Departamento de Computação/UFS

Otimização de Wrappers de recompensas na Implementação do Algoritmo Deep Q-Networks (DQN) no Jogo *Freeway* do Atari 2600

Optimization of Reward Wrappers in the Implementation of the Deep Q-Networks (DQN) Algorithm in the Atari *Freeway* Game

Heitor Saulo Dantas Santos [1]; Itor Carlos Souza Queiroz [1]; Lanna Luara Novaes Silva [1]; Lavínia Louise Rosa Santos [1]; Rômulo Menezes De Santana [1]

[1] *Departamento de Computação (DCOMP)*
Universidade Federal de Sergipe (UFS)
Av. Marechal Rondon, s/n– Jardim Rosa Elze– CEP 49100-000
São Cristóvão – SE– Brazil

heitor.santos@dcomp.ufs.br, itor_carlos@academico.ufs.br, lannaluara@academico.ufs.br, lavinialouise@academico.ufs.br, rmsantana@dcomp.ufs.br

Este estudo investiga a otimização de reward wrappers na implementação do algoritmo Deep Q-Networks (DQN) para o jogo Freeway do Atari 2600. Foram treinados quatro agentes baseados em DQN: dois utilizando redes neurais convolucionais (CNN e CNN-R) e dois com perceptrons multicamadas (MLP e MLP-R), sendo que CNN-R e MLP-R empregaram sistemas de recompensas personalizados. O objetivo foi superar as limitações do sistema de recompensas original, que apenas recompensa a travessia completa da rua, buscando acelerar e tornar o aprendizado mais eficiente. O reward wrapper desenvolvido introduziu recompensas intermediárias por progresso e penalidades por colisões, utilizando dados da RAM do jogo para uma detecção precisa de estados e eventos. Os resultados demonstraram que o sistema personalizado melhorou a eficiência dos agentes, embora ainda apresente desafios para alcançar um comportamento adaptativo semelhante ao de um jogador humano. Como trabalhos futuros, sugere-se a adoção de estratégias como aprendizado multi-agente e redes neurais recorrentes (RNNs) para capturar padrões temporais mais complexos e aprimorar a tomada de decisões.
Palavras-chave: Aprendizado por Reforço, Deep Q-Network, Freeway, Recompensa.

1. Introdução

A área de treinamento de agentes inteligentes em jogos digitais é explorada desde o surgimento desses jogos. Com o passar dos anos, novas técnicas e modelos surgiram, entre elas o Aprendizado por Reforço e as Redes Neurais. O Atari 2600 Video Computer System é um console de videogame lançado em 1977 pela empresa Atari, que possui diversos jogos que se tornaram uma rica fonte para treinamentos de modelos. Em 2013, a DeepMind introduziu o Deep Q-Network (DQN), aplicando Aprendizado Profundo para treinar agentes no Atari 2600 como Breakout e Pong [1]. Diante disso, o objetivo deste artigo é apresentar os dados de treinamento de um agente inteligente para jogar o jogo Freeway do console, utilizando DQN, Convolutional Neural Networks (CNN) e Multilayer Perceptron (MLP) buscando otimizar o sistema de recompensas.

O jogo selecionado como objeto de estudo foi o Freeway, em que o objetivo é controlar uma galinha a atravessar uma rua de forma linear passando por automóveis (Figura 1). Uma problemática existente com o sistema de recompensa do Freeway é que o seu funcionamento dificulta o Aprendizado por Reforço (RL - Reinforcement Learning). O RL utiliza de recompensas e penalidades para aprender a melhor política [2], entretanto a galinha do jogo só

recebe pontuação ao atravessar a rua por completo e não recebe penalidades na pontuação ao colidir com um automóvel. Com isso, sua recompensa é tardia, pois depende de uma sequência de ações corretas que durante o treinamento o modelo de pontuação não reflete esse caminho traçado. Todavia, o Freeway é um jogo em um ambiente determinístico e estático, o que significa que há padrões específicos a serem explorados. Sabendo disso, os experimentos apresentados nesse artigo buscaram, além de explorar alternativas mais diretas e intuitivas ao aplicar modelos de treinamento, desenvolver um sistema de recompensa personalizado que considerasse as problemáticas apresentadas.

