Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

TREINANDO UMA REDE NEURAL PARA DIRIGIR EM UM SIMULADOR DE CORRIDA

Bruno Biagi de Lima Piveta

Dr. Rubens Barbosa Filho (Orientador)

Dourados - MS
2024

Treinando uma rede neural para dirigir em um simulador de corrida

Bruno Biagi de Lima Piveta

Projeto apresentado ao curso de Ciência da Computação da Universidade Estadual de Mato Grosso do Sul como parte dos requisitos para aprovação na disciplina Projeto Final de Curso.

Dourados, 04 de novembro de 2024.

Prof. Dr. Rubens Barbosa Filho (Orientador)

P764t Piveta, Bruno Biagi de Lima

Treinando uma rede neural para dirigir em um simulador de corrida / Bruno Biagi de Lima Piveta. – Dourados,MS: UEMS, 2024.
33 p.

Trabalho de Conclusão de Curso (Graduação) – Ciência da Computação – Universidade Estadual de Mato Grosso do Sul, 2024.
Orientador: Profº. Drº. Rubens Barbosa Filho

1. Aprendizado de máquina (*machine learning*). 2. Redes neurais artificiais. 3. Aprendizado de máquina por reforço (*Deep Q-Network*) 4. Inteligência Artificial (IA). 5. Simulador de corrida I. Barbosa Filho, Rubens. II. Título.

CDD 23 ed. 006.31

Ficha Catalográfica elaborada pela bibliotecária da Universidade Estadual de Mato Grosso do Sul (UEMS)
Aline Perdomo Soutelo, inscrita sob o CRB n. 3668 - 1ª Região.

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

TREINANDO UMA REDE NEURAL PARA DIRIGIR EM UM SIMULADOR DE CORRIDA

Bruno Biagi de Lima Piveta

Novembro de 2024

Banca Examinadora:
Prof. Dr. Rubens Barbosa Filho (Orientador)
Área de Computação – UEMS
Prof. Dr. Diogo Fernando Trevisan
Área de Computação – UEMS
Prof. Dr. Osvaldo Vargas Jaques
Área de Computação – UEMS

Resumo

Neste trabalho foi feita uma implementação de uma rede neural para dirigir um carro em um simulador de corrida, o projeto foi feito em *Python* com o simulador sendo feito usando a biblioteca *Pygame* e a rede neural usando a biblioteca *Pytorch*. A rede neural implementada usou como base o algoritmo de aprendizado por reforço *Deep Q-Learning*. O treinamento do agente foi feito utilizando seis pistas de corridas diferentes. O resultado do treinamento foi satisfatório com o agente completando consistentemente voltas nas pistas onde ele foi treinado.

Palavras-chave: *Aprendizado de Máquina*, *Aprendizado por reforço*, *Simulador de corrida*.

Sumário

Lista de ilustrações

1 INTRODUÇÃO

A Inteligência Artifical (IA) é um tema que cada vez mais vem ganhando notoriedade no mundo contemporâneo. Está presente nas mais variadas áreas, desde laser e entreterimento como em jogos eletrônicos e algoritmos de recomendações das redes sociais até aréas como medicina e agricultura.

O Aprendizado de máquina (*machine learning*) é um subcampo de IA que se foca em criar sistemas que podem treinar e aprender usando dados para melhorar em uma determinada tarefa. Uma IA pode ser treinada para reconhecer padrões em um conjunto de dados, reconhecimento facial, responder à perguntas, dirigir carros autômatos, entre várias outras aplicações.

IA é uma ferramenta poderosa que pode ser usada nas mais diversas situações, porém surge a pergunta de como tal ferramenta pode ser implementada nessas situações, tendo em vista isso, este projeto visa ser um exemplo simples de como uma IA usando aprendizado de máquina pode ser implementada.

O objetivo geral deste projeto é desenvolver um sistema de IA usando redes neurais artificiais e aprendizado de máquina. Esse sistema será usado para a aprendizagem de um agente que consiga dirigir um carro para percorrer as pistas criadas. O objetivo geral pode ser dividido em cinco objetivos específicos:

O1 Desenvolver um simulador de corrida simples;

O2 Criar um conjunto de pistas que a IA terá que percorrer;

O3 Desenvolver uma IA usando redes neurais artificiais;

O4 Treinar um agente para percorrer as pistas criadas;

O5 O agente deve conseguir completar de forma consistente voltas pela pista sem que colida com as paredes.

Para implementar o aprendizado da rede neural foi utilizado o algoritmo *Deep Q-Learning*, esse algoritmo foi escolhido por ser um algoritmo que não depende de conhecimento prévio sobre o ambiente e faz a escolhas de suas ações visando maximizar a recompensa tanto imediata quanto futura, com isso o algoritmo sempre visará percorrer o caminho mais rápido pela pista.

O sistema desenvolvido nesse projeto possui várias aplicações, podendo ser adaptado para outros jogos ou simuladores com funcionalidades parecidas, ou ser adaptado para controlar um carro de controle remoto, drone ou um kart para percorrer um percurso fechado.

O texto será organizado em cinco Capítulos:

O primeiro Capítulo foi a introdução do projeto. O segundo Capítulo será um estudo sobre redes neurais, aprendizado por reforço e o algoritmo *Q-Learning*. O terceiro Capítulo será uma revisão de trabalhos realizados na área. O quarto Capítulo será a implementação do projeto, com o desenvolvimento de tanto o simulador de corrida quanto da rede neural e o modelo que será treinado. O quinto Capítulo será a apresentação dos resultados do treinamento do agente. O sexto Capítulo será a conclusão.

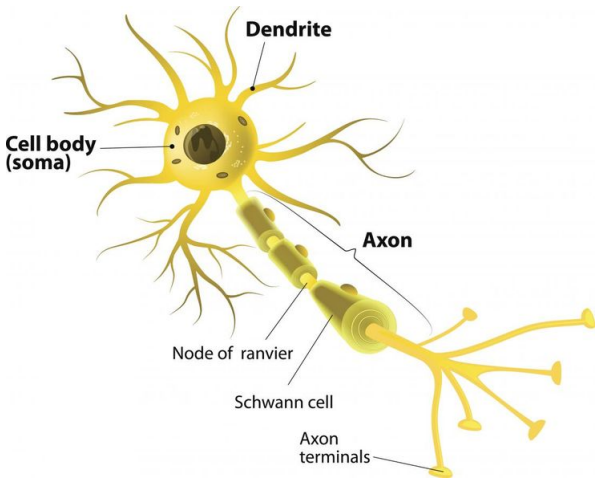
2 APRESENTAÇÃO DE CONCEITOS

Nesse Capítulo será apresentado o que é uma rede neural natural, o que é uma rede neural artificial e como ela adapta as redes neurais naturais, o que é aprendizado por reforço e por fim uma explicação do algoritmo *Deep Q-learning*.

