UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOÃO VITOR BRUNIERA LABRES

ANÁLISE DE VIABILIDADE DE UTILIZAÇÃO DE DEEP Q-LEARNING PARA CRIAÇÃO DE UM CONTROLADOR NO TORCS

CHAPECÓ
2019

JOÃO VITOR BRUNIERA LABRES

ANÁLISE DE VIABILIDADE DE UTILIZAÇÃO DE DEEP Q-LEARNING PARA CRIAÇÃO DE UM CONTROLADOR NO TORCS

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Dr. Fernando Bevilacqua

CHAPECÓ
2019

Labres, João Vitor Bruniera

Análise de viabilidade de utilização de Deep Q-Learning para criação de um controlador no TORCS / João Vitor Bruniera Labres. – 2019.
52 f.: il.

Orientador: Dr. Fernando Bevilacqua.
Trabalho de conclusão de curso (graduação) – Universidade Federal da Fronteira Sul, curso de Ciência da Computação, Chapecó, SC, 2019.

1. INTELIGÊNCIA ARTIFICIAL. 2. DEEP REIFORCEMENT LEARNING. 3. JOGOS. 4. TORCS. I. Bevilacqua, Dr. Fernando, orientador. II. Universidade Federal da Fronteira Sul. III. Título.

E-mail: joao.labres@estudante.uffs.edu.br

JOÃO VITOR BRUNIERA LABRES

ANÁLISE DE VIABILIDADE DE UTILIZAÇÃO DE DEEP Q-LEARNING PARA CRIAÇÃO DE UM CONTROLADOR NO TORCS

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Dr. Fernando Bevilacqua

Aprovado em: 04/12/2019.

BANCA AVALIADORA

Dr. Fernando Bevilacqua – UFFS

Dr. Denio Duarte - UFFS

Dr. Giancarlo Dondoni Salton - UFFS

AGRADECIMENTOS

Agradeço primeiramente à Deus por me conceder saúde, força e coragem para chegar ao final deste trabalho. Aos meus pais e familiares por todo o apoio durante esta caminhada. Aos meus amigos que estiveram ao meu lado, me ajudando e oferecendo o café nosso de cada dia.

Agradeço ao meu orientador Dr. Fernando Bevilacqua por todas as revisões, conversas e orientações que tivemos durante o período de trabalho, não somente referentes ao trabalho, mas também para a vida. Também aos membros das bancas que muito contribuíram para a execução e aprimoramento deste trabalho: Me. Adriano Padilha, Dr. Denio Duarte e Dr. Giancarlo Salton.

“Don’t Panic”

(Douglas Adams, The Hitchhiker’s Guide to the Galaxy)

RESUMO

Visando o crescimento das indústrias de jogos e de carros autônomos, a análise de viabilidade de um algoritmo de inteligência artificial tem um papel fundamental para o desenvolvimento de novos jogos e carros. O presente trabalho analisa a viabilidade de utilização de *Deep Q-Learning* utilizando o TORCS, um simulador de corridas de carros de código aberto. A alteração de uma função de recompensa foi testada durante o desenvolvimento e apresentou um comportamento promissor para o início do treinamento. Para isso, são propostos testes de eficiência e aprendizado. Não foi possível chegar a um controlador completo, utilizando-se somente *Deep Q-Learning*, por tanto, não sendo viável a utilização. Uma discussão é feita sobre os resultados alcançados e as propostas de alterações são compatíveis com modelos já utilizado em outros trabalhos.

Palavras-chave: INTELIGÊNCIA ARTIFICIAL. DEEP REIFORCEMENT LEARNING. JOGOS. TORCS.

ABSTRACT

Aiming at the growth of the autonomous car and game industries, the feasibility analysis of an artificial intelligence algorithm plays a key role in the development of new games and cars. This paper analyzes the feasibility of using Deep Q-Learning using TORCS, an open source car racing simulator. Changing a reward function was tested during development and showed promising behavior for the start of training. For this, efficiency and learning tests are proposed. It was not possible to reach a complete controller using only *Deep Q-Learning*, so it was not feasible to use. A discussion is made about the results achieved and the proposed changes are compatible with models already used in other works.

Keywords: ARTIFICIAL INTELLIGENCE. DEEP REINFORCEMENT LEARNING. GAMES. TORCS.

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O mundo dos jogos digitais vem crescendo a cada ano e junto é necessário desenvolver novas tecnologias para eles. Contudo, as tecnologias também passam por uma evolução constante que torna o processo custoso para as indústrias acompanharem. Novas soluções precisam ser estudas antes de serem aplicadas, o que é, muitas vezes, caro para uma empresa de desenvolvimento de jogos.

Alguns jogos são focados na diversão dos jogadores, negligenciando o realismo das mecânicas, outros passam a ser simuladores realistas. Para a diversão é necessário manter um jogador cada vez mais entretido dentro do jogo, criando novas formas de jogar. Os simuladores buscam cada vez mais realismo, tanto em gráficos ou físicas parecidas com as encontradas no mundo real quanto em simulação da inteligência artificial (IA). Um exemplo de simuladores são os de corridas de carros. Os jogadores, muitas vezes, não possuem recursos para correr em carros reais, mas gostam de sentirem-se como se estivessem pilotando automóveis de corrida.

Com base nas pesquisas realizadas, um dos simuladores mais utilizados é o TORCS[1], uma ferramenta de código aberto que permite testes em computadores com baixo poder computacional, tornando-se uma solução que pode ser considerada estado da arte. Os controladores, carros autônomos com os quais o jogador compete, vêm sendo desenvolvidos em diversos trabalhos acadêmicos. Alguns utilizam-se de uma máquina finita de estados (FSM) e outros de técnicas de IA. Porém, uma das sub-áreas da IA pouco explorada nesse contexto é a de *Deep Reinforcemente Learning*. Desenvolvido em 2013 por Mnih et al. (28), ele se tornou a base para o AlphaZero[2] que vem melhorando e sobrepujando outras ferramentas.

Já existem trabalhos que realizam pesquisas para jogos de corrida (Sallab et al. (32), Ganesh et al. (14), Chen et al. (9) e Munoz; Gutierrez; Sanchis (29)) e outros que somente utilizam simuladores como cama de testes para seus algoritmos (Mnih et al. (27), Lillicrap et al. (20) e Koutnı'k et al. (18)). Este trabalho busca ser uma referência para futuros trabalhos acadêmicos que possam utilizar os resultados como base para novos métodos, e também ser uma contribuição para empresas que possuem menos condições de conduzir investigações mais detalhadas para suas aplicações. Finalmente, pode contribuir com o desenvolvimento de carros autônomos para corridas.

Dentro do contexto de jogos digitais e inteligência artificial, este trabalho vem a somar na investigação de quais técnicas podem resultar em um melhor controlador. Percebendo as dificuldades encontradas em trabalhos relacionados, como validar o aprendizado e treinar com mais de um carro na pista, este foca somente em controlar o carro de forma autônoma e sem outros competidores na pista. Este comportamento simula o modo de classificação, onde o carro tenta completar o circuito no menor tempo possível. Já existe pelo menos uma empresa que

[1] http://torcs.sourceforge.net/

[2] https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games%2Dchess-shogi-and-go

desenvolve carros autônomos de corrida[3], por isso pode-se ver a colocação desse trabalho em um âmbito acadêmico e comercial.

Os resultados alcançados neste trabalho apontam para uma inviabilidade de utilização do *deep Q-Learning* como uma forma de controle completo de um controlador. Toda via, utilizando uma função de recompensa modificada pelos autores, o algoritmo obteve bons tempos de voltas quando comparado com outros trabalhos e também um aprendizado consistente na condução do carro em um cenário simplificado.

1.1 PROBLEMATIZAÇÃO

Ao se observar o mundo dos jogos, pode-se notar uma grande variedade de jogos digitais disponíveis. Uma das áreas é a de corridas, mais especificamente corridas de carros. Estes jogos necessitam de uma simulação muito parecida com o mundo real. Porém, para simular ainda corrida real, é necessário desenvolver controladores para correr contra o jogador. Para descobrir qual é a melhor forma de desenvolver um controlador, faz-se necessário elaborar vários testes com diversos algoritmos.

Tendo em vista como humanos aprendem a dirigir, treinando em diversos cenários diferentes, é possível perceber que o desenho da pista também influencia o aprendizado da máquina. Os treinamentos realizados precisam abranger a maior quantidade possível de cenários, porém, específicos em alguns aspectos. Muitas curvas ou retas, por exemplo, modificam a forma que o carro deve ser pilotado, alterando a velocidade e o ângulo do veículo. Conforme ilustrado na Figura 1, existem várias situações que podem modificar o comportamento de um jogador e de um controlador. Se treinada somente em uma pista com muitas retas, Figura 1 (a), o controlador tende a não saber fazer curvas da maneira apropriada. Se executar os treinos em uma pista oval, apresentado na Figura 1 (c), ele aprende a fazer curvas, porém, pode aprender somente para um lado. Então, pode ser interessante utilizar uma pista parecida com a Figura 1 (b), que possui retas e também curvas para ambos os lados e com velocidades altas e baixas.

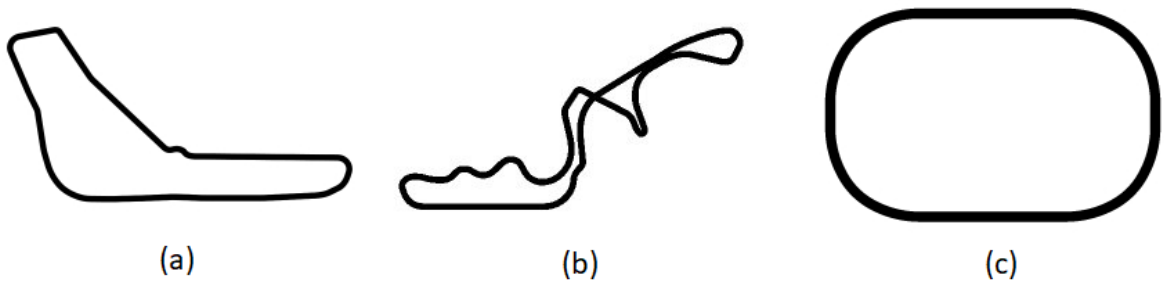


Figura 1 – Exemplificação, com 3 pistas, das várias possibilidades de desenhos de traçado encontrados no TORCS - (a) Forza, (b) Wheel 2 e (c) F-Speedway

[3] https://roborace.com/

Desenvolver diversos treinos com variáveis diferentes em algoritmos diferentes é custoso para empresas de jogos. Se faz necessário, então, o aprofundamento dos testes em algorítimos específicos. Estes testes aprofundados auxiliam tanto uma empresa de jogos a escolher qual a melhor opção dentro de um contexto, quanto a cientistas procurarem novas formas de solucionar problemas utilizando algoritmos poucos utilizados. Ainda, pode-se estender a utilização de trabalhos relacionados a pesquisas em carros autônomos de corrida. Em buscas realizadas, pode-se constatar uma falta de pesquisas específicas para *Deep Reinforcemente Learning* aplicado a jogos de corrida.

Como realizar testes em ambientes reais demandam recursos como carro, pista e dinheiro, uma saída é a utilização de simulações. Isso é apresentado em um vídeo da Roborace[4] sobre simulação de um carro para corridas autônomas. Em outros vídeos, pode-se perceber que ainda é necessário evoluir os testes, pois ainda acontecem acidentes, sem vítimas humanas.

[4] `https://www.youtube.com/watch?v=d1nLdsC2cK4` - acesso em: 14 junho 2019

2 OBJETIVOS

2.1 OBJETIVO GERAL

Avaliação de viabilidade da utilização do *Deep Q-Learning*, quando aplicado ao TORCS, um jogo de simulação de corrida de carros.

2.2 OBJETIVOS ESPECÍFICOS

- Identificar e avaliar dados disponíveis no simulador que possam servir como entrada para o treinamento da rede neural;
- Analisar e encontrar um conjunto de configurações disponíveis no TORCS, que possam ser utilizadas para desenvolver o treinamento da inteligência artificial;
- Realizar o treinamento de redes neurais com os parâmetros encontrados para comparar com outros trabalhos.
- Identificar os parâmetros que sirvam de *baseline* para comparar o método utilizado neste trabalho com outros trabalho.

2.3 JUSTIFICATIVA

No cenário em que empresas de jogos estão competindo para conseguir lançar um produto que venda mais que o da concorrência, o desenvolvimento de pequenos módulos se torna cada vez mais importante. Visando o crescimento de seus jogos, empresas com menores condições monetárias acabam não tendo recursos suficientes para investirem em pesquisa e desenvolvimento. Isso porque precisam destinar tempo à arte do jogo e em pesquisas sobre qual método pode trazer melhores resultados para aquela aplicação específica, por exemplo. A falta deste investimento pode interferir diretamente na imersão e na jogabilidade dos usuários.

Um robô que sempre perde pode fazer com que os jogadores fiquem entediados pela falta de desafio, porém um que ganha todas as corridas pode deixar os jogadores irritados com o excesso de desafio, como mostra Tognetti et al. (37). Uma das teorias que modela o engajamento e emoções dos usuários é conhecida como *flow* ilustrado na Figura 2. Essa expressão foi denominada por Csikszentmihalyi (10) e refere-se à ideia de manter a pessoa engajada no jogo por mais tempo, causando a perda da noção do tempo, por exemplo. Uma das formas é equilibrando o nível de habilidade do jogador com o nível de desafio.

Em pesquisas realizadas, encontrou-se uma quantia consideravelmente baixa de 24 artigos com os termos *"car race" deep reinforcement learning q-learning* e outros 86 artigos com os termos *"TORCS" AND "deep q-learning"*. As *strings* de busca referem-se, respectivamente, a um controle geral de carros de corrida utilizando *deep Q-Learning* (DQL) como

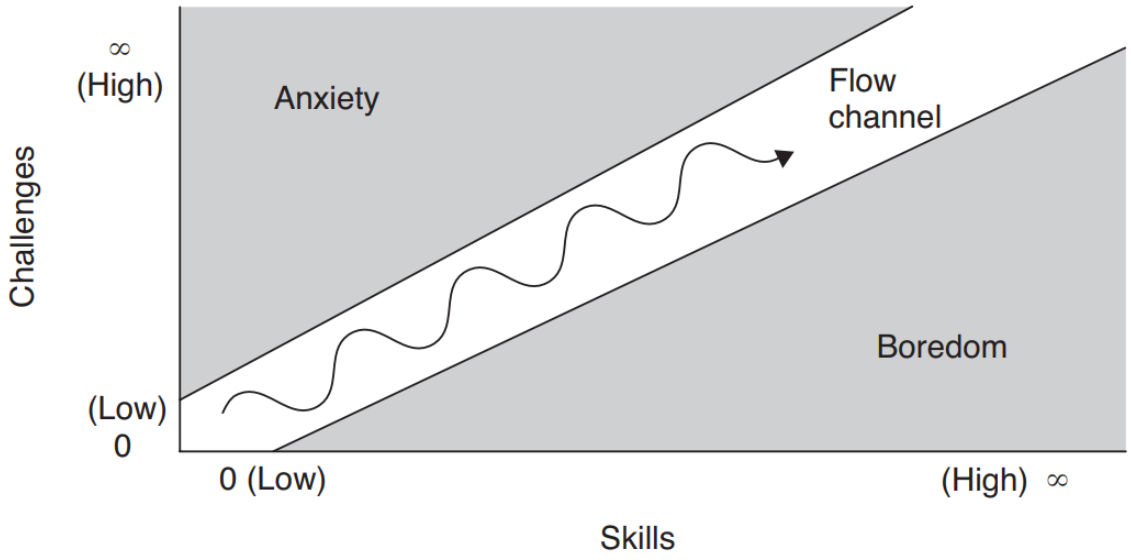


Figura 2 – Gráfico flow

Fonte: reproduzido de Schell (34)

forma de aprendizado para a condução do veículo, sem focar no controle de outras função de um controlador, e a ferramenta TORCS como métrica de desempenho, juntamente com o *deep Q-Learning*, sem ser voltado especificamente em corridas. Nesse caso, inclui-se o desenvolvimentos de algoritmos para outros objetivos e que são colocados à prova no TORCS. Com isso, pode-se perceber uma carência por pesquisas aprofundadas nas áreas de criação de robôs com IA voltados para jogos de corrida de carros, como o TORCS.

