

**UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**IMPLEMENTAÇÃO DE ESTRATÉGIAS DE
COMPORTAMENTO PARA AGENTES EM
UMA SIMULAÇÃO 3D DE FUTEBOL**

TRABALHO DE GRADUAÇÃO

Eric Tomás Zancanaro

Santa Maria, RS, Brasil

2015

IMPLEMENTAÇÃO DE ESTRATÉGIAS DE COMPORTAMENTO PARA AGENTES EM UMA SIMULAÇÃO 3D DE FUTEBOL

Eric Tomás Zancanaro

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da
Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a
obtenção do grau de

Bacharel em Ciência da Computação

Orientador: Prof. Dr. Giovani Rubert Librelotto

**408
Santa Maria, RS, Brasil**

2015

**Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação**

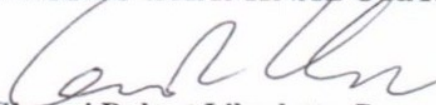
A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

**IMPLEMENTAÇÃO DE ESTRATÉGIAS DE COMPORTAMENTO PARA
AGENTES EM UMA SIMULAÇÃO 3D DE FUTEBOL**

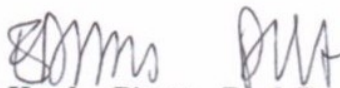
elaborado por
Eric Tomás Zancanaro

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

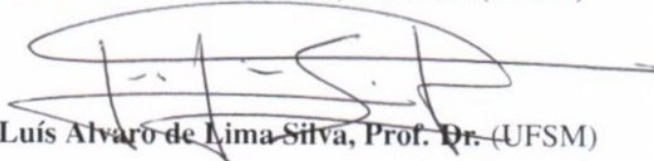
COMISSÃO EXAMINADORA:



Giovani Rubert Librelotto, Dr.
(Presidente/Orientador)



Eduardo Kessler Pivetta, Prof. Dr. (UFSM)



Luís Alvaro de Lima Silva, Prof. Dr. (UFSM)

Santa Maria, 11 de Dezembro de 2015.

AGRADECIMENTOS

Agradeço a minha família, por me proporcionar a oportunidade de vivenciar a universidade e me manter focado, mesmo estando a centenas de quilômetros de distância.

Agradeço aos professores da instituição, por me apresentar ao vasto mundo da computação, em especial ao professor Giovani Librelotto e a banca deste trabalho, por auxiliar na conclusão desta primeira jornada por este mundo.

Agradeço, por fim, aos amigos e colegas que fizeram com que esta jornada seja inesquecível.

A todos, obrigado.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

IMPLEMENTAÇÃO DE ESTRATÉGIAS DE COMPORTAMENTO PARA AGENTES EM UMA SIMULAÇÃO 3D DE FUTEBOL

AUTOR: ERIC TOMÁS ZANCANARO

ORIENTADOR: GIOVANI RUBERT LIBRELOTTO

Local da Defesa e Data: Santa Maria, 11 de Dezembro de 2015.

Com o objetivo de promover as pesquisas nas áreas de inteligência artificial e robótica, a RoboCup promove anualmente competições de futebol de robôs. Estas competições demandam trabalho em diferentes áreas do conhecimento, sendo igualmente importantes o desenvolvimento das capacidades motoras dos robôs participantes e a criação de comportamentos para o controle dos mesmos.

Buscando adquirir experiência aplicável ao controle de robôs físicos, utilizamos as capacidades da simulação computacional para criar comportamentos a serem seguidos por um time de jogadores simulados, capazes de participar de forma autônoma de uma partida de futebol.

Utilizando a plataforma de simulação SimsPark, trabalhamos com um ambiente de simulação tri-dimensional, capaz de representar uma aproximação dos desafios encontrados na competição de futebol de robôs reais.

Os comportamentos foram implementados utilizando-se de um *framework* disponibilizado pela equipe alemã MagmaOffenburg, o qual provê funcionalidades motoras básicas para os agentes.

Para a criação dos comportamentos foram definidos três papéis existentes em um time de futebol, cada qual possui um comportamento adequado ao objetivo identificado para cada um destes papéis em uma partida completa. Estes comportamentos criam um time capaz de desempenhar as funções básicas esperadas em uma partida de futebol.

Palavras-chave: RoboCup. Simulação Computacional. Comportamento. Java. Futebol de Robôs. magmaOffenburg.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

IMPLEMENTATION OF BEHAVIOR STRATEGIES FOR AGENTS IN A 3D SOCCER SIMULATION

AUTHOR: ERIC TOMÁS ZANCANARO

ADVISOR: GIOVANI RUBERT LIBRELOTTO

Defense Place and Date: Santa Maria, December 11th, 2015.

Aiming to promote research on fields such as artificial intelligence and robotics, RoboCup promotes annual robot soccer competitions. These competitions demand work on different knowledge fields, being the development of motor skills as important as the creation of behaviors capable of controlling the robots.

Looking to gather experience applicable to controlling physical robots, we utilize computer simulation to create behaviors capable of being followed by a team of simulated players, capable of autonomously participating in a soccer match.

Working with the SimsPark simulation platform, we have a three-dimensional environment capable of replicating an approximation of the challenges faced in the real robots competition.

The behaviors were implemented by use of a framework made available by the German team MagmaOffenburg, a framework which provides the agents with basic motor skills.

Three roles were defined for the creation of the behaviors, each of which adequate to the objectives identified for these roles in a full soccer match. These behaviors create a team capable of fulfilling the basic functions expected in a soccer match.

Keywords: RoboCup. Computer Simulation. Behavior. Java. Robot Soccer.

SUMÁRIO

LISTA DE FIGURAS	8
1 INTRODUÇÃO	9
2 FUNDAMENTOS E REVISÃO DE LITERATURA	11
2.1 RoboCup	11
2.2 Simulação computacional	12
2.3 A liga de simulação 3D	12
2.4 SimsPark e rcserver3D	14
2.5 MagmaOffenburg e magma-AF	15
2.5.1 Communication	17
2.5.2 Protocol	17
2.5.3 Model	17
2.5.4 Control	17
2.5.5 DecisionMaking	18
2.6 Sumário do Capítulo	18
3 DESENVOLVIMENTO	20
3.1 Definição de Requisitos	21
3.2 Estratégias de Comportamento	22
3.2.1 Defesa	23
3.2.2 Meia	24
3.2.3 Ataque	25
3.3 Estudo das classes do <i>framework</i>	27
3.3.1 ThoughtModel	27
3.3.2 IBelief	28
3.3.3 IBehavior	28
3.3.4 Vector3D e Pose2D	29
3.4 Estrutura do decisor	29
3.4.1 decideSoccer()	30
3.4.2 Funções de decisão	31
3.4.3 Funções de ação	33
3.5 Sumário do Capítulo	37
4 ESTUDOS DE CASO	38
4.1 Avanço ofensivo	38
4.2 Teste da defesa	40
4.3 Sumário do Capítulo	43
5 CONCLUSÃO	44
REFERÊNCIAS	45

LISTA DE FIGURAS

2.1	Anatomia do modelo <i>Nao</i> utilizado pelo rcsserver3D.....	13
2.2	Arquitetura em camadas do MagmaAF.....	16
3.1	Estrutura do comportamento dos meias.....	25
3.2	Estrutura do comportamento de ataque.....	26
3.3	Fluxo do decisor.....	30
3.4	Vetor de chute.....	35
4.1	Formação ofensiva.....	39
4.2	Realização de um passe.....	39
4.3	Finalização da jogada.....	40
4.4	Capacidade de finalização em diferentes ângulos.....	41
4.5	Formação defensiva.....	42
4.6	Formação da barreira.....	42

1 INTRODUÇÃO

A RoboCup é uma competição internacional cujo objetivo é proporcionar o crescimento da robótica e da inteligência artificial, proporcionando um desafio a comunidade científica focada nestas áreas. No ano de 2015 o evento foi realizado na cidade de Hefei¹, China, e contou com a primeira participação da Universidade Federal de Santa Maria (UFSM), realizada através da equipe TauraBots.

O desenvolvimento de um robô humanóide capaz de participar da prova de futebol de robôs demanda trabalho em diversas áreas, acarretando na divisão da equipe em diversos times de desenvolvimento. Além do grupo responsável pela manutenção das partes elétricas e mecânicas dos robôs, existem acadêmicos dedicados a aperfeiçoar as capacidades perceptivas do robô e acadêmicos responsáveis pelos algoritmos de comportamento.

Como o acesso ilimitado ao robô por parte de todas as equipes é algo inviável, os grupos devem adquirir ferramentas que possibilitem a continuidade de seus trabalhos. A equipe que desenvolve os algoritmos de comportamento, por exemplo, visa o uso de simuladores computacionais para o teste de suas estratégias, possibilitando assim o progresso desta área independentemente das demais.

Dado o interesse do grupo pelo uso de ferramentas de simulação, foram identificados dois ambientes possíveis para a criação do comportamento dos robôs. A primeira opção identificada foi o uso de simuladores em um ambiente de duas dimensões, possibilitando a criação de estratégias de alto nível para o controle de times de humanóides completos. Esta opção provê ao grupo um alto nível de abstração, facilitando a implementação das estratégias mas removendo grande parte dos desafios encontrados com o robô real. Esta opção não atende as necessidades e interesses da equipe, que busca adquirir experiência transferível ao trabalho realizado no robô físico. Por este motivo o grupo decidiu pela segunda opção de ambiente: um simulador em três dimensões.

A simulação em um ambiente 3D provê aos integrantes da equipe um ambiente realista, capaz de simular boa parte das dificuldades encontradas com o robô real, ao mesmo tempo em que permite a criação de comportamentos complexos que consideram diversos cenários presentes em uma partida completa. Testando-se diferentes estratégias no cenário simulado, a equipe pode adquirir experiência importante para a definição dos módulos comportamentais do robô real.

¹ <http://www.robocup2015.org/>

Este trabalho tem por objetivo implementar diferentes estratégias de jogo para um agente autônomo em uma simulação 3D, incluindo nas mesmas a capacidade de adaptação a diferentes contextos apresentados durante uma partida. O agente simulado deve considerar seu papel na equipe na escolha de suas ações, bem como definir de forma autônoma qual papel deve adotar ao longo da partida, sendo capaz de modificar seu comportamento instantaneamente caso considere necessário. Definido seu papel, o agente deverá considerar informações sobre o mundo em que se encontra, processando-as para definir qual a melhor ação a ser tomada a cada momento e mantendo-se coeso quanto ao seu papel na equipe. Um agente determinado para funções de ataque deve considerar informações e táticas diferentes daquelas compreendidas por um agente da defesa.

O trabalho propõe a implementação do comportamento de agentes seguindo três papéis: ataque, defesa e meia. A troca de papéis durante a execução será permitida pelo modelo implementado, embora o modo como esta definição é dada seja abstraído pela implementação.

Para a construção dos algoritmos de comportamento será utilizado um conjunto de ferramentas disponibilizado através de um *framework* de código aberto. Este conjunto de ferramentas possibilita a criação de estratégias de comportamento sobre uma base prévia de algoritmos de controle básico de humanóides, possibilitando o foco no módulo de tomada de decisões, responsável por ditar as ações a serem tomadas pelos robôs dentro do cenário simulado.

Este trabalho foi dividido em quatro seções, iniciando pela revisão dos conceitos importantes e das ferramentas empregadas na criação dos algoritmos. O terceiro capítulo apresenta as metodologias empregadas durante o desenvolvimento dos algoritmos, seguido por um capítulo dedicado a estudos de casos, onde os comportamentos implementados são postos a prova. Ao final, o texto apresenta a conclusão alcançada com o término do trabalho.

2 FUNDAMENTOS E REVISÃO DE LITERATURA

Neste capítulo será realizada uma breve apresentação dos conceitos utilizados durante o decorrer deste trabalho, bem como uma revisão das ferramentas utilizadas para o mesmo.

