UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO CÉSAR BARBOZA ROSSETTI

UMA ESTRATÉGIA DE FUTEBOL DE ROBÔS BASEADA EM APRENDIZADO POR REFORÇO

BAURU
Janeiro/2023

RODRIGO CÉSAR BARBOZA ROSSETTI

UMA ESTRATÉGIA DE FUTEBOL DE ROBÔS BASEADA EM APRENDIZADO POR REFORÇO

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Renê Pegoraro

BAURU
Janeiro/2023

R829e

Rossetti, Rodrigo César Barboza
Uma Estratégia de Futebol de Robôs Baseada em Aprendizado por Reforço / Rodrigo César Barboza Rossetti. -- Bauru, 2023
44 f. : il.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências, Bauru
Orientador: Renê Pegoraro

1. Inteligência artificial. 2. Aprendizado do computador. 3. Redes neurais (Computação). I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Rodrigo César Barboza Rossetti

Uma Estratégia de Futebol de Robôs Baseada em Aprendizado por Reforço

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Renê Pegoraro
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Profª. Dra. Simone das Graças Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Prof. Dr. Wilson Massashiro Yonezawa
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Bauru, 16 de janeiro de 2023.

Resumo

O uso de técnicas de aprendizado de máquina se popularizou muito nos últimos anos com avanços em diversas áreas como carros autônomos, geração de imagens e texto. O futebol de robôs, onde vários agentes interagem, é um ambiente propício para aplicação e verificação de técnicas de aprendizado. Neste trabalho foi aplicado técnicas de aprendizado por reforço para treinar uma estratégia de futebol de robôs completamente autônoma para mitigar os problemas da estratégia baseada em autômatos finitos determinísticos. Para tal, o ambiente do futebol de robôs do time Carrossel Caipira foi adaptado como um problema padronizado de aprendizado por reforço utilizando a biblioteca Gym e então treinado com o algoritmo Soft Actor-Critic com implementação da biblioteca Stable Baselines 3, por ser ideal para problemas com espaços de ação contínuos. Inicialmente, o goleiro foi treinado junto ao volante e atacante da estratégia anterior para fins de testes antes de continuar o treinamento junto ao volante e atacante atuais ao se constatar que o ambiente estava funcionando e o goleiro aprendendo. As recompensas obtidas durante o treinamento foram salvas e analisadas graficamente junto ao comportamento dos jogadores observado durante o treinamento. Para o goleiro, ele aprendeu a defender o gol conforme o aprendizado progrediu, mas para o volante e atacante é necessário mais tempo de treinamento para aprenderem bem suas posições. Também foi possível observar a tendência de posicionamento dos robôs em certas posições do campo para maximizar suas recompensas. O trabalho foi desenvolvido com base nas regras da competição IEEE Very Small Size Soccer da qual participa a equipe Carrossel Caipira.

Palavras-chave: Aprendizado profundo. Aprendizado por reforço. Futebol de Robôs.

Abstract

The use of machine learning techniques has become very popular in recent years with advances in several areas such as autonomous cars, image and text generation. Robot soccer, where several agents interact, is a favorable environment for the application and verification of learning techniques. In this work, reinforcement learning techniques were applied to train a completely autonomous robot soccer strategy to mitigate the problems of the strategy based on deterministic finite automata. To this end, the robot soccer environment of the Carrossel Caipira team was adapted as a standardized reinforcement learning problem using the Gym library and then trained with the Soft Actor-Critic algorithm with the implementation of the Stable Baselines 3 library, as it is ideal for problems with continuous action spaces. Initially, the goalkeeper was trained with the midfielder and striker of the previous strategy for testing purposes before continuing training with the current midfielder and striker after it was verified that the environment was working and the goalkeeper was learning. The rewards obtained during training were saved and graphically analyzed along with the players' behavior observed during training. For the goalkeeper, he learned to defend the goal as learning progressed, but for the midfielder and striker more training time is needed to learn their positions well. It was also possible to observe the tendency of positioning the robots in certain positions in the field to maximize their rewards. This work was developed based on the rules of the IEEE Very Small Size Soccer competition in which the Carrossel Caipira team participates.

Keywords: Deep learning. Reinforcement learning. Robot soccer.

Lista de figuras

Lista de abreviaturas e siglas

DQN	Deep Q Network
DNN	Deep Neural Network ou Rede Neural Profunda
RNA	Rede Neural Artificial
VSSS	Very Small Size Soccer
MDP	Markov Decision Process
IA	Inteligência Artificial
TD	Diferença Temporal
SAC	Soft Actor Critic

Sumário

1 Introdução

A cada dia aumentam os investimentos realizados em pesquisa na área de robótica. Algumas empresas de grande porte já realizam grandes investimentos no desenvolvimento de novas tecnologias. Segundo dados de pesquisas da Federação Internacional de Robótica, coletados e analisador por Jurkat, Klump e Schneider (2022), já existem cerca de três milhões de robôs operando em fábricas em todo o mundo e este número está em crescimento desde 2010. Além disso, cerca de US$ 13,2 bilhões foram gastos nos últimos anos em novas instalações. A média global é de 126 robôs para cada 10 mil trabalhadores na indústria, quase o dobro de 2015. Neste cenário, o Brasil tem nove robôs para cada 10 mil trabalhadores (International Federation of Robotics, 2021).

Hoje os robôs são cada vez mais proeminentes nas indústrias de manufatura, com partes – e as vezes, completos – processos de produção automatizados. A nova tecnologia robótica, incentivada pelo desenvolvimento da computação e tecnologias da informação e comunicação, tem profundas implicações para a organização da produção, comércio e empregos pela sua capacidade de adaptação e aprendizagem e de realização de tarefas precisas. O Boston Consulting Group sugere que uma verdadeira revolução robótica está prestes a acontecer, com muitas indústrias manufatureiras atingindo um ponto de inflexão em que a robotização se tornará comercialmente viável (BACKER et al., 2018).

Nesse contexto, uma importante área de pesquisa na robótica é a de sistemas com múltiplos robôs (do inglês, *Multi Robot Systems* - MRS), área emergente nas últimas três décadas. Nessa linha de pesquisa, é possível destacar sistemas de robôs cooperativos que interagem e compartilham um mesmo ambiente, tendo como finalidade atingir um objetivo global. A importância do estudo desses sistemas decorre do fato que o uso de múltiplos robôs permite uma maior capacidade de trabalho, o que pode viabilizar tarefas que não podem ser executadas por um único robô, reduzir o tempo de execução de tarefas e aumentar a confiabilidade do sistema (KHAN; RINNER; CAVALLARO, 2016).

O futebol de robôs é um domínio muito popular da área de sistemas de múltiplos robôs e tornou-se um meio para o estudo de tecnologias multiagentes, como cooperação, colaboração e coordenação (KIM; VADAKKEPAT, 2000).

Já há mais de duas décadas, o futebol de robôs tem sido tomado como referência para o estudo da robótica, devido à sua complexidade e abrangência de diversos assuntos dentro da robótica. Segundo Lund e Pagliarini (1998), o futebol de robôs tem sido definido como um novo marco para a inteligência artificial. Em contraste com desafios de inteligência artificial anteriores, como xadrez, o futebol de robôs é um jogo físico e dinâmico, em que controle em tempo real é essencial.

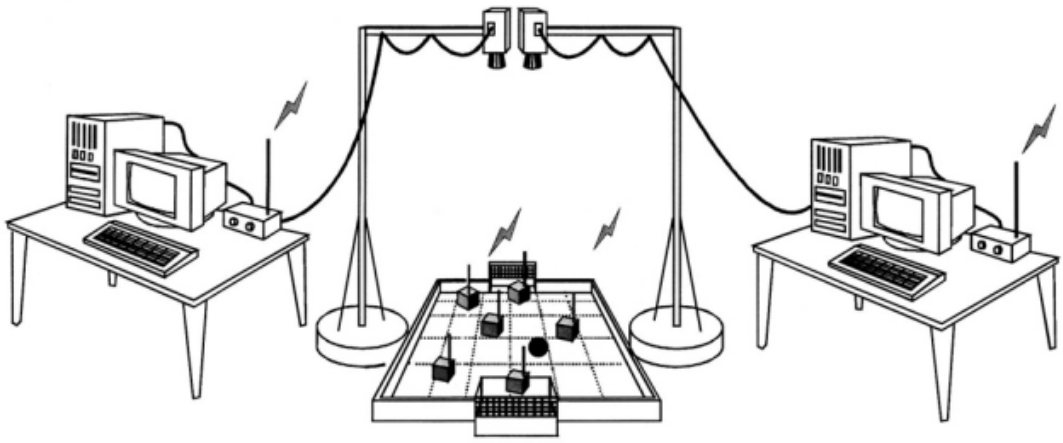
Espera-se que as competições como de futebol de robôs forneçam um ambiente adequado para testar e comparar tecnologias e algoritmos entre si para determinar qual obtém melhor desempenho em determinada tarefa, e fomentem novas pesquisas na área.

1.1 Competição

O Departamento de Computação da Faculdade de Ciências da UNESP, campus de Bauru, participa de competições de futebol de robôs, na modalidade de micro robôs (chamada atualmente IEEE *Very Small Size Soccer* - VSSS), desde 1998, quando alcançou o vice-campeonato na 1ª Copa Brasil de Futebol de Robôs - CBFR'98, e o título de vice-campeão mundial no futebol de robôs na categoria MIROSOT (*Micro-Robot Soccer Tournament*) da FIRA (*Federation of International Robot-soccer Association*), nas competições da FIRA realizadas na França.

Nesta categoria, as partidas são disputadas por dois times, formados por três robôs que devem caber em um cubo com arestas de 7,5 cm. Os robôs devem ser controlados de forma autônoma e remota por um computador, sem intervenção humana, exceto caso seja necessário o reposicionamento manual dos robôs, como em uma situação de pênalti, por exemplo. As partidas são disputadas em um campo de 150cm x 130cm, onde os robôs interagem, com o objetivo de defender o próprio gol e levar a bola até o gol adversário.

