UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
COORDENAÇÃO DE CIÊNCIA DA COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO MENDES GARCÊS

UMA NOVA PROPOSTA PARA O TIME DE FUTEBOL ROBÓTICO SIMULADO UFMA2D

São Luís
2019

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
COORDENAÇÃO DE CIÊNCIA DA COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO MENDES GARCÊS

UMA NOVA PROPOSTA PARA O TIME DE FUTEBOL ROBÓTICO SIMULADO UFMA2D

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Paulo Rogério de Almeida Ribeiro

São Luís
2019

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Garcês, Rodrigo Mendes.
Uma nova proposta para o time de futebol robótico simulado UFMA2D / Rodrigo Mendes Garcês. - 2019.
52 f.

Orientador(a): Prof. Dr. Paulo Rogério De Almeida Ribeiro.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2019.

1. Multirrobôs cooperativos. 2. Robocup Simulation 2D League. 3. Sistema multi-robótico. I. Ribeiro, Prof. Dr. Paulo Rogério De Almeida. II. Título.

RODRIGO MENDES GARCÊS

UMA NOVA PROPOSTA PARA O TIME DE FUTEBOL ROBÓTICO SIMULADO UFMA2D

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Data da Defesa: 08 de janeiro de 2019

Banca Examinadora

Paulo Rogério de A. Ribeiro

Prof. Dr. Paulo Rogério de Almeida Ribeiro
Universidade Federal do Maranhão
Orientador

Prof. Dr. Geraldo Braz Júnior
Universidade Federal do Maranhão
Membro da Banca

Prof. Dr. Alexandre César Muniz de Oliveira
Universidade Federal do Maranhão
Membro da Banca

Prof. Dr. Luciano Reis Coutinho
Universidade Federal do Maranhão
Membro da Banca

São Luís
2019

AGRADECIMENTOS

Em primeiro lugar, a Deus, por me permitir chegar até aqui. Segundamente aos meus pais, José Benedito e Maria do Livramento, por darem todo o suporte necessário ao longo desses anos.

Aos meus amigos que me ajudaram durante esta longa caminhada, em especial Bruno Eduardo, Jorge Lucas, Jorge luís, Hugo, Tarcio, Erick, Daniel, Alexandre Pessoa, Samir, João Baluz e Matheus Menezes.

Aos professores que tive a oportunidade de conhecer durante a graduação, em especial ao professor Dr. Paulo Rogério de Almeira Ribeiro, orientador deste trabalho, por sua paciência e dedicação, respondendo as minhas inúmeras perguntas.

Agradeço a todos que direta ou indiretamente contribuíram para a realização desde trabalho, inclusive aqueles que me deram uma palavra de apoio quando pensei em desistir do projeto.

Em todas as circunstâncias, dai graças,
porque esta é a vosso respeito
a vontade de Deus em Jesus Cristo.
(Bíblia Sagrada, I Tessalonicenses 5, 18)

RESUMO

O futebol robótico apresenta-se como um problema capaz de envolver robótica e inteligência artificial em diversos níveis. Apresenta um ambiente extremamente dinâmico, capaz de adicionar um certo fator de aleatoriedade às partidas, o que se torna ainda mais complexo por se tratar de um sistema autônomo multiagentes, em que estes precisam, além de compreender o ambiente a sua volta, identificar os adversários e a bola, recebendo sempre informações com ruído, de modo a simular um ambiente real.

Este trabalho objetiva a atualização do time base utilizado no time de futebol robótico UFMA2D e a implementação de estratégias reais utilizando a ferramenta gráfica de edição de formação *Fedit*, bem como fornecer um panorama mais abrangente sobre os times base disponíveis atualmente.

A partir desta nova análise sobre os times base, mudou-se o time base do UvA Trilearn para o Agent2d, e desenvolveram-se novas estratégias reais de saída de bola, com as formações (4-3-2-1) e (4-3-3), as quais foram combinadas e implementadas utilizando a ferramenta gráfica de edição de formação *Fedit*, sendo esta fornecida juntamente com o Agent2d.

Os resultados obtidos com estas modificações se mostraram bastante animadores, superando com ampla vantagem as implementações anteriores do UFMA2D, e superando com pouca margem o Agent2d, o que demonstra que as estratégias reais implementadas no time, apesar de simples, conseguiram aumentar a eficiência do time de maneira significativa.

Palavras-chave: Robocup Simulation 2D League. Sistema multi-robótico. Multirrobôs cooperativos.

ABSTRACT

Robotic football presents itself as a problem capable of involving robotics and artificial intelligence on several levels. It presents an extremely dynamic environment, capable of adding a certain factor of randomness to matches, which becomes even more complex because it is an autonomous multi-agent system, in which they need, in addition to understanding the surrounding, identify opponents and the ball, always receiving information with noise, in order to simulate a real environment.

This work aims at updating the base team used in the UFMA2D robotic soccer team and implementing real strategies using the Fedit graphical training editing tool, as well as providing a more comprehensive overview of the currently available base teams.

From this new analysis on the base teams, the base was moved from UvA Trilearn UvA to Agent2d, and new real ball-out strategies were developed, with formations (4-3-2-1) and (4-3-3), which were combined and implemented using the graphical editing tool *Fedit* , which is supplied together with Agent2d.

The results obtained with these modifications were quite encouraging, surpassing with great advantage the previous implementations of UFMA2D, and surpassing with little margin the Agent2d, which shows that the real strategies implemented in the team, although simple, managed to increase the efficiency of the team significantly.

Keywords: Robocup Simulation 2D League. Multi-robots system. Cooperative multirobots.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FIFA	Fédération Internationale de Football Association
FIRA	Federation of International Robot-Soccer Association
GPL	General Public License
PET	Programa de Educação Tutorial
RCSS	RoboCup Soccer Simulation
TDP	Team Description Paper
UDP	User Datagram Protocol

SUMÁRIO

1 INTRODUÇÃO

Desde que foi criada em 1996 (KITANO et al., 1997), a RoboCup fornece um problema padrão para o futebol de robôs. A comunidade RoboCup promove o desenvolvimento de robôs e algoritmos inteligentes, por meio de competições, sendo estas utilizadas por cientistas e estudantes do mundo todo. Tais competições são divididas em várias categorias: RoboCup soccer (futebol 2d e 3d, com robôs físicos e simulados), RoboCup rescue (salvamento em condições complexas), RoboCup @home (robôs para auxiliar na vida doméstica), RoboCup industrial (robôs industriais) e RoboCup junior (destinada a crianças e adolescentes). Neste trabalho será abordado a RoboCup soccer simulation 2d, subcategoria da RoboCup Scoccer simulation league.

Sua primeira competição oficial foi realizada em 1997, com a participação de 40 equipes, distribuídas entre as categorias reais e simuladas, e cerca de 5000 espectadores. Neste ano e no seguinte, a plataforma de simulação sofreu diversas modificações, a fim de corrigir diversos bugs e falhas, como por exemplo, a limitação da comunicação entre os agentes do time e a separação da comunicação dos dois times. Tais modificações aprimoraram a qualidade da simulação e a estabilidade do simulador, enquanto tornavam a simulação mais justa e igualitária, para times iniciantes e profissionais. Desde então, a plataforma sofreu poucas modificações, basicamente para melhorar a performance e corrigir bugs encontrados *a posteriori*.

Atualmente, existem duas entidades responsáveis pela realização das competições: a RoboCup Federation (KITANO et al., 1997) e a FIRA (FIRA, 2017). As duas se diferenciam na maneira de abordar o problema. Enquanto na FIRA os robôs tem baixo grau de autonomia, tomando pequenas decisões de forma autônoma, mas seguindo diversas restrições, na RoboCup os robôs precisam resolver problemas complexos de maneira totalmente autônoma. Pela relativa simplicidade em compreender os desafios propostos e pela abrangência do tema, os eventos organizados pelas duas entidades conseguem atrair cada vez mais pessoas a cada edição anual.

As competições de futebol robótico simulado se tornaram um laboratório para as áreas de inteligência artificial e sistema multiagentes, bem como algoritmos de posicionamento e de redução de ruído, dentre outras, uma vez que sua estrutura, embora simulada, é altamente complexa, simulando a realidade, incluindo muitas variáveis reais tais como: atrito, vento, inércia, falha de comunicação por ruído e variação dos dados obtidos pela presença de ruído, sendo um ambiente altamente dinâmico.

Por conta de tais características, as competições de futebol simulada crescem ano após ano, com cada vez mais equipes interessadas em participar. Mas a construção de um time mostra-se algo realmente complexo, uma vez que são necessárias muitas implementações de baixo nível, bem como o conhecimento de todos os protocolos utilizados. Diante desta dificuldade em iniciar a construção de um novo time, surgiram os time base sendo estes uma implementação implementação completa, mas sem a parte de controle inteligente, normalmente disponibilizada por algumas equipes profissionais.

Tais times base são de extrema relevância, uma vez que disponibilizam um código fonte funcional e normalmente bem documentado, sendo disponibilizado sob alguma licença que permita tanto a modificação quanto a redistribuição, desde que sejam citados os autores originais. E isso proporciona uma gigantesca economia de tempo na formação de uma nova equipe, uma vez que esta já parte de uma implementação funcional e razoavelmente eficiente, sem precisar se preocupar com as implementações de baixo nível nem necessitar de um estudo profundo sobre as implementações do servidor e seus métodos de comunicação.

1.1 Justificativa

O Robocup Simulation 2D League tem se demonstrado um desafio bastante complexo, pois todos os jogadores (agentes) funcionam de maneira autônoma, sem qualquer interferência humana, em tempo real e com recursos bastante limitados. Neste cenário, nota-se a dificuldade não só em conseguir desenvolver as estratégias, mas também saber quando usar cada estratégia desenvolvida, visto que o ambiente da simulação é extremamente dinâmico. Demonstra-se a importância de um algoritmo inteligente, ainda que simples, para definir qual estratégia usar em determinada situação, a fim de obter um placar o mais favorável possível.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho visa aprimorar o trabalho já existente no time UFMA2D, a fim de melhorar a performance do time, além de compreender melhor o funcionamento da plataforma de simulação do Robocup Simulation 2D League.

1.2.2 Objetivos Específicos

- Mudança de time base.
- Implementação de estratégias reais no time utilizando a ferramenta gráfica de edição de formação *Fedit*.

2 REFERENCIAL TEÓRICO E TRABALHOS RELACIONADOS

Sessão descritiva das principais ligas e subligas que compõe o RoboCup Soccer, bem como a estrutura detalhada de funcionamento e da arquitetura da subliga de simulação 2d, utilizada neste trabalho. Descreve também um estudo revisado sobre alguns times base, a fim de buscar inovações que possam ser implementadas no time de futebol robótico UFMA2D.

2.1 História da RoboCup

A RoboCup foi criada em 1996 (KITANO et al., 1997), após o IBM deep blue(IBM, 2018) derrotar o campeão mundial de xadrez daquele ano, como um novo desafio padrão da inteligência artificial, com a missão de derrotar o time campeão da copa do mundo de 2050 utilizando apenas sensores equivalentes aos dos humanos, e controlados de forma autônoma.

Em 1997 foram realizadas as primeiras competições oficiais, já com regras bem definidas, passando ainda por um período de ajustes até o ano seguinte para corrigir falhas que não foram previstas oficialmente. E após esse período, sofreu poucas modificações a fim de melhorar a compatibilidade e estabilidade da plataforma, sendo considerada uma plataforma bastante estável e bem definida desde então, sendo considerada como um dos problemas padrão de robótica.

