TANILSON DIAS DOS SANTOS

Jogadas Coletivas por meio de Comunicação Multi-Agente para a equipe iBots da Categoria de Simulação 2D da RoboCup

PALMAS

JUNHO DE 2011

Jogadas Coletivas por meio de Comunicação Multi-Agente para a equipe iBots da Categoria de Simulação 2D da RoboCup

TANILSON DIAS DOS SANTOS
Graduando em Ciência da Computação

Prof. M.Sc Alexandre Tadeu Rossini da Silva, Ciência da Computação
Orientador

Monografia submetida ao curso de graduação em Ciência da Computação como parte dos requisitos necessários para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal do Tocantins - UFT
Campus Universitário de Palmas - TO

PALMAS, TO - BRASIL
JUNHO DE 2011.

Santos, Tanilson Dias dos.

Jogadas Coletivas por meio de Comunicação Multi-Agente para a equipe iBots da Categoria de Simulação 2D da RoboCup / Tanilson Dias dos Santos - Palmas, 2011

xiv, 83 p.:il color; 31cm.

Monografia (graduação) - Curso de Ciência da Computação, Fundação Universidade Federal do Tocantins, 2011.

Orientador: Prof. M.Sc. Alexandre Tadeu Rossini da Silva

1.Futebol de robôs. **2.** Inteligência artificial. **3.** Lógica *fuzzy*. **4.** Sistemas multi-agentes. **I.** Título.

Jogadas Coletivas por meio de Comunicação Multi-Agente para a equipe iBots da Categoria de Simulação 2D da RoboCup

TANILSON DIAS DOS SANTOS

Graduando em Ciência da Computação

Aprovada por:

Prof. M.Sc. Rogério Azevedo Rocha / UFT

Prof.

Prof.

Monografia submetida ao curso de graduação em Ciência da Computação como parte dos requisitos necessários para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal do Tocantins - UFT

Campus Universitário de Palmas - TO

PALMAS, TO - BRASIL

JUNHO DE 2011.

"A descoberta consiste em ver o que todo mundo viu e pensar o que ninguém pensou"

Albert Von Szent-Györgyi Nagyrápolt

Dedico este trabalho aos meus pais, Tânia e Ariston, e à minha irmã, Aristiane, que sempre me ajudaram nas horas de dificuldades e nunca me negaram palavras de apoio.

À todos os meus colegas de graduação, com os quais convivi os quatro anos da graduação, que sempre estiveram presentes comigo nos momentos de dificuldade, incertezas e conquistas.

Ao pessoal da minha querida cidade de Brejinho de Nazaré e todos os meus parentes.

Dedico também à todos amigos que fiz no grupo Eletrobras Eletronorte, na regional de transmissão do Tocantins, os quais foram pessoas que me ajudaram muito a amadurecer profissionalmente.

Agradecimentos

Agradeço, primeiramente, a Deus por todas as graças que tem permitido que eu alcance.
De modo especial à minha mãe, Tânia Andrade, pois ela, mais do que ninguém, sempre pelejou para que eu pudesse alcançar essa vitória.
Ao meu pai, Ariston Oliveira, pelo exemplo de vida, honestidade e perseverança.
Ao amigo Horiano Gomes pela ajuda prestada que possibilitou concluir este trabalho.
Aos amigos Diego Antonio, Eduardo Paixão e Lais Gomes por ensinar o valor da persistência em alcançar um ideal e pelas palavras de incentivo e determinação comportalhidas ao longo do curso.
Aos amigos Vitor Sousa e Edcarllos Santos por mostrarem o valor da humildade e da calma e pelas experiências compartilhadas nas horas de dificuldade.
Aos amigos Laercio Pontin, Danilo Queiroz, Gustavo Cunha, Gleice Ramos, Maely Passos, por todos momentos de descontração compartilhados.
À todos colegas do curso de Ciência da Computação, em especial ao amigo Felipe Albuquerque que muito ajudou nos momentos de incerteza.
À todos amigos de minha querida cidade, Brejiho de Nazaré.
Aos meus amigos Nilton e Pedro, e às suas esposas, que me ampararam no primeiro ano em Palmas, por se prontificarem em me ajudar na hora em que eu precisava.
Ao meu orientador, Alexandre Rossini, pela ajuda no desenvolvimento do trabalho, revisão de textos e pela dedicação e seriedade que sempre mostrou, além dos bons momentos compartilhados no curso.
À todos que tiveram envolvidos de forma direta e indireta neste trabalho.

Tanilson Dias dos Santos

Resumo

O processo de interação entre agentes, em um time de futebol de robôs, é importante para escolher um melhor conjunto de ações (individuais e coletivas) a fim de atingir o principal objetivo da equipe, vencer. Uma forma de melhorar o desempenho dos agentes e, consequentemente, da equipe é aumentar o grau de cooperação existente entre eles. A execução de jogadas coletivas depende intimamente da troca fonética de mensagens entre os jogadores e técnico da equipe. É interessante explorar todos os artifícios possíveis de se utilizar durante a partida, e nesse caso, a existência (e bom uso) de determinadas jogadas coletivas pode representar a diferença entre vitória e derrota do time em situações críticas. O foco abordado nesse projeto foi o desenvolvimento e controle da utilização de jogadas coletivas de bola parada. É interessante que os agentes levem em conta ao tomar decisões as mensagens enviadas pelos companheiros de equipe. Essas mensagens devem fornecer dados relevantes para as tomadas de decisão no atual ciclo e em ciclos futuros. Um algoritmo *fuzzy* é adotado como mecanismo de suporte a decisão para a escolha da melhor jogada em determinada situação de jogo. O time utilizado para os testes foi a *framework* disponibilizada pela equipe *Helios_Base*, atual campeã da categoria de simulação 2D, a *Agent2D*. Foram realizadas algumas alterações no código da ferramenta e uma extensão à biblioteca *librcsc*, que é uma das bibliotecas-base para a realização de simulações.

Palavras-chave: *Robocup*, Comunicação, Inteligência Artificial, Futebol Robôs, *Simulation League*, Lógica *Fuzzy*, Jogadas Coletivas.

Abstract

The process of interaction between agents in a soccer team of robots, it is important to choose a better set of actions (individually and collectively) to achieve the main goal of the team, win. One way to improve agent performance and thus the team is to increase the degree of cooperation between them. The collective creation of plays intimately dependent phonetic exchange of messages between the players and coach. It is interesting to explore every possible device to use during the match, and if so, the existence (and good use) of certain collective moves can mean the difference between victory and defeat of the team in critical situations. The focus addressed in this project was to develop and control the use of collective pause ball movies. It is interesting that the agents take into account when making decisions messages sent by teammates. These messages should provide relevant data for decision making in the current cycle and future cycles. A *fuzzy* algorithm is adopted as the decision support mechanism for choosing the best move in a particular game situation. The team used for the tests was the *framework* provided by the team *Helios_base*, champions of the category of 2D simulation, the *Agent2D*. Underwent some changes to the code of the tool and a library extension to *librcsc*, which is a base of libraries for the simulations.

Keyword: *RoboCup*, *Communication, Artificial Intelligence, Soccer Robots*, *Simulation League* , *Fuzzy Logic*, *Played Collective.*

Sumário

Lista de Figuras

Lista de Tabelas

Siglas e Abreviações

ASCII	:	*American Standard Code for Information Interchange*
CCD	:	*Charge-Coupled Device*
FIRA	:	*Federation of International Robot-soccer Association*
IA	:	Inteligência Artificial
IEEE	:	Instituto de Engenheiros Eletricistas e Eletrônicos
LARC	:	*Latin American Robotics Competition*
LCA	:	Linguagem de Comunicação entre Agentes
LISP	:	*List Processing*
ROBOCUP	:	*Robot World Cup Soccer, Games and Conferences*
SBC	:	Sociedade Brasileira de Computação
SMA	:	Sistema Multi-Agente
UDP	:	*User Datagram Protoco*l

Glossário de Termos

Agente	:	uma entidade real ou virtual, capaz de agir em um ambiente, de se comunicar com outros agentes e que é movido por um conjunto de inclinações
Agent2D	:	*framework* disponibilizada pela equipe *Helios* para desenvolvimento de equipes para categoria de simulação 2D da *Robocup*
Crisp	:	é um valor puro ou preciso, ou seja, pertencente aos conjuntos ou sistemas booleanos
Defuzzificação	:	transformação do valor *fuzzy* em uma saída *crisp*
Diagrama de *Voronoi*	:	é uma decomposição de um espaço métrico em regiões de acordo com a distância a determinados pontos
Flag	:	é um mecanismo lógico semelhante a um semáforo
Framework	:	ferramenta que fornece as condições iniciais para o desenvolvimento de um trabalho com futebol de robôs, dita o fluxo de controle da aplicação
Fuzzificação	:	transformação da entrada *crisp* em valor *fuzzy*
Jogada Coletiva	:	colaboração entre agentes resultantes da comunicação utilizada entre eles para realização de uma jogada no futebol, no caso deste trabalho
Librcsc	:	biblioteca-base para utilização da *Agent2D*
Triangularização de *Delaunay*	:	dá-se esse nome à triangularização aplicada sobre o diagrama de Voronoi
XFuzzy	:	ferramenta de auxilio à modelagem, teste e desenvolvimento de conjuntos *fuzzy*

1 Introdução

O futebol de robôs é um problema padrão de investigação internacional que reúne grande parte dos desafios presentes em problemas do mundo real a serem resolvidos em tempo real. Um problema padrão de investigação internacional, estimulante do ponto de vista científico, coloca um vasto conjunto de problemas aos pesquisadores da área e ao mesmo tempo procura despertar grande interesse no público em geral e refletindo também nos meios de comunicação. A padronização de problemas é importante, uma vez que metodologias distintas podem ser comparadas, além de incentivar a pesquisa em Inteligência Artificial e Robótica Inteligente.

Os problemas de investigação contidos no futebol de robôs cobrem uma área mais ampla do que aparentam, já que o jogo pode ser visto apenas como uma simples brincadeira. No entanto, o futebol de robôs constitui um domínio bem mais complexo. Dentre os problemas nele explorados, podem ser citados a coordenação, cooperação, comunicação entre máquinas, aprendizagem, planejamento em tempo real, decisão estratégica, tática, comportamento, visão, controle, locomoção e sistemas sensoriais. Além disso, as soluções encontradas para o futebol de robôs podem ser estendidas para outras aplicações, possibilitando, por exemplo, o uso da robótica em locais de difícil acesso para humanos, ambientes insalubres e situações de risco de vida iminente, incluindo a exploração espacial.

No futebol de robôs, se desenvolvida apropriadamente, a comunicação é um dos fatores que pode ajudar uma equipe a ter melhor desempenho. Isso porque permite negociação e cooperação direta entre os agentes envolvidos. Porém, no futebol robótico, todos os dados transmitidos e toda informação que é enviada e recebida pelos robôs durante a partida está sujeita a interferências (ruído) (CHEN et al., 2003). Isso ocorre uma vez que o simulador reproduz as interferências sonoro-visuais existentes em jogos de futebol. Assim como no futebol de humanos, a comunicação entre os jogadores pode definir jogadas que necessitem de um posicionamento mais ofensivo/defensivo da equipe. Essa postura adotada pela colaboração direta (pois um jogador comunica diretamente ao companheiro de equipe sua intenção) é chamada de 'jogada coletiva', ou seja, jogadas realizadas entre

agentes decorrente da troca de mensagens entre eles.

1.1 Justificativa

O desenvolvimento de uma equipe simulada foi definido uma vez que é mais barato projetá-la e manuteni-la do que uma equipe de robôs físicos, já que não apresenta custos de estrutura material. Na categoria simulada há a necessidade somente de *software* adequado para gerenciamento e visualização das ações dos robôs, fornecido pela federação **RoboCup**. Além disso justifica-se este trabalho uma vez que dá continuidade à pesquisa realizada anteriormente por Silva et al. (2010), Silva (2010) e Ferreira (2010), já que estes trabalhos fazem parte do projeto de pesquisa "*Futebol de Robôs: Uma Plataforma Cooperativa para Sistemas Inteligentes*". Neste projeto foi desenvolvida a equipe *iBots*, que representou a UFT na categoria Simulação 2D da *Robocup* na Competição Latino-Americana de Robótica (*LARC - Latin American Robotics Competition*). No trabalho de Silva (2010) há algumas sugestões de trabalhos futuros e a implementação de jogadas coletivas é uma delas.

1.2 Objetivo

O objetivo que norteou este trabalho foi o de desenvolver jogadas coletivas que façam uso da comunicação entre agentes da equipe *iBots*, a fim de aprimorar a cooperação entre eles, atualmente implementada de forma indireta. A ideia de aprimorar a equipe *iBots* e, assim obter um melhor desempenho, veio da observância do baixo grau de colaboração que existia entre os agentes. Cada agente, ao tomar uma decisão, espera que os demais companheiros de equipe tomem as decisões a fim de alcançar com êxito cada próxima ação a ser realizada. Por exemplo, no toque (ou passe) um jogador percebe o outro desmarcado e lança a bola para este segundo esperando que ele não se movimente e esteja preparado para receber a bola. Atualmente o toque e todas as outras habilidades implementadas na equipe *iBots* ocorrem sem que haja comunicação entre os agentes (cooperação indireta). Com a existência de uma cooperação direta, um agente ao se desmarcar poderia sinalizar ao companheiro de posse da bola para tocar em um determinado ponto, neste caso possibilitando aumentar os acertos de passes. Esse tipo de comportamento é o foco das jogadas coletivas das quais tratam este trabalho.

1.3 Descrição do Problema

O processo de interação entre agentes, em um time de futebol de robôs, é importante para escolher um melhor conjunto de ações (individuais e coletivas) a fim de atingir o principal objetivo da equipe, vencer. Em outras palavras, isso significa que a existência e bom uso de jogadas coletivas pode representar a diferença entre vitória e derrota em situações de equilíbrio técnico entre equipes em uma partida. Uma forma de melhorar o desempenho dos agentes e, consequentemente da equipe, é criar cooperação direta entre eles por meio de comunicação. Neste caso, a criação de jogadas coletivas depende intimamente da troca de mensagens entre os agentes (jogadores/jogadores e jogadores/técnico).

Durante uma partida de futebol, as jogadas coletivas, resultantes da comunicação entre agentes (robóticos ou humanos), pode ser realizada de diversas maneiras e em diferentes situações. Desse modo, o primeiro problema envolvido se refere à definição de quando poderão ser realizadas jogadas coletivas no decorrer de uma partida, ou seja, definir se as jogadas coletivas são defensivas, ofensivas, de contra-ataque, bola parada, etc. Outro problema envolvido está em projetar as jogadas coletivas que serão implementadas na equipe: ("jogadas ensaiadas", tabela entre jogadores, linha de impedimento, etc.).

Uma vez definidas as jogadas coletivas e em que situações de jogo poderão ser realizadas, surge o problema de definir os conteúdos das mensagens que serão usadas para a comunicação. É importante ressaltar que a estrutura das mensagens na categoria de simulação 2D da *RoboCup* é definida pelo simulador, assim o conteúdo das mensagens deve ser elaborado seguindo as especificações e restrições do simulador.

Outro problema aparece para realizar uma troca de mensagens eficiente, pois os agentes precisam realizar uma avaliação no ambiente de jogo (reproduzido nos modelos de mundo de cada um dos agentes) para determinar qual jogada será executada em um determinado momento. Neste ponto é necessário de um sistema de tomada de decisão, que pode ser reativo (que necessita ser projetado para todas as situações esperadas) ou cognitivo (que possui uma base de conhecimento e, por isso, capaz de generalizar a partir de algumas poucas situações planejadas). A decisão deve ser repassada pelo agente que processou a informação aos demais agentes que participarão da jogada.

Isso significa que, após definir qual jogada coletiva será executada, o problema passa por identificar os jogadores com melhores condições de participar e comunicá-los acerca do seu envolvimento na jogada. Não se deve tratar somente o envio das mensagens, mas também a interpretação de seu significado para cada agente receptor. É necessário ainda

que cada agente saiba analisar os momentos oportunos para enviar mensagens aos demais agentes de sua equipe para não "queimar" tentativas com o insucesso da execução. Esse problema, que deve ser tratado, ocorre porque o número de mensagens que podem ser ouvidas é limitado a fim de não congestionar o canal de comunicação (XAVIER et al., 2006).

Os últimos problemas se referem à implementação computacional das soluções dos problemas anteriormente descritos. Na prática, o desenvolvimento de jogadas coletivas extrapola a alteração/adição do código fonte da equipe e da *framework* do sistema de tomada de decisão. É necessário realizar uma extensão da biblioteca *librcsc*[1], que é uma das bibliotecas que servem de plataforma para a utilização da *framework Agent2D* (que é a ferramenta utilizada atualmente pela equipe *iBots*), para suportar a implementação da solução.

1.4 Metodologia

A etapa inicial foi constituída de pesquisa bibliográfica para estudar técnicas de comunicação implementadas em outras equipes. Nessa etapa, foi realizado um estudo investigando soluções de acordo com as características de cada equipe. Concomitantemente foram analisadas as habilidades já implementadas da equipe *iBots* e as diferentes necessidades de comunicação entre os agentes para o desenvolvimento de jogadas coletivas. Posteriormente foram implementadas jogadas de bola parada com diferentes concepções estratégicas, ordenando-as da mais defensiva à mais ofensiva. O passo seguinte foi definir e implementar um algoritmo cognitivo para decidir qual a melhor jogada a ser adotada para a situação de jogo analisada, levando em consideração posicionamento dos jogadores, regiões do campo menos povoadas e potenciais oportunidades de gol. Por fim, foram realizados testes com a finalidade de comparar a equipe antes e depois da implementação das jogadas coletivas propostas neste trabalho.

1.5 Organização da Monografia

Este trabalho está dividido em cinco capítulos. O capítulo 2 é dedicado à revisão bibliográfica, onde são apresentados os principais conceitos envolvidos neste trabalho e necessários para o seu entendimento. O capítulo 3 dá detalhes específicos do desenvol-

[1] *Librcsc-4.0.0* - É um software livre que atua sob a licença LGPL. É também um dos três projetos *open source* que vem sendo desenvolvidos pela equipe *Helios* (AKIYAMA; SHIMORA, 2010).

vimento, escolha das técnicas e soluções adotadas. O capítulo 4 apresenta os testes e os resultados obtidos com a aplicação da proposta e uma comparação com a versão anterior do time. No capítulo 5 são realizadas as conclusões e considerações finais com sugestões de trabalhos futuros.

2 Revisão Bibliográfica

Neste capítulo são contextualizados os problemas e apresentados os principais conceitos envolvidos neste trabalho. Assim, é descrito o futebol de robôs, suas ligas e suas categorias. Ainda são apresentadas as principais *frameworks* da liga de simulação 2D da *RoboCup*, sistemas multi-agentes com as linguagens de comunicação entre agentes e, por fim, técnicas de Inteligência Artificial, em especial a lógica *fuzzy*.

2.1 Futebol de Robôs

O surgimento do futebol de robôs tem origem prática ligada diretamente com a aplicação da Inteligência Artificial (IA) em jogos. O xadrez foi o primeiro jogo a se tornar um problema padrão em que foram utilizadas técnicas de IA, onde uma máquina se confrontaria com um ser humano, e data de 1950, segundo Strassmann (2006). Desde então instigou-se a criação de máquinas (agentes) que cada vez mais desafiassem e equiparassem, ou até mesmo superassem, o potencial humano em atividades específicas.

O desenvolvimento do primeiro time de futebol de robôs tem origem em meados de 1993. No mesmo ano, um grupo de pesquisadores japoneses, formado por Minoru Asada, Yasuo Kuniyoshi e Hiroaki Kitano, decidiu iniciar uma competição de robótica nomeada *Robot J-League* (ROBOCUP, 2010). Essa foi a liga pioneira nos estudos ligados ao futebol de robôs, dela derivaram todas as federações e ligas da atualidade. O sucesso de implantação da *Robot J-League* foi tão grande que, logo um mês depois de ser criada, começaram a surgir solicitações de todo o mundo para os membros fundadores. Com a intenção de que o projeto fosse internacionalizado, nascia a *RoboCup* e se transformaria na primeira liga profissional de futebol de robôs do mundo. A figura 2.1 mostra o *banner* da *RoboCup* 2010.

A *Robot J-League* foi a primeira a realizar um campeonato com foco em robótica e

Figura 2.1: *Banner* RoboCup 2010 (ROBOCUP, 2010)

IA, porém uma ideia semelhante já havia sido proposta em 1992[1] por Mackworth (1993), que sugeriu o futebol como base para o estudo e desenvolvimento de robôs inteligentes. Logo após isso, vários outros pesquisadores notaram a vastidão de campos que o futebol de robôs aborda, tais como atuação em ambientes insalubres e de alta periculosidade, reconhecimento de objetos em tempo real, colaboração direta entre agentes, exploração espacial, entre outros (STRASSMANN, 2006).

Atualmente, existem duas grandes federações internacionais que trabalham diretamente com o futebol de robôs, a *Fira* e a *RoboCup*. Cada uma dessas federações tem suas próprias regras, focos e categorias. A seguir são apresentadas as principais características de cada uma das categorias.

1. *Robot World Cup Soccer Games and Conferences* (**RoboCup**)- tem grande ênfase nos aspectos teóricos e inovações. O futebol competitivo, apesar de não ser o único foco, é um dos principais pretextos para disseminação de pesquisas nas áreas de ciência e tecnologia da *RoboCup*. A investigação da área se propõe a resolver problemas em ambientes dinâmicos e não determinísticos com colaboração multiagente, algumas vezes homogêneos outras heterogêneos. Todos os robôs são autônomos e suas principais categorias são:

[1] O artigo original "*On Seeing Robots*", embora datado de 1992, foi publicado posteriormente em 1993 no livro *Computer Vision: Systems, Theory and Applications*, pp. 1-13.

- **RoboCupRescue**: objetiva a investigação e utilização de robôs em missões de salvamento e resgate decorrentes de grandes catástrofes (SILVA, 2010). Procura tratar problemas que envolvem múltiplos agentes heterogêneos para trabalhar de forma ordenada trazendo informações sobre vítimas, infra-estruturas abaladas, suporte à decisão para avaliações do ambiente, entre outras. É dividida em duas grandes ligas:

  Robot League - exige que os robôs demonstrem suas capacidades de mobilidade, sensores de percepção, planejamento, mapeamento, interface de operação prática, isso enquanto procura vítimas em um ambiente desestruturado;

  Simulation League - esta categoria possui uma proposta dupla, sendo a primeira o desenvolvimento de uma estrutura que represente uma simulação fiel de ambientes abalados por catástrofes naturais; e a segunda proposta, desenvolver agentes para trabalharem em ambientes específicos.

- **RoboCupJunior**: tem foco na educação e visa estimular a introdução dos jovens e crianças no ramo da robótica e desenvolver suas habilidades. Possui três subdivisões:

  Soccer - robôs autônomos são colocadas em pares em cada uma das equipes no ambiente. Esta categoria não possui um goleiro. A bola emite um feixe de luz para facilitar que o robô a encontre e o campo é cercado por um pequeno muro;

  Dance - um ou mais robôs, ornamentados de acordo com a temática proposta pela equipe, devem mover-se de forma criativa (em uma espécie de dança) de acordo com a música que a equipe escolher;

  Rescue - esta categoria tem robôs pré-modelados e possui os mesmos objetivos da grande categoria *RoboCupRescue*, com a diferença de que os obstáculos são mais simples.No chão, o caminho a ser seguido é previamente traçado, com algumas pequenas imperfeições a serem corrigidas pelo robô, para que os sensores infra-vermelho dos robôs captem informações do trajeto.

- **RoboCupSoccer**: liga de futebol de robôs que engloba robôs simulados e físicos de diferentes tamanhos e características. Esta categoria é composta por robôs autônomos e tem foco no futebol competitivo. Atualmente possui cinco grandes subvisões, que são:

  Simulation - se divide em duas outras subcategorias: *2D* e *3D*. É uma das

mais antigas ligas da *RoboCup Soccer*. Os jogadores aqui são agentes virtuais que atuam em um campo de futebol simulado. O foco desta categoria é o desenvolvimento de técnicas e estratégias inteligentes que possam ser aplicadas a sistemas multiagentes;

Humanoid - esta liga se subdivide em mais três, de acordo com o tamanho do robô, e pode ser *Teen Size*, *Kid Size* ou *Adult Size*. Os robôs são bípedes, possuem aparência humanóide, devem perceber o ambiente, aliados, adversários e bola através da percepção visual. Os jogadores também devem ser capazes de ponderar sobre sua pose (posição e orientação) em campo e tomar decisões sobre qual comportamento adotar em cada situação de jogo;

Figura 2.2: Robôs da Categoria *Humanoid* (ELAKIRI, 2011)

Mixed Reality - é uma categoria pouco conhecida, anteriormente chamada de *Phisycal Visualization* (GERNDT et al., 2008). Na prática, representa uma variação da *simulation league*, porém mistura simulação e realidade, o que dá a esta categoria uma propriedade particular de oferecer um campo de testes para comportamentos abstratos em ambientes complexos e permitir interação física entre os robôs. O principal campo de pesquisa abordado é a interoperação de um grande conjunto de robôs autônomos em várias aplicações e ambientes. O robô utilizado na categoria é o *Citizen Microrobot*, ver imagem 2.3, que é simples e tem baixo custo;

Figura 2.3: *Citizen Micro-robot*

Small Size - esta é a famosa categoria *F-180*. Aborda praticamente todos os problemas propostos anteriormente e tem foco mais voltado para sistemas altamente dinâmicos e híbridos com processamentos centralizado e distribuído;

Standard Plataform - todos os times desta categoria devem utilizar o mesmo robô, o *Aldebaran's Nao Humanoids* (ver figura 2.4 da página 10). Atualmente o robô custa cerca de $ 12.000 (doze mil euros). O foco da pesquisa é o desenvolvimento de *softwares*, já que o *hardware* é idêntico a todos.