2.1 RoboCup

Em 1993, pesquisadores japoneses postularam a ideia de utilizar o futebol para promover a ciência e a tecnologia. Neste ano foi instituída a J-League, competição de robótica centrada no desafio de desenvolver robôs e sistemas capazes de jogar futebol. Menos de um mês após o lançamento da liga, a reação de pesquisadores de fora do Japão convence o grupo de pesquisadores a expandir a iniciativa internacionalmente, criando assim a RoboCup.

Em 1996, foi realizada a Pre-RoboCup-96, um evento contando com oito times competindo em uma liga de simulações. Este pré-evento buscava avaliar as condições de organizar uma RoboCup em larga escala (Robocup org., 2015). No ano seguinte realizou-se a primeira conferência e competição RoboCup, contando com 40 times e mais de 5.000 espectadores.

O objetivo principal da organização do evento é o de promover as pesquisas em robótica e inteligência artificial (KITANO et al., 1998), utilizando um desafio atraente tanto ao público geral quanto aos pesquisadores dedicados. Utilizando-se do caso do computador Deep Blue (CAMPBELL; HOANE; HSU, 1980), que derrotou o enxadrista campeão do mundo, Garry Kasparov, 50 anos após a criação do primeiro computador, a RoboCup almeja a existência, em um período de 50 anos, de um time de robôs humanóides autônomos capazes de derrotar o time vencedor da copa do mundo mais recente (KITANO et al., 1998).

Para cumprir este objetivo foram instituídas três modalidades de competição. A primeira é denominada futebol de robôs reais, aonde modelos físicos competem em partidas com regras adaptadas do esporte. Outra modalidade é a competição de robôs especialistas, utilizada para demonstrações de modelos com capacidades excepcionais em funções específicas, sendo elas relacionadas ou não ao futebol. Por fim existe a modalidade de futebol de robôs em software (KITANO et al., 1998). Dedicada a expandir o alcance dos incentivos de pesquisa da organização, esta modalidade prevê a implementação de robôs em *software*, através do desenvolvimento das técnicas de simulação computacional.

2.2 Simulação computacional

Simulação computacional refere-se aos *softwares* capazes de imitar eventos ou operações do mundo real, utilizando-se da formalização dos padrões e especificações empregadas na tarefa a ser simulada. A simulação computacional opera sobre um conjunto de partes capazes de trabalhar associadas, visando um objetivo em comum, denominado sistema (CHWIF; MEDINA, 2006).

Os simuladores apresentam uma abstração da realidade chamada de *modelo de simulação*. Os modelos de simulação consistem na simplificação das interações existentes entre as partes de um sistema, mantendo sempre uma aproximação do comportamento real (CHWIF; MEDINA, 2006). Esta abstração deve ser feita com base na compreensão dos fatores essenciais para o comportamento do sistema para a atividade a ser simulada.

No contexto da RoboCup existem dois modelos de simulação em uso nas competições atuais. O primeiro modelo apresenta um nível muito alto de abstração, representando o ambiente em apenas duas dimensões e ignorando as limitações físicas dos robôs simulados, provendo as equipes com um sistema eficaz para a criação de estratégias focadas no desenvolvimento de times com controle centralizado, priorizando a criação de estratégias de alto nível. Neste modelo os algoritmos de comportamento conseguem uma visão global do sistema, independente da percepção individual de cada robô presente na simulação.

O segundo modelo utilizado na competição simula um ambiente em três dimensões, com representações da física aplicada sobre os robôs humanóides e das capacidades limitadas dos sensores responsáveis por perceber o ambiente a seu redor. Este modelo foi responsável pela criação de um desafio ainda maior para as equipes participantes, visto que as limitações impostas pelas partes mecânicas simuladas dos robôs deve ser superado para a criação de times capazes de competir efetivamente. Atualmente o foco das equipes participantes concentra-se no desenvolvimento dos algoritmos de movimentação e localização dos agentes, deixando o controle do comportamento em alto nível para a simulação em duas dimensões.

2.3 A liga de simulação 3D

A liga de simulação 3D é um dos eventos pivotais da RoboCup, trazendo consigo uma representação mais realista dos desafios encontrados na criação de um robô autônomo capaz de jogar futebol. A competição é executada no simulador *RoboCup Simulated Soccer Server*

3D (*rcsserver3D*), software de código aberto desenvolvido a partir do motor para simulações físicas SimsPark (OBST; ROLLMANN, 2004)). Visando a melhoria contínua do sistema, um dos requisitos para a qualificação de equipes para a liga é a contribuição científica para a comunidade da RoboCup, através da participação dos integrantes no desenvolvimento do simulador e da publicação de artigos referentes a competição (RoboCup org., 2015), promovendo a integração entre desenvolvedores das equipes participantes.

Inicialmente o simulador provia apenas a simulação de robôs esféricos (OBST; ROLLMANN, 2004), os quais eram usados para a concepção de estratégias de alto nível para uma partida de futebol. Com a melhoria do sistema de simulação, a representação de robôs mais sofisticados foi possível. Atualmente o simulador utiliza um modelo de humanóide *Nao* baseado na linha de robôs desenvolvida pela Aldebaran Robotics (Aldebaran Robotics, 2015)). O modelo atual reflete em software a anatomia mecânica do modelo físico, simulando as capacidades motoras do humanóide através da representação das juntas, perceptores e atuadores do humanóide (figura 2.1).

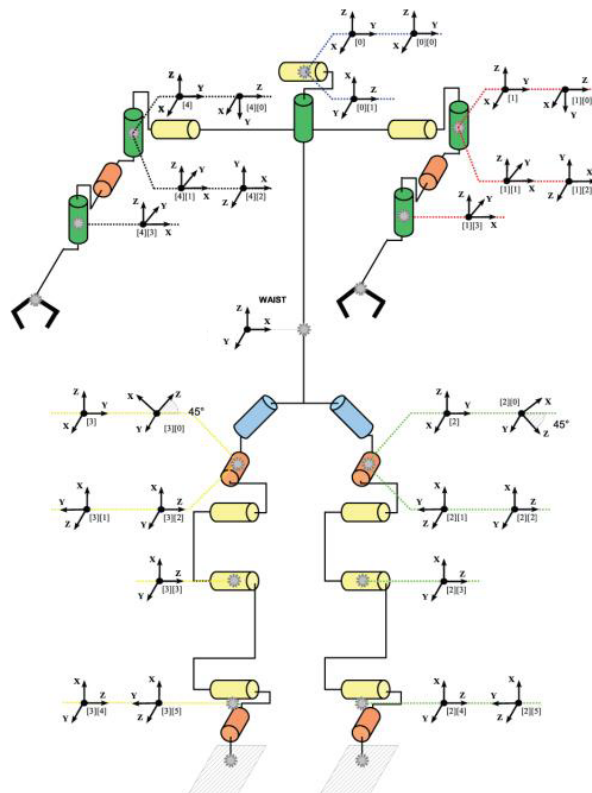


Figura 2.1: Anatomia do modelo *Nao* utilizado pelo *rcsserver3D*

Referimo-nos aos robôs simulados nesta plataforma como agentes, os quais interagem com o ambiente simulado pelo servidor através de uma interface TCP (OBST; ROLLMANN,

2004), podendo receber mensagens sobre o estado atual da simulação e enviar respostas que detalham ações a serem tomadas pelo agente simulado.

Os times competidores são formados por 11 agentes, sendo cada um deles um processo separado. A comunicação direta entre processos é considerada violação das regras (DORER; GLASER, 2014), porém a comunicação entre os agentes é permitida, desde que feita por meio do servidor através das mensagens estruturadas pelo mesmo. Tal regra limita a quantidade de informação que pode ser compartilhada entre os agentes, bem como impossibilita a utilização de um controle centralizado, mantendo o foco na criação de agentes totalmente autônomos.

As regras da competição são alteradas anualmente, refletindo o nível de desempenho das equipes participantes. Estas regras são aplicadas pela utilização de um árbitro automático, capaz de detectar infrações a regras simples, bem como modificar o estado de jogo quando da realização de um gol ou quando a bola sai dos limites do campo. Para a aplicação de regras mais complexas, um árbitro humano é designado durante a competição para acompanhar e modificar, caso necessário, a partida através de uma interface de monitoramento (Robocup org., 2010).

O sistema não possui uma interface gráfica, mas a RoboCup disponibiliza duas aplicações capazes de se conectar ao servidor para a visualização dos jogos, o rcsmonitor3D, com uma interface simplista e fácil de utilizar, e o RoboViz (STOECKER; VISSER, 2011), aplicação em Java com funcionalidades avançadas, através da qual um árbitro humano é capaz de interferir na partida, reposicionando objetos e modificando os estados de jogo.

2.4 SimsPark e rcserver3D

Desenvolvido por iniciativa de pesquisadores da RoboCup, o SimsPark é um sistema de simulação genérico capaz de simular a física atuante sobre múltiplos agentes em um ambiente tridimensional. Com um grande foco na flexibilidade e na intercambialidade de componentes (OBST; ROLLMANN, 2004), o simulador provê um ambiente robusto e adaptável para diversos cenários e problemas.

Embora seu propósito inicial tenha sido o uso nas competições de futebol da RoboCup, o desenvolvimento do SimsPark focou na implementação de serviços de simulação genéricos (OBST; ROLLMANN, 2004), mantendo funções específicas da simulação de futebol em um conjunto de plugins.

A fim de oficializar as ferramentas para a liga de simulação 3D foi criado o rcserver3D. Utilizando o motor de simulação criado pelo SimSpark, esta aplicação concentra as funcionalidades

dades e modelos necessários para a simulação de um jogo de futebol, bem como fornece uma interface para a comunicação via TCP com agentes externos.

A comunicação entre o servidor e os agentes é realizada através de uma mensagem com estrutura padronizada, a qual contém informações sobre o estado individual dos componentes do modelo simulado. Esta mensagem contém informações sobre os objetos detectados pelo conjunto de perceptores do agente, bem como os estados atuais de cada uma das juntas apresentadas no modelo Nao.

Como resposta o agente envia uma mensagem contendo os estados a serem refletidos nos atuadores do modelo, realizando o controle individual de cada uma das partes móveis presentes no humanóide.

Criar agentes capazes de disputar uma partida de futebol no ambiente de simulação 3D é uma tarefa complexa que demanda conhecimento de diversas áreas, dificultando o trabalho de quem almeja iniciar o desenvolvimento de seus próprios algoritmos. Por este motivo, algumas das equipes resolveram disponibilizar parte de seus códigos para auxiliar na expansão do número de participantes da liga.

Uma destas equipes é formada por um grupo de estudantes de Offenburg, na Alemanha, os quais desenvolveram um conjunto de ferramentas capaz de impulsionar iniciantes no desenvolvimento de seus agentes. A equipe *MagmaOffenburg* disponibiliza de forma aberta um *framework* completo capaz de realizar as funções básicas necessárias para a participação do agente em uma partida de futebol.

2.5 MagmaOffenburg e magma-AF

MagmaOffenburg é um time de estudantes da Universidade de Ciência Aplicada Offenburg (*University of Applied Science Offenburg*) na Alemanha, os quais, supervisionados pelo professor Klaus Dorer, competem na liga de simulação 3D da Robocup desde o ano de 2008 (DORER et al., 2011).

Algumas das conquistas do time incluem: 3º lugar na RoboCup 2014 em João Pessoa e 4º lugar na RoboCup 2013, realizada no México. Embora não seja um dos times mais bem-sucedidos da competição, a equipe ganhou notoriedade com a publicação de trabalhos dedicados a auxiliar aspirantes a ingressar na construção de agentes para a simulação 3D. Inspirados pela iniciativa do time Little Green Bats (DIJK et al., 2007), cujo código foi utilizado para o início das atividades do time alemão, os integrantes do grupo magmaOffenburg criaram e mantêm o

magmaOffenburg agent framework (DORER et al., 2011).