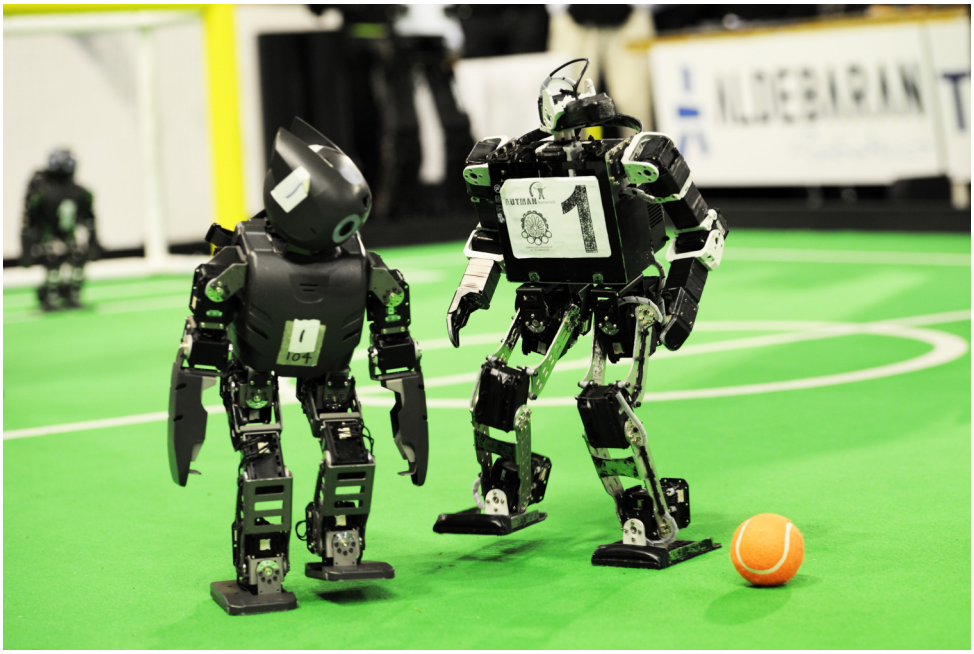
Inicialmente, a RoboCup limitava-se ao RoboCup Soccer, sendo composta das ligas 2d e 3d, com robôs físicos ou simulados. Com o passar dos anos, foram sendo adicionadas novas ligas: RoboCup Rescue (salvatagem em áreas de difícil acesso), RoboCup@Home (Auxiliar nas tarefas domésticas), RoboCup Industrial (Tarefas relacionadas ao ambiente de trabalho, armazéns e indústria) e RoboCup Junior (Modalidade voltada à educação e divulgação da robótica para crianças e adolescentes).

O problema da RoboCup Soccer consiste em um time de jogadores autônomos de futebol, com capacidade de sensoriamento reduzida , de tal modo que seja o mais próximo possível da capacidade de sensoriamento dos humanos (KITANO, 1998). Tal controle é realizado em tempo de execução. As categorias do RoboCup Soccer são descritas a seguir:

2.1.1 RoboCup Soccer - Humanoid

Na liga humanoide, ilustrada na Figura 1, robôs com físico e sensores semelhantes aos humanos jogam futebol contra outro time. E ao contrário de outros robôs humanoides, a percepção e o modelo de mundo não são simplificados utilizando sensores "extra humanos, como sensores de distância. Além dos desafios já inerentes ao futebol, ainda há outros adicionais à categoria, como: andar, correr e chutar a bola enquanto mantém o equilíbrio, percepção da bola, dos outros jogadores e do campo, auto-localização e jogo de equipe.

Figura 1 – RoboCup humanoid



Fonte: RoboCup Humanoid Kid Size (2013)

2.1.2 RoboCup Soccer - Middle size

Na liga *middle size*, ilustrada na Figura 2, os robôs possuem um tamanho médio, com um formato predefinido, com restrições de peso e altura, mas cada equipe pode projetar o seu próprio robô e utilizar diferentes sensores, desde que todos os sensores sejam integrados (sem sensores externos que possam garantir alguma vantagem ao time.). Possui foco em mecatrônica, controle e cooperação multiagentes.

Figura 2 – RoboCup middle size

Fonte: RoboCup middle Size (2013)

2.1.3 RoboCup Soccer - Small size

Na liga *small size*, ilustrada na Figura 3, os robôs são redondos e pequenos, devendo caber em um círculo de 18cm de diâmetro, e com no máximo de 15 cm de altura, sendo o desenvolvimento tanto do hardware como do software de responsabilidade do próprio time, o que permite uma certa variação entre os times. O campo mede 12 metros de comprimento por 9 de largura, sendo verde com marcações brancas. Os times são compostos de 8 jogadores cada um, identificados utilizando 3 marcadores coloridos para identificação dos jogadores (um em cada lado e mais um na frente), devendo a combinação dos marcadores ser única para cada jogador da partida, de ambos os times.

Uma câmera, ou até quatro, dependendo do ambiente em que é instalado o campo, são posicionadas acima do campo, sendo conectadas a um computador localizado fora do campo, responsável por fornecer as informações sobre os jogadores e a bola, utilizando um software *open source* para realizar o reconhecimento de todos os elementos do campo, software este que foi desenvolvido pelo RoboCup em parceria com participantes de algumas equipes da referida categoria e está em constante desenvolvimento, a fim de facilitar a criação de novas equipes para a categoria.

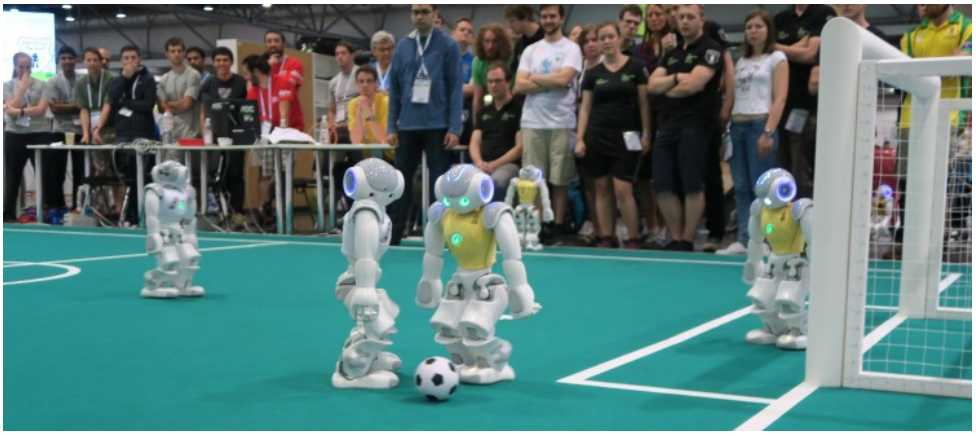
Figura 3 – RoboCup small size

Fonte: Wiki RoboCup (2015)

2.1.4 RoboCup Soccer - Standard plataform

Na liga *standard plataform*, ilustrada na Figura 4, dois times competem utilizando robôs padrão, operando de forma totalmente autônoma, ao mesmo tempo em que precisam coordenar as ações como um time. O campo, assim como nas outras ligas, é plano e verde, com marcações em branco, cabendo a cada robô identificar em tempo real as marcações do campo, a bola e os adversários. Há uma grande variedade de códigos *open source* para realizar uma grande variedade de tarefas, como reconhecer o apito do árbitro, identificar a bola e gerenciar a mecânica/ física do robô, a fim de realizar as ações com a maior eficiência possível.

Figura 4 – RoboCup standard plataform



Fonte: RoboCup Standard Plataform (2018)

2.1.5 RoboCup Soccer - Simulation League

A liga simulada (*simulation league*) se divide em liga simulada 2d (*simulation league 2d*) e liga simulada 3d (*simulation league 3d*). Possui a vantagem de não necessitar de robôs e infraestrutura física, possibilitando que equipes com menor orçamento consigam desenvolver times competitivos. As subligas são detalhadas a seguir.

2.1.5.1 RoboCup Soccer - Simulation League 2d

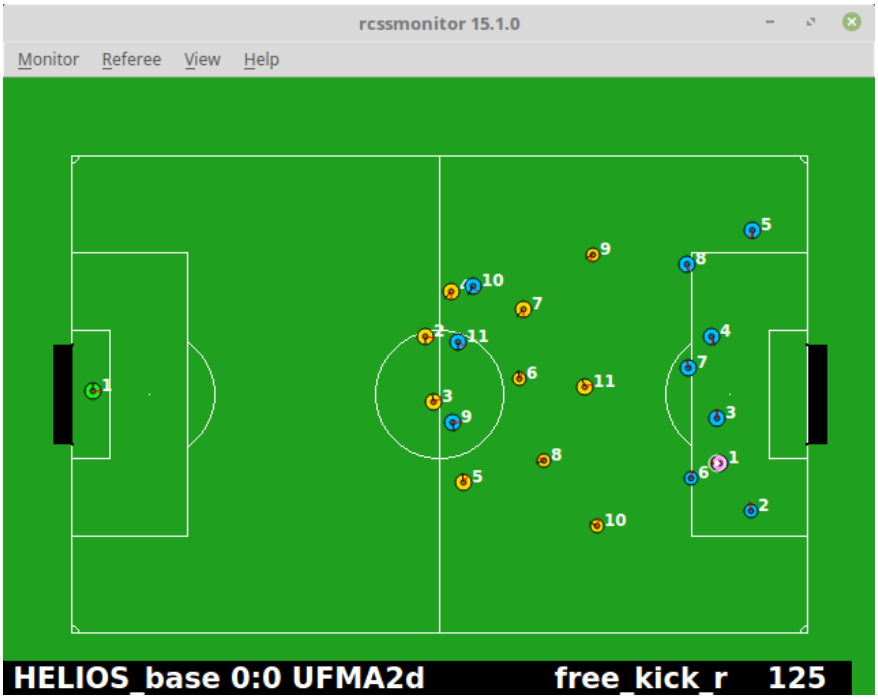
A subliga de futebol robótico simulado 2d, ilustrada na Figura 5, é semelhante a um jogo de futebol de botão visto de cima, mas baseia-se nas regras oficiais de futebol de campo da FIFA.

Cada time é composto por 11 jogadores, sendo estes agentes autônomos, e um técnico. Toda a comunicação dos jogadores (entre dois ou mais jogadores e entre jogador e técnico) possui ruído aleatório, para simular um cenário de implementação real, o que inclui ruído em todas as leituras dos sensores e nos comandos enviados aos atuadores. já o técnico consegue obter informações sem ruídos, tais quais são, mas tem comunicação bastante limitada com os jogadores, a fim de impedir que este controle os jogadores utilizando as informações sem ruído, o que poderia dar uma grande vantagem ao time.

2.1.5.2 RoboCup Soccer - Simulation League 3d

A subliga de futebol robótico simulado 3d, ilustrada na Figura 6, surgiu da necessidade de uma plataforma simulada que fosse mais complexa que o 2d, levando em conta fatores como equilíbrio estático e dinâmico, além de fatores como a percepção do campo, da bola e dos adversários.

Figura 5 – RoboCup simulation league 2d executando no *soccer monitor*



Os círculos amarelos e o verde representam os jogadores e o goleiro do time da esquerda, respectivamente, enquanto os círculos azuis e o rosa representam os jogadores e o goleiro do time da direita. A bola é representada por um círculo branco.
Fonte: Autor (2019)

Figura 6 – RoboCup simulation league 3d

Fonte: RoboCup Simulation Soccer 3D (2018)

2.2 RoboCup Simulation League 2d - Soccer Server

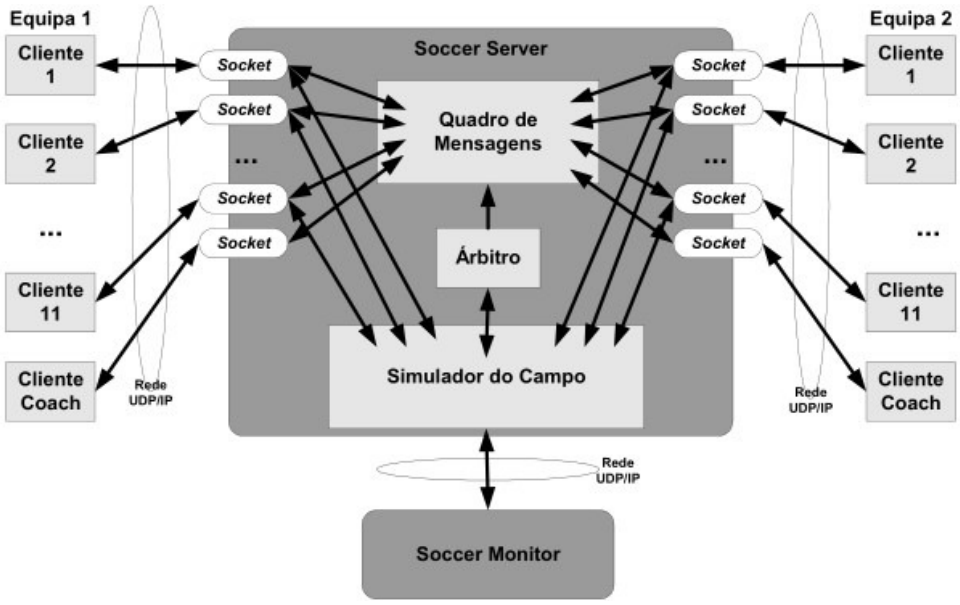
No ambiente de simulação 2d há um servidor central (*soccer server*), que concentra todas as mensagens trocadas entre todos os integrantes dos dois times, com um quadro por time, via *sockets* UDP, possibilitando que estes sejam implementado em qualquer linguagem que possua tal comunicação, ou mesmo que sejam executadas em máquinas distintas, desde que seja possível a comunicação entre elas. A Figura 7 mostra a arquitetura do *Soccer server*, em que cada cliente representa um jogador ou o técnico. O *soccer server* apresenta como uma de suas características

um tempo limite para que uma ação seja executada pelo jogador. Se este não executar uma ação no tempo determinado, o servidor executa uma ação aleatoriamente pelo jogador.

