**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CUROS DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO**

# PROTÓTIPO DE FORMAÇÃO EM TIMES DE FUTEBOL DE ROBÔS UTILIZANDO ROBÓTICA BASEADA EM COMPORTAMENTO

**JULIO CESAR MAFRA**

**BLUMENAU**
**2004**

**2004/2-28**

**JULIO CESAR MAFRA**

# PROTÓTIPO DE FORMAÇÃO EM TIMES DE FUTEBOL DE ROBÔS UTILIZANDO ROBÓTICA BASEADA EM COMPORTAMENTO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso do curso de Ciência da Computação — Bacharelado.

Prof. Jomi Fred Hübner - Orientador

**BLUMENAU**
**2004**

**2004/2-28**

# PROTÓTIPO DE FORMAÇÃO EM TIMES DE FUTEBOL DE ROBÔS UTILIZANDO ROBÓTICA BASEADA EM COMPORTAMENTO

Por

**JULIO CESAR MAFRA**

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso, pela banca examinadora formada por:

| | |
|---|---|
| Presidente: | Prof. Jomi Fred Hübner – Orientador, FURB |
| Membro: | Prof. Mauro Marcelo Mattos |
| Membro: | Paulo César Rodacki Gomes |

Blumenau, 15 de dezembro de 2004

Dedico este trabalho a todos aqueles que me ajudaram diretamente na realização deste.

## AGRADECIMENTOS

Primeiramente, agradeço a minha família e namorada pela força, compreensão, paciência, apoio e carinho; a Deus, pelo seu imenso amor e graça; a Sênior Sistemas Ltda., amigos colegas do ambiente de trabalho e de estudo; aos professores desta entidade pelo aprendizado e pelo crescimento como ser humano.

Agradecimento especial ao meu orientador professor Jomi Fred Hübner pela força, paciência, conhecimento e apoio dispensados nesse período do desenvolvimento desse trabalho.

## RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo de formação em times de futebol de robôs utilizando robótica baseada em comportamento. Mais especificamente, este trabalho procura apresentar a utilização da robótica baseada em comportamento através de campos potenciais e dos esquemas motor e perceptivo para possibilitar a formação de times em um ambiente dinâmico. Esse conhecimento será utilizado na criação de cinco agentes que irão definir uma formação de time. Como resultado, tem-se cinco agentes desenvolvidos para atuação no simulador TBSim do ambiente TeamBots.



Palavras chaves: Robótica Baseada em Comportamento; Campos Potenciais; Esquema Motor.

## ABSTRACT

This work presents the development of a software prototype for soccer robot team´s formation using behavior-based robotics. It intends to show the utilization of behavior-based robotics through potencial fields and motor and perceptive schemas to allow the formation of soccer teams in a dynamic environment. This knowledge will be used to create five agents who will determinate a team formation. As a result, we have five agents developed to work in the TBSim simulator of TeamBots package.



Key-Words: Behavior- Based Robotics; Potencial Fields; Motor Schema.

## LISTA DE ILUSTRAÇÕES

## LISTA DE SIGLAS

API – *Application Program Interface*

RoboCup – *The Robot World Cup Soccer Games and Conference*

IJCAI – *Internacional Joint Conference on Artificial Intelligence*

ICMAS – International Conference on Multi-Agent Systems

TBSim – TeamBot Simulator

FSA – *Finite State Acceptor Diagrams*

RS – *Robot Schema*

**SUMÁRIO**

# 1 INTRODUÇÃO

Nos jogos de futebol de robôs, existe a necessidade de estabelecer estratégias de jogo para facilitar a obtenção do objetivo que é vencer o jogo. Esses esquemas de jogo podem ser definidos de algumas maneiras. A robótica baseada em comportamento engloba metodologias como a de campos potenciais, que podem facilitar o desenvolvimento de estratégias. A necessidade que o protótipo deste trabalho visa atender é demonstrar como a robótica baseada em comportamento pode ser aplicada na formação de times para atingir o objetivo do jogo.

Nesses ambientes simulados, os times de futebol são formados por jogadores que atuam, por exemplo, na função de atacante, defensor e goleiro. Cada uma dessas funções possui uma região de atuação e características relativas à função. O goleiro deve ficar na região próxima ao gol para defender o mesmo de ataques do adversário. Os laterais devem ficar nas regiões laterais e oferecer suporte tanto no ataque como na defesa e meio-de-campo. O meio-de-campo e atacante devem estar prontos para atuar no ataque a fim de fazer gols.

O conceito de atuação acima descrito está ligado a uma formação de time composta de cinco jogadores sendo: um goleiro, dois laterais, um meio-de-campo e um atacante. Um time com essa formação que se objetiva desenvolver nesse trabalho.

## 1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho de conclusão de curso é desenvolver um protótipo de formação em times de futebol de robôs, que jogam futebol em um ambiente simulado de duas dimensões (2D), utilizando a robótica baseada em comportamento para manter uma formação adequada e vencer o jogo mais rapidamente. Esse time é composto de agentes reativos que representam os robôs.

Os objetivos específicos do trabalho são:

a) desenvolver uma classe Java para ser utilizada no TeamBots, conjunto de programas e pacotes Java para pesquisadores em robótica móvel na área de sistemas multi-agentes, que possibilite determinar a área de atuação do agente jogador utilizando campos potenciais;
b) determinar a área de atuação de cada agente, com base na função do agente jogador;
c) desenvolver agentes jogadores com as seguintes funções distintas: goleiro, lateral, meio-campo e atacante.

## 1.2 ESTRUTURA DO TRABALHO

Apresentados os objetivos do trabalho neste capítulo, o capítulo dois apresenta a fundamentação teórica para o trabalho. A primeira seção deste capítulo descreve a RoboCup, destacando sua origem, características e organização; a segunda seção apresenta o ambiente de desenvolvimento de times de robôs de pequeno porte, o TeamBots, suas características e APIs relevantes ao trabalho; a terceira seção apresenta a robótica baseada em comportamento com os campos potenciais e arquitetura de esquema motor e assuntos relacionados aos mesmos.

O capítulo três apresenta a especificação, implementação e funcionamento dos agentes que compõem o protótipo sugerido. Esse capítulo irá tratar isoladamente cada agente por terem funções e áreas de atuação distintas.

No capítulo quatro, são apresentadas as conclusões provenientes do desenvolvimento desse trabalho, bem como as possíveis extensões que a partir dele podem ser desenvolvidas.

# 2 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica necessária para este trabalho é dividida da seguinte forma: a primeira seção deste capítulo descreve a RoboCup, destacando sua origem, características e organização; a segunda seção apresenta o ambiente de desenvolvimento de times de robôs de pequeno porte, o TeamBots, suas características e APIs relevantes ao trabalho; a terceira seção apresenta a robótica baseada em comportamento com os campos potenciais e arquitetura de esquema motor e assuntos relativos aos mesmos.

## 2.1 ROBOCUP

A *The World Cup Robot Soccer* (RoboCup) é a tentativa de promover a pesquisa nas áreas de inteligência artificial e robótica através da tarefa comum de avaliar várias teorias, algoritmos e arquiteturas de agentes. Essa diversa faixa de tecnologias abrange as pesquisas na área da inteligência artificial e robótica tais como desenvolvimento de um agente autônomo, colaboração de multi-agentes, estratégias, raciocínio e planejamento em tempo-real, robótica inteligente, sensores e assim por diante (KITANO, 1998).

Apesar do objetivo primário da RoboCup ser uma copa do mundo com robôs de verdade, a RoboCup oferece um software como plataforma de pesquisa sobre os aspectos de software. A Liga de robô de software, chamada também de liga do simulador, possibilita que vários pesquisadores façam parte desse programa (KITANO, 1998).

No caso da RoboCup, o objetivo mais recente é "até 2050 desenvolver um time de robôs humanóides totalmente autônomos que possa vencer o time humano campeão no futebol " (ROBOCUP, 2004).

A primeira RoboCup, RoboCup-97, ocorreu em Nagoya no Japão, durante a *Fifteenth Internacional Joint Conference on Artificial Intelligence* (IJCAI´97) como parte especial da programação do evento (KITANO, 1998).

Essa fase contou com pelo menos quarenta times. Desde então as competições vêm acontecendo anualmente na IJCAI ou na *International Conference on Multi-Agente Systems* (ICMAS) com a participação de pesquisadores de todo mundo (LCMI, 2000).

A RoboCup está dividida em cinco faixas de competição:

a) Liga de Simulação (*Simulation league*): software movimenta jogadores (agentes) independentes para jogar futebol em um campo virtual no computador. A competição é dividida em dois tempos de 5 minutos (ROBOCUP, 2004);
b) Liga de Robô Pequeno (*Small-size Robot League (f-180)*): robôs com não mais do que 18 cm de diâmetro jogam futebol com uma bola alaranjada de golf em times de no máximo cinco integrantes em um campo com tamanho próximo ao de uma mesa de tênis de mesa. A partida é dividida em duas partes iguais de 10 minutos (ROBOCUP, 2004);
c) Liga de Robô Médio (*Middle-size Robot League (f-2000)*): robôs com não mais do que 20 cm de diâmetro jogam futebol com uma bola alaranjada de futebol em times de no máximo quatro integrantes em um campo com tamanho de 12x8 metros. A partida é dividida em duas partes iguais de 10 minutos (ROBOCUP, 2004);
d) Liga de Robô de Quatro Pernas (*Four-legged Robot League*): times com quatro robôs de entretenimento de quatro pernas (robô AIBO da SONY) jogam futebol em um campo de 3x5 metros. A partida é dividida em duas partes iguais de 10 minutos (ROBOCUP, 2004);
e) Liga Humanóide (*Humanoid League*): robôs humanóides autônomos bípedes se enfrentam “andando” e “chutando” em partidas do tipo “cobrança de *penalty*” e “um contra um” (ROBOCUP, 2004).

## 2.2 ROBÓTICA BASEADA EM COMPORTAMENTO

A área de robótica baseada em comportamento é um meio de desenvolver técnicas que são utilizadas para a navegação de robôs em ambiente. Os sistemas reativos estão contidos dentro desta área e partem do princípio da reatividade dos agentes dentro do ambiente em que se encontram (ARKIN, 1998).

Os agentes reativos são programados para reagir com base na análise do atual estado do ambiente. As informações do ambiente, que são utilizadas para a reação do agente reativo, são obtidas através de sensores ou outros dispositivos semelhantes, capazes de analisar o ambiente (ARKIN, 1998).

A programação dos agentes reativos pode ser feita através da metodologia de campos potenciais, do esquema motor e da percepção em relação ao ambiente, que são apresentados a seguir (ARKIN, 1998).

### 2.2.1 Campos potenciais

Khatib e Krogh desenvolveram a metodologia de campos potenciais como a base de geração de trajetórias sem obstáculos para sistemas robóticos móveis e sistemas de manipulação (ARKIN, 1998).

Esse método gera um campo representando um espaço de navegação baseado numa função potencial arbitrária. A função utilizada é a de atração eletro-estática de Coulomb, semelhante à lei da gravitação universal, onde a força potencial atua na área entre o robô e os objetos do ambiente (ARKIN, 1998).

A cada objeto existente está associado a um vetor com uma força ou magnitude (por convenção é um número real entre zero e um) e um sentido ou direção. Essas duas propriedades são vistas na Figura 1, na representação gráfica do vetor onde a força é representada pelo tamanho do vetor; e a direção para qual o vetor aponta, representa em que sentido a força está atuando (ARKIN, 1998).