Figura 2.4: *Aldebaran's Nao Humanoids*

Middle Size - os jogos desta categoria são disputados com equipes de até seis robôs que não podem possuir mais que 50cm de diâmetro. A bola usada possui as mesmas dimensões da oficial da *FIFA*[2]. Os robôs podem se

[2]**FIFA** -*Fédération Internationale de Football Association*

comunicar usando rede *wireless*, já que os objetivos aqui são a cooperação, a autonomia e o planejamento;

2. *Federation of International Robot-soccer Association* (**FIRA**) - está mais voltada para as tecnologias de construção de robôs. Sua primeira edição aconteceu na Coréia do Sul, no ano de 1996, e contou com a participação de 23 times de 10 países diferentes (SILVA, 2003). A seguir são apresentadas as categorias da *FIRA*.

   - **HuroSoft** (*Humanoid Robot World Cup Soccer Tournament*): nesta categoria os robôs possuem um limite máximo de altura de 150cm, peso máximo de 30kg e e são bípedes. O vencedor de uma partida, nesta categoria, é determinado pela maior pontuação sobre sete eventos avaliados: *robot dash, penalty kick, obstacle run, lift and carry, weight lifting, marathon, and basketball* (FIRA, 2011);
   - **RoboSot** (*Autonomous Robot-soccer Tournament*): cada robô possui as dimensões 20cm x 20cm x 40cm. As equipes são compostas de um a três robôs, onde um deles pode ser o goleiro, e podem ser autônomos ou semi-autônomos[3];

Figura 2.5: Robôs da Categoria *Robosot* (FIRA, 2011)

   - **MiroSot** (*Micro Robot World Cup Soccer Tournament*): as dimensões máximas para os robôs nesta categoria são 7,5cm x 7,5cm x 7,5cm. A bola utilizada na partida é de *golf* na cor alaranjada. O robô é autônomo, por isso os membros humanos da equipe só devem iniciar seu robô e colocá-lo no local de campo. É considerada a mais fácil em questão de desenvolvimento e implementações de robôs. Nesta categoria uma partida é jogada por duas equipes compostas de três robôs, sendo um deles o goleiro, e é permitido um máximo de três membros

[3]Quando é auxiliado por um computador externo.

humanos por equipe, sendo o técnico, o treinador e o gerenciador da equipe (técnico, treinador e o gerenciador da equipe). Os robôs são compostos por dois motores controlados por um microprocessador, bateria própria e sistema de comunicação sem fio. Cada time deve ter seu próprio *hardware*: uma câmera de vídeo CCD e sistema de aquisição de imagens, um computador, sistema de transmissão de dados e os três robôs (MARTINS et al., 2005) (SCHERRER, 2006);

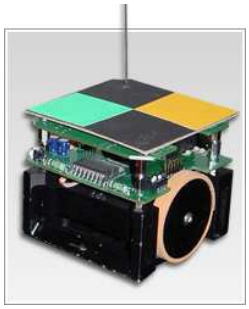


Figura 2.6: Robô da Categoria *Mirosot* (FIRA, 2011)

- **NaroSot** (*Nano Robot World Cup Soccer Tournament*): as dimensões máximas para os robôs desta categoria são 4cm x 4cm x 5cm. A partida é disputada por duas equipes, onde cada uma é composta por cinco robôs e um deles pode ser o goleiro;
- **SimuroSot** (*Simulated Robot-soccer Tournament*): categoria de simulação 3D, onde existe mediação de um servidor[4] no ambiente de jogo. O ambiente de simulação é composto por gráficos em 3D que exibe visualmente o andamento do jogo. A ausência de *hardware* diminui o custo e facilita a disseminação das pesquisas na área (GOLVEIA, 2006).
- **AndroSot** *(Android Soccer Tournament)*: a operação do robô desta categoria mistura métodos de controle automático, para a visão, e controle remoto, para a comunicação *wireless*. O limite de peso do robô é de 600g e altura máxima é de até 50cm (FIRA, 2011);
- **Amiresot** - a partida, nessa categoria, é disputada por um robô e dois técnicos humanos de cada equipe. O robô utilizado aqui é o *Amire Robot*, a bola usada é uma bola de tênis amarela, o robô possui dimensões de 130cmX90cm. Os técnicos humanos só devem iniciar seus robôs e colocá-los no local de campo definido pelo árbitro.

[4] O servidor atua como simulador, mostra os robôs, o campo, o placar, entre outros.

Atualmente a *RoboCup*[5] é a federação mais popular no mundo e o nome vem de uma contração do nome completo da competição, "***Robot Soccer World Cup***".

O principal objetivo da *RoboCup* é: ***"Por volta do ano de 2050, desenvolver uma equipe de robôs humanóides totalmente autônoma capaz de vencer o atual campeão mundial de futebol humano"***(traduzido de http://www.RoboCup.org/about-R Mesmo que este objetivo não seja alcançado, a disseminação das pesquisas em relação ao futebol de robôs já são frutos que a *RoboCup* pode colher. Em pouco mais de uma década a *RoboCup* evoluiu de um pequeno encontro com poucos cientistas para um evento com participação internacional.

O primeiro simulador de futebol de robôs foi desenvolvido pela equipe Noda, no *Electro Technical Laboratory* (ETL), do Japão. Esta equipe anunciou o *Soccer Server* versão 0, escrito em LISP, e logo foi ovacionado como primeiro sistema para o domínio do futebol de robôs com suporte multi-agente. A versão 1.0 do *Soccer Server* já foi escrita em C++, que foi distribuída gratuitamente pela *web*. A primeira demonstração pública deste simulador foi feita na *International Joint Conference* (IJCAI-95) (ROBOCUP, 2010).

O primeiro evento oficial da *RoboCup* aconteceu na cidade japonesa de Nagoya, no ano de 1997, e contou com a participação de 12 equipes internacionais em um total de 40 equipes inscritas. O público presente na conferência foi de mais de 5 mil espectadores, desde então o evento não tem parado de crescer. Em 2006 existiam, por todo o mundo, mais de 300 grupos de pesquisa ativos com trabalhos referentes à *RoboCup*, segundo Strassmann (2006). O número de pesquisadores a utilizar o futebol de robôs como domínio de suas pesquisas vem crescendo desde então, como cita Robocup (2010), não só os cientistas e pesquisadores da área têm interesse como observa-se também investimento privados e públicos, principalmente do governo do Japão. A grande quantidade de desafios presentes e a popularidade mundial do esporte transformaram o futebol na principal abordagem da *RoboCup*.

2.1.1 Simulação 2D da *RoboCup*

A liga de simulação de futebol da *RoboCup*, possui duas modalidades: simulação 2D e 3D. Em ambas a partida é disputada por 2 times, onde cada equipe é constituída por 11 robôs na categoria 2D e quatro na categoria 3D. Os robôs destas categorias são virtuais, ou seja, não apresentam *hardware*.

[5]Ver no site: http://www.RoboCup.org/

O desenvolvimento de uma equipe na categoria 2D demanda a implementação de um algoritmo que irá ser executado no ambiente do simulador, uma vez que não há necessidade de *hardware* específico para os robôs. Esse algoritmo desenvolvido representa o time com todos os seus agentes (jogadores e técnico). Assim é necessário utilizar um simulador[6] disponibilizado pela *RoboCup* para a competição.O simulador é composto por servidor, chamado *Soccer Server* ou *rcssserver*, e encontra-se na versão 14.0.3; por um monitor, chamado *Soccer Monitor* ou *rcssmonitor*, e está na versão 14.1.1 do monitor; e pelo *LogPlayer* na versão 14.0.1. Os três módulos básicos (também válidos para a simulação 3D) são o servidor, o cliente e o monitor. Fraccaroli (2008) explica que:

1. **o servidor** é o módulo responsável por gerenciar os jogadores e o técnico. Recebe e envia informações do ambiente simulado para os clientes (posição da bola, posição do adversário, posição atual do agente, tempo decorrido, entre outras) através de uma comunicação *UDP/IP*;

2. **o cliente** é o módulo responsável por atuar no ambiente simulado. A grosso modo, o módulo cliente é o conjunto de todos os algoritmos que dão instrução para os agentes. Em outras palavras, é a equipe propriamente dita;

3. **o monitor** é o módulo responsável por mostrar o que acontece no jogo através de uma interface gráfica do campo virtual. Por meio dessa interface é possível notar os agentes interagindo de acordo com as informações enviadas pelo servidor. Através do monitor também é possível analisar melhor um jogo através de funções extras. Entre as funções dispostas pelo monitor estão a possibilidade de aplicar *zoom* sobre o campo de futebol, visualizar o nível de energia (*stamina*) de cada jogador e seu campo de visão, o número do jogador, entre outras funcionalidades. A imagem 2.7 mostra o monitor da liga de simulação 2D.

Segundo Xavier et al. (2006), a cada intervalo de simulação, durante uma partida, existe uma sequência lógica de passos (apresentados a seguir) que se repetem até que sejam alcançados um total de seis mil ciclos[7]:

1. o servidor envia, aos clientes e ao monitor, informações sobre posições dos agentes, bola, *flags*, condições do próprio agente, entre outras. Isso pode ser considerado

[6] Veja arquitetura lógica de interação entre o simulador e seus clientes na figura 2.9 da página 20.

[7] Lembrando que um ciclo corresponde a 100ms e que existe um intervalo depois de passados 3000 ciclos.

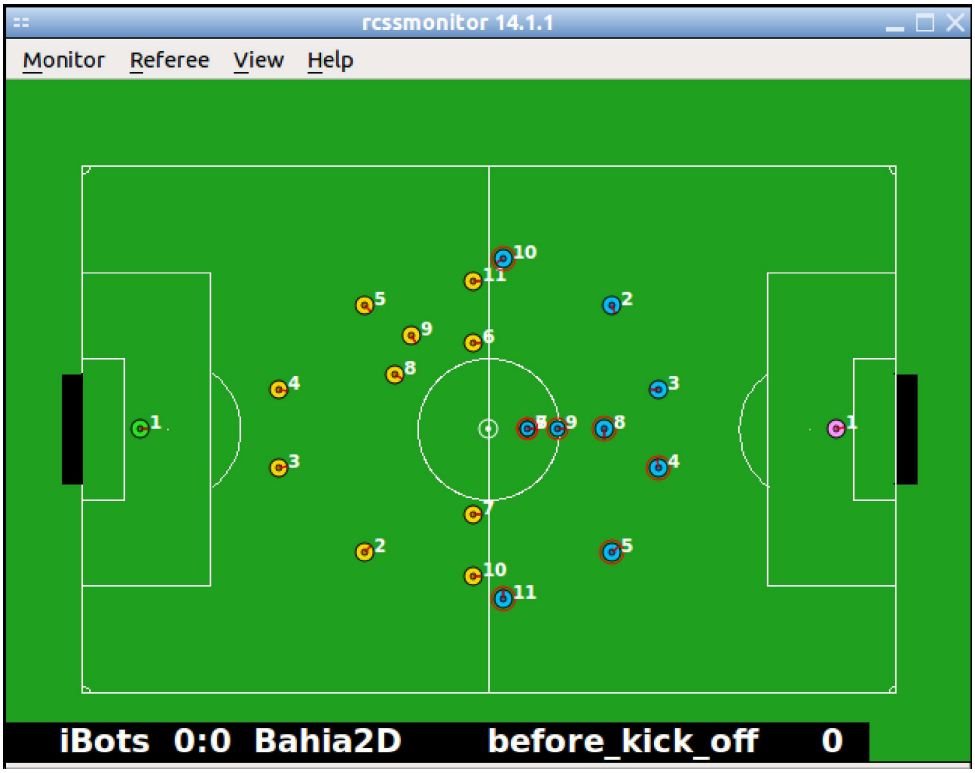


Figura 2.7: Monitor da *Simulation League 2D*

como uma percepção dos jogadores, ou seja, simula a visão e a audição do jogador e o que está acontecendo ao seu redor;

2. quando o monitor recebe as informações do servidor ele as interpreta de forma visual para que o ser humano possa visualizar o que acontece;

3. um cliente, ao receber informações do servidor, analisa (ou espera-se que analise) e toma uma decisão (ou não), enviando uma resposta ao servidor na forma de um comando para ser executado;

4. o servidor calcula novas condições de acordo com as condições anteriores e nos comandos enviados pelos clientes.

A maior competição onde os melhores times tem a oportunidade de testar e comparar seus times é a Copa do Mundo da *RoboCup*. A tabela 2.1, mostra as classificações da liga de simulação 2D dos últimos três anos. Pode-se notar que não houve muita variação dentre as primeiras posições nesses últimos anos, isso pode ser ocasionado pelo fato de novas equipes demandarem de um tempo relativamente grande para atingir o nível das melhores.

Tabela 2.1: Resultados dos últimos três anos da *RoboCup*

Resultado Final - Categoria Simulação 2D						
Col./Ano	2008		2009		2010	
1	DesertEgl	XU	WrightEagle	USC	HELIOS	NIT
2	WrightEagle	USC	Helios	NIT	WrightEagle	USC
3	HfutEngine	HUT	Oxsy	LBU	Oxsy	LBU

XU -Xiamen University (China)

USC - University of Science and Technology of China (China)

HUT - Hefei University of Technology (China)

LBU - Lucian Blaga University (Romênia)

NIT - National Institute of Advanced Industrial Science and Technology (Japão)

Outros Eventos Promovidos pela *RoboCup*

Um evento que recebe incentivos da *RoboCup*, *IEEE*, *SBC*, entre outras instituições é a Competição Latino Americana de Robótica (*LARC*). A *LARC* é um evento anual e caracteriza-se por ser a princial competição de robótica da América Latina. Foi disputada no Brasil entre os anos de 2005 e 2010 e em 2011 será realizada em Bogotá, na Colômbia. Este evento tem como principal objetivo motivar a imaginação, a inovação e o desenvolvimento tecnológico entre os estudantes (ROBOCUP, 2010). Essa competição de robótica também possibilita o desenvolvimento analítico de habilidades para resolver problemas e uso de conhecimentos de mecânica, eletrônica e *software* de uma maneira prática.

A *LARC* 2010[8] aconteceu na cidade de São Bernardo do Campo (imagem 2.8), Estado de São Paulo, e contou com a participação de mais de 600 competidores inscritos em 133 times de oito países diferentes. Especificamente na categoria Simulação 2D, 33 equipes se inscreveram mas somente 9 compareceram à competição. A equipe *iBots* da UFT, resultante dos trabalhos de Silva (2010) e Silva et al. (2010), debutou em competições na LARC 2010 e, infelizmente, foi eliminada na primeira fase (junto com Instituto Militar de Engenharia e a Universidade Estadual da Bahia). A participação na competição serviu para situar o desenvolvimento da equipe *iBots* em relação ao nível dos competidores de Brasil e América Latina e de intercâmbio com outras equipes, resultando na aquisição de informações importantes que refletiram no trabalho de Ferreira (2010). A *OBR*[9] (Olimpíada Brasileira de Robótica) ocorreu anexo à *LARC*. Os resultados dos últimos anos de

[8] *LARC 2010* - veja mais no site da *Joint Conference*: http://www.larc2010.fei.edu.br/

[9] OBR - É um evento de robótica cujo público alvo são crianças e adolescentes (ROBOCUP, 2010). Essa categoria possui uma participação bem escassa, apesar das facilidades que são permitidas às equipes, como a utilização de hardware pré-moldado, por exemplo.

competição estão descritos na tabela 2.2 da página 17.

Tabela 2.2: Resultados dos últimos anos da *LARC*

Resultado Final - Categoria Simulação 2D			
Ano/Col.	1	2	3
2010	GEAR SIM- USP	MECATEAM- UFBA	USPDROIDS- ICMC
2009	GEAR SIM- USP	PET_SOCCER_- UFES	GPR-2D- Unioeste
2008	PET_SOCCER_- UFES	BAHIA 2D- UNEB	MECATEAM- UFBA
2007	PET_SOCCER_- UFES	GARGALOS- UFES	BAHIA 2D- UNEB
2006	CDU- FEI	MECATEAM- UFBA	PET_SOCCER_- UFES
2005	ITANDROIDS- ITA	OXENTETEAM - UFBA	FEISIM– FEI

USP - Universidade de São Paulo - São Carlos/Brasil

UFBA - Universidade Federal da Bahia/Brasil

ICMC - Instituto de Ciências Matemáticas e da Computação da Universidade de São Paulo/Brasil

UNEB - Universidade Estadual da Bahia / Faculdades Integradas da Bahia/Brasil

UFES - Universidade Federal do Espírito Santo/Brasil

ITA - Instituto Tecnológico da Aeronáutica

Unioeste - Universidade Estadual do Oeste do Paraná

Figura 2.8: *Banner* Larc 2010 (LARC, 2010)

As Regras

A ordem durante o jogo e o controle de toda a partida são mantidos por um agente específico do servidor, chamado árbitro. O árbitro verifica se todos os clientes estão agindo em conformidade com as regras, se algum dos parâmetros enviados ao servidor é violado ou assume um valor irregular, entre outros. Entre as atribuições do ábitro também estão as de julgar faltas, aplicar penalidades, controlar o tipo de jogo que segue a cada ciclo e demais eventualidades que possam ocorrer. A atuação do árbitro pode ser caracterizada de dois modos básicos, sendo o primeiro de mediação das ações dos agentes (verifica a consistência

dos comandos enviados) e o segundo modo é o modo de penalização (acontece quando envia mensagens do tipo fonéticas aos clientes conectados ou quando atua diretamente sobre o agente por causa de uma violação de regra). Como as demais mensagens fonéticas, as mensagens do árbitro também são ouvidas por todos os jogadores em campo, com a ressalva de não haver interferência ou ruído, como ocorre com as mensagens de áudio dos clientes. As penalizações aplicadas pelo árbitro vão desde uma simples cobrança de tiro de meta, de lateral e mudança de modo de jogo até a paralização de determinado agente durante alguns ciclos por realizar comando ilegal.

Em campeonatos oficiais as regras da categoria de simulação 2D geralmente não variam muito de um evento para outro. No campeonato de Atlanta 2007 foi definida a versão corrente do simulador e a configuração das máquinas, sendo obrigatório o uso do sistema operacional Linux. Nesta definição cada equipe pode usar até quatro máquinas sem o uso da programação concorrente (ROBOCUP2007, 2007). Geralmente é solicitado às equipes que participam da competição que escrevam um *script* padrão, pois todos os jogos são iniciados automaticamente por um *script manager* da liga (ou por um árbitro humano).

A respeito da manutenção técnica da equipe, é importante enfatizar que, em campeonatos oficiais, alterações nos códigos binários podem ser realizadas somente até 15 minutos antes de começar o primeiro jogo da liga. Também é solicitado que cada equipe forneça nove *scripts* de *start* (iniciar) do time e *kill* (matar) o algoritmo de sua equipe. Esses *scripts* devem ser obrigatoriamente chamados *start1, start2, start3, start4, kill1, kill2, kill3 e kill4*, e devem ser colocados em local lógico especificado (ROBOCUP2007, 2007). Geralmente também é citada nas regras de torneios a técnica de execução de códigos, que será utilizada para o teste dos binários de cada equipe. Além disso, é solicitado que o código tenha o mínimo de legibilidade.

Assim como nos esportes, no futebol de robôs também existe uma espécie de código de ética, chamado *Fair Play*. A intenção deste código é tornar o jogo de futebol justo, dentro das regras e restrições impostas pelo simulador virtual. A evasão destas restrições é considerada violação do compromisso de *fair play* e é severamente penalizada pela equipe organizadora.

Existe também a questão da participação remota, que só é permitida em casos extremos. Há a possibilidade de acontecerem problemas com a equipe que podem ser mais difíceis de serem solucionados com uma participação remota. Talvez esse tipo de participação seja um dos pontos mais variáveis entre cada campeonato, pois é uma demanda que existe e que é importante que seja suprida sem que haja algum tipo de favorecimento

para qualquer equipe.

2.1.2 A Comunicação na Simulação 2D da *RoboCup*

Na categoria de simulação 2D da *RoboCup* o principal objetivo é a colaboração entre agentes heterogêneos em um ambiente dinâmico e não-determinístico tentando alcançar uma meta comum. Esse ambiente é dinâmico porque no futebol, tanto no robótico quanto no humano, o ambiente altera enquanto o agente decide e não determinístico porque uma ação não tem um efeito conhecido, direto e garantido. Por outro lado, por exemplo, no xadrez o ambiente é estático e determinístico. No caso do futebol, o ambiente dinâmico é decorrente das posições dos jogadoes, da velocidade da bola, da quantidade de energia (*stamina*) que cada um possui, do esquema tático adotado, entre outros.

No futebol de robôs uma importante ferramenta é a comunicação, essencial para a colaboração direta entre jogadores de uma equipe. A comunicação pode ser usada de diversas formas dependendo das regras referentes às diferentes ligas. Na liga de robôs físicos, por exemplo, a comunicação pode ser realizada através de transmissão por ondas de rádio, já nas ligas de simulação 2D e 3D a comunicação é mediada por um programa específico, o servidor de mensagens.

A Arquitetura do Servidor

Ao contrário do que ocorre na maioria das aplicações de Sistemas Multi Agentes (SMA's), no domínio do futebol de robôs simulado os agentes não podem se comunicar diretamente uns com os outros. Todos os jogadores utilizam o mesmo canal de comunicação e possuem uma largura de banda reduzida. Isso é explicável porque existe um mediador na comunicação entre os agentes, o servidor *Soccer Server*.

Uma partida de futebol de robôs é disputada por dois times. Cada time é composto por 11 jogadores[10] que se conectam ao simulador, em uma arquitetura **cliente-servidor**, através de *sockets UDP/IP*. Cada cliente tem um *socket*[11] dedicado, ligado diretamente ao *Soccer Server*, veja figura 2.9 da página 20. Normalmente tem-se 22 *sockets* de jogadores, mais dois *sockets* de cada técnico e o *socket* do *Soccer Monitor*[12], ligados simultaneamente

[10] Além dos jogadores, que são agentes, existe também o agente técnico, chamado *coach*, que não participa diretamente do jogo, porém também faz parte da equipe e pode ser eventualmente usado para mudanças táticas, por exemplo.

[11] *Socket* é uma interface para a comunicação via rede.

[12] Existem casos, principalmente em campeonatos oficiais, onde mais de um monitor pode estar conectado ao *Soccer Server*.

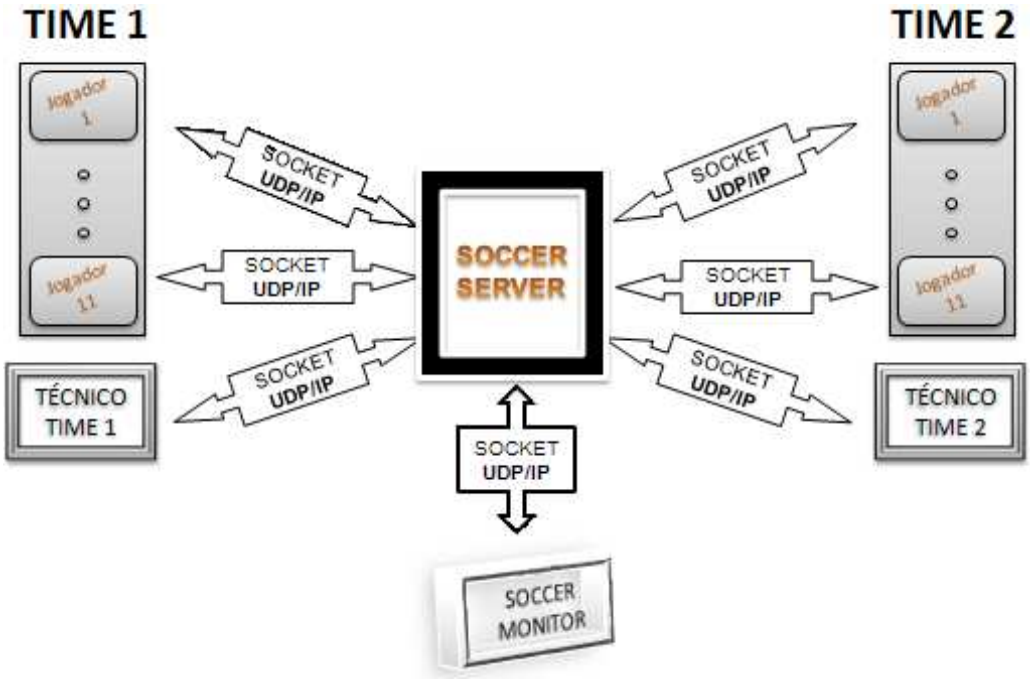


Figura 2.9: Interação simulador-clientes. Adaptado de Silva e Santos (2010)

ao *Soccer Server*, totalizando 25 *sockets*.

A comunicação deve ser realizada exclusivamente via troca de mensagens, isso vale tanto para os clientes quanto para o *Soccer Monitor*. O quadro de mensagens é onde ficam armazenadas as requisições enviadas pelos clientes e as respostas enviadas pelo servidor. Existe um nível de acesso regulado ao quadro de mensagens do *Soccer Server*. O árbitro é o único que pode acessar diretamente o quadro de mensagens do *Soccer Server*, os demais clientes só podem acessar as mensagens armazenadas depois delas serem analisadas pelo árbitro, ver figura 2.10 da página 21. É o árbitro quem define o que pode ou não ser executado e mostrado pelo *Soccer Monitor*. O resultado das ações dos clientes é exibido no *Soccer Monitor*, mas antes as ações passam pelo simulador de campo, que é onde os jogadores colhem informações sobre o estado atual de jogo para atualizar seu modelo de mundo[13]. Note que os agentes jogadores obtêm as informações do modelo de mundo, enquando os agentes técnicos têm o privilégio de acessar a informação sem as interferências acrescidas pelo simulador de campo, ou seja, diretamente do quadro de mensagens, ver figuras 2.10 e 2.11 das páginas 21 e 22 respectivamente.

Para manter um alto nível de fidelidade aos acontecimentos relacionados ao futebol, o *Soccer Server* encarrega-se de limitar a distância a que uma determinada mensagem do tipo fonética enviada por um jogador pode ser ouvida. Há limitações também no que diz

[13] As informações sensoriais, obtidas pelo robô, são interpretadas a fim de gerar uma modelo do ambiente de modo que possa tomar decisões sobre este modelo. Também chamado de visão de mundo.

