



UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

JOÃO PEDRO FIGUEIRÔA NASCIMENTO

INVESTIGAÇÃO SOBRE UTILIZAÇÃO DE
APRENDIZAGEM POR REFORÇO PARA MÓDULO DE
DEFESA APLICADO AO FUTVASF2D

JUAZEIRO - BA

2019

UNIVERSIDADE FEDERAL DO VALE DO SÃO FRANCISCO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

JOÃO PEDRO FIGUEIRÔA NASCIMENTO

INVESTIGAÇÃO SOBRE UTILIZAÇÃO DE
APRENDIZAGEM POR REFORÇO PARA MÓDULO DE
DEFESA APLICADO AO FUTVASF2D

Trabalho apresentado à Universidade Federal do Vale do São Francisco - Univasf, Campus Juazeiro, como requisito da obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Rosalvo Ferreira de Oliveira

JUAZEIRO - BA

2019

Planet Earth is blue and there's nothing I can do

David Bowie

Space Oddity

AGRADECIMENTOS

Agradeço primeiramente a Deus e a minha família, principalmente meus pais, João Alves e Regina Coeli, e a meus irmãos, Maria Alice e Luciano, pelo apoio e carinho durante todo o caminho da graduação.

À minha noiva, Nathanaele, por mesmo longe se fazer presente e me incentivar a continuar em frente sempre.

Agradeço ao “Clubinho Guaraná” composto por Ellen, Daniel, Talita, Carolina, Isaac, Esron e Mauricio, pelos momentos compartilhados de alegria, estresse e, as vezes, tristeza. Um grupo não só de leitura, mas de colaboração mútua e respeito. Um agradecimento especial a Ellen por estar comigo desde o começo da jornada.

Agradeço a meu orientador, Rosalvo, por me apresentar a oportunidade de dar início ao projeto FutVasf2D e trabalhando comigo por 2 anos seguidos até finalmente a conclusão.

Obrigado aos professores Ricardo Ramos e Jorge Cavalcanti que contribuíram com a melhoria deste trabalho.

Obrigado ainda a Brauliro e Ricardo, novamente, com conselhos profissionais e acadêmicos que contribuíram para formação e ofereceram momentos de lazer que se mostraram um oásis em meio a todo o estresse do curso, se tornando amigos, ousou dizer.

Obrigado à Deise por sempre ajudar com toda a burocracia da UNIVASF, até quando parecia que não tinha jeito.

Por fim agradeço a todos que estiveram comigo neste caminho, professores, colegas e amigos. Obrigado por me ajudar a me tornar quem eu sou hoje.

RESUMO

Este trabalho se debruça sobre a copa do mundo de futebol de robôs (*RoboCup Soccer*), que incentiva a produção de pesquisas na área de inteligência artificial e robótica, enquanto parte do projeto FutVasf2D que busca o desenvolvimento de um time oficial de futebol de robôs na liga de simulação 2D para UNIVASF, investigando o efeito de diferentes modelos de mundo aplicados à técnica de aprendizado por reforço *Q-Learning* sobre a performance do time na posição de defesa. Este trabalho busca encontrar modelos que facilitem o aprendizado dos agentes na tentativa de interceptar bolas lançadas pelo adversário e de capturar a bola em posse do adversário. Para alcançar o objetivo foram desenvolvidos modelos através de diferentes combinações de estados, ações, recompensas e métodos de implementação. Os modelos foram treinados e avaliados através da ferramenta HFO que cria situações de defesas aleatoriamente agilizando o processo de treinamento. Os resultados encontrados foram comparados entre si e entre o time base original (*WrightEagleBASE*), mas apresentaram desempenho abaixo do esperado, levando discussões de possíveis falhas no processo.

Palavras-chave: Futebol de Robôs, Aprendizado por reforço, *Q-Learning*, *RoboCup Simulation 2D*, Defesa.

ABSTRACT

This work dwell on the soccer world cup (RoboCup Soccer), which encourages researchers in the artificial intelligence and robotics fields, being part of the FutVasf2D project that looks for the development of an official robot soccer team in the 2d simulation league to the UNIVASF, investigating the effects of different world designs applied to the reinforcement technique Q-Learning about the performance of the team in defense position. This work tries to find designs that improve the learning of the agents into trying to intercept balls launched by the adversaries and capture the ball in the opponent possession. To reach this objective designs with differents combination of states, actions, rewards and implementation methods were developed. The designs were trained e evaluated through the HFO tools, which creates situations of defense randomly making the learning process faster. The found results were compared between themselves and the original base team (WrightEagleBASE), but showed performance below the expected, leading to discussions about the possible faults in the process.

Key-words: *Robot Soccer, Reinforcement Learning, Q-Learning, RoboCup Simulation 2D, Defense.*

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura geral de um agente.	11
Figura 2 – Representação do tabuleiro de jogo-da-velha.	16
Figura 3 – Aprendizado por reforço.	21
Figura 4 – Tabuleiro do problema exemplo.	25
Figura 5 – Representação dos estados e solução encontrada pelo <i>Q-Learning</i> . . .	25
Figura 6 – Etapas de KDD.	28
Figura 7 – Diagrama de classes simplificado para mecanismo de defesa.	34
Figura 8 – Algoritmo da classe <i>BehaviorDefensePlanner</i>	35
Figura 9 – Algoritmo da classe <i>BehaviorFormationPlanner</i>	36
Figura 10 – Algoritmo da classe <i>BehaviorBlockPlanner</i>	37
Figura 11 – Algoritmo da classe <i>BehaviorMarkPlanner</i>	38
Figura 12 – Algoritmo da classe <i>BehaviorInterceptPlanner</i>	39
Figura 13 – HFO	40
Figura 14 – Médias finais em comparação ao original	42
Figura 15 – Gráficos de Modelos 2 e 3	43

LISTA DE TABELAS

Tabela 1	–	<i>Q-Table</i>	23
Tabela 2	–	<i>Q-Table</i> resultante da execução do experimento.	26
Tabela 3	–	<i>Resultados</i>	41

LISTA DE CÓDIGOS

Código 1 – Algoritmo de <i>Q-Learning</i>	24
---	----

LISTA DE ABREVIATURAS E SIGLAS

CBR	Competição Brasileira de Robótica
CMAC	Computador Aritmético de Modelo Cerebelar
DCBD	Descoberta de Conhecimento em Base de Dados
HAQL	<i>Q-Learning</i> Acelerado Heuristicamente
HFO	Ofensa de Meio Campo
KDD	Descoberta de Conhecimento em Base de Dados, do inglês <i>Knowledge Discovery in Databases</i>
MDP	Processo de Decisão Markoviano
PIBIC	Programa Institucional de Bolsas de Iniciação Científica
QRL	Aprendizado por Reforço Qualitativo
QSR	Raciocínio Espacial Qualitativo
RPROP	<i>Backpropagation</i> Resiliente
SARSA	Estado-Ação-Recompensa-Estado-Ação
UNIVASF	Universidade Federal do Vale do São Francisco

SUMÁRIO

1	INTRODUÇÃO	11
1.1	MOTIVAÇÃO	12
1.2	DEFINIÇÃO DO PROBLEMA	13
1.3	OBJETIVOS	14
1.3.1	Objetivo Geral	14
1.3.2	Objetivos Específicos	14
1.4	ORGANIZAÇÃO DO TRABALHO	14
2	FUNDAMENTAÇÃO TEÓRICA	16
2.1	TRABALHOS RELACIONADOS	16
2.2	APRENDIZADO POR REFORÇO	20
2.2.1	Q-Learning	23
3	MATERIAIS E MÉTODOS	27
3.1	MODELAGEM DO MUNDO	27
3.1.1	Estados	28
3.1.2	Ações	32
3.1.3	Recompensas	32
3.2	IMPLEMENTAÇÃO EM <i>WRIGHTEAGLEBASE</i>	33
3.3	METODOLOGIA EXPERIMENTAL	37
4	RESULTADOS	41
5	CONCLUSÃO	44
5.1	TRABALHOS FUTUROS	45
	REFERÊNCIAS	46
	APÊNDICE A CÓDIGOS-FONTE	49

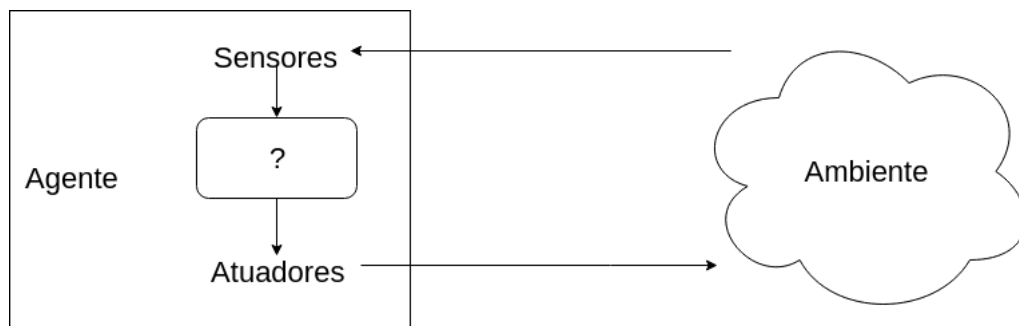
1 INTRODUÇÃO

Como descrito por Kitano et al. (1997), a *RoboCup*, a copa do mundo de futebol de robôs, procura fomentar as pesquisas acerca de robótica e inteligência artificial fornecendo um problema padrão capaz de avaliar teorias, algoritmos e arquiteturas de agentes. O projeto da *RoboCup* é dividido em várias ligas, desde robôs humanoides até a liga de simulação 2D. Proporcionando assim, um ambiente criativo e desafiador para que novas tecnologias sejam criadas e aplicadas por alunos de graduação.

Segundo Henn, Henrio e Nakashima (2017), a liga de simulação 2D representa um jogo de futebol, em um *framework* multiagente, no qual o ambiente é um campo de futebol em duas dimensões e os agentes são os jogadores, criando a vantagem de livrar os pesquisadores de toda a parte mecânica e eletrônica para que o foco se volte à análise de dados e construção da estratégia. Henn, Henrio e Nakashima (2017) ainda afirmam que a principal parte da liga é a modelagem de uma estratégia ou método efetivo o suficiente para obter performance superior aos adversários. Por fim, os autores relatam que uma das estratégias com melhores resultados é a utilização de agentes inteligentes.

De acordo com Russell e Norvig (2016), um agente inteligente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre seu ambiente por intermédio de atuadores. A Figura 1 ilustra a arquitetura clássica de um agente, onde é possível perceber que o agente interage com o ambiente por meio de sensores e atuadores.

Figura 1 – Arquitetura geral de um agente.



Fonte: (RUSSELL; NORVIG, 2016)

No domínio da liga de simulação 2D da *RoboCup*, o ambiente é composto pela representação 2D do campo, as traves, a bola e os jogadores. Esse ambiente é captado pelos agentes através dos seus sensores classificados como físicos, visuais ou acústicos, que, respectivamente, servem para perceber o estado do agente, referente ao vigor, velocidade e

posição; as posições e velocidades dos demais objetos no ambiente, incluindo jogadores e mensagens oriundas dos aliados. Os atuadores dos agentes correspondem aos mecanismos de movimento, rotação e chute (ROBOCUP, 2019). Um dos aspectos que torna o desenvolvimento de agentes inteligentes para jogos de futebol de robôs desafiador é a característica de seu ambiente:

- **Parcialmente observável:** O ambiente para um agente jogador de futebol de robôs é parcialmente observável, pois os seus sensores são imprecisos e também porque partes do estado estão simplesmente ausentes nos dados do sensor (SCHMILL; OATES; COHEN, 2000);
- **Estocástico:** De acordo com Hayes-Roth (1995), se o próximo estado do ambiente é completamente determinado pelo estado atual e pela ação executada pelo agente, dizemos que o ambiente é determinístico; caso contrário, ele é estocástico. Desta forma, o ambiente para um agente jogador de futebol de robôs é estocástico, pois o próximo estado não depende apenas da sua ação;
- **Episódico:** Em um ambiente de tarefa episódico, a experiência do agente é dividida em episódios atômicos (WOOLDRIDGE, 2001). Cada episódio consiste na percepção do agente, e depois na execução de uma única ação. É crucial que o episódio seguinte não dependa das ações executadas em episódios anteriores. Desta forma, o ambiente de tarefa para o agente jogador de futebol de robôs é episódico;
- **Dinâmico:** Ambientes estáticos são fáceis de manipular, porque o agente não precisa continuar a observar o mundo enquanto está decidindo sobre a realização de uma ação, nem precisa se preocupar com a passagem do tempo. No entanto, o ambiente de tarefa para o agente jogador de futebol de robôs é dinâmico, pois o ambiente pode ser alterado enquanto o agente está deliberando (FRANKLIN; GRAESSER, 1996);
- **Multiagente:** Um ambiente ainda pode ser classificado como agente único e multiagente. Agente único não existem outros agentes que utilizam a mesma medida de performance, como exemplo podemos citar: um agente que resolve um jogo de palavras cruzadas sozinho está claramente em um ambiente de agente único, enquanto que um agente jogador de futebol de robôs está em ambiente multiagente, tanto cooperativo (agentes do mesmo time) como também competitivos (agentes do time adversário) (POGGI; ADORNI, 1996).