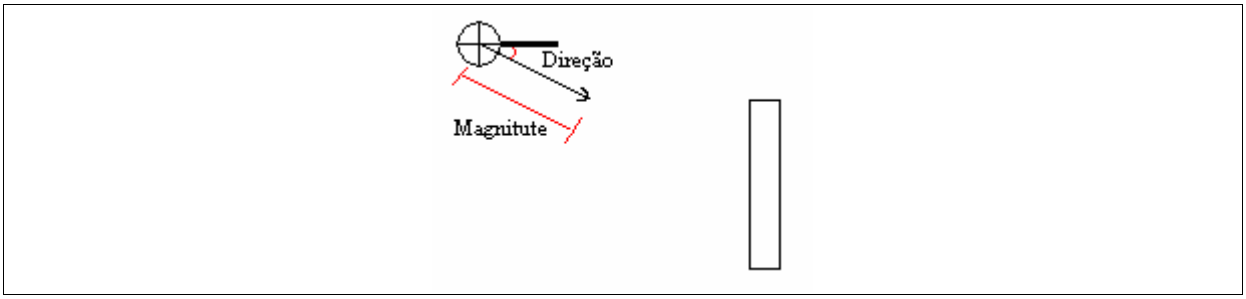



FIGURA 1 – Demonstração gráfica, exibindo as propriedades de um vetor.

Os sistemas reativos baseados em comportamento que usam os campos potenciais não geram planos baseados em todo campo, mas somente nas percepções que o robô tem do ambiente. Essa característica é importante quando o mundo é dinâmico (têm objetos se movendo, invalidando técnicas estáticas de planejamento) ou a leitura dos sensores é incerta ou ruidosa (ARKIN, 1998).

#### 2.2.1.1 Lei de Coulomb

A lei de Coulomb diz: a intensidade da força eletrostática entre duas cargas é diretamente proporcional ao produto delas e inversamente proporcional ao quadrado da distância que as separa. A fórmula dessa lei é exibida abaixo acompanhada de explicações.

$$F = K_0 \frac{|q| . |Q|}{d^2}$$

O símbolo $q$ e $Q$ representam as cargas e $d^2$ representa o quadrado da distância entre as cargas. $K_0$ é uma constante de proporcionalidade denominada constante eletrostática do vácuo e seu valor é $K_0 = 9,0 . 10^9 N.M^2/C^2$.

Maiores informações sobre a lei de Coulomb podem ser vistas em Filho.

2.2.1.2 Tipos de campos potenciais

Existem cinco tipos básicos de campos potenciais, que são enumerados abaixo:

a) campo de atração: o robô "sente" a força atrativa de determinado objeto, atraindo o robô para o objeto. Essa força que aponta em direção do objeto, diminui à medida que o robô se aproxima do mesmo. Um exemplo pode ser visto na Figura 2;
b) campo de repulsão: ao contrário da força atrativa, a força repulsiva afasta o robô do objeto que exerce a força repulsiva. Essa força aponta em direção contrária ao objeto e diminui à medida que o robô se afasta do mesmo. Um exemplo pode ser visto na Figura 3;
c) campo tangencial: esse campo faz com que o robô tangencie um objeto ficando próximo ao objeto, porém sem poder tocar o mesmo. Um exemplo pode ser visto na Figura 4;
d) campo perpendicular: os vetores apontam perpendicularmente para algum objeto ou limite do campo, podendo ser um campo de atração ou repulsão. Um exemplo pode ser visto na Figura 5;
e) campo uniforme: os vetores apontam para uma única direção. A velocidade será proporcional ao tamanho do vetor. Um exemplo pode ser visto na Figura 6.

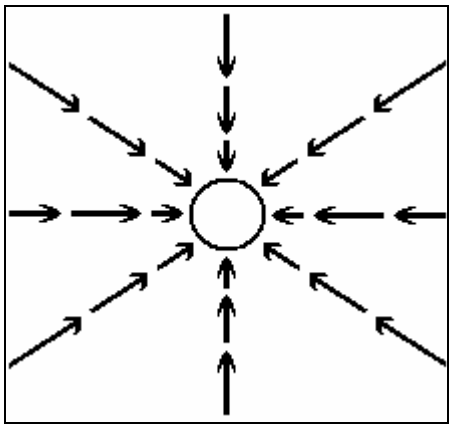


FIGURA 2 – Exemplo de campo de atração.

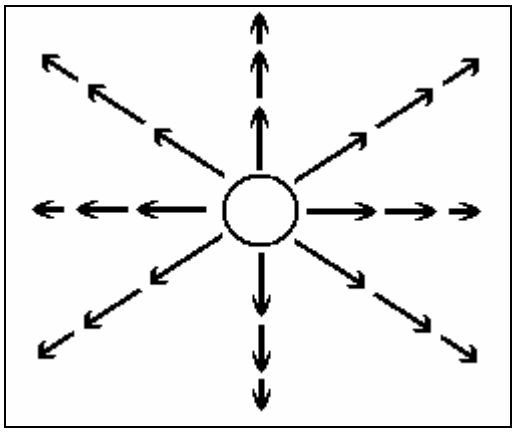


FIGURA 3 – Exemplo de campo de repulsão.

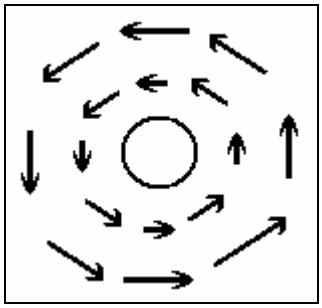


FIGURA 4 – Exemplo de campo tangencial.

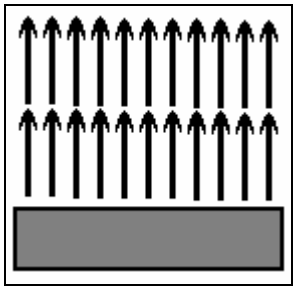


FIGURA 5 – Exemplo de campo perpendicular.

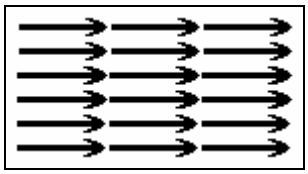
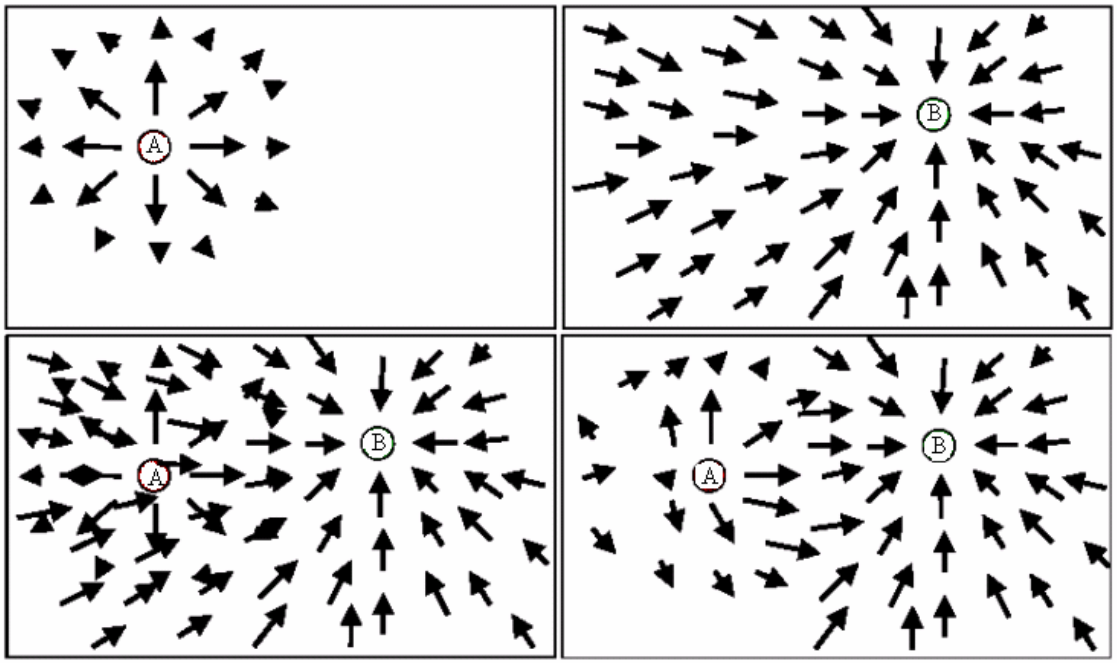


FIGURA 6 – Exemplo de campo uniforme.

Por exemplo, para o desenvolvimento de agentes jogadores de futebol, as traves são tratadas como atratores e os obstáculos (adversários e até mesmo jogadores do mesmo time) são tratados como repulsores. Os diferentes campos são criados, baseados em funções potenciais, para representar o relacionamento entre o robô e cada um dos objetos dentro do campo sensorial do robô. Depois, esses campos são combinados, normalmente através de superposicionamento, para produzir um único campo global. Para o mapeamento de um caminho, pode ser definida uma trajetória com pouco ou sem obstáculos.

Na Figura 7, no quadro superior esquerdo tem-se um objeto A que exerce uma força através de um campo de repulsão e no quadro superior direito um objeto Bque exerce uma

força através de um campo de atração dentro do ambiente percebido. No quadro inferior esquerdo é feito o sobre-posicionamento das duas forças percebidas. No quadro inferior direito é visto o resultado, ou seja, o mapeamento produzido pela atuação das forças dos campos potenciais percebidos nesse ambiente.




Fonte: Adaptado de Murphy (2000, p. 132)

FIGURA 7 – Exemplo de sobre-posicionamento de campos potenciais

A Figura 8 representa um campo de atração que é exercido pelo objeto 0 (zero) nos objetos 1, 2 e 3. Os objetos localizados próximos do círculo de menor atração sofrem uma força de atração menor por estarem mais próximos do objeto atrator, ao contrário dos objetos que se localizam próximos do círculo de maior atração. Os objetos localizados além do círculo de maior atração sofrerão a força máxima de atração, independente da distância do objeto atrator.

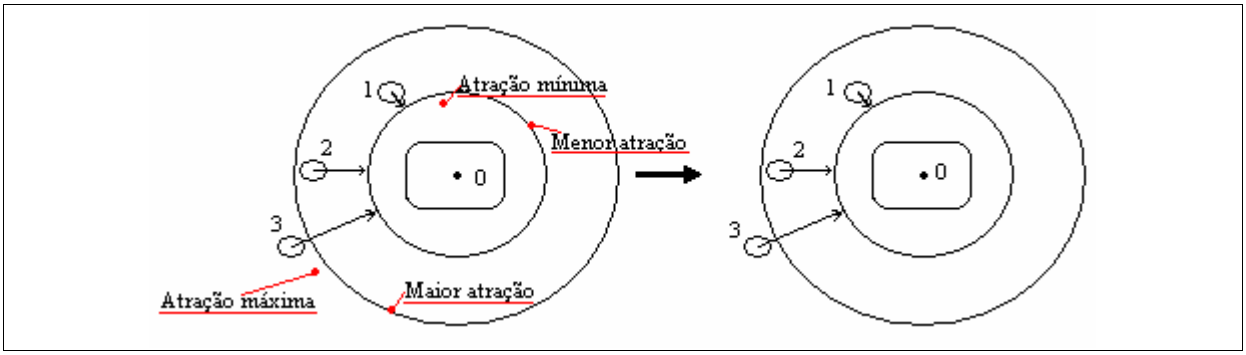



FIGURA 8 – Demonstração da influência da força atratora de um objeto.

A Figura 9 representa um campo de repulsão que é exercido pelo objeto 0 (zero) nos objetos 1, 2 e 3. Os objetos localizados próximos do círculo de maior repulsão sofrem uma

força maior de repulsão por estarem mais próximos do objeto repulsor, ao contrário dos objetos que se localizam próximos do círculo de menor repulsão. Os objetos localizados além do círculo de menor repulsão sofrerão a força mínima de repulsão, independente da distância do objeto repulsor.