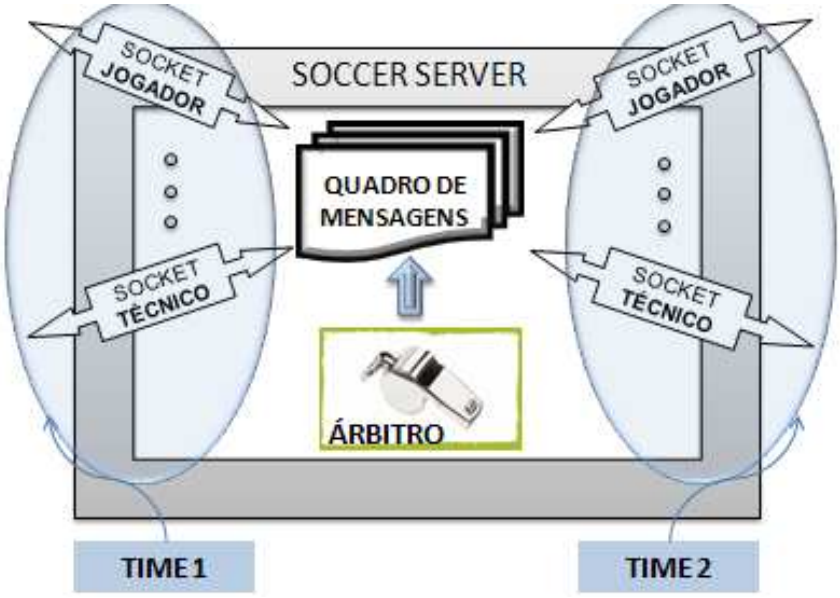


Figura 2.10: Acesso ao quadro de mensagens. Adaptado de Almeida (2008)

respeito à consistência das mensagens, já que quanto maior a distância que o jogador que ouve a mensagem está do jogador que fala, pior será a qualidade[14] da mensagem recebida.

De modo geral, as principais premissas de comunicação do futebol de robôs são:

- *(**say** "mensagem")* - a *"mensagem"* é enviada em *broadcast* para todos jogadores dentro de uma distância padrão definida pelo *Soccer Server* e armazenada em uma constante de nome *audio-max-dist*, que atualmente corresponde a 50m;
- *(**hear** "mensagem")* - assim que um comando de fala é recebido, o servidor repassa para todos os jogadores[15] a mensagem recebida através de um comando *hear*, que pode possuir diversos argumentos.

2.1.3 A Comunicação Jogador X Jogador

A comunicação entre os jogadores de um mesmo time é realizada através de mensagens fonéticas enviadas em *broadcast*[16]. Não existem limitações relativas à quantidade de mensagens enviadas, porém existe um limitante relacionado ao número de mensagens que podem ser ouvidas. É adotado pelo servidor que cada jogador pode ouvir somente duas mensagens por ciclo (BOER; KOK, 2002).

[14] A qualidade da mensagem é diminuída através da atribuição de ruído, embutido na mensagem fonética pelo simulador de campo do *Soccer Server*.
[15] Dentro da distância de áudio definida nas características do servidor.
[16] A mensagem fonética enviada por um jogador é ouvida por todos outros jogadores dentro de uma distância de áudio definida pelo Servidor.

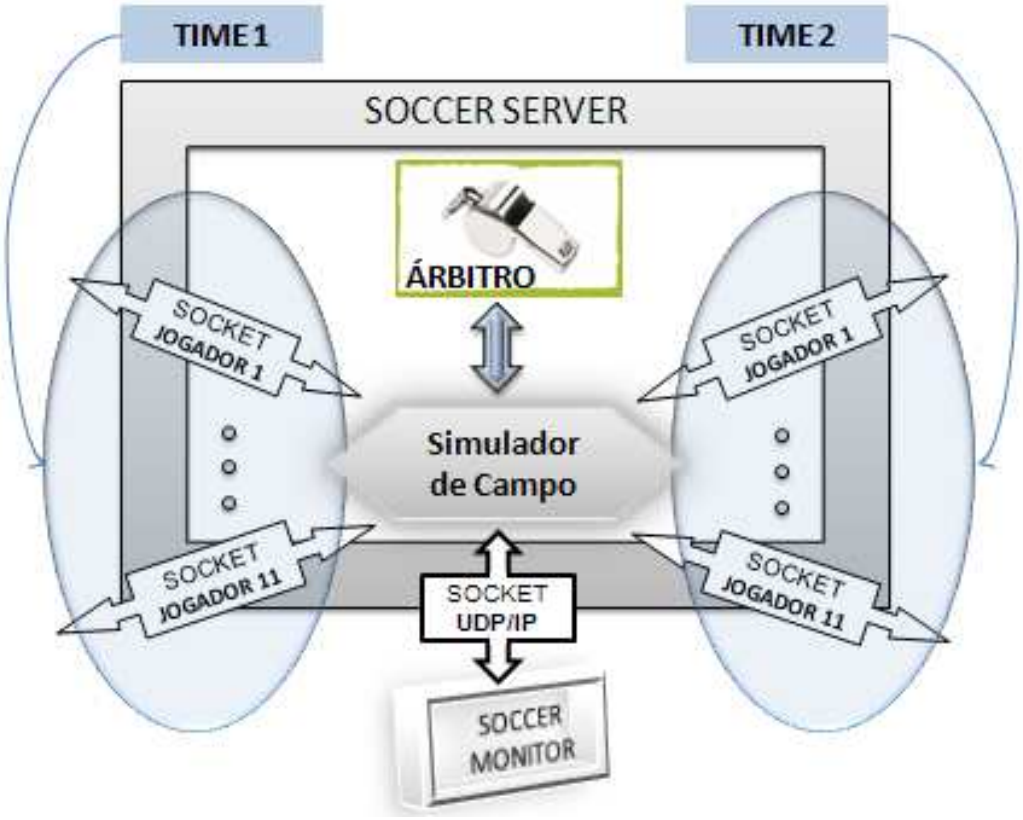


Figura 2.11: Acesso ao simulador de campo. Adaptado de Fraccaroli (2008)

As mensagens dos jogadores possuem também um limitante de tamanho, também definido pelo servidor, que corresponde atualmente a 10 *bytes*. Nas mensagens podem ser usadas quase todos os caracteres *ASCII*. O time pode utilizar esse alfabeto disponível para trocar mensagens, atualizar dados e realizar jogadas coletivas. Deve-se lembrar que pode ser útil incluir no corpo da mensagem mecanismos que possibilitem a conferência, correção e/ou descarte das mensagens.

As mensagens fonéticas de um determinado agente são enviadas para todos os jogadores dentro de uma distância de alcance de aúdio, definida por uma constante de nome *audio_cut_dist*. Um agente pode chamar um método que direciona sua atenção para outro jogador invocando o comando ***attention-to****(objeto)*, que recebe como argumento o objeto da atenção. Com isso, as mensagens fonéticas enviadas pelo jogador, foco de atenção, terão privilégio ao chegar ao *buffer* do jogador que ouve. As mensagens fonéticas enviadas por um jogador são recebidas no ciclo seguinte pelos outros jogadores em campo, não havendo atraso temporal de mais de um ciclo, como ocorre com as mensagens do técnico. A sintaxe de comando de envio para os jogadores é: *(**say** "mensagem")*. Já a formação com a qual o jogador recebe a mensagem é variável e precisa ser tratada para que se possa avaliar o áudio capturado. São variações das mensagens de áudio percebidas pelos

jogadores:

- *(**hear** ciclo direcao "mensagem")* - quando o jogador que ouve a mensagem não sabe ao certo de qual jogador está ouvindo, então é dada somente a direção da qual a mensagem vem. A direção dada pode variar de [-180 , 180], indicando o ângulo de onde vem a mensagem ouvida, ou seja, o agente não reconhece o emissor mas sabe de qual região veio a mensagem;
- *(**hear** ciclo jogadorOrigem "mensagem")* - especifica qual jogador enviou a mensagem. Inclui auto-mensagens, no caso de o jogador ouvir uma mensagem que ele próprio enviou;
- *(**hear** ciclo árbitro "mensagem")* - mensagens enviadas pelo árbitro tem maior prioridade que as outras mensagens;
- *(**hear** ciclo técnico "mensagem")* - são as mensagens enviadas pelo técnico do time.

A Comunicação Técnico X Jogador

O técnico é um agente privilegiado e o bom uso de sua comunicação pode ter efeitos positivos para o time. Como o técnico pode receber as percepções do campo sem qualquer erro e/ou ruído, sua análise de jogo é considerada mais confiável. Não há distância limitante de áudio para dissipação das mensagens do técnico, tal como do árbitro. Porém, essas mensagens incorrem sobre outro problema, que é a demora de recepção pelos jogadores. Outro empecilho é que a comunicação é mais limitada do que a dos demais agentes. O agente técnico tem um número restrito de vezes que pode se comunicar com seu time e também existe um intervalo de tempo mínimo, 300 ciclos entre duas mensagens enviadas por ele (BOER; KOK, 2002).

Para balancear as limitações quanto ao número de vezes que o técnico pode se comunicar com seu time, com os privilégios concedidos pelo simulador, são definidos formatos especiais para suas mensagens. O técnico pode enviar mensagens de cinco tipos básicos: *info, advice, define, meta, freeform*[17]. As mensagens do tipo *info, advice, define e meta* sofrem atraso temporal, pois podem ser enviadas durante a fase de jogo (modo *play_on*), ou seja, quando a bola não está parada; já as mensagens *freeform* são encaminhadas sem atraso temporal aos jogadores, pois só podem ser lançadas em momentos de *non-play-on*

[17]Mensagens *freeform* do técnico não possuem limitante de tamanho de 10 *bytes* como as dos demais clientes. O único limitante desse tipo de mensagem é a quantidade de vezes que pode ser usada.

(curto momento de jogo em que a bola está parada e é sinalizada como de posse de uma das equipes). Independente do tipo, existe um limitante no número de mensagens que o técnico pode enviar.

Existem dois tipos de técnico: o *online-coach* e o *trainer*. O *online-coach* pode ser usado em partidas oficiais, já o *trainer* tem maior liberdade e pode ser usado somente para treinar o time durante a fase de desenvolvimento (equivalente ao período de treinamento e preparação tática do futebol). O *trainer* pode desativar o ábitro automático e mover a bola e os jogadores.

Aplicação da Comunicação Técnico X Jogador

A aplicação do mecanismo de comunicação do técnico, descrita por Ferreira (2010), resultou em uma melhora perceptivelmente positiva na equipe *iBots*. Para isso, o técnico foi programado para analisar o andamento da partida e decidir o esquema tático a ser utilizado (5-4-1, 4-5-1, 4-4-2, 4-3-3, 3-5-2, 3-4-3).

Na categoria de simulação 2D, cada equipe recebe os jogadores de seu time de um sorteio realizado pelo *Soccer Server* no início de cada partida. Esses jogadores são retirados de uma lista com 18 tipos distintos, diferenciando-se basicamente em: aceleração máxima, velocidade máxima, área de chute, taxa de descanso e erro de chute.A utilização dessa heterogeneidade de forma a distribuir os agentes taticamente em campo, de acordo com suas características, é um bom exemplo de aplicação da comunicação técnico x jogadores. Pode-se destacar ainda, como outras formas da utilização de mecanismos de comunicação do técnico, a selecção de jogadores para atuarem em determinadas zonas do campo ou auxiliar no desenvolvimento de habilidades que facilitem a comunicação para reposicionamento dos agentes e/ou marcação de jogadores oponentes.

2.1.4 Linguagens de Comunicação entre Agentes

Quando existe um ambiente multiagentes, é comum o uso de uma linguagem de comunicação que seja responsável pela interação através de troca de mensagens entre os agentes envolvidos. No futebol de robôs, isso não é possível entre os agentes jogadores, porém o agente técnico pode usar uma linguagem de comunicação para trabalhar com sua equipe.

Dentre as várias Linguagens de Comunicação entre Agentes (LCAs) existentes, a ***KQML*** (*Knowledge Query and Manipulation Language*) é a mais mencionada e adotada

nos mais diversos ambientes multiagentes por ter uma estrutura simples e várias características importantes para aplicação no futebol de robôs. Segundo Scherrer (2006), são propriedades básicas da *KQML*:

- possibilidade de usar qualquer linguagem de programação para escrever o conteúdo das mensagens;
- as informações necessárias para entender o que há nas mensagens já estão contidas na própria comunicação;
- o mecanismo de transporte da troca de mensagens é transparente. A mensagem sai do emissor e chega no receptor.

A tabela 2.3 da página 25 mostra os parâmetros de uma mensagem *KQML*. Observe que esses parâmetros são bastante significativos, possuindo nomes que têm ligação direta com sua funcionalidade. Logo em seguida, na tabela 2.4 da página 26, é mostrado um segmento de código, possível de ser montado em *KQML*, que descreve a sintaxe usada na linguagem. Essa descrição é dada com um exemplo de como a linguagem é utilizada na prática, veja na sequência.

Tabela 2.3: Parâmetros de uma mensagem *KQML*.(REIS, 2003).

Parâmetro	**Significado**
`:content`	Conteúdo da mensagem
`:sender`	Emissor da mensagem
`:receiver`	Receptor da mensagem
`:language`	Linguagem do conteúdo da mensagem
`:ontology`	Ontologia utilizada no conteúdo da mensagem
`:force`	Especifica se o conteúdo da mensagem é definitivo ou o emissor o poderá alterar no futuro
`:reply-with`	Definição se o emissor da mensagem aguarda por uma resposta e se tal for verdadeiro, qual o identificador para essa resposta
`:in-reply-to`	Referência ao identificador de resposta fornecido por um reply-with prévio

O exemplo da tabela 2.4 mostra que o agente <*agenteEmissor*>, envia para o agente <*agenteReceptor*> uma mensagem cujo conteúdo é <*conteudoMensagem*>. Ainda são especificadas nessa mensagem a linguagem (<*linguagem*>) e a ontologia (<*ontologia*>) utilizada. O identificador <*performativa*>, poderia ser, neste caso, *ask-one*, o que significa

que uma resposta é esperada. A resposta a esta mensagem deve ser dada contendo o identificador <*identificadorResposta*>.

Tabela 2.4: Exemplo de código de uma mensagem *KQML*

```
(<performativa>
      :content ( <conteudoMensagem> )
      :sender <agenteEmissor>
      :receiver <agenteReceptor>
      :reply-with <identificadorResposta>
      :language <linguagem>
      :ontology <ontologia>
)
```

Junto com a *KQML* é largamente utilizado o formato ***KIF***(*Knowledge Interchange Format*). O *KIF* destina-se a representar explicitamente o conhecimento sobre um domínio de discurso específico. Foi desenvolvido primariamente como forma de definir o conteúdo de mensagens expressas em *KQML*. A linguagem dá suporte à utilização de estruturas de tipos de dados simples como *string, números* e *caracteres*. Possui também operadores lógicos (*and, or, not, etc*) pois a linguagem é baseada na lógica de predicados de primeira ordem com extensões para o suporte de raciocínio não monótono e de definições (REIS, 2003). A descrição da linguagem inclui a especificação da sintaxe e da semântica.

Outra linguagem de comunicação multi-agente bem difundida é a *FIPA*[18] *ACL* (*Agent Communication Language*), que é semelhante em formato à *KQML* (REIS, 2003). A *FIPA* procura providenciar uma semântica mais compreensível do que a linguagem *KQML*. Uma vantagem da *FIPA* com relação à *KQML* é a disponibilização de performativas mais claras e específicas para a tarefa que deve ser executada, incluindo negociações. Veja a variedade de performativas na tabela B.2 da página 80.

O desenvolvimento de uma LCA é de extrema importância para a troca de mensagens e consequente melhora na coloboração entre os agentes no futebol de robôs. Para isso a linguagem deve fornecer uma sintaxe, uma semântica e uma pragmática bem definidas. Com a reunião destes três recursos da linguagem é possível associar um significado para as mensagens desejadas (SCHERRER, 2006).

A primeira linguagem padrão de alto nível utilizada para treinar uma equipe de fu-

[18] ***FIPA*** – Foundation for Intelligent Physical Agents

tebol de robôs foi a *Coach Unilang* (ALMEIDA, 2008). Esta é uma linguagem baseada em conceitos comuns do futebol real e robótico, tais como: regiões, períodos de tempo, situações, táticas, formações, tipos e comportamentos de jogadores.

A *RoboCup* optou por usar uma linguagem de baixo nível chamada *CLang*, como a linguagem padrão no simulador oficial. No entanto, devido à sua ineficácia, foram introduzidos conceitos originais da linguagem de alto nível *Coach Unilang* (ALMEIDA, 2008). Atualmente as duas linguagens são aceitas na versão corrente do servidor.

2.1.5 Algumas *Frameworks* Utilizadas

Na categoria de simulação 2D da *RoboCup*, diversas equipes disponibilizam partes de seu código fonte sob forma livre para o desenvolvimento e aperfeiçoamento de novas equipes. Esses códigos são as chamadas *framework*. Com as *frameworks* é possível começar um desenvolvimento guiado, ou seja, não existindo a necessidade de iniciar totalmente do zero uma equipe. As *frameworks* mais conhecidas atualmente são o *UvaTrilearn, Agent2D* e o *Wrighteagle*, todas contêm uma modelagem das habilidades básicas que os jogadores devem possuir.

O *UvaTrilearn* é uma *framework* das mais largamente utilizadas até hoje, se não a mais utilizada, por diferentes equipes. O código fonte dessa *framework* está se tornando uma ferramenta obsoleta, motiva pela descontinuidade de atualizações do projeto que não seguem novos recursos permitidos pelo simulador da *RoboCup*. A última atualização do *UvaTrilearn* data de 2003 (FERREIRA, 2010). As ferramentas ligadas ao futebol de robôs estão em constante e rápida atualização, assim também as *frameworks* incorrem nessa necessidade. A ferramenta que veio substituir o *UvaTrilearn* em popularidade é a *framework Agent2D*. A *Agent2D* possui ampla cobertura com relação à habilidades básicas, isso permite ao desenvolvedor centrar seu trabalho na construção de habilidades mais complexas. As vantagens da utilização do *UvaTrilearn* atualmente se restringem à documentação abundante, o que é indicado para equipes que estão iniciando seus estudos na área de futebol de robôs. Sobre o *Agent2D*, existe uma pequena faixa de documentação bem escassa em inglês e somente um manual escrito em Japonês, o que dificulta o entendimento da estrutura de funcionamento da *framework*.

As principais *frameworks* disponíveis:

- *UvaTrilearn* - é uma *framework* disponibilizada pelo time de futebol de robôs simulado da Universidade de Amsterdã, da Holanda. Trata-se de uma equipe de

simulação 2D bastante conhecida pela sua documentação e, talvez por isso, a sua *framework* serve como ponto de partida para equipes iniciantes nesta categoria;

- *Agent2D* - uma *framework* que está sendo largamente usada atualmente por diversos times de futebol em simulação 2D. A *Agent2D* foi desenvolvida pelo Instituto Nacional de Ciência e Tecnologia Industrial Avançada (*National Institute of Advanced Industrial Science and Technology*) do Japão para a equipe *Helios* (atual campeã mundial), porém como principal diferença a existência de um nível estratégico mais refinado do que a *framework UvaTrilearn* (AKIYAMA; SHIMORA, 2010). A *Agent2D* está licenciada sob a licença *GPL*;

- *WrightEagle* - é uma equipe chinesa desenvolvido pela *University of Science and Technology of China* para participar da *RoboCup* (CHEN et al., 2009). É a *framework* do time que ficou na segunda colocação do mundial da *RoboCup*, em 2010, realizado em Singapura.

2.1.6 Influência da *Framework*

Em 2010, a categoria de Simulação 2D de futebol da *RoboCup* na *LARC* contou com a participação de 9 equipes brasileiras. Para ser mais específico, foram três times representando instituições de São Paulo (*MauaBots, Gear, USPDroids-2D*), dois times representando instituições da Bahia (*MecaTeam, Bahia_2D*), dois times representando instituições do Espírito Santo (*Pet_Soccer_2D, Pahnois*), um time representando instituição do Tocantins (*iBots*) e um time representando o Rio de Janeiro (*RobotIME*).

Tabela 2.5: Classificação final da *LARC 2010*

Classificação	Equipe	Instituição	*Framework* Utilizada
1	GEAR	EESC-USP	*AGENT2D*
2	MECATEAM	UFBA	*AGENT2D*
3	USPDROIDS-2D	ICMC-USP	*AGENT2D*
4	PET_SOCCER_2D	UFES	*UVATRILEARN*
5	PAHNOIS	UFES	*UVATRILEARN*
6	MAUABOTS	IMT	*UVATRILEARN*
7	BAHIA_2D	UNEB	*UVATRILEARN*
8	IBOTS	UFT	*UVATRILEARN*
9	ROBOIME	IME	*UVATRILEARN*

Nota-se que a escolha da *framework* para desenvolvimento do time influenciou na classificação final das equipes, apresentada na tabela 2.5. As primeiras posições foram

conquistadas por equipes que utilizaram a *framework Agent2D* como time-base. Isto pode ser justificado pelo fato da *framework* fornecida pelo *UvaTrilearn* não ter acompanhado as recentes atualizações tanto do simulador quanto do próprio código-base. Essas constatações fizeram a equipe *iBots*, que usava a *framework UvaTrilearn*, a adotar como *framework* a *Agent2D* após a *LARC* (FERREIRA, 2010).

2.2 Inteligência Artificial

A Inteligência Artificial (IA) é uma ampla área de pesquisa da computação dedicada a buscar métodos ou dispositivos computacionais que possuam ou simulem a capacidade racional de resolver problemas, pensar ou, de alguma forma, ser inteligente. Existem vários campos de estudo dentro da IA com o objetivo de prover a capacidade de raciocínio nas máquinas. Entre as muitas técnicas de IA, cita-se algumas das mais utilizadas: *Algoritmos Genéticos*, *Lógica Fuzzy*,*Redes Neurais Artificiais*.

Sistemas de Inteligência Artificial Distribuída permitem que vários processos autônomos, chamados agentes (organizados em sociedades de agentes ou Sistemas Multi-Agentes), realizem atos de inteligência global por meio de processamento local e comunicação interprocessos.

Nas próximas subseções são descritos mais detalhadamente os dois principais ramos da IA abordados neste trabalho: Sistemas Multi-Agentes e Lógica *Fuzzy*. Os Sistemas Multi-Agentes são necessários, de acordo com a arquitetura da categoria de Simulação 2D da *RoboCup*, uma vez que os jogadores e o técnico de cada uma das equipes são agentes. Já a Lógica *Fuzzy* foi adotada na implementação do sistema de tomada de decisão para avaliar o ambiente de jogo (reproduzido nos modelos de mundo de cada um dos agentes) e definir qual jogada coletiva será executada em um determinado momento.

2.2.1 Sistemas Multi-Agentes

Sistemas Multi-Agente (*SMA*) são os sistemas compostos por múltiplos agentes autônomos que possuem um comportamento social. Antes de aprofundar sobre *SMA*, é importante conhecer o que é um agente e quais suas características. Somente na sequência são apresentadas algumas aplicações e trabalhos sobre *SMAs*.

Segundo Rezende (2005), um agente é uma entidade real ou virtual, capaz de agir em um ambiente, de se comunicar com outros agentes, que é movida por um conjunto de inclinações (sejam objetivos individuais a atingir ou uma função de satisfação a otimizar); que possui recursos próprios; que é capaz de perceber seu ambiente (de modo limitado); que dispõe (eventualmente) de uma representação parcial deste ambiente; que possui competência e oferece serviços; que pode eventualmente se reproduzir e cujo comportamento tende a atingir seus objetivos utilizando as competências e os recursos que dispoõe e levando em conta os resultados de suas funções de percepção e comunicação, bem como suas representações internas.

De acordo com Reis (2003) e Carvalho (2004), os agentes podem ser de duas espécies básicas:

- reativos - os agentes reativos consideram apenas as informações correntes de um ambiente para a tomada de decisões, ou seja, eles não possuem uma "memória" para armazenar suas "experiências", agindo por "instinto". Um agente reativo apresenta um modelo de funcionamento por *ação X reação / estímulo X resposta* (PINTO, 2008). Se esse agente é puramentes reativo irá sempre reagir executando ações previamente definidas que são diretamente resultantes dos estímulos por ele percebidos;

- deliberativos - também conhecidos como agentes cognitivos, os agentes deliberativos são aqueles que aprendem ao interagir com o ambiente e utilizam essas experiências para auxiliar em uma tomada de decisão futura. A principal característica deste tipo de agente é a presença de um mecanismo de planejamento. Os sistemas que trabalham com agentes deliberativos geralmente realizam tarefas mais complexas que os agentes dos sistemas multi-agentes reativos. Um agente cognitivo possui ainda, além dos sistemas de recepção e percepção, um mecanismo de revisão e outro de raciocínio e decisão, o que dá ao agente um "caráter social" sobre os outros agentes (PINTO, 2008).

As agentes podem interatuar por meio de cooperação indireta, ou seja, sem a existência de comunicação entre eles de acordo com inferência de uns sobre as ações de outros (situação da equipe *iBots* antes desta monografia). No entanto os agentes também podem se comunicar e, de acordo com Fernández (1998), as formas de comunicação mais comuns entre agentes são:

- comunicação primitiva: a comunicação restrita a um conjunto de sinais (na maioria das vezes 2) com interpretação fixa;

- quadro-negro (*blackboard*): a sociedade de agentes não se comunicam diretamente, mas sempre por meio do quadro-negro - uma espécie de mural;

- troca de mensagens (*message passing*): agentes podem se comunicar sem intermediários, ou seja, diretamente uns com os outros enviando mensagens assíncronas para um ou mais agentes. Os agentes devem conhecer o seu entorno para saber exatamente a quais agentes devem enviar a mensagem;

- comunicação federativa (*federal system*): pode ser considerado um sistema de troca de mensagens em que o número de agentes presente na sociedade é muito grande, onde o tempo computacional necessário para enviar uma mensagem em broadcast é muito alto.

O tipo de comunicação permitida pelo simulador da *RoboCup* durante uma partida é a troca de mensagens (*message passing*), mediada pelo árbitro, mas durante o treinamento a comunicação por quadro-negro é suportada.

Os agentes que fazem parte de um *SMA* podem apresentar diversos graus de interação entre si. As interferências de um agente sobre o ambiente ou sobre outro agente podem representar consequências positivas, negativas ou neutras. Pode-se observar alguns tipos de interação entre agentes na tabela B.1 retirada de (RUSSELL et al., 2003), no apêndice deste trabalho. Essas interações podem ser contínuas ou representar mudanças de tipo com o passar do tempo. Por exemplo, em um dado momento um agente pode ter uma relação de competição com outro, mas passado esse fato, ambos passam a conviver em cooperação quando pertencentes a uma mesma sociedade.

