UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Um Sistema Multiagente para o Simulador Soccerserver

por

DANIELA DUARTE DA SILVA BAGATINI

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de Mestre
em Ciência da Computação

Prof. Dr. Luis Otávio Campos Alvares
Orientador

Porto Alegre, abril de 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bagatini, Daniela Duarte da Silva

Um Sistema Multiagente para o Simulador Soccerserver / por Daniela Duarte da Silva Bagatini. – Porto Alegre: PPGC da UFRGS, 2001.

149p.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001. Orientador: Alvares, Luis Otávio Campos.

1.Inteligência Artificial Distribuída. 2.Sistemas Multiagentes. 3.Arquitetura de Agentes. 4.RoboCup. 5.Simulador Soccerserver. I.Alvares, Luis Otávio Campos. II. Título

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann
Superintendente de Pós-Graduação: Prof. Philippe Olivier Alexandre Navaux
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Agradecimentos

Agradeço a Deus a luz divina com a qual me guia, os benefícios que, se atribuísse ao acaso dos acontecimentos ou apenas ao meu próprio esforço, indigna seria. Mais importante que o lugar que Ele ocupa em mim, é a intensidade da Sua presença em tudo o que faço.

À minha família que me deu coragem e determinação para traçar o caminho em busca dos meus ideais; o abraço, o beijo e o futuro de esperanças em retribuição ao amor a mim dedicado.

Lembrar de todas as pessoas que, de certa forma (direta ou indiretamente), contribuíram para o andamento da dissertação, seria um desafio comprometedor. De qualquer maneira não posso omitir a contribuição valiosa dos amigos aos quais agradeço.

À colega e amiga Rejane Frozza retribuo em forma de mensagem o tudo de bom que recebi: "*não encontrei um só homem com o qual não pudesse aprender algo*". A amizade solidificada a cada dia fez-me mais confiante, mais forte.

Aos companheiros e amigos, Irineu, Denise, Ícaro e Paulo, por terem ajudado a colocar em prática as idéias discutidas e amadurecidas e aos demais amigos (Carlos, Débora, André, Paulo, Adriane, João Paulo, Eliane e Astrogildo), que auxiliaram de alguma forma na realização do presente trabalho, dedico o meu apreço.

À Professora Cláudia Moreira, pela revisão ortográfica completa e competente, o meu reconhecimento. Os erros eventualmente remanescentes são frutos da inquietante mania de mexer em textos prontos e, portanto, da minha inteira responsabilidade.

Ao meu orientador, Professor Doutor Luis Otávio Álvares, que tratou com cuidadoso respeito as limitações e inquietudes próprias dos que buscam mais conhecimento, a minha eterna gratidão. Não ter medido esforços para, com dedicação, orientar com clareza e firmeza, fez com que o caminho se tornasse mais límpido e estimulante.

À CAPES pelo apoio financeiro que possibilita os estudos tão necessários ao aprimoramento pessoal e profissional.

"Descobri como é bom chegar quando se tem paciência, e para chegar onde quer que seja, aprendi que não é preciso dominar a força. É preciso antes de mais nada, querer".

(Amyr Klink)

Sumário

Lista de Abreviaturas

IA	Inteligência Artificial
IAD	Inteligência Artificial Distribuída
SMA	Sistemas Multiagentes
RoboCup	*The Robot World Cup Initiative*
UFRGS	Universidade Federal do Rio Grande do Sul
FIFA	*Federation Internationale de Football Association*
IJCAI	*International Joint Conference on Artificial Intelligence*
MIKE	*Multi-agent Iteractions Knowledgeably Explained*
UDP/IP	*User Datagram Protocol/ Internet Protocol*
ASCII	*American Standard Code for Information Interchange*
UFSC	Universidade Federal de Santa Catarina
CMUnited	*Carnegie Mellon University*
AT Humboldt	*Humboldt-University of Berlin*
CBR	*Case Based Reasoning*
BDI	*Belief Desires Intentions*
SBSP	*Situation Based Strategic Positioning*
DPRE	*Dynamic Positioning and Role Exchange*
ISI	*University of Southern California's Information Sciences Institute*
ISIS	*ISI Synthetic*
RPAC	Pacote de recebimento (*receive packet*)
SPAC	Pacote de envio (*send packet*)
VC	Variável de condição
PI	Ponto de interseção
cs	Relógio interno do sistema

Lista de Figuras

Lista de Tabelas

Resumo

O interesse de pesquisa da comunidade de Inteligência Artificial em Sistemas Multiagentes tem gerado o crescimento da utilização de técnicas de agentes nas mais diversas áreas da ciência da computação. Isso ocorre, principalmente, devido à variedade de aplicações em que esses sistemas podem ser usados, como por exemplo: jogos de computadores, interfaces adaptativas, simulação e controle de processos industriais.

The Robot World Cup Initiative (*RoboCup*) é uma tentativa de estimular a área de Inteligência Artificial e, principalmente de Sistemas Multiagentes, por promover um problema padrão, jogar futebol, onde uma ampla cadeia de tecnologias podem ser integradas, examinadas e comparadas. A utilização do ambiente da RoboCup para a simulação de uma partida de futebol (simulador Soccerserver) permite a avaliação de diferentes técnicas de Sistemas Multiagentes (planejamento de estratégias, conhecimento em tempo real, colaboração de agentes, princípios de agentes autônomos, entre outros) e estimula as pesquisas, investigações e testes que possibilitem a construção gradativa de agentes avançados.

O presente trabalho tem por objetivo o desenvolvimento de um time de futebol para o simulador Soccerserver. A idéia principal é desenvolver agentes jogadores que demonstrem um nível considerável de competência para a realização de suas tarefas, como percepção, ação, cooperação, estratégias pré-definidas, decisão e previsão.

Inicialmente, apresenta-se uma visão geral sobre Inteligência Artificial Distribuída e sobre o simulador Soccerserver, pré-requisitos para o restante do trabalho. A seguir, é realizado um estudo sobre algumas arquiteturas de agentes (clientes) do Soccerserver. A arquitetura proposta na dissertação, suas principais características e a sua materialização em um protótipo desenvolvido correspondem à parte principal do trabalho. Finalmente são apresentados os testes realizados e as conclusões do trabalho.

Palavras-chave: Inteligência Artificial Distribuída, Sistemas Multiagentes, Arquitetura de Agentes, RoboCup, Simulador Soccerserver

TITLE: “A MULTI-AGENT SYSTEM FOR THE SOCCERSERVER SIMULATOR”

Abstract

The research efforts in Multi-Agent Systems (MAS) carried out by the Artificial Intelligence (AI) community have generated a growth in the use of techniques of agents in the most diverse areas of computer science. This occurs, mainly, due to the variety of applications in which these systems can be used, for example: computer games, adaptive interfaces, simulation and industrial process control.

The Robot World Cup Initiative (RoboCup) is an attempt to stimulate the area of AI and MAS, for promoting a standard problem, the soccer game, where a wide set of technologies can be integrated, examined and compared. The use of the RoboCup environment for the simulation of a soccer match (Soccer Server simulator) allows the evaluation of different techniques of MAS (planning of strategies, knowledge in real time, colaboration among agents, principles of autonomous agents, and others) and stimulates the research, inquiries and tests that make possible the gradual construction of advanced agents.

The purpose of the present work is to develop a soccer team for the Soccerserver Simulator. The main idea is to develop agent players that show a considerable level of competence in performing their tasks, for example perception, action, cooperation, strategy, decision and foresight.

Initially, we present a general outline about Distributed Artificial Intelligence and the Soccerserver Simulator, what provides the necessary background for the rest of the work. Next, we present a research on several agents architectures (clients) of the Soccerserver. The architecture proposed in this dissertation, its main characteristics and its materialization into a prototype, are the main part of the work. Finally, we conclude with the prototype test and the future works.

Keywords: Distributed Artificial Intelligence, Multi-Agent Systems, Agent Architecture, RoboCup, Soccerserver Simulator

1 Introdução

Atualmente, os paradigmas com foco no ambiente dos problemas e na forma como eles são tratados vêm ganhando destaque. Isso deve-se ao fato dos mesmos proporcionarem a aproximação entre os sistemas do mundo real e suas simulações no computador. Fala-se, hoje, na programação orientada a agentes, onde a idéia é construir sistemas de software constituídos de unidades autônomas com capacidade de representar conhecimento e estabelecer comunicações de natureza rica.

O diferencial do novo paradigma é fortemente apoiado na inteligência comunitária, ao contrário das abordagens anteriores da Inteligência Artificial (IA), onde o centro da modelagem era um único agente provido de alguma espécie de inteligência. O novo paradigma tem se mostrado bastante adequado à produção de software em geral, principalmente para atender necessidades do mundo real (como por exemplo, o mercado industrial).

A pesquisa sobre agentes é, no momento, um campo bastante popular. Grande parte desse estudo, entretanto, concentra-se na disciplina de IA, mais precisamente em Inteligência Artificial Distribuída (IAD), onde foram realizados estudos sobre o comportamento de agentes inteligentes, tanto tomados individualmente quanto em populações ou sociedades de vários agentes. O termo agente, entretanto, não possui uma definição unanimemente aceita, em parte, devido à multiplicidade de enfoques sobre os quais é estudado. Um agente pode desempenhar diversos papéis, o que torna muito difícil formular em poucas palavras uma definição para o mesmo.

Considera-se um agente como uma entidade individual, inserida em um meio ambiente, onde é capaz de agir com base em uma representação parcial deste ambiente [HUH 97].

A integração de vários agentes propicia um trabalho cooperativo na busca pela solução de um problema. Esses agentes interagem em um meio ambiente comum e apresentam comportamento autônomo. Eles são capazes de perceber, descobrir e agir, juntamente com outros agentes, para alcançar seus objetivos [ALV 99], [DEM 93] e [OLI 96]. De acordo com [PAL 96], "*o que distingue as arquiteturas multiagentes de outras arquiteturas é que elas oferecem soluções razoáveis para certa classe de problemas a um custo aceitável. Tais problemas seriam, por exemplo, do tipo que não pode ser solucionado com os recursos disponíveis em tempo hábil com sistemas monolíticos*".

Um objetivo da IAD, portanto, é construir sistemas capazes de solucionar problemas de forma cooperativa, com a utilização de Sistemas Multiagentes (SMA). O interesse pela tecnologia de agentes em áreas da Ciência da Computação, como na IA, tem refletido o potencial de suas aplicações. Há algum tempo esses sistemas são utilizados em uma variedade de ambientes, com resultados satisfatórios. Dentre as diversas aplicações

desenvolvidas a partir do enfoque de SMA, estão: cartografia [ALV 96], robótica [STE 94], Internet [CAZ 97], simulação [STE 94], entre outras.

A simulação do jogo de futebol constitui-se em um bom domínio para o estudo de SMA. Ela pode ser usada para avaliar diferentes técnicas de SMA, proporcionando um domínio complexo com características realísticas. Além disso, o futebol é bastante simples, possibilitando simular um jogo no computador com razoável aproximação [NOD 96a].

A iniciativa RoboCup, trata-se de um campeonato mundial de "futebol", onde diversos grupos de pesquisa, entre eles grupos de SMA, desenvolvem seu time de jogadores. Os jogadores estão inseridos em um campo de futebol, onde cada um pode perceber os estímulos de outros jogadores ou do próprio ambiente e agir sobre este ambiente.

O simulador Soccerserver é uma plataforma comum que permite a simulação da partida de futebol e a avaliação de diversas técnicas de SMA (explora idéias como: percepção, ação, cooperação, planejamento e decisão), estimulando a pesquisa e a investigação na construção gradativa de agentes avançados. O intuito é construir agentes que apresentem comportamentos inteligentes (chutar, interceptar, defender, etc.) e que estão inseridos em um ambiente dinâmico com características muito próximas da realidade (por exemplo: tempo real).

A RoboCup tem proporcionado resultados significativos na área de SMA. Atualmente, arquiteturas de agentes jogadores apresentam um alto nível de sofisticação e evolução [SCE 2000]. Por esses motivos, essa iniciativa, a cada ano, tem se tornado mais numerosa em termos de participantes e cada vez mais vem sendo difundida.

A maior motivação na realização deste trabalho, foi a possibilidade de desenvolver agentes distribuídos para um ambiente dinâmico que adiciona complexidade do mundo real em suas simulações. Este trabalho também proporcionou a integração de diversas tecnologias, tais como: estudo de SMA (combinando uma série de desafios que partem da arquitetura do agente), arquitetura cliente-servidor, decisão em tempo real.

As razões para a construção de tal time são: estudar o simulador Soccerserver (características, arquitetura e comandos), aplicar aspectos fundamentais de SMA (planejamento, organização, interação, entre outros) à arquitetura do agente, realizar a avaliação das técnicas empregadas e proporcionar a troca de idéias com os grupos de pesquisa da área de SMA.

Outro aspecto motivador é que a elaboração de uma arquitetura de um time de futebol para o simulador Soccerserver será um primeiro passo em direção ao desenvolvimento de um time a ser implementado na UFRGS. Ressalta-se o benefício do estudo para a pesquisa acadêmica sobre SMA, onde o cliente também proporciona um ambiente para a aplicação de SMA.

O objetivo desta dissertação é apresentar a arquitetura individual do agente UFRGS e, a partir dessa arquitetura, desenvolver um time de futebol para o simulador Soccerserver. Para atingir tais objetivos, foram traçadas algumas diretrizes básicas para o trabalho:

- O estudo de várias arquiteturas de agentes jogadores para o simulador, como as encontradas em [COS 99a], [STO 99a], [BUR 99], [REI 2001] e [TAM 99]. Tais arquiteturas foram analisadas, contribuindo para uma melhor definição das qualidades que um cliente deveria apresentar, facilitando o desenvolvimento dos jogadores.
- O estudo de SMA e do simulador Soccerserver.
- A proposta e a implementação de uma arquitetura de um agente jogador para o simulador, além da avaliação da mesma.

Foram feitas várias tentativas no sentido de utilizar outros times como ponto de partida. No entanto, encontraram-se dificuldades em trabalhar tanto com times mais simples, que não apresentaram boa documentação, quanto com times mais avançados, compostos por inúmeras linhas de código. A utilização de bibliotecas prontas, também não proporcionaria um conhecimento mais aprofundado da arquitetura base do time.

Por esses motivos, optou-se por desenvolver o agente UFRGS a partir do zero, desde a comunicação com o servidor até a especificação das ações do agente. A implementação de uma arquitetura simples permitiu a compreensão do funcionamento e das características determinantes do simulador Soccerserver. Com a evolução do trabalho, estabeleceu-se um conjunto de atividades para o agente que foi empregado na arquitetura desenvolvida.

Com isso, espera-se que o presente trabalho sirva como estímulo para aumentar as pesquisas no campo de SMA, bem como, ampliar o número de aplicações reais utilizando SMA. Espera-se também, que este trabalho propicie a continuação e o aprimoramento do agente UFRGS, adicionando ao mesmo características mais cognitivas, além de aprimorar as funções mais básicas, procurando eliminar a incerteza adicionada pelo simulador.

1.1 Organização do Texto

A seguir, apresenta-se uma breve descrição da organização deste trabalho, destacando os pontos importantes de cada capítulo.

No Capítulo 2, apresenta-se uma breve introdução à Inteligência Artificial Distribuída e aos Sistemas Multiagentes; destacando-se as arquiteturas de agentes e os cenários. Neste capítulo, tem-se uma visão geral da Arquitetura de Subsunção [BRO 86] e descreve-se o tópico Funcionalidade Emergente.

O Capítulo 3 aborda o estudo realizado sobre a RoboCup e o simulador Soccerserver. São apresentados os objetivos, os desafios, a origem e a análise da estrutura da Copa Mundial de Robôs. A arquitetura, protocolos e comandos de controle do simulador são descritos no decorrer deste capítulo.

O Capítulo 4 destaca os clientes (jogadores) desenvolvidos para o simulador, apresentando a arquitetura de quatro agentes, suas características e seus objetivos.

No Capítulo 5, descreve-se uma proposta de uma arquitetura cliente com base em componentes simples e técnicas de SMA. A arquitetura do agente individual foi definida com o objetivo de desenvolver agentes para o Soccerserver. Apresenta-se as características do agente UFRGS e a sua funcionalidade.

O Capítulo 6 apresenta detalhes de implementação do time desenvolvido e os resultados obtidos com a pesquisa realizada.

No Capítulo 7, encontram-se as conclusões e os trabalhos futuros, possibilitando a extensão do estudo realizado.

2 Inteligência Artificial Distribuída

Muitas das pesquisas realizadas pela IA procuram desenvolver sistemas inteligentes baseados no "comportamento individual humano", utilizando conceitos como: heurísticas e métodos baseados em conhecimento e planejamento, para alcançar a solução de um problema.

As primeiras idéias de distribuição em IA surgiram na década de 70. Segundo [BIT 98], "*estas idéias aliadas à maturidade alcançada pelas tecnologias de redes e de sistemas distribuídos, deram origem à área de IAD, onde se desenvolvem métodos de IA para a solução distribuída de problemas*".

Essa abordagem traduz o comportamento do sistema como algo essencialmente ligado ao coletivo. Há classes de problemas que são naturalmente distribuídos e usam domínios de conhecimento distintos para a sua resolução.

No decorrer deste capítulo, será exposta uma sucinta introdução aos assuntos IAD, Arquitetura de Subsunção e funcionalidade emergente. Essa introdução busca resgatar o domínio no qual o presente trabalho se insere, com o objetivo de proporcionar uma melhor compreensão durante o andamento do mesmo.

2.1 O Agente

A Inteligência Artificial Distribuída surgiu como uma sub-área da IA que, desde então, procura otimizar e solucionar problemas de concepção decentralizada, onde o processo de busca de solução de problema parte da individualidade para a coletividade.

Os problemas abrangidos pela IAD possuem características distribuídas e são solucionados com o auxílio de entidades independentes que, interagindo, procuram encontrar uma solução global para o problema. Essas entidades individuais que se comunicam em um ambiente distribuído, contribuindo para a solução de um problema, são chamadas de agentes (Figura 2.1).

Segundo [ALV 99] com base em [FER 91], "*um agente é uma entidade real ou virtual, imersa num ambiente sobre o qual é capaz de agir, que dispõe de uma capacidade de percepção e de representação parcial deste ambiente, que pode se comunicar com outros agentes e que possui um comportamento autônomo, conseqüência de suas observações, de seu conhecimento e das suas interações com os outros agentes*".

[STE 94] define agente como: "*um agente deve ser capaz de perceber as mudanças e agir em seu meio ambiente – por exemplo, mudar sua própria posição ou manipular objetos. Isto deve acontecer por mecanismos automáticos – que não requeiram*

a intervenção de outros agentes ou a intervenção humana para ser executado. Dessa forma, o agente deve ser capaz de decidir o que fazer e quando".

O termo agente, portanto, designa uma entidade autônoma que age em um meio ambiente de acordo com seus próprios objetivos e seu estado corrente de conhecimento [BOI 92].

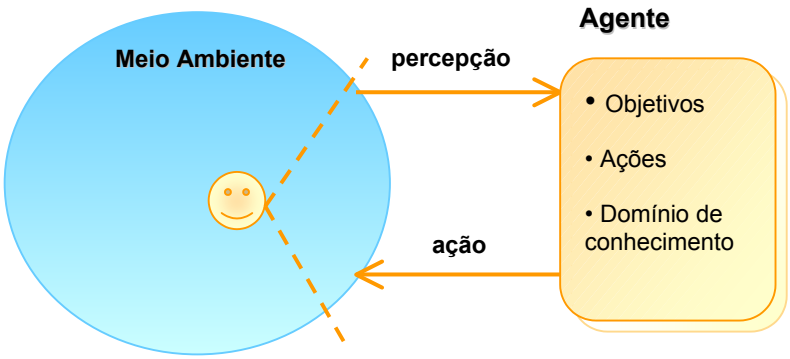


FIGURA 2.1 - Agente.

Os agentes são capazes de interagir de modo a cooperar para que uma determinada meta seja alcançada de maneira satisfatória, semelhante ao que ocorre com a própria sociedade humana, onde pessoas que compartilham um objetivo comum e um senso de equipe conseguem desempenhar seu trabalho mais depressa e facilmente.

Esses exemplos servem de inspiração para a IAD. O foco central da IAD está no comportamento social, ela parte do princípio de que um comportamento social inteligente pode derivar da interação entre os agentes membros de uma sociedade.

2.1.1 Sistemas Multiagentes

As pesquisas de SMA (Figura 2.2), abrangem o estudo de sistemas formados por várias entidades independentes, ou agentes. São agentes autônomos, com existência própria, que independem da existência de outros agentes e que podem ter, cada um deles, diferentes objetivos que no conjunto levam à solução de um problema.

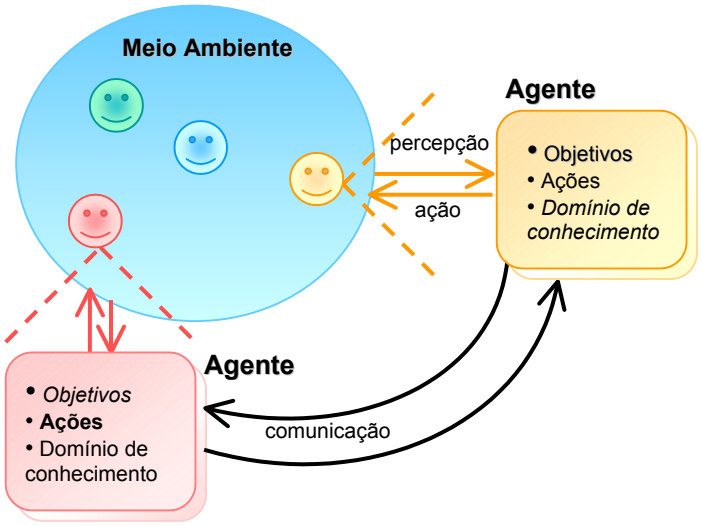


FIGURA 2.2 - Sistemas Multiagentes.

Os SMA podem ser utilizados em projetos de sistemas complexos, na busca de uma solução com o uso de vários agentes representados por seus objetivos e interesses. Por exemplo: um domínio no qual SMA podem ser utilizados é a agenda de horários de um hospital (como apresentado por Keith Decker (1996) em [STO 2000]). Diferentes pessoas concorrem por um horário, com diferentes critérios, elas devem ser representadas por agentes separados em função de seus interesses. O próprio futebol de robôs é um domínio rico indicado para o uso de SMA, pois pode avaliar diferentes técnicas de SMA de maneira direta, implementando agentes autônomos que podem jogar uns contra os outros e cooperar com seus companheiros de time.

A arquitetura do agente demonstra como ele está implementado de acordo com as suas propriedades, sua estrutura e como os módulos que o compõem podem interagir, garantindo a sua funcionalidade e as suas obrigações. Segundo [WOO 95], existem três tipos de arquiteturas de agentes: arquitetura cognitiva ou deliberativa, arquitetura reativa e arquitetura híbrida.

2.1.2 A Arquitetura de um Agente Cognitivo

As arquiteturas cognitivas (deliberativas) possuem um modelo simbólico do mundo (Figura 2.3, baseada em [RUS 95]). Os agentes podem conter uma representação explícita do ambiente e dos outros agentes que fazem parte do seu mundo. Conforme [WOO 95], as decisões sobre as ações a serem realizadas são tomadas através de raciocínio lógico baseado em reconhecimento de padrões, por exemplo.

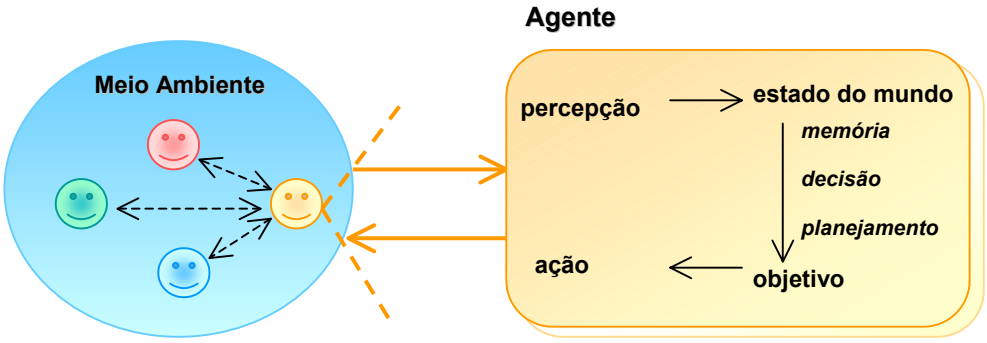


FIGURA 2.3 - Exemplo de Arquitetura de Agente Cognitivo.

Essas arquiteturas possuem por base as organizações sociais humanas e são caracterizadas pela complexidade dos seus agentes que dispõem de capacidade de raciocínio com base em sua representação do mundo (conhecimento). Os agentes cognitivos possuem objetivos e uma memória que permite planejar suas ações futuras e derivar as possíveis soluções para o problema a ser solucionado, além de se comunicarem através de troca de mensagens.

Em geral, esses agentes estão inseridos em sociedades que contêm um pequeno número de membros. São agentes com aptidão para tratar de informações diversas, como as interações com os outros agentes e com o ambiente [CAZ 97].

2.1.3 A Arquitetura de um Agente Reativo

As arquiteturas reativas não possuem nenhum tipo de modelo simbólico do mundo (Figura 2.4, baseada em [RUS 95]). Os agentes reagem às mudanças de seu meio ambiente sem utilizar raciocínio simbólico complexo [WOO 95]. São capazes de agir em seu meio ambiente com base nas suas percepções e nos seus objetivos de modo a satisfazê-los.

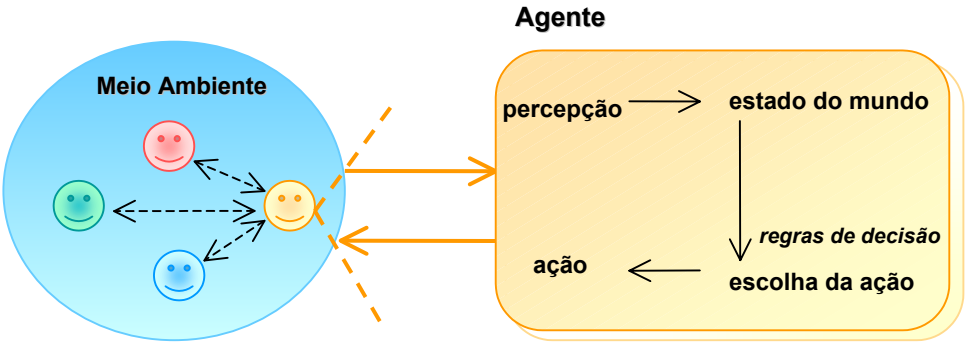


FIGURA 2.4 - Exemplo de Arquitetura de Agente Reativo.

Esses agentes autônomos possuem: arquitetura de percepção e ação, característica distribuída e decentralizada, interações dinâmicas com o meio ambiente, mecanismos intrínsecos e poder com pesquisas limitadas e informação incompleta. Eles não possuem memória de seus atos, não planejam suas ações futuras e não se comunicam diretamente com os outros agentes, cada agente toma conhecimento das ações dos outros agentes apenas pelas modificações do ambiente no qual está inserido.

Os agentes reativos buscam inspiração nos modelos de organizações biológicas e etológicas, como por exemplo: sociedades de formigas. Aparentemente esses pequenos seres não inspiram a convicção de que existe complexidade interna, no entanto, é visível e admirável a sua capacidade de organização. A inteligência nem sempre é manifestada por propriedades de um único indivíduo, a genialidade de um trabalho em conjunto pode ser visualizada em um time de futebol. A composição em grupo acaba exibindo uma estrutura sofisticada, que vai além das tarefas básicas de cada um.

As vantagens de um sistema reativo estão no fato do mesmo apresentar bom desempenho quando opera em domínios onde é difícil realizar uma modelagem completa e precisa. Esses sistemas também são eficientes e de resposta extremamente rápida, por perceberem o ambiente através de estímulo-resposta, dispensando qualquer tipo de modelagem, buscando suas ações diretamente nas percepções que têm do mundo [CAZ 97]. O custo computacional dessa arquitetura é baixo, pois somente há reação às ações que ocorrem no ambiente.

Um exemplo de arquitetura reativa é a “Arquitetura de Subsunção” de Rodney Brooks [BRO 86], onde existe uma hierarquia de níveis de comportamentos tarefa-ação, em que cada comportamento compete com outros para ter controle sobre o agente. Essa arquitetura surgiu da necessidade de um mecanismo para o controle de robôs móveis e autônomos.

2.1.4 A Arquitetura de um Agente Híbrido

Nas arquiteturas híbridas, os agentes possuem características reativas com relação aos eventos que ocorrem no ambiente e características deliberativas com a definição simbólica do mundo para a tomada de decisões. Existe a mistura de componentes das arquiteturas deliberativas e reativas, com o objetivo de tornar a arquitetura híbrida adequada e funcional para a construção de agentes.

O desafio que a arquitetura de um agente deve superar é o de organizar o raciocínio e a percepção do agente de maneira adequada. Por exemplo, é desejável ao agente manter a racionalidade sobre eventos futuros, enquanto existe um problema no meio ambiente, ou ainda, deve-se evitar que percepções contínuas interrompam o raciocínio do agente, do contrário o mesmo poderia não ser capaz de realizar cadeias complexas de raciocínio.

A cooperação entre os agentes é um fator significativo para o bom desempenho de suas ações no ambiente. A cooperação diz respeito à maneira com que os agentes estão organizados, para que o esforço individual (tanto na formação do plano, quanto na sua

execução) se reflita no objetivo comum do sistema. Em outras palavras, mostra como as tarefas executadas individualmente por cada agente podem ser coordenadas para obter uma funcionalidade eficiente de todo conjunto. A distribuição de tarefas e a implementação paralela são vistas como um ingrediente que proporciona comportamentos rápidos e robustos.

Um agente pode ser visto como uma coleção de comportamentos com sua própria competência específica. A especificação do comportamento do agente representa o processo de ação do agente no sentido de atingir uma situação final, como a solução do problema. A especificação do comportamento do agente só não esclarece a funcionalidade que é exposta quando o agente está em atuação. Este funcionamento é altamente dependente do dinamismo do meio ambiente e essas características são exploradas para servir ao bom desempenho do sistema.

A seguir, comenta-se uma série de características encontradas em agentes, buscando produzir uma idéia geral de um conceito que os identifique. Tais características agrupam-se em torno de duas noções, respectivamente denominadas *fraca* e *forte* (conforme Woodridge (1995) apresentado em [PAL 96]).

2.1.5 A Noção Fraca de Agente

Na noção fraca, o termo agente é empregado em sua forma mais geral para denotar uma entidade baseada em *hardware* ou (mais freqüentemente) em *software*, com as seguintes propriedades:

- Autonomia: o agente possui controle de suas ações e de seu estado interno, independente da existência de outros agentes ou da intervenção humana.
- Percepção: o agente deve ser capaz de perceber as mudanças do estado do mundo no qual está inserido, bem como, responder a essas transformações.
- Atuação: o agente deve ter a capacidade de agir em seu meio ambiente de forma a alcançar a solução de problemas e atingir sua meta.
- Meio ambiente: um agente deve estar inserido em um ambiente. Deve ser capaz de agir e interagir com o mesmo e com os outros membros da sociedade.
- Comunicação: um agente deve ser capaz de comunicar-se com os outros agentes e, assim, realizar a troca de conhecimento. A comunicação pode acontecer de maneira não-intencional, por exemplo: através do ambiente. Essa mensagem pode ser assimilada pelos agentes da comunidade por meio de suas percepções aos estímulos do mundo.

2.1.6 A Noção Forte de Agente

Segundo [PAL 96], na noção forte, um agente é uma entidade que, além das propriedades apresentadas, emprega conceitos mais usualmente aplicados a seres humanos

e caracterizados por estados mentais, tais como: crença, intenção e compromisso. Em geral, os agentes que se enquadram nesta noção mais forte possuem uma ou mais das seguintes propriedades:

- Comportamento social: um agente pode raciocinar sobre a atuação dos demais membros da comunidade e interagir com os mesmos e com o ambiente no qual se encontram, com o objetivo de cooperar e de realizar suas tarefas.
- Racionalidade: dada uma seqüência perceptiva, o agente escolhe, segundo seus conhecimentos, as ações que satisfazem melhor o seu objetivo.
- Adaptabilidade: um agente deve ser capaz de adaptar-se ao seu meio ambiente.
- Benevolência: a idéia é que cada agente irá sempre tentar fazer o que lhe for pedido, ajudando outros agentes a atingirem suas metas.

2.1.7 O Funcionamento do Agente

Basicamente o agente exibe o seguinte funcionamento, segundo [PEB 95]:

A entrada (percepção) de um agente é originada por um sensor ou talvez pelo seu próprio estado interno. A entrada pode ser considerada como o estado do mundo do agente e, como tal, é o conjunto de informação do agente. O agente utiliza a sua própria percepção do mundo para realizar a tomada de decisão e, então, seleciona a ação a ser executada. A memória ou o conhecimento do agente também pode ser usado como uma entrada.

A saída (ação) do agente pode ser considerada como a sua atuação ou o seu modo de agir no mundo. Em algumas circunstâncias, a saída do agente também proporciona uma entrada para o mesmo agente.

O mecanismo interno de um agente transforma a entrada em uma saída apropriada. O mecanismo interno deve permitir que o agente trabalhe seguramente na realização de sua tarefa corrente. O processo interno pode incorporar a variação de memória, que pode ser usada para aumentar a entrada sensorial com as experiências prévias do agente.

2.1.8 As Vantagens na Utilização da IAD

As vantagens na utilização da IAD nas soluções de problemas distribuídos são as seguintes [BON 88]:

- Adaptabilidade: sistemas de IAD são mais apropriados para lidar com problemas de controle decentralizado, visto a sua característica distribuída.
- Custo: um grande número de pequenas unidades computacionais pode ser mais interessante, quando os custos em comunicação não são relevantes.
- Desenvolvimento e Gerenciamento: a inerente modularidade do sistema em IAD permite o desenvolvimento das partes de maneira distribuída e independente. A própria atividade de decompor um problema em problemas

mais simples ocasiona uma simplificação da tarefa e permite uma maior facilidade de compreensão do mesmo [CAZ 97].

- Eficiência e Velocidade: a concorrência e a distribuição dos processos em diferentes máquinas pode aumentar a velocidade de computação e raciocínio.
- Confiabilidade: os sistemas distribuídos podem exibir maior confiabilidade e controle de segurança quando comparados aos sistemas centralizados, pois podem prover redundâncias de dados e múltiplas verificações. Se um problema é distribuído entre sistemas diferentes, a busca pela solução pode continuar mesmo que um dos sistemas apresente falhas [RIC 93].
- Limitações de recursos: os agentes computacionais individuais ligados a recursos escassos podem, através da cooperação, superar problemas complexos.
- Especialização: pode-se especializar cada agente de acordo com seu domínio de conhecimento e aplicação.
- Problemas distribuídos: alguns problemas são melhores descritos como uma coleção de agentes separados; existe uma melhor forma de representação para o problema ou domínio, porque os elementos são naturalmente distribuídos.

2.1.9 Os Cenários dos SMA

Os SMA diferem de sistemas de um único agente de forma significativa. Por exemplo, em um SMA, o meio ambiente dinâmico pode ser determinado por outros agentes. Incertezas podem ser adicionadas ao domínio, ou seja, outros agentes podem intencionalmente afetar o meio ambiente de maneiras imprevisíveis.

Como entidade independente que é, cada agente, inserido no meio ambiente, pode ser modelado de maneira diferente. Podem existir agentes com grau de heterogeneidade diversificado, com ou sem habilidade de comunicação direta. Cada agente pode ainda possuir seus objetivos, suas percepções, seus conhecimentos, suas ações e seu domínio. Esses agentes também podem interagir diretamente.

A seguir são apresentados, de forma concisa, alguns cenários de SMA [STO 2000a]:

- Homogêneos não-Comunicantes (*Homogeneous Non-communicating Agents*). Agentes homogêneos não-comunicantes são agentes autônomos que apresentam a mesma estrutura interna, incluindo objetivos, domínios de conhecimento, ações possíveis e o mesmo procedimento de seleção entre suas ações. A única diferença entre os agentes são as suas percepções e conseqüentemente a ação que eles escolhem. Nesse cenário, assume-se que os agentes não podem se comunicar diretamente. A Figura 2.5 ilustra um cenário multiagente não-comunicante indicando que os objetivos do agente, as ações, e o domínio de conhecimento são os mesmos, representado por fontes de letras idênticas.

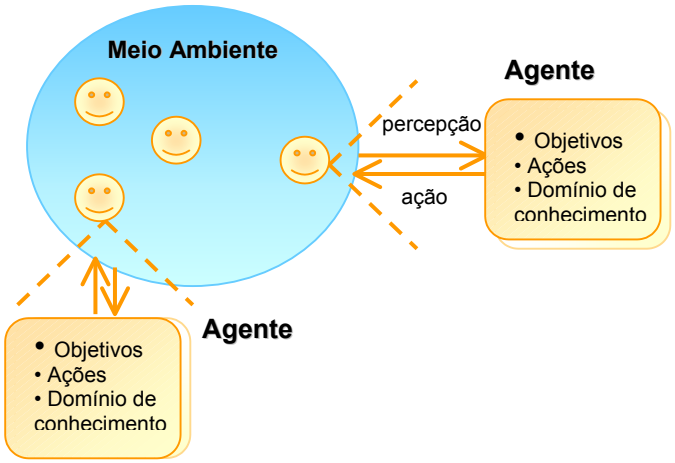


FIGURA 2.5 - Agentes Homogêneos não-Comunicantes.

- Heterogêneos não-Comunicantes (*Heterogeneous Non-communicating Agents*). Os agentes podem ser heterogêneos, por terem objetivos, modelos e ações distintas. Possuindo diferentes objetivos, esses agentes podem cooperar para a realização do objetivo do outro ou, ao contrário, eles podem ativamente tentar inibir o outro. Como os agentes homogêneos, esses estão situados de forma diferente no meio ambiente, o que indica que eles estão recebendo diferentes percepções de entrada e suas necessidades correspondem a ações diversas. Entretanto, nesse cenário, os agentes podem ter objetivos, ações e/ou domínio de conhecimento distintos, como indicado pelas diferentes fontes de letras na Figura 2.6.

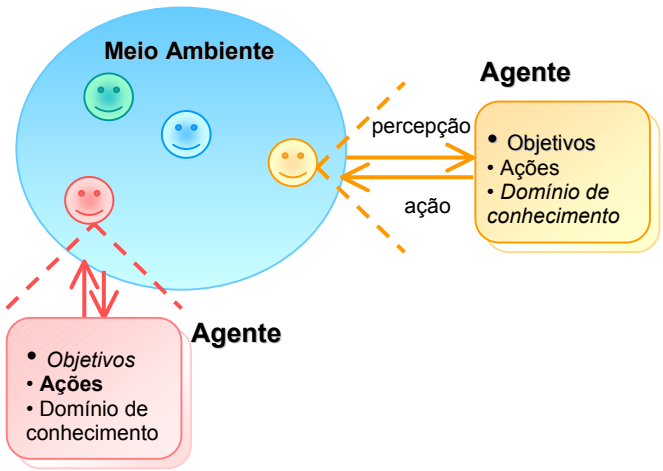


FIGURA 2.6 - Agentes Heterogêneos não-Comunicantes.

- Homogêneos Comunicantes (*Homogeneous Communicating Agents*). Cada agente pode comunicar-se diretamente com os outros agentes. Com o auxílio da comunicação, os agentes podem coordenar-se de forma mais efetiva. Como no caso homogêneo não-comunicante, os agentes são idênticos e eles estão situados no meio ambiente de forma diferente. Entretanto, nesse cenário, os agentes podem realizar comunicação direta, como indicado pela Figura 2.7. A comunicação também pode ser visualizada como parte da interação do agente com o meio ambiente.

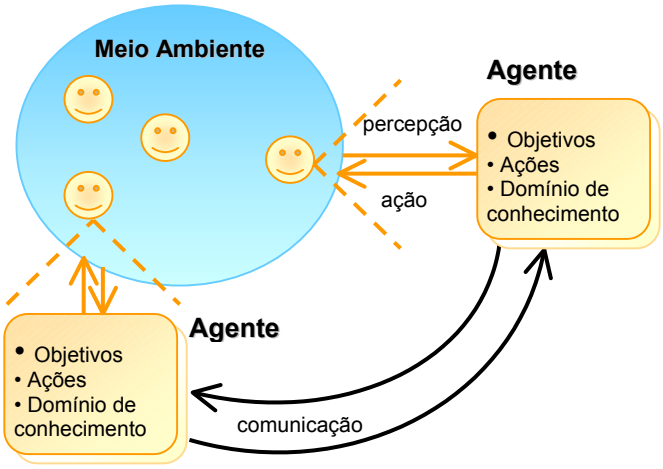


FIGURA 2.7 - Agentes Homogêneos Comunicantes.

- Heterogêneos Comunicantes (*Heterogeneous Communicating Agents*). Cada agente heterogêneo pode ser muito complexo e poderoso, entretanto, maior poder pode ser obtido quando adiciona-se habilidades de agentes heterogêneos que se comunicam. Esses agentes podem trocar mensagens e, desta forma, planejar suas ações. Esse cenário multiagente é apresentado na Figura 2.8.

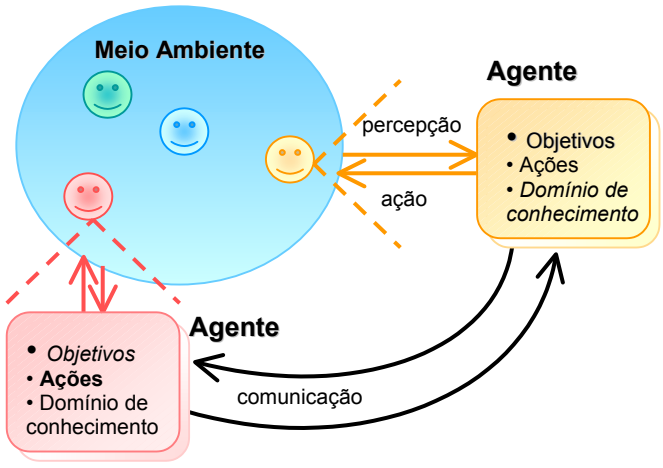


FIGURA 2.8 - Agentes Heterogêneos Comunicantes.

2.2. Arquitetura de Subsunção

Em [BRO 86], foi proposta uma arquitetura reativa para a construção de agentes autônomos, chamada de Arquitetura de Subsunção. Trata-se de uma abordagem também conhecida como baseada em comportamento, pelo fato da realização das tarefas basearem-se na pré-especificação dos comportamentos, ou seja, cada tarefa é responsável por um comportamento específico do agente, como por exemplo: desviar_de_obstáculos, alcançar_o_objeto ou contornar_obstáculo.

As experiências realizadas com a Arquitetura de Subsunção (como podem ser observados em [STE 94]), apresentaram sucesso em ambientes desconhecidos, embora dependentes das estruturas de conhecimento pré-definidas e programadas em cada módulo. Essa arquitetura tem sido extensivamente utilizada. Tal arquitetura apresenta um novo movimento para o estudo da inteligência de maneira "*bottom up*", concentrando-se nos sistemas físicos (com inspiração na engenharia e nos sistemas biológicos) situados no mundo, que executam de maneira autônoma suas tarefas. Por essa razão, é projetada para atuar em ambientes reais, dinâmicos, onde os agentes atendam a várias tarefas com diferentes níveis de urgência.

Conforme Noel Rode (1994) apresentado por [ARN 98], dentre as vantagens da Arquitetura de Subsunção destacam-se:

- Tem princípios e procedimentos bem definidos para o desenvolvimento de sistemas baseados em comportamento [BRO 91a].
- Existem muitas implementações documentadas que obtiveram sucesso [MAE 90].
- Seu enfoque incremental, *bottom-up*, necessita que qualquer solução desenvolvida seja funcional no mundo real e com dados reais; com o mínimo de representação possível [BRO 91a].

O enfoque utilizado por Rodney Brooks foi o de construir incrementalmente um sistema de controle por níveis de abstração (camadas) [BRO 93]. Cada atividade ou comportamento individual tem origem e resultado nas percepções e ações do agente, respectivamente. O termo atividade é usado devido às camadas (sistema de atividades) decidirem quando agir por elas mesmas e não através da chamada de uma subrotina de alguma outra camada.

O sistema de controle prevê, primeiramente, um nível baixo de competência do agente. Outros níveis de competência podem ser adicionados incrementalmente acima do sistema de controle (de forma evolucionária), sendo capaz de influenciar nas regras dos níveis mais baixos, sem modificar esses níveis, fazendo com que o agente tenha um desempenho inteligente e competente.

O nível de competência é uma especificação informal de um conjunto de comportamentos do sistema sobre o ambiente no qual ele se encontra. Um alto nível de competência implica em um conjunto de comportamentos mais específicos, provendo restrições adicionais em níveis mais baixos.

Conforme o que foi relatado, o problema é decomposto em diferentes níveis, ou seja, o agente possui por tarefa a satisfação de seus sub-objetivos. O uso de regras locais pode proporcionar aos agentes a alcançar os seus sub-objetivos e reduzir a complexidade da tarefa.

Cada comportamento tem uma tarefa associada, as de baixo nível são básicas, enquanto que as de alto nível são capazes de suprimir a ação dos níveis inferiores. Vários sub-objetivos podem coexistir, no entanto, existem diferentes prioridades a cada nível de comportamento, indicando a tarefa a ser executada pelo agente. Os vários níveis individuais de comportamento extraem da percepção do agente somente aqueles aspectos que são relevantes para si, onde cada comportamento exibe a conclusão de uma tarefa com base nos aspectos que emergem do comportamento do agente como um todo.