1.1 MOTIVAÇÃO

FutVasf2D é um projeto que nasceu em 2017 com auxílio do Programa Institucional de Bolsas de Iniciação Científica (PIBIC) para o desenvolvimento de um time oficial de futebol robôs para Universidade Federal do Vale do São Francisco (UNIVASF) na categoria

de simulação 2D. O primeiro trabalho do projeto teve como objetivo o desenvolvimento de um módulo de decisão para o momento do chute a gol. O projeto está tendo continuidade com, além do presente trabalho, outro PIBIC que visa o desenvolvimento de um módulo de passes.

Durante o processo de desenvolvimento deste trabalho almeja-se por em prática conhecimentos tratados forma teórica na área de aprendizado de máquinas enfrentando o desafio da aplicação das técnicas tratadas em um ambiente complexo e prático. A técnica de aprendizado por reforço aplicada neste trabalho é a *Q-Learning* devido ao fato desta ser uma técnica comum e já utilizada por diversos trabalhos na área.

Com o término deste trabalho espera-se a participação do time desenvolvido na Competição Brasileira de Robótica (CBR) como meio de divulgar o projeto e angariar interessados no desenvolvimento de novas estratégias e melhoramento das já desenvolvidas no meio acadêmico da UNIVASF.

1.2 DEFINIÇÃO DO PROBLEMA

De acordo com Oliveira et al. (2009), uma partida de futebol apresenta situações que exigem dos jogadores a tomada de decisão acerca da ação a ser tomada. Quando um jogador está com a bola, por exemplo, ele deve decidir se deve passar a bola para um aliado, driblar um adversário ou chutar a bola para o gol. Todas essas ações são decididas através de avaliações do conhecimento acerca da situação atual do jogo. O mesmo pode ser afirmado para situações onde o time adversário está com posse de bola, neste caso, os jogadores devem sempre tomar a melhor decisão com base no estado atual para evitar sofrer gol e recuperar a posse de bola do adversário.

Alguns times que competem ou competiram em campeonatos da liga fornecem código-fonte de times, chamados de times bases, a fim de facilitar o desenvolvimento de novos times para novos competidores. É o caso dos times HELIOS, do Japão, que fornece o time base Agent2D¹ ou o WrightEagle, da China, que fornece o WrightEagleBASE², utilizado neste projeto. Tais times são desenvolvidos com estratégias simples para que possam ser melhoradas pelas novas equipes.

Gabel, Riedmiller e Trost (2008) afirmam que o uso de técnicas de aprendizagem para o desenvolvimento da capacidade defensiva de um time vêm sendo quase negligenciadas, de forma que as pesquisas focam, em sua maioria, em tarefas como a de passe, chute a gol ou posicionamento do jogador com posse de bola.

Ainda segundo Gabel, Riedmiller e Trost (2008), estratégias de defesa são divididas em 2 partes, o posicionamento dos jogadores de forma a melhor interceptar passes ou

¹ <http://rctools.osdn.jp/pukiwiki/>

² <https://wrighteagle2d.github.io/>

tentativas de chute ou marcar oponentes, e o de atacar o jogador com posse de bola a fim de obtê-la.

Neste trabalho, serão levadas em conta ambas as facetas das estratégias de defesa, de forma que o problema proposto pode ser definido como a identificação de melhor ação a ser tomada, utilizando dados obtidos do ambiente através de seus sensores, entre:

- Mover-se para uma melhor posição;
- Marcar um jogador adversário;
- Interceptar a bola e
- Bloquear avanço do jogador com posse de bola.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Investigar performance de modelos no desenvolvimento de módulo responsável por tomar decisões voltadas para defesa para o time de futebol de robôs da liga de simulação 2D FutVasf2D utilizando a técnica de aprendizado por reforço *Q-Learning*.

1.3.2 Objetivos Específicos

1. Desenvolver modelos aptos a representar o ambiente do futebol de robôs da liga de simulação 2D em algoritmos de aprendizado por reforço aplicados na solução do problema proposto;
2. Implementar os modelos propostos no algoritmo *Q-Learning* no time *WrightEagle-BASE*, usado como base para o FutVasf2D;
3. Treinar os times implementados através da execução de partidas com situações controladas;
4. Comparar resultados e identificar falhas e melhorias.

1.4 ORGANIZAÇÃO DO TRABALHO

Além do presente capítulo, Introdução, estão presentes neste trabalho 4 capítulos, Fundamentação Teórica (2), Materiais e Métodos (3), Resultados (4) e Conclusão (5).

O Capítulo 2 fundamenta teoricamente o trabalho apresentando na Seção 2.1 trabalhos relacionados que reforçam a metodologia por trás da execução deste, a Seção 2.2 explana o conceito e funcionamento do aprendizado por reforço e, na Subseção 2.2.1, faz o

mesmo com o algoritmo *Q-Learning*, ponto fundamental para o desenvolvimento deste trabalho.

O Capítulo 3 explica o processo proposto para o desenvolvimento dos modelos e coleta de resultados. Na Seção 3.1 é exposto o conceito de modelagem do ambiente e seus desafios. A Seção 3.2 explica o processo de implementação e as variações aplicadas, enquanto a Seção 3.3 trata da metodologia experimental, como o treinamento e a forma de avaliação.

O Capítulo 4 apresenta a descrição dos modelos desenvolvidos, a fundamentação de cada um e os resultados obtidos em seus respectivos treinamentos.

No Capítulo final 5 constam as devidas conclusões tomadas a partir do processo de desenvolvimento deste trabalho e dos resultados obtidos pelos experimentos, procurando responder indagações oriundas da análise dos dados do capítulo anterior. Também é trabalhado na Seção 5.1, o traçado de uma continuidade para o projeto iniciado apresentando propostas de novas investigações e o caminho a ser tomado.

2 FUNDAMENTAÇÃO TEÓRICA

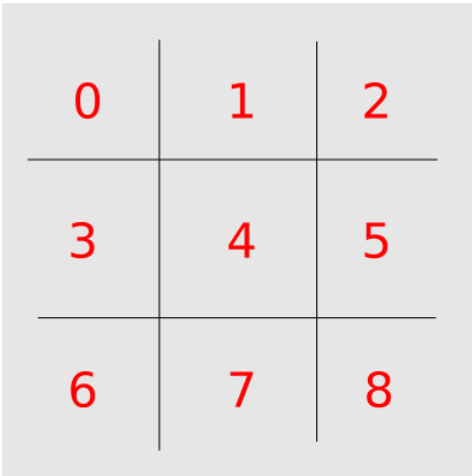
2.1 TRABALHOS RELACIONADOS

Diversos trabalhos foram redigidos utilizando aprendizado por reforço no domínio da *RoboCup Soccer Simulation 2D*, cada um desses trabalhos apresentam modelagens diferentes para o mundo.

A modelagem de mundo é a forma como o ambiente é representado, ou simulado. Um agente inteligente precisa constantemente entender como está o ambiente no qual ele está inserido, por isso é importante a definição da representação do mundo de forma que melhor se adéque ao processo de aprendizado do agente.

Um agente empregado no aprendizado do jogo-da-velha, pode ser utilizado com exemplo. O agente precisa entender como o jogo está sendo encaminhado para que o mesmo possa decidir em que posição jogar, porém, o agente isolado não tem acesso ao estado do jogo e precisa de uma representação do mesmo para que possa entendê-lo. Uma abordagem válida é numerar o tabuleiro conforme a Figura 2 e atrelar a cada célula um valor que varia de acordo com o que é marcado na mesma, uma escolha razoável é atribuir o valor “-1” à células preenchidas pelo jogador adversário, “0” à células não preenchidas e “1” às preenchidas pelo próprio agente. Nota-se que com essa representação o agente consegue perceber em que situação se encontra e assim escolher uma ação. Podemos chamar cada uma das células de variável, já que nelas se encontram os valores que determinam qual o estado atual do ambiente.

Figura 2 – Representação do tabuleiro de jogo-da-velha.



0	1	2
3	4	5
6	7	8

Fonte: O Autor

É fácil encontrar exemplos de problemas inseridos em ambientes que são mais complexos que o citado. Para esses problemas é comum encontrar variáveis de valores contínuos, nesses casos é necessário pensar em como representar as variáveis de forma discreta para que seja possível se obter um número finito de estados. Alguns dos trabalhos citados adiante demonstram maneiras de interpretar os dados do ambiente para que possa ser aplicada técnica de aprendizado.

Gabel, Riedmiller e Trost (2008) desenvolveram jogadores de defesa com comportamento agressivo, ou seja, jogadores que interferem e tentam tomar a bola do jogador adversário, utilizando aprendizado por reforço. O algoritmo desenvolvido foi chamado de NeuroHassle e utiliza redes neurais e uma abstração do mundo em 9 dimensões, distância entre o jogador e oponente com posse de bola; vetor de velocidade do jogador; velocidade absoluta do oponente com posse de bola; vetor de posicionamento da bola; ângulo do corpo do jogador em relação ao oponente com posse de bola; ângulo do corpo do adversário em relação ao gol aliado; ângulo definido pela posição do gol aliado, a posição do oponente com posse de bola e a posição do jogador. O treinamento foi feito de forma episódica em partidas de competições anteriores. O time gerado obteve uma taxa de falha menor que 20% e, em competição, cerca de 30 tomadas de bola por partida.

Uma variação de redes neurais com a utilização de *backpropagation*, chamada de *Backpropagation Resiliente* (RPROP, do inglês *Resilient Backpropagation*), foi utilizada por Riedmiller e Gabel (2005) no treinamento de jogadores sem posse de bola, ao mesmo tempo que a escolha de ações do jogador com posse de bola. Para isto foi feita a discretização do domínio em um Processo de Decisão Markoviano (MDP, do inglês *Markovian Decision Process*) que destacou as variáveis posição e velocidade da bola e a posição de todos os atacantes aliados e defensores adversários, além das possíveis ações, que para os jogadores com posse de bola consistem em passar a bola diretamente para um aliado, lançar a bola de forma que um aliado possa interceptar a bola antes de um adversário e driblar, já para os jogadores sem posse de bola, é possível se mover em 8 direções a partir do local atual ou a partir do local inicial ou mover-se para o local inicial. Este trabalho resultou no time vencedor do ano de 2007 da liga de simulação 2D da *RoboCup Soccer*.

A técnica *Q-Learning* é aplicada em alguns trabalhos, como o de Neri et al. (2012b), que utiliza desta técnica a fim de obter um sistema de tomada de decisões focado no ataque. Nesse trabalho, os estados foram discretizados em 7 estados, caracterizados como quando é possível lançar a bola para um aliado melhor posicionado; um adversário mais próximo está atrás do jogador com posse de bola; o adversário mais próximo está a mais de 7 metros; o adversário mais próximo está a uma distância entre 7 e 6 metros; o adversário mais próximo está a uma distância entre 6 e 5 metros; o adversário mais próximo está a uma distância menor de 4 metros e há uma linha direta entre o jogador com posse de bola e o gol adversário. Além disso, 7 ações podem ser tomadas, são elas, passar a bola para

um jogador mais próximo do gol adversário, avançar devagar, avançar rapidamente, passar a bola para um jogador próximo, driblar, segurar a bola ou chutar ao gol.

O modelo foi treinado e testado através da execução de conjuntos de 10 partidas. O conjunto de treinamento se deu contra o time base do Helios apenas na situação de posse de bola. Após o treinamento, o time foi tuilizado no primeiro conjunto de testes contra o time base do Helios, além disso, outro foi executado entre Helios e o PetSoccer, além de um entre o time treinado e o PetSoccer para comparação, e por fim outro contra o time chinês WrightEagles. No primeiro conjunto foram realizados 6 gols contra o Helios, o Helios executou 47 gols contra o PetSoccer, o time treinado conseguiu 61 gols contra o PetSoccer e 6 contra o WrightEagles. Mostrando que a utilização do *Q-Learning* resultou numa melhoria, ganhando a maioria dos jogos. A mesma técnica foi utilizada no time brasileiro GPR-2D de 2012, descrito em Neri et al. (2012a), com o objetivo de treinar a tomada de decisão para quando o jogador estiver com posse de bola. O time foi classificado e chegou a segunda fase do campeonato mundial da *RoboCup Soccer Simulation 2D*.

Liu e Li (2008 e 2009), utilizam *Q-Learning* com uma abstração do conjunto de agentes como um grande agente e o conjunto de suas ações como uma ação deste grande agente chamando os dois algoritmos criados de *Regional Cooperative Q-Learning* e *Sparse Cooperative Q-Learning* (*Q-Learning* Cooperativo Regional e *Q-Learning* Cooperativo Esperso). Nesses trabalhos foram identificados estados onde não é necessário que as ações sejam decididas em conjunto (não coordenados) e onde é (coordenados), utilizando uma abordagem baseado em campos potenciais. Na abordagem regional foram encontrados 5012 estados coordenados e 34390 não coordenados, já na abordagem esparsa, foram 4720 coordenados e 34682 não coordenados. O modelo foi treinado com uma simplificação de partidas com dois agentes defendendo e um adversário atacando e a região (reduzida para 40 x 20) dividida em 200 células. Foram rodados experimentos com 200000 e 500000 episódios comparando a técnica aplicada com a técnica MDP e a linear independente. Para 200000 episódios foi obtida uma média de tempo de captura de 6,57 segundos para a abordagem regional e 6,49 para a esparsa, mostrando-se o melhor dentre 3 métodos enquanto com 500000 episódios apresenta uma média de 6,49 segundos na abordagem regional e 6,40 na abordagem esparsa, maior apenas que o do método MDP que apresentou 6,23 segundos.