Desenvolvido em Java com foco na expansibilidade o magma-AF é um framework com as funcionalidades básicas para a criação e controle de agentes simulados capazes de participar de uma partida de futebol. A aplicação traz consigo os protocolos de comunicação e conexão com o servidor, um modelo de agentes e do mundo simulado e um conjunto de comportamentos básicos.

A arquitetura em camadas do sistema provê as seguintes funcionalidades: comunicação com o servidor, tradução das mensagens vindas e destinadas a camada de comunicação, modelagem das informações de mundo e do agente, centralização das crenças e ações do agente e, por fim, tomada de decisões. Esta arquitetura pode ser visualizada na figura 2.2.

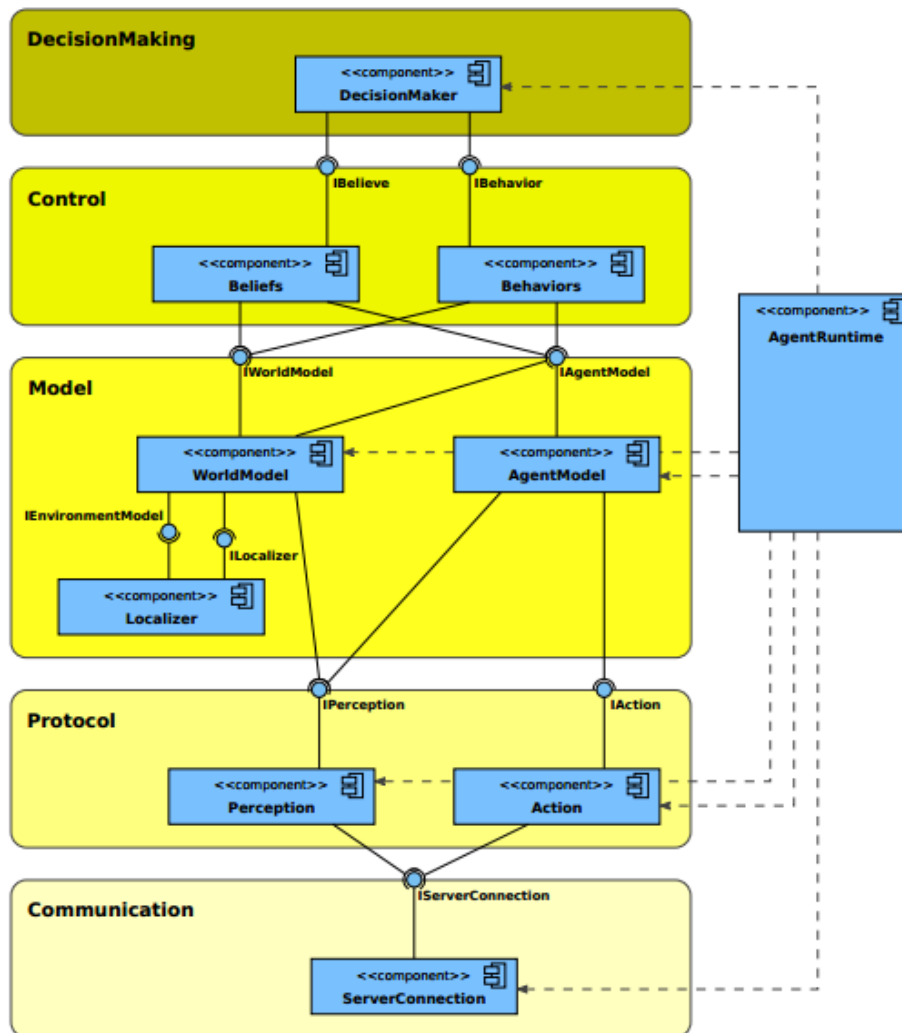


Figura 2.2: Arquitetura em camadas do MagmaAF

2.5.1 Communication

A camada de comunicação implementa os métodos de conexão e troca de mensagens entre o servidor e o agente. Realizada via TCP a comunicação consiste na passagem de *strings* contendo as informações sobre o estado do mundo e do agente no passo corrente de simulação.

2.5.2 Protocol

A camada de protocolo implementa a tradução das mensagens recebidas do servidor referentes a percepção do agente, informações como a posição dos objetos no campo de visão do agente, orientação do giroscópio e forças atuantes sobre o acelerômetro são extraídas da string recebida da camada de comunicação para serem utilizadas pelo agente.

Além da tradução das mensagens recebidas, esta camada realiza ainda a tradução das ações tomadas pelo agente em uma mensagem capaz de ser processada pelo servidor, preparando a resposta a ser enviada pela camada de comunicação.

2.5.3 Model

Acima da camada de protocolo existem os modelos representando o agente e o mundo no sistema. Contendo informações sobre todos os sensores e atuadores presentes no robô simulado, o módulo *AgentModel* encapsula toda a informação que o agente tem de si mesmo e pode utilizar para executar diferentes ações.

WorldModel é a parte do sistema responsável por armazenar as informações do mundo simulado. Este módulo contém duas representações de coordenadas: local, usando como referencial a posição do agente no estado de simulação atual, e global, a qual utiliza como referencial de origem o centro do campo.

Este modelo contém ainda informações referentes aos objetos presentes na simulação, em especial a bola e os jogadores, dimensões e posicionamento das traves, bem como informações como o tempo de jogo e o placar atual.

2.5.4 Control

Esta camada é uma abstração dos modelos de mundo e de jogador, criando uma interface de interação entre a tomada de decisão e a execução das ações no modelo de agente. Esta camada apresenta um conjunto de crenças (*believes*), as quais representam as informações contidas na

camada de modelo, e um conjunto de comportamentos básicos (*behaviors*) a serem realizados pelo agente.

As crenças apresentam informações sobre o mundo e sobre o agente que serão relevantes para a tomada de decisões. São estruturadas com um valor de certeza entre 0 e 1 para definir o quão certa é esta informação no estado atual.

A classe de comportamentos encapsula o controle dos atuadores presentes no agente simulado, refletindo as ações necessárias para movimentar e posicionar as partes individuais do robô.

Os comportamentos incluídos no framework são ações básicas a serem tomadas por agentes em uma partida de futebol. Ações como andar até um ponto, procurar a bola e estabilizar a postura são incluídas de forma a possibilitar a criação de um agente capaz de realizar tais funções em uma partida. O conjunto de comportamentos é, assim como o de crenças, expansível e personalizável.

2.5.5 DecisionMaking

Por fim encontra-se o módulo responsável pela tomada de decisões. Este módulo acessa o modelo de agente via o conjunto de crenças, processa as informações relevantes para o estado atual e dita um comportamento a ser tomado pelo agente. Como base o sistema oferece um decisor simples, consistindo na caminhada até a bola e eventual chute.

Esta camada é o foco do desenvolvimento deste trabalho, sendo a única parte do framework modificada no decorrer do desenvolvimento. É nesta camada que são implementados todos os algoritmos de decisão aqui propostos, utilizando-se das capacidades providas pelas camadas inferiores.

2.6 Sumário do Capítulo

A RoboCup proporciona um grande desafio a comunidade científica, agrupando pesquisadores em torno de um objetivo em comum: promover o avanço das áreas de robótica e inteligência artificial. O cumprimento deste objetivo está atrelado a diferentes modalidades de competição, abrangendo assim diversas áreas de pesquisa.

Uma das áreas abrangidas pelo desafio da RoboCup é a área de simulação computacional, utilizando as ferramentas de simulação como plataforma para o desenvolvimento das demais áreas

de interesse da organização. As duas ligas de simulação da RoboCup possuem características distintas, apresentando diferentes níveis de abstração e novos desafios aos participantes.

A liga de simulação 3D provê aos pesquisadores uma representação aproximada dos desafios encontrados com a criação de robôs autônomos reais, trazendo para o *software* alguns dos problemas enfrentados na construção de robôs capazes de jogar futebol. Este ambiente promove a interação entre as equipes através do desenvolvimento das ferramentas de simulação em código aberto, estimulando a cooperação e o avanço contínuo da competição e das equipes participantes.

Embora muito atrativa aos olhos de iniciantes, a construção de agentes capazes de agir no simulador não é tarefa fácil, sendo de grande valia o compartilhamento de código feito por algumas das equipes, em especial o trabalho realizado pela equipe *MagmaOffenburg* através do *framework* disponibilizado pela equipe alemã.

Este trabalho utiliza a ferramenta disponibilizada gratuitamente pela equipe, implementando algoritmos de comportamento em cima da camada de tomada de decisões. As demais camadas do *framework* foram estudadas durante o processo de criação dos algoritmos, mas permaneceram inalteradas.

3 DESENVOLVIMENTO

A implementação de um agente simulado engloba o controle individual de todos os atuadores e perceptores do modelo da simulação, bem como o processamento das informações para decidir como o agente deve responder aos estímulos do mundo ao seu redor. Construir um software capaz de realizar todas estas tarefas requer um trabalho extenso nas áreas de movimentação e percepção, deixando o controle comportamental como foco secundário.

Desta forma, trabalhos como o da equipe magmaOffenburg destacam-se como grandes impulsores de novatos na área de simulação 3D, provendo aos aspirantes um sistema capaz de realizar as funções básicas de um robô capaz de jogar futebol. O sistema desenvolvido pela equipe alemã oferece capacidades motoras básicas para o agente, como deslocar-se até um ponto, levantar-se do chão e chutar a bola, bem como a coleta de informações por parte dos perceptores. Este conjunto de capacidades é utilizado para a definição de um módulo de tomada de decisão, responsável por decidir quais ações deverão ser tomadas pelo agente em um momento de jogo.

Utilizando o *framework* magma-AF, disponibilizado de forma *open-source* pela equipe, podemos focar na definição em alto nível das estratégias de comportamento, criando algoritmos que permitem a adaptação do agente a diferentes contextos presentes na partida. Através do conjunto de informações coletadas pelos perceptores, definimos quais ações deverão ser tomadas pelo agente, criando assim agentes complexos capazes de realizar diversas funções em um campo de futebol. Chamamos tais algoritmos de decisores.

Este trabalho foca na criação de um decisor que leve em conta o contexto do jogo, bem como o papel determinado para o agente naquele momento. O agente deverá ter autonomia para mudar seu papel caso julgue necessário. A princípio consideraremos que a definição de papéis será realizada de forma autônoma pelo sistema, não sendo realizada a implementação desta troca neste trabalho.

Não houveram modificações nos algoritmos motores dos agentes, permanecendo réplicas dos algoritmos fornecidos pelo *framework*. Tais algoritmos, embora suficientes para o teste inicial das estratégias aqui propostas, não provém capacidades competitivas aos agentes implementados, sendo este um trabalho de propósito puramente exploratório.

3.1 Definição de Requisitos

Os algoritmos de comportamento implementados neste trabalho devem ser adequados as regras dispostas pela liga de simulação 3D da RoboCup. Embora o conjunto total de regras não seja aplicável no contexto deste trabalho devido a imposição de sistemas e hardware para a execução das simulações, é importante que sejam cumpridas as normas referentes ao andamento da partida, garantindo que o comportamento dos agentes em tempo de jogo seja próximo ao esperado das equipes da competição. Em especial, duas regras da competição influenciam diretamente a implementação das estratégias de comportamento.

A primeira delas, denominada *illegal defense*, refere-se ao posicionamento dos defensores e determina que em nenhum momento do jogo podem existir três jogadores do mesmo time dentro da sua própria área de penaltis (DORER; GLASER, 2014). A segunda regra a ser observada é referida por *ball crowding* e diz que, quando um adversário aproxima-se da bola, não podem haver mais de dois jogadores do time a uma distância de 1 metro da mesma (DORER; GLASER, 2014).