Este tema também reflete na indústria, mais especificamente na automobilística e carros autônomos, como exposto por Sallab et al. (32). Carros de corrida, como os da Fórmula 1, são desenvolvidos para serem rápidos, porém ainda mantendo a segurança dos pilotos como prioridade. Um caso da interseção destas duas áreas é o desenvolvimento de um carro autônomo de corrida pela Roborace[1].

Nos próximos capítulos serão apresentados alguns fundamentos básicos para o entendimento do trabalho, seguido de trabalhos relacionados, após é descrito a forma em que os testes foram conduzidos, na sequência os resultados são exibidos, por fim é feita uma discussão sobre os resultados e uma conclusão.

[1] `https://roborace.com/`

3 REVISÃO BIBLIOGRÁFICA

Esta seção apresenta uma breve revisão de alguns fundamentos básicos para a elaboração e entendimento do trabalho.

3.1 JOGOS

Em 1993 a revista Time[1] publicou em seu site um grande avanço na indústria de jogos entre jovens, o que tornava este setor uma "máquina de fazer dinheiro". Em 2018, segundo a empresa de pesquisas Newzoo[2], a indústria de jogos faturou aproximadamente 138 bilhões de dólares. Para manter os jogadores entretidos e comprando jogos, é necessário desenvolver títulos cada vez mais atrativos, sendo com novos desafios, jogabilidades, histórias ou gráficos, para seu público alvo. Trabalhos para engajamento dos jogadores com a criação de partes de um mundo (Bevilacqua (2)), pistas (Togelius; De Nardi; Lucas (36) e Cardamone; Loiacono; Lanzi (6)) ou as fases de um jogo (Pedersen; Togelius; Yannakakis (30)), vem sendo desenvolvidos.

Na área de jogos de corrida pode-se citar o trabalho desenvolvido pela Codemasters[3], empresa especializada neste ramo de jogos digitais. A qualidade gráfica dos jogos da Fórmula 1 estão cada vez mais próximas de imagens reais, como mostrado na Figura 3. Essa evolução (conforme apresentado na Figura 4) traz mais desafios para empresas menores que, teoricamente, precisam gastar mais tempo do desenvolvimento para aprimorar os gráficos de seus jogos.

Com os recursos temporais e monetários voltados para os gráficos, muitas empresas visam utilizar algoritmos mais genéricos e tradicionais para seus carros. Esses controladores utilizam matemática para calcular a direção praticamente perfeita. Esta direção muitas vezes torna o jogo mais desafiador porém menos parecido que o que um humano pode exercer. O comportamento além das habilidades do jogador pode interferir na sensação de imersão do jogador. Isso abre um espaço para exploração, pela indústria de jogos, de técnicas de inteligências artificiais para jogos de corridas.

3.2 SIMULAÇÃO

Ideias sobre a utilização da simulação computacional como uma grande área da computação podem ser empregadas também na simulação de corridas. D'Angelo; Ferretti; Ghini (11) colocam alguns motivos e vantagens para se utilizar a simulação computacional, ao invés da implementação real do sistema. Dados como: os custos de implementar um sistema compu-

[1] http://content.time.com/time/subscriber/article/0,33009,979289-1,00.html acesso em: 24 maio 2019

[2] https://newzoo.com/insights/articles/global-games-market-reaches-137%2D9-billion-in-2018-mobile-games-take-half/ acesso em: 24 maio 2019

[3] http://www.codemasters.com/

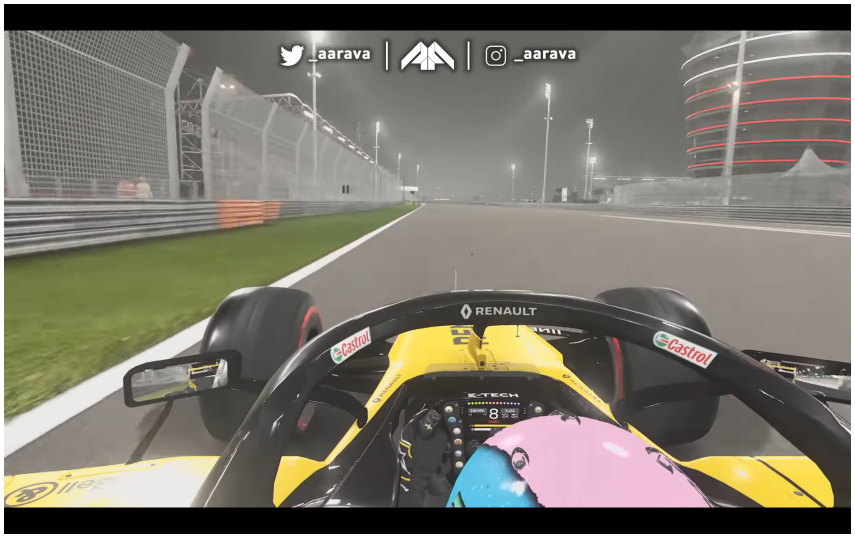


Figura 3 – Exemplo de visual do F1 2019, um jogo lançado em 2019, mostrando a evolução gráfica deixando-o cada vez mais parecido com a vida real

Fonte: https://www.youtube.com/watch?v=pxDzhAvhVkk - Acesso em: 24 maio 2019

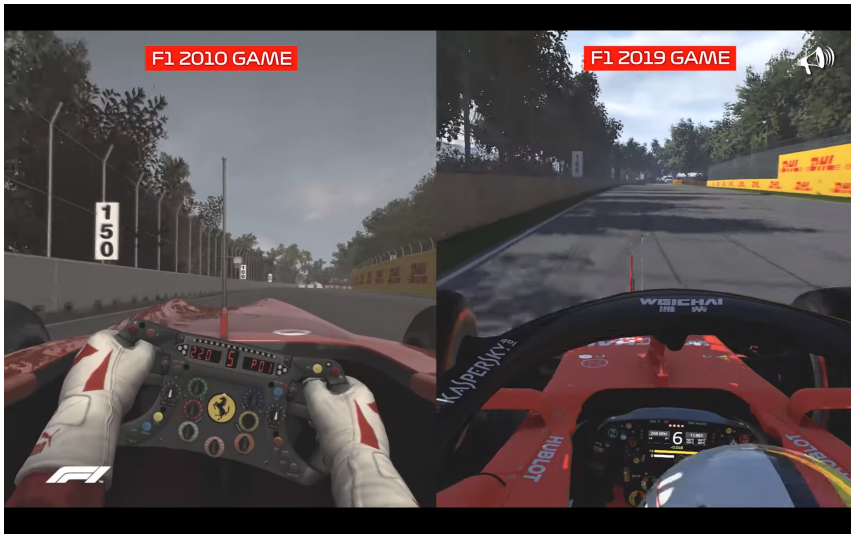


Figura 4 – Exemplo comparativo do visual dos jogos digitais F1 2010 e do F1 2019, mostrando a evolução gráfica que empresas aplicaram no últimos anos de desenvolvimento

Fonte: https://www.youtube.com/watch?v=Sn9oQOc3BFA - Acesso em: 13 junho 2019

tacional, riscos para a vida de alguma pessoa e a possibilidade de se alterar o sistema de várias formas tornando-o mais ”robusto” e maleável para o propósito que lhe é necessário.

Para a área específica de simulação de corridas de carros existem centenas de ”jogos de corrida” disponíveis no mercado, gratuitos ou pagos, que podem ser utilizado na simulação, treinamento ou validação de um algoritmo de IA. Inicialmente, para este trabalho, foram qualificados pela qualidade gráfica de alguns jogos considerados de "última geração": o F1 2018[4]

[4] http://www.codemasters.com/game/f1-2018/

e o Project Cars 2[5]. Eles possuem uma maneira assíncrona de leitura de dados da simulação, processo chamado de telemetria, com uma API UDP. Eles utilizam uma API de uma via, ou seja, somente enviando os dados, sem a possibilidade de retornar alguma ação processada pelo algoritmo de aprendizagem. Por esse motivo, foi procurada uma alternativa que disponibilizasse uma interface de comunicação de dois caminhos - adquirir os dados e devolver os movimentos computados - ou que já fosse utilizada em outros trabalhos.

Em uma busca na literatura referente a um simulador que propiciasse as formas de manipulação mencionados, encontraram-se alguns trabalhos relacionados à área (Munoz; Gutierrez; Sanchis (29), Sallab et al. (32), Cardamone; Loiacono; Lanzi (5), Koutnı'k et al. (18) e Mnih et al. (27)) que utilizam o TORCS, um simulador de corridas de carros.

3.3 THE OPEN RACING CAR SIMULATOR (TORCS)

Desenvolvido originalmente por Eric Espié e Christophe Guionneau[6] e agora mantido, principalmente, por Bernhard Wymann, o TORCS é um jogo digital de simulação de corrida de carros. Ele é utilizado tanto para trabalhos de pesquisa em inteligência artificial, colocando-o como estado da arte na área de pesquisas, quanto para diversão. Ele está disponível para Linux (todas arquiteturas, 32 e 64 bit), FreeBSD, OpenSolaris, MacOSX e Windows (32 e 64 bit). Segundo Wymann et al. (40), TORCS é um moderno, modular e altamente portável simulador de corridas de carro e, por esse motivo, é utilizado como uma ferramenta para pesquisas em inteligência artificial, imitando algumas características reais de um carro, como a aerodinâmica e consumo de combustível. Ele possui recursos gráficos simples, como exemplificado na Figura 5 onde são ilustrados alguns carros disponíveis, o que possibilita rodar em computadores com baixo poder de processamento gráfico.

O TORCS possui o sistema de carregamentos em módulos externos, possibilitando o desenvolvimento e os treinamentos de modo independente aos outros controladores, desde que sigam o padrão de codificação de sua API. Também possui um modo de utilização via linha de comando, facilitando a criação de *scripts* para treinamentos de algoritmos, automatizando-os com diferentes hiper-parâmetros, sem a necessidade de execução manual de cada algoritmo.

Sendo este um software licenciado GNU General Public License (GPL 2) e com maior parte de sua arte como Free Art License[7], por isso a ferramenta pode ser utilizada livremente. Nota-se sua utilização em pesquisas sobre carros autônomos, como os trabalhos de Sallab et al. (32) e Chen et al. (9), por ser um ambiente de simulação que proporciona uma fácil alteração de código e manipulação de variáveis.

O TORCS possui um sistema de controladores programados em C++/C carregados externamente através de módulos, utilizando fórmulas que descrevem a melhor ação para manter

[5] https://www.projectcarsgame.com/
[6] http://torcs.sourceforge.net/index.php?name=Sections&op=viewarticle&artid=19
[7] http://artlibre.org/licence/lal/pt/

Figura 5 – Exemplo de visual do TORCS

o robô o mais perfeito possível. Entre eles pode-se destacar o robô "Berniw"[8], que possui várias fórmulas para calcular a melhor trajetória que o carro deve tomar, tornando-o um dos melhores robôs disponíveis.

O TORCS dispõe de um total de 42 circuitos disponíveis para corridas. Alguns simulam circuitos reais, como é o caso do Wheel 2 e Forza, com traçados antigos dos circuitos de Suzuka (Japão) e Monza (Itália), respectivamente. Aparentemente, o circuito Wheel 2 pode ser um bom candidato para realizar o treinamento de uma rede neural (utilizado em Cardamone; Loiacono; Lanzi (5)), por possuir uma variedade de cenários (curvas de alta e baixa velocidade para ambos os lados e retas). Outra pista utilizada em alguns artigos (Sallab et al. (32), Macedo et al. (24)) é a CG Speedway Number 1, que possui várias retas rápidas, fazendo-se necessário utilizar o freio para não sair da pista nas curvas.

Os modos de corrida do TORCS consistem em: Corrida Rápida, Não-campeonato, Resistência, Campeonato, Corrida de Desafio e Prática. É possível selecionar cada um dos participantes e as pistas para as corridas. O modo Prática é utilizado para treinamento das habilidades dos pilotos, este modo pode ser utilizado para treinamento de algoritmos, sendo simples de reiniciar e recuperar os dados caso seja necessário.

O TORCS ainda possui duas formas de execução do jogo: *Graphical User Interface* (GUI - conforme ilustrado na Figura 5) e *Command Line* (CL) - ilustrado na Figura 6. Nesse último modo, pode-se ler alguns dados da simulação como: tempo total da simulação, voltas lideradas e a distância percorrida. O modo GUI é o modo padrão de execução e o CL pode ser acessado executando via terminal utilizando "-r" e passando uma configuração XML. Não é possível executar com um jogador configurado para modo CL, pois, não foi desenvolvido um meio de capturar as entradas do teclado ou controle para jogar.