2.1 Neurônios

Neurônios são a base do sistema nervoso, essenciais para a realização das mais básicas ações até as mais complexas. São responsáveis desde o batimento cardíaco e a respiração, até o conhecimento e a personalidade. Os neurônios podem ser classificados em três tipos: os sensoriais, responsáveis por levar as informações recebidas pelos receptores externos e internos para o sistema nervoso central; os motores, responsáveis por levar as informações do sistema nervoso central para os órgãos e músculos; e os de transmissão, responsáveis pela comunicação entre os neurônios sensoriais, motores e o sistema nervoso central. (NEWMAN, 2023)

Figura 1 – Diagrama de um neurônio. (NEWMAN, 2023)



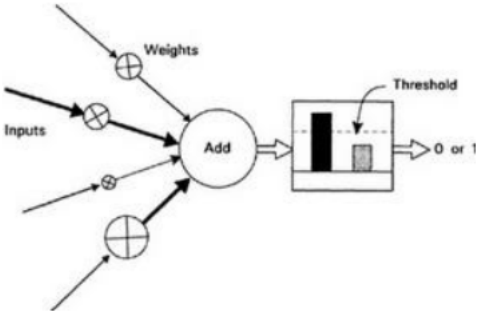
Os neurônios são compostos do corpo celular, que recebe as informações e contêm o núcleo da célula; dos dendritos que recebem sinais químicos ou elétricos pelas junções de comunicação com outros neurônios e carregam essas informações para o corpo celular; e os axônios que carregam as informações do corpo celular para outros neurônios. Os neurônios são conectados por sinapses, que ligam os dendritos e axônios. As sinapses permitem a comunicação entre os neurônios, elas podem ser tanto químicas como elétricas. Neurônios se comunicam a partir de sinais elétricos: quando um neurônio recebe sinais o suficiente para passar um certo limite, o neurônio passará um sinal para os outros neurônios que

estão conectados com ele. A Figura 1 mostra a representação de um neurônio, onde é possível ver os dendritos, o corpo celular e o axônio.

2.2 Redes Neurais Artificiais

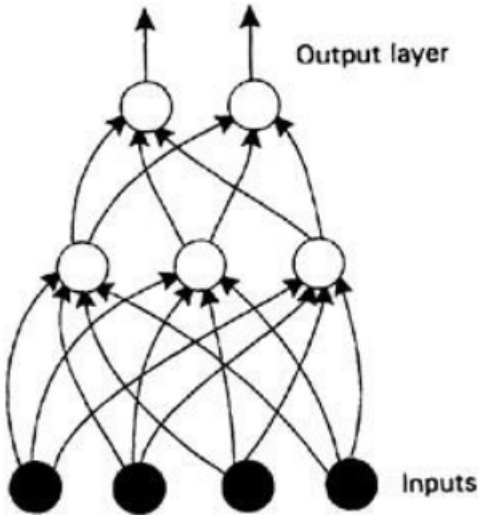
Redes neurais artificiais são modelos computacionais inspirados no funcionamento do cérebro animal. Redes neurais são amplamente utilizadas em uma variedade de aplicações, desde reconhecimento de padrões até visão computacional e processamento de linguagem natural.

Figura 2 – Neurônio artificial simples. (GURNEY, 1997)



Uma rede neural artificial é constituída por uma rede de nodos, conectados por arestas, onde cada aresta possui um peso e cada entrada para um nodo é multiplicado pelo peso da aresta. Depois, o nodo soma todos os valores recebidos pelas arestas e verifica se esse valor excede um certo limite: se não, o nodo passa o valor 0 para os próximos nodos; se sim, ele passa um valor diferente de 0 (geralmente 1) para os próximos nodos. Na Figura 2 é possível ver um nodo simples.

Figura 3 – Exemplo simples de uma rede neural. (GURNEY, 1997)



A rede de nodos pode ser dividida em três camadas diferentes: a camada de entrada, onde toda a informação irá entrar para a rede neural; a camada oculta, onde a entrada será processada para gerar um resultado; e a camada de saída, onde o resultado gerado pela rede será retornado. A Figura 3 mostra o esquema de uma rede neural artificial.

Como as redes neurais artificiais se baseiam no cérebro animal, é possível encontrar vários paralelos entre ambos: os nodos são equivalentes aos neurônios e as arestas são análogas às sinapses. Tanto os neurônios naturais quanto os artificiais se comunicam a partir de sinais que passam a outros neurônios nas respectivas redes. A camada de entrada é correspondente aos neurônios sensoriais, a camada oculta é equivalente aos neurônios de transmissão e a camada de saída representa os neurônios motores.

2.3 Aprendizado por Reforço

A teoria de aprendizado de máquina possui quatro tipos de aprendizagem: o aprendizado supervisionado, semi-supervisionado, não supervisionado e o aprendizado por reforço. O aprendizado por reforço (*Reinforcement Learning*) pode ser definido como um agente (rede neural) que interage com um ambiente a partir de ações e recebe uma recompensa ou punição, baseado no resultado de suas ações. Esse agente então, se adapta para que suas ações produzam mais recompensas e menos punições (GLORENNEC, 2001).

O ciclo de execução de um algoritmo de aprendizagem por reforço pode ser resumido à (GLORENNEC, 2001):

1. Em um determinado tempo t, o agente está no estado $\mathrm{x}(t)$.
2. O agente então escolhe uma das possíveis ações $\mathrm{a}(t)$ dentro deste estado;
3. O agente executa a ação, que provoca:
   - O avanço para um novo estado $\mathrm{x}(t+1)$;
   - O cálculo e recebimento de um reforço $\mathrm{r}(t)$.
4. t é incrementado em 1.
5. Se o estado não for um estado terminal, volta para o passo 2.

Aprendizado por reforço é muito utilizado quando um agente interage com um ambiente em busca de um objetivo e a quantidade de estados possíveis e ações que podem ser realizadas em cada estado dificulta a implementação do aprendizado supervisionado, entre alguns outros pontos.

2.4 Q Learning

O algoritmo *Q-Learning*, proposto por Christopher Watkins em 1989, é um algoritmo de aprendizado (EVEN-DAR; MANSOUR, 2001), (GLORENNEC, 2001), o algoritmo mantém uma tabela chamada *Q-Table* que contém, para cada par de estado-ação possível no ambiente, os *Q-values* que denotam a qualidade da ação em um estado específico, a tabela é inicializada com valores arbitrários ou todos iguais a zero. A *Q-Table* é atualizada após a seleção de uma ação e a observação de uma recompensa com os dados observados. A ideia é ajustar o valor Q do estado atual e a ação escolhida para se aproximar do valor da função ótima Q*, que é obtido através de um processo iterativo (EVEN-DAR; MANSOUR, 2001), (GLORENNEC, 2001).

O algoritmo *Deep Q-Learning* (DQL) é uma extensão do algoritmo *Q-Learning* tradicional que incorpora redes neurais profundas (*Deep Neural Network*) para estimar a *Q-Function*. Ele foi introduzido no artigo "*Human-level control through deep reinforcement learning*"de 2015 (MNIH et al., 2015). O DQL permite lidar com ambientes de estados grandes e complexos, tornando-o adequado para tarefas de aprendizado por reforço em que a representação dos estados não pode ser facilmente armazenada em uma *Q-Table* convencional. O DQL utiliza da rede neural profunda para estimar a *Q-function*, com a entrada da rede sendo o estado atual do ambiente e a saída um conjunto de *Q-Values* para cada ação possível (EVEN-DAR; MANSOUR, 2001), (GLORENNEC, 2001).

Uma das principais características do DQL é o uso de uma memória de *replay*. Em vez de atualizar a rede neural com dados imediatamente após cada ação, as experiências (uma tupla com estado, ação, recompensa e próximo estado) são armazenadas na memória de *replay*. A cada etapa do treinamento, um lote (*batch*) de experiências é amostrado aleatoriamente da memória de replay, o que ajuda a quebrar a correlação temporal entre as experiências e torna o treinamento mais estável (EVEN-DAR; MANSOUR, 2001), (GLORENNEC, 2001).

Durante o treinamento, a rede neural profunda do DQL é atualizada para se ajustar melhor aos *Q-Values* desejados. Isso é feito minimizando a diferença entre os *Q-Values* previstos pela rede e os *Q-Values* reais usando uma função de perda, como o erro quadrático médio (MSE). Os *Q-Values* reais são calculados usando a equação de Bellman:

$$Q(s, a) = r(s, a) + \gamma Q(s', a) \tag{2.1}$$

Onde s é um estado e a é uma ação do estado e s' é o estado após a ação a, Q(s, a) é o *Q-Value* do par s-a, r(s, a) a recompensa da ação a no estado s, γ(gamma) é fator de desconto das recompensas futuras, ou seja, o quanto será considerado as recompensas futuras para a otimização e Q(s', a) *Q-Value* do par s'-a (EVEN-DAR; MANSOUR, 2001), (GLORENNEC, 2001).