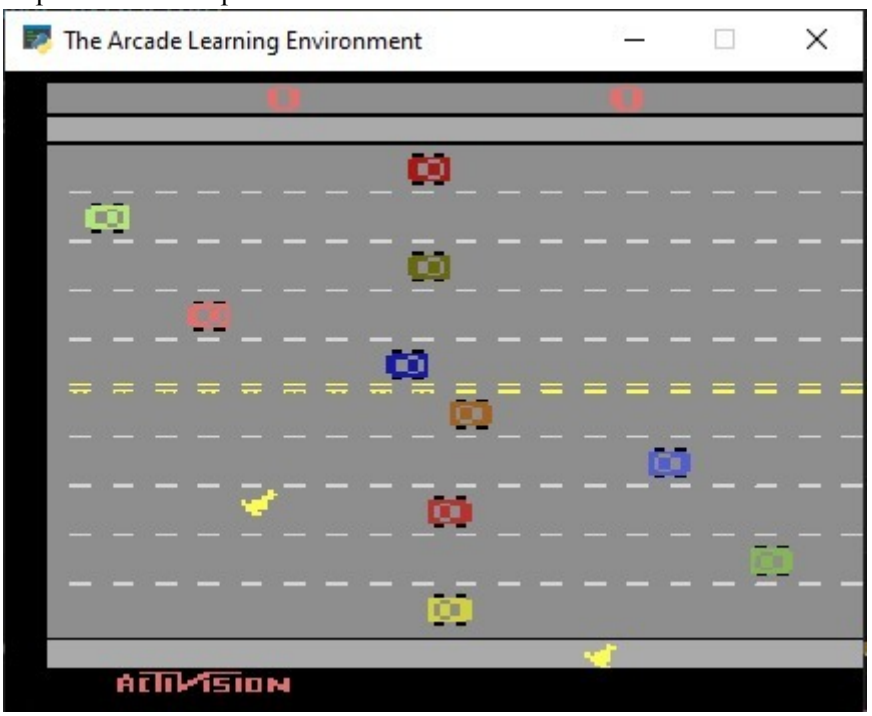

Figura 1: Ambiente do Freeway

A abordagem escolhida foi a utilização de dois modelos Deep Q-Network (DQN) com políticas diferentes a título de comparação e uma melhor análise de ambos os desempenhos. A primeira política é uma rede neural totalmente conectada com o Multilayer Perceptron (MLP) e a segunda política é a Convolutional Neural Networks (CNN). O sistema de recompensas desenvolvido foi implementado e aplicado a ambas políticas e os resultados dos experimentos serão apresentados ao longo do artigo.

A seguir será apresentado a metodologia detalhada e as especificações de implementação que embasaram os experimentos. Na seção 3 será apresentado os dados e especificações dos experimentos realizados. Na seção 4 será apresentado os resultados, comparações entre os modelos e o detalhamentos das soluções adquiridas. Por fim, as conclusões experimentais do que foi demonstrado e possíveis melhorias futuras.

2. METODOLOGIA

A metodologia utilizada foi baseada nos treinamentos de 4 agentes inteligente: DQN com política CNN e sistema de recompensa original (**CNN**), DQN com política MLP e sistema de recompensa original (**MLP**), DQN com política CNN e sistema de recompensa personalizado (**CNN-R)**, DQN com política MLP e sistema de recompensa personalizado (**MLP-R**). Para facilitar a referência aos agentes durante o artigo, os agentes serão identificados como as siglas destacadas. A seguir será apresentada a descrição de cada agente, os detalhes do sistema de recompensa personalizado, descrição de frameworks e bibliotecas e, por último, a descrição do hardware utilizado para treinamento.

2.1 Especificações dos agentes

Todos os agentes foram implementados utilizando a arquitetura Deep Q-Network (DQN), que combina Q-learning com redes neurais profundas para aproximar a função de valor Q, estimando a recompensa futura esperada para cada par estado-ação. O treinamento é baseado na equação de Bellman, que minimiza a diferença entre o valor Q previsto e o valor Q alvo. [3] A implementação foi feita com a biblioteca Stable-Baselines3, que inclui técnicas como *experience replay* (armazenamento de transições anteriores para estabilidade) e ε-greedy (balanceamento entre exploração e explotação) [4].

Foram utilizadas duas políticas de rede neural na DQN: Convolutional Neural Network (CNN) e Multilayer Perceptron (MLP). A CNN foi escolhida por sua capacidade de processar dados visuais, como os frames do jogo Freeway, extraindo padrões espaciais diretamente dos pixels. A arquitetura implementada é a NatureCNN, proposta por Mnih et al. (2015) [3], composta por três camadas convolucionais (32 filtros 8x8, 64 filtros 4x4 e 64 filtros 3x3, com strides 4x4, 2x2 e 1x1, respectivamente), seguidas por uma camada densa de 512 neurônios e uma camada linear final para mapear as ações (subir, descer ou ficar parado). Todas as camadas usam a função de ativação ReLU.

Já a MLP, mais simples que a CNN, mapeia observações pré-processadas (como a posição do agente e a distância até os carros) diretamente para ações. Sua arquitetura consiste em duas camadas densamente conectadas (64 neurônios cada, com ReLU) e uma camada linear final para as ações, embora também funcione para imagens, é uma política mais eficiente quando a entrada não é visual [5].

No treinamento dos agentes dois sistemas de recompensas foram testados, o sistema original do jogo Freeway e uma versão personalizada dos sistema de recompensas a qual será melhor explicada posteriormente, de maneira geral, essa nova versão inclui penalidades por colisões e recompensas intermediárias por progresso na direção correta. Esses dois tipos de sistemas foram usados com o objetivo de comparar também o desempenho dos agentes em relação a essas pontuações, buscando identificar como acelerar a convergência dos agentes para políticas mais eficientes.

2.1.1 DQN com política CNN e sistema de recompensas original (CNN)

Este agente foi o primeiro desenvolvido; ele utiliza o DQN com a política padrão CNN em que o agente começa a fazer uso da aleatoriedade para identificar padrões do ambiente para começar a aprender. Segue o pseudo código do modelo com os parâmetros utilizados.

```
##cria o ambiente Freeway usando o Gym Atari
##inicializa modelo com seguintes parâmetros
ModeloDQNCNN(
   -  Política de aprendizado: Cnn Policy
   -  Ambiente de treinamento: ambiente do Freeway
   -  Taxa de aprendizado: 0.0005
   -  Tamanho do buffer de replay: 100000
   -  Passos antes de iniciar o aprendizado: 1000
   -  Tamanho do lote de treinamento: 32
   -  Fator de desconto: 0.99
   -  Intervalo de atualização da rede-alvo: 1000
   -  Frequência de treinamento a cada 4 ações
   -  Nível de detalhamento da saída: 1
)
salva_modelo(arquivo)
treina_modelo(total de passos)
```