Uma decomposição do processamento da informação é a proposta da Arquitetura de Subsunção (Figura 2.9) [COE 99]. Ela permite que um nível de comportamento continue executando sem ser influenciado pela mudança de outros ou pela adição de novos comportamentos. Portanto, é construído inicialmente um sistema de controle por níveis [ARN 98]. Cada nível trabalha com as percepções e ações do agente. À cada etapa, somente é necessário construir um comportamento e integrá-lo com os já existentes. A idéia é, primeiramente, construir um sistema autônomo, completo e simples e testá-lo em um mundo real. Depois, deve-se construir um comportamento adicional que opere com o sistema anterior, e novamente testá-lo no mundo real, de forma que o primeiro nível do sistema continue a ser executado sem perceber a existência do segundo nível.

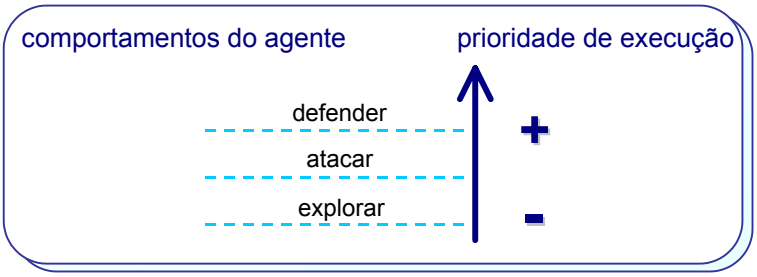


FIGURA 2.9 - Exemplo de Arquitetura de Subsunção.

O paradigma, baseado em comportamento, vê a inteligência em termos de comportamentos simples e coerentes. Essa inteligência aparece em conseqüência de eventos ou oportunidades no ambiente. Cada nível funciona de forma independente, cada um com seu próprio sub-objetivo. O sistema de controle pode ser naturalmente oportunista, quando as circunstâncias propícias são apresentadas e poderá também responder satisfatoriamente às mudanças do estado do mundo [BRO 91b]. A funcionalidade do sistema, como percepção, planejamento, modelagem, aprendizado, e outras, emerge da interação dos componentes do sistema.

Conforme [ARN 98], isso traduz a maioria das coisas que se percebe como comportamento inteligente complexo. Esses comportamentos emergem das interações entre comportamentos mais simples que não precisam ser explicitamente planejados, representados ou reconhecidos pelos outros agentes. A organização da inteligência trata-se

de utilizar a inteligência reativa para aspectos dinâmicos do ambiente, em resposta às condições do mesmo.

A Arquitetura de Subsunção proporciona toda uma funcionalidade com base em níveis de formas de vida mais simples, como insetos, por exemplo. De acordo com [BRO 91a], quando se examinam os níveis muito simples de inteligência, não se encontram representações explícitas e modelos completos do mundo. Isso significa que é melhor utilizar o mundo como próprio modelo do agente.

Segundo H. A. Simon (1969) citado por [BRO 91b], a complexidade do comportamento de uma formiga, por exemplo, é maior que a complexidade de seu meio ambiente e de sua complexidade interna. Brooks também relata que muitas das atitudes e ações humanas são tomadas sem que um complexo raciocínio seja feito. Segundo ele, "muitas das coisas feitas em nosso dia não são planejadas, são atividades rotineiras numa relação dinâmica com o mundo".

A Arquitetura de Subsunção proporciona o aparecimento de comportamentos mais sofisticados através da combinação dos comportamentos simples. As capacidades do agente manifestam-se da combinação dos comportamentos, da percepção e da mobilidade do agente em um meio ambiente dinâmico. Isso também permite que o agente possa evoluir a partir de suas necessidades básicas.

A preocupação de Brooks estava no fato de se perder grande parte do tempo de processamento no planejamento e na ação, procurando tornar a operação dos agentes mais rápida, com pouca computação na percepção e na construção do modelo do mundo.

A arquitetura em questão é aplicável a todo sistema que tenha características do tipo [ARN 98]:

- Precisa ser operado em tempo real;
- Precisa atingir múltiplos objetivos simultaneamente;
- Explora uma arquitetura paralela;
- Precisa estar informado dos eventos externos a ele, em tempo real;
- Precisa processar dados de sensores;
- Existem condições, envolvendo detalhes não importantes para os objetivos do sistema central, que precisam ser encontradas de forma que os objetivos sejam alcançados.

A Arquitetura de Subsunção apresenta características como:

- Não existe representação explícita: Podem haver representações parciais do modelo do mundo, extraindo-se os aspectos relevantes, ou seja, o próprio mundo serve como modelo para o agente. Assim, evitam-se representações incorretas do ambiente dinâmico.
- Não existe raciocínio explícito: O raciocínio deve ser emergente da combinação dos diferentes comportamentos do agente.

- Inteligência: Interpretada como o produto dos comportamentos associados à complexidade do ambiente.
- Adaptação: Capaz de responder às mudanças do mundo e, desta forma, controlar as circunstâncias de maneira naturalmente oportuna.
- Robustez: Lida com as inconsistências do mundo e incertezas provenientes das entradas, ou seja, saber lidar com um ambiente imprevisível, atendendo às necessidades com vários níveis de urgência.
- Extensibilidade: Proporciona que novos comportamentos possam ser incorporados ao agente, sem prejudicar o conjunto de comportamentos existentes.
- Múltiplos sub-objetivos: Cada comportamento pode trabalhar com sub-objetivos individuais e concorrentes.

2.3. Funcionalidade Emergente

Um agente individual não precisa necessariamente executar um comportamento complexo para desempenhar um trabalho global inteligente [FRO 97]. Na natureza é fácil perceber que espécies que vivem juntas, e não parecem ser muito inteligentes, com poucas capacidades, podem ilustrar um comportamento bem estruturado, podendo até mesmo aparentarem ser o mesmo organismo. A idéia que a funcionalidade emergente expressa é que a soma dos comportamentos elementares de um agente pode exibir um comportamento global mais organizado do que cada comportamento individual.

O comportamento emergente trata-se de um fato que vai além das expectativas da organização dos sistemas baseados em comportamento. É por causa das ações simultâneas (mudanças rápidas) e da dinâmica individual de interação com o mundo que o comportamento total do sistema é capaz de apresentar uma funcionalidade emergente.

O trabalho realizado por [WAV 92] é baseado em representações explícitas de comportamento, onde o agente por si só pode produzir um comportamento robusto e eficiente. Segundo Wavish, a atividade de um agente é produzida, primeiramente, pela inter-relação do mesmo e o seu meio ambiente. Quando o agente está interagindo com o meio ambiente, seu comportamento cresce e, conseqüentemente, o comportamento do sistema do qual ele faz parte, emerge das interações dos componentes representados por comportamentos explícitos.

Esse comportamento emergente não é representado diretamente, ou pelo menos, não é representado por uma função simples de comportamento do sistema. A inteligência do sistema emerge da interação dos agentes.

Muitos estudos têm sido feitos sobre o trabalho emergente baseado no tema de auto-organização de uma sociedade. Em estudos baseados em comportamento, [STE 90] define: "*um objetivo é alcançado imediatamente por interações de componentes mais primitivos, entre eles mesmos e com o mundo*".

Em [BRO 89], é apresentado um exemplo de comportamento emergente de um robô de seis pernas. O comportamento “caminhando” não é explicitamente representado, mas é perceptível ao observador esse comportamento de caminhada.

O conceito de emergente em si não oferece orientações de como construir cada sistema, nem dá idéia como poderia funcionar. A única certeza está no fato da funcionalidade emergente manifestar a passagem da ação individual do agente reativo para uma ação coletiva.

3 Os Sistemas Multiagentes e a RoboCup

De acordo com [TAM 99], cada vez mais SMA são designados para uma variedade de aplicações em domínios complexos e dinâmicos. Em cada domínio, as interações entre agentes têm crescido, tornando-se um desafio para a pesquisa de SMA. Vários são os exemplos utilizados para avaliar o desempenho desses sistemas, mas cada exemplo é simples e pequeno. A iniciativa RoboCup propõe a simulação de um jogo de futebol. Trata-se de um problema complexo e grande, conveniente à avaliação de SMA.

Reconhecida como um desafio do ponto de vista de SMA, a RoboCup visa a estimular a pesquisa na construção de agentes avançados que possam apresentar comportamento inteligente. Além disso, o futebol requer técnicas e estratégias que podem ser usadas adequadamente em SMA (habilidades do agente, cooperação, entre outras).

Com o objetivo de organizar o conhecimento sobre a Copa Mundial de Robôs, neste capítulo, será exposta uma introdução a RoboCup e ao simulador Soccerserver.

3.1 Um Problema Padrão

A RoboCup é uma tentativa de estimular uma extensa variedade de pesquisas e produzir avanços rápidos em tecnologias chaves nas áreas de robótica e SMA [ROB 2001]. Ela proporciona uma tarefa comum para a avaliação de várias teorias, algoritmos e arquiteturas de agentes. Também, oferece a possibilidade de confronto e intercâmbio de idéias em um domínio comum, além de estimular os grupos de pesquisas a estabelecerem e manterem contatos entre si [KIT 97a].

Essa iniciativa tem por finalidade usar o jogo de futebol (*soccer*) como uma plataforma para pesquisas, cujos projetos principais baseiam-se em agentes autônomos [MAE 90], colaboração em sistemas multiagentes, aquisição de estratégias de pesquisa em tempo real, entre outros.

De acordo com [KIT 97b], um “problema padrão” tem sido a força básica para pesquisas em IA. O computador que joga xadrez é um típico exemplo de problema padrão [RUS 95]. Esse tipo de pesquisa permitiu a descoberta e o progresso de várias tecnologias. Mas, as críticas atribuídas, freqüentemente focalizam o fato de que esse estudo é uma tarefa abstrata e ignora dificuldades do mundo real.

[ASA 99] reforça: "*necessitamos de um novo desafio*, *esse desafio precisa fomentar um conjunto de tecnologias para a próxima geração de indústrias. Nós achamos que a RoboCup cumpre a demanda ilustrada na Tabela 3.1, a qual diferencia as características de domínio do computador que joga xadrez e a RoboCup.*"

TABELA 3.1 - Comparação entre o jogo de xadrez e a RoboCup [ASA 99].

Características	***Xadrez***	***RoboCup***
Meio ambiente	estático	dinâmico
Mudança de estado	turnos de tempos	tempo real
Acessibilidade de informações	completo	incompleto
Sensores de leitura	simbólico	não-simbólico
Controle	central	distribuído

Segundo [KIT 97b], a comunidade científica, em geral, discute os problemas, centralizados em pesquisas sérias (como o próprio jogo de xadrez), mas inerentes a domínios específicos que não são normalmente necessários em outros domínios. Já as pesquisas que usam sistemas do mundo real, são potenciais e fundamentais na comparação de tecnologias para a realização de tarefas reais.

Sendo assim, a RoboCup reúne a necessidade de lidar com a complexidade do mundo real (embora um mundo limitado) como um fornecedor de problema considerável e de pesquisas sofisticadas (por exemplo: cognição). Oferece tarefas de pesquisas integradas, abrangendo a ampla área da IA e da robótica, incluindo pesquisas como comportamento reativo e cognitivo, aquisição de estratégias, aprendizado, planejamento em tempo real, sistemas multiagentes, visão, decisões estratégicas, controle motor, controle de robôs inteligentes, e muitas outras [KIT 97c].

A RoboCup destina-se a criar robôs (físicos ou simulados) que demonstrem um alto nível de competência para a realização de tarefas, como chutar, interceptar, entre outras, além de colaborarem em um problema dinâmico. Um time poderá produzir um plano estratégico e executá-lo de maneira coordenada, monitorando e selecionando as ações apropriadas, de forma a colaborar na resolução do problema.

A maior característica da RoboCup está no fato de promover pesquisas com a oportunidade de demonstrar os resultados da mesmas, na forma de competição em um ambiente dinâmico.

3.2 A Pesquisa em Questão

Segundo [NOD 96a], "*A Copa de Robôs (RoboCup) procura fomentar a IA e as pesquisas de SMA e robôs inteligentes, promovendo um problema padrão onde várias tecnologias podem ser integradas e examinadas.*"

O futebol tem sido usado em uma variedade de exemplos para testar e demonstrar o desempenho dos diversos tipos de sistemas de IA. A razão dessa tendência, é que o futebol inclui problemas característicos abordados pela IA, como o processamento em

tempo real, a robustez contra ruído, a cooperação em SMA e o processamento de informações incompletas. Ele é usado para avaliar diversas técnicas de SMA de maneira direta: implementando times que podem jogar um contra o outro.

Do ponto de vista dos SMA, o futebol produz um bom exemplo de problema do mundo real, possibilitando simular um jogo com razoável aproximação. Por outro lado, os SMA proporcionam a distribuição de controle e a efetiva comunicação. As características abaixo demonstram que o futebol é um domínio apropriado para avaliação de SMA [MAT 96]:

- Complexidade bastante realística, como percepção limitada.
- Possibilidade direta de comparação.
- Robustez: um time pode ter uma estratégia de falha, caso ocorra algum acidente (planos de jogo e estratégias podem falhar quando uma mensagem sensorial é perdida).
- Adaptabilidade: é requerida para modificações dinâmicas de planos, de acordo com o comportamento do time oponente.
- Tempo real: uma partida não pode ser interrompida. Cada jogador é requerido para tarefas em tempo real.
- Eficiência de cooperação: a comunicação não é precisa. Efetivamente, devem ser providenciadas simples combinações de sinais entre jogadores.

O futebol requer agentes separados para controlar os jogadores individualmente. Nesse domínio de pesquisa, pode se simplificar a programação dos agentes dividindo as tarefas. Cada jogador tem a sua própria entrada sensorial, podendo ter, cada um, uma tarefa específica no campo. Um jogador pode estar em posição para defender, enquanto outro está preparando um ataque [STO 99b].

Por cada agente possuir o seu próprio sistema de visão, o problema torna-se mais interessante, adicionando novos desafios; uma vez que cada jogador pode perceber somente parte do meio ambiente. Dessa forma, alguma descoberta global sobre o meio ambiente deve ser construída a partir de uma visão local. A cooperação entre a comunidade de agentes pode proporcionar a construção de um conhecimento social.

3.2.1 Os Objetivos

Uma das características da RoboCup está no fato de ser explicitamente a primeira pesquisa em que o objetivo final é de um time campeão da copa de humanos. Segundo [ASA 99], chama-se esse tipo de objetivo de *grand challenge project*. A busca por esse objetivo pode ser considerada como a maior realização no campo da robótica.

De acordo com [ASA 99], no caso da RoboCup, o objetivo final é: "*Por volta do século 21, um time de robôs humanóides autônomos deve enfrentar o time campeão da copa do mundo conforme as regulamentações da FIFA.*"

Embora a meta seja a copa do mundo com robôs reais (um time de humanóides), a RoboCup proporciona uma plataforma comum para pesquisa em aspectos de *software*. Até então, o futebol não era tratado como um problema comum, pois existiam plataformas não comuns, com cenários distintos, que dificultavam a comparação do desempenho de cada sistema que usava o futebol como um exemplo.

Dentre um conjunto de pequenos objetivos, busca-se avaliar os méritos e não méritos das propostas de pesquisa usando SMA (objetivos, planos, conhecimentos, capacidades, emoções e outros) e robótica. Ao mesmo tempo, procura-se manter a generalização das arquiteturas para diversas outras aplicações.

O benefício de um projeto como a RoboCup está em, gradualmente, aprimorar a ciência, a engenharia e a tecnologia, proporcionando um objetivo principal a ser atingido. Algumas das possibilidades da utilização das tecnologias associadas com a RoboCup está em [NOD 95]:

- Robôs para desenvolver tarefas perigosas: limpar terrenos radioativos, dispositivos de bomba e minas, realizar tarefas de baixo d'água e explorar lugares.
- Robôs de pesquisa e resgate: resgate a sobreviventes de tremor, resgate de nadadores em perigo, resgate em minas, regaste em avalanches e outros.
- Futuros sistemas de transporte.
- Robôs de brinquedo: para entretenimento e educação.
- Robôs para socorrer idosos e deficientes.
- Robôs para trabalhar na indústria.
- Baterias de alta eficiência e uso de energia.
- Novos materiais e aparelhos.
- Evolução de Software.
- Vida Artificial.

3.2.2 Os Desafios

Vários são os desafios proporcionados pela RoboCup, iniciando na arquitetura do agente, habilidades individuais, controle autônomo e, dentre outros, o trabalho em grupo.

O trabalho desenvolvido por um conjunto de agentes é bastante complexo em domínios dinâmicos como o futebol. Requer alta flexibilidade de coordenação e comunicação para superar as incertezas, desafios, cumprir responsabilidades e assimilar inesperada descoberta de oportunidades. É difícil antecipar e pré-planejar todas as características de coordenação possíveis. Além disso, o planejamento de coordenação em domínios com tantas possibilidades de ação é uma tarefa complicada. O dinamismo do domínio, causado pelas ações não previstas do oponente, torna a situação consideravelmente mais complexa.

Uma das maiores razões pelas quais a RoboCup atrai tantas pesquisas, está no fato da mesma permitir a integração de várias tecnologias dentro de um time de agentes. Além de adicionar aspectos do mundo real.

3.2.3 A Origem

Em 1988 era proposta a primeira pesquisa relacionada ao futebol (por Elisabeth Andre, de acordo com [STO 2000a]). Foi desenvolvido um sistema chamado Soccer, que era capaz de analisar o jogo de futebol humano, produzindo comentários em linguagem natural em tempo real.

Conforme [STO 2000a], o futebol de robôs foi introduzido como um interessante e promissor domínio de pesquisa para a IA em 1992 por [MAC 93]. O Dynamite foi o primeiro sistema de futebol de robôs usado para introduzir uma estratégia de decisão chamada "deliberação reativa" [SAH 93]. O robô possui um módulo de alto nível (cognitivo) e um módulo executor. O primeiro é responsável por selecionar uma ação apropriada ou um objetivo para a situação corrente. O executor consiste de ações de plano de baixo-nível, como seguir o caminho, defender, parar.

[ASA 94] desenvolveu os primeiros robôs equipados com capacidade de sentidos. Esses robôs usam aprendizagem e técnicas de treinamento. Eles aprendem a bater em uma bola parada dentro da área de gol. Uma contribuição desse trabalho é a combinação de comportamentos de baixo nível, como chutar e evitar um oponente, além da construção de estado e espaço de ação que reduz a complexidade de tarefas de aprendizado.

O simulador oficial da RoboCup, o Soccerserver [NOD 95], proporciona o domínio e o suporte aos usuários que desejam construir seus próprios agentes. Embora sirva como uma proposta de ilustração, o futebol de robôs proporciona complexidade e características interessantes como testes para SMA. Mesmo sendo um jogo, vários aspectos do mundo real são assegurados. A idéia chave do futebol é que os agentes não devem somente controlar a eles mesmos, mas também controlar a bola, que é uma parte passiva do meio ambiente.

O Soccerserver é realístico em muitos aspectos, como: a visão do jogador é limitada; o jogador pode comunicar-se enviando recado; todos os jogadores são controlados por processos separados; cada jogador tem 10 companheiros e 11 oponentes; cada jogador tem energia limitada; ações e sensores têm ruídos; e o jogo ocorre em tempo real.

A idéia da primeira RoboCup foi discutida em 1993, resultando em dois anos em estudos de viabilidade, um anúncio público da iniciativa em 1995 (IJCAI'95 em Montreal), e a competição preliminar em Osaka no Japão em 1996. A primeira Copa Mundial de Robôs, RoboCup-97, aconteceu em Nagoya, Japão, em Agosto de 1997, e incluiu a participação de mais de 40 times. A Segunda Copa Mundial de Robôs, RoboCup-98, aconteceu em Paris, em Julho de 98, com mais de 50 times participantes. A terceira Copa Mundial de Robôs, a RoboCup-99, foi em Stockholm onde 120 times estavam inscritos. Em Melbourne 2000, 150 times foram pré-registrados a estarem presentes na competição (Tabela 3.2). A próxima RoboCup acontecerá em Seattle em 2001 [ROB 2001].

TABELA 3.2 - Eventos RoboCup.

RoboCup	***Local/Data***	***Times inscritos***
Primeira	Nagoya, Japão, Agosto de 1997	+ 40
Segunda	Paris, França, Julho de 1998	+ 50
Terceira	Stockholm, Suíça, Julho de 1999	+ 120
Hoje	Melbourne, Austrália, Agosto 2000	+ 150
Amanhã	Seattle	

O fator mais extraordinário, perceptível a cada novo evento, está no aprimoramento do jogo desde a inauguração da Copa Mundial em 1997 [KIT 97c]. O esforço do agente para achar a bola e marcar o gol foi um dos primeiros desafios a serem alcançados. Hoje, concentra-se em estratégias mais difíceis, como cooperação e aprendizado de agentes.

Existe uma Federação RoboCup, que é uma organização internacional, formada por um grupo de pesquisadores com o objetivo de promover a ciência e a tecnologia, usando o jogo de futebol como domínio, tanto para pesquisas em robótica como para pesquisas em agentes. Ela envolve mais de 35 países e 150 universidades, institutos de pesquisas nacionais e empresas privadas.

3.2.4 A Análise da Estrutura

A RoboCup apresenta uma série de atividades, dentre elas estão as competições e as conferências internacionais. Essas competições ocorrem através de três ligas principais, que são [COA 2000]:

- ***Simulator League*** (Liga do Simulador): cada time consiste de 11 programas. Cada um dos 11 membros de um time é controlado separadamente. O simulador oficial usado chama-se Soccerserver (desenvolvido por Itsuki Noda [NOD 95], apresentado na Figura 3.1). Cada jogador tem capacidade de percepção (visão e audição) e energia de movimento, ambas limitadas. A comunicação pode ser realizada entre os jogadores e regras rigorosas do jogo de futebol são aplicadas. Essa liga destina-se, principalmente, às pesquisas com alto interesse em SMA complexos e abordagens de aprendizado.

  A liga de simuladores continua sendo a mais popular da RoboCup. Em Stockholm 1999, foram 37 times participantes, em Melbourne 2000, foram 43 times qualificados para a apresentação entre 150 inscritos.

  O agente jogador da liga do simulador apresenta as seguintes características [ASA 99]:

- Cooperação entre os agentes, envolvendo habilidades de baixo nível, como passar a bola e habilidades de alto nível, como execuções de estratégias.
- O modelo do agente possui habilidades primitivas, como reconhecer a intenção do agente de passar a bola e habilidades complexas, como reconhecimento de um plano de alto nível.
- Aprendizado de SMA *on-line* e *off-line*, com habilidades como passar e interceptar a bola e aprendizado de estratégias complexas (por exemplo: aprender estratégias do time oponente).

- ***Small-size real robot league*** (Liga dos Robôs Reais de Pequeno Porte): O campo possui o mesmo tamanho e a mesma cor de uma mesa de ping-pong. Essa liga apresenta cinco robôs por time, que jogam uma partida com uma bola de *golf*.
- ***Midle-size real robot league*** (Liga dos Robôs Reais de Médio Porte): O campo é do tamanho e da cor de 3 mesas de ping-pong. Essa liga apresenta cinco robôs por time, utilizando a bola de Futsal-4.

No restante do manuscrito, será considerada apenas a liga do simulador.

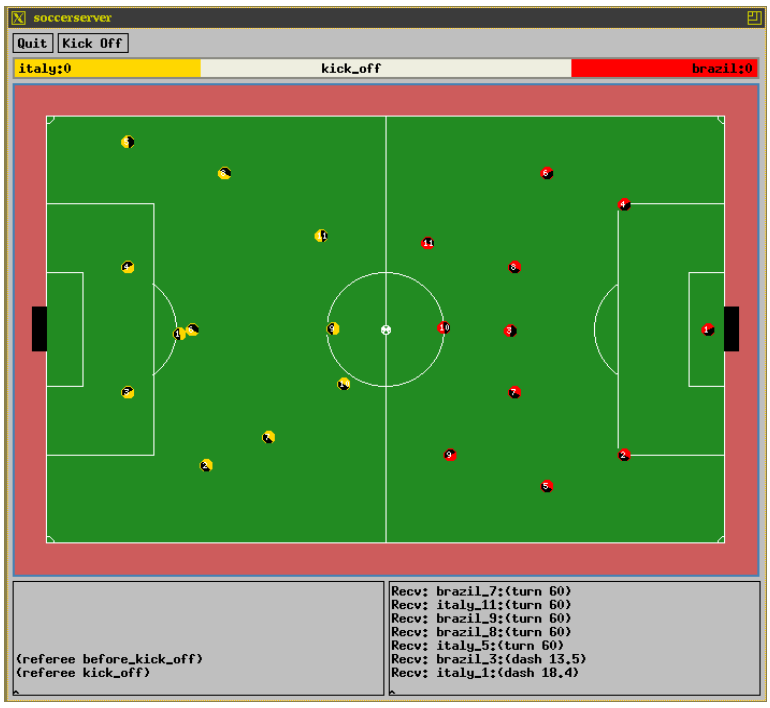


FIGURA 3.1 - O Simulador Soccerserver.

Durante a competição (liga do simulador), os times são divididos em grupos, de forma a obter o melhor de cada grupo para a próxima rodada, até que sejam conhecidos os três finalistas. A qualificação de cada time é determinada de acordo com o número de vitórias, a diferença de gols, o número de gols, a contagem de jogos e o número de empates.

Sistemas comentaristas automáticos são usados na liga de simulação, como por exemplo o MIKE (*Multi-agent Interactions Knowledgeably Explained*). O MIKE é capaz de produzir textos, interpretando as ações realizadas pelos agentes [TAN 98]. Esses sistemas utilizam técnicas de análises estatísticas para avaliar os times, gerando uma documentação. O LogMonitor é uma outra ferramenta para analisar jogos gravados [TOM 2000]. Ele pode ser usado não somente para o aprimoramento da colaboração entre o time, mas também, para avaliar os próprios jogadores.

Outra ferramenta que auxilia na análise, na avaliação e no entendimento do comportamento do time é o Issac [TAM 2000]. Trata-se de um agente analista que realiza uma análise *off-line* do time.

3.3 O Simulador da RoboCup

O simulador oficial da Copa Mundial de Robôs, o Soccerserver*,* foi desenvolvido com a intenção de proporcionar um ambiente de testes que possibilite a comparação do desempenho de diferentes SMA. Também permite observar testes com novas técnicas de cooperação de agentes, onde cada jogador é requerido para jogar cooperativamente em situações dinamicamente variadas.

O Soccerserver apresenta muitas complexidades do futebol do mundo real, incluindo, dentre as mais importantes, as seguintes [ASA 99] e [STO 99b]:

- Os jogadores possuem um número limitado de informações sensoriais.
- Os jogadores podem mandar mensagens somente através do Soccerserver*,* as quais são acessíveis apenas para jogadores a uma certa distância.
- Os jogadores possuem uma energia limitada (*stamina*) que é consumida por ações de caminhada (*dash*).
- As ações e informações sensoriais mandadas aos jogadores contêm "ruídos" adicionados pelo próprio simulador.
- O jogo ocorre em tempo real e em um ambiente dinâmico.

O campo de futebol e todos os objetos que estão nele são bidimensionais. O Soccerserver apresenta um campo virtual, chamado *Soccermonitor*. O tamanho do campo é o oficial das regras de futebol humano. O comprimento é de 105m (metros) e a largura do campo é de 68m. A largura do gol é dupla, sendo 14.02m, isso porque a simulação bidimensional dificulta a realização de um gol.

Cada agente pode ver somente uma porção do mundo (percepção limitada), dependendo, da direção da sua cabeça. Todas as informações recebidas estão em coordenadas polares com relação a própria posição do agente.

Existem, no campo de futebol, objetos móveis (a bola e os jogadores) e objetos estacionários (gols, *flags* e linhas). Os objetos estacionários (também chamados de marcas) são usados pelo cliente para determinar a posição absoluta dos jogadores.

Cada objeto móvel é um círculo que possui as características apresentadas na Tabela 3.3 a seguir [NOD 95].

TABELA 3.3 - Características dos objetos móveis [NOD 95].

Características	***Descrição***
Tamanho	Raio do círculo do objeto. O tamanho dos jogadores e da bola é respectivamente 1m e 0.15m (valor *default*).
Posições	Localização bidimensional do centro do objeto no campo.
Velocidade	Velocidade bidimensional do objeto no campo.
Aceleração	Aceleração da velocidade.
Direção	Direção do objeto. No caso do jogador, o objeto pode acelerar ao longo da direção. A direção do jogador também significa uma direção central vista pelo jogador.
Decaimento	Parâmetro de decaimento de velocidade do jogador (*default* 0.5). No caso da bola, o valor *default* é de 0.8.
Parâmetros de ruídos	Razão de incerteza do movimento do objeto. Se o parâmetro é 0, o objeto se move exatamente de acordo com a velocidade. Se o parâmetro é positivo, ruídos são adicionados no movimento. A magnitude do ruído é proporcional ao parâmetro e à velocidade.

3.3.1 A Arquitetura do Soccerserver

O Soccerserver proporciona um servidor de conecções UDP/IP e um *display* gráfico Xwindows (*Soccermonitor*), onde são apresentados o "campo de futebol virtual" e os jogadores de ambos os times.

A arquitetura do Soccerserver consiste de três módulos [COR 2001]: um módulo simulador de campo, um módulo árbitro e um módulo de mensagens (Figura 3.2). O módulo simulador de campo calcula os movimentos dos objetos no campo e verifica colisões entre eles. O módulo árbitro controla o jogo segundo as regras atribuídas pelo simulador. E o módulo de mensagens gerencia a comunicação entre os clientes.

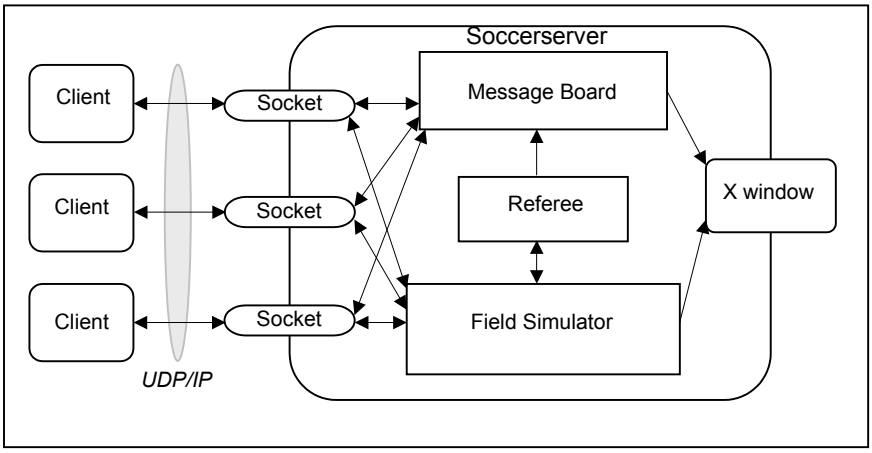


FIGURA 3.2 - Arquitetura do Simulador Soccerserver.

3.3.2 As Características do Simulador

Segundo [COR 2001] e [NOD 2001], dentre as principais características atribuídas ao simulador, destacam-se:

- Sistema cliente-servidor.
- Proporciona um campo virtual. Um cliente é o cérebro do jogador em campo e controla as ações do jogador.
- Sistema aberto. Os clientes podem ser escritos por qualquer sistema de programação que tenha interface UDP/IP.
- O Soccerserver foi implementado em C++ usando o compilador g++ no sistema X-window com Athena widget-set.
- Múltipla plataforma. O Soccerserver roda em SunOS 4.1.x, Solaris 2.x, DEC OSF/1, NEWS-OS, Linux e Irix 5.
- O Soccerserver consiste de dois programas: Soccerserver e Soccermonitor. O Soccerserver é o simulador de um campo que calcula os movimentos dos jogadores e da bola. O Soccermonitor é a interface que mostra a janela do campo em um monitor.

3.3.3 O Objetivo do Jogo

O objetivo do time, em uma partida de futebol, é ganhar a partida, marcando o máximo de gols possíveis e impedindo o oponente de marcar gols. Para que o objetivo possa ser alcançado, os jogadores devem possuir habilidades ofensivas e defensivas, dependendo do seu papel no jogo (atacante, defesa, meio de campo).

3.3.4 Inconsistências e Incertezas

O simulador apresenta características interessantes, como a incerteza e a inconsistência de informações. Cada programa cliente deve controlar efetivamente seu jogador, sabendo lidar com alguns fatores randômicos na ação dos jogadores e com restrições de desempenho do próprio jogador. O jogador, além de trabalhar com essas incertezas, deve saber agir em tempo real. Portanto, o projeto do cliente deve levar em conta tais considerações e processar cada informação seguramente.

A incerteza na ação é um fator importante. Nem sempre os comandos enviados ao servidor são executados. O comando pode ser perdido na rede ou ainda a informação enviada pelo cliente pode não ser aceita pelo simulador, por causa do tempo de envio da mensagem. Os comandos enviados ao Soccerserver possuem um tempo de resposta e nada garante que, na próxima informação visual, o efeito do comando seja percebido.

Assim, a execução de comandos não é exata. Por exemplo: o servidor aceita apenas um comando de ação a cada 100ms. Se mais que um comando for enviando dentro de um mesmo ciclo de simulação, o Soccerserver poderá não aceitar um desses comandos.

Os movimentos dos jogadores não são tão seguros. O comando *turn* 90 (giro) não significa que o jogador vira exatamente 90°, mas sim, que o jogador modifica a direção em um ângulo aproximado de 90°. O servidor adiciona ruídos para os movimentos dos objetos. A bola vai para uma direção aproximada, quando um cliente envia um comando *kick* (chutar). Portanto, os clientes podem possuir mecanismos para confirmar efeitos de comandos enviados por si mesmos.

Outro fator é o limite do sensor de informação. Cada cliente pode ter uma informação visual estreita, como por exemplo: 60°. Para ter a informação visual completa do campo (360°), o cliente deve rotacionar o jogador usando o comando *turn* e integrar as informações visuais enviadas pelo servidor.

Os jogadores não sabem os valores de posição exatos de muitos objetos distantes. O máximo de ruído adicionado pelo simulador à informação de distância é por volta de 10%. Por exemplo, quando a distância é aproximadamente 100m, o ruído adicionado à distância é por volta de 10m.

As informações sobre os jogadores visualizados são perdidas, quando as distâncias entre os objetos são longas. Por exemplo: o cliente obtém o nome do time e o número do uniforme de um jogador em sua visão, se o jogador está perto o bastante. Se o jogador está distante (*unum_too_far_length*), o cliente não pode obter o número do uniforme, e se o jogador está muito longe (*team_too_far_length*), o cliente não pode igualmente obter o nome do time (Figura 3.3). Portanto, para saber cada informação sobre os jogadores que estão muito longe, o cliente deve inferir uma história passada da informação enviada pelo servidor.

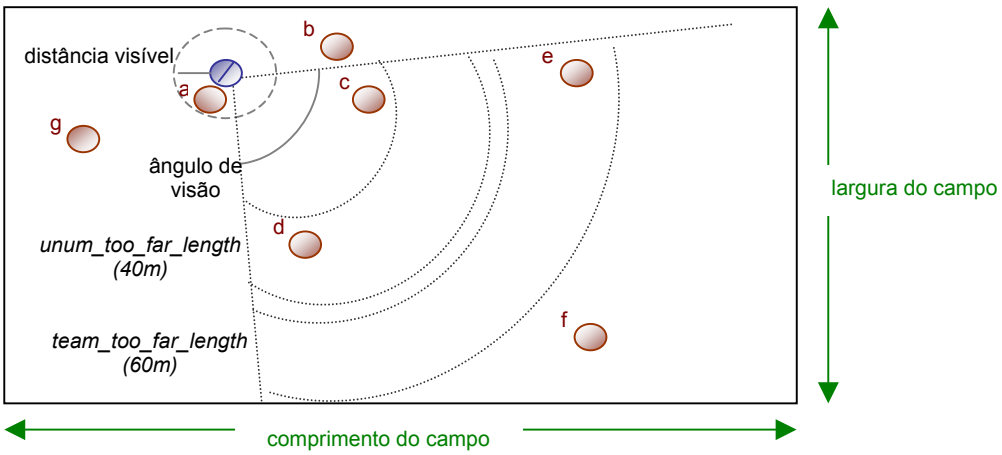


FIGURA 3.3 - Alcance de Visão de um Agente.

Outros fatores interessantes próximos da realidade são fornecidos pelo simulador. Cada jogador possui uma certa quantidade de energia (*stamina*), que é decrementada quando o jogador caminha. Por exemplo: quando um cliente envia um comando de caminhada (*dash* 80), a energia decresce 80 pontos. Essa energia é incrementada constantemente, a cada ciclo de simulação, até alcançar o máximo permitido (3500 pontos).

Os movimentos dos objetos são simulados passo a passo e incrementados da seguinte maneira: dado um comando do cliente, a velocidade de um objeto é adicionada à sua posição, enquanto a velocidade decai para uma certa razão e incrementa de acordo com sua aceleração. As colisões são simplificadas, se um objeto coincidir com um outro objeto, após um movimento, cada objeto é movido para trás até não sobrepor o outro objeto (a velocidade é multiplicada por –0.1).

Existem três parâmetros relevantes associados ao jogador: energia, força e "*recovery*". A energia é usada quando o jogador está caminhando e é reposta a cada ciclo. A força determina o quão efetivo é a caminhada. O "*recovery*" controla quanto de energia é recuperada a cada ciclo.

Uma força negativa pode causar ao agente a caminhada para trás, acarretando um consumo de energia duas vezes maior.

Como mencionado acima, cada cliente tem informações limitadas sobre o campo. A comunicação e o fato de compartilhar cada informação entre os clientes é uma forma de vencer o problema. Porém no Soccerserver não é tão simples. O servidor proporciona um canal de comunicação, mas com capacidade restrita, que limita o tamanho da mensagem, a distância na qual a mensagem poderá ser escutada e o número de mensagens.

A sincronização entre o agente e o simulador é mais um dos desafios a ser enfrentado na implementação do agente. A incorreta sincronização entre o agente e o Soccerserver pode levar o jogador a apresentar um comportamento completamente indesejado.

3.3.5 Planos de Jogo

O futebol explora as vantagens proporcionadas pelos SMA, como a cooperação entre jogadores. Esse é um fator determinante para decidir a habilidade dos times. A cooperação entre os jogadores pode ser demonstrada, por exemplo, através de um simples passe.

Um passe é a ação mais básica em um time. Para passar uma bola entre os jogadores, esses devem ter um consenso sobre o curso dos passes. No caso de passes entre dois jogadores, não é fácil achar um consenso.

De acordo com o exemplo extraído de [MAT 96], considere uma situação como na Figura 3.4. Neste caso, foram considerados dois planos de passes: "um ou dois passes" (Figura 3.5) e "através do passe" (Figura 3.6). Para quebrar a defesa, jogadores do ataque (ofensivos) precisam decidir qual dos dois planos de passe devem adotar.

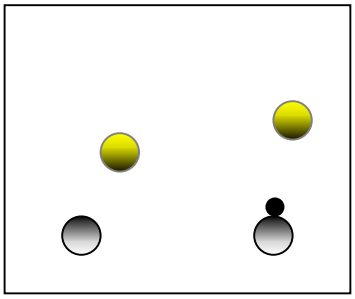


FIGURA 3.4 - Planos de Passe.

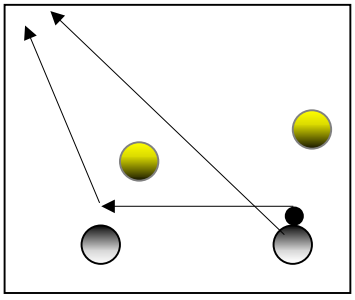


FIGURA 3.5 - Plano de Passe 1.

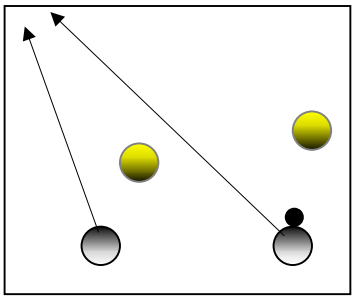


FIGURA 3.6 - Plano de Passe 2.

Outra combinação básica de jogo é a interceptação feita pela defesa. No jogo (conforme exemplo apresentado por [NOD 96b]), um defensor próximo do oponente, persegue o oponente do lado esquerdo ou direito, enquanto outros jogadores da defesa cobrem outro lado (Figura 3.7). Esse jogo envolve problemas com o passe, onde se torna necessário pesquisar um consenso e escolher um lado para cortar.

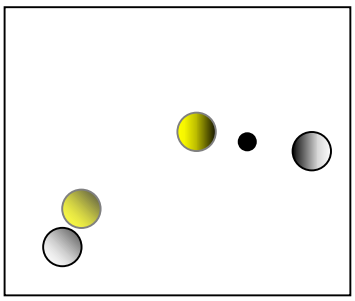


FIGURA 3.7 - Interceptação da Defesa.

3.3.6 A Partida

A partida de futebol é basicamente controlada pelo módulo árbitro. O árbitro anuncia suas decisões a todos os clientes em forma de mensagens auditivas. No entanto, faltas como a obstrução são difíceis de serem julgadas automaticamente, porque elas envolvem a intenção do jogador. Nas competições oficiais, existe a possibilidade de um árbitro humano marcar a falta. Esse árbitro humano pode marcar o chute livre (*free_kick_l* ou *free_kick_r*) para ambos os times ou ainda, colocar a bola em jogo em qualquer lugar do campo (realizar um *drop ball*).

De acordo com as regras do Soccerserver, o módulo árbitro controla o jogo da seguinte forma (como apresentado por [COS 99b] e [COR 2001]):

- **Gol**: quando a bola está dentro do gol, o árbitro anuncia o gol transmitindo uma mensagem para todos os clientes, atualiza o placar, move a bola para o centro do campo e suspende a execução da partida por cinco segundos, além de

modificar o modo do jogo para *kick_off* (saída de bola), e por fim, reinicia a partida. Enquanto isso, os clientes devem retornar para o seu lado do campo. Se o jogador permanecer no campo do adversário após os cinco segundos, o árbitro move o jogador para uma posição randômica do seu lado do campo.

- **Bola fora de campo**: quando a bola está fora de campo (fora das dimensões do campo), o árbitro move a bola para uma posição lateral do campo, marca do escanteio ou da pequena área e modifica o estado do jogo para *kick_in* (reposição de bola), *corner_kick* (escanteio) ou *goal_kick* (tiro de meta), movendo a bola para a posição apropriada.
- **Desobstrução**: quando o goleiro "segura" a bola, ou quando a partida se encontra em um dos seguintes modos: *kick_off*, *throw_in*, *corner_kick* ou *goal_kick*, o árbitro remove os jogadores adversários no raio de 9.15m da bola, para o perímetro do círculo de mesmo raio, centrado na bola. Quando ocorre um *goal_kick*, os jogadores são movidos para fora da área de pênalti, e não podem entrar na área enquanto o tiro de meta não for cobrado.
- **Bola em jogo**: quando o jogo encontra-se em um dos seguintes estados: *kick_off*, *throw_in*, *corner_kick* ou *goal_kick*, o árbitro modifica o estado do jogo para *play_on* (bola em jogo) após a bola ter sido colocada em jogo (*kick*) por um dos jogadores do time que detém a sua posse.
- **Final do primeiro tempo e final de jogo**: o árbitro suspende o jogo quando o primeiro tempo (3000 ciclos de simulação) ou o segundo tempo (6000 ciclos de simulação) termina (*half_time*). Cada tempo corresponde a 5 minutos (podendo ser modificado) de jogo. O árbitro difundirá uma mensagem informando o novo estado do jogo.

Cada cliente de cada time realiza a comunicação com o servidor por um comando "*init*". No momento em que todos os clientes estão prontos para jogar, o comissário do jogo (uma pessoa que invoca o servidor), dá inicio ao primeiro tempo do jogo. Quando a primeira metade de tempo termina, o servidor suspende o jogo. Durante esse intervalo, os competidores podem modificar os seus programas clientes.

Antes de iniciar o segundo tempo, cada cliente estabelece a comunicação novamente com o servidor por um comando "*reconnect*". Quando todos os clientes estão prontos, o comissário inicia o segundo tempo. Ao término do segundo tempo, o servidor pára o jogo. Poderá acontecer também uma prorrogação em um tempo extra, até ocorrer um gol. Esses passos são melhores detalhados no decorrer deste capítulo, no item 3.3.8 Inicialização.

Cada partida pode assumir os seguintes estados [COS 99b] e [NOD 2001]:

- *before_kick_off*: o jogo está parado, aguardando o início. Esse estado ocorre no início do jogo, antes de um *kick_off* ou durante os cinco segundos de intervalo dados pelo árbitro após o acontecimento de um gol. Todos os jogadores devem estar em seu lado do campo. Caso contrário, o árbitro modifica a posição do jogador randomicamente.

- *time_over*: final do jogo.
- *play_on*: quando a bola está em jogo.
- *kick_off_l* ou *kick_off_r*: saída da bola para o time da esquerda ou da direita, respectivamente.
- *kick_in_l* ou *kick_in_r*: reposição da bola em jogo para o time da esquerda ou para o time da direita, respectivamente.
- *free_kick_l* ou *free_kick_r*: chute livre para o time da esquerda ou para o da direita, respectivamente.
- *corner_kick_l* ou *corner_kick_r*: cobrança de escanteio para o time da esquerda ou para o time da direita.
- *goal_kick_l* ou *goal_kick_r*: cobrança de tiro de meta para o time da esquerda ou para o time da direita.
- *goal_l* ou *goal_r*: ocorrência de um gol em favor do time da esquerda ou do time da direita.

3.3.7 Os Protocolos

Um cliente estabelece a comunicação com o servidor por meio de um *socket* UDP/IP. A partir desse momento, o servidor designa um jogador no campo de futebol para o cliente. Através do *socket*, o cliente envia ao simulador os comandos para controlar o jogador e recebe as informações sensoriais sobre o meio ambiente (campo de futebol). Os protocolos de controle facilitam a criação de programas clientes, que utilizam algum tipo de sistema de programação que suporte *sockets*.

O Soccerserver simula os movimentos dos objetos (bola e jogador) e as colisões entre eles. O simulador é simplificado e procura calcular as modificações em tempo real, sem perder a essência da dinâmica do futebol. O simulador também trabalha independentemente da comunicação com os clientes. Dessa forma, os clientes podem presumir que as situações no campo modificam-se dinamicamente.