O algoritmo do *Deep Q-Learning* é dividido em duas partes, a predição e o treinamento, a predição é onde a rede neural faz escolhas baseados no estado atual e recebe a

recompensa e o novo estado baseado nas escolhas e guarda os dados na memória *replay*, e o treinamento é onde os dados da memória *replay* são usados para treinar a rede neural. O algoritmo é apresentado na Figura 4.

Figura 4 – Algoritmo *Deep Q-Learning*

```
Inicializa memória replay MEMORIA com capacidade N
Inicializa rede neural Q
Inicializa rede neural Q_ALVO
Inicializa epsilon com valor entre 0 e 1
Para episodio = 1 até M faça
    recebe estado inicial s1
    Para passo = 1 até P faça
    #Predição
        aleatorio = número aleatório entre 0 e 1
        Se aleatorio < epsilon então
            a1 = número aleatório
        Senão
            a1 = escolha da rede neural Q baseada no estado s1 Q(s1)
        Executa ação a1 e recebe a recompensa r1 e o estado s2
        guarda transição (s1, a1, r1, s2) em MEMORIA
        s1 = s2

    #Treinamento
        Escolhe um lote aleatório das transições (s1, a1, r1, s2) guardadas em MEMORIA
        Se s2 não é um estado final:
            novo_Q = r1 + gamma*Q_ALVO(s2)
        Senão
            novo_Q = r1

        Calcule a perda e atualize Q

        A cada C passos Q_ALVO = Q
    Fim_Para
Fim_Para
```

3 REVISÃO DE LITERATURA

Nesse Capítulo será apresentado um resumo de trabalhos das áreas de aprendizado por reforço e carros autômnatos estudados durante o projeto.

O artigo "*Human-level control through deep reinforcement learning*" (MNIH et al., 2015) teve como objetivo desenvolver um algoritmo de redes neurais que conseguisse se sobressair em diversas tarefas desafiadoras, para isso foi desenvolvido o algoritmo *Deep Q-Learning*, que utiliza a junção do aprendizado por reforço com redes neurais profundas.

Para verificar o desempenho do algoritmo, ele foi testado em 49 jogos da plataforma ATARI 2600, comparando seus resultados com outros algoritmos de aprendizado por reforço e um profissional em testar jogos, onde apresentou resultados superiores aos outros algoritmos de aprendizado por reforço em quase todos os jogos, e teve um desempenho similar ou superior ao profissional na maioria dos jogos.

No artigo "*Deep Reinforcement Learning for Autonomous Driving*" (WANG; JIA; WENG, 2019) teve como objetivo criar um carro autômato usando o sistema *The Open Racing Car Simulator* (TORCS) como ambiente e o algoritmo *Deep Deterministic Policy Gradient* que foi escolhido por ser baseado em política, o que gera um desempenho melhor em conjuntos de ações contínuos, e por ser determinístico.

Os teste do agente foram feitos usando o modo de competição do TORCS, onde o agente apresentou um desempenho melhor do que seus competidores, principalmente em curvas.

Em "*End-to-End Deep Reinforcement Learning for Lane Keeping Assist*" (SALLAB et al., 2016) o objetivo foi criar um sistema de aprendizado de reforço para assistência de faixa, foram usados dois algoritmos, um algoritmo com saídas determinísticas (DQL) e um com saídas contínuas (*Deep Deterministic Actor Critic* - DDAC).

Os testes foram realizados no sistema TORCS onde ambos foram bem sucedidos em apresentar as funções de assistência de faixa, com o algoritmo DDAC apresentando um caminho mais suave comparado ao algoritmo DQL.

No artigo "Deep Reinforcement Learning framework for Autonomous Driving" (SALLAB et al., 2017) foi proposto uma possível estrutura para modelos de direção automática utilizando redes neurais profundas com os algoritmos Deep Q-Learning e DDAC juntamente com modelos de atenção usados para extrair apenas as informações necessárias dos dados de entrada facilitando seu uso em sistemas embarcados. Para testar a estrutura foi usado o sistema TORCS, onde ambos os algoritmos foram bem sucedidos.

O artigo "*Comparative Study of End-to-end Deep Learning Methods for Self-driving Car*" (YOUSSEF; HOUDA, 2020) foram comparados dois modelos de inteligência artificial para a criação de um sistema de assistência de faixa, o primeiro modelo sendo a

aprendizagem profunda por imitação (IL), o segundo sendo aprendizado profundo por reforço (DRL), para comparar ambos foi usado o simulador CARLA.

Os testes mostraram que o algoritmo baseado em IL possuía um desempenho melhor do que o algoritmo baseado em DRL em percursos que o agente já havia treinado, com o algoritmo baseado em DRL conseguindo se adaptar melhor em percursos novos.

No artigo "*Deep Reinforcement Learning for Simulated Autonomous Vehicle Control*" (YU; PALEFSKY-SMITH; BEDI, 2016) foi implementado e testado um agente usando o algoritmo DQL para controlar um carro simulado no jogo de corrida JavaScript Racer. Ao final do treinamento o agente conseguia percorrer grandes partes da pista sem bater, porém o agente apresentou dificuldade em evitar obstáculos presentes na pista.

Em "*Deep Reinforcement Learning for Autonomous Driving*" (DEVELOPING..., 2022) O objetivo foi criar um sistema de planejamento de trajeto utilizando DQL, usando dados do open CV para treinamento. Para testar o sistema, foram utilizados um conjunto de dados de condução contendo informações sobre o ambiente, os testes demonstraram que o sistema é funcional e eficiente em suas escolhas.

O artigo "*Deep Reinforcement Learning for Autonomous Driving*" (ZHANG; DU; TIAN, 2019) teve como objetivo desenvolver um agente usando o algoritmo Double DQL (DDQL) que foi treinado em um ambiente virtual criado na engine UNITY e testado em um ambiente físico usando um carro de controle remoto modificado para ser controlado por uma placa Raspberry Pi, com o agente treinado no ambiente virtual completando o teste no ambiente físico.

4 Desenvolvimento

Neste Capítulo é detalhada a implementação do projeto, que inclui tanto a parte do simulador, quanto a parte da rede neural. Na Seção 4.1 são abordados os requisitos gerais que o simulador e a rede neural precisam atender e como eles irão se comunicar. A Seção 4.2 aborda como o simulador foi implementado. A Seção 4.3 detalha como a rede neural foi implementada. Por fim, a seção 4.4 cobre o menu usado para iniciar os treinamentos e escolher as pistas

4.1 Requisitos

O simulador de corrida foi desenvolvido com o objetivo de atender à definição dos seguintes requisitos:

- Guardar e desenhar as pistas de corridas que o agente irá percorrer;
- Ter um objeto "carro"que poderá: acelerar, frear e virar tanto para esquerda quanto para a direita;
- O objeto carro deve realizar as ações escolhidas pelo agente de IA;
- Cada pista de corrida terá pontos de controle em seu percurso, tais pontos serão usados para recompensar o agente e para demonstrar o progresso do agente na pista;
- Verificar a colisão entre o carro e a pista;
- Reiniciar a corrida caso o carro colida com a pista;
- Comunicar a posição atual do carro em relação as paredes da pista ao agente;
- Calcular a recompensa baseado no resultado da ação tomada pelo agente.

A rede neural foi definida de forma a atender as seguintes necessidades:

- Receber informações do simulador sobre o estado atual;
- Calcular os comandos para o carro no simulador;
- Receber a recompensa calculada pelo simulador baseado na ação tomada;
- Atualizar a rede neural para maximizar as recompensas e minimizar as punições;
- Salvar a rede neural.