Figura 1 – Ambiente do futebol de robôs



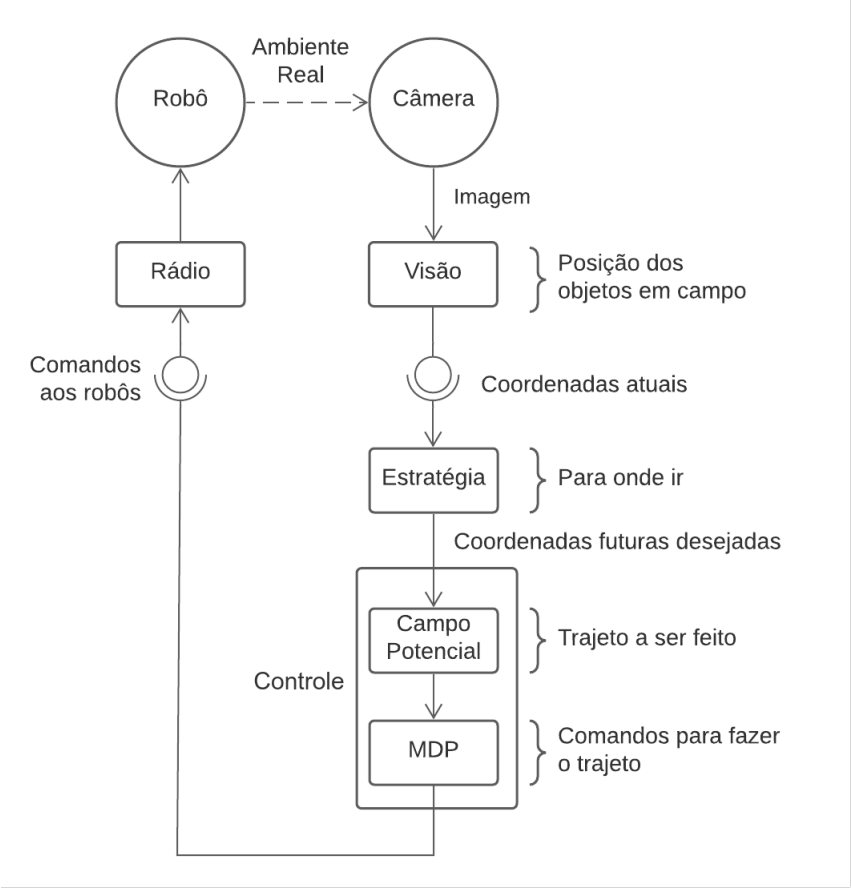
Fonte: CBROBOTICA (2022)

No ambiente (Figura 1), cada time posiciona uma câmera a 2m sobre o campo e alinhada ao seu centro para capturar as imagens do campo. Os robôs são identificados por meio de pares de etiquetas coloridas colocadas em cima deles. A cor da primeira etiqueta é usada para identificar o time: azul ou amarelo; a segunda é usada para identificar o robô. Cada etiqueta deve caber em um quadrado de lado 3,5 cm ou em um círculo de raio 4 cm. A bola utilizada é laranja e tem aproximadamente 42.7mm de diâmetro e 46g de massa (CBROBOTICA, 2022).

1.2 Sistema do Carrossel Caipira

O software do futebol de robôs pode ser dividido em três módulos fundamentais para que os robôs possam locomover-se autonomamente (11), de forma que o sistema consiga responder às seguintes questões: "onde está?", "para onde vai?" e "como vai?". Cada módulo do sistema responderá uma dessas perguntas para obter a "resposta" e fazer com que o robô entenda o comando final (ROSSETTI et al., 2020).

Figura 2 – Diagrama simplificado dos módulos do time Carrossel Caipira no ambiente real



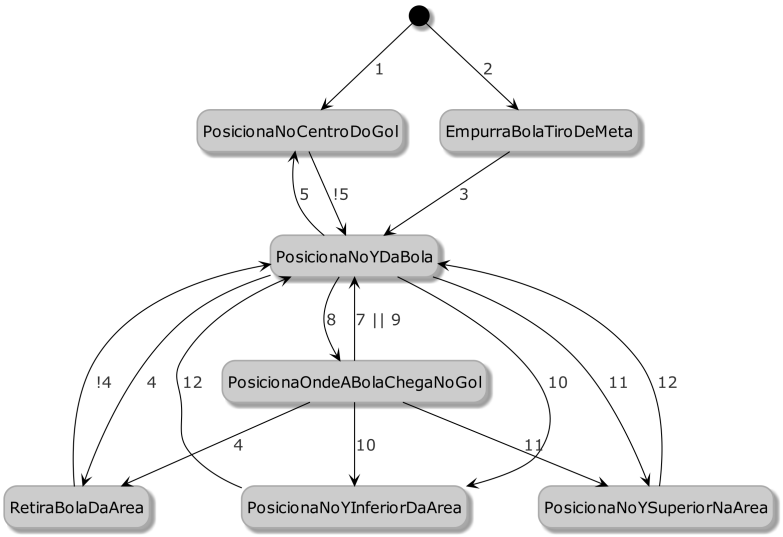
Fonte: Rossetti et al. (2020)

As imagens capturadas e digitalizadas pela câmera são processadas pelo módulo VISÃO, que calcula, a partir das cores das etiquetas e da bola, o vetor $(x, y, \hat{a}ngulo)$ dos robôs e da bola em campo. Esses dados são passados para o módulo ESTRATÉGIA, que determina o vetor $(x, y, \hat{a}ngulo, velocidade)$ objetivo para onde os robôs devem se locomover para realizar a estratégia de defesa ou ataque definida. O módulo CONTROLE transforma esse vetor objetivo em comandos enviados às rodas via rádio, de forma que os robôs consigam chegar ao objetivo desejado.

O módulo ESTRATÉGIA atual do time Carrossel Caipira de futebol de robôs foi desenvolvido com base no paradigma de programação orientada a autômatos – os comportamentos

dos robôs são representados por autômatos finitos determinísticos (AFD). Cada estado desses autômatos descreve um objetivo para os robôs, e as arestas conectando os estados são as condições de trocas de estados.

Figura 3 – Máquina de estados do goleiro



Fonte: Rossetti et al. (2020)

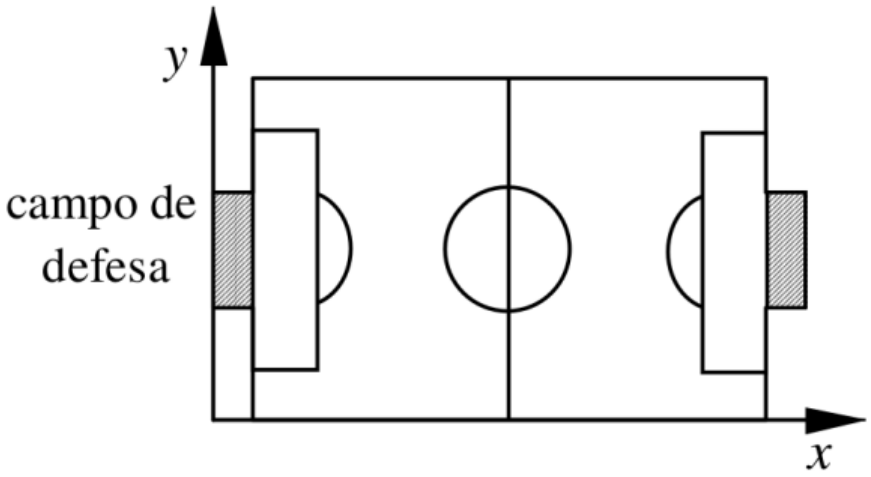
Quadro 1 – Condições de trocas de estados utilizadas pela estratégia.

Condição	Descrição
1	Inicia o autômato, ativando um estado inicial
2	Condição de tiro de meta satisfeita
3	Fim do tiro de meta
4	Bola posicionada dentro da nossa área de gol
5	Bola posicionada dentro do campo de ataque
6	Bola posicionada no campo superior
7	Bola indo em direção ao gol adversário
8	Bola vindo em direção ao nosso gol
9	A bola não está em movimento
10	A bola chegará no nosso gol por baixo de fora da área
11	A bola chegará no nosso gol por cima de fora da área
12	A bola chegará no nosso gol pelo centro de fora da área
13	Atacante posicionado à frente do volante
14	Bola posicionada à frente do volante
15	Bola posicionada à frente do atacante
16	O atacante está alinhado com a bola
17	Volante posicionado próximo da área do gol adversário

Fonte: Rossetti et al. (2020)

Como exemplo, a Figura 3 descreve o comportamento do goleiro. Conforme descrito na Quadro 1, os números nas arestas são condições que devem ser satisfeitas para que o goleiro mude de um estado para outro. Por exemplo, a condição 1 é o inicio normal do autômato, a 2 é o inicio em caso de tiro de meta, e a 10 é previsão de que a bola chegará no gol por baixo de fora da área, ao considerar o sistema de coordenadas do campo como o apresentado na Figura 4.

Figura 4 – Sistema de coordenadas usado no time Carrossel Caipira



Fonte: Rossetti et al. (2021)

1.3 Justificativa

O futebol de robôs é um ambiente complexo onde inúmeros estados e ações são possíveis a cada instante de tempo, exige cooperação entre os robôs e a antecipação das ações realizadas pelos oponentes.

Este ambiente caótico torna difícil a criação de autômatos para descrever o comportamento dos robôs. A Figura 3 descreve o comportamento do goleiro com 7 estados e 16 arestas, mas conforme o número de estados aumenta, mais arestas são necessárias para conectar os novos estados aos pré-existentes. A estratégia do atacante, por exemplo, tem 10 estados e 34 arestas.

O problema de se ter autômatos muito grandes é que torna-se mais difícil depurar os estados e transições e modificar o autômato de forma coerente com os estados existentes. Além disso, a representação visual da estratégia como mostrado na Figura 3, em que seja fácil identificar o comportamento do robô, fica inviável.

Outro problema é que os autômatos utilizados são determinísticos, ou seja, para cada instante de tempo, ou o robô permanece no estado atual ou apenas uma transição será

verdadeira e o robô mudará de estado. É difícil garantir que isso é sempre verdade, então em algumas situações alguns estados podem ficar inalcançáveis.

Mudar os autômatos para não-determinísticos levará ao problema oposto: as arestas precisarão ser criadas considerando que em uma situação dois ou mais estados podem ser ideais, então escolhe-se um aleatoriamente entre eles. Porém, pode acontecer que em uma situação específica e não considerada, uma aresta que idealmente não deveria ser verdadeira, torne-se verdadeira, e ela seja escolhida para o robô trocar de estado. De qualquer forma é fundamental pensar as arestas com cuidado para que isso não ocorra, mas como o ambiente é caótico, alguma situação pode não ser identificada, resultando em situações não tratadas. Uma forma de mitigar isto é utilizar aprendizado por reforço para otimizar uma estratégia para os jogadores independente da situação encontrada por eles.

Utilizar sub-estados seria uma forma de organizar melhor hierarquicamente a máquina de estados dos jogadores, isolando situações em estados e tratando ações dos jogadores como arestas que levam a sub-estados. Porém, isto ainda não resolve o problema de depuração das arestas em situações raras de jogo, e também aumenta ainda mais o número de estados e ações.

Mesmo assim, com três robôs a estratégia dos autômatos obtém resultados razoáveis e apesar da sua complexidade não é difícil de modificá-la. Porém conforme o número de robôs aumenta, a sua viabilidade cai, pois a complexidade dos autômatos e interações aumenta exponencialmente e torna-se mais difícil aprimorá-los.

Nesse trabalho foi treinada uma estratégia de futebol de robôs completamente autônoma utilizando aprendizado por reforço, a fim de mitigar esses problemas e possibilitar a melhoria do desempenho do time.

Jogos como o futebol de robôs são frequentemente ambientes de referência para o teste, treinamento e comparação de algoritmos de aprendizado por reforço, por serem ambientes controlados e simples, mas que permitem alta complexidade. Além disso, os modelos treinados podem ser transferidos para outros domínios.

1.4 Objetivos

O principal objetivo desse trabalho é aplicar técnicas de aprendizado por reforço para treinar uma estratégia de futebol de robôs a fim de lidar com situações não tratadas e caóticas, difíceis de serem previstas com a estratégia atual, além de possibilitar a melhoria do desempenho do time Carrossel Caipira em competições de futebol de robôs.

Os objetivos secundários incluem:

- Adaptar o sistema da equipe Carrossel Caipira para que possa ser modelado como um

problema de aprendizado por reforço.

- Aplicar técnicas de aprendizado por reforço na estratégia do futebol de robôs.
- Treinar os robôs utilizando essas técnicas e analisar os resultados obtidos.