Segundo [COR 2001], toda a comunicação entre o servidor e cada cliente é feita via *socket*, utilizando *strings* ASCII, onde cada S-expressão é uma unidade de comunicação. As principais primitivas de comunicação entre o servidor e o cliente são as seguintes:

- *turn*
- *dash*
- *kick*
- *say*
- *see*
- *hear*

Detalhes sobre esses comandos são apresentados no item 3.3.9 Comandos de Controle e no item 3.3.10 Informações Sensoriais.

3.3.8 A Inicialização

Conforme [NOD 95], para que uma partida possa ocorrer, é necessário que, primeiramente, cada cliente estabeleça uma comunicação com o servidor via *socket* (o número *default* da porta do servidor é 6000) de duas maneiras: "iniciando" e "reiniciando" o jogador. O "início" é usado, quando um cliente quer nomear um novo jogador em campo. O "reinício" é utilizado, quando um cliente quer indicar um jogador existente (para o segundo tempo do jogo).

Para o início do jogo, o cliente envia a seguinte mensagem ao servidor:

(*init TeamName* (*Pos_x Pos_y*)[(*version VerNum*)] [(*goalie*)])

O servidor designa um jogador que pertencerá ao time *TeamName* (o nome do time poderá conter os caracteres: *(-|-|a-z|A-Z|0-9)+*), e posiciona-o nas coordenadas (*Pos_x Pos_y*). Se a inicialização do jogador for bem sucedida, o servidor retorna a seguinte mensagem para o cliente:

(*init Side UniformNumber PlayMode*)

Onde *Side*, *Uniform_Number*, *PlayMode* indicam respectivamente, o lado do campo em que o time sai jogando ("l" - esquerda ou "r" – direita do campo), o número do uniforme do jogador (de 1 à 11), e o estado do jogo (*before_kick_off*, *kick_off*, *play_on*, *throw_in*, *corner_kick*, e *goal_kick*).

Se o servidor falhar ao designar um novo jogador, ele enviará a seguinte mensagem para o cliente:

(*error no_more_team_or_player*)

Quando todos os clientes estiverem prontos para jogar, o comissário do jogo, iniciará o jogo pressionando o botão *kick_off* da janela do servidor (*Soccermonitor*) e o primeiro tempo começará.

Para o reinício do jogo, um cliente envia a seguinte mensagem ao servidor:

(*reconnect TeamName UniformNumber*)

Se o servidor, com sucesso, reiniciou o jogador, ele retornará uma mensagem para o cliente:

(*init Side UniformNumber PlayMode*)

Se o servidor falhar ao reiniciar o jogador, o servidor enviará uma das seguintes mensagens para o cliente:

(*error no_more_team_or_player*)

(*error reconnet*)

3.3.9 Os Comandos de Controle

Os comandos de controle são as mensagens enviadas do cliente para o servidor com respeito a ação a ser executada pelo jogador. Há seis tipos usuais de comandos de controle: *turn*, *dash*, *kick*, *say*, *chage_view* e *move.*

Os comandos *turn*, *dash* e *kick* são usados para controlar o jogador durante o jogo. O comando *move* é usado para mover o jogador para uma certa posição, quando o jogo está no estado *before_kick_off*, antes do início. No caso do goleiro, o *move* também é usado para posicionar o jogador após o mesmo pegar a bola.

As ações primitivas como: *kick*, *dash* e *turn* são usadas em blocos de construção para implementar ações mais sofisticadas. Por exemplo: um *drible* poderá envolver um chute, uma caminhada e um giro.

O agente poderá enviar uma ação a cada período de simulação (ciclos de 100ms). Algumas primitivas poderão ser enviadas ao simulador apenas uma vez a cada ciclo, como por exemplo a primitiva *dash*. Outras poderão ser enviadas várias vezes dentro de um mesmo ciclo, como indicado na Tabela 3.4, apresentada em [COR 2001].

TABELA 3.4 - Comandos de controle do cliente [COR 2001].

Do cliente para o servidor	**Somente um por ciclo**
(*catch Direction*)	Sim
(*change_view Width Quality*)	Não
(*dash Power*)	Sim
(*kick Power Direction*)	Sim
(*move* X Y)	Sim
(*say Message*)	Não
(*sense_body*)	Não
(*turn Moment*)	Sim
(*turn_neck Angle*)	Sim

Caso o servidor não reconheça o comando enviado pelo cliente, ou se for um

comando ilegal, alguns tipos de erros poderão ser enviados em resposta ao cliente, como: (*error unknown_command*) ou (*error illegal_command_form*).

A seguir, apresenta-se a descrição dos comandos de controle (ações) utilizados pelo Soccerserver [NOD 95] e [COR 2001]:

- (***move*** X Y): move o jogador para a posição (x, y). A origem das coordenadas é o centro do campo. O valor de x é negativo para localizar o jogador em seu próprio lado do campo. Considerando o jogador de frente para o centro do campo, o lado direito do jogador contém um y positivo e o lado esquerdo contém um y negativo. Esse comando só é válido no modo *before_kick_off* e para o goleiro após pegar a bola (após efetuar o comando *catch* (pegar) com sucesso).
- (***turn*** *Moment*): modifica a direção do jogador de acordo com o parâmetro *Moment*. O *Moment* pode ser um giro de -180 (*minmoment*) ~ 180 (*maxmoment*) graus, em torno do próprio eixo do jogador. Existe o aspecto de inércia, que torna mais difícil o giro quando o jogador está em movimento. Quando o jogador está em sua velocidade máxima (1.0), o máximo de *turn* efetivo que ele poderá fazer é -30 ~ 30.
- (***turn_neck*** *Angle*): modifica o ângulo da cabeça do jogador relativo ao seu corpo. O ângulo da cabeça deve ser entre o intervalo de –90 ~ 90. Esse ângulo indica qual é o ângulo de visão do jogador.
- (***dash*** *Power*): incrementa a aceleração do jogador para a direção atual (direção do corpo), de acordo com a potência especificada (*Power)*. A potência pode assumir valores inteiros do intervalo -30 ~ 100. Se *Power* for negativo, então o jogador efetua um *dash* para trás, consumindo assim, duas vezes mais a sua energia. Para o jogador realizar uma corrida, *dash* seguidos devem ser enviados ao simulador.
- (***say*** *Message*): transmite mensagens para todos os jogadores que estão num raio de 50m de quem enviou a mensagem. A mensagem é informada imediatamente ao cliente através do comando *hear*. Essa mensagem pode consistir de caracteres do alfabeto, números e símbolos ("+-*/_"), com tamanho máximo de 512 caracteres.
- (***kick*** *Power Direction*): chuta a bola com a potência –30 ~ 100, na direção –180 ~ 180. Esse comando só terá efeito se a bola estiver na margem de chute do jogador (*kickablearea*), como especificado pelo Soccerserver. O comando *kick* é similar ao *dash*, exceto que a aceleração é aplicada na bola e não no jogador. A área de chute (*kickableare*) do jogador é dada pelos parâmetros do simulador: *player_size* + *ball_size* + *kickable_margin*. Uma maior força no chute pode ser conseguida quando a bola está diretamente em frente ao jogador e num ângulo fechado.
- (***change_view*** *Width Quality*): modifica a profundidade e a qualidade da visão do jogador. A profundidade corresponde à estreita (*narrow,* 45°), normal (90°)

ou ampla (*wide,* 180°) e a qualidade da informação visual pode ser baixa (*low*) ou alta (*high*). No caso de alta qualidade de visão, o servidor envia ao jogador uma informação detalhada sobre a posição dos objetos, bem como, um maior número de objetos pode ser visualizado pelo jogador. Com baixa qualidade de visão, o servidor envia um menor número de objetos e uma informação reduzida sobre os objetos.

A freqüência da informação visual também se modifica de acordo com o ângulo e a qualidade de visão do jogador. No caso de um ângulo normal e de uma alta qualidade de visão, a informação visual é enviada ao jogador a cada 150ms (este intervalo pode ser modificado pelo parâmetro *send_step*). Com um ângulo amplo, a freqüência é aproximadamente de 300ms e com um ângulo estreito a freqüência é aproximadamente 75ms, supondo uma qualidade de visão alta. Por exemplo: ângulo = estreito, qualidade = baixa, a informação pode ser enviada pelo simulador a cada 37.5ms.

- (***catch*** *Direction*): utilizado apenas pelo agente goleiro para tentar agarrar a bola na direção dada por *Direction*. A possibilidade de sucesso é definida no parâmetro do Soccerserver chamado: *catch_probability*. Para que o goleiro possa efetivamente pegar a bola, ele deve se encontrar em uma área definida como *goalie_cacthable_area*, um retângulo de 2m x 1m. A probabilidade de agarrar a bola, se ele está no retângulo, é dada pelo *catch_probability*. O goleiro só pode fazer um *catch* a cada poucos ciclos (especificado pelo parâmetro *catch_ban_cycle*), se o goleiro tentar pegar a bola no mesmo *ban_cycle*, o comando é ignorado pelo simulador. Se o goleiro pegar a bola com sucesso, o servidor altera o modo de jogo para *free_kick*. Uma vez pega a bola, o goleiro pode se mover dentro da área de pênalti, juntamente com a bola, utilizando o comando *move*.

O modelo de ação utilizado pelo simulador é discreto, ou seja, todos os movimentos de um objeto em um ciclo ocorrem simultaneamente ao final do mesmo. Ao término do ciclo, o servidor toma todas as ações recebidas e aplica-as nos objetos em campo, partindo da posição corrente e da informação de velocidade para então calcular a nova posição e a velocidade corrente de cada objeto. Os comandos de ação podem, ainda, não causar os efeitos esperados por causa da situação física e dos ruídos.

3.3.10 As Informações Sensoriais

Um cliente possui três tipos de informações sensoriais: as informações visuais, as informações auditivas e as informações do próprio corpo, que são transmitidas por mensagens *see*, *hear* e *sense_body*, respectivamente [NOD 95] e [COR 2001]. Essas informações servem como fontes de percepção, não somente do mundo externo, mas dos efeitos das ações e da situação do agente.

As informações visuais são mandadas pelo Soccerserver ao cliente, informando a distância e o ângulo relativo aos pontos estacionários (como: *flags* -bandeiras-, linhas, entre

outros) aos jogadores (nome do time, número do uniforme, distância e ângulo dos jogadores, com certo alcance, baseado no campo visual corrente do agente) e à bola (distância, variação de distância, ângulo e variação do ângulo da bola).

Os jogadores podem ver os objetos que estão em seu campo visual, ou seja, dentro de seu ângulo de visão, na direção da sua cabeça. Esse setor visível pelo jogador depende de fatores como os parâmetros do simulador: *sense_step* e *visible_angle*, que determinam o intervalo entre cada informação visual e o grau de visão do jogador, respectivamente. Na maioria dos casos, a freqüência da informação visual é de 150ms e o ângulo de visão normal é de 90 graus.

O jogador estará influenciando na freqüência e na qualidade da visão sempre que alterar os parâmetros de profundidade (*view_width*) e de qualidade de visão (*view_quality*) do comando *change_view*.

O jogador pode também "ver" um objeto se esse está na sua *visible_distance*. Se o objeto está na vizinhança (3m), mas fora do campo visual. O jogador pode saber somente o tipo do objeto (bola, jogador ou pontos estacionários), mas não exatamente a identificação do objeto.

As informações de visão são enviadas ao cliente, no seguinte formato:

(***see*** *Time ObjInfo ObjInfo ...*)

Onde o *Time* indica o tempo corrente e o *ObjInfo* contém as informações sobre o objeto visível, contendo o seguinte formato:

(*ObjName Distance Direction DistChng DirChng BodyDir HeadDir*)

ObjName::= (player *Teamname UniformNumber*)

| (*goal Side*)

| (*ball*)

| (*flag* [l|c|r] [t|b])

| (*line* [l|c|r|t|b])

Distance::= distância relativa do objeto (em relação ao centro do círculo)

Direction::= direção relativa do objeto

DistChng::= distância modificada

DirChng::= direção modificada

BodyDir::= direção do corpo do jogador

HeadDir::= direção da cabeça do jogador

Um exemplo de uma informação visual enviada pelo servidor a um cliente pode ser visualizada na Figura 3.8.

```
Recebido: (see 29 ((goal r) 90 4) ((flag c) 38.1 -2) ((flag c t) 56.3 -38)
((flag c b) 45.2 44) ((flag r t) 99.5 -15) ((flag r b) 92.8 26) ((flag p l t)
28.5 -77) ((flag p l c) 8.3 -72 -0.166 -2.2) ((flag p r t) 78.3 -11) ((flag p
r c) 73.7 3) ((flag p r b) 73.7 19) ((flag g r t) 90.9 0) ((flag g r b) 89.1
9) ((flag t l 40) 47.5 -83) ((flag t l 30) 47.9 -71) ((flag t l 20) 50.4 -60)
((flag t l 10) 54.6 -50) ((flag t 0) 60.3 -41) ((flag t r 10) 66.7 -35) ((flag
t r 20) 74.4 -29) ((flag t r 30) 82.3 -25) ((flag t r 40) 90 -21) ((flag t r
50) 99.5 -18) ((flag b l 30) 31.5 87) ((flag b l 20) 35.2 71) ((flag b l 10)
40.9 58) ((flag b 0) 47.9 49) ((flag b r 10) 56.3 43) ((flag b r 20) 64.7 38)
((flag b r 30) 73.7 34) ((flag b r 40) 83.1 31) ((flag b r 50) 91.8 29) ((flag
r t 30) 101.5 -11) ((flag r t 20) 98.5 -6) ((flag r t 10) 96.5 0) ((flag r 0)
94.6 5) ((flag r b 10) 94.6 11) ((flag r b 20) 95.6 17) ((flag r b 30) 96.5
23) ((ball) 40.4 0) ((player t1 2) 14.9 -24 -0.298 -0.4 10 10) ((player t1 3)
11 6 -0.22 0.2 -3 -3) ((player t1 4) 20.1 -40 -0.402 -0.5 24 24) ((player t1
5) 33.1 -46 -0 -0.3 49 49) ((player t1) 24.5 23) ((player t1 7) 24.5 -10 -0.49
-0.1 10 10) ((player t1 8) 33.1 -31 -0 -0.3 49 49) ((player t1) 36.6 34)
((player t1) 40.4 -1) ((player) 44.7 -29) ((player) 90 -2) ((player) 66.7 3)
((player) 66.7 -3) ((player) 66.7 9) ((player) 60.3 19) ((player) 60.3 -15)
((player) 54.6 1) ((player t2) 54.6 20) ((player) 54.6 -28) ((player t2) 40.4
0) ((player) 44.7 26) ((line r) 90.9 -79))
```

FIGURA 3.8 - Informação Visual.

No campo de futebol existem 55 pontos estacionários, que são: *flags* (o gol conta como uma *flag*) e 4 linhas, que podem ser visíveis pelo jogador, como apresentado na Figura 3.9.

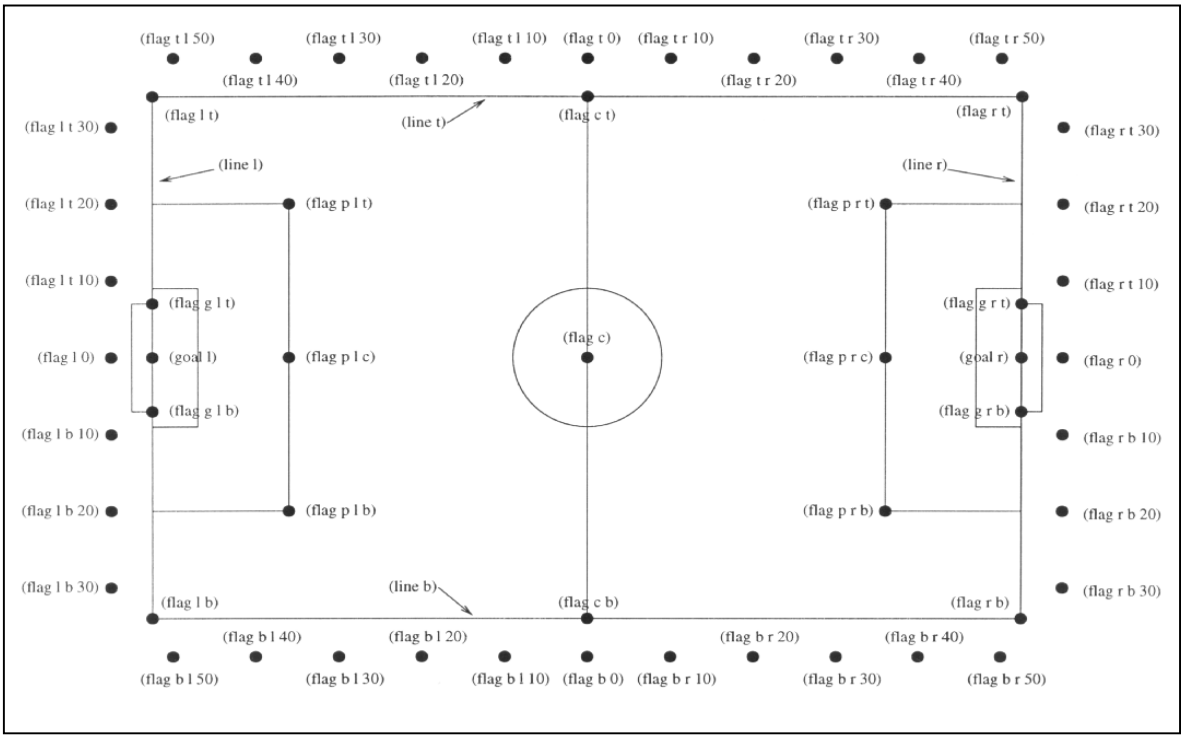


FIGURA 3.9 - Pontos Estacionários.

Os caracteres "l, c, r, t, b" que acompanham as *flags* e linhas, significam respectivamente: esquerda, centro, direita, acima e abaixo, e indicam o local no campo em que a *flag* está posicionada. Por exemplo: (*flag p r c*), onde "p" significa que é uma *flag* da área de pênalti, "r" significa que a *flag* é da direita e "c" do centro. O objeto (*goal* r), por exemplo, representa o centro da linha do gol da direita. Alguns tipos de *flags* estão localizados a 5m fora do campo. Por exemplo, (*flag* t l 20) está a 5m da linha do topo e a 20m à esquerda da linha do centro.

No caso do objeto (*line Distance Direction*), a informação de distância corresponde à distância do ponto onde o centro da linha de visão do jogador cruza a linha do campo. A informação de direção é o valor absoluto do ângulo formado entre a linha de visão do jogador e a linha lateral do campo que o jogador está vendo.

Toda comunicação entre clientes é feita através do servidor. A comunicação direta entre os agentes não é permitida. Um cliente pode enviar uma mensagem ao servidor através de um comando *say*, e então a mensagem é transmitida para todos os clientes por uma informação *hear*.

As mensagens transmitidas por um jogador, somente serão informadas aos jogadores distantes até 50m do jogador que as enviou. Sendo assim, o atacante não poderá escutar o que foi dito pelo goleiro, conforme ocorre no jogo de futebol real. Por esse mesmo motivo, toda a comunicação é controlada pelo simulador, evitando que situações pouco realísticas ocorram.

As informações auditivas são mensagens assincronamente enviadas por outros jogadores através do Soccerserver ou pelo próprio árbitro (por exemplo: *corner_kick* ou *free_kick*).

A informação auditiva apresenta o seguinte formato:

(***hear*** *Time Direction Message*)

Essa mensagem é enviada imediatamente, quando um cliente envia o comando (*say Message*). O parâmetro *Time* indica o tempo corrente e *Direction*, indica quem enviou a mensagem (por exemplo: *self* -ele mesmo-, *referee* -árbitro-).

Um exemplo de uma informação auditiva enviada pelo servidor a um cliente pode ser visualizada na Figura 3.10.

```
Recebido: (hear 39 referee foul_r)
```

FIGURA 3.10 - Informação Auditiva.

Informações sobre o próprio jogador podem ser obtidas através do comando *sense_body*. Esse comando é enviado pelo servidor para cada jogador, a cada *sense_step* (normalmente 100ms), indicando o *status* do corpo do jogador da seguinte forma:

(***sense_body*** *Time*

(view_mode View_Quality ViewWidth)

(*stamina Stamina Effort*)

(*speed AmountOfSpeed*)

(*head_angle HeadDirection*)

(*kick KickCount*)

(*dash DashCount*)

(*turn TurnCount*)

(*say SayCount*)

(*turn_neck TurnNeckCount*))

Onde, *View_Quality* é *high* ou *low* e *View_Width* é *narrow*, *normal* e *wide*.

AmountOfSpeed é uma aproximação da quantidade de velocidade do jogador. *HeadDirection* é a direção relativa da cabeça do jogador. As variáveis *Count* são o número total de comandos executados pelo jogador.

Um exemplo de uma informação *sense_body* enviada pelo servidor a um cliente pode ser visualizada na Figura 3.11.

```
Recebido: (sense_body 42 (view_mode high normal) (stamina 3500 1)
(speed 0.02) (head_angle 0) (kick 3) (dash 21) (turn 2) (say 0)
(turn neck 0))
```

FIGURA 3.11 - Informação do Estado do Jogador.

3.3.11 O Modo Treinador

Um agente privilegiado chamado “treinador” pode possibilitar a execução de seções de treinamento (certas ações, tais como, dribles, lançamentos e jogadas ensaiadas são testadas). Dessa forma, pode-se obter um maior controle sobre o jogo durante o processo de desenvolvimento dos agentes, proporcionando a oportunidade de se utilizar métodos de aprendizado.

O treinador tem uma visão global do posicionamento dos jogadores em campo e pode se comunicar com seus companheiros. O agente treinador apresenta as seguintes habilidades (como pode ser visto em [COR 2001]):

- Controlar o estado da partida (*play_mode*);
- Difundir mensagens “auditivas”;
- Mover os objetos (bola e jogadores) para qualquer posição do campo, independente do estado da partida;
- Obter informações a respeito de qualquer um dos objetos móveis (bola e jogadores) no campo.

Ao agente treinador pode-se atribuir o total controle do jogo, para isso, é necessário desativar o módulo árbitro da partida. Nesse caso, o agente treinador torna-se responsável pelo monitoramento do estado da partida, efetuando as respectivas mudanças e verificações do estado do jogo (*play_mode*) de acordo com suas próprias regras. No entanto, ambos, tanto o árbitro, quanto o treinador, podem ter o controle da partida. O treinador pode ser usado para controlar as seções de treinamento ou fornecer aos jogadores informações sobre o seu desempenho.

O agente treinador, em 1998 foi utilizado apenas na etapa de desenvolvimento dos jogadores. A partir da RoboCup-*99*, uma nova categoria foi adicionada à atual categoria de simuladores, onde foi permitida a utilização do agente treinador durante a competição. O objetivo é utilizar o agente treinador para observar a partida e passar informações estratégicas aos jogadores.

Os times podem obter vantagens utilizando o agente treinador como, por exemplo, escutar os conselhos do treinador a respeito da formação do time em campo. Dessa forma, o time poderá partir de uma função ofensiva para uma defensiva ou, simplesmente, mudar sua formação. O treinador também pode ser responsável por aprender as jogadas do time oponente, realizando uma modelagem do adversário e, assim, passar instruções para o seu time.

3.3.12 Logplayer

O Logplayer permite a repetição (*replay*) de um jogo gravado. O servidor escreve as informações enviadas para o Soccermonitor dentro de um *logfile*, e desta forma, gera um arquivo de *log*. Esse arquivo pode ser lido em um *logplayer* (programa que acompanha o Soccerserver), que conectado ao Soccermonitor permite rever o jogo.

3.3.13 Soccermonitor

O Soccerserver e o Logplayer enviam ao Soccermonitor três diferentes tipos de informação:

- Informações necessárias para desenhar a cena.
- A mensagem do jogador e do árbitro apresentadas na janela do Soccermonitor.

- Informações do monitor para desenhar círculos, linhas e pontos (não usado pelo servidor).

O monitor envia ao servidor os seguintes comandos:

- (*dispinit*): uma primeira mensagem como monitor.
- (*dispstart*): para iniciar o jogo (*kick_off*), o início do segundo tempo ou a prorrogação. Ignorado quando o jogo já está ocorrendo.
- (*dispfoul x y side*): para indicar a situação de uma falta marcada pelo árbitro humano. Onde x e y são as coordenadas da falta. O lado é LEFT(1) para um *free_kick* para o time da esquerda, NEUTRAL(0) para *drop_ball* e RIGHT(-1) para um *free_kick* para o time da direita.
- (*dispdiscard side unum*): apresenta um cartão vermelho (*kick_him_out*). Lado pode ser LEFT ou RIGHT, *unum* é o número do jogador (1 - 11).

Como um grande desafio, a RoboCup é definitivamente estimulante em uma ampla variedade de pesquisas e tem produzido rápidos avanços nas principais tecnologias. O crescimento do número de participantes na RoboCup permite a rápida expansão da pesquisa. Com as ligas, a RoboCup proporciona a oportunidade de aprender e compartilhar soluções em três diferentes plataformas de arquitetura de agentes.

4 Clientes Soccerserver

Neste capítulo, serão introduzidas as arquiteturas dos agentes de quatro times que participaram da RoboCup. Inicialmente, será apresentada a arquitetura do agente UFSC-Team 98 e as alterações realizadas para a melhoria da mesma. A seguir, será introduzida a arquitetura do agente CMUnited 98, observando-se aspectos do agente, bem como, o aperfeiçoamento proposto e aplicado ao agente CMUnited 99. Finalizando, tem-se a arquitetura do agente AT Humboldt 97 e do F.C. Portugal Team 2000.

4.1 A Arquitetura do Agente UFSC-Team

A arquitetura do agente UFSC-Team 98 foi desenvolvida na Universidade Federal de Santa Catarina, por Guilherme Bittencourt, Luciano Rottava da Silva e Augusto César Pinto Loreiro da Costa [UFS 2001]. A implementação realizada na linguagem de programação C++ explora idéias como: percepção, ação, comunicação, cooperação, planejamento e decisão.

De acordo com [COS 98], a arquitetura UFSC-Team 98 apresentada na RoboCup'98, propõe um agente cognitivo concorrente. A cooperação entre os agentes tem por base as informações visuais e as estratégias de jogo pré-definidas, que especificam as ações a serem executadas pelo agente jogador.

Essa arquitetura possui três processos concorrentes chamados: *Interface*, *Coordinator* e *Expert*, que se comunicam através de um canal de suporte (*socket*), conforme ilustrado na Figura 4.1.

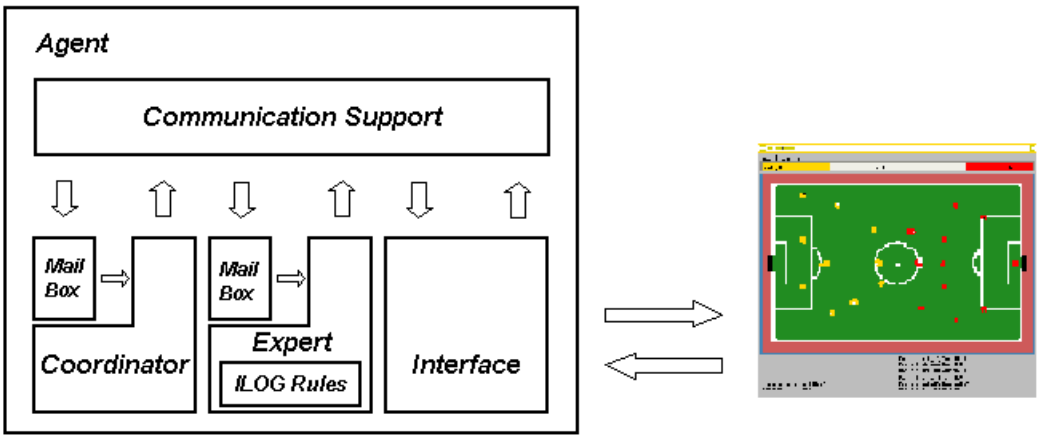


FIGURA 4.1 - Arquitetura do Agente UFSC-Team 98.

4.1.1 Componentes da Arquitetura do Agente UFSC-Team 98

Interface

O processo *Interface* realiza a comunicação com o Socceserver, através de troca de mensagens. O simulador envia ao processo *Interface* as informações visuais, auditivas (mensagens do árbitro ou dos demais agentes) e do corpo do agente. Essas informações são redirecionadas ao processo *Coordinator* para serem tratadas.

A função central do processo *Interface* está na manipulação das informações de percepção (vindas do servidor) e de ação (enviadas ao servidor). Trata-se de um processo reativo que transmite ao simulador a ação escolhida pelo agente.

Esse processo também possui outras funcionalidades, como: rotacionar o jogador ocioso e armazenar informações visuais.

Coordinator

O processo *Coordinator* tem por função realizar a comunicação (manipular as mensagens auditivas dos demais agentes e do árbitro redirecionadas pelo processo Interface) e conduzir o processo de cooperação entre os jogadores (baseado no ambiente *Expert-Coop*, como pode ser visto em [BIT 97]).

Esse processo pode receber as mensagens enviadas por outros processos da arquitetura pelo canal *socket*.

Expert

O processo *Expert* é responsável pelo comportamento cognitivo do agente. Esse processo envolve capacidades do agente como: planejar, tomar decisões, entre outras atividades.

O processo *Expert* possui um sistema baseado em conhecimento que armazena as informações sensoriais, mensagens do árbitro e de outros agentes do time. Com base nessas informações, permite a tomada de decisões apropriadas, atribuindo, ao agente, um comportamento específico de acordo com sua participação em "jogadas ensaiadas" (regras armazenadas na base de conhecimento).

Conforme [COS 98], o conhecimento do agente consiste em uma base de regras projetada para manipular as informações de percepção. O conhecimento social consiste em um conjunto de regras que armazena os diferentes papéis que um agente pode assumir, quando está envolvido em uma jogada.

O UFSC-Team 98 apresentou alguns problemas, tais como [COS 99a]: considerável tempo de resposta, com relação à sincronização do agente ao meio ambiente e complexidade do sistema baseado em conhecimento ao tomar as decisões do agente (tratam informações de alto nível como jogadas ensaiadas, até informações de baixo nível como o

valor da potência e direção do chute). A arquitetura do agente UFSC-Team foi migrada para uma arquitetura de agentes autônomos apresentada na Figura 4.2.

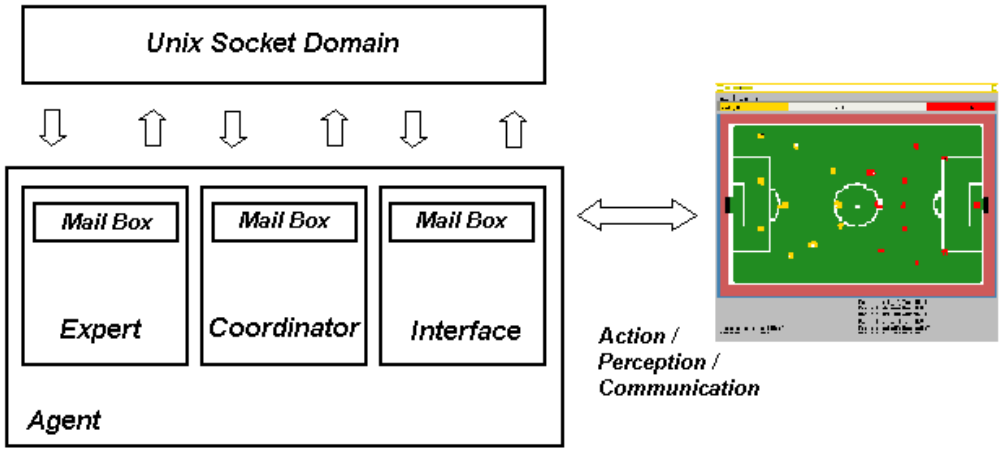


FIGURA 4.2 - Nova Arquitetura do Agente UFSC-Team.

A nova arquitetura, como descrito por [COS 99b], permite que o agente reaja aos estímulos do meio ambiente, fazendo planos, estabelecendo objetivos e realizando complexas estratégias concorrentes de cooperação de agentes em tempo real.

A arquitetura de agentes autônomos para o UFSC-Team mantém os mesmos três processos *Interface*, *Coordinator* e *Expert*, com os mesmos níveis de decisão (reativo, instintivo e cognitivo), onde cada um dos processos encapsulam diferentes motores de inferência, que são responsáveis por um dos três níveis de decisão [COS 99c].

Interface

O motor de inferência do nível reativo está presente no processo *Interface*. Esse nível mantém o mesmo objetivo apresentado anteriormente no UFSC-Team 98: recebe do Soccerserver as informações de percepção e envia ao Soccerserver a ação a ser executada pelo agente jogador em tempo real.

Segundo [COS 99c], nesta arquitetura, o nível reativo é composto pelos seguintes componentes: ***Mail Box***, ***Controlador Difusos***, ***Filtro de Entrada*** e ***Filtro de Saída***.

- O *Mail Box* é responsável pela recepção e pelo armazenamento das informações visuais enviadas pelo Soccerserver.
- O Controle difuso é responsável por um comportamento reativo específico, como por exemplo: Inicializar_Jogador, Saída_de_Bola, Chute_em_Gol, Driblar_Oponente, Conduzir_Bola_Avante, entre outros. Um controlador será ativado a cada momento, de acordo com as informações de percepção recebidas, determinando os comandos a serem enviados ao Soccerserver (situação que será efetivada).

Cada controle difuso tem seu próprio conjunto de variáveis lingüísticas. O filtro de saída é responsável por extrair, da informação de percepção, o respectivo valor que será usado pelo conjunto das variáveis lingüísticas de saída. O conjunto de controles, associados a cada agente, dependerá do grupo do qual o mesmo fará parte. Por exemplo: chutar a bola no gol oponente - para os jogadores do ataque e de meio campo.

A maioria dos controles difusos possuem quatro saídas: *kick_direction*, *kick_power*, *turn_moment*, *dash_power*. Os controles *pass_ball* e *kick_to_goal* possuem as saídas *kick_direction* e *kick_power*, já o *move_to_position* possui as saída *turn_moment* e *dash_power*.

Conforme apresentado em [COS 99b], os controles difusos permitem a sincronização de agentes (ajustando a relação entre entrada e saída) selecionando um dado controle capaz de satisfazer às necessidades do agente em tempo real. O controlador ativo será o comportamento mais adequado para cada situação de jogo.

- O Filtro de Entrada é responsável por extrair da informação de percepção os valores que serão atribuídos às variáveis lingüísticas que, posteriormente, serão utilizadas pelo controlador difuso ativo.
- O Filtro de Saída é responsável pelo controlador difuso de saída ativo e/ou combinação deles, seguindo alguns critérios como, por exemplo, saída nula, não é enviada ao Soccerserver. Quando existirem os comandos *turn* e *dash*, simultâneos, será enviado primeiro o comando *turn* com os respectivos valores e, após um atraso intencional, o comando *dash* é enviado ao simulador.

Coordinator

O motor de inferência do nível instintivo encontra-se no processo *Coordinator*. Esse nível é responsável por atualizar as variáveis simbólicas utilizadas para modificar a base de conhecimento usada pelo nível cognitivo e escolher um comportamento para o agente que seja adequado ao estado do jogo, de forma a alcançar uma meta local.

Uma meta local pode ser obtida através de uma seqüência de comportamentos a partir de um sistema especialista, contido no *Coordinator*. O referido sistema escolhe, a cada mudança de jogo, o comportamento reativo mais adequado. A meta local é estabelecida pelo nível cognitivo e isso determina o conjunto de regras a ser utilizado pelo motor de inferência.

Dada uma entrada (percepções), as regras (conjunto de regras usadas no motor de inferência) são capazes de reconhecer modificações no estado do jogo. O resultado da execução das regras pode ser qualquer mudança na base de conhecimento do nível cognitivo ou a seleção do controlador difuso mais apropriado no nível reativo, dirigindo o agente do estado corrente para o objetivo global.

Cada estado do jogo é definido por um conjunto de condições, dentre elas: informações de percepção de entrada e mensagens do árbitro vindas do processo de *Interface*. As informações visuais e do corpo do jogador são armazenadas em um *buffer* de

mensagens síncronas (*Sync Msg*), e as mensagens do árbitro e dos demais jogadores são armazenadas em um outro *buffer* destinado às mensagens assíncronas (*Assync Msg*). O sistema especialista é executado a cada atualização desses *buffers* (memória) ou quando um novo objetivo local é recebido do nível cognitivo.

O nível instintivo mantém-se monitorando as condições associadas ao comportamento e, se algum desses não for mais satisfeito, suas regras são usadas para inferir um novo comportamento reativo, especificando um objetivo e realizando as ações na direção da meta desejada.

Expert

O motor de inferência do nível cognitivo é implementado no processo *Expert*. Consiste em um sistema baseado em conhecimento simbólico e orientado a objetos, que é responsável por determinar o objetivo local e geral do agente. Esse nível trata as informações simbólicas recebidas do nível instintivo e as mensagens assíncronas recebidas dos outros agentes UFSC-Team.

O nível cognitivo escolhe o objetivo local e passa para o nível instintivo. Indiretamente essa escolha pode modificar as regras no motor de inferência no nível instintivo, que resultará em diferentes comportamentos reativos a serem selecionados.

O nível cognitivo é usado para realizar o planejamento de estratégias, os futuros objetivos locais de acordo com o resultado presente, e a especificação de pedidos cooperativos para alcançar o objetivo global. Esses pedidos serão tratados pelo processo *Coordinator*, resultando em objetivos locais compatíveis com o objetivo global desejado.

O processo *Expert* é composto por três bases: *KB Dynamic* (armazena informações simbólicas do nível instintivo e mensagens assíncronas), *Static KB* (armazena o conhecimento inferido pelo processo cognitivo sobre o jogo, por exemplo o tempo de jogo), *Export KB* (armazena as saídas do processo *Expert*, como os objetivos locais para serem enviados ao nível instintivo, informações usadas em estratégias de cooperação e mensagens enviadas para outros agentes UFSC-Team) [COS 99c].

O processo *Expert* também é responsável pela cooperação (utiliza a estratégia de cooperação chamada: Conhecimento Social Dinâmico), política de tempo-real e algoritmos de gerenciamento para comunicação de sistemas distribuídos [COS 99c].

4.2 A Arquitetura do Agente CMUnited

A arquitetura CMUnited, desenvolvida por Manuela Veloso, Peter Stone e Patrick Riley na Carnegie Mellon University, trata-se de um sistema multiagente em tempo real, onde agentes individuais cooperam entre si para atingir um objetivo comum, enquanto agem autonomamente [STO 2001]. Baseado num paradigma de agentes padrão, a arquitetura permite aos agentes perceberem e conhecerem o ambiente no qual estão inseridos e selecionarem uma ação para agir no mundo [VEL 97].

A arquitetura do agente CMUnited 98 é composta por três diferentes estados: o *World State*, o *Locker-Room Agreement* e o *Internal State*. Essa arquitetura também apresenta dois diferentes tipos de comportamento: *Internal Behavior* e *External Behavior*.

A estrutura dos comportamentos é similar, trata-se de conjuntos de condições e ações, onde as condições são expressões lógicas sobre as entradas, e as ações são os comportamentos do agente. Os dois tipos de comportamentos tratam-se de regras pré-estabelecidas de profundidade arbitrária, que resulta em tipos de comportamentos correspondentes à saída.

O *Internal State* é modificado pelo *Internal Behavior* e o *External Behavior* produz os comandos de saída (Figura 4.3).

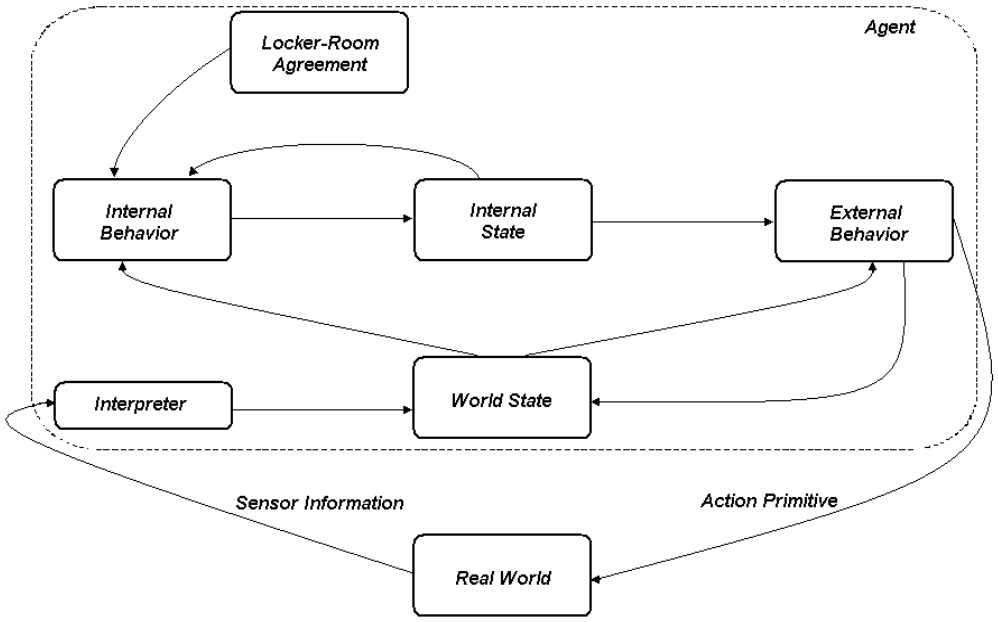


FIGURA 4.3 - Arquitetura do Agente CMUnited 98.

4.2.1 Componentes da Arquitetura do Agente CMUnited 98

World State

O *World State* reflete a concepção que o agente possui do ambiente no qual está inserido. O agente constrói o modelo do mundo com base em suas informações sensoriais e por efeitos prévios de suas ações. O estado do mundo pode ser alterado com o resultado da informação sensorial processada ou por meio de uma ação (de acordo com as previsões do módulo de *External Behavior*).

O agente possui uma memória prévia que contém uma representação possível do estado do jogo baseado nas observações passadas. Essa memória pode armazenar e modificar o conhecimento do agente na falta de informações. O agente, de posse das observações passadas, pode determinar a posição dos objetos que não estão visíveis no ciclo corrente.

Quando o agente recebe uma mensagem, ele interpreta-a e altera o *World State* para refletir a informação transmitida. Também armazena o conteúdo da mensagem em uma variável especial chamada *Last_ Message*.

Locker-Room Agreement

O *Locker-Room Agreement* é responsável pela sincronização do time em períodos de baixa comunicação. Esse módulo, por meio de jogadas pré-estabelecidas armazenadas, pode realizar uma nova sincronização quando for necessário, porém, geralmente mantém-se inativo.

Esse componente define a flexibilidade da estrutura do time, bem como, os protocolos de comunicação inter-agentes. O conceito de formações flexíveis presentes no *Locker-Room Agreement* consiste em papéis flexíveis, ou seja, os papéis são definidos independentemente do agente que os ocupa. Pelo fato dos agentes serem homogêneos (exceto o goleiro), eles podem mudar a sua função ao longo do tempo. Cada papel irá definir o comportamento do jogador em questão.

O CMUnited apresenta diferentes formações (defensivas e ofensivas). Uma definição completa de todas as formações faz parte do *Locker-Room Agreement*. Portanto, os agentes conhecem as posições de seus companheiros. Os jogadores possuem coordenadas de áreas fixas que indicam as regiões nas quais o agente pode se mover. Entretanto, o agente possui a flexibilidade de jogar em qualquer posição.

O *Locker-Room Agreement* pode ser usado para eliminar ou reduzir a necessidade de comunicação futura e também para incrementar a confiabilidade da comunicação. Os jogadores, para distinguir suas mensagens das mensagens de outro time, podem combinar um número de código com o qual todas as mensagens podem iniciar.

De acordo com [STO 99a], o *Locker_Room Agreement* trata-se de um compromisso entre os agentes comunicantes, onde agentes combinam sobre protocolos e decomposição de tarefas para usar durante períodos dinâmicos com comunicação restringida. Durante esses períodos dinâmicos, agentes confiam nos outros, seguindo o acordo.

Baseado no *Locker-Room Agreement*, o *Internal Behavior* atualiza o *Internal State* do agente. Se a mensagem de entrada requer uma resposta, variáveis no *Internal State* são manipuladas pelo *Internal Behavior.* Trata-se da resposta atual que pode ser dada pelo agente. Todas as variáveis são em seguida referenciadas pelo *External Behavior* que, então, determina quando a resposta pode ser enviada ao Soccerserver [STO 99a].

Internal State

O *Internal State* armazena as variáveis internas do agente. Esse módulo pode refletir o estado anterior do agente e o estado do mundo corrente, possivelmente como especificado pelo *Locker-Room Agreement*. Por exemplo: o papel do agente pode ser armazenado como parte do *Internal State*. O agente altera seu *Internal State* via *Internal Behavior*.

Os comportamentos são chamados no momento de alterar o *Internal State* do agente ou realizar uma ação no mundo. O processo de decisão do agente utiliza diversas variáveis definidas, como, por exemplo:

- *distance_to_their_goal*: a distância ao gol do oponente.
- *distance_to_our_goal*: a distância ao próprio gol.
- *closest_opponent*: a distância ao oponente mais próximo.
- *closer_to_goal*: indica o(s) companheiro(s) que está mais próximo ao gol oponente do que o agente que está com a bola.
- *can_shoot*: quando *distance_to_their_goal* < 25 and (*opponents_in_front_of_goal* <= 1 and *shot_margin* n <= 6)
- *shot_margin*: (*ball_to_goalie_cycles*) – (*goalie_to_ball_cycles*)
- *blocking_point*: o ponto na trajetória da bola, no qual o goleiro tem o melhor posicionamento.
- *ball_to_goalie_cycles*: número de ciclos que a bola leva para chegar ao *blocking_point*.
- *goalie_to_ball_cycles*: número de ciclos que o goleiro leva para chegar ao *blocking_point*.