[8] `http://www.berniw.org/tutorials/robot/tutorial.html`

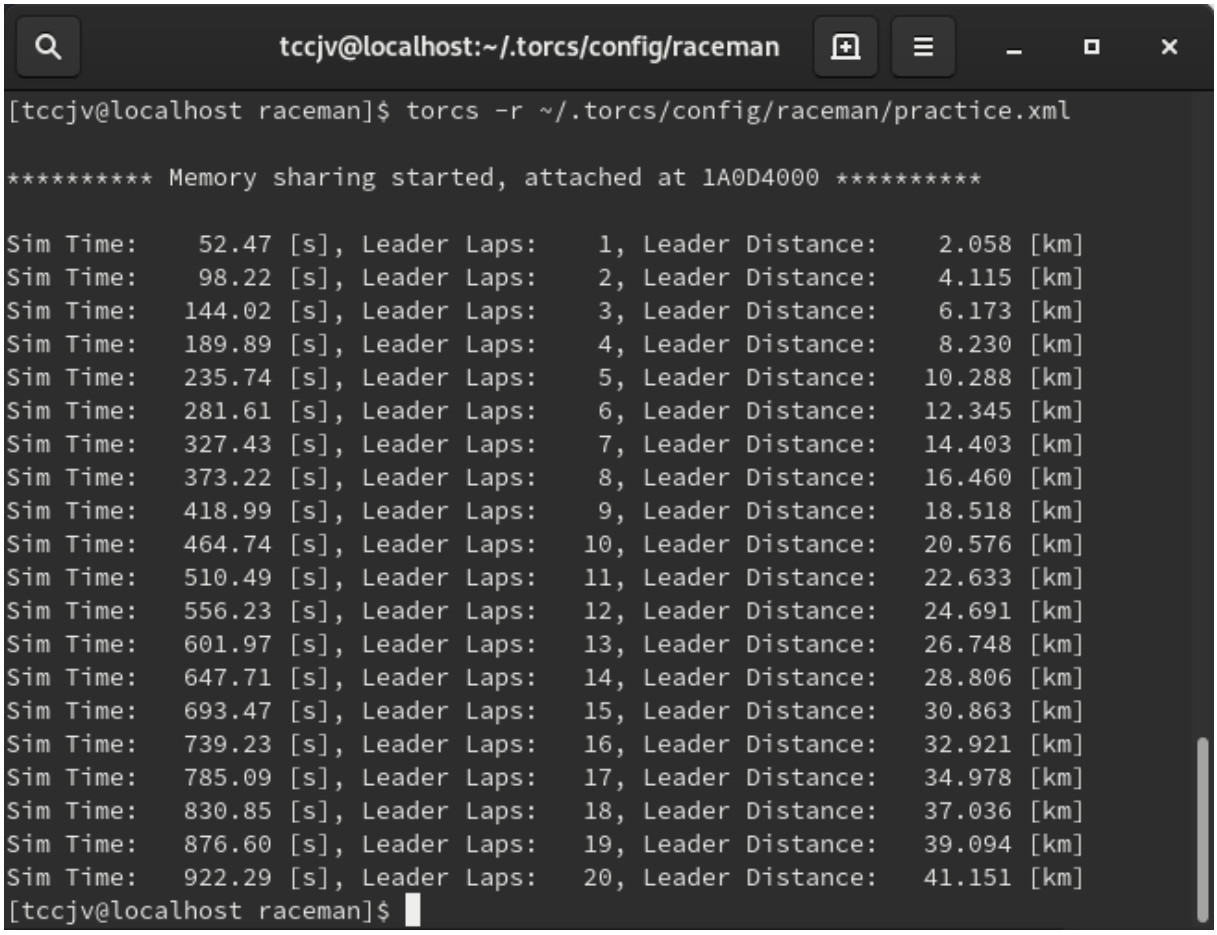


Figura 6 – Exemplo de visual do TORCS em modo CL

Nas pesquisas de fundamentação teórica, foram encontrados dois softwares que realizam um tratamento simplificado dos dados vindos da API do TORCS. O primeiro, citado por Espié et al. (12) é o trabalho de Caldeira (4), porém, não foram encontradas muitas referências em outros trabalhos. O segundo, feita por Loiacono et al. (23), facilitando, segundo eles, a criação de um "campeonato de IAs", chamado por eles de "Simulated Car Racing Competition Software (SCR)". Sua utilização é mencionada significativamente na literatura - inclusive com códigos já disponíveis para utilização e modificação.

Pelas vantagens de não necessitar um retrabalho sobre o código de implementação, foi escolhida a utilização do SCR, disponível para download[9] com as alterações do TORCS necessárias para sua utilização.

3.4 SIMULATED CAR RACING COMPETITION SOFTWARE

Segundo Loiacono; Cardamone; Lanzi (21), o Simulated Car Raning Competition Software (SCR) foi desenvolvido com o intuito de utilizar-se no Simulated Car Racing Championship. Esta competição, de âmbito internacional, era realizada em grandes conferências

[9] Código com o *patch* `https://github.com/fmirus/torcs-1.3.7`

no campo da Computação Evolutiva e no campo da Inteligência Computacional e Jogos. A ferramenta foi disponibilizada para ser instalada[10] junto com o TORCS rodando na versão 1.3.7.

O TORCS possui um sistema de módulos para o carregamento de seus robôs, porém, como trazem os autores do SCR, isso apresenta três desvantagens: 1) corridas não são em tempo real: utilizam-se chamadas bloqueantes; 2) robôs podem pegar os dados de outros carros, alterando sua estratégia com base neles; e 3) como o TORCS é feito em C/C++, a API se restringe a aceitar somente estas duas linguagens. O SCR amplia as possibilidades de utilização, facilitando a comunicação e proibindo algumas vantagens que poderiam se obter usando a desvantagem 2.

Os dados, listados na Tabela 1 que cada competidor recebe de seu respectivo carro, são padronizados, respeitando as variáveis que podem ser escolhidas por cada robô no momento da inicialização. As informações possibilitam que o robô saiba a posição do carro em relação à pista e os outros competidores e com isso controle o seu veículo. Como resposta para o SCR, cada carro pode utilizar atuadores, apresentados na Tabela 2, que atuam sobre a forma que o carro será processado na próxima iteração do TORCS.

Na inicialização do controlador, podem ser escolhidas combinações de sensores para a utilização. Por exemplo, um controlador irá utilizar os sensores do ângulo do carro, distância das bordas da pista e a velocidade em no eixo x do carro. Ao analisar os dados recebidos o controlador "percebe" que o seu ângulo é zero, ou seja paralelo à pista, e por isso em uma reta, e decide escolher o atuador de acelerador para fazer o carro ir para frente. Quando o ângulo começa a variar é provável que ele esteja entrando em uma curva, e com isso retira a aceleração e começa a utilizar o atuador para virar o volante e tentar reduzir o ângulo para zero novamente.

Um fluxo de dados entre o TORCS, o SCR e o controlador pode ser descrito como um processo de tratamento e transferência de dados. O fluxograma que mostra as passagens de informações pode ser visto na Figura 7. O TORCS encaminha os dados da simulação para o SCR, via mensagem no *socket* do SCR, o SCR recebe os dados, converte para o seu padrão e disponibiliza nas variáveis assinaladas para o controlador.

3.5 CARROS AUTÔNOMOS

Carros são utilizados para os mais diversos meios, desde transportes de cargas até corridas. Pesquisas sobre veículos autônomos vêm crescendo (Sallab et al. (32), Waymo[11], Bimbraw (3)), porém, ainda com algumas falhas, mostrando que, mesmo sendo uma área promissora, ainda é necessário estudo mais aprofundado. A ideia de um carro autônomo é um veículo que se desloca "sozinho". No lugar de uma pessoa tem-se um agente controlador treinado para avaliar o ambiente e tomar as decisões para guiar o carro. Segundo Mitchell

[10] Código para *patch* manual `https://github.com/barisdemirdelen/scr-torcs-1.3.7`
[11] `https://waymo.com/`

Tabela 1 – Tabela de sensores disponibilizados pelo SCR que podem ser utilizadas como entradas para a rede neural

Nome	Variação (unidade)	Descrição
angle	$[-\pi,+\pi]$ (rad)	Ângulo entre a direção do carro e o eixo da pista.
curLapTime	[0,+∞) (s)	Tempo percorrido durante a volta atual.
damage	[0,+∞) (pontos)	Dano atual do carro (quanto maior o valor mais dano sofrido).
distFromStart	[0,+∞) (m)	Distância do carro a partir da linha inicial.
distRaced	[0,+∞) (m)	Distância cobrida pelo carro desde o início da corrida.
fuel	[0,+∞) (l)	Quantidade de combustível disponível.
gear	[-1,0,1,...,6]	Marcha atual: -1 ré, 0 neutro e 1ª à 6ª marchas.
lastLapTime	[0,+∞) (s)	Tempo da última volta completada.
opponents	[0,200] (m)	Vetor de 36 posições: cada sensor abrange um ângulo de 10 graus com uma distância de 200m e retorna a distância do oponente mais próximo dentro da área.
racePos	1,2,...,N	Posição de corrida.
rpm	[0,+∞) (rpm)	Número de rotações por minuto do motor.
speedX	(−∞,+∞) (km/h)	Velocidade do carro no eixo longitudinal.
speedY	(−∞,+∞) (km/h)	Velocidade do carro no eixo transversal
speedZ	(−∞,+∞) (km/h)	Velocidade do carro no eixo Z.
track	[0,200] (m)	Vetor de 19 posições com sensores de distância: cada sensor retorna a distância do carro até a borda pista em um intervalo de 200 metros.
trackPos	(−∞,+∞)	Distância entre o carro e o eixo da pista. Valores maiores que 1 ou menores que -1 significam que o carro está fora da pista.
wheelSpinVel	[0,+∞] (rad/s)	Vetor de 4 posições representando a velocidade de rotação das rodas.
z	[−∞,+∞] (m)	Distância do centro de massa do carro da superfície da pista.

Fonte: adaptado de Loiacono; Cardamone; Lanzi (21)

(25), algoritmos para carros autônomos são métodos para conduzir corretamente o carro em diferentes estradas.

A evolução das tecnologias automobilísticas também pode ser vista nas corridas. Muitas montadoras de carros utilizam competições para testar novas invenções que depois são levadas aos carros populares. Mesmo evoluindo ano após ano, os carros de corrida acabam ficando limitados pela segurança dos pilotos.

Aplicando a ideia de carros autônomos, se não houver pilotos humanos, as corridas poderiam ser mais rápidas, pois, não colocam a vida de humanos em risco. Cria-se, então, a possibilidade de colocar os carros em seu limite, desenvolvendo tecnologias voltadas à velocidade. Esta subárea já vêm sendo estudada, como o caso da Roborace, utilizando aprendizado

Tabela 2 – Tabela de atuadores virtuais que podem ser as ações da rede neural

Nome	Variação	Descrição
accel	[0,1]	Pedal de aceleração virtual (0 não pressionado, 1 totalmente pressionado).
brake	[0,1]	Pedal de freio virtual (0 não pressionado, 1 totalmente pressionado).
clutch	[0,1]	Pedal de embreagem virtual (0 não pressionado, 1 totalmente pressionado).
gear	-1,0,1,...,6	Marcha selecionada.
steering	[-1,1]	Valor de giro do volante: -1 totalmente para virado para a esquerda e +1 totalmente para direita, que corresponde a um ângulo de 0.366519 rad.
meta	0,1	Comando de controle: 0 não faça nada, 1 solicita ao servidor que a corrida seja reiniciada.

Fonte: adaptado de Loiacono; Cardamone; Lanzi (21)

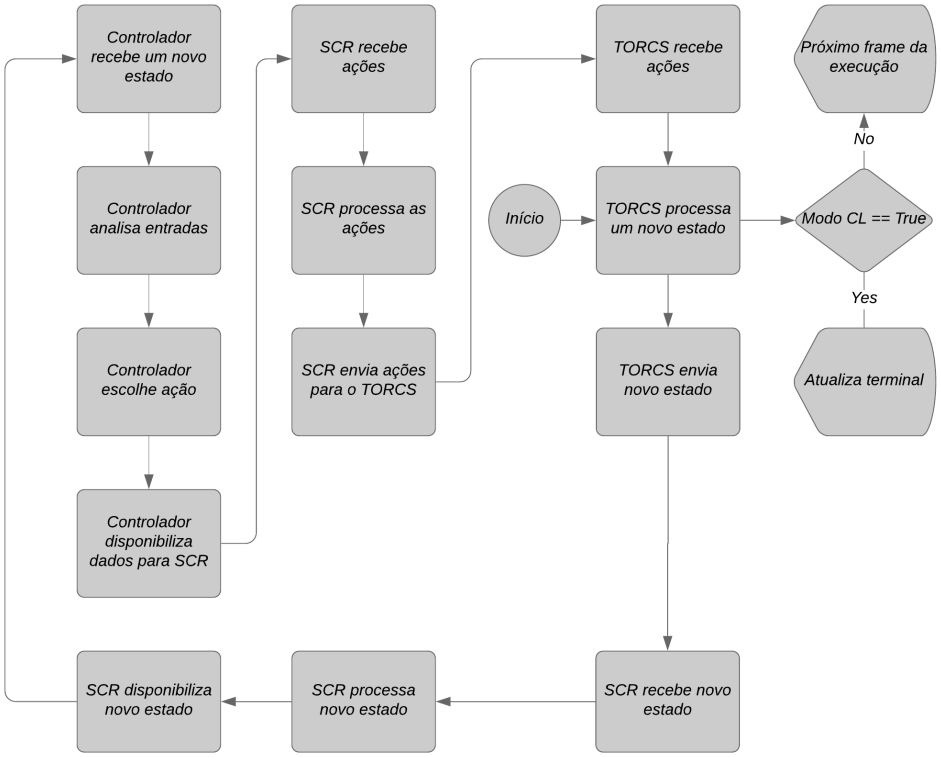


Figura 7 – Fluxograma as transições das informações entre o TORCS, o SCR e o controlador

de máquina, para carros elétricos autônomos competirem em uma pequena corrida.

3.6 REDES NEURAIS

Os neurônios, segundo Haykin (15) e ilustrado na Figura 8, são unidades de processamentos da informações e são formadas pelas entradas, pesos para as entradas, um somador

para unir as entradas e uma função de ativação. Os pesos, junto com as entradas, simbolizam as sinapses que acontecem no cérebro. Quanto maior o peso atrelado à entrada, maior será a importância dela para a ativação daquele neurônio. A ativação, por sua vez, é feita através de uma função matemática. A função de ativação é fundamental para limitar a ativação do neurônio, permitindo que ele seja ativado a partir de um determinado valor ou gerando uma amplitude para a saída.

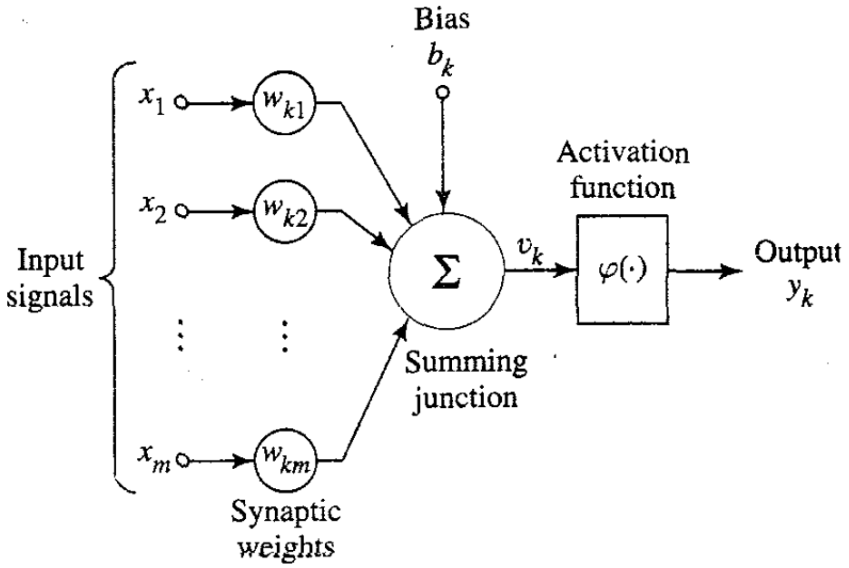


Figura 8 – Exemplo de neurônio artificial desenvolvido para simular um neurônio cerebral

Fonte: reproduzido de Haykin (15)

Os pesos sinápticos de cada entrada são iniciados com valores aleatórios. Para cada novo exemplo apresentado para o neurônio, é verificada se a saída foi a esperada ou não e os pesos são atualizados, aumentando o peso dos que são mais representam a saída necessária e diminuindo os que menos representam. O processo de atualização dos pesos, normalmente, é custoso computacionalmente, pois, é necessário atualizar os pesos de todas as entradas e, quanto mais entradas, mais pesos necessitam ser atualizados. Frequentemente, vários neurônios são utilizados para entradas e encadeados, o que faz com que aumente substancialmente a quantidade de alterações necessárias. Quando agrupados de forma que a saída de um neurônio seja a entrada de um outro, criando camadas de neurônios interligados, cria-se uma rede neural. As camadas intermediárias, isto é, as que não são nem entradas e nem as saídas da rede neural, são chamadas de camadas ocultas.