Alguns trabalhos utilizam variações do *Q-Learning*, como Carvalho e Oliveira (2011), que desenvolveu um módulo de drible unindo Computador Aritmético de Modelo Cerebelar (CMAC, do inglês *Cerebellar Model Arithmetic Computer*) e Estado-Ação-Recompensa-Estado-Ação (SARSA, do inglês *State-Action-Reward-State-Action*), este último algoritmo foi concebido como *Q-Learning* conexionista modificado. Neste trabalho a tarefa de drible foi mapeada como um problema episódico adequado para o trabalho com aprendizado por reforço onde foram destacadas duas ações primitivas, segurar a bola e

driblar (que consiste em girar o corpo a uma determinada angulação, chutar a bola a uma distância determinada e correr para interceptá-la) e as seguintes variáveis para compor um estado para o jogador que deverá driblar, o ângulo global de um objeto, o ângulo relativo entre dois objetos, a distância entre dois objetos, a altura e largura do campo e se o jogador está perto do topo ou da parte mais baixa do campo. O treinamento do modelo criado foi feito em 5 experimentos de 50000 episódios nos quais o jogadores não podiam recuperar a *stamina* por si só utilizando o ambiente padrão e um ambiente de treino de 20 x 20. Para testar foram executados 10000 cenários com configurações aleatórias. No fim do treinamento foi obtido um número de vitórias de cerca de 53% e 58% dos testes pós-treino.

O SARSA também foi empregado em Homem et al. (2017) em conjunto com o Raciocínio Espacial Qualitativo (QSR, do inglês *Qualitative Spatial Reasoning*), chamando o resultado de Aprendizado por Reforço Qualitativo (QRL, do inglês *Qualitative Reinforcement Learning*) e aplicou no desenvolvimento de um mecanismo de ataque para um time de simulação 2D. Para alcançar o objetivo foi utilizado o formalismo eOPRAm para discretizar o mundo em regiões qualitativas pelo qual foram identificadas um total de 14 variáveis, são elas a posição e orientação do jogador, a distância e ângulo do jogador em relação à bola, se o jogador pode ou não chutar, distância e ângulo do jogador em relação ao gol adversário, maior ângulo aberto para o gol, distância entre os aliados e o adversário mais próximo, maior ângulo aberto para passe para cada aliado, numeração de cada aliado e de cada adversário.

As possíveis ações foram definidas como driblar, chutar e passar a bola para um aliado específico. O treinamento foi realizado utilizando a ferramenta Ofensa de Meio Campo (HFO, do inglês *Half-Field Offense*) 30 vezes de forma independente com 1000 episódios em 3 situações diferentes, a primeira com apenas um jogador atacando sem nenhuma defesa, a segunda com apenas o atacante e o goleiro e a terceira com o atacante, o goleiro e um zagueiro. Foi realizada uma comparação da técnica utilizada com a abordagem quantitativa utilizando análise de variância. Na primeira situação o resultado foi de uma média de 99,3% de gols se mostrando melhor que a quantitativa com certeza de 95%, enquanto a segunda e terceira situação se mostraram melhor com margem de 1% com 81,2% e 47,6% contra 78,3% e 40,6% respectivamente.

Além dessas variações, algumas foram empregadas com o intuito de acelerar o aprendizado, este foi o caso de Celiberto et al. (2007) e Bianchi e Mantaras (2010). O primeiro, utilizou a chamada *Q-Learning* Acelerado Heuristicamente (HAQL, do inglês *Heuristically Accelerated Q-Learning*) para treinar um goleiro e jogador de defesa. Para tanto, foi identificada uma heurística aplicável ao domínio e implementada no algoritmo *Q-Learning* ao mesmo tempo que é feita a discretização do mundo. As variáveis de estado para o algoritmo foram a posição dos agentes e da bola numa grade e a direção para a qual o agente está virado definida por norte, sul, leste ou oeste. As ações que podem ser

tomadas são mudar a direção para a qual o agente está virado para um objeto específico, mover em direção a bola, mover-se com a bola, passar a bola para o goleiro, chutar a bola para longe do gol mover-se para perto de um oponente. Foram executadas 10 sessões de treinamento para o algoritmo criado, *Q-Learning* puro e a função heurística pura com 100 episódios cada, onde cada episódio consiste de uma partida de 3000 ciclos. Foi analisada a curva de aprendizado e o número de gols sofridos para cada algoritmo. E utilizado o teste *t-student* que validou a hipótese de que a aplicação de heurística no *Q-Learning* acelera o aprendizado com nível de confiança de 95% e uma taxa de gols sofridos menor que as demais opções testadas.

Já Bianchi e Mantaras (2010), utilizam heurística baseada em casos para acelerar o *Minmax-Q*, que também é uma variação do *Q-Learning*, para melhorar a tomada de decisões. Os casos destacado para utilização no algoritmo foram o de que caso o jogador esteja com a bola e não exista adversário no caminho, avança para o gol; se houver um adversário bloqueando, mova para cima ou para baixo; se houver um aliado mais próximo do gol, passa a bola e se o adversário estiver com a bola e o jogador estiver próximo, fica na frente do adversário. O treinamento foi feito com 20000 jogos de 10 tentativas em um domínio chamado “futebol expandido de Littman”. Chegou-se ao resultado de que a solução encontrada não é ótima, mas a hipótese de que o aprendizado é acelerado foi comprovado, através do teste de *t-student* e uma comparação dos resultados das partidas e das curvas de aprendizado.

Diante dos exemplos apresentados e considerando a natureza do problema, fica decidido utilizar uma técnica de aprendizado por reforço, mais especificamente *Q-Learning*, para o treinamento de um modelo de tomada de decisões de defesa do time em desenvolvimento, *FutVasf2D*.

2.2 APRENDIZADO POR REFORÇO

Segundo Shalev-Shwartz e Ben-David (2014), aprendizado por reforço é uma técnica de aprendizado intermediária, entre supervisionado e não supervisionado, onde o agente precisa realizar mais previsões durante os “testes”.

Russell e Norvig (2016) afirmam que aprendizado por reforço é necessário quando não há um bom “professor” para a tarefa a ser executada. Apontam ainda que, através de ações aleatórias, o modelo é capaz de “aprender” a realizar previsões sobre o ambiente. Logo, o papel do aprendizado por reforço é, utilizando-se de recompensas, encontrar uma função eficaz para o agente.

Uma analogia apontada por Russell e Norvig (2016) é a que compara o aprendizado por reforço com a forma que seres vivos aprendem. Animais são capazes de perceber quando específicas sensações são recompensas positivas ou negativas e essa é uma característica

que facilita o adestramento de cães, por exemplo, no qual o cão consegue entender que fez algo positivo através de recompensas como petiscos ou carinho ou negativo como a repreensão em determinado tom de voz.

Aprendizado por reforço é aprender o que fazer - como mapear situações em ações - de forma a maximizar uma recompensa numérica. Aquele que aprende não é informado quais ações tomar, no entanto, deve descobrir quais ações resultam em uma maior recompensa tentando executá-las. (SUTTON; BARTO, 2018)

Segundo Sutton e Barto (2018), existem alguns elementos que devem ser conceituados em prol de melhor entender o aprendizado por reforço, são eles, política, sinal de recompensa, função valor e modelo de ambiente.

- **Política** é o que define a forma como o agente deve se comportar em determinado momento. É um mapeamento entre os possíveis estados do ambiente e ações.
- **Sinal de recompensa** é um valor numérico que define o quão determinada ação afeta de forma positiva ou negativa na tentativa de alcançar o objetivo.
- **Função de valor** se refere ao que o agente espera acumular de recompensa no futuro. É como a função de recompensa, porém, ao invés de um valor imediato, retorna um valor para um intervalo de tempo maior.
- **Modelo de ambiente** é uma simulação do ambiente que permite que sejam feitas interferências em como o ambiente se comportará. Modelos são usados para prever como estará o ambiente sem precisar alterá-lo de fato para melhor planejar as ações.

A Figura 3 resume o funcionamento do aprendizado por reforço como o já dito.

Figura 3 – Aprendizado por reforço.



Fonte: O Autor

Russell e Norvig (2016) descrevem como o aprendizado por reforço pode variar, tanto em relação ao tipo de ambiente quanto ao agente:

- O ambiente pode ser acessível ou inacessível. No primeiro caso o agente consegue detectar o estado do ambiente através de sensores, enquanto que em ambientes inacessíveis o agente precisa manter e atualizar um estado interno.
- O agente pode saber *a priori* sobre os estados do ambiente e como suas ações podem interferir no mesmo ou pode ter que aprender essas informações durante o treinamento.
- Recompensas podem ser recebidas em qualquer estado ou apenas em estados terminais.
- Recompensas podem ser valores que o modelo tenta maximizar existentes no sistema, como pontuação num jogo de tênis de mesa, ou valores simbólicos que funcionam como “dicas” de quão bem o agente está indo.
- Os agentes podem ser passivos ou ativos quanto a aprendizagem. Os agentes passivos apenas observam o ambiente mudar e tenta entender o valor de estar em vários estados, enquanto o ativo age conforme aprende e pode usar o gerador do problema para explorar estados desconhecidos.

Sutton e Barto (2018) afirmam que o aprendizado por reforço depende bastante do conceito de estado e o conceito de estado aplicado neste domínio é o estado de um MDP. Ainda segundo Sutton e Barto (2018), MDP é uma forma idealizada matematicamente de um problema de aprendizado por reforço para o qual é possível realizar declarações teóricas precisas.

Em geral, Chan e Asgarpour (2006) definem MDP como uma quádrupla (S, K, R, T) , onde S é o conjunto finito de possíveis estados, K é o conjunto finito de possíveis ações a se tomar, R é o conjunto de recompensas imediatas para cada relação estado-ação e T é o conjunto de probabilidades de transições para cada relação estado-ação.

Existem diversos *designs* básicos para os agentes, como no aprendizado por reforço os agentes devem receber recompensas de acordo com a utilidade da ação, Russell e Norvig (2016) destacam dois principais *designs*:

- Onde o agente aprende a função de utilidade para os estados e usa isto para prever quais ações podem trazer um resultado melhor no futuro.
- Quando o agente aprende uma função para a relação ação-estado e o retorno desta função define qual melhor ação tomar para cada estado.

Este último *design* é a base do método a ser utilizado neste trabalho e é conhecido como *Q-Learning*.

2.2.1 Q-Learning

Segundo Watkins e Dayan (1992), *Q-Learning* “confere aos agentes a capacidade de aprender a agir otimamente em um domínio Markoviano experimentando as consequências de suas ações sem a necessidade da construção de mapas do domínio”.

Como já explicado, estar em um ambiente Markoviano envolve um ambiente discreto com número finito de estados e possíveis ações.

Q-Learning utiliza uma ferramenta chamada de *Q-table*, de acordo com Simonini (2018), esta ferramenta consiste em uma tabela onde cada coluna corresponde a uma possível ação a ser tomada e cada linha um possível estado a ser alcançado. Cada célula, então, corresponde ao valor de recompensa máximo esperado para o par ação-estado, aqui chamado de Q . Um exemplo de representação para esta tabela é demonstrada na Tabela 1. É possível perceber que a tabela funciona como uma espécie de guia para escolher a melhor ação para o estado atual.

	Ação 1	Ação 2	...	Ação N
Estado 1	0.1	2.3	...	10
Estado 2	0.5	1.2	...	5.6
...
Estado M	11.5	3.7	...	0.3

Tabela 1 – *Q-Table*

Para se atualizar o valor Q na tabela para um estado s_i quando tomada a ação a_j , Simonini (2018) fornece a seguinte equação:

$$Q_{novo}(s_i, a_j) = Q(s_i, a_j) + \alpha[R(s_i, a_j) + \gamma \max_{a'}(Q(s_k, a')) - Q(s_i, a_j)] \quad (2.1)$$

Onde α é a taxa de aprendizagem, γ é taxa com a qual o valor de Q deve decair com o tempo, s_k é o estado a ser alcançado ao se tomar a ação a_i e a' são todas as possíveis para que seja calculado o valor máximo de Q para o estado s_k .

O algoritmo a ser seguido para treinar se chegar à melhor *Q-Table* é descrito no pseudocódigo em Código 1 e é uma adaptação do algoritmo fornecido por Simonini (2018).

Código 1 – Algoritmo de *Q-Learning*

```

1 Inicializar Q-table
2 Enquanto não alcançar critério de parada
3     Escolher a próxima ação a
4     Executar a
5     Medir recompensa
6     Atualizar Q

```

Fonte: O autor

O critério de parada para o algoritmo pode ser adotado como a convergência dos valores da tabela ou o número de episódios rodados e a recompensa gerada depende do problema abordado. A utilização do *Q-Learning* consiste na execução do treinamento, onde é calculada toda a *Q-Table* e então a execução do algoritmo novamente, sem necessariamente atualizar a tabela e utilizando os valores já encontrados para decidir a ação a ser tomada.

É interessante destacar aqui, que a escolha da próxima ação, no treinamento, é feita a partir de uma probabilidade que decidirá se será tomada uma ação aleatória (chamado de exploração) ou se será consultada a tabela encontrada até agora.