2.1.2 DQN com política CNN e sistema de recompensas personalizado (CNN-R)

Como o anterior, este agente também fez uso de DQN com política CNN porém com o sistema de recompensa personalizado e que foi desenvolvido durante os experimentos. Esse sistema será apresentado em detalhes na Seção 3 - Experimentos. Também foi utilizado um ambiente vetorizado para otimização que será mais detalhado no tópico 2.4. Segue o pseudo código do modelo com os parâmetros utilizados e a recompensa personalizada

```
      ##cria o ambiente Freeway usando o Gym Atari
      FreewayRewardWrapper(ambiente)     #aplicação     da     recompensa
personalizada
      ##inicializa modelo com seguintes parâmetros
      ModeloDQNCNN(
      Política de aprendizado: Cnn Policy
      Ambiente de treinamento: ambiente do Freeway
      Taxa de aprendizado: 0.0003
      Tamanho do buffer de replay: 50000
      Passos antes de iniciar o aprendizado: 50000
      Tamanho do lote de treinamento: 64
      Fator de desconto: 0.78
      Intervalo de atualização da rede-alvo: 1000
      Frequência de treinamento: 4 ações
      Fator de exploração: 0.12
      Fator de exploração final: 0.04
      Nível de detalhamento da saída: 1
      Local de treinamento: GPU, se disponível
      )
      treina_modelo(total  de  passos,  reward_callback)#treina  com  o
callback para armazenar a recompensa média por episódio.
      salva_modelo(arquivo)
```

2.1.3 DQN com política MLP e sistema de recompensas original (MLP)

Esse agente foi implementado através do uso de uma arquitetura de Rede Neural Profunda (DQN) com uma política baseada em Multilayer Perceptron (MLP). O sistema de recompensas original do jogo foi adotado. O pseudocódigo deste agente segue o mesmo padrão e valores de parâmetros que o agente CNN do subtópico 2.1.1, muda-se apenas a política de aprendizado para “MlpPolicy”.

2.1.4 DQN com política MLP e sistema de recompensas personalizado (MLP-R)

Esse agente também fez uso de DQN com política MLP porém com o sistema de recompensa personalizado que foi desenvolvido durante os experimentos que serão especificados melhor posteriormente. O pseudocódigo deste agente utiliza a mesma estrutura e valores de parâmetros descritos para o agente CNN-R no subtópico 2.1.2, alterando apenas a política de aprendizado para "MlpPolicy". Como o CNN-R, um ambiente vetorizado também foi utilizado.

2.2 Descrição de frameworks e bibliotecas

Para o treinamento foi usada a linguagem de programação Python 3.13.0 e durante a implementação dos treinamentos foram utilizadas algumas bibliotecas e frameworks. Também não houve a necessidade se utilizar o dataset do Freeway, pois o treinamento foi feito de forma interativa ao treinar o agente no ambiente e ele aprender com tentativa e erro. A seguir são apresentados as bibliotecas e frameworks utilizados:

a) **gymnasium [6]:** biblioteca usada para criar e interagir com ambientes de aprendizado por reforço; fornece ambientes prontos de jogos Atari.

```
import gymnasium as gym #importação
```

b) **stable_baselines3[4]:** framework que implementa algoritmos de aprendizado por reforço profundo (Deep RL); fornece o DQN e suas políticas de CNN e MLP.

```
!pip install stable_baselines3 #instalação
from stable_baselines3 import DQN #importação
from stable_baselines3.dqn import CnnPolicy
from stable_baselines3.dqn import MlpPolicy
```

c) **ale_py [7]:** fornece a interface para os jogos do Atari, permitindo que `gymnasium` os utilize

```
import ale_py
```

d) **numpy, time, os, matplotlib.pyplot:** bibliotecas do Python utilizadas como auxílio durante a implementação

e) **torch**: fornece suporte para operações em tensores e computação acelerada por GPU

2.3 Descrição do Hardware utilizado

Como forma de melhor análise de desempenho dos agentes inteligentes treinados foram utilizados diferentes hardware. Ao longo do artigo as máquinas serão referenciadas de acordo com as especificações contidas na Tabela 1

Tabela 1: Especificações de Hardware

	Hardware		
Identificador	**RAM**	**CPU/GPU**	**SO/Ambiente**
Máquina 1	16 GB	Intel Core i5-9300H	Windows 10
Máquina 2	12 GB	12 GB (VRAM)	Google Colab
Máquina 3	16 GB	Apple Silicon M1 (MPS)	MacOS

2.4 Utilização do framework Stable-Baselines3 para vetorização do ambiente

Como citado no tópico 2.1, os agentes que utilizam da recompensa personalizada foram implementados utilizando um ambiente vetorizado. Para essa abordagem, foram implementadas as classes VecFrameStack e DummyVecEnv do framework Stable-Baselines3. Os Ambientes Vetorizados são uma técnica utilizada para empilhar múltiplos ambientes independentes em um único ambiente. Em vez de treinar um agente de Aprendizado por Reforço (RL) em apenas um ambiente por etapa, essa abordagem permite que ele seja treinado simultaneamente em N ambientes por etapa. Como consequência, as ações enviadas ao ambiente passam a ser representadas por um vetor de dimensão N, o que também se aplica às observações, recompensas e sinais de término de episódio (dones). Os ambientes vetorizados são necessários para a aplicação de wrappers destinados a técnicas como empilhamento de quadros (frame-stacking) e normalização. Além disso, ao utilizar essa abordagem, os ambientes são redefinidos automaticamente ao final de cada episódio. Dessa forma, a observação retornada para o i-ésimo ambiente, quando done[i] for verdadeiro, será a primeira observação do próximo episódio, e não a última do episódio recém-finalizado. Caso seja necessário acessar a observação final real do episódio encerrado—ou seja, aquela associada ao evento done gerado

pelo ambiente subjacente—é possível obtê-la por meio da chave terminal_observation presente nos dicionários info retornados pelo VecEnv [4]. A execução do código chegou a ser aproximadamente 10 vezes mais rápida quando comparadas às soluções sem vetorização de ambiente e aplicação do construtor DummyVecEnv. Além disso, é possível sobrescrever e personalizar métodos dessas classes.

