UNIVERSIDADE FEDERAL DE ALFENAS

LUCAS DA SILVA JUSTINO
RODRIGO SILVA DE PAULO

IMPLEMENTAÇÃO DE MECANISMOS BÁSICOS DE FUTEBOL PARA UM TIME DE ROBOCUP SOCCER SIMULATION

ALFENAS/MG

2023

LUCAS DA SILVA JUSTINO
RODRIGO SILVA DE PAULO

IMPLEMENTAÇÃO DE MECANISMOS BÁSICOS DE FUTEBOL PARA UM TIME DE ROBOCUP SOCCER SIMULATION

Monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, pela Universidade Federal de Alfenas.
Área de concentração: Ciências Exatas.

Orientador: Prof. Dr. Luiz Eduardo da Silva

ALFENAS/MG
2023

RESUMO

O futebol de robôs consiste em uma coleção de robôs autônomos capazes de reconhecer o ambiente em que estão inseridos e seus pontos de referência, o campo de futebol e seus elementos, como a bola, o gol e os jogadores. Sendo uma das principais referências da área, a *Robocup Soccer Simulation 2D* é uma competição que faz parte da *Robocup*, uma competição que promove o avanço de pesquisas na área de robótica e inteligência artificial. Na modalidade 2D, os times de robôs são apresentados por agentes virtuais em um ambiente simulado. Este estudo se concentra na otimização da equipe de futebol de robôs *UFABC-TrillearnBase*, inspirada no *UvA-Trillearn* da Universidade de Amsterdã. O objetivo principal é aprimorar o desempenho da equipe durante as partidas, introduzindo mecanismos de passe, drible e cruzamento, levando em consideração a posição dos jogadores em campo para uma tomada de decisão mais estratégica. Isso possibilita estabelecer uma base competitiva sólida para a Universidade Federal de Alfenas em torneios futuros. Os resultados evidenciam melhorias significativas e um desempenho notável em comparação à equipe base, destacando a clara superioridade do time modificado, que saiu vitorioso em todos os testes realizados, alcançando o objetivo do trabalho. As técnicas implementadas oferecem maior controle sobre a posse de bola e ampliam as opções de jogadas ofensivas. Como sugestão para trabalhos futuros, recomenda-se explorar o uso de técnicas de aprendizado de máquina e aprendizado por reforço para alcançar melhorias adicionais.

Palavras-chave: *UFABC-TrillearnBase; UvA-Trillearn; Robocup Soccer Simulation 2D*, Futebol de Robôs Simulado.

ABSTRACT

Robotic soccer involves a collection of autonomous robots capable of recognizing their environment and its references, such as the soccer field and its elements like the ball, goal, and players. As one of the primary references in this field, RoboCup Soccer Simulation 2D is a competition within RoboCup that drives advancements in robotics and artificial intelligence research. In the 2D category, robot teams are represented by virtual agents in a simulated environment. This study focuses on optimizing the UFABC-TrillearnBase robot soccer team, inspired by the UvA-Trillearn from the University of Amsterdam. The main goal is to enhance the team's performance during matches by introducing mechanisms for passing, dribbling, and crossing, considering players' positions on the field for more strategic decision-making. This aims to establish a solid competitive foundation for the Federal University of Alfenas in future tournaments. The results demonstrate significant improvements and notable performance compared to the base team, emphasizing the clear superiority of the modified team, which emerged victorious in all conducted tests, meeting the goal of the study. The implemented techniques provide better ball control and broaden offensive play options. As a suggestion for future work, it is recommended to explore the use of machine learning and reinforcement learning techniques to achieve additional improvements.

Keywords: *UFABC-TrillearnBase; UvA-Trillearn; Robocup Soccer Simulation 2D, Simulated Robot Soccer.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FIFA	Fédération Internationale de Football Association
UFABC	Universidade Federal do ABC
UNIFAL-MG	Universidade Federal de Alfenas
UVA	Universiteit van Amsterdam

SUMÁRIO

1. INTRODUÇÃO

O futebol, ao longo dos séculos, consolidou-se como um esporte de relevância global, transcendendo fronteiras culturais e temporais. Embora suas origens sejam datadas do século XIX, sabe-se que a humanidade praticava esportes com características semelhantes há milhares de anos com vestígios de atividades esportivas análogas referindo-se à China de 3000 a.C. (Maior, 2008).

No futebol moderno, dois times de 11 jogadores cada se enfrentam e um árbitro se ocupa da correta aplicação das normas do esporte. Segundo dados da FIFA, cerca de 270 milhões de pessoas participam de suas várias competições, tornando-o o esporte mais popular do mundo (FIFA, 2023). Devido à sua grande popularidade em todo o mundo, tornou-se um modelo para incentivar o desenvolvimento e a pesquisa em diversas áreas de estudo, incluindo a simulação de tomada de decisão em tempo real em ambientes suscetíveis a constantes alterações provocadas por agentes externos.

As ferramentas de simulação, têm se mostrado um excelente meio para a aquisição rápida de conhecimento, evitando a necessidade de criar equipamentos físicos que demandam altos investimentos e longos períodos de desenvolvimento. Assim, a área da simulação demonstra-se um meio eficaz e ágil para a condução de estudos em vários campos de pesquisa, (How Simulation Tools are Transforming Education and Training, 2018).

Com base nesses conceitos, surgiu a competição *RoboCup Soccer Simulation 2D*, a competição mais antiga das ligas de *RoboCup Soccer* (Santos, 2023). Esta competição engloba uma série de torneios nos quais partidas de futebol são simuladas por meio de um computador. Nesse contexto, não há presença de robôs físicos; em vez disso, é criado um ambiente simulado onde os espectadores acompanham as partidas através de uma interface que exibe as ações e informações dos “jogadores”.

Na *RoboCup Soccer Simulation 2D*, duas equipes com 11 jogadores, denominados “agentes”, disputam partidas de futebol em um campo virtual bidimensional. A simulação é feita em um servidor em que cada jogador recebe informações de maneira individual e deve tomar decisões básicas, como correr, chutar, tocar a bola, entre outras, (ROBOCUP, 2023).

O objetivo principal deste trabalho é aprimorar e criar estratégias essenciais para uma equipe, tomando como base um time da *UFABC Robotics* que se inspirou no competitivo *UvA-Trillearn* da Universidade de Amsterdã, o *UFABC-TrillearnBase*. Buscou-se desenvolver mecanismos fundamentais que levem a um desempenho superior ao do time

original, podendo assim servir como base para uma equipe capaz de representar a Universidade Federal de Alfenas em torneios competitivos.

Esta monografia segue uma estrutura detalhada para apresentar os diversos aspectos do estudo realizado. No capítulo 1, a Introdução oferece uma contextualização do problema, apresenta os objetivos da pesquisa e expõe a organização do trabalho. Na sequência, no Capítulo 2 é apresentada a revisão bibliográfica, abrange os principais conceitos, teorias e revisão crítica de artigos e fontes relevantes, analisando sua contribuição para o desenvolvimento deste estudo. No capítulo 3, Objetivos, concentra-se o foco central deste trabalho, apresentando de maneira específica os propósitos e metas estabelecidos. O Capítulo 4, Metodologia, detalha a abordagem de pesquisa adotada, descrevendo os procedimentos metodológicos, a lógica desenvolvida, os métodos aplicados e as ferramentas utilizadas para a condução do estudo. Os resultados obtidos são apresentados no Capítulo 5, exibidos através de uma análise, representada por meio da tabela e gráfico comparativo entre o desempenho do time modificado em relação ao time original. O Capítulo 6, Conclusão, realiza a síntese dos resultados comparados aos objetivos propostos, fornecendo considerações finais com base nos dados coletados. Adicionalmente, são oferecidas sugestões para futuras pesquisas, visando a expansão e aprofundamento deste estudo.

