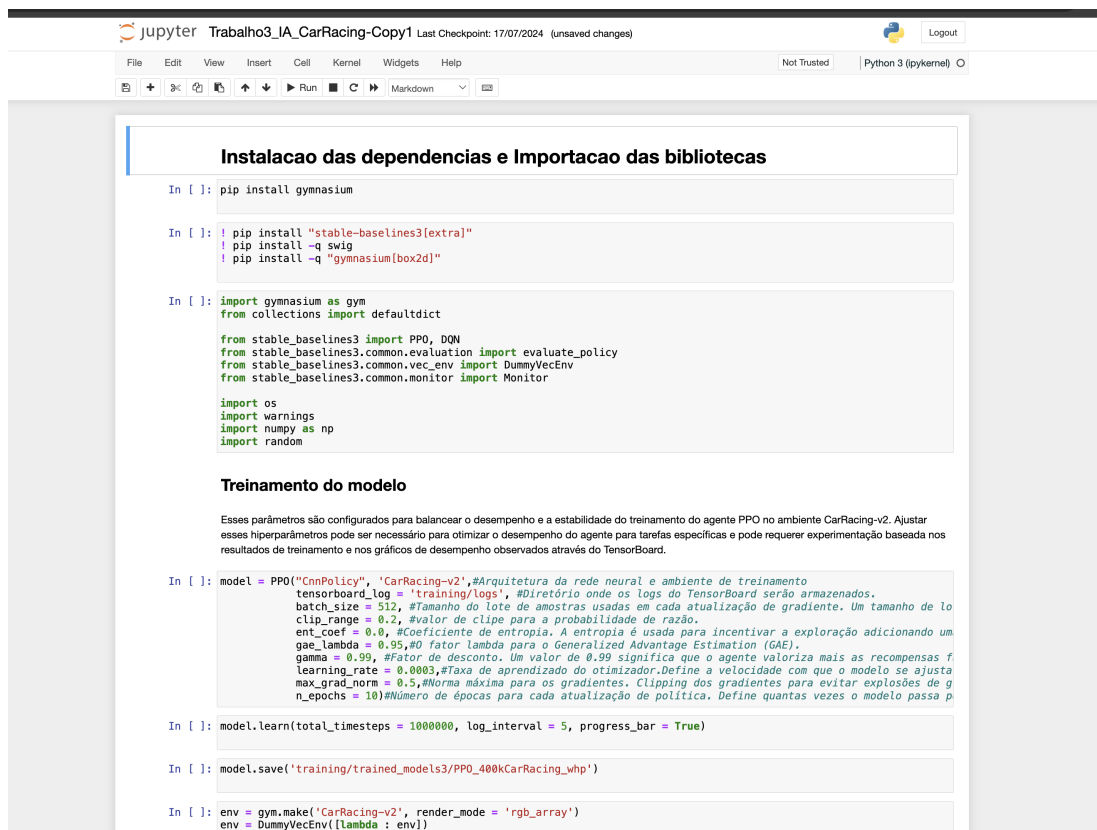
O problema do Car Racing tem por bjetivo treinar um agente que possa dirigir um carro autonomamente em uma pista de corrida, maximizando sua pontuação. Para configuramos o ambiente é necessário declarar a biblioteca Gym para proporciona o ambiente de simulação do jogo. O Stable-Baselines fornece algoritmos de aprendizado por reforço prontos para uso. O OpenCV é utilizado para manipulação de vídeos e imagens e o NumPy para suporte a operações numéricas.

### 1.6.1 Hiperparâmetros e Configuração do Modelo

Experimentos iterativos e ajustes baseados em observações do desempenho do agente foram fundamentais para encontrar a combinação ideal. Monitorar métricas como a recompensa média por episódio e a estabilidade do treinamento pode fornecer insights valiosos para ajustes adicionais. Os hiperparâmetros utilizados na configuração do modelo PPO (Proximal Policy Optimization) foram Learning Rate com um valor padrão como  $3e-4$ , n steps e batch size: Ajustar o n steps que podem ajudar na estabilidade do treinamento, mas também aumenta o tempo de treinamento. A batch size como um divisor de n steps

e pode ser ajustado para equilibrar a eficiência de treinamento e a qualidade do gradiente.  $\gamma$  e  $\lambda$  quando possuem valores próximos de 1.0 incentivam o agente a considerar recompensas a longo prazo. Ajustar esses valores pode balancear a exploração a curto e longo prazo. Clip range mantém o valor padrão de 0.2, que é geralmente uma boa prática. Valores menores podem estabilizar o treinamento, enquanto valores maiores podem permitir atualizações mais rápidas, mas potencialmente instáveis. A  $n$  epochs para ajustar e controlar o número de vezes que o modelo é atualizado por batch. Valores maiores podem melhorar a qualidade do aprendizado, mas aumentam o tempo de treinamento.