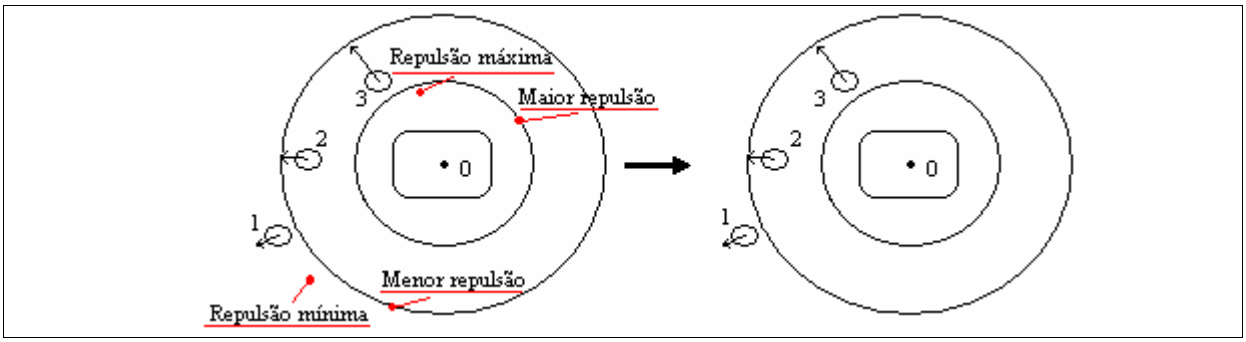



FIGURA 9 – Demonstração da influência da força repulsora de um objeto.

### 2.2.2 Arquitetura de esquemas: esquema motor (esquema motor e esquema perceptivo)

Para desenvolver agentes que utilizam os conceitos de robótica baseada em comportamento, é necessário entender a arquitetura que é utilizada na mesma.

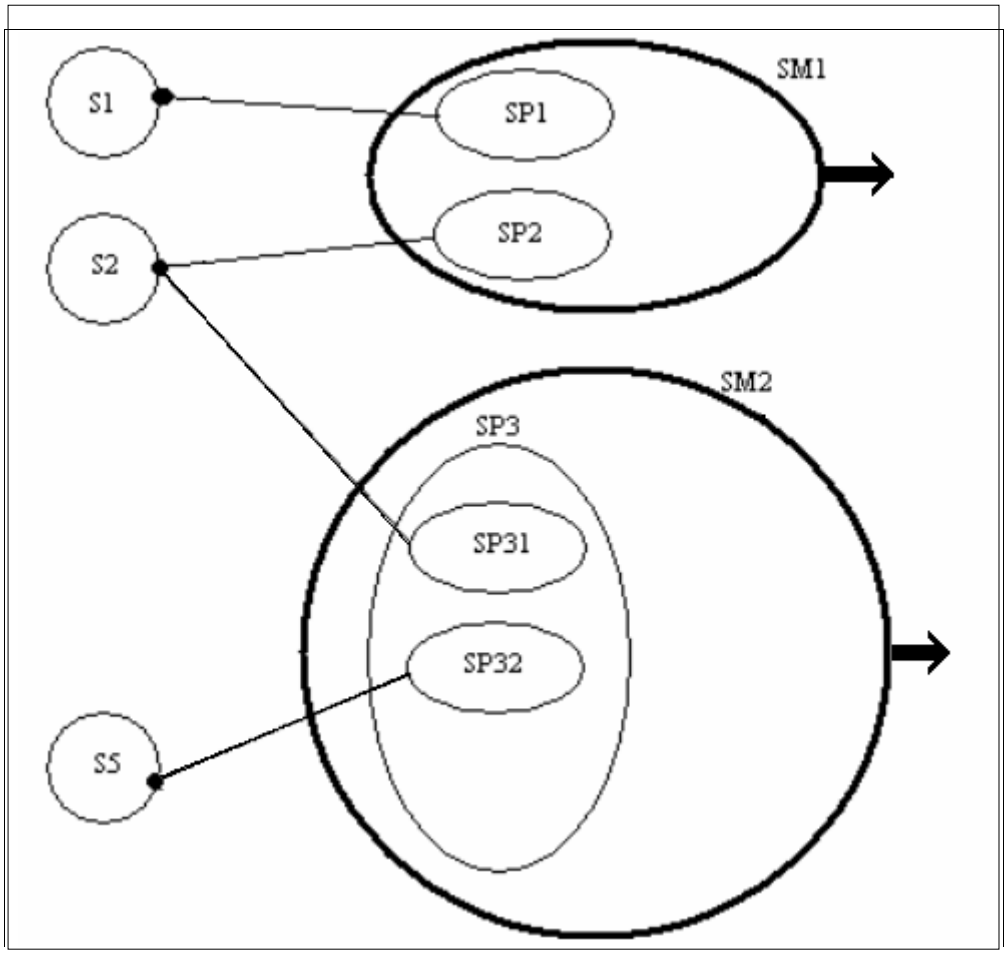
Arbib definiu que o comportamento é composto por um esquema motor e um esquema perceptivo. O esquema motor representa a forma da atividade física e o esquema perceptivo engloba as percepções. Esses dois esquemas são como as peças de um quebra-cabeça; as duas precisam estar juntas para ser um comportamento. Esses são blocos básicos de construção para navegação autônoma dentro dessa arquitetura.

Dentro dos conceitos de POO (Programação Orientado a Objetos), as classes esquema motor e esquema perceptivo, são derivadas da classe esquema. Um comportamento primitivo somente tem um esquema motor e um esquema perceptivo. No caso de uma seqüência de comportamentos, o comportamento resultante pode ser representado de duas maneiras. Uma delas é considerar que o comportamento é resultado de uma composição de comportamentos primitivos.

O esquema perceptivo está dentro de cada esquema motor. Esses esquemas perceptivos fornecem as informações específicas sobre o ambiente para aquele determinado comportamento. Esquemas perceptivos podem ser formados por sub-esquemas perceptivos.

Segundo Arkin (1998), cada esquema motor tem como saída um vetor ação (composto por magnitude e direção) que define a direção em que o robô deverá se mover em resposta ao estímulo percebido através do esquema perceptivo.

A Figura 10 apresenta dois esquemas motor. O esquema motor identificado como SM1 é composto por dois esquemas de percepção que recebem a informação do ambiente através dos sensores S1 e S2. O esquema motor SM2 é composto de um recurso perceptivo SP3 que é o resultado de dois outros esquemas perceptivos, SP31 e SP32, que recebem as informações do ambiente através dos sensores S2 e S5. Os sensores S3 e S4 estão representados porém não fazem parte desses dois esquemas motores.




Fonte: Adaptado de Arkin (1998, p. 144)

FIGURA 10 – Representação de esquemas perceptivo e motor.

Os conceitos de esquema motor e perceptivo se caracterizam com o comportamento humano e a psicologia cognitiva, como segue:

a) através da entrada sensorial, o comportamento gera uma ação motora como saída;
b) um comportamento pode ser representado por um esquema, que é nada mais que uma construção de um objeto-orientado na programação;

c) o comportamento é ativado por gatilhos;
d) a transformação das entradas sensoriais em ações motoras de saídas pode ser dividida em dois sub-processos: esquema perceptivo e esquema motor.

Em aplicações mais avançadas, o agente pode ter a opção de mais de um esquema perceptivo ou motor para melhorar o comportamento. Por exemplo, uma pessoa normalmente usa a visão (esquema perceptivo padrão) para caminhar por um ambiente (esquema motor). Mas se for um local escuro, a pessoa pode usar o tato (esquema perceptivo alternativo) para achar uma maneira de sair do ambiente escuro. Nesse caso, sabe-se que um esquema alternativo pode ser utilizado em um ambiente com diferentes condições.

Outra maneira de criar um comportamento é através da escolha entre esquemas perceptivos e esquemas motores alternativos, baseados na situação do ambiente, por exemplo.

Esses esquemas são agrupados através de um mecanismo chamado de montagem ou construção (ver seção 2.2.3 para saber mais sobre construções de comportamento). Cada montagem codifica uma rede de esquemas ou de outras montagens (ARKIN, 1998).

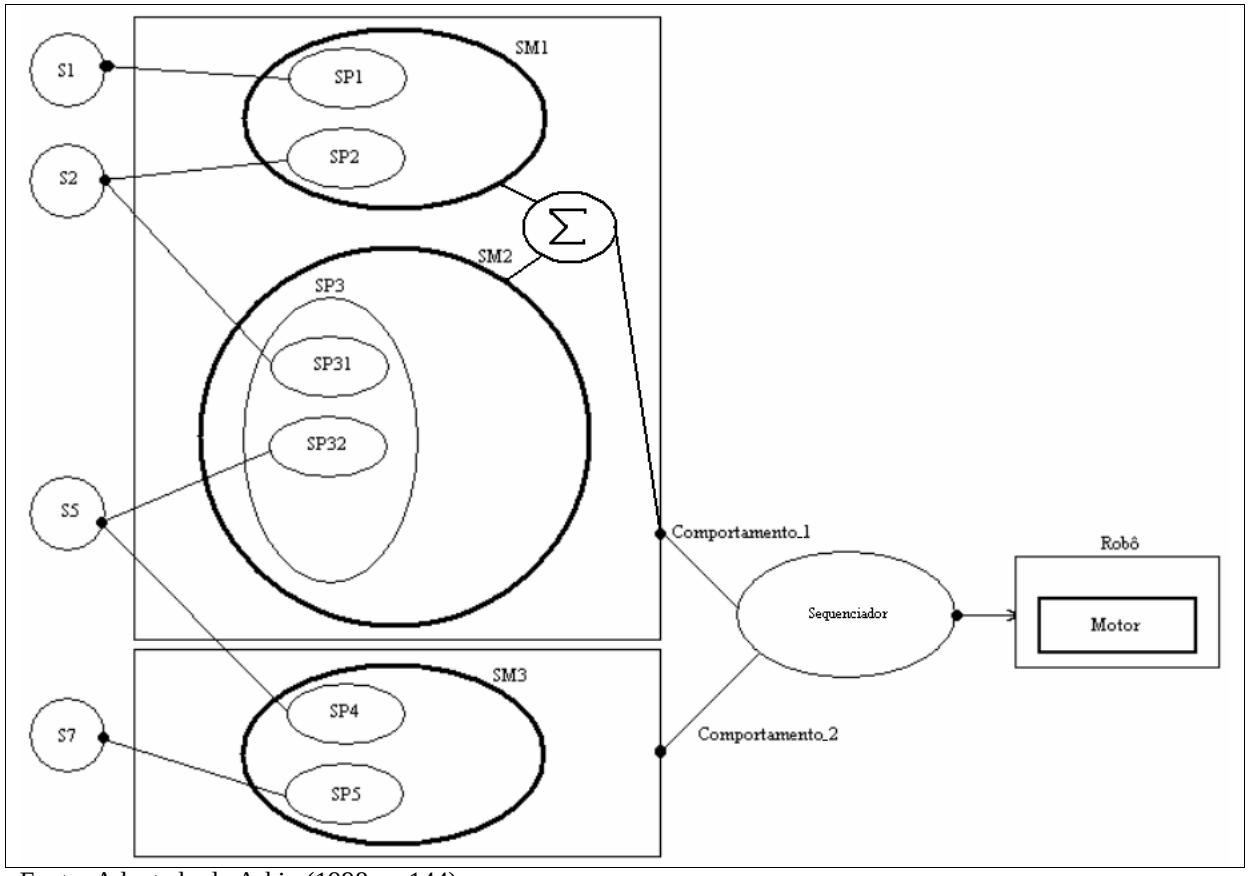
### 2.2.3 Construções de comportamentos

Construções de comportamentos são os pacotes a partir dos quais os sistemas robóticos baseados em comportamento são construídos. Uma construção é definida como uma coleção de comportamentos ou construções primitivas. O uso de construções nasceu da abstração, onde se parte de comportamentos mais simples para atingir comportamentos de mais alto nível. A abstração possibilita reutilizar construções de uma maneira modular, simples e fácil para construir sistemas baseados em comportamento (ARKIN, 1998).

O elemento de resposta motora do robô é composto por dois componentes:
a) força: indica a magnitude da resposta, que pode ou não estar relacionada à força de um dado estímulo;
b) direção: indica a direção da ação de resposta. A realização desse componente direcional de resposta necessita do conhecimento dos movimentos mecânicos do robô. Pode ser ou não dependente da força do estímulo.