2 Revisão Bibliográfica

Aprendizado de máquina é um método para ensinar computadores, programas ou agentes a realizarem uma tarefa sem serem explicitamente programados. É uma área focada no desenvolvimento de algoritmos para analisar e fazer previsões baseadas em dados de treinamento fornecidos previamente. Os três paradigmas de aprendizado de máquina são: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço. Cada um foca em uma forma diferente de aprendizado.

Segundo Sutton e Barto (2018), no aprendizado por reforço, um agente realiza ações em um ambiente e recebe um sinal de recompensa ou penalidade dele, que é então utilizado para atualizar seu comportamento. Um exemplo de aprendizado por reforço é um carro autônomo, que aprende a navegar por uma cidade sendo recompensado por dirigir com segurança e eficiência e penalizado por quebrar as leis de trânsito.

Ainda de acordo com Sutton e Barto (2018), aprendizado por reforço difere do aprendizado supervisionado na medida em que este utiliza um conjunto de exemplos rotulados fornecidos por um supervisor externo para treinamento. O rótulo associado a uma situação é a ação correta que o sistema deve tomar naquela situação. O objetivo do treinamento é extrapolar ou generalizar respostas para situações não presentes no conjunto de treinamento, por exemplo, através de regressão. Aprendizado por reforço também difere do aprendizado não supervisionado. Neste caso, o objetivo é encontrar estruturas ou padrões ocultos em um conjunto de dados não rotulados, por exemplo através do agrupamento de dados. Apesar de ambos serem métodos não supervisionado, o objetivo do aprendizado por reforço é maximizar um sinal de recompensa.

Uma das mais conhecidas aplicações de aprendizado por reforço em jogos é o AlphaGo, um programa de computador desenvolvido pela DeepMind que foi capaz de derrotar jogadores humanos no jogo de tabuleiro Go. O AlphaGo usou uma combinação de redes neurais profundas e aprendizado por reforço para aprender como jogar e tomar decisões sobre quais movimentos fazer. O programa foi capaz de aprender e melhorar com o tempo por meio do autojogo, tornando-se um dos melhores jogadores de Go do mundo ao vencer o então campeão europeu de Go, Fan Hui, por 5 a 0 (SILVER et al., 2016).

Outros exemplos do uso de aprendizado por reforço em jogos incluem o treinamento de Inteligências Artificiais (IAs) para jogar videogames, como Atari (MNIH et al., 2013) ou jogos de tiro em primeira pessoa (KEMPKA et al., 2016), e o desenvolvimento de IAs que podem aprender a jogar jogos de tabuleiro estratégicos, como xadrez ou shogi (SILVER et al., 2018). Em todos esses casos, os algoritmos de aprendizado por reforço permitem que a IA aprenda por meio de tentativa e erro, melhorando seu desempenho ao longo do tempo à medida que

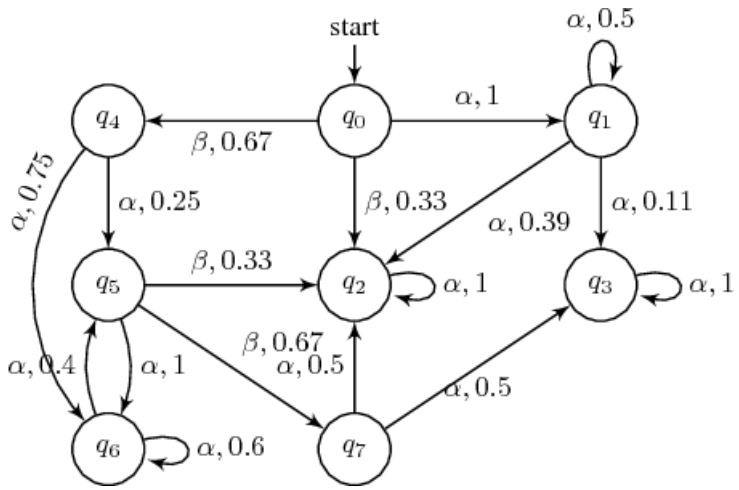
aprende a maximizar sua recompensa.

2.1 Processos de Decisão de Markov

Um processo de decisão de Markov, ou MDP (*Markov Decision Process*) é uma estrutura matemática para modelar tomada de decisão frequentemente usado em aprendizado por reforço. É uma maneira clara de modelar o problema de aprender através da interação para atingir um objetivo.

A Figura 5 apresenta um exemplo de MDP, com estados $Q = \{q_i, i = 0, ..., 7\}$, ações $\sum = \{\alpha, \beta\}$ e função de probabilidade P. Essa MDP possui 7 estados, sendo o estado q_0 inicial. Para cada estado atual, pode-se realizar uma das ações possíveis e trocar para outro estado com probabilidade p_i: por exemplo, no estado q_0, pode-se realizar a ação α e chegar no estado q_1 com probabilidade 1, ou realizar a ação β, que pode levar ao estado q_4 com probabilidade 0,67 ou ao estado q_2 com probabilidade 0,33.

Figura 5 – Exemplo de um processo de decisão de Markov



Fonte: Fu e Topcu (2014)

A Figura 3 do autômato do goleiro pode ser entendida como uma MDP. No entanto, em vários problemas como no futebol de robôs, a MDP do ambiente é desconhecida e não é possível contruí-la previamente. Nesse caso, a Figura 3 é apenas uma aproximação da MDP real. A vantagem de utilizar aprendizado por reforço é aprender a MDP à medida que o agente interage com o ambiente (SUTTON; BARTO, 2018).

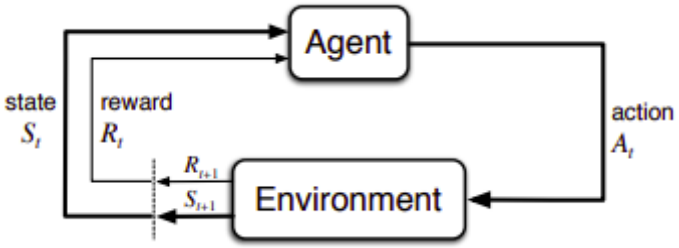
2.1.1 A Interface Agente-Ambiente

O aprendizado por reforço lida com o treinamento de um tomador de ações, chamado de agente, imerso em um ambiente aprendendo através do método de tentativa e erro. O ambiente fornece ao agente informações sobre o estado que se encontra e o agente toma

ações no ambiente de modo a modificar seu estado. Ao executar uma ação no ambiente, o agente também recebe um valor de recompensa, que pode ser positivo ou negativo e que diz ao agente o quão bom sua ação foi para o sucesso da tarefa. Dessa forma, a tarefa do agente de aprendizado por reforço é maximizar a recompensa cumulativa no ambiente no qual é treinado (KAELBLING; LITTMAN; MOORE, 1996).

Essa interação entre agente e ambiente pode ser modelada como um processo de decisão de Markov, como mostrado na Figura 6. Nesse caso, o agente recebe um estado S_t, com base nesse estado realiza uma ação A_t, e o ambiente retorna um novo estado S_{t+1} e uma recompensa R_t.

Figura 6 – Interação entre agente e ambiente em um processo de decisão de Markov



Fonte: Sutton e Barto (2018)

No domínio do futebol de robôs, os **agentes** são os três robôs controlados por cada time. O **ambiente** inclui o campo onde os robôs se movimentam, a bola e o juiz. Ou seja, ele inclui as regras que definem o comportamento do ambiente, como a checagem de situações de falta pelo juiz, o reposicionamento em caso de falta e a delimitação do campo, como o gol, a área do gol, a linha do meio de campo, etc.

As **ações** que os robôs podem tomar dependem da granularidade do sistema. Ou seja, é possível considerar que em baixo nível as ações são as velocidades de rotação de cada roda dos robôs para que eles se movimentem em campo, ou que em alto nível são as coordenadas x e y do objetivo do robô. Como o projeto é focado na parte da estratégia e não do controle do time, será considerado como ação do robô o vetor objetivo $(x, y, \hat{a}ngulo, velocidade)$, isto é, a posição x e y em campo em que ele deve chegar, o $\hat{a}ngulo$ que ele deve assumir ao chegar nessa posição, e a $velocidade$ que ele deve utilizar para chegar nessa posição. Esse vetor será passado para o controle, que então será responsável por utilizar esse objetivo para movimentar o robô em campo e fazê-lo chegar na posição (x, y) desejada com $(\hat{a}ngulo, velocidade)$ desejados, conforme a arquitetura mostrada na Figura 2.

O **estado** deve descrever unicamente a "situação" atual do jogo, e é composto de informações extraídas do ambiente ou do agente, como a posição e ângulo dos jogadores e da bola em campo, a imagem capturada pela câmera, informações de falta, como tipo e lado, a cor de cada time, dados de sensores, etc. O conjunto desses dados descreve o estado do

agente, mas nem todos esses dados são úteis, então alguns podem ser removidos para agilizar o treinamento.

A **recompensa** depende da tarefa na qual o robô está sendo treinado. Por exemplo, para o goleiro, pode-se definir uma recompensa de -100 para cada gol recebido, e para o atacante uma recompensa de +100 para cada gol feito.

2.1.2 MDPs Finitas

Formalmente, dado um conjunto de estados, ações e recompensas ($\mathcal{S}$, $\mathcal{A}$, e $\mathcal{R}$) finitos, as variáveis aleatórias R_t e S_t tem uma distribuição de probabilidade discreta bem definida dependente apenas do estado e ação anterior. Isto é, a probabilidade p do agente sair de um estado s e ir para um estado s' ao realizar uma ação a, e receber uma recompensa r é (SUTTON; BARTO, 2018):

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{2.1}$$

A soma das probabilidades deve ser 1:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1 \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}(s). \tag{2.2}$$

A partir dessa função de quatro argumentos p é possível computar qualquer coisa sobre o ambiente. Por exemplo, a probabilidade de transição de estado, isto é, a probabilidade de dado um estado s e ação a, chegar em um estado s' é:

$$p(s'|s, a) \doteq Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a) \tag{2.3}$$

Também é possível computar a recompensa esperada para um par de estado-ação, isto é, a recompensa esperada ao tomar a ação a em um estado s:

$$r(s, a) \doteq E\{R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'\} = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a) \tag{2.4}$$

Isso diz o quão "bom" é tomar essa ação nesse estado.

2.1.3 Objetivos e Recompensas

Em aprendizado por reforço, o objetivo do agente é formalizado em termos de um sinal especial, passado em cada instante de tempo do ambiente ao agente, chamado de recompensa. Formalmente, o objetivo do agente é maximizar a recompensa total que recebe (SUTTON; BARTO, 2018).

Segundo Sutton e Barto (2018), para que o agente aprenda uma tarefa, é preciso que ele não aprenda apenas a escolher a melhor ação imediata com base na recompensa do próximo

estado, e sim com base na recompensa acumulada de todos os estados a longo prazo. Dessa forma, o agente não será "cego", isto é, mesmo que a recompensa de um estado s_1 seja menor que a de outro s_2, é possível que a recompensa dos estados alcançáveis a partir de s_1 compense essa diferença, então o agente deve escolher a ação que leva ao estado s_1;

Essa é a noção de *retorno esperado* G_t:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.5}$$

Onde R_T é um *estado terminal*, delimitando o fim da tarefa. Tarefas que possuem estados terminais são episódicas. Considera-se um **episódio** a sequência de ações necessárias para o agente sair de um estado inicial e chegar a um estado final. Por exemplo, no futebol de robôs um episódio é delimitado por um gol. Contrariamente, tarefas **contínuas** são infinitas (SUTTON; BARTO, 2018).