Fazendo-se uso dessas variáveis, as decisões são tomadas seguindo o seguinte processo. Por exemplo:

- Se *can_shoot*: *kick*
- Se (*distance_their_goal* < 17 e *opponents_in_front_of_goal* <= 1) ou *shot_margin* <= 3: *kick_to_goal*

Internal Behavior

O *Internal Behavior* é alterado com base no *Internal State* corrente, no *World State* e no *Locker-Room Agreement* do time.

Entre os comportamentos interno e externo existe uma camada de aprendizado. O CMUnited utiliza uma hierarquia de comportamentos, através de uma máquina de aprendizado de vários níveis. Os níveis mais baixos contêm funções como interceptar e passar a bola; são comportamentos externos que envolvem ações diretas no ambiente. Os

comportamentos de alto nível, como estratégias de posicionamento e adaptação, são comportamentos internos e envolvem modificações no estado interno do agente.

O tipo de aprendizado usado em cada nível depende da característica da tarefa. Redes neurais e árvores de decisão são usadas pelo agente para aprender a interceptar e a passar a bola, respectivamente.

Exemplos de modos de comportamento são apresentados a seguir:

- *Localize*: encontra a localização do próprio campo se esta não é conhecida.
- *Face Ball*: encontra a bola e olha para ela.
- *Handle Ball*: usado quando a bola pode ser chutada.
- *Active Offence*: vai em direção a bola o mais rápido possível. Usado quando nenhum companheiro próximo está em poder da bola.
- *Auxiliary Offence*: fica pronto para receber um passe. Usado quando um companheiro próximo está em poder da bola.
- *Passive Offence*: move-se para a posição que parece ser útil ofensivamente no futuro.
- *Active Defence*: vai até a bola mesmo quando outro companheiro já está indo. Usado no fim do campo defensivo.
- *Auxiliary Defense*: marca o oponente.
- *Passive Defense*: segue o oponente ou vai para uma posição que parece ser útil defensivamente no futuro.

External Behavior

O *External Behavior* envia comandos para o Soccerserver, com base no *World State* e o *Internal State*. As ações do agente afetam o mundo e, dessa forma, alteram também a percepção futura do agente.

O agente CMUnited 99 usa a mesma aproximação para modelar o estado do mundo e as habilidades do agente de baixo nível, como o agente CMUnited 98, incluindo: chutar, driblar, interceptar a bola, tender ao gol, defender, e limpar jogada.

Como apresentado por [STO 99b], um desafio na criação e no uso de um agente autônomo complexo é permitir que suas escolhas de ações atendam às necessidades do mundo que muda dinamicamente, entendendo por que eles agem e como eles fazem.

O CMUnited 99, além de usar uma camada de aprendizado, uma estrutura de time flexível, e as habilidades do agente, apresenta, dentre as principais inovações [STO 2000b]:

- Módulo de treinamento *offline* (*Offline Training*), que possibilita a repetição de cenários de treinamento, como: habilidade de chutar. Esse módulo pode ainda verificar a medida de desempenho de um jogador em um chute ou em outras

atividades e passar instruções aos clientes (como os valores dos parâmetros de um comando);

- Introdução do conceito chamado "*layered disclosure*", no qual os agentes autônomos incluem, em sua arquitetura, a base necessária para permitir que revelem, quando requerido, o motivo explícito de suas ações. Uma pessoa pode requerer informações em qualquer nível de detalhe e até retroativo, mesmo quando o agente está agindo. Esse conceito foi introduzido já que um desafio em criar e usar agentes complexos e autônomos é seguir suas escolhas e ações em um ambiente que muda dinamicamente e entender por que eles agem e como agem.
- Otimização paramétrica para o modo de comportamento e condições de manejo da bola;
- Planejamento *on-line*;
- Modelagem do oponente e previsão.

Na modelagem do oponente, o agente deve ajustar efetivamente seu comportamento em resposta às ações do adversário. Deve ser capaz de classificar o comportamento do adversário em um grupo que determina qual mudança de estratégia é apropriada. Primeiro, é preciso definir características importantes do time para separá-lo em classes de comportamento. Então, usando essas características, pode-se definir as classes possíveis do comportamento do oponente [STO 2000b].

O agente treinador introduzido na RoboCup-99 ofereceu a oportunidade de recolher informações globais do oponente. O time faz uso do treinador *on-line* para fazer a classificação do oponente. O treinador passa a classificação para cada um dos jogadores. Cada agente, então, muda a estratégia de uma maneira pré-definida baseada na classificação do treinador.

4.3 A Arquitetura do Agente AT Humboldt 97

O principal interesse do time AT Humboult 97 em relação ao futebol artificial está em aplicar e estudar técnicas de IAD, SMA, tecnologias orientadas a agentes, e Raciocínio Baseado em Casos (*Case Based Reasoning* - CBR [LEA 97]).

O time desenvolvido na Humboldt-University of Berlin por Hans-Dieter Burkhard, Markus Hannebauer e Jean Wendler, possui os seguintes objetivos científicos com relação a sua participação na RoboCup:

- arquitetura do agente (arquitetura de *subsumption* (subsunção) e arquitetura BDI (*Belief – Desires – Intentions*));
- cooperação em SMA (cooperação emergente e melhoria de comunicação, negociação e planos de jogadas pré-definidas);

- aprendizagem de agentes (treinamento de capacidades e procedimentos de decisão - aprendizado *off-line* - e por adaptação ao comportamento oponente - aprendizado *on-line*);
- e decisão usando informações vagas (técnica especial para tratar informações incompletas ou vagas vindas do servidor, de fato, o AT Humboldt não chegou a utilizar CBR na RoboCup'97).

Conforme [BUR 97], o raciocínio baseado em casos permite aos agentes aprenderem com as experiências passadas, especialmente nas situações onde não há informações suficientes para introduzir uma regra.

O desenvolvimento estratégico global manteve-se em atenção ao interesse de utilizar a arquitetura de *subsumption* e de BDI (crenças, desejos e intenções para deliberação). Segundo [BUR 97], o modelo BDI é adequado ao conceito de agentes para o futebol, pois os seus componentes podem ser identificados no processo de planejamento realizado pelo agente AT Humboldt 97.

Alguns avanços para a competição em Nagoya (1997) conceberam especialmente estratégias de posicionamento *fuzzy*, posições de escanteio, de chute a gol e posições iniciais. O projeto foi implementado em C++ sobre as plataformas Solaris e Linux.

O jogador identifica as regras (goleiro, defesa, atacante, entre outros) por um número da camisa dado a ele pelo Soccerserver. O comportamento especial de cada jogador, de acordo com as regras, é portanto, devido a diferentes valores de parâmetros (como o número do jogador).

A Figura 4.4 apresenta a arquitetura do agente. As setas de espessura grossa indicam o fluxo de informação principal. Os losangos simbolizam componentes concorrentes. Um agente consiste de todos estes componentes.

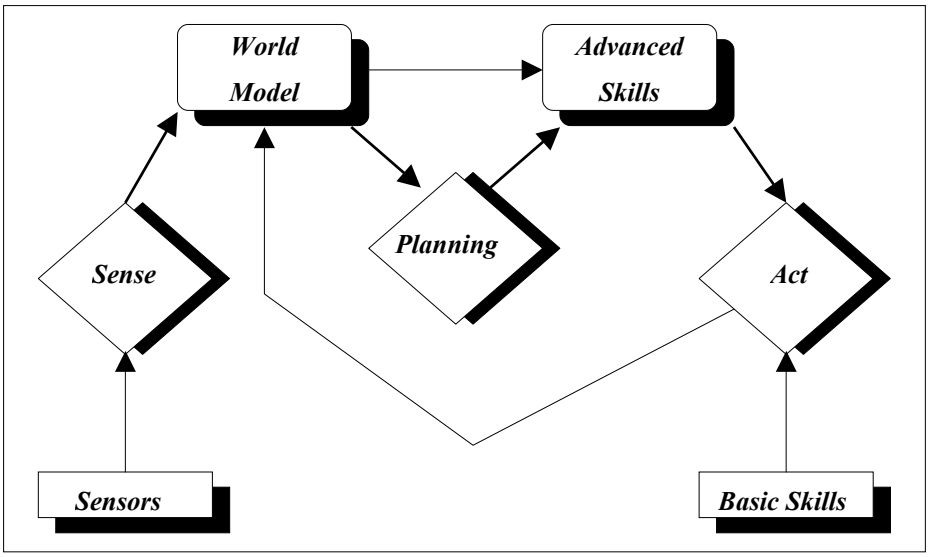


FIGURA 4.4 - Arquitetura do Agente AT Humboldt 97.

4.3.1 Componentes da Arquitetura do Agente AT Humboldt 97

Sensors e Basic Skills

O componente *Sensors* analisa gramaticalmente e transforma o código da *string* da informação sensorial que o jogador recebe do Soccerserver. O componente *Basic Skills* proporciona métodos para enviar os comandos básicos para o servidor.

World Model

O componente *World Model* (modelo do mundo) armazena um número ajustável de estados do mundo para manter um caminho histórico do jogador (memória).

Segundo [BUR 97], "*O futebol é um modelo consistente de um meio ambiente dinâmico e incerto"*. As informações imprecisas devem ser avaliadas e inferências são necessárias por informações perdidas. Isso leva a uma certa estabilidade de convicção do agente. Essa tarefa de correção de informações é realizada pelo componente *World Model.*

A posição absoluta do agente no campo é calculada usando a informação visual (a partir de linhas, *flags* e goleiras, considerando todos os casos possíveis). A velocidade do agente é estimada a partir dos comandos enviados para o servidor. Adicionalmente, o jogador grava a sua "energia".

Esse componente também procura prever situações futuras, usando o conhecimento das posições e das velocidades do jogador. Essa habilidade é extensivamente usada por *Planning* e *Advanced Skills* para estimar as conseqüências de comandos futuros. O *Advanced Skills* pode, por exemplo, acompanhar (simular) o objeto bola por algum tempo e monitorar a posição e a velocidade da mesma. Os aspectos adicionais, como vento, podem ser levados em conta.

O componente *Belief* (crença) trata-se do *World Model*. A estrita relatividade dos parâmetros do servidor conduz a uma imagem individual do mundo em cada agente. Adicionalmente, o agente não pode confiar na precisão dos dados recebidos e interpolados, por essa razão, ele apenas acredita (*belief*). As rotinas de atualização (incluindo a simulação de objetos não visualizados), a simulação de situações esperadas futuramente e o histórico das situações são também considerados como partes da "crença".

Planning

O componente *Planning* (planejamento) envolve o planejamento e o processo que conduz os planos a seqüências concretas de comando. Por exemplo: a direção do chute necessita algumas preparações (parar a bola, preparar a bola para chutar, finalmente chutar). Conseqüentemente, os jogadores devem aprender seqüências de ações parametrizadas, onde uma cuidadosa análise pode especificar um esqueleto para uma habilidade de sucesso, através de parâmetros ativados, de acordo com uma dada situação.

Um novo processo de planejamento é iniciado cada vez que chega uma nova informação sensorial. A situação é classificada da seguinte forma: se a bola está sob controle, o agente é capaz de passar a bola ou driblar. Se o jogador não tem o controle da

bola, ele pode decidir onde interceptá-la, olhar em volta ou correr para um certa posição. A procura do objetivo é feita por uma árvore de decisão usual. Algumas decisões são triviais ("A bola está no alcance do chute?") outras são difíceis ("Devo correr até a bola ou meu companheiro deve fazer isso?"). A decisão final é feita por uma simulação de alcançabilidade: se o agente supõe ser o primeiro de seu time a alcançar a bola, ele vai correr. Se não, ele confia nos companheiros e corre para a sua posição inicial.

Após a seleção de um objetivo, o jogador deve achar a melhor maneira de alcançá-lo. Essa fase do processo de planejamento produz um plano de longa duração com alguns planos parciais (*partial plans*). Os planos são baseados nas habilidades dos agentes. Isso significa que os planos estão em uma correspondência direta com *Advanced Skills*, que são blocos de construção para a execução de planos de longa duração. Tão logo o agente tenha decidido por um plano parcial, esse é fornecido ao componente *Advanced Skills* para que possa realizar a execução.

A cooperação entre os jogadores do time emerge da confiança no comportamento dos companheiros. O jogador confia no fato de que o companheiro mais próximo da bola tentará interceptá-la, quando esta passar em sua direção.

De acordo com [BUR 97], "*A cooperação emergente já tem dado muitos resultados excitantes: as ações dos jogadores são de acordo com suas expectativas e o comportamento de seu time. Regras diferentes resultam em um uso eficiente do campo inteiro. Mas, nós também temos identificado situações, onde comunicações podem melhorar o comportamento, porém isso ainda não foi implementado*".

O componente *Desires* (desejos) corresponde ao componente *Planning*. Trata-se de metas (desejos do agente) que são selecionadas de uma biblioteca fixa de objetivos. Diferentes objetivos (até opostos) podem ser alcançáveis, mas o agente seleciona apenas um deles (utilizando uma árvore de decisão).

Advanced Skill

O *Advanced Skill* (habilidades avançadas) é composto por planos pré-definidos. Esse componente possui uma biblioteca de habilidades que facilita o manejo da bola e de movimentos ótimos. A tarefa técnica desse componente é separar um plano de longa duração em planos de curta duração (seqüências concretas de comandos com todos os parâmetros). Esses planos de curta duração não são maiores que o intervalo entre informações sensoriais consecutivas. Desse modo, os planos de longa duração são executados pelas chamadas iteradas de *Advanced Skills* depois de cada informação de entrada.

A estratégia mais comum é adotar um plano de longo período durante a execução (se necessário), começando com algumas ações iniciais para alcançar uma situação bem definida. Depois, as ações são feitas na seqüência adotada de acordo com o plano, sendo assim, cada ação depende do sucesso da execução de seus predecessores.

Um processo estritamente novo pode começar a cada nova informação sensorial, ou seja, tem-se um novo plano começando exatamente no momento da chegada da nova informação. Como apresentado por [BUR 97], "*se nós necessitássemos sempre de ações iniciais (novas), então poderíamos nunca vir a dar continuidade a um plano. Para superar*

esse problema, o Advanced Skills é desenvolvido para lidar imediatamente com qualquer situação que possa aparecer no começo ou durante a execução de um plano de longo período".

Como efeito secundário, o *Advanced Skills* é capaz de conceber a maneira mais rápida de alcançar o objetivo de um modo muito flexível a situações arbitrárias de início.

O *Advanced Skills* usa *Basic Skills* para enviar comandos básicos ao servidor. As principais habilidades avançadas dos jogadores são: chute direto (*Directed Kick),* vai para a bola *(Go to Position)* e *Drible*. A habilidade *Go to Position* capacita o agente a alcançar todas as posições absolutas no campo, produz um *turn* (se necessário) e/ou dois ou três *dash*. Se requerido, esse procedimento evita obstáculos como outros jogadores. A habilidade *Drible* move a bola para uma certa direção sem perder o contato com ela, inclui a produção de vários *kick, turn* e *dash* combinados.

A habilidade *Directed Kick* é uma das vantagens competitivas do AT Humboldt 97. Essa habilidade permite ao jogador chutar a bola em qualquer direção com uma força escolhida (o quanto possível), trabalha com situações difíceis como altas velocidades e situações onde o próprio jogador é um obstáculo para a direção desejada. Se a direção e a velocidade desejada não podem ser alcançadas, essa habilidade procura combinar as exigências o melhor possível.

O componente *Intentions* (intenções) é composto por planos parciais para alcançar objetivos por ações apropriadas, identificadas na função do componente *Advanced Skills*. As ações são dividas em dois estágios. Inicialmente, a melhor possibilidade de alcançar um objetivo escolhido é computada e fixada como uma intenção. Isso corresponde a um plano de longo período com alguns parâmetros que podem ser vistos como planos parciais. Seu fim previsto é a realização do objetivo selecionado.

Portanto, a execução da intenção é separada em pequenos pedaços que são implementados como planos de curta duração no componente *Advanced Skills.* Esses planos de curta duração são "atômicos" e não podem ser interferidos por informações de entrada. Em conjunto eles formam um plano de longa duração suficientemente complexo para alcançar grandes objetivos. Há uma certa superposição com os procedimentos de decisão. Isso é necessário, a partir do momento em que o processo de decisão tem que olhar para os desejos alcançáveis. A realização da intenção baseia-se nas capacidades do agente que são implementadas no *Advanced Skills*.

A consideração desses agentes como construções BDI é apropriada, desde que, para cada nova informação de percepção, um processo de deliberação completo ocorra, atualizando as crenças, a escolha de um desejo, o compromisso com uma intenção e a execução de um plano [BUR 97].

O problema surge do fato de que esse compromisso com intenções é, em grande parte, executado independentemente de intenções anteriores. Isso pode contradizer o princípio da estabilidade. O processo de deliberação deve manter intenções antigas, enquanto não há sérias indicações opostas.

Pelo fato de um novo processo de deliberação iniciar cada vez que chega uma nova informação sensorial e novos planos serem criados, a estratégia de planejamento tem que assegurar a estabilidade de planos de longo período e evitar uma troca constante de

objetivos ou intenções. Portanto, deve-se garantir essa estabilidade do objetivo, até em uma nova inicialização de um completo processo de planejamento, normalmente os mesmos passos são escolhidos, pois a situação não muda radicalmente. Isso significa que os mesmos objetivos e intenções são também criados [BUR 99].

4.3.2 Sincronização Precisa

É essencial manter-se sincronizado com o servidor, visto que se trata de um ambiente de tempo real. Dessa forma, alguns componentes da arquitetura executam em paralelo. No AT Humboldt 97, os componentes *Sense* e *Act* são executados concorrentemente com a rotina principal para permitir um processo de raciocínio sem perturbações e manter-se em sincronia com o servidor.

Adicionalmente, um manejador pega uma mensagem a cada 100 ms no alto da fila de comandos e inicia o ciclo de simulação do *World Model.* Se a informação de tempo de uma informação de chegada difere do tempo interno do agente, ciclos de simulação adicionais são feitos até que a sincronia seja alcançada. Isso é essencial, pois a estimativa da velocidade do jogador depende somente dos comandos enviados e do tempo passado.

4.4 A Arquitetura do Agente F.C. Portugal Team

O time F.C. Portugal Team, campeão da RoboCup'2000, foi desenvolvido nas Universidades do Porto e de Aveiro, por Luís Paulo Reis, José Nuno Lau e Luís Seabra Lopes. Esse time apresenta, dentre as suas principais características, uma nova pesquisa para a coordenação de agentes, utilizando o Posicionamento Estratégico Baseado em Situação (*Situation Based Strategic Positioning* - SBSP) e Trocas de Papéis e Posicionamento Dinâmico (*Dynamic Positioning and Role Exchange* - DPRE). Além de outras técnicas de coordenação em SMA, algumas técnicas de domínios mais específicos e situações baseadas em seleção de Ação de Ataque (*Attacking Action Seletion*), em seleção de Ação de Defesa (*Defending Action Selection*) e em Posicionamento *Playoff* (*Playoff Positioning*).

Segundo [REI 00], "*estas técnicas são eficientes na coordenação de um time homogêneo de agentes que possuem conhecimento prévio comum e que devem alcançar, a cada momento, um conjunto limitado de tarefas dinâmicas em um meio ambiente dinâmico".*

O conhecimento do agente é formado pela percepção e pela previsão do estado do mundo. As habilidades de baixo nível são baseadas no agente CMUnited 99, assim como as modificações no estado do mundo e a previsão das ações. Foram adicionadas ainda, informações de situação, seleção da ação de ataque e de defesa e o uso da comunicação para transmitir o estado do mundo entre os jogadores, procurando obter um estado do mundo mais correto.

No módulo de decisão de alto nível, são realizadas as verificações à respeito da próxima ação a ser efetuada pelo agente. Várias árvores de decisão e conhecimentos sobre a formação, sub-formação, protocolos de comunicação e situações de jogo são usados para proporcionar a decisão de alto nível do agente.

O F.C. Portugal Team, concentrou esforços na capacidade de decisão inteligente de alto nível e habilidades sociais (coordenação e comunicação).

Conforme [REI 2001], o *loop* do agente inicia pelo módulo de decisão de formação corrente. Esse módulo é diretamente conectado com o segundo módulo, que se preocupa com a análise da situação do jogo. O *loop* de controle segue com a Troca de Posicionamento Dinâmico (*Dynamic Positioning Exchange*) e a análise da comunicação. Se o jogo estiver parado, o posicionamento *playoff* é realizado. Se uma situação crítica não é identificada, significa que o agente não necessita imediatamente ativar uma ação. Dessa forma, uma situação global, baseada em posicionamento estratégico, é realizada. Se a situação é crítica e o agente tem a bola, o movimento de ação da bola é selecionado, caso contrário, uma ação de defesa apropriada é feita.

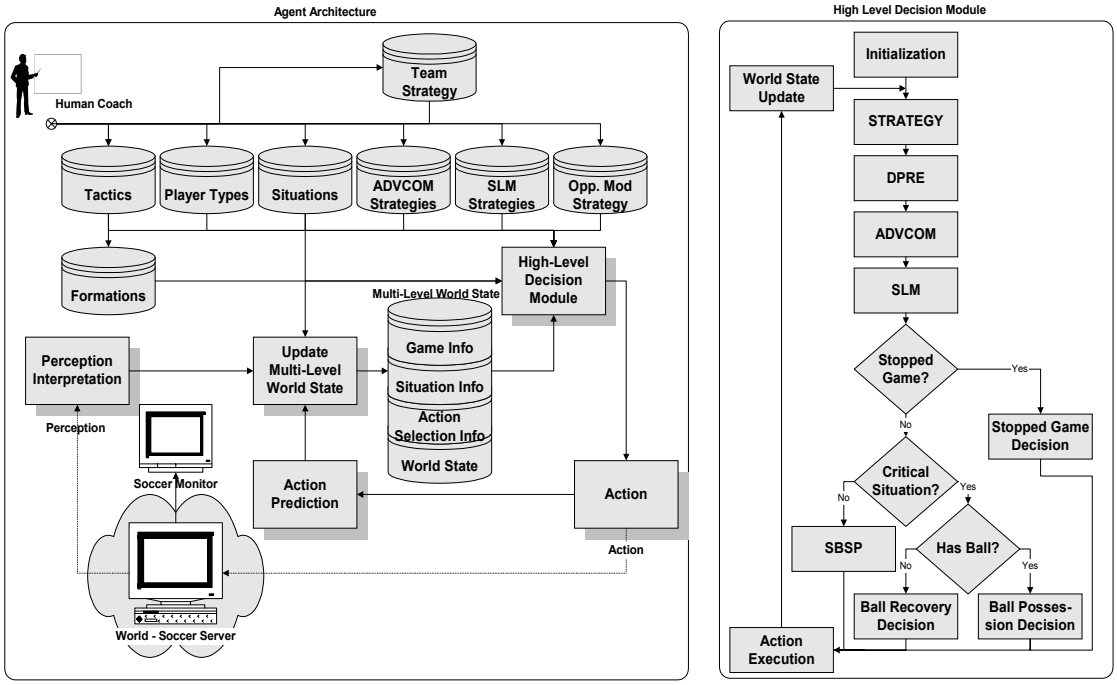


FIGURA 4.5 - Arquitetura do Agente F.C. Portugal Team.

4.4.1 Componentes da Arquitetura do Agente F.C. Portugal Team

Formações, sub-formações e análise da situação

Neste módulo, o agente usa um conjunto de regras pré-definidas, informações de alto nível de jogo (como o resultado e o tempo) e, se disponível, informações do treinador para decidir qual é a formação corrente. A informação de formação inclui um conjunto de posições flexíveis que podem ser modificadas de acordo com as informações de baixo nível do jogo (como a posição corrente da bola e dos jogadores).

Este módulo é diretamente conectado com o segundo módulo do *loop* de controle principal, que está encarregado da análise da situação do jogo. Essa análise inclui saber quem está atacando e a identificação da situação de jogo específica (como *free_kick*). Para cada situação uma sub-formação diferente é utilizada.

Posicionamento estratégico baseado em situação

O SBSP é uma das principais inovações do F.C. Portugal Team. Em situações não críticas, os jogadores procuram se posicionar estrategicamente. Essas posições são determinadas dinamicamente, usando as informações do jogo e disponibilizando os papéis de todos os agentes do time. Desse modo, mesmo quando não há informação de percepção de entrada, um dado jogador, através da análise detalhada da situação, pode saber com boa aproximação todas as posições dos companheiros no campo.

Segundo [REI 2001], "*isto é muito similar com o que acontece no futebol real em que todos os jogadores, analisando a situação de jogo, são capazes de identificar situações conhecidas e adivinhar as posições dos companheiros de acordo com essa situação*". Essa técnica assegura que os jogadores estão bem distribuídos em campo.

Troca de papéis e posicionamento dinâmico

O DPRE permite o posicionamento e a troca de papéis entre os agentes. Isso é possível, porque todos os agentes são homogêneos (exceto o goleiro). Se, em uma dada situação, uma troca de posições e papéis entre dois agentes proporcionar o aumento do desempenho do time, então, essa troca é realizada e comunicada a todos os agentes. De acordo com [REI 2001], "*embora a coordenação seja mais rápida usando a comunicação auditiva, a DPRE não necessita de comunicação para alcançar a coordenação global*".

Seleção de ação ativa

Existem dois módulos de Seleção de Ação Ativa, são eles: Controle de Bola e Seleção da Ação de Disputa pela Bola. O módulo de Seleção de Ação de Controle de Bola é usado apenas quando o agente tem a posse da bola. Ele permite que o agente decida a melhor ação a ser realizada. Essa seleção é baseada no conceito de matriz de passe, de matriz de chute e de matriz de drible (*Pass Matrix*, *Shoot Matrix* e *Drible Matrix*).

A matriz de passe ilustra as características de vários tipos de passes por cada um dos jogadores do time. Dentre os tipos de passes inclui: passes diretos para o jogador de

destino, passes ao longo de linhas de baixo congestionamento, passes para pontos de baixo congestionamento perto do jogador de destino e passes para a posição esperada do jogador.

As matrizes de chute e de *drible* apresentam características de vários tipos de chutes (para o meio do gol, escanteio e outros) e *dribles*. A decisão final da ação ativada a ser executada é desempenhada usando uma árvore de decisão.

A ação de disputa da bola refere-se às ações realizadas para tentar recuperar o controle da bola. Isso é conseguido com a seleção de uma ação quando o agente não tem a bola e a situação merece a atenção dele. O agente é motivado pela proximidade da bola, por situação de perigo em seu próprio gol ou em gol oponente, por uma ação coordenada ou pela execução de um plano.

Semelhante ao módulo SBSP, a seleção da ação deve ter conhecimento que o agente possui energia e capacidade de força limitadas, e que não é aconselhável que o agente perca capacidade de reposição para desempenhar uma ação em uma situação não crítica. O método de controle do F.C. Portugal Team de energia executa uma análise de situação que associa diversos limites de energia com várias situações críticas ordenadas hierarquicamente.

A ação de defesa inclui vários tipos de comportamentos de intercepção como, traçar a trajetória da bola, do oponente, proteger o gol e marcar a linha de passe. Antes da ação de defesa ser analisada, uma estrutura é gerada, incluindo posição presente e futura e a informação de ação.

Modelando o oponente

O módulo do modelo do oponente é distribuído entre o treinador (alto nível) e os jogadores (baixo nível). Esse módulo analisa o jogo e usa uma máquina de aprendizado associada com algumas heurísticas simples para aprender várias características do time oponente. Essas características incluem, por exemplo: o número do goleiro, a posição do goleiro, a direção e a velocidade do chute à gol, a formação global do time oponente e a previsão das posições futuras dos jogadores do time oponente.

O F.C. Portugal Team procura aprimorar, em seus trabalhos futuros, o desenvolvimento das técnicas de codificação de comunicação, o aumento das habilidades de baixo nível do agente (chute e *drible*), o desenvolvimento de novas técnicas de modelagem do oponente, desenvolvimento de planos locais para situações específicas, uma análise mais exaustiva de situações de *playoff* e o desenvolvimento de uma nova técnica de coordenação para permitir que dois agentes tenham, ao mesmo tempo, mesmo posicionamento e papel (chamada Cobertura Dinâmica – *Dynamic Covering*) [REI 2001].

Obviamente, tais arquiteturas têm sido pesquisadas e aprimoradas com o intuito de avançar o conhecimento sobre áreas de pesquisas em SMA. A possibilidade de estudar esses times serviu como base e visão sobre o assunto, facilitando a compreensão para o desenvolvimento de uma nova arquitetura de um agente para o Soccerserver.

Existem ainda outros times próprios para o estudo e para servirem como base no desenvolvimento de um time inicial (como: [DOR 99], [KOS 99]). Pode-se citar aqui a arquitetura ISIS (*ISI Synthetic*), um dos primeiros times a participar na categoria de simulação da RoboCup (participou da RoboCup97, onde conseguiu o terceiro lugar), desenvolvido na ISI (*University of Southern California's Information Sciences Institute*). O agente ISIS apresenta um modelo de time baseado em combinação de intenções. Detalhes sobre essa arquitetura podem ser obtidos em [TAM 99].

O time Erika 99 é outro exemplo. Esse time apresenta características como arquitetura BDI, pré-planejamento, modelo do mundo, previsão [MAS 99]. O agente Erika 99 executa os seus comportamentos conforme as suas intenções, repetidamente, até alcançar seus desejos.

5 Uma Arquitetura Cliente para o Soccerserver

Este capítulo relata a proposta de uma arquitetura para um agente (jogador), baseada em conceitos e estudos realizados nos Capítulos 2, 3 e 4 e em inúmeras discussões no grupo de trabalho. Essa proposta serviu de base para a elaboração concreta de um time de futebol para o simulador Soccerserver, proporcionando um primeiro passo em direção ao desenvolvimento de um time implementado na UFRGS.

Inicialmente, é introduzida a arquitetura do agente individual, possibilitando uma visão geral do jogador. Após, tem-se uma descrição mais detalhada de cada módulo que compõe a arquitetura. Segue-se com uma descrição da comunicação e da estrutura concorrente do jogador. Finalmente, são apresentados um exemplo do funcionamento do agente e algumas considerações sobre o agente.

5.1 Visão da Arquitetura do Agente Individual

A idéia principal do time UFRGS é desenvolver agentes jogadores que demonstrem um nível considerável de competência para a realização de suas tarefas, tais como: percepção, ação, cooperação emergente, estratégias pré-definidas, decisão e previsão.

O time UFRGS consiste de onze agentes, onze programas independentes. Cada agente distribuído constrói um modelo do estado do mundo corrente e, baseado em um conjunto de comportamentos, escolhe uma ação apropriada para realizar no campo de futebol. Através de ações autônomas, cada agente contribui para alcançar o objetivo do time.

As arquiteturas, descritas no Capítulo 4, foram estudadas e analisadas, contribuindo para uma melhor definição dos elementos e das características que um cliente para o simulador deveria apresentar, facilitando a elaboração do jogador.

Após o estudo e a análise dessas arquiteturas, com a observação direta dos times desenvolvidos para o Soccerserver, estabeleceu-se a proposta da arquitetura, contemplando algumas características apresentadas pelos clientes que foram estudados [PAG 97]. Idéias como: modelo do mundo, memória e Arquitetura de Subsunção, foram incorporadas à arquitetura do agente UFRGS.

5.1.1 Características da Arquitetura

O agente UFRGS projetado é dotado das seguintes características:

- Apresenta uma arquitetura de um agente autônomo concorrente.
- Trata-se de uma arquitetura de um agente reativo. No entanto, a arquitetura proporciona aspectos mais elaborados, como, memória e previsão.
- Apresenta agentes homogêneos comunicantes. Esses agentes contêm a mesma estrutura interna, incluindo objetivos, domínios de conhecimento, ações possíveis e o mesmo procedimento de seleção entre suas ações. Cada agente possui uma função (ou papel) específica, de forma que há jogadores que pertencem ao ataque, outros à defesa e ainda outros ao meio de campo, além do goleiro. No entanto, as arquiteturas são iguais.
- Realiza a seleção do comportamento a ser efetuado baseado na Arquitetura de Subsunção, de acordo com a prioridade do comportamento para o estado do mundo corrente.
- Possui variáveis de controle de alto nível, que compõem as regras de comportamento do agente.
- Apresenta um modelo do mundo e um modelo do próprio jogador, baseado nas percepções visual, auditiva e informações do corpo do agente.
- Possui memória, isso implica dizer que o agente pode relembrar os *t-2* estados do jogo (estados anteriores).
- Permite a realização de previsões baseadas nas ações de saída e no modelo que o agente tem do mundo.
- Apresenta controle sob as restrições impostas pelo simulador, como por exemplo: o turno dos comandos de ação é respeitado, o chute só é permitido dentro da área de chute do jogador, o jogador, no caso o goleiro, não tentará pegar a bola se a mesma estiver fora da sua área de ação.
- Permite que comportamentos não disponíveis nos comportamentos elementares do agente possam ser implementados através de programação convencional. Novos comportamentos podem ser facilmente acrescentados de forma direta no módulo de ações do agente. Para isso, é necessário apenas conhecer o conjunto de variáveis de controle.

5.1.2 A Arquitetura do Agente UFRGS

O desenvolvimento de um agente para o simulador Soccerserver possibilita a aplicação e avaliação de SMA, desde a elaboração da arquitetura do agente, definição das habilidades individuais, escolha do processo de seleção de ações, além de explorar e visualizar a cooperação emergente entre os jogadores. Também, proporciona a integração

de diversas tecnologias, começando pela comunicação do cliente com o servidor, realização de processamento concorrente e controle em tempo-real.

O objetivo do trabalho é realizar a modelagem de uma arquitetura cliente para o simulador, aplicando aspectos fundamentais de SMA. A implementação do agente permitiu a visualização do funcionamento do mesmo.

A arquitetura geral de um agente individual proposta é apresentada na Figura 5.1. Pode-se identificar quatro módulos principais: Interfaces de Entrada e de Saída, Percepção do Mundo e Escolha da Ação.

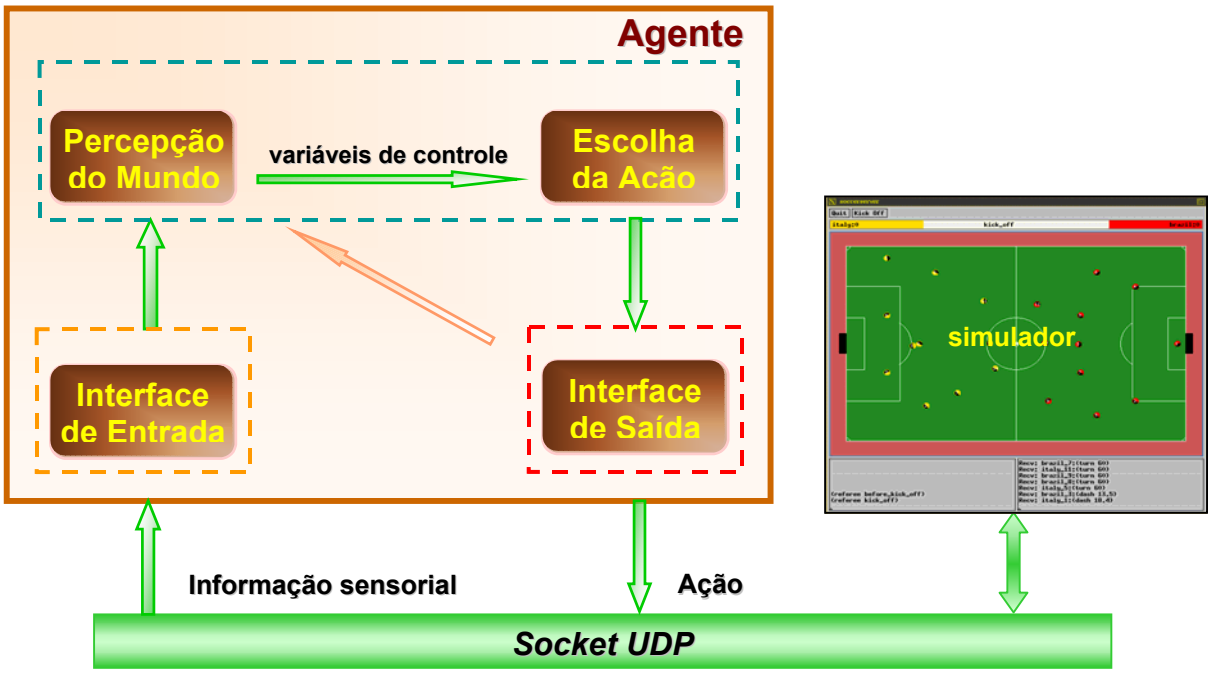


FIGURA 5.1 - Arquitetura do Agente Individual UFRGS.

Essa arquitetura apresenta um SMA capaz de agir em um ambiente de tempo real, onde agentes individuais buscam cooperar entre si para atingir um objetivo comum, enquanto agem autonomamente. A arquitetura permite aos agentes perceberem e conhecerem o ambiente no qual estão inseridos e selecionar uma ação para agir no ambiente. Uma das prioridades dessa arquitetura é possibilitar que o agente possa reavaliar constantemente e rapidamente suas tarefas no campo de futebol, adaptando-se ao seu meio ambiente conforme necessário, de acordo com as mudanças do mesmo.

Os agentes constroem um modelo do mundo baseado em suas percepções que, associado a um conjunto de comportamentos, estabelece uma ação apropriada para o estado do mundo corrente. Esse modelo que o agente tem, é atualizado a cada nova informação de percepção recebida do servidor.

5.1.3 Componentes da Arquitetura do Agente UFRGS

Interface de Entrada e Interface de Saída

A Interface de Entrada e a Interface de Saída realizam toda a comunicação, recebendo mensagens (percepções) e enviando mensagens (ações) para o Soccerserver, respectivamente. Tanto a interface de entrada, quanto a de saída, são executadas concorrentemente com o processo principal, permitindo ao agente um processamento sem perturbações e mantendo a sincronia com o servidor.

Busca-se, com a concorrência, obter um melhor desempenho do agente em tempo real, visto que alguns componentes da arquitetura, como a Interface de Entrada e Saída, são executados "paralelamente".

Como apresentado pela Figura 5.2, a Interface de Entrada e a Interface de Saída proporcionam a comunicação entre o simulador e o agente. O processo Interface de Entrada realiza todo um processamento da informação sensorial enviada pelo Soccerserver e prepara essa informação para ser utilizada pelo módulo chamado Percepção do Mundo.

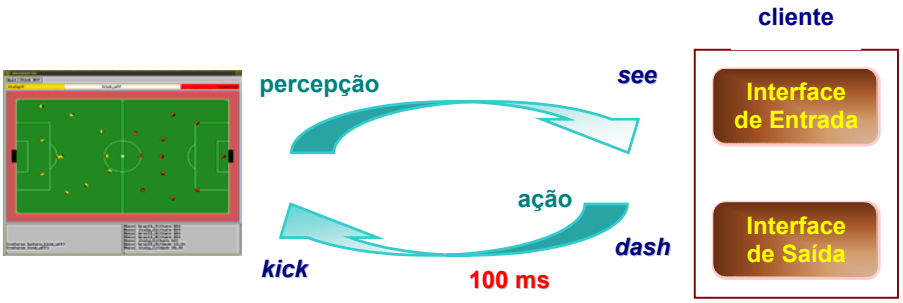


FIGURA 5.2 - Interface entre o Simulador e o Agente.

O processo de Interface de Entrada gerencia o andamento das informações de entrada de maneira ordenada. As mensagens recebidas são colocadas em uma fila e, conforme a demanda, são disponibilizadas ao processo principal. Essa fila contém, no máximo, três *slots*, suficientes para manter em espera as mensagens que não conseguem ser imediatamente processadas. Observou-se que um número maior de *slots* poderia acarretar o armazenamento de informações ultrapassadas, visto o domínio dinâmico do simulador.

Existe um *buffer* compartilhado entre o processo de entrada e o processo principal. O processo Interface de Entrada é responsável por monitorar e atualizar este *buffer*. As informações armazenadas na fila são adicionadas no *buffer* uma a uma, sempre que o mesmo estiver disponível.

Dada uma informação de percepção, o processo Interface de Entrada acomoda a informação na última posição vazia da fila. Esse mesmo processo disponibiliza a informação no *buffer* compartilhado, para que possa ser utilizada pelo módulo Percepção

do Mundo, como apresentado na Figura 5.3. São armazenadas na fila as informações visuais e auditivas.

As mensagens enviadas pelo árbitro possuem maior importância, isso deve-se ao fato dessas estarem associadas às mudanças do modo do jogo, exigindo uma atitude do jogador, diferente da adotada quando a bola está em jogo (*play_on*). Portanto, supondo um caso extremo, quando chega uma mensagem do árbitro e a fila está cheia, nesse caso, um comando *see* poderá ser suprimido em favor do comando *hear*.

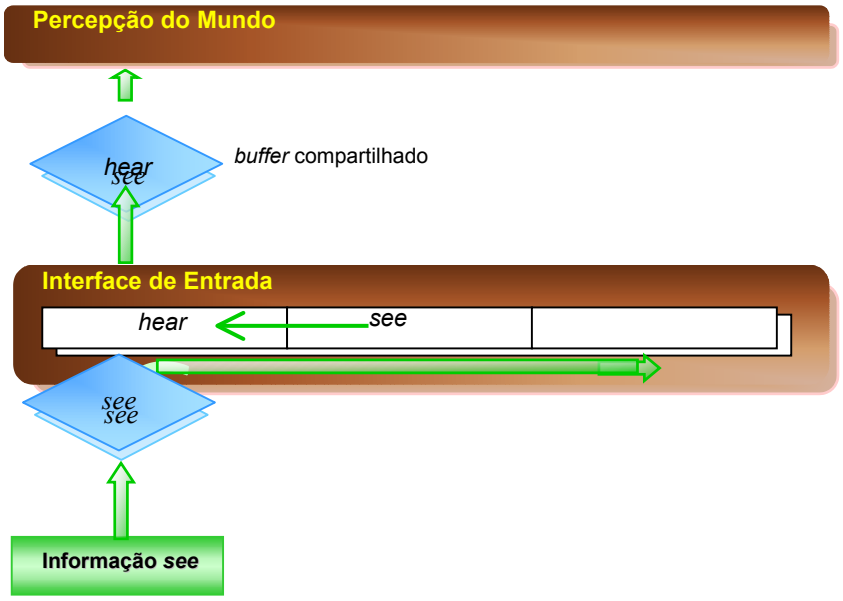


FIGURA 5.3 - Funcionamento do Processo Interface de Entrada.

A Interface de Saída envia ao Soccerserver a ação escolhida pelo agente. Esse processo pode, ainda, disponibilizar ao módulo Percepção do Mundo a última ação enviada ao servidor, possibilitando a realização de previsões quanto ao estado do mundo no próximo ciclo. Por exemplo: uma ação de giro pode resultar na atualização da direção dos objetos estacionários e da bola, que estão visíveis pelo jogador no ciclo corrente. No entanto, a atualização dos objetos móveis, envolve uma série de cuidados, visto que esses objetos podem estar em constante movimento.

Realizar uma previsão quanto ao estado do mundo, pode também possibilitar ao jogador que não recebeu nenhuma informação durante um período de tempo, efetuar uma ação com base nas informações passadas (de sua memória) e na sua última ação. Com isso, garante-se que o agente estará em condições de agir constantemente.

O simulador recebe comandos de ação de clientes distribuídos durante um ciclo de 100ms e, então, modifica todo o estado do mundo ao final do ciclo. Algumas restrições impostas pelo simulador são consideradas pelo módulo de Interface de Saída, como por exemplo, respeitar o critério do simulador em que apenas um comando de ação (dentre eles:

turn, *kick*, *dash*) deve ser enviado por ciclo de simulação. Sendo assim, não mais do que uma ação é mandada pelo agente em um mesmo ciclo, evitando-se com isso que ações sejam perdidas ou desconsideradas pelo simulador.

A perda de uma ação poderia acarretar vantagens aos jogadores oponentes que agissem durante o ciclo. Para prevenir isso, o processo de Interface de Saída deve proporcionar a sincronização adequada entre o agente e o Soccerserver.

Um *clock* interno ao agente é acionado sempre ao final de um ciclo. O *clock* indica o momento no qual uma ação poderá ser enviada ao servidor. Esse assunto será melhor abordado no item 5.2 Comunicação.

Percepção do Mundo

A mensagem recebida do processo Interface de Entrada é analisada gramaticalmente (*parser*) no módulo Percepção do Mundo. As informações presentes na mensagem sobre cada objeto em campo, visível pelo agente, são armazenadas no Modelo do Mundo. O estado do jogo é atualizado a cada nova informação de percepção.

Esse módulo também realiza a análise gramatical das mensagens auditivas e do comando *sense_body* (além do *error* e o *ok*), armazenando as informações no modelo que o jogador tem de si.

O *parser* (interpretador) tem por característica analisar a mensagem de entrada, classificando os parâmetros dessa mensagem. Ele reconhece a natureza da mensagem (visão, audição e sentidos do corpo) e identifica informações como: tipo do objeto, distância, direção, entre outras. A Figura 5.4 apresenta uma mensagem visual de entrada (*see*) e a decomposição da mesma após o *parser*.

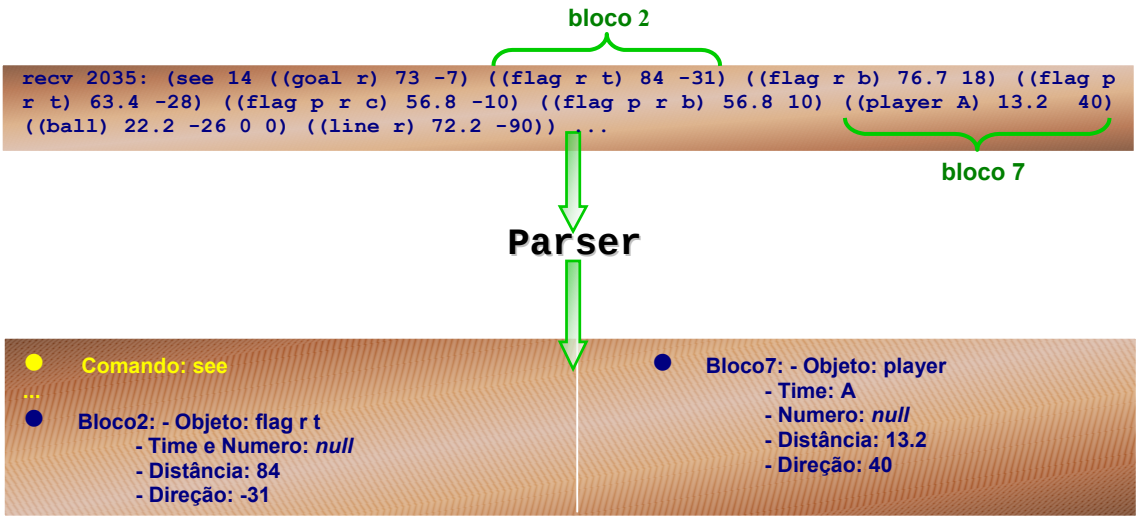


FIGURA 5.4 - Interpretador de Mensagens.