4.2 Simulador de Corrida

Para a implementação do simulador de corrida foi usada a biblioteca *Pygame*, um conjunto de modulos, criados junto à biblioteca SDL para criação de jogos e programas multimídia na linguagem *Python* (PYGAME, 2024).

O simulador de corrida pode ser dividido em três componentes:

- O carro que o agente vai controlar;
- As pistas de corrida que definem o percurso que o agente precisa percorrer com o carro;
- O controlador que vai verificar o resultado da ação tomada pelo agente, verificando a colisão com a pista e calculando as recompensas e punições que serão enviadas para o agente.

4.2.1 Classe Carro

Para desenvolver o carro foi criada uma nova classe, os objetivos dessa classe são: permitir a movimentação e colisão do carro; receber e executar as ações do agente; verificar a posição atual do carro em relação à pista. A Figura 5 mostra o carro.

Figura 5 – Carro

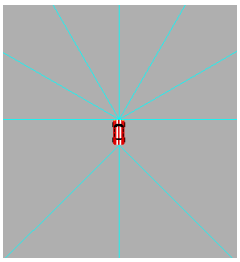
Para movimentação do carro a classe possui parâmetros para guardar a posição, velocidade, ângulo e aceleração. A classe também possui funções para acelerar e desacelerar o carro, rotacionar o carro para a esquerda ou direita e atualizar a posição do carro baseado em sua velocidade e ângulo.

Para a colisão do carro com a pista a classe possui uma lista com quatro pontos que juntos formam um retângulo que envolve o carro, esse retângulo possui a posição e ângulo do carro.

Para executar a ação do agente, a classe possui uma função que recebe uma ação que pode ser: acelerar, desacelerar, rotacionar para esquerda ou rotacionar para direita. Baseado na ação escolhida pelo agente é chamada a função correspondente à acão.

Para verificar a posição atual do carro na pista foi criado uma lista de linhas com o nome de linhas de visão que servem para calcular a distância do carro com as paredes e pontos de controle. As linhas tem início na parte da frente ou trâs carro e seguindo até um ponto a uma determinada distância e ângulo. A Figura 6 mostra essas linhas desenhadas em azul claro.

Figura 6 – Linhas de visão do carro



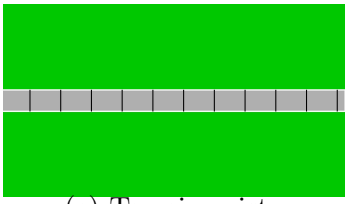
4.2.2 Pistas de corrida

O primeiro passo na criação do simulador foi definir o ambiente que o agente vai interagir, para isso foram criadas pistas de corrida que serviram para limitar o espaço que o agente pode percorrer e definir o caminho a ser percorrido, tendo em vista esses objetivos as pistas foram divididas em duas partes, a primeira parte sendo as paredes que limitam o espaço e a segunda parte sendo os pontos de controle que definem o caminho a ser percorrido.

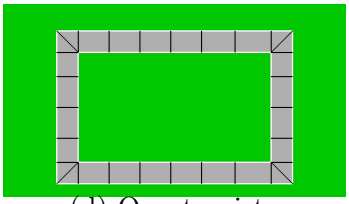
Cada parede é composta de uma lista de pontos que formam um polígono, o espaço entre ambos polígonos é a área que o carro pode percorrer. Os pontos de controle são compostos de dois pontos que quando conectados formam uma linha. A criação das pistas foi feita de forma manual.

Figura 7 – Pistas criadas

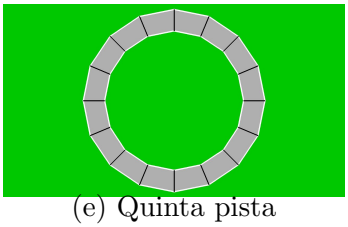
(a) Primeira pista



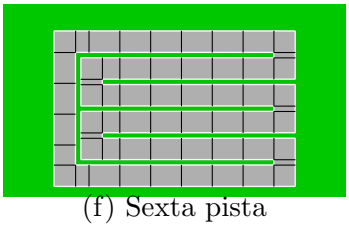
(b) Segunda pista



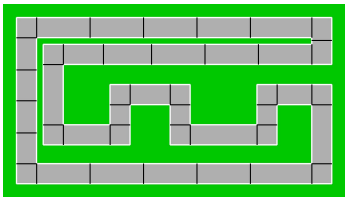
(c) Terceira pista



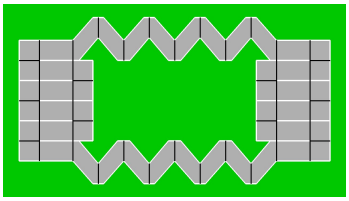
(d) Quarta pista



(e) Quinta pista



(f) Sexta pista



A Figura 7 apresenta todas as pistas criadas. As pistas Nela é possível visualizar as várias partes das pistas: as linhas brancas representam as paredes da pista, as linhas

pretas representam os pontos de controle, a área em cinza é o espaço que o agente pode percorrer e a área em verde representa a área inacessível ao agente.

4.2.3 Controlador do simulador

Para controlar o simulador foi criada uma classe, ela é responsável por fazer a comunicação entre o agente e o carro, verificar e restaurar o progresso do agente na pista, calcular a recompensa do agente baseado no resultado da ação tomada e atualizar a tela durante o treinamento.

A execução do simulador pode ser dividida em oito partes, sendo elas:

1. O simulador envia as informações do estado atual para o agente;
2. O agente escolhe uma ação a partir do estado atual e envia para o simulador;
3. O controlador desenha o estado atual na tela,;
4. O simulador envia a ação do agente para o carro;
5. O carro executa a ação do agente e atualiza o estado;
6. O simulador calcula as colisões e verifica se o carro atravessou o ponto de controle atual e se o novo estado é um estado final;
7. O simulador retorna a recompensa recebida, se o novo estado é final, e a pontuação atual do episódio;
8. O agente recebe o novo estado do simulador.

Uma execução completa de todas as partes é chamada de passo, esses passos serão executados indefinidamente enquanto o simulador estiver treinando o agente.

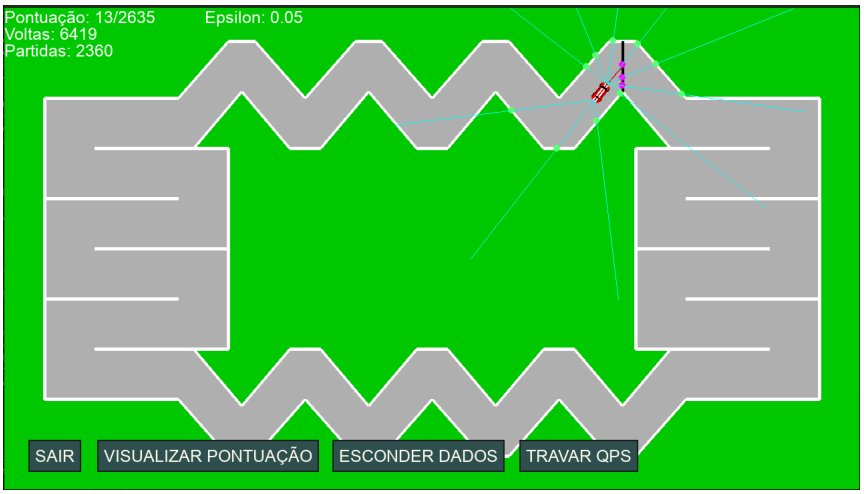
As recompensas calculadas pelo controlador podem ser tanto positivas quanto negativas, o agente recebe uma pequena recompensas positivas quando o carro se aproxima do ponto de controle ou quando o carro atravessa o ponto de controle com o valor variando dependendo do quão perto do meio da pista o carro estava. O agente recebe uma recompensa negativa caso: colida com uma parede da pista, chegue a uma determinada distância das paredes da pista ou se afaste do ponto de controle. A pontuação do agente é a quantidade de pontos de controle que o carro cruzou dentro de um episódio, essa pontuação é usada para visualizar a evolução do carro conforme o treinamento avança.