Como pode-se observar na Figura 7, cada agente, seja ele jogador ou técnico, ou o *soccer monitor*, podem comunicar-se somente por intermédio do servidor, de modo a garantir uma comunicação padronizada, auditável e que ao mesmo tempo impeça que um determinado time consiga obter uma vantagem indevida.

Um dos clientes que podem se conectar ao *soccer server* é o *soccer monitor*, sendo este um cliente para visualização da partida, ilustrado na Figura 5, conectando-se *soccer server* via *socket UDP*, de tal modo que pode-se utilizar o *soccer monitor* para visualização remota, como por exemplo, para exibir a partida em um telão. O *soccer monitor* é amplamente utilizado nos torneios para permitir que cada equipe tenha a sua própria visão da partida, visto que, ao contrário dos outros clientes, não há limitação de quantos clientes de visualização de partida podem ser conectados simultaneamente.

Figura 7 – Arquitetura do soccer server



Cada time possui onze jogadores e mais um técnico, sendo representados como agentes conectados ao servidor via *socket*, representados dos lados direitos e esquerdo. Vemos ao centro os componentes que compõe o simulador, e na parte de baixo o cliente de visualização *soccer monitor*
Fonte: (REIS, 2003)

2.3 Levantamento de times base

Nesta etapa, foi realizado um novo levantamento dos times base disponíveis disponíveis na internet, a fim de identificar possíveis melhorias que pudessem ser implementadas no time proposto.

Um time base é uma implementação básica disponibilizada gratuitamente na internet, normalmente oriunda de times profissionais, que disponibilizam parte de seus códigos para ser livremente reutilizado, desde que referenciado o autor original. Normalmente, junto com os arquivos fonte, vem a documentação dos arquivos, classes e métodos implementados, bem como a comunicação entre eles e com o servidor.

2.3.1 Helios base (Agent2D)

O time Helios, de origem japonesa, disputa a liga de simulação 2d desde 2000, tendo sido 4 vezes campeão (ROBOCUP, 2018) e 4 vezes vice campeão, sendo atualmente o segundo time mais vitorioso. E com tantos anos de desenvolvimento ativo, eles disponibilizaram parte de seus códigos, com as implementações de baixo e médio nível, de maneira gratuita, distribuído sobre a licença GNU GPL, sendo denominado de Agent2D.

O Agent2D foi desenvolvido em C++ e continua em desenvolvimento ativo. Possui ampla documentação sobre as habilidades básicas, mas bastante restrita quanto as outras áreas, principalmente na comunicação entre as classes que compõe o código do time. Sua documentação é majoritariamente em japonês, incluindo um manual de funcionamento do time, embora não muito detalhado.

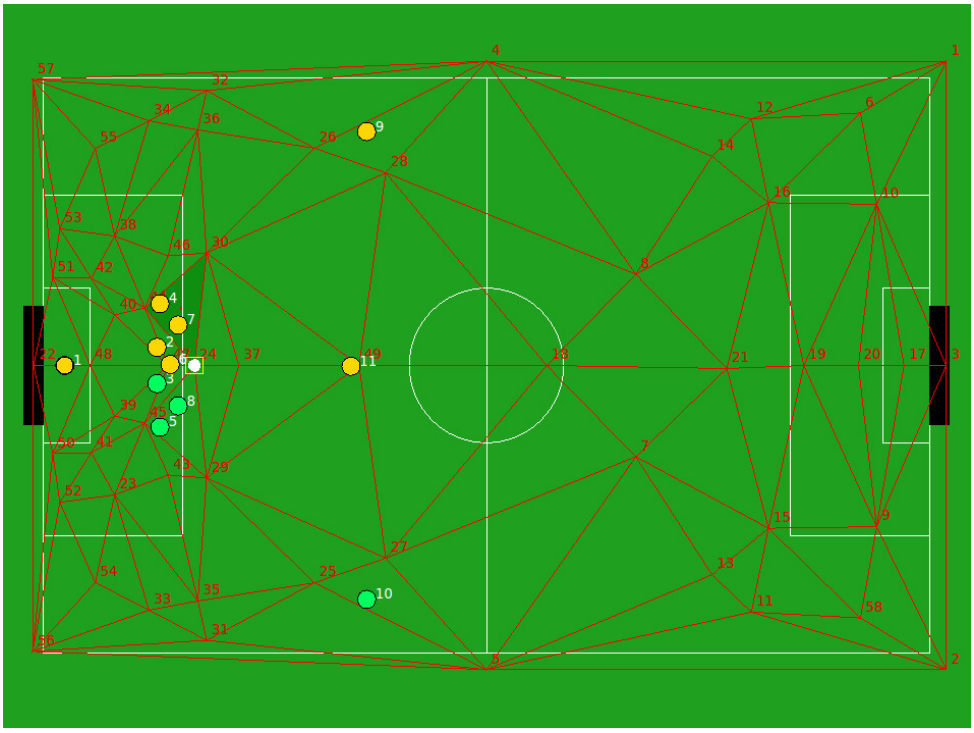
O código base já conta com alguns comportamentos básicos para os agentes, utilizando-se de uma estrutura conhecida como *Action chain*, que busca maximizar a recompensa obtida executando uma série (cadeia) de ações, modeladas como um grafo, em que cada nó é um par ação/estado, sendo a ação algo que o agente executa, como um passe ou um drible, e o estado é a previsão do estado final após a ação ser executada. Tal modelagem permite que os agentes possam prever os resultados de uma série de ações distintas (com um ou mais passos), e executar a que obtiver o melhor resultado, aumentando consideravelmente a performance do time.

O time utiliza como base a biblioteca *librcsc2*, sendo esta uma biblioteca específica para comunicação e gerenciamento no Robocup 2d, e seu código é separado em comportamento (*Behavior*) e ação (*Action*). Cada jogador possui sua própria implementação de comportamento, e esta, embora simples, é mais complexa do que a encontrada no time base UvA Trilearn (HIDEHISA, 2007). Cada jogador pode individualmente interceptar a bola, driblar, passar e chutar, de acordo com a situação corrrente do jogo. As estratégias em time continuam simples, mas a performance geral do time é relativamente boa.

Juntamente com o Agent2D, foi disponibilizado o Fedit (*Formation Editor*), ilustrado na Figura 8, sendo este um programa *open source*, que permite criar novas estratégias de modo gráfico, o que é um diferencial entre os times base, e implementá-las no time Agent2D. Tal ferramenta possibilitou que o time se tornasse extremamente popular, principalmente entre equipes iniciantes. (PROKOPENKO et al., 2013) (SILVA et al., 2010) (COSTA et al.,) (FRACCAROLI; CARLSON; COMPUTAÇAO,). Tal ferramenta, bem como o próprio time utilizam dois algo-

ritmos para obter uma formação estável sem que seja necessário definir todas as posições dos jogadores e da bola possíveis. Tais algoritmos são a triangulação de Delaunay (Abordada na subseção 2.3.1.1), utilizada para definir a malha de triângulos, e o algoritmo de sombreamento de Gouraud (Abordado na subseção 2.3.1.2), que realiza a interpolação linear das posições dos jogadores e da bola em tempo de execução.

Figura 8 – Fedit sendo executado

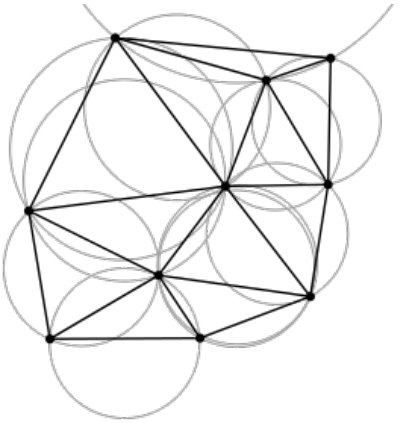


Os triângulos são representados pelas linhas vermelhas e os seus vértices são identificados pelos números vermelhos. O vértice selecionado atualmente é representado pelo quadrado amarelo. Os jogadores da metade superior do campo são representados pelos círculos amarelos, e os da metade inferior pelos círculos verdes, enquanto a bola é representada pelo círculo branco. Os jogadores possuem cores distintas para a utilização da função de espelhamento, que replica a movimentação dos jogadores da metade de cima nos jogadores de baixo. E a numeração dos jogadores está a direita de cada jogador, em branco.
Fonte: Autor (2019)

2.3.1.1 Triangulação de Delaunay

Triangulação de Delaunay é um método para triangulação de regiões planas com base em um conjunto de pontos fornecidos. A triangulação para um conjunto P de pontos no plano é uma triangulação DT(P) tal que nenhum ponto em P está dentro da circunferência de qualquer triângulo em DT (P). A triangulação de Delaunay maximiza o ângulo mínimo de todos os ângulos de todos os triângulos que compõem a triangulação, ou seja, ajusta os ângulos de todos os triângulos de modo que a sua soma seja mínima, a fim de obter a formação mais otimizada e estável. A Figura 9 mostra um exemplo de triangulação de Delaunay.

Figura 9 – Exemplo de triangulação de Delaunay



Fonte: (WIKIPEDIA, 2013)

Se são fornecidos mais de três pontos, então pode-se obter uma triangulação única. Há atualmente diversos algoritmos para calcular a triangulação, e o escolhido pelo Agent2d é algoritmo de sombreamento de Gouraud, detalhado na subseção seguinte, por ser simples o suficiente para ser executado em tempo real (levando em conta as restrições do simulador) mas ainda assim oferecendo boa precisão. Permitindo assim que um pequeno conjunto de pontos seja suficiente para definir uma formação tática em toda a extensão do campo.

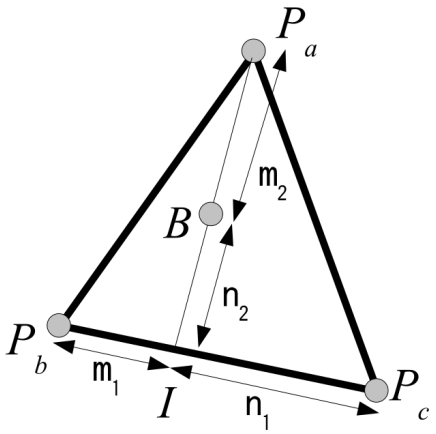
Além da facilidade da redução de pontos, o arquivo gerado é bastante simples, podendo ser facilmente lido por outro programa, o que aumenta ainda mais as possibilidades de aplicação do time.

2.3.1.2 Algoritmo de interpolação linear

O algoritmo de interpolação linear utilizado pelo time Agent2d é o mesmo algoritmo de sombreamento de Gouraud (*Gouraud shading algorithm*) (GOURAUD, 1971), sendo este amplamente utilizado no domínio da computação gráfica para cálculo de sombreamento em superfícies e objetos. O processo de funcionamento do algoritmo é ilustrado na Figura 10. O valor de saída dos vértices P_a, P_b e P_c são O(P_a), O(P_b) e O(P_c), respectivamente, e deseja-se calcular o valor de saída de B, sendo este O(B). O algoritmo funciona da seguinte forma:

1. O algoritmo calcula o ponto de intersecção (I) entre os segmentos P_bP_c e a linha P_aB.
2. O valor de saída de I, O(I), é calculado como:
   $O(I) = O(P_b) + (O(P_c) - O(P_b))\frac{m_1}{m_1 + n_1}$, onde $|\overrightarrow{P_bI}| = m_1$ e $|\overrightarrow{P_cI}| = n_1$.
3. O valor de saída de B, O(B), é calculado como:
   $O(B) = O(P_a) + (O(I) - O(P_a))\frac{m_2}{m_2 + n_2}$, onde $|\overrightarrow{P_aB}| = m_2$ e $|\overrightarrow{B_1}| = n_2$.

Figura 10 – Interpolação linear utilizando o algoritmo de sombreamento de Gouraud



(HIDEHISA, 2007)

2.3.2 WrightEagle base

O time WrightEagle, desenvolvido pela *University of Science and Technology of China*, é atualmente o maior detentor de títulos da liga simulada 2d, com 6 vitórias e 5 prêmios de segundo lugar. Disponibiliza parte de seus códigos , com o nome de WrightEagle base, a fim de contribuir com a comunidade científica. Seu código é dividido em comportamento (*Behavior*) e ação (*Action*).