O conjunto de regras da RoboCup dita também que não pode haver comunicação entre processos, impedindo a comunicação direta entre os agentes. Estas disposições impedem a criação de um decisor que considere as informações de múltiplos agentes de forma simultânea, impedindo a criação de algoritmos de controle centralizado. Como não existem informações compartilhadas pelos agentes do time, o decisor implementado deve ser executado por agentes completamente autônomos, assim cada agente presente na simulação será executado como um processo diferente, totalmente independente dos demais.

Além das limitações encontradas nas regras da competição, este trabalho dispõe-se a criar agentes capazes de atuar em qualquer um dos papéis estabelecidos para o time. Os algoritmos devem ainda prever a possibilidade de mudança do papel do agente no decorrer da partida.

É importante observar ainda que a simulação do ambiente tri-dimensional impõe algumas limitações aos agentes. Neste ambiente as informações obtidas pelo agente devem ser coletadas pelo seu conjunto de perceptores, sujeitos a imprecisões. Este fator faz com que um agente não possua conhecimento preciso de todos os objetos do sistema, mantendo informações limitadas a sua percepção de mundo.

Somado a dificuldade em coletar informações precisas sobre o ambiente, os agentes estão ainda sujeitos a imprecisões físicas, tais quais desequilíbrio na caminhada e chutes imprecisos,

incorrendo em quedas e chutes mal direcionados. Estas imprecisões precisam ser consideradas na implementação das estratégias de comportamento, buscando mitigar o efeito das mesmas no desempenho dos agentes.

Estes requisitos impõem restrições na implementação da tomada de decisões por parte dos agentes, visto que um determinado agente tem informações apenas sobre seu papel e sobre o ambiente ao seu redor, não sendo capaz de determinar com precisão qual a função dos demais jogadores do seu time. Dessa forma o agente deve levar em conta apenas informações que o mesmo é capaz de coletar em seu estado atual para decidir por sua próxima ação. Deve ainda evitar a realização de ações que incorram em maior incidência de falhas físicas, a ponto de reduzir o número de quedas ocorridas durante a partida.

Com estes requisitos em mente precisamos então definir as funções a serem cumpridas pelos agentes do time, separando-as de acordo com os três papéis definidos. Devemos identificar atitudes coerentes com o papel do agente e adaptá-las a ponto de cumprir os requisitos aqui especificados.

A definição das necessidades específicas de cada uma das funções implementadas foi realizada através da observação de partidas das edições anteriores da RoboCup. Muitas das equipes participantes aparentam a utilização de estratégias de comportamento simples, mantendo seu foco na melhoria dos algoritmos de corrida e chute dos agentes simulados.

Utilizando-se das forças de seus algoritmos, equipes com agentes mais rápidos aparentam focar na utilização de um agente capaz de manter a posse de bola e aproximar-se das traves adversárias, evitando o contato com jogadores que ameacem a posse de bola. Quando na defensiva, estas equipes utilizam agentes próximos ao seu gol para criar uma barreira, buscando impedir a finalização do gol adversário.

3.2 Estratégias de Comportamento

A implementação dos comportamentos segue os seguintes princípios: o agente deve determinar no início de cada ciclo se está em condições de agir, ou seja, se está em pé e o jogo não está parado. Havendo determinado que se encontra pronto para jogar, o agente deve então considerar seu papel no time e seguir o comportamento adequado.

Com base em um time de futebol convencional, foram definidos três papéis distintos: defesa, meia e ataque. Para determinar qual o comportamento exigido de um agente de cada função, foram estabelecidos objetivos para cada uma delas, sendo o comportamento estabelecido

de forma que os agentes consigam cumprir tal objetivo quando em situação de jogo. Cada uma destas funções constitui a implementação de um comportamento distinto, com base no objetivo a ser alcançado pela função.

Um jogador da defesa possui dois objetivos: evitar a marcação de um gol adversário e retirar a bola do campo de defesa. Durante o jogo um agente que assume a função de defensor deve ser capaz de identificar qual destes objetivos deve ser cumprido no momento atual, respondendo com ações relevantes ao estado de jogo identificado. Por exemplo, se o agente determina que um adversário detém a posse de bola no seu campo defensivo, ele deve preparar-se para cumprir o primeiro objetivo.

Um jogador de ataque, por sua vez, possui um único objetivo bem delineado: Marcar um gol. Para tal são necessárias que sejam cumpridas algumas condições: a bola deve estar no seu campo ofensivo e próxima o suficiente do gol adversário, de forma que o agente consiga chutá-la com o intuito de pontuar.

Em seguida são detalhados os comportamentos referentes as três funções estabelecidas pelo decisor implementado neste trabalho.

3.2.1 Defesa

Como mencionado anteriormente, um defensor possui dois objetivos em uma partida, sendo estes os pontos iniciais da construção dos algoritmos de comportamento deste trabalho.

O objetivo primário de um agente com esta função é evitar a marcação de gols adversários. Para que o cumprimento deste objetivo seja necessário, necessitamos que as seguintes condições sejam verdadeiras: o adversário está próximo da bola e existe um risco do adversário chutar em direção ao meu gol. Caso tais afirmações sejam constatadas, o agente deve ser capaz de interceptar ou impedir o chute adversário, buscando assim evitar a marcação de um gol.

A primeira condição é simples de se constatar, basta observar a posição da bola e dos agentes em seu entorno. Caso um deles seja um adversário, a condição é validada. Para determinar se existe perigo do adversário marcar um gol, o agente deve levar em consideração a posição da bola em relação ao gol do seu time, estabelecendo uma “zona de perigo”, dentro da qual existe a possibilidade de um chute adversário.

Constatada a veracidade das afirmações acima o defensor deve decidir entre dois comportamentos capazes de evitar a marcação do gol adversário: posicionar-se de forma a bloquear a bola após o chute ou disputar a posse de bola com o adversário, movendo-se até a mesma para

evitar a realização do chute. A primeira possibilidade é realizada através da do cálculo de uma linha possível de chute adversário utilizando a posição da bola e do gol, a qual será utilizada pelo agente para determinar uma posição capaz de bloquear a bola em movimento.

Caso o agente determine que é o jogador do time mais próximo da bola, ele tentará interceptá-la antes do chute do adversário. Caso contrário, tentará formar um bloqueio entre a posição do adversário e a linha do gol. Em ambos os casos o defensor deve observar o cumprimento da regra *illegal defense*, determinando nova posição caso possa infringir a mesma.

O segundo objetivo dos defensores será cumprido quando não detectar um adversário próximo o suficiente da bola para ameaçar a marcação de um gol e quando não exista um agente do time capaz de realizar tal objetivo. Caso exista um agente do time próximo da bola, o defensor não precisa realizar tal tarefa, ela será realizada pelo agente que se encontra em melhores condições para tal. Caso contrário, o defensor deverá mover-se até a bola e chutá-la na direção do campo adversário, colocando a bola longe da sua própria área.

3.2.2 Meia

Um agente que assume a função de meia terá os seguintes objetivos: 1- Evitar o avanço adversário no meu campo defensivo. 2- Apoiar o ataque em seus objetivos.

Seu objetivo primário é evitar que o adversário consiga avançar a bola na direção do seu campo defensivo, agindo como a primeira linha de defesa do time. Tal função é cumprida quando o agente detecta a posse de bola adversária, seja ela em qualquer posição do campo. De tal forma, o agente deverá disputar a posse de bola em todos os momentos do jogo, atuando tanto no campo defensivo quanto no campo ofensivo. Recuperando a posse de bola no campo defensivo, um meia tentará realizar um passe para agentes posicionados a frente da bola, a fim de levar a bola até o campo de ataque.

O segundo objetivo entra em ação quando o próprio time possui a bola, sendo um jogador desta função capaz de avançar a bola na direção do campo ofensivo, possibilitando o cumprimento das funções de ataque do time. Quando cumprindo esta função, o agente deve ser capaz de posicionar-se relativo ao agente que detém a posse de bola no momento, criando uma segunda linha ofensiva. Este comportamento tem por objetivo ampliar a presença ofensiva do time, possibilitando a manutenção da bola no campo do adversário por mais tempo. O agente deve ainda ser capaz de controlar a bola caso possível, levando-a na direção do gol adversário e realizando a finalização caso aproxime-se o suficiente.

O diagrama mostrado na figura 3.1 ilustra as decisões tomadas pelos agentes que cumprem este papel. Neste diagrama, os pontos de decisão são denotados pelos nodos grifados, os quais representam funções de decisão. Estas funções retornam valores booleanos que indicam o caminho a ser seguido. Os demais nodos representam a ação a ser realizada pelo agente naquele momento. Ao final da execução obteremos um objeto do tipo *IBehavior*, o qual codifica a ação indicada.

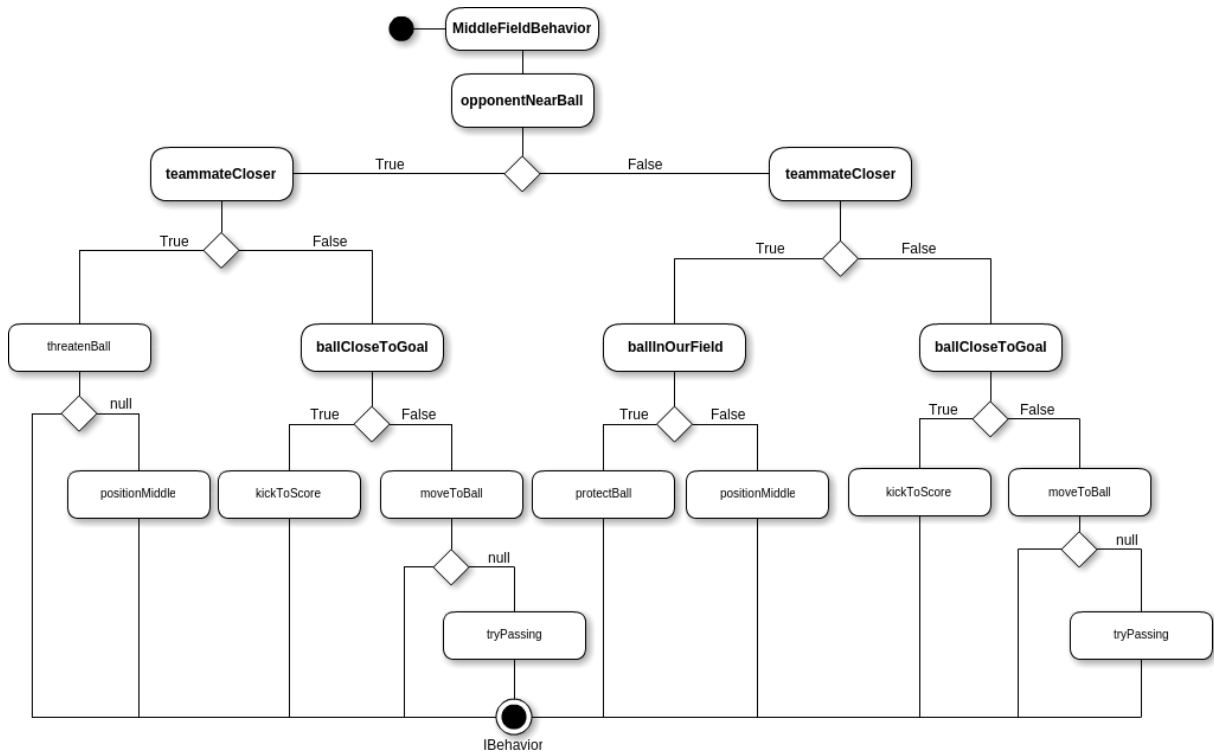


Figura 3.1: Estrutura do comportamento dos meias

3.2.3 Ataque

Um atacante possui apenas um objetivo durante todo o jogo: marcar um gol. Para tal são necessários diversos passos, realizados antes do chute final.