Existem várias formas de categorizar as redes neurais, normalmente são organizadas por arquiteturas de aprendizado. Estas podem ser categorizadas como aprendizado supervisionado e o não supervisionado. Pode-se descrever o aprendizado supervisionado como aquele que, apresentando um conjunto de dados com rótulos, pode-se entregar para a máquina uma parte destes dados e ela irá “perceber” as semelhanças entre eles. Já um método não supervisionado pode ser definido como uma forma da máquina aprender sem ter uma saída necessariamente

conhecida. Ela pode, ainda, criar grupos com os dados - este caso é chamado de *clustering*. Além destas duas formas de aprendizado, existe uma terceira que, com os avanços da capacidade computacional, vem ganhando espaço nas pesquisas, uma que utilizam reforços positivos e negativos para o aprendizado, chamada de *Reinforcement Learning*.

3.7 APRENDIZADO POR REFORÇO

Segundo Watkins (39), o aprendizado por reforço (Reinforcement Learning - RL) pode ser considerado uma forma parecida de como os animais aprendem, através de estímulos positivos e negativos, ou tentativa e erro. Dentro do reino animal, um predador precisa, por exemplo, aprender a hora correta para dar a investida sobre a presa, ou caso contrário a presa escapa. Normalmente, no reino animal, só se tem uma tentativa entre o que pode ser a última refeição e a garantia de mais alguns dias de vida.

Os computadores possuem uma vantagem no aprendizado por reforço: podem realizar várias iterações com o propósito de alcançar o melhor resultado. Se uma ação tomada gerou um resultado ruim, basta recompensá-la negativamente e durante a próxima execução, possivelmente, ela não será tomada. Esse processo pode repetir-se enquanto não se consegue o melhor resultado para aquela situação. Quando garante-se que será escolhido o melhor resultado para cada um dos cenários que o agente enfrenta, tem-se a maximização das recompensas. Esse processo pode ser comparado com a definição de Mitchell (25), dizendo que um algoritmo aprenda a realizar determinada tarefa.

De fato, o aprendizado por reforço é uma aplicação do Processo de Decisão de Markov, que pode ser divida em: um conjunto finito de estados (S), um conjunto de ações A(x) para x em S e um conjunto R_t recompensas para tempo discreto t = $\{1, 2, 3, \cdots\}$. Um exemplo utilizado pelo site freeCodeCamp[12] utiliza o jogo digital Super Mario (visualmente ilustrada na Figura 9): o agente de RL recebe um estado inicial S^0 do ambiente, neste caso o jogo Super Mario. Baseado neste estado, o agente toma uma ação A^0 - seja ela pular, andar ou atirar. Como em um primeiro momento não é possível saber qual ação gera a recompensa, ela é aleatória. A ação enviada para o ambiente, que passa para um estado S^1 (aplicando a ação escolhida pelo usuário), devolvendo ao agente uma recompensa R^1 para a ação e muda para o novo estado. Esta sequência de ações, estados e recompensas seguem em ciclo até que atinja-se um estado final - neste exemplo morte do personagem ou atingir o objetivo.

3.7.1 Q-Learning

Watkins (39) o criador do algoritmo de Q-Learning com o intuito de medir a qualidade das ações tomadas pelo computador. Ele utilizou as ideias da Árvore de Decisão de Markov

[12] https://www.freecodecamp.org/news/a-brief-introduction-to-reinforcement%2Dlearning-7799af5840db/?gi=6df0f5fb0cdb acesso em: 27 maio 2019

Figura 9 – Exemplo iterações do *Reinforcemente Learning* utilizando Super Mario

Fonte: reproduzido de freeCodeCamp[12]

para construir seu algoritmo. O Algoritmo 1 apresenta uma implementação onde α é a taxa de aprendizado, γ é o fator de desconto, R é a recompensa e S′ é o estado após o ambiente executar a ação A.

Segundo Loiacono et al. (22) o Q-Learning pode ser considerado o mais simples algoritmo de aprendizado por reforço e se tornou muito popular dentro da computação. Ele utiliza uma tabela de consulta (*look-up table* - ou Q-table) para salvar os valores das ações - $Q(\cdot,\cdot)$ - tomadas para assim poder calcular a qualidade da ação atual.

Algoritmo 1 – Q-Learning

```
1 Inicialize Q(s, a), ∀ s ∈ S, a ∈ A(s), arbitrariamente, e Q(estado terminal, ·) = 0;
2 foreach episódio do
3 |   Inicialize o estado S;
4 |   foreach passo do episódio && estado S não é terminal do
5 |   |   Escolha A de S usando a política derivada de Q (e.g., ϵ-greedy);
6 |   |   Tome a ação A, observe R, S′;
7 |   |   Q(S, A) ← Q(S, A) + α[R + γmax_a Q(S′, a) - Q(S, A)];
8 |   end
9 end
```

Fonte: adaptado de Sutton; Barto (35)

Este algoritmo é utilizado em alguns trabalhos desta área de pesquisa (Loiacono et al. (22) e Sallab et al. (33)), porém, ele possui algumas limitações para o aprendizado de atividades mais complexas.

3.7.2 Deep Q-Learning

Mnih et al. (28), criadores da *Deep Q-Network* (DQN) e pesquisadores da DeepMind[13], desenvolveram uma forma mais avançada de utilizar técnicas de *Deep Learning* e *Reinforcement Learning* juntas. O algoritmo foi capaz de superar os resultados de seres humanos em alguns jogos do Atari 2600. A Figura 10 exemplifica o desempenho do modelo criado pelos autores em alguns jogos eletrônicos, comparando-o com o algoritmo de G Bellemare; Veness; Bowling (13), utilizando como entrada para a rede somente pixels e a pontuação.

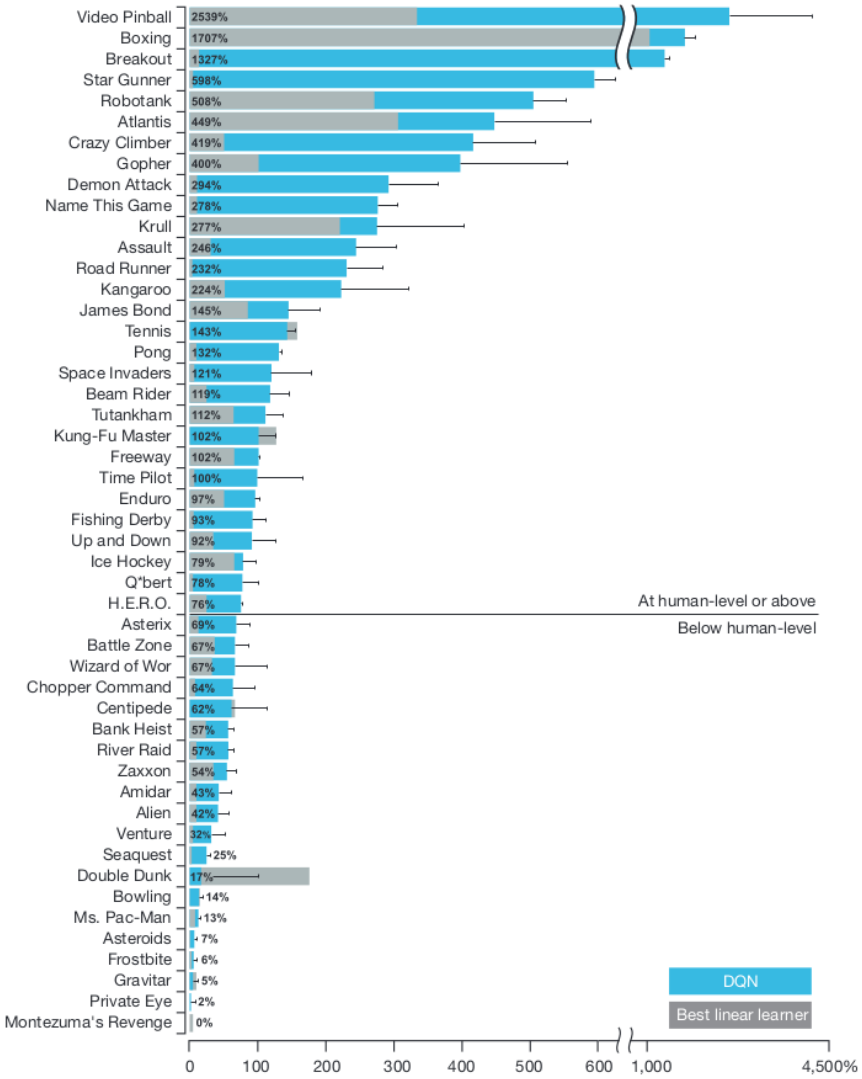


Figura 10 – Comparação do DQN por Mnih et al. (26)

Fonte: reproduzido de Mnih et al. (26)

Os resultados da pesquisa de Mnih et al. (28) auxiliaram no desenvolvimento da AlphaGo[14]. Alguns anos depois a própria DeepMind evoluiu o seu programa criando um

[13] https://deepmind.com/
[14] https://deepmind.com/research/alphago/

novo algoritmo, o AlphaZero, que obteve resultados mais rápido e melhores que seus antecessores. Segundo o autor, este novo algoritmo ultrapassou a pontuação do AlphaGo Zero em apenas três dias de treinamento. Em comparação, o AlphaGo Zero precisou de 21 dias para ultrapassar o algoritmo anterior, o AlphaGo.

O *Deep Q-Learning* possui vários algoritmos que podem ser utilizados. Um destes é implementado por Ganesh et al. (14), o *Deep Deterministic Policy Gradient* (DDPG), também utilizado por Lau (19). Essa implementação é baseada no trabalho de Lillicrap et al. (20) como algoritmo para a resolução do problema. É utilizada uma memória temporária para armazenar uma quantidade de passos para calcular uma perda (*loss*) daquele passo com base nas informações dos outros passos.

O algoritmo do DDPG é formado por duas redes neurais (exemplificado na Figura 11). A rede da Figura 11 (a), chamada de Actor, recebe como entradas os sensores escolhidos e providos pelo SCR, possuindo duas camadas ocultas, a primeira com 300 neurônios e a segunda com 600, e uma saída para escolher as ações tomadas. A rede da Figura 11 (b), chamada de Critic, tem como entradas os mesmos sensores acrescido das ações tomadas. É adicionada uma camada para somar esses valores e, assim, poder "criticar" a qualidade das ações tomadas.

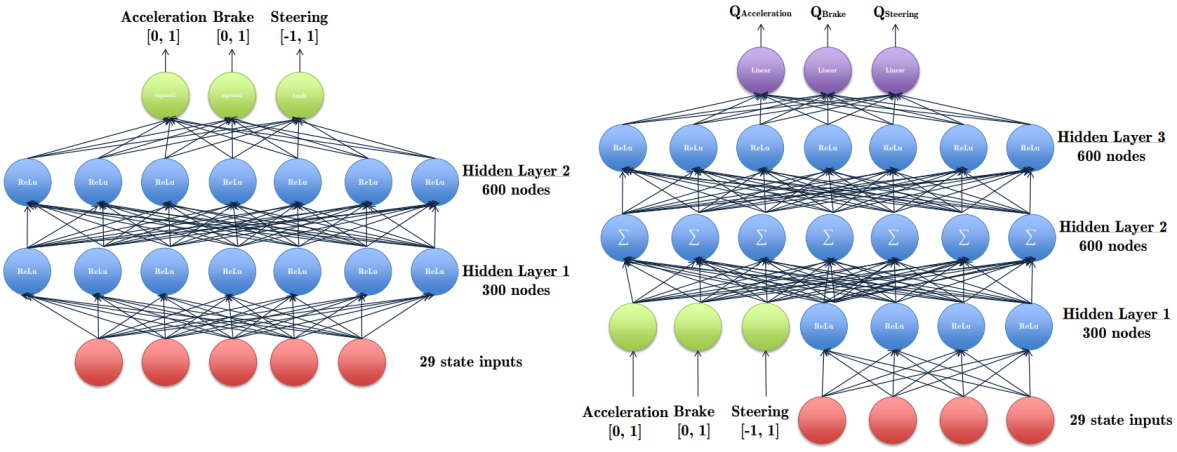


Figura 11 – Representação das redes utilizadas pelo algoritmo DDPG

Fonte: reproduzido de Ganesh et al. (14)

A inicialização dos pesos da rede neural é feita com valores aleatórios, fazendo com que as primeiras ações sejam aleatórias. Para que a rede comece a convergir para as ações desejadas, no início do treinamento, são inseridos ruídos nas ações. Lillicrap et al. (20) utilizaram o processo de Ornstein-Uhlenbeck(Uhlenbeck; Ornstein (38)) para gerar o ruído.

4 TRABALHOS RELACIONADOS

A Tabela 3 mostra uma lista de técnicas utilizadas em trabalhos relacionados à área de aprendizado de máquina para jogos de corrida. Estes trabalhos serão explicados na sequência.

Tabela 3 – Tabela de técnicas

Referência	**Técnica**
Macedo et al. (24)	Máquina Finita de Estados
Cardamone; Loiacono; Lanzi (7)	Evolutionary Neural Network
Kim et al. (16)	Heurística
Sallab et al. (32)	Deep Deterministic Actor Critic
Cardamone et al. (8)	Genetic Algorithm
Quadflieg et al. (31)	Evolutionary Strategy
Koutnı'k; Schmidhuber; Gomez (17)	Convolutional Neural Network
Loiacono et al. (22)	Q-Learning
Ganesh et al. (14)	Deep Q-Learning
Munoz; Gutierrez; Sanchis (29)	NeuroEvolution of Augmenting Topologies

Fonte: adaptado de Kim et al. (16)

Macedo et al. (24) utilizam uma Máquina Finita de Estados (FSM) para desenvolver seu robô. O controlador foi desenvolvido utilizando algoritmos genéticos para evoluir os parâmetros e conseguir o melhor resultado. As avaliações desenvolvidas se basearam na distância (em metros) e no tempo (em segundos) total percorrido pelo algoritmo em 6 pistas, utilizando 10.000 ticks e 10 voltas, respectivamente. Com o desenvolvimento realizado para o *Simulated Car Racing Championship* de 2015, conseguiram o quarto lugar.

Utilizando NeuroEvolution, Cardamone; Loiacono; Lanzi (7) apresentam um algoritmo que é capaz de transportar o aprendizado adquirido em seu treinamento em determinadas pistas para outras. Com uma técnica semelhante com a utilizada em RL, avaliando as ações tomadas que geram melhores resultados empregando mais recursos computacionais para as execuções. Os treinamentos duravam 2000 voltas e evoluíam 100 controladores. É avaliado o tempo médio (em segundos) de volta em algumas pistas.