Essa fórmula 2.5 tem um problema de não funcionar com tarefas contínuas e não considerar o tempo necessário para chegar em um estado bom. Espera-se que chegar em um estado em 10 ações seja melhor do que em 100 ações. Para resolver esse problema pode-se multiplicar cada recompensa por uma taxa de desconto γ, em que $0 < \gamma \leq 1$. A nova função de retorno descontado é (SUTTON; BARTO, 2018):

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.6}$$

A taxa de desconto define o valor presente de recompensas futuras. Para valores de γ próximos de 0, o agente pode aprender a escolher ações a fim de maximizar recompensas mais imediatas, desconsiderando as recompensas futuras, e para valores próximos de 1 ele pode aprender a escolher ações piores imediatamente mas que levem a estados melhores no futuro. O γ é um parâmetro que deve ser escolhido e otimizado para o problema em questão (SUTTON; BARTO, 2018).

2.1.4 Políticas e Funções de Valor

Segundo (SUTTON; BARTO, 2018), os algoritmos de aprendizado por reforço envolvem estimar funções de valor que avaliam o quão bom é para o agente estar em um certo estado, ou o quão bom é realizar uma ação em um certo estado. Essa noção de "quão bom" pode ser definida como o retorno esperado como definido em 2.6. Essas funções são definidas com base em formas de agir, chamadas **políticas**.

Formalmente, uma política é o mapeamento de estados para probabilidades de selecionar cada ação possível. Se o agente está seguindo uma política π em um instante de tempo t, então $\pi(a|s)$ é a probabilidade de que $A_t = a$ se $S_t = s$. Políticas comuns são a aleatória, em que todas as ações possuem a mesma probabilidade de serem escolhidas e a determinística, em que sempre a mesma ação é escolhida para o estado (SUTTON; BARTO, 2018).

Funções de valor (ou valor de estado) estimam o retorno esperado dado o estado atual s e considerando que o agente siga a política π, ou seja, o quão bom é estar em um estado s sob a política π (SUTTON; BARTO, 2018):

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right], \forall s \in (S) \tag{2.7}$$

Analogamente, pode-se estimar o retorno esperado de tomar uma ação a em um estado s, considerando que o agente siga a política π, ou seja, o quão bom é escolher a ação a no estado s sob a política π:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a \right] \tag{2.8}$$

Essa função $q_\pi(s,a)$ é chamada de função de valor de ação (ou valor de estado-ação) e é a base dos algoritmos baseados em *Q-learning* que serão discutidos na subseção 2.4.1.

A função de valor de estado satisfaz uma propriedade recursiva. Ao abrir essa expressão obtém-se a seguinte equação (SUTTON; BARTO, 2018):

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s'] \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[r + \gamma v_\pi(s') \right], \forall s \in \mathcal{S} \end{aligned} \tag{2.9}$$

A equação 2.9 é conhecida como **equação de Bellman** para $v_\pi(s)$. Ela expressa o valor de um estado como o valor dos estados subsequentes. Ela é a média ponderada de todos os retornos esperados com suas probabilidades de acontecerem. Ela também diz que o valor do estado inicial deve ser igual ao valor descontado do próximo estado esperado, mais o valor dos estados alcançáveis a partir dele (SUTTON; BARTO, 2018).

Segundo Sutton e Barto (2018), para os problemas de aprendizado se considera que existe uma política com uma função de valor de estado e valor de ação ótimas que são máximas entre todas as políticas possíveis. O objetivo do aprendizado por reforço é aproximar uma política da política ótima e as funções de valor das funções de valores ótima.

2.2 Programação Dinâmica

Segundo Cormen et al. (2022), programação dinâmica é um método de resolução de problemas de otimização que se baseia em quebrá-los em subproblemas menores, então resolver cada subproblema uma vez, e armazenar as soluções dos subproblemas em uma tabela. As

soluções dos subproblemas podem então ser usadas para construir uma solução para o problema original.

Pode-se obter a política ótima após encontrar as funções de valor de estado e ação ótimas que satisfazem as equações de otimalidade de Bellman (SUTTON; BARTO, 2018):

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \end{aligned} \tag{2.10}$$

ou

$$\begin{aligned} q_*(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{q_*}(s', a')] \end{aligned} \tag{2.11}$$

Os dois métodos fundamentais para resolver MDPs usando programação dinâmica são iteração de política e iteração de valor.

Segundo Sutton e Barto (2018), no contexto do aprendizado por reforço, programação dinâmica é usado para computar políticas ótimas dado um perfeito modelo do ambiente como uma MDP. Isso na maioria dos problemas, incluindo o futebol de robôs, é inviável devido a sua alta complexidade, e logo a aplicabilidade é limitada a problemas simples .

A programação dinâmica exige que todo o problema seja conhecido, incluindo as transição e a função de recompensa. No aprendizado por reforço, o agente normalmente interage com o ambiente em tempo real e aprende sobre as transição e a função de recompensa à medida que avança. Isso significa que o problema não é conhecido por completo e a solução não pode ser pré-computada usando programação dinâmica. Outro problema é que geralmente é necessário o uso de uma tabela ou *array*. Isso pode consumir muita memória para problemas com muitos estados e ações possíveis (KALEJAIYE, 2019).

2.3 Métodos de Monte Carlo

Segundo Sutton e Barto (2018), os métodos de Monte Carlo são algoritmos usados no aprendizado por reforço para estimar o valor de um estado ou ação, amostrando os resultados de vários episódios escolhidos aleatoriamente e calculando a média de suas recompensas. Isso é uma vantagem sobre a programação dinâmica pois o conhecimento completo do problema não é mais necessário.

Uma das principais vantagens dos métodos de Monte Carlo é que eles não necessitam de conhecimento ou um modelo do ambiente, sendo capazes de aprender diretamente da experiência bruta. Isso os torna mais flexíveis, mas também significa que podem exigir uma grande quantidade de amostras para aprender com precisão (SUTTON; BARTO, 2018).

2.3.1 Treinamento *Off-Policy* e *On-Policy*

Métodos em aprendizado por reforço podem ser classificados em *off-policy* ou *on-policy*. Métodos *on-policy* aprendem o valor de ações e estados seguindo a política atual, e atualizando essa mesma política enquanto aprende. Métodos *off-policy* ao contrário, aprendem o valor de ações e estados avaliando as consequências de ações tomadas sob uma política diferente (SUTTON; BARTO, 2018).

Aprendizado *off-policy* pode ser mais eficiente do que *on-policy* pois permite que o agente aprenda de um conjunto maior de experiências, incluindo experiências impossíveis sob a política atual. No entanto isso também torna mais difícil de aprender com precisão pois os valores são estimados baseados em uma política diferente (SUTTON; BARTO, 2018).

2.3.2 Exploração e Aproveitamento

De acordo com Sutton e Barto (2018), o dilema entre exploração (*exploration*) e aproveitamento (*exploitation*) é um problema comum no aprendizado por reforço, no qual o agente deve equilibrar a necessidade de explorar o ambiente, isto é, tentar novas ações que podem levar a melhores resultados, e aproveitar seu conhecimento atual, isto é, tomar boas ações já conhecidas.

Nos métodos de Monte Carlo e alguns outros métdos, esse dilema é tipicamente tratado usando uma política $\epsilon - greedy$ que permite ao agente tentar novas ações com alguma probabilidade ϵ, mesmo que não sejam as melhores ações esperadas de acordo com as estimativas de valor atuais. Isso permite que o agente aprenda sobre as consequências de diferentes ações e melhore suas estimativas do valor de cada estado (SUTTON; BARTO, 2018).

No geral, esse dilema é uma consideração importante porque afeta a capacidade do agente aprender sobre o ambiente e encontrar a política ótima. O equilíbrio entre a exploração e o aproveitamento pode ajudar o agente a aprender de forma mais eficiente e eficaz.

2.4 Aprendizado por Diferença Temporal

Aprendizado por diferença temporal é um método que envolve aprender de uma sequência incompleta de experiência. O agente aprende o valor de um estado considerando a recompensa que recebe após tomar a ação e o valor do estado subsequente. Isso contrasta com os métodos de Monte Carlo, em que aprende-se o valor de um estado fazendo a média das recompensas recebidas dos episódios completos (SUTTON; BARTO, 2018).

Dessa forma o aprendizado acontece para cada passo de um episódio em vez de esperar até o fim do episódio para receber a recompensa, o que torna o aprendizado por diferença temporal mais eficiente do que métodos de Monte Carlo ao permitir que o agente aprenda da

sua experiência mais rapidamente. Porém, isso também pode ser uma desvantagem, pois o agente aprende com base em informações incompleta em relação a recompensas e estados, em vez de episódios completos de experiência, além disso, esses métodos também podem ter maior variância (SUTTON; BARTO, 2018).

De acordo com Sutton e Barto (2018) a regra de atualização de um método de aprendizado por diferença temporal pode ser dado por:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)], \tag{2.12}$$

onde o termo α representa a taxa de aprendizado do sistema e controla o *trade-off* entre nova informação ser aprendida e informação velha ser mantida. O termo $T_{t+1} + \gamma V(S_{t+1})$ é chamado de *TD Target* ou alvo do treinamento, e V aproxima a função de valor de estado v.

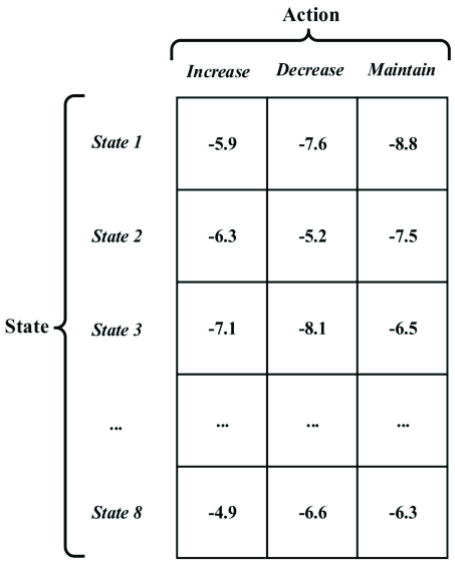
2.4.1 Q-Learning

O *Q-learning* é um método de aprendizado que tenta aprender uma função de valor de ação Q próxima de uma função de valor ótima q_* (SUTTON; BARTO, 2018).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{2.13}$$

A maneira mais comum de representar essa função Q é com tabelas, sendo um eixo para os estados e outro para as ações. Para problemas com muitos estados e ações, métodos como *Q-learning* são inviáveis computacionalmente pois utilizam uma representação matricial, o que torna inviável armazenar a tabela na memória (SUTTON; BARTO, 2018).