2. REVISÃO BIBLIOGRÁFICA

O trabalho de Gomes et al. (GOMES, 2018), *Spartron: Soccer Simulation 2D Team Description Paper*, apresenta as atividades realizadas pela equipe para a participação da Competição Brasileira de *Soccer Simulation 2D*. Nesse projeto, são apresentadas algumas informações sobre o funcionamento dos jogos, uma breve introdução da ferramenta *Fedit*, usada com o intuito de desenvolver e editar as formações táticas do time, e uma breve descrição do algoritmo que é usado para o controle da "*stamina*" (energia) de cada jogador, destacando o problema de cansaço elevado nos últimos 1000 ciclos de cada tempo da partida.

Embora os conhecimentos sobre a ferramenta *Fedit* sejam interessantes, não a incorporamos neste trabalho porque mantivemos a formação inicial já implementada no time base. Como a equipe não é totalmente competitiva, ela não leva os jogadores ao máximo. Como resultado, nos últimos 1000 ciclos de cada tempo da partida, não há um alto nível de fadiga, eliminando a necessidade de implementar o algoritmo mencionado por Gomes et al. (2018).

O trabalho *HELIOS2018: RoboCup 2018 Soccer Simulation 2D League Champion,* desenvolvido por Akiyama et al.(Akiyama et al., 2019), apresenta uma das formas de modelar o comportamento cooperativo entre os agentes é o planejamento de sequências de ações usando métodos de busca em árvore. O time apresenta um método de busca online de comportamento cooperativo, que gera e avalia sequências de ações de chute entre vários jogadores da equipe. O método utiliza uma função de avaliação que representa a qualidade de cada ação candidata, e seleciona a melhor sequência de ações baseada no valor de avaliação. O time também utiliza uma análise do adversário para adaptar a estratégia da equipe ao time oponente online.

Embora as técnicas de modelagem do comportamento cooperativo e busca em árvore sejam apresentadas como excelentes meios de tomada de decisão, foi decidido por não utilizar essas técnicas avançadas de desenvolvimento de estratégias propostas, pois a abordagem para este trabalho se concentra na implementação de ações de jogo mais diretas e simples.

O trabalho HELIOS 201*9: Team Description Paper*, desenvolvido por Akiyama et al. (AKIYAMA, 2018), relata uma proposta de implementação de dois métodos importantes para o desenvolvimento: um modelo de formação usando triangulação e uma estrutura de planejamento de sequência de ação. Eles propõem utilizar um modelo que identifica qualquer relação entre os jogadores, e para isso, utilizam o *Word2Vec*, que traduz as ações

e jogadores para um vetor. Após isso, o método extrai as ações contidas nos arquivos log gerados pelo simulador, obtém os vetores de ações e de jogadores gerados pelo *Word2Vec* e realizam o agrupamento destes valores, criando uma matriz de distância para a equipe.

A principal contribuição deste trabalho ao desenvolvimento deste projeto, seria a forma como eles utilizam esse modelo para a classificação de grupos de jogadores, podendo separá-los por ações, o que facilitaria a movimentação deles durante a partida, buscando obter o melhor resultado. Por não aplicarmos métodos de aprendizado de máquina, não utilizamos a classificação dos jogadores como meio de determinar suas ações durante a partida.

O trabalho escrito por Tanilson Santos (SANTOS, 2011), sobre Jogadas Coletivas por meio de Comunicação Multi-Agente para a equipe iBots da Categoria de Simulação 2D da RoboCup, se concentra na dinâmica da comunicação e interação entre os agentes que compõem as equipes de futebol de robôs na categoria de simulação 2D da competição RoboCup. Explora a maneira como ocorre a troca de informações entre jogadores e técnicos, detalhando o processo de transmissão e recepção de mensagens nesse ambiente simulado. O estudo também estabelece um paralelo com o jogo real ao analisar a troca de passes. Ele destaca que essa comunicação muitas vezes acontece de forma indireta, o que pode resultar no passe sendo feito para uma posição anterior do jogador, não necessariamente refletindo sua posição atual.

O trabalho aborda de maneira detalhada a comunicação entre os jogadores, destacando a importância de uma abordagem mais direta na implementação. No entanto, ao desenvolver a estratégia de passe, foi adotada uma função que calcula automaticamente a posição do jogador, permitindo que o passe alcance o jogador escolhido sem a necessidade de uma comunicação mais direta ser implementada.

O trabalho conduzido por Costa et al. (Costa, 2016) sobre o *Asimov Soccer Simulation 2D* destaca a fundação e as estratégias adotadas por esse time. O trabalho aponta o uso do time base *Agent2D*, que já possuía funcionalidades essenciais, como ponto de partida. Além disso, explora as estratégias para aprimorar o desempenho do time, incluindo o uso do algoritmo Q-*Learning* na fase ofensiva, a otimização da *stamina* dos jogadores (ou seja, a gestão de energia) e a seleção dos tipos de jogadores com base em suas características específicas descritas pelo *Agent2D*.

Apesar do método proposto neste trabalho oferecer uma estratégia para criar jogadas ofensivas, optamos por não adotá-la devido à complexidade resultante dos diversos contextos de jogo, o que dificultaria a aplicação do algoritmo *Q-Learning* de maneira efetiva. Além disso, não recorremos à base do *Agent2D* devido à indisponibilidade de sua

referência. Quanto à otimização da *stamina*, essa etapa foi dispensada, uma vez que o time selecionado por nós não apresentava uma queda significativa no desempenho relacionada à energia dos jogadores. Como não utilizamos o *Agent2D* como base, a classificação dos jogadores não se deu com base em suas características específicas.

O trabalho de Silva et al. (SILVA, 2019), sobre um Ambiente Virtual para Simulação de Futebol de Robôs Aplicado ao Estudo de Programação, Sistemas Inteligentes e Robótica, tem como objetivo principal introduzir ideias básicas da programação orientada a objetos para definir alguns conceitos como a passagem de parâmetros, ponteiros, recursividade, objetos, classes e heranças.

Os passos seguidos no trabalho são claros, principalmente a configuração de um ambiente capaz de executar as simulações que é disponibilizado através de uma máquina virtual com as dependências do *Soccer Simulator* já instaladas. Porém, ao instalar a máquina virtual disponibilizada e executar o time do UFABC-Trillearn, nos deparamos com a falta de algumas bibliotecas essenciais, exigindo uma configuração manual do ambiente.

Para alcançar a configuração adequada do ambiente de execução e do servidor, foram realizadas diversas pesquisas em artigos e sites. Infelizmente, muitas das dependências estavam desatualizadas ou simplesmente não estavam mais disponíveis. Diante dessa dificuldade e da escassez de documentação, elaboramos um passo a passo detalhado, que pode ser consultado no Apêndice A, abrangendo toda a configuração do ambiente.

3. OBJETIVO

O objetivo deste trabalho é otimizar o desempenho de um time base de futebol de robôs na competição *Robocup Soccer Simulation*, por meio da implementação de mecanismos essenciais de jogo. A meta central é alcançar um desempenho significativamente aprimorado em comparação ao estado inicial do time.

O sucesso na obtenção de um desempenho superior e no refinamento das habilidades propostas neste estudo tem o potencial de posicionar a UNIFAL-MG como uma entidade competitiva e promissora na pesquisa em futebol de robôs.