Um exemplo de utilização do *Q-Learning* foi desenvolvido pelo autor e está disponível em um repositório no *GitHub*¹. O problema consiste de um tabuleiro como o mostrado na Figura 4 onde o personagem deve aprender a se mover em uma das quatro direções (cima, baixo, esquerda ou direita), evitando o monstro, desviando das paredes e tentando chegar ao objetivo, representado por uma bandeira.

A modelagem deste problema consistiu em um estado para cada possível posição do personagem (25 possibilidades, identificadas na Figura 5a) e quatro ações (subir, descer, ir para direita ou ir para a esquerda). As recompensas foram modeladas de forma que, uma ação que resulte em o personagem chegando em uma célula vazia, implica em 1 ponto de recompensa, o personagem tentando ir para uma célula ocupada por uma parede ou fora do tabuleiro retorna -1 ponto, o personagem tentando entrar na célula ocupada pelo monstro resulta em -20 pontos e chegar ao objetivo 20 pontos. Foram executados 5000 episódios para o treinamento. O treinamento resultou na Tabela 2.

Estando o personagem inicialmente no estado 20, observando os resultados na *Q-Table*, é possível traçar o caminho para vitória como o mostrado na Figura 5b que, notavelmente, é o caminho mais eficiente.

¹ NASCIMENTO, J. P. F. [joaopedrofn/learnToWalk](https://github.com/joaopedrofn/learnToWalk). 2018. Disponível em: <<https://github.com/joaopedrofn/learnToWalk>>.

Figura 4 – Tabuleiro do problema exemplo.



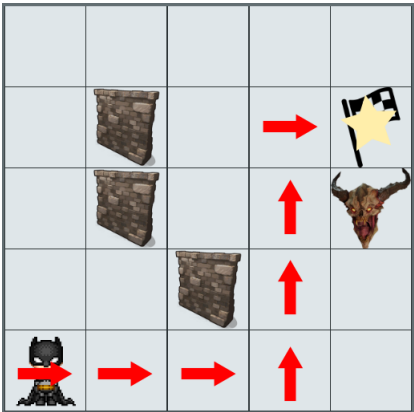
Fonte: O Autor

Figura 5 – Representação dos estados e solução encontrada pelo *Q-Learning*

(a) Respresentação de estados 1



(b) Solução



Fonte: O Autor

	UP	DOWN	LEFT	RIGHT
0	2,036	3,539	2,090	5,136
1	3,112	2,965	4,019	6,719
2	4,634	4,575	4,910	9,256
3	7,253	13,260	6,720	13,36
4	11,289	20	9,250	11,322
5	4,147	3,144	1,406	1,004
6	0	0	0	0
7	6,719	6,642	7,139	13,36
8	9,256	9,256	9,256	20
9	0	0	0	0
10	3,548	2,898	1,090	1,130
11	0	0	0	0
12	9,256	4,674	4,593	9,251
13	13,360	6,720	6,720	-14,279
14	0	0	0	0
15	3,175	3,216	0,985	3,216
16	1,216	3,586	2,987	1,216
17	0	0	0	0
18	9,256	5,153	4,720	5,153
19	-19,999	4,076	6,720	2,966
20	2,988	1,216	1,216	3,586
21	3,216	1,586	3,216	4,185
22	2,185	2,185	3,586	5,153
23	6,720	3,153	4,185	4,185
24	4,956	2,184	5,153	2,183

Tabela 2 – *Q-Table* resultante da execução do experimento.

3 MATERIAIS E MÉTODOS

Como já especificado, o trabalho tem como objetivo a investigação de modelos a serem utilizados no desenvolvimento de um módulo de defesa para o time de Futebol de Robôs na categoria de simulação 2D, FutVasf2D. O processo de modelagem de um mundo em um MDP exige a discretização do mesmo de forma que sejam obtidos um número finito de ações e de estados com os quais o algoritmo de aprendizado de reforço possa trabalhar, sendo assim, a próxima etapa é definir como o algoritmo escrito pode ser implementado no time base escolhido (*WrightEagleBASE*).

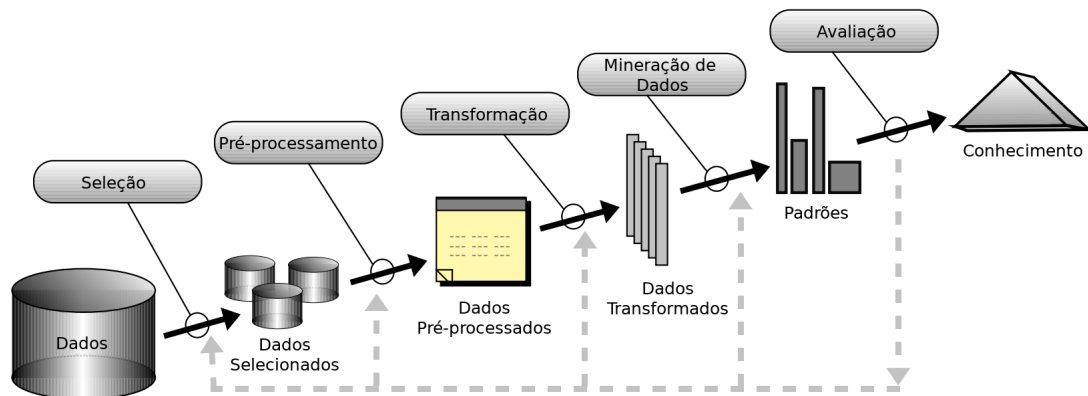
3.1 MODELAGEM DO MUNDO

Os trabalhos apresentados na Seção 2.1 oferecem várias possibilidades de como adaptar o ambiente da partida para um problema que se encaixe às definições de MDP. Algumas das variáveis destacadas por Gabel, Riedmiller e Trost (2008), Celiberto et al. (2007), Bianchi e Mantaras (2010) e Homem et al. (2017) estão descritos a seguir:

- Posição do jogador;
- Posição da bola;
- Orientação do corpo do jogador;
- Distância entre o jogador e a bola;
- Velocidade do jogador;
- Distância entre jogador e adversário mais próximo;
- Ângulos diversos, como ângulo entre jogador e adversário, ângulo formado entre bola, jogador e gol adversário.

Segundo Fayyad, Piatetsky-Shapiro e Smyth (1996), a Descoberta de Conhecimento em Base de Dados (do inglês *Knowledge Discovery in Databases* - KDD) é o processo de identificar padrões válidos, novos, potencialmente úteis, e inteligível em dados. O KDD é composto de 5 etapas, seleção, pré-processamento, transformação, mineração de dados e avaliação, como mostrado na Figura 6, junto com cada artefato gerado. Na etapa de pré-processamento são realizadas manipulação de variáveis pré-existent e identificações de novas.

A construção de novas variáveis é uma forma sistemática de incorporar conhecimento em um projeto de KDD. Essas variáveis serão úteis na etapa de mineração de dados, que

Figura 6 – Etapas de KDD.

Fonte: (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996)

no caso de aprendizado por reforço, acontece no momento do treinamento. Segundo Hastie, Friedman e Tibshirani (2009), apesar de ser um dos mais antigos, o processo de construção de novas variáveis é um dos mais desafiadores. De acordo com Witten et al. (2016), a melhor forma de construir variáveis é a manual, baseando-se no entendimento do problema e significado de cada atributo. No geral, essa tarefa depende muito mais do conhecimento sobre o domínio do que da construção do algoritmo, por tanto, o conhecimento sobre o domínio é essencial (ZHAO; SINHA; GE, 2009). O uso de conhecimento de domínio para construção de variáveis aumenta o poder preditivo do modelo (ZHAO; SINHA; GE, 2009). A estratégia de enriquecimento de dados para alcançar uma maior capacidade discriminatória é ainda maior em áreas como as de reconhecimento de som e imagem, de acordo com Gao et al. (2008). As soluções de melhor performance em competições internacionais utilizaram a estratégia de construção de novas variáveis a fim de incorporar conhecimento do domínio, segundo Adeodato et al. (2008, 2009).

Partindo das informações e exemplos obtidos através da revisão bibliográfica, foram modeladas algumas variações de formas de representar o mundo. Quatro componentes são importantes neste processo, os estados que podem ser alcançados pelo agente, as ações que o mesmo pode tomar, as recompensas e punições a serem atribuídas para o resultado das ações em cada estado e como será feita a implementação de todo o modelo no agente. Os 3 primeiros componentes são cobertos nas subseções abaixo enquanto a implementação é abordada na Seção 3.2.

3.1.1 Estados

A definição dos estados é uma etapa certamente crucial e complexa, neste trabalho foram tomadas duas abordagens. A primeira foi sobre a identificação de variáveis cujos

diferentes possíveis valores formam os estados, a outra se baseia na identificação de possíveis papéis de função que os agentes podem tomar no processo de defesa e uma simplificação das possíveis situações que os mesmos podem estar encaixados.

Especialmente, mas não exclusivamente, na primeira abordagem, é necessário um cuidado sobre a questão de quais serão os possíveis valores de cada variável, visto que esta decisão impactará na complexidade e quantidade de estados além do significado de cada um. Várias das variáveis abordadas a seguir possuem caráter contínuo e uma amplitude de possíveis valores muito grande, por conta disto foi feita um agrupamento de seus valores em 3 classes através da análise dos valores assumidos pelas variáveis na execução de várias partidas. As variáveis identificadas e seus possíveis valores foram as seguintes:

- **Posição do jogador em relação à bola no eixo horizontal** (P_{hj}), ou seja, se o jogador está mais a direita ou a esquerda da bola, esta variável pode assumir valores booleanos onde VERDADEIRO implica que o jogador está a esquerda e FALSO a direita;
- **Posição do jogador em relação à bola no eixo vertical** (P_{vj}), se identifica se o jogador pode ser encontrado mais acima ou abaixo da bola, assim como o item passado, os valores assumidos são VERDADEIRO para quando o jogador está acima da bola e FALSO quando abaixo, esta variável e a anterior são úteis para questão da direção de movimentação do jogador;
- **Distância entre o jogador e bola** (D_{jb}), mede o afastamento do agente em relação à bola, as classes atribuídas para as variáveis de distância são as mesmas (PERTO, MÉDIO e LONGE), diferindo apenas nos valores atribuídos a cada classe, nesta, os valores para PERTO é sempre menor ou igual a 10 (unidade de medida utilizada pelo servidor), para MÉDIO os valores são entre 10 e 20 e LONGE engloba todos acima de 20;
- **Distância entre o aliado mais próximo da bola e a bola** (D_{tb}), aqui os intervalos são menores, podendo assumir PERTO quando o valor da distância é menor ou igual a 5, MÉDIO quando entre 5 e 15 e LONGE quando acima de 15;
- **O agente é o aliado mais próximo da bola** (F), trata-se de um valor booleano a fim de identificar se o jogador “pensante” é o aliado mais próximo à bola;
- **Distância do adversário com posse de bola do gol** (D_{ag}), mede o afastamento do jogador com posse de bola do gol aliado, aqui os valores para PERTO são todos os que forem iguais ou abaixo de 11, MÉDIO quando entre 11 e 22 e LONGE quando acima de 33;

- **Compactação** (σ), consiste numa variável construída a partir das distâncias dos jogadores entre si, com foco no adversário com posse de bola. Esta variável é importante pois serve como uma maneira de reconhecer a dispersão dos jogadores no campo possibilitando o fechamento de um “cerco” com menos brechas em torno da bola. O cálculo desta variável é feito como um desvio padrão com base na distância média entre os jogadores aliados e o jogador com posse de bola, como na Equação 3.1, onde d_i é a distância entre o jogador i e o adversário com posse de bola e m é a média dessas distâncias, as classes atribuídas a essa variável foram ESPARSO (valores acima de 14), COMPACTO (entre 14 e 7) e MUITO COMPACTO (abaixo de 7).

$$\sigma = \sqrt{0,1 \times \sum (d_i - m)^2} \quad (3.1)$$

O estado do modelo descrito, então, passa a ser representado por uma 7-tupla, $(P_{hj}, P_{vj}, D_{jb}, D_{tb}, F, D_{ag}, \sigma)$. Como consequência da quantidade de variáveis e de seus valores, o campo de estados passa a assumir 648 possibilidades.