O simulador pode chegar ao estado final de duas formas, a primeira é uma colisão entre o carro e as paredes da pista, isso gera uma recompensa negativa, a segunda é caso o carro fique um determinado número de passos sem atravessar nenhum ponto de controle, isso não gera recompensa.

Para comunicar o estado atual ao agente o controlador gera uma lista de valores, a lista pode ser dividida em três partes, a primeira parte é a distância entre o ponto inicial das linhas de visão e o ponto de colisão delas com as paredes da pista, a segunda parte é a distância entre o ponto inicial das linhas de visão e o ponto de colisão com o ponto de controle, a última parte é a velocidade atual do carro e se o carro está se aproximando ou afastando do ponto de controle.

O controlador permite duas visualiizações para a tela, a primeira é a visualização da partida onde é possível ver o carro percorrendo a pista com alguns dados sobre a partida atual, a segunda é a visualização do gráfico que demonstra o progresso do agente durante as partidas anteriores.

Figura 8 – Tela de visualização da partida



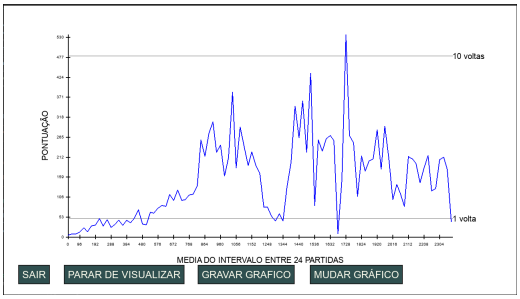
A visualização da partida, vista na Figura 8, nela é desenhado a pista, o carro, o próximo ponto de controle que o carro tem que chegar, as linha de visão do carro e suas colisões, dados sobre a pontuação atual da partida e a pontuação máxima atingida, a quantidade de voltas concluidas ao total durante o treinamento, a quantidade de partidas executadas e o *epsilon*, que será abordado na próxima seção. A tela também possui botões para: sair do treinamento, trocar a visualização para o gráfico, esconder ou mostrar os dados e travar os quadros por segundo da tela que também trava a quantidade de passos por segundo.

A visualização do gráfico possue dois gráficos, ambos podem ser vistos na Figura 9, a tela dos gráficos apresenta o gráfico com o eixo x e y, sendo o eixo x o número da partida e o eixo y a pontuação, a linha em azul é a pontuação das partida, a linha preta cortando o gráfico sinaliza a pontuação de uma volta, aparecendo outras linha para dez voltas, cem voltas e assim por diante, a tela também apresenta botões na parte infeiror, sendo eles: sair do treinamento, mudar a visualização para a partida, gravar uma imagem do gráfico, e mudar o tipo de gráfico.

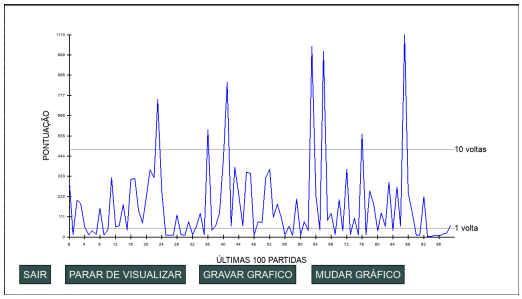
A diferença entre ambos os gráficos são os dados que apresentam, o primeiro gráfico

Figura 9 – Tela de visualização do gráfico

(a) Gráfico com a pontuação média entre determinado número de partida



(b) Gráfico com a pontuação das últimas 100 partidas



divide as partidas em grupos e calcula a média desse grupos, essa média é então colocada no gráfico. O segundo gráfico apresenta a pontuação das últimas cem partidas realizada.

4.3 Aplicação da rede neural

Para implementação da rede neural foi usada a biblioteca *Pytorch*, A aplicação da rede neural foi divida em três partes:

- Implementação do modelo;
- Implementação do agente;
- Implementação do laço de execução.

4.3.1 Modelo

Uma classe foi criada para o modelo, ela herda da classe *Module* da biblioteca *Pytorch*. O modelo é dividido em uma camada de entrada, uma camada de saída e uma camada oculta conectando ambas. Para fazer a predição a entrada começa na camada de entrada, passa pela camada oculta e termina na camada de saída. O modelo também possui funções para salvar e carregar os pesos do modelo a partir de um arquivo.

4.3.2 Agente

O agente criado é uma implementação de *Deep Q-Learning* como visto na seção 2.4. Para implementar o agente foi criada uma nova classe que possui: o valor de *epsilon*, o valor de gamma, uma memória de *replay*, uma rede alvo, taxa de aprendizado, um otimizador e o critério usado.

A classe pode: Receber informações sobre o estado atual do simulador; escolher uma ação baseado nas informações recebidas; guardar os dados na memória *replay* que é composta por: estado inicial do passo, a ação tomada, a recompensa, o próximo estado do passo e caso o estado é final; e treinar o agente baseado nos dados salvos.

A variável *epsilon* da classe representa uma porcentagem que é a chance da escolha do agente ser aleatória ao invés de ser baseado em seu conhecimento, a funcionalidade de tal variável está ligada ao dilema *exploration-exploitation* (exploração-exploração), esse dilema envolve a decisão entre a ação que o agente acredita ser a melhor com base em seu conhecimento (*exploitation*), ou uma ação aleatória que possa levar a novos estados e situações (*exploration*) (GLORENNEC, 2001).

No começo, o agente não possui conhecimento sobre os estados, ações e possíveis recompensas dentro do ambiente então a exploração (*exploration*) se torna muito importante para que o agente aprenda, porém com o tempo, o agente começa a ter um entendimento melhor sobre seu ambiente e pode começar a explorar(*exploitation*) esse conhecimento para maximizar as recompensa, mas, mesmo com o agente tendo conhecimento sobre o ambiente a exploração (*exploration*) ainda tem o papel importante de explorar ações que possam levar a um ganho maior do que a ação escolhida pelo agente (GLORENNEC, 2001).

A variiável *epsilon* tem como objetivo dividir a fase de exploração (*exploration*) e a fase de exploração(*exploitation*), essa variável começa em um valor inicial e, a cada volta completada pelo agente, seu valor é multiplicado por uma taxa de decaimento, esse processo é repetido até que o valor de *epsilon* chegue a um valor minímo.

Para tornar o treinamento mais estável, uma segunda rede neural chamada "*Target Network*"ou rede alvo é usada. A cada poucos passos de treinamento, a rede alvo é copiada da rede principal. Isso ajuda a evitar oscilações nas estimativas de Q durante o treinamento. Com a rede treinada, ela pode ser usada para escolher ações, prevendo o *Q-Value* baseado no conhecimento adquirido durante o treinamento (EVEN-DAR; MANSOUR, 2001), (GLORENNEC, 2001).

Para atualizar a rede neural são usados um critério e um otimizador, o critério utiliza uma função de perda para calcular o valor da perda, que é a diferença entre o resultado predito pelo agente e o resultado esperado. O otimizador tem o objetivo de atualizar os pesos da rede neural baseado no valor da perda, o quanto os valores dos pesos são atualizados pelo otimizador é determinado por uma variável chamada taxa de aprendizagem.