Sendo assim, iniciar o processo de construção de uma base sólida e um time habilidoso, preparado para representar a universidade e participar de campeonatos e contribuir com o envolvimento da instituição na inovação e no avanço tecnológico.

4. METODOLOGIA

Este estudo sobre as estratégias implementadas no contexto da competição de futebol de robôs simulados 2D da *RoboCup*, adotou uma abordagem metodológica composta por etapas distintas, dividida nas seguintes etapas:

1. A configuração do ambiente;
2. Avaliação do time base;
3. Implementação das estratégias.

4.1 AMBIENTE DE JOGO

4.1.1 PARTIDA

A partida de futebol na *RoboCup Soccer Simulation 2D* é disputada por dois times de robôs controlados por agentes virtuais em um ambiente bidimensional simulado. Cada equipe é composta por 11 jogadores, representados por agentes autônomos que interagem com o ambiente virtual, onde há um campo, uma bola e os elementos do jogo, como gols e jogadores.

Durante o jogo, os agentes controlam os jogadores, tomando decisões com base nas informações recebidas do ambiente simulado. Eles têm a capacidade de reconhecer a posição da bola, dos outros jogadores e do campo, permitindo-lhes realizar ações como se movimentar, chutar, passar a bola e tentar marcar gols.

As partidas seguem regras similares às de um jogo oficial de futebol. Elas incluem elementos como impedimento, faltas diretas e indiretas, lançamentos laterais e cartões. Os agentes controlam os jogadores virtuais, cada um cumprindo um papel específico em campo, como em uma partida real.

A partida é baseada em ciclos de tempo discretos, onde cada ciclo representa um intervalo de tempo fixo. Durante cada ciclo, os agentes dos times processam as informações do ambiente, tomam decisões e executam ações com base nessas informações.

4.1.2 SERVIDOR[1]

O servidor é um elemento fundamental para possibilitar a interação entre jogadores e coordenar todas as atividades realizadas durante o jogo, atuando como um sistema central que possibilita a competição entre diferentes equipes. O sistema funciona com uma

[1] The RoboCup Soccer Simulator documentation (rcsoccersim.readthedocs.io)

estrutura cliente-servidor e não apresenta restrições quanto à composição das equipes, desde que sigam os requisitos de comunicação via UDP/IP.

As principais funcionalidades do servidor incluem:

- **Estabelecimento e manutenção de conexões individuais:** O servidor estabelece e mantém conexões individuais com os clientes (jogadores, técnico).
- **Recebimento e processamento de requisições:** Processa as requisições dos clientes relacionadas às ações a serem realizadas no jogo, como chutar, correr, girar, entre outras.
- **Atualização contínua do ambiente de jogo:** Garante a coesão e sincronização das atividades de todos os participantes, assegurando a dinâmica do jogo.
- **Fornecimento de informações sensoriais:** Fornece dados visuais sobre a posição dos objetos no campo e detalhes sobre recursos individuais, como energia e velocidade, aos jogadores.

O servidor opera como um sistema em tempo real, trabalhando com ciclos definidos de duração específica. Cada ciclo requer a execução de ações dentro de um intervalo determinado, exigindo que as requisições dos jogadores cheguem ao servidor no tempo correto.

É crucial destacar que o desempenho de cada jogador impacta diretamente no funcionamento da equipe como um todo. Um atraso ou falha na execução de ações por parte de um jogador pode afetar significativamente o desempenho global da equipe durante o jogo.

4.1.3 MONITOR[2]

O Monitor de Futebol é disponibilizado em duas versões: o rcssmonitor e o rcssmonitor_classic. Ambos desempenham um papel crucial ao exibir informações essenciais durante o jogo, como placar, nomes das equipes e posicionamento dos jogadores e da bola. Enquanto o rcssmonitor_classic oferece funcionalidades básicas, o rcssmonitor, baseado no *frameview* de Artur Merke, expande sua capacidade com recursos adicionais, enriquecendo a experiência de visualização do jogo.

As funcionalidades do Monitor de Futebol incluem:

- **Zoom e depuração:** Permite ampliar áreas específicas do campo, fornecendo uma visão detalhada para depuração e análise de questões específicas do jogo.

[2] The RoboCup Soccer Simulator documentation (rcsoccersim.readthedocs.io)

- **Leitura de dados em tempo real:** Disponibiliza as posições e velocidades atuais dos jogadores e da bola no console a qualquer momento, permitindo uma compreensão dinâmica do desenrolar da partida.
- **Exibição de informações detalhadas:** Apresenta uma variedade de dados, como o campo de visão dos jogadores, níveis de energia e, no caso de jogadores diferentes, o tipo específico de jogador em campo.
- **Interatividade aprimorada:** Oferece a capacidade de movimentar jogadores e a bola usando o mouse, promovendo uma experiência interativa e facilitando a análise tática durante o jogo.

Embora não seja um requisito obrigatório para a execução do jogo no servidor, o Monitor de Futebol desempenha um papel crucial ao possibilitar uma visualização detalhada das atividades durante a partida. Sua utilidade vai desde oferecer uma visão geral do jogo até facilitar o trabalho de árbitros humanos, permitindo que iniciem a partida quando ambos os times estão prontos, além de proporcionar uma ferramenta valiosa para análise e melhoria das estratégias de jogo.

4.2 CONFIGURAÇÃO DO AMBIENTE

Inicialmente, para viabilizar a pesquisa, foi adotada uma abordagem de configuração em ambiente virtual utilizando o *Oracle VM VirtualBox*[3] com sistema operacional Ubuntu 20.04 como base para executar a configuração das bibliotecas e dependências essenciais do *RoboCup Soccer Simulation 2D*. Este processo incluiu a instalação e configuração do monitor[4] e servidor[5] necessários para o ambiente de simulação, descritas por um passo a passo, detalhado no Apêndice A.

A Figura 1 apresenta a imagem de um monitor exibindo o campo sem equipes conectadas.

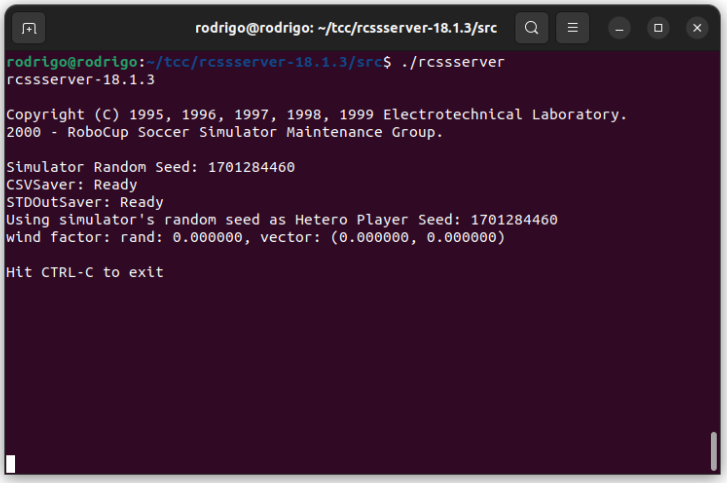
[3] Disponível em: https://www.virtualbox.org/
[4] Disponível em: https://github.com/rcsoccersim/rcssmonitor
[5] Disponível em: https://github.com/rcsoccersim/rcssserver

Figura 1: Monitor.

Fonte: Autores, 2023.

A Figura 2 apresenta um servidor sendo executado em um terminal no linux Ubuntu 20.04.