Munoz; Gutierrez; Sanchis (29) usam a ideia de transferência de aprendizado para ensinar o seu algoritmo a pilotar. Os dados foram adquiridos de um robô disponível, um ser humano jogando e o controlador vencedor de uma competição de 2008. A rede neural foi treinada com estes dados e depois foi colocada a prova em 17 pistas, listadas na Figura 12. Algumas delas também aparecem em outros trabalhos, com isso, foram utilizadas como comparação. Os resultados obtidos mostram que a rede aprende menos quando utilizado dados de um humano jogando quando comparado com dados de controladores.

Cardamone et al. (8) apresenta um otimizador de trajetos para um controlador no TORCS. Eles aplicaram um algoritmo genético para encontrar o melhor resultado entre uma curvatura mínima e o menor caminho. Ambas características são consideradas importantes para aumentar

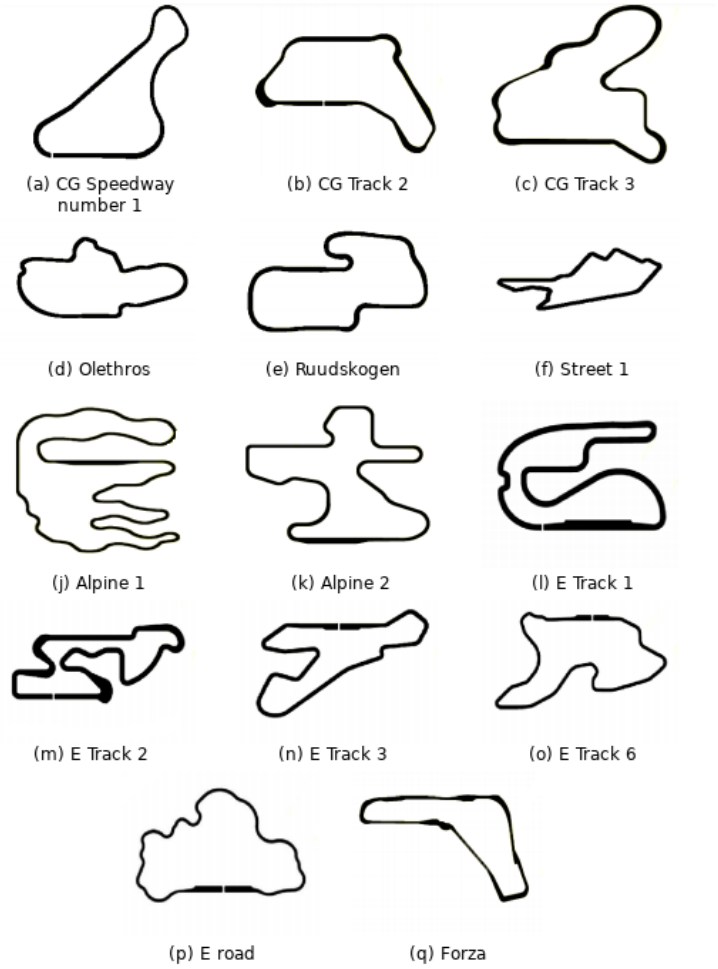


Figura 12 – Lista de 17 pistas utilizadas por Munoz; Gutierrez; Sanchis (29)

Fonte: Adaptado de Munoz; Gutierrez; Sanchis (29)

a média de velocidade do carro, por conta do que é chamado de *apex* (local mais próximo que se passa próximo de uma curva). Os resultados encontrados nos testes diminuíram os tempos das voltas em todos os circuitos, mostrando-se um aspecto importante para o controlador.

Quadflieg et al. (31) segue a mesma linha de trabalho de Cardamone et al. (8), aprender o melhor traçado para a corrida. Como exemplificado na Figura 13, duas partes visualmente distintas de uma pista podem gerar dados muito parecidos para os sensores, o que pode causar um planejamento errado da ação, acelerando quando já deveria frear, por exemplo. Para resolver tal situação, foram mapeados alguns cenários para classes de curvas e planejadas novas ações com base neles. Foi utilizada a ideia de "começar em baixa velocidade", limitando a aceleração do carro para que ele aprendesse de forma simples e depois aumentando o limite.

Koutnı'k; Schmidhuber; Gomez (17) utilizam o TORCS como entrada visual para o desenvolvimento de seu robô. Através do processamento da tela, o agente aprendeu a identificar padrões de curvas e retas. Os resultados mostram que a abordagem ainda está abaixo da expectativa para um controlador somente visual, porém, é um pequeno passo para que sejam exploradas novas formas de visão computacional.

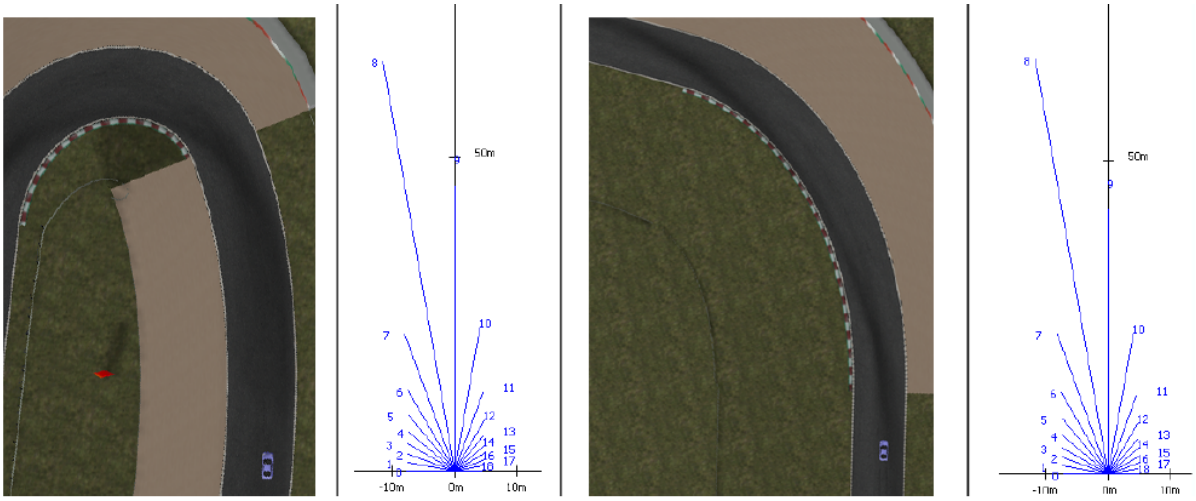


Figura 13 – Gráfico com retorno quase idênticos de 19 sensores do TORCS em duas situações diferentes

Fonte: Reproduzido de Quadflieg et al. (31)

Loiacono et al. (22) desenvolvem um módulo de ultrapassagens para um controlador do TORCS. A ultrapassagem pode ser considerada um dos módulos mais importantes para corridas pois com elas é possível alcançar melhores posições na corrida. Para isso os autores aplicam Q-Learning para treinar o robô a realizar ultrapassagens de forma eficiente. Eles utilizaram alguns cenários onde é possível ultrapassar como: retas longas e frear com atraso. Como resultado, a taxa de sucesso nas ultrapassagens foi maior que a alcançada pelo controlador Berniw.

Sallab et al. (32) utilizam o *Deep Deterministic Actor Critic* (DDAC), uma variação do Deep Q-Network, para realizar o treinamento de seu controlador. O DDAC trabalha com uma política independente da função, separando o algoritmo em duas redes distintas: 1) Actor Network: rede neural responsável por tomar as ações com bases nos estados, e 2) Critic Network: rede neural responsável por "criticar"as ações tomadas em um determinado estado. Os resultados foram visíveis quanto a melhoria/constância da rota: o carro não altera seguidamente a posição do volante em uma linha reta. Este pode ser um bom indicativo de qual função usar, caso seja compatível com a capacidade computacional disponível para realizar o trabalho.

Ganesh et al. (14) utilizam Deep Deterministic Policy Gradient (DDPG) para treinar o seu modelo. O DDPG também utiliza a ideia do DDAC, porém, com alterações na forma de trabalhar as saídas. É utilizado um sistema de aprendizado desenvolvido por Lillicrap et al. (20) que consegue trabalhar em um ambiente contínuo. Muitos trabalhos utilizam saídas discretas, o que pode ser uma vantagem no controle do carro. A exploração de novas funções de recompensa, pistas e políticas de exploração são propostas pelos autores.

Kim et al. (16) utilizam heurística e técnicas de IA para realizar o trabalho de ensinar o controlador. O trabalho apresenta uma boa separação de dados de pistas que podem ser usados como base para o desenvolvimento da IA proposta. Os autores separaram algumas pistas utilizadas no processo de aprendizado, ilustradas na 14, com base na dificuldade das pistas. Eles

utilizaram a velocidade média alcançada para realizar a categorização. As pistas do grupo A foram as mais lentas e por isso consideradas difíceis, as mais rápidas foram colocadas no grupo E, considerando-as mais fáceis.

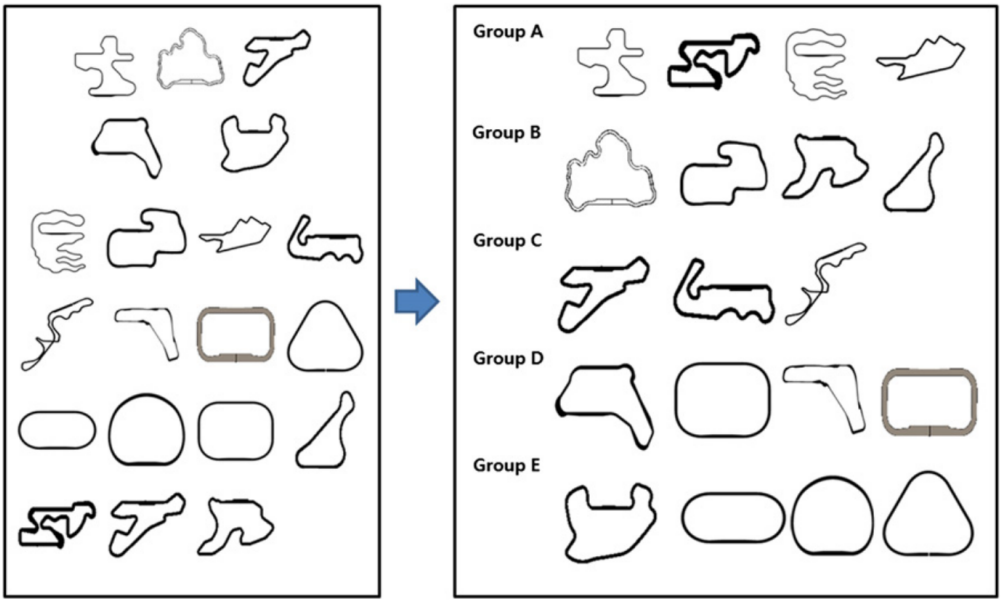


Figura 14 – Pistas utilizados por Kim et al. (16)

Fonte: Reproduzido de Kim et al. (16)

Muitos trabalhos visam testar diferentes formas de melhorar um controlador já existente ou criar novos. De fato, nenhum que utiliza DQL é focado em criar um controlador completo do zero. O foco em uma pista mais fácil, grupo E do trabalho de Kim et al. (16), de pode trazer vantagens para o processo de aprendizado. A função de recompensa pode ter um papel fundamental para a convergência do modelo, sendo a forma de avaliação se as ações que estão sendo tomadas são boas ou não. Técnicas diferentes podem ser utilizadas e combinadas para gerar um modelo, porém, a generalização dele parece ser um dos principais desafios para o trabalho.

5 METODOLOGIA

5.1 IMPLEMENTAÇÃO

Para o desenvolvimento do trabalho, foram utilizados os *frameworks* TensorFlow e Keras com a linguagem Python. O código[1] foi adaptado da implementação de Ganesh et al. (14), que por sua vez foi adaptado de Lau (19) disponível no Github[2]. Este código utiliza *Deep Deterministic Policy Gradient* (DDPG) e foi portado para uma versão mais recente dos *frameworks* para testes. O algoritmo foi executado em um computador disponibilizado pela Universidade Federal da Fronteira Sul (UFFS). Este possui as seguintes especificações: 8GB memória RAM, processador Intel® Core™ i5-4570 e sem utilizar placa de processamento gráfico (GPU).

Como descrito em Loiacono; Cardamone; Lanzi (21), é possível rodar a aplicação TORCS em modo texto. Para isso é preciso adicionar o comando "-r race_config.xml" junto ao comando de execução do TORCS no terminal. Esse comando carrega o arquivo XML que contém as configurações da corrida. Nela podem ser escolhidas as pistas somente alterando um campo, o que ajuda na automatização do processo de treino.

Toda via, durante o desenvolvimento do agente, foi encontrada uma dificuldade em utilizar o TORCS em modo CL, pois, o computador utilizado não possuía capacidade de processamento suficiente para executar as ações na velocidade mínima necessária. No lugar disso, o arquivo XML foi configurado para executar sempre a mesma pista em modo GUI. O modo GUI do TORCS ainda gerou o problema de convergência do modelo, pois, o computador ainda não possuía capacidade de processamento suficiente. O problema de convergência pode estar ligado com a demora para o SCR receber a nova ação, visto que este executa com chamadas não bloqueantes, enviando para o TORCS a última ação recebida novamente.

Para resolução deste empecilho, foi diminuída a velocidade de execução do TORCS e de toda a simulação para 50% do tempo normal. Com a redução o SCR não precisava encaminhar na mesma frequência, o que tornou possível executar o treinamento. A redução da velocidade de execução faz com que os *ticks* - ciclos de tempo dentro do jogo (coloquialmente relacionado com *Frames Per Second (FPS)*) - aumentasse de 20ms para 40ms.

Para fins de treino máximo, foi utilizada a *flag* "-nofuel", desabilitando o consumo de combustível. O combustível é uma informação importante dentro da estratégia de corrida, porém, não foi avaliada por demandar codificação humana para realizar uma troca de estados, isto poderia dificultar a execução deste trabalho. Também seria necessário desenvolver uma estratégia para calcular o melhor momento de parar para reabastecer, para entrar no *box* destinado no início da corrida e controlar o gasto de combustível.

[1] https://github.com/joaovitorblabres/TCC-JV-SCR-TORCS
[2] https://github.com/yanpanlau/DDPG-Keras-Torcs

5.2 DESENVOLVIMENTO DE PROTÓTIPOS

Antes da escolha definitiva dos hiper-parâmetros utilizados foram desenvolvidos protótipos. Os protótipos serviram para descartar possíveis modelos, após uma investigação preliminar, que não tiveram resultados tão relevantes. Foram efetuados testes com alguns modelos de rede como o apresentado por Ganesh et al. (14) e em modelos derivados. Por conta da grande variedade de sensores que são disponibilizados pelo software que realiza a conexão entre os agentes controladores e o TORCS, os sensores variaram como entradas da rede neural. Com isso foi possível identificar quais sensores a rede neural melhor aprendeu durante o treinamento.

Os sensores utilizados, exibidos na Tabela 4, somam um total de 29 entradas para a rede neural. Foram testadas alterações de entradas variando entre 22 e 29, retirando os sensores: trackPos, speedZ, wheelSpinVel e rpm. A remoção que apresentou o melhor resultado foi retirando o sensor de posição na pista. Com isso o total de sensores utilizados como entradas foi de 28.