Figura 7 – Exemplo de uma tabela-Q com três ações (colunas) e 8 estados (linhas)



Fonte: Qu et al. (2020)

2.5 Deep Q-Learning

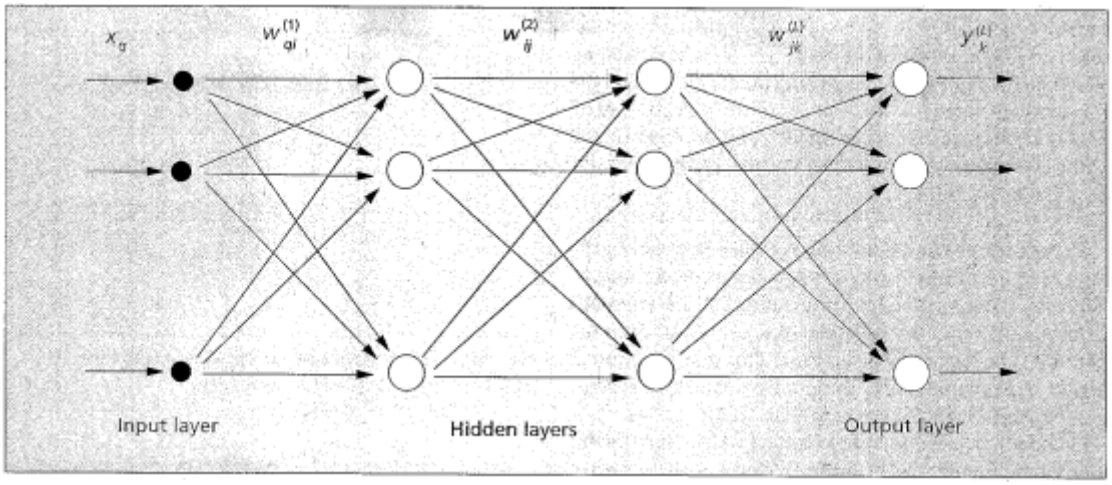
Os algoritmos apresentados anteriormente dependem de uma representação matricial ou vetorial para aprender. Para problemas com muitos estados e ações possíveis, isso torna-se inviável computacionalmente. Uma alternativa a computar os valores Q em uma tabela é utilizar aproximação de função com redes neurais artificiais, e essa é a proposta do *Deep Q-Learning* (DQN).

Na DQN, os valores Q são estimados usando uma rede neural profunda, que é treinada para prever os valores Q para um determinado estado e ação. A rede é treinada usando uma variante da equação de Bellman, que define os valores Q em termos do retorno futuro esperado da ação.

2.5.1 Redes Neurais Profundas

Redes Neurais Artificiais (RNA) são técnicas computacionais inspiradas na estrutura neural de organismos inteligentes que adquirem conhecimento através da experiência. Uma RNA é baseada em uma coleção de nós conectados chamados de neurônios artificias que, inspirados nos neurônios biológicos, podem transmitir sinais para os outros neurônios, assim como receber e processar sinais de outros neurônios (Figura 8).

Figura 8 – Exemplo de rede neural artificial



Fonte: Jain, Mao e Mohiuddin (1996)

Esses neurônios são normalmente organizados em camadas, de forma que os sinais trafegam da primeira camada (de entrada) até a última camada (de saída). O sinal de uma conexão é um número real, e a saída de cada neurônio é calculada por alguma função não linear da soma de suas entradas. Neurônios e conexões normalmente têm um peso que pode aumentar ou diminuir a força do sinal em uma conexão, e esse peso é ajustado conforme o aprendizado progride (JAIN; MAO; MOHIUDDIN, 1996).

Essas redes são chamadas de "profundas" pois possuem um grande número de camadas, tipicamente mais de 10. Essas camadas são usadas para extrair e transformar características dos dados de entrada, e a cada final da rede é usada para fazer previsões ou decisões baseadas nas características transformadas.

Segundo Goodfellow, Bengio e Courville (2016), o treinamento de um neurônio artificial envolve o ajuste dos pesos das conexões entre as entradas e a saída do neurônio, para que ele possa prever com precisão a saída de uma determinada entrada. Uma **função de ativação** é um componente chave de um neurônio artificial e é usada para introduzir não linearidade na saída do neurônio. Sem uma função de ativação, um neurônio artificial seria um modelo linear, que seria limitado em sua capacidade de modelar padrões complexos e relacionamentos em dados.

Uma função de perda é uma medida de quão bem uma rede neural artificial é capaz de prever a saída esperada para uma determinada entrada. Ela é diferença entre a saída gerada pela rede e a saída esperada, é um valor escalar que representa a "perda" ou erro entre os dois. A função de perda é usada para otimizar os pesos das conexões entre os neurônios da rede, para que a rede possa fazer previsões precisas sobre novos dados (GOODFELLOW; BENGIO; COURVILLE, 2016).

Uma definição comum de função de perda é a de quadrados mínimos:

$$L = (y_{previsto} - y_{esperado})^2 \tag{2.14}$$

A partir desse cálculo, pode-se usar um método para propagar esse erro pela rede, ajustando os pesos para melhorar a rede. Esse mecanismo é o **Backpropagation**, um algoritmo para treinamento de redes neurais artificiais, que é usado para calcular o gradiente da função de perda em relação aos pesos das conexões entre os neurônios da rede. O gradiente é usado para atualizar os pesos na direção que reduz a perda, usando um algoritmo de otimização como o gradiente descendente estocástico (SGD) (WYTHOFF, 1993).

2.6 *Soft Actor Critic*

Soft Actor-critic (SAC) é um algoritmo de aprendizado de reforço *off-policy* profundo que combina ideias de métodos *actor-critic* para treinar um agente imerso em um ambiente e que realize ações contínuas.

A principal vantagem do SAC é sua aplicabilidade em ambientes com espaços de ação contínuos, isto é, em que o espaço de ação é representado por um conjunto de valores reais em vez de discretos. Por exemplo, a ação que o robô deve realizar no futebol de robô é o objetivo $(x, y, \hat{a}ngulo, velocidade)$, onde as variáveis x, y e $\hat{a}ngulo$ são reais. Em várias aplicações de aprendizado por reforço com o SAC na realidade, utiliza-se variáveis reais para modelar as

ações dos agentes, como locomoção quadrúpede de robôs e manipulação de mão robóticas para agarrar objetos, dentre outras tarefas (HAARNOJA et al., 2018b).

De acordo com Christodoulou (2019), o algoritmo pode ser implementado para lidar com espaços de ação discretos ao modificar a regra de atualização da política, mas será considerado e apresentado neste trabalho apenas o caso contínuo pela sua aplicação no futebol de robôs.

No SAC, o agente possui duas redes neurais: uma rede "agente" que é usada para selecionar ações e uma rede "crítico" que é usada para estimar a função valor de ação Q. A rede do agente é treinada para maximizar o retorno esperado das ações que ela seleciona, enquanto a rede do crítico é treinada para estimar os valores-Q das ações tomadas pelo agente (HAARNOJA et al., 2018a).

2.6.1 Aprendizado Por Reforço Por Entropia Máxima

Segundo Goodfellow, Bengio e Courville (2016, p.73-74), entropia é uma quantidade que diz o quão aleatória uma variável aleatória é. A intuição por trás da entropia na teoria da informação é de que aprender que um evento improvável aconteceu é mais informativo do que aprender que um evento provável aconteceu, então os eventos raros tem mais informação do que os eventos comuns. Por exemplo, no lance de uma moeda viciada, em que a probabilidade de cair cara é maior do que coroa, a entropia é mais baixa pois há uma maior certeza de cair cara. Já no caso de uma moeda comum, a entropia é alta pois cada resultado é igualmente provável.

A entropia H de uma variável aleatória x pode ser computada a partir de sua função de densidade P através de:

$$H(P) = \mathop{\mathbb{E}}_{x\sim P}[-\log P(x)] \tag{2.15}$$

Segundo Haarnoja et al. (2018a), uma vantagem de utilizar a entropia no aprendizado por reforço é encorajar a exploração para que o agente não fique preso em um ótimo local, ao mesmo tempo que desiste de ações pouco promissoras, assim sendo uma forma de equilibrar o dilema entre exploração e aproveitamento.

Para formalizar, no aprendizado por reforço por entropia máxima, o agente recebe uma recompensa bônus em cada instante de tempo proporcional à entropia naquele instante de tempo, e desta forma o objetivo do aprendizado pode ser definido como (HAARNOJA et al., 2018a):

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big)], \tag{2.16}$$

onde o parâmetro de temperatura $\alpha > 0$ determina a importância relativa do termo da entropia em relação à recompensa, e portanto controla a estocacidade da política ótima.

Agora a função V pode ser calculada com esses bônus em cada instante de tempo (HAARNOJA et al., 2018a):

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^{t}\Bigg(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Bigg)\Bigg| s_0 = s] \tag{2.17}$$

Igualmente, a função Q pode ser calculada usando esse bônus em cada instante de tempo exceto o primeiro:

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^{t}R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^{t}H\left(\pi(\cdot|s_t)\right)\Bigg| s_0 = s, a_0 = a] \tag{2.18}$$

2.6.2 O Algoritmo

SAC é um algoritmo *off-policy* pois utiliza um *replay buffer* para amostrar os próximos estados, enquanto as próximas ações vem da política atual. O *replay buffer* é uma memória que armazena o estado e recompensa recebidas para cada estado anterior e ação tomada pelo agente, enquanto ele interage com o ambiente. A vantagem de utilizar o *replay buffer* é que a média da distribuição de comportamento é calculada sobre muitos de seus estados anteriores, suavizando o aprendizado e evitando oscilações ou divergência nos parâmetros (MNIH et al., 2013).

Segundo Haarnoja et al. (2018a), SAC aprende concorrentemente uma política π_θ e duas funções Q, Q_{ϕ_1}, Q_{ϕ_2}. A função de perda das redes-Q no SAC é:

$$L(\phi_i, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right], \tag{2.19}$$

onde $\mathcal{D}$ é o *replay buffer* e y é o alvo do aprendizado, dado por:

$$y(r,s',d) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s',\tilde{a}') - \alpha\log\pi_\theta(\tilde{a}'|s')\right), \qquad \tilde{a}'\sim\pi_\theta(\cdot|s'), \tag{2.20}$$

onde d é indica se o estado é terminal (1) ou não (0) e $\tilde{a}'$ é amostrado da política enquanto r e s' do *replay buffer*.

Com as definições 2.17 e 2.18, $V^{\pi}(s)$ e $Q^{\pi}(s,a)$ estão conectados por (HAARNOJA et al., 2018a):

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a\sim\pi}[Q^{\pi}(s,a)] + \alpha H\left(\pi(\cdot|s)\right) \tag{2.21}$$

O algoritmo deve para cada estado agir para maximizar essa função $V^{\pi}(s)$.

3 Metodologia

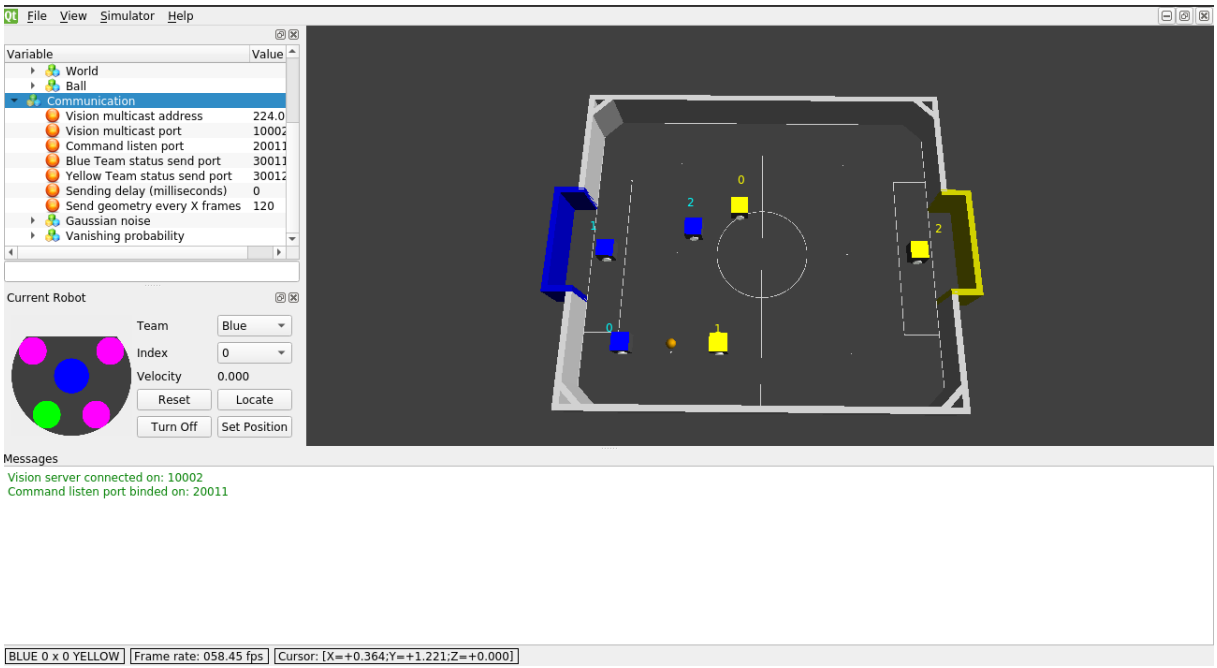
Neste capítulo é apresentado as ferramentas utilizadas para a realização do projeto e alguns detalhes de implementação pertinentes a elas, assim como a metodologia relacionada ao treinamento da estratégia.