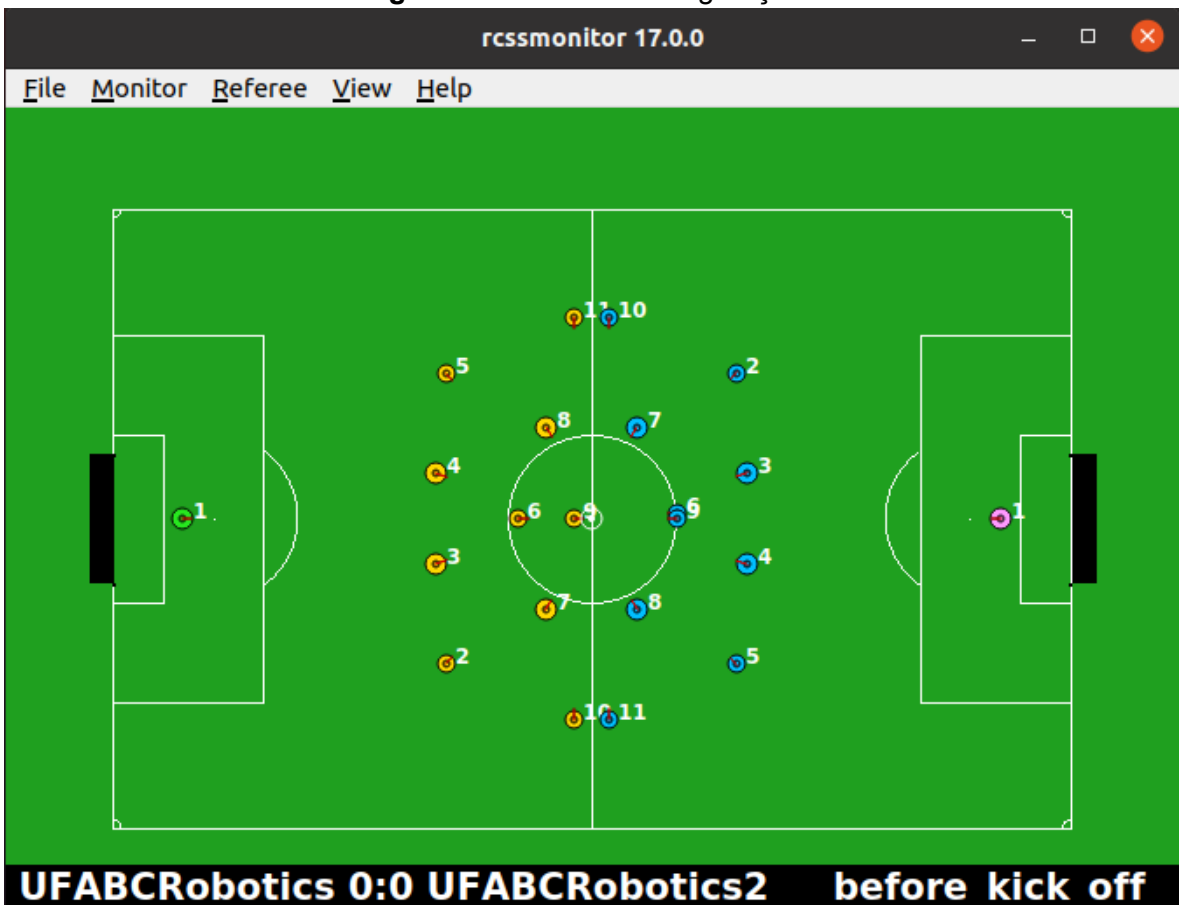
Figura 2: Servidor.



Fonte: Autores, 2023.

Para garantir o funcionamento correto do ambiente, realizamos um teste executando uma partida em que o time base jogava contra si mesmo, verificando assim se todas as configurações estavam operando corretamente. A Figura 3 apresenta o monitor com as equipes conectadas e aguardando o início da partida.

Figura 3: Partida de configuração.



Fonte: Autores, 2023.

4.3 AVALIAÇÃO DO TIME BASE

O time *UFABC-TrillearnBase*[6] é uma equipe desenvolvida por estudantes e pesquisadores da Universidade Federal do ABC (UFABC) no Brasil, focada em competições de futebol de robôs, particularmente na *RoboCup*, uma competição internacional que promove o avanço da pesquisa em robótica e inteligência artificial.

Esse time possui funcionalidades já integradas em sua estrutura. Ele mantém um esquema tático com uma formação inicial que se mantém ao longo da partida. Além disso, executa tarefas básicas, como interceptar a bola, realizar chutes e desempenhar ações defensivas, incluindo as funções de defesa do goleiro.

O código é dividido em várias classes interconectadas que desempenham papéis distintos na simulação. Algumas dessas classes fundamentais incluem:

[6] Disponível em: https://github.com/UFABCRobotics/trilearnBaseUFABC

- ***WorldModel*:** Responsável por representar o estado atual do ambiente de jogo, incluindo informações sobre jogadores, bola, posições e velocidades. Ele tem métodos para atualizar, prever e manipular dados sobre o estado do jogo.

- ***BasicPlayer:*** Define habilidades que os agentes podem executar, como chutar e se movimentar. A execução dessas habilidades depende do estado atual do ambiente.

- ***SenseHandler* e *ActHandler*:** Lidam com a entrada e saída de dados. O *SenseHandler* interpreta as informações recebidas do servidor, enquanto o *ActHandler* gera comandos de ação com base nas decisões tomadas pelo agente.

- ***Formations*:** Contém informações sobre diferentes formações táticas de equipe e métodos para determinar posições estratégicas dos jogadores no campo.

- ***Object*:** É parte do conjunto de classes responsáveis por representar os objetos presentes na simulação do ambiente de futebol de robôs.

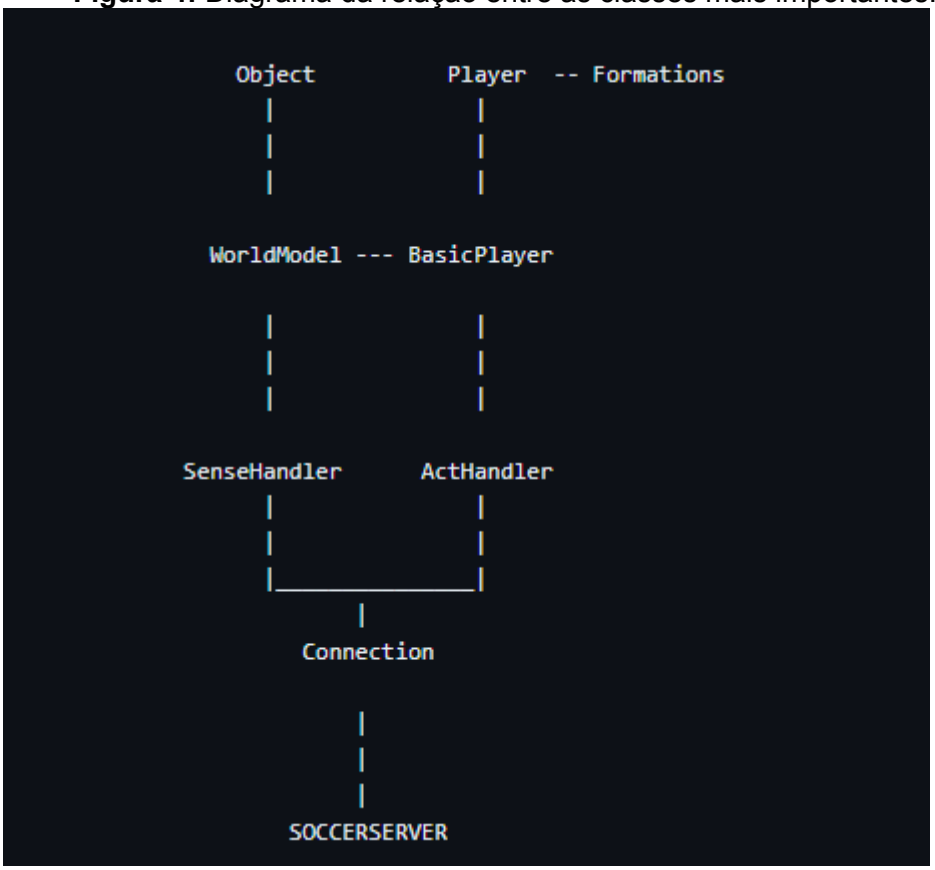
- ***Connection*:** Responsável por estabelecer e gerenciar conexões com um socket (um ponto de comunicação em uma rede). Ela possui métodos para enviar e receber mensagens por meio desse socket.