A Figura 11 demonstra dois comportamentos. O Comportamento_1 é o resultado de dois esquemas motores, SM1 e SM2, e o Comportamento_2 é composto pelo esquema motor SM3. Os comportamentos passam por seqüenciador, que recebe como parâmetro a posição do

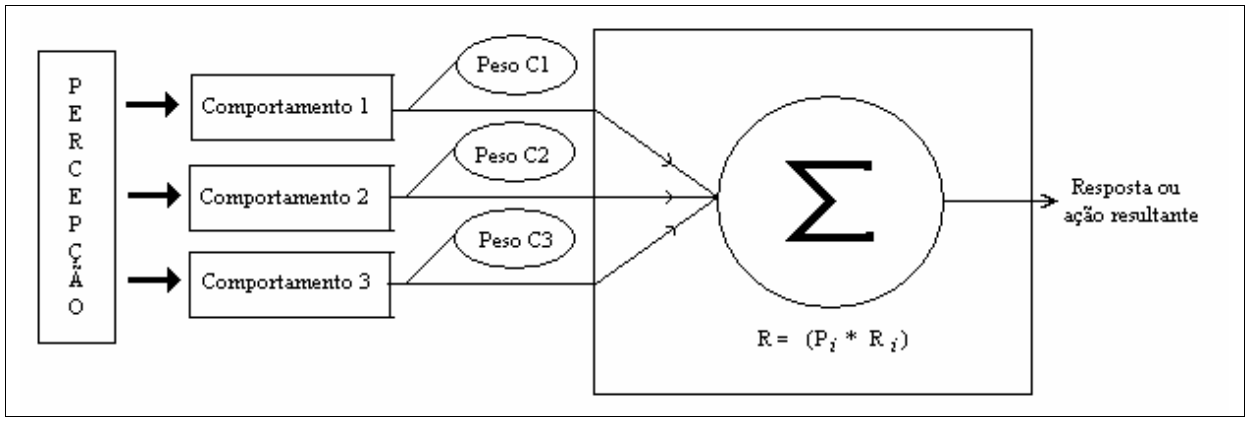
comportamento que deve ser retornado, antes da instrução ir para o robô. Para unir esses dois comportamentos é feito uso da fusão comportamental, que possibilita usar mais de uma saída de comportamentos, para obter a ação resultante. O resultado final é a combinação da saída dos comportamentos. Cada força possui um peso relativo ao comportamento que é utilizado como multiplicador do vetor antes de ser efetuado o somatório.




Fonte: Adaptado de Arkin (1998, p. 144)

FIGURA 11 – Representação gráfica de comportamentos compostos por esquemas motor e perceptivo.

Os esquemas motores criados geram *arrays* que contém os vetores a ele associados. Considerando-se que vários comportamentos podem ser concorrentemente ativados dentro do sistema robótico, cada matriz de um esquema motor é multiplicada por um peso e então somadas para obter a ação resultante da construção de comportamento. Essa ação resultante é repassada ao robô para que ative o sistema motor e se movimente no ambiente. A representação do que acaba de ser explanado pode ser visto na Figura 12. Os pesos são representados por C1, C2 e C3 e a resposta ou ação resultante (R) é o resultado do somatório dos produtos da multiplicação dos comportamentos (R*i*) pelos respectivos pesos (P*i*).




Fonte: Adaptado de Arkin (1998, p. 114)

FIGURA 12 – Representação da fusão de comportamentos através do somatório de comportamentos.

Maiores informações a respeito de robótica baseada em comportamento, como expressar comportamentos, os métodos que podem ser utilizados e assuntos relacionados poderão ser consultados em Arkin (1998) e Murphy (2000).

## 2.3 TEAMBOTS

TeamBots é o conjunto de programas e pacotes Java para pesquisadores em robótica móvel na área de sistemas multi-agentes. Teambots é distribuído com seu código-fonte aberto. O ambiente de simulação é totalmente escrito em Java. Atualmente os robôs desenvolvidos no TeamBots podem ser executados nos robôs que utilizam a tecnologia Nomadic (BALCH, 2000).

O pacote TeamBots suporta prototipação, simulação e execução de sistemas que controlam sistemas de múltiplos robôs. Sistemas para controle de robôs desenvolvidos com o TeamBots podem ser executados no programa simulador TBSim (BALCH, 2000).

### 2.3.1 Características

Uma das características mais importantes do ambiente TeamBots é o suporte a prototipação e simulação do mesmo sistema de controle que é executado em robôs móveis. O ambiente TeamBots é extremamente flexível. Ele suporta a execução de múltiplos robôs heterogêneos com sistema de controles heterogêneos. Ambientes experimentais complexos ou simples podem ser criados com paredes, estradas, outros robôs e obstáculos circulares. Todos esses objetos podem ser criados editando o arquivo de configuração (BALCH, 2000).

### 2.3.2 TBSim

TBSim faz parte do pacote de programas do ambiente TeamBots. TBSim é um programa que tem por objetivo realizar a simulação das condições encontradas no mundo real (obstáculos, outros robôs, bola de golfe, tamanho da área de atuação, etc...) para robôs da categoria de médio porte utilizada nas competições da RoboCup (BALCH, 2000).

O TBSim é utilizado para testar sistemas que controlam robôs implementados a partir da API *abstractrobot* da biblioteca Teambots.

#### 2.3.2.1 Arquivo de descrição do ambiente

O arquivo de descrição do ambiente contém a descrição do ambiente no qual os robôs vão atuar. Este arquivo tem várias seções e é de fácil entendimento, a seguir estas seções serão apresentadas e uma breve descrição de sua função no ambiente.

a) Bounds: define o tamanho do campo que será visível no simulador, este tamanho é definido em metros, se a área definida pelos limites forem diferentes dos obstáculos que delimitam a área de atuação dos robôs, isto poderá causar perda da visibilidade do robô na tela do simulador;

```
1 //======
2 // SIMULATION BOUNDARY
3 //======
4
5 bounds -1.47 1.47 -.8625 .8625
6
```

QUADRO 1 – Exemplo de configuração de bounds

b) Seed: a instrução SEED configura um número aleatório para a distribuir os jogadores. O valor padrão é -1;

```
 8 //======
 9 // SEED
10 //======
11
12 seed 3
13
```

QUADRO 2 – Funções

c) Time: configura a velocidade de execução do simulador em relação às respostas de tempo real. Configurando com 0.5, a simulação processará na metade da velocidade normal, assim como quando configurado com 1 a simulação ocorrerá em tempo real, e quando configurado com 4, a simulação processará 4 vezes mais rápida que a

velocidade normal;

```
32
33 //======
34 // TIME
35 //======
36
37 time 2.0 // 2x tempo real
38
```

QUADRO 3 – Funções

d) Timeout: a instrução *timeout* configura o tempo de duração da partida em milisegundos. O simulador automaticamente termina e encerra o aplicativo quando este tempo é alcançado. Se não for informada a instrução *timeout* a simulação não termina automaticamente;

```
22 //======
23 // TIMEOUT
24 //======
25 timeout 10000 // dez segundos
26
```

QUADRO 4 – Funções

e) Max time step: configura a tempo máximo que pode decorrer entre duas simulações. Força um pulo em computadores mais lentos, ou se a troca de processos termina o seu tempo de execução;

```
28 //======
29 // MAX TIME STEP
30 //======
31
32 maxtimestep 50 // 1/10 de segundo
33
```

QUADRO 5 – Funções

f) Background Color: configura a cor de fundo da tela do simulador. A cor deve ser informada no formato hexadecimal no formato “xRRGGBB”, onde RR indica a intensidade da cor vermelha ( o valor pode ser de 00 até FF), GG indica a intensidade da cor verde e BB indica a intensidade da cor azul. Para o campo de futebol é usado o verde escuro “x009000”;

```
35 //======
36 // BACKGROUND COLOR
37 //======
38
39 background x009000
40
```

QUADRO 6 – Funções

g) Objects: a instrução *object* faz com que um objeto seja criado no simulador. Sintaxe: *object objecttype x y theta forecolor backcolor visionclass*. O parâmetro *objecttype* indica o tipo de objeto que vai ser criado. Este deve ser informado

usando o caminho completo da localização da classe que representa este objeto. Os parâmetros *x*, *y* e *theta* indicam a posição inicial do objeto. O parâmetro *forecolor* e *backcolor* indica as cores de frente e de fundo do objeto. O parâmetro *visionclass* é usado para classificar cada tipo de objeto criado no ambiente. Com isso é possível simular a visão dos sensores dos robôs e poder organizar usando este identificador. Na simulação para a RoboCup foi criado um objeto especial que tem por objetivo desenhar o campo de futebol, o nome deste objeto é SocFieldSmallSim. Este objeto não tem nenhuma interação com os robôs ou com a bola. É criado no inicio da simulação;

```
42 //======
43 // OBJECTS
44 //======
45 object EDU.gatech.cc.is.simulation.SocFieldSmallSim 0 0 0 0 x009000 x000000 0
46
47 object EDU.gatech.cc.is.simulation.ObstacleInvisibleSim 2.047 1.4396 0 1.0
48     x000000 x000000 0
```

QUADRO 7 – Funções

h) Robot: a instrução informa ao simulador que este deve criar um robô com um sistema de controle. Sintaxe: *robot robottype controlsystem x y theta forecolor backcolor visionclass*. Nos parâmetros *robottype* e *controlesystem* devem ser informados o caminho completo da localização da classe do tipo do robô. Os parâmetros *y* e *theta* são ignorados pelos robôs, pois estes já têm posição iniciais pré-definidas. O parâmetro *x* indica se o robô esta a leste (positivo/direito) ou a oeste (negativo/esquerdo). É possível usar diferentes cores num mesmo time usando os parâmetros *forecolor* e *backcolor* (cor da frente e de fundo respectivamente). O parâmetro *visionclass* tem o mesmo funcionamento que na instrução *object*.

```
60
61 //======
62 // ROBOTS
63 //======
64
65 //======WEST TEAM======
66 robot EDU.gatech.cc.is.abstractrobot.SocSmallSim Kechze
67 //------------your control system name goes here ^^^^^^^^
68     -1.2   0     0 xEAEA00 xFFFFFF 1
69 robot EDU.gatech.cc.is.abstractrobot.SocSmallSim Kechze
70 //------------your control system name goes here ^^^^^^^^
71     -.5    0     0 xEAEA00 xFFFFFF 1
```

QUADRO 8 – Funções

Maiores informações a respeito de TeamBots poderão ser consultadas Teambots (2000) e Balch (2000).

### 2.3.3 API AbstractRobot

Essa API é a base para a criação dos agentes jogadores. A seguir são apresentadas as principais classes e interfaces que fazem parte da API AbstractRobot.

a) ControlSystemS: super classe para todos os tipos de sistemas de controle para robôs do TeamBots. Quando um sistema de controle de robôs é criado a partir da extensão desta classe, este pode ser executado no TBSim;
b) ControlSystemSS: essa classe estende a classe ControlSystemS. É a classe que o usuário do TeamBots deve estender para criar seus agentes jogadores;
c) SimpleInterface: define as compatibilidades básicas que todas as classes de robôs precisam ter. Um robô simples pode detectar obstáculos, sua posição, girar e mover. A intenção dessa classe é poder ser estendida para vários tipos de robôs reais;
d) Simple: implementa a interface SimpleInterface, permitindo assim, usar o mesmo sistema de controle para robôs diferentes tendo o mesmo sistema de simulação.
e) KinSensor: define a interface de um robô que pode perceber outros robôs;
f) KickActuator: define a interface para o ativador de chute;
g) GoalSensor: define a interface para o sensor de gol (local da trave);
h) BallSensor: define a interface para o sensor da bola;
i) SocSmall: estende as interfaces SimpleInterface, KinSensor, KickActuator, GoalSensor e BallSensor. Determina a interface para a simulação do hardware de um robô de pequeno porte da RoboCup. A simulação de tamanho e percepção é compatível com as especificações dos regulamentos da RoboCup de robôs de pequeno porte. A implementação desta interface permite a sua simulação no TBSim;
j) SocSmallSim: implementa a interface SocSmall, possibilitando a simulação.