Para a segunda abordagem, o campo de estados foi mais simples, baseando-se em conceitos básicos de defesa no futebol real, foram atribuídos os papéis de Primeira Defesa, Segunda Defesa e Terceira defesa para o jogador mais próximo da bola, o segundo mais próximo e o terceiro mais próximo, respectivamente. Além disso, o valor de compactação já descrito foi reaproveitado para implementar o conceito de balanceamento no restante do time, resultando na modificação da Equação 3.1 para a Equação 3.2 e utilizando o mesmo conceito de classe anterior, atribuindo-o porém a novos estados ao invés de valores de um estado único.

$$\sigma = \sqrt{\frac{\sum (d_i - m)^2}{7}} \quad (3.2)$$

Desta forma essa abordagem resulta num campo de 7 estados que são:

- **Primeira Defesa**, se o agente pensante assume o papel de primeira defesa;
- **Primeira Defesa Muito Perto**, se o agente pensante assume o papel de primeira defesa e está a uma distância menor ou igual a 5 da bola;
- **Segunda Defesa**, se o agente tomando a decisão assume o papel de segunda defesa;
- **Terceira Defesa**, se agora é assumido o papel de terceira defesa;
- **Esparso**, se o jogador não se encaixa nos estados anteriores e os jogadores estão muito espalhados no campo;

- **Compacto**, se os jogadores estão menos espalhados;
- **Muito Compacto**, se os jogadores sem papel definido estão juntos no campo;

Podendo esta modelagem ser considerada muito simples, em outra tentativa foram atribuídas variações para os jogadores com papéis definidos, resultante no seguinte campo de 12 estados:

- **Primeira Defesa Muito Perto**, se o agente pensante assume o papel de primeira defesa e está a uma distância menor que 5 da bola;
- **Primeira Defesa Perto**, se o agente pensante assume o papel de primeira defesa e está a uma distância menor que 6 e maior que 5 da bola;
- **Primeira Defesa Médio**, se o agente pensante assume o papel de primeira defesa e está a uma distância menor que 7 e maior que 6 da bola;
- **Primeira Defesa Longe**, se o agente pensante assume o papel de primeira defesa e está a uma distância maior que 7 da bola;
- **Primeira Defesa Atrás**, se o agente pensante assume o papel de primeira defesa e está atrás da bola;
- **Segunda Defesa**, se o agente tomando a decisão assume o papel de segunda defesa;
- **Segunda Defesa Com Primeiro Atrás**, se o agente tomando a decisão assume o papel de segunda defesa e o agente que assume o papel de primeira defesa se encontra atrás da bola;
- **Terceira Defesa**, se agora é assumido o papel de terceira defesa;
- **Terceira Defesa Com Segundo Atrás**, se agora é assumido o papel de terceira defesa e o jogador que assume o papel de segunda defesa se encontra atrás da bola;
- **Esparso**, se o jogador não se encaixa nos estados anteriores e os jogadores estão muito espalhados no campo;
- **Compacto**, se os jogadores estão menos espalhados;
- **Muito Compacto**, se os jogadores sem papel definido estão juntos no campo;

Esses foram os estados utilizados nos experimentos descritos no decorrer deste trabalho, na subseção seguinte será abordada a questão da escolha de possíveis ações para os agentes.

3.1.2 Ações

A escolha das ações foi pensando no que pode ser feito, enquanto no papel de defesa numa partida de futebol, a fim de capturar a bola e impedir o avanço do time adversário. A implementação original do time base define 3 possíveis ações para a situação de defesa, mover-se para uma melhor posição, bloquear o avanço do jogador com posse de bola e marcar adversários para impedir a recepção de bola. Para este trabalho foram mantidas estas ações com algumas modificações e adições.

Primeiramente, a ação de se mover foi dividida em 5 possíveis ações, mover-se para cima, para baixo, para esquerda, para direita ou diretamente em direção a bola, desta forma é eliminada a avaliação original do time base garantido que o aprendizado seja exclusivamente baseado no algoritmo *Q-Learning*.

Além dessa divisão foi adicionada a ação de interceptação e a possibilidade de não fazer nada. A ação de interceptação consiste na tentativa de dominar a bola quando a mesma está ao alcance do agente.

Vale destacar que essas foram as ações levantadas para ser utilizadas, mas isto não implica que em todos os modelos implementados tenham sido utilizadas todas as ações.

3.1.3 Recompensas

As recompensas são partes essenciais do aprendizado por reforço, visto que são elas as responsáveis por informar ao agente se a ação tomada obteve resultados positivos ou negativos e o quão positivo ou negativo foi a ação. Esta decisão não é fácil de ser tomada e por isso foram feitas algumas implementações a fim de procurar otimizar esses valores.

A primeira tentativa de modelar um bom sistema de recompensas se baseou na ocorrência dos principais eventos na simulação, gols, bolas jogadas para fora do campo, captura de bola e passes. Esta modelagem adotou como recompensa para captura de bola +20, para gols sofridos -20 e para bolas lançadas para fora o valor +10, visto que implica no sucesso na tentativa de impedir possíveis ações para o agente adversário, convenção mantida em todas as modelagens seguintes, e para passe foi atribuído o valor -5 numa tentativa de impedir o time adversário de se articular.

A segunda implementação aboliu a recompensa de passes e adicionou uma verificação no avanço e recuo da bola, atribuindo -5 ao avanço da bola, +5 ao recuo da bola somada à aproximação do agente da captura da bola e +2 ao recuo da bola sem avanço dos aliados. Esta modelagem se mostrou bastante falha pela punição excessiva no avanço da bola, visto que ele é uma característica constante nas situações simuladas, desta forma a última implementação removeu esta punição e atribuiu recompensa neutra a esta e como padrão para não identificação de nenhuma das situações anteriores.

3.2 IMPLEMENTAÇÃO EM *WRIGHTEAGLEBASE*

O último ponto a ser abordado sobre a modelagem está na implementação do algoritmo, mas para que se possa tratar da implementação realizada, se faz necessário primeiro entender o funcionamento atual do time base escolhido.

O time base *WrightEagleBASE* conta com um mecanismo modularizado para tomada de decisões. A sua implementação depende do “tipo” de classe *Behavior* (comportamento), que é constituído por classes mais especializadas, como comportamento de bloqueio ou de interceptação, e menos especializadas, como comportamento de defesa ou ataque. As classes menos especializadas, na implementação original, criam uma forma de hierarquia para execução das mais especializadas que irão decidir se deve tomar a ação correspondente a classe ou não.

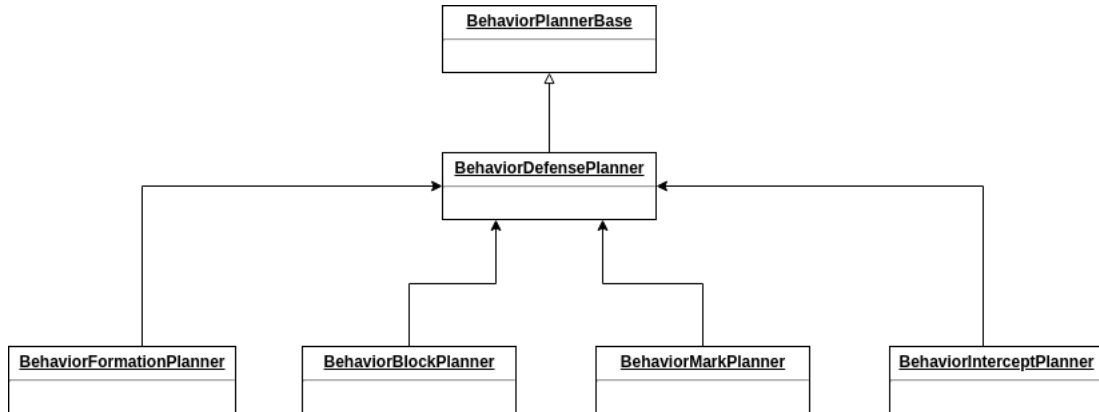
Este trabalho foca em 5 dessas classes, *BehaviorDefensePlanner*, *BehaviorFormationPlanner*, *BehaviorBlockPlanner*, *BehaviorMarkPlanner* e *BehaviorInterceptPlanner*, que são descritas a seguir. O diagrama representado na Figura 7, mostra como é o diagrama de classes simplificado da implementação original focando nas classes *Behavior*, mas especificamente na *BehaviorDefensePlanner*, vale destacar que a classe *BehaviorPlannerBase* corresponde a uma classe abstrata que é implementada por todas as demais.

- ***BehaviorDefensePlanner*** decide entre quais comportamentos o agente deve decidir quando o time está sem posse de bola e em qual ordem, na implementação original apenas define uma ordem de execução dos testes de cada possível ação de defesa;
- ***BehaviorFormationPlanner*** coordena a movimentação do time, decide quando o agente precisa ser realocado;
- ***BehaviorBlockPlanner*** é responsável por definir quando o agente deve tentar bloquear o adversário;
- ***BehaviorMarkPlanner*** decide se há a necessidade de determinado agente marcar um adversário;
- ***BehaviorInterceptPlanner*** busca a possibilidade de interceptar a bola e se deve tentar realizar tal ação.

Um exemplo de como isso é aplicado é dado pela implementação original do mecanismo de defesa que acontece da seguinte maneira. A classe de defesa define que a ordem de execução dos planejadores mais específicos é a seguinte:

1. *BehaviorFormationPlanner*

Figura 7 – Diagrama de classes simplificado para mecanismo de defesa.



Fonte: O Autor

2. *BehaviorBlockPlanner*

3. *BehaviorMarkPlanner*

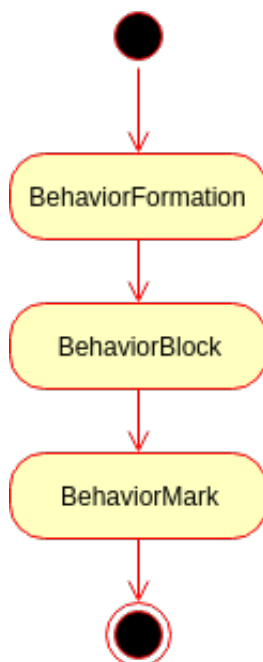
Desta forma, o agente primeiro decidirá se precisa se movimentar e para que local e só em seguida decidirá se precisa realizar um bloqueio e apenas depois disso, se precisa marcar algum jogador e qual. Desta forma, se garante que o jogador só se preocupará com bloqueio ou marcação depois de estar na posição adequada. O fluxograma do algoritmo de decisão das 5 classes, *BehaviorDefensePlanner*, *BehaviorFormationPlanner*, *BehaviorBlockPlanner*, *BehaviorMarkPlanner* e *BehaviorInterceptPlanner* podem ser vistas nas Figuras 8, 9, 10, 11 e 12, respectivamente.

Na implementação dos modelos, as classes mais especializadas (*BehaviorMarkPlanner*, *BehaviorBlockPlanner*, *BehaviorFormationPlanner* e *BehaviorInterceptPlanner*) foram modificadas ou deixaram de ser utilizadas para melhor se adequar ao processo de tomada de ação proposto.

Desde a primeira implementação do mecanismo de movimento a classe *BehaviorFormationPlanner* deixou de ser utilizada para dar lugar a implementação direta do mecanismo de movimentação provido pelo *framework* do time base. Em uma implementação futura o mesmo foi feito com as demais classes, mas a princípio as classes foram apenas modificadas removendo as condições que decidiam se a ação deveria ou não ser executada.

Além das classes *Behavior*, são importantes citar duas classes modificadas no desenvolvimento deste trabalho, são elas *Player* e *Agent*. A primeira foi utilizada para a implementação da utilização da *Q-Table* por se tratar de uma etapa externa aos comportamentos possibilitando a detecção de eventos que encerra o episódio, como a captura de

Figura 8 – Algoritmo da classe *BehaviorDefensePlanner*



Fonte: O Autor

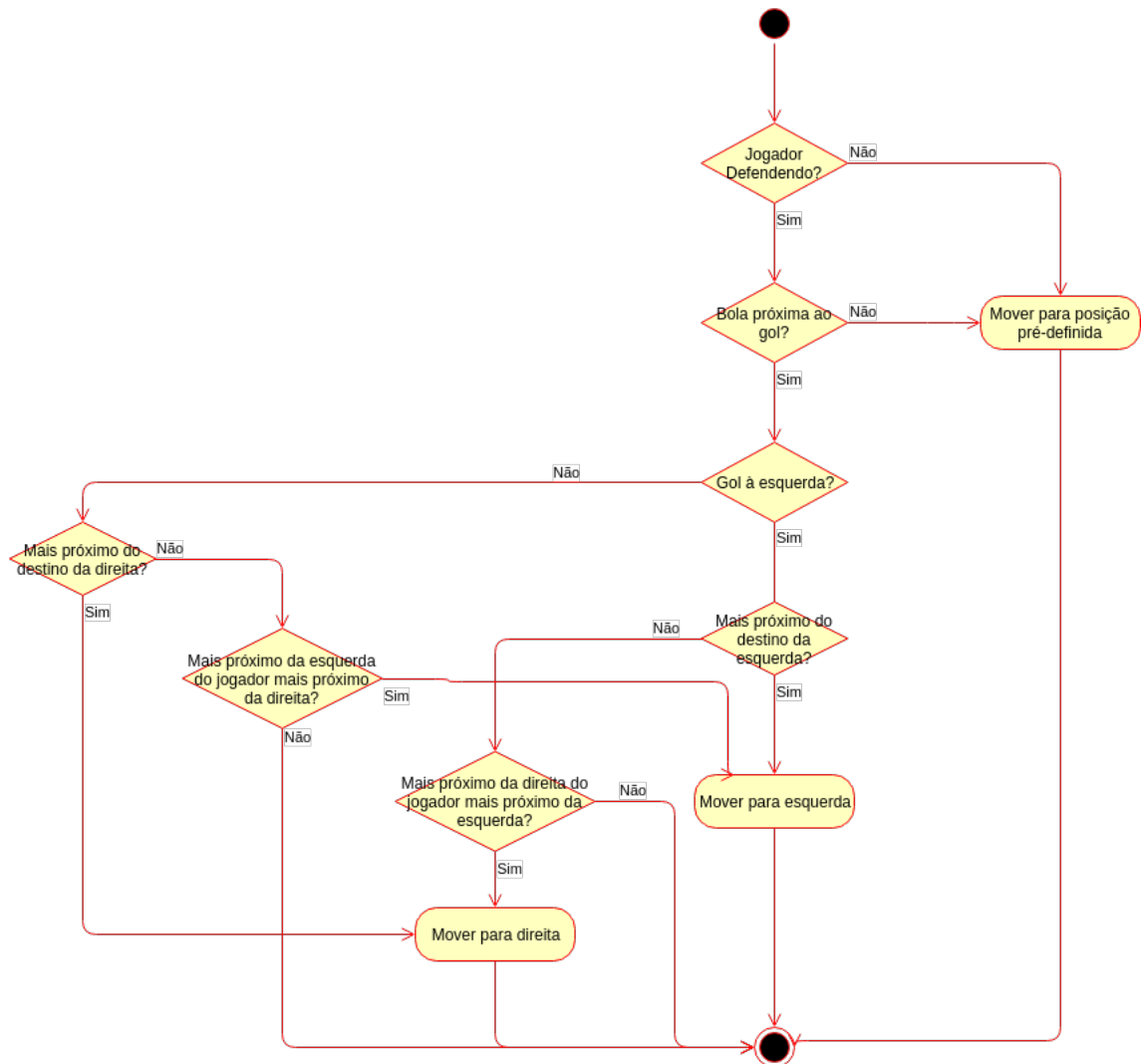
defesa, que fazem com que a classe *BehaviorDefensePlanner* não seja acessada no instante devido. Já a classe *Agent* foi utilizado apenas como meio de guardar informações sobre o agente a ser compartilhados entre as classes, visto que esta é a classe responsável pela descrição do agente.