- ***VecPosition*:** Lida com a representação e manipulação das posições no campo de futebol. Oferece métodos para trabalhar com coordenadas cartesianas (x,y).

- ***SoccerComand*:** Representa os comandos que o agente pode enviar para o servidor para executar as ações durante a partida.

A Figura 4 apresenta um diagrama do relacionamento entre as classes mais importantes.

Figura 4: Diagrama da relação entre as classes mais importantes.



Fonte: UFABC-Robotics, 2016.

Ao examinar detalhadamente o desempenho do time *UFABC-TrillearnBase*, foram identificadas áreas passíveis de aprimoramento. O time apresenta uma estratégia limitada especialmente ao se encontrar com a posse de bola, o que ocasiona um baixo desempenho nas partidas. Atualmente, a abordagem se restringe unicamente a tentativas de chutes ao gol, independentemente da posição dos jogadores no campo.

As observações das dinâmicas de jogo nos levaram a considerar a implementação de mecânicas fundamentais de passes, dribles e cruzamentos, visando proporcionar ao time uma performance mais competitiva, diversificando suas estratégias quando estiver em posse da bola.

4.4 IMPLEMENTAÇÃO DAS ESTRATÉGIAS

A implementação das estratégias de jogo neste estudo foi realizada sem alterar a estrutura ou o funcionamento básico da equipe já existente. A tomada de decisão dos jogadores é executada por meio da função *'deMeer5',* presente no arquivo *PlayerTeam*.

Na estratégia atual do time base, ao reconhecer que um jogador está em posse da bola – ação determinada pela função *isBallKickable* na classe *WorldModel*, que verifica se o

jogador tem alcance da bola – a abordagem adotada é executar um chute em direção ao gol, como ilustrado na Figura 5.

Figura 5: Estratégia de chute ao gol do time base.

```
else if( WM->isBallKickable()) {
   VecPosition posGoal( PITCH_LENGTH/2.0,
             (-1 + 2*(WM->getCurrentCycle()%2)) * 0.4 * SS->getGoalWidth() );
   soc = kickTo( posGoal, SS->getBallSpeedMax() );
   ACT->putCommandInQueue( soc );
   ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
   Log.log( 100, "kick ball" );
 }
```

Fonte: Autores, 2023.

Para executar o chute, uma posição é calculada usando *VecPosition*, com o parâmetro do *PITCH_LENGTH*, que representa o tamanho do campo. Ao dividir essa medida por 2, obtem-se o valor da extremidade horizontal do campo adversário, assegurando que o chute seja em direção ao gol e definindo a coordenada X do chute. Para determinar a coordenada Y, são utilizados os métodos *getCurrentCycle*, que retorna o ciclo atual da partida, e *getGoalWidth*, que fornece a largura do gol. Isso garante que o chute seja realizado em uma posição aleatória dentro das dimensões do gol.

Depois de calcular a posição do chute, essa localização é enviada para a função *kickTo*. Isso resulta na geração de um comando de chute direcionado para essa posição do gol, atribuído no objeto soc. Esse comando de chute é então adicionado à fila de comandos a serem enviados para o servidor.

A estratégia adotada pelo time quando está com a posse de bola foi refinada para proporcionar um controle mais assertivo do jogo, abrindo espaço para a criação de oportunidades mais consistentes para marcar gols.

Nesse aprimoramento, a tomada de decisões foi aperfeiçoada considerando a posição de cada jogador no campo, que possui dimensões de 105x68, onde sua coordenada central é (0,0).

Quando um jogador é identificado como estando com a posse da bola, sua tomada de decisão não se resume mais a simplesmente chutar ao gol. Uma nova estratégia foi adicionada, que envolve verificação do posicionamento do jogador, como demonstra a implementação apresentada na Figura 6.

Figura 6: Implementação da tomada de decisão baseada na posição do campo.

```
else if (WM->isBallKickable()) {
    if (WM->isAfterOpponentLineBox(posBall)) {
      if (WM->isNearSideLine(posBall)){
        soc = cross(posBall);
      }
      else {
        soc = goalKick();
      }
    }
    else {
      ObjectT optionPass = WM->getOptionPass(posAgent);
      if (optionPass == OBJECT_TEAMMATE_UNKNOWN){
        soc = dribble(VecPosition(10, 0).getDirection(), DRIBBLE_SLOW);
      }
      else {
        VecPosition pass = getThroughPassShootingPoint(optionPass,
                                    WM->getGlobalPosition(optionPass),0);
        soc = directPass(pass, PASS_NORMAL);
      }
    }
    ACT->putCommandInQueue(soc);
    ACT->putCommandInQueue(turnNeckToObject(OBJECT_BALL, soc));
  }
```

Fonte: Autores, 2023.

Se o jogador estiver posicionado à frente da linha da área adversária, o que é verificado pela função *isAfterOpponentLineBox*, a decisão de sua ação dependerá da proximidade em relação às linhas laterais do campo, determinada pela função *isNearSideLine*.

Para viabilizar essa abordagem, essas duas funções essenciais foram implementadas no *WorldModel*, como apresentam as Figuras 7 e 8.

Figura 7: Implementação da função isAfterOpponentLineBox.

```
bool WorldModel::isAfterOpponentLineBox(VecPosition pos){
      return pos.getX() > 36;
}
```

Fonte: Autores, 2023.

A função é responsável por avaliar se uma posição específica está à frente da linha da área adversária, utilizando as coordenadas encapsuladas em um objeto *VecPosition*. Ela verifica se a coordenada X é maior que 36, indicando uma ultrapassagem da linha da área adversária e identificando que o jogador está próximo ao gol.

Figura 8: Implementação da função isNearSideLine.

```
bool WorldModel::isNearSideLine(VecPosition pos){
      return pos.getY() > 18 || pos.getY() < -18;
}
```

Fonte: Autores, 2023.

A função tem a responsabilidade de verificar se um jogador está próximo à linha lateral. Ela realiza essa avaliação comparando a coordenada Y. Se essa coordenada for maior que 18, isso indica proximidade do jogador à linha lateral superior. Por outro lado, se a coordenada for menor que -18, sugere que o jogador está próximo à linha lateral inferior.

Se estiver próximo de uma das linhas laterais, a escolha será realizar um cruzamento da bola. Isso é feito atribuindo ao objeto soc o resultado da função *cross*, implementada em *BasicPlayer* e detalhada na seção Estratégia de Cruzamento. Se estiver mais centralizado no campo, a decisão do jogador será chutar ao gol.

A estratégia de chutar a gol foi mantida a partir da configuração original do time, entretanto, foi transferida para uma função na classe *BasicPlayer,* como apresentado na Figura 9, visando aprimorar a organização do código. Essa mudança foi realizada para melhorar a estrutura e a clareza do código-fonte, possibilitando uma gestão mais eficiente das diferentes partes da lógica de jogo.