Maiores informações a respeito da API AbstractRobot poderão ser consultadas em Balch (2000).

### 2.3.4 API Clay

Clay é um grupo de classes Java utilizadas para criar sistemas robóticos baseados em comportamento. Esta API trabalha dentro dos conceitos da arquitetura de esquema motor (ver na seção 2.3.2 informações sobre a arquitetura de esquema motor) e tem a vantagem da sintaxe Java para facilitar a combinação, mistura e abstração dos comportamentos. Esta API

pode ser utilizada para criar sistemas reativos simples ou configurações hierárquicas complexas com aprendizado e armazenamento (BALCH, 2000).

2.3.4.1 Configurando um comportamento com Clay

O bloco básico para construções do Clay é o nodo ou nó. Existem duas importantes fases na vida do nodo: a inicialização e o tempo de execução. A maioria dos nodos têm somente dois métodos, correspondendo a essas fases: o *constructor* (construtor), usado na inicialização; e *Value()*, chamado repetitivamente em tempo de execução (BALCH, 2000).

Os nodos podem conter outros nodos nele. A utilização de outros nodos é especificada na inicialização usando o nodo construtor.

No Quadro 9, o nodo detect_obstacles é criado usando a classe va_Obstacles_r. O próximo nodo, avoid_obstacles, é gerado através da utilização do nodo detect_obstacles dentro do objeto v_Avoid_va.

```
detect_obstacles = new va_Obstacles_r(abstract_robot);
avoid_obstacles  = new v_Avoid_va(2.0, 1.0, detect_obstacles);
```

Fonte: Adaptado de Balch (2000)

QUADRO 9 – Representação de encapsulação de nodos.

2.3.4.2 Principais classes da API Clay

A seguir serão enumeradas as classes mais pertinentes a esse trabalho de conclusão de curso, seguidas de breve explanação sobre sua função e um exemplo para ilustrar o uso.

a) b_CanKick_r: o método *Value()* retorna verdadeiro se o robô está em condições de chutar a bola, isso no que diz respeito à distância entre robô e bola;
b) v_Select_i: o método *Value()* retorna um valor inteiro que indicará qual esquema motor de uma construção foi selecionado como saída do comportamento;
c) va_Obstacles_r: o método *Value()* retorna uma lista de vetores contendo os obstáculos percebidos pelo robô;
d) va_Opponents_r: o método *Value()* retorna uma lista de vetores contendo os adversários percebidos pelo robô;
e) va_Teammates_r: o método *Value()* retorna uma lista de vetores contendo os parceiros percebidos pelo robô;
f) v_Attract_va: o método *Value()* retorna um vetor contendo a direção e a magnitude

do vetor atrator resultante;

g) v_Average_vv: o método *Value()* retorna um vetor com o ponto médio dos pontos passados como parâmetro;
h) v_Avoid_va: o método *Value()* retorna um vetor contendo a direção e a magnitude do vetor repulsor resultante;
i) v_GlobalPosition_r: o método *Value()* retorna vetor com a posição do robô em coordenadas globais;
j) v_LinearAttraction_v: o método *Value()* retorna um vetor de atração que aponta para o objetivo da atração;
k) v_OurGoal_r: o método *Value()* retorna a posição central do objeto gol do próprio time;
l) v_StaticWeightedSum_va: o método *Value()* retorna um vetor resultante da soma dos esquemas motores que compõem a construção, levando em consideração os pesos estabelecidos em tempo de desenvolvimento;
m) v_Swirl_vv: o método *Value()* retorna um vetor que exerce um campo tangencial em relação ao ponto passado como parâmetro;
n) v_TheirGoal_r: o método *Value()* retorna a posição central do objeto gol do time adversário;
o) NodeVec2: nó de Vec2. utilizado para chamar o método *Value()* do Vec2 contido no objeto.

2.3.4.3 Execução

Uma vez tendo sido especificado, o sistema baseado no pacote Clay é chamado repetitivamente em tempo de execução, baseado na atual situação do robô. A configuração é todo o sistema de comportamento encapsulado em um único objeto. Por convenção, normalmente esse objeto é identificado como *configuration* (BALCH, 2000).

A cada ciclo de tempo, a configuração é ativada através da chamada do *configuration.Value()*.Este método implicitamente ativa essa chamada de qualquer nodo embutido nesse objeto e assim por diante como numa hierarquia *top-down*, dessa forma todos são inicializados (BALCH, 2000).

```
detect_obstacles = new va_Obstacles_r(abstract_robot);
avoid_obstacles  = new v_Avoid_va(2.0, 1.0, detect_obstacles);
```

Fonte: Adaptado de Balch (2000)

QUADRO 10 – Representação de encapsulação de nodos.

No Quadro 10, quando o nodo avoid_obstacles é gerado, causará a execução do método *Value()* da classe v_Avoid_va e em conseqüência executará o método *Value()* da classe va_Obstacles_r, porque v_Avoid_va esta sendo utilizado como parâmetro.

Maiores informações a respeito da API Clay poderá ser consultado em Balch (2000).

### 2.3.5 Classe Vec2

A classe Vec2 é utilizada para manipulação de vetores de duas dimensões e está contida na API Util.

A seguir é apresentada uma breve explicação da estrutura da classe Vec2 utilizada.

a) x: componente do plano cartesiano que identifica a abscissa. Deve receber valor apenas através do método setx;
b) y: componente do plano cartesiano que identifica a ordenada. Deve receber valor apenas através do método sety;
c) r: componente do plano polar que identifica a força, ou intensidade, ou magnitude, do vetor. Deve receber valor apenas através do método setr;
d) t: componente do plano polar que identifica a direção, ou sentido, do vetor. Deve receber valor apenas através do método sett;
e) add: método que adiciona o vetor que é passado como parâmetro, ao vetor que chama o método. O parâmetro recebido é do tipo Vec2;
f) sub: método que subtrai o vetor que é passado como parâmetro, do vetor que chama o método. O parâmetro recebido é do tipo Vec2;
g) toString: método que retorna uma expressão alfanumérica com as informações do plano cartesiano e polar do vetor;
h) setx: método que passa valor para o componente x. O parâmetro recebido é do tipo *double*;
i) sety: método que passa valor para o componente y. O parâmetro recebido é do tipo *double*;
j) setr: método que passa valor para o componente r. O parâmetro recebido é do tipo *double*;

k) sett: método que passa valor para o componente t. O parâmetro recebido é do tipo *double*.

Maiores informações sobre a API Util, a classe Vec2 e sua estrutura completa poderá ser consultado em Balch (2000).

# 3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo serão apresentados os requisitos do problema, a modelagem e descrição da implementação. Após, será apresentada a operacionalidade das implementações. Esta se dividirá em duas etapas: a primeira descreve um comportamento simples usado para facilitar o entendimento do protótipo, e a segunda um comportamento mais complexo, onde será implementada a formação de um time de robôs jogadores de futebol. Para a definição dessa formação será feita a combinação de comportamentos.

## 3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O objetivo do software é definir uma área de atuação para o agente robô jogador de futebol, com base nos quatro parâmetros passados. Esta área de atuação representa um campo de atração ao qual o agente robô jogador de futebol está sujeito. O robô deve permanecer dentro da área de atuação definida e sempre que o robô sair dessa área, será atraído para ela através da força de um vetor que representa a força de atração da área de atuação. Essa força apontará para um ponto, denominado ponto de atração, que deve estar dentro dos limites da área de atuação definida. Na Figura 13 é apresentada a definição da área de atuação através dos quatro parâmetros passados, na Figura 14 são identificados os limites da área de atuação.

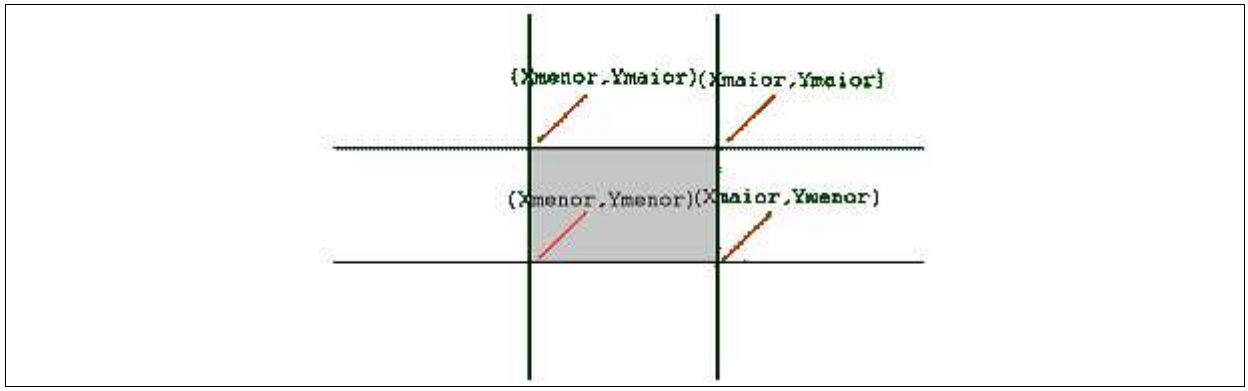



FIGURA 13 – Definição da área de atuação.

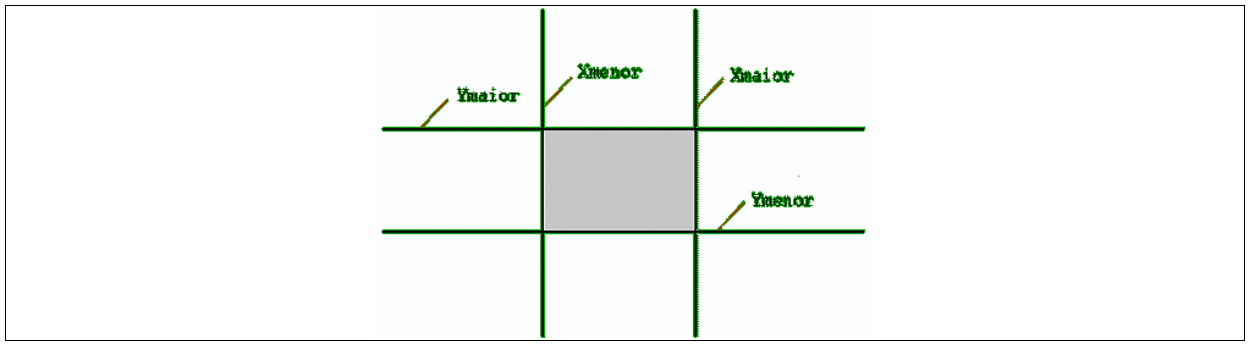



FIGURA 14 – Identificação dos limites da área de atuação.

Os requisitos que foram identificados para o desenvolvimento desse trabalho, são os seguintes:

a) criticar os quatro parâmetros passados que formam dois pontos cartesianos que representam o ponto inicial e o ponto final da área de atração dos jogadores, identificados na Figura 15 pelos pontos A, B, C, D;
b) definir as quatro extremidades da área de atração a fim de facilitar a identificação do ponto de atração. As extremidades são definidas com base nos quatro parâmetros passados, identificados na Figura 15 pelo retângulo de cor preta;
c) identificar o ponto de atração, identificado na Figura 15 pelo ponto X;
d) calcular o vetor resultante, identificado na Figura 15 pelo vetor de cor azul, entre o vetor de posição do robô, identificado na Figura 15 pelo vetor de cor vermelha, e o vetor do ponto de atração, identificado na Figura 15 pelo vetor de cor amarela;
e) retornar o vetor da atração.