O processo Percepção do Mundo também abriga a memória do jogador (Figura 5.5). Essa memória permite ao jogador relembrar o estado do jogo nos últimos três ciclos. Essa informação é usada, quando necessária, como por exemplo, se a bola não está visível no ciclo corrente, a última informação conhecida da bola pode ser usada, ou ainda, combinações entre as informações durante os três últimos ciclos podem verificar se um jogador (companheiro ou adversário) está se aproximando da bola.

Outra característica da memória é permitir que interpolações sejam feitas, no caso, poderia ter-se a informação da bola em *t-2* e em *t* e descobrir em *t-1*.

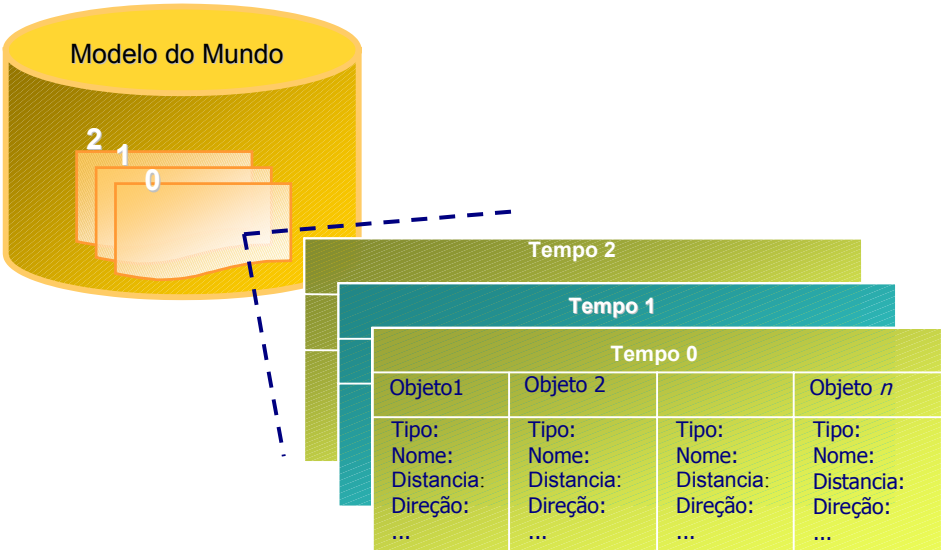


FIGURA 5.5 - Modelo do Mundo.

O jogador ainda mantém um modelo de si. Esse modelo contém as características do próprio jogador, muitas delas fornecidas pelo comando *sense_body*, como energia, velocidade e quantidades de ações realizadas. Além disso, o modelo armazena a posição e a direção absoluta do jogador, nome do seu time, nome do time adversário, o papel do jogador, tempo de jogo (primeiro ou segundo tempo), mensagem do árbitro, lado de campo em que saiu jogando, entre outros.

Após a mensagem ser analisada e as informações de entrada serem armazenadas no Modelo do Mundo, o módulo Percepção do Mundo calcula os parâmetros de interesse do módulo posterior (Escolha da Ação). O resultado dos cálculos é comunicado ao módulo Escolha da Ação através de "variáveis de controle". Essas variáveis são criadas com base no conhecimento que o jogador possui do seu ambiente e de si, e desta forma, caracterizam-se por refletir o estado do mundo corrente do agente (Figura 5.6).

Condição	(vejo_a_bola) e (sei_a_distância_da_bola) e (KICKABLEAREA >= distância_a_bola)
Var. de Controle	POSSO_CHUTAR

Condição	(vejo_companheiro) e (vejo_adversário) e (sei_distância_direção_do_companheiro) e (sei_distância_direção_do_adversário) e (cálculo_distância_entre_dois_pontos) e (AREA_DE_MARCAÇÃO < distância)
Var. de Controle	COMPANHEIRO_DESMARCADO

Condição	(vejo_companheiro_desmarcado) e (vejo_gol_adversário) e (sei_distância_direção_do_companheiro) e (sei_distância_direção_do_gol_adversário) e (cálculo_distância_entre_dois_pontos) e (DISTÂNCIA_PERTO_DO_GOL_ADVERSÁRIO > distância)
Var. de Controle	COMPANHEIRO_DESMARCADO_PERTO_DO_GOL_ADVERSÁRIO

FIGURA 5.6 – Exemplo de Variáveis de Controle.

As habilidades disponíveis aos jogadores incluem ações como chutar, girar e caminhar. Para chegar à decisão de uma ação, utiliza-se um sistema baseado em regras. Trata-se de um conjunto de condições e ações, onde as condições são as variáveis de controle do agente. Os comportamentos, portanto, derivam de regras pré-estabelecidas (Arquitetura de Subsunção). Esses comportamentos representam a ação a ser realizada pelo agente, no sentido de atingirem uma solução para o estado do mundo corrente.

O agente UFRGS possui um total de 20 variáveis de controle. Tais variáveis são apresentadas na Tabela 5.1. Essas variáveis também possuem o objetivo de facilitar o desenvolvimento dos comportamentos.

TABELA 5.1 - Variáveis de Controle.

Váriaveis de Controle	*Descrição*	*Valores Possíveis*
1. BOLA_APROXIMA	verifica se a bola está se aproximando do jogador, recordando a posição da bola em tempos passados.	1: a bola aproxima 0: a bola não aproxima -1: não vê a bola
2. POSSO_CHUTAR	verifica se a bola está dentro da *kickablearea* do jogador.	1: vê a bola e pode chutar 0: vê a bola e não pode chutar -1: não vê a bola
3. MEU_LADO	verifica o lado do campo do jogador (lado em que saiu jogando).	1: lado de campo do jogador 0: lado de campo do adversário
4. ESTOU_DESMARCADO	verifica se o jogador está desmarcado (a área de marcação é de aproximadamente 3m).	1: jogador está desmarcado 0: jogador não está desmarcado -1: não vê adversário

TABELA 5.1 - Variáveis de Controle.

5. ESTOU_FORA_DE_ CAMPO	verifica se o jogador está fora de campo.	1: jogador está fora de campo 0: jogador não está fora de campo
6. COMPANHEIRO_ PERTO_DA_BOLA	verifica se existe algum companheiro perto da bola. Só serão considerados os companheiros que estão mais perto da bola do que o próprio jogador e que são visíveis pelo mesmo.	número: número da camisa do companheiro posição_modelo: posição do jogador no modelo do mundo (índice para a posição no modelo do mundo que contém informações sobre o jogador)
7. COMPANHEIRO_ APROXIMA_DA_BOLA	verifica se algum companheiro está se aproximando da bola.	número: número da camisa do companheiro posição_modelo: posição do jogador no modelo do mundo
8. COMPANHEIRO_PODE _CHUTAR	verifica se o companheiro pode chutar.	número: número da camisa do companheiro posição_modelo: posição do jogador no modelo do mundo
9. COMPANHEIRO_ DESMARCADO	verifica se o companheiro está desmarcado.	número: número da camisa do companheiro posição_modelo: posição do jogador no modelo do mundo
10. COMPANHEIRO_ PERTO_DO_GOL	verifica se o companheiro está perto do gol adversário. Só serão considerados os companheiros que estão mais perto do gol do que o próprio jogador.	número: número da camisa do companheiro posição_modelo: posição do jogador no modelo do mundo
11. ADVERSARIO_PERTO_ DA_BOLA	verifica se o adversário está perto da bola. Só serão considerados os adversários que estão mais perto da bola do que o próprio jogador.	número: número da camisa do adversário posição_modelo: posição do jogador no modelo do mundo
12. ESTADO_DO_JOGO	verifica o estado do jogo corrente.	*before_kick_off,corner_kick_l(r), free_kick_l(r), goal_kick_l(r), kick_in_l(r), kick_off_l(r),play_on, time_over, foul_l(r), goal_l(r), half_time,time_up, time_extended time_up_without_a_team,*
13.ADVERSARIO_ DESMARCADO	retornam todos os adversários que estão desmarcados.	número: número da camisa do adversário posição_modelo: posição do jogador no modelo do mundo
14. COMP_PERTO_DO_GO L_DESMARCADO	retornam todos os companheiros que estão mais perto do gol (em ordem de proximidade ao gol) do que o próprio jogador e desmarcados.	número: número da camisa do adversário posição_modelo: posição do jogador no modelo do mundo
15. COMP_PERTO_DA_ BOLA_DESMARCADO	retornam todos os companheiros que estão mais perto da bola (em ordem de proximidade a bola) do que o próprio jogador e desmarcados.	número: número da camisa do adversário posição_modelo: posição do jogador no modelo do mundo

TABELA 5.1 - Variáveis de Controle.

16. COMP_PERTO_DE_MIM	verifica se há algum companheiro num raio de 25m de distância.	número: número da camisa do adversário posição_modelo: posição do jogador no modelo do mundo
17. ADV_PERTO_DE_MIM	verifica se há algum adversário num raio de 25m de distância.	número: número da camisa do adversário posição_modelo: posição do jogador no modelo do mundo
18. COMP_DESMARCADO_PERTO_DE_MIM	retornam todos os companheiros desmarcados que estão num raio de 25m de distância.	número: número da camisa do adversário posição_modelo: posição do jogador no modelo do mundo
19. ADV_DESMARCADO_PERTO_DE_MIM	retornam todos os adversários desmarcados que estão num raio de 25m de distância.	número: número da camisa do adversário posição_modelo: posição do jogador no modelo do mundo
20. MEU_TIME_TEM_POSSE_DE_BOLA	verifica se algum companheiro (inclusive o próprio jogador) tem a posse da bola (ou seja, a bola esta na *kickablearea* de um companheiro ou do jogador).	1: se meu time tem a posse da bola 0: se meu time não tem a posse da bola

Esse módulo também realiza vários cálculos necessários ao jogador, como por exemplo, o cálculo da posição e da direção absoluta do jogador em campo. Embora uma grande concentração de esforços tenha sido aplicada para que estes valores fossem o mais corretos possíveis e bons resultados tenham sido obtidos, a inconsistência dos valores enviados ao jogador (como valores de distâncias), tornou esta tarefa difícil de ser solucionada.

O módulo Percepção do Mundo não toma decisões com respeito às ações do agente. Ao contrário, toda a decisão com relação às ações do agente é realizada com base nas variáveis de controle, pelo módulo Escolha da Ação.

Escolha da Ação

Num domínio dinâmico como o do simulador Soccerserver, a seleção das variáveis de controle, a escolha da ação e a execução dos comandos são fortemente dependentes do estado corrente do meio ambiente, como percebido pelo agente individual.

A reavaliação das condições do agente a cada momento evita que o mesmo possa criar ou dar continuidade a planos que não mais correspondam à realidade do agente. A própria incerteza das percepções pode ocasionar um planejamento inconsistente.

A natureza dinâmica do servidor justifica a utilização de um SMA com características reativas. Em um ambiente de tempo real, o processamento das informações é limitado por restrições de tempo. O sistema pode realizar uma decisão oportuna, entretanto,

se ele não consegue desfrutar do intervalo de tempo disponível para agir, podem ocorrer decisões tardias, que levam o agente a um comportamento sub-ótimo (por exemplo: perder a oportunidade de agir).

O processo Escolha da Ação, de posse das variáveis (condições), define uma ação (sub-objetivo) a ser executada pelo jogador, baseada em regras de decisão (plano reativo).

As regras especificadas para o agente guiam as ações dos agentes no processo de satisfação de seus comportamentos. Para especificar essas regras, utilizam-se comportamentos pré-definidos. Portanto, a atuação do agente no ambiente é modelada através de regras, que determinarão quais comportamentos do jogador devem ser executados e sob quais condições.

Cada comportamento equivale a um par condição/ação. Fazendo uso das variáveis de controle, criam-se regras como a apresentada na Figura 5.7.

Prioridade 1	
Condição	(posso_pegar) e (¬ estou_com_a_bola)
Ação	PEGAR a bola

FIGURA 5.7 - Par Condição/Ação do Comportamento do Agente Goleiro.

A prioridade da regra (Figura 5.7) corresponde à importância que a mesma tem para um determinado jogador. Por exemplo: um goleiro tem como sua tarefa mais significativa, pegar a bola. Esse comportamento deve sobrepor a todos os outros comportamentos do goleiro. Outro exemplo: o atacante tem como função principal chutar ao gol do adversário, conseqüentemente esse comportamento possui uma alta prioridade e, por esse motivo, é definido acima de outros comportamentos com menor prioridade, como marcar o adversário.

A condição (Figura 5.7), corresponde a testes condicionais sobre as variáveis de controle. Esta condição é responsável por acionar ou não, a execução de uma ação que será associada ao agente. A ação (Figura 5.7) corresponde a tarefa a ser desempenhada pelo agente.

Os comportamentos diferenciam-se conforme as responsabilidades de cada jogador. Portanto, as funções específicas de cada agente são distinguidas neste processo. A escolha das ações a serem realizadas pelo agente é detalhada através de uma árvore de decisão.

O exemplo ilustrado na Figura 5.7 pode ser observado na forma de árvore de decisão do agente goleiro, apresentada na Figura 5.8 (cujo comportamento será explicado na item 6.6.2 Comportamento do Agente Goleiro).

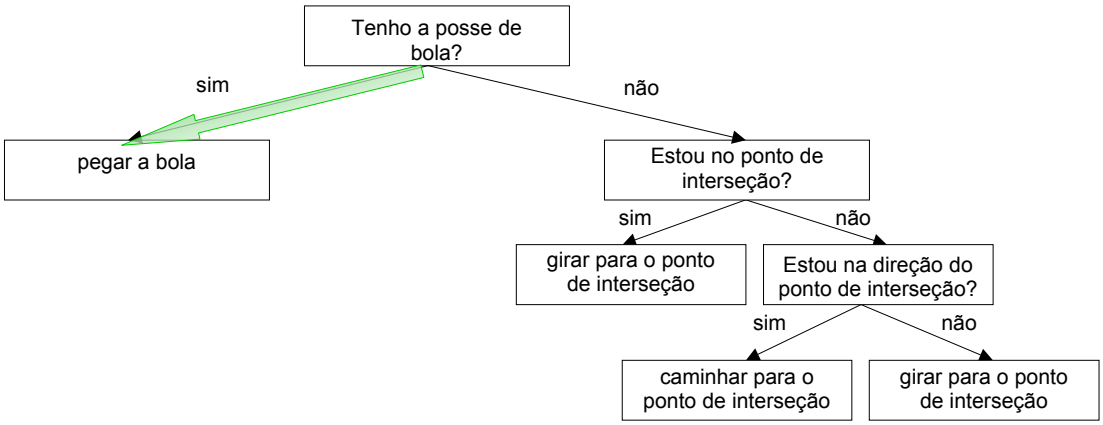


FIGURA 5.8 - Árvore de Decisão Resumida do Agente Goleiro.

Dentre as características da arquitetura do agente UFRGS destaca-se a utilização da Arquitetura de Subsunção de Brooks [BRO 91a]. Essa arquitetura estabelece prioridades entre a execução dos comportamentos atribuídos aos agentes. Ela é utilizada para hierarquizar os comportamentos do agente. Dessa forma, alternar, excluir ou adicionar um comportamento deve influenciar no processo de escolha da ação sem afetar os demais comportamentos.

As regras de decisão são criadas conforme a ordem de execução das ações. Por exemplo: um jogador, em uma cobrança de escanteio, deve ter como prioridade chutar a bola para o companheiro desmarcado mais perto do gol oponente; caso essa condição não seja satisfeita, ele deverá chutar a bola para o companheiro mais perto do gol oponente. Caso contrário, então, ele chuta para uma posição estratégica, onde algum dos seus companheiros possa, logo, ter o controle da bola, como apresentado na Figura 5.9.

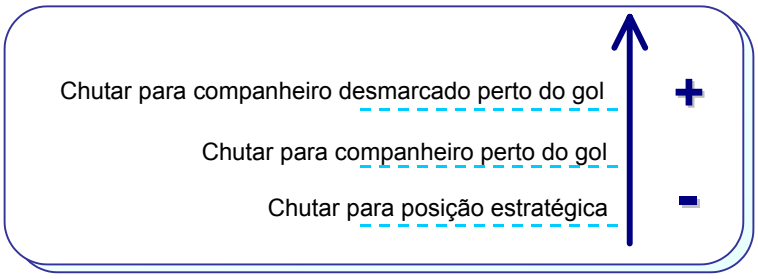


FIGURA 5.9 – Prioridade dos Comportamentos para um Escanteio.

Como apresentado em [BUR 97], a decisão de uma jogada é realizada por uma simulação de alcançabilidade: se o agente jogador supõe ser o mais próximo do seu time a

alcançar a bola, então ele vai correr até a bola. Caso contrário, o agente confia nos companheiros e procura se reposicionar.

Assim, a cooperação entre os jogadores emerge da confiança no comportamento dos companheiros, em função da percepção que o agente possui do ambiente e de seu conhecimento, ou seja, o comportamento emergente parte do conhecimento local de cada agente sobre o estado do ambiente. O jogador confia no fato de que o companheiro mais próximo da bola tentará interceptá-la, quando vier em sua direção.

A funcionalidade emergente permite que um sistema com componentes simples possa exibir um comportamento, como um todo, mais organizado do que o comportamento das partes individuais, evoluindo para um comportamento em grupo inteligente [FRO 97]. O próprio comportamento social resulta de regras individuais comuns.

Buscou-se, através de comportamentos elementares bem definidos, obter resultados eficientes que proporcionam o aparecimento de comportamentos e cooperação emergente entre os agentes. A partir de regras básicas, permitiu-se que os comportamentos mais elaborados venham a ocorrer, possibilitando um movimento final dinâmico ou, por vezes, inesperado.

A combinação de primitivas básicas seguidas, como: chutar, girar e caminhar, resultaram em um comportamento mais sofisticado, como o "drible" apresentado em [MAT 96]. A idéia é: o jogador deve ir alternando suas ações entre chutes, caminhadas e giros, mantendo a posse de bola. Para isso, os chutes devem ser curtos a ponto de afastar a bola o mínimo possível do controle do jogador. Critérios como distância e direção do adversário também são levados em consideração.

Muitas das ações adicionam aleatoriedade nos movimentos do jogador. Isso possibilita ao jogador agir de forma diversificada, com imprevisibilidade.

5.2 Comunicação

Como o simulador trabalha em tempo real, para tornar o cliente mais eficiente e otimizar o processamento através de concorrência, faz-se o uso de três *threads* [LEW 96], como apresentado na Figura 5.10.

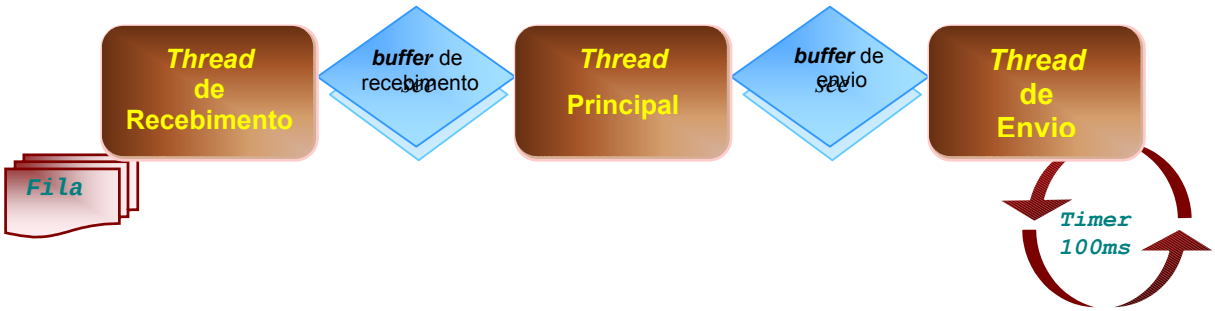


FIGURA 5.10 - Processamento Concorrente.

A sincronização entre as *threads* é realizada pelo mecanismo de exclusão mútua (MutEx), que é responsável por proteger as variáveis compartilhadas (região crítica), ou seja, os *buffers* de recebimento e de envio entre as *threads*. Evita-se, dessa maneira, que duas *threads* possam acessar o mesmo *buffer* ao mesmo tempo.

A *thread* de entrada possui uma fila que armazena as mensagens vindas do servidor. Sempre que possível, quando o *buffer* está disponível, é atribuido ao *buffer* de recebimento a primeira informação da fila. Caso a fila esteja vazia, a informação vinda do simulador é adicionada imediatamente no *buffer*. O recebimento dessas mensagens é realizado por mecanismos de *socket* bloqueante, permitindo um melhor aproveitamento do tempo da CPU para o processamento principal, onde se concentra a maior parte do processamento realizado pelo agente [HAL 2001].

A sincronização das ações de saída é realizada pela *thread* de Envio. Ela é responsável por controlar a quantidade de mensagens enviadas em cada ciclo. Um comando de ação (mensagem armazenada no *buffer* de envio) é enviado ao servidor no momento em que a *thread* de Envio recebe um sinal do *clock* interno ao agente.

O *clock* realiza a contagem de intervalos de 100ms e, ao final do ciclo, dispara um alarme, indicando que uma mensagem pode ser enviada (liberando a *thread* de envio para mandar a mensagem). Trata-se de um *clock* do próprio sistema operacional (no caso Linux). Dessa forma, o ciclo de sincronização é iniciado a cada 100ms.

5.4 Um Exemplo

Um exemplo do funcionamento do agente é ilustrado na Figura 5.11.

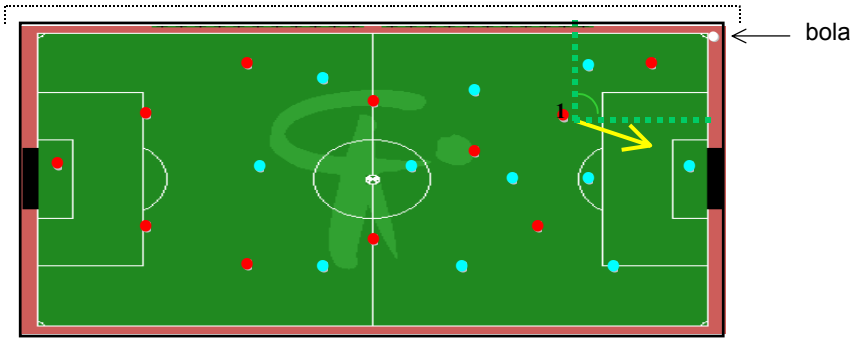


FIGURA 5.11 - Uma Situação de Jogada.

Nessa situação, ocorreu um escanteio e o jogador **1** percebe que existe um companheiro mais perto da bola do que ele. Dessa forma, o jogador, então, toma a decisão de caminhar para a área do gol do oponente e preparar-se para uma possível recepção de um passe, como ilustrado na Figura 5.12.

Esse é um comportamento individual e simples do agente que, num todo, acaba resultando num comportamento em grupo (entre os jogadores) mais organizado (comportamento emergente).

Cond	**ESCANTEIO e companheiro_perto_da_bola**
Ação	comportamento(ir para a área do gol oponente)

FIGURA 5.12 - Escolha da Ação.

5.5 Considerações sobre o Agente

Para a criação dos agentes jogadores, foi necessário realizar o estudo sobre o simulador Soccerserver, SMA e comunicação entre cliente/servidor. Na elaboração da arquitetura, foram analisados vários clientes existentes que participaram da RoboCup. Implementou-se a arquitetura com base nos conceitos apresentados, desenvolvendo vários comportamentos para os agentes.

A proposta objetivou proporcionar a viabilidade da aplicação do estudo realizado e de técnicas de SMA. Foi elaborada uma arquitetura para um agente jogador com características que facilitam a tarefa de implementação, procuram proporcionar maior eficiência, fornecer subsídios para que os comportamentos dos agentes possam ser interessantes e inesperados, além de permitir que o agente escolha a ação que mais se aproxime do seu objetivo no estado atual (possibilita ao agente reavaliar constantemente a sua situação em campo).

A arquitetura do agente UFRGS, em comparação com outras arquiteturas desenvolvidas para o Soccerserver, descritas neste trabalho, oferece a simplicidade e a facilidade tanto de implementação quanto de entendimento do funcionamento interno ao agente. Essa arquitetura proporciona ao agente realizar suas tarefas com base em suas características reativas.

A documentação sobre o agente e o simulador, quanto à implementação realizada, procura esclarecer, cuidadosamente, todas as características e o desenvolvimento da arquitetura proposta, com o intuito de permitir e encorajar a continuidade do aprimoramento do agente.

6 O Protótipo do Agente UFRGS

Desenvolveu-se um protótipo do agente UFRGS com o objetivo de verificar se as técnicas abordadas eram satisfatórias para que um agente pudesse jogar no ambiente do Soccerserver, com a capacidade de perceber e atuar no seu mundo. A implementação também exigiu uma compreensão em profundidade do funcionamento e das características determinantes do simulador.

Os primeiros testes com a arquitetura foram feitos com a implementação de agentes simples. Esses agentes tinham poucos comportamentos e atribuía-se às suas ações parâmetros com valores fixos. Esses não estavam preocupados com o gasto de energia, drible, passe, ou até mesmo, com a intercepção da bola. No entanto, eram capazes de executar ações básicas como levar a bola ao gol adversário e chutar, com boa precisão.

É importante salientar que o resultado da implementação é um protótipo criado para validar as técnicas através da atuação do jogador em campo. Buscou-se, desde o princípio, tornar as ações do agente as melhores, dentro do limite de recursos utilizados.

A seguir, acompanha-se algumas etapas da implementação do agente UFRGS e os resultados obtidos, considerando a descrição da arquitetura apresentada no Capítulo 5 (Arquitetura do Agente UFRGS).

6.1 Implementação

A implementação do agente UFRGS foi desenvolvida na linguagem de programação C, para rodar em ambiente Linux. O compilador utilizado foi gcc 2.91.66. O agente UFRGS também apresenta uma versão para Solaris 5.2.

6.2 Comunicação com o Servidor

O mecanismo utilizado para estabelecer a comunicação do agente com o simulador são os *sockets* UDP bloqueantes. A escolha pelo *socket* bloqueante atribui-se ao fato do mesmo despender menor tempo de CPU, de forma a ocupá-la apenas no momento da chegada de uma mensagem.

A comunicação ocorre através de funções *send* e *receive*, conforme ilustrado na Figura 6.1 e na Figura 6.2.

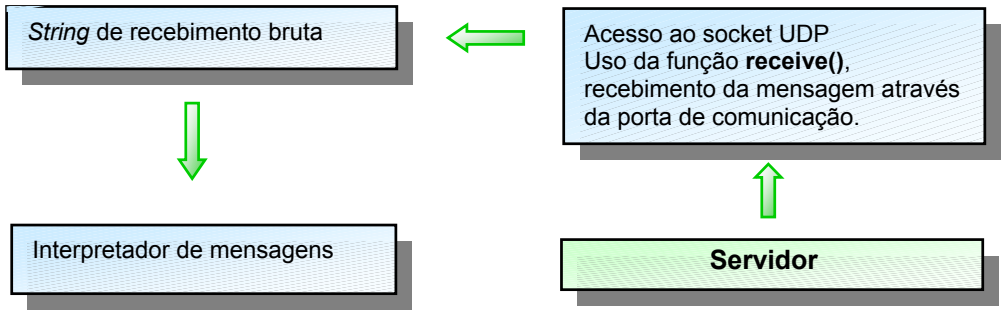


FIGURA 6.1 - Recebimento da Mensagem.

A Figura 6.1 corresponde a entrada de uma mensagem vinda do servidor. A função *receive*() realiza a interface de comunicação com o servidor e, posteriormente, disponibiliza a *string* bruta para o interpretador de mensagens.

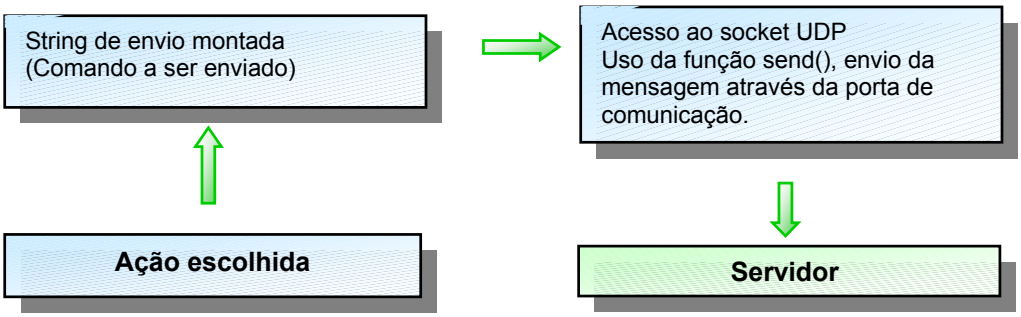


FIGURA 6.2 - Envio da Mensagem.

A Figura 6.2 apresenta o envio de uma ação a ser executada pelo jogador em campo. A ação escolhida pelo agente é convertida em um comando de saída. A *string* é enviada ao servidor através da porta de comunicação, por uma função *send*().

6.3 Módulos Concorrentes

Para tornar o agente UFRGS mais competitivo e melhorar seu desempenho em tempo-real, utilizou-se três *threads*. Cada *thread* é uma rotina diferente de controle que pode ser executada com suas instruções independentemente, permitindo um processo *multithread* para desempenhar várias tarefas concorrentemente. A primeira *thread* se preocupa com o recebimento das mensagens vindas do simulador, a segunda *thread* realiza o processamento interno ao agente, enquanto uma terceira trabalha no envio dos comandos de ações ao simulador.

Utilizou-se dois *mutexes* como uma forma de realizar sincronização por mecanismo de exclusão mútua, denominados: RPAC (*buffer* sincronizado de mensagens de entrada, compartilhado entre a *thread* de Recebimento e a Principal) e SPAC (*buffer* que fica entre a *thread* Principal e a *thread* de Envio). Uma visão geral da estrutura das *threads* pode ser vista na Figura 6.3.

O *mutex* proporciona um único proprietário para a seção de código (seção de código crítica, compartilhada entre as *threads*- *buffer*) entre chamadas de Mutex_Lock() e Mutex_Unlock(). A primeira *thread*, que chama *lock*, toma posse do *mutex*. Se houver alguma chamada subseqüente tentando realizar um *lock*, esta irá falhar colocando a *thread* que chamou para dormir. Quando o proprietário chama *unlock*, a *thread* seguinte é acordada, dando assim, a chance para um outro proprietário.

Outro mecanismo utilizado é a variável de condição (VC) que proporciona um meio ambiente seguro, permitindo a execução de uma *thread* somente se uma condição é satisfeita. A VC é responsável por sinalizar a atualização do *buffer* e da *flag* (chamada de predicado). O predicado irá indicar se uma mensagem foi recebida ou transmitida (0 ou 1).

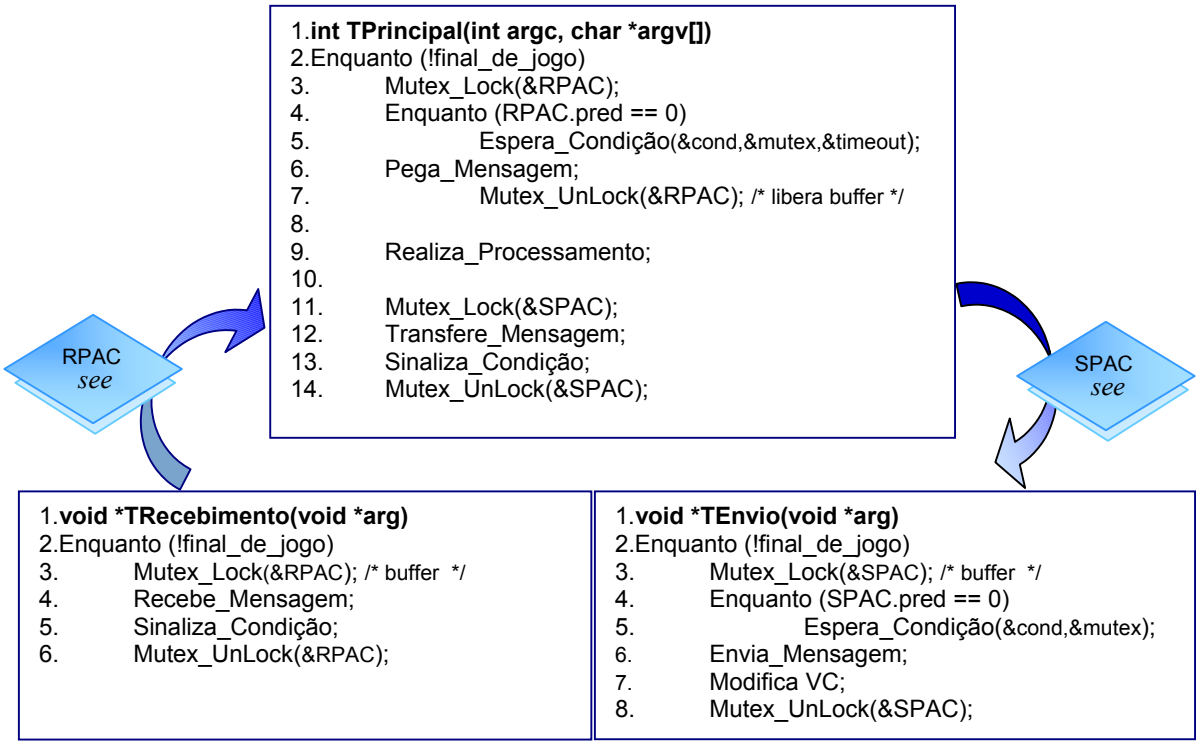


FIGURA 6.3 - Estrutura das *Threads* de Recebimento, Principal e Envio.

A *thread* que obtém o *mutex* testa a condição sobre o *mutex* protegido (ver linhas 3 e 4 da Figura 6.3 TPrincipal). Nenhuma outra *thread* poderá alterar algum aspecto da condição sem deter o *mutex*. Se a condição for verdadeira, a *thread* completa a sua tarefa, liberando o *mutex* quando apropriado (ver linhas 6 e 7 da Figura 6.3 TPrincipal). Se a

condição não é verdadeira, o *mutex* é liberado e a *thread* vai dormir em VC (ver linha 5 da Figura 6.3 TPrincipal). Quando outra *thread* mudar a condição, um sinal (ver linha 5 da Figura 6.3 TRecebimento) acorda a *thread* que estava dormindo. A *thread* então requer o *mutex* e reavalia a condição. Caso a condição seja falsa a *thread* volta a dormir. Caso contrário, a *thread* passa a executar a sua tarefa (ver linha 6 da Figura 6.3 TPrincipal). A *thread* principal é, no caso, do tipo impaciente, isso porque, a *thread* dorme por um tempo fixo (*timeout*) limitado pela VC (ver linha 5 da Figura 6.3 TPrincipal).

O funcionamento entre a TRecebimento e TPrincipal pode ser visualizado na Figura 6.4.

TRECEBIMENTO	TPRINCIPAL
	Adquire o mutex Testa VC Falsa! Libera o mutex Dorme em VC
Adquire o mutex Faz o processamento! Modifica a VC Libera o mutex	
	Re-adquire o mutex Re-testa a condição Sucesso! Faz o processamento! Libera o mutex

FIGURA 6.4 - Exemplo de Funcionamento das *Threads* Recebimento e Principal.

A Figura 6.5 mostra um exemplo de funcionamento entre a TPrincipal e a TEnvio.

TPRINCIPAL	TENVIO
Adquire o *mutex* Faz o processamento! Modifica a VC Libera o *mutex*	
	Adquire o *mutex* Testa VC Sucesso! Faz o processamento! Libera o *mutex*

FIGURA 6.5 - Exemplo de Funcionamento das *Threads* Principal e Envio.

Na *thread* de recebimento foi implementada uma fila de mensagens. Essa fila é capaz de armazenar as três últimas mensagens (podendo ser ampliada) vindas do

Soccerserver e tem como função principal evitar que as mensagens de entrada sejam perdidas. Basicamente, a fila apresenta o funcionamento ilustrado pela Figura 6.6.

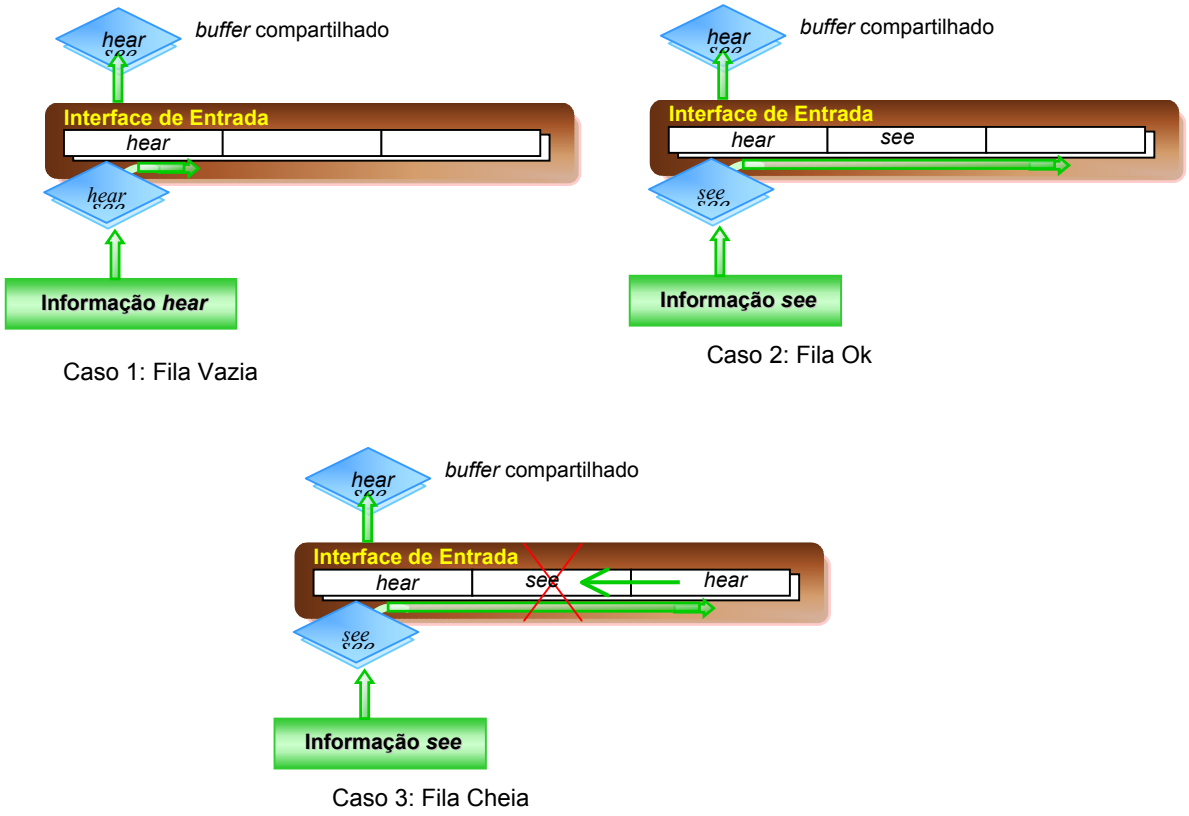


FIGURA 6.6 - Funcionamento da Fila de Entrada.

No caso 1, a fila está vazia, a mensagem de entrada além de ser copiada para RPAC (*buffer*) é também armazenada na fila (até que esta seja utilizada no módulo superior). No caso 2, tem-se um *slot* vazio para o armazenamento da mensagem. Já no caso 3, a mensagem *see*, mais antiga, como a do segundo *slot*, é eliminada e, assim, realiza-se uma realocação da fila. O motivo pelo qual a mensagem eliminada é a *see* e não a *hear*, dá-se ao fato da mensagem *hear* ter maior importância, por ser uma mensagem do árbitro ou de outro jogador.

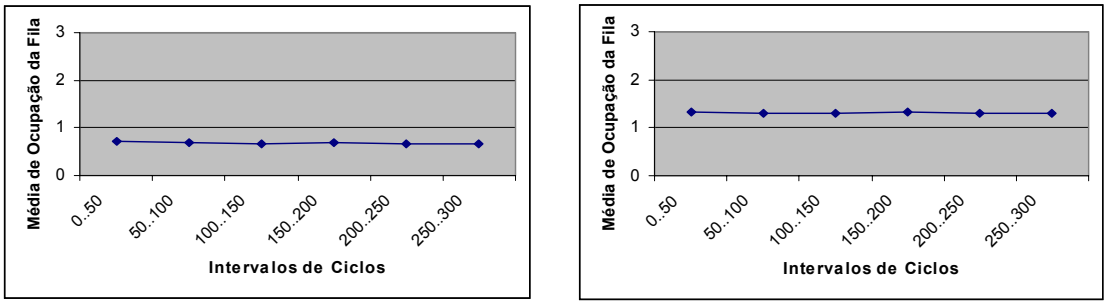
A definição da fila de entrada e o algoritmo resumido da mesma são apresentados na Figura 6.7.

```
char fila [MAX_FILA][BUFFER_MAX];
char buffer_tempo[BUFFER_MAX];

Se não RPAC vazio então
        Procura_slot_vazio
        Se fila_cheia então
            Se existe_mensagem_see_mais_antiga então
                    Elimina_mensagem_mais_antiga
                    Avança_fila
                    Armazena_mensagem
        Senão
            Armazena_mensagem_no_slot_vazio
        Atualiza RPAC
Senão
        Se existe_mensagem_na_fila então
                Avança_fila
                Atualiza RPAC
        Armazena_mensagem_no_slot_vazio
```

FIGURA 6.7 - Definição da Fila de Entrada.

A Figura 6.8 apresenta o andamento da fila de entrada. Na Figura 6.8.a, pode-se visualizar a média de ocupação da fila em intervalos de ciclos. Neste caso, a qualidade de visão utilizada pelo jogador era normal e a profundidade era alta. Percebe-se que entre os ciclos 0..50, a média de *slots* ocupados é menor que um, isto também é verdadeiro nos demais ciclos. Na Figura 6.8.b, a qualidade de visão era estreita e a profundidade era alta. A média de ocupação da fila permaneceu em um *slot* ocupado.



(A) (B)

FIGURA 6.8 – Utilização da Fila de Entrada.

O Anexo 1 apresenta parte do *logfile* gerado durante uma partida, que demonstra a situação da fila de entrada.

A *thread* de Envio mantém o controle dos ciclos de simulação, permitindo que apenas um comando de ação seja enviando ao simulador, a cada ciclo. Para isso, a *thread*

de Envio realiza o controle de um ciclo de 100ms através do relógio do sistema operacional (*cs*). A Figura 6.9 apresenta o tempo gasto em processamento para cada *thread* e o atraso proposicional de saída causado pelo *clock* interno ao agente (estes dados foram obtidos através de diversos experimentos que demonstraram estabilidade em seus valores). A Figura 6.9.a mostra a entrada de uma mensagem *see*, no momento o *clock* estava em 138ms. Ainda em 138ms a mensagem é adquirida pela *thread* Principal, que termina o processamento no tempo 141ms. De 141 à 146ms, a ação escolhida pelo agente permanece em estado de espera na *thread* de Envio (no SPAC). Então, no tempo 146ms, o comando de ação é adquirido pela *thread* de Envio e mandado para o simulador.

Na Figura 6.9.b, tem-se a próxima mensagem de entrada (200ms), trata-se de um comando *sense_body*. Entre 200 e 246ms o comando de ação ficou aguardando (em SPAC) pelo aviso do *clock*. Nesse caso, houve um atraso de 46ms. A Figura 6.9.c, apresenta a mensagem de entrada *see*, foram utilizados 3ms para o processamento, desde a sua entrada até o final do processamento. No entanto, ocorreu 53ms de espera até que o comando fosse enviado.

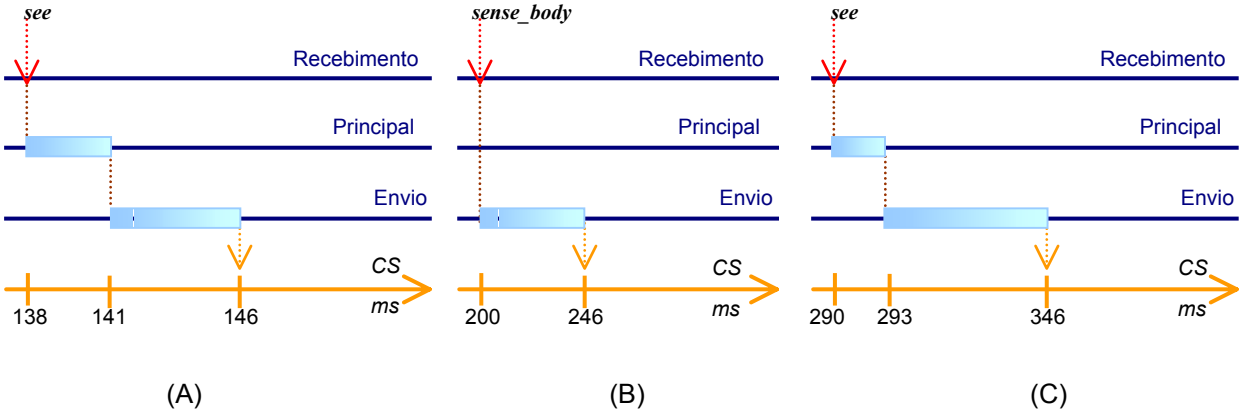


FIGURA 6.9 - Sinais de *Clock*.