A classe *Player*, portanto, foi tomada com uma certa preocupação devido ao tempo (medido em ciclos) para se considerar que a ação tomada obteve resultados. Para tratar desta questão, foi desenvolvido uma fila de ações e seus respectivos estados para que depois de um determinado número de ciclos pudessem ser avaliadas, com exceção de estados terminais (como gols, capturas e gols fora do campo) que quando ocorrem é tomada a primeira ação da fila e em seguida limpada novamente, tomando assim a ação mais velha que poderia ter causado o alcance a tal estado. O tamanho da fila foi alterado para tentar chegar ao melhor valor, assumindo os valores de 2 ciclos, 5 ciclos, 10 ciclos ou 1 ciclo (ação imediata).

Vale destacar ainda a implementação da *Q-Table* em si, a principio houve a tentativa de utilizar apenas um arquivo binário com a estrutura de dados de matriz de *double* para guardar as informações entre todos os agentes aliados em campo, porém esta abordagem se mostrou defeituosa por conta da concorrência do acesso.

A questão é que o time base não faz utilização de *threads*, utilizando clientes completamente independentes entre si, em relação a processos, o que fazia da única forma

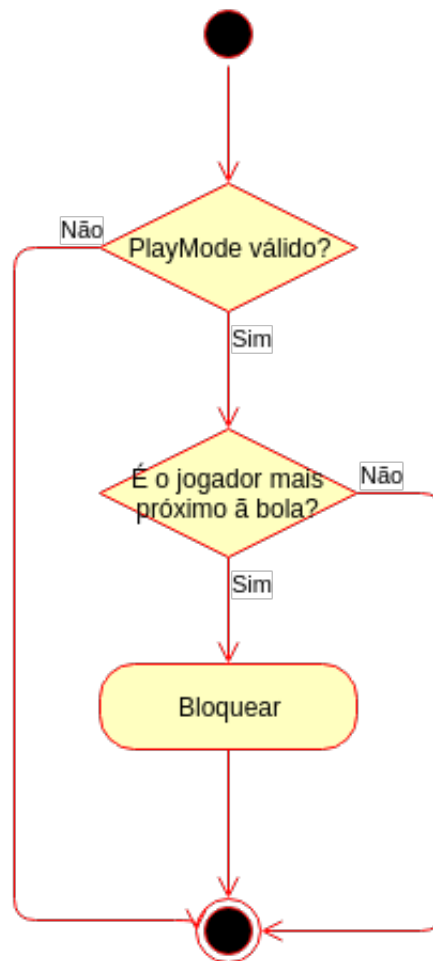
Figura 9 – Algoritmo da classe *BehaviorFormationPlanner*



Fonte: O Autor

de controlar o acesso ao arquivo a criação de exclusão de um arquivo de trava. O problema surgido desta abordagem é a velocidade de execução dos ciclos que acarretava na verificação por parte de mais de um agente no momento em que o arquivo de trava ainda não havia sido criado por nenhum outro, causando sobreposição da tabela sempre que um novo agente terminava de processar uma ação.

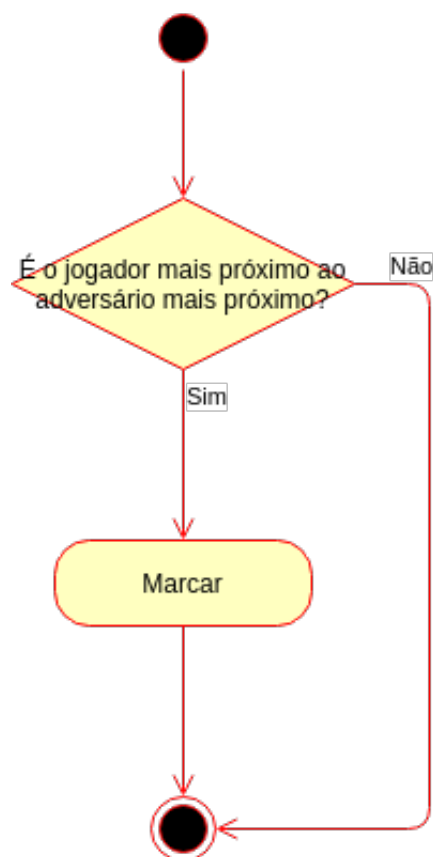
A solução deste problema foi a criação de uma tabela para cada jogador, isto resolve o problema da concorrência e trata cada jogador por sua função específica no jogo, mas torna o aprendizado potencialmente mais lento, devido a falta de compartilhamento de experiência. Outra questão desta abordagem é que obriga a execução do treinamento de todos os jogadores ao mesmo tempo.

Figura 10 – Algoritmo da classe *BehaviorBlockPlanner***Fonte:** O Autor

3.3 METODOLOGIA EXPERIMENTAL

Os modelos implementados foram treinados com o auxílio da sub tarefa HFO, mostrado na Figura 13, que consiste numa especialização do *RoboCup Simulated Soccer* onde é utilizado apenas metade do campo com alguns jogadores no ataque e alguns na defesa, foram experimentadas seções de treinamento utilizando 10 contra 10 jogadores. Com o HFO, é possível treinar os jogadores para defesa de forma mais objetiva focando apenas em situações de defesa. Seria possível a execução do treinamento utilizando partidas completas, no entanto, isso tornaria o treinamento mais lento, já que seriam encontradas muitas situações diferentes do alvo do trabalho, resultando em muito tempo na execução de uma partida para poucos episódios treinados.

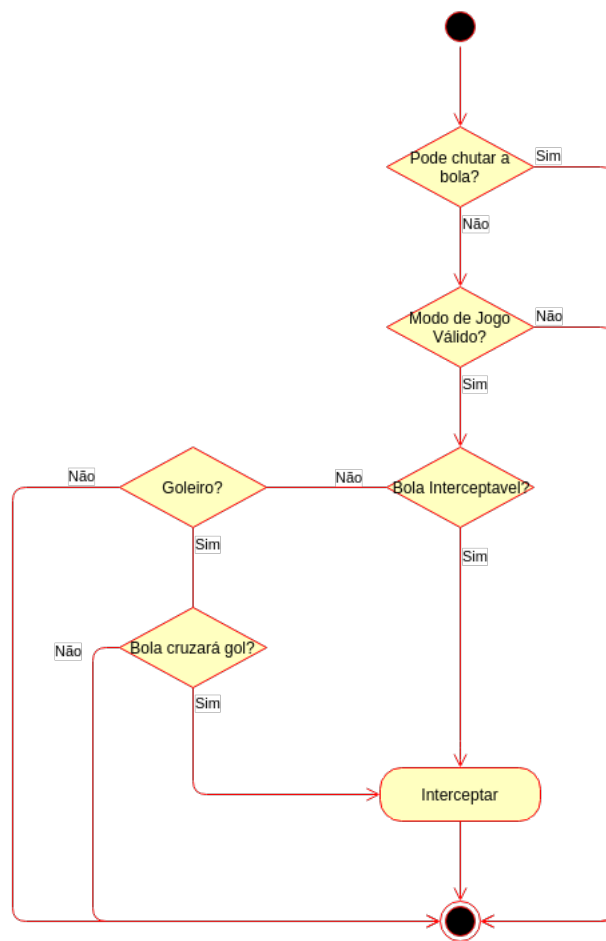
Devida necessidade de comparar vários modelos e o quão cada um evolue com o tempo foi utilizado o mecanismo de treinamento para também verificar o desempenho do modelo em execução. Para este fim foram feitas modificações no código-fonte com o

Figura 11 – Algoritmo da classe *BehaviorMarkPlanner*

Fonte: O Autor

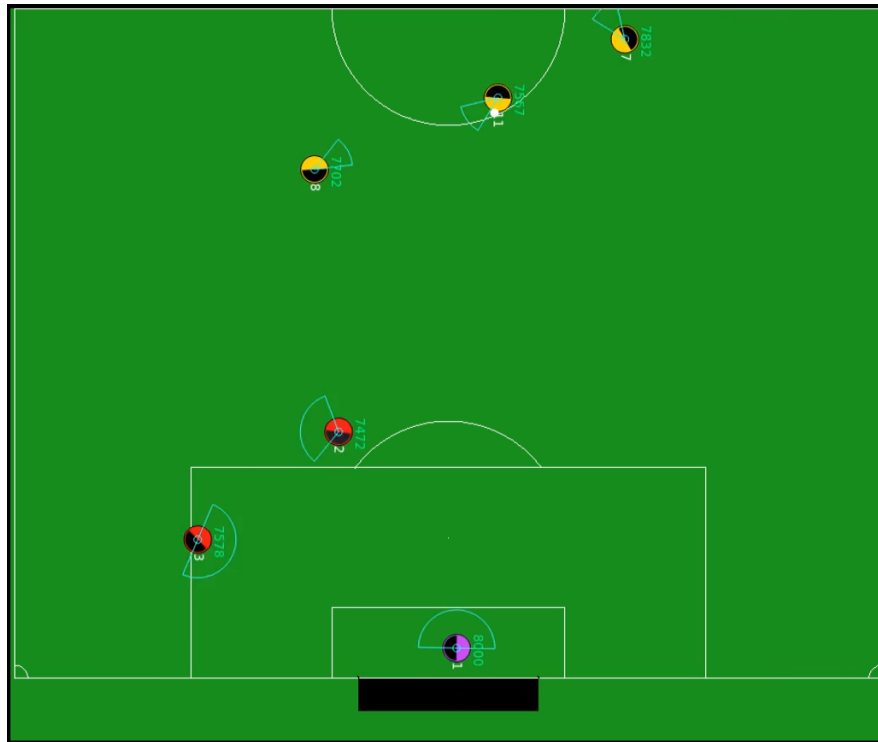
intuito de contabilizar o número de gols sofridos em intervalos de 3000 ciclos (metade da quantidade de ciclos de uma partida comum da categoria) e foram registradas as variações destes resultados para cada modelo, inclusive o time original sem a técnica empregada.

Figura 12 – Algoritmo da classe *BehaviorInterceptPlanner*



Fonte: O Autor

Figura 13 – HFO



Fonte: <<http://www.cs.utexas.edu/~AustinVilla/sim/halffieldoffense/>>

4 RESULTADOS

A partir daqui serão identificados os 4 modelos que demonstraram maior importância para o estudo:

1. O primeiro modelo testado, utilizando a primeira abordagem de estados (baseada apenas em variáveis) com o sistema de recompensas utilizando avanço da bola como punição e fila de 5 ações;
2. Modelo que utiliza o mesmo sistema de estados que o anterior, mas já abole a punição por avanço e diminui o tamanho da fila para 2 ações;
3. Este utiliza os mesmos estados e recompensas do anterior, porém utiliza uma fila de apenas uma ação (verificação imediata) e utiliza diretamente o mecanismo de ações ignorando a utilização das classes de comportamento mais especializadas;
4. O último modelo descrito é o que utiliza o sistema mais simples de papéis, além de utilizar-se da fila de 5 ações e o mesmo sistema de recompensas que os dois anteriores.

Após a execução de grande número de episódios para treinamento dos modelos propostos foram separados em planilhas as quantidades de gols resultantes para cada bloco de 3000 ciclos para cada um. Em cima desse registro foi feita uma média geral, uma média dos primeiros 100 registros e uma média de apenas os últimos 100 registros, levando em conta que se espera que tenha havido uma melhora na defesa de cada um.

Na Tabela 3 estão listados os modelos e as médias calculadas e por fim o valor para comparação do time original. Uma comparação visual é ilustrada pela Figura 14.

Modelo	Média Geral	Média dos primeiros 100 blocos	Média dos últimos 100 blocos
Modelo 1	44,08	44,09	44,11
Modelo 2	40,40	40,21	35,75
Modelo 3	39,13	44,1	37,85
Modelo 4	44,68	44,4	44,78
Original	26,45		

Tabela 3 – Resultados

A observação da Tabela 3 faz com que seja possível perceber que os modelos com melhor evolução, ou alguma evolução, são os Modelos 2 e 3. Por conta desta observação foram realizados um número maior de execuções nos mesmos para que fossem traçados gráficos (Figura 15) cruzando o número de blocos executados no teste e o número de gol

Figura 14 – Médias finais em comparação ao original

Fonte: O Autor

sofridos no último bloco juntamente com a curva de tendência linear formada por este cruzamento de ambos modelos para que se possa ser feito a visualização do melhoramento dos mesmos.