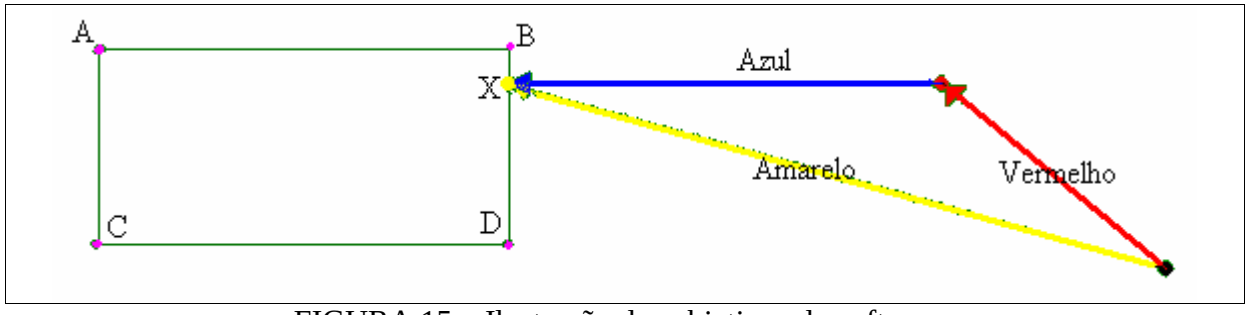



FIGURA 15 – Ilustração dos objetivos do software.

## 3.2 ESPECIFICAÇÃO E IMPLEMENTAÇÃO

Nesta seção será apresentada a especificação e a implementação do protótipo da classe desenvolvida. Inicialmente é apresentado fluxograma e o diagrama de seqüência. Posteriormente é apresentada e explicada a modelagem da classe desenvolvida. Nesta seção é

apresentada a utilização da classe como um comportamento de um agente jogador. O diagrama de classes das principais classes pode ser visto no Apêndice A.

### 3.2.1 Classe v_RectangularAttraction_v

Para poder determinar a área de atuação dos robôs jogadores, foi necessário desenvolver uma nova classe que será utilizada na definição dos comportamentos dos robôs. A classe desenvolvida trabalha com o conceito de campo potencial, mais especificamente como campo de atração visto na seção 2.2.1.

Seguindo os padrões de nomenclatura já existente para as classes, a nova classe recebeu o nome de v_RectangularAttraction_v. Os caracteres que iniciam o nome até o caractere "_" identifica o tipo do parâmetro principal que é passado para a classe; em seguida vem o nome que identifica o objetivo da implementação da classe, que se encontra limitado pelos caracteres "_". Finalmente, após o segundo caractere "_" até o final, se encontra o tipo de dado que a classe retornará.

### 3.2.2 Modelagem e implementação

No caso da classe v_RectangularAttraction_v tem-se como parâmetro de entrada e como retorno uma instância de Vec2, classe que é utilizada para representar vetores de duas dimensões que contém componentes cartesianos e polares, identificados pela letra "v". O nome que identifica o objetivo da implementação da classe é *RectangularAttraction* que é, traduzindo, atração retangular.

A Figura 16 apresenta a modelagem da classe v_RectangularAttraction_v, que é uma extensão de NodeVec2.

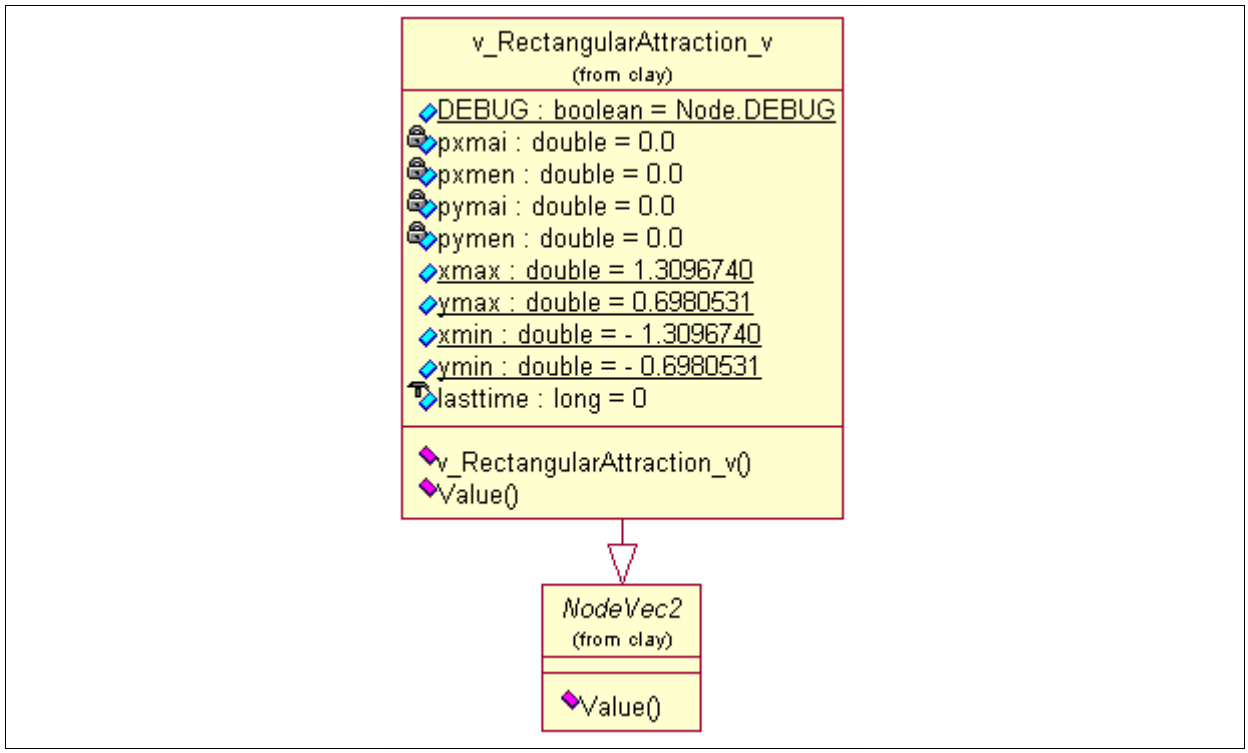



FIGURA 16 – Classe v_RectangularAttraction_v.

A classe v_RectangularAttraction_v tem o construtor mostrado no Quadro 12.

```
24    public v_RectangularAttraction_v(double ptoxini, double ptoyini,
25                                     double ptoxfim, double ptoyfim,
26                                     Vec2 im1)
```

QUADRO 12 – Sintaxe v_RectangularAttraction_v.

Os parâmetros deste construtor são os seguintes:

- ptoxini é um parâmetro tipo *double* que identifica a coordenada abscissa do ponto inicial da área de atração;
- ptoyini é um parâmetro tipo *double* que identifica a coordenada ordenada do ponto inicial da área de atração;
- ptoxfim é um parâmetro tipo *double* que identifica a coordenada abscissa do ponto final da área de atração;
- ptoyfim é um parâmetro tipo *double* que identifica a coordenada ordenada do ponto final da área de atração;
- im1 é um parâmetro tipo Vec2 que identifica o posicionamento do jogador dentro do campo de futebol.

Com os parâmetros passados é possível determinar a área retangular de atuação do jogador.

3.2.2.1 Validação de parâmetros

Quando é criado um objeto da classe v_RectangularAttraction_v, é feita uma consistência dos parâmetros das ordenadas e abscissas passados para verificar se estão dentro dos limites da extensão do campo. Para essa verificação foram definidos atributos tipo estáticos contendo os limites da extensão do campo, conforme mostrado no Quadro 13. Caso algum desses parâmetros estiver inválido, é enviada uma mensagem para o console avisando que existem parâmetros inválidos. Essa consistência é feita apenas para os parâmetros relativos as coordenadas.

```
20    public static final double xmax = 1.3096740;
21    public static final double ymax = 0.6980531;
22    public static final double xmin = -1.3096740;
23    public static final double ymin = -0.6980531;
```

QUADRO 13 – Atributos estáticos para consistência de parâmetros.

Efetuada a consistência, é feita uma normalização dos pontos, a fim de facilitar o cálculo do ponto de atração. Nesta normalização, é verificado qual é a menor e a maior abscissa e ordenada, e estas são passadas para variáveis que serão utilizadas no cálculo do ponto de atração. As variáveis são exibidas no Quadro 14.

```
16    private double      pxmai = 0.0;
17    private double      pxmen = 0.0;
18    private double      pymai = 0.0;
19    private double      pymen = 0.0;
```

QUADRO 14 – Variáveis utilizadas no cálculo do ponto de atração.

3.2.2.2 Funcionamento da classe v_RectangularAttraction_v

Cada vez que a classe é chamada através do método *Value*, primeiramente é verificado qual é o ponto para o qual a força irá atrair o robô jogador e em seguida é feito o cálculo do vetor resultante da atração.

Na Figura 17 é exibido o diagrama de seqüência que tem como objetivo demonstrar o funcionamento da simulação, focando a chamada da classe v_RectangularAttraction_v.

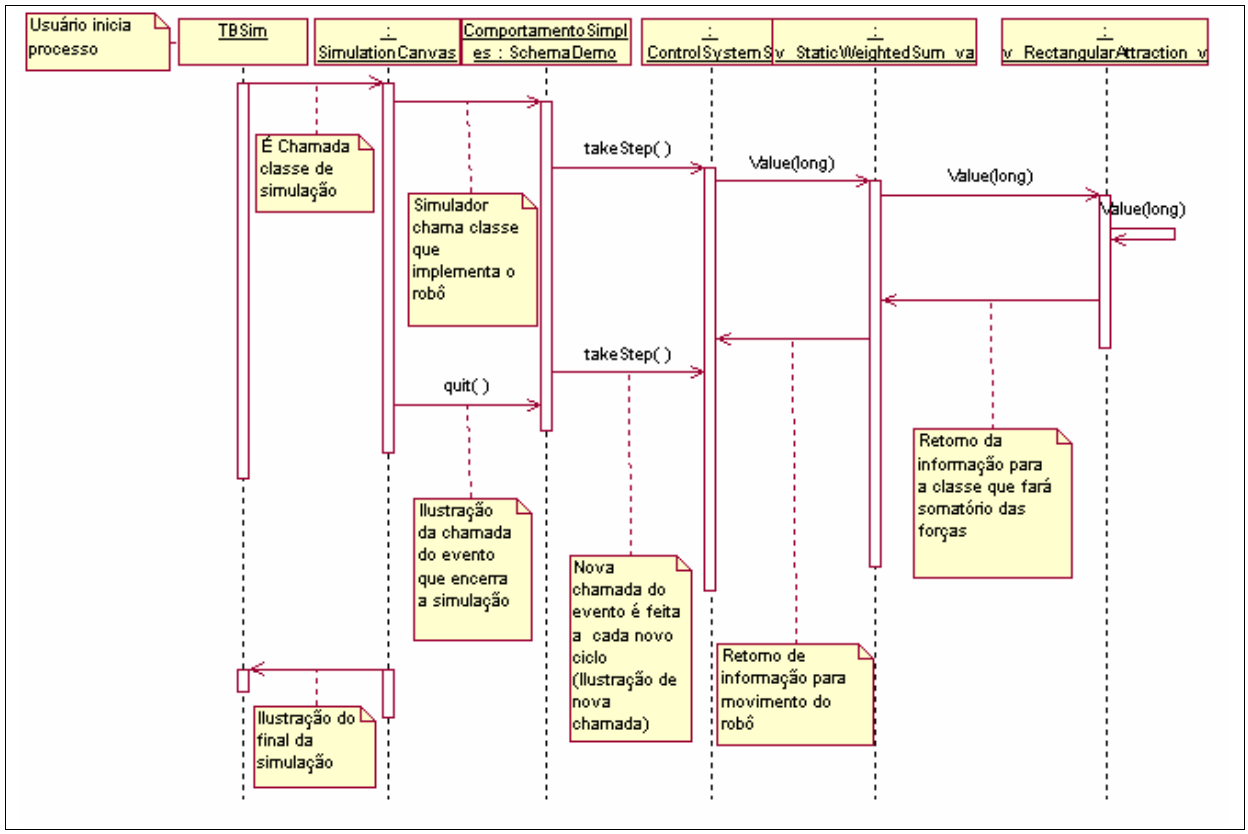



FIGURA 17 – Diagrama de seqüência da execução da chamada da classe.