Pode-se verificar, com base nas observações feitas sobre os testes (como o exemplo apresentado na Figura 6.9), que do recebimento da mensagem até a escolha da ação, o tempo gasto no processamento é muito inferior ao tempo que a ação espera por um sinal de final de ciclo, para então ser enviada ao simulador.

6.4 Estrutura Principal

A comunicação interna (entre os módulos) ao agente ocorre através de chamadas a rotinas. O *loop* principal do agente resume-se em: interpretar a mensagem de entrada, atualizar o estado do mundo, realizar o processamento interno, escolher um comando de ação a ser realizado e enviar a mensagem. A Figura 6.10 apresenta um algoritmo resumido do *loop* principal do agente UFRGS.

```
Loop do agente
        Recebe_Mensagem();
        Analisa_Tipo_Mensagem();
                Caso seja uma mensagem sense_body;
                        Interpreta_Mensagem();
                        Atualiza_Mundo();
                Caso seja uma mensagem see;
                        Interpreta_Mensagem();
                        Atualiza_Mundo();
                        Calcula_Variáveis();
                        Escolha_Ação();
                Caso seja uma mensagem hear;
                        Interpreta_Mensagem();
                        Atualiza_Mundo();
                        Escolha_Ação();
        Envia a ação();
```

FIGURA 6.10 - *Loop* Principal do Agente UFRGS.

Os agentes atuam enquanto a condição de parada for diferente de final do jogo. A cada nova mensagem de entrada, o agente interpreta a mesma e atualiza o modelo do mundo. Em caso de mensagens do tipo *see*, um conjunto de variáveis de controle é calculado. Após, é realizada a escolha de um comportamento. Dessa forma, o sistema faz uma seleção da regra de comportamento adequada à situação do jogo, escolhendo, assim, uma ação para ser executada pelo jogador.

6.5 Especificação do Modelo do Mundo

O conhecimento do agente é representado por uma combinação de estruturas de dados e de procedimentos de interpretação da percepção do jogador. Essas características levam o agente a realizar uma ação a ser executada.

As informações adquiridas pelo agente são dispostas de forma organizada no modelo do mundo do agente, de forma a deixar explícito coisas importantes. A base de conhecimento também facilita a computação no sentido de que a informação pode ser armazenada e recuperada rapidamente.

A representação do modelo do mundo do agente foi implementada em estruturas. As principais estruturas como a do modelo do mundo e o modelo do próprio agente é apresentada na Tabela 6.1. Cabe salientar que os objetos *flags* não são armazenados no modelo do mundo.

TABELA 6.1 - Estruturas de Implementação.

Estruturas	***Informações***
Objeto (informações de cada objeto visível pelo agente)	Tipo do objeto (jogador, bola ou gol); Time (nome do time); Número (número do jogador); Distância (distância ao objeto); Direção (direção ao objeto);
Eu (informações sobre próprio agente)	Função (goleiro, zagueiro, meio de campo, atacante); Número (número do jogador); Tempo (primeiro ou segundo tempo de jogo); Meu_time (nome do time do jogador); Time_adversário (nome do time adversário); Lado_campo (lado do campo em que o jogador saiu jogando); Ângulo absoluto do jogador em campo; Mensagem (mensagem do árbitro); Posição x (posição absoluta x do jogador em campo); Posição y (posição absoluta x do jogador em campo); Ciclo (ciclo do simulador); Qualidade da visão (baixa, alta); Largura da visão (estreita, normal, ampla); Energia (energia do jogador); Velocidade (velocidade do jogador); Direção da cabeça (em relação ao corpo); Quantidade de chutes realizados durante a partida; Quantidade de *dash* realizados durante a partida; Quantidade de giros realizados durante a partida; Quantidade de *say* realizados durante a partida; Quantidade de giros de pescoço realizados durante a partida.

As informações armazenadas no modelo do mundo servem tanto para o sistema buscar o valor de um dado, como também para reagir aos estímulos do ambiente. Sendo assim, armazena-se as informações para posteriormente recuperá-las de forma que o agente possa utilizar as informações.

6.6 Regras de Comportamentos

A escolha da ação a ser realizada vai depender da verificação de uma série de condições. Como abordado no Capítulo 5 (Arquitetura do Agente UFRGS), utilizou-se um sistema baseado em regras de comportamento para modelar as ações do agente. A representação baseada em regras possui razoável expressividade, sendo bastante adequada

para representar associações empíricas, além de apresentar uma sintaxe fácil de entender e de aplicar.

Maiores vantagens podem ser extraídas, apenas realizando uma análise cuidadosa na ordem de prioridades das regras. No agente UFRGS, a ordem das regras seguem alguns critérios de urgência, essas estão dispostas conforme a função do agente, sendo assim, o agente pode resgatar suas ações mais freqüentes rapidamente.

6.6.1 Especificação dos Comportamentos

O comportamento do jogador foi construído na forma de uma árvore de decisão (Figura 6.11). Parte-se do princípio que a mensagem do árbitro tem maior importância, exigindo uma resposta mais rápida. O ramo, que dá seguimento à mensagem do árbitro, contém os modos de partida diferentes de *play_on*, por exemplo: *kick_in*, *free_kick*. Esse ramo destina-se a obter um comportamento para o agente no caso de "bola parada".

Sendo assim, quando ocorrer uma parada de jogo como por motivo de escanteio, falta ou lateral para o time UFRGS, um de seus jogadores, irá realizar a cobrança, conforme a situação do jogo observado pelo agente. Esse jogador realiza a avaliação da jogada a ser executada e logo repõe a bola em jogo.

FIGURA 6.11 - Raiz da Árvore de Decisão do Agente.

O outro ramo da árvore destina-se ao modelo de jogo *play-on*, ou seja, "a bola está em jogo". Parte-se do princípio da posse da bola pelo jogador. Dois comportamentos táticos foram construídos: um para a posse da bola e outro para a não posse da bola. O desenvolvimento da árvore irá depender da função do jogador no campo. Diferenciam-se alguns comportamentos mais básicos de cada jogador, por exemplo: enquanto um atacante procura "driblar", um zagueiro tenta marcar.

Cada folha da árvore de decisão tem um conjunto de pré-condições que devem ser satisfeitas para que uma ação correspondente seja ativada. Essas pré-condições correspondem a um ramo da árvore que resulta no comportamento do jogador.

Os agentes possuem o mesmo conjunto de variáveis, exceto o goleiro (que possui a variável posso_pegar). Essas variáveis, juntamente com o conhecimento que o agente possui do mundo, guiam o processo da escolha de uma ação. As ações (*kick*, *dash*, *turn*) são as mesmas para todos os agentes, no entanto, os comportamentos diversificam-se entre as diferentes funções dos agentes.

6.6.2 Comportamento do Agente Goleiro

A idéia geral do comportamento do goleiro baseia-se em um semi-círculo formado em torno da pequena área. Esse semi-círculo é denominado "área de ação" do goleiro (Figura 6.12). O goleiro procura estar dentro dela, sempre posicionado no ponto onde a bola possa interceptar o semi-círculo.

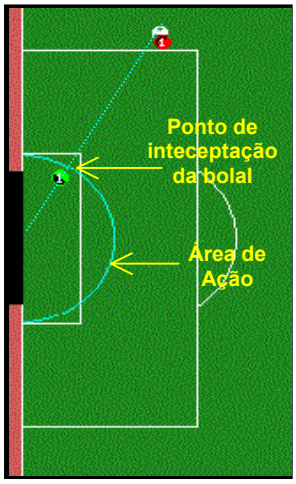


FIGURA 6.12 - Área de Ação do Goleiro.

Os principais comportamentos do goleiro são:

1. Saber onde se localiza a bola.

Se o goleiro deixar de enxergar a bola, ele tentará localizá-la. Para isso, o goleiro verifica se o último ângulo conhecido da bola era positivo ou negativo. Logo após, ele gira no mesmo sentido. Caso não consiga determinar o último ângulo conhecido, o goleiro fará a procura pela bola, executando um giro no sentido escolhido arbitrariamente.

A única situação em que o goleiro não tenta procurar a bola é quando ele se encontra fora de sua "área de ação" e está regressando a mesma. A "área de ação" do goleiro é uma região limitada por um semi-círculo criado em torno do ponto central de sua goleira, e tem como principal função auxiliar no posicionamento do mesmo.

O fato do goleiro não procurar a bola quando está fora da sua área de ação, foi uma solução encontrada para realizar a correção do seguinte problema: quando o goleiro saía de

sua área de ação por algum motivo (como correr atrás da bola, por exemplo), ele tinha grande dificuldade em regressar.

Quando o goleiro tinha visão da bola, ele calculava o ponto entre a bola e o centro da goleira que ficava no limite da área de ação, onde ele deveria se posicionar. Esse ponto foi chamado de ponto de intersecção (PI).

Tendo calculado o PI, o goleiro virava-se para ele. Quando estava de frente para o ponto, acabava por perder a posição da bola. O goleiro geralmente fica entre o PI e a bola quando está fora da área de ação, ou seja, não consegue ver os dois ao mesmo tempo. Não tendo mais a posição da bola, não podia recalcular o próprio ponto de intersecção. Procurava novamente a bola, calculava o PI, virava-se para o PI e perdia a bola, e assim sucessivamente.

Para corrigir essa situação de indecisão, decidiu-se fazer com que o goleiro sempre volte para a sua área de ação, quando não enxergar a bola ou quando esta estiver muito longe dele (mais de 9 metros).

2. Agarrar a bola, sempre que possível.

Sempre que o goleiro tiver a possibilidade de pegar a bola, ele pegará. Para que o goleiro possa pegar a bola, ela deve estar em uma região chamada "*goalie_cacthable_area*" que é o retângulo de dimensões 2x1 metros em torno dele. Existe, também, outro parâmetro que influencia no sucesso do ato de pegar a bola, que é o "*catch_probability*". Ele define a probabilidade de sucesso de um comando "*catch*", ou seja, mesmo que o goleiro tente pegar a bola no lugar certo, existe a possibilidade de ele não conseguir.

Para determinar "o lugar certo", onde a bola estará, desenvolveu-se uma rotina que calcula a posição da bola para o ciclo seguinte de execução. Durante testes realizados usando o comando *catch* associado à direção e distância da posição futura da bola, o goleiro mostrou melhor aproveitamento nas defesas do que utilizando os parâmetros da posição presente.

Uma circunstância em que o aproveitamento do goleiro ainda deixa a desejar é para a defesa de chutes fortes. A variação muito grande na posição da bola a cada ciclo, o ruído introduzido pelo servidor e a baixa taxa de amostragem na recepção de informações visuais são as principais dificuldades encontradas para se calcular uma posição futura para a bola precisa e tornar o *catch* mais eficaz.

3. Verificar a trajetória da bola e interceptá-la, caso ela intercepte sua área de ação.

O goleiro sempre procura calcular a trajetória que a bola irá percorrer. Caso a trajetória da bola intercepte a sua área de ação em algum ponto, o goleiro tentará posicionar-se nesse ponto, de modo a poder pegar a bola quando ela passar.

Para calcular a trajetória da bola, toma-se a posição da mesma em dois tempos diferentes e verifica-se o deslocamento. Soma-se o deslocamento à última posição da bola, e tem-se uma previsão aproximada da próxima posição. Multiplicando o deslocamento por

um fator ‘n’ obtém-se uma aproximação para vários tempos futuros, ou seja, uma trajetória aproximada.

Porém, se a bola chutada não passar pela área de ação, o goleiro simplesmente se posiciona em um ponto entre a posição atual da bola e o centro da goleira (conforme item 4).

4. Posicionar-se de maneira a se entrepor entre a bola e sua goleira.

Esta tarefa será executada sempre que a trajetória da bola não tiver previsão de interceptar a área de ação do goleiro (como visto no item 3).

A ação consiste em calcular o PI (entre a bola e o centro da goleira, que esteja nos limites da área de ação) e fazer com que o goleiro se posicione próximo ao ponto (Figura 6.13).

FIGURA 6.13 - Ponto de Intersecção.

Para calcular o PI é necessário saber a posição absoluta da bola. Entre a bola e o centro da goleira traça-se uma reta, e calcula-se o PI entre essa reta e o semi-círculo que limita a área de ação.

Com base nas coordenadas do PI e nas coordenadas e ângulo do goleiro, pode-se calcular a distância e a direção do PI visto pelo goleiro.

A idéia de calcular um PI procurou proporcionar ao goleiro a chance de cobrir com maior facilidades toda a extensão da goleira, fechando o ângulo do chute do atacante.

5. Correr atrás da bola, se esta estiver próxima dele.

Analisando as ações do goleiro até aqui citadas, pode-se ter a impressão de que o goleiro está limitado apenas à sua área de ação, e não à área da qual ele dispõe segundo os limites do campo.

Para “desamarrar” o goleiro desses limites, o comportamento “correr para a bola” foi utilizado. Essa ação consiste em fazer com que o goleiro corra para a bola quando esta estiver a uma distância não muito grande dele (aproximadamente 9m), mesmo que ele tenha que sair da sua área de ação (Figura 6.14). Para que essa ação seja executada quando a bola se encontra fora da área de ação, também é necessário que não haja outro companheiro mais perto da bola do que o goleiro. Caso exista outro companheiro mais próximo, é mais seguro manter o goleiro em sua posição e deixar que o outro jogador domine a bola.

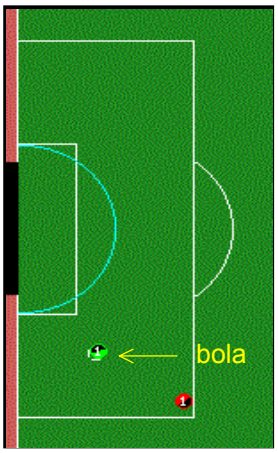


FIGURA 6.14 - Saída da Área de Ação.

Portanto, se a bola estiver dentro da área de ação, o goleiro tentará pegá-la de qualquer forma. Se a bola estiver fora da área de ação, porém não muito longe do goleiro, e não houver outro jogador companheiro perto dela, o goleiro irá tentar alcançá-la. Em qualquer outra circunstância, o goleiro ficará em sua área de ação, próximo ao PI.

6. Caso possua posse de bola, repô-la em jogo.

Recolocar a bola em jogo é uma tarefa simples. Quando o goleiro agarra a bola ele move-se para um certo local (dentro da grande área) e procura jogar a bola para um de seus companheiros. Ele realiza uma análise sobre o estado do mundo e então, se possível, seleciona o companheiro mais favorável a receber a bola. Essa análise envolve características como: companheiro desmarcado, companheiro na defesa, companheiro no ataque.

Outro caso onde o procedimento acima citado ocorre são nas cobranças de tiro de meta.

A seguir, apresenta-se parte da árvore de decisão do goleiro (Figura 6.15) a partir do critério da posse de bola em modo *play_on* de jogo.

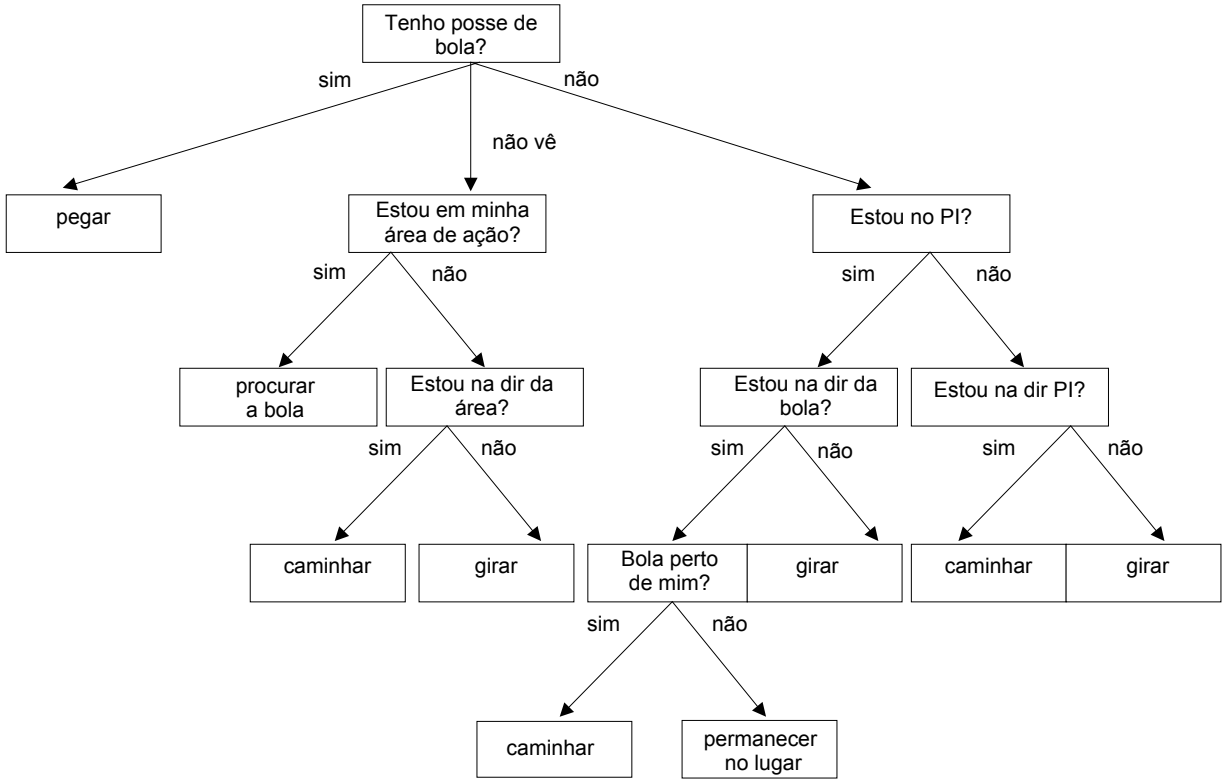


FIGURA 6.15 - Parte da Árvore de Decisão do Agente Goleiro.

O agente goleiro pode executar tarefas como, "pegar a bola" de forma automática, sem necessitar que outras verificações de menor relevância, como retornar à área de ação, sejam feitas anteriormente. O comportamento pegar a bola, portanto, apresenta uma prioridade maior para esse tipo de agente.

A árvore completa do agente goleiro é apresentada no Anexo 2.

6.6.3 Resultados Obtidos com o Goleiro

Nos testes de desempenho realizados (através de observações diretas) com o goleiro, constatou-se que ele pode executar bem as funções básicas, tais como: reposição de bola; tiro de meta; posicionamento.

Para o posicionamento e também movimentação em geral, verificou-se que o goleiro se desloca muito devagar em relação aos outros jogadores. O problema tem como causa o fato de sua visão exigir maior qualidade, o que faz com que ele receba informações visuais com freqüências mais baixas. Uma possível solução seria alternar a visão do goleiro entre um ângulo mais amplo, quando a bola está longe e um ângulo mais fechado, quando a bola está perto ou ainda, utilizar o que foi denominado de "comandos no escuro" para o goleiro. Os "comandos no escuro" são possíveis comandos para ciclos futuros, que não executariam nenhuma ação por falta de informação visual.

Uma tentativa de reduzir a largura da faixa visível do goleiro acarretou instabilidades, devido ao recebimento de informações visuais repetidas e, também, porque o goleiro passou a perder a bola de vista com muita facilidade. Outra dificuldade constatada durante os testes foi prever o comportamento da bola para chutes fortes, ou seja, bolas rápidas são muito mais difíceis de defender quando se tem uma freqüência de informação visual baixa.

De modo geral, o goleiro demonstrou que pode realizar as tarefas destinadas a ele.

6.6.4 Comportamento do Agente Meio de Campo

Os jogadores UFRGS, exceto o goleiro, apresentam características similares e têm como principal referencial, a bola. Diferem entre si apenas na prioridade de suas funções mais específicas. Dentre seus comportamentos, o jogador meio de campo possui, como idéia principal, estar em controle da bola para realizar um passe, conduzir a bola ou ainda, por ventura, chutar ao gol oponente. Esse jogador prioriza comportamentos como o de chutar a bola para o companheiro.

A seguir, apresenta-se os principais comportamentos do meio de campo. A escolha por descrever o jogador meio de campo, deu-se pelo fato do mesmo também representar os comportamentos dos demais jogadores. Portanto, a descrição que segue, serve para todos os jogadores (exceção do goleiro). Apenas a prioridade das ações é encontrada com grau de urgência diferente, dependente da função do jogador em campo.

1. Saber onde se localiza a bola.

Assim como o goleiro, os demais jogadores (zagueiro, meio de campo e atacante) tendem a localizar a bola sempre que a mesma não estiver visível. Tanto o meio de campo, quanto o atacante e o zagueiro, possuem uma área de posicionamento em campo. Essa área corresponde a uma região em torno da posição inicial do jogador. Essa região é limitada por um círculo imaginário, que possui como centro a posição inicial do jogador e tem como objetivo auxiliar no posicionamento do mesmo (Figura 6.16).

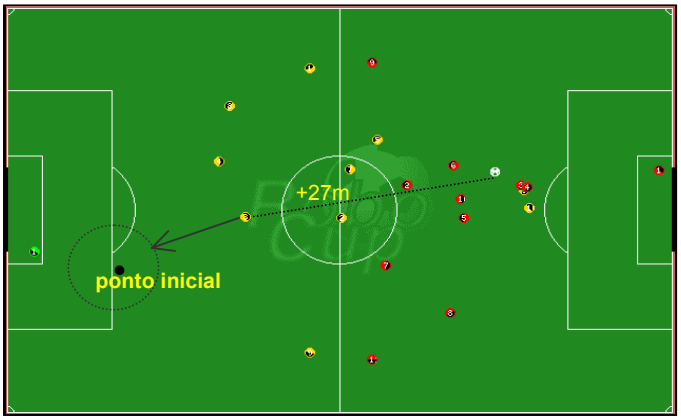


FIGURA 6.16 - Retorna à Posição Inicial.

O jogador procura manter-se em sua área de posicionamento em momentos nos quais não faz parte de uma jogada, ou seja, sempre que estiver a grande distância da bola (aproximadamente 27m) e o seu time tiver a posse da bola.

Da mesma forma, como o goleiro, correções foram feitas para o problema de indecisão do jogador de retornar à sua área de posicionamento e procurar a bola (como será descrito no item 4).

2. Passar a bola para o companheiro.

Para realizar um passe, o jogador que tem o controle da bola, verifica a disponibilidade de seus companheiros (como, se seu companheiro está desmarcado). Ele tentará chutar a bola para o companheiro mais apropriado (Figura 6.17).

Caso o jogador que tenha a posse de bola esteja marcado, ele tentará chutar para o companheiro procurando evitar o adversário que está próximo.

FIGURA 6.17 - Passe.

Para realizar esse comportamento, o jogador segue algumas pré-condições como: verificar os companheiros mais próximos dele ou do gol adversário, analisando quais estão desmarcados. Sempre em um passe, o jogador procura ter como referencial o gol do adversário. Essa condição procura proporcionar que o mesmo, ao realizar um passe, prefira um passe em direção ao gol do adversário e não ao seu próprio gol.

3. Interceptar a bola.

A tarefa de interceptar a bola corresponde à ação do jogador de ir para um ponto da trajetória da bola, no qual ele possa ter o domínio da mesma.

O jogador procura inteceptar a bola, calculando um ponto de interceptação. Esse ponto corresponde à interceptação entre duas linhas. Uma delas é a linha formada por dois pontos, que são o deslocamento da bola em dois ciclos consecutivos (o atual e o anterior), a outra é uma linha perpendicular traçada à primeira que passa pelo ponto do jogador, conforme ilustrado na Figura 6.18.

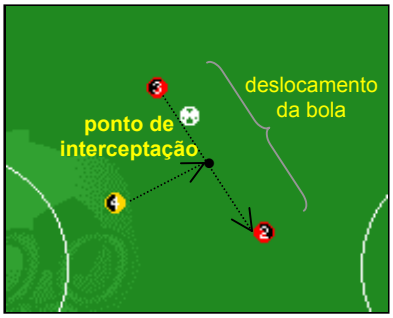
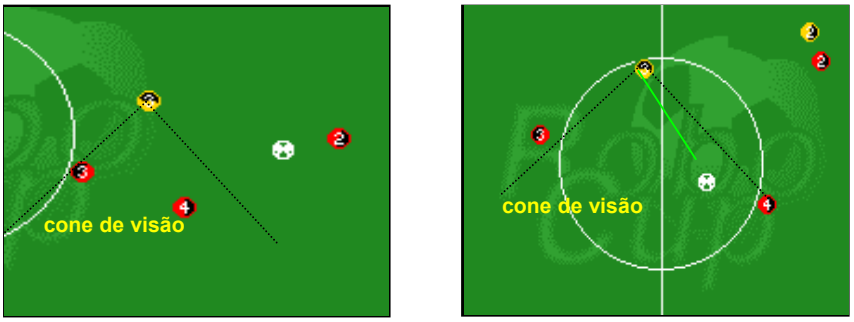


FIGURA 6.18 - Interceptação da Bola.

Conhecido o ponto de interceptação, deve-se saber se a linha da trajetória da bola irá passar perto do jogador, para isso, realiza-se uma estimativa da distância que a bola vai percorrer até parar e da distância total da bola ao ponto de interceptação. Caso o valor dessa estimativa seja menor que a distância total da bola ao ponto, a ação não é executada, pois a bola não chegaria ao ponto. Caso contrário, o jogador calcula a direção para a qual ele vai se virar para que fique de frente para o ponto (utilizou-se o ângulo absoluto do jogador) e finalmente correr para esse ponto.

Algumas instabilidades foram observadas nesse comportamento, como em momentos nos quais o jogador perde a visão da bola. Conforme o jogador se aproxima do ponto de interceptação, a bola pode acabar saindo do seu cone de visão, ocasionando o comportamento de procurar a bola. Em muitos casos, o jogador abandonava a ida ao ponto de interceptação e começava a procurar a bola.

Para solucionar esse inconveniente, considerou-se que, se a distância do jogador ao ponto de interceptação for pequena (menor do que 8m, por exemplo), o jogador continua a sua caminhada em direção à interceptação da bola. Caso contrário, é feita uma reavaliação do ponto de interceptação, a cada nova informação visual. É permitido ao jogador que se mantenha caminhando em direção ao ponto de interceptação quando sua distância é menor do que 8m, isso, porque, quando o mesmo encontra-se a uma pequena distância da linha da trajetória da bola, é mais provável que, no ângulo de visão do jogador, não esteja aparecendo a bola (Figura 6.19.a). Já para distâncias maiores, é possível visualizar a bola, como pode ser visto na Figura 6.19.b.



(A) (B)

FIGURA 6.19 - Cone de Visão do Jogador.

4. Posicionar-se em campo.

Esta tarefa será executada sempre que o jogador estiver a grande distância da bola e seu time possuir o controle da mesma. Nesse caso, o jogador irá retornar a um ponto dentro da sua área de posicionamento, que não necessariamente seja o seu ponto inicial. A única restrição quanto a esse comportamento, está no fato de o jogador ficar em uma região em volta do ponto inicial, na qual a sua distância não seja maior do que 5m ao ponto (Figura 6.20.a). Estando em sua região de posicionamento, o jogador passa a executar as demais ações. A Figura 6.20.b mostra que, após chegar em sua posição, o jogador realiza um giro referente ao comportamento de procurar a bola.

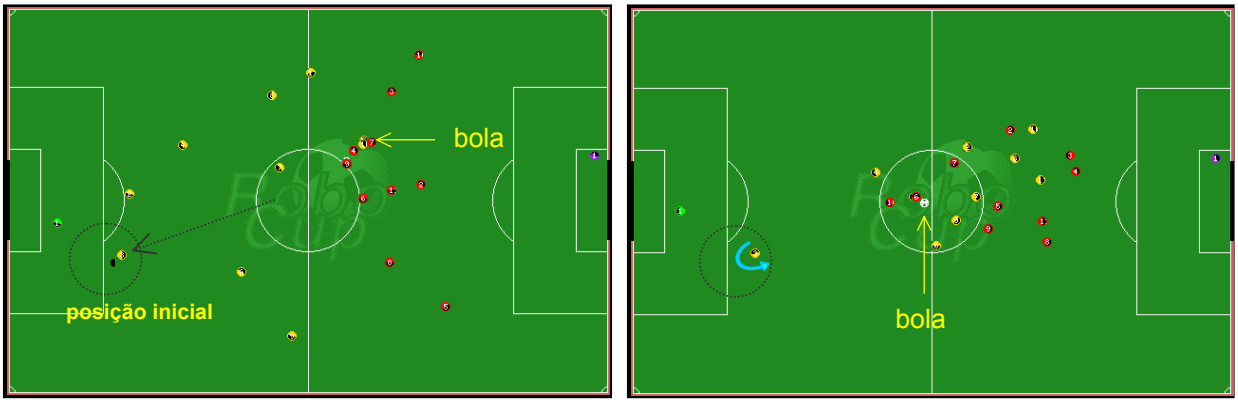


FIGURA 6.20 - Posicionamento do Jogador em Campo.

O comportamento "posicionar-se em campo" também possibilita a perda das informações da bola, visto que o jogador tende a se virar para o ponto inicial, ocasionando a perda da visão da bola, o que possibilita o comportamento de procurá-la. Para a solução desse problema foi incluída uma condição na regra de procurar a bola: sempre que a última ação for girar para o ponto inicial, deve-se ignorar a regra de procurar a bola e seguir para a posição. Como essa ação é feita quando o jogador está a grande distância da jogada, não é necessário fazer a procura da bola (em seguida), logo, a ação de voltar para a região de posicionamento pode ser realizada.

5. Marcar o adversário.

O problema da marcação envolve vários fatores e tal comportamento pode ser utilizado por diferentes jogadores. No entanto, as diferentes prioridades podem definir esse comportamento da seguinte forma: um zagueiro pode estar sempre procurando marcar, já um meio de campo irá marcar no momento em que nenhum outro comportamento for satisfeito.

Para o comportamento marcar acontecer, o jogador deve saber qual adversário marcar. Uma solução simples que apresenta razoável resultado é a de procurar marcar o jogador adversário mais próximo que está desmarcado.

Uma das alternativas para realizar o comportamento de marcação é representada pelo esquema da Figura 6.21.

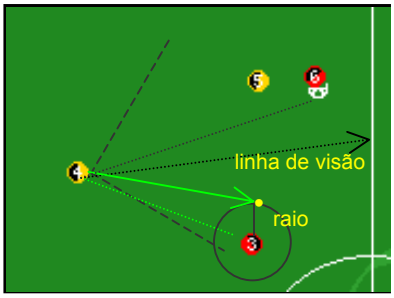


FIGURA 6.21 - Marcar o Adversário.

Quando a bola está visível, considera-se a região de marcação um circulo de raio de aproximadamente 5m em volta do adversário. O jogador procura ir para uma região entre a bola e o adversário. Para achar a direção na qual o jogador irá virar-se, toma-se como referência a direção do adversário somada de um certo desvio que fará com que o jogador fique entre a bola e seu oponente. A equação para achar a direção fica então da seguinte forma: direção de marcação = direção do oponente + desvio (calculado através do arco tangente do cateto oposto -raio- sobre o cateto adjacente -distância ao adversário-).

No caso da bola não estar visível, o jogador vai em direção ao adversário. Na maior parte, esse comportamento faz com que o jogador fique atrás de seu adversário. No entanto, quando o jogador chega até o oponente, ele começa a procurar a bola e ao achar a bola, ele pode calcular a direção e se locomover para a região entre a bola e o adversário, (caso ainda não esteja nela).

6. Ir para a bola.

Outro comportamento atribuído ao jogador é o de ir para a bola sempre que não existir companheiro perto da bola (aproximadamente em um raio de 3m da bola). Nesse caso, o jogador considera-se o mais próximo da bola e vai em direção a ela, mesmo estando a uma grande distância. Esse comportamento também permite que o jogador possa se libertar de sua região de posicionamento, proporcionando uma maior movimentação em campo.

A ação de ir para a bola, pode acarretar a ida de dois ou mais jogadores do time em direção a bola, ocasionando o agrupamento dos jogadores, isto ocorre pelo fato dos mesmos, não visualizarem o seu companheiro de time, como mostra a Figura 6.22. Neste caso, os jogadores 7 e 8 vão em direção à bola.

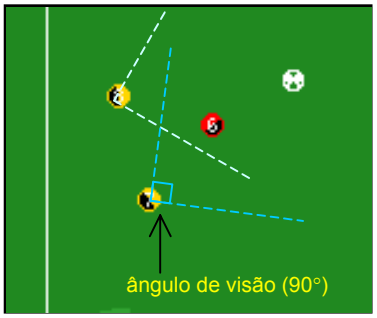


FIGURA 6.22 - Ir para a Bola.

7. Manter-se desmarcado.

A ação consiste em caminhar desviando do adversário, procurando afastar-se do mesmo. O jogador tende a ir para uma direção que o deixe mais próximo da bola. No caso dele não estar enxergando a bola, atribui uma direção randômica para seu movimento. O jogador tente a realizar o desvio enquanto o adversário ainda estiver em sua área de marcação. Para executar tais tarefas, o jogador deve estar sendo marcado pelo adversário, e ter algum companheiro perto da bola. Essas condições evitam que o jogador desvie de seu objetivo principal que é o de ter o controle da bola para o seu time.

A seguir, apresenta-se parte da árvore de decisão do meio de campo (Figura 6.23) a partir do critério da posse de bola em modo *play_on* de jogo. Basicamente, os agentes zagueiro, meio de campo e atacante apresentam os mesmos comportamentos, maiores diferenças podem ser percebidas na disposição dos ramos.

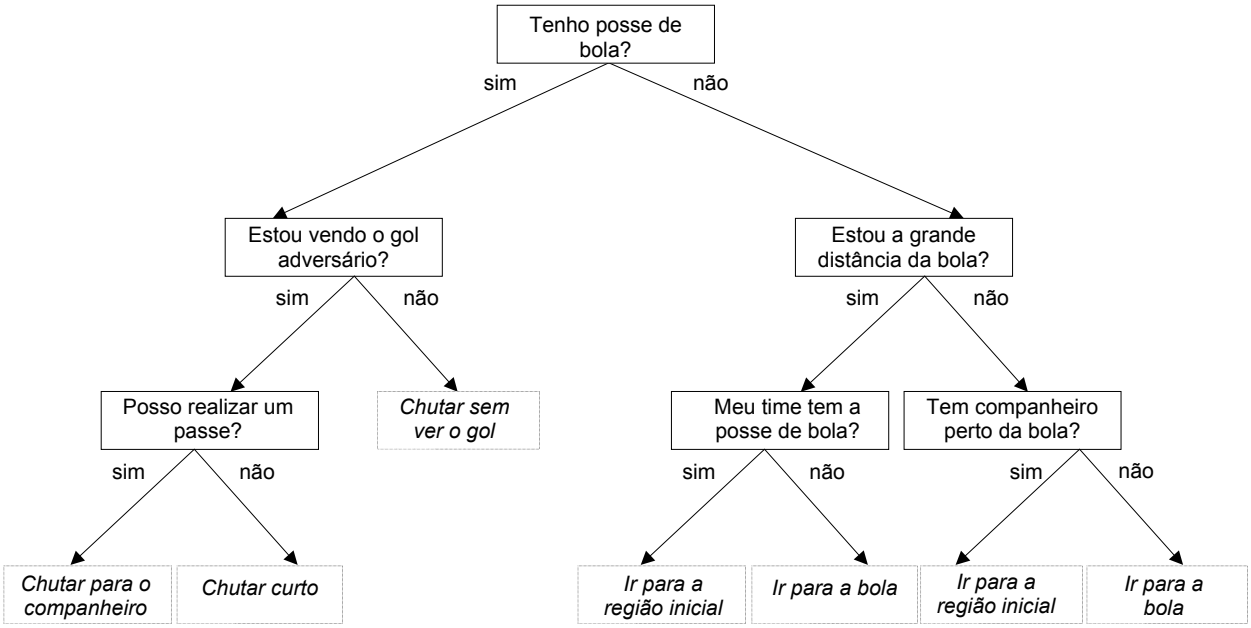


FIGURA 6.23 - Parte da Árvore de Decisão do Agente Meio de Campo.

As tarefas como “chutar para o companheiro” possuem um grau de prioridade maior para o jogador meio de campo.

Os retângulos pontilhados, resumem o resultado obtido após a verificação de um conjunto de condições, ou seja, o resultado representa um ramo percorrido da árvore. Por exemplo, a ação “chutar para o companheiro” requer que as seguintes condições sejam satisfeitas:

1. estou vendo o gol do adversário?
2. estou na direção do gol do adversário?
3. estou atrás da bola?
4. existe companheiro perto de mim?
5. existe companheiro desmarcado?
6. estou desmarcado?

Para a verificação das condições acima, é utilizada a combinação das seguintes variáveis de controle:

- companheiro perto de mim
- estado do jogo
- posso chutar
- companheiro desmarcado
- estou desmarcado
- além de variáveis como: direção da bola e direção do gol do adversário.

A árvore completa do agente meio de campo pode ser observada no Anexo 4. O Anexo 4 ilustra um número abaixo de cada folha, que indica o número do comportamento. No Anexo 3 e 5 são apresentadas as árvores do agente zagueiro e do atacante, respectivamente. O Anexo 6 ilustra um exemplo dos comportamentos criados para o agente meio de campo (numerados de acordo com a árvore do mesmo). Este exemplo foi extraído do código fonte do agente.

6.6.5 Resultados Obtidos com o Meio de Campo

Tarefas como marcar, fugir da marcação e outras, em um ambiente com informações incompletas e imprecisas como o do Soccerserver não são simples, um grande número de cálculos é necessário. Embora, na teoria, os cuidados com a definição dos comportamentos e o desenvolvimento das rotinas necessárias, parecessem funcionar corretamente, os testes mostraram dificuldades para o jogador efetuar tais tarefas. A

necessidade de um maior detalhamento das informações e de uma considerável quantidade de cálculos tornou o trabalho difícil de ser realizado.

A execução de comportamentos mais simples como, localizar a bola, passar a bola para o companheiro, posicionar-se em campo, ir para a bola, demonstraram um bom desempenho do jogador. Esses comportamentos não necessitam de verificações complicadas que envolvam um grande número de cálculos ou estimativas. Ao contrário, as rotinas simples utilizadas proporcionaram uma agilidade na decisão e na realização dessas ações.

6.7 Dificuldades

Algumas dificuldades foram enfrentadas durante o desenvolvimento dos comportamentos do agente. Vários comportamentos envolveram a criação de funções, principalmente matemáticas, de modo a viabilizar a realização de uma ação apropriada. Dentre elas, pode-se citar a função que calcula as coordenadas absolutas tanto da bola, quanto do próprio jogador; a função que retorna a distância e a direção de um ponto a partir de suas coordenadas; o cálculo da área de atuação do jogador.

Vários esforços foram concentrados no desenvolvimento do cálculo da posição absoluta do jogador em campo. Diversas funções foram desenvolvidas até que fosse possível obter um melhor resultado para a posição, de modo a reduzir o erro inserido pelo simulador. Um exemplo do cálculo da posição absoluta é apresentado na Figura 6.24.

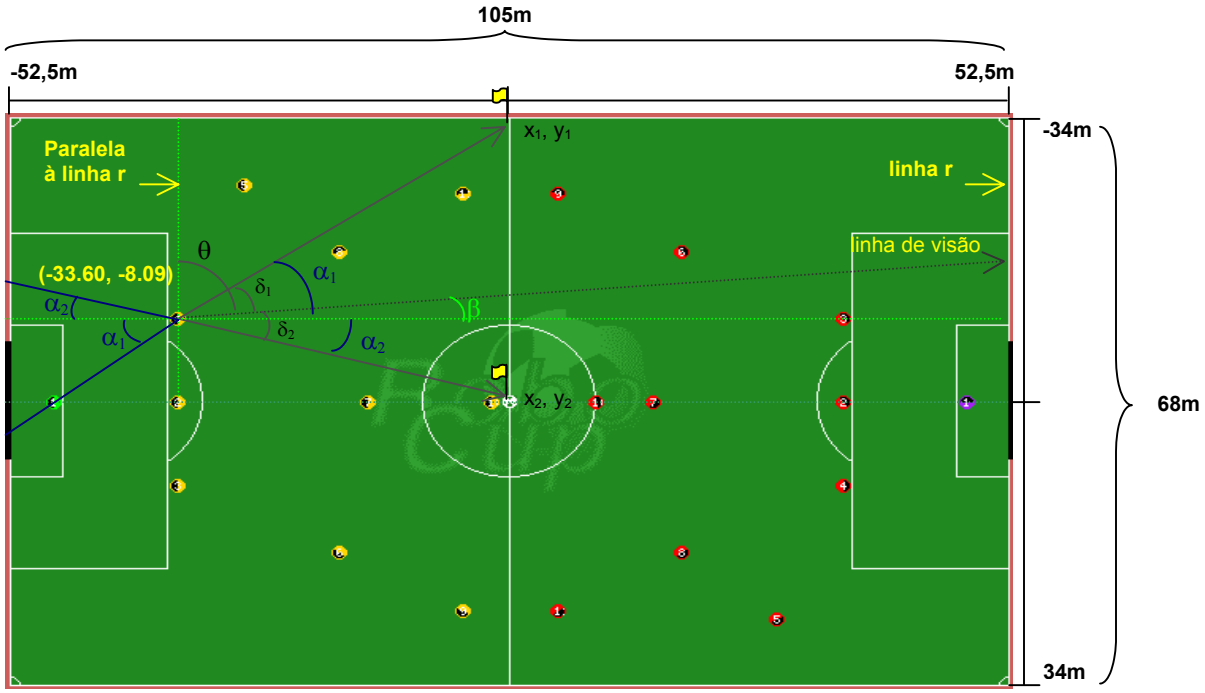


FIGURA 6.24 - Cálculo da Posição Absoluta.

A posição absoluta do jogador é calculada com base na direção (δ_1 e δ_2) e na posição (x, y) das *flags*, além da direção de uma das linhas (θ) do campo, visíveis pelo jogador. A partir de θ, pode-se calcular o ângulo de normalização com relação à linha horizontal imaginária do jogador (β). Este ângulo indica o sinal e o valor dos ângulos (α_1 e α_2), utilizados no cálculo da posição absoluta. De modo geral, foi usada a seguinte fórmula, para o cálculo:

$$tg\alpha_1 = \frac{y_1 - y}{x_1 - x} \qquad tg\alpha_2 = \frac{y_2 - y}{x_2 - x}$$

Com isto obteve-se que:

$t_1 = posição\_flag_1.y - (\tan(\alpha_1) * posição\_flag_1.x)$
$t_2 = posição\_flag_2.y - (\tan(\alpha_2) * posição\_flag_2.x)$
$posição\_x = (t_1 - t_2)/ \tan(\alpha_2) - \tan(\alpha_1)$
$posição\_y = t_1 + (\tan(\alpha_1) * posição\_x)$

Para conseguir maior precisão do valor da posição do jogador, realizou-se uma média das posições calculadas a partir de todas as *flags* visíveis pelo mesmo.

Procurou-se também, tentar evitar que as ramificações da árvore crescessem demais. Embora ele consiga realizar suas tarefas com um conjunto simples e pequeno de regras, comportamentos mais complexos exigiram que um maior número de regras fossem definidas. Em muitos casos, a habilidade do agente cresceu a uma taxa proporcional ao crescimento da base de regras de comportamentos, mas isso causou o aumento do conjunto de regras. Esse crescimento pode acarretar dificuldades no controle das ações executadas e na compreensão do funcionamento do agente.

6.8 Jogos

Para a verificação dos resultados, um dos passos foi conferir a funcionalidade do sistema. Através da implementação tornou-se possível apurar o desempenho do jogador individualmente, bem como, observar o desempenho dos jogadores diante de outros times. Para testar a aplicabilidade do agente no simulador, foram realizados testes que demonstraram a comunicação, o processamento interno e a escolha da ação (como apresentado no início deste capítulo).

Individualmente, o agente demonstrou determinação em suas tarefas em direção ao seu objetivo. Constantemente, buscava pela bola, tentando conduzi-la ao gol do adversário. Por outro lado, ficaram evidentes pontos fracos do time que serão discutidos aqui.

Foram realizados jogos com o time UFRGS frente a adversários mais sofisticados que participaram da RoboCup. Cabe salientar que os comentários apresentados aqui são baseados apenas na observação direta feita das partidas realizadas.

Dentre os jogos realizados contra times recentes como: Essex Wizards (8-0) (privatewww.essex.ac.uk/~kkosti/Files/ew99.tar.gz), CMUnited (8-0) (www.cs.cmu.edu/

~pstone/RoboCup/CMUnited98-source.tar.gz) e Humboldt (6-0) (www.ki.informatik.hu-berlin.de/RoboCup/RoboCup97/agent.tgz). Times como CMUnited e Essex, demonstraram grande agilidade em jogadas ensaiadas. Esses jogadores apresentaram ter bom conhecimento das posições dos companheiros em campo. Percebeu-se a facilidade desses times em antecipação às jogadas, como em prever a posição futura da bola. Outro fator observado foi a dificuldade do time em dominar a bola nas jogadas divididas contra o adversário.

Embora tenham ocorrido várias jogadas bem sucedidas em conjunto com seus companheiros, houve momentos nos quais o agrupamento entre os agentes UFRGS acabava ocasionando uma situação de retranca. Mesmos próximos, os jogadores não conseguiam ver uns aos outros, por esse motivo, acreditavam que não havia nenhum companheiro próximo da bola, o que ocasionava a ida de vários jogadores em direção a bola.

Por não existir nenhuma preocupação em conhecer o posicionamento de todos os jogadores em campo, grande parte dos passes feitos pelo jogador UFRGS eram realizados apenas em momentos onde o mesmo pudesse ver o seu companheiro.

O confronto com um time mais simples como o Andhill (0-0) (ci.etl.go.jp/pub/soccer/client/RoboCup98/andhill98.RoboCup98.tar.gz) e PSI Team (1-2) (www.botik.ru/~soccer/Team-99/Team-PSI_1999.tar.gz) proporcionou uma comparação mais adequada, visto a composição e simplicidade do mesmo.