Figura 9: Implementação da função goalKick.

```
SoccerCommand BasicPlayer::goalKick(){
  VecPosition posGoal(PITCH_LENGTH / 2.0, (-1 + 2 * (WM->getCurrentCycle() % 2))*
                          0.4 * SS->getGoalWidth());
 return kickTo(posGoal, SS->getBallSpeedMax());
}
```

Fonte: Autores, 2023

Se o jogador não estiver à frente da linha da área adversária, sua ação será procurar uma opção de passe por meio da função *getOptionPass*, que é detalhada na seção de Estratégia de Passe. Essa função retorna para o objeto *optionPass* o identificador do jogador que foi selecionado como alvo do passe.

Com esse identificador, a função *getThroughPassShootingPoint* calcula um ângulo ideal para o passe, retornando um *VecPosition* com as coordenadas calculadas. Posteriormente, a função *directPass* atribui ao objeto soc o comando para realizar o passe, que será então enviado ao servidor.

Caso esse identificador seja desconhecido, verificado através da comparação com *OBJECT_TEAMMATE_UNKNOWN*, isso indicaria que não foi identificada uma boa opção para o passe. Nesse caso, a decisão seria realizar um drible. A função drible implementada em *BasicPlayer* retorna o comando que é atribuído ao objeto soc e enviado ao servidor. Mais detalhes sobre essa estratégia serão apresentados na seção de estratégia de drible.

4.4.1 ESTRATÉGIA DE PASSE

No futebol, os passes são a base para um bom desempenho em uma partida, é quando um jogador chuta a bola para outro da mesma equipe, buscando um melhor posicionamento em campo. Passar a bola é controlar o jogo, ditando o ritmo e mantendo a posse com o objetivo de encontrar uma forma de avançar, seja com passes curtos para manter a bola próxima ou longos para atingir o ataque.

Existem dois tipos de passes: o passe rápido, que seria basicamente um lançamento, onde a bola seria tocada em um ponto futuro do jogador, tentando quebrar a linha de marcação adversária e assim o jogador do nosso time correr até a bola. E existe também o passe normal, que é um toque mais curto, na posição mais próxima ao jogador. Utilizamos a técnica do passe normal, evitando que o jogador adversário consiga antecipar o passe, e assim tomar a posse da bola, o que poderia resultar em um ataque promissor do adversário.

Para melhorar nossa eficácia em campo e aumentar nossas chances de marcar gols, adotamos uma estratégia de passes simplificada, implementando a função "*getOptionPass*" na classe *WorldModel*. Essa função se baseia em identificar os jogadores mais bem posicionados no campo para receber a bola e retornar seu identificador. Eles devem estar a uma distância inferior a 25 do jogador com a posse da bola, além de manter uma distância superior a 5 de todos os jogadores adversários. Além disso, é essencial que esses jogadores estejam à frente do jogador com a posse da bola.

Nossa abordagem inicial consiste em analisar as posições de todos os nossos jogadores no campo. Em seguida, selecionamos aleatoriamente um jogador que esteja dentro das condições mencionadas, como demonstra a implementação da Figura 10.

Figura 10: Implementação da função getOptionPass.

```
ObjectT WorldModel::getOptionPass(VecPosition posAgent){
 ObjectT tPlayers[] = {OBJECT_TEAMMATE_2, OBJECT_TEAMMATE_3,
                       OBJECT_TEAMMATE_4, OBJECT_TEAMMATE_5,
                       OBJECT_TEAMMATE_6, OBJECT_TEAMMATE_7,
                       OBJECT_TEAMMATE_8, OBJECT_TEAMMATE_9,
                       OBJECT_TEAMMATE_10, OBJECT_TEAMMATE_11};
 ObjectT oPlayers[] = {OBJECT_OPPONENT_1, OBJECT_OPPONENT_2,
                       OBJECT_OPPONENT_3, OBJECT_OPPONENT_4,
                       OBJECT_OPPONENT_5, OBJECT_OPPONENT_6,
                       OBJECT_OPPONENT_7, OBJECT_OPPONENT_8,
                       OBJECT_OPPONENT_9, OBJECT_OPPONENT_10,
                       OBJECT_OPPONENT_11};
 ObjectT optionsPass[10];
 int posOption = 0;
 for (int i = 0; i < 10; i++)
 {
   if(getAgentObjectType() != tPlayers[i])
   {
     VecPosition posPlayer = getGlobalPosition(tPlayers[i]);
     if(posAgent.getDistanceTo(posPlayer) < 25)
     {
       bool isUnmarked = true;
       for (int j = 0; j < 11 && isUnmarked; j++)
       {
         isUnmarked = isUnmarked &&
               posPlayer.getDistanceTo(getGlobalPosition(oPlayers[j])) > 5;
       }
       if (isUnmarked && posPlayer.getX() > posAgent.getX() + 5)
       {
         optionsPass[posOption] = tPlayers[i];
         posOption++;
       }
     }
   }
 }
 if(posOption == 0)
   return OBJECT_TEAMMATE_UNKNOWN;
 return optionsPass[getCurrentCycle() % posOption];
}
```

Fonte: Autores, 2023.

A função *getOptionPass* recebe como parâmetro um objeto *VecPosition* representando a posição do agente que pretende realizar um passe. Para cada identificador de companheiros de equipe listados no vetor *ObjectT tPlayers*, a função executa uma série de verificações para determinar se o companheiro é uma opção viável para receber o passe.

Para ser considerada uma boa opção de passe, o companheiro de equipe alvo não deve ser o agente atual com a posse da bola. O identificador do agente com a bola é obtido pela função *getAgentObjectType*. Caso o identificador do alvo do passe seja diferente do agente com a bola, é verificado se a distância do alvo do passe é menor que 25 em relação ao agente com a bola, a posição do alvo é obtida através da função *getGlobalPosition* e armazenada na variável *posPlayer*.

Para avaliar se o jogador alvo está livre para receber o passe, a função verifica a distância entre o alvo e todos os oponentes listados no vetor *ObjectT oPlayer*. Essa verificação é realizada utilizando a função *getDistanceTo* para calcular a distância entre o alvo e cada oponente. Se a distância for maior que 5, o jogador é considerado livre para receber o passe.

O status de "livre" ou "marcado" do jogador é indicado pela variável booleana *isUnmarked*. Se *isUnmarked* for *true*, significa que o jogador está livre; se for *false*, indica que o jogador está marcado por algum oponente.

Além disso, para ser considerado uma boa opção de passe, o jogador alvo deve estar à frente do agente com a posse da bola. Isso é verificado comparando a posição X do alvo do passe com a posição X incrementada em 5 do jogador com a bola. Essa comparação garante que o jogador destinatário do passe esteja sempre à frente do agente atual.

Os identificadores dos jogadores considerados boas opções de passe são adicionados ao vetor *optionsPass*. Ao final da função, se houver jogadores no vetor, é retornado um identificador aleatório dentre as boas opções encontradas. Caso nenhum jogador tenha sido adicionado ao vetor, a função retorna a referência *OBJECT_TEAMMATE_UNKNOWN*, indicando que nenhum jogador viável foi identificado para o passe. Essa referência representa um jogador desconhecido ou inexistente.

Essa abordagem nos permite avançar com mais segurança, mantendo a pressão no ataque e minimizando as chances de perda de posse na defesa, que poderiam resultar em

ataques perigosos do adversário. Dessa forma, nossa equipe consegue manter a iniciativa e diminuir os riscos defensivos, visando sempre a meta oposta para marcar gols.