Tabela 4 – Tabela de Sensores utilizados

Nome	**Quantidade de sensores**
angle	1
track	19
speedX	1
speedY	1
speedZ	1
wheelSpinVel	4
rpm	1
trackPos	1

Durante testes foram utilizados alguns algoritmos de aprendizado profundo. Os primeiros modelos testados foram os de *Deep Q-Network* (DQN), *Actor Critic Advantage* (A2C) e o *Deep Deterministic Policy Gradient* (DDPG) utilizando os códigos do repositório de Morvan Zhou[3]. Após algumas semanas rodando os modelos DQN e A2C, ambos não conseguiram convergir para um modelo bom, por isso foram descartados. Por isso foi utilizado o código de DDPG feito por Lau (19).

Os protótipos serviram para verificar os resultados parciais de funções de ativação e a quantidade de camadas e neurônios para a rede neural, bem como as funções de recompensa para avaliação da ação tomada. Os protótipos foram testados em pistas que já serviram como base para trabalhos relacionados ou que podem gerar uma melhor aprendizagem, na visão do autor, por possuir características específicas. Pode-se considerar que um protótipo não teria uma evolução nos resultados caso ele caísse nos estados 1, 2 ou 3 da Tabela 5. A avaliação dos melhores foi feita através dos tempo de volta (em segundos).

[3] https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow

Os protótipos foram salvos em arquivos de configuração. Com os modelos salvos foi possível continuar treinando-os pelo tempo que fosse necessário ou alcançando o estado final 4 da Tabela 5 - completar dez voltas na pista escolhida.

Tabela 5 – Tabela de estados finais

Número	Estado	Descrição
1	Carro preso (Macedo et al. (24))	O carro é considerado preso quando, por mais de um determinado tempo, não pode se mover, ou se mover abaixo de uma velocidade determinada, por um determinado tempo (500 *ticks* simulados - cerca de 10 segundos simulados)
2	Velocidade negativa	Quando o carro, ao invés de ir para frente, começa a ir para trás por mais de 500 *ticks* simulados
3	Limite de passos	Atingir um limite de passos determinado que possa ser considerado suficiente para completar a tarefa - pode variar de pista para pista
4	Completar dez voltas	Quando o carro completar dez voltas na pista escolhida.

Após a seleção dos protótipos com melhor desempenho, foram analisadas e descritas as características de cada um. Quando encontradas características distintas com bons resultados, e que ainda não haviam sido testadas, elas serão unidas para realizar novos testes e verificação dos resultados. Intuitivamente, essa união poderia gerar resultados melhores que os encontrados até aquele momento, mas isso não foi constatado nos testes.

5.2.1 Hiper-parâmetros

Todos os hiper-parâmetros utilizados como base foram os encontrados no código de Lau (19) e de Ganesh et al. (14). Algumas variações foram testadas, como nas taxas de aprendizado e tamanho do *batch*, mas não surtiram efeitos positivos para a rede, por isso foram descartadas. Os hiper-parâmetros utilizadas foram os originais, presentes na Tabela 6.

Tabela 6 – Tabela de Hiper-parâmetros

Hiper-parâmetro	Valor
Tamanho do *buffer*	100.000
Tamanho do *batch*	32
Gamma γ	0,99
Tau τ	0,001
Taxa de aprendizado rede *Actor*	0,0001
Taxa de aprendizado rede *Critic*	0,001

A quantidade de unidades das camadas internas da rede foram as mesmas utilizadas nos trabalhos supracitados, 300 na primeira camada e 600 na segunda. A Figura 15 exemplifica as duas redes utilizadas, a Figura 15 (a) Actor e Figura 15 (b) a Critic. A rede Critic possui mais

camadas, que atuam para avaliar as ações tomadas, uma vez que, elas também fazem parte das entradas. Para fins de exploração foi realizado um teste com os dois terços (2/3) do tamanho, 200 e 400, respectivamente, e metade do tamanho, 150 e 300.

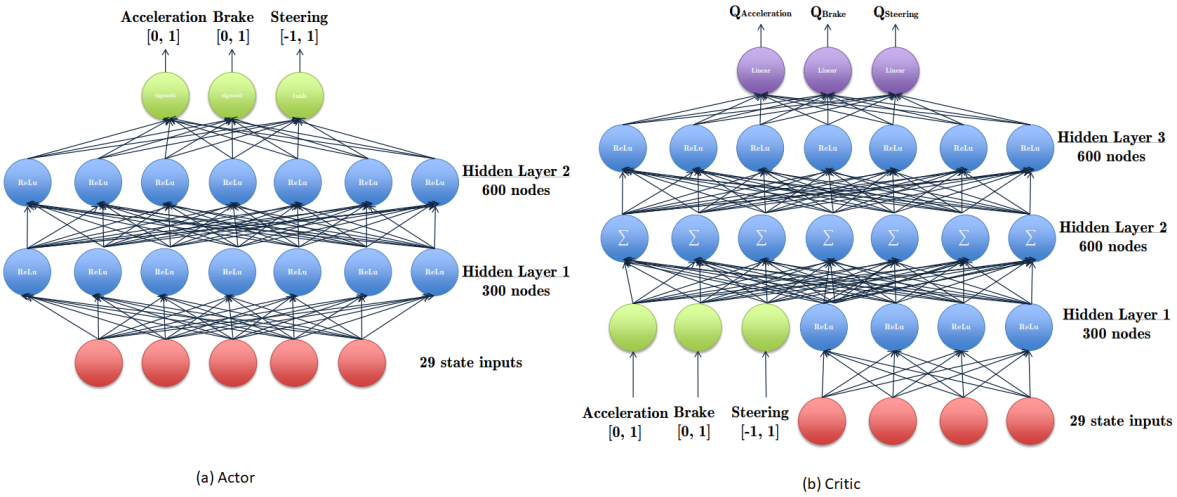


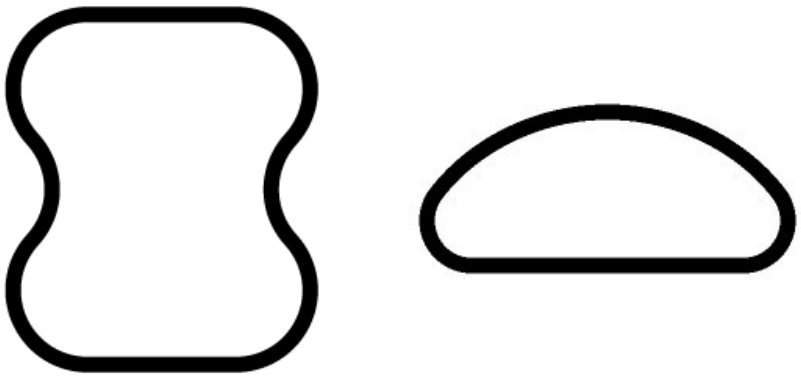
Figura 15 – Figura representativa da rede neural

Fonte: Reproduzido de Ganesh et al. (14)

5.2.2 Pistas escolhidas

Para a realização dos treinamentos foram escolhidas duas pistas. A primeira pista escolhida foi uma pista no formato oval, a Michigan Speedway (ilustrada na Figura 16 (b)). Michigan Speedway possui um grau de inclinação baixo nas curvas, somente com curvas para a esquerda. O baixo grau de inclinação das curvas foi uma vantagem para o início dos testes. Em alguns em testes executados, em outras pistas ovais, os modelos apresentaram problema em manter o carro na pista. Este problema provavelmente foi causado pela falta de controle de tração no carro, o que fazia o carro girar no próprio eixo. A segunda pista escolhida foi a E-Track 5 (ilustrada na Figura 16 (a)), uma pista também com formato oval, porém, com curvas para esquerda e direita.

A primeira parte dos testes, a escolha dos protótipos, foi realizada nas pistas Michigan Speedway e E-Track 5, por serem pistas simples para o controlador completar os testes iniciais. A pista E-Track 5 foi escolhida por ter curvas para a esquerda e para direita e ainda ser simples. A segunda parte foi realizar mais alguns treinamentos em pistas em que ainda não teriam sido treinados. Para realizar a validação dos resultados, foram escolhidas algumas pistas já presentes na literatura.



(a) E-Track 5 (b) Michigan Speedway

Figura 16 – Pistas utilizadas para o treino

5.2.3 Função de Recompensa

Para a evolução da rede neural, é necessária a utilização de uma função de recompensa. Lau (19) demonstra em seu trabalho a função que está descrita na Figura 17. A função utiliza-se da variação do cosseno do ângulo do carro em relação à pista como reforço positivo. Como reforço negativo são aplicadas as variações do seno do ângulo e a posição da pista, ambos em seus valores absolutos.

$$R = speedX * cos(angle) - |speedX * sin(angle)| - |speedX * trackPos|$$

Figura 17 – Função de Recompensa 1

É possível observar o comportamento do gráfico da função em relação ao sensor *angle*, ilustrado na Figura 18. Quanto mais próximo de zero o ângulo estiver, maior será a recompensa recebida. Quando o sensor estiver próximo de zero a variação do carro em relação a pista será mínima, ou seja, o carro estaria em uma posição ideal, paralelo com as laterias da pista. Pode-se ressaltar que o valor do sensor *angle* varia entre $[-\pi, \pi]$.

Um problema enfrentado durante o desenvolvimento dos protótipos foi de não conseguirem controlar o carro em uma linha reta. O agente ativava totalmente o atuador do volante para a esquerda e para direita, isto é, valores de -1 e 1 na saída. Este comportamento pode estar atrelado à tentativa do controlador buscar sempre o máximo de recompensa, ou seja, manter sempre o ângulo do veículo em 0 e não conseguir mantê-lo, naquele momento, com movimentos mais suaves (valores próximos de 0 na saída). Para tentar reduzir as variações, foi colocado um intervalo arbitrário de -0.1 à 0.1 em que a recompensa era sempre 1. Isto, possivelmente, auxiliou a reduzir a variância do carro, pois, melhorou a condução em linhas retas e nas curvas.

A Função de Recompensa 1 apresentou um comportamento pouco promissor em alguns testes. Como ilustrado nos gráficos da Figura 19 rodando 180 episódios, algumas vezes o modelo

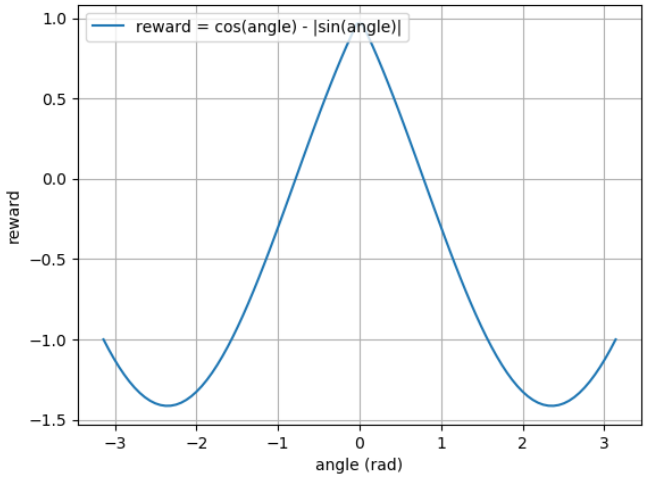


Figura 18 – Gráfico da Função de Recompensa 1 em relação ao sensor *angle*

recebe recompensa negativa por fazer avanços na distância total percorrida. A Figura 19 (a) mostra a distância percorrida pelo carro em cada episódio e a Figura 19 (b) mostra a média de recompensa recebida. Entre os episódios 11 e 15 a distância percorrida chegou a 140m, porém, a maior recompensa média recebida foi de -4,7, quando o carro percorreu 31m. Este exemplo fez com que uma nova função fosse pesquisada.

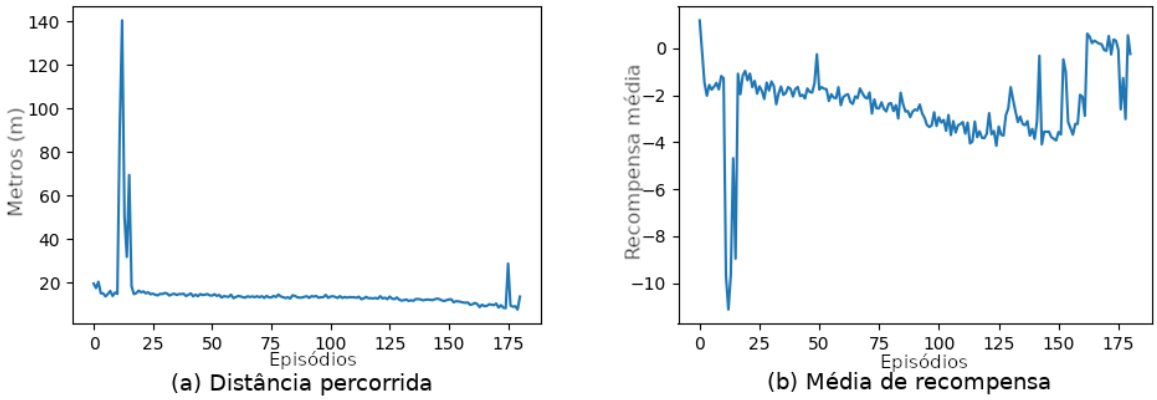


Figura 19 – Gráficos sobre correlação entre distância percorrida e média da recompensa para a função 1

Outras funções de recompensas foram desenvolvidas e testadas, totalizando 16 alterações, até chegar em uma função que apresentou uma evolução nos resultados. A função desenvolvida pelos autores possui semelhanças com a outra função e está descrita na Figura 20. A parte alterada trata da posição na pista. Ela trabalha de forma semelhante à Função de Recompensa 1, recompensando com o valor máximo quando está mais próxima do ângulo ideal. A modificação resultou em colocar o valor máximo de recompensa quando o carro está próximo posição ideal na pista sem penalizar muito quando ele está mais longe do centro.

$$R = speedX * cos(angle) - |speedX * sin(angle)| + speedX * cos(trackPos) - |speedX * sin(trackPos)|$$

Figura 20 – Função de Recompensa 2

A Função de Recompensa 2, apresentou um comportamento mais promissor que a Função de Recompensa 1 no mesmo teste. Na Figura 21 são apresentados os resultados após executar 180 episódios. A Figura 21 (a) mostra a distância percorrida pelo carro em cada episódio e a Figura 21 (b) mostra a média de recompensa recebida. Como esperado, a recompensa média recebida pelo controlador descreveu melhor a distância percorrida recompensando de maneira positiva as tentativas de avançar com o carro, mesmo que ainda não na posição ideal. O controlador completou uma volta no episódio 158 e completou o desafio de 10 voltas nos episódios 169, 170, 172 e 173.