Em geral os *SMAs* incluem diversos agentes que são por vezes homogêneos outras heterogêneos. Cada agente atua de forma autônoma tentando resolver os problemas a ele propostos, atuando assincronamente com relação aos outros inseridos no mesmo ambiente. Em um sistema, os agentes podem participar de diversas organizações e possuir uma esfera de influência modificável, ver figura 2.12 da página 32.

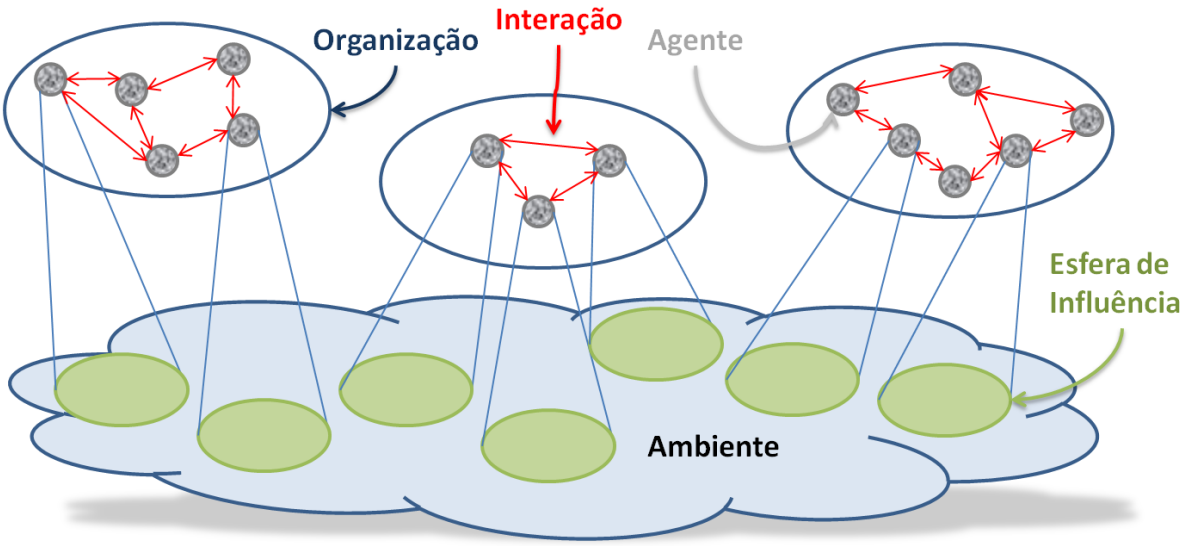


Figura 2.12: Estrutura de um Sistema Multi-Agente. Adaptado de (REZENDE, 2005)

Reis (2003) apresenta algumas vantagens do uso de SMA na resolução de problemas::

1. resolução mais rápida de problemas devido ao processamento concorrente;
2. diminuição da comunicação devido ao processamento estar localizado junto à fonte de informação e a comunicação ser realizada a alto-nível;
3. aumento da flexibilidade e escalabilidade resultantes da possibilidade de interconexão de múltiplos sistemas com arquiteturas distintas;
4. aumento da tolerância a falhas devido à inexistência de um ponto singular de falha;
5. aumento da capacidade de resposta devido aos sensores, sistemas de processamento e atuadores estarem localizados em conjunto, no interior dos agentes;
6. facilidade acrescida de desenvolvimento de sistemas devido à modularidade resultante da decomposição dos problemas e da decomposição dos sistemas em agentes semi-autônomos.

2.2.2 Lógica *Fuzzy*

Também chamada de lógica nebulosa ou difusa, provê um método de traduzir expressões verbais, vagas, imprecisas e qualitativas, comuns na comunicação humana em valores numéricos, o que facilita converter a experiência humana em uma forma compreensível pelos computadores (SHAW; SIMõES, 1999). Para exemplificar melhor, enquanto a lógica bivalente tradicional consegue diferenciar somente preto e branco (um ponto ou é branco ou é preto), a lógica *fuzzy* consegue distinguir também os tons de cinza (é um tipo de lógica multivalente), ver figura 2.13 da página 33.

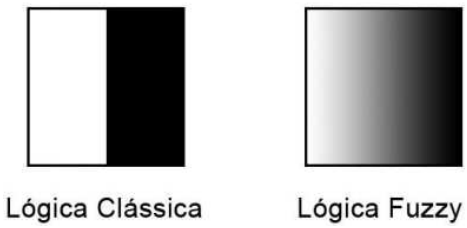


Figura 2.13: Lógia Fuzzy X Lógica Clássica (KOHAGURA, 2007)

Antes de ir mais a fundo sobre lógica *fuzzy*, é preciso entender melhor o conceito de conjunto *fuzzy* definido por *Lotfi Zadeh* (ZADEH, 1965). A grosso modo, a *Teoria dos Conjuntos Fuzzy* diz que dado um elemento que pertence a um domínio, é verificado o grau de pertinência do elemento em relação ao conjunto. Cada conjunto *fuzzy* é caracterizado pela função de pertinência, geralmente representada por $\mu(x)$. Essa função pode

assumir os mais diferentes formatos, tendo que respeitar algumas condições, dentre elas, ser convexa e normal.

Como citado por Sandri e Côrrea (1999), em seu trabalho, formalmente pode-se definir o termo conjunto *fuzzy*, como mostra a definição a seguir:

Definição 2.2.2.1 *Formalmente, um conjunto fuzzy A do universo de discurso* Ω *é definido por uma função de pertinência* $\mu_A : \Omega \rightarrow [0,1]$. *Essa função associa a cada elemento x de* Ω *o grau* $\mu_A(x)$, *com o qual x pertence a A. A função de pertinência* $\mu_A(x)$ *indica o grau de compatibilidade entre x e o conceito expresso por A:*

- $\mu_A(x)$=*1 - indica que x é completamente compatível com A;*
- $\mu_A(x)$=*0 - indica que x é completamente incompatível com A;*
- *0*<$\mu_A(x)$<*1 - indica que x é parcialmente compatível com A, com grau* $\mu_A(x)$.

Tendo como base a definição descrita em 2.2.2.1, nota-se que a lógica *fuzzy* pode tratar as mesmas situações que ocorrem na lógica tradicional. Isso incorre no caso especial em que $\mu_A : \Omega \rightarrow \{0,1\}$, ou seja, a pertinência ou exclusão ao conjunto é total, tornando então esse conjunto *"crisp"*[19].

Operadores *Fuzzy*

Na lógica tradicional as operações com conjuntos são possibilitadas atomicamente pelos operadores binários **E** (interseção) e **OU** (união) e pelo operador unário **NÃO** (complemento). Do mesmo modo, na lógica *fuzzy* existem esses operadores. *Zadeh* expressou que os operadores *fuzzy* devem, para as mesmas entradas, fornecer as mesmas saídas dos operadores da lógica bivalente tradicional. *Zadeh* propôs para o operador **E** o mínimo: *Mín*[μ_A(x), μ_B(y)]; para o operador **OU** o máximo: *Máx*[μ_A(x), μ_B(y)]; e para o operador **NÃO** o complemento: *1-* μ_A(x). No entanto, outros operadores **E** e **OU** foram propostos na literatura. Os principais operadores **E** (interseção) são apresentados na tabela 2.6 e os operadores **OU** (união) na tabela 2.7.

Além dos operadores *fuzzy* citados anteriormente, *Zadeh* propôs o operador de implicação. Este novo operador *fuzzy* deve, para as mesmas entradas, fornecer as mesmas saídas dos operadores da lógica bivalente tradicional. O operador *fuzzy* de implicação proposto por *Zadeh* é: *Mín*[1, 1 - μ_A(x) + μ_B(y)].

[19] ***Crisp***- um conjunto é chamado *crisp*, quando todos elementos são classificados como pertencentes ou não ao conjunto. Não existe pertinência parcial em conjuntos *crisp*.

Tabela 2.6: Principais operadores *fuzzy* **E** (interseção)

Operador	**E** (interseção)
Zadeh	$Mín[\mu_A(x), \mu_B(y)]$
Média	$[\mu_A(x) + \mu_B(y)] / 2$
Produto	$\mu_A(x) * \mu_B(y)$
Diferença Limitada	$Máx[0, \mu_A(x) + \mu_B(y) - 1]$

Tabela 2.7: Principais operadores *fuzzy* **OU** (união)

Operador	**OU** (união)
Zadeh	$Máx[\mu_A(x), \mu_B(y)]$
Média	$\{2*Mín[\mu_A(x), \mu_B(y)] + 4*Máx[\mu_A(x), \mu_B(y)]\} / 6$
Soma Probabilística	$[\mu_A(x) + \mu_B(y)] - [\mu_A(x) * \mu_B(y)]$
Soma Limitada	$Mín[1, \mu_A(x) + \mu_B(y)]$

Sistema de Inferência *Mamdani*

Mamdani (1974) propôs um sistema de inferência *fuzzy* para a utilização dos conceitos da lógica *fuzzy* que se tornou bastante popular, principalmente em aplicações de controle. Este sistema de inferência é o adotado na solução proposta deste trabalho.

Um sistema de inferência *fuzzy* do tipo *Mamdani* é composto de um conjunto de regras de produção do tipo **Se <*premissa*> Então <*conclusão*>**, que definem ações de controle em função das diversas faixas de valores que as variáveis de estado do problema podem assumir (SANDRI; CÔRREA, 1999). O termo antecedente é composto por um conjunto de condições que, quando satisfeitas (mesmo que parcialmente), determinam o processamento do consequente da regra por um mecanismo de inferência *fuzzy*. Por sua vez, o consequente é composto de um conjunto de ações ou diagnósticos que são gerados com o disparo da regra. Os consequentes são processados em conjunto até que se chegue a uma resposta de saída.

Na figura 2.14 é mostrada a representação do fluxo de dados do sistema de inferência *fuzzy* proposto por *Mamdani*.

Resumidamente, a fuzzificação consiste em mapear cada entrada *crisp*[20] do sistema em um conjunto *fuzzy* de entrada. O processo de inferência transforma um conjunto *fuzzy* de entrada ativado em um conjunto *fuzzy* de saída, a partir da base de regras

[20] é um valor puro ou preciso, ou seja, pertencente aos conjuntos ou sistemas *booleanos*.

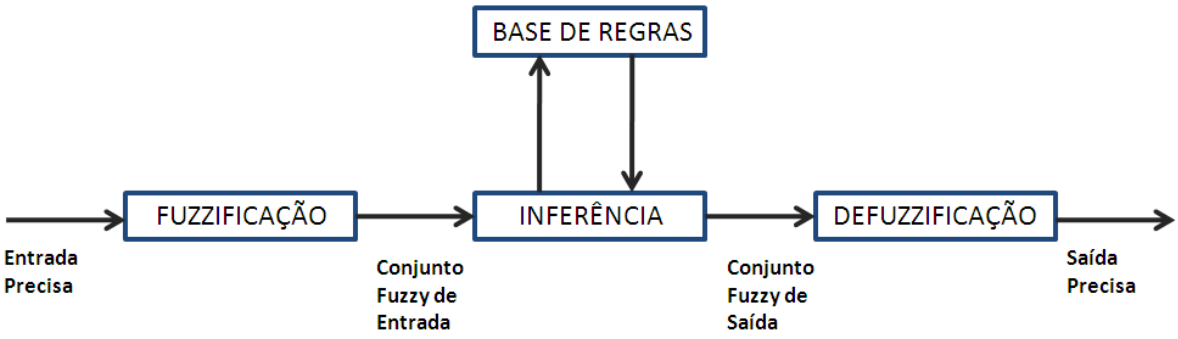


Figura 2.14: Sistema de Inferência *Fuzzy* proposto por *Mamdani*

fornecida por especialistas ou extraídas de dados numéricos. Por fim, o processo de defuzzificação converte o conjunto *fuzzy* de saída em um valor *crisp*. Para o seu sistema de inferência, Mamdani ainda propôs dois novos operadores de implicação. Implicação mínimo: *Mín*[μ_A(x), μ_B(y)]; e Implicação produto: μ_A(x)*μ_B(x).

Para entender melhor o funcionamento do sistema de inferência proposto por *Mamdani*, observe o exemplo de um controlador de tráfego *fuzzy* implantado em um túnel, retirado de Aguiar e Júnior (1999) . Esse controlador deve informar as condições do trânsito em pontos estratégicos de saída do túnel, veja como podem ser modelados esses conjuntos na figura 2.15 da página 37. No exemplo existem seis conjuntos *fuzzy*, que são valores linguísticos sobre a variável linguística[21] velocidade: *estagnada, muito lenta, lenta, congestionada, normal* e *rapida*. Considere que esses valores sejam também usados para mostrar em um painel eletrônico as condições do tráfego nesse túnel, no seguinte formato: *"A situação do tráfego no túnel* ***<nome do túnel>*** *está* ***<valor linguístico>****"*.

Considere que a entrada para o sistema de inferência seja tripla (e simbolizada por *fluxo_V1, fluxo_V2* e *fluxo_V3*) e indica a velocidade média dos carros em alguns pontos do túnel. A saída da inferência, que é fornecida como entrada ao processo de defuzzificação, também é um valor linguístico da variável velocidade apresentada anteriormente (*estagnada, muito lenta, lenta, congestionada, normal* e *rápida*).

Os valores '*fluxo_V1*', '*fluxo_V2*' e '*fluxo_V3*' ao entrarem no sistema são fuzzificados,

[21] Variável linguística - variável cujos valores são nomes de conjuntos *fuzzy*. Formalmente, caracretiza-se por uma quíntupla (**N, T(N), X, G, M**), onde: **N** é o nome da variável (ex.: velocidade); **T(N)** é o conjunto finito de termos **N**, ou seja, o conjunto de valores linguísticos de **N** (ex.: estagnado, muito lento, lento, congestionado, normal e rápido); **X** é o universo de discurso, ou seja, espaço *fuzzy* completo de variação de uma variável do modelo (ex.: 0 a 100km/h); **G** é a regra sintática para gerar os valores de **N** como uma composição de termos de **T(N)**, conectivos lógicos, modificadores e delimitadores (ex.: velocidade lenta, velocidade não muito lenta); **M** é a regra semântica, para associar a cada valor gerado por **G** um conjunto *fuzzy* em **X** (ex.: associa os valores acima a conjuntos fuzzy cujas funções de pertinência exprimem seus significados). (TANSCHEIT, s/d)

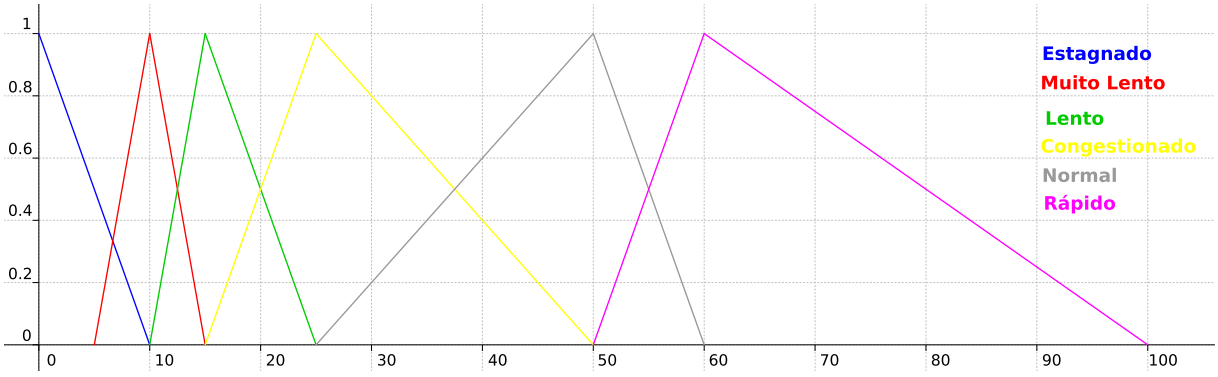


Figura 2.15: Conjuntos Fuzzy da variável *velocidade*

ou seja, os valores de entrada são mapeados em valores linguísticos.

Para ilustrar o procedimento descrito, suponha que em um dado instante tenha-se *fluxo_V1 = 15km/h, fluxo_V2 = 20km/h e fluxo_V3 = 25km/h*. Na fuzzificação, o valor do entrada 'fluxo_V1' é mapeada como *lenta*, com grau de pertinência 1; '*fluxo_V2*' como *lenta*, com grau de pertinência 0.5, e *congestionada*, com grau de pertinência 0.5; e '*fluxo_V3*' como *congestionada*, com grau de pertinência 1. Ver figura 2.16 da página 37.

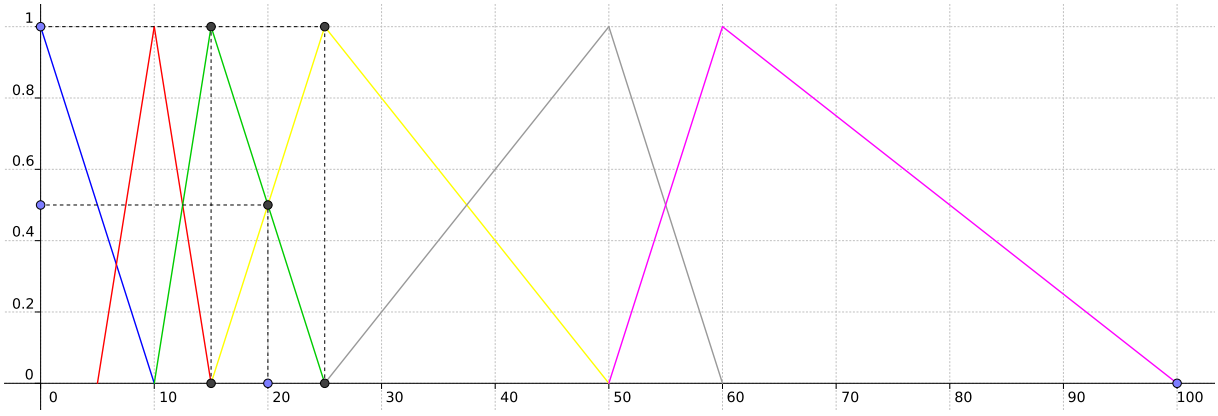


Figura 2.16: Fuzzificação das entradas 'fluxo_V1=15km/h, 'fluxo_V2'=20km/h e 'fluxo_V3'=25km/h

Observe que a entrada '*fluxo_V1*' é *lenta*, enquanto '*fluxo_V2*' é ao mesmo tempo parcialmente *lenta* e parcialmente *congestionada* e '*fluxo_V3*' está *congestionada*. Os valores linguísticos e seus graus de pertinência compreendem a saída do processo de fuzzificação e representam a entrada da inferência, que é a etapa seguinte. O processo de inferência consulta a base de regras da figura 2.17 a fim de, a partir dos operadores *fuzzy*, determinar o conjunto de saída, que servirá como entrada no processo de defuzzificação. Neste exemplo está sendo usado os operados **E** e **OU**, propostos por *Zadeh*, e implicação mínimo, proposto por *Mamdani*.

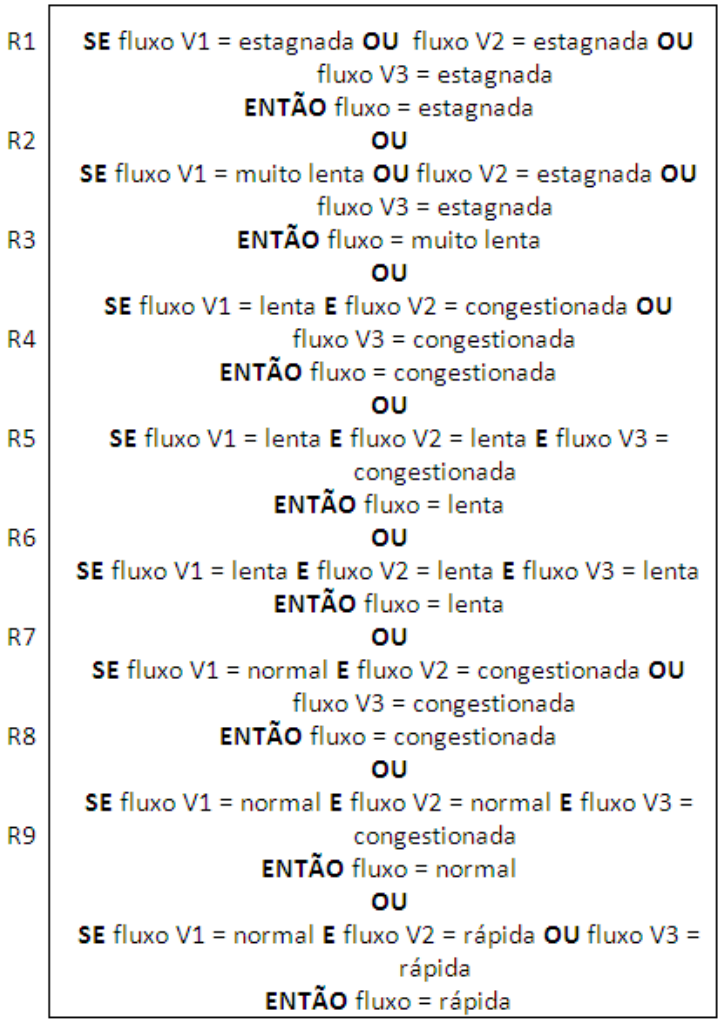


Figura 2.17: Base de Regras do Controlador de Tráfego

Note que somente as regras R3, R4 e R6 foram ativadas, ou seja, têm consequentes maiores do que zero. Assim, obtém-se na regra R3:

SE *fluxo_V1 = lenta* **E** *fluxa_V2 = congestionada* **OU** *fluxo_V3 = congestionada*
ENTÃO *fluxo = congestionada*

SE 1.0 **E** 0.5 **OU** 1.0 **ENTÃO** 1.0

deste modo, o valor linguístico do consequente da regra R3 é congestionada com 1,0 de grau de pertinência. Veja o resultado na figura 2.18.

Para a regra R4:

SE *fluxo_V1 = lenta* **E** *fluxo_V2 = lenta* **E** *fluxo_V3 = congestionada* **ENTÃO** *fluxo = lento*

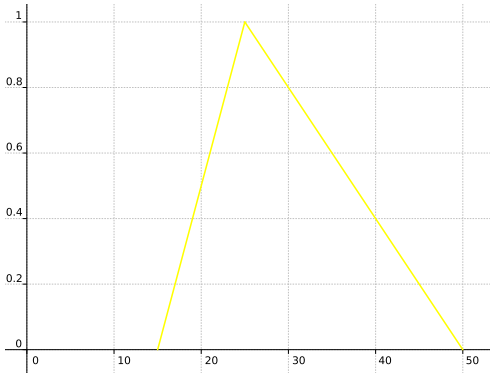


Figura 2.18: Consequente da regra R3 ativado como congestionada com grau de pertinência 1,0. Adaptado de Aguiar e Júnior (1999)

SE 1.0 **E** 0.5 **E** 1.0 **ENTÃO** 0.5

deste modo, o valor linguístico do consequente da regra R4 é lenta com 0.5 de grau de pertinência. Veja na o resultado na figura 2.19.

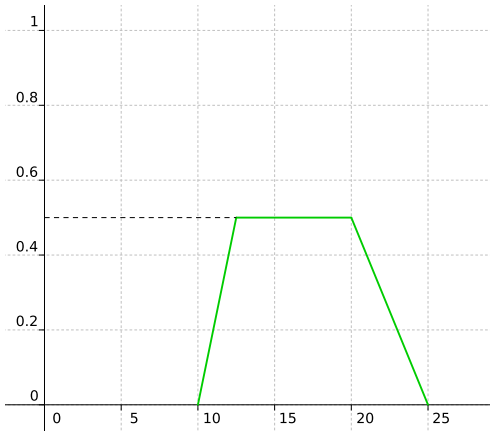


Figura 2.19: Consequente da regra R4 ativado como lenta com grau de pertinência 0,5. Adaptado de Aguiar e Júnior (1999)

Para a regra R6:

SE *fluxo_V1 = normal* **E** *fluxo_V2 = congestionada* **OU** *fluxo_V3 = congestionada*
ENTÃO *fluxo = congestionada*

SE 0.0 **E** 0.5 **OU** 1.0 **ENTÃO** 1.0

deste modo, o valor linguístico do consequente da regra R6 é congestionada com 1.0 de grau de pertinência, assim como na regra R3.

Por fim, no processo de inferência, falta a aplicação do operador **OU** sobre os consequentes ativados das regras R3, R4 e R6. Como resultado obtém-se o conjunto, expresso na figura 2.20, que é a entrada fornecida para o processo de defuzzificação.

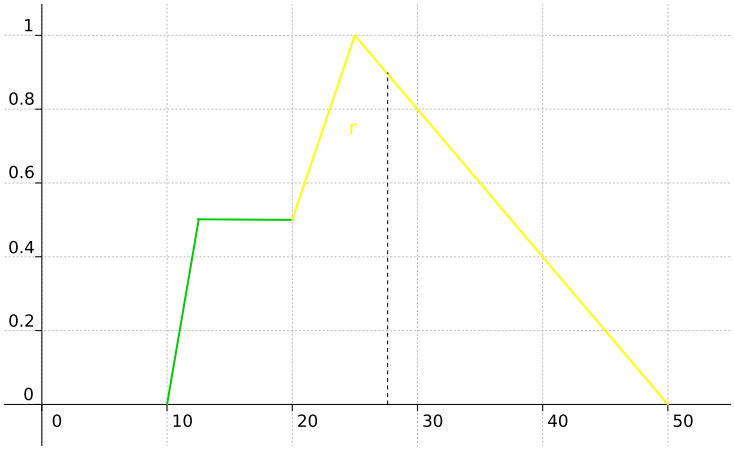


Figura 2.20: União dos consequentes ativados das regras R3, R4 e R6.

Existem diversos métodos para defuzificar um conjunto *fuzzy*, dentre eles: Centro de Gravidade (COG - *Center Of Gravity*), Máximo, Média dos Máximos, Altura, Altura Modificada (REZENDE, 2005). Neste exemplo é usado como mecanismo de deffuzificação o *"Método do Centróide"*(COG), ou centróide, que encontra o centro geométrico do conjunto de saída (figura 2.21), o que corresponde à abcissa do baricentro (KOHAGURA, 2007). O centróide é calculado como mostram as equações 2.1 e 2.2.