4.4.2 ESTRATÉGIA DE DRIBLE

O drible é uma técnica essencial no futebol. Consiste em enganar o adversário com movimentos rápidos e habilidosos, podendo passar a bola por entre as pernas ou contornando o jogador adversário. É uma forma de superar a marcação de uma forma individual e avançar no campo, criando oportunidades de ataque para a equipe. Utilizar o drible é importante para desequilibrar a defesa adversária e criar chances de marcar gols.

A técnica de drible se baseia em avançar a bola em uma direção específica, com o objetivo de superar o adversário. Para atingir esses objetivos existem três formas de drible apresentadas no time: o *'DRIBBLE_WITHBALL'*, que é basicamente fazer o movimento com a bola bem próxima ao jogador, tornando difícil a recuperação pelo adversário; o *'DRIBBLE_SLOW'*, que é realizado com a bola um pouco mais distante do jogador, mas fazendo com que tenha um leve ganho de performance e assim passar por seu adversário; o *'DRIBBLE_FAST'* , que consiste no jogador avançar a bola mais longe do seu corpo tendo o máximo de ganho de performance, porém facilita a vida do marcador para que ele possa recuperar a posse de bola, visto que o jogador não tomará uma decisão rápida para evitar isso, já que a bola vai estar longe do seu domínio.

Diante das opções de dribles, optamos por adotar a estratégia do *'DRIBBLE_SLOW'*, que se apresenta como uma escolha intermediária entre as duas outras técnicas. Este tipo de drible oferece uma progressão satisfatória no campo, comparável ao drible com a bola, porém com menor risco de perda da posse de bola, diferentemente do drible rápido.

A técnica de drible foi incorporada na tomada de decisão do jogador para melhorar o desenvolvimento do jogo, buscando equilibrar o avanço progressivo com a segurança da manutenção da posse, quando não existem boas opções de passe, como apresentado na Figura 6. Assim, oferecendo a oportunidade de realizar um chute ao gol, efetuar um cruzamento preciso ou conduzir a bola até encontrar uma boa opção de passe, dependendo da sua localização no campo.

4.4.3 ESTRATÉGIA DE CRUZAMENTO

O cruzamento é uma técnica fundamental no futebol. Consiste em enviar a bola da lateral do campo em direção à área de ataque, buscando alcançar um companheiro de equipe para finalizar a gol. É uma estratégia utilizada para criar oportunidades de marcar, aproveitando a presença de jogadores na área adversária. O cruzamento demanda precisão e *timing* para ser eficaz, sendo uma arma valiosa para explorar as defesas adversárias e criar momentos de perigo durante a partida.

Para realizar o cruzamento da bola na área adversária, utilizamos da posição do jogador. Consiste no jogador estar próximo a linha de fundo e da linha lateral, onde fazemos essa verificação a partir das funções "*isAfterOpponentLineBox*" apresentada na Figura 7 e *isNearSideLine* apresentada na Figura 8, seja quando ele receber um passe ou chegar até essa posição através do drible em cima do adversário. Ao chegar nessa posição, o jogador faz um cruzamento, ou seja, ele toca a bola para o meio da área, onde terá um jogador pronto para finalizar ao gol.

Para realizar essa estratégia durante o jogo, implementamos a função *'cross'*. Inicialmente, a função analisa a posição da bola para determinar se o jogador em posse dela está na metade superior ou inferior do campo. Com essa informação, ela decide a direção do cruzamento. Após identificar a área em que o jogador se encontra e definir a posição do cruzamento, o movimento é executado, consistindo em um passe direcionado para o meio da área adversária.

As Figuras 11 e 12 apresentam respectivamente a implementação das funções "*getCrossDirection*" e "*cross*".

Figura 11: Implementação da função "*getCrossDirection*".

```
VecPosition WorldModel::getCrossDirection(VecPosition pos){
     return VecPosition(pos.getX(), pos.getY() + (pos.getY()  > 18) ?
                                                              -10 : 10);
}
```

Fonte: Autores, 2023.

Figura 12: Implementação da função "*cross*".

```
SoccerCommand BasicPlayer::cross(VecPosition posBall){
  VecPosition directionCross = WM->getCrossDirection(posBall);
  return kickTo(directionCross, SS->getBallSpeedMax());
}
```

Fonte: Autores, 2023.

5. RESULTADOS

Com o intuito de avaliar os impactos das alterações feitas, foram realizados 10 jogos entre a equipe que adotou as mudanças e o time original. Os resultados destacaram um notável e expressivo domínio do time com as estratégias implementadas, com 10 vitórias. Os resultados das partidas estão apresentados na Tabela 1.

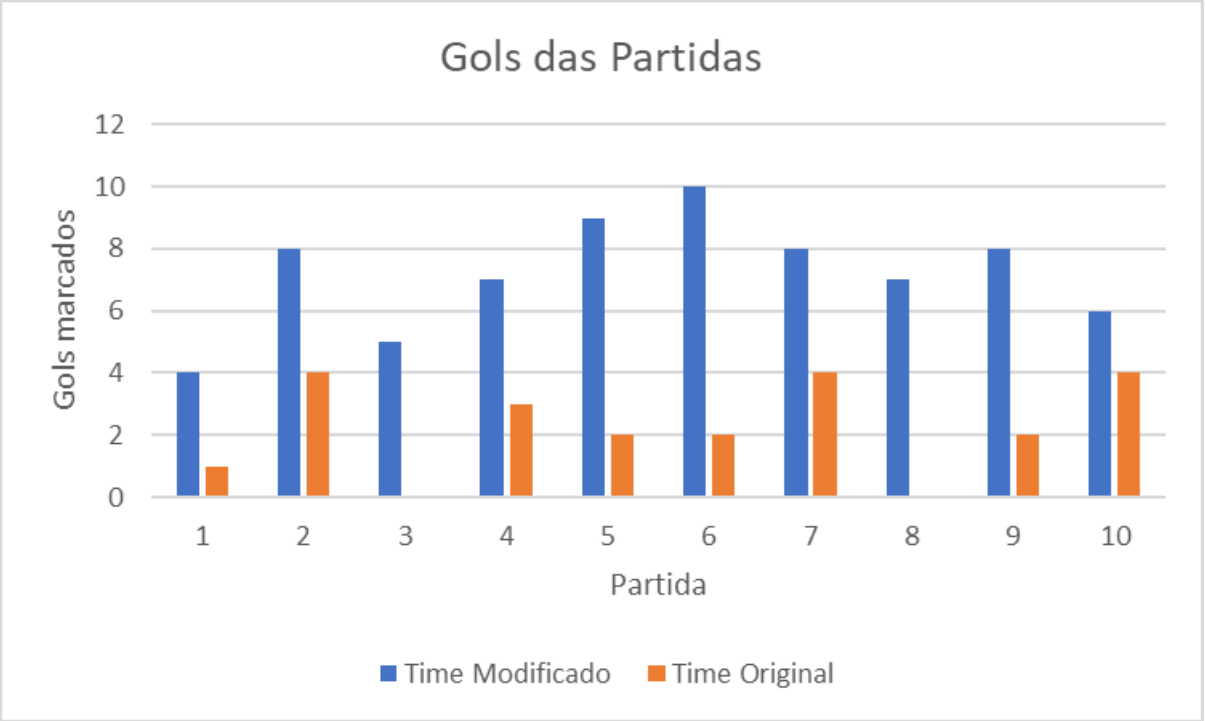
Tabela 1: Apresentação dos resultados dos jogos.

Jogo	Time Modificado	Time Original
1	4	1
2	8	4
3	5	0
4	7	3
5	9	2
6	10	2
7	8	4
8	7	0
9	8	2
10	6	4

Fonte: Autores, 2023.