O fator primordial para que haja a possibilidade da marcação de um gol, é que a bola deve estar próxima o suficiente do gol adversário para que o agente consiga chutá-la e pontuar no jogo. Sendo assim, um atacante deve ser capaz de trabalhar com a bola no campo ofensivo, estando ela próxima do gol adversário ou do centro de campo.

A forma mais efetiva de avançar a bola no campo adversário é carregá-la na direção de gol através de movimentos curtos, mantendo a bola próxima aos seus pés durante seu avanço. Carregando a bola o agente precisa reduzir a velocidade de seu movimento, ficando vulnerável a

avanços adversários. Por esta razão, um atacante deve identificar a presença de jogadores que ameacem a posse de bola e reagir de acordo, realizando um passe para agentes a sua frente, caso encontre receptores adequados.

Estando a bola a uma distância do gol julgada propícia para a marcação do gol, o agente deve definir uma linha entre a bola e o centro do gol adversário, determinando uma direção para realizar o chute.

Um atacante que não detem a posse de bola pode realizar duas ações diferentes, auxiliar o agente com a posse de bola no momento ou avançar no campo para receber um passe. A primeira opção é realizada mantendo-se próximo o suficiente da bola para reagir caso o jogador com a bola encontre dificuldades ou sofra uma queda. Deve ainda garantir que não estará infringindo a regra referente a *ball crowding*.

Sua segunda opção é realizar um avanço no campo adversário com o objetivo de posicionar-se para receber um passe. Para tal o agente deve determinar a posição atual da bola, posicionando-se próximo o suficiente para manter-se ao alcance de um passe aliado.

Quando a bola encontra-se no campo defensivo, um agente com função de ataque deve posicionar-se próximo ao centro de campo, encontrando-se pronto para agir quando o restante do time recuperar a posse de bola.

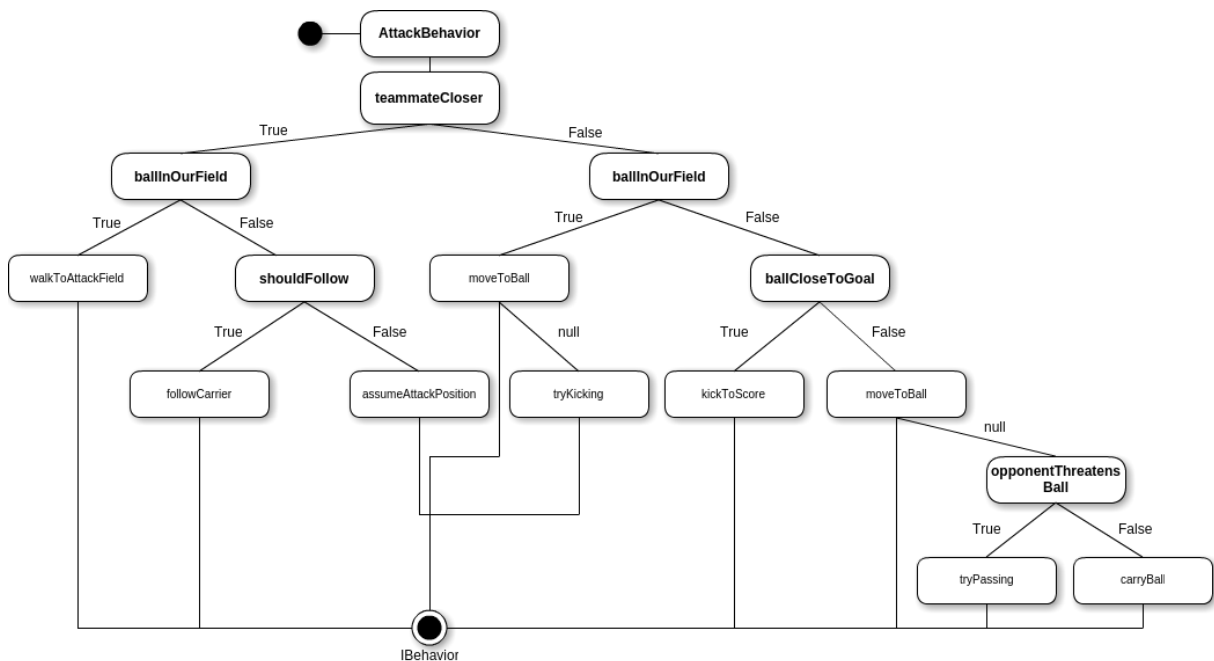


Figura 3.2: Estrutura do comportamento de ataque

A tomada de decisões por parte dos atacantes é ilustrada na figura 3.2.

3.3 Estudo das classes do *framework*

A implementação dos algoritmos de comportamento foi realizada através da extensão da classe de comportamento básico fornecida pelo *framework*, denominada SoccerDecisionMaker. Tal classe contém uma instância dos modelos de agente e de mundo, armazenados na classe ThoughtModel, bem como um mapa dos *beliefs* e *behaviors* inclusos com o software. A tomada de decisões do sistema é realizada através de uma chamada a função *decideSoccer*, a qual retorna um objeto que armazena a ação a ser tomada pelo agente no próximo ciclo de simulação.

```
public SoccerDecisionMaker (Map<String, IBelief>believes, Map<String, IBehavior> behaviors,
    IThoughtModel thoughtModel, int serverVersion)
```

3.3.1 ThoughtModel

Para a tomada de decisões complexas, é necessário que levemos em conta diversas informações referentes tanto ao ambiente de jogo, quanto aos agentes e objetos em campo. O *thoughtModel* cumpre esta função, armazenando e processando informações referentes ao passo de simulação atual. Entre os objetos armazenados nesta classe, destacam-se listas de agentes e suas posições conhecidas no campo, bem como um modelo de mundo, o qual encapsula informações sobre o ambiente.

O modelo de mundo armazena informações referentes ao campo de jogo, bem como dos objetos pertencentes a partida, tais quais a bola e os postes. Esta classe provê ao agente conhecimento sobre as dimensões do campo, a posição conhecida dos objetos visíveis durante a partida e as informações específicas do agente, encapsuladas nas classes *IVisibleObject* e *IPlayer*.

Neste modelo de mundo existe uma abstração do campo utilizando coordenadas retangulares, na qual o centro de campo representa a origem do sistema de coordenadas, sendo o eixo X referente ao comprimento do campo (distância entre ambos os gols), o eixo Y referente a largura e por fim o eixo Z representa a altura. O campo defensivo do adversário será representado com as coordenadas de X positivas, sendo (15,0,0) a coordenada do centro do gol adversário. Os modelos são responsáveis por transformar as informações recebidas pelos perceptores do agente em posições retangulares que seguem esta abstração.

3.3.2 IBelief

O conjunto de *beliefs* provê o agente com funções que encapsulam conhecimentos do agente sobre o mundo, realizando o processamento de informações captadas pelo agente e retornando um valor de verdade determinado pelo mesmo. A sua utilização é dada através da criação de um objeto *IBelief* através da lista de possíveis crenças do sistema. O trecho a seguir instancia uma crença sobre a posição da bola e do agente, e testa se é ou não possível chutar a bola.

```
IBelief ballKickable = believes.get (IBeliefConstants.BALL_KICKABLE);
if (ballKickable.getTruthValue() >= 0.6){...}
```

Um valor de verdade acima de 0.6 representa uma certeza superior a 60% de que o agente é capaz de chutar a bola na posição em este se encontra. A utilização dos *beliefs* não é estritamente necessária, mas provê uma abstração dos cálculos utilizados para determinar a condição especificada por cada um deles, encapsulando alguns dos pontos de decisão.

3.3.3 IBehavior

A realização das ações do agente é encapsulada no conjunto de behaviors provenientes do sistema, os quais devem ser repassados a classe principal do sistema para a execução dos movimentos do robô simulado. Para que a ação determinada no decisor seja realizada, o mesmo precisa retornar um objeto que implemente a interface *IBehavior*, o qual será utilizado para refletir os movimentos necessários nos atuadores do agente. A utilização destes objetos difere de acordo com a ação desejada pelo decisor. Um comportamento de corrida apresenta funções diferentes das providas pelo comportamento de chute, por exemplo.

O código a seguir exemplifica o movimento do agente até a posição indicada por *destination*, posicionando-se de frente para o lado ofensivo.

```
RunToPositionrun= (RunToPosition)behaviors.get (IBehaviorConstants.RUN_TO_POSITION);
// setPosition(pose2D, velMinima para frente, velMinima para o lado, velMax para frente, passiveMovement)
run.setPosition(new Pose2D(destination, Angle.ZERO), 30, 0, 60, false);
return run;
```

A execução deste comportamento dá-se através da instanciação de um objeto da classe *RunToPosition*, o qual implementa a interface *IBehavior*. Este objeto possui um método dedicado a receber a posição até a qual o agente deve se movimentar, recebendo como parâmetros um objeto *Pose2D* representando a posição a ser assumida pelo agente, três parâmetros de velocidade, indicando a velocidade do agente quando se movimentando para a frente ou para os lados e por

fim um parâmetro que indica se o movimento é para sua posição inicial em campo.

A realização do chute, por sua vez é mais simples, sendo necessária apenas a criação de um objeto e o eventual retorno do mesmo para que as camadas inferiores executem o movimento de chute.

```
IBehavior toExecute = behaviors.get (IBehaviorConstants.SHOOT_TO_GOAL);
return toExecute;
```

3.3.4 Vector3D e Pose2D

Para determinar a posição do agente no campo são utilizadas coordenadas retangulares obtidas através da abstração implementada pelo modelo de mundo. Tais coordenadas são representadas pela classe *Vector3D*, a qual provê a representação do sistema de coordenadas com três dimensões, bem como implementações de funções algébricas como adição de vetores e cálculo de distâncias.

A classe *Pose2D* é utilizada pelo sistema para determinar o posicionamento circular do agente em relação ao campo através do ângulo que o mesmo faz com a linha de fundo. Objetos desta classe são compostos por um objeto *Vector3D* que indica a posição no campo assumida pelo agente e um ângulo, o qual indica o lado para qual o agente está voltado.

No sistema, um agente em uma pose com ângulo 0 está voltado para a linha de fundo adversária, enquanto um ângulo de 90 graus representa um agente de frente para a ala esquerda do campo.

```
// Agente no centro do campo, voltado para o campo adversário.
Pose2D pose = new Pose2D (new Vector3D (0, 0, 0), 0);
```

3.4 Estrutura do decisor

A tomada de decisões por parte do agente pode ser compreendida de forma análoga a estrutura de uma árvore binária, existindo um ponto de partida comum para todos os agentes e ramificações referentes ao estado atual da partida de futebol. A tomada de decisão por parte de cada um dos papéis é representada por uma sub-árvore própria, como pode ser observado na figura 3.3.

Neste contexto, os nós da árvore são representados por funções booleanas que determinam em tempo de execução se uma condição é ou não válida no ciclo atual, extendendo seus ramos às ações adequadas para cada contexto identificado. Estas decisões são tomadas com base nas

informações disponibilizadas pelo *ThoughtModel*, bem como pelo conjunto de vrenças contido no sistema. Através destas informações, o agente consegue identificar qual a melhor ação a ser tomada a cada momento da partida.

Os nós folhas da árvore são representados pelas funções que ditam as ações a serem tomadas pelo agente, através do retorno de objetos do tipo *IBehavior*.