3.1 Simulador e Juiz

Simuladores são com frequência usados em aprendizado por reforço pois fornecem um ambiente controlado e seguro para o agente treinar e aprender suas políticas, sem o risco de danificar sistemas do mundo real. Simuladores também permitem a pesquisadores controlar e manipular o ambiente de maneiras que não é possível no mundo real (ZHAO; QUERALTA; WESTERLUND, 2020).

No caso do futebol de robôs, há dificuldade de treinamento pois os robôs reais possuem bateria e outros componentes mecânicos, então não é possível fazer longas sessões de treinamento sem recarregá-los ou consertá-los e há o perigo de serem danificados durante os testes. Por outro lado, ao usar um simulador, é possível treiná-lo ainda mais rápido sem as dificuldades impostas pelo mundo real.

Figura 9 – Simulador FiraSim



Fonte: O Autor

Por causa disso, o simulador oficial da competição, o FiraSim foi escolhido como ambiente de treinamento da estratégia. A interface do simulador pode ser apreciada na Figura 9.

Outro programa necessário para o treinamento da estratégia é o Juiz (Figura 10). Ele monitora o jogo por situações de falta que são comunicadas ao cliente. Através dele é possível saber se um gol aconteceu para definir as recompensas correspondentes aos jogadores, identificar o início e fim de episódios, acompanhar o tempo de jogo e placar, além de criar situações de falta para treinar a estratégia.

Figura 10 – Tela do Juiz

Fonte: O Autor

A comunicação com o simulador e o juiz é feita usando a biblioteca Google Protobuf.

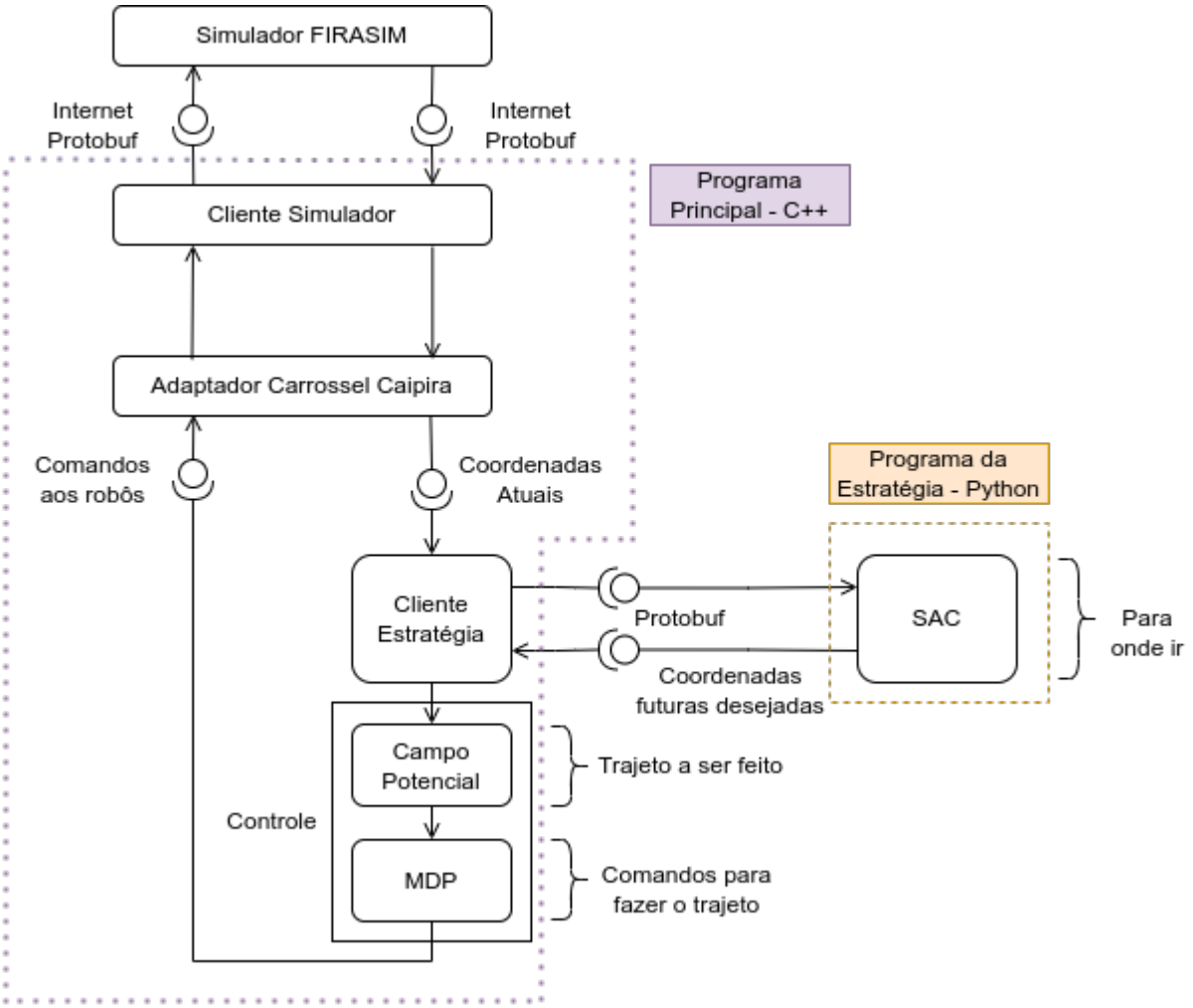
3.2 Comunicação Usando Docker e Protobuf

O Protobuf é o formato de serialização de dados do Google que é independente de plataforma e linguagem. Foi feito para ser leve, eficiente e extensível.

Conforme a Figura 11, o Protobuf é o modo utilizado pelo simulador e o juiz para comunicação. Mensagens são descritas de maneira estruturada em arquivos com extensão *.proto* que são compilados para gerar as classes na linguagem escolhida. Essas classes podem então ser usadas para serializar e desserializar mensagens.

Utilizando o Protobuf, o programa principal do time consegue receber informações do simulador e juiz, como a posição dos robôs e da bola em campo e situações de falta. Esses dados são então passados para o programa da estratégia que foi feito em Python, e o programa irá receber o vetor objetivo do robô, que então será passado para o controle que irá decidir como movimentar o robô em campo e enviar os comandos de volta para o FiraSim.

Figura 11 – Sistema simulado do time Carrossel Caipira



Fonte: O Autor

A interface do simulador com o programa principal tem como propósito fornecer informações de jogo ao cliente e receber comandos para movimentar os robôs em campo. O simulador envia a posição (x, y), orientação e velocidade dos robôs e da bola em campo, assim como informações de falta do Juiz, como tipo, time e quadrante, e o cliente envia em resposta as velocidades de rotação de cada roda dos robôs.

A interface do programa principal para a estratégia tem como propósito repassar as informações recebidas do simulador, assim como outros dados adicionais úteis para compor o estado único dos jogadores, como por exemplo a cor do time, para o programa da estratégia.

A estratégia então processa essas informações em um estado e o utiliza para treinar o algoritmo de aprendizado por reforço, recebendo uma ação para este estado. Esta ação é o vetor objetivo $(x, y, \hat{a}ngulo, velocidade)$ que será passado ao programa principal via a mesma interface.

3.3 Docker

Com a pandemia de COVID-19 durante os anos de 2020 e 2021, a competição de futebol de robôs aconteceu de maneira remota, com as equipes se conectando a uma máquina remota para executar seus clientes do FiraSim e poderem jogar.

Para permitir que as equipes executassem seus clientes na máquina remota de forma segura e padronizada, considerando que as equipes possuem ambientes de desenvolvimento diferentes, tornou-se um requisito a utilização do Docker para a conexão com FiraSim.

O Docker é uma plataforma de contêineres que permite aos desenvolvedores empacotar e implantar aplicativos de maneira leve e portátil. Os contêineres são ambientes isolados que contêm tudo o que um aplicativo precisa para executar, incluindo código, bibliotecas, dependências e tempo de execução. Isso facilita a execução consistente do aplicativo em diferentes ambientes, como desenvolvimento, teste, preparação e produção.

Com base nisso foi feito a transferência do ambiente da equipe para um contêiner do Docker, com cinco serviços independentes: firasim, juiz, simulador da equipe, cliente amarelo do time e cliente azul do time.

Com isso surge um problema adicional em que todos os serviços estão rodando em um contêiner, mas a estratégia desenvolvida nesse trabalho foi feita em Python na máquina *host*, então é necessário criar uma forma de comunicação entre o contêiner e o *host*. Para isto, basta compartilhar um diretório do sistema operacional da máquina *host* com o contêiner.

3.4 Bibliotecas e Frameworks

O código da estratégia foi desenvolvido em Python 3, considerando que muitas das principais bibliotecas de aprendizado por reforço focam nessa linguagem. A biblioteca Stable Baselines 3 foi usada por possuir implementações otimizadas dos principais algoritmos de aprendizado por reforço, incluindo o *Soft Actor Critic* que foi usado nesse projeto. Ela também fornece utilização padronizada através do Gym.

O Gym é um conjunto de ambientes de testes para aprendizado por reforço. Ele define um modelo para ambientes de aprendizado por reforço, em uma classe com 5 métodos:

- *constructor*: deve definir o espaço de ação e o espaço de estados.

- *step*: dado uma ação, deve retornar o estado, recompensa e informação se o estado é terminal.
- *reset*: reinicia o ambiente. É chamado quando o estado for terminal.
- *render*: método para renderizar o ambiente.
- *close*: fecha o ambiente.

O sistema do time foi encapsulado em um ambiente do Gym definindo esses métodos com as informações recebidas via Protobuf. Por exemplo, para o goleiro é possível verificar se a falta é do tipo *KICKOFF*[1] para o time, e se for verdade isso significa que o oponente marcou um gol e a recompensa será definida como negativa.

Outras bibliotecas auxiliares incluem o Numpy para fazer operações com *arrays* e o Matplotlib para gerar os gráficos de recompensa dos jogadores.