Também foram realizados testes mais individuais, onde colocou-se de 2 à 3 jogadores de cada time. Neste momento, teve-se um melhor rendimento do agente UFRGS. Esses testes demonstraram que a falta de coordenação entre os 11 jogadores acabava acarretando um desempenho em grupo muitas vezes indesejado. Em um grupo de três jogadores (sendo um goleiro), pode-se observar melhor as ações escolhidas pelo agente em uma dada situação. Percebeu-se uma maior facilidade do agente UFRGS em chegar até o gol do adversário, bem como, em procurar ter a posse de bola.

6.9 Outros Trabalhos

Vários trabalhos foram desenvolvidos com o objetivo de aprimorar o funcionamento do agente. Dentre eles, destaca-se o desenvolvimento de um sistema *fuzzy* para a otimização da energia dos jogadores. Esse sistema procura utilizar a energia da melhor forma possível durante toda a partida, monitorando a aceleração aplicada no comando *dash*. Para isso, leva-se em conta o tempo da partida, a energia e a prioridade (se a aceleração aplicada no *dash* for 100, por exemplo, a prioridade é considerada grande, caso contrário, pode ser considerada média ou pequena). Esses parâmetros podem indicar o esforço que deverá ser aplicado pelo jogador em uma ação de caminhada.

Da mesma forma, desenvolveu-se uma função capaz de calcular a aceleração do *dash*, baseada na energia do jogador. Ambas proporcionaram um melhor rendimento do jogador em campo, estabilizando a energia do mesmo.

Outra proposta para otimizar a funcionalidade do jogador, é a utilização do que se chamou de “comandos no escuro”. Durante a escolha de uma ação, seleciona-se também as

ações futuras que dão continuidade ao comportamento que está sendo executado pelo agente. No caso da falta de uma informação em um dado ciclo, a ação seguinte pode ser executada.

Outros trabalhos também auxiliaram na avaliação da implementação realizada. Foram desenvolvidos alguns trabalhos na disciplina de Sistemas Multiagentes do Programa de Pós-Graduação em Computação da UFRGS, nos quais os alunos utilizaram a implementação realizada para desenvolver seus próprios comportamentos.

Realizou-se uma análise preliminar do desempenho dos jogadores desenvolvidos. Ao longo do desenvolvimento do programa, as atitudes do jogador tornaram-se melhores, quando optou-se por implementações simplificadas. Por exemplo, em princípio, o comportamento de evitar o impedimento era controlado pela posição do atacante em relação ao time adversário. No entanto, tal abordagem mostrou-se de difícil implementação devido às limitações de tempo para o desenvolvimento do trabalho. Um comportamento alternativo foi elaborado, no qual utiliza-se o ângulo absoluto do jogador. Embora simples (e por simples) esse comportamento mostrou-se mais adequado, uma vez que proporciona bons resultados na maioria dos casos.

Pelo fato deste trabalho ter possibilitado o início de um estudo, especificação e implementação no domínio do Soccerserver, envolvendo várias tecnologias, a busca por um resultado eficiente, que pudesse ser desenvolvido em tempo hábil, foi potencialmente relevante na escolha de um agente reativo.

Por conseqüência do próprio domínio do simulador, como a sua característica de tempo-real, procurou-se possibilitar ao agente responder aos estímulos do ambiente de maneira simples e rápida.

É possível perceber que certos procedimentos manifestaram-se a partir da combinação dos comportamentos definidos ao agente. Embora não exista um processo de cooperação explícito entre os jogadores, cada um realiza suas ações, tendo conhecimento da situação de seus companheiros e adversários visíveis em campo. O jogador procura dividir com seus companheiros uma jogada.

7 Conclusões

O interesse na utilização da abordagem de agentes em diversas áreas provém da necessidade de aplicar novas técnicas para a construção de sistemas. Os SMA, há algum tempo, têm proporcionado resultados satisfatórios, principalmente em aplicações de sistemas reais.

Ambientes como o do simulador Soccerserver propiciam o desenvolvimento de SMA e a realização de testes de desempenho. O Soccerserver é um simulador de um jogo de futebol, trata-se de um ambiente dinâmico, capaz de adicionar características bastante realísticas em suas simulações. Além disso, permite a integração de diversas tecnologias.

O presente trabalho teve como objetivo realizar a proposta da arquitetura de um agente jogador de futebol e a implementação de um time para o simulador Soccerserver. Para tal, foi necessário estudar o simulador Soccerserver e as arquiteturas de clientes desenvolvidos para o mesmo.

A seguir, apresenta-se uma visão geral do trabalho, objetivando mostrar as contribuições geradas. Posteriormente, serão comentadas algumas possibilidades de trabalhos futuros.

Inicialmente, no Capítulo 1, teve-se o intuito de oferecer uma compreensão preliminar sobre o assunto abordado e, assim, situar o presente trabalho. Apresentou-se, no Capítulo 2, uma breve introdução à Inteligência Artificial Distribuída. Esse capítulo tratou de aspectos sobre a definição de agentes e de SMA. Apresentou-se, também, questões sobre a Arquitetura de Subsunção [BRO 86] (utilizada na arquitetura do agente UFRGS), e o assunto Funcionalidade Emergente (que auxiliou na compreensão das atitudes do agente que foram além das suas definições).

No Capítulo 3, discorreu-se sobre a RoboCup e realizou-se a análise de sua estrutura, com ênfase na categoria de simulação. Apresentou-se o futebol como um problema padrão para os SMA e as principais características do simulador oficial da RoboCup, o Soccerserver. Esse capítulo procurou demonstrar a complexidade e a utilidade de se desenvolver um agente para o simulador. Com esse propósito, foram focadas questões chaves desse ambiente como tempo-real, comunicação cliente-servidor, percepção limitada e incerteza de informações. Mostrou-se como o simulador vem sendo amplamente divulgado e utilizado por grupos de pesquisa do mundo inteiro, que se dedicam ao desenvolvimento de agentes jogadores para o Soccerserver, com o objetivo de demonstrar e testar suas tecnologias.

Posteriormente, no Capítulo 4, foi feito um estudo de clientes desenvolvidos para o simulador. Nesse momento, foram observadas diferentes arquiteturas de agentes para o Soccerserver e as melhorias que permitiram, com o tempo, aprimorar o desempenho das mesmas. Esse estudo foi de extrema importância, visto que possibilitou identificar aspectos

fundamentais que um agente deveria conter, além de fornecer respostas para dúvidas de como o agente deveria ser construído.

O ponto fundamental desse trabalho envolveu a proposta para o desenvolvimento de um agente simples e flexível, que possibilitou um primeiro contato com o simulador. O estudo realizado sobre os clientes, além das diversas discussões no grupo de trabalho, serviram para atingir o objetivo do trabalho que era a elaboração de uma nova arquitetura de um agente para o simulador. A arquitetura do agente UFRGS proposta foi apresentada no Capítulo 5.

Essa arquitetura é composta por quatro módulos responsáveis por realizar o *loop* do jogador, são eles: Interface de Entrada, Percepção do Mundo, Escolha da Ação e Interface de Saída. Foram identificados os fundamentos conceituais a cerca do desenvolvimento e das habilidades do agente com o objetivo de utilizá-los na criação do agente UFRGS. A idéia era concentrar-se no desenvolvimento inicial, utilizando uma abordagem clara e simples que permitisse o funcionamento do agente e possibilitasse com facilidade a inclusão de novas características ao mesmo. Parte desses fundamentos foram alcançados pelo agente através das seguintes decisões de projeto:

- Optou-se por uma arquitetura baseada em aspectos de agentes reativos que permitiu ao jogador desempenhar suas tarefas mais básicas de maneira adequada, em curto espaço de tempo. Embora se tenha buscado atribuir ao agente comportamentos mais sofisticados, o projeto, desde o princípio, dedicou-se aos primeiros passos do desenvolvimento. A escolha de trabalhar com agente reativos, ocorreu pelo fato de cada agente possuir um conjunto de comportamentos, que são ativados por estímulos no ambiente onde estão inseridos, não se preocupando em planejar e nem em negociar suas ações, mas agindo de forma autônoma.
- A comunicação entre o Soccerserver e o agente ocorre através do recebimento de uma informação de percepção e do envio de uma ação a ser realizada pelo jogador em campo. Durante o processo de entrada e saída o agente organiza o modelo do mundo e realiza o processamento interno em busca de um comportamento a ser efetuado no ambiente. Recursos de processamento concorrente possibilitaram ao agente responder às suas percepções do mundo em tempo-real, conforme as exigências do simulador.
- A representação do modelo do mundo do agente é simplificada, possibilitando facilidades no armazenamento das informações e a rápida recuperação das mesmas.
- Mecanismos internos ao agente como a fila de entrada e o relógio interno possibilitaram otimizar o funcionamento do mesmo. A fila de entrada impediu que as mensagens mais importantes vindas do servidor fossem perdidas. Já o relógio proporcionou uma forma de sincronizar as ações enviadas ao simulador.
- A definição dos comportamentos de forma independente permitiu modificar o funcionamento do agente alterando a ordem das regras existentes e, ainda, proporcionou uma forma facilitada para a inclusão ou exclusão de regras.

Contudo, a implementação dos comportamentos mais sofisticados ficou limitada aos recursos utilizados. Dado o limite de tempo definido para a confecção deste trabalho, foi impossível desenvolver funções mais robustas que pudessem lidar com a freqüente inconsistência das informações e, desta forma, aperfeiçoar o comportamento do agente. A razão da ausência dessas capacidades está relacionada à necessidade demasiadamente grande de pesquisa e implementação podendo, contudo, ser alvo de uma nova dissertação de mestrado.

Como vantagens da arquitetura UFRGS, pode-se citar:

- Apresenta um acesso rápido e fácil ao modelo do mundo do agente.
- Incorpora conhecimento prático em regras "se-então".
- A habilidade do agente cresce a taxa proporcional ao crescimento da base de regras de comportamento.
- Pode resolver um grande número de problemas de forma relativamente simples.

Após a realização deste, obteve-se uma visão das possíveis soluções (como: transformar uma informação visual de entrada em uma ação de saída do agente) e até mesmo das dificuldades (como: evitar que um agente fique a espera de uma informação visual para realizar uma ação) encontradas no desenvolvimento de um cliente, além de conhecer o contexto atual que abrange a área observada.

Certamente, a adição de características mais robustas pode possibilitar o aperfeiçoamento e preencher a lacuna de problemas encontrados durante a implementação do agente. Foi possível desenvolver toda a base necessária para que um agente pudesse estabelecer comunicação com o servidor, mecanismos de controle interno e, principalmente, implementar as idéias propostas pela arquitetura.

Cabe salientar que as idéias propostas são idéias iniciais, em que os componentes elementares buscam maior eficiência. Levando-se em consideração todos os pontos observados, pode-se constatar a possibilidade do desenvolvimento de um cliente para o simulador Soccerserver.

7.1 Trabalhos Futuros

Os resultados obtidos com a implementação do agente UFRGS estimulam a continuidade do projeto, tanto sob o ponto de agregar ao sistema recursos não implementados, tais como a inclusão da previsão do mundo, quanto a adicionar características mais cognitivas, como aprendizado. Visando a continuação do estudo, apresenta-se um maior aprofundamento da arquitetura proposta e a extensão de sua aplicabilidade, objetivando sua melhoria, como:

- Incorporar novas regras, possibilitando comportamentos mais complexos.
- Adicionar novas funções de cálculo, procurando obter resultados mais precisos.

- Aprimorar a arquitetura com técnicas mais sofisticadas, como: cooperação entre agentes, previsões para o estado do mundo, planejamento de jogadas e aprendizado ou adaptação ao comportamento do oponente.
- Proporcionar aos jogadores o uso de papéis flexíveis, permitindo que o jogador possa alterar o seu posicionamento em campo de acordo com as informações da situação corrente do jogo.
- Aprimorar a distribuição dos jogadores em campo através da definição de formações. Os jogadores podem realizar uma análise da situação do jogo. Conhecendo a situação e posição de seus companheiros em campo, o jogador então pode identificar a formação corrente.
- Desenvolver o agente treinador, juntamente com a definição de "jogadas ensaiadas" para o time, buscando tirar proveito da capacidade de visão global provida pelo Soccerserver para esse tipo de agente. Isto permitirá uma melhora no posicionamento do time em campo, buscando aproveitar melhor os espaços livres. O treinador ainda pode sanar deficiências de marcação, selecionando um posicionamento de marcação mais eficaz, bem como, melhorar a tática de posicionamento de ataque. Outra vantagem é a possibilidade de antecipar os movimentos do adversário, analisando o jogo realizado pelo mesmo.
- Utilizar as ferramentas de análise de jogo, como: Mike, LogMonitor e o Issac, para interpretar as ações realizadas pelos agentes e avaliar o time. Estas ferramentas podem auxiliar no aprimoramento da colaboração entre os jogadores de um mesmo time.

7.2 Contribuição

Ao atingir os objetivos propostos para o presente trabalho, foi obtido um cliente para o simulador Socceserver. Isso tornou possível o confronto com os clientes desenvolvidos para o Soccerserver, bem como, proporcionou o intercâmbio de idéias através do contato com vários grupos de pesquisa espalhados pelo mundo e que fazem parte dessa iniciativa.

O fato deste trabalho ter sido pioneiro na UFRGS no desenvolvimento de um agente para o Soccerserver, possibilitou uma ampla e clara visão das vantagens e da riqueza de detalhes proporcionadas pelo ambiente para pesquisas na área de SMA que se dedicam a criação de agentes capazes de percepção parcial e de agirem em seu mundo, cooperando com o seu grupo.

As preocupações constantes com o aspecto de eficiência levaram à proposta de uma arquitetura simplificada. Durante os testes, foi possível constatar a rapidez de processamento do agente. Grande parte do tempo disponível ao agente foi perdida na espera entre a escolha da ação e o envio da mesma ao simulador. Isso significa que o tempo de espera poderia ser utilizado por um processamento de alto nível, como por exemplo: na elaboração de uma estratégia em grupo.

Um aspecto importante é que grande parte das pesquisas da IA e, principalmente, de SMA podem ser realizadas com o simulador. O desenvolvimento de um cliente também proporcionou um ambiente para aplicação e avaliação de SMA em tempo real, promovendo pesquisas nessa área, com a oportunidade de demonstrar resultados.

A maior contribuição obtida, com o presente trabalho, acredita-se ter sido, a definição da arquitetura (como: especificação do modelo do mundo, elaboração da base de comportamento, variáveis de controle, definição da estrutura de previsão) do agente e a implementação da base necessária para a criação dos agentes. Grande parte do tempo deste trabalho foi dedicada a definição e a criação das tecnologias necessárias (como: comunicação, processamento concorrente, interpretação de mensagens, controle dos ciclos de simulação, fila de entrada) para o funcionamento do agente.

Anexo 1 Exemplo de utilização da fila de entrada

RPAC (buffer de recebimento) na TRecebimento: (hear 13 referee play_on)
Fila[0]: (hear 13 referee play_onull)
Fila[1]: null
Fila[2]: null
RPAC.msg na principal: (hear 13 referee play_on)

RPAC (buffer de recebimento) na TRecebimento: (sense_body 13 (view_mode high normal) (stamina 3500 1) (speed 0) (head_angle 0) (kick 1) (dash 8) (turn 0) (say 0) (turn_neck 0))
Fila[0]: null
Fila[1]: null
Fila[2]: null
RPAC.msg na principal: (sense_body 13 (view_mode high normal) (stamina 3500 1) (speed 0) (head_angle 0) (kick 1) (dash 8) (turn 0) (say 0) (turn_neck 0))

RPAC (buffer de recebimento) na TRecebimento: (see 13 ((goal r) 53 0) ((flag c) 0.4 3 -0.008 0) ((flag r t) 62.8 -32) ((flag r b) 62.8 32) ((flag p r t) 41.7 -28) ((flag p r c) 36.2 0) ((flag p r b) 41.7 29) ((flag g r t) 53.5 -7) ((flag g r b) 53.5 7) ((flag t r 40) 56.3 -43) ((flag t r 50) 63.4 -37) ((flag b r 40) 56.3 43) ((flag b r 50) 64.1 37) ((flag r t 30) 65.4 -27) ((flag r t 20) 61.6 -19) ((flag r t 10) 58.6 -9) ((flag r 0) 58 0) ((flag r b 10) 58.6 9) ((flag r b 20) 61.6 19) ((flag r b 30) 65.4 27) ((ball) 0.5 2 0.07 -0.4) ((player) 49.4 -11) ((player ufrgs) 36.6 0) ((player ufrgs) 36.6 -15) ((player ufrgs 4) 36.6 15 -0 0 156 156) ((player ufrgs) 40.4 42) ((player ufrgs 6) 24.5 -44 -0 -0 143 143) ((player ufrgs 7) 14.9 0 -0 -0 163 163) ((player ufrgs 8) 24.5 44 -0 0 -118 -118) ((player ufrgs 10) 10 0 -0 -0 156 156) ((line r) 53 -89))
Fila[0]: (see 13 ((goal r) 53 0) ((flag c) 0.4 3 -0.008 0) ((flag r t) 62.8 -32) ((flag r b) 62.8 32) ((flag p r t) 41.7 -28) ((flag p r c) 36.2 0) ((flag p r b) 41.7 29) ((flag g r t) 53.5 -7) ((flag g r b) 53.5 7) ((flag t r 40) 56.3 -43) ((flag t r 50) 63.4 -37) ((flag b r 40) 56.3 43) ((flag b r 50) 64.1 37) ((flag r t 30) 65.4 -27) ((flag r t 20) 61.6 -19) ((flag r t 10) 58.6 -9) ((flag r 0) 58 0) ((flag r b 10) 58.6 9) ((flag r b 20) 61.6 19) ((flag r b 30) 65.4 27) ((ball) 0.5 2 0.07 -0.4) ((player) 49.4 -11) ((player ufrgs) 36.6 0) ((player ufrgs) 36.6 -15) ((player ufrgs 4) 36.6 15 -0 0 156 156) ((player ufrgs) 40.4 42) ((player ufrgs 6) 24.5 -44 -0 -0 143 143) ((player ufrgs 7) 14.9 0 -0 -0 163 163) ((player ufrgs 8) 24.5 44 -0 0 -118 -118) ((player ufrgs 10) 10 0 -0 -0 156 156) ((line r) 53 -89))
Fila[1]: NULL
Fila[2]: NULL
RPAC.msg na principal: (see 13 ((goal r) 53 0) ((flag c) 0.4 3 -0.008 0) ((flag r t) 62.8 -32) ((flag r b) 62.8 32) ((flag p r t) 41.7 -28) ((flag p r c) 36.2 0) ((flag p r b) 41.7 29) ((flag g r t) 53.5 -7) ((flag g r b) 53.5 7) ((flag t r 40) 56.3 -43) ((flag t r 50) 63.4 -37) ((flag b r 40) 56.3 43) ((flag b r 50) 64.1 37) ((flag r t 30) 65.4 -27) ((flag r t 20) 61.6 -19) ((flag r t 10) 58.6 -9) ((flag r 0) 58 0) ((flag r b 10) 58.6 9) ((flag r b 20) 61.6 19) ((flag r b 30) 65.4 27) ((ball) 0.5 2 0.07 -0.4) ((player) 49.4 -11) ((player ufrgs) 36.6 0) ((player ufrgs) 36.6 -15) ((player ufrgs 4) 36.6 15 -0 0 156 156) ((player ufrgs) 40.4 42) ((player ufrgs 6) 24.5 -44 -0 -0 143 143) ((player ufrgs 7) 14.9 0 -0 -0 163 163) ((player ufrgs 8) 24.5 44 -0 0 -118 -118) ((player ufrgs 10) 10 0 -0 -0 156 156) ((line r) 53 -89))

RPAC (buffer de recebimento) na TRecebimento: (sense_body 14 (view_mode high normal) (stamina 3500 1) (speed 0) (head_angle 0) (kick 2) (dash 8) (turn 0) (say 0) (turn_neck 0))
Fila[0]: NULL
Fila[1]: NULL
Fila[2]: NULL
RPAC.msg na principal: (sense_body 14 (view_mode high normal) (stamina 3500 1) (speed 0) (head_angle 0) (kick 2) (dash 8) (turn 0) (say 0) (turn_neck 0))

RPAC (buffer de recebimento) na TRecebimento: (see 14 ((goal r) 53 0) ((flag c) 0.4 3 -0 -0) ((flag r t) 62.8 -32) ((flag r b) 62.8 32) ((flag p r t) 41.7 -28) ((flag p r c) 36.2 0 -0 -0) ((flag p r b) 41.7 29) ((flag g r t) 53.5 -7) ((flag g r b) 53.5 7) ((flag t r 40) 56.3 -43) ((flag t r 50) 63.4 -37) ((flag b r 40) 56.3 44) ((flag b r 50) 63.4 37) ((flag r t 30) 65.4 -27) ((flag r t 20) 61.6 -19) ((flag r t 10) 58.6 -9) ((flag r 0) 58 0) ((flag r b 10) 58.6 9) ((flag r b 20) 61.6 19) ((flag r b 30) 65.4 27) ((ball) 0.7 2 0.154 -0.2) ((player ufrgs) 49.4 -11) ((player ufrgs) 36.6 0) ((player ufrgs) 36.6 -15) ((player ufrgs) 36.6 15) ((player ufrgs) 40.4 42) ((player ufrgs 6) 24.5 -44 -0 -0 143 143) ((player ufrgs 7) 14.9 0 -0 -0 163 163) ((player ufrgs) 24.5 44) ((player ufrgs 10) 10 0 -0 -0 156 156) ((line r) 53 -89))
Fila[0]: (see 14 ((goal r) 53 0) ((flag c) 0.4 3 -0 -0) ((flag r t) 62.8 -32) ((flag r b) 62.8 32) ((flag p r t) 41.7 -28) ((flag p r c) 36.2 0 -0 -0) ((flag p r b) 41.7 29) ((flag g r t) 53.5 -7) ((flag g r b) 53.5 7) ((flag t r 40) 56.3 -43) ((flag t r 50) 63.4 -37) ((flag b r 40) 56.3 44) ((flag b r 50) 63.4 37) ((flag r t 30) 65.4 -27) ((flag r t 20) 61.6 -19) ((flag r t 10) 58.6 -9) ((flag r 0) 58 0) ((flag r b 10) 58.6 9) ((flag r b 20) 61.6 19) ((flag r b 30) 65.4 27) ((ball) 0.7 2 0.154 -0.2) ((player ufrgs) 49.4 -11) ((player ufrgs) 36.6 0) ((player ufrgs) 36.6 -15) ((player ufrgs) 36.6 15) ((player ufrgs) 40.4 42) ((player ufrgs 6) 24.5 -44 -0 -0 143 143) ((player ufrgs 7) 14.9 0 -0 -0 163 163) ((player ufrgs) 24.5 44) ((player ufrgs 10) 10 0 -0 -0 156 156) ((line r) 53 -89))
Fila[1]: NULL
Fila[2]: NULL
RPAC.msg na principal: (see 14 ((goal r) 53 0) ((flag c) 0.4 3 -0 -0) ((flag r t) 62.8 -32) ((flag r b) 62.8 32) ((flag p r t) 41.7 -28) ((flag p r c) 36.2 0 -0 -0) ((flag p r b) 41.7 29) ((flag g r t) 53.5 -7) ((flag g r b) 53.5 7) ((flag t r 40) 56.3 -43) ((flag t r 50) 63.4 -37) ((flag b r 40) 56.3 44) ((flag b r 50) 63.4 37) ((flag r t 30) 65.4 -27) ((flag r t 20) 61.6 -19) ((flag r t 10) 58.6 -9) ((flag r 0) 58 0) ((flag r b 10) 58.6 9) ((flag r b 20) 61.6 19) ((flag r b 30) 65.4 27) ((ball) 0.7 2 0.154 -0.2) ((player ufrgs) 49.4 -11) ((player ufrgs) 36.6 0) ((player ufrgs) 36.6 -15) ((player ufrgs) 36.6 15) ((player ufrgs) 40.4 42) ((player ufrgs 6) 24.5 -44 -0 -0 143 143) ((player ufrgs 7) 14.9 0 -0 -0 163 163) ((player ufrgs) 24.5 44) ((player ufrgs 10) 10 0 -0 -0 156 156) ((line r) 53 -89))

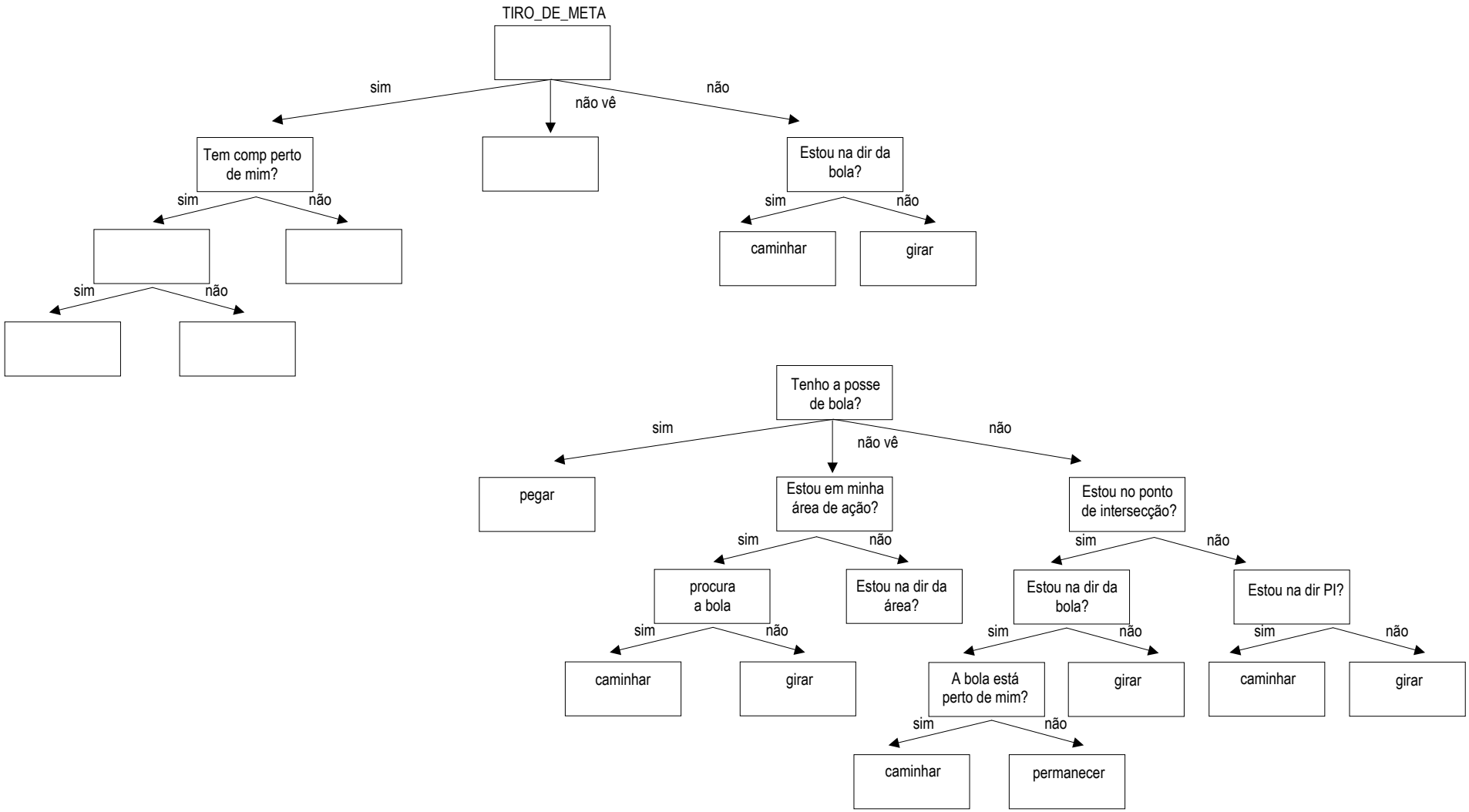
RPAC (buffer de recebimento) na TRecebimento: (sense_body 15 ...)
Fila[0]: NULL
Fila[1]: NULL
Fila[2]: NULL
RPAC.msg na principal: (sense_body 15 ...)

RPAC (buffer de recebimento) na TRecebimento: (sense_body 16 ...)
Fila[0]: null
Fila[1]: NULL
Fila[2]: NULL
RPAC.msg na principal: (sense_body 16 ...)

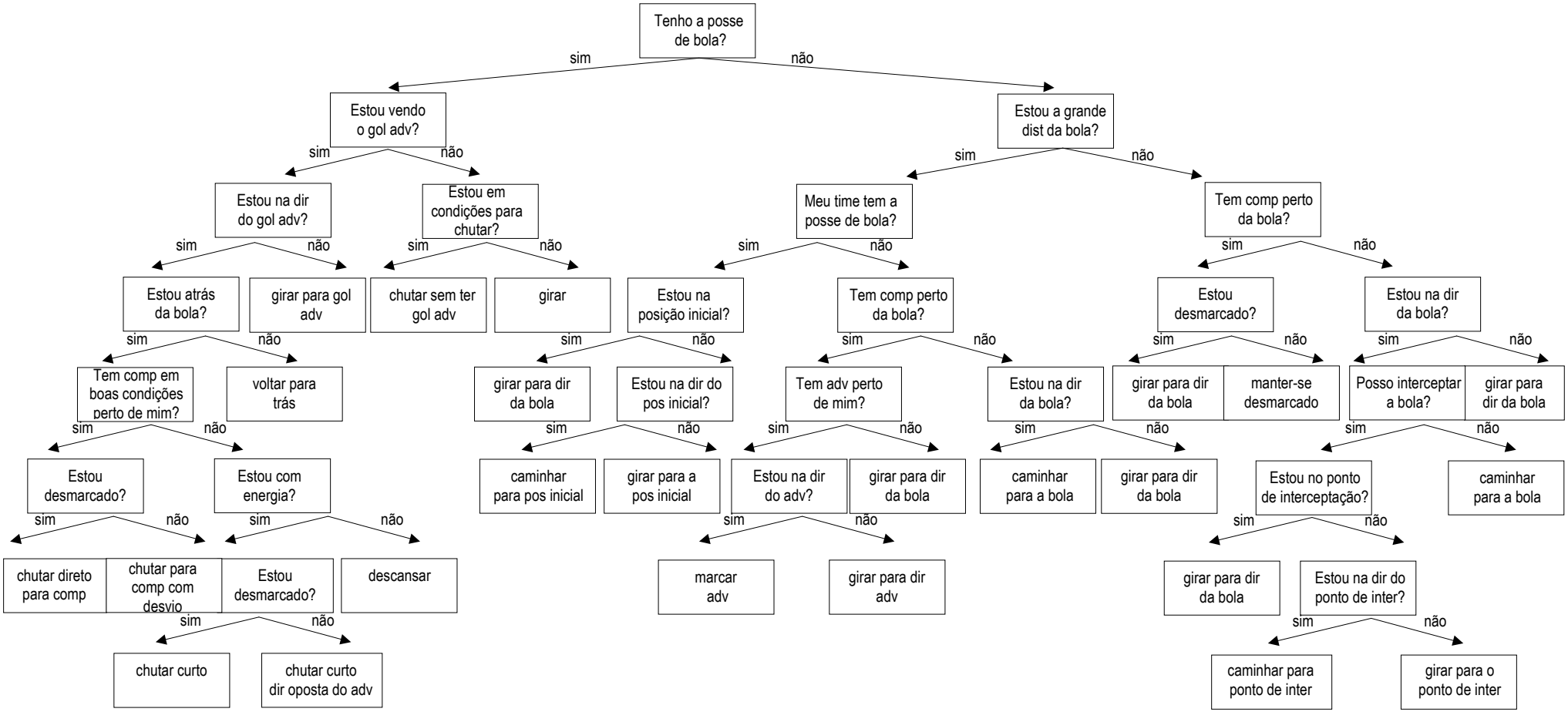
RPAC (buffer de recebimento) na TRecebimento: (see 17 ...)
Fila[0]: (see 17)
Fila[1]: NULL
Fila[2]: NULL
RPAC.msg na principal: (see 17 ...)

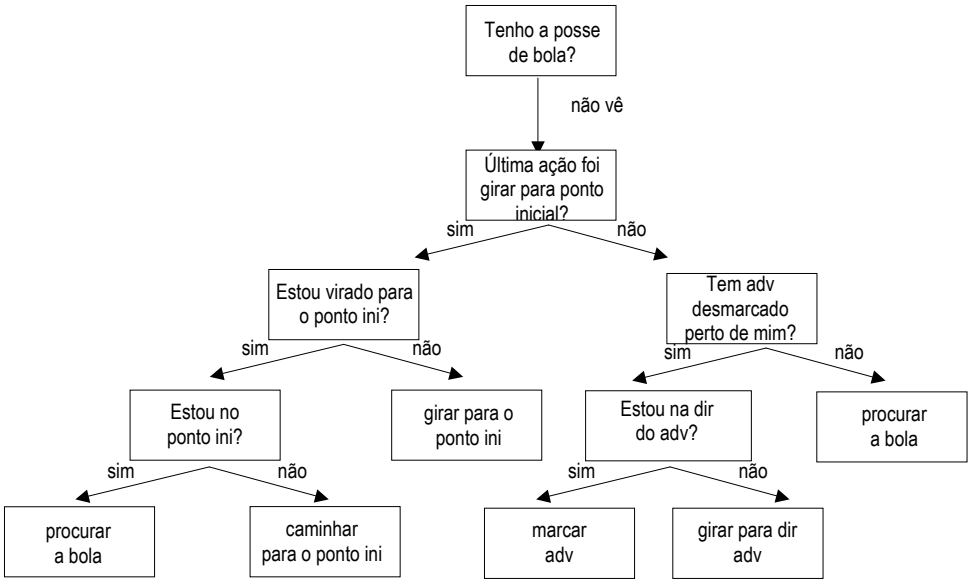
RPAC (buffer de recebimento) na TRecebimento: (sense_body 17 ...)
Fila[0]: NULL
Fila[1]: NULL
Fila[2]: NULL
RPAC.msg na principal: (sense_body 17 ...)

Anexo 2 Árvore de decisão para o comportamento do jogador goleiro.

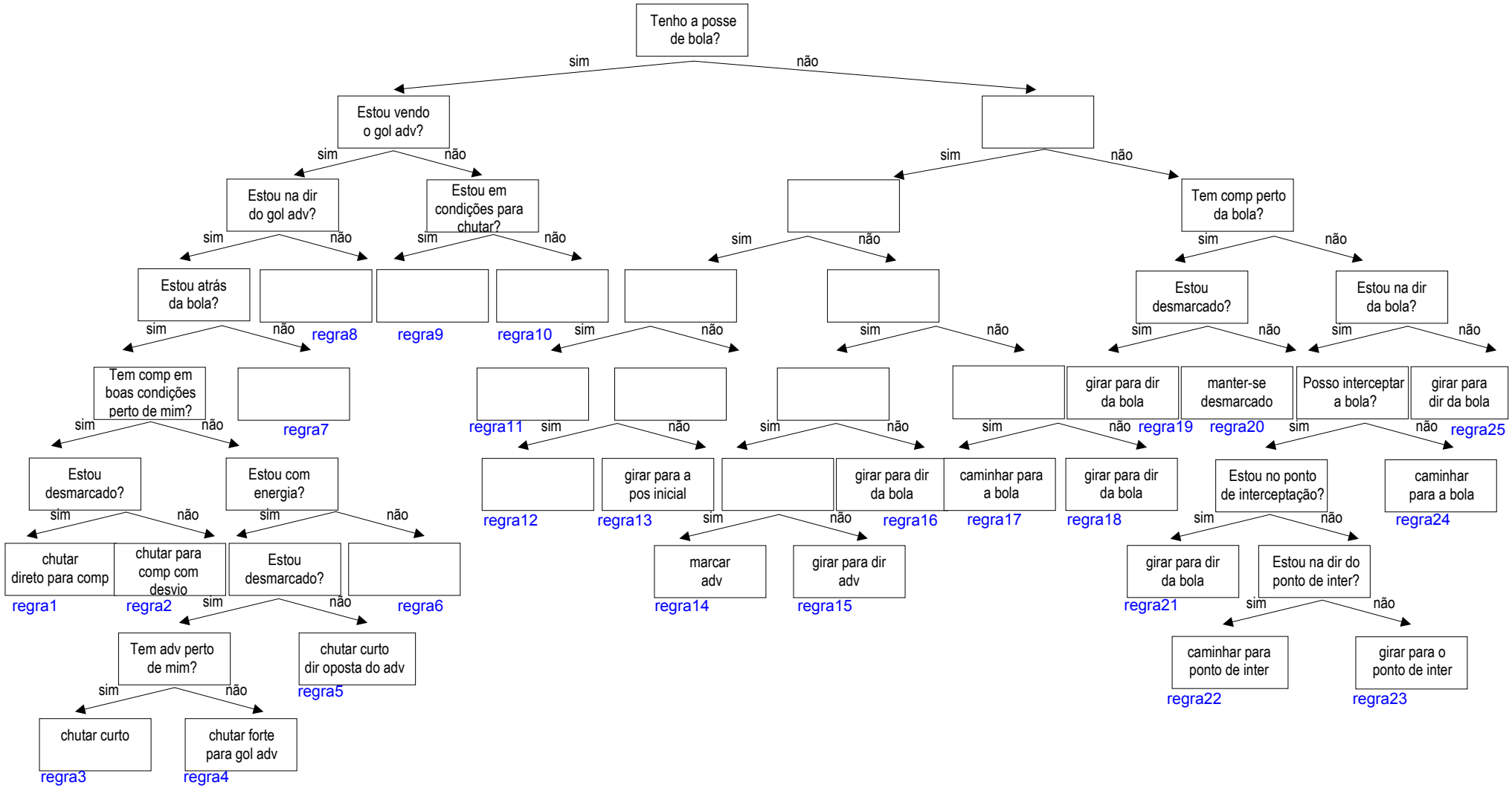


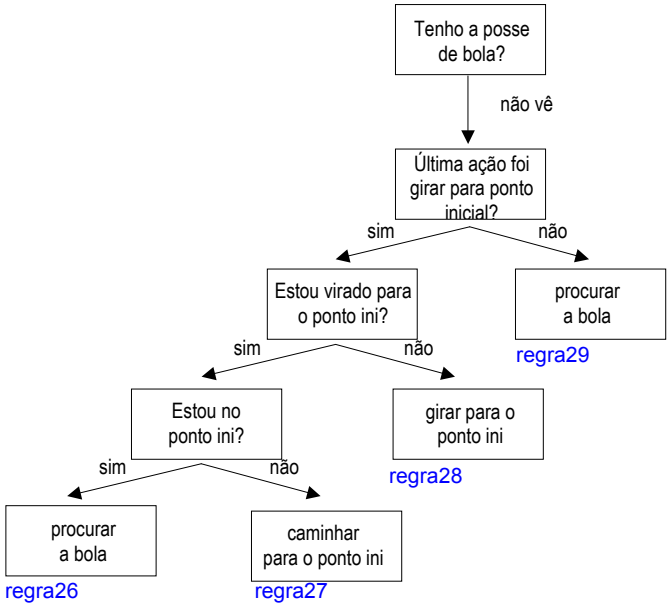
Anexo 3 Árvore de decisão para o comportamento do jogador zagueiro.



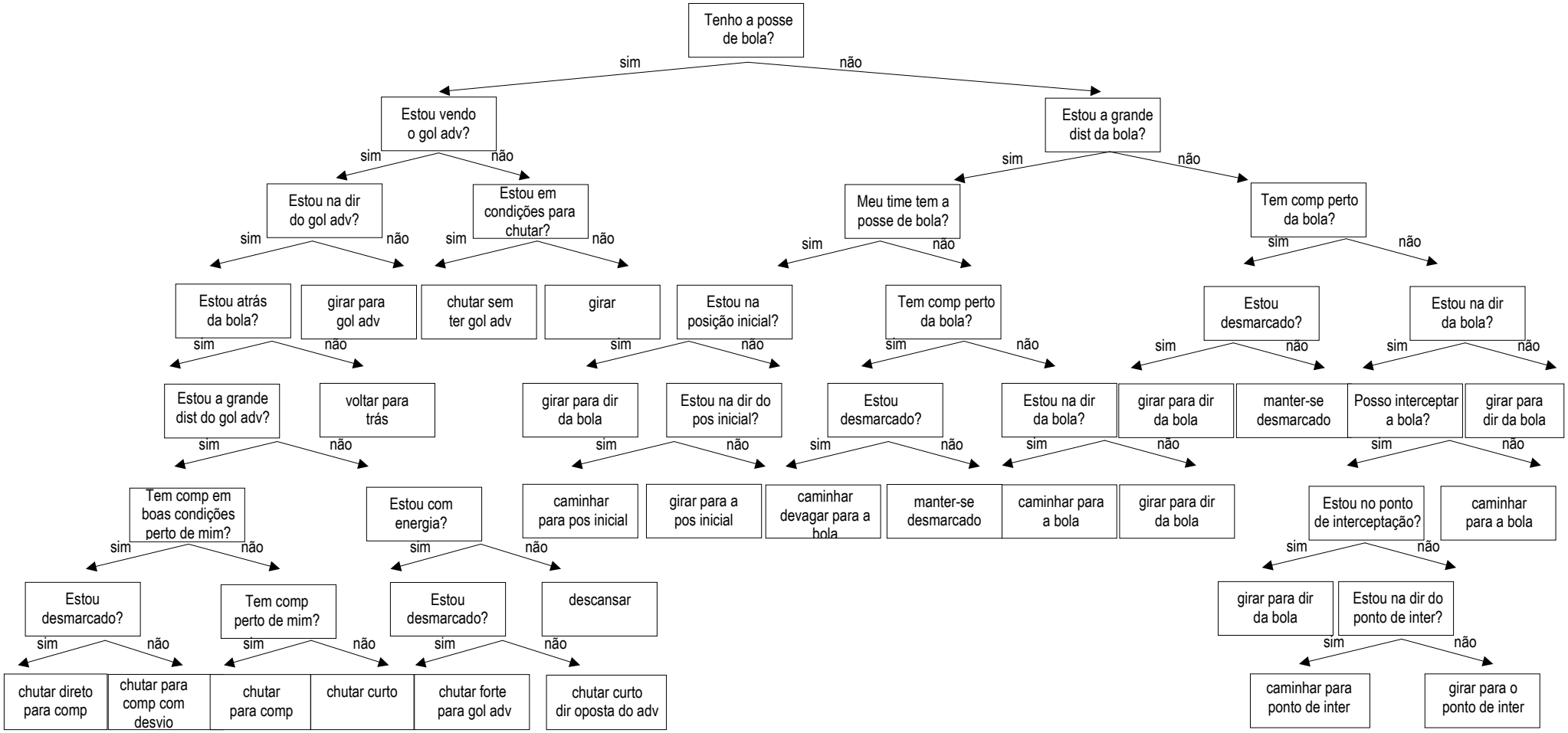

Tenho a posse de bola?
não vê
Última ação foi girar para ponto inicial?
sim
não
Estou virado para o ponto ini?
Tem adv desmarcado perto de mim?
sim
não
sim
não
Estou no ponto ini?
girar para o ponto ini
Estou na dir do adv?
procurar a bola
sim
não
sim
não
procurar a bola
caminhar para o ponto ini
marcar adv
girar para dir adv

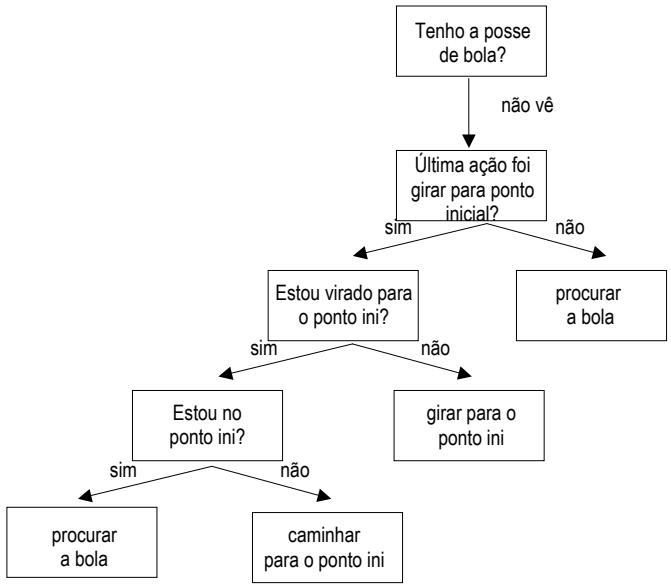
Anexo 4 Árvore de decisão para o comportamento do jogador meio de campo.




Tenho a posse de bola?
não vê
Última ação foi girar para ponto inicial?
sim
não
Estou virado para o ponto ini?
procurar a bola
regra29
sim
não
Estou no ponto ini?
girar para o ponto ini
regra28
sim
não
procurar a bola
caminhar para o ponto ini
regra26
regra27

Anexo 5 Árvore de decisão para o comportamento do jogador atacante.