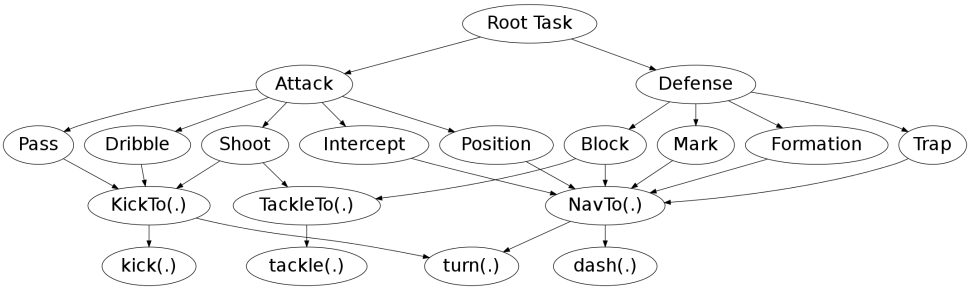
O time conseguiu modelar o ambiente do simulador como uma cadeia de Markov, mesmo sendo um problema parcialmente observável e altamente dinâmico. Cada um dos jogadores é representado por seis variáveis (ângulo da cabeça(α) e do corpo(β), posição(x,y) e velocidade(x,y)), e a bola é representada por quatro variáveis (posição(x,y) e velocidade(x,y)), o que dá um total de 136 variáveis. Se cada variável for discretizada com 10^3 valores, serão necessários 10^{408} estados, inviabilizando uma solução *offline*.

Já para a utilização de uma solução *online* puramente baseada em Markov, a restrição de tempo (100ms) acaba impactando de maneira extremamente negativa no espaço de busca, por permitir que somente uma parte do espaço de busca seja analisada. Para resolver isso, propôs-se o algoritmo MaxQ, que divide a tarefa de encontrar a solução ótima em várias subtarefas, organizadas de maneira hierárquica, com o compartilhamento de subtarefas comuns entre várias tarefas, o que permite que sejam analisadas possibilidades em menos tempo. Tendo como maior desvantagem a necessidade de se conhecer muito bem o problema no qual será aplicado o algoritmo, caso contrário, as subtarefas não irão convergir para um resultado próximo ao ótimo em tempo de execução. E tal feito não era eficiente quando o problema apresentava um ambiente dinâmico.

Para solucionar este problema de domínio da aplicação alvo, surgiu o MaxQ-OP (BAI; WU; CHEN, 2015) , utilizando as vantagens do MaxQ para planejamento online, apresentando como principal vantagem a decomposição da cadeia de Markov em subtarefas hierárquicas sem a necessidade de definir as subtarefas manualmente, o que torna o algoritmo muito eficiente para

aplicação em tempo de execução. E tal algoritmo foi aplicado no time WrightEagle, despontando como um dos melhores times da atualidade. A Figura 11 ilustra a divisão da cadeia de Markov em subtarefas aplicadas ao time.

Figura 11 – Estrutura de tarefas do time WrightEagle



***kick*, *turn*, *dash* e *tackle* são as habilidades primitivas, definidas pelo simulador. *kickTo*, *TackleTo* e *NavTo* são subtarefas de baixo nível definidas a partir das habilidades primitivas. *Shoot*, *dribble*, *Pass*, *Position*, *Intercept*, *Block*, *Trap*, *Mark* e *Formation* são as subtarefas de alto nível consistindo de uma ou mais subtarefas de baixo nível. *Attack* e *Defense* são as subtarefas que determinam se o objetivo é uma tática ofensiva e defensiva. *Root task* é a tarefa principal, sendo esta programada manualmente, e que em tempo de execução determina se o jogador deve tentar uma jogada ofensiva ou defensiva.**
Fonte: (BAI; WU; CHEN, 2015)

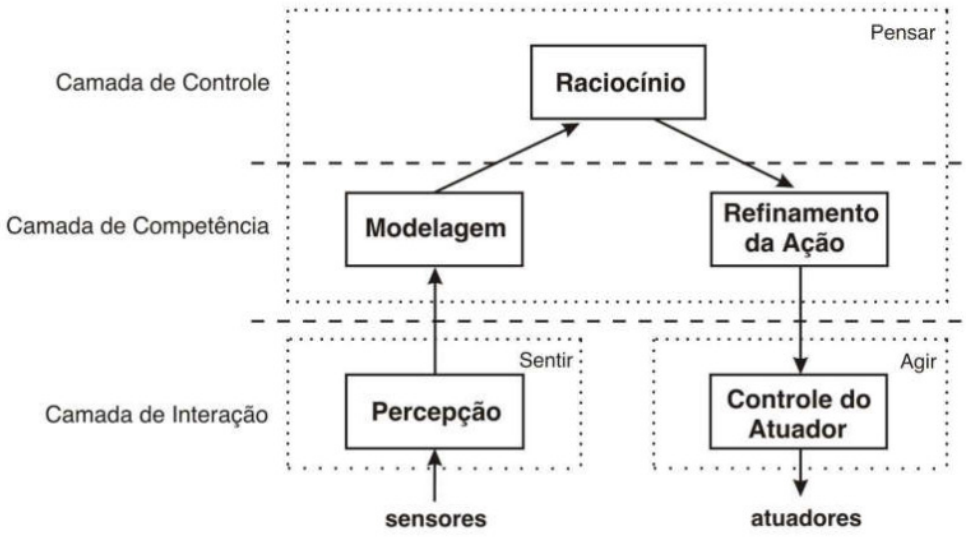
2.3.3 UvA Trilearn base

O time UvA Trilearn (BOER; KOK, 2002), da Universidade de Amsterdã, na Holanda, disponibilizou em 2001 parte do seu código-fonte, sob o nome de UvA Trilearn base, sofrendo alterações em 2002 e algumas correções de compatibilidade em 2003. Possui extensa documentação, que abrange a maior parte, se não todo, do código fonte, que é escrito em C++. O código é dividido em três camadas: camada de iteração, camada de competência e camada de controle, como ilustrado na Figura 12.

A camada de interação é responsável pela interação com ambiente de simulação do servidor, com a entrada de dados no sistema através dos sensores dos jogadores e a saída (ação executada) em cada agente (jogador). A camada de competência utiliza os dados obtidos da camada de interação para construir um modelo abstrato do mundo e implementar as habilidades de cada agente individualmente. A camada de controle contém o raciocínio do sistema, sendo a responsável por escolher a melhor estratégia para o momento corrente. Esta última camada é ausente no time base, a fim de que novos times possam implementar suas próprias estratégias sem precisar se preocupar com as implementações de baixo nível.

O time apresenta como maior contribuição sua arquitetura de três camadas altamente flexível, sendo esta capaz de realizar processamento paralelo, com métodos eficazes para a estimação de velocidade presente e futura de um determinado elemento. Possui também três formações táticas básicas, sendo elas (3-4-3), (4-4-2) e (4-3-3), utilizando a formação (4-3-3) por padrão, por se tratar de uma formação equilibrada e extremamente versátil.

Figura 12 – Arquitetura do time UvA Trilearn



O time apresenta uma arquitetura de três camadas, capaz de realizar processamento paralelo. A informação é recebida dos sensores na camada de interação, sendo tratada e repassada à camada superior, que processa as informações e repassa á camada de controle para a tomada de decisão. Esta, por sua vez, encaminha as decisões tomadas à segunda camada, para o refinamento da ação, que envia à camada de interação para que os comandos sejam executados pelos atuadores do robô. Fonte: (BOER; KOK, 2002)

2.3.4 BahiaRT (bahia2d)

O time BahiaRT é desenvolvido em C++, tendo como base o time UvA Trilearn, com parte do seu código fonte disponibilizado. Possui tanto métodos herdados do UvA Trilearn quanto métodos próprios. Apresenta como diferencial métodos para tratamento de dados no nível reativo (equivalente à camada de interação do UvA Trilearn), a fim de que os dados já sejam convertidos em informação sem a necessidade de que o desenvolvedor precise tratá-los manualmente, gerando uma economia considerável de tempo.

No nível instintivo (equivalente à camada de competência do UvA Trilearn) os dados são recebidos e analisados, a fim de realizar a melhor ação, enquanto no UvA Trilearn base as ações são tomadas de maneira aleatória. Já o nível cognitivo (equivalente à camada de controle do UvA Trilearn) é responsável por tomar decisões elaboradas, com base nas informações recebidas das camadas inferiores, tentando prever as jogadas do time adversário e utilizar a estratégia mais eficiente tomando como parâmetros todos os fatores supracitados.

2.4 Histórico do UFMA2D

O time UFMA2D, nomeado aqui para UFMA2D 2017, foi implementado em Gomes (2017), devido à baixa participação de equipes nordestinas em competições da RoboCup, com somente a participação da equipe baiana BahiaRT. Diante dessa constatação, iniciou-se um estudo a fim de levantar e analisar os times base existentes, sob vários aspectos, principalmente

sobre a documentação e a facilidade de programação do time.

Como resultado, escolheu-se o time base UvA Trilearn, por apresentar uma sólida documentação e código altamente modular, sendo totalmente escrito em C++, o que facilitou bastante a implementação inicial do time.

Para tal, realizou-se um estudo sobre o código fonte, a fim de entender como as classes se comunicavam e como o time funcionava do ponto de vista da programação. Além do desafio de entender como o time funcionava, havia também o desafio de fazer o código fonte, relativamente desatualizado, executar em uma versão recente do simulador. Para tal, contou-se com a documentação em português e código fonte atualizado (modificado a fim de ser compatível com versões mais novas do simulador, mas mantendo todas as características originais) do PET Computação[1] da Universidade Federal do Espírito Santo.

Com o time funcional e com relativo conhecimento sobre o time, optou-se por implementar 3 estratégias básicas: chute ao gol, chute ao gol de acordo com o ciclo da partida e drible. A primeira estratégia consiste em chutar ao gol assim que o jogador atinge uma certa proximidade do gol. A segunda estratégia difere da primeira por chutar no canto esquerdo do gol se o ciclo for par e no canto direito se o ciclo for ímpar, a fim de criar alguma variabilidade. E a terceira estratégia consiste em percorrer o campo driblando com a bola até chegar a uma distância relativa do gol, utilizando a segunda estratégia para realizar o chute.

[1] O site, a documentação e o time base não estão mais disponíveis online desde 2017.

3 METODOLOGIA

Este trabalho visa realizar a atualização do time base utilizado no time de futebol robótico UFMA2D (GOMES, 2017) e a implementação de estratégias reais utilizando a ferramenta gráfica de edição de formação *Fedit*, bem como fornecer um panorama mais abrangente sobre os times base disponíveis atualmente. Para tal, foram realizados diversos testes, com análise e discussão dos resultados obtidos. Todos os testes foram executados utilizando a plataforma de simulação oferecida pela RoboCup, a fim de garantir a lisura e a reprodutibilidade dos resultados.

Esta nova etapa foi dividida da seguinte forma:

1. Atualização de *scripts* de configuração e execução, a fim de adequar-se às novas versões do sistema operacional Ubuntu;
2. Atualização de máquina virtual do simulador, a fim de corrigir problemas existentes;
3. Realizar uma revisão bibliográfica ampliada sobre times base;
4. Escolha de um novo time base para o time UFMA2D, a fim de melhorar o desempenho do time;
5. Implementação de estratégias reais no time UFMA2D
6. Discussão dos resultados obtidos.

3.1 Escolha de um novo time base para o time de futebol robótico UFMA2D

Em 2018 o time sofreu algumas atualizações, nomeado aqui para UFMA2D 2018, com a adição de cinco novas formações e um algoritmo de decisão baseado em testes condicionais, tomando como critério o tempo da partida, a fim de melhorar a performance do time, tanto pela maior variabilidade de estratégias do time, quanto pela diminuição das situações em que o time ficava travado e executava alguma ação aleatoriamente. Tal abordagem mostrou uma ligeira evolução sobre a implementação original, melhorando discretamente a capacidade de finalização do time, mas ainda sendo bastante deficiente quando em confronto com um time dotado de tomada de decisão inteligente, por não conseguir variar suas estratégias além do que fora programado.

Diante da pouca melhora com a adição de novas estratégias no time e do estudo mais detalhado dos times bases existentes, constatou-se que o time Helios base (Agent2d) apresenta uma interface mais simples para a programação de estratégias, sendo esta o *Fedit*, enquanto que os outros times não possuem quaisquer meio gráfico disponível para a edição de formações e/ou estratégias.