Assim como foi explicado anteriormente, o uso de ambientes vetorizados permite executar várias instâncias de um mesmo ambiente em paralelo, otimizando o processo de coleta de amostras durante o treino. Dessa forma, cada passo de interação abrange múltiplas observações, reduzindo o tempo total de treinamento. Os métodos de construção, como make_vec_env ou diretamente DummyVecEnv, criam esses ambientes paralelos de forma simples. Quando integrados a modelos DQN, PPO ou A2C da biblioteca stable_baselines3, facilitam a amostragem síncrona ou assíncrona dos dados, permitindo alto aproveitamento de hardware. A vetorização também possibilita callbacks obtendo estatísticas de episódios simultâneos, auxiliando na análise e monitoramento do agente em cenários complexos. Cada ambiente gera recompensas e estados que são replicados para a rede, otimizando o uso do batch de treinamento e tornando o processo mais robusto e escalável.

3. EXPERIMENTOS

Como informado, os experimentos foram feitos com cada agente sendo executado nas três máquinas descritas no tópico 2.3 Descrição do Hardware utilizado. A seguir serão detalhadas as recompensas utilizadas, os parâmetros que afetam o treinamento e o processo utilizando o posicionamento da RAM para a recompensa personalizada.

3.1 Valores de Recompensas Utilizados

Durante um treinamento os valores de recompensa podem ser maiores que zero – indicando uma recompensa positiva – e menores que zero, indicando penalidades por uma ação ruim. Cada um desses valores refletem no aprendizado e desempenho do agente ao treinar pelo ambiente do jogo. Os agentes que utilizaram o sistema de recompensa original do jogo possuem somente o +1 se atravessar a rua, enquanto o sistema de recompensas desenvolvido possui valores para outras ações. As ações identificada e suas respectivas recompensas implementadas estão na Tabela 2 a seguir:

Tabela 2: Recompensas

Id	Recompensas				
	Descrição da Ação	Valor			
		CNN	MLP	CNN-R	MLP-R
Ação 1	Galinha avançou e não colidiu	0	0	+0.25	+0.5
Ação 2	Galinha avançou e retornou, com colisão	0	0	-0.95	-0.95
Ação 3	Retornou (com ou sem colisão)	0	0	0	0
Ação 4	Atravessou com sucesso	+1	+1	+10	+3.5
Ação 5	Ficar parado	0	0	-0.035	-0.50

3.2 Parâmetros de Treinamento

Os parâmetros de treinamento são configurações essenciais que guiam o processo de aprendizado dos modelos, influenciando diretamente a eficiência, estabilidade e desempenho final dos agentes. A seguir, são descritos os principais parâmetros utilizados:

a) **learning_rate (taxa de aprendizado):** define o tamanho dos ajustes realizados nos pesos da rede neural durante o treinamento, equilibrando a velocidade e a estabilidade da convergência.
b) **buffer_size (tamanho do buffer):** determina a capacidade da memória de replay, que armazena experiências passadas para amostragem durante o treinamento, promovendo a diversidade dos dados.
c) **learning_starts (início do aprendizado)**: especifica o número de passos iniciais em que o agente coleta experiências sem atualizar a rede, garantindo um buffer inicial suficiente para amostragem.
d) **batch_size (tamanho do lote):** define quantas experiências são amostradas do buffer para cada atualização da rede, influenciando a eficiência e a estabilidade do treinamento.
e) **gamma (fator de desconto):** controla a importância das recompensas futuras em relação às imediatas, sendo essencial para o cálculo do valor esperado das ações.
f) **target_update_interval (intervalo de atualização da rede alvo):** define com que frequência a rede alvo é atualizada, ajudando a estabilizar o treinamento ao reduzir a volatilidade das estimativas de Q-values.
g) **train_freq (frequência de treinamento):** determina quantos passos do ambiente são realizados entre cada atualização da rede.
h) **exploration_fraction (fração de exploração) e exploration_final_eps (valor final de epsilon):** controlam a política de exploração ε-greedy, definindo a redução gradual da taxa de exploração ao longo do tempo.
i) **verbose (nível de detalhamento):** ajusta a quantidade de informações exibidas durante o treinamento, facilitando o monitoramento do processo.
j) **total_timesteps (quantidade de passos):** número total de passos no ambiente durante o treinamento. Define a exposição do agente a diferentes situações, influenciando a robustez da política aprendida. Um número maior permite exploração mais ampla, mas aumenta o custo computacional. Um número menor pode limitar o aprendizado, porém é mais eficiente em termos de recursos.

Esses parâmetros foram cuidadosamente configurados e testados com o objetivo de alcançar a otimização do desempenho dos agentes no jogo Freeway. Os parâmetros, já apresentados nos pseudocódigos dos agentes no tópico 2.1 Especificações dos Agentes, são detalhados a seguir visando facilitar a compreensão do impacto dos valores implementados em ambas as versões dos sistemas para uma análise mais clara de suas funções e efeitos no treinamento.

3.2.1 Agentes com sistema de recompensas original (CNN e MLP)

Na implementação dos dois agentes que utilizam o sistema de recompensas original foram definidos os mesmos valores de parâmetros, tendo como objetivo comparar as políticas CNN e MLP sob as mesmas condições de treinamento. O learning_rate foi configurado como 0.0005, um valor relativamente baixo que permite ajustes suaves nos pesos da rede, garantindo estabilidade durante o aprendizado. O buffer_size de 10000 define uma memória de replay ampla, capaz de armazenar uma quantidade significativa de experiências, o que ajuda a diversificar as amostras e evitar overfitting. O learning_starts de 1000 assegura que o agente colete dados suficientes antes de iniciar o treinamento e o batch_size de 32 equilibra eficiência e estabilidade nas atualizações da rede. O gamma de 0.99 prioriza recompensas futuras, incentivando o agente a adotar estratégias de longo prazo. O target_update_interval de 1000 passos define a frequência de atualização da rede alvo, reduzindo a volatilidade das estimativas de Q-values, o train_freq de 4 atualiza a rede a cada 4 passos, mantendo equilíbrio entre exploração e aprendizado e por fim, o verbose de 1 ativa um monitoramento moderado do progresso. Esses parâmetros buscaram promover um treinamento estável e eficiente para ambas as políticas.