Tenho a posse de bola?
não vê
Última ação foi girar para ponto inicial?
sim
não
Estou virado para o ponto ini?
procurar a bola
sim
não
Estou no ponto ini?
girar para o ponto ini
sim
não
procurar a bola
caminhar para o ponto ini

Anexo 6 Escolha da ação para o comportamento do jogador meio de campo.

```
/************************************************************/
/* Modulo Escolha da Ação DO MEIO DE CAMPO                  */
/* Descricao: Define os comportamentos do agente            */
/* Data Atualizacao: 07/02/2001                             */
/* Funcao escolha_acao                                      */
/*      Define a acao a ser executada pelo jogador.         */
/* Parametros de entrada: buffer, estrutura tempo_ptr (modelo*/
/* do mundo do agente), estrutura eu (informacoes do proprio */
/* jogador) e estrutura de varivaies de controle.           */
/* Parametros de saida: resp - ok ou erro.                  */
/************************************************************/
int escolha_acao(char *buffer, struct tipo_objetos_tempo *tempo_ptr, struct tipo_eu eu, struct variaveis var, struct tipo_bola
*tempo_ptr_bola)
{
       int   resp; /* ok ou erro */
       resp = comportamentos(buffer,tempo_ptr,eu,var,tempo_ptr_bola,tempo_ptr_bola[0].dist_PI,tempo_ptr_bola[0].dir_PI);
       return(resp);
}

//_____________________________________________________________________________< Regras >
//  COMPORTAMENTOS DO JOGADOR MEIO DE CAMPO
int comportamentos(char *buffer, struct tipo_objetos_tempo *tempo_ptr, struct tipo_eu eu, struct variaveis var, struct tipo_bola
*tempo_ptr_bola, float dist_PI, float dir_PI)
{
       struct tipo_coordenada posicao_inter;
       struct tipo_bola jogador;
       struct tipo_bola inter_dist_dir;
       struct tipo_ponto ponto_interceptacao;
       int   resp; /* 0 = erro   1 = ok */
       float dir_check;
       float velocidade;
       char  proximo_comando[255];
       int   indice_do_mais_perto = TRASH;
       float dir_marcacao = NAO_TEM;

       ClearNextCommand();
       SetAfterDash(1); // liga e desliga comandos no escuro
       SetAfterTurn(0);
       SetAfterKick(0);
```

```
/*************************************************************/
/************************ Tenho a Bola ***********************/
/*************************************************************/
/* REGRA 1_____________________________________________________________________________________________________________*/
        indice_do_mais_perto = acha_jogador_para_passe(var);
/* REGRA 1 Chutar direto para o companheiro____________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && ((DIR_BOLA <= 90.) && (DIR_BOLA >= -90.)) && ((DIR_GOL_ADV <=
        DIR_GIRO) && (DIR_GOL_ADV >= -DIR_GIRO)) && (indice_do_mais_perto!= TRASH) && (var.estou_desmarcado != FALSE))
        {
                if(tempo_ptr[0].objeto[var.comp_perto_de_mim[indice_do_mais_perto].posicao_modelo].distancia < 10.)
                        resp = envia_kick(tempo_ptr[0].objeto[var.comp_perto_de_mim[indice_do_mais_perto].posicao_modelo].distancia*10.,
                        tempo_ptr[0].objeto[var.comp_desmarcado_perto_de_mim[indice_do_mais_perto].posicao_modelo].direcao, buffer);
                else
                        resp = envia_kick(tempo_ptr[0].objeto[var.comp_perto_de_mim[indice_do_mais_perto].posicao_modelo].distancia*5.,
                tempo_ptr[0].objeto[var.comp_desmarcado_perto_de_mim[indice_do_mais_perto].posicao_modelo].direcao, buffer);
                ultima_acao = 1;
                return(resp);
        }

/* REGRA 2 Chutar para o companheiro com desvio________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && ((DIR_BOLA <= 90.) && (DIR_BOLA >= -90.)) && ((DIR_GOL_ADV <=
        DIR_GIRO) && (DIR_GOL_ADV >= -DIR_GIRO)) && (indice_do_mais_perto!= TRASH) && (var.estou_desmarcado == FALSE))
        {
                resp = envia_kick(tempo_ptr[0].objeto[var.comp_perto_de_mim[indice_do_mais_perto].posicao_modelo].distancia*7.,
                tempo_ptr[0].objeto[var.comp_desmarcado_perto_de_mim[indice_do_mais_perto].posicao_modelo].direcao +
                (random()%10)*((random()%2)*2-1), buffer);
                ultima_acao = 2;
                return(resp);
        }

/* REGRA 3 Chutar curto________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && ((DIR_BOLA <= 90.) && (DIR_BOLA >= -90.)) && ((DIR_GOL_ADV <=
        DIR_GIRO) && (DIR_GOL_ADV >= -DIR_GIRO)) && (indice_do_mais_perto == TRASH) && (eu.tempo_eu_ptr[0].sense_body.energia >= MIN_ENERGIA)
        && (var.estou_desmarcado != FALSE) && (DIST_ADV_PERTO != naotem))
        {
                resp = envia_kick(20,(DIR_GOL_ADV+random()%5), buffer);
                ultima_acao = 3;
                SetAfterKick(1);
                sprintf(proximo_comando,"(dash %.2f)",corre_para_bola((DIST_BOLA+3), (DIST_BOLA+5), eu.tempo_eu_ptr[0].sense_body.energia));
                SetAfterKickCommand(proximo_comando);
                return(resp);
        }

/* REGRA 4 Chutar forte para dir_gol_adv_______________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && ((DIR_BOLA <= 90.) && (DIR_BOLA >= -90.)) && ((DIR_GOL_ADV <=
        DIR_GIRO) && (DIR_GOL_ADV >= -DIR_GIRO)) && (indice_do_mais_perto == TRASH) && (eu.tempo_eu_ptr[0].sense_body.energia >= MIN_ENERGIA)
        && (var.estou_desmarcado != FALSE) && (DIST_ADV_PERTO == naotem))
        {
                resp = envia_kick(80,(DIR_GOL_ADV+random()%5), buffer);
                ultima_acao = 4;
                SetAfterKick(1);
```

```
                sprintf(proximo_comando,"(dash %.2f)",corre_para_bola((DIST_BOLA+3), (DIST_BOLA+5), eu.tempo_eu_ptr[0].sense_body.energia));
                SetAfterKickCommand(proximo_comando);
                return(resp);
        }

/* REGRA 5 Chutar curto em direcao oposta ao adversario_____________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && ((DIR_BOLA <= 90.) && (DIR_BOLA >= -90.)) && ((DIR_GOL_ADV <=
        DIR_GIRO) && (DIR_GOL_ADV >= -DIR_GIRO)) && (indice_do_mais_perto == TRASH) && (eu.tempo_eu_ptr[0].sense_body.energia >= MIN_ENERGIA)
        && (var.estou_desmarcado == FALSE) && (DIST_ADV_PERTO == naotem))
        {
                resp = envia_kick(KICKABLEAREA*5.0, tempo_ptr[0].objeto[var.adv_perto_de_mim[0].posicao_modelo].direcao*(-1), buffer);
                ultima_acao = 5;
                SetAfterKick(1);
                sprintf(proximo_comando,"(turn %.2f)",KICKABLEAREA*2.5);
                SetAfterKickCommand(proximo_comando);
                return(resp);
        }

/* REGRA 6 Descansar _______________________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && ((DIR_BOLA <= 90.) && (DIR_BOLA >= -90.)) && ((DIR_GOL_ADV <=
        DIR_GIRO) && (DIR_GOL_ADV >= -DIR_GIRO)) && (indice_do_mais_perto == TRASH) && (eu.tempo_eu_ptr[0].sense_body.energia < MIN_ENERGIA))
        {
                resp = envia_dash(0, buffer);
                ultima_acao = 6;
                return(resp);
        }

/* REGRA 7 Voltar pra tras_________________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && ((DIR_GOL_ADV <= DIR_GIRO) && (DIR_GOL_ADV >= -DIR_GIRO)) &&
        ((DIR_BOLA >= 90.) || (DIR_BOLA <= -90.)))
        {
                resp = envia_dash(-20,buffer);
                ultima_acao = 7;
                return(resp);
        }

/* REGRA 8 Girar para dir_gol_adv__________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && ((DIR_GOL_ADV >= DIR_GIRO) || (DIR_GOL_ADV <= -DIR_GIRO))
        && (DIR_GOL_ADV != naotem))
        {
                resp = envia_turn(DIR_GOL_ADV, eu.tempo_eu_ptr[0].sense_body.velocidade,buffer);
                ultima_acao = 8;
                SetAfterTurn(1);
                sprintf(proximo_comando,"(dash %.2f)",20);
                SetAfterTurnCommand(proximo_comando);
                return(resp);
        }

/* REGRA 9 _________________________________________________________________________________________________________________________*/
dir_check  = verifica_dir_ang_abs_vs_ang_bola(eu.ang_abs_jog, DIR_BOLA);
velocidade = VEscBola(DIST_BOLA,tempo_ptr[1].objeto[posi_bola_no_modelo].distancia);
```

```
/* REGRA 9 Chutar curto sem ter dir_gol_adv_____________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && (DIR_GOL_ADV == naotem) && (dir_check == naotem))
        {
                resp = envia_kick(20,chuta_sem_dir_gol_adv(eu.ang_abs_jog, DIR_BOLA, POS_X, POS_Y),buffer);
                ultima_acao = 9;
                SetAfterKick(1);
                sprintf(proximo_comando,"(turn %.2f)",(chuta_sem_dir_gol_adv(eu.ang_abs_jog, DIR_BOLA, POS_X, POS_Y))/3);
                SetAfterKickCommand(proximo_comando);
                return(resp);
        }

/* REGRA 10 Girar para ajuste _________________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == TRUE) && (DIR_GOL_ADV == naotem) && (dir_check != naotem))
        {
                resp = envia_turn(dir_check, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
                ultima_acao = 10;
                return(resp);
        }


/************************************************************/
/******************** Não tenho a Bola **********************/
/************************************************************/
/* REGRA 11 _________________________________________________________________________________________________________________________*/
posicao_inter = calcula_coordenada_PI(tempo_ptr_bola[0].pos_bola_x,tempo_ptr_bola[0].pos_bola_y,15,eu.numero);

/* REGRA 11 Girar para a bola_________________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA >= 27) && (var.meu_time_tem_posse_de_bola == TRUE) &&
        (((POS_X >= (posicao_inter.x - 5)) && (POS_X <= (posicao_inter.x + 5))) && ((POS_Y >= (posicao_inter.y - 5)) &&
        (POS_Y <= (posicao_inter.y + 5)))) && (DIR_BOLA != NAO_TEM))
        {
                resp = envia_turn(DIR_BOLA, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
                ultima_acao = 11;
                return(resp);
        }

/* REGRA 12 _________________________________________________________________________________________________________________________*/
jogador = calcula_onde_vai_jogador_ad(posicao_inter.x,posicao_inter.y,POS_X,POS_Y,eu.ang_abs_jog);

/* REGRA 12 Caminhar para a posicao inicial___________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA >= 15.) && (var.meu_time_tem_posse_de_bola)  &&
        (jogador.dist_PI > 5.) && ((jogador.dir_PI >= -DIR_GIRO) && (jogador.dir_PI <= DIR_GIRO)))
        {
                resp = envia_dash(corre_para_bola(jogador.dist_PI,jogador.dist_PI,eu.tempo_eu_ptr[0].sense_body.energia),buffer);
                ultima_acao = 12;
                return(resp);
        }

/* REGRA 13 Girar para a posicao inicical _____________________________________________________________________________________________*/
```

```
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA >= 15.) && (var.meu_time_tem_posse_de_bola == TRUE)
        && (jogador.dist_PI > 5.) && ((jogador.dir_PI < - DIR_GIRO) || (jogador.dir_PI > DIR_GIRO)))
        {
                resp = envia_turn(jogador.dir_PI, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
                ultima_acao = 13;
                if(jogador.dist_PI > 5.)
                {
                        //SetAfterTurn(1);
                        sprintf(proximo_comando,"(dash %.2f)",corre_para_bola(jogador.dist_PI,
                        jogador.dist_PI,eu.tempo_eu_ptr[0].sense_body.energia));
                        SetAfterTurnCommand(proximo_comando);
                }
                return(resp);
        }

/* REGRA 14 ____________________________________________________________________________________________________________________*/
        if(var.adv_desmarcado_perto_de_mim[0].numero != TRASH)
         if ((DIR_BOLA != NAO_TEM) && (var.adversario_desmarcado[0].posicao_modelo != TRASH)
         && (var.adversario_desmarcado[0].numero != NAO_TEM))
         {
                dir_marcacao = acha_dir_marcacao(DIR_BOLA, tempo_ptr[0].objeto[var.adversario_desmarcado[0].posicao_modelo].direcao,
                                        tempo_ptr[0].objeto[var.adversario_desmarcado[0].posicao_modelo].distancia);
        }

/* REGRA 14 Marcar o adversario_________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar != TRUE) && (DIST_BOLA >= 27.) && (var.meu_time_tem_posse_de_bola == 0)
        && (var.companheiro_perto_da_bola[0].posicao_modelo != TRASH) && (var.adversario_desmarcado[0].posicao_modelo != TRASH)
        && (var.adversario_desmarcado[0].numero != NAO_TEM) && ((dir_marcacao <= DIR_GIRO_GRANDE) && (dir_marcacao >= -DIR_GIRO_GRANDE))
        && (tempo_ptr[0].objeto[var.adversario_desmarcado[0].posicao_modelo].distancia > 5.) && (DIST_BOLA != NAO_TEM))
        {
                resp = envia_dash(corre_para_bola(tempo_ptr[0].objeto[var.adversario_desmarcado[0].posicao_modelo].distancia,
                                                tempo_ptr[1].objeto[var.adversario_desmarcado[0].posicao_modelo].distancia,
                                                eu.tempo_eu_ptr[0].sense_body.energia),buffer) ;
                ultima_acao = 14;
                return(resp);
        }

/* REGRA 15 Girar para o adversario_____________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar != TRUE) && (DIST_BOLA >= 27.) && (var.meu_time_tem_posse_de_bola == 0)
        && (var.companheiro_perto_da_bola[0].posicao_modelo != TRASH) && (var.adversario_desmarcado[0].posicao_modelo != TRASH)
        && (var.adversario_desmarcado[0].numero != NAO_TEM) && (DIST_BOLA != NAO_TEM) && ((dir_marcacao > DIR_GIRO_GRANDE) ||
        (dir_marcacao < -DIR_GIRO_GRANDE)))
        {
                ultima_acao = 15;
                if(tempo_ptr[0].objeto[var.adversario_desmarcado[0].posicao_modelo].distancia > 3.)
                {
                        SetAfterTurn(1);
                        sprintf(proximo_comando,"(dash
                        %.2f)",corre_para_bola(tempo_ptr[0].objeto[var.adversario_desmarcado[0].posicao_modelo].distancia,

                        tempo_ptr[1].objeto[var.adversario_desmarcado[0].posicao_modelo].distancia,
```

```
                                                                    eu.tempo_eu_ptr[0].sense_body.energia));
                SetAfterTurnCommand(proximo_comando);
            }
            resp = envia_turn(dir_marcacao, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
            return(resp);
        }

/* REGRA 16 Girar para a bola_____________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA >= 27.) && (var.meu_time_tem_posse_de_bola == FALSE)
        && (var.companheiro_perto_da_bola[0].posicao_modelo != TRASH) && (var.adv_perto_de_mim[0].posicao_modelo == TRASH) && (DIR_BOLA !=
        naotem) && (DIST_BOLA != naotem))
        {
            resp = envia_turn(DIR_BOLA, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
            ultima_acao = 16;
            SetAfterTurn(1);
            sprintf(proximo_comando,"(dash %.2f)",corre_para_bola(DIST_BOLA,
                                                    tempo_ptr[1].objeto[posi_bola_no_modelo].distancia,
                                                    eu.tempo_eu_ptr[0].sense_body.energia));
            SetAfterTurnCommand(proximo_comando);
            return(resp);
        }

/* REGRA 17 Caminhar para a bola_____________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA >= 27.) && (var.meu_time_tem_posse_de_bola == FALSE)
        && (var.companheiro_perto_da_bola[0].posicao_modelo == TRASH) && ((DIR_BOLA <= DIR_GIRO) && (DIR_BOLA >= -DIR_GIRO))  && (DIST_BOLA !=
        naotem))
        {
            resp = envia_dash(corre_para_bola(DIST_BOLA,
            tempo_ptr[1].objeto[posi_bola_no_modelo].distancia,eu.tempo_eu_ptr[0].sense_body.energia), buffer);
            ultima_acao = 17;
            acao_ciclo  = 1;
            return(resp);
        }

/* REGRA 18 Girar para a bola_____________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA >= 27.) && (var.meu_time_tem_posse_de_bola == FALSE)
        && (var.companheiro_perto_da_bola[0].posicao_modelo == TRASH) && ((DIR_BOLA > DIR_GIRO) || (DIR_BOLA < -DIR_GIRO)) && (DIR_BOLA !=
        naotem) && (DIST_BOLA != naotem))
        {
            resp = envia_turn(DIR_BOLA, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
            ultima_acao = 18;
            SetAfterTurn(1);
            sprintf(proximo_comando,"(dash %.2f)",corre_para_bola(DIST_BOLA,
                                                    tempo_ptr[1].objeto[posi_bola_no_modelo].distancia,
                                                    eu.tempo_eu_ptr[0].sense_body.energia));
            SetAfterTurnCommand(proximo_comando);
            return(resp);
        }

/* REGRA 19 Girar para a bola_____________________________________________________________________________________________________
```

```
	if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA < 27.) && (var.companheiro_perto_da_bola[0].numero != 
	TRASH) && (var.estou_desmarcado) && (DIR_BOLA != naotem) && (inter_dir == NAO_TEM))
	{
		resp = envia_turn(DIR_BOLA, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
		ultima_acao = 19;
		if(DIST_BOLA>10)
		{
			SetAfterTurn(1);
			sprintf(proximo_comando,"(dash %.2f)",corre_para_bola(DIR_BOLA,
			tempo_ptr[1].objeto[var.adv_perto_de_mim[0].posicao_modelo].distancia, eu.tempo_eu_ptr[0].sense_body.energia));
			SetAfterTurnCommand(proximo_comando);
		}
		return(resp);
	}

/* REGRA 20 Fugir da marcacao_____________________________________________________________________________________________________*/
	if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA < 27.) &&
	(var.companheiro_perto_da_bola[0].posicao_modelo != -1) && (var.estou_desmarcado == 0) &&
	((tempo_ptr[0].objeto[var.adv_perto_de_mim[0].posicao_modelo].direcao >= DIR_GIRO) ||
	(tempo_ptr[0].objeto[var.adv_perto_de_mim[0].posicao_modelo].direcao <= -DIR_GIRO)))
	{
		resp =  envia_dash (100, buffer);
		ultima_acao = 20;
		return(resp);
	}

/* REGRA 20.1 ____________________________________________________________________________________________________________________*/
//j=1+(int) (10.0*rand()/(RAND_MAX+1.0));

/* REGRA 20.1 Girar para Fugir da marcacao_____________________________________________________________________________________*/
	if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA < 27.) &&
	(var.companheiro_perto_da_bola[0].posicao_modelo != -1) && (var.estou_desmarcado == 0) &&
	((tempo_ptr[0].objeto[var.adv_perto_de_mim[0].posicao_modelo].direcao < DIR_GIRO)
	&& (tempo_ptr[0].objeto[var.adv_perto_de_mim[0].posicao_modelo].direcao > -DIR_GIRO)))
	{
		if((DIR_BOLA != NAO_TEM) && (tempo_ptr[0].objeto[var.adv_perto_de_mim[0].posicao_modelo].direcao != DIR_BOLA))
		resp =  envia_turn ((tempo_ptr[0].objeto[var.adv_perto_de_mim[0].posicao_modelo].direcao + (rand()%10)),
		eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
		else
			resp =  envia_turn (DIR_BOLA, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
		ultima_acao = 20.1;
		return(resp);
	}

/* REGRA 21 ______________________________________________________________________________________________________________________*/
	ponto_interceptacao = calcula_ponto_de_interceptacao_da_bola(tempo_ptr_bola, eu);
	if(inter_dir != NAO_TEM)
	{
		inter_x = ponto_interceptacao.x;
		inter_y = ponto_interceptacao.y;
		inter_dist = ponto_interceptacao.distancia;
```

```
		inter_dir = ponto_interceptacao.direcao;
	}

/* REGRA 21 Girar para a bola_____________________________________________________________________________________________________________*/
	if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA < 27) &&
	(var.companheiro_perto_da_bola[0].posicao_modelo == TRASH) && ((DIR_BOLA <= DIR_GIRO) && (DIR_BOLA >= -DIR_GIRO)) &&
	(inter_dir != NAO_TEM) && (inter_dist < 2.))
	{
		resp = envia_turn(DIR_BOLA, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
		ultima_acao = 21;
		if(DIST_BOLA>10)
		{
			SetAfterTurn(1);
			sprintf(proximo_comando,"(dash %.2f)",corre_para_bola(DIST_BOLA,
								tempo_ptr[1].objeto[posi_bola_no_modelo].distancia,
								eu.tempo_eu_ptr[0].sense_body.energia));
			SetAfterTurnCommand(proximo_comando);
		}
		return(resp);
	}

/* REGRA 22 Caminhar para ponto de interceptacao da bola_________________________________________________________________________________*/
	if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && /*(DIST_BOLA < 27)
	&& (var.companheiro_perto_da_bola[0].posicao_modelo == TRASH) && ((DIR_BOLA <= DIR_GIRO) && (DIR_BOLA >= -DIR_GIRO))*/
	(ponto_interceptacao.direcao != NAO_TEM) && (ponto_interceptacao.distancia > 2.) &&
	((ponto_interceptacao.direcao <= 23.) && (ponto_interceptacao.direcao >= -23.)))
	{
		resp = envia_dash(100/*corre_para_bola(ponto_interceptacao.distancia,ponto_interceptacao.distancia,
		eu.tempo_eu_ptr[0].sense_body.energia)*/, buffer);
		 ultima_acao = 22;
		inter_dist_dir = calcula_onde_vai_jogador_ad(inter_x, inter_y, POS_X, POS_Y, eu.ang_abs_jog);
		inter_dist = inter_dist_dir.dist_PI;
		inter_dir = inter_dist_dir.dir_PI;
		return(resp);
	}

/* REGRA 23 _____________________________________________________________________________________________________________________________*/
	if(inter_dist < 2.)
	{
		inter_x    = NAO_TEM;
		inter_y    = NAO_TEM;
		inter_dist = NAO_TEM;
		inter_dir  = NAO_TEM;
	}

/* REGRA 23 Girar para ponto de interceptacao da bola_____________________________________________________________________________________*/
	if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) /*&& (DIST_BOLA < 27)
	&& (var.companheiro_perto_da_bola[0].posicao_modelo == TRASH) && ((DIR_BOLA <= DIR_GIRO) && (DIR_BOLA >= -DIR_GIRO))*/
	&& (ponto_interceptacao.direcao != NAO_TEM) && (ponto_interceptacao.distancia > 2.) && ((ponto_interceptacao.direcao > 23.)
	|| (ponto_interceptacao.direcao < -23.)))
	{
```

```
                resp = envia_turn(ponto_interceptacao.direcao, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
                ultima_acao = 23;
                acao_ciclo  = 1;
                SetAfterTurn(1);
                sprintf(proximo_comando,"(dash %.2f)",corre_para_bola(ponto_interceptacao.distancia, ponto_interceptacao.distancia,
                                eu.tempo_eu_ptr[0].sense_body.energia));
                SetAfterTurnCommand(proximo_comando);
                return(resp);
        }

/* REGRA 24 Caminhar para a bola_____________________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA < 27)
        && (var.companheiro_perto_da_bola[0].posicao_modelo == TRASH) && ((DIR_BOLA <= DIR_GIRO) && (DIR_BOLA >= -DIR_GIRO)) &&
        (ponto_interceptacao.direcao == NAO_TEM))
        {
                resp = envia_dash(corre_para_bola(DIST_BOLA,tempo_ptr[1].objeto[posi_bola_no_modelo].distancia,
                eu.tempo_eu_ptr[0].sense_body.energia), buffer);
                ultima_acao = 24;
                return(resp);
        }

/* REGRA 25 Girar para a bola_____________________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar == FALSE) && (DIST_BOLA < 27)
        && (var.companheiro_perto_da_bola[0].posicao_modelo == TRASH) && ((DIR_BOLA > DIR_GIRO) || (DIR_BOLA < -DIR_GIRO))
        && (DIR_BOLA != naotem))
        {
                resp = envia_turn(DIR_BOLA, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
                ultima_acao = 25;
                if(DIST_BOLA > 10.)
                {
                        SetAfterTurn(1);
                        sprintf(proximo_comando,"(dash %.2f)",corre_para_bola(DIST_BOLA,tempo_ptr[1].objeto[posi_bola_no_modelo].distancia,
                        eu.tempo_eu_ptr[0].sense_body.energia));
                        SetAfterTurnCommand(proximo_comando);
                }
                return(resp);
        }


/*********************************************************/
/********************** Não Ve Bola **********************/
/*********************************************************/
/* REGRA 26 Procurar a bola_____________________________________________________________________________________________________________*/
        if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar != TRUE) && (DIR_BOLA == naotem) && (ultima_acao == 13) &&
        ((jogador.dir_PI >= -DIR_GIRO) && (jogador.dir_PI <= DIR_GIRO)) && (jogador.dist_PI < 5.))
        {
                resp = envia_turn(60, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
                ultima_acao = 26;
                return(resp);
        }
```

```
/* REGRA 27 Caminhar para a posicao inicial_____________________________________________________________________________________________________*/
       if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar != TRUE) && (DIR_BOLA == naotem) && (ultima_acao == 13) &&
       ((jogador.dir_PI >= -DIR_GIRO) && (jogador.dir_PI <= DIR_GIRO)) && (jogador.dist_PI > 5.))
       {
              resp = envia_dash(corre_para_bola(DIST_BOLA,tempo_ptr[1].objeto[posi_bola_no_modelo].distancia,
              eu.tempo_eu_ptr[0].sense_body.energia),buffer);
              ultima_acao = 27;
              return(resp);
       }

/* REGRA 28 Girar para a posicao inicial_____________________________________________________________________________________________________*/
       if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar != TRUE) && (DIR_BOLA == naotem) && (ultima_acao == 13) &&
       ((jogador.dir_PI < -DIR_GIRO) || (jogador.dir_PI > DIR_GIRO)) && (jogador.dist_PI > 5.))
       {
              resp =  envia_turn(jogador.dir_PI, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
              ultima_acao = 28;
              return(resp);
       }

/* REGRA 29 Procurar a bola_____________________________________________________________________________________________________*/
       if ((var.estado_do_jogo == PLAY_ON) && (var.posso_chutar != TRUE) && (DIR_BOLA == naotem) && (ultima_acao != 13))
       {
              resp = envia_turn(90, eu.tempo_eu_ptr[0].sense_body.velocidade, buffer);
              ultima_acao = 29;
              return(resp);
       }
```

Bibliografia

[ALV 96] ALVARES, L. O.; BAEIJS, Christof; DEMAZEAU, Yves. SIGMA – Um Sistema Holístico de Generalização Cartográfica. In: CONGRESSO E FEIRA PARA USUÁRIOS DE GEOPROCESSAMENTO, GIS, 2., 1996, Curitiba, Paraná. **Anais...** Curitiba: Ed. SAGRES, 1996. p.657-666.

[ALV 99] ALVARES, Luis Otávio; SICHMAN, Jaime. Introdução aos Sistemas Multiagentes. In: ESCOLA DE INFORMÁTICA NORTE DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, ENI, 1., 1999, Belém. **Anais...** Belém: Unama, 1999. p.170-200.

[ARN 98] ARNOLD, Gustavo V. **Uma Extensão da Linguagem RS para o Desenvolvimento de Sistemas Reativos Baseados na Arquitetura de Subsunção**. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

[ASA 94] ASADA, Minoru et al. Coordination of multiple behaviors acquired by a a vision-based reinforcement learning. In: IEEE/RSJ/GI INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, IROS, 1994, Munich, Germany. **Proceedings...** [S.l.:s.n.], 1994. p.917-924. Disponível em: <http://citeseer.nj.nec.com/asada94coordination.html>. Acesso em: 07 fev. 2001.

[ASA 99] ASADA, Minoru et al. RoboCup: Today and tomorrow – What we have learned. **Artificial Intelligence**, Amsterdam, v.110, n.2, p.193-214. 1999.

[BAS 98] BASTOS, R. M. **O Planejamento de Alocação de Recursos Baseados em Sistemas Multiagentes**. Porto Alegre: CPGCC da UFRGS, 1998. Tese de Doutorado.

[BIT 97] BITTENCOURT, Guilherme; COSTA, Augusto L. Expert-coop: An environment for cognitive multi-agent systems. In: CONFERENCE ON MANAGEMENT AND CONTROL OF PRODUCTION AND LOGISTICS, MCPL, 1997, Campinas, São Paulo. **Proceedings...** [S.l.:s.n.], 1997. v.2, p.492-497. Disponível em: <http://www.das.ufsc.br/pub/publicacoes/mestrado/loreiro.augusto.97.ps.gz >. Acesso em: 07 fev. 2001.

[BIT 98] BITTENCOURT, Guilherme. **Inteligência Artificial:** ferramentas e teorias. Florianópolis: Ed. UFSC, 1998.

[BOI 92] BOISSER, Oliver; DEMAZEAU, Yves. A Distributed Artificial Intelligence View on General Purpose Vision Systems. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, 3., 1992, Amsterdam. **Proceeding…** Amsterdam: North-Holland, 1992. p.311-330.

[BON 88] BOND, Alan H.; GASSER, Les. Distributed Artificial Intelligence. An Analysis of Problems and Research in DAI. In: BOND, Alan H.; GASSER, Les (Eds.). **Readings in Distributed Artificial Intelligence.** San Mateo, Califórnia: Morgan Kaufman, 1988. p.3-35.

[BUR 97] BURKHARD, Hans-Dieter; HANNEBAUER, Markus; WENDLER, Jan. AT Humboldt – Development, Practice and Theory. In: ROBOCUP: ROBOT SOCCER WORLD CUP 1., 1997, Nagoya. **Proceedings...** Berlin: Springer-Verlag, 1997. p.357-372.

[BUR 99] BURKHARD, Hans-Dieter et al. AT Humboldt in RoboCup-99. In: ROBOCUP: ROBOT SOCCER WORLD CUP 3., 1999, Stockholm, Sweden. **Procedings...** Berlin: Springer-Verlag, 1999. p.1-6. Disponível em: <www.ida.liu.se/ext/robocup/simul/ATHumboldt99/paper.ps>. Acesso em: 07 fev. 2001.

[BRO 86] BROOKS, Rodney A. A Robust layered control system for a mobile robot. **IEEE Journal of Robotics and Automation**, New York, v. RA-2, n. 1, p.14-23, Mar. 1986.

[BRO 89] BROOKS, Rodney A. A robot that walks: Emergent behavior from a carefully evolved network. **Neural Computation** [S.l.], v.1, n.2, p.253-262, 1989.

[BRO 93] BROOKS, Rodney A. Elephants don't play chess. In: MAES, Pattie (Eds.). **Designing Autonomous Agents:** Theory and Pratice from Biology to Engineering and Back. Cambridge: MIT Press, 1993. p.3-15.

[BRO 91a] BROOKS, Rodney A. Intelligence Without Reason. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJICAI, 12., 1991, Sydney. **Proceedings...** Austrália: [s.n.], 1991. p.569-595.

[BRO 91b] BROOKS, Rodney A. Intelligence Without Representation. **Artificial Intelligence**, Amsterdan, v. 47, p.139-160, Jan.1991.

[CAZ 97] CAZELLA, Sílvio C. **Uma arquitetura para coordenar a interação de agentes na Internet**. Porto Alegre: CPGCC da UFRGS, 1997. Dissertação de Mestrado.

[COR 2001] CORTEN, E et al. **Soccerserver manual**. Technical Report RoboCup-1999-001, RoboCup, 2001. Disponível em: <http://www.dsv.su.se/~johank/RoboCup/manual/>. Acesso em: 07 fev. 2001.

[COA 2000] CORADESCHI, Silvia et al. Overview of RoboCup-99. **AI Magazine**, Menlo Park, v.21, n.3, p.11-18, Fall 2000.

[COS 98] COSTA, Augusto L.; BITTENCOURT, Guilherme. UFSC-team: A cognitive Multi-Agent Approach to the RoboCup'98 Simulator League. In: ROBOCUP: ROBOT SOCCER WORLD CUP 2., 1998, Paris, França. **Proceedings...** Berlin: Springer-Verlag, 1999. Disponível em: <http://www.das.ufsc.br/~loreiro/download/UFSC-Team.ps.gz>. Acesso em: 07 fev. 2001.

[COS 99a] COSTA, Augusto L.; BITTENCOURT, Guilherme. From the Concurrent Architecture to the Concurrent Autonomous Agents Architecture. In: ROBOCUP: ROBOT SOCCER WORLD CUP 3., 1999, Stockholm, Sweden. **Proceedings...** Berlin: Springer-Verlag, 2000.

[COS 99b] COSTA, Augusto L.; BITTENCOURT, Guilherme. **SoccerServer:** O Simulador para o futebol de Robôs da RoboCup Federation. In: ENCONTRO NACIONAL DE INTELIGÊNCIA ARTIFICIAL, ENIA, 1999, Rio de Janeiro. Disponível em: <http://www.das.ufsc.br/~loreiro/download/Tutorial_ENIA99.ps.gz>. Acesso em: 07 fev. 2001.

[COS 99c] COSTA, Augusto L. **Conhecimento Social Dinâmico:** Uma Estratégia de Cooperação para Sistemas Multiagentes Cognitivos. Florianópolis: Programa de Pós-Graduação em Engenharia Elétrica da UFSC, 1999. Exame de Qualificação.

[DEC 96] DECKER, Keith S. Task environment centered simulation. In: Prietula, M.; Carley, K.; Gasser L. (Eds.). **Simulating Organizations:** Computational Models of Institutions and Groups. [S.l.]: AAAI Press/MIT Press, 1996. Disponível em: <http://citeseer.nj.nec.com/decker96task.html>. Acesso em: 07 fev. 2001.

[DEM 93] DEMAZEAU, Yves. Distributed Artificial Intelligence & Multi-Agent Systems. In: SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 10., 1993, Porto Alegre. **Anais...** Porto Alegre: Instituto de Informática da UFRGS, 1993.

[DOR 99] DORER, Klaus. Extended Behavior Networks for the magmaFreiburg Team. In: ROBOCUP: ROBOT SOCCER WORLD CUP 3., 1999, Stockholm, Sweden. **Proceedings...** Berlin: Springer-Verlag, 2000. p.79-83. Disponível em: <http://www.ida.liu.se/ext/robocup/simul/magmaFreiburg/paper.ps>. Acesso em: 07 fev. 2001.

[FER 91] FERBER, Jacques; Gasser L. Intelligence artificielle distribuée. In: INTERNATIONAL WORKSHOP ON EXPET SYSTEMS & THEIR APPLICATIONS, 1991, Avignon, France. **Procedings...** [S.l.:s.n.], 1991.

[FRO 97] FROZZA, Rejane; ALVARES, Luis Otávio. Um Ambiente para o Desenvolvimento de Sistemas Multiagentes Reativos. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, SEMISH, 24., 1997, Brasília. **Anais...** Brasília: UnB, 1997. p.375-386.

[HAL 2001] HALL, Brian. **Brian Hall Beej's Guide to Network Programming Using Internet Sockets**. Disponível em: <http://www.dee.isep.ipp.pt/~pviana/igr/beej>. Acesso em: 07 fev. 2001.

[HUH 97] HUHNS, Michael N.; SINGH, Munindar P. Agents and Multiagents Systems: Themes, Approaches, and Challenges. In: HUHNS, Michael N.; SINGH, Munindar P. (Eds.). **Readings in Agents**. San Francisco: Morgan Kaufmann, 1997. p.1-23.

[KIT 97a] KITANO, Hiroaki et al. RoboCup: The Robot World Cup Initiative. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 1., 1997, Marina del Ray, Califórnia. **Proceedings...** [S.l.:s.n.], 1997. Disponível em: <http://www.robocup.org/RoboCup/roboCup-Agent97.ps>. Acesso em: 07 fev. 2001.

[KIT 97b] KITANO, Hiroaki et al. RoboCup - A challenge problem for AI -. **AI Magazine**, Menlo Park, v.18, n.1, p.73-85, Spring 1997.

[KIT 97c] KITANO, Hiroaki et al. The Robocup Synthetic Agent Challenge 97. In: ROBOCUP: ROBOT SOCCER WORLD CUP 1., 1997, Nagoya, Japan. **Proceedings...** Berlin: Springer-Verlag, 1997. p.62-73.

[KOS 99] KOSTIADIS, Kostas; HU, Huosheng. Essex Wizards. In: ROBOCUP: ROBOT SOCCER WORLD CUP 3., 1999, Stockholm, Sweden. **Proceedings...** Berlin: Springer-Verlag, 2000. p.17-25. Disponível em: <http://www.ida.liu.se/ext/robocup/simul/Essex-Wizards/paper.ps>. Acesso em: 07 fev. 2001.

[LEA 97] LEAKE, David; BARLETTA, Ralph. **Case-Based Reasoning**. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, IJCAI, 1997, Nagoya, Japan. **Proceedings...** [S.l.:s.n.], 1997. Disponível em: <http://www.ijcai.org/past/ijcai-97/CI/TUTORIAL/mp4.html>. Acesso em: 07 fev. 2001.

[LEW 96] LEWIS, Bil; BERG, Daniel J. **Threads Primer** – A Guide Multithreaded Programming. Mountain View: SunSoft Press, 1996.

[MAC 93] MACKWORTH, Alan K. On seeing robots. In: BASU, A.; LI, X. (Eds.). **Computer Vision:** Systems, Theory, and Applications. Singapore: World Scientific Press, 1993. v.38. p. 1-13. Disponível em: <http://citeseer.nj.nec.com/context/50845/0>. Acesso em: 07 fev. 2001.

[MAE 90] MAES, Pattie. Situated agents can have goals. In: MAES, Pattie (Ed.). **Designing Autonomous Agents:** Theory and Pratice from Biology to Engineering and Back. Cambridge: MIT Press, London, 1990. p.49-70.

[MAS 99] MATSUMURA, Takeshi. Behavior Based Modelling. In: ROBOCUP: ROBOT SOCCER WORLD CUP 3., 1999, Melbourne, Australia. **Proceedings...** Berlin: Springer-Verlag, 2000. p.13-16. Disponível em: <http://www.ida.liu.se/ext/robocup/simul/ERIKA/paper.ps>. Acesso em: 07 fev. 2001.

[MAT 96] MATSUBARA, Hitoshi; NODA, Itsuki; HIRAKI, Kazuo. Learning of Cooperative actions in multi-agent systems: a case study of pass in soccer. In: SEN, S. (Ed.). SPRING SYMPOSIUM ON ADAPTATION, COEVOLUTION AND LEARNING IN MULTI-AGENT SYSTEMS, AAAI, 1996, Portland, Oregon. **Proceedings...** [S.l.:s.n.], 1996. p.63-67. Disponível em: <http://citeseer.nj.nec.com/matsubara96learning.html>. Acesso em: 07 fev. 2001.

[NOD 95] NODA, Itsuki. Soccer Server: a simulator of RoboCup. In: JAPANESE SOCIETY FOR ARTIFICIAL INTELLIGENCE AI-SYMPOSIUM 95: Special Session on RoboCup, JSAI, 1995, Tokyo, Japan. **Proceedings...** [S.l.:s.n.], 1995. Disponível em: <http://citeseer.nj.nec.com/noda95soccer.html>. Acesso em: 07 fev. 2001.

[NOD 96a] NODA, Itsuki; MATSUBARA, Hitoshi. Soccer Server and Researches on Multi-Agent Systems. In: INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS - WORKSHOP ON ROBOCUP, IROS, 1996, Osaka, Japan. **Proceedings...** [S.l.:s.n.], 1996. p.1-7. Disponível em: <http://citeseer.nj.nec.com/noda96soccer.html>. Acesso em: 07 fev. 2001.

[NOD 96b] NODA, Itsuki; MATSUBARA, Hitoshi; HIRAKI, Kazuo. Learning Cooperative Behavior in Multi-agent Environment – a case study of choice of play-plans in soccer. In: PACIFIC RIM INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, PRICAI, 4., 1996, Cairns. **Proceedings...** Australia: Springer-Verlag, 1996. p.570-579. Disponível em: <http://citeseer.nj.nec.com/context/97168/0>. Acesso em: 07 fev. 2001.

[NOD 2001] NODA, Itsuki. **Simulador Soccerserver**. Disponível em: <http://ci.etl.go.jp/~noda/soccer/server>. Acesso em: 07 fev. 2001.

[OLI 96] OLIVEIRA, Flávio M. Inteligência artificial Distribuída. In: ESCOLA REGIONAL DE INFORMÁTICA, 4., 1996, Canoas, RS. **Anais...** Canoas: Departamento de Informática da ULBRA, 1996. p.54-73.

[PAG 97] PAGELLO, E. et al. A Reative Architecture for RoboCup Competition. In: ROBOCUP: ROBOT SOCCER WORLD CUP 1., 1997, Nagoya, Japan. **Proceedings...** Berlin: Springer-Verlag, 1997. p.434-442.

[PAL 96] PALAZZO, Luiz Antonio M. **Aspectos da Modelagem de Sistemas de Informações Inteligentes**. Porto Alegre: CPGCC da UFRGS, 1996. Exame de Qualificação.

[PEB 95] PEBODY, Miles. How do you choose your Agentes? How do you Distribute your Processes?. In: STEELS, Luc (Ed.).**The Biology and Technology of Intelligent Autonomous Agents**. Berlin: Springer-Verlag, 1995.

[REI 2001] REIS, Luis Paulo; LAU, Nuno; LOPES, Luis Seabra. **FC Portugal Team Description Paper**. Disponível em: <http://www.ieeta.pt/robocup/documents/FCPortugalTeamDescription.ps.zip>. Acesso em: 07 fev. 2001.

[RIC 93] RICH, Elaine; KNIGHT, Kevin. **Inteligência Artificial**. São Paulo: Makron Brooks, 1993.

[ROB 2001] ROBOCUP. **Página Oficial da RoboCup**. Disponível em: <http://www.robocup.org>. Acesso em: 07 fev. 2001.

[RUS 95] RUSSEL, Stuart; NORVIG, Peter. **Artificial Intelligence:** A Modern Approach. New Jersey: Prentice-Hall, 1995.

[SAH 93] SAHOTA, Michael K. et al. Real-time control of soccer-playing robots using off-board vision: the dynamite testbed. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS, SMC, 1995, Vancouver, Canada. **Proceeding...** Vancouver: British Columbia, 1995. p. 3690-3663. Disponível em: <http://citeseer.nj.nec.com/context/4427/0>. Acesso em: 07 fev. 2001.

[SCE 2000] SCERRI, Paul; YDREN, Johan. End User Specication of RoboCup Teams. In: ROBOCUP: ROBOT SOCCER WORLD CUP 3., 2000, Melbourne, Australia. **Proceedings...** Berlin: Springer-Verlag, 2000. Disponível em: <http://citeseer.nj.nec.com/context/1264298/0>. Acesso em 07 fev. 2001.

[STE 90] STELLS, Luc. Cooperating Between Distributed Agents Through Self-Organization. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, 1990, Amsterdam. **Proceedings...** Amsterdam: North-Holland, 1990.

[STE 94] STEELS, Luc; BROOKS, Rodney. **The Artificial Life Route to Artificial Intelligence**. Hillsdale, New Jersey: Lourence Erlboum Associates, 1995.

[STO 99a] STONE, Peter; VELOSO, Manuela; RILEY, Patrick. The CMUnited-98 Champion Simulator Team. In: ROBOCUP: ROBOT SOCCER WORLD CUP 2., 1998, Stockholm, Sweden. **Proceedings...** Berlin: Springer-Verlag, 1999. Disponível em: <http://www.research.att.com/~pstone/Papers/98springer/simulator/champ98.ps.gz>. Acesso em: 07 fev. 2001.

[STO 99b] STONE, Peter; VELOSO, Manuela. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. **Artificial Intelligence**, Amsterdam, v.110, n.2, p.241-273, June 1999.

[STO 2000a] STONE, Peter; RILEY, Patrick; VELOSO, Manuela. The CMUnited-99 Champion Simulator Team. In: ROBOCUP: ROBOT SOCCER WORLD CUP 3., 2000, Melbourne, Australia. **Proceedings...** Berlin: Springer-Verlag, 2000. Disponível em: < http://www.research.att.com/~pstone/Papers/99springer/simulator/champ99.ps.gz>. Acesso em: 07 fev. 2001.

[STO 2000b] STONE, Peter; VELOSO, Manuela. Multiagent Systems: A Survey from a Machine Learning Perspective. **Autonomous Robots**, Los Angeles, v.8, n.3, p. 345-383, June 2000. Disponível em: < http://www.research.att.com/~pstone/Papers/99auro/survey.ps.gz>. Acesso em: 07 fev. 2001.

[STO 2001] STONE, Peter. **CMUnited**. Disponível em <http://www.cs.cmu.edu/afs/cs/usr/pstone/public/papers.html>. Acesso em: 07 fev. 2001.

[TAM 99] TAMBE, Milind et al. Building agent teams using an explicit teamwork and learning. **Artificial Intelligence**, Amsterdam, v.110, n.3, p.215-239, June 1999.

[TAM 2000] TAMBE, Milind; RAINES, Taylor; MARSELLA, Stacy. Agent Assistants for Team Analysis. **AI Magazine**, Menlo Park, v.21, n.3, p.27-31, Fall 2000.

[TAN 98] TANAKA-Ishii, Kumiko et al. MIKE: An Automatic Commentary System for Soccer. In: INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS, ICMAS, 1998, Paris. **Proceeding...** Washington: IEEE Computer Society, 1998. p.285-292.

[TOM 2000] TOMOICHI, Takahashi. LogMonitor: Analyzing Good Plays to Train Player Agents. **AI Magazine**, Menlo Park, v.21, n.3, p.25, Fall 2000.

[UFS 2001] UFSC Team. Disponível em: <http://www.lcmi.ufsc.br/ufsc-team/>. Acesso em: 07 fev. 2001.

[VEL 97] VELOSO, Manuela et al. CMUnited: A team of robotic soccer agents collaborating in an adversarial environment. In: INTERNATIONAL WORKSHOP ON ROBOCUP, IJCAI, 1997, Nagoya, Japan. **Proceedings...** [S.l.:s.n.], 1997. Disponível em: <http://www.cs.cmu.edu/~mmv/papers/CROSSROADS98.ps.gz>. Acesso em: 07 fev. 2001.

[WAV 92] WAVISH, Peter. Exploiting Emergent Behavior in Multi-Agent Systems. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, 1992, Amsterdam. **Proceedings...** Amsterdam: North-Holland, 1992. p. 297-310.

[WOO 95] WOOLDRIDGE, Michael; JENNINGS, Nicholas R. **Intelligent Agents:** Theory and Pratice. [S.1.:s.n.], 1995.