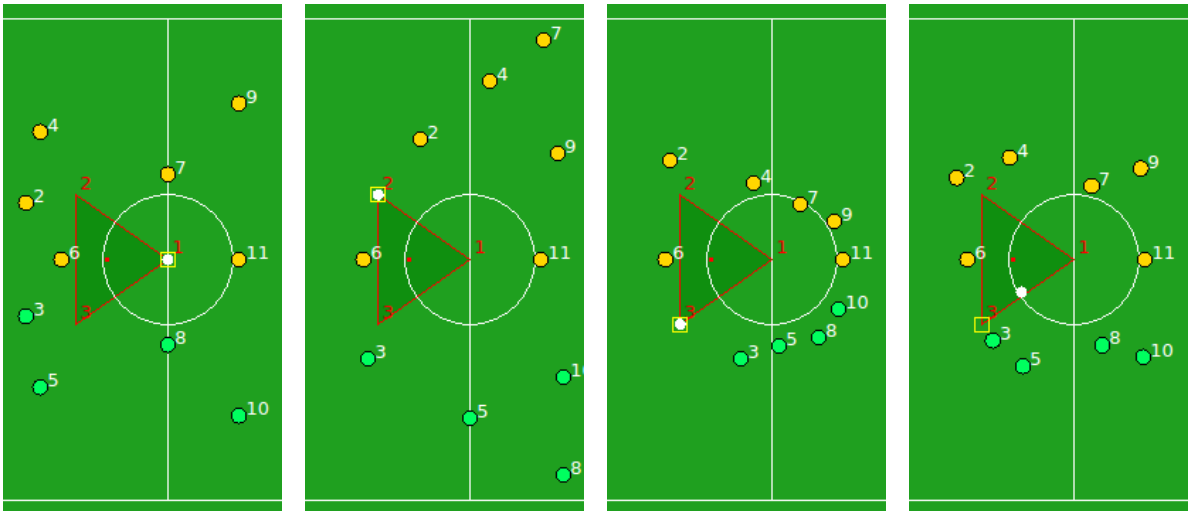
O *Fedit* é distribuído junto com o Agent2d, tendo sido criado para facilitar o desenvolvimento de novas formações de maneira gráfica, algo até então não existente em um time base e o único até hoje. A ideia relativamente simples pareceu bastante interessante no começo, mas revelou um grande problema: como armazenar a formação gerada e passar isso de uma maneira simples ao time. A resposta foi a utilização da triangulação de Delaunay. Na abordagem proposta pelo time, ao invés de armazenar todas as posições possíveis da bola e de todos os jogadores, são armazenadas as posições da bola e dos jogadores em pontos pré-definidos pelo programador, como mostram as Figuras 13 e 14.

Figura 13 – Exemplo de triangulação no *Fedit*

Na parte esquerda estão a lista de posições da bola (*index*) definidas pelo programador, sendo representados no campo como os vértices numerados, e e cada posição contém a posição de cada um dos onze jogadores do time.
Fonte: Autor (2019)

Na Figura 13, na parte esquerda da tela, há as posições da bola definidas pelo programador, sendo três neste caso. Cada posição do *index* representa uma posição da bola, e cada posição da bola contém a posição de cada um dos onze jogadores do time. Na parte direita da tela está representado o campo com as respectivas marcações, a bola e todos os jogadores, além das posições do *index* definidas pelo programador.

Figura 14 – Exemplo ilustrativo de triangulação no *Fedit*



(a) Primeiro vértice **(b) Segundo vértice** **(c) Terceiro vértice** **(d) Interpolação dos vértices 1 e 3**

Em (a) temos as posições da bola e dos jogadores para o primeiro vértice, em (b) as posições para o segundo vértice, em (c) as posições para o terceiro vértice, e em (d) uma posição intermediária entre o primeiro e o terceiro vértices, interpolados utilizando a triangulação de Delaunay.
Fonte: Autor (2019)

Na Figura 14 estão representados três vértices (posições do *index*) distintos, em que cada vértice representa a posição da bola, e cada posição da bola contém o posicionamento de todos os jogadores naquela situação em específico. Na Figura 14a a bola está no primeiro vértice, com uma formação (5-2-3), na Figura 14b a bola está no segundo vértice, com uma formação (2-1-2-5), na Figura 14c a bola está no terceiro vértice, com uma formação (2-2-1-2-3), e na Figura 14d a bola está em uma posição intermediária entre os vértices 1 e 3, interpolados utilizando a triangulação de Delaunay.

A triangulação de Delaunay pode ser utilizada para interpolar qualquer ponto intermediário definido no triângulo. No exemplo citado, foram utilizados os vértices 1 e 3 para a obtenção do novo ponto, por ser um ponto contido na aresta que liga os dois vértices, com a contribuição do vértice 2 sendo nula neste caso, mas poderiam ser utilizados os vértices 1 e 2, ou qualquer outro ponto intermediário entre os três vértices, desde que este esteja contido dentro do triângulo, incluindo o ponto central do triângulo.

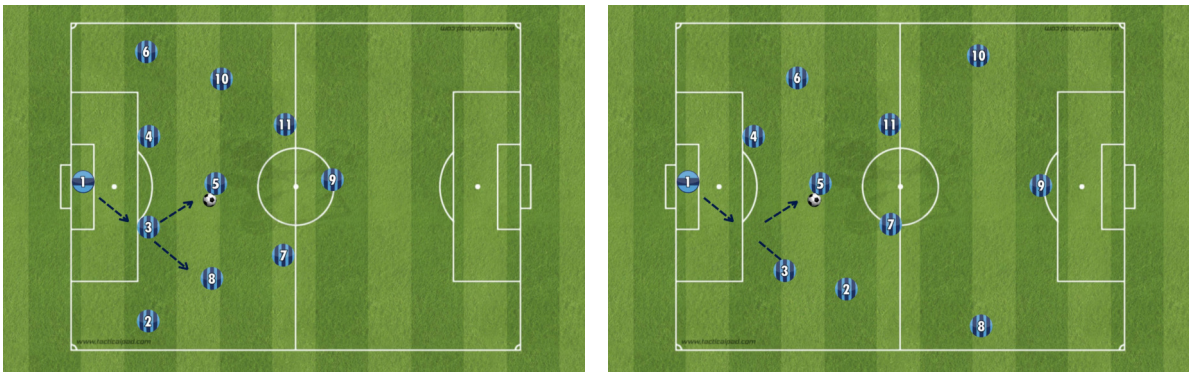
4 IMPLEMENTAÇÃO DE ESTRATÉGIAS REAIS NO TIME UFMA2D

As estratégias (4-3-2-1) e (4-3-3) foram implementadas tendo o time Agent2d como base, a fim de aumentar a competitividade do time, além de testar quais tipos de formações conseguem ser mais eficientes em um cenário genérico. Para tal, contou-se com a colaboração de um professor do Departamento de Educação Física [1] da Universidade Federal do Maranhão. Optou-se por estratégias de bola parada, por ser até então a maior deficiência do time.

4.1 Estratégias reais

A estratégia sugerida consiste em duas estratégias de saída de bola distintas,com uma e duas variações, respectivamente, constituindo assim um total de cinco estratégias, sendo representadas nas Figuras 15, 16, 17, 18 e 19. São representados dois estados de cada estratégia, inicial e final, a fim de demonstrar de maneira mais clara variação da formação conforme o posicionamento da bola.

Figura 15 – Primeira estratégia



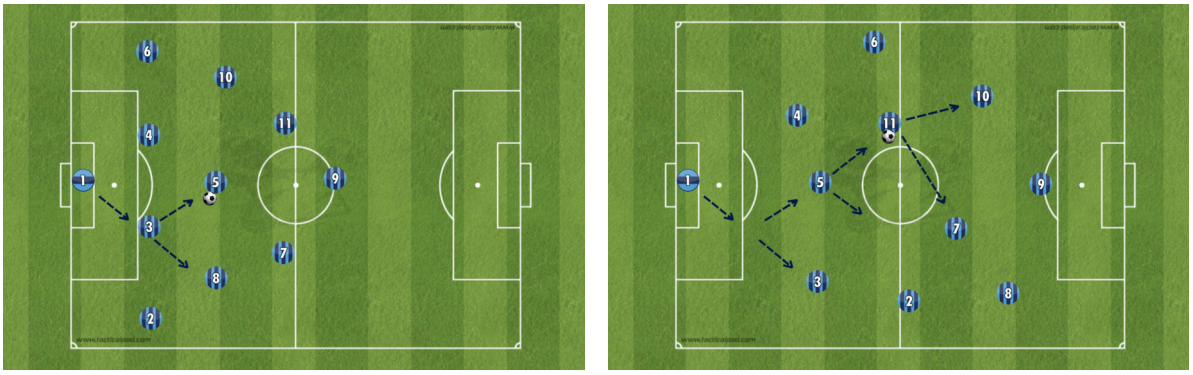
(a) Estratégia 1 inicial **(b) Estratégia 1 final**

Os círculos azulados representam os jogadores e as setas tracejadas pretas representam as possibilidades de passe. Formação inicial (a) (4-3-2-1) e formação final (b) fechada, mantendo a triangulação, com pelo menos três jogadores próximos.
Fonte: Autor (2019)

A primeira estratégia conta com formação inicial (4-3-2-1) (Figura 15a), sendo esta uma formação equilibrada com foco na defesa, e a medida que o time avança a formação é alterada conforme a posição da bola, de modo a manter sempre pelo menos três jogadores próximos (pelo menos dois destes em condição de ataque), preferencialmente mantendo a triangulação, e com formação final fechada (Figura 15b). A variação da estratégia (Figura 16a) consiste na modificação da formação final, que fica aberta (Figura 16b), para dificultar a marcação pelo time adversário.

[1] Prof. Dr. Antônio Coppi Navarro

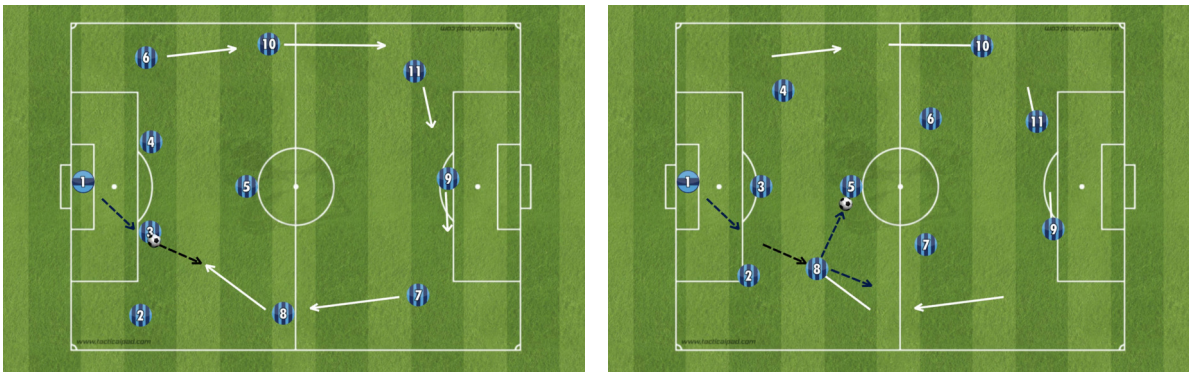
Figura 16 – Variação da primeira estratégia



(a) Estratégia 1 variação 1 inicial **(b) Estratégia 1 variação 1 final**

Os círculos azulados representam os jogadores e as setas tracejadas pretas representam as possibilidades de passe. Formação inicial (a) (4-3-2-1) e formação final (b) aberta, com jogadores de ataque mais avançados que na estratégia original, mantendo a triangulação, com pelo menos três jogadores próximos.
Fonte: Autor (2019)

Figura 17 – Segunda estratégia

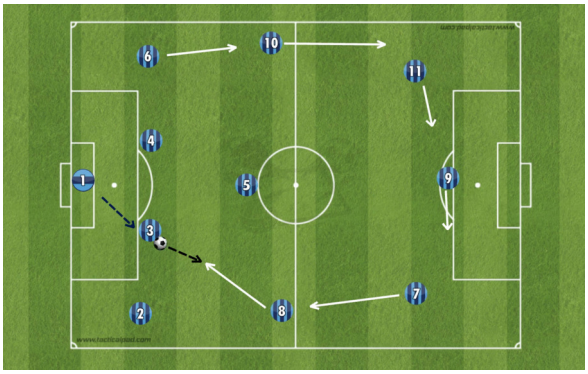


(a) Estratégia 2 inicial **(b) Estratégia 2 final**

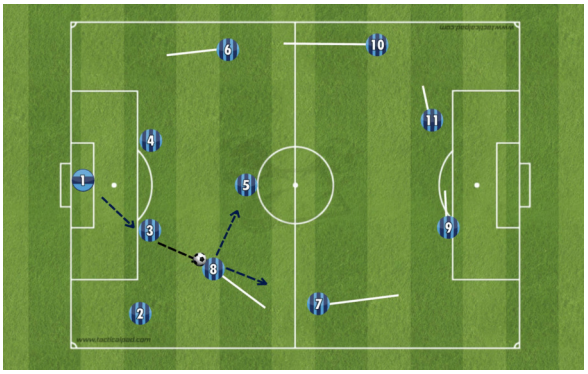
Os círculos azulados representam os jogadores, as setas tracejadas pretas representam as possibilidades de passe e as setas brancas representam a movimentação do jogador. Formação inicial (a) (4-3-3) aberta, com rotação dos jogadores no sentido horário, como observado em (b).
Fonte: Autor (2019)

A segunda estratégia conta com formação inicial (4-3-3) aberta (Figura 17a), realizando a rotação dos jogadores a fim de dificultar a marcação pelo time adversário. Busca sempre manter a posição dos jogadores de defesa, para impedir uma reação súbita do time adversário. Busca também manter a triangulação, a fim de garantir que sempre há pelo menos duas opções de passe (Figura 17b). A primeira variação da estratégia (Figura 18a) consiste em uma variação de passe, com formação final mais aberta (Figura 18b) e a segunda variação (19a) consiste em adiantar os jogadores para aumentar a probabilidade de uma jogada ofensiva (Figura 19b).