A Figura 13 apresenta uma representação em formato de gráfico dos gols marcados por cada time na realização das partidas.

Figura 13: Gráfico de gols das partidas.



Fonte: Autores, 2023.

Após os jogos, surgiram indícios significativos da superioridade do time modificado, influenciada pelas estratégias mencionadas. Este time conseguiu marcar 72 gols, com uma média de 7.2 gols por partida, em contraste com os 22 gols registrados pelo time base. Assim, as modificações implementadas contribuíram para um aumento de 50 gols.

6. CONCLUSÃO

Os resultados obtidos destacam a notória superioridade do time modificado em relação ao time original. As técnicas implementadas desempenharam um papel fundamental na considerável diferença de gols marcados durante os jogos, permitindo um controle mais efetivo da posse de bola e na criação de jogadas. Além disso, proporcionaram maior flexibilidade tática com base no posicionamento dos jogadores em campo.

A introdução da técnica de passe não apenas melhorou o desenvolvimento do jogo, mas também proporcionou um controle mais apurado da bola, evitando chutes sem critério que frequentemente resultam na perda da posse e em menor número de oportunidades para marcar gols.

Em conjunto com as estratégias de passe, os dribles ofereceram ao time uma nova abordagem no desenrolar das partidas, permitindo manter a posse da bola em situações onde não havia opções de passe viáveis, além de avançar no campo, criando cenários variados de jogo.

A tática de cruzamento, por sua vez, abriu oportunidades para jogadas pelos lados do campo, introduzindo um novo recurso ofensivo que resultou em um aumento significativo nas chances de gols criadas. Foi por meio dessa estratégia que a maioria dos gols foi convertida.

Em síntese, a implementação coordenada dessas técnicas proporcionou não apenas um jogo mais controlado e criativo, mas também fortaleceu a capacidade ofensiva do time, refletida na notável diferença de gols marcados.

Embora as técnicas atualmente em uso tenham contribuído significativamente, há espaço para aprimorar os resultados por meio de abordagens alternativas. Uma dessas possibilidades é a aplicação de métodos como o aprendizado de máquina e o aprendizado por reforço para aprimorar as decisões tomadas durante as partidas em campo.

Além disso, para evoluir ainda mais, seria interessante direcionar esforços para otimizar o desempenho defensivo, especialmente no que diz respeito ao goleiro, visando a redução do número de gols sofridos.

7. REFERÊNCIAS

AKIYAMA, Hidehisa; NAKASHIMA, Tomoharu; FUKUSHIMA, Takuya; SUZUKI, Yudai; OHORI, An. **HELIOS2018: RoboCup 2018 Soccer Simulation 2D league Champion**. Fukuoka University, Fukuoka, Japan, [S. l.], p. 1-4, 23 out. 2019.

AKIYAMA, Hidehisa; NAKASHIMA, Tomoharu; FUKUSHIMA, Takuya; ZHONG, Jiarun; SUZUKI, Yudai; OHORI, An. **HELIOS 2019: Team Description Paper**. Fukuoka University, Fukuoka, Japan, [S. l.], p. 1-4, 10 jul. 2018.

COSTA, Alexandra; TEIXEIRA, Giovany; BATISTA, Leonardo; CAMPOS, Lucas; RODRIGUES, Maykel; RIOS, Renan; MENDES, Roberto. **Asimov Soccer Simulation 2D**. Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo, 2016.

GOMES, Dêmis Carlos Fonseca; ALVES, Antônia Maria de Paula; SOUSA, Danilo Júnior Fonseca de; SILVA, Fábio Santana da; COSTA, Paulo César Pereira. **SPARTRON: SOCCER SIMULATION 2D TEAM DESCRIPTION PAPER**. Instituto Federal de Educação, Ciência e Tecnologia do Tocantins, 2018.

How Simulation Tools are Transforming Education and Training - ETC. Disponível em: https://www.etcourse.com/simulation-tools-transform-education-and-training.html. Acesso em: 7 dez. 2023.

MAIOR, Ludovico. **Unidade didática, análise combinatória, esportes e conhecimentos em ação**. Universidade Estadual de Ponta Grossa, 2008.

ROBOCUP Soccer Simulation League – 2D soccer simulation and 3D soccer simulation. Disponível em: https://ssim.robocup.org/. Acesso em: 28 nov. 2023.

Robocup Soccer Simulation 2d. Disponível em: https://www.instructables.com/Robocup-Soccer-Simulation-2d/. Acesso em: 28 nov. 2023.

RoboCup Simulation 2D League. Disponível em: https://www.robocup.org/leagues/24. Acesso em: 7 dez. 2023.

SANTOS, Tanilson. **Jogadas Coletivas por meio de Comunicação Multi-Agente para a equipe iBots da Categoria de Simulação 2D da RoboCup.** Universidade Federal de Tocantins, 2011.

SILVA, Paulo; CAETANO, Amanda; ALMEIDA, Rodrigo; PRADO, Marcos. **Ambiente virtual para simulação de futebol de robôs aplicado ao estudo de programação, sistemas inteligentes e robótica**. In: Escola Regional de Computação Bahia, Alagoas e Sergipe (ERBASE), 2019, Ilhéus. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 477-486.

The RoboCup Soccer Simulator Users Manual — The RoboCup Soccer Simulator documentation. Disponível em: https://rcsoccersim.readthedocs.io/en/latest/index.html. Acesso em: 28 nov. 2023.

Time base TrilearnBase — UFABCRobotics. Disponível em: https://github.com/UFABCRobotics/trilearnBaseUFABC. Acesso em: 28 nov. 2023.

APÊNDICE A - Instalação e configuração dos arquivos do Robocup

Para executar as configurações corretamente, é crucial seguir todos os comandos na sequência fornecida, sem pular nenhum, e utilizar a versão do Ubuntu 20.04. Isso se deve ao fato de que algumas dependências essenciais não estão mais disponíveis em versões mais recentes do sistema operacional.

Instalação de Pacotes Essenciais

Os comandos apresentados a seguir instalam todas as dependências necessárias para a configuração do ambiente, execute-os:

- sudo apt update
- sudo apt install build-essential
- sudo apt install qt5-default
- sudo apt install libfontconfig1-dev
- sudo apt install libaudio-dev
- sudo apt install libxt-dev
- sudo apt install libglib2.0-dev
- sudo apt install libxi-dev
- sudo apt install libxrender-dev

- sudo apt-get install build-essential
- sudo apt-get install pkg-config
- sudo apt-get install autoconf
- sudo apt-get install automake
- sudo apt-get install m4
- sudo apt-get install libtool
- sudo apt-get install libevent-dev
- sudo apt-get install libboost-all-dev
- sudo apt-get install perl

Instalação do Monitor

Após a instalação das dependências, faça o download do monitor no link:

https://github.com/rcsoccersim/rcssmonitor

Após o download, extraia o arquivo em uma pasta e execute os comandos:

- ./bootstrap
- ./configure
- make

- sudo make install

Instalação do Servidor

Faça o download do servidor no link:

https://github.com/rcsoccersim/rcssserver

Instale as dependências necessárias para executar o servidor:

- sudo apt-get install flex
- sudo apt-get install bison

Após a realização do download, extraia o servidor em uma pasta e execute os comandos:

- ./bootstrap
- ./configure
- make
- sudo make install

Execução do Servidor

Ao realizar a instalação de todos os arquivos necessários, execute o comando para iniciar o servidor e o monitor:

- rcsoccersim

Instalação do Time UFABC

Instale a dependência abaixo para poder executar o time:

- sudo apt-get install tcsh

Após executar o servidor, realize os seguintes comandos para colocar o time em campo:

- ./start.sh