Quando o usuário inicia a execução do simulador TBSim, a classe *SimulationCanvas* é criada para preparação do ambiente e criação dos robôs jogadores. A classe ComportamentoSimples tem o objeto *abstract_robot* que dispara o método *takeStep* do robô, da classe *ControlSystemS*. Esse método chama o método *Value* da construção definida. Ao executar o método *Value*, será feito o cálculo do vetor resultante da construção, com base nos pesos passados da definição da construção. Para obter o valor de cada vetor, chamará o método *Value* da classe v_RectangularAttraction_v. Conforme será explicado em 3.2.2.2.2, esse método retorna o vetor de atração resultante. A construção retornará o vetor que definirá que definirá para que direção e com que velocidade o robô se deslocará. O método *takeStep* será executado enquanto estiver ocorrendo a simulação do jogo.

### 3.2.2.2.1 Ponto de atração

Ponto de atração é o ponto para onde o vetor resultante deve atrair o agente robô jogador de futebol O processamento apresentado nessa sessão pertence ao método *Value* da classe v_RectangularAttraction_v.

Para definir o ponto de atração, o campo de futebol é dividido em nove regiões, conforme é exibido na Figura 18.

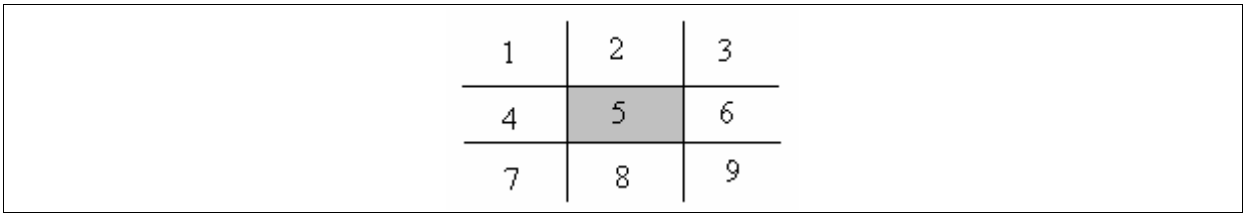



FIGURA 18 – Divisão do campo em relação à área definida como parâmetro.

Com base nessa divisão foram criadas as condições para estabelecer o ponto de atração, conforme segue:

a) se o robô se encontrar na região 1, que é menor que a abscissa menor e maior que a ordenada maior, então o ponto de atração será formado pela abscissa menor e pela ordenada maior;
b) se o robô se encontrar na região 2, que é maior que a abscissa menor e menor que a abscissa maior; e maior ou igual à ordenada maior, então o ponto de atração será formado pela abscissa do robô e pela ordenada maior;
c) se o robô se encontrar na região 3, que é maior que a abscissa maior e maior que a ordenada maior, então o ponto de atração será formado pela abscissa maior e pela ordenada maior;
d) se o robô se encontrar na região 4, que é menor ou igual à abscissa menor e maior que a ordenada menor e menor que a ordenada maior, então o ponto de atração será formado pela abscissa menor e pela ordenada do robô;
e) se o robô se encontrar na região 5, que é a área de atuação, então o ponto de atração será o ponto onde se encontra o robô;
f) se o robô se encontrar na região 6, que é maior que a abscissa maior e maior que a ordenada menor e menor que a ordenada maior, então o ponto de atração será formado pela abscissa maior e pela ordenada do robô;
g) se o robô se encontrar na região 7, que é menor que a abscissa menor e menor que a ordenada menor, então o ponto de atração será formado pela abscissa menor e pela ordenada menor;
h) se o robô se encontrar na região 8, que é maior que a abscissa menor e menor que a abscissa maior; e menor ou igual à ordenada maior, então o ponto de atração será formado pela abscissa do robô e pela ordenada menor;
i) se o robô se encontrar na região 9, que é maior que a abscissa maior e menor que a ordenada menor, então o ponto de atração será formado pela abscissa maior e pela

ordenada menor.

Inicialmente o algoritmo que definia o ponto de atração verificava se o agente robô jogador de futebol estava dentro de cada área para então definir o ponto de atração. Esse processamento prejudicava a performance da classe. Esta verificação foi modificada e passou-se a tratar a posição do agente robô jogador de futebol em relação às abscissas e às ordenadas. Como resultado, houve uma redução de nove para sete blocos de condições. Para definir se o robô estava dentro da área de atuação, destes sete blocos, seis tratam ou abscissas ou ordenadas e um trata ambas. Caso o robô estiver dentro da área de atuação definida, então as coordenadas do ponto de atração deverão ser iguais a zero. O quadro 16 é apresentado o algoritmo de verificação das coordenadas.

```
Se (Xmenor <= Xrobo >= Xmaior)  E  (Ymenor <= Yrobo >= Ymaior)
Entao Comeco
  Xponto = zero
  Yponto = zero
FIM
```

QUADRO 16 – Algoritmo que verifica se o robô está na área de atuação.

Se o robô estiver fora da área de atuação, então é efetuada uma análise separada para a abscissa e a ordenada do robô. Para essa análise foram utilizados os algoritmos apresentados nos Quadros 17 e 18 respectivamente.

```
Se (Xmenor > Xrobo < Xmaior)
Entao Xponto = Xmenor

Se (Xmenor < Xrobo < Xmaior)
Entao Xponto = Xrobo

Se (Xmenor < Xrobo > Xmaior)
Entao Xponto = Xmaior
```

QUADRO 17 – Algoritmo da abscissa.

```
Se (Ymenor > Yrobo < Ymaior)
Entao Yponto = Ymenor

Se (Ymenor < Yrobo < Ymaior)
Entao Yponto = Yrobo

Se (Ymenor < Yrobo > Ymaior)
Entao Yponto = Ymaior
```

QUADRO 18 – Algoritmo da ordenada.

No Quadro 19, é apresentado o código da implementação do cálculo do ponto de atração, que foi reduzido, conforme explicado anteriormente, para simplificar o processamento.

```
108             if(!( ((pxmen <= goal.x) && (pxmai >= goal.x)) &&
109               ((pymen <= goal.y) && (pymai >= goal.y)) ))
110             {
111                 /* regiao Esquerdo */
112                 if ((pxmen < goal.x) && (pxmai < goal.x))
113                     last_val.setx(-pxmai);
114                 /* regiao Direito */
115                 else if ((pxmen > goal.x) && (pxmai > goal.x))
116                     last_val.setx(-pxmen);
117                 /* regiao Centro */
118                 else if ((pxmen <= goal.x) && (pxmai >= goal.x))
119                     last_val.setx(-goal.x);
120 
121                 /* regiao Alto */
122                 if ((pymen < goal.y) && (pymai < goal.y))
123                     last_val.sety(-pymai);
124                 /* regiao Baixo */
125                 if ((pymen > goal.y) && (pymai > goal.y))
126                     last_val.sety(-pymen);
127                 /* regiao Centro */
128                 else if ((pymen <= goal.y) && (pymai >= goal.y))
129                     last_val.sety(-goal.y);
```

QUADRO 19 – Programação para definir ponto de atração.

3.2.2.2.2 Vetor resultante da atração

Com os vetores que indicam o ponto de atração e o posicionamento do robô, é feito o cálculo do vetor resultante, que representa a diferença entre o posicionamento do robô e o ponto de atração, e informa o sentido e força que levará o robô à área passada como parâmetro.

A Figura 19 a representa graficamente essa situação. A área retangular em preto representa a área de atuação definida através dos parâmetros. O ponto amarelo é o ponto de atração calculado e a linha amarela representa o vetor do ponto de atração. O ponto vermelho representa o robô e sua localização e a linha vermelha representa o vetor do posicionamento do robô. A linha azul representa o vetor resultante da subtração entre o vetor do ponto de atração e o vetor do posicionamento do robô.

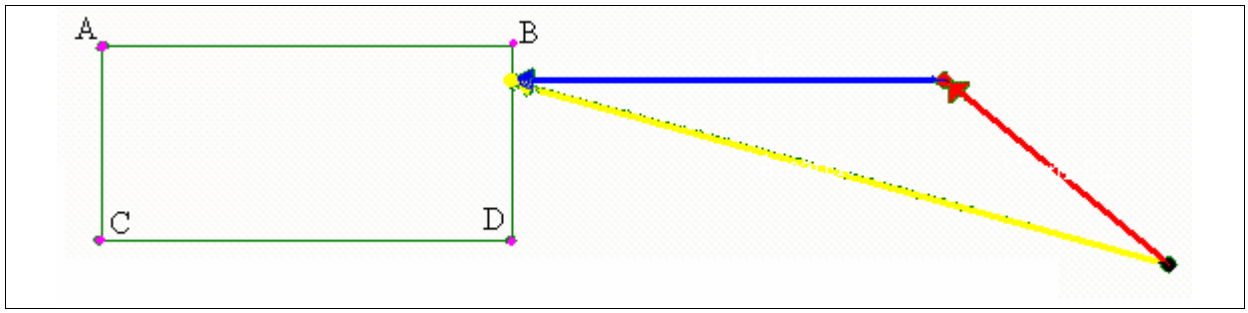



FIGURA 19 – Ilustração da área retangular de atração, vetor posicionamento do robô, vetor ponto de atração e vetor resultante.

### 3.2.3 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Após a especificação da classe v_RectangularAttraction_v, serão apresentados os comportamentos criados para formar um time de robôs jogadores de futebol utilizando a robótica baseada em comportamento.

Inicialmente será apresentado um comportamento simples que envolve somente os jogadores de um time com dois comportamentos sendo englobados em uma construção. Posteriormente, será apresentado um comportamento mais complexo, envolvendo o time de robôs jogadores de futebol com a formação desenvolvida, sendo construída com mais de um comportamento e mais de uma construção, e no outro time de robôs jogadores de futebol com uma formação já existente na API Clay.

#### 3.2.3.1 Comportamento simples

Nessa sessão será apresentada, explicada e demonstrada a classe do agente robô jogador de futebol implementado.

Na Figura 20, é mostrado o digrama da classe ComportamentoSimples que implementa o agente jogador de futebol. A classe é estendida da classe ControlSystemSS, como pode ser observado na definição da classe.

Foram implementados dois métodos, *configure()* e *takeStep()*, que serão explicados em seguida.

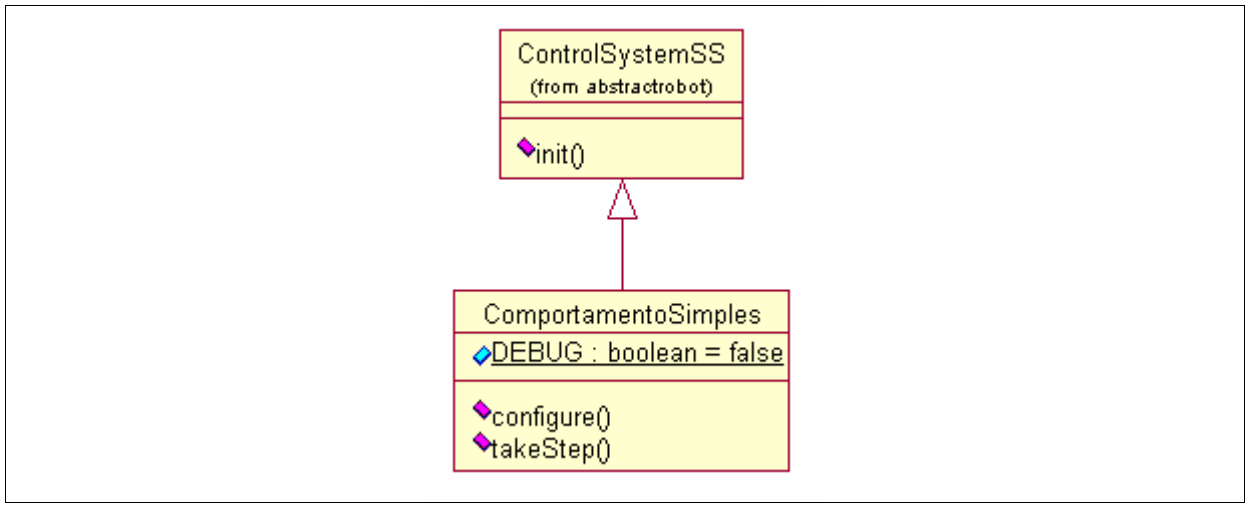



FIGURA 20 – Modelagem da classe ComportamentoSimples, com representação da classe estendida ControlSystemSS.