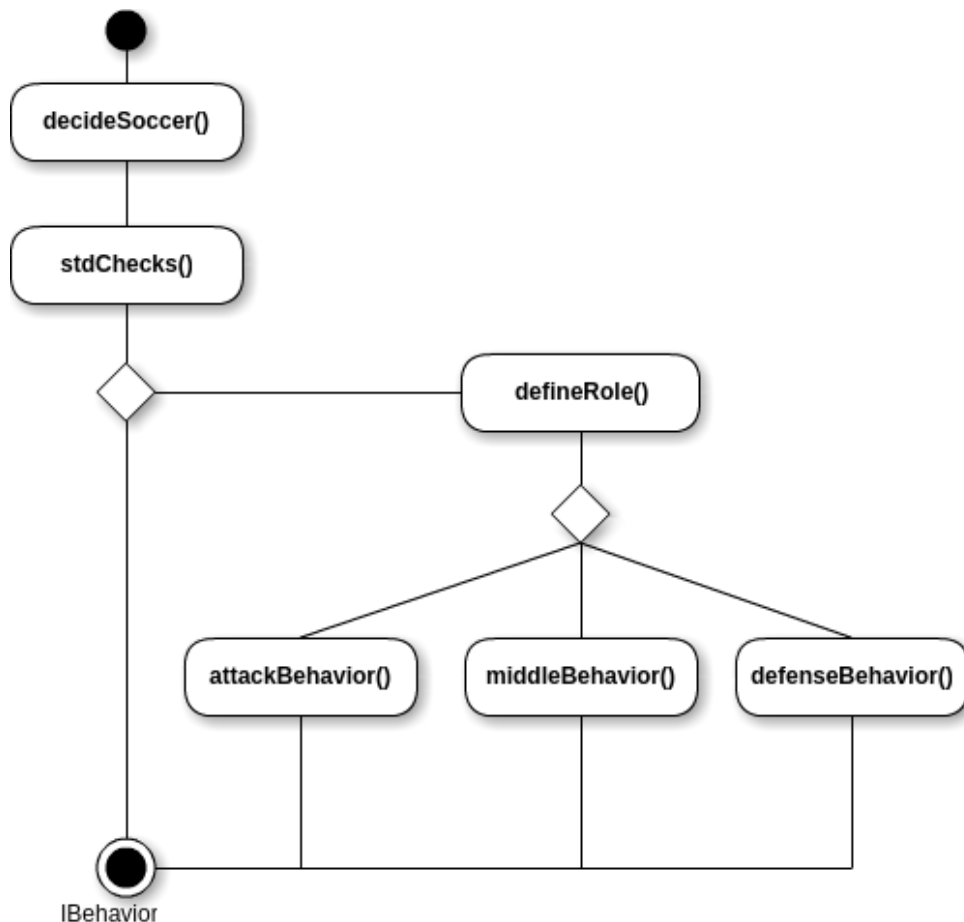


Figura 3.3: Fluxo do decisor.

3.4.1 decideSoccer()

O ponto de partida do decisor, esta função é responsável por chamar as rotinas de verificação padrão do agente, as quais determinam se o mesmo está em condições de jogo. Confirmada a capacidade do agente de agir no ciclo de simulação, ele deve verificar qual função assumirá neste ponto da partida, passando então a execução dos algoritmos específicos de cada uma das funções possíveis.

As rotinas de verificação do agente estão presentes na função *stdChecks*. Esta função trata das ações necessárias para o bom funcionamento do agente durante o ciclo de simulação. Suas

tarefas iniciam-se com a garantia de que o robô consegue manter o foco na bola, procurando-a caso perca a referência de sua posição. Além disso, é aqui que o agente deverá verificar se está em pé e em condições de realizar ações como caminhar ou chutar a bola. Caso o agente detecte problemas como uma queda ou instabilidade do agente, esta função retornará behaviors responsáveis por corrigir os problemas detectados, deixando o agente pronto para executar ações de jogo.

Além das verificações acima, este conjunto de funções realiza também o posicionamento imediato do agente quando do início da partida ou detecção de um gol, período no qual os agentes podem ser transportados para a sua posição inicial através da chamada da função *BeamHome*. Esta função transporta o agente diretamente para sua posição inicial no campo, deixando-o posicionado para a próxima saída de bola.

Completadas as verificações, o agente deve então decidir por seu papel neste momento da simulação. O método de escolha de papéis não foi explorado neste trabalho, sendo implementado apenas um método simples de definição, baseado no número do uniforme do agente em questão. Nesta implementação o agente manterá o mesmo papel durante toda a partida.

Definido seu papel o agente deverá acessar o conjunto de decisões especificado para aquele papel, seguindo o fluxo apresentado nas figuras 3.2. Este fluxo representa a tomada de decisões do agente, destacando as funções de decisão grifadas, enquanto as funções de ação encontram-se nas extremidades do diagrama.

3.4.2 Funções de decisão

As funções de decisão são implementadas através do teste das informações coletadas pelo agente sobre o ambiente da partida, retornando um valor booleano a ser analisado pelo decisor. Este valor é utilizado pelo agente para definir seu próximo passo, seja ele analisar novas informações para continuar compondo um contexto de atuação, seja decidir por realizar uma ação naquele momento.

A função *dangerousBallPosition* avalia a proximidade da bola com a posição do gol aliado, retornando *true* caso a bola encontre-se a uma distância pré-determinada do centro do gol. Esta avaliação é realizada através do cálculo da distância da posição verificada da bola até a posição conhecida das traves, armazenada no modelo de mundo do agente.

```
// Retorna true caso a bola esteja próxima do meu gol.
public boolean dangerousBallPosition() {
    Vector3D ownGoal = thoughtModel.getWorldModel().getOwnGoalPosition();
    IVisibleObject ball = thoughtModel.getWorldModel().getBall();
```

```

return (ball.getPosition().distance(ownGoal) <= 10);
}

```

De forma semelhante atua a função `ballInOurField`, utilizada para determinar se a bola encontra-se no campo de defesa do agente. Esta função utiliza-se das posições de ambas as traves e calcula a distância destas até a posição atual da bola, retornando `true` quando a distância da bola até as traves adversárias for maior do que a distância entre a bola e a trave aliada.

```

// Retorna true caso a bola esteja no meu campo defensivo
public boolean ballInOurField() {
    Vector3D ownGoal = thoughtModel.getWorldModel().getOwnGoalPosition();
    Vector3D theirGoal = thoughtModel.getWorldModel().getOtherGoalPosition();
    IVisibleObject ball = thoughtModel.getWorldModel().getBall();

    return (ball.getDistanceToXY(ownGoal) < ball.getDistanceToXY(theirGoal));
}

```

A função `ballCloseToGoal` atua de forma análoga a primeira função aqui descrita, determinando se a bola encontra-se próxima o suficiente do gol adversário para a realização de um chute a gol. Estas funções são responsáveis por determinar as ações do agente de acordo com a posição da bola no campo, porém não são suficientes para a determinação de alguns fatores do jogo.

A função `teamMateCloser`, por exemplo, é utilizada para a determinação do comportamento de todos os agentes. Esta função é responsável por identificar se o agente deve disputar a bola ou se existe outro jogador do seu time em melhor posição para fazê-lo. Esta função faz uso do conjunto de `believes` implementados pelo `framework`, retornando `true` caso o objeto apresente um valor de verdade acima de 0.7. Da mesma forma a função `opponentNearBall` utiliza um objeto do conjunto de `believes` para determinar a presença de um adversário próximo da bola.

```

// Retorna true caso acredite que exista um jogador do meu time mais próximo da bola.
public boolean teamMateCloser() {
    IBelief teammateCloser = believes.get(IBeliefConstants.TEAMMATE_CLOSER_TO_BALL);

    return (teammateCloser.getTruthValue() < 0.7);
}

```

```

// Retorna true caso exista um adversário próximo da bola.
public boolean opponentNearBall() {
    float opponentWithBall = believes.get(IBeliefConstants.OPPONENT_NEAR_BALL).getTruthValue();

    return (opponentWithBall >= 0.6);
}

```

Por fim, a função `teamMateAtPosition` é utilizada para determinar se existe um jogador ocupando a posição passada como parâmetro da função. Esta função é utilizada pelo agente quando o mesmo precisa determinar uma posição para assumir no campo. Sua implementação utiliza-se de uma lista de jogadores com posição conhecida, ordenada de acordo com a proximidade dos jogadores com este agente. Obtendo esta lista, o agente analisará a posição dos seis

primeiros agentes encontrados, pois agentes distantes não precisam ser levados em consideração neste teste.

```
// Retorna true caso exista um jogador próximo da posição argumento.
public boolean teamMateAtPosition(Vector3D position){
    List<IPlayer> teammates = thoughtModel.getPlayersAtMeList();
    int i;
    for(i=0; i< 6; i++)
        if(teammates.get(i).getPosition().distance(position) <= 0.5)
            return true;

    return false;
}
```

3.4.3 Funções de ação

Estas funções são responsáveis por determinar a próxima ação do agente, calculando posições a serem assumidas e indicando o momento de realizar um chute. Estas funções têm dois valores possíveis de retorno: um objeto que implemente a interface *Ibehavior*, representando uma ação a ser realizada pelo agente, e o valor *null* indicando que a ação não deve continuar sendo realizada neste ciclo.

Inicialmente as funções de ação realizam cálculos simples e retornam uma ação básica, como no exemplo da função *moveToBall*. Esta função acessa a posição conhecida da bola através do *thoughtModel* e calcula uma posição próxima da bola, mantendo o agente na direção da linha de fundo do campo adversário. Um dos problemas identificados na implementação das funções foi o fato de que o agente responde melhor as tentativas de chute com a perna direita, logo esta função determina uma posição que permita este tipo de chute, adicionando uma compensação no valor de *y*. Além do cálculo da posição de movimento desejada, esta função calcula ainda a possibilidade de chute, retornando um objeto do tipo *RunToPosition* caso não esteja posicionado para chutar e o valor *null* caso o agente encontre-se próximo o suficiente da bola para realizar o chute.

```
// Caminha até a bola. Retorna null caso já esteja próximo da bola para chutar.
public IBehavior MoveToBall(){
    IThisPlayer thisPlayer = thoughtModel.getWorldModel().getThisPlayer();
    Vector3D myPosition = thisPlayer.getPosition();
    Vector3D position = thoughtModel.getWorldModel().getBall().getPosition();
    // x offset: atrás da bola, em direção ao gol adversário. y offset: chuta melhor com a direita, fica a esquerda da bola.
    Vector3D destination = new Vector3D(position.getX() - 0.15, position.getY() + 0.05,
        position.getZ());
    IBelief ballKickable = believes.get(IBeliefConstants.BALL_KICKABLE);
    Vector3D me = new Vector3D(myPosition.getX(), myPosition.getY(), 0); // Desconsidera o eixo Z.
    Vector3D dest = new Vector3D(destination.getX(), destination.getY(), 0);
    // Maior precisão no teste de posição sem os valores do eixo Z.
    if (me.distance(dest) <= 0.08 && ballKickable.getTruthValue() >= 0.6) {
        return null; // Já está posicionado próximo o suficiente para chutar a bola.
    }
    else{
        // Caminha até a bola
        RunToPosition run = (RunToPosition) behaviors.get(IBehaviorConstants.RUN_TO_POSITION);
```

```

run.setPosition(new Pose2D(destination, Angle.ZERO), 30, 0, 60, false);
return run;
}
}

```

As funções *positionMiddle*, *positionDefense* e *assumeAttackPosition* são funções análogas, dedicadas a posicionar um agente quando este não age em função específica. A primeira função posiciona os defensores próximos ao próprio gol, porém fora da área de penaltis, garantindo que estes agentes estejam preparados para evitar o chute a gol da equipe adversária. Esta função utiliza como base para o posicionamento dos defensores sua posição inicial da partida, utilizando um valor simples de compensação para determinarr a posição final a ser assumida pelo agente. Para garantir o posicionamento do agente fora da área de penaltis, a distância da posição de destino até o centro do gol é utilizada.