3.2.2 Agentes com sistema de recompensas personalizado (CNN-R e MLP-R)

Assim como nos agentes com sistema de recompensa original, os parâmetros dos agentes com recompensa personalizada foram mantidos iguais para ambas as políticas (CNN-R e MLP-R), O learning_rate de 0,0003 foi escolhido para permitir um aprendizado mais rápido em comparação ao valor anterior, adaptando-se às recompensas personalizadas e o buffer_size de 50000 aumenta a capacidade da memória de replay, armazenando mais experiências e melhorando a diversidade das amostras. O learning_starts de 50000 garante que o agente colete uma quantidade significativa de dados antes de iniciar o treinamento. O batch_size de 64 amplia o número de experiências usadas em cada atualização da rede, promovendo uma maior estabilidade no aprendizado e o gamma de 0.78 reduz a importância das recompensas futuras, refletindo a priorização de recompensas personalizadas de curto e médio prazo. O target_update_interval, o train_freq e o verbose seguem o mesmo padrão e valores dos agentes com sistema de recompensa original. O exploration_fraction de 0.12 e o exploration_final_eps de 0.04 ajustam a política de exploração ε-greedy, reduzindo a taxa de exploração mais rapidamente para focar em estratégias já aprendidas. Esses parâmetros foram configurados buscando otimizar o desempenho dos agentes com recompensas personalizadas.

3.3 Recompensa personalizada e posicionamento da RAM

Inicialmente, os modelos foram treinados sem Wrappers de recompensa, utilizando apenas a pontuação fornecida pelo jogo a cada travessia bem-sucedida da pista. Essa abordagem produzia resultados inconstantes, pois o agente dependia exclusivamente de exploração aleatória para maximizar o retorno, sem considerar colisões com os carros como evento negativo. Em seguida, buscou-se aprimorar o sistema de recompensa do Freeway de diversas maneiras.

A recompensa personalizada utiliza do estudo da posição de colisões e do posicionamento da RAM para mapear os estados do jogo e assim ajustar os padrões de recompensa. Essa abordagem foi inspirada em um repositório que utiliza do estudo do posicionamento da RAM em jogos do Atari [8]. A posição de colisão é a posição em que a galinha colide com um dos automóveis e recua (Ação 2). Nesta seção, será apresentado detalhadamente o processo para desenvolver essa abordagem. Segue o pseudocódigo para melhor entendimento:

```
1  FreewayRewardWrapper():
2      inicializa a recompensa e a posição do jogador em 0
3      função reset(estado atual do ambiente):
4          inicializa a ram #self.env.unwrapped.ale.getRAM()
5          inicializa a última pontuação na ram[103]
6          inicializa a última posição da galinha em ram[14]
7          limpa as recompensas do episódio atual
8          retorna estado atual do ambiente
9      função reward(ram, ação):
10         inicializa a posição da galinha em ram[14]
11         inicializa a pontuação na ram[103]
12         inicializa carros com a faixa de posição dos carros na ram
13         reward = 0
14         posicao na faixa = verify_car_lane()
15         se posição da galinha > ultima posicao #moveu para cima
16             reward = reward + 0.25
17                  se  (check_collision(carros,  posicao  na  faixa)  ou
check_collision_with_action e posicao atual < ultima posicao)
18             reward = reward - 0.95 # colisao identificada
19         se posicao atual = ultima posicao
20             reward = reward - 0.035 # ficou parada
21     # amplificar impacto da pontuação
22         se pontuação atual > ultima pontuacao
23             reward = (pontuacao - ultima pontuacao) * 10
```

```
24          atualiza estados (pontuação e posicao) anteriores
25          retorna reward
```

A lógica principal para avaliação de recompensas concentra-se na classe FreewayRewardWrapper, onde a função verify_car_lane mapeia a posição em RAM (ram[14]) para uma das faixas de tráfego especificadas em faixas, e as funções check_collision e check_collision_with_action penalizam ações que não evitem colisões ou não contribuam para o progresso na pontuação. A customização do sistema de recompensas reforça avanços positivos (como subir na tela) e desencoraja estagnação e colisões. Quando o agente pontua (ram[103] aumenta), a recompensa é amplificada, equilibrando motivação para movimentos corretos e punições por decisões equivocadas.

A primeira tentativa de detecção de colisões focou em observar a ação do agente e sua posição. Se o agente recuasse sem comando explícito para isso, o evento era interpretado como colisão. Recorrer às informações da RAM foi essencial para obter a posição em y do agente e as coordenadas x dos carros. Entretanto, toda vez que o agente atravessava a rua e retornava ao ponto inicial, essa mecânica equivocadamente computava o fato como colisão, penalizando ações corretas. Após analisar essas inconsistências, métodos mais precisos de detecção de colisões foram examinados, identificando a posição x do agente em um endereço específico (25) na RAM, o que permitiu correlacionar a localização do agente ao posicionamento em y de cada faixa de rodovia. Desse modo, pode-se detectar o momento exato em que ocorrem colisões de maneira mais confiável. A combinação dessas estratégias resultou em um sistema de detecção mais eficiente, aperfeiçoando o treinamento do modelo ao equilibrar avanços e punições de forma mais robusta.