Figura 18 – Primeira variação da segunda estratégia



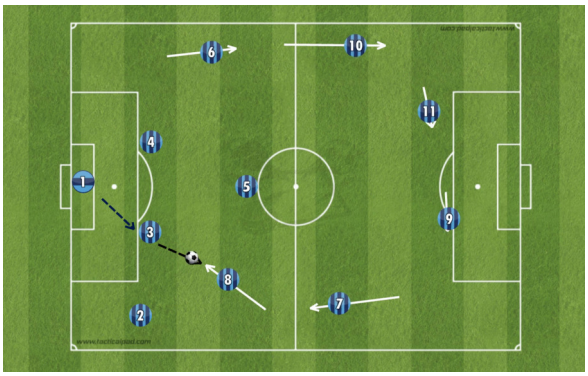
(a) Estratégia 2 variação 1 inicial



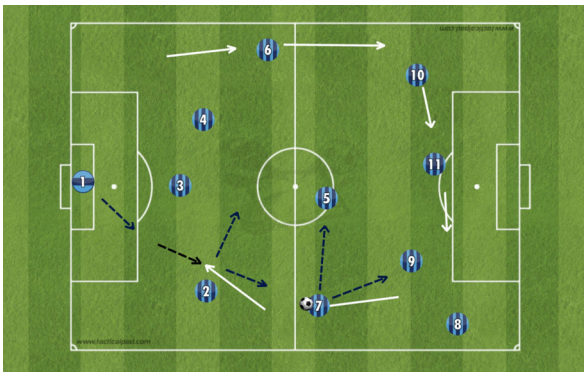
(b) Estratégia 2 variação 1 final

Os círculos azulados representam os jogadores, as setas tracejadas pretas representam as possibilidades de passe e as setas brancas representam a movimentação do jogador. Formação inicial (a) (4-3-3) aberta, com rotação dos jogadores no sentido horário e formação final (b) mais aberta que a estratégia original.
Fonte: Autor (2019)

Figura 19 – Segunda variação da segunda estratégia



(a) Estratégia 2 variação 2 inicial



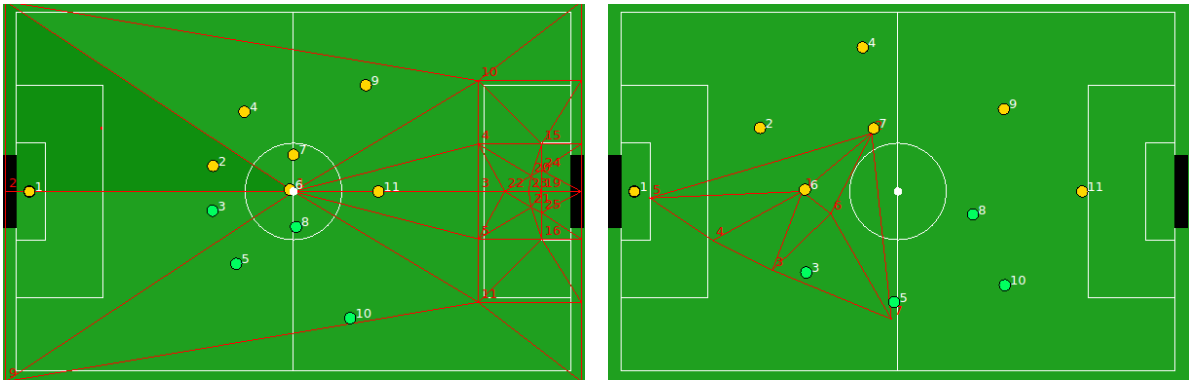
(b) Estratégia 2 variação 2 final

Os círculos azulados representam os jogadores, as setas tracejadas pretas representam as possibilidades de passe e as setas brancas representam a movimentação do jogador. Formação inicial (a) (4-3-3) aberta, com rotação dos jogadores no sentido horário e formação final (b) mais avançada.
Fonte: Autor (2019)

4.2 Implementação das estratégias no time UFMA2D

Para a implementação das estratégias propostas, criou-se um modelo equivalente utilizando o *Fedit*. Como todas as estratégias envolvem jogadas distintas, optou-se pela criação de um único modelo que contivesse todas as estratégias, de tal modo que o time possa escolher qual delas utilizar dependendo da posição atual da bola. Tal estratégia permite que em situações semelhantes ações distintas sejam tomadas, o que é bastante eficaz para reduzir o poder ofensivo do time adversário. A Figura 20 mostra o comparativo entre a implementação original do Agent2d e a implementação proposta. O arquivo de configuração da implementação encontra-se no anexo.

Figura 20 – Estratégia implementada



(a) Estratégia original **(b) Estratégia modificada**

Na implementação original (a), embora hajam muito mais pontos, estes são muito distantes no campo defensivo, fazendo com que os jogadores se movam de forma bastante genérica quando a bola está próxima ao próprio gol, utilizando esta estratégia. Já na implementação proposta (b), os pontos são próximos no campo defensivo, e como cada ponto possui o posicionamento de cada um dos jogadores, estes conseguem se posicionar de maneira estratégica de acordo com a posição da bola, quando a bola está no campo defensivo, utilizando esta estratégia.
Fonte: Autor (2019)

O arquivo modificado foi o *src/indirect-freekick-our-formation.conf*, sendo o responsável pelas táticas de saída de bola a partir do goleiro. Embora a implementação original (Figura 20a) tenha muito mais pontos, eles são muito distantes no campo defensivo, o que significa que a interpolação realizada devido à triangulação acaba não sendo muito precisa, ao mesmo tempo em que os jogadores não conseguem se posicionar estrategicamente, o que acaba aumentado consideravelmente as chances de um contra ataque efetivo.

Já na implementação proposta (Figura 20b), focou-se em transformar a saída de bola a partir do goleiro em uma situação de marcação eficiente e bastante dinâmica, aumentando as chances de que um jogador esteja livre no ataque ao mesmo tempo em que mantém a defesa bastante equilibrada, aumentando a chance de conversão de uma situação de perigo em uma situação bastante vantajosa para o time.

Devido à natureza dinâmica do time, esta situação pode ser ativada em outras situações de bola parada que não sejam exclusivamente saída de bola a partir do goleiro. O time calcula de forma autônoma qual ação ou conjunto de ações causa efeito mais benéfico ao time. Tomemos como exemplo ilustrativo uma situação em que o goleiro seja o jogador mais próximo para realizar uma saída de bola parada. Por proximidade, o goleiro deveria executar a ação, mas por ser uma solução óbvia, resultaria em uma forte marcação do time adversário sobre o goleiro.

Em tempo de execução e de maneira autônoma, o time pode escolher um jogador da defesa ou do meio de campo, que pode executar a ação mesmo com a bola relativamente próxima ao goleiro, por ser muito difícil ou arriscado o goleiro executar a ação, normalmente devido à forte marcação adversária nas proximidades do gol. E embora bastante simples, esta tomada

de decisão se mostra bastante eficiente por conseguir evitar decisões obvias quando necessário, confundindo o time adversário. Conseguindo não só eliminar o risco iminente de um gol, como também diminuir consideravelmente a marcação do time adversário por um curto período de tempo, visto que este precisa de um certo tempo para reposicionar seus jogadores.

5 RESULTADOS OBTIDOS

Para atestar a eficácia da nova implementação do time (UFMA2D 2019) consistindo do time base Agent2d conjuntamente com as estratégias reais implementadas no time, realizaram-se quatro etapas de teste, sendo a primeira contra a primeira implementação do time (UFMA2D 2017) (GOMES, 2017), a segunda contra a segunda implementação do time (UFMA2D 2018), a terceira contra o time base Agent2d e a quarta contra o time Helios2018, atual campeão da RoboCup e responsável pelo time base Agent2d, no qual o UFMA2D é atualmente baseado.

A primeira implementação do time tomou como base o time base UvATrilearn, com a adição da estratégia de passe e uma estratégia simples de chute ao gol. A segunda implementação adicionou cinco novas formações táticas, sendo estas combinadas com as estratégias já existentes a fim de aumentar a variabilidade das jogadas.

Os testes contra as implementações anteriores do time, ambas baseadas no time base UvATrilearn, se mostraram necessários para demonstrar de maneira clara a evolução do time desde a sua concepção, e os testes contra o time Agent2d e contra o Helios2018 para demonstrar que apesar de simples, as modificações realizadas são suficientes para melhorar o desempenho do time.

Todos os testes foram executados em uma máquina rodando Linux mint 18.3, que é baseado no Ubuntu 16.04LTS, sendo esta a versão mais recente suportada pelo simulador da RoboCup. Foram realizados 10 jogos em cada uma das etapas. E para fins de melhor legibilidade das tabelas, os times foram renomeados: UFMA2D 2017 como UFMA2017, UFMA2D 2018 como UFMA2018 e UFMA2D 2019 como UFMA2D.

5.1 Resultados contra a primeira implementação do time

Tabela 1 – Resultados do UFMA2D 2019 (UFMA2D) contra UFMA2D 2017 (UFMA2017)

	Gols			Cartões amarelo	
Partida	UFMA2D	UFMA2017	Vencedor	UFMA2D	UFMA2017
1	15	0	UFMA2D	3	0
2	11	0	UFMA2D	2	0
3	24	0	UFMA2D	2	0
4	25	0	UFMA2D	1	0
5	21	0	UFMA2D	0	0
6	18	0	UFMA2D	1	0
7	17	0	UFMA2D	3	0
8	16	0	UFMA2D	2	0
9	17	0	UFMA2D	2	0
10	20	0	UFMA2D	0	0
Média	18.4	0	100% UFMA2D	1.6	0

Nos testes contra a primeira implementação (UFMA2D 2017), representados na Tabela 1, que apresentava apenas comportamentos bem básicos, com uma considerável aleatoriedade, o time se mostrou muito eficiente. No princípio o time proposto apresentou certa dificuldade, mas conseguiu decodificar as estratégias utilizadas pelo time adversário e desenvolveu algumas séries de jogadas que resultavam quase sempre em gol. E após um certo tempo de partida, o time proposto conseguia fazer gols em sequencia [1] utilizando basicamente as mesmas jogadas, por ter identificado as falhas do time adversário.

Tal comportamento do time proposto se deve à abordagem *Action chain* implementada no próprio time base Agent2d, o qual lhe serviu de base. O time consegue calcular o impacto de suas decisões em tempo de execução, bem como prever com certo grau de precisão as ações do time adversário. Isto permite que o time proposto consiga decodificar o estilo de jogo do time adversário e desenvolver técnicas efetivas especificamente para o adversário.

O time proposto também apresentou uma considerável taxa de cartões amarelo, por prever jogadas e se adiantar, mas o time adversário apresentou demora em executar a ação prevista, ficando simplesmente parado, sem executar nenhuma ação. Isto causou muitas situações em que o simulador considerou como infração, por um ou mais jogadores do time proposto bloqueando o jogador do time adversário, principalmente quando o jogador adversário que sofre o bloqueio está com a bola.

Observou-se também que o time adquiriu padrões de jogo distintos entre as partidas, modificando o padrão também quando o adversário apresentava uma condição considerada ofensiva, procurando sempre estabelecer uma vantagem sobre o adversário. Tal comportamento evidencia que o algoritmo de tomada de decisão é capaz de identificar condições de perigo e buscar um novo ponto de equilíbrio em tempo de execução, mas ao mesmo tempo é incapaz de identificar quando o time adversário trava, executando as ações previstas e muitas vezes sendo punido, quando o mais recomendável seria ignorar a previsão e posicionar os jogadores de maneira estratégica ou simplesmente esperar o adversário destravar e executar a ação.