A segunda função posiciona os agentes de forma a acompanharem o avanço da bola no campo ofensivo, utilizando a posição conhecida da bola, esta função calcula uma posição atrás da mesma, de forma a manter uma linha secundária de agentes no campo ofensivo.

```

public IBehavior positionDefense(){
    Vector3D destination = thoughtModel.getWorldModel().getHomePosition(); //Posição inicial do agente.
    Vector3D myGoal = thoughtModel.getWorldModel().getOwnGoalPosition(); //Posição do meu gol.
    Vector3D offset = new Vector3D(-0.3,0,0);
    while(destination.distance(myGoal) >= 2) //Posiciona-se a duas unidades de distancia das traves,com base em sua
        posição inicial
        destination = destination.add(offset);

    RunToPosition run = (RunToPosition) behaviors.get(IBehaviorConstants.RUN_TO_POSITION);
    //setPosition(pose2D, velMinima para frente, velMinima pro lado, velMax para frente, passiveMovement)
    run.setPosition(new Pose2D(destination, Angle.ZERO), 30, 0, 60, false);
    return run;
}

```

```

public IBehavior positionMiddle(){
    IVisibleObject ball = thoughtModel.getWorldModel().getBall();
    Vector3D ballpos = ball.getPosition();
    Vector3D destination = new Vector3D(ballpos.getX() - 2.5, ballpos.getY() - 0.5,
        ballpos.getZ());
    if(teamMateAtPosition(destination))
        destination.add(new Vector3D(0,-0.5,0));

    RunToPosition run = (RunToPosition) behaviors.get(IBehaviorConstants.RUN_TO_POSITION);
    //setPosition(pose2D, velMinima para frente, velMinima pro lado, velMax para frente, passiveMovement)
    run.setPosition(new Pose2D(destination, Angle.ZERO), 30, 0, 60, false);
    return run;
}

```

O passe será realizado através da função *tryPassing*, a qual procura um jogador capaz de receber o passe e posiciona o agente de forma a realizar um chute na direção do jogador escolhido no passo anterior. A direção de chute é calculada utilizando como base a posição atual da bola e a posição do agente que deve receber o passe, criando um vetor de chute. Para tal subtraímos a posição da bola da posição do agente, recebendo como retorno um vetor apontando para o receptor do passe. Após a obtenção deste vetor, devemos normalizá-lo através da função

normalize da classe `Vector3D`, obtendo um vetor utilizado para determinar a posição a ser assumida para o passe. Para tal adicionamos o vetor normal a posição da bola, juntamente com um valor de compensação X , indicando que o agente deve posicionar-se a X unidades de distância da bola, na direção do vetor normal. Utilizando um valor negativo, indicamos que o agente deve posicionar-se na direção oposta a do vetor, mantendo a bola entre si e a posição do receptor. Esta operação pode ser visualizada na figura 3.4.

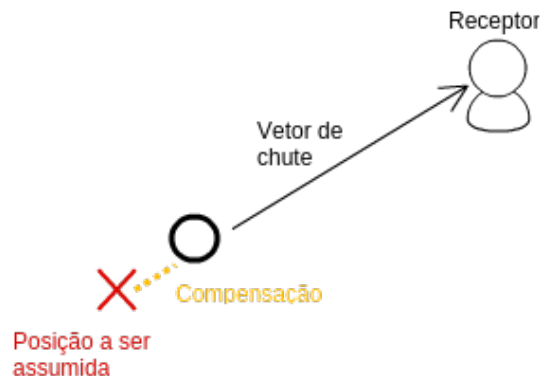


Figura 3.4: Vetor de chute.

```

IPlayer receiver = chooseReceiver();
if(receiver == null) //sem agentes capazes de receber o passe.
    return carryBall();
IPlayer thisPlayer = thoughtModel.getWorldModel().getThisPlayer();
Vector3D receiverPosition = receiver.getPosition(); //Posição do receptor.
IVisibleObject ball = thoughtModel.getWorldModel().getBall();
Vector3D meReceiver = receiverPosition.subtract(ball.getPosition()); //Vetor de chute, direção entre a
bola e o receptor do passe.
Vector3D passPosition = ball.getPosition().add(-0.15,meReceiver.normalize()); //Posição atrás da
bola, em direção ao receptor.

```

Este método de cálculo de vetor de chute é empregado também na função *kickToScore*, utilizada para que o agente realize um chute na direção do gol adversário. Neste caso calculamos o vetor de chute com base na posição do centro do gol adversário e na posição da bola. Determinada a posição a ser assumida pelo agente, esta função deve calcular a proximidade do agente com a posição desejada, retornando um comportamento de chute quando o mesmo encontrar-se pronto para chutar a bola na direção correta. Para uma maior precisão neste cálculo, excluimos os valores do eixo Z das posições do agente e de destino, realizando o cálculo da distância considerando apenas duas dimensões.

```

IBelief ballKickable = believes.get(IBeliefConstants.BALL_KICKABLE);
Vector3D me = new Vector3D(myPosition.getX(), myPosition.getY(), 0); //Desconsidera o eixo Z para o
chute, bola nao sai do chao.
Vector3D dest = new Vector3D(scorePosition.getX(), scorePosition.getY(), 0); // Exclui o eixo Z do
vetor weBall.
// Se estou na bola e posso chutar, chuto no gol.
if (me.distance(dest) <= 0.08 && ballKickable.getTruthValue() >= 0.6) {
    toExecute = behaviors.get(IBehaviorConstants.SHOOT_TO_GOAL);
} else { // Caso contrário, corro até a posição calculada para o chute.
    RunToPosition run = (RunToPosition) behaviors.get(IBehaviorConstants.RUN_TO_POSITION);

```

```

Pose2D position = new Pose2D(scorePosition.getX(),
    scorePosition.getY(), Angle.rad(shootVector.getAlpha()));
run.setPosition(position, 30, 0, 60, false);
return run;
}
return toExecute;

```

Além das funções que empregam o chute, a função *carryBall* também faz uso deste método de cálculo para determinar a direção em que o agente deve caminhar quando tem a posse de bola. Para controlar a bola, o agente calcula a posição a ser assumida utilizando o método descrito acima, porém o valor de compensação utilizado para determinar a posição é um valor baixo, neste caso -0.3. Este valor faz com que o agente movimente a bola sempre que tenta alcançar a posição, mantendo o contato com a bola durante a caminhada e avançando-a no campo ofensivo.

Em geral, o agente deverá caminhar na direção do gol adversário, mantendo a atenção em jogadores adversários que ameacem a posse de bola. O conjunto de ações contidas no *framework* não apresenta ações capazes de realizar o controle preciso da bola, dificultando a criação de comportamentos capazes de realizar dribles. Por esta razão o agente busca realizar um passe sempre que detecta a possibilidade de perder a posse de bola.

```

public IBehavior carryBall() {
    // Objetos importantes para o cálculo do destino
    Vector3D otherGoal = thoughtModel.getWorldModel().getOtherGoalPosition(); // Centro do gol adversário
    IVisibleObject ball = thoughtModel.getWorldModel().getBall();
    Vector3D meGoal = otherGoal.subtract(ball.getPosition()); // Vetor direção do gol.
    Vector3D scorePosition = ball.getPosition().add(-0.03, meGoal.normalize()); // Posição atrás da bola,
    em direção ao gol.
    RunToPosition run = (RunToPosition) behaviors.get(IBehaviorConstants.RUN_TO_POSITION);
    Pose2D position = new Pose2D(scorePosition.getX(),
        scorePosition.getY(), Angle.rad(meGoal.getAlpha()));
    run.setPosition(position, 30, 0, 80, false);
    return run;
}

```

A função *assumeDefensePosition* é responsável por posicionar os defensores quando detectam perigo de um chute a gol por parte do adversário. Esta função utiliza a posição percebida da bola e do agente adversário para calcular uma linha possível de chute, assumindo que o adversário tentará chutar na direção do centro do gol. Para tal, utiliza-se do método de cálculo da direção do chute descrito na função de passe, calculando um vetor que indica a direção do centro do gol até a posição da bola. A este vetor devemos adicionar uma distância de forma a posicionar os defensores com o objetivo de bloquear a bola em curso após o chute. Além do cálculo da posição inicial a ser tomada, diretamente entre a bola e o centro do gol, o agente deve também observar o posicionamento de outros defensores que assumam a mesma tarefa. Caso perceba-se a presença de um jogador do próprio time na posição desejada, esta função calcula uma posição alternativa com base na posição atual do agente executor, posicionando-o ao lado do

jogador anteriormente posicionado, formando um bloqueio. Esta função deve também garantir que os agentes não escolham posições fora do campo, posicionando-os antes da linha de fundo do campo.

```
//:Espera três jogadores na barreira
for(int i=0; i< 2; i++){
    if(teamMateAtPosition(blockPosition)) { //Já existe um defensor na posição, fico ao seu lado.
        double NewY = blockPosition.getY() < myPosition.getY() ? 0.2:-0.2;
        blockPosition.add(new Vector3D(0,NewY,0);
        valor1 > valor2 ?valor1: valor2;
    }
}
//-15 é a linha de fundo do campo defensivo
if(blockPosition.getX() <= -15){
    blockPosition = new Vector3D(-15.1,blockPosition.getY(),blockPosition.getZ()); //Fica dentro dos
    limites do campo.
}
```

3.5 Sumário do Capítulo

Este trabalho define estratégias de comportamento para agentes que se enquadram em três papéis diferentes: defesa, meia e ataque. Estas estratégias de comportamento seguem as normas definidas para a realização correta das partidas da RoboCup, levando em consideração as dificuldades apresentadas pelo ambiente de simulação em três dimensões.

Para a implementação destas estratégias foi necessário o estudo das classes de armazenamento de informações implementadas pelo *framework* utilizado para a criação dos algoritmos, sendo estas fundamentais para o funcionamento dos agentes.

Conhecendo as ferramentas disponíveis para a implementação das estratégias, utilizamos na definição de dois conjuntos de funções, um deles responsável pelo conjunto de decisões do agente e outro responsável pelas ações a serem realizadas durante a partida. Estes conjuntos de funções formam o comportamento do agente através de uma estrutura similar às estruturas de árvores.

4 ESTUDOS DE CASO

Os casos aqui descritos tem a função de ilustrar o funcionamento do time segundo as estratégias de comportamento definidas neste trabalho.

Nestes cenários, todos os agentes são executados como processos autônomos, capazes de comunicação com o servidor via protocolo TCP-IP. Este fato, somado ao processamento das informações do simulador e do monitor gráfico resulta em alto consumo de recursos computacionais, impedindo a realização de testes com times completos.

Desta forma, criamos dois cenários diferentes, cada um responsável por demonstrar a capacidade de uma parte do time. O primeiro cenário contém agentes desempenhando funções ofensivas enquanto o segundo apresenta as capacidades defensivas do time.

4.1 Avanço ofensivo

Este cenário põe em foco a parte ofensiva das estratégias de comportamento. Para este propósito utilizaremos quatro agentes, sendo três atacantes e um meia. Em oposição a esta formação utilizaremos três agentes estáticos representando uma formação defensiva. O cenário criado representa uma situação peculiar de jogo, em que o agente encontra resistência adversária e deve atuar em conjunto com o time para finalizar a marcação de um gol.

Quando um agente mantém a posse de bola o time deve avançar em conjunto na direção das traves até encontrar um adversário que impeça o avanço direto. A formação ofensiva do time pode ser visualizada na figura 4.1. Nesta figura vemos os jogadores de número 9, 10 e 11 desempenhando o papel de atacantes, enquanto o jogador de número 8 acompanha-os realizando a função de meia.

O agente de número 10 detem a posse de bola, enquanto o agente 9 acompanha-o com o objetivo de recuperar a bola em caso de falha do primeiro. O agente de número 11 por sua vez, avança na direção do campo adversário, buscando manter uma distância fixa da bola e permanecer ao alcance de um passe dos jogadores aliados.