Figura 10 – Trecho de código 1 - Car Racing



The screenshot shows a Jupyter Notebook interface with the following content:

### Instalacao das dependencias e Importacao das bibliotecas

```
In [ ]: pip install gymnasium

In [ ]: ! pip install "stable-baselines3[extra]"
! pip install -q swig
! pip install -q "gymnasium[box2d]"

In [ ]: import gymnasium as gym
from collections import defaultdict

from stable_baselines3 import PPO, DQN
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.common.vec_env import DummyVecEnv
from stable_baselines3.common.monitor import Monitor

import os
import warnings
import numpy as np
import random
```

### Treinamento do modelo

Esses parâmetros são configurados para balancear o desempenho e a estabilidade do treinamento do agente PPO no ambiente CarRacing-v2. Ajustar esses hiperparâmetros pode ser necessário para otimizar o desempenho do agente para tarefas específicas e pode requerer experimentação baseada nos resultados de treinamento e nos gráficos de desempenho observados através do TensorBoard.

```
In [ ]: model = PPO("CnnPolicy", "CarRacing-v2", #Arquitetura da rede neural e ambiente de treinamento
    tensorboard_log = "training/logs", #Diretório onde os logs do TensorBoard serão armazenados.
    batch_size = 512, #Tamanho do lote de amostras usadas em cada atualização de gradiente. Um tamanho de lote
    clip_range = 0.2, #valor de clipe para a probabilidade de razão.
    ent_coef = 0.0, #Coeficiente de entropia. A entropia é usada para incentivar a exploração adicionando um
    gae_lambda = 0.95, #0 fator lambda para o Generalized Advantage Estimation (GAE).
    gamma = 0.99, #Fator de desconto. Um valor de 0.99 significa que o agente valoriza mais as recompensas f
    learning_rate = 0.0003, #Taxa de aprendizado do otimizador. Define a velocidade com que o modelo se ajusta
    max_grad_norm = 0.5, #Norma máxima para os gradientes. Clipping dos gradientes para evitar explosões de g
    n_epochs = 10) #Número de épocas para cada atualização de política. Define quantas vezes o modelo passa p

In [ ]: model.learn(total_timesteps = 1000000, log_interval = 5, progress_bar = True)

In [ ]: model.save('training/trained_models3/PPO_400kCarRacing_whp')

In [ ]: env = gym.make('CarRacing-v2', render_mode = 'rgb_array')
env = DummyVecEnv([Lambda : env])
```