Para treinar o agente, primeiro é feito uma predição dos *Q-Values* dos estados iniciais do lote usando a rede alvo, a predição dos *Q-Values* reais é feita usando a equação de Bellman (equação 2.1). Com ambos os *Q-Values* é então calculada a perda usando o erro quadrático médio como função de perda e o modelo é otimizado usando o otimizador Adam da biblioteca *Pytorch*.

4.3.3 Ciclo de execução

Juntando tanto a execução do simulador quanto o treinamento do agente, o ciclo de execução de um passo do projeto pode ser resumido à:

1. O agente recebe o estado atual do simulador;
2. O agente escolhe uma ação a partir do estado atual;
3. O simulador recebe a ação, atualiza o estado e retorna a recompensa, se está em um estado final e a pontuação atual;
4. O agente recebe o novo estado do simulador;
5. O agente é treinado com as experiências obtidas;
6. Caso o novo estado seja um estado final então:
    a) Incrementa o número de episódios do agente;
    b) A rede alvo é atualizada conforme o intevalo de atualização específicado;
    c) Atualiza o *epsilon* multiplicando o *epsilon* atual com sua taxa de decaimento até que chege no valor minímo;
    d) Treina o agente com um lote aleatório da memória de *replay*;
    e) Salva os pesos da rede neural do agente;
    f) Guarda a pontuação do episódio e a pontuação média nas lista;
    g) Salva a pontuação e pontuação média do episódio nos arquivos;
    h) Cria um gráfico com as pontuações e pontuações médias de todos os episódios executados;

4.4 Menu

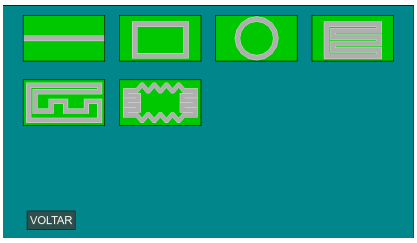
Para facilitar a escolha das pistas e configuração dos modelos foi criado um menu, ele é dividido em telas, cada tela tem uma funcionalidade como: escolher a pista que o agente irá treinar, escolher as configurações básicas do agente, importar agentes de outras pistas e escolher entre os agentes treinados na mesma pista.

A Figura 10 mostra todas as telas do menu, na primeira tela é possível escolher qual pista será realizado o treinamento, selecionada uma pista o menu vai para a segunda tela. Na segunda tela é possível selecionar entre continuar treinando o último modelo treinado na pista selecionada que leva para o simulador, carregar qualquer modelo treinado na pista que leva a terceira tela, ou criar um novo modelo que leva à quarta tela.

A terceira tela permite selecionar qualquer modelo já treinado em determinada pista, ao ser selecionado um modelo a tela muda para o simulador. A quarta tela permite selecionar entre criar um novo modelo, que leva à sexta tela, ou carregar um modelo de outra pista, que leva à quinta tela. Na quinta tela é apresentado uma lista com as pistas que possuem modelos treinados, ao escolher uma pista a tela passa a mostrar todos os

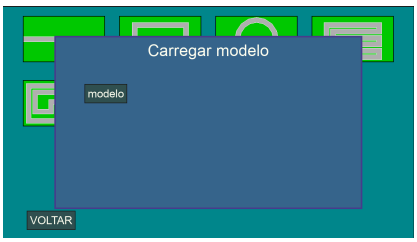
Figura 10 – Telas do menu

(a) Escolha da pista para treinamento



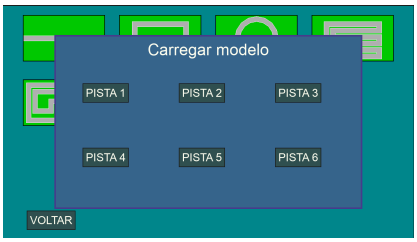
(b) Continuar, criar ou carregar o modelo

(c) Carregar modelo da pista

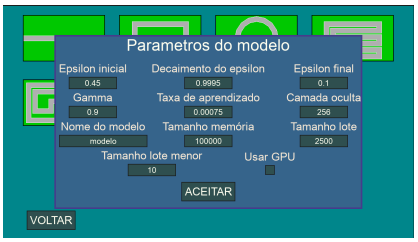


(d) Criar modelo ou carregar de outra pista

(e) Lista de pistas com modelos



(f) Configuração do modelo



modelos treinados da pista e ao escolher um dos modelos a tela transiciona para a sexta tela.

A sexta tela permite configurar um novo modelo, nela é possível expecificar: os valores do *epsilon* inicial, o decaimento do *epsilon* e o *epsilon* final; o valor de *gamma*; a taxa de aprendizado; o tamanho da camada oculta; o nome do modelo; o tamanho da memória *replay*; a quantidade de dados da memória usados no treinamento ao final de uma partida; a quantidade de dados da memória usados no treinamento ao final de um passo; e se o treinamento da rede neural será feito usando a GPU ou não.

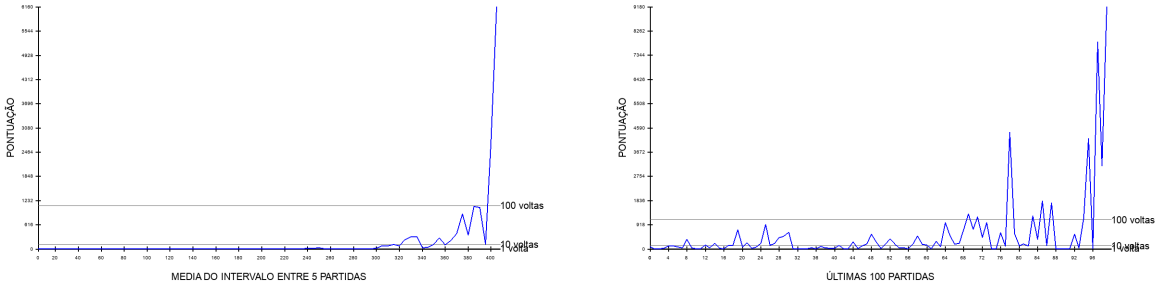
5 Resultados

Nesse Capítulo serão abordados os resultados do treinamento do agente nas seis pistas, primeiramente as configurações e resultados do treinamento serão mostrados na seção 5.1.

5.1 Treinamentos

O treinamento do agente foi feito começando das pistas mais simples e avançando até as mais complexas, com o treinamento de uma pista completo, o modelo do agente foi carregado para a próxima pista. A seguir serão apresentados os resultados dos treinamentos nas pistas.

Figura 11 – Gáficos da primeira pista



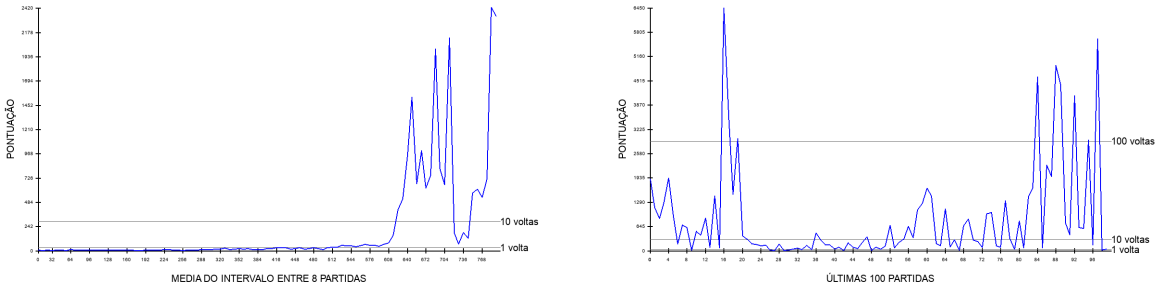
A Figura 11 mostra os gráficos do progresso do agente na primeira pista durante o treinamento. As configurações usadas para o treinamento foram:

- *epsilon* inicial: 0,45
- decaimento do *epsilon*: 0,999
- *epsilon* minímo: 0,1
- *gamma*: 0,9
- taxa de aprendizado: 0,00075
- tamanho da camada oculta: 256
- Tamanho da memória *replay*: 100.000
- Tamanho do lote da partida: 2.500
- Tamanho do lote do passo: 10

O agente realizou 5.110 voltas em 405 partidas com uma média de 12 voltas por partida, a maior pontuação atingida foi 9.183 (834 voltas completas na pista).