[1] Uma das partidas está disponível em <https://www.youtube.com/watch?v=Rlm52Yo_jug>

5.2 Resultados contra a segunda implementação do time

Tabela 2 – Resultados do UFMA2D 2019 (UFMA2D) contra UFMA2D 2018 (UFMA2018)

	Gols			Cartões amarelo	
Partida	UFMA2D	UFMA2018	Vencedor	UFMA2D	UFMA2018
1	32	0	UFMA2D	1	0
2	33	0	UFMA2D	0	0
3	31	0	UFMA2D	2	0
4	30	0	UFMA2D	1	0
5	31	0	UFMA2D	1	0
6	33	0	UFMA2D	1	0
7	35	0	UFMA2D	1	0
8	34	0	UFMA2D	0	0
9	27	0	UFMA2D	0	0
10	37	0	UFMA2D	2	0
Média	32.3	0	100% UFMA2D	0.9	0

Já nos testes contra a segunda implementação (UFMA2D 2018), a diferença de pontuação foi ainda maior, como pode ser observado na Tabela 2, mas ainda sofrendo com penalidades em situações que o time adversário demorou mais do que o previsto para executar uma ação [2]. O menor desempenho do UFMA2018 quando comparado ao UFMA2016 se deve à menor aleatoriedade do UFMA2D 2018, uma vez que tenta sempre agir de acordo com algumas estratégias pré definidas. O fator aleatoriedade acaba de certo modo beneficiando o time quando jogando contra um time com tomada de decisão inteligente, por este não conseguir prever jogadas fora do padrão. E como a segunda implementação (UFMA2D 2018) possui mais situações de jogo cobertas pela programação, isto é, utiliza menos o fator aleatoriedade que a primeira implementação (UFMA2D 2017), a segunda implementação acaba ficando em desvantagem, por contar com estratégias simples que não conseguem se adaptar em tempo de execução.

Devido a maior quantidade de estratégias da segunda implementação implicou em um maior número de situações cobertas pela programação, resultando em menos situações onde o time executou uma ação aleatória ou simplesmente não executou ação alguma. Isto pode ser facilmente observado pela expressiva diminuição do número de cartões amarelos recebidos pelo time proposto, quando comparado os testes da primeira e os da segunda implementação.

Além da redução dos cartões amarelo recebidos, o time proposto se beneficiou da variação de estratégia da segunda implementação (UFMA2D 2018), por não ser um mecanismo de tomada de decisão inteligente, aprendendo todos os padrões utilizados pelo time, devido ao *Action chain*, e desenvolvendo uma estratégia altamente eficiente para cada um. Tal identificação de padrões diminuiu consideravelmente a modificação dos padrões, uma vez que estes já incluíam as modificações possíveis, basicamente repetindo padrões até o fim da partida.

[2] Uma das partidas está disponível em <https://www.youtube.com/watch?v=3afvp4VI-S0>

Mas apesar do aprendizado, como o time adversário apresentava comportamentos distintos entre as partidas (por não apresentar todos os comportamentos possíveis em uma única partida), eram geradas estratégias distintas pelo time proposto, ainda que o time adversário estivesse em uma formação semelhante a de uma partida anterior. Tal fenômeno tem grande importância para evitar um super ajuste ao time adversário, o que poderia prejudicar de maneira significativa o sistema de previsão de jogadas.

5.3 Resultados contra o time Agent2d

Tabela 3 – Resultados do UFMA2D 2019 (UFMA2D) contra Agent2d

	Gols			Cartões amarelo	
Partida	UFMA2D	Agent2d	Vencedor	UFMA2D	Agent2d
1	2	4	Agent2d	0	0
2	3	2	UFMA2D	0	0
3	6	4	UFMA2D	0	0
4	1	2	Agent2d	0	0
5	3	3	UFMA2D	0	0
6	2	1	UFMA2D	0	0
7	5	6	Agent2d	0	0
8	1	0	UFMA2D	0	0
9	5	2	UFMA2D	0	0
10	3	1	UFMA2D	0	0
Média	3.1	2.5	70% UFMA2D	0	0

Na Tabela 3 temos os resultados do time proposto contra o time que lhe deu base, o Agent2d. Como esperado, os placares foram bastante apertados, mostrando uma ligeira vantagem para o time proposto sobre o time base. Como o Agent2d não fica "travado", o time UFMA2D não sofreu nenhuma punição grave durante todos os jogos nessa etapa de teste. Observou-se um jogo com intensa marcação e poucas oportunidades de gol para ambos os lados. Mas a grande diferença se mostrou pelo posicionamento do time quando a bola estava nas proximidades do gol e a estratégia de bola parada era ativada por alguma penalidade do time adversário. O time proposto apresentou uma melhor distribuição dos jogadores no campo, oferecendo um comportamento bastante ofensivo ao mesmo tempo em que conseguia manter a defesa bem protegida [3].

Tal comportamento permitiu explorar uma fraqueza do time original, em que o time ficava muito disperso na formação de bola parada, com somente os jogadores defensivos a uma curta distância do gol, conforme a Figura 20a. Tirando proveito deste fato, e com a estratégia proposta, criou-se uma condição bastante favorável ao time proposto durante a saída de bola parada, permitindo ao time proposto posicionar estrategicamente os seus jogadores, enquanto implementação do Agent2d continuava em uma formação genérica e bastante dispersa.

[3] Uma das partidas está disponível em <https://www.youtube.com/watch?v=wjnioK41iD4>

5.4 Resultados contra o time Helios2018 (atual campeão da RoboCup)

Tabela 4 – Resultados do UFMA2D 2019 (UFMA2D) contra helios2018

	Gols			Cartões amarelo	
Partida	UFMA2d	Helios2018	Vencedor	UFMA2d	Helios2018
1	0	11	Helios2018	1	0
2	0	10	Helios2018	1	0
3	0	7	Helios2018	1	3
4	0	16	Helios2018	0	1
5	0	9	Helios2018	0	2
6	0	18	Helios2018	0	1
7	0	11	Helios2018	0	0
8	0	11	Helios2018	1	2
9	0	12	Helios2018	1	2
10	1	9	Helios2018	1	0
Média	0.1	11.4	100% Helios2018	0.6	1.1

Na Tabela 4 temos os resultados contra o time Helios2018. Neste caso, aconteceu algo semelhante ao que aconteceu nos testes da implementação proposta contra as implementações anteriores, em que o time inteligente previu ações, mas o time sem tomada de decisão em tempo real simplesmente não realizou nenhuma ação. Nos casos anteriores o UFMA2D era o time inteligente e as implementações anteriores o time sem tomada de decisão. Já neste caso, o Helios2018 é o time inteligente e o UFMA2D é o time sem tomada de decisão, quando comparado com o adversário.

Houve o mesmo problema com uma quantidade excessiva de cartões amarelo, normalmente em situações que o time de menor capacidade, o UFMA2D neste caso, não executou ação alguma enquanto o adversário, o time Helios2018 neste caso, previu uma ação, adiantando os jogadores para que esta não se efetivasse. Mas como o UFMA2D não realizou a ação, o Helios2018 acabou sendo penalizado em muitas situações, inclusive com cartões amarelo.

6 CONCLUSÃO

O presente trabalho demonstrou uma significativa evolução sobre as implementações anteriores do time, não só pela troca do time base, mas também pela implementação de estratégias reais no time, o que serviu como um excelente ponto de partida nesta nova fase do time. Os resultados obtidos foram bons, superando com ampla vantagem as implementações anteriores, e até mesmo o próprio time base Agent2d.

No caso dos resultados contra as implementações anteriores, a expressiva diferença se deve principalmente ao sistema de tomada de decisão em tempo real, componente ausente no UvA Trilearn (no qual foram baseadas as implementações anteriores do time), que além de identificar jogadas ótimas baseadas na situação corrente do jogo, foi bastante eficiente em identificar os jogadores e formações do time adversário, ainda que esse processo demorasse um tempo.

Percebeu-se uma diferença entre antes e depois da identificação das estratégias do time adversário. Antes, o time apresentou um comportamento genérico e incerto, executando ações que muitas vezes resultavam na perda da bola, e a previsão de jogadas sendo praticamente inútil. Já após a identificação, o time começou a executar jogadas certeiras, raramente resultando na perda da bola ou condição de perigo, e a previsão de jogadas se mostrou bastante eficiente.

Quando executado contra times sem uma tomada de decisão inteligente, o time adversário as vezes se vê em uma condição que não foi programado, e simplesmente fica parado até o servidor executar uma ação aleatória, após um determinado intervalo de tempo, como visto na seção 2.2. Esse tipo de comportamento é bastante prejudicial ao time proposto, uma vez que o time prevê uma jogada do adversário e executa as ações para impedi-la, mas o time adversário simplesmente não faz nada. E quando o servidor executa uma ação pro adversário aleatoriamente, muitas vezes o time proposto já se adiantou a jogada, bloqueando os jogadores adversários, o que acaba gerando penalidades para o time proposto.

Nos resultados contra o próprio time base Agent2d, os resultados mostraram uma ligeira vantagem gerada pelo melhor posicionamento dos jogadores nas penalidades de bola parada. tal reposicionamento se mostrou bastante eficiente e decisivo em muitos momentos da partida, evidenciando a necessidade e a eficiências das estratégias reais.

Já nos resultados contra o time Helios2018, apesar das derrotas, foi um resultado bastante expressivo, visto que este é o atual campeão da RoboCup. Este resultado abre uma nova possibilidade para a implementação de novas estratégias reais, bem como a implementação de algoritmos inteligentes para a tomada de decisão em tempo real, de modo que o time proposto consiga utilizar diferentes formações táticas em uma mesma partida ainda que o time adversário mantenha um estilo de jogo constante. Tal variação pode ser bastante benéfica ao time proposto por dificultar a identificação das suas estratégias pelo time adversário, conseguindo reduzir o

poder ofensivo do poder adversário.

Propõe-se como trabalho futuro expandir o time UFMA2D com a adição de novas estratégias reais, principalmente para o goleiro, pois este ainda conta com uma estratégia bastante simples e pouco eficiente, principalmente nas situações em que ele está cercado por adversários. Sugere-se também um mecanismo para identificar se o adversário não travou durante o ciclo com a bola, visto que essa condição é extremamente prejudicial ao time. E embora não seja comum em times com mecanismo de tomada de decisão, este tipo de comportamento pode ocorrer na prática, e conseguir identificá-lo pode significar a não aplicação de uma penalidade ao time.

REFERÊNCIAS

BAI, A.; WU, F.; CHEN, X. Online planning for large markov decision processes with hierarchical decomposition. **ACM Transactions on Intelligent Systems and Technology (TIST)**, ACM, v. 6, n. 4, p. 45, 2015.

BOER, R. de; KOK, J. **The incremental development of a synthetic multi-agent system: The uva trilearn 2001 robotic soccer simulation team**. Tese (Doutorado) — Master's thesis, University of Amsterdam, The Netherlands, 2002.

COSTA, A. F. et al. Asimov soccer simulation 2d.

FIRA. **FIRA**. 2017. <http://www.fira.net/main/>. Accessed: 2017-05-30.

FRACCAROLI, E. S.; CARLSON, P. M.; COMPUTAÇAO, C. da. Uma abordagem fuzzy para modelagem de times de futebol de rob ^os.

GOMES, J. A. **Criação de um Time de Futebol Robótico para o Simulation League do Robocup**. 2017. Monografia (Bacharel em Ciência da computação), UFMA (Universidade Federal do Maranhão), São Luís, Brasil.

GOURAUD, H. Continuous shading of curved surfaces. **IEEE transactions on computers**, IEEE, v. 100, n. 6, p. 623–629, 1971.

HIDEHISA, A. Helios2007 team description paper. **RoboCup World Cup**, 2007.

IBM. **IBM Deep Blue**. 2018. <https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/>. Accessed: 2018-10-19.

KITANO, H. **RoboCup-97: robot soccer world cup I**. [S.l.]: Springer Science & Business Media, 1998. v. 1395.

KITANO, H. et al. Robocup: The robot world cup initiative. In: ACM. **Proceedings of the first international conference on Autonomous agents**. [S.l.], 1997. p. 340–347.

PROKOPENKO, M. et al. Gliders2013: Tactical analysis with information dynamics. In: **RoboCup 2013 symposium and competitions: Team description papers, Eindhoven, The Netherlands**. [S.l.: s.n.], 2013.