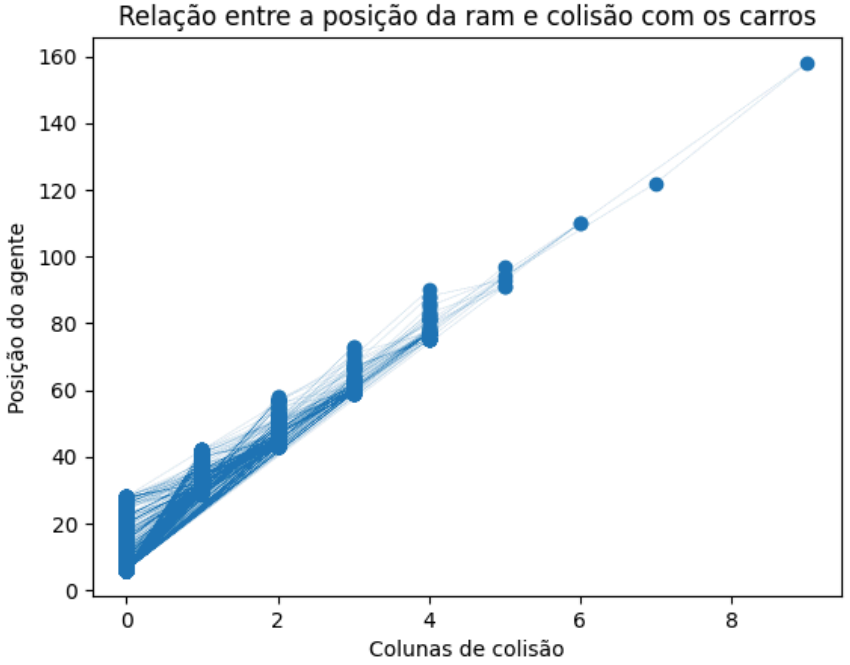

Figura 2: Gráfico da posição do agente nas colisões

4. Resultados e Discussão

4.1 Resultados da execução dos modelos em cada máquina

Na Tabela 3, são apresentados os resultados de execução dos treinamentos nas diferentes máquinas, em que ***p*** representa a quantidade de passos executado e ***t*** o tempo total do

treinamento. Como forma de registro e comparação, na tabela abaixo os testes foram padronizados para 100.000 passos; máquinas ‘sem teste“, foram máquinas que o agente referente não foi testado.

Tabela 3: Modelo em cada máquina

Agente	Execução dos agentes					
	Máquina 1		Máquina 2		Máquina 3	
	p	t	p	t(s)	p	t(s)
CNN	100.000	14760	100.000	563	100.000	5400
MLP	100.000	11232	100.000	394	100.000	1200
CNN-R	*sem teste*	*sem teste*	*sem teste*	*sem teste*	100.000	1080
MLP-R	*sem teste*	*sem teste*	*sem teste*	*sem teste*	100.000	43

4.2 Resultados de desempenho da CNN

A DQN com política CNN e recompensa original ao ser executada nas máquinas com 100.000 passos o tempo de execução foi os mostrados na Tabela 3. A duração foi dentro do esperado levando em consideração as configurações de hardware de cada máquina. A galinha se movia para frente, colidia, retornava e atravessava a rua. Após um tempo de treinamento foi observado que ela ficava mais inteligente e atravessava mais a rua, porém voltava a cometer os mesmos erros do início do treinamento. A pontuação máxima utilizando visualmente a recompensa original e pontuação do jogo foi de 24 pontos. No gráfico a seguir é apresentada a evolução das recompensas durante o treinamento em cada episódio. Nota-se que no início em que o agente está aprendendo a recompensa é 0, mas entre o episódio 20 e 30 ele começa a aprender rapidamente e entra em crescente no aprendizado, mas continua decaindo em alguns momentos por conta dos erros citados acima. A linha em laranja representa a média das recompensas obtidas.

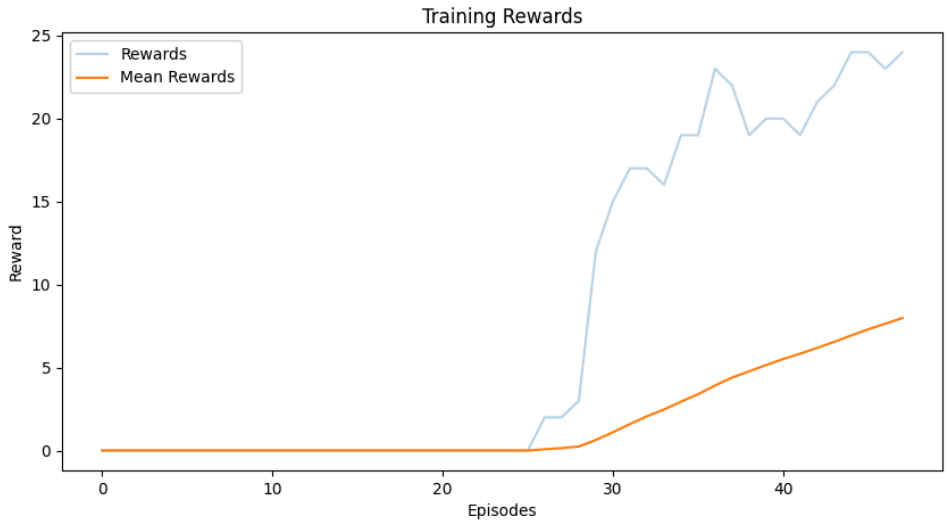

Figura 3: Gráfico da Recompensa no treino da CNN padrão

4.3 Resultados de desempenho da CNN-R

A DQN com política CNN e com recompensa personalizada ao ser executada com 600.000 passos não apresentou os resultados esperados. Foram criadas manipulações das recompensas e punições, além da alteração da taxa de exploração e, ainda assim, o agente aprendeu que deveria usar somente a ação 1 (movimentar-se para frente) até atravessar a rua.

Apesar de acontecerem múltiplas colisões nesse processo, o agente conseguiu pontuar (atravessar completamente a rua) consideravelmente bem. O gráfico a seguir mostra a evolução das recompensas durante o treinamento da CNN-R. A linha azul representa as recompensas por episódio, com variação significativa ao longo do tempo e a linha laranja indica a média das recompensas, que cresce gradualmente até estabilizar por volta do 50º episódio.