3.5 Recompensas

O goleiro recebe $+1$ de recompensa em cada instante de tempo, exceto caso receba um gol. Neste caso, ele recebe uma recompensa de -1000. Desta forma, ele é desincentivado a sair do gol e avançar para o lado do adversário, pois deixará o gol vulnerável e potencialmente receberá um gol com recompensa de -1000. Portanto, para maximizar sua recompensa que incrementa a $+1$ cada instante de tempo, ele deverá defender o gol do time o máximo de tempo possível.

Para o atacante é o contrário, ele recebe uma recompensa de -1 a cada instante de tempo e $+1000$ caso faça um gol. Desta forma, ele é incentivado a marcar gols o mais rápido possível para maximizar sua recompensa que diminui a cada instante de tempo.

Para o volante considerou-se um papel intermediário, então ele recebe -1 de recompensa se a bola estiver na metade do campo do time, $+1$ se ela estiver na metade do campo adversário, -1000 se receber um gol e $+1000$ se fizer um gol. Desta forma, o volante é incentivado a levar a bola ao campo adversário, auxiliar o atacante a marcar gols e auxiliar o goleiro a defender gols.

3.6 Entrada e Saída

A entrada do sistema é o vetor de estado, composto pelas variáveis contínuas x e y para todos os robôs e a bola em campo, a variável contínua $\hat{a}ngulo$ para os três robôs controlados, a variável discreta de cor do time, que pode ser azul ou amarela, e as variáveis discretas relacionadas a faltas ou situações de jogo: tipo, time e quadrante.

[1] Situação indicada pelo juiz para início ou reinício de jogo, por exemplo em caso de gol.

A saída do sistema, ou ação de cada robô, é o vetor objetivo $(x, y, \hat{a}ngulo, velocidade)$ de cada um dos robôs controlados que será passado para o controle que deverá fazer os robôs movimentarem-se para cumprir esse objetivo.

3.7 Implementação

Conforme apresentado na seção 3.2 e seção 3.4, a estratégia foi desenvolvida em Python utilizando a biblioteca Stable Baselines 3, com comunicação via Protobuf com o juiz e o simulador para obter informações sobre o estado do jogo e falta e enviar comandos de reposicionamento e movimentação aos jogadores.

O algoritmo *Soft Actor Critic* foi escolhido pois é um dos algoritmos estado da arte em aprendizado por reforço para ações contínuas (HAARNOJA et al., 2018b). Este é o caso do futebol de robôs, em que as variáveis x, y e $\hat{a}ngulo$ que compõem o objetivo dos jogadores, são contínuas, o que inviabiliza a utilização de métodos discretos.

Junto a ele foi utilizado a política *MlpPolicy* do Stable Baselines 3, que implementa *actor-critic* usando um perceptron multicamada com 2 camadas de 64 neurônios.

Para treinar cada jogador basta chamar o método SAC do Stable Baselines 3 no modelo com o ambiente adaptado do futebol de robôs e passar o número de instantes de tempo[2] desejados para o treinamento. Cada instante de tempo é uma chamada dos métodos *step* ou *reset*, que varia por segundo dependendo do poder computacional disponível.

O goleiro foi então treinado por 100.000 instantes de tempo junto ao volante e atacantes da estratégia dos autômatos para fins de testes, pois ao treinar um único jogador, o poder computacional disponível a ele é maior do que treinar os três ao mesmo tempo, e é possível realizar mais chamadas da função *step* do ambiente por segundo, logo ele aprende mais rapidamente.

O goleiro jogou contra o time com a estratégia anterior dos autômatos, e várias situações específicas foram testadas no juiz e simulador durante o treinamento para determinar se o goleiro continuaria aprendendo, como defesa de pênalti, cobrança de tiro de meta, defesa de ataque duplo, defesa sem volante, etc.

As recompensas do treinamento foram salvas a cada instante de tempo e um gráfico foi gerado para determinar se o goleiro realmente estava aprendendo a maximizar a recompensa. Por fim, como constatado no gráfico da seção 4.1, o ambiente estava funcionando e o goleiro aprendendo, pois as suas recompensas aumentaram.

Em seguida, o goleiro, volante e atacante foram treinados juntos por cerca de 40.000 instantes de tempo. Os dados de treinamento foram salvos e são apresentados no Capítulo 4.

[2] O instante de tempo, ou época, representa o período entre cada interação com do sistema de controle do futebol e o aprendizado com o ambiente simulado do futebol de robôs.

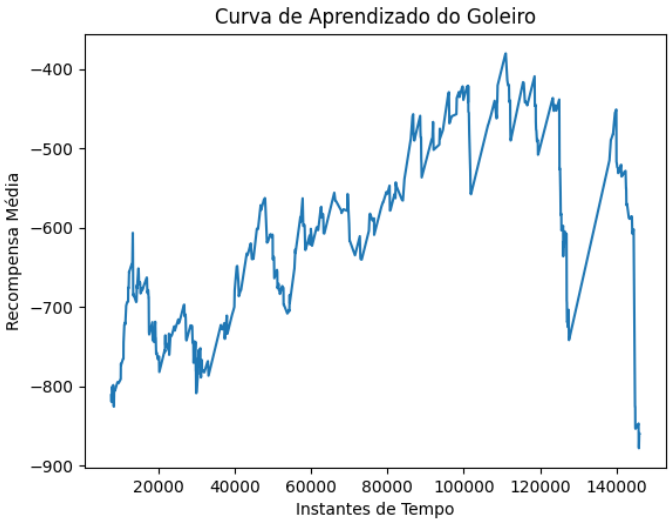
4 Resultados

A Figura 12, Figura 13 e Figura 14 mostram a recompensa média a cada 50 instantes de tempo obtidos por cada jogador durante o treinamento. A média móvel de 50 instantes de tempo é para suavizar as curvas e visualizar a tendência de aprendizado, desconsiderando as recompensas individuais que podem variar muito de um instante ao outro.

Na seção 4.1, seção 4.2 e seção 4.3 serão discutidos os resultados obtidos por cada jogador, e na seção seção 4.4 um comportamento geral da estratégia observado durante o treinamento.

4.1 Goleiro

Figura 12 – Curva de Aprendizado do Goleiro



Fonte: O Autor

O gráfico do goleiro na Figura 12 foi o que mais apresentou crescimento, passando de -800 recompensa no início de treinamento para -400 recompensa no pico por volta de 100.000 instantes de tempo.

Para o goleiro a recompensa é $+1$ por instante de tempo e -1000 se receber um gol. Além disso, sempre que um gol é feito o episódio acaba, então a recompensa indica o quanto de tempo, em média, o goleiro conseguiu defender o gol por episódio, que passou de 200 no início do aprendizado para 600 instantes de tempo.

A recompensa média até os 100.000 instantes de tempo possui vários picos e vales característicos do aprendizado por reforço, pois é através da exploração que o agente pode encontrar

ações melhores a serem tomadas, além do ambiente ser inerentemente não-determinístico, o que gera aleatoriedade. Mas ainda sim, a tendência é que o goleiro aprendeu a defender o gol por cada vez mais tempo.

Por volta de 100.000 instantes de tempo a recompensa média reduziu abruptamente. Isto aconteceu pois o goleiro deixou de treinar sozinho com o volante e atacante da estratégia dos autômatos para treinar com o volante e atacantes que usam o SAC. Essa mudança de comportamento do volante e atacante que serve como entrada da rede neural do goleiro fez com que o goleiro tivesse que adaptar a sua política aprendida ao novo volante e atacante. Além disso, o novo volante e atacante estavam iniciando seus treinamentos, então não conseguiam jogar com a mesma qualidade do que os anteriores.

Por esses motivos, a recompensa média do goleiro cai por volta de 120.000 instantes para -700 recompensa, antes de retornar a cerca de -450 e logo cair novamente para cerca de -850, o menor valor experimentado. O pico de -450 recompensa é o goleiro reaprendendo seu comportamento com o novo volante e atacante: mesmo que eles joguem pior que os anteriores, o goleiro pode adaptar o que aprendeu e continuar o treinamento sem precisar treinar novamente por 80.000 instantes de tempo para chegar nesse nível. Da mesma forma, de 100.000 até 120.000 instantes de tempo ele conseguiu manter com pouca variação a recompensa média durante o início do treinamento com o novo volante e atacante.

A queda em seguida para -850 recompensa é pelo mesmo motivo. Se o aprendizado continuasse, possivelmente apareceriam mais grandes picos e vales, pois o goleiro precisa reaprender a jogar com o novo volante e atacante ao mesmo tempo que mantém o que aprendeu com os anteriores.

4.2 Atacante

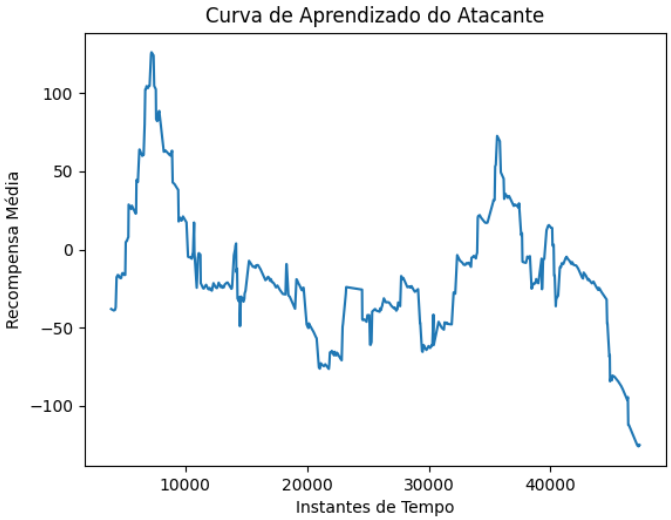
O treinamento do atacante se inicia no instante de tempo 100.000 do treinamento do goleiro. Conforme a Figura 13, o atacante inicia seu treinamento e logo chega a uma recompensa de 100 após 7.000 instantes de tempo, mas em seguida cai e mantém-se entre 0 e -50, antes de subir novamente para 70 após 35.000 instantes de tempo e cair novamente.

O atacante recebe uma recompensa de -1 por instante de tempo e $+1000$ se ele realizar um gol, então a recompensa média diz o quanto de tempo, em média, foi necessário para o atacante realizar um gol. Se a recompensa for negativa, ele possivelmente não conseguiu realizar um gol ou demorou mais do que 1.000 instantes.

O pico inicial é possivelmente por causa da exploração inicial do algoritmo, assim como é possível observar nos gráficos dos outros jogadores. Após este pico, a recompensa se estabiliza entre 0 e -50, o que significa que o atacante ainda está aprendendo a fazer gols. O pico por volta de 35.000 instantes de tempo indica que o atacante conseguiu realizar um gol, seguido

de um pico menor mas positivo, mostrando que o atacante está aprendendo lentamente a realizar gols. A queda final é possivelmente por causa da influência do goleiro, como descrito na seção 4.1, pois se o goleiro não conseguir defender os gols, o episódio acabará para o atacante com uma recompensa negativa. Esta influência dos outros jogadores na recompensa obtida por ele torna difícil compreender seu comportamento a partir do gráfico.