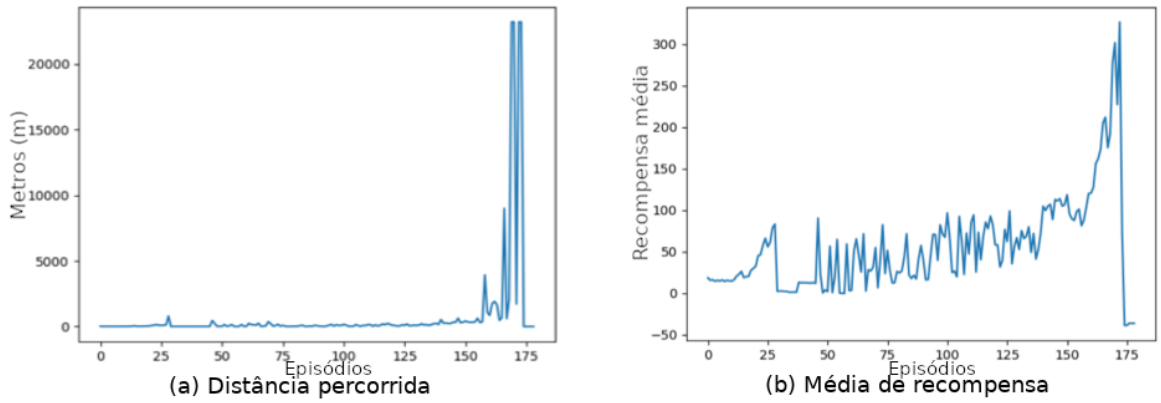
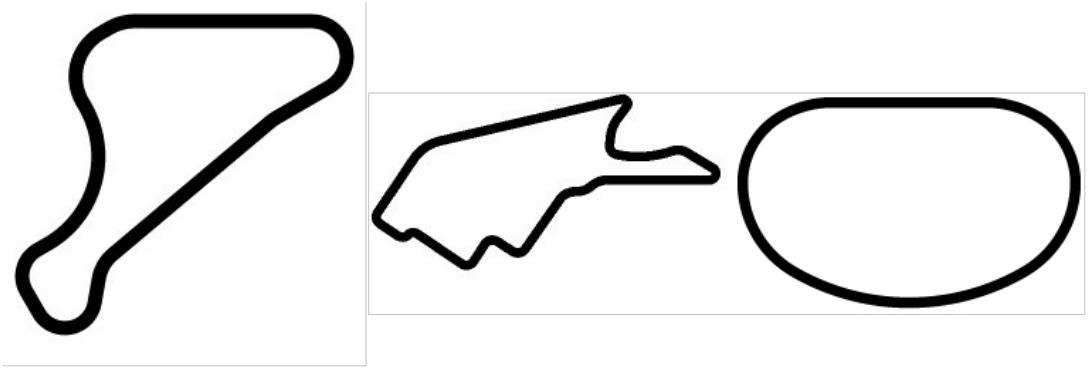


Figura 21 – Gráficos sobre correlação entre distância percorrida e média da recompensa

5.2.4 Avaliação dos modelos

A proposta foi fixada na capacidade do carro completar uma volta válida, ou seja, alcançar o estado final 4 da Tabela 5, no menor tempo (em segundos) possível. Esta proposta foi fixada dessa forma após a leitura de alguns artigos e notado a dificuldade extra em trabalhar com mais de um carro na pista. Os sensores de distância sofrem com interferências caso hajam outros carros e por isso é interessante desenvolver o carro sozinho.

Para a validação do treinamento do agente, foi necessário selecionar algumas pistas para realizar o aprendizado e outras para avaliação. A apresentação de circuitos diferentes é importante para evitar que o modelo proposto “memorize” um caminho e isso gere falsos entendimentos sobre o fato de "aprender a dirigir". Foram escolhidas pistas já utilizadas em outros trabalhos, como as utilizadas por Munoz; Gutierrez; Sanchis (29) ou Kim et al. (16): CG Speedway 1, Street 1 e D-Speedway, ilustradas na Figura 22.



(a) CG Speedway 1 (b) Street 1 (c) D-Speedway

Figura 22 – Pistas usadas para comparar os modelos

Para avaliação dos modelos testados, tanto de protótipo quanto finais, serão considerados estados finais da Tabela 5. Estes servem para medir a evolução do modelo e também são usados como parâmetros de finalização do treinamento.

6 RESULTADOS

6.1 PROTÓTIPOS

Uma série de modelos foram analisados para compor os resultados. Para realizar um teste mais aprofundado, foram escolhidos os modelos com 28 e 29 entradas, com as duas funções de recompensa apresentadas. Todos os testes foram tabelados e na ausência de algum dado utilizou-se o símbolo $*$. No caso onde não foi completada nenhuma volta [NC] - Não Completou - foi utilizado.

Cada teste foi executado por 100 episódios. O objetivo do controlador em cada episódio foi: completar 10 voltas na pista Michigan Oval, realizando-as no menor tempo possível. Os resultados são apresentados na Tabela 7, onde são mostradas as quantidade de voltas completadas, melhor volta válida e inválida completada pelo controlador e a média do tempo de volta. Os modelos avaliados na Tabela 7 foram treinados na própria pista Michigan Speedway. A título de comparação foi adicionada uma linha com um teste com um ser humano[1] fazendo 10 voltas e o recorde disponível no site do TORCS[2].

Tabela 7 – Execuções de testes nos modelos selecionados treinados na pista Michigan Speedway

Modelo	Quantidade de voltas	Melhor volta válida (s)	Melhor volta inválida (s)	Média geral de voltas (s)
28 entradas + Função 1	178	48,2	47,8	53,8
28 entradas + Função 2	918	32,8	34,7	34,4
29 entradas + Função 1	928	37,2	37,9	37,8
29 entradas + Função 2	908	37,0	36,9	38,5
Humano	10	32,2	32,6	32,8
USRobotics	*	29,5	*	*

O mesmo processo de avaliação foi utilizado para o treinamento na pista E-Track 5: executar 100 episódios com no máximo 10 voltas em cada. Os resultados são apresentados na Tabela 8, onde também foi adicionado o tempo de um ser humano para comparação. Percebe-se que o primeiro modelo não apresentou um resultado condizente com os outros modelos em ambos os testes. Estes resultados negativos podem estar ligados com o comportamento da Função de Recompensa 1, não recompensando os avanços mínimos e forçando o carro ser quase perfeito desde o início. Esta falta de incentivo inicial fez com que não fosse compensatório para o modelo testar novos padrões. A linha na Tabela 8, para este modelo, mostra onde o agente não completou nenhuma volta após 2000 episódios.

Para comparação os modelos treinados na pista Michigan Speedway foram testados na pista E-Track 5 sem nenhum treinamento extra. Os resultados para este segundo teste são apresentados na Tabela 9, onde são mostradas as quantidade de voltas completadas, melhor

[1] O teste foi feito pelo autor do trabalho

[2] `http://www.berniw.org/trb/tracks/track_view.php?viewtrackid=17`

Tabela 8 – Execuções de testes nos modelos selecionados treinados na pista E-Track 5

Modelo	Quantidade de voltas	Melhor volta válida (s)	Melhor volta inválida (s)	Média geral de voltas (s)
28 entradas + Função 1	[NC]	[NC]	[NC]	[NC]
28 entradas + Função 2	825	28,6	42,9	32,3
29 entradas + Função 1	949	32,4	32,0	33,8
29 entradas + Função 2	881	33,3	32,9	34,2
Humano	10	25,2	*	25,9

volta válida e inválida completada pelo controlador e a média do tempo de volta. Os modelos avaliados na Tabela 9 foram treinados na pista Michigan Speedway. Também foi adicionada uma linha com um teste com um ser humano fazendo 10 voltas, nenhuma volta inválida foi completada.

Tabela 9 – Execuções de testes nos modelos treinados na pista Michigan Speedway executando na pista E-Track 5

Modelo	Quantidade de voltas	Melhor volta válida (s)	Melhor volta inválida (s)	Média geral de voltas (s)
28 entradas + Função 1	11	44,2	*	44,3
28 entradas + Função 2	818	29,8	31,2	31,1
29 entradas + Função 1	874	34,6	34,7	39,4
29 entradas + Função 2	708	32,3	*	33,8
Humano	10	25,2	*	25,9

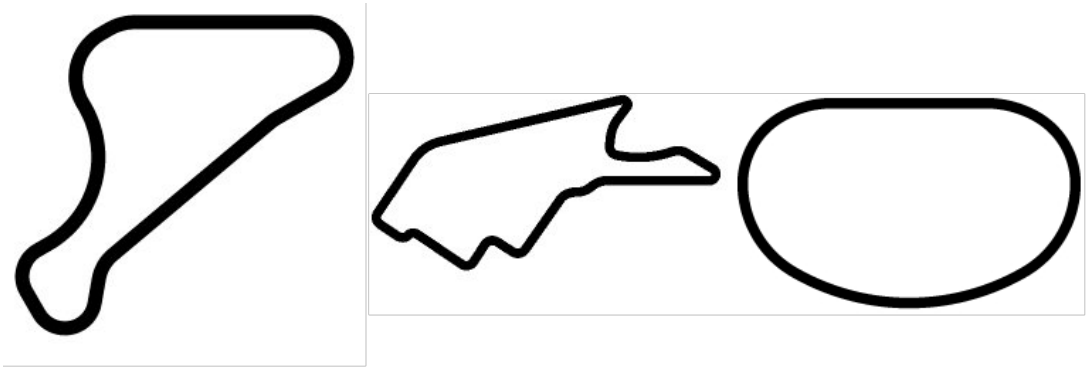
A Tabela 9 exemplifica a capacidade dos agentes, com exceção do primeiro modelo, em fazer curvas para ambos os lados, mesmo sendo ensinado em uma pista com curvas somente para a esquerda. Este comportamento pode estar relacionado com a necessidade de seguir o melhor traçado na pista, avaliado na função de recompensa, fazendo com que, desde o início, o agente realize movimentos para direita e esquerda. Com base nos testes realizado e comparações entre os modelos, o que apresentou um melhor resultado e evolução foi o modelo com 28 entradas e com a função de recompensa 2 treinado na pista Michigan Speedway.

6.2 MODELO FINAL

Com o melhor modelo dos testes selecionado, este foi treinado por mais 2000 episódios na pista E-Track 5 para que aprendesse outros comportamentos e assim ter seu desempenho avaliado. O controlador desenvolvido neste trabalho executou por 3 voltas nas pistas escolhidas, assim como Munoz; Gutierrez; Sanchis (29), para poder selecionar a melhor volta.

Os trabalhos selecionados para realização de comparação de tempo foram os trabalhos desenvolvidos por: Munoz; Gutierrez; Sanchis (29), Kim et al. (16) e Macedo et al. (24) - nos dados deste trabalho, não estão inclusos os melhores tempo, somente o total, por conta disso, será considerada a média do tempo. Também foi adicionada uma linha para cada pista com um

ser humano controlando por 10 voltas e colocando o melhor tempo. As comparações foram tabuladas na Tabela 10 onde mostram as três pistas que os trabalhos possuem: CG Speedway number 1 (Figura 23 (a)), Street 1 (Figura 23 (b)) e D-Speedway (Figura 23 (c) - Munoz; Gutierrez; Sanchis (29) não utiliza esta pista). Estas pistas foram escolhidas por serem cenários diferentes, para testar a capacidade de aprendizado do modelo, como pode ser visualizado na Figura 23.



(a) CG Speedway 1 (b) Street 1 (c) D-Speedway

Figura 23 – Pistas usadas para comparar os modelos

Tabela 10 – Comparação de resultados obtidos por outros trabalhos

Pista	**Modelo**	**Melhor volta válida (s)**
CG Speedway number 1	Munoz; Gutierrez; Sanchis (29)	47,4
CG Speedway number 1	Kim et al. (16)	47,3
CG Speedway number 1	Macedo et al. (24)	48,3
CG Speedway number 1	28 Entradas + Função 2	[NC]
CG Speedway number 1	Humano	37,2
Street 1	Munoz; Gutierrez; Sanchis (29)	93,5
Street 1	Kim et al. (16)	94,8
Street 1	Macedo et al. (24)	108,6
Street 1	28 Entradas + Função 2	[NC]
Street 1	Humano	74,1
D-Speedway	Munoz; Gutierrez; Sanchis (29)	*
D-Speedway	Kim et al. (16)	50,7
D-Speedway	Macedo et al. (24)	60,7
D-Speedway	28 Entradas + Função 2	[NC]
D-Speedway	Humano	40,4
CG Speedway number 1[3]	DanDroid Team[4]	31,3
Street 1[5]	wdbee robotics Team[6]	69,4

[3] http://www.berniw.org/trb/tracks/track_view.php?viewtrackid=10
[4] http://www.berniw.org/trb/teams/team_view.php?viewteamid=45
[5] http://www.berniw.org/trb/tracks/track_view.php?viewtrackid=38
[6] http://www.berniw.org/trb/tracks/track_view.php?viewtrackid=50

Após realizar mais treinamentos com o modelo que obteve o melhor resultado, foi notado um retrocesso na capacidade de dirigir do modelo. Os fatores principais foram: falta de capacidade de frenagem e movimentos com o volante virtual para direita e esquerda. A repetição dos movimentos faz com que o carro diminua a velocidade, comportamento que, provavelmente, repõe a falta da utilização do freio, retirado pelo ruído no início dos treinamentos. A mesma repetição de movimentos foi a causa das saídas e giros no próprio eixo na pista D-Speedway.

Ao perceber-se o comportamento de falta de freio foi refeito o treinamento com ruído em 20% dos passos, agora aplicado-o positivamente no freio, na pista CG Speedway number 1. A quantidade máxima de ruído foi a metade da capacidade total de freio do carro, para evitar que o carro ficasse parado sem conseguir avançar. Os resultados deste segundo treinamento ajudaram o carro avançar mais nas pistas CG Speedway number 1 e Street 1, mas ainda não foi o suficiente para fazer completar uma volta. O carro não virava mais o volante virtual para os lados nas retas por ter outra fonte de diminuir a velocidade, mas não era o suficiente para diminuição da velocidade do carro para fazer todas as curvas.

A partir dos resultados negativos, observados pela falta de capacidade de frenagem, mais uma vez foi refeito o treinamento, porém, com a capacidade máxima do freio disponível. Durante o novo treinamento foi completada uma volta na pista CG Speedway, mostrando possível evolução no modelo. Porém, esta evolução não foi confirmada, o modelo não conseguiu desenvolver o comportamento de frenagem por completo, ou seja, quando se aproximava de uma curva ele não freava. Por outro lado, o agente aperfeiçoou a capacidade de controle, sendo suficiente para realizar bons tempos na pista D-Speedway.

Observando este novo resultado, o agente foi colocado em novos testes na pista oval. Realizando o mesmo desafio, 100 episódios com 10 voltas cada, o modelo obteve tempos satisfatoriamente regulares e bons quando comparados com um humano, apresentados na Tabela 11. Por este motivo ele foi considerado o modelo final.

Tabela 11 – Execuções de testes no modelo final na pista D-Speedway

Modelo	Quantidade de voltas	Melhor volta válida (s)	Melhor volta inválida (s)	Média geral de voltas (s)
Modelo Final	807	41,6	41,7	44,2
Kim et al. (16)	*	50,7	*	*
Macedo et al. (24)	*	60,7	*	*
Humano	*	40,4	*	*

Um vídeo foi gravado para demonstrar uma volta do modelo na pista D-Speedway. O vídeo está disponível no link: `https://youtu.be/wVDzcKMc5d8`. Pode-se perceber uma pequena variação na movimentação do carro, o que pode ser tratado com mais treino na pista específica.