- Caso Discreto:

$$VC_{COG} = \frac{\sum_{i=1}^{n} x(i)A(i)}{\sum_{i=1}^{n} A(i)} \tag{2.1}$$

- Caso Contínuo:

$$VC_{COG} = \frac{\int_{i=1}^{n} xA(x)dx}{\int_{i=1}^{n} A(x)} \tag{2.2}$$

Para o exemplo, o valor defuzzificado é *27.615*, aproximadamente. Isso significa que os carros no túnel estão em uma velocidade que os deixa entre os conjuntos *congestionada* e *lenta*, porém o valor *27.615* tem maior grau de pertinência no conjunto *congestionada* do que no conjunto *lenta*. Logo a saída sintática desse processamento seria: *A situação do tráfego no túnel* ***<nome do túnel>*** *está* ***Congestionada***.

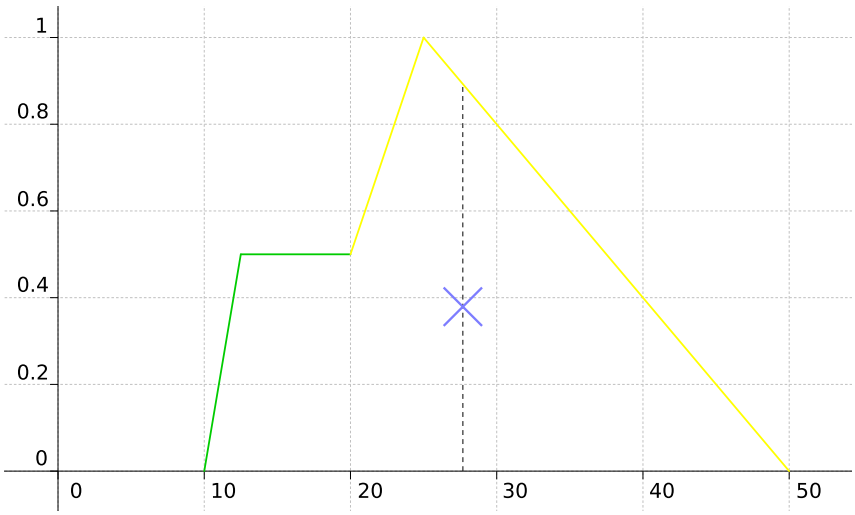


Figura 2.21: Centróide do conjunto de saída do exemplo

3 Metodologia

Este capítulo trata da metodologia de desenvolvimento que foi utilizada no trabalho. São mostradas inicialmente algumas soluções adotadas por outras equipes para problemas semelhantes. Em seguida é descrita a implementação da solução proposta. São mostrados exemplos com a ferramenta utilizada na descrição e correção do sistema e realizados breves comentários sobre os testes e resultados parciais efetuados durante a fase de modelagem.

3.1 Soluções Adotadas por Algumas Equipes

Soluções *fuzzy* são largamente aplicadas em ambientes multi-agente dinâmicos em tempo real, no caso do futebol de robôs simulados com necessidade de rápido processamento, ou seja, que consiga processar dentro de um ciclo do simulador. Pode-se notar bons resultados relativos ao uso de tais métodos no universo do futebol robótico em simulação 2D, como mostra Mattoso (2010) ao desenvolver um sistema de posicionamento de jogadores por meio de algoritmo *fuzzy*. O autor mostra a utilização de conjuntos *fuzzy* para controlar o posicionamento do goleiro em relação à bola e aos atacantes do time adversário. Como resultado de seu trabalho, o algoritmo foi capaz de corrigir erros do goleiro, principalmente em relação à posição no eixo *"x"* e para jogadas vindas pela lateral do gol. Mattoso (2010) comenta ainda que: o motor de inferência do goleiro é ativado com base na distância que os atacantes do time adversário e a bola estão do gol de sua equipe; para auxiliar no controle do posicionamento dos jogadores, fazem uso da aplicação de uma variação do algoritmo *GostrategicPosition*, desenvolvido pela equipe *UvaTrilearn*. Essa variação foi aplicada em união com o algoritmo *fuzzy*, descrito para posicionamento do goleiro, gerando um híbrido que segundo o autor mostrou resultados positivos. A defesa se posicionou, após a aplicação da solução, de forma mais alinhada, ocasionando ocorrência de mais impedimentos no time adversário e impedindo que acontecessem lançamentos próximos à área de defesa. Segundo os autores, ainda foi notado que o time como um todo, assumiu seu posicionamento esperado mais rapidamente, como consequência obteve

uma defesa mais forte e um contra-ataque mais perigoso.

Resultados promissores são apresentados por Fraccaroli (2008) em seu time ao aplicar técnicas *fuzzy* para realizar a escolha do posicionamento tático de sua equipe. A seleção da formação é adotada dentro de um conjunto com três formações pré-definidas, de acordo com a posição da bola e dos jogadores adversários. O autor ainda classifica essas formações como defensiva, defensiva aberta e de ataque. É citado ainda que a ferramenta utilizada para modelagem dos conjuntos foi a *toolbox de fuzzy* do *Matlab*. As entradas do sistema são a posição da bola e a posição do adversário, ambas tomadas em relação ao eixo das abcissas (*"x"*). A partir dessas duas variáveis linguísticas define-se, pelo motor de inferência, qual esquema tático será adotado.

Diversos times de futebol robótico por todo país tem adotado soluções *fuzzy* para melhorar a estratégia de suas equipes. O trabalho de Silva et al. (2007) mostra o desenvolvimento de diversos controladores *fuzzy* para as mais variadas habilidades de cada agente. Sua solução foi proposta de acordo com as áreas de atuação de cada agente, para os atacantes, o chute e o posicionamento foram aperfeiçoados, os meiocampistas tiveram seu posicinamento e tomada de decisões abordados, goleiro e defensores tiveram o posicionamento tratado. A ferramenta adotada pelo autor para modelagem dos conjuntos foi a *XFuzzy 3.0*. Os controladores *fuzzy* desenvolvidos pelo autor no seu trabalho foram:

- **posição de chute** - tem como objetivo encontrar o ponto do gol adversário, onde a possibilidade de marcar seja a maior possível, sendo a bola chutada da posição atual do agente. A saída desse conjunto é o valor correspondente ao canto do gol adversário onde a bola será chutada. Essa saída é influenciada pelas saídas de dois outros conjuntos que são os da posição global do goleiro em campo e posição do jogador atacante com relação ao goleiro;

- **posicionamento sem a bola** - tenta colocar os atacantes em boa posição para receber a bola sem entrarem em linha de impedimento. Para isso a equipe utilizou dois controladores com um total de cinco entradas e duas saídas, onde a posição do jogador e da bola nos eixos *"x"* e *"y"* e a posição de impedimento no eixo *"x"* são os variáveis de entrada e a saída é o par ordenado que designa a posição que o agente deve adotar;

- **tomada de decisão dos meias ofensivos** - guia a decisão dos meias ofensivos quando o jogador está com a posse de bola. Para isso, leva em consideração a posição global dos agentes, a posição do adversário mais próximo, sua posição em relação

ao gol, a posição do agente aliado mais próximo com relação ao gol, a quantidade de adversários próximos ao agente companheiro e a distância até o aliado;

- **posicionamento do meia defensivo** - para os meias defensivos foi modelado um controlador que mantivesse o agente dentro de sua área de atuação levando em consideração a posição da bola e sua posição atual;

- **posicionamento dos laterais** - o objetivo deste controlador é mover um jogador, quando ele estiver sem a posse de bola, ao longo de toda a lateral do campo, tendo como ponto de atração a bola. Esse comportamento é aplicado a dois jogadores e faz com que elem atuem pelo lado direito e esquerdo do campo como "alas";

- **posicionamento dos zagueiros** - este controlador tenta manter os zagueiros aproximadamente no centro da linha de defesa, com possibilidades de visualizar melhor a bola. Já que os alas encarregam-se das laterais, então os zagueiros podem se preocupar em proteger melhor o centro da defesa;

- **posicionamento do goleiro** - as variáveis de entrada para este controlador são as posições do jogador que está de posse da bola e a posição do goleiro. Dadas estas variáveis, o valor resultante da deffuzificação é a posição onde o goleiro deve estar para ter maior possibilidades de defesa. Este controlador foi o julgado mais ineficiente pelo autor, pois após a aplicação da solução o goleiro ainda se mostrava adiantado em algumas jogadas.

Existem ainda autores que adotaram outras alternativas para melhorar o desempenho de suas equipes, é o caso de Silva (2010) que aplicou um algoritmo de aprendizado por reforço muito famoso, o *Sarsa*. A aplicação deste algoritmo no trabalho do autor mostrou uma redução no número de gols sofridos pela sua equipe. O autor ainda comenta que do universo de 300 partidas disputadas entre diferentes equipes para testes houve alguns momentos em que sua equipe chegou a uma vantagem sofrendo cerca de menos de 25% dos gols sofridos inicialmente, o melhor caso diante dos resultados apresentados.

Existem trabalhos que exploram o uso de redes neurais na liga de simulação, é o que sugere Ferreira (2010). O foco desse autor foi a implementação e uso de esquemas táticos dinâmicos que pudessem ser trocados pelo técnico da equipe durante o jogo. A ideia foi utilizar uma rede neural do tipo *perceptron* de múltiplas camadas como mecanismo de tomada de decisão acerca de qual formação tática a equipe deve assumir de acordo com o desenvolvimento da partida. Os resultados observados pelo autor possuem alto grau de relevância, pois notou-se melhora no time através da aplicação de uma mudança tênue.

3.2 Implementação

Uma análise *Top-Down* do algoritmo da *Agent2D* pode mostrar alguns dos possíveis caminhos que cada agente pode traçar em sua área de atuação. O comportamento desses agentes é determinado por um conjunto de algoritmos, arquivos de cabeçalho, *scripts* e aquivos binários (dispersos em aproximadamente 1000 itens, incluindo arquivos e diretórios, sendo que 327 desses itens fazem parte da ferramenta e 655 da biblioteca *librcsc*).

Antes de iniciar uma fase de implementação, é interessante que se tenha bem definido um projeto-base a ser seguido. A primeira pergunta que deve ser respondida em relação ao desenvolvimento de jogadas coletivas pode ser generalizada como uma ideia de colaboração multi-agente. Como cita Reis (2003), existem quatro questionamentos importantes que devem ser respondidos no ato da comunicação. Esses pontos têm aplicação para ambientes multi-agentes em geral e, nesse caso específico, foco no futebol robótico. São eles: o que comunicar? quando comunicar? a quem comunicar? como comunicar?

3.2.1 O Que Comunicar?

Independentemente do fim a que se destina, as mensagens enviadas não podem (e para o caso do futebol de robôs, nem suportam) conter toda a informação que o agente emissor possui. Como o canal e o tamanho das mensagem são limitados, é interessante que sejam passados aos agentes aliados somente informações relevantes e precisas. Para isso, de início, é necessário definir as jogadas coletivas que serão comunicadas.

As jogadas coletivas foram desenvolvidas para dar à equipe um comportamento diferenciado em modos de jogo (ver subseção 3.2.2) que não possuíam tratamento específico, como caso de faltas, tiros livres e impedimentos. Essas jogadas foram inseridas como extensão às atuais habilidades mantidas hoje pela equipe *iBots*. O código que trata a postura que o agente tem que assumir quando for detectada uma situação de invocação de jogadas coletivas, encontra-se anexa ao arquivo *bhv_set_play_free_kick.cpp*, é o qual contém a classe ***Bhv_SetPlayFreeKick***. Ao responder a pergunta "o que comunicar?", por meio da execução de uma jogada coletiva, recomenda-se ao agente o que fazer no momento de cada invocação. As jogadas desenvolvidas também tem que regular comportamentos atípicos (em determinado momento a jogada se torna inviável, um dos jogadores não consegue alcançar o destino, entre outros), para os casos em que, por exemplo, algum ruído atrapalhe o andamento da jogada.

O primeiro momento de decisão, antes da chamada de uma jogada, é o de análise

do mundo ao redor do jogador, ver imagem 3.1 da página 45. Essa análise da última atualização do modelo de mundo visa coletar dados que serão passados como argumento para o sistema de apoio a decisão *fuzzy*. A coleta de informação visa identificar a posição em campo onde a bola está parada, a distância da bola ao gol adversário e o ângulo onde o probabilidade de acerto de um lance seja maior. Após o envio dessas informações ao sistema de suporte a decisão, este sistema retorna um valor correspondente à jogada que deve ser executada.

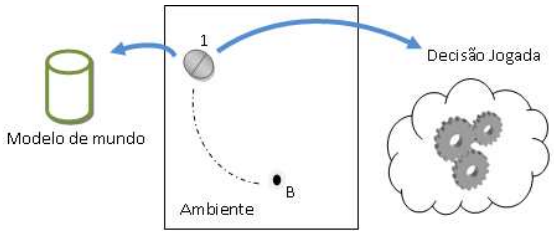


Figura 3.1: Jogador Percebe Jogada, Busca Dados e Toma Decisão

Quando cada um dos agentes já está posicionado e sabe o que deve fazer, o jogador com a posse da bola dá início à jogada coletiva. A seguir é descrita a informação que será comunicada, ou seja, as jogadas coletivas desenvolvidas neste trabalho e como cada uma delas trabalha.

- **Tabelas pela Direita e pela Esquerda** - é interessante que esta jogada seja invocada quando o jogador, ao cobrar uma falta, identifica o aliado que está em melhor posição para uma jogada mais ofensiva. O comportamento geral dos jogadores para essa jogada é o seguinte: se o jogador percebe que é o mais próximo da bola, ele se desloca para a posição onde a bola está parada; durante o deslocamento, o jogador verifica os aliados que estão mais próximos e que podem colaborar com a jogada; quando os aliados que participarão da jogada são definidos, o jogador que vai cobrar a falta ou o tiro livre envia mensagens para os destinatários que irão participar da jogada para que estes destinatários se posicionem de acordo com suas instruções; se o jogador é um agente que não irá cobrar a falta, então ele invoca uma chamada de posicionamento tático padrão, segundo o comportamento normal, se em sua memória de áudio sobre as jogadas não houver nenhuma mensagem em que nos últimos dois ciclos o jogador que armazenou seja um dos destinatários; se ocorrer que o jogador seja o primeiro ou o segundo destinatário da mensagem, ele se posiciona segundo a instrução contida na mensagem, deixando o comportamento padrão de lado e aguardando a cobrança da penalidade. Um modelo gráfico da intenção da jogada coletiva pode ser visto na figura 3.2 da página 46; após a cobrança da falta

há uma mudança no modo de jogo, o comportamento que foi direcionado para o agente deve ser preparado para que, nessa situação, as ações do agente continuem a colaborar para o sucesso da jogada. Quando o jogador que estava mais próximo da bola a passa para o aliado, ele se desloca à frente para receber a bola que será devolvida por um companheiro de equipe.

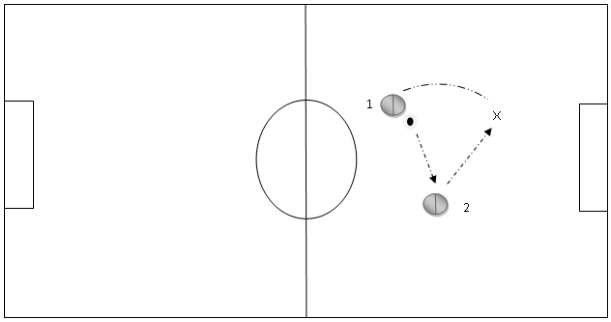


Figura 3.2: Exemplo da Jogada *Tabela*

- **Lançamento com Abertura pelas Laterais** - este tipo de jogada consiste em posicionar um jogador um pouco à frente do agente que irá cobrar a penalidade e outro jogador na lateral do campo. Esta jogada possui um lançamento mais longo que a tabela, por isso é necessário que o agente esteja bem posicionado para receber a bola sem que haja interceptação adversária. O lançamento com abertura pelas laterais é executado em situações em que a bola esteja mais longe do gol oponente. A jogada coletiva é visualizada graficamente na imagem 3.3 da página 46.

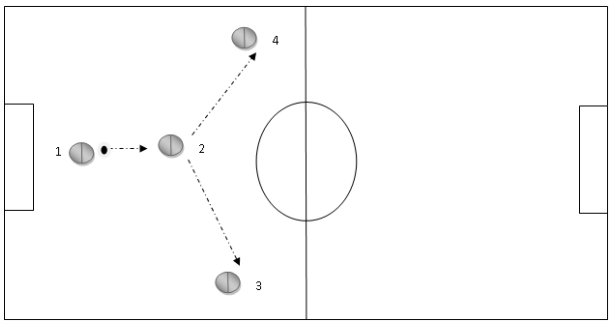


Figura 3.3: Exemplo da Jogada *Meio Abertura Laterais*

- **Laterais com Lançamento para o meio** - faz-se uso desta jogada esperando bons resultados tanto de longe quanto perto do gol adversário, pois ela tenta realizar um lançamento de bola com abertura pela lateral e posteriormente um lançamento para o meio. Este tipo de jogada pode culminar em boas oportunidades de gol e também "abrir" o jogo em momentos de *play_on* em que há muitos jogadores aglomerados no

meio-campo. Veja a representação gráfica da intenção dessa jogada na imagem 3.4 da página 47.

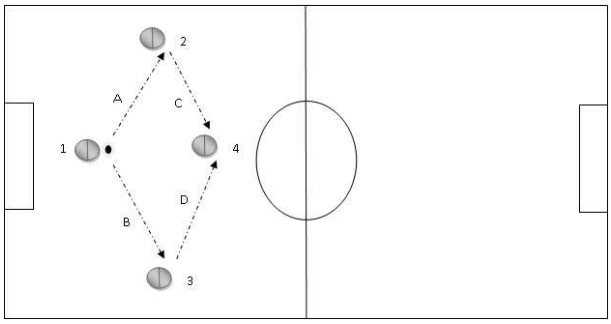


Figura 3.4: Exemplo da Jogada *Lateral Meio*

- **Assistência para Chute** - das jogadas coletivas desenvolvidas, esta é a mais ofensiva, rápida e direta. Consiste em o jogador com a posse de bola, visualizando que há possibilidades de um aliado próximo ao gol adversário chutar a gol, enviar uma mensagem solicitando o ponto em que o aliado deve se posicionar para receber a bola. Quando o aliado estiver próximo ao ponto o lançamento é efetuado. Assim que a bola chega, o aliado chuta a gol se possível. O comportamento esperado da jogada pode ser visto na figura 3.5 da página 47.

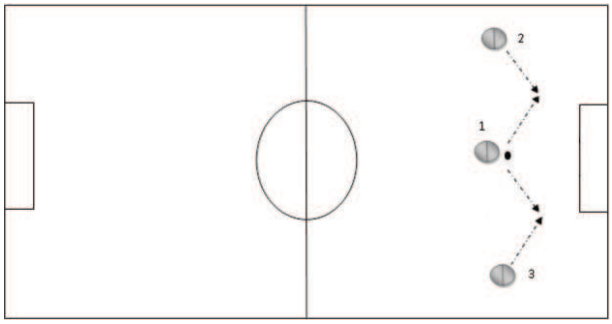


Figura 3.5: Exemplo da Jogada *Assistência*

- **Inversões** - a inversão é uma jogada que consiste em lançar a bola de um lado a outro do campo. Essa jogada deve ser chamada quando percebe-se uma aglomeração adversária na região do campo onde a bola se encontra, então, é entendido que pode ser melhor tentar uma jogada pelo lado oposto. Veja exemplo na imagem 3.6, na página 48.

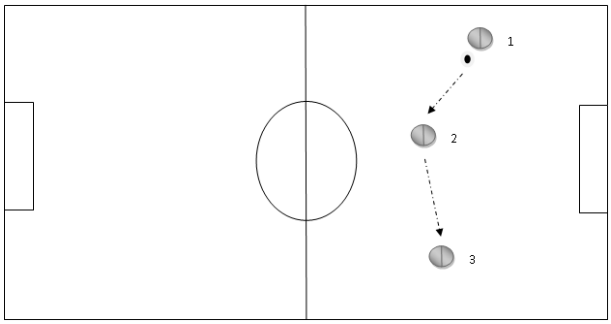


Figura 3.6: Exemplo da Jogada *Inversão*

3.2.2 Quando Comunicar?

Em domínios com reduzida largura de banda para operações de comunicação, é muito importante decidir corretamente o momento temporal em que é efetuada a comunicação. Para as situações estudadas neste trabalho, é necessário considerar os modos de jogo definidos pelo árbitro. Por vezes, se a informação a comunicar não for muito importante, é preferível não realizar qualquer comunicação, deixando, desta forma, a largura de banda do canal de comunicação disponível para os outros agentes presentes no sistema.

Neste trabalho os momentos definidos para a utilização das jogadas coletivas foram as jogadas de bola parada, identificadas pelas *flags* de modo de jogo *GameMode::FreeKickFault_*, *GameMode::FreeKick_* e *GameMode::OffSide_*, em que as duas primeiras representam modos de cobrança de falta e a última impedimento. Os modo de jogo estão definidos em um arquivo da biblioteca *librcsc-4.0.0*, na seguinte localização: /usr/local/include/rcsc/-game_mode.h A classe recebe o mesmo nome do arquivo (*Class GameMode*{}) e os tipos de jogo são definidos como enumerações, como pode ser visto na tabela 3.1 da página 49.

A mudança de cada modo de jogo é regulada pelo árbitro *online*, que sinaliza para os agentes o que está acontecendo através da mudança dos modos de jogo e através de mensagens fonéticas. O estudo desses modos de jogo foram importantes, pois foi baseado neles que o desvio condicional para chamada da jogada coletiva, realizado neste trabalho, é ativado. Sempre que uma jogada de impedimento, tiro livre ou falta são detectados, o algoritmo que avalia qual a melhor jogada para a situação analisada é invocado.

Os modos de jogo, descritos na tabela 3.1, servem como sinalizadores para marcar um determinado acontecimento. *BeforeKickOff* sinaliza que o jogo ainda não iniciou (ou reiniciou); *TimeOver* indica fim de jogo; *KickOff_* marca pontapé inicial (início e reinício de jogo); *KickIn_* e *CornerKick_* representam cobrança de lateral e escanteio, respectivamente; *FreeKick_*, *FreeKickFault_* e *IndFreeKick_* sinalizam modos de cobrança de falta;

Tabela 3.1: Modos de Jogo

BeforeKickOff	TimeOver	PlayOn	KickOff_
KickIn_	FreeKick_	CornerKick_	GoalKick_
AfterGoal_	OffSide_	PenaltyKick_	FirstHalfOver
Pause	Human	FoulCharge_	FoulPush_
FoulMultipleAttacker_	FoulBallOut_	BackPass_	FreeKickFault_
CatchFault_	IndFreeKick_	PenaltySetup_	PenaltyReady_
PenaltyTaken_	PenaltyMiss_	PenaltyScore_	PenaltyOnfield_
PenaltyFoul_	GoalieCatch_	ExtendHalf	MODE_MAX

*GoalKick_* significa que o goleiro capturou a bola; *OffSide_* impedimento; *PenaltySetup_* indica que a partida irá para pênaltis; *PenaltyReady_* indica que o campo está pronto para a cobrança do pênalti; *PenaltyKick_* sinaliza cobrança de pênaltis; *PenaltySetup_* o goleiro se posiciona para a cobrança do pênalti; *PenaltyTaken_* o pênalti está sendo cobrado; *PenaltyMiss_* o cobrador errou na cobrança do pênalti; *PenaltyScore_* o jogador marcou o gol na cobrança do pênalti; *PenaltyOnfield_* indica que o próximo modo de jogo é *PenaltySetup_*; *PenaltyFoul_* indica que o próximo modo de jogo é *PenaltyMiss_*; *FirstHalfOver* indica que se passou a primeira metade de jogo; *Pause* sinaliza pausa no jogo; *Human* sinaliza interferência humana; *FoulCharge_, FoulPush_, FoulMultipleAttacker_, FoulBallOut_* simbolizam diferentes tipos de falta cometidas; *BackPass_* significa que a bola deve ser recuada para o goleiro; *ExtendHalf* indica tempo adicional (acréscimo).

Apesar de as jogadas coletivas adotadas e implementadas neste trabalho serem realizadas somente para as jogadas de bola parada, as jogadas coletivas podem ser facilmente estendidas para jogadas coletivas de bola "em movimento", *play_on*. Isso porque a estrutura definida das mensagens (ver subseção 3.2.3) não se limita às jogadas de bola parada, ou seja, já compreende outras situações a serem definidas em trabalhos futuros.

3.2.3 A Quem Comunicar?

No futebol de robôs todas as mensagens, trocadas entre os jogadores, assumem a forma *broadcast*, ou seja, um agente envia uma mensagem para todos dentro de uma pré-determinada distância de alcance de áudio. Porém, em outras aplicações multi-agente as mensagens podem ser enviadas para um, vários ou todos os agentes. A decisão do receptor da mensagem assume particular importância nos casos de comunicação direta entre dois agentes. O agente receptor deve estar preparado para tomar uma decisão assim que recebe uma mensagem do emissor destinada a ele e descartar as demais.

Na liga de simulação 2D a comunicação pode ocorrer de diferentes maneiras, sendo a primeira unidirecional, caso comunicação realizada entre técnico e algum jogador de seu time, ou multidirecional, que é a comunicação realizada entre os jogadores. É dita que a comunicação entre os jogadores em campo é multidirecional poque eles podem trocar mensagens entre si em tempo real[1]. Além de multidirecional, as mensagens dos agentes também são enviadas em *broadcast*[2]. O técnico é um agente privilegiado que possui acesso a todas as informações do jogo sem ruído e pode comunicar-se com seu time, porém essa comunição é limitada a intervalos de tempo discretamente definidos pelo simulador para cada tipo de mensagem.