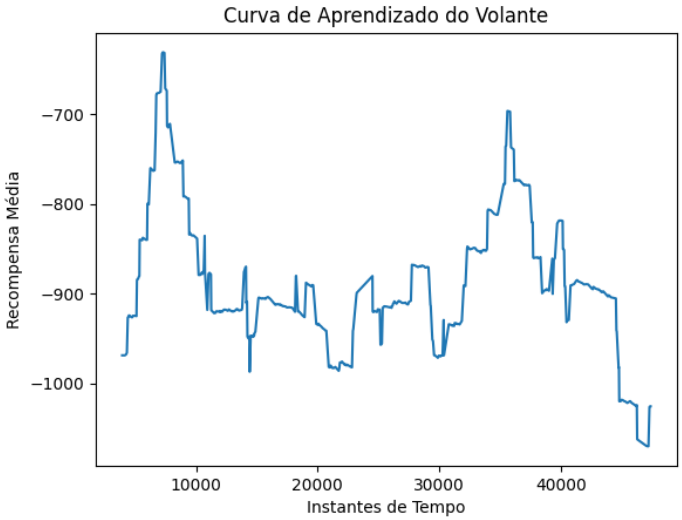
Figura 13 – Curva de Aprendizado do Atacante



Fonte: O Autor

4.3 Volante

Figura 14 – Curva de Aprendizado do Volante



Fonte: O Autor

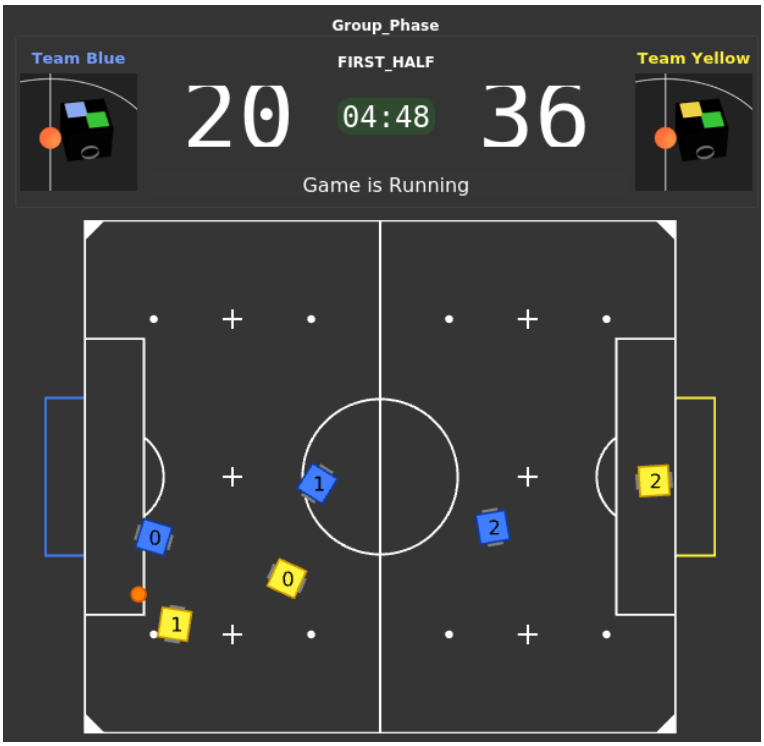
O treinamento do volante se inicia no instante de tempo 100.000 do treinamento do goleiro. O volante recebe -1000 se receber um gol, +1000 se realizar um gol, -1 se a bola estivar na metade do campo do time e +1 se estiver na metade do campo adversário. Desta forma, sua recompensa média é influenciada tanto pelo atacante que deve aprender a realizar gols, quanto pelo goleiro que deve aprender a defender gols. Como é possível observar na Figura 14, ela possui formato semelhante a Figura 13. Desta forma, é difícil compreender o comportamento exclusivo do volante a partir do gráfico, pois sua recompensa depende dos outros jogadores.

O pico inicial de -700 recompensa é possivelmente devido a exploração inicial do algoritmo. Em seguida ele se estabiliza por volta de -900, até subir para -700 recompensa após cerca de 35.000 instantes de tempo, o que é devido ao atacante realizar gols conforme Figura 13. A queda final é influência do goleiro, pois a recompensa média abaixo de -1000 só é possível se receber um gol e a bola ficar mais tempo no campo do time do que no campo adversário.

Portanto, a tentativa de treinar os três jogadores ao mesmo tempo não gerou bons resultados, visto que o aprendizado de um influencia no aprendizado do outro.

4.4 Estratégia

Figura 15 – Comportamento dos Jogadores



Fonte: O Autor

A Figura 15 mostra uma captura de tela durante o treinamento, do time azul da estratégia treinada jogando contra o time amarelo da estratégia dos autômatos. É possível observar um comportamento primitivo dos jogadores ainda com pouco treinamento: o goleiro posicionado perto da área do gol para poder defender melhor caso a bola chegue no campo do time, o volante em uma posição intermediária para ajudar a levar a bola do campo do time para o campo adversário, e o atacante no campo adversário para maximizar suas chances de gols quando a bola chegar a ele.

5 Conclusões e Trabalhos Futuros

Neste trabalho, o sistema da equipe Carrosel Caipira foi adaptado como um problema de aprendizado por reforço definido por uma MDP, capaz de devolver a recompensa para ações realizadas pelos jogadores.

O treinamento foi feito com o algoritmo SAC, porém há a possibilidade de que outros algoritmos apresentem resultados melhores ou mais rápidos. Como o ambiente foi encapsulado como um ambiente do Gym, qualquer biblioteca de aprendizado por reforço capaz de usar um ambiente do Gym é capaz de treinar a estratégia, então basta trocar o algoritmo, recomeçar o treinamento e analisar os resultados.

Durante o treinamento do goleiro, ele foi treinado sozinho com o volante e atacante da estratégia dos autômatos por 100.000 instantes de tempo. Isto proporcionou uma aprendizagem mais rápida e com maior recompensa média do que treinar os três robôs juntos do zero. Desta forma, ele pode aprender mais independente dos outros jogadores do time. É portanto uma possibilidade mesclar a estratégia desenvolvida com a estratégia anterior, ao treinar os robôs individualmente para posições específicas em conjunto com a estratégia dos autômatos, ou até mesmo treiná-los para situações específicas, como cobrança de pênaltis.

Os resultados obtidos pelo goleiro mostram que ele estava aprendendo a defender por cada vez mais tempo o gol, enquanto os obtidos pelo volante e atacante foram inconclusivos exceto por um ligeiro aumento na recomenda média após 35.000 instantes de tempo. Com mais tempo de treinamento, possivelmente 100.000 instantes de tempo como o goleiro, seria possível observar uma maior recompensa média. Por fim, foi notado a tendência de posicionamento dos jogadores em certas regiões do campo para maximizar suas recompensas.

Outros pontos para melhorar incluem: a estrutura do sistema, que está escrito em C++ mas a estratégia em Python, se comunicando com o Protobuf e dentro de um contêiner do Docker. Todas essas camadas adicionam uma ineficiência no treinamento que pode ser removida; modificar o sistema para suportar várias instâncias de treino paralelas através de *multithreading*; Melhorar a função de recompensa dos jogadores para agilizar o treinamento.

Por fim, a estrutura desenvolvida pode facilitar trabalhos futuros de aprendizado de máquina na equipe do Carrossel Caipira, e somado a mais tempo de treinamento do modelo treinado, podem gerar uma boa estratégia de futebol de robôs.

Referências

BACKER, K. D.; DESTEFANO, T.; MENON, C.; SUH, J. R. Industrial robotics and the global organisation of production. OECD, 2018.

CBROBOTICA. *Regras IEEE Very Small Size Soccer (VSSS) - Série B*. 2022. Disponível em: https://www.cbrobotica.org/wp-content/uploads/2022/05/vssRules.pdf. Acesso em: 27 maio 2022.

CHRISTODOULOU, P. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to algorithms*. [S.l.]: MIT press, 2022.

FU, J.; TOPCU, U. Probably approximately correct mdp learning and control with temporal logic constraints. *arXiv preprint arXiv:1404.7073*, 2014.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.

HAARNOJA, T.; ZHOU, A.; ABBEEL, P.; LEVINE, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: PMLR. *International conference on machine learning*. [S.l.], 2018. p. 1861–1870.

HAARNOJA, T.; ZHOU, A.; HARTIKAINEN, K.; TUCKER, G.; HA, S.; TAN, J.; KUMAR, V.; ZHU, H.; GUPTA, A.; ABBEEL, P. et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

International Federation of Robotics. World robotics 2021. 2021. Disponível em: https://ifr.org/ifr-press-releases/news/robot-sales-rise-again. Acesso em: 29 janeiro 2023.

JAIN, A. K.; MAO, J.; MOHIUDDIN, K. M. Artificial neural networks: A tutorial. *Computer*, IEEE, v. 29, n. 3, p. 31–44, 1996.

JURKAT, A.; KLUMP, R.; SCHNEIDER, F. Tracking the rise of robots: the ifr database. *Jahrbücher für Nationalökonomie und Statistik*, De Gruyter Oldenbourg, v. 242, n. 5-6, p. 669–689, 2022.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, v. 4, p. 237–285, 1996.

KALEJAIYE, L. B. T. Treinamento de agentes jogadores de futebol usando aprendizado por reforço. 2019.

KEMPKA, M.; WYDMUCH, M.; RUNC, G.; TOCZEK, J.; JAŚKOWSKI, W. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In: IEEE. *2016 IEEE conference on computational intelligence and games (CIG)*. [S.l.], 2016. p. 1–8.

KHAN, A.; RINNER, B.; CAVALLARO, A. Cooperative robots to observe moving targets. *IEEE transactions on cybernetics*, IEEE, v. 48, n. 1, p. 187–198, 2016.

KIM, J.-H.; VADAKKEPAT, P. Multi-agent systems: a survey from the robot-soccer perspective. *Intelligent Automation & Soft Computing*, Taylor & Francis, v. 6, n. 1, p. 3–17, 2000.

LUND, H. H.; PAGLIARINI, L. Robot soccer with lego mindstorms. In: SPRINGER. *Robot Soccer World Cup*. [S.l.], 1998. p. 141–151.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

QU, C.; ZHANG, Y.; ZHANG, X.; YANG, Y. Reinforcement learning-based data association for multiple target tracking in clutter. *Sensors*, MDPI, v. 20, n. 22, p. 6595, 2020.

ROSSETTI, R. C. B.; SANTOS, L. A. dos; CORREIA, J. V. M.; SHIMABUKURO, T. S.; OLIMPIO, J. P.; PEGORARO, R.; BATISTA, M. *O Time Carrossel Caipira de Futebol de Robôs*. 2020. Disponível em: https://github.com/VSSSLeague/vss/blob/master/tdp/TDP2020/carrossel.pdf. Acesso em: 27 maio 2022.

ROSSETTI, R. C. B.; SANTOS, L. A. dos; CORREIA, J. V. M.; PEGORARO, R.; BATISTA, M. *O Time Carrossel Caipira de Futebol de Robôs*. 2021.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. V. D.; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M. et al. Mastering the game of go with deep neural networks and tree search. *nature*, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016.

SILVER, D.; HUBERT, T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A.; LANCTOT, M.; SIFRE, L.; KUMARAN, D.; GRAEPEL, T. et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, American Association for the Advancement of Science, v. 362, n. 6419, p. 1140–1144, 2018.

SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.

WYTHOFF, B. J. Backpropagation neural networks: a tutorial. *Chemometrics and Intelligent Laboratory Systems*, Elsevier, v. 18, n. 2, p. 115–155, 1993.

ZHAO, W.; QUERALTA, J. P.; WESTERLUND, T. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: IEEE. *2020 IEEE symposium series on computational intelligence (SSCI)*. [S.l.], 2020. p. 737–744.