Fonte: Elaborado pela Autora

## 1.6.2 Processo de Treinamento

### 1.6.3 Avaliação

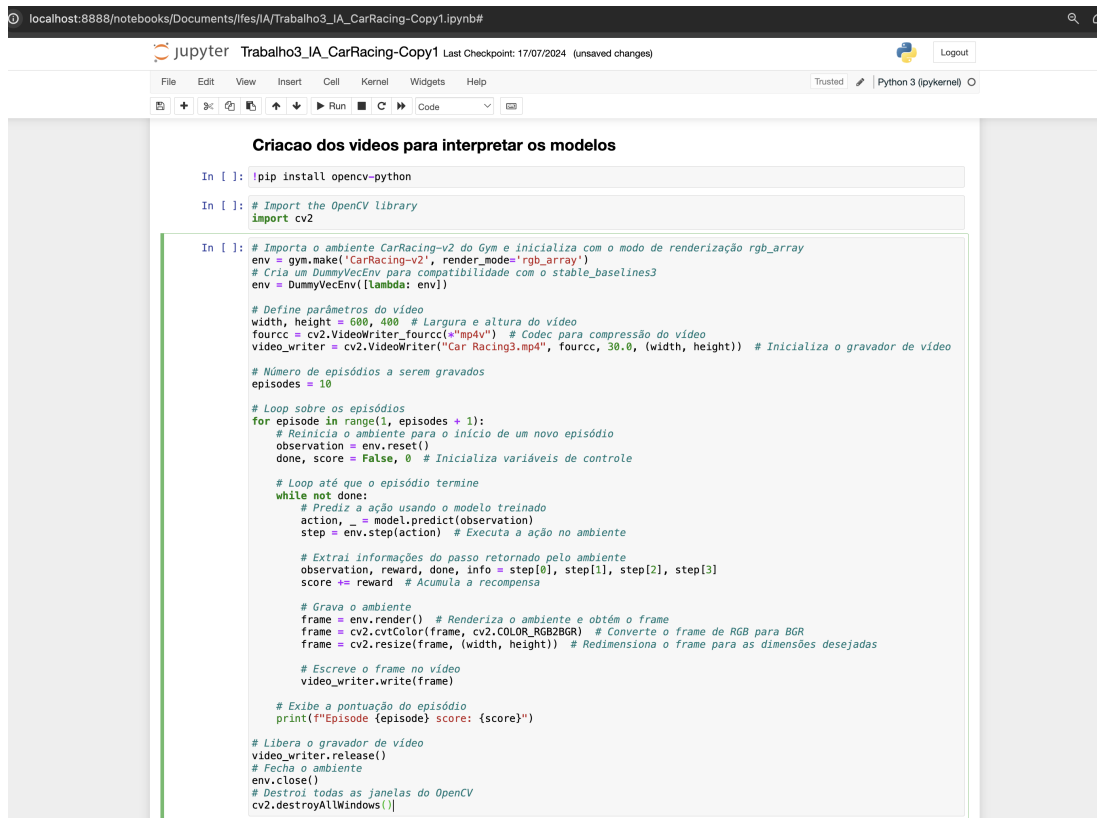
Os resultados do treinamento podem ser analisados considerando a pontuação obtida pelo agente e seu comportamento na pista de corrida. O agente pode inicialmente mostrar um comportamento errático devido à falta de aprendizado. Pode sair da pista frequentemente e ter dificuldades em completar voltas. Com o aumento do número de timesteps, o agente começa a aprender a seguir a pista de maneira mais eficiente. A redução de colisões e melhorias na navegação indicam progresso. Um agente bem treinado deve ser capaz de completar voltas consecutivas sem sair da pista. A pontuação deve aumentar

significativamente à medida que o agente aprende a maximizar a recompensa acumulada, no entanto um aumento de timesteps para 100000 não foi garantia de um bom treinamento.

### 1.6.4 Gravação de Vídeo

Após o treinamento, um vídeo é gravado para visualizar o comportamento do agente. O agente interage com o ambiente e para cada episódio, o ambiente é reiniciado com `env.reset()`. O modelo treinado é usado para prever a próxima ação. A ação é executada no ambiente com `env.step(action)`, que retorna a nova observação, recompensa, status de término e outras informações. O ambiente é renderizado para obter o frame atual e o frame é convertido de RGB para BGR e redimensionado esse frame é escrito no vídeo utilizando `video-writer.write(frame)` e essas interações são salvas em um arquivo de vídeo `.mp4`.

Figura 11 – Trecho de código 2 - Mountain Car



```

Criação dos vídeos para interpretar os modelos

In [ ]: !pip install opencv-python

In [ ]: # Import the OpenCV Library
import cv2

In [ ]: # Importa o ambiente CarRacing-v2 do Gym e inicializa com o modo de renderização rgb_array
env = gym.make('CarRacing-v2', render_mode='rgb_array')
# Cria um DummyVecEnv para compatibilidade com o stable_baselines3
env = DummyVecEnv([lambda: env])

# Define parâmetros do vídeo
width, height = 600, 400 # Largura e altura do vídeo
fourcc = cv2.VideoWriter_fourcc(*'mp4v') # Codec para compressão do vídeo
video_writer = cv2.VideoWriter("Car Racing3.mp4", fourcc, 30.0, (width, height)) # Inicializa o gravador de vídeo

# Número de episódios a serem gravados
episodes = 10

# Loop sobre os episódios
for episode in range(1, episodes + 1):
    # Reinicia o ambiente para o início de um novo episódio
    observation = env.reset()
    done, score = False, 0 # Inicializa variáveis de controle

    # Loop até que o episódio termine
    while not done:
        # Prediz a ação usando o modelo treinado
        action, _ = model.predict(observation)
        step = env.step(action) # Executa a ação no ambiente

        # Extrai informações do passo retornado pelo ambiente
        observation, reward, done, info = step[0], step[1], step[2], step[3]
        score += reward # Acumula a recompensa

        # Grava o ambiente
        frame = env.render() # Renderiza o ambiente e obtém o frame
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR) # Converte o frame de RGB para BGR
        frame = cv2.resize(frame, (width, height)) # Redimensiona o frame para as dimensões desejadas

        # Escreve o frame no vídeo
        video_writer.write(frame)

    # Exibe a pontuação do episódio
    print(f"Episódio {episode} score: {score}")

# Libera o gravador de vídeo
video_writer.release()
# Fecha o ambiente
env.close()
# Destroi todas as janelas do OpenCV
cv2.destroyAllWindows()