O processo de seleção que determina quais jogadores serão receptores finais da mensagem é realizado atráves de uma análise da posição da bola em campo. Logo depois que a jogada coletiva é selecionada, o agente que vai cobrar a falta analisa quais os jogadores estão em melhores condições para participar da jogada. Para isso, considera como distância mínima de distância dos adversários para a bola de 9,5m. O processo de análise dos melhores jogadores a participar da jogada é o seguinte: é traçada uma circunferência de raio igual a 9m; são capturados todos adversários com maior potencial de interferência na jogada (que estejam até 15m de distância); entre cada um desses jogadores e a bola é traçado um vetor; para cada vetor é calculado o ângulo entre o vetores mais próximos (em sentido horário e anti-horário, e considerando como extremos os vetores paralelos ao eixo "y" que saem da bola com sentidos diferentes); são selecionados pontos na borda da circunferência que pertencem a cada vetor que sai da bola e tem ângulo igual à metade do ângulo entre cada vetor; dos pontos selecionados na borda da circunferência é verificada a distância do oponente mais próximo; o ponto que tiver a maior distância do oponente mais próximo é considerado o melhor; é selecionado entre os aliados em campo o que esteja mais próximo do ponto definido; o segundo ponto é escolhido levando em consideração a jogada e a posição do primeiro ponto; o segundo jogador que participará da jogada é definido com base na proximidade com o segundo ponto.

Os posicionamentos dos jogadores que participarão da jogada coletiva sofrem ligeiras alterações dependendo de qual jogada é invocada (por exemplo, deslocamento para direita quando for invocada uma jogada pela direita ou um posicionamento um pouco mais

[1] Dize-se que os agentes trocam mensagens em tempo real porque a única demora que existe é o tempo da mensagem chegar ao servidor. Ou seja, as mensagens fonéticas que chegam ao servidor são repassadas no ciclo seguinte.

[2] *Broadcast* - Nesse caso, dizer que as mensagens são repassadas por *broadcast*, significa que não há como especificar qual jogador exatamente irá ouvi-la. A mensagem é repassada a todos os jogadores dentro de uma distância de alcance de áudio (atualmente 50m), com a atribuição de ruído proporcional à distância do ouvinte.

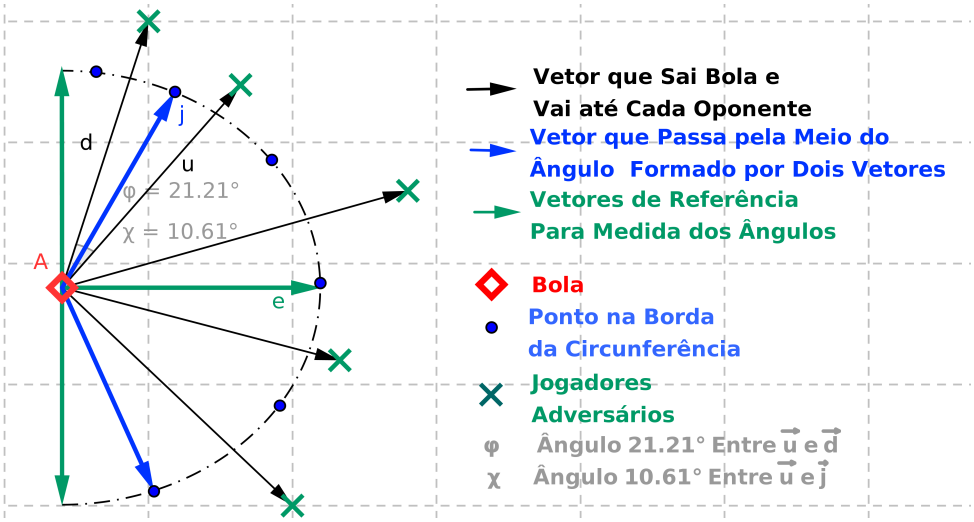


Figura 3.7: Seleção do Ângulo

à frente quando a jogada for uma assistência para chute). Após a seleção da jogada e o cálculo dos pontos para posicionamento dos jogadores, a mensagem é enviada em *broadcast*. Cada jogador deve perceber, ao receber a mensagem, se é um dos destinatários.

3.2.4 Como Comunicar?

No caso de existirem diversos meios de comunicação disponíveis ao agente, é necessário selecionar qual o meio mais adequado para executar a comunicação. A presente abordagem assume a comunicação fonética.

Logo após a decisão ser tomada (logo que chega ao ponto onde deve cobrar o tiro livre - a falta ou o impedimento -, antes de dar início à jogada), o agente-jogador que selecionou a jogada a ser invocada, codifica uma mensagem de áudio. Essa mensagem fornece informações sobre qual jogada será executada, quais agentes participarão, o que cada agente-jogador deve fazer e o *byte* de conferência. Quando cada jogador recebe a mensagem, verifica se é um dos destinatários, se não for descarta ou repassa esta segundo cada comportamento. Esse processo pode ser observado mais claramente na figura 3.8 da página 52.

Considerando que o tamanho da mensagem de áudio que o jogador pode enviar é de 10 *bytes* e que um desses já é dedicado ao cabeçalho que identifica que a mensagem é uma jogada coletiva, então sobram 9 caracteres que devem possuir o mínimo de informação relevante à jogada. Foi adotada a seguinte estrutura para as mensagens: o segundo caractere armazena a letra que representará qual das jogadas coletivas é a selecionada; o terceiro e o quarto caracteres indicam os números dos jogadores aliados que participarão

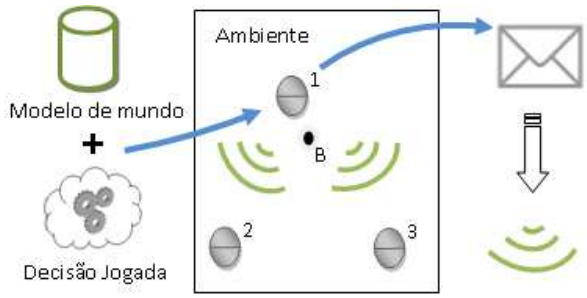


Figura 3.8: Jogador Une informações, Codifica e Envia Mensagem

da jogada; os quatro caracteres posteriores armazenam as coordenadas onde os jogadores devem estar para iniciar a jogada; os dois últimos espaços desse vetor são destinados aos caracteres de conferência. A imagem 3.9 da página 52 mostra mais detalhadamente essa estrutura.

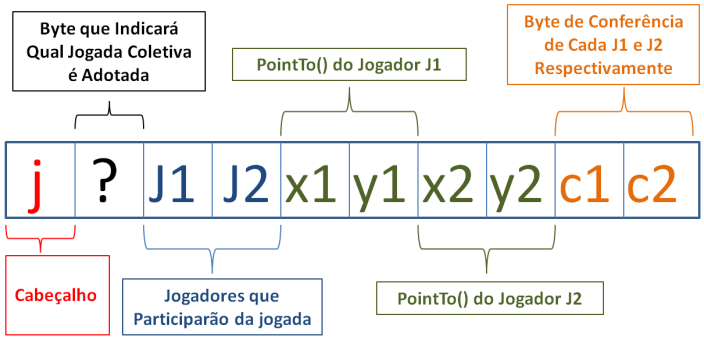


Figura 3.9: Corpo das Mensagens de Jogadas Coletivas

Cabeçalho

O cabeçalho da mensagem a letra que indica o conteúdo de que a mensagem trata. A seguir os cabeçalhos possíveis de serem montados, descritos no arquivo *say_message_parser* são apresentados. Note que todas as mensagens comunicadas de jogada coletivas desenvolvidas neste trabalho têm como cabeçalho o caractere 'j'.

‘ **i** ’ - descreve que a mensagem é de **I**nterceptação de bola;

‘ **b** ’ - a mensagem vai tratar da **B**ola;

‘ **B** ’ - o conteúdo da mensagem será da **B**ola em relação a um jogador;

‘ **G** ’ - está passando uma mensagem para o **G**oleiro sobre a bola e os jogadores;

‘ **g** ’ - está passando uma mensagem sobre o **G**oleiro;

‘ **R** ’ - a mensagem é sobre 3 jogadores e a informação é ***R****aw* (pura);

‘ **r** ’ - indica que o jogador deve se recuperar ***R****ecovery*;

‘ **Q** ’ - a mensagem é sobre 2 jogadores;

‘ **P** ’ - a mensagem é sobre 1 ***P****layer*;

‘ **p** ’ – mensagem que indica que será realizado um ***P****ass*;

‘ **h** ’ – solicita que a bola seja passada para ele (imagine como se o jogador dissesse “**H**ei”);

‘ **o** ’ - formação de linha de ***O****ffside* (linha de impedimento);

‘ **O** ’ - informação sobre os jogadores ***O****ponents*;

‘ **d** ’ - mensagem que indica formação de linha de defesa (***D****efenseLine*);

‘ **D** ’ – indica para o aliado qual lado é melhor para ele efetuar um ***D****ribble*;

‘ **S** ’ - diz respeito a mensagem cujo próprio jogador que enviou é o destinatário ***S****elfMessages*;

‘ **s** ’ - a mensagem é sobre a ***S****tamina* do jogador;

‘ **T** ’ – mensagem sobre os aliados ***T****eamMates*;

‘ **e** ’ – goleiro oponente **E** jogadores próximos;

‘ **w** ’ – mensagem solicitando que jogadore aguarde requisição ***W****ait*;

‘ **j** ’ - cabeçalho significando que a mensagem ouvida trata de uma **J**ogada coletiva.

Byte **da Jogada Coletiva Adotada**

O segundo caractere da mensagem de jogada coletiva corresponde ao identificador que indica qual jogada coletiva é adotada, ver possíveis jogadas no algoritmo 3 da página 58. Este *byte* pode assumir qualquer caracteres com valor inteiro entre 0 e 9 (0 para Tabela pela Direita; 1 para Tabela pela Esquerda; 2 para Lateral com Lançamento pela Esquerda; 3 Assistência para Chute pela Esquerda; 4 para Inversão pela Esquerda; 5 para Lançamento com Abertura pela Lateral Esquerda; 6 para Lançamento com Abertura pela Lateral Direita; 7 para Inversão pela Direita; 8 Assistência para Chute pela Direita; 9 para Lateral com Lançamento para a Direita), ou seja, ASCII entre 48 e 57 em notação decimal.

Jogadores que Participarão da Jogada

Os jogadores indicados como **J1** e **J2** na imagem 3.9 são os aliados que participarão da jogada coletiva. Esses jogadores são denotados pelos terceiro e quarto byte da mensagem. Considerando que existem 11 jogadores em campo e que eles são enumerados com uniformes em um intervalo inteiro que varia de 1 até 11, então esses *bytes* foram preparados para representando os jogadores de números de 1 até 9 com seus respectivos correspondentes *ASCII* (caractere '1' para o jogador número 1, ... , caractere '9' para o jogador número 9). Os jogadores de números 10 e 11 são representados pelos caracteres *ASCII* 'A' e 'B', respectivamente. Esse padrão foi definido pois além de ser uma sequência lógica se for pensar em valores hexadecimais, os valores que começam em 'A' e vão até 'Z' formam a maior sequência contínua (juntamente com seus correspondentes com letras minúsculas) de valores que são aceitos pelo simulador como corpo das mensagens.

A mesma codificação utilizada para montagem deste *byte* foi aproveitada para montar também os *bytes* de *PointTo()*, descritos a seguir.

PointTo do Jogador J1 e do Jogador J2

Os comandos PointTo(), descritos na estrutura da mensagem mostrada na imagem 3.9, correspondem às posições que devem ser ocupadas pelos jogadores J1 e J2 para o início da jogada coletiva. As posições a serem ocupada são representadas por x1,y1 (para J1) e x2,y2 (para J2).

É importante acrescentar que a dimensão do campo é de 105x68, compreendendo o intervalo [-52.5,52.5] no eixo das abcissas e [-34.0,43.0] no eixo das ordenadas. Dessa forma, cada *byte* do comando *PointTo()* receberá valores no intervalo discreto de inteiros entre 0 e 53, já que o maior valor absoluto em um dos eixos é 52.5. É importante acrescentar que o sinal de cada componente da coordenada é inserido em conjunto com o *byte* de conferência. Veja a representação desses valores na tabela 3.2.

Byte de Conferência

Os *bytes* de conferência C1 e C2 são gerados utilizando a *prova dos nove*[3], calculada com o número do jogador e os valores das coordenadas inseridas no comando *Pointto()*. Cada um dos *byte* de conferência pode assumir um valor máximo igual a 8, visto que na prova dos nove os dígitos de conferência gerados pertencem ao conjunto (0,1,2,3,4,5,6,7,8).

[3] ***Prova dos Nove*** - é uma forma segura de verificar se uma operação aritmética está correta.

Tabela 3.2: Correspondência entre valor representado x caractere utilizado x ASCII em Decimal

Valor Representado	Caractere Utilizado	ASCII em Decimal
0	0	48
...	...	...
9	9	57
10	A	65
...	...	...
35	Z	90
36	a	97
...	...	...
53	r	114

Sendo assim, o *byte* de conferência precisa ser capaz de representar 9 valores, isso é feito utilizando no mínimo 4 *bits* ($2^4 = 16$ valores), já que com 3 *bits* somente 8 valores poderiam ser representados. Dese modo, ainda sobram 4 *bits* por *byte* de conferência que não são utilizados. Porém, ainda é necessário representar o sinal dos números referentes ao *pointto()*, para isso são necessários 2 *bits* (um para o sinal do eixo das abcissas, outro para o sinal do eixo das ordenadas). Agora sim, a mensagem está formatada e completa.

3.3 A *Framework* e a Comunicação

Após respondidas as perguntas anteriores, o próximo passo é entender como é utilizada a comunicação na *framework* escolhida. A *Agent2D* traz uma inovação que outras ferramentas de times-base não haviam trabalhado ainda que é a existência de uma memória de áudio. Esta memória de áudio é descrita por uma classe que se encontra no arquivo audio_memory.cpp e guarda as últimas mensagens de áudio em uma lista para cada tipo respectivo de mensagem. O tipo da mensagem, ou seja, o assunto do qual ela trata é definido pelo seu cabeçalho. As mensagens incluídas na lista de mensagens de áudio são separadas de acordo com seus respectivos tipos que estão definidos no mesmo arquivo como *structs*, ou seja, estruturas de dados compostas. Essas *structs* possuem um inicializador semelhante a um construtor de uma classe. Para auxiliar no acesso à memória de áudio para as mensagens ouvidas por cada agente de uma provável jogada, foi necessário inserir nesse arquivo a estrutura jogada, um método *virtual* ***void setJogada()*** e uma lista da estrutura jogada. Os algoritmos 1 e 2 mostram alguns segmentos das estruturas adicionadas ao arquivo *audio_memory.cpp*. Essa etapa de desenvolvimento

trata do armazenamento da mensagem para uma futura utilização. O que interessa nesse ponto é a montagem da mensagem (descritas na subseção 3.2.4) para envio e posterior recebimento por outro agente.

Algoritmo 1 Adicão de struct em Audio_Memory.h

```
349     struct Jogada {
350         int sender_; //!< numero do jogador que enviou a mensagem
351         int qual_jogada_; //!< numero referente a jogada
352         Vector2D jog1_pos_; //!< primeiro jogador que ira participar
353         Vector2D jog2_pos_; //!< segundo jogador que ira participar
354         int j1_; //!< numero do primeiro jogador que ira participar
355         int j2_; //!< numero do segundo jogador que ira participar
356         char conf1_; //!< caracter de conferencia do primeiro jogador
357         char conf2_; //!< caracter de conferencia do segundo jogador
358
359 /*! \brief inicializacao de todos os membros da struct jogada */
360
361         Jogada( const int sender,
362                 const int qual_jogada,
363                 const Vector2D & jog1_pos ,
364                 const Vector2D & jog2_pos,
365                 const int j1, const int j2,
366                 const char conf1, const char conf2
367                 )
368             : sender_( sender )
369             , qual_jogada_( qual_jogada )
370             , jog1_pos_(jog1_pos)
371             , jog2_pos_(jog2_pos)
372             , j1_(j1) , j2_(j2)
373             , conf1_(conf1), conf2_(conf2)
374           { }
375     };
```

Algoritmo 2 Adicão de Vector e GameTime em Audio_Memory.h

```
431     std::vector< Jogada > M_jogada; //!< informacao ouvida sobre a jogada
432     GameTime M_jogada_time; //!< ciclo em que a ultima jogada foi ouvida
```

Os primeiros métodos invocados ao enviar uma mensagem fonética são os métodos *Say<tipo>()*, descritos na classe *sample_communication.h* na *framework Agent2D*. O *token "<tipo>"* trata dos tipos de invocações que podem ser realizadas pelos métodos *Say<tipo>()*. Entre estes métodos podem ser citados, dentro outros, *sayBall(*agent)*, *sayPlayers(*agent)*, *saySelf(*agent)* e *sayJogada(*agent)*, implementado neste trabalho para as jogadas coletivas. Estes métodos recebem como argumento o tipo *Player_agent* e retornam valores binários, que indicam sucesso ou fracasso na montagem da mensagem de áudio. Quando invocados, os métodos *Say<tipo>()* coletam todas as informações que precisam para enviar a mensagem fonética e solicitam uma chamada ao método *agent->addSayMessage()*, que adiciona a mensagem montada em uma lista de mensagens a

serem enviadas para o servidor. O argumento desse método é um objeto cujo tipo corresponde ao "<*tipo*>" do comando *Say<tipo>()* invocado. O construtor desse objeto é invocado tendo como parâmetros os dados coletados antes de sua chamada. Todos os possíveis tipos de objetos a serem invocados como parâmetros de *agent->addSayMessage()* estão descritos no arquivo *say_message_builder.h*.

O *agent->addSayMessage()*, após ter preenchido o objeto de seu argumento, lança uma chamada ao método *M_effector.addSayMessage(message)*, que recebe como argumento o objeto *saymessage* passado como argumento para este método. A sequência dessas chamadas tem como objetivo adicionar os objetos *saymessage* na lista *M_say_messages*, que é um atributo da classe *action_efector*.

Após preparadas, as mensagens aguardam o momento de serem montadas para enviar para o servidor. Isso acontece quando é feita uma invocação ao método *MakeCommand()*. Este método formata os comandos de todas as ações que os agentes desejam enviar ao servidor, como mover-se para um ponto, mudar o tipo de visão, dar um carrinho, chutar a gol, entre outras. Quando a invoção do *makeSayCommand()* é efetuada para cada tipo de mensagem que foi enviada, é realizada uma codificação do seu conteúdo através da chamada do método *toStr()* de cada mensagem na lista de *M_say_messages*. Esse método codifica a mensagem para que ela possua um tamanho dentro do suportado pelo servidor de mensagens (atualmente 10 *bytes*). Ao final dessa codificação é adicionado no final da mensagem a letra 'j' que indica o conteúdo de que a mensagem trata.

3.3.1 Extensões à Biblioteca

Inicialmente, a necessidade que teve que ser suprida foi a de um conjunto numérico finito que identificasse cada jogada coletiva e que todos os jogadores pudessem ter acesso. Isso aconteceu com a extensão do *namespace rcsc*, o qual foi adicionado o arquivo que pode ser visto no algoritmo 3 da página 58.

Acréscimos ao arquivo *say_message_builder* também tiveram que ser realizados. Foi criada uma classe chamada *JogadaMessage*, que possui oito métodos públicos e alguns atributos privados. O encapsulamento de dados dessa classe foi desenvolvido mantendo a mesma estrutura de outras classes de suporporte à comunicação. A classe em questão herda todos seus métodos da classe *SayMessage*, que é uma classe abstrata onde a maioria de seus métodos são virtuais puros. O construtor da classe recebe seus argumentos de forma a preencher todos atributos do objeto, quando a classe é instanciada. Existem ainda métodos que retornam o tamanho da mensagem transmitida e o caractere de cabeçalho.

Algoritmo 3 Arquivo de Enumerações de Jogadas

```
1  /*
2       Enumeracao de jogadas que foram implementadas
3   */
4
5  #ifndef RCSC_ENUMERACAO_H
6  #define RCSC_ENUMERACAO_H
7
8  #include <stdio.h>
9
10 namespace rcsc {
11
12 enum jogadas{
13    tabela_esq = 1, //jogada ofensiva pela direita
14    lateral_lanc_esq = 2, //jogada ofensiva pela esquerda
15    assistencia_chute_esq = 3, //jogada ofensiva central
16    inversao_esq = 4, //jogadas semi-ofensivas tentando chegar mais proximo
          do gol adversario
17    meio_aberto_lateral_esq = 5,
18    meio_aberto_lateral_dir =6,
19    inversao_dir =7,
20    assistencia_chute_dir = 8, //jogada defensiva de bola recuada
21    lateral_lanc_dir =9, //jogada defensiva aberta pela direita
22    tabela_dir = 0 //jogada defensiva aberta pela esquerda
23 };
24 }
25 #endif
```

Dois métodos tiveram que ser desenvolvidos: de conversão de um número inteiro para a jogada coletiva; e a conversão da jogada coletiva em um número inteiro correspondente. Talvez o método mais interessante dessa classe seja o *toStr()*, compartilhado por todas as classes que herdam de *SayMessage* e reimplementado em cada uma. Esse método formata e insere o conteúdo da variável que contém os comandos que posteriormente são enviados ao *Soccer Server*.

Cada um dos métodos fonéticos na *librcsc* possui o seguinte formato de invocação: *Say<tipoInvocação>*(). Para manter esse padrão, também foi inserido no código um método com o nome *sayJogada()*, que recebe como argumento um tipo *PlayerAgent*. Essa classe faz parte do escopo *SampleCommunication*, assim como os demais métodos *say()*. A invocação do método *sayJogada()* serve como gatilho que dispara uma sequência de ações que montam a mensagem de aúdio, inclusive a função *fuzzy* que avalia qual a melhor jogada para a situação analisada. Após o retorno do método *fuzzy*, são passados os argumentos da função que adiciona a mensagem fonética ao conjunto de comandos que deve ser executado pelo agente.

3.4 Definição das Variáveis e Conjuntos *Fuzzy*

Foram considerados para a tomada de decisão quatro variáveis *fuzzy*: distância ao gol adversário, posição em Campo, ângulo com menor incidência de adversários e conjunto de jogadas coletivas desenvolvidas. Inicialmente pensou-se em considerar a dispersão dos jogadores por regiões do campo ao invés de um ângulo com menor incidência de adversários, mas esta ideia foi abandonada. O motivo do abandono foi que o processamento era demasiadamente "pesado" por requerer discretização do campo, divisão do campo em regiões, análise estatística de dispersão dos jogadores e muitos algoritmos iterativos ou recursivos para medir as distâncias entre os jogadores e encontrar as regiões com mais e menos jogadores e adversários. Em contrapartida, encontrar um lado (leia-se um melhor ângulo) é um processamento que necessita de menos recursos computacionais e, por isso, é mais rápido. A definição das funções de pertinência de cada uma dessas variáveis (distância ao gol adversário, posição em Campo, ângulo com menor incidência de adversários e conjunto de jogadas coletivas desenvolvidas) se deu por tentativa e erro, com a aplicação de testes de mesa e testes com a ferramenta *xfuzzy*. Para cada variável *fuzzy* foram testados intervalos de domínio diferentes (aumentando o tamanho de cada conjunto), números de funções variadas (aumentando ou diminuindo o número de funções que representavam cada universo de discurso), funções diferentes (funções de diferentes formas: senoidal, trapezoidal, triangular) e híbridos desses tipos. Cada uma das variáveis *fuzzy* foi examinada de forma isolada, depois inserida na ferramenta e testada em conjunto com as outras.

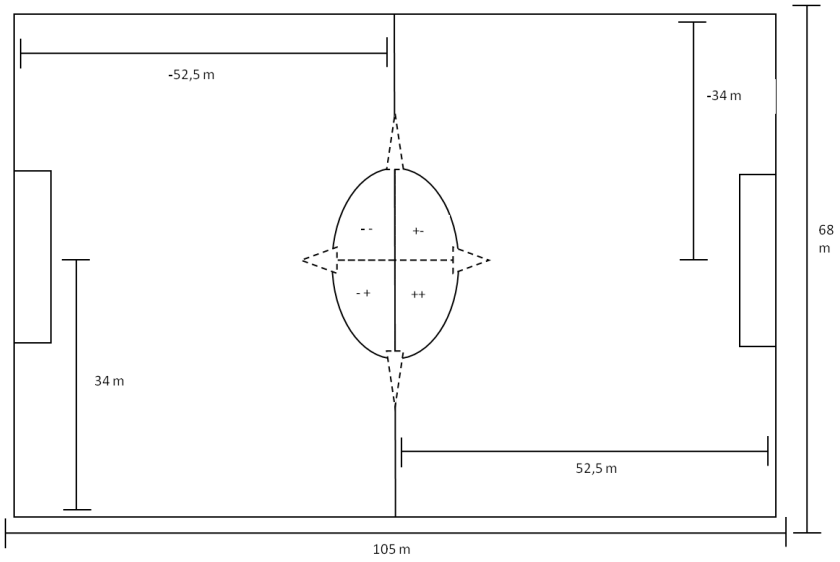


Figura 3.10: Dimensões do Campo e Orientação dos Eixos

A figura 3.10 exibe as dimensões do campo simulado e a orientação dos eixos ordenados

(para baixo e para direita positivos). Baseado nessas dimensões, foram modelados os conjuntos de cada variável *fuzzy*.

A variável linguística *"Posição em Campo"* abrange o intervalo [-34,34] e tem os valores linguísticos *Esquerda, Meio e Direita* (veja figura 3.11 da página 60).

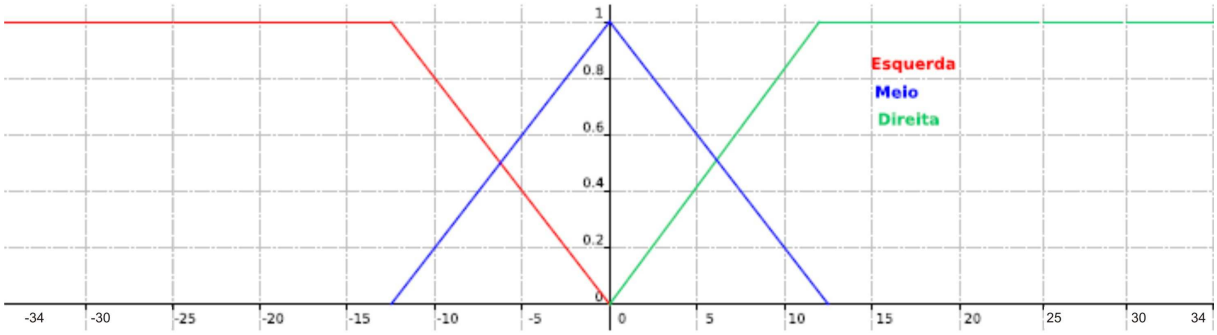


Figura 3.11: Variável linguística *posição em campo*

Do mesmo modo que a variável "*posição em campo*", a variável que avalia a "*distância ao gol adversário*" também se baseia nas medidas do campo simulado, por isso o intervalo dessa variável *fuzzy* é [0,126]. Note que 126m é a maior distância euclidiana entre dois pontos, uma vez que o campo de jogo tem 68X105 de dimensão. A variável linguística "*distância ao gol adversário*" está representada na figura 3.12 da página 60.