6.2.1 Teste com mais controladores

A fim de validar um caso base de um controlador de jogos de corrida, ele foi colocado junto com um outro carro para avaliar a capacidade de desviar, ou ultrapassar, um carro adversário. A estratégia de ultrapassagem é um ponto fundamental para o sucesso em uma corrida de carros, pois é necessária para subir de posições dentro de uma corrida. Todavia, conforme apresentado no trabalho de Loiacono et al. (22), esta tarefa pode ser muito complexa, necessitando de milhares de episódios para se conseguir um comportamento de ultrapassagem que os autores consideram bom. Por isso ela não foi considerada como primordial em um primeiro momento.

Pela falta de treinamento para essa tarefa específica, esperava-se que ele não conseguisse realizar com sucesso, o que de fato aconteceu. O modelo final não conseguiu desviar do outro carro, batendo nele nos testes realizados. Por mais que ele consiga se manter na trajetória após a batida, ele não consegue desviar e realizar a ultrapassagem.

6.3 DISCUSSÃO DOS RESULTADOS

Algumas partes importantes para um controlador, como o desenvolvimento de estratégias de *pit stop* e ultrapassagem, foram desconsideradas, desenvolvendo-se somente um agente para controlar o carro. Estes pontos faltantes, provavelmente, não serão desenvolvidos em uma única rede neural de controle do carro, sendo controladas por meio externo a rede que ativem uma rede neural secundária que irá tratar os casos específicos. O processo de desenvolvimento do controlador em módulos de atuação já foi realizado por outros trabalhos e este parece ser o processo mais cabível para um futuro próximo dos controladores.

Existem alguns problemas que foram enfrentados que podem ter sido cruciais para a falta de capacidade de testes mais abrangentes, como a baixa disponibilidade de recursos computacionais. Ganesh et al. (14) e Lau (19) utilizaram placas de processamento gráfico consideradas rápidas (NVIDIA GTX 1080 e NVIDIA Titan-X respectivamente) em comparação com o processador utilizado neste trabalho. Assim como Betz et al. (1) utilizam-se de uma NVIDIA Drive PX2, um hardware desenvolvido para carros autônomos. A aceleração dos cálculos proporcionado pela GPU poderia ter ajudado na quantidade de testes realizados, limitados pelo tempo ainda disponível após descobrir o problema de convergência já citado. Toda via, os resultados alcançados foram bons, quando comparados com outros trabalhos, mesmo sem possuir recursos mais avançados.

Outro fator que pode ter influenciado nos resultados finais foi a quantidade de vezes que o controlador foi treinado. Ganesh et al. (14) e Lau (19) não mostram qual foi a quantidade total de episódios ou passos que foram utilizados para conseguir os resultados que encontraram. Alterando algumas configurações de pistas e recompensas durante os treinamentos podem ser testadas para verificar se existe uma melhora na capacidade de direção.

A escolha da pista também pode ter influenciado no resultado final, ambos trabalhos

(Ganesh et al. (14) e Lau (19)) utilizam pistas mais complicadas para o início dos treinos. A escolha da pista mais simples mostrou a capacidade de aprendizado, porém, a escolha pode não ter sido boa para a generalização do modelo. Avaliar outras pistas pode ser um bom processo para a continuação do trabalho.

A redução na quantidade de entradas da rede neural se mostrou efetiva para o desafio proposto. A remoção da posição atual na pista foi trabalhada também no trabalho de Lau (19), que mostra que, ao retirar o mesmo sensor na função de recompensa, o agente começou a aprender a apex da pista. A aproximação do melhor caminho possível também foi desenvolvido no trabalho de Cardamone et al. (8), mostrando-se um fator importante para a equiparação dos modelos com Deep Q-Learning e os tradicionais. Pode-se testar novas formas de utilização das funções, como, por exemplo, alterando-as durante a execução e episódios, possibilitando avaliações diferentes dos cenários apresentados, e a remoção de mais sensores que talvez não afetem na qualidade do agente.

Existem indícios, tantos nos trabalhos já realizados quanto no caso da Roborace, que é possível criar um controlador autônomo. Não é possível saber qual técnica foi utilizada pela Roborace e as universidade parceiras para realizar o aprendizado, porém, segundo Betz et al. (1) foram utilizadas técnicas similares com uma máquina finita de estados.

Com a análise dos dados e resultados obtidos, pode-se considerar que, para os parâmetros testados, não foi possível criar um controlador completo com o Deep Q-Learning. Com tudo, ainda pode-se acreditar em um futuro com a utilização de combinação do métodos para se conseguir um controlador. Uma vez que, têm-se indícios que somente com uma rede não será possível realizar todas as ações, a utilização de módulos pode ser a mais bem aproveitada para o desenvolvimento. Há de se ressaltar a necessidade de mais processamento e memória RAM para esse processo, porém, com a melhoria dos componentes, é provável que seja possível realizar essa implementação.

7 CONCLUSÃO

Alcançou-se com o presente trabalho os objetivos propostos de se criar uma análise para o desenvolvimento de um controlador para o TORCS com Deep Q-Learning. A possibilidade de utilização desse método se mostrou inviável para criar um controlador completo por conta da quantidade de ações necessárias. Conseguiu-se alcançar um modelo que consegue pilotar um carro, porém, não realiza ultrapassagens ou controla consumo de combustível.

Contrapondo preconceitos, o agente consegue aprender a realizar curvas para ambos os lados, mesmo sendo ensinado em pistas com curvas somente para esquerda. A redução das entradas foi importante para encontrar novas formas de aprendizado na área de pesquisa e demonstrar uma capacidade de aprendizado mais variada.

Pode-se dizer que é possível que no futuro exista um controlador feito somente com inteligência artificial e que ele aprenda a fazer todas as operações separadas. Todavia, é necessário que haja evolução no campo de redes neurais ou que sejam desenvolvidas novas formas de trabalhar com as atuais. Enquanto mais evoluções não acontecem, entende-se como a melhor abordagem o fato de se utilizar métodos já tradicionais ou mesclar as duas, como já acontece com o trabalho da Roborace.

Observando os métodos já utilizados e os novos que estão surgindo, um novo controlador modular completo com o Deep Q-Learning pode ser desenvolvido como uma proposta de trabalho futuro. Bem como a utilização de mais pistas durante o processo de aprendizado, modificando a pista depois de alcançar uma meta estabelecida ou uma quantidade de episódios. Aliando a isso, pode-se testar outras funções de recompensa, bem como a alteração dela também após algumas iterações.

REFERÊNCIAS

1 BETZ, J. et al. A Software Architecture for an Autonomous Racecar. In: 2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring). [S.l.: s.n.], abr. 2019. p. 1–6. DOI: 10.1109/VTCSpring.2019.8746367.

2 BEVILACQUA, Fernando. **Ferramenta para Geração em Tempo Real de Bordas de Mapas Virtuais Pseudo-infinitos para Jogos 3D**. Jul. 2009. Diss. (Mestrado) – Universidade de Santa Maria, Brazil.

3 BIMBRAW, Keshav. Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology. **ICINCO 2015 - 12th International Conference on Informatics in Control, Automation and Robotics, Proceedings**, v. 1, p. 191–198, jan. 2015. DOI: 10.5220/0005540501910198.

4 CALDEIRA, Clara Marques. **TORCS Training Interface : uma ferramenta auxiliar ao desenvolvimento de pilotos do TORCS**. [S.l.: s.n.], dez. 2013. Monografia (graduação)—Universidade de Brasília, Brasília, 2013. Disponível em: <http://bdm.unb.br/handle/10483/6807>.

5 CARDAMONE, Luigi; LOIACONO, Daniele; LANZI, Pier Luca. Evolving competitive car controllers for racing games with neuroevolution. In: PROCEEDINGS of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09. [S.l.]: ACM Press, 2009. DOI: 10.1145/1569901.1570060.

6 _____. Interactive evolution for the procedural generation of tracks in a high-end racing game. In: ACM. PROCEEDINGS of the 13th annual conference on Genetic and evolutionary computation. [S.l.: s.n.], 2011. p. 395–402.

7 _____. Learning to Drive in the Open Racing Car Simulator Using Online Neuroevolution. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 2, n. 3, p. 176–190, set. 2010. ISSN 1943-068X. DOI: 10.1109/TCIAIG.2010.2052102.

8 CARDAMONE, Luigi et al. Searching for the optimal racing line using genetic algorithms. In: IEEE. PROCEEDINGS of the 2010 IEEE Conference on Computational Intelligence and Games. [S.l.: s.n.], 2010. p. 388–394.

9 CHEN, Chenyi et al. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. In: 2015 IEEE International Conference on Computer Vision (ICCV). [S.l.]: IEEE, dez. 2015. DOI: 10.1109/iccv.2015.312.

10 CSIKSZENTMIHALYI, Mihaly. Play and Intrinsic Rewards. **Journal of Humanistic Psychology**, SAGE Publications, v. 15, n. 3, p. 41–63, jul. 1975. DOI: 10.1177/002216787501500306.

11 D'ANGELO, G.; FERRETTI, S.; GHINI, V. Simulation of the Internet of Things. In: 2016 International Conference on High Performance Computing Simulation (HPCS). [S.l.: s.n.], jul. 2016. p. 1–8. DOI: 10.1109/HPCSim.2016.7568309.

12 ESPIÉ, Bernhard Wymann Eric et al. **TORCS, The Open Racing Car Simulator**. 2014.

13 G BELLEMARE, Marc; VENESS, Joel; BOWLING, Michael. Investigating Contingency Awareness Using Atari 2600 Games. **Proceedings of the National Conference on Artificial Intelligence**, v. 2, jan. 2012.

14 GANESH, Adithya et al. **Deep Reinforcement Learning for Simulated Autonomous Driving**. [S.l.]: Stanford University, 2016.

15 HAYKIN, Simon. **Neural Networks: A Comprehensive Foundation**. 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0132733501.

16 KIM, Kyung-Joong et al. Generalization of TORCS car racing controllers with artificial neural networks and linear regression analysis. **Neurocomputing**, v. 88, p. 87–99, jul. 2012. DOI: 10.1016/j.neucom.2011.06.034.

17 KOUTNI'K, Jan; SCHMIDHUBER, Jürgen; GOMEZ, Faustino. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In: ACM. PROCEEDINGS of the 2014 Annual Conference on Genetic and Evolutionary Computation. [S.l.: s.n.], 2014. p. 541–548.

18 KOUTNI'K, Jan et al. Evolving large-scale neural networks for vision-based torcs, 2013.

19 LAU, Yanpan. **Using Keras and Deep Deterministic Policy Gradient to play TORCS**. 2016.

20 LILLICRAP, Timothy P et al. Continuous control with deep reinforcement learning. **arXiv preprint arXiv:1509.02971**, 2015.

21 LOIACONO, Daniele; CARDAMONE, Luigi; LANZI, Pier Luca. Simulated Car Racing Championship: Competition Software Manual. **CoRR**, abs/1304.1672, 5 abr. 2013. arXiv: http://arxiv.org/abs/1304.1672v2 [cs.AI].

22 LOIACONO, Daniele et al. Learning to overtake in TORCS using simple reinforcement learning. In: IEEE Congress on Evolutionary Computation. [S.l.: s.n.], jul. 2010. p. 1–8. DOI: 10.1109/CEC.2010.5586191.

23 LOIACONO, Daniele et al. The WCCI 2008 simulated car racing competition. In: 2008 IEEE Symposium On Computational Intelligence and Games. [S.l.]: IEEE, dez. 2008. DOI: 10.1109/cig.2008.5035630.

24 MACEDO, B. H. F. et al. Evolving Finite-State Machines Controllers for the Simulated Car Racing Championship. In: 2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). [S.l.: s.n.], nov. 2015. p. 160–172. DOI: 10.1109/SBGames.2015.19.

25 MITCHELL, Thomas M. **Machine Learning**. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

26 MNIH, Volodymyr et al. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group, v. 518, n. 7540, p. 529, 2015.

27 MNIH, Volodymyr et al. Asynchronous methods for deep reinforcement learning. In: INTERNATIONAL conference on machine learning. [S.l.: s.n.], 2016. p. 1928–1937.

28 MNIH, Volodymyr et al. Playing Atari with Deep Reinforcement Learning. **CoRR**, abs/1312.5602, 2013. arXiv: 1312.5602. Disponível em: <http://arxiv.org/abs/1312.5602>.

29 MUNOZ, Jorge; GUTIERREZ, German; SANCHIS, Araceli. Controller for torcs created by imitation. In: IEEE. 2009 IEEE Symposium on Computational Intelligence and Games. [S.l.: s.n.], 2009. p. 271–278.

30 PEDERSEN, C.; TOGELIUS, J.; YANNAKAKIS, G. N. Modeling Player Experience for Content Creation. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 2, n. 1, p. 54–67, mar. 2010. ISSN 1943-068X. DOI: 10.1109/TCIAIG.2010.2043950.

31 QUADFLIEG, Jan et al. Learning the track and planning ahead in a car racing controller. In: IEEE. PROCEEDINGS of the 2010 IEEE Conference on Computational Intelligence and Games. [S.l.: s.n.], 2010. p. 395–402.

32 SALLAB, Ahmad EL et al. Deep Reinforcement Learning framework for Autonomous Driving. **Electronic Imaging**, Society for Imaging Science & Technology, v. 2017, n. 19, p. 70–76, jan. 2017. DOI: 10.2352/issn.2470-1173.2017.19.avm-023.

33 SALLAB, Ahmad El et al. End-to-end deep reinforcement learning for lane keeping assist. **arXiv preprint arXiv:1612.04340**, 2016.

34 SCHELL, J. **A Arte De Game Design: O Livro Original**. [S.l.]: Taylor & Francis, 2010. ISBN 9788535241983. Disponível em: <https://books.google.com.br/books?id=4spMYgEACAAJ>.

35 SUTTON, Richard S.; BARTO, Andrew G. **Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series)**. [S.l.]: A Bradford Book, 2015.

36 TOGELIUS, Julian; DE NARDI, Renzo; LUCAS, Simon M. Towards automatic personalised content creation for racing games. In: IEEE. 2007 IEEE Symposium on Computational Intelligence and Games. [S.l.: s.n.], 2007. p. 252–259.

37 TOGNETTI, Simone et al. Enjoyment Recognition from Physiological Data in a Car Racing Game. In: PROCEEDINGS of the 3rd International Workshop on Affective Interaction in Natural Environments. Firenze, Italy: ACM, 2010. (AFFINE '10), p. 3–8. ISBN 978-1-4503-0170-1. DOI: 10.1145/1877826.1877830. Disponível em: <http://doi.acm.org/10.1145/1877826.1877830>.

38 UHLENBECK, G. E.; ORNSTEIN, L. S. On the Theory of the Brownian Motion. **Phys. Rev.**, American Physical Society, v. 36, p. 823–841, 5 set. 1930. DOI: 10.1103/PhysRev.36.823. Disponível em: <https://link.aps.org/doi/10.1103/PhysRev.36.823>.

39 WATKINS, Christopher John Cornish Hellaby. **Learning from delayed rewards**. 1989. Tese (Doutorado) – King's College, Cambridge.

40 WYMANN, Bernhard et al. Torcs, the open racing car simulator. **Software available at http://torcs. sourceforge. net**, v. 4, n. 6, 2000.