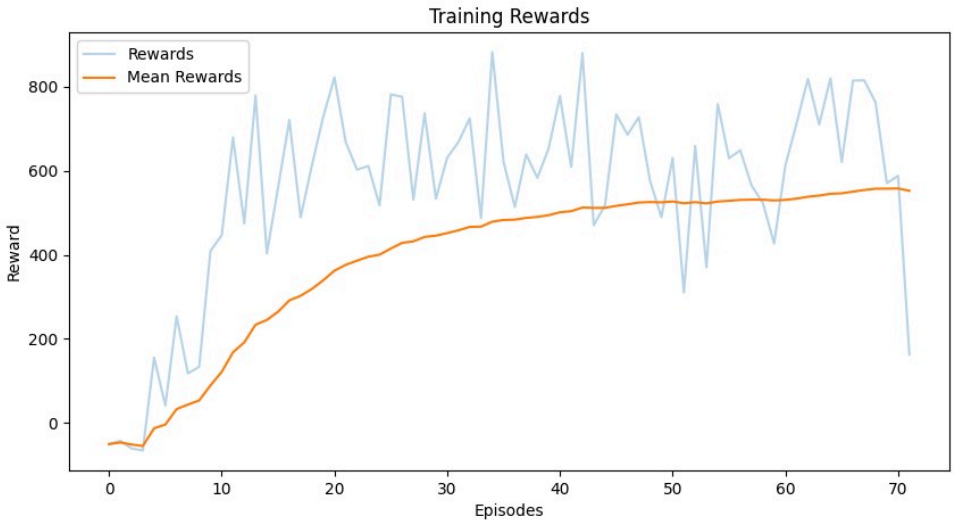

Figura 4: Gráfico da Recompensa no treino da CNN personalizada

4.4 Resultados de desempenho da MLP

A DQN com política MLP e recompensa original ao ser executada com 100.000 passos durou um tempo de treinamento de 6 minutos e 34 segundos na Máquina 1 e 2 horas e 53 minutos na Máquina 3. A duração foi dentro do esperado levando em consideração as configurações de hardware de cada máquina. A galinha se comportou de maneira semelhante ao treinar com a CNN, mas o treinamento foi consideravelmente mais rápido em ambas as máquinas. Outras execuções realizadas também demonstraram esse aumento de velocidade ao se treinar com MLP. A pontuação máxima utilizando visualmente a recompensa original e pontuação do jogo foi de 6 pontos.

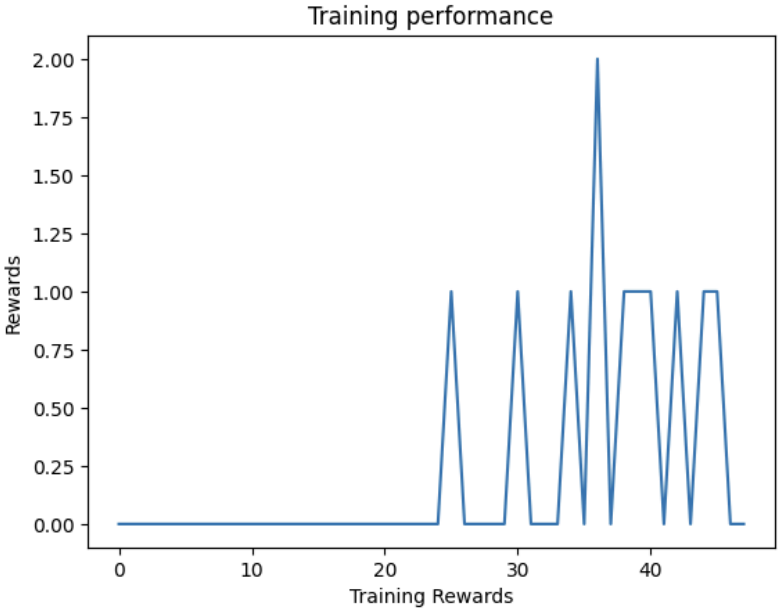

Figura 5: Gráfico da Recompensa no treino da MLP padrão

4.5 Resultados de desempenho da MLP-R

A DQN com política MLP e recompensa personalizada ao ser executada com 100.000 passos durou um tempo de treinamento de 43 segundos na Máquina 4, fazendo uso de ambientes vetorizados para maior velocidade de execução do treino. A duração foi dentro do esperado levando em consideração as configurações de hardware de cada máquina. O comportamento do agente não foi bem sucedido: ela se movimentou para cima e para baixo múltiplas vezes, mas praticamente sem sair do lugar e não atravessou a rua. O gráfico a seguir exibe as recompensas durante o treinamento da MLP-R. A linha azul mostra variações abruptas nos episódios finais, enquanto a linha laranja (média das recompensas) apresenta crescimento gradual até estabilizar. Apesar da melhora tardia, o agente teve dificuldades para aprender, com desempenho inferior ao esperado.