Os gráficos foram traçados com auxílio da ferramenta de planilhas Google Sheets¹ que permitiu a captura da equação da curva de tendência traçada. O Modelo 2 é representado pela equação 4.1 e o Modelo 3 é representado pela Equação 4.2, onde G representa a média de gols sofridos por bloco e b o número de blocos executados.

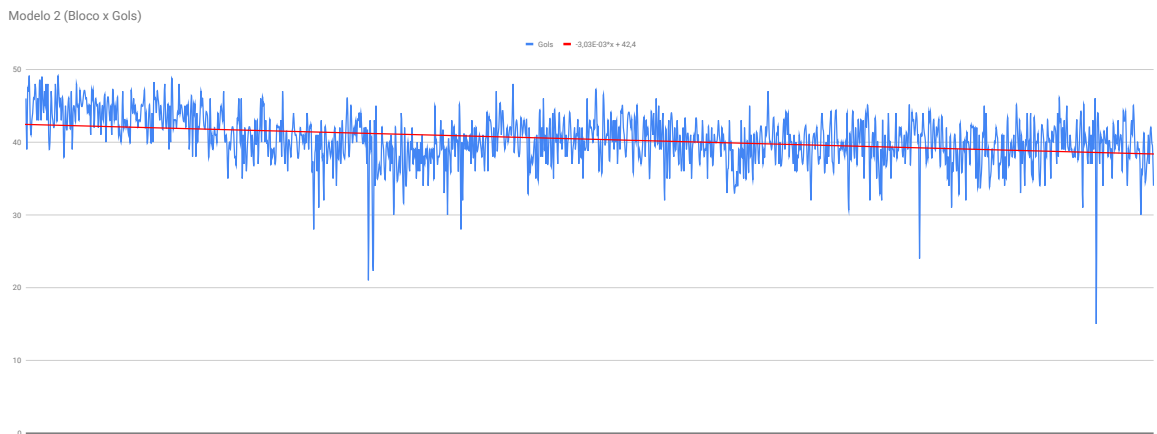
$$G = -3,03 \times 10^{-3} \times b + 42,4 \quad (4.1)$$

$$G = -3,67 \times 10^{-3} \times b + 42,1 \quad (4.2)$$

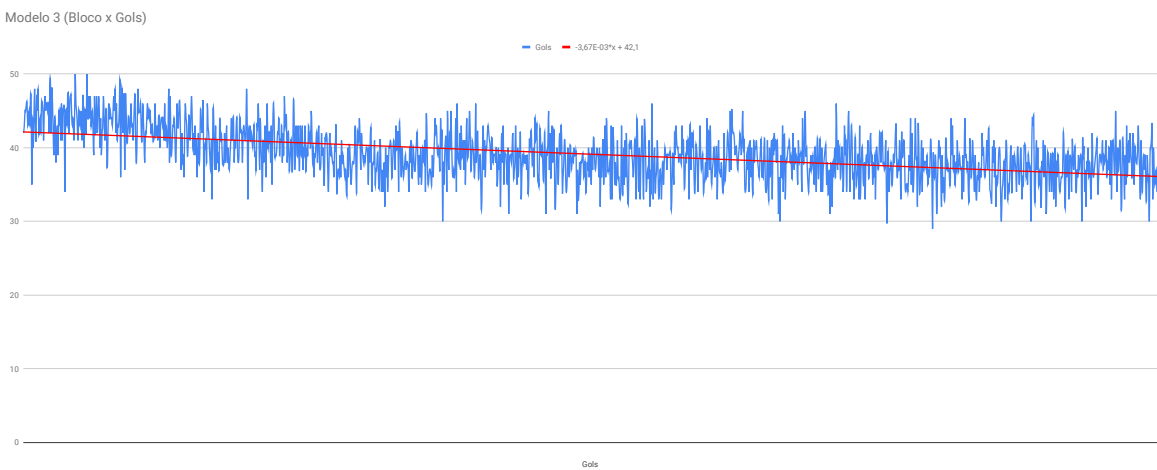
¹ <https://docs.google.com/spreadsheets>

Figura 15 – Gráficos de Modelos 2 e 3

(a) Modelo 2



(b) Model 3



Fonte: O Autor

5 CONCLUSÃO

Os resultados encontrados através da execução dos experimentos são longe de ideais, visto que não foi alcançada uma taxa de gols que conseguisse ser menor que a do time original, porém, é possível tomar notas sobre o processo que pode ter levado a tais resultados e inferir possíveis ações que podem levar ao resultado desejado.

Uma verdade na execução deste trabalho é de que, pela sua natureza, foi necessário a execução de treinamentos extensos para vários modelos direntes. Isto por si só não consistiria em problema, se não fosse uma pequena falha na projeção do time base que impedia a plena compatibilidade com o HFO.

A falha em questão corresponde a um problema de memória e acesso à ponteiros que causa a interrupção dos clientes durante as seções de treinamento, a correção desta falha não foi possível por conta da complexidade do código-fonte do time base desenvolvido por anos por componentes da equipe *WrightEagle*. Isto somado à necessidade de iniciar as seções manualmente, o que impossibilitava a escrita de um *script* que reconhecesse a queda dos clientes e o executassem novamente, impediu a execução de seções de treinamento longas e ininterruptas.

Outro prolema relacionado à execução dos treinamentos é quantidade de recursos utilizadas pelo treinamento, já que o HFO utiliza o máximo que consegue dos recursos para executar ciclos rápidos agilizando o treinamento. O consumo em questão impossibilitou a execução de treinamento de mais modelos simultaneamente.

Tudo isso somado a quantidade de variáveis que precisavam ser testadas para que se pudesse obter uma otimização do modelo e da implementação do mesmo impediu o avanço do projeto em tempo ábil.

Mas ainda assim, é possível observar que os gráficos dos resultados dos Modelos 2 e 2 apresentam sinais de evolução na inteligência dos agentes quanto a tentativa de evitar gols.

Levando em conta todos os modelos testados e os valores obtidos, é possível comparar e observar que a tentativa de avaliar a ação tomada após alguns ciclos de sua execução não apresentou os frutos esperados, pelo contrários, os dois modelos em destaque foram aqueles com menor número de ações na fila. Também é possível observar que a abordagem de estados por variáveis se mostrou mais eficiente que aquela por papéis situacionais.

Estes resultados, embora não o que se esperava ao se iniciar o projeto, são útei para execução de novas pesquisas na área, principalmente dando continuidade a esta pesquisa e

ao projeto FutVasf2D.

5.1 TRABALHOS FUTUROS

O resultado deste trabalho serve para apontar novos caminhos a serem tomados para melhoria de mecanismos de defesa, como o refino dos modelos aqui indicados ou a especialização em subtarefas da defesa. Vale a pena focar nos modelos com melhores resultados deste trabalhos a fim de reproduzi-los e melhora-los.

Também se faz interessante a realização de pesquisas de aplicação do *Q-Learning* em outros tipos de mecanismos, como no drible ou passe de bola.

Uma outra possibilidade de pesquisa é a de aplicação de outras técnicas de aprendizagem de máquina para o mecanismo aqui pesquisado como fim de comparação.

REFERÊNCIAS

- ADEODATO, P. J. et al. The role of temporal feature extraction and bagging of mlp neural networks for solving the wcci 2008 ford classification challenge. *IEEE*, 2009. Citado na página 28.
- ADEODATO, P. J. et al. The power of sampling and stacking for the pakdd-2007 cross-selling problem. **International Journal of Data Warehousing and Mining (IJDWM)**, IGI Global, v. 4, n. 2, p. 22–31, 2008. Citado na página 28.
- BIANCHI, R. A.; MANTARAS, R. L. de. Case-based multiagent reinforcement learning: Cases as heuristics for selection of actions. In: **ECAI**. [S.l.: s.n.], 2010. p. 355–360. Citado 3 vezes nas páginas 19, 20 e 27.
- CARVALHO, A.; OLIVEIRA, R. Reinforcement learning for the soccer dribbling task. In: *IEEE. Computational Intelligence and Games (CIG), 2011 IEEE Conference on*. [S.l.], 2011. p. 95–101. Citado na página 18.
- CELIBERTO, L. A. et al. Heuristic reinforcement learning applied to robocup simulation agents. In: SPRINGER. **Robot Soccer World Cup**. [S.l.], 2007. p. 220–227. Citado 2 vezes nas páginas 19 e 27.
- CHAN, G.; ASGARPOOR, S. Optimum maintenance policy with markov processes. **Electric power systems research**, Elsevier, v. 76, n. 6-7, p. 452–456, 2006. Citado na página 22.
- FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. Advances in knowledge discovery and data mining. In: FAYYAD, U. M. et al. (Ed.). Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996. cap. From Data Mining to Knowledge Discovery: An Overview, p. 1–34. ISBN 0-262-56097-6. Disponível em: <<http://dl.acm.org/citation.cfm?id=257938.257942>>. Citado 2 vezes nas páginas 27 e 28.
- FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: SPRINGER. **International Workshop on Agent Theories, Architectures, and Languages**. [S.l.], 1996. p. 21–35. Citado na página 12.
- GABEL, T.; RIEDMILLER, M.; TROST, F. A case study on improving defense behavior in soccer simulation 2d: The neurohassle approach. In: SPRINGER. **Robot Soccer World Cup**. [S.l.], 2008. p. 61–72. Citado 3 vezes nas páginas 13, 17 e 27.
- GAO, X. et al. Wavelet-based contourlet in quality evaluation of digital images. **Neurocomputing**, Elsevier, v. 72, n. 1-3, p. 378–385, 2008. Citado na página 28.
- HASTIE, T.; FRIEDMAN, J.; TIBSHIRANI, R. **The elements of statistical learning**. 2. ed. [S.l.]: Springer series in statistics New York, NY, USA:, 2009. Citado na página 28.
- HAYES-ROTH, B. An architecture for adaptive intelligent systems. **Artificial Intelligence**, Elsevier, v. 72, n. 1-2, p. 329–365, 1995. Citado na página 12.

HENN, T.; HENRIO, J.; NAKASHIMA, T. Optimizing player's formations for corner-kick situations in robocup soccer 2d simulation. **Artificial Life and Robotics**, Springer, v. 22, n. 3, p. 296–300, 2017. Citado na página 11.

HOMEM, T. P. D. et al. Improving reinforcement learning results with qualitative spatial representation. In: IEEE. **Intelligent Systems (BRACIS), 2017 Brazilian Conference on**. [S.l.], 2017. p. 151–156. Citado 2 vezes nas páginas 19 e 27.

KITANO, H. et al. Robocup: A challenge problem for ai and robotics. In: SPRINGER. **Robot Soccer World Cup**. [S.l.], 1997. p. 1–19. Citado na página 11.

LIU, L.; LI, L. Regional cooperative multi-agent q-learning based on potential field. In: IEEE. **Natural Computation, 2008. ICNC'08. Fourth International Conference on**. [S.l.], 2008. v. 6, p. 535–539. Citado na página 18.

LIU, L.; LI, L. Sparse cooperative multi-agent q-learning based on vector potential field. In: IEEE. **Global Congress on Intelligent Systems**. [S.l.], 2009. p. 99–103. Citado na página 18.

NASCIMENTO, J. P. F. **joaopedrofn/learnToWalk**. 2018. Disponível em: <<https://github.com/joaopedrofn/learnToWalk>>. Citado na página 24.

NERI, J. R. F. et al. Team description paper gpr-2d 2012. 2012. Citado na página 18.

NERI, J. R. F. et al. A proposal of qlearning to control the attack of a 2d robot soccer simulation team. In: IEEE. **Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian**. [S.l.], 2012. p. 174–178. Citado na página 17.

OLIVEIRA, R. et al. A data mining approach to solve the goal scoring problem. In: IEEE. **Neural Networks, 2009. IJCNN 2009. International Joint Conference on**. [S.l.], 2009. p. 2347–2352. Citado na página 13.

POGGI, A.; ADORNI, G. A multi language environment to develop multi agent applications. In: SPRINGER. **International Workshop on Agent Theories, Architectures, and Languages**. [S.l.], 1996. p. 325–339. Citado na página 12.

RIEDMILLER, M.; GABEL, T. Brainstormers 2 d team description 2007. In: . [S.l.: s.n.], 2005. Citado na página 17.

ROBOCUP. **RoboCupSoccer - Simulation 2D**. 2019. Disponível em: <<https://www.robocup.org/leagues/24>>. Citado na página 12.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. [S.l.]: Malaysia; Pearson Education Limited,, 2016. Citado 4 vezes nas páginas 11, 20, 21 e 22.

SCHMILL, M. D.; OATES, T.; COHEN, P. R. Learning planning operators in real-world, partially observable environments. In: **AIPS**. [S.l.: s.n.], 2000. p. 246–253. Citado na página 12.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. **Understanding machine learning: From theory to algorithms**. [S.l.]: Cambridge university press, 2014. Citado na página 20.