REIS, L. P. G. **Coordenaçao em sistemas multi-agente: Aplicaçoes na gestao universitária e futebol robótico**. Tese (Doutorado), 2003.

ROBOCUP. **Tabela de classificação da liga simulada 2d**. 2018. <https://en.wikipedia.org/wiki/RoboCup_2D_Soccer_Simulation_League>. Accessed: 2018-11-24.

SILVA, A. T. et al. ibots 2010: Descriçao do time. **Competiçao Latino Americana de Robótica**, 2010.

WIKIPEDIA. **Delaunay triangulation**. 2013. <https://en.wikipedia.org/wiki/Delaunay_triangulation>. Accessed: 2018-12-23.

Apêndices

APÊNDICE A – INSTALAÇÃO E CONFIGURAÇÃO DE ARQUIVOS

Este manual foi baseado no tutorial disponível em <https://github.com/herodrigues/robocup2d/wiki/Instalando-o-simulador>, contendo muitas atualizações e instalação de dependências extras para a instalação e o correto funcionamento do *Fedit*. Esta versão é somente para referência, a versão mais atualizada pode ser obtida em <https://github.com/rodrigogarces/RoboCup2d>

Dependências

Este comando instala todas as dependências necessárias para o correto funcionamento do futebol robótico, bem como a instalação dos times base Agent2d e UvA Trilearn, por serem os mais utilizados. Todos os passos devem ser seguidos na sequência especificada, não podendo ser omitida qualquer parte deste tutorial.

O comando apenas instala e configura as bibliotecas que ainda não estiverem instaladas no sistema. Caso a biblioteca já esteja instalada, o sistema acusa que a mesma está instalada e prossegue para o próximo comando. Este tutorial só funciona até o Ubuntu 16.04, pois é a última versão suportada oficialmente pelo repositório da RoboCup. Para utilizar em uma versão mais recente, você deve baixar os fontes e compilar manualmente.

Instala dependências

```
$ sudo apt-get install -y g++ build-essential libboost-all-dev qt4-default
    qt4-dev-tools qt4-qmake libaudio-dev libgtk-3-dev libxt-dev doxygen tcsh
```

Adiciona o repositório para posterior instalação do simulador

```
$ sudo apt-add-repository -y ppa:gnurubuntu/rubuntu
$ sudo apt-get update
```

Instalação

Rcssserver (servidor)

```
$ sudo apt-get install -y rcssserver
```

Monitor

```
$ sudo apt-get install -y rcssmonitor
```

Soccersim (servidor e monitor juntos)

```
$ sudo apt-get install -y rcsoccersim
```

Librcsc

```
$ wget http://c3sl.dl.osdn.jp/rctools/51941/librcsc-4.1.0.tar.gz
$ tar -zxpf librcsc-4.1.0.tar.gz
$ cd librcsc-4.1.0
$ sudo ./configure
$ sudo make
$ sudo make install
```

Fedit2

```
$ git clone https://github.com/KN2C/rctools.git
$ cd rctools/fedit2-0.0.0/
$ export QT_SELECT=4
$ qmake fedit2.pro
$ make
```

Agent2d

```
$ wget http://c3sl.dl.osdn.jp/rctools/55186/agent2d-3.1.1.tar.gz
$ tar -zxpf agent2d-3.1.1.tar.gz
$ cd  agent2d-3.1.1
$ sudo ./configure
$ sudo make
```

UVA Trilearn (atualizado e adaptado pelo PET de engenharia da computação da UFES)

```
$ cd ~
$ echo "Instalando o UVA Trilearn"
$ wget http://inf.ufes.br/~pet/projetos/Simulacao_2D/Sim2D/
  trilearn_base_sources-3.3-v13-by_PET.tar.gz
$ tar -zxpf trilearn_base_sources-3.3-v13-by_PET.tar.gz
$ cd  trilearn_base_sources-3.3-v13-by_PET
$ sudo ./configure
```

Execução

Correção do problema do separador decimal

Antes de iniciar com o simulador, corrija o problema do separador decimal, pois o simulador foi originalmente pensado para trabalhar com o ponto sendo o separador decimal, o que faz com que apresente funcionamento errôneo quando executado em uma linguagem (idioma) que utilize vírgula como separador decimal. Existem dois métodos. O primeiro não

exige nenhum conhecimento técnico, mas exige que seja executado a cada vez que iniciar a máquina E desejar utilizar o simulador. O segundo método exige um mínimo de conhecimento sobre linux, mas precisa ser executado uma única vez, para configuração. Depois não é necessário executar novamente, como ao reiniciar a máquina.

Método 1

Execute o comando no terminal

```
$ export LC_NUMERIC="C"
```

Este comando garante que o simulador não terá problema em trabalhar com o separador decimal sendo a vírgula. Este comando deve ser executado a cada sessão do usuário [em que pretenda usar o simulador], caso contrário, o simulador irá executar, mas a simulação não irá rodar de maneira satisfatória. Você só precisa executar o comando no primeiro terminal.

Método 2

Abra o arquivo que contém os comando do terminal com o comando

```
$ sudo nano ~/.bashrc
```

Caso sua distribuição não possua o arquivo .bashrc, crie-o! Adicione a seguinte linha ao final do arquivo

```
$ export LC_NUMERIC="C"
```

Salve o arquivo e reinicie a máquina

Soccersim (Servidor e monitor juntos)

O seguinte comando executa o servidor e o monitor juntos, sendo extremamente útil para fins de testes na máquina local, por executar uma partida de forma automática, bastando para tanto somente iniciar os times (que irão se conectar ao servidor na porta padrão 6000.).

```
$ rcsoccersim
```

Servidor

```
$ rcssserver
```

Monitor

```
$ rcssmonitor
```

Caso apareça o erro "could not read or create config directory '/home/<usuário>/.rcssserver/', feche a abra novamente o rcsoccersim. Caso o erro persista, rode o programa como root. Este

erro pode acontecer somente ao abrir o rcsoccersim pela primeira vez. Agora, você tem dois times base, o Agent2, da equipe Helios, e o UVA Trilearn, da equipe de mesmo nome. Ambos os times são implementações prontas das funções de baixo nível das respectivas equipes, e são distribuídos gratuitamente para que os iniciantes gastem o tempo fazendo o que realmente conta, a estratégia. Ou seja, todas (ou pelo menos a maioria) das funções básicas já vem implementadas, de modo que o iniciante precisa apenas implementar funções específicas e a estratégia de sua equipe, reduzindo de maneira muito significativa a quantidade de tempo e esforço gastos na fase inicial de implementação de uma nova equipe.

Agent2d

Primeira equipe

```
$ cd <pasta do Agent2d>/src
$ ./start.sh
```

Segunda equipe

```
$ cd <pasta do Agent2d>/src
$ ./start.sh -t <nome da equipe>
```

UvA Trilearn

Primeira equipe

```
$ cd <pasta do UvA Trilearn>
$ ./start.sh
```

Segunda equipe

```
$ cd <pasta do UvA Trilearn>
$ ./start.sh localhost <nome da equipe>
```

Referências

<https://github.com/herodrigues/robocup2d/wiki/Instalando-o-simulador>
<https://veroneze.wordpress.com/2016/05/30/robocup-simulation-league-2d-instalar-o-ambiente/>
<http://inf.ufes.br/~pet/projetos/Simulacao_2D/Sim2D/simulador.pdf>
<https://www.linuxquestions.org/questions/ubuntu-63/error-in-installing-fedit2-4175632099/>

Script de instalação automática

```
#!/bin/bash
echo "this tutorial is based on https://github.com/herodrigues/robocup2d-tutorial/blob/master/sections/installing-the-soccer-simulator.md"
echo "fedit2 installation is based on https://www.linuxquestions.org/questions/ubuntu-63/error-in-installing-fedit2-4175632099/"
echo "Install process can take 5~30 min, deppending on your pc and internet speed"

echo "Installing dependencies"
sudo apt-get install -y g++ build-essential libboost-all-dev qt4-default qt4-dev-tools qt4-qmake libaudio-dev libgtk-3-dev libxt-dev doxygen tcsh

echo "Adding robocup 2d repository"
sudo apt-add-repository -y ppa:gnurubuntu/rubuntu
sudo apt-get update

echo "Installing rcssserver rcssmonitor rcsoccersim rcsslogplayer"
sudo apt-get install -y rcssserver rcssmonitor rcsslogplayer

cd ~
echo "Installing librcsc"
wget http://c3sl.dl.osdn.jp/rctools/51941/librcsc-4.1.0.tar.gz
tar -xf librcsc-4.1.0.tar.gz
cd librcsc-4.1.0
sudo ./configure
sudo make
sudo make install

cd ~
echo "Installing fedit2"
git clone https://github.com/KN2C/rctools.git
cd rctools/fedit2-0.0.0/
export QT_SELECT=4
qmake fedit2.pro
make

cd ~
echo "Installing agent2d"
wget http://c3sl.dl.osdn.jp/rctools/55186/agent2d-3.1.1.tar.gz
tar -xf agent2d-3.1.1.tar.gz
cd agent2d-3.1.1
sudo ./configure
sudo make

cd ~
echo "Installing UvA Trilearn"
```

```
43 wget http://inf.ufes.br/~pet/projetos/Simulação_2D/Sim2D/
       trilearn_base_sources-3.3-v13-by_PET.tar.gz
44 tar -xf trilearn_base_sources-3.3-v13-by_PET.tar.gz
45 cd trilearn_base_sources-3.3-v13-by_PET
46 sudo ./configure
47 sudo make
48
49 echo "Edit decimal separator"
50 sudo echo "export LC_NUMERIC=C" >> ~/.bashrc
51
52 echo "Installation complete"
53 echo "Open a new terminal instance or reload terminal (source ~/.bashrc) to
       apply changes"
```

APÊNDICE B – SCRIPT AUXILIAR PARA A REALIZAÇÃO DE TESTES AUTOMÁTICOS

Esta versão do script é somente para referência, a versão mais atualizada pode ser obtida em <https://github.com/rodrigogarces/RoboCup2d>

Tendo em vista a necessidade de realizar muitas partidas a fim de obter estatística, e o tedioso processo envolvido na execução de uma partida, foi desenvolvido um *script* que realiza várias partidas paralelamente, sendo necessário apenas executar o *script*. Este pode ser facilmente adaptado para executar muitas partidas em lotes, dividindo o total em pequenos grupos de partidas que podem se executadas em simultâneo, sem que o desempenho dos times seja comprometido pela capacidade limitada de processamento, tendo em vista que todos os algoritmos de tomada de decisão são executados em tempo real e projetados para trabalhar com o tempo fixo de 100ms (tempo do ciclo de execução do simulador).

O *script* funciona com times baseados no Agent2d (inclusive o Helios) e em qualquer outro time que permita especificar as portas de execução via parâmetro. Já o UvA Trilearn infelizmente não é compatível, por não permitir que as portas de execução sejam especificadas via parâmetro, e mesmo modificando as portas direto no código fonte e recompilando o time, não conseguiu se conectar de maneira satisfatória ao simulador. E mesmo fazendo a execução de duas partidas simultâneas manualmente, o UvA Trilearn enfrentou problemas de conexão, o que demonstra uma limitação interna do time.

O *script* foi dividido em duas partes: o multigame.sh, responsável por executar as partidas, e o masterkill.sh, responsável por matar todas as instâncias criadas pelo primeiro script. Mas cuidado, execute o masterkill somente após o término de todas as partidas, visto que ele mata todas as instâncias do simulador, inclusive alguma outra que você possa ter aberto manualmente. E o log das partidas fica salvo na mesma pasta em que o multigame está.

```
1  #!/bin/bash
2  #Rodrigo Garcês – 2018
3
4  def=6000          #Default server port (for fisrt instance)
5  match=10          #Simultaneous match number
6  interval=6        #interval betwwen execute a new match (simultaneous)
7
8  match=$((match – 1))
9
10 for (( c=0; c<=$match; c++ ))
11 do
12
13     server=$(($def))          #Server port
14     coach=$(($def + 1))       #Coach port
15     olcoach=$(($def + 2))     #Online coach port
```

```
16
17      #echo $server
18      #echo $coach
19      #echo $olcoach
20
21
22      rcssserver server::auto_mode=on server::port=$server server::coach_port
        =$coach server::olcoach_port=$olcoach& # automatically start the game
23
24      cd ~/UFMA2D/src
25      ./start.sh -p $server -P $olcoach&
26
27      cd ~/agent2d-3.1.1/src
28      ./start.sh -p $server -P $olcoach&
29
30      rcssmonitor --server-port $server&
31
32
33      def=$(($def + 3))
34      sleep $interval
35  done
```