Durante a criação das estratégias de comportamento, não foi obtido o controle preciso da bola, desta forma, a realização de um passe é a melhor alternativa para evitar a tomada de bola pelo adversário. Ao encontrar um adversário em sua frente, o agente deve ser capaz de identificar um jogador do próprio time capaz de receber um passe, realizando-o em seguida. A realização do passe é mostrada na figura 4.2. Na figura vemos o jogador 10 após finalizar o posicionamento,



Figura 4.1: Formação ofensiva.

realizando o movimento de chute para concluir o passe na direção do jogador de número 11.



Figura 4.2: Realização de um passe.

Como o ambiente tri-dimensional impõe dificuldades físicas sobre os agentes, não existe

forma de garantir a realização correta e precisa do passe. Este fator é um dos motivos para o posicionamento assumido pelo agente de número 9, o qual encontra-se pronto para assumir a posse da bola caso o passador não conclua com exatidão sua tarefa e continuar o avanço em direção as traves adversárias.

Aproximando-se das traves adversárias o agente efetuará um chute com o objetivo de marcar um gol. Na figura 4.3 vemos a conclusão do cenário apresentado, com o chute a gol realizado pelo agente de número 11.



Figura 4.3: Finalização da jogada.

Neste cenário, o agente encontra-se de frente para o gol adversário, o que não apresenta dificuldades para a realização do chute a gol. Em partidas completas, um agente deve ser capaz de realizar a marcação de gols de diferentes locais, realizando chutes de variados ângulos. A figura 4.4 demonstra a capacidade dos agentes aqui implementados de realizar chutes a gol de duas posições diferentes.

4.2 Teste da defesa

Este cenário ilustra o funcionamento da parte defensiva do time. Para tal utilizamos um conjunto de 5 agentes, sendo quatro destes cumprindo papel de defensores e um deles cumprindo papel de meia.

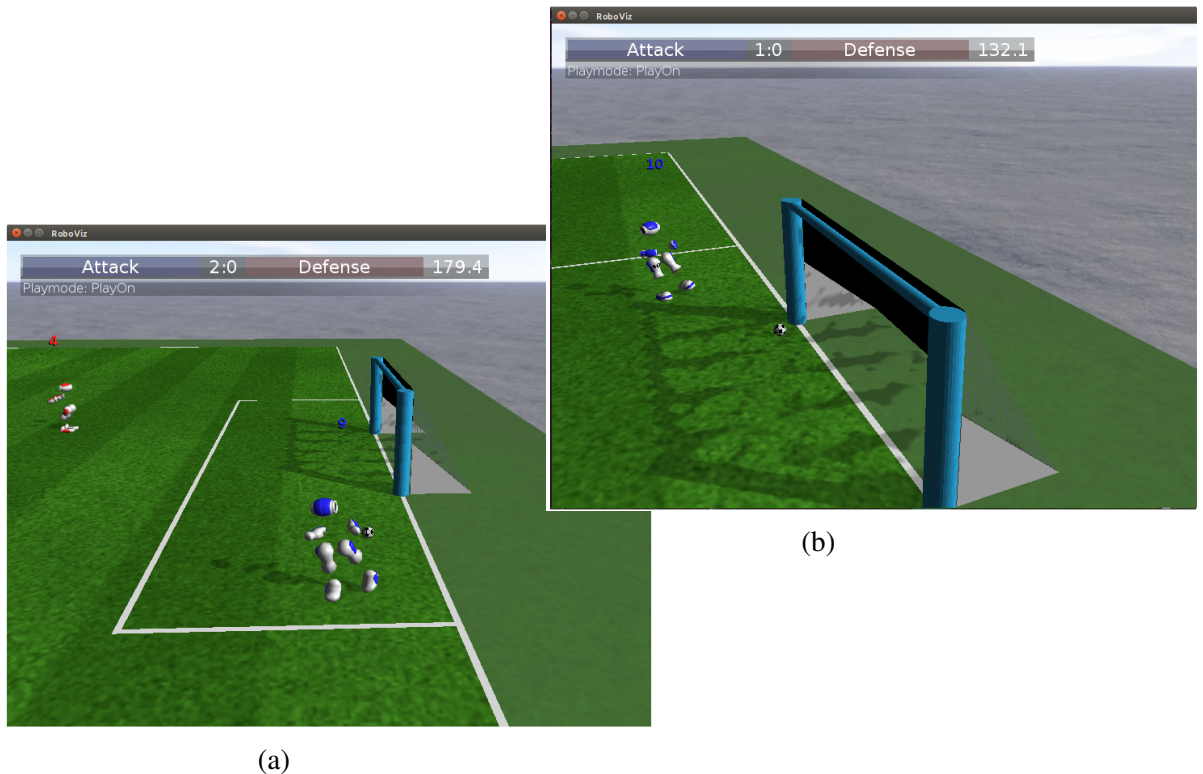


Figura 4.4: Capacidade de finalização em diferentes ângulos.

O time adversário consiste de três agentes exercendo a função de atacantes segundo o comportamento estabelecido neste trabalho.

Em um primeiro momento, vemos a formação defensiva do time, na qual os defensores posicionam-se próximos da própria área de penaltis, esperando o avanço adversário enquanto os meias tentam disputar a posse de bola. A figura 4.5 ilustra esta formação defensiva.

Neste cenário, os meias são responsáveis por disputar a bola no campo defensivo, formando a primeira linha de defesa do time. Os defensores permanecem posicionados desta forma até detectarem a aproximação da bola do seu gol. Quando isto é detectado, os agentes passam a exercer seu objetivo de evitar a marcação do gol adversário. Detectando a presença da bola nas proximidades do gol, os defensores devem coordenar duas ações diferentes, a disputa de bola, realizada por apenas um dos agentes, e a formação de uma barreira.

A formação da barreira tem por objetivo impedir a passagem da bola caso o adversário concretize um chute a gol. Para tal, os agentes calculam uma possível linha de chute e uma posição a ser assumida nesta linha. Sabendo a posição a ser assumida devem então levar em consideração a presença de agentes aliados na barreira, modificando a posição desejada de forma a manter-se ao lado do agente já posicionado na barreira.

A figura 4.6 demonstra a formação do bloqueio pelos agentes de número 4 e 5, enquanto



Figura 4.5: Formação defensiva.

o agente de número 2 recupera-se de uma queda ao avançar para disputar a posse de bola.



Figura 4.6: Formação da barreira.

4.3 Sumário do Capítulo

Neste capítulo criamos dois cenários para ilustrar o funcionamento das estratégias implementadas em uma partida de futebol de robôs. Estes cenários demonstram as capacidades ofensivas e defensivas do time.

Quando no ataque, os agentes são capazes de controlar a bola em um avanço na direção do gol adversário, utilizando a posição atual da bola para determinar a posição a ser assumida em campo. Um dos agentes é responsável por manter a posse de bola e realizar um passe caso encontre-se ameaçado por um adversário.

Os agentes defensores, por sua vez, mantêm-se em uma formação compacta, aguardando a atuação dos meias na disputa de bola em primeira instância. Quando os adversários aproximam-se do gol, os defensores são capazes de disputar a posse de bola e formar um bloqueio com o intuito de interceptar a bola quando da ocorrência do chute adversário.

5 CONCLUSÃO

Este trabalho tinha por objetivo criar estratégias de comportamento para robôs em uma simulação tridimensional, visando auxiliar no processo de criação dos módulos comportamentais do robô físico da equipe TauraBots. A equipe fora dividida em grupos, um deles responsável pelo desenvolvimento das estratégias de comportamento do robô, enquanto outro grupo desenvolve os algoritmos de controle motor do robô.

Por esta razão este trabalho focou na definição dos módulos comportamentais, não exibindo preocupações com os algoritmos de controle dos movimentos de baixo nível dos agentes. Para isso obtivemos o suporte de uma ferramenta open-source capaz de prover tais algoritmos para a criação dos comportamentos deste trabalho.

O estudo do *framework* magmaAF agrega conhecimento importante para a criação de módulos comportamentais para o robô físico, pois demonstra de forma prática os objetos importantes para a tomada de decisão dos agentes, bem como ilustra as dificuldades encontradas quando trabalhamos com a abstração das funções de baixo nível.

A utilização desta ferramenta com foco somente na camada de tomada de decisões nos permite explorar comportamentos sem considerar a fundo o funcionamento das capacidades básicas do agente. Esta abordagem impõe também limitações ao desempenho dos agentes, visto que os algoritmos de movimentação disponibilizados não são otimizados para uso nas competições. Desta forma, os agentes implementados são limitados nas ações que podem executar, ficando o programador impossibilitado de criar ações complexas que requerem movimentos mais precisos dos agentes.

Observando as limitações impostas pelo ambiente de simulação e pelo *framework*, foram criadas estratégias de comportamento capazes de gerir de forma simples um time de agentes em uma simulação de futebol. As estratégias aqui descritas foram implementadas seguindo um fluxo bem definido, utilizando-se do conhecimento obtido pelos agentes em tempo de execução para definir as ações a serem realizadas durante o decorrer de uma partida simulada.

Estas estratégias apresentam um ponto de partida para a definição de comportamentos mais complexos que podem ser futuramente trabalhados pela equipe. Para desenvolvimentos futuros no ambiente de simulação, seria adequado o estudo e aprimoramento das funções de baixo nível, melhorando as habilidades motoras dos agentes de forma a propiciar maior número de alternativas na criação de estratégias de comportamento.

REFERÊNCIAS

- Aldebaran Robotics. **Aldebaran Robotics homepage**. <http://www.smallvcm.com/>.
- CAMPBELL, M.; HOANE, A. J. J.; HSU, F.-h. Deep Blue. **Artificial intelligence**, Philadelphia, PA, USA, v.134, n.1, p.57–83, 1980.
- CHWIF, L.; MEDINA, A. C. **Modelagem e Simulação de Eventos Discretos**: teoria e aplicações. 1st.ed. São Paulo, SP, Brasil: Bravarte, 2006.
- DIJK, S. v. et al. **Little Green BATS Humanoid 3D Simulation Team Description Paper**. http://www.researchgate.net/publication/266577123_Little_Green_BATS_Humanoid_3D_Simulation_Team_Description_Paper, acessado em Setembro de 2015.
- DORER, K. et al. **The magmaOffenburg 2011 RoboCup 3D Simulation Team**. <http://robocup.hs-offenburg.de/nc/downloads/>, acessado em Agosto de 2015.
- DORER, K.; GLASER, S. **RoboCup German Open 2014 3D Simulation League Rules Version 1.0**. http://www.robocupgermanopen.de/sites/default/files/rules_GO_2015_3DSim.pdf, acessado em Setembro de 2015.
- KITANO, H. et al. RoboCup A Challenge Problem for AI. **AI Magazine**, [S.l.], v.18, n.1, p.73–85, 1998.
- OBST, O.; ROLLMANN, M. Spark – A Generic Simulator for Physical Multi-agent Simulations. In: **Multiagent System Technologies**. [S.l.]: Springer Berlin Heidelberg, 2004. p.243–257.
- Robocup org. **SimsPark User Manual**. <http://sourceforge.net/projects/simspark/>, acessado em Agosto de 2015.
- Robocup org. **About Robocup**. <http://www.robocup.org/about-robocup/>, acessado em Agosto de 2015.
- RoboCup org. **RoboCup 2015 Calls for Participation**. http://wiki.robocup.org/wiki/Soccer_Simulation_League#Calls_for_Participation_.28RoboCup_2015.29, acessado em Agosto de 2015.

STOECKER, J.; VISSER, U. RoboViz: programmable visualization for simulated soccer. In: ROBOT SOCCER WORLD CUP XV, Istanbul, Turkey. **Anais...** [S.l.: s.n.], 2011. v.1.