No primeiro gráfico é possível ver a progressão do agente durante todo o treinamento, no começo o agente completava entre 1 e 10 voltas antes de terminar a partida, no final do treinamento o agente completava entre 10 e 100 voltas, chegando a 800 no final. No segundo gráfico é possível verificar uma variação no desempenho do agente nas últimas 100 partidas realizadas, essa variação é causada pela atualização da rede neural do agente.

Figura 12 – Gáficos da segunda pista



A Figura 12 mostra os gráficos do progresso do agente na segunda pista durante o treinamento. As configurações usadas para o treinamento foram:

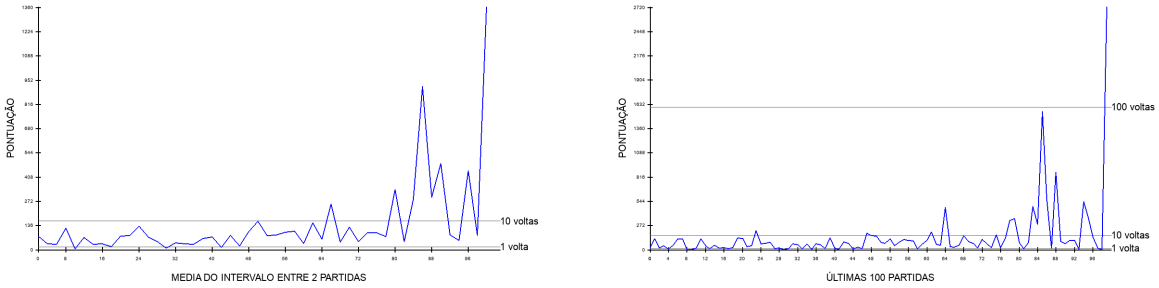
- *epsilon* inicial: 0,45
- decaimento do *epsilon*: 0,9992
- *epsilon* minímo: 0,05
- *gamma*: 0,9
- taxa de aprendizado: 0,00075
- tamanho da camada oculta: 256
- Tamanho da memória *replay*: 100.000
- Tamanho do lote da partida: 2.500
- Tamanho do lote do passo: 10

O decaimento do *epsilon* foi aumentado para que o agente possa esplorar mais escolhas aleatóriias na pista, e o *epsilon* final foi diminuido para o agente ter mais controle sobre as ações do carro após o *epsilon* chegar em seu valor minímo.

O agente realizou 5.491 voltas em 796 partidas com uma média de 6 voltas por partida, a maior pontuação atingida foi 8.694 (299 voltas completas na pista).

No primeiro gráfico é possível ver a progressão do agente durante todo o treinamento, no começo o agente completava em média 1 volta antes de terminar a partida, no final do treinamento o agente completava entre 10 e 80 voltas, com uma queda causada pelo treinamento do agente, logo após voltando à ter um bom desempenho . No segundo gráfico é possível verificar uma variação no desempenho do agente nas últimas 100 partidas realizadas.

Figura 13 – Gáficos da terceira pista



A Figura 13 mostra os gráficos do progresso do agente na terceira pista durante o treinamento. As configurações usadas para o treinamento foram:

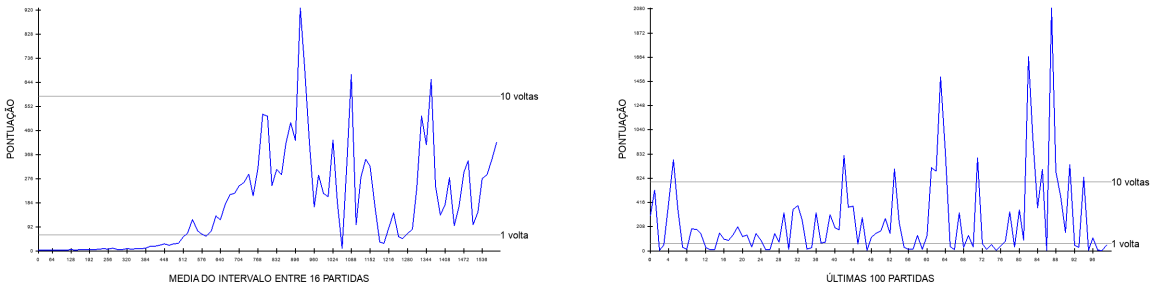
- *epsilon* inicial: 0,45
- decaimento do *epsilon*: 0,9992
- *epsilon* minímo: 0,05
- *gamma*: 0,9
- taxa de aprendizado: 0,00075
- tamanho da camada oculta: 256
- Tamanho da memória *replay*: 100.000
- Tamanho do lote da partida: 2.500
- Tamanho do lote do passo: 10

O agente realizou 1.030 voltas em 101 partidas com uma média de 10 voltas por partida, a maior pontuação atingida foi 2.726 (170 voltas completas na pista).

No primeiro gráfico é possível ver a progressão do agente durante todo o treinamento, no começo o agente completava em média de 1 a 10 voltas antes de terminar a partida, no final do treinamento o agente completava entre 5 e 50 voltas. No segundo gráfico é possível verificar uma variação no desempenho do agente nas últimas 100 partidas realizadas.

A Figura 14 mostra os gráficos do progresso do agente na quarta pista durante a partida. As configurações usadas para o treinamento foram:

Figura 14 – Gáficos da quarta pista



- *epsilon* inicial: 0,45
- decaimento do *epsilon*: 0,9993
- *epsilon* minímo: 0,05
- *gamma*: 0,9
- taxa de aprendizado: 0,00065
- tamanho da camada oculta: 256
- Tamanho da memória *replay*: 100.000
- Tamanho do lote da partida: 2.500
- Tamanho do lote do passo: 10

O valor do decaimento do *epsilon* foi novamente aumentado. O valor da taxa de aprendizado foi diminuído limitar o impacto de cada dado do lote de treinamento nos pesos da rede neural.

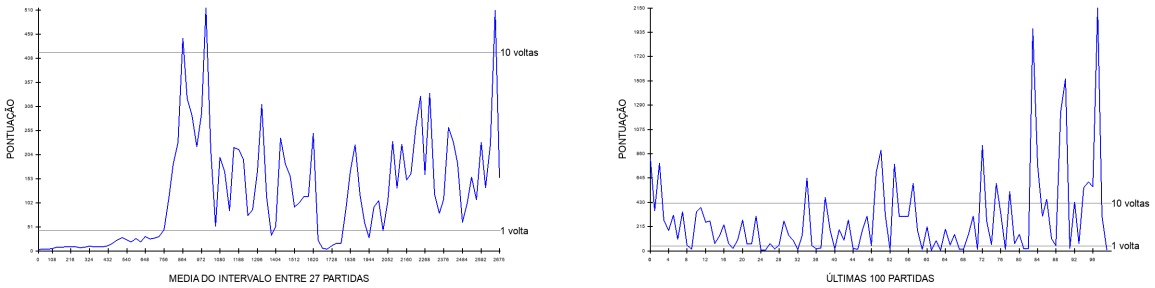
O agente realizou 4.183 voltas em 1.582 partidas com uma média de 2 voltas por partida, a maior pontuação atingida foi 3.176 (53 voltas completas na pista).

No primeiro gráfico é possível ver a progressão do agente durante todo o treinamento, no começo o agente não conseguia completar nenhuma volta antes de terminar a partida, na metade do treinamento ele teve um aumento de desempenho completando entre 1 e 10 voltas, no final do treinamento o agente completava em média 5 voltas. No segundo gráfico é possível verificar uma variação no desempenho do agente nas últimas 100 partidas realizadas.