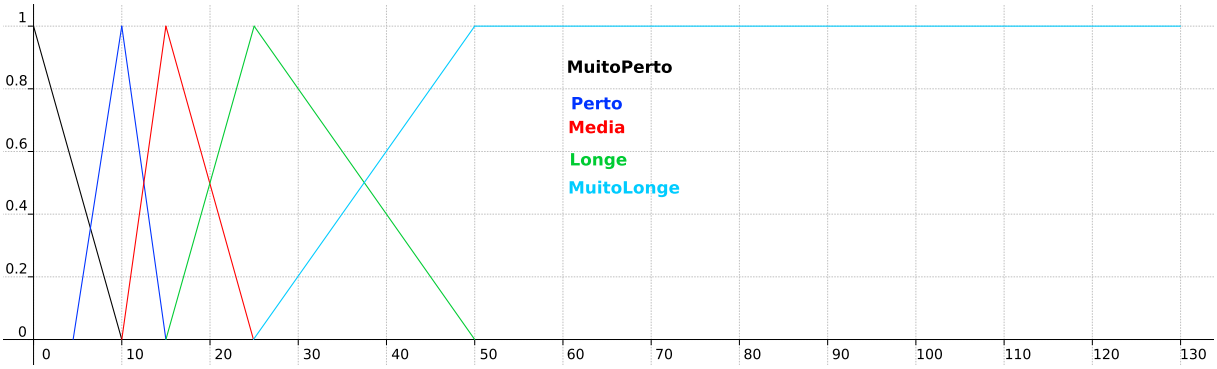


Figura 3.12: Variável linguística *distância do gol adversário*

A variável "*ângulo com menor incidência de jogadores adversários*" visa aumentar a efetividade da jogada coletiva. Para isso, é necessário avaliar a possibilidade de os jogadores aliados receberem a bola. Para o cálculo, projeta-se um vetor entre a bola e cada um dos jogadores do time adversário que estão em um raio de 15m da bola. Em seguida, é calculado o ponto de maior possibilidade de um aliado receber a bola sem interferencia do outro time. Para entender melhor esse processo, veja subseção 3.2.3 da página 49. O campo de visão considerado pode ser observado na figura 3.13 e a variável

linguística "*ângulo com menor incidência de jogadores adversários*" está apresentada na figura 3.14.

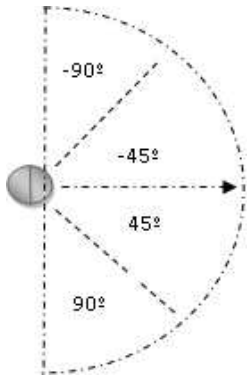


Figura 3.13: Campo de visão avaliado

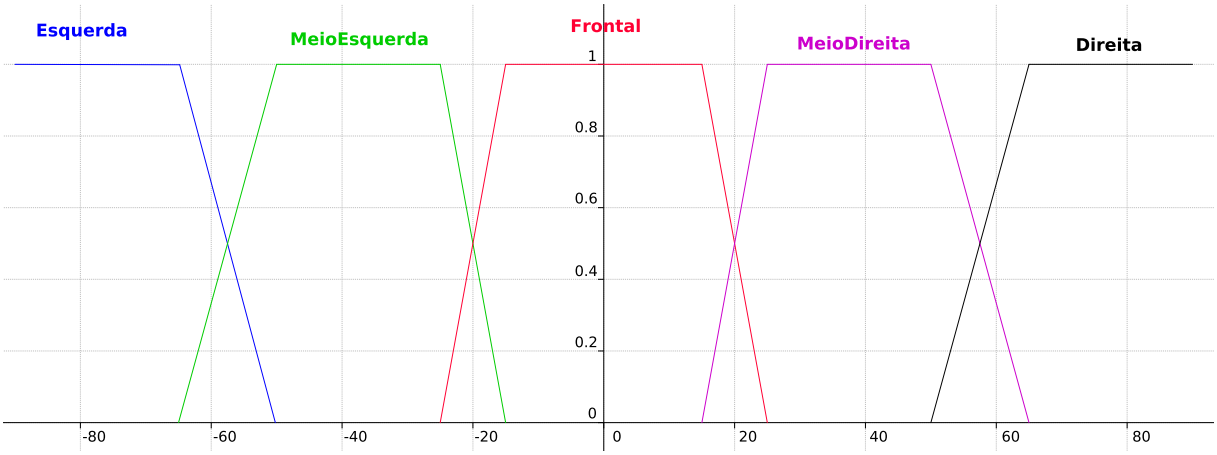


Figura 3.14: Variável linguística *ângulo com menor incidência de jogadores adversários*

A base de regras, do sistema de inferência *fuzzy* deste trabalho, trabalha com três entradas (*distância ao gol adversário, posição em campo* e *ângulo com menor incidência de adversários*) e uma saída (*jogadas coletivas*). A variável linguística *"jogadas coletivas"* tem 10 valores linguísticos, representando diferentes jogadas coletivas, e está representada na 3.12 da página 60. O sistema de inferência *fuzzy*, a partir da base de regras, ativa os conjuntos de saída referentes às jogadas coletivas e, com a defuzzificação, a jogada a ser realizada em determinado momento é decidida. Essa base de conhecimento é composta por pouco mais de trinta regras, que podem ser vistas no apêndice C.1.

3.5 Ferramenta *XFuzzy-3.0.0*

O aplicativo *XFuzzy-3.0.0* é uma ferramenta que auxilia na modelagem de conjuntos *fuzzy*, para isso utiliza uma linguagem formal definida chamada *XFL3*. Segundo

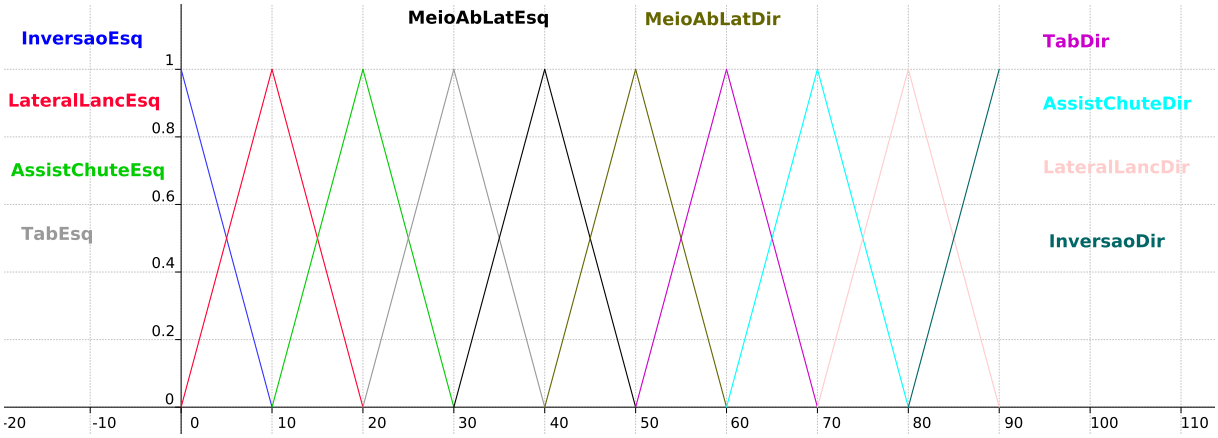


Figura 3.15: Variável linguística *jogadas coletivas*

IMSE-CNM (2003), a linguagem *XFL3* é uma extensão da *XFL* e pode atuar tanto definindo estruturas do sistema quanto modelando operadores *fuzzy*. A linguagem é baseada em *Java* o que a torna flexível e, consequentemente, orientada a objeto. Essa linguagem determina a descrição de um sistema *fuzzy* em duas etapas, sendo a primeira a definição lógica da estrutura do sistema (que são salvas em arquivo com extensão ".xfl") e a segunda é a definição matemática de funções *fuzzy* (que são incluídas nos arquivos de extensão ".pkg"). A sintaxe básica da linguagem opera com oito palavras reservadas: *import, operatorset, rulebase, type, extends, using, if* e *system*.

A ferramenta *xfuzzy* foi manuseada para gerar gráficos e monitorar as saídas do sistema de acordo com cada grupo de entradas. Baseado nas análises das saídas, puderam ser efetuadas correções na definição das funções *fuzzy*, mudar o formato delas, o tamanho do intervalo e procurar híbridos que respondessem o mais próximo possível às soluções esperadas (como o usado para a variável *fuzzy "posição em campo"*). Durante a fase de correção da base de regras e definição dos conjuntos, alguns testes de combinações com diferentes regras e conjuntos foram realizados. Os gráficos gerados durante a primeira etapa de testes deste trabalho podem ser vistos nas imagens 3.16, 3.17, e 3.18 (páginas 63 e 63).

Da análise do gráfico 3.16 é possível verificar que existe uma variação mais brusca com relação a seleção da jogada, quando a distância do gol adversário diminui e a posição em campo (em relação ao eixo "y") é mais centralizada. Foi estipulado para plotagem desse gráfico o valor do ângulo ideal igual a zero (ângulo livre frontalmente).

A figura 3.17 mostra como a jogada selecionada varia quando a relação observada é a distância ao gol adversário e o ângulo com menor incidência de adversários. Foi assumido para plotar esse gráfico o valor fixo do eixo "y" (posição da bola em campo) igual a zero.

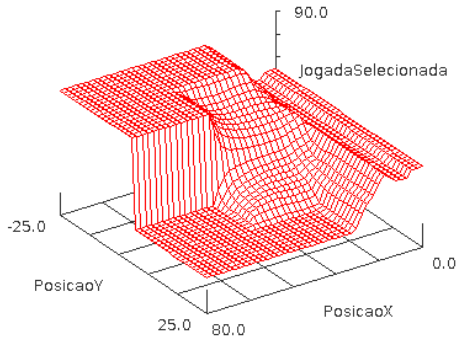


Figura 3.16: Relação Jogada X Distância X Posição

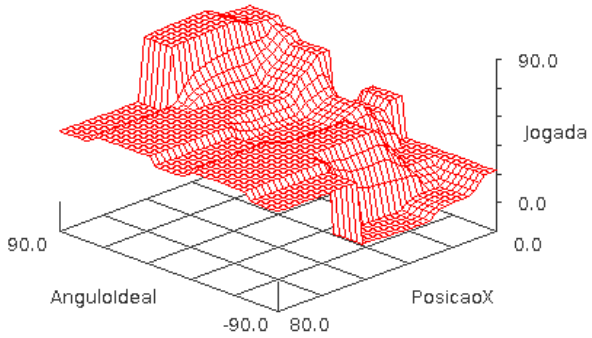


Figura 3.17: Relação com Posição da Ordenada Fixa

Como nos gráficos plotados citados anteriormente, o gráfico 3.18 assume como *default* o valor do centro do conjunto do meio (15) como fixo. Empiricamente constatou-se que as variações da jogada de saída são mais constantes neste caso.

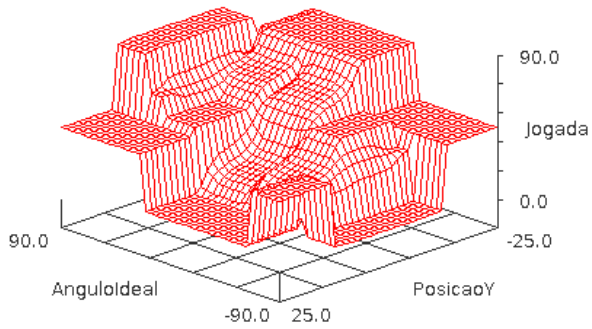


Figura 3.18: Relação com Distância Fixa

Da análise desses conjuntos nota-se que a variação das jogadas sofrem maior influência do ângulo do que das coordenadas onde a jogada inicia. Isso é notado uma vez que a variação das relações entre os eixos e a jogada de saída é menor do que a variação entre qualquer dos eixos e o ângulo. Nos gráficos gerados com a base de regras e os conjuntos

finais notou-se mudanças em alguns resultados, mas manutenção do padrão geral (variação maior quando a bola está próxima ao gol e chamada de jogadas para o lado oposto quando em extremo do campo). Os gráficos resultantes dos testes com os conjuntos definitivos podem ser vistos no apêndice C deste trabalho.

3.6 Acoplação do Sistema *Fuzzy*

A união do sistema de apoio à decisão *fuzzy* ao comportamento dos agentes se deu pela inserção de algumas classes de controle nos mesmos arquivo onde foram inseridas as jogadas coletivas, *bhv_set_play_free_kick.cpp* e *bhv_set_play_free_kick.h*. A escolha da inserção do código do controlador nesse arquivo foi feita por ser o único lugar onde o objeto que seleciona a jogada é utilizado. Diversas classes auxiliam na saída do resultado final, porém dá-se mais ênfase à classe que concentra o processamento principal, chamada de *DecidirJogada*{}, que é invocada pelo método *Bhv_SetPlayFreeKick::tomada_decisao()*.

No escopo de *tomada_decisao()* estão inseridos os métodos que fazem chamadas aos ao sistema de inferência *fuzzy Mandani* (fuzzificação, consulta à base de regras, inferência e defuzzificação). A base de regras usada pelo motor de inferência procura cobrir as principais situações onde se espera invocar cada tipo de jogada coletiva.

No sistema de inferência *fuzzy* as entradas não-*fuzzy* ou precisas (resultantes de medições e observações do simulador) são mapeadas para os conjuntos *fuzzy* (de entrada) relevantes, processo chamado de fuzzificação. Na sequência, as entradas fuzzificadas ativam regras (da base de regras) relevantes para uma dada situação. Fazendo-se uso de operadores lógicos *fuzzy* propostos por Zadeh (1965) (NEGAÇÃO – complemento –, E – mínimo – e OU – máximo) e por Mamdani (1974) (IMPLICAÇÃO – mínimo) sobre as regras ativadas, o processo de inferência obtém o conjunto *fuzzy* de saída. Na defuzzificação é efetuada uma interpretação dessa informação, ou seja, o conjunto *fuzzy* de saída é convertido em uma saída não-*fuzzy* ou precisa pelo método do centróide.

3.7 Aceleração do Treinamento

Uma forma interessante de realizar mais jogos em um espaço de tempo menor é através da aceleração do treinamento. A aceleração é dada pela alteração da *flag synch_mode*, na linha 265 do arquivo *server.conf*, no diretório oculto em *pastapessoal/.rcssserver/server.conf*, onde são atribuídos alguns valores de configuração do servidor. *Synch_mode* é por padrão

assumida como valor *false*, isso faz com que o servidor controle o monitor para que a velocidade do modo de alta sincronização dos jogadores seja baixa, ou seja, na prática isso implica que os robôs ficam com uma velocidade próxima do futebol de humanos. Contudo, a aceleração do treinamento (período de testes de uma equipe) é desejável uma vez que obtém-se economia de tempo.

Esse resultado pode ser alcançado através da mudança do valor da *flag synch_mode* de *false* para *true*, que reduzirá o tempo de jogo pela metade. O tempo que levaria para realizar uma partida no modo normal, agora poderão ser executadas duas partidas. No entanto vale ressaltar que para que isso ocorra, o algoritmo não pode responder, em cada ciclo normal, com mais da metade de um ciclo.

No diretório oculto também existem outras configurações, não só do *soccerserver* mas também do *player*, que podem ser mudadas para fins de análise de jogo. Por exemplo, existe no arquivo de definições do servidor uma variável que marca a duração de alguns comandos enviados ao *soccerserver*. Cita-se como exemplo o caso do *pointoTo()*, que por padrão dura 20 ciclos. Esses valores, entre outros, podem ser alterados para treinar casos especiais de alguma estratégia pré-definida.

3.8 Técnicas e Meios Selecionados

A simulação 2D é um ambiente amplamente incerto. Nunca se sabe com absoluta certeza a verdade sobre algum fato, seja este a posição de um jogador, velocidade da bola ou até mesmo a qualidade das mensagens de áudio transmitidas. Sobre todos os aspectos citados, e muitos outros omitidos, há um certo grau de incerteza inserido. Esse fato leva a pensar sobre os tipos de soluções que podem ser adotadas para trabalhar com incertezas. Sistemas *fuzzy* baseados em conhecimento trabalham muito bem com problemas desse tipo. Na seção 3.1 foram mostradas algumas soluções para a liga de simulação de futebol de robôs.

Na liga de simulação, ao decidir uma ação que o agente deverá realizar, deve-se sempre considerar a previsão do ambiente nos próximos ciclos (mesmo sendo um ambiente dinâmico), ou existe a possibilidade de insucesso na execução das ações. Uma boa análise de jogo aliada ao aumento da cooperação direta traz como vantagens a harmonia encontrada pelo time. Os resultados da aplicação dos mecanismos desenvolvidos neste trabalho podem ser vistos mais detalhadamente no capítulo 4.

A ferramenta *xfuzzy* foi de grande importância para a etapa de testes, uma vez que

antes de sua utilização os testes para definição dos conjuntos estavam sendo realizados manualmente, o que era cansativo e demorado. O modo gráfico da ferramenta possibilitou a realização de um número maior de testes e uma visualização mais precisa dos resultados. A aplicação de correções também se mostrou satisfatória ao utilizar a ferramenta.

O futebol de robôs é um problema padrão bastante motivador. O desafio de desenvolver/aplicar técnicas e táticas tentando contribuir de forma positiva para a equipe (acompanhando o estado da arte), foi uma das razões para a escolha da utilização da *framework Agent2D*. Desde a sua definição como ferramenta principal, a equipe tem melhorado o desempenho, ver Ferreira (2010).

4 Testes e Resultados

Alguns testes com a base de regras do sistema de inferência *fuzzy* foram realizados em situações nos limitantes superior, inferior e no centro de cada conjunto. Os resultados destes testes podem ser acompanhados no apêndice C.2. A monitoração das saídas mostra que a seleção da jogada sofre maior influência do ângulo selecionado, já que quando este é fixo, as jogadas resultantes da invocação da inferência apresentam menor variação. As saídas mais "altas" representam jogadas pelo lado direito do campo e as mais baixas, jogadas pelo lado esquerdo. Pode-se observar ainda que, também, há mudanças mais constantes conforme a bola se aproxima do gol oponente, esse comportamento se deve ao fato de a base de conhecimento abranger mais regras para estas situações.

Foram ainda realizados testes visando possibilitar comparação da equipe *iBots* implementada com as soluções propostas neste trabalho (chamada *iBots Nova*) em relação à equipe *iBots* sem as soluções propostas (chamada *iBots Antiga*). Para isso, essas duas equipes realizaram cinco partidas contra a equipe a *FCPortugal (versão 2007)*. Dos testes foram coletadas as seguintes informações:

- tempo de posse de bola, em ciclos, da equipe *iBots (Nova e Atinga)*, desde o momento da chamada para realizar uma jogada coletiva até o momento em chuta a gol ou perde a bola para a equipe *FCPortugal* ou altera o modo de jogo, sair de *PlayOn*;
- se durante o tempo de posse de bola, citado anteriormente, a equipe *iBots (Nova e Antiga)* fez gol ou não;
- tempo de posse de bola da equipe *FCPortugal*, desde o momento da roubada de bola até chutar a gol ou perder a bola para a equipe *iBots (Antiga ou Nova)* ou alterar o modo de jogo, sair de *PlayOn*;

Especificamente sobre a equipe *iBots Nova*, obteve-se ainda a informação de que a jogada coletiva foi executada como planejada ou parcialmente executada (somente um

jogador, diferente do que inicia a jogada coletiva, participou da jogada) mas ficou com posse de bola ou parcialmente executada e perdeu a posse de bola ou não foi possível executar.

Para obter as informações citadas anteriormente durante as partidas, foi necessário implementar algoritmos específicos. Todas elas são informações que o simulador e a *framework Agent2D* não disponibilizam. A implementação desses algoritmos demandou muito tempo o que resultou escassez de tempo e resultou em poucos jogos de testes.

Os dados das partidas realizadas nos testes foram sumarizados e estão apresentadas na tabela 4.1. É necessário acrescentar que todas as informações são referentes somente ao período referente ao ataque resultante do momento de chamada para realizar uma jogada coletiva até o término do ataque seguinte à roubada de bola pelo *FCPortugal*.

Tabela 4.1: Comparação Entre *iBots* Antiga e *iBots* Nova

	iBots Antiga	iBots Nova
Média da posse de bola	45 ciclos	49,8 ciclos
Média da posse de bola FCPortugal	52 ciclos	25,3 ciclos
Média de chutes que resultaram em gols	0,4	0,3

Diante dos resultados obtidos, nota-se que: houve um pequeno aumento da posse de bola com a equipe *iBots Nova*; uma redução significativa no tempo de posse de bola da *FCPortugal*, ou seja, a roubada de bola da equipe *iBots* apresentou-se mais eficiente; a média de chutes que resultaram em gols teve uma leve redução.

É importante expressar que cada partida é uma nova partida, ou seja, uma nova história completamente diferente porque o ambiente do simulador é dinâmico não e não determinístico. Com isso, para se obter resultados mais fidedignos é necessário realizar mais jogos de teste para coletar mais informações. Diante dos resultados obtidos, o mais expressivo é a roubada de bola mais rápida pela equipe *iBots Nova*. Por outro lado, a média da posse de bola da equipe *iBots* e a média de chutes que resultaram em gols foram bem parecidos, a diferença pode ter sido provocada pela alta aleatoriedade do ambiente simulado, o que tende a reduzir com mais jogos de teste.

Foi obtida ainda a informação de que aproximadamente 33,33% da jogadas coletivas foi executada como planejada; aproximadamente 66,67% das jogadas coletivas foi executada parcialmente e ficou com a posse de bola; 0,0% foi executada parcialmente e perdeu a posse de bola; e 0,0% não foi possível executar. Nota-se em nenhum momento, durante a chamada de uma jogada coletiva, a bola foi roubada pelo adversário *FCPortugal*. Para

validar esses resultados é importante realizar testes com outras equipes de diferentes níveis.

5 Conclusão

Como contribuições deste trabalho citam-se a modelagem e aplicação de sistema de inferência fuzzy para decidir qual jogada coletiva adotar em momentos de bola parada, o que pode ser expandido para outras situações simplesmente acrescentando novas regras; definição própria da estrutura de mensagem para realização de jogadas coletivas; a mensagem carrega consigo um *byte* de conferência, aumentando a confiabilidade da mensagem uma vez que o simulador pode acrescentar ruído. Como contribuições mais importantes para a equipe *iBots* pode se citar: uso de mensagens fonéticas para cooperação entre os agentes, o que não ocorria; e realização de jogadas coletivas em bolas paradas, o que abre horizontes para jogadas coletivas com bola "rolando" como jogadas ensaiadas e linha de impedimento.

Na fase de testes, observou-se visualmente que as jogadas coletivas em situações de bola parada um comportamento inusitado. Quando o jogador vai cobrar a saída de bola, ele se desloca para a bola e enquanto isso os companheiro de equipe permanecem parados e os adversários continuam estáticos, efetuando uma marcação. Mas quando o agente que vai cobrar a saída de bola envia mensagens de áudio para seus companheiros de equipe, estes companheiros correm para se posicionar e dar condições à execução da jogada e inicia uma ação coletiva de reposicionamento. Esse reposicionamento causa "confusão" temporária nos jogadores da equipe adversária, o que é favorável à ação a ser executada e é indício do potencial de exploração que surge com as primeiras jogadas coletivas implementadas.

Essa movimentação dos companheiros de equipe se deve ao fato de o comportamento padrão da triangularização de *Delaunay*, adotada pela *framework Agent2D*, tentar manter equidistância entre os jogadores. Assim que os jogadores selecionados começam a se mover para a posição de início da jogada, uma ação conjunta de reposicionamento é disparada.

Deve-se acrescentar ainda que as jogadas coletivas implementadas neste trabalho acontecem poucas vezes, sendo que em algumas raras vezes sua ocorrência tende a zero. Apesar

disso, é esperado da equipe que quanto mais esse tipo de situação aconteça, mais essas jogadas desenvolvidas venham a colaborar com o aumento da efetividade da equipe.

Sugere-se como trabalhos futuros:

- aplicação de comunicação para jogadas que não são de bola parada, ou seja, aplicação de jogadas coletivas para o modo *PlayOn*;
- aplicação de jogadas coletivas para grupos de jogadores. Por exemplo, jogadas coletivas para todos jogadores de defesa, de meio campo, de ataque, de lateral, entre outros possíveis;
- expandir a comunicação para que ela possa ser utilizada para outros tipos de interações entre os agentes (companheiros de equipe comunicar sobre informações do adversário, por exemplo), incluindo o técnico;
- utilizar a comunicação e habilidades do agente técnico (que tem acesso às informações sem ruído e pode construir um modelo de mundo global da partida);
- desenvolvimento de algoritmos híbridos e outras técnicas de suporte a decisão para comparação com a solução baseada em sistema de inferência *fuzzy* deste trabalho;
- unir uma memória de trabalho à base de regras do sistema desenvolvido;
- desenvolvimento de jogadas coletivas por meio de aprendizado por reforço.

Referências Bibliográficas

AGUIAR, H.; JÚNIOR, O. *Lógica Difusa - Aspectos Práticos e Aplicações*. [S.l.]: Sindicato Nacional dos Editores de Livros, 1999. ISBN 85-7193-024-4.

AKIYAMA, H.; SHIMORA, H. *HELIOS2010 Team Description*. [S.l.], 2010.

ALMEIDA, R. M. F. de. Análise e previsão das formações das equipas no domínio do futebol robótico. p. 24–60, oct 2008.

BOER, R. de; KOK, J. *The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team*. Dissertação (Mestrado) — Faculty of Science University of Amsterdam, feb 2002.

CARVALHO, F. G. de. *Comportamento em Grupo de Personagens do Tipo Black&White*. Dissertação (Mestrado) — Pontífica Universidade Católica do Rio de Janeiro, mar. 2004.