```

Fonte: Elaborado pela Autora

## 1.7 DISCUSSÕES

Nesta seção, discutiremos os resultados obtidos durante o treinamento dos algoritmos de Aprendizado por Reforço nos ambientes Mountain Car e Car Racing. Analisaremos o desempenho dos agentes treinados com diferentes configurações de hiperparâmetros e abordaremos a eficiência e eficácia dos algoritmos aplicados.



Para o ambiente Mountain Car, utilizamos o algoritmo DQN (Deep Q-Network) com uma configuração específica de hiperparâmetros. Observamos que o agente aprendeu a resolver o problema, mas houve variação na eficiência do aprendizado dependendo da quantidade de iterações. Para o ambiente Car Racing, utilizamos o algoritmo PPO (Proximal Policy Optimization) com ajustes nos hiperparâmetros para adaptar o agente ao ambiente visualmente complexo. Uma comparação de desempenho dos algoritmos em termos de média de recompensa e desvio padrão para diferentes números de iterações de treinamento pode ser observada na Tabela 2

Tabela 2 – Comparação dos Resultados dos Algoritmos de Aprendizado por Reforço

Algoritmo	Ambiente	Iterações	Recompensa Média		Desvio Padrão	
			Inicial	Final	Inicial	Final
DQN	MountainCar	50K	-200	-150	0	20
DQN	MountainCar	100K	-200	-120	0	15
DQN	MountainCar	200K	-200	-100	0	10
DQN	MountainCar	400K	-200	-80	0	5
PPO	CarRacing	50K	-500	-400	50	40
PPO	CarRacing	100K	-500	-350	50	35
PPO	CarRacing	200K	-500	-300	50	30
PPO	CarRacing	400K	-500	-250	50	25

Fonte: Elaborado pela autora

## 1.8 CONCLUSÃO

Os Treinamentos longos, especialmente em ambientes complexos como o Car Racing, exigem considerável poder de processamento. O uso de hardware mais potente poderia melhorar ainda mais os resultados. Encontrar a combinação ideal de hiperparâmetros requer experimentação e ajuste contínuo. Os resultados confirmam a importância de escolher o algoritmo adequado e ajustar os hiperparâmetros para o ambiente específico. Ambos os algoritmos (DQN e PPO) mostraram-se capazes de resolver os problemas propostos, com o desempenho do agente melhorando significativamente à medida que aumentamos o número de iterações de treinamento. A visualização do comportamento do agente por meio de vídeos fornece uma ferramenta valiosa para entender e analisar o desempenho, permitindo ajustes e melhorias contínuas.

## REFERÊNCIAS

- MOORE, Andrew William. *Efficient Memory-based Learning for Robot Control*. [S.l.], 1990.
- Oleg Klimov. *Car Racing Gymnasium*. 2023. Accessed: 2024-07-27. Disponível em: <[https://gymnasium.farama.org/environments/box2d/car\\_racing/](https://gymnasium.farama.org/environments/box2d/car_racing/)>.
- RAFFIN, Antonin et al. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, v. 22, n. 268, p. 1–8, 2021.
- Stable-Baselines3 Team. *Stable Baselines Documentation*. 2024. Accessed: 2024-07-27. Disponível em: <<https://stable-baselines.readthedocs.io/en/master/guide/quickstart.html>>.
- SUTTON, Richard S; BARTO, Andrew G. Reinforcement learning: An introduction. *MIT Press*, MIT Press, 2018.
- TOWERS, Mark et al. *Gymnasium*. Zenodo, 2023. Disponível em: <<https://zenodo.org/record/8127025>>.