SIMONINI, T. **Diving deeper into Reinforcement Learning with Q-Learning**. freeCodeCamp.org, 2018. Disponível em: <<https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe>>. Citado na página 23.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018. Citado 2 vezes nas páginas 21 e 22.

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992. Citado na página 23.

WITTEN, I. H. et al. **Data Mining: Practical machine learning tools and techniques**. [S.l.]: Morgan Kaufmann, 2016. Citado na página 28.

WOOLDRIDGE, M. Intelligent agents: The key concepts. In: SPRINGER. **ECCAI Advanced Course on Artificial Intelligence**. [S.l.], 2001. p. 3–43. Citado na página 12.

ZHAO, H.; SINHA, A. P.; GE, W. Effects of feature construction on classification performance: An empirical study in bank failure prediction. **Expert Systems with Applications**, Elsevier, v. 36, n. 2, p. 2633–2644, 2009. Citado na página 28.

APÊNDICE A – CÓDIGOS-FONTE

Código 2 – *Player* - Avaliação de Ação e Atualização de *Q-Table*

```

1  //QTable AREA
2  {
3      if (++mpAgent->generalCycleCounter >= 3000)
4      {
5          mpAgent->generalCycleCounter = 0;
6
7          std::cout << "Block of cycles " << ++mpAgent->countDone << ": "
            ↳ << mpAgent->goalCounter << " Goals" << std::endl;
8          mpAgent->goalCounter = 0;
9      }
10     std::stringstream qTableString;
11     qTableString << "qTable" << mpAgent->GetSelfUnum();
12     std::ifstream qTableFileIn(qTableString.str(), std::ios::binary);
13     double qTable[7][9];
14     qTableFileIn.read((char *)&qTable, sizeof(qTable));
15     qTableFileIn.close();
16
17     int curState = mpAgent->lastStateOccurred;
18     int actionToTake = mpAgent->lastActionTaken;
19     if (curState != -1)
20     {
21         double reward = 0;
22         int thatState = mpAgent->lastActionsState.size() ?
            ↳ mpAgent->lastActionsState[0] : -1;
23         int thatAction = mpAgent->lastActions.size() ?
            ↳ mpAgent->lastActions[0] : -1;
24         ServerPlayMode thatPM = mpAgent->lastActionsPM.size() ?
            ↳ mpAgent->lastActionsPM[0] : SPM_Null;
25         vector<double> actionSpace{qTable[curState][0],
            ↳ qTable[curState][1], qTable[curState][2],
            ↳ qTable[curState][3], qTable[curState][4],
            ↳ qTable[curState][5], qTable[curState][6],
            ↳ qTable[curState][7]};
26         int maxFromCurrent = greedySelection(actionSpace);

```

```

27     qTable[curState][8]++;
28     switch (mpObserver->GetServerPlayMode())
29     {
30     case SPM_Captured:
31         mpAgent->cycleCounter = 0;
32         mpAgent->lastActions.clear();
33         mpAgent->lastActionsState.clear();
34         mpAgent->lastActionsPM.clear();
35         reward = 20;
36         qTable[thatState][thatAction] =
            ↳ learn(qTable[thatState][thatAction], maxFromCurrent,
            ↳ reward);
37         break;
38     case SPM_OutOfBounds:
39         mpAgent->cycleCounter = 0;
40         mpAgent->lastActions.clear();
41         mpAgent->lastActionsState.clear();
42         mpAgent->lastActionsPM.clear();
43
44         reward = 10;
45         qTable[thatState][thatAction] =
            ↳ learn(qTable[thatState][thatAction], maxFromCurrent,
            ↳ reward);
46         break;
47     case SPM_Goal_Train:
48         mpAgent->goalCounter++;
49         mpAgent->cycleCounter = 0;
50         mpAgent->lastActions.clear();
51         mpAgent->lastActionsState.clear();
52         mpAgent->lastActionsPM.clear();
53         reward = -20;
54         qTable[thatState][thatAction] =
            ↳ learn(qTable[thatState][thatAction], maxFromCurrent,
            ↳ reward);
55         break;
56     case SPM_TimeOut:
57         mpAgent->cycleCounter = 0;
58         mpAgent->lastActions.clear();

```

```

59         mpAgent->lastActionsState.clear();
60         mpAgent->lastActionsPM.clear();
61         break;
62     default:
63         break;
64 }
65 char side = mpObserver->OurInitSide();
66 Vector goal(side == '1' ? 51.162 : -51.162, 0);
67 if (mpAgent->cycleCounter == 5)
68 {
69     mpAgent->cycleCounter = 0;
70     mpAgent->lastActions.erase(mpAgent->lastActions.begin());
71     mpAgent->lastActionsState
72         .erase(mpAgent->lastActionsState.begin());
73     mpAgent->lastActionsPM
74         .erase(mpAgent->lastActionsPM.begin());
75     switch (mpObserver->GetServerPlayMode())
76     {
77     case SPM_PlayOn_11:
78     case SPM_PlayOn_10:
79     case SPM_PlayOn_9:
80     case SPM_PlayOn_8:
81     case SPM_PlayOn_7:
82     case SPM_PlayOn_6:
83     case SPM_PlayOn_5:
84     case SPM_PlayOn_4:
85     case SPM_PlayOn_3:
86     case SPM_PlayOn_2:
87     case SPM_PlayOn_1:
88         if (mpAgent->lastBallPosition.Dist(goal) >
89             ⇨ mpAgent->lastGoalDist)
90         {
91             if (
92                 mpAgent->lastBallPosition
93                     .Dist(mpAgent->lastPosition) <
94                     ⇨ mpAgent->lastPlayerDist)
95                 reward = 5;
96             else

```

```

95         reward = 2;
96     }
97     break;
98     default:
99     break;
100 }
101
102     qTable[thatState][thatAction] =
103         ↪ learn(qTable[thatState][thatAction], maxFromCurrent,
104         ↪ reward);
105 }
106
107     mpAgent->lastGoalDist = mpAgent->lastBallPosition.Dist(goal);
108     mpAgent->lastPlayerDist =
109         ↪ mpAgent->lastBallPosition.Dist(mpAgent->lastPosition);
110
111     std::ofstream qTableFileOut(qTableString.str(),
112         ↪ std::ios::binary);
113     qTableFileOut.write((char *)&qTable, sizeof(qTable));
114     qTableFileOut.close();
115     mpAgent->cycleCounter++;
116 }
117 }
118 //END OF QTable AREA

```

Fonte: Os autores.

Código 3 – *BehaviorDefensePlanner* - Identificação de Estado e Escolha de Ação Para Model 4 (Mais Simples)

```

1  void BehaviorDefensePlanner::Plan(std::list<ActiveBehavior>
2  ↪ &behavior_list)
3  {
4      std::stringstream qTableString;
5      qTableString << "qTable" << mSelfState.GetUnum();
6      std::ifstream qTableFileIn(qTableString.str(), std::ios::binary);
7      double qTable[7][9];
8      qTableFileIn.read((char *)&qTable, sizeof(qTable));

```

```

8      qTableFileIn.close();
9      mAgent.lastPosition = mSelfState.GetPos();
10     mAgent.lastBallPosition = mWorldState.GetBall().GetPos();
11
12     //GETTING VARIABLES
13     double teammatesDistToOpp[5];
14     int index = 0;
15     double avgDist = 0;
16     for (int i = 1; i <= 11; i++)
17     {
18         if (i != mWorldState.GetTeammateGoalieUnum())
19         {
20             double aux = mPositionInfo.GetBallDistToTeammate(i);
21             teammatesDistToOpp[index++] = aux;
22         }
23     }
24     int n = sizeof(teammatesDistToOpp) / sizeof(teammatesDistToOpp[0]);
25     std::sort(teammatesDistToOpp, teammatesDistToOpp + (n));
26     avgDist = teammatesDistToOpp[3] + teammatesDistToOpp[4] +
27     ↪ teammatesDistToOpp[5] + teammatesDistToOpp[6] +
28     ↪ teammatesDistToOpp[7] + teammatesDistToOpp[8] +
29     ↪ teammatesDistToOpp[9];
30     avgDist /= 7;
31     double sum = 0;
32     for (int i = 3; i < 10; i++)
33     {
34         sum += (teammatesDistToOpp[i] - avgDist) *
35         ↪ (teammatesDistToOpp[i] - avgDist);
36     }
37     double density = sqrt(sum / 7);
38     //END OF VARIABLES
39
40     int curState;
41     if (teammatesDistToOpp[0] ==
42     ↪ mPositionInfo.GetBallDistToTeammate(mSelfState.GetUnum()))
43         if (teammatesDistToOpp[0] <= 5)
44             curState = FirstDefenderRealClose;
45         else

```

```

41         curState = FirstDefender;
42     else if (teammatesDistToOpp[1] ==
    ↪ mPositionInfo.GetBallDistToTeammate(mSelfState.GetUnum()))
43         curState = SecondDefender;
44     else if (teammatesDistToOpp[2] ==
    ↪ mPositionInfo.GetBallDistToTeammate(mSelfState.GetUnum()))
45         curState = ThirdDefender;
46     else if (density < 7)
47         curState = Together;
48     else if (density < 14)
49         curState = MidRanged;
50     else
51         curState = Away;
52     mAgent.lastStateOccurred = curState;
53     Vector ballPosition = mWorldState.GetBall().GetPos();
54     vector<double> actionSpace{qTable[curState][0], qTable[curState][1],
    ↪ qTable[curState][2], qTable[curState][3], qTable[curState][4],
    ↪ qTable[curState][5], qTable[curState][6], qTable[curState][7]};
55     double epsilon = (1 - (qTable[curState][8] / 50000));
56     int actionToTake = greedyEpSelection(actionSpace, epsilon > 0 ?
    ↪ epsilon : 0.1);
57     mAgent.lastActionTaken = actionToTake;
58     double power = mSelfState.CorrectDashPowerForStamina(
59         ServerParam::instance().maxDashPower());
60     switch (actionToTake)
61     {
62     case MoveNorth:
63         if (mSelfState.GetPos().Y() + 2 <= 25)
64             Dasher::instance().GoToPoint(mAgent,
    ↪ Vector(mSelfState.GetPos().X(), mSelfState.GetPos().Y() +
    ↪ 10), 1.0, power, false, true);
65         break;
66     case MoveSouth:
67         if (mSelfState.GetPos().Y() - 2 >= -25)
68             Dasher::instance().GoToPoint(mAgent,
    ↪ Vector(mSelfState.GetPos().X(), mSelfState.GetPos().Y() -
    ↪ 10), 1.0, power, false, true);
69         break;

```

```

70     case MoveWest:
71         if (mSelfState.GetPos().X() - 2 >= -51)
72             Dasher::instance().GoToPoint(mAgent,
73                 ↪ Vector(mSelfState.GetPos().X() - 10,
74                 ↪ mSelfState.GetPos().Y()), 1.0, power, false, true);
75         break;
76     case MoveEast:
77         if (mSelfState.GetPos().X() + 2 >= 51)
78             Dasher::instance().GoToPoint(mAgent,
79                 ↪ Vector(mSelfState.GetPos().X() + 10,
80                 ↪ mSelfState.GetPos().Y()), 1.0, power, false, true);
81         break;
82     case MoveToBall:
83         Dasher::instance().GoToPoint(mAgent, ballPosition, 1.0, power,
84             ↪ false, true);
85         break;
86     case InterceptAction:
87         BehaviorInterceptPlanner(mAgent).Plan(behavior_list);
88         break;
89     case BlockAction:
90         BehaviorBlockPlanner(mAgent).Plan(behavior_list);
91         break;
92     case MarkAction:
93         BehaviorMarkPlanner(mAgent).Plan(behavior_list);
94         break;
95     case StayStill:
96     default:
97         break;
98 }
99
100 mAgent.lastActions.push_back(actionToTake);
101 mAgent.lastActionsState.push_back(curState);
102 PlayMode pm = mWorldState.GetPlayMode();
103 ServerPlayMode spm = SPM_Null;
104 switch (pm)
105 {
106     case PM_Goal_Opps:
107         spm = SPM_Goal_Train;

```



```
103         break;
104     case PM_Captured:
105         spm = SPM_Captured;
106         break;
107     case PM_OutOfBounds:
108         spm = SPM_OutOfBounds;
109         break;
110     case PM_Play_On_11:
111         spm = SPM_PlayOn_11;
112         break;
113     case PM_Play_On_10:
114         spm = SPM_PlayOn_1;
115         break;
116     case PM_Play_On_9:
117         spm = SPM_PlayOn_9;
118         break;
119     case PM_Play_On_8:
120         spm = SPM_PlayOn_8;
121         break;
122     case PM_Play_On_7:
123         spm = SPM_PlayOn_7;
124         break;
125     case PM_Play_On_6:
126         spm = SPM_PlayOn_6;
127         break;
128     case PM_Play_On_5:
129         spm = SPM_PlayOn_5;
130         break;
131     case PM_Play_On_4:
132         spm = SPM_PlayOn_4;
133         break;
134     case PM_Play_On_3:
135         spm = SPM_PlayOn_3;
136         break;
137     case PM_Play_On_2:
138         spm = SPM_PlayOn_2;
139         break;
140     case PM_Play_On_1:
```

```
141         spm = SPM_PlayOn_1;
142         break;
143     default:
144         break;
145     }
146     mAgent.lastActionsPM.push_back(spm);
147 }
```

Fonte: Os autores.