A Figura 15 mostra os gráficos do progresso do agente na quinta pista durante a partida. As configurações usadas para o treinamento foram:

- *epsilon* inicial: 0,4
- decaimento do *epsilon*: 0,9993

Figura 15 – Gáficos da quinta pista



- *epsilon* minímo: 0,05
- *gamma*: 0,9
- taxa de aprendizado: 0,00065
- tamanho da camada oculta: 256
- Tamanho da memória *replay*: 100.000
- Tamanho do lote da partida: 2.500
- Tamanho do lote do passo: 15

O *epsilon* inicial foi diminuido para que o agente tenha mais controle no começo do treinamento. O tamanho do lote do passo foi aumentado para que cada treinamento utilize mais dados da memória *replay*.

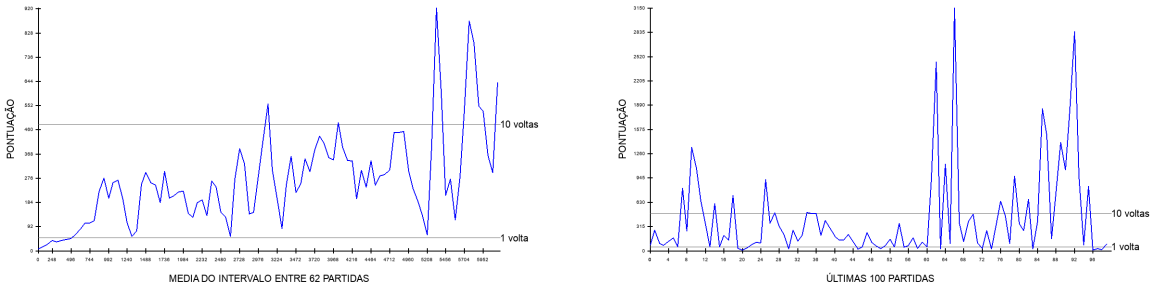
O agente realizou 6.852 voltas em 2.677 partidas com uma média de 2 voltas por partida, a maior pontuação atingida foi 2.362 (56 voltas completas na pista).

No primeiro gráfico é possível ver a progressão do agente durante todo o treinamento, no começo o agente não conseguia completar nenhuma volta antes de terminar a partida, tendo uma melhora no desempenho e conseguindo completar de 1 a 10 voltas, no final do treinamento o agente completava em média 5 voltas. No segundo gráfico é possível verificar uma variação no desempenho do agente nas últimas 100 partidas realizadas.

A Figura 16 mostra os gráficos do progresso do agente na sexta pista durante a partida. As configurações usadas para o treinamento foram:

- *epsilon* inicial: 0,4
- decaimento do *epsilon*: 0,9993
- *epsilon* minímo: 0,05
- *gamma*: 0,9

Figura 16 – Gáficos da sexta pista



- taxa de aprendizado: 0,00065
- tamanho da camada oculta: 256
- Tamanho da memória *replay*: 100.000
- Tamanho do lote da partida: 2.500
- Tamanho do lote do passo: 15

O agente realizou 31.361 voltas em 6.114 partidas com uma média de 5 voltas por partida, a maior pontuação atingida foi 6.554 (136 voltas completas na pista).

No primeiro gráfico é possível ver a progressão do agente durante todo o treinamento, no começo o agente conseguia completar cerca de 5 voltas ntes de terminar a partida, mantendo seu desempenho no meio do treinamento, no final do treinamento o agente completava entre 5 e 15 voltas. No segundo gráfico é possível verificar uma variação no desempenho do agente nas últimas 100 partidas realizadas.

6 Conclusão

O objetivo geral deste projeto foi desenvolver um sistema de IA usando redes neurais artificiais e aprendizado de máquina. Esse sistema foi usado para a aprendizagem de um agente que consiga dirigir um carro para percorrer as pistas criadas. O objetivo geral foi dividido em cinco objetivos específicos:

O1 Desenvolver um simulador de corrida simples;

O2 Criar um conjunto de pistas que a IA terá que percorrer;

O3 Desenvolver uma IA usando redes neurais artificiais;

O4 Treinar um agente para percorrer as pistas criadas;

O5 O agente deve conseguir completar de forma consistente voltas pela pista sem que colida com as paredes.

O projeto começou com um estudo de conceitos sobre redes neurais, aprendizado por reforço e *Deep Q-Learning*, o estudo permitiu um maior entendimento sobre como o projeto poderia ser implementado. O desenvolvimento do projeto criou um entendimento mais profundo sobre a implementação de *machine learning* e redes neurais, e como integrá-las com outro sistemas. Com o estudo dos conceitos feito foi feita então a implementação do projeto.

A implementação teve inicio pelo simulador que foi criado usando a biblioteca *Pygame*. A rede neural foi então implementada usando a biblioteca *Pytorch*, tendo como base o algoritmo *Deep Q-Learning*, um algoritmo de aprendizado por reforço. Com a implementação do projeto concluída foram realizados os treinamentos do agente nas pistas criadas.

Com os treinamentos finalizados e os dados resultantes apresentados, é possível verificar que todos os treinamentos consistentemente conseguiram completar voltas nas pistas sem que falhasem por colidir com uma parede ou por não chegarem no próximo ponto de controle dentro do tempo limite, em certos casos podendo completar mais de cem voltas antes de falhar. Em todos os treinamentos o agente completou em média duas ou mais voltas por partida.

Tendo em vista estes resultados é possível concluir que o treinamento foi um sucesso, alcançando o objetido do projeto.

Melhorias possíveis a esse projeto seriam a adição de novas pistas de corrida com formatos diferentes, a criação de obstáculos na pista, a implementação de paralelismo para o treinamento de vários agentes simultâneamente e a implementação do simulador em um ambiente tridimensional.

Referências bibliográficas

DEVELOPING a Deep Q-Learning and Neural Network Framework for Trajectory Planning. 2022. 16

EVEN-DAR, E.; MANSOUR, Y. Learning rates for q-learning. In: HELMBOLD, D.; WILLIAMSON, B. (Ed.). *Computational Learning Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. ISBN 978-3-540-44581-4. 13, 23

GLORENNEC, P. Reinforcement learning: an overview. 04 2001. 12, 13, 23

GURNEY, K. *An Introduction to Neural Networks*. USA: Taylor & Francis, Inc., 1997. ISBN 1857286731. 7, 11

MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, v. 518, p. 529–533, 2015. 13, 15

NEWMAN, T. *All you need to know about neurons*. 2023. ⟨https://www.medicalnewstoday.com/articles/320289⟩. Acesso em: 7 de ago. 2024. 7, 10

PYGAME. 2024. ⟨https://www.pygame.org/wiki/about⟩. Acesso em: 7 de ago. 2024. 18

SALLAB, A. E. et al. *End-to-End Deep Reinforcement Learning for Lane Keeping Assist*. 2016. 15

SALLAB, A. E. et al. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, Society for Imaging Science & Technology, p. 70–76, jan. 2017. 15

WANG, S.; JIA, D.; WENG, X. *Deep Reinforcement Learning for Autonomous Driving*. 2019. 15

YOUSSEF, F.; HOUDA, B. M. Comparative study of end-to-end deep learning methods for self-driving car. *International Journal of Intelligent Systems and Applications*, 2020. 15

YU, A.; PALEFSKY-SMITH, R.; BEDI, R. *Deep Reinforcement Learning for Simulated Autonomous Vehicle Control*. 2016. 16

ZHANG, Q.; DU, T.; TIAN, C. *Self-driving scale car trained by Deep reinforcement learning*. 2019. 16