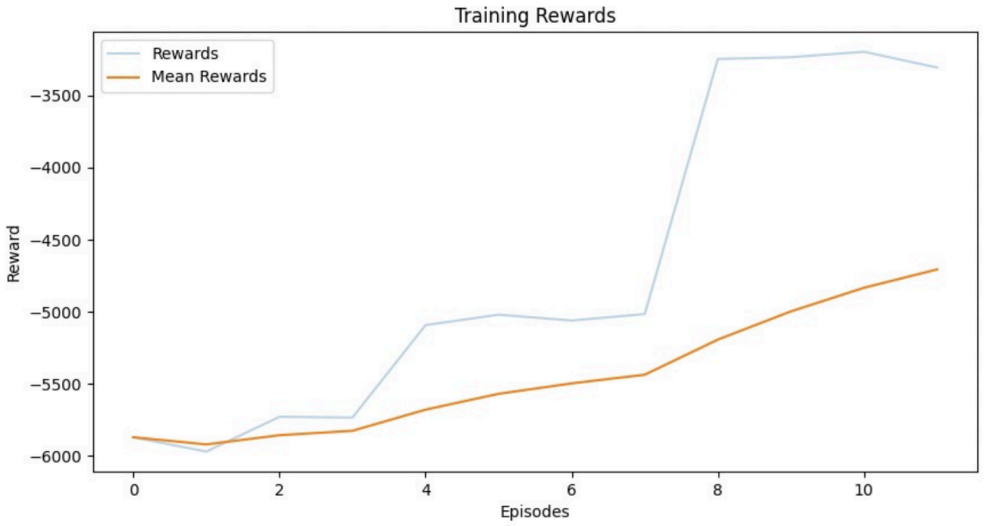

Figura 6: Gráfico da Recompensa no treino da MLP personalizada

5. Conclusão

Os experimentos realizados demonstraram que a modificação do sistema de recompensas teve um impacto significativo na aprendizagem do agente no jogo Freeway. A implementação do Reward Wrapper personalizado permitiu acelerar o processo de aprendizado ao fornecer feedback mais granular sobre as ações do agente. Entretanto, os resultados indicam que, embora a adaptação das recompensas tenha melhorado a eficiência do treinamento, ainda existem desafios consideráveis para alcançar um comportamento semelhante ao de um jogador humano.

A principal dificuldade encontrada foi equilibrar as recompensas e penalidades de forma a incentivar uma estratégia mais natural e eficiente. Os testes mostraram que, em alguns casos, os agentes desenvolvem comportamentos subótimos, como avançar de maneira persistente sem considerar colisões ou hesitar em tomar ações estratégicas. Essa limitação reforça a necessidade de ajustes finos nos pesos das recompensas e possivelmente a combinação de diferentes técnicas de aprendizado por reforço, como métodos de aprendizado hierárquico ou aprendizado por imitação.

Além disso, a análise da RAM do jogo Freeway revelou informações valiosas sobre a dinâmica interna do jogo, possibilitando a exploração de abordagens alternativas. A utilização desses dados para inferir estados mais ricos e detalhados pode ser um caminho promissor para a melhoria do desempenho dos agentes. No futuro, técnicas mais avançadas de extração de

features e engenharia de recompensas podem ser empregadas para aprimorar a performance dos agentes e torná-los mais adaptáveis a diferentes cenários.

Os tempos de execução dos experimentos também evidenciam o impacto das modificações realizadas. Considerando o cenário padrão, a arquitetura CNN padrão levou 563 segundos e a MLP levou 394 segundos na Máquina 2, enquanto na Máquina 3 os tempos foram de 5400 segundos para CNN e 1200 segundos para MLP padrão. Já com o sistema de recompensas alterado, a CNN levou 1080 segundos e a MLP levou apenas 43 segundos. Tais resultados sugerem que a modificação das recompensas e a otimização computacional por meio da vetorização não apenas afetaram nos comportamentos dos agentes, mas também impactaram diretamente na eficiência computacional do treinamento dos modelos, em especial no caso da MLP, com uma redução drástica no seu respectivo tempo.

Como trabalhos futuros, sugere-se investigar estratégias como aprendizado multi-agente, onde diferentes agentes cooperam ou competem para melhorar a eficiência do aprendizado, além da aplicação de redes neurais recorrentes (RNNs) para capturar padrões temporais mais complexos. Outra possibilidade é testar arquiteturas híbridas que combinam DQN com métodos baseados em vantagem (A2C, PPO), o que pode resultar em políticas mais robustas e eficientes, já que o Freeway por natureza tem como objetivo a competição entre dois dois jogadores para competir quem pontua com mais eficiência, essa seria uma abordagem mais natural e robusta para o melhor desempenho do agente.

Por fim, este estudo reforça a importância da adaptação de sistemas de recompensa no aprendizado por reforço, destacando tanto os benefícios quanto os desafios dessa abordagem. Os insights obtidos fornecem um ponto de partida valioso para pesquisas futuras que busquem aprimorar a capacidade dos agentes de aprendizado por reforço em jogos e outros domínios complexos.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] MNIH, Volodymyr; KAVUKCUOGLU, Koray; SILVER, David; et al. **Playing Atari with Deep Reinforcement Learning.** [s.l.: s.n.], 2013. Disponível em: <https://arxiv.org/pdf/1312.5602>.

[2] Sutton, R. S., & Barto, A. G. (2018). **Reinforcement Learning: An Introduction** (2nd Edition). MIT Press. Capítulo 1: The Reinforcement Learning Problem (pp. 1-20)

[3] MNIH, VOLODYMYR, et al. "**Human-level control through deep reinforcement learning.**" nature
518.7540 (2015): 529-533.

[4] Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations — Stable Baselines3 2.6.0a1 documentation. Readthedocs.io. Disponível em: <https://stable-baselines3.readthedocs.io/en/master/>. Acesso em: 21 fev. 2025.

[5] MOREIRA, S. Rede Neural Perceptron Multicamadas. Disponível em: <https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9>.

[6] Gymnasium Documentation. Farama.org. Disponível em: <https://gymnasium.farama.org/index.html>. Acesso em: 21 fev. 2025.

[7] ALE Documentation. Farama.org. Disponível em: <https://ale.farama.org/index.html>. Acesso em: 21 fev. 2025.

[8] atari-representation-learning/atariari/benchmark/ram_annotations.py at master · mila-iqia/atari-representation-learning. GitHub. Disponível em: <https://github.com/mila-iqia/atari-representation-learning/blob/master/atariari/benchmark/ram_annotations.py>. Acesso em: 21 fev. 2025.