CHEN, M. et al. *RoboCup Soccer Server*. [S.l.], feb 2003.

CHEN, X. et al. Wrighteagle team description for robocup@home 2009. *Robocup*, 2009.

ELAKIRI. abr. 2011. [Online; accessed 1-Abril-2011]. Disponível em: <http://www.elakiri.com/forum/showthread.php?p=6640168>.

FERNÁNDEZ, C. Á. I. *Definición de una metodología para el desarrollo de sistemas multiagente*. Tese (Doutorado) — DEPARTAMENTO DE INGENIERÍA DE SISTEMAS TELEMÁTICOS, 1998.

FERREIRA, G. B. Uso dinâmico de esquemas táticos com redes neurais perceptron de múltiplas camadas na equipe ibots de futebol de robôs simulados. *Universidade Federal do Tocantins*, nov. 2010.

FIRA. *Federation of International Robot-soccer Association*. abr. 2011. [Online; accessed 18-maio-2011]. Disponível em: <http://www.fira.net>.

FRACCAROLI, E. S. Sistema fuzzy aplicado ao time uspsim_2008 da categoria robocup simulation 2d. *Universidade de São Paulo, Escola de Engenharia de São Carlos*, 2008.

GERNDT, R. et al. On the aspects of simulation in the robocup mixed reality soccer systems. *Workshop Proceedings of SIMPAR Workshop Proceedings of SIMPAR*, p. 159–166, nov. 2008.

GOLVEIA, H. B. Sistema estrategista para futebol de robôs. *FACULDADE DE INFORMÁTICA DE PRESIDENTE PRUDENTE*, 2006.

IMSE-CNM. *FUZZY LOGIC DESIGN TOOLS*. [S.l.], 2003.

KOHAGURA, T. Lógica fuzzy e suas aplicações. *Universidade Estadual de Londrina*, nov. 2007.

LARC. *Latin American Robotics Competition*. 2010. [Online; accessed 19-Janeiro-2011]. Disponível em: <http://www.larc10.fei.edu.br/>.

MACKWORTH, A. On seeing robots. *World Scientific Press*, p. 1–13, 1993.

MAMDANI, E. H. Application of fuzzy algorithms for control of simple dynamic plant. *IEEE (Control and Science)*, v. 121(12), p. 1585–1588, 1974.

MARTINS, M. F. et al. Um protocolo confiável e flexível de comunicaÇão para futebol de robôs. *VII SBAI / II IEEE LARS*, set. 2005.

MATTOSO, D. E. Sistema especialista fuzzy para posicionamento dos jogadores aplicado ao futebol de robôs. *Universidade Federal da Bahia*, 2010.

PINTO, A. dos S. *Simulação e Avaliação de Comportamentos em Sistemas Multi-Agentes baseados em Modelos de Reputação e Interação*. Dissertação (Mestrado) — UNIVERSIDADE DO VALE DO RIO DOS SINOS, 2008.

REIS, L. P. *Coordination in Multi-Agent Systems: Applications in University Management and Robotic Soccer*. Tese (Doutorado) — Faculdade de Engenharia da Universidade do Porto, jul. 2003.

REZENDE, S. O. *Sistemas Inteligentes: Fundamentos e Aplicações*. Manole. [S.l.]: Rede Cooperativa de Pesquisa em Inteligência Artificial, 2005. ISBN 85-204-1683-7.

ROBOCUP. *Robocup Federation*. 2010. [Online; accessed 18-Setembro-2010]. Disponível em: <http://www.robocup.org/>.

ROBOCUP2007, A. C. *Rules Soccer Simulation League 2D RoboCup2007 Atlanta*. [S.l.], jun. 2007. Disponível em: <http://wiki.cc.gatech.edu/robocup/index% -.php/Soccer_Simulation>.

RUSSELL, S. J. et al. *Artificial Intelligence*. Elsevier. [S.l.]: Editora Campus, 2003.

SANDRI, S.; CÔRREA, C. Lógica nebulosa. *V Escola de Redes Neurais: Conselho Nacional de Redes Neurais*, p. c073–c090, jun. 1999.

SCHERRER, E. L. Coordenação de agentes inteligentes aplicada a um time de futebol de robôs simulado para a robocup small size robot league. *CENTRO UNIVERSITÁRIO FEEVALE*, jun. 2006.

SHAW, I. S.; SIMõES, M. G. Controle e modelagem fuzzy. *FAPESP*, 1999.

SILVA, A. T. R. da et al. ibots 2010: Descrição do time. *Latin American Robotics Competition*, out. 2010.

SILVA, B. A. da; SANTOS, L. R. dos. Tópicos de robótica móvel. *Universidade Federal da Grande Dourados*, 2010.

SILVA, H. da L. et al. Controladores fuzzy para agentes robôs jogadores de futebol. *Núcleo de Arquitetura de Computadores e Sistemas Operacionais (ACSO)*, abr. 2007.

SILVA, H. G. da. Futebol de robôs: Comportamento social cooperativo. *Universidade Federal do Tocantins*, jul. 2010.

SILVA, L. R. da. *ANÁLISE E PROGRAMAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS DA PLATAFORMA EYEBOT*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, mar. 2003.

STRASSMANN, D. S. Desenvolvimento de uma arquitetura baseada em subsunção para um time de futebol de robôs. *UNIVERSIDADE FEDERAL DE SANTA CATARINA CENTRO TECNOLÓGICO*, ago. 2006.

TANSCHEIT, R. Lógica fuzzy: Formatos e operações. s/d. Disponível em: <http://url20.ca/TansCheit>.

XAVIER, R. O. et al. O time de futebol simulado itandroids-2d. *Anais do XXVI Congresso da SBC*, 2006.

ZADEH, L. A. Fuzzy sets. *journal Information and Control*, p. 338–353, 1965.

APÊNDICE A – Dificuldades Encontradas

Alguns bons desafios foram superados durante a etapa de desenvolvimento desse trabalho, podemos citar como principais a utilização de algoritmos *fuzzy*, os qual não foram estudados formalmente no curso. Posteriormente podemos citar as dificuldades relacionadas ao desenvolvimento dos testes. O trabalho com diversas *frameworks* foi realmente motivador. A projeção e extensão das propostas na *librcsc* e na *Agent2D* foram bem trabalhosas já que fazem parte de um universo de pesquisa bem diferente do que está presente no cotidiano. Há também a questão do trabalho pioneiro, até onde se sabe, com simulação 2D na região norte do Brasil e aplicação no futebol robótico.

A aprendizagem de novas linguagens, tanto de programação, como C++, quanto de definição como a (*XFL3*), levaram algum tempo. Sendo até cansativo às vezes o *debug* de alguns segmentos de código. Mas após algum tempo de trabalho o desenvolvimento se tornou bem mais rápido.

Demora de compilação da *framework*, em média 10 a 15 minutos de compilação. Demora de compilação também da biblioteca, em médias 5 ou 10 minutos. Isso faz o processo de desenvolvimento ser lento e cansativo. Há situações em que ainda é necessário compilar a biblioteca, depois a *framework*, geralmente para esses casos o processo de compilação dura pouco mais que 30 minutos, sendo 5 minutos para compilação e instalação da biblioteca e 25 minutos para compilação e linkagem da ferramenta e biblioteca.

A inserção de arquivos na *framework* se mostrou um empecilho realmente desgastante. Primeiramente, é interessante lembrar que os binários gerados pela aplicação estão intimamente ligados aos arquivos objeto gerados durante a etapa de compilação e não é simples inserir um novo arquivo, uma vez que o script *make*, pré-estabelecido, utiliza linkagem dinâmica de bibliotecas. A solução para este problema foi tentar criar um arquivo de comando *cmake* que permitisse a inserção de novos arquivos e implementasse uma forma de linkagem genérica para todos arquivos.

A sincronização dos agentes de acordo com a mudança nos modos de jogo também

mostrou-se uma barreira de difícil superação, já que a expectativa inicial seria de que as jogadas coletivas seriam tratadas inteiramente em modos de jogo onde a bola estivesse parada e estática. Porém a continuação da jogada, exige que o comportamento do agente seja adaptado de forma a favorecer o sucesso da ação conjunta.

A.0.1 Configurações da Máquina e do Ambiente

Os testes dos conjuntos *fuzzy* e as extensões à biblioteca e à *framework* foram realizados em uma máquina com processador *Intel Celeron M 440*, com 3GB de memória RAM DDR3, HD de 80GB e rodando o sistema operacional *Ubuntu Maverick Meerkat*, versão 10.10.

A.1 Instalação e Dependências

O ambiente *Soccer Server + Soccer Monitor_Frameview + Times Base* necessita que alguns requisitos de softwares sejam sanados para que possa executar a simulação com sucesso. A instalação das dependências listada na tabela A.1, na página 77, é o mínimo exigido para que possamos "rodar"uma partida de futebol entre os agentes, no simulador.

Tabela A.1: Lista de Dependências para o *Soccer Simulator*

Pacote a Instalar	**Descrição**
`csh`	C shell, muito utilizado em ambientes BSB. Aumenta as opções de trabalho via linha de comando.
`bison`	Oferece várias opções de geração de código, entre essas, definição de namespaces específicos.
`liboost`	(todas dependências) Coleção de bibliotecas usadas para debug e link temporal de aplicações distribuídas.
`filesystem`	(todas dependências) Oferece funcionalidades para manipulação de *paths*, arquivos e diretórios. O objetivo é facilitar a portabilidade de saídas geradas por aplicações C++.
`QtCore4`	É uma cross-plataforma para aplicações que utilizam *frameworks* paa aplicações C++.
`libgd`	Biblioteca de Código aberto para criação dinâmica de imagens.
`cross`	Suporte a aplicações multi-agentes do tipo cliente-servidor.
`fork`	Suporte a segmentação de processos distribuídos.
`libtool`	Pacote usado para construção de sistemas e aplicações em *Grid*.
`whether`	Provê conectividade *Multicast*
`posix`	É uma família de normas definidas pelo IEEE e designada formalmente por IEEE 1003, que tem como objetivo garantir a portabilidade do código-fonte de um programa a partir de qualquer sistema operacional que atenda as normas POSIX.
`qt4`	Automatiza a geração do *Makefile*.
`qtcore`	Módulo que provê extensão às funcionalidades oferecidas pelo qt4
`libxm`	Módulo suporte a serialização/deserialização. Ajuda também no desenvolvimento de cabeçalhos de código C++.
`gprof`	É uma ferramenta para análise dinâmica da execução de programas escritos em linguagens C e C++.
`automake`	É uma ferramenta que automatiza as gerações dos *Makefile.in* para cada chamada dos *Makefile.am*.
`boost-devel`	Provê *templates* de fluxos de dados para trabalhos com C++.
`qt-devel`	Conjunto de ferramentas que permite desenvolvimento de projetos qt.
`desktop-file-utils`	Pacote para trabalhar em ambiente *Desktop*.
`libaudio`	Utilizada para codificação e decodificação de sinais de áudio.
`librcsc`	Plataforma base para "rodar"a *Agent2D*.

APÊNDICE B – Tabelas

B.1 Tipos Interação Entre Agentes

B.2 Performativa FIPA-ACL

Tabela B.1: Tipos de Interação Entre Agentes. (REZENDE, 2005)

Tipos de Interação	**Interação X ↔ Y**	**Características da Interação**
Neutralismo	0 ↔ 0	Não ocorre interação entre os agentes
Competição	- ↔ -	Ambos os agentes são atingidos negativamente, pois competem pelos mesmos recursos. Por exemplo, em um leilão eletrônico, o lance de um agente, faz o outro ficar mais longe do seu objetivo.
Amensalismo	- ↔ 0	Um agente é afetado pela ação não proposital de outro agente. Por exemplo, um agente, ao atuar, pode apagar informações de uma fonte usada por outros agentes.
Parasitismo	+ ↔ -	Um agente depende do outro para sua existência e afeta o segundo negativamente. É exemplo desse tipo de relação, a atuação de um vírus em um computador
Predação	+ ↔ -	A meta de um agente (predador) é a eliminação de outro agente (presa). No caso do leilão eletrônico temos como exemplo o caso em que um agente verifica que outro está prestes a vencer o leilão. Este pode agir impedindo a comunicação do competidor como leilão, eliminando-o da concorrência.
Comensalismo	+ ↔ 0	A interação beneficia apenas um dos agentes (o comensal), mas não prejudica o outro (o hóspede). Por exemplo, um agente requerendo informações não confidenciais a outro.
Proto-Cooperação	+ ↔ +	A interação entre os agentes otimiza a obtenção dos resultados, embora não seja vital. Por exemplo, agentes que tenham por meta fazer um levantamento de preços de diversos produtos. Ambos podem realizar essa tarefa isoladamente, porém a atividade será acelerada com a divisão do trabalho.
Simbiose	+ ↔ +	A interação entre os agentes é obrigatória para obtenção de suas metas. Por exemplo, dois agentes que precisem trabalhar continuamente em suas tarefas, porém as máquinas em que estão "hospedados"ficam ligados apenas em períodos alternados. Uma maneira de garantir a sobrevivência de ambos é "hospedar"os dois no mesmo servidor que esteja ligado.

Tabela B.2: Performativas da linguagem *FIPA-ACL*.(REIS, 2003).

Performativa	**Significado**
`accept-proposal`	Aceitação de proposta numa negociação
`agree`	Aceitação de desempenhar uma dada ação
`cancel`	Cancelamento da execução de uma dada ação
`cfp`	"*Call for proposals*" . Utilizada para iniciar uma dada negociação
`confirm`	Confirmação da veracidade de uma dada mensagem
`disconfirm`	Inverso da mensagem anterior
`failure`	Uma tentativa de executar uma dada ação (usualmente requisitada por outro agente) que falhou
`inform`	Uma das performativas mais importantes da FIPA ACL. Permite comunicar uma informação aos outros agentes
`inform-if`	Informação sobre a veracidade de determinada informação
`inform-ref`	Informação sobre um dado valor
`not-understood`	Indicação de que uma dada mensagem não foi percebida
`propagate`	Pedido de propagação a um conjunto de agentes de uma dada mensagem
`propose`	Envio de proposta, por exemplo, como resposta a uma mensagem cfp
`proxy`	Permite enviar uma mensagem que vai ser reenviada a um conjunto de agentes
`query-if`	Pedido de informação sobre a veracidade de determinada informação
`query-ref`	Pedido de informação sobre um dado valor
`refuse`	Recusa de executar determinada ação
`reject-proposal`	Recusa de uma proposta efectuada no contexto de uma dada negociação
`request`	Pedido a um dado agente para executar determinada ação
`request-when`	Pedido para executar uma dada acção quando uma determinada condição for verdadeira
`request-whenever`	Pedido para executar uma dada acção sempre que uma determinada condição seja verdadeira
`Subscribe`	Pedido para ser informado acerca das alterações relacionadas com determinado fato ou informação

APÊNDICE C – Extras

C.1 Base de Regras

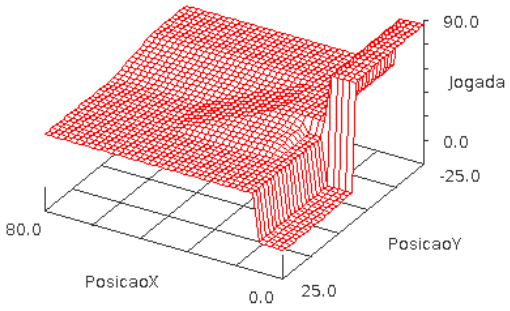
Algoritmo 4 Base de Regra Utilizadas pelo Sistema de Inferência

```
43 rulebase ProcessoDecisao (DistanciaGolAdv PosX, PosicaoCampo PosY,
       MelhorAngulo AngIdeal : Jogada jogada) using DefinicaoOperador {
44   if(PosX == MuitoPerto & PosY == Esquerda) -> jogada = InversaoDir;
45   if(PosX == MuitoPerto & PosY == Direita) -> jogada = InversaoEsq;
46   if(PosX == MuitoPerto & PosY == Meio & AngIdeal == MeioEsquerda) ->
       jogada = AssistenciaChuteEsq;
47   if(PosX == MuitoPerto & PosY == Meio & AngIdeal == MeioDireita) -> jogada
       = AssistenciaChuteDir;
48   if(PosX == Perto & (+(PosY == Esquerda)) & AngIdeal == Direita) -> jogada
       = InversaoDir;
49   if(PosX == Perto & (+(PosY == Direita)) & AngIdeal == Esquerda) -> jogada
       = InversaoEsq;
50   if(PosX == Perto & PosY == Meio & (AngIdeal == Direita | (+(AngIdeal ==
       MeioDireita)))) -> jogada = AssistenciaChuteDir;
51   if(PosX == Perto & PosY == Meio & (AngIdeal == Esquerda | (+(AngIdeal ==
       MeioEsquerda)))) -> jogada = AssistenciaChuteEsq;
52   if(PosX == Media & PosY == Esquerda & AngIdeal == Frontal) -> jogada =
       MeioAberLatDir;
53   if(PosX == Media & (+(PosY == Direita)) & AngIdeal == Frontal) -> jogada
       = MeioAberLatEsq;
54   if(PosX == Media & (+(PosY == Esquerda)) & (AngIdeal == Direita |
       AngIdeal == MeioDireita)) -> jogada = InversaoDir;
55   if(PosX == Media & (+(PosY == Direita)) & (AngIdeal == Esquerda |
       AngIdeal == MeioEsquerda)) -> jogada = InversaoEsq;
56   if(PosX == Media & PosY == Meio & (+(AngIdeal == Esquerda))) -> jogada =
       LateralLancEsq;
57   if(PosX == Media & PosY == Meio & (+(AngIdeal == Direita))) -> jogada =
       LateralLancDir;
58   if(PosX == Media & PosY == Meio & AngIdeal == MeioEsquerda) -> jogada =
       AssistenciaChuteEsq;
59   if(PosX == Media & PosY == Meio & AngIdeal == MeioDireita) -> jogada =
       AssistenciaChuteDir;
60   if(PosX == Longe & PosY == Esquerda & AngIdeal == Frontal) -> jogada =
       MeioAberLatDir;
```

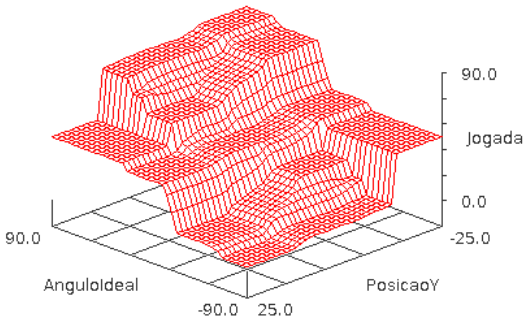
Algoritmo 5 Continuação do Algoritmo 4

```
61 if(PosX == Longe & PosY == Direita & AngIdeal == Frontal) -> jogada =
       MeioAberLatEsq;
62   if(PosX == Longe & PosY == Direita & (AngIdeal == MeioEsquerda | AngIdeal
         == Esquerda)) -> jogada = InversaoEsq;
63   if(PosX == Longe & PosY == Esquerda & (AngIdeal == MeioDireita | AngIdeal
         == Direita)) -> jogada = InversaoDir;
64   if(PosX == Longe & PosY == Meio & AngIdeal == MeioEsquerda) -> jogada =
       TabelaEsq;
65   if(PosX == Longe & PosY == Meio & AngIdeal == MeioDireita) -> jogada =
       TabelaDir;
66   if(PosX == Longe & PosY == Meio & AngIdeal == Esquerda) -> jogada =
       LateralLancEsq;
67   if(PosX == Longe & PosY == Meio & AngIdeal == Direita) -> jogada =
       LateralLancDir;
68   if(PosX == Longe & PosY == Meio & AngIdeal == Frontal) -> jogada =
       MeioAberLatDir;
69   if(PosX == MuitoLonge & PosY == Meio & AngIdeal == Frontal) -> jogada =
       MeioAberLatEsq;
70   if(PosX == MuitoLonge & PosY == Meio & AngIdeal == MeioEsquerda) ->
       jogada = TabelaEsq;
71   if(PosX == MuitoLonge & PosY == Meio & AngIdeal == MeioDireita) -> jogada
         = TabelaDir;
72   if(PosX == MuitoLonge & (PosY == Meio | PosY == Direita) & AngIdeal ==
       Esquerda) -> jogada = InversaoEsq;
73   if(PosX == MuitoLonge & (PosY == Meio | (+(PosY == Esquerda))) & AngIdeal
         == Direita) -> jogada = InversaoDir;
74   if(PosX == MuitoLonge & PosY == Esquerda & AngIdeal == Frontal) -> jogada
         = MeioAberLatDir;
75   if(PosX == MuitoLonge & PosY == Direita & AngIdeal == Frontal) -> jogada
       = MeioAberLatEsq;
76   if(PosX == MuitoLonge & PosY == Esquerda & AngIdeal == MeioDireita) ->
       jogada = LateralLancDir;
77   if(PosX == MuitoLonge & PosY == Direita & AngIdeal == MeioEsquerda) ->
       jogada = LateralLancEsq;
78  }
```

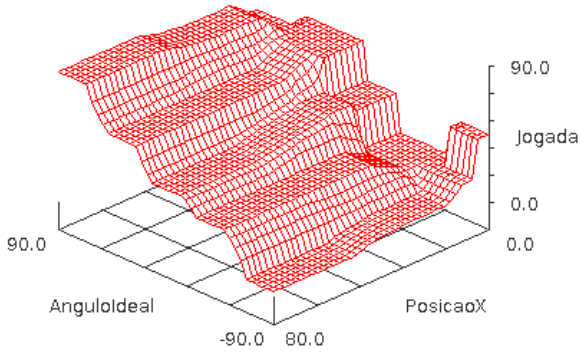
C.2 Gráficos de Testes nos Conjuntos *Fuzzy*



(a) Ângulo Fixo em 0

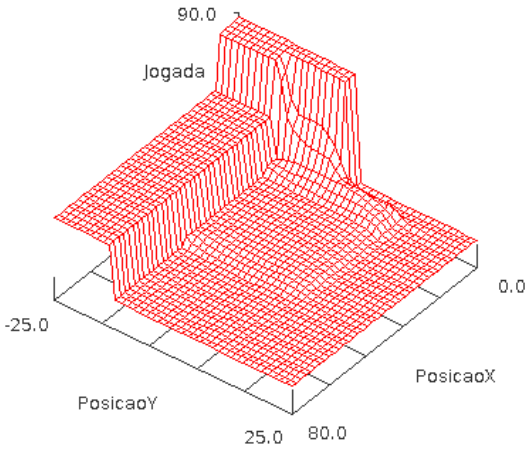


(b) Distância Fixa em 15

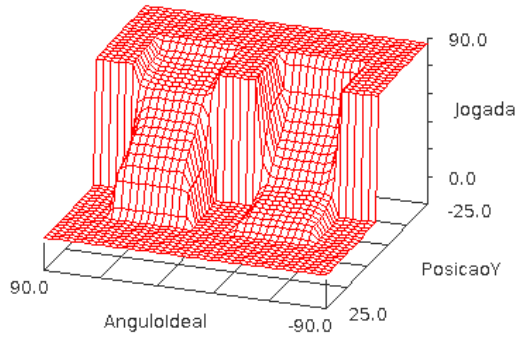


(c) Posição Fixa em 0

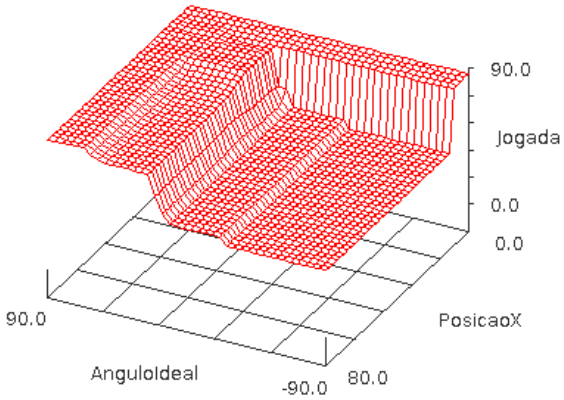
Figura C.1: Testes com Valores Fixos no Centro de Cada Conjunto



(a) Ângulo Fixo em -90

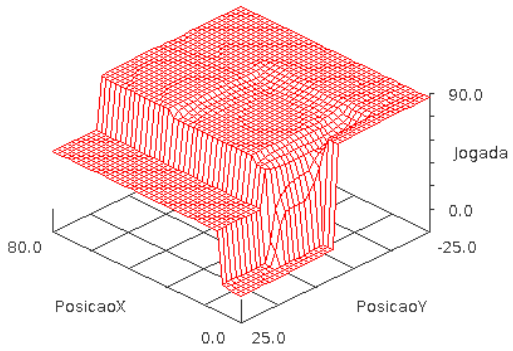


(b) Distância Fixa em 0

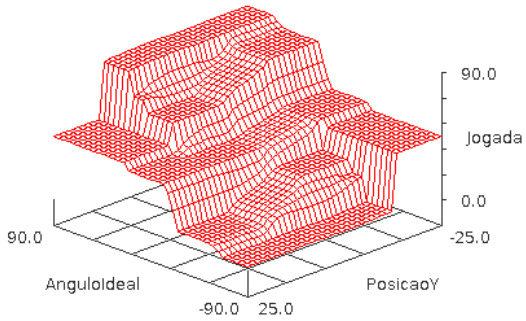


(c) Posição Fixa em -25

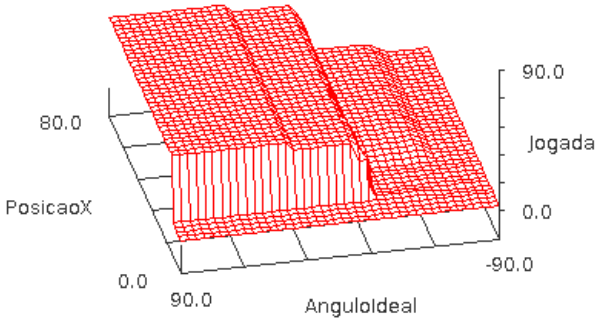
Figura C.2: Testes com Valores Fixos nos Extremos Inferiores de Cada Conjunto



(a) Ângulo Fixo em 90



(b) Distância Fixa em 80



(c) Posição Fixa em 25

Figura C.3: Testes com Valores Fixos nos Extremos Superiores de Cada Conjunto