No Quadro 21 é exibida a implementação do método *configure()*. Nesse método é definido um esquema motor e dois esquemas perceptivos.

O esquema perceptivo é identificado por PS_TEAMMATES, responsável por informar quais parceiros de time são percebidos pelo robô. O esquema motor é identificado por MS_AVOID_TEAMMATES, que tem como objetivo evitar que os robôs jogadores de um mesmo time fiquem muito próximos um dos outros. Em seguida, é feita a inicialização da variável, ou objeto, que armazenará a movimentação que será feita pelo robô. A última instrução desse método é outro esquema perceptivo que retorna se o robô pode chutar a bola.

```
16      public void configure()
17          {
18
19          if (DEBUG) System.out.println("ComportamentoSimples.Configure()");
20          abstract_robot.setObstacleMaxRange(3.0);
21
22          //======
23          // Esquema perceptivo
24          //======
25          // Posição dos parceiros de time percebidos pelo robô
26          NodeVec2Array
27          PS_TEAMMATES = new va_Teammates_r(abstract_robot);
28
29          //======
30          // Esquema motor
31          //======
32          // Esquema motor para evitar parceiros
33          MS_AVOID_TEAMMATES
34              = new v_Avoid_va(1.0, abstract_robot.RADIUS+0.1,
35                  PS_TEAMMATES);
36
37          // Inicialização da variável que aponta comportamento
38          steering_configuration = null;
39
40          // Retorna se o robô pode chutar a bola
41          kick_configuration = new b_CanKick_r(abstract_robot);
42          }
```

QUADRO 21 – Código da implementação do método *configure()*.

No método *takeStep()* é feito o processamento da ação que será feita pelo agente robô jogador de futebol a cada passo da simulação. Nela está contida a construção do comportamento que utiliza dois esquemas motores; o de repulsão de agentes robôs jogadores de futebol parceiros e o de atração da área retangular de atuação do agente robô jogador de futebol.

No Quadro 22 é exibida a primeira parte da implementação do método *takeStep*() onde é armazenada na variável **n_robo** o número que identifica cada agente robô jogador de futebol. Essa informação será utilizada para definir o comportamento de cada agente já que todos foram implementados com a mesma classe.

Em seguida pode ser identificado o esquema perceptivo responsável por fornecer o posicionamento do agente robô jogador de futebol. Essa informação será utilizada como parâmetro na utilização da classe v_RectangularAttraction_v.

```
45    public int takeStep()
46        {
47        Vec2    result;
48        boolean kick_result;
49
50        long curr_time = abstract_robot.getTime();
51        long n_robo = abstract_robot.getPlayerNumber(curr_time);
52
53        Vec2    interseccao = new Vec2();
54        interseccao = abstract_robot.getPosition(curr_time);
55        System.out.println("************************************************************");
56        System.out.println("Posicao:" + interseccao.toString());
57
58
59        StringBuffer sb = new StringBuffer();
60        System.out.println(sb.append("Robo Num....: ").append(n_robo));
61
62        // Posicionamento do robô dentro do campo
63        Vec2    PS_GLOBAL_POS = new Vec2(0,0);
64        PS_GLOBAL_POS.sub(abstract_robot.getPosition(curr_time));
65
```

QUADRO 22 – Código da implementação do método *takeStep()*, parte 1.

Na segunda parte da implementação, exibida parcialmente no Quadro 23, demonstra-se a construção e seleção do comportamento do agente robô jogador de futebol.

Primeiro é definido o esquema motor identificado por MS_MOVE_TO_AREA0, respectivo ao agente robô jogador de futebol identificado pelo número 0, que irá atrair o jogador para a área passada como parâmetro.

Em seguida, a construção do comportamento, identificado por AS_COMPORTAMENTOSIMPLES0, é feita através do esquema motor MS_AVOID_TEAMMATES e MS_MOVE_TO_AREA0. Nesse momento, cada comportamento que irá compor a construção recebe um peso e outro comportamento, no caso

os esquemas motores definidos. O vetor resultante dessa construção de comportamento será passada para uma variável identificada por steering_configuration.

```
67          if (n_robo == 0) // área E
68          {
69              // Esquema motor para área retangular
70              NodeVec2
71              MS_MOVE_TO_RAREA0
72                  = new v_RectangularAttraction_v(+1.3, +0.001, +1.27, -0.001, PS_GLOBAL_POS);
73
74              //======
75              // CONSTRUÇÃO
76              //======
77              // Esquema motor para evitar parceiros
78              v_StaticWeightedSum_va AS_COMPORTAMENTOSIMPLES0 =
79                  new v_StaticWeightedSum_va();
80              AS_COMPORTAMENTOSIMPLES0.weights[0]  = 0.8;
81              AS_COMPORTAMENTOSIMPLES0.embedded[0] = MS_AVOID_TEAMMATES;
82
83              AS_COMPORTAMENTOSIMPLES0.weights[1]  = 1.2;
84              AS_COMPORTAMENTOSIMPLES0.embedded[1] = MS_MOVE_TO_RAREA0;
85              steering_configuration = AS_COMPORTAMENTOSIMPLES0;
86          }
87          else if (n_robo == 2) // área A
```

QUADRO 23 – Código da implementação do método *takeStep()*, parte 2.

Na Figura 21 é apresentada uma ilustração do que ocorreu no Quadro 23.

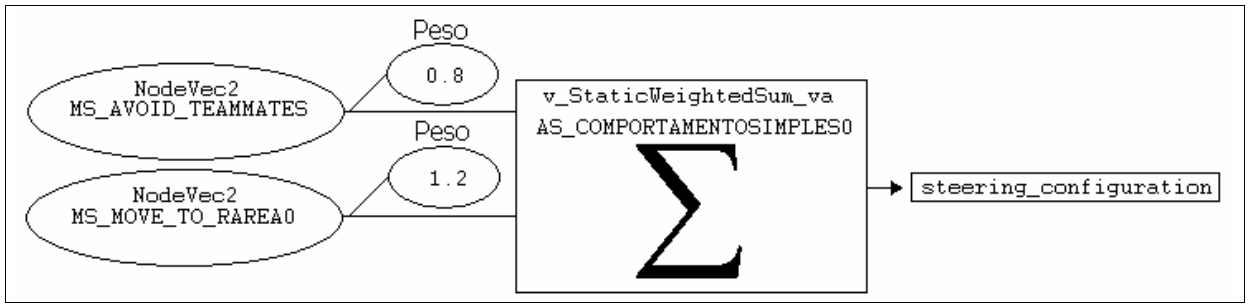



FIGURA 21 – Ilustração de como é gerada a ação do agente robô jogador de futebol.

Se o agente robô jogador de futebol pudesse perceber todos os seus parceiros, então as forças que atuariam sobre ele seriam as identificadas na Figura 22. As linhas vermelhas identificam forças de repulsão geradas pelo comportamento MS_AVOID_TEAMMATES; e a linha verde identificaria a força de atração gerada pelo comportamento MS_MOVE_TO_AREA0, que define a área de atuação identificada pela cor amarela.

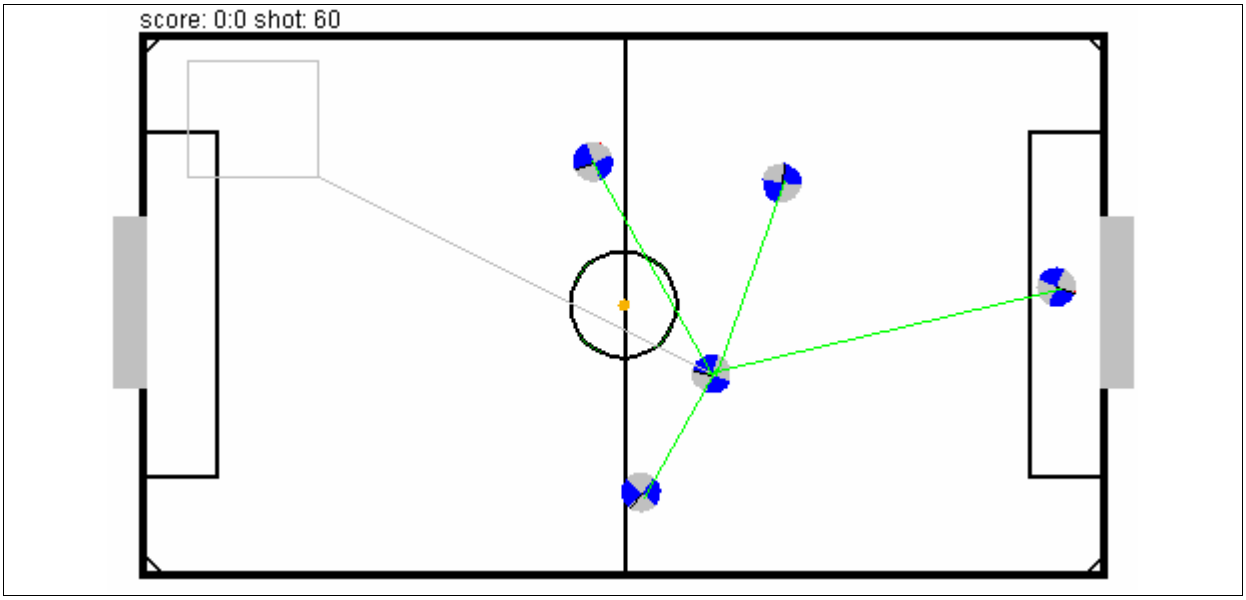



FIGURA 22 – Ilustração de forças incidentes no agente robô jogador de futebol número 3.

No Quadro 24 esta apresentada a parte final da implementação. Após conhecer-se o comportamento que gera a ação do agente robô jogador de futebol, é verificada a velocidade com que o robô irá se movimentar não é maior que 1. Se for, é alterada para 1.

Em seguida é passado ao robô direção e velocidade que deve se deslocar. Caso o robô possa chutar a bola, é passada a instrução para que ele chute-a.

Finalizando a chamada do método, é definido o valor para o retorno do método.

```
177         // Passagem de comportamento/ação para o robô
178         result = steering_configuration.Value(curr_time);
179
180         if (result.r > 1.0)
181             result.setr(1.0);
182
183         kick_result = kick_configuration.Value(curr_time);
184
185         // passa informações ao robô
186         abstract_robot.setSteerHeading(curr_time, result.t);
187         abstract_robot.setSpeed(curr_time, result.r);
188
189         // Caso robô estiver em condições de chutar (alcance da bola), chuta
190         if (kick_result)
191             abstract_robot.kick(curr_time);
192
193         // Retorna que robô está OK
194         return(CSSTAT_OK);
195         }
196     }
```

QUADRO 24 – Código da implementação do método *takeStep()*, parte 3.

Na Figura 23 são exibidos quadros da execução do comportamento simples. No primeiro quadro é exibido o objetivo de cada um dos agentes, que deve ser obtido sem que os robôs se esbarrem. Nos demais quadros são exibidos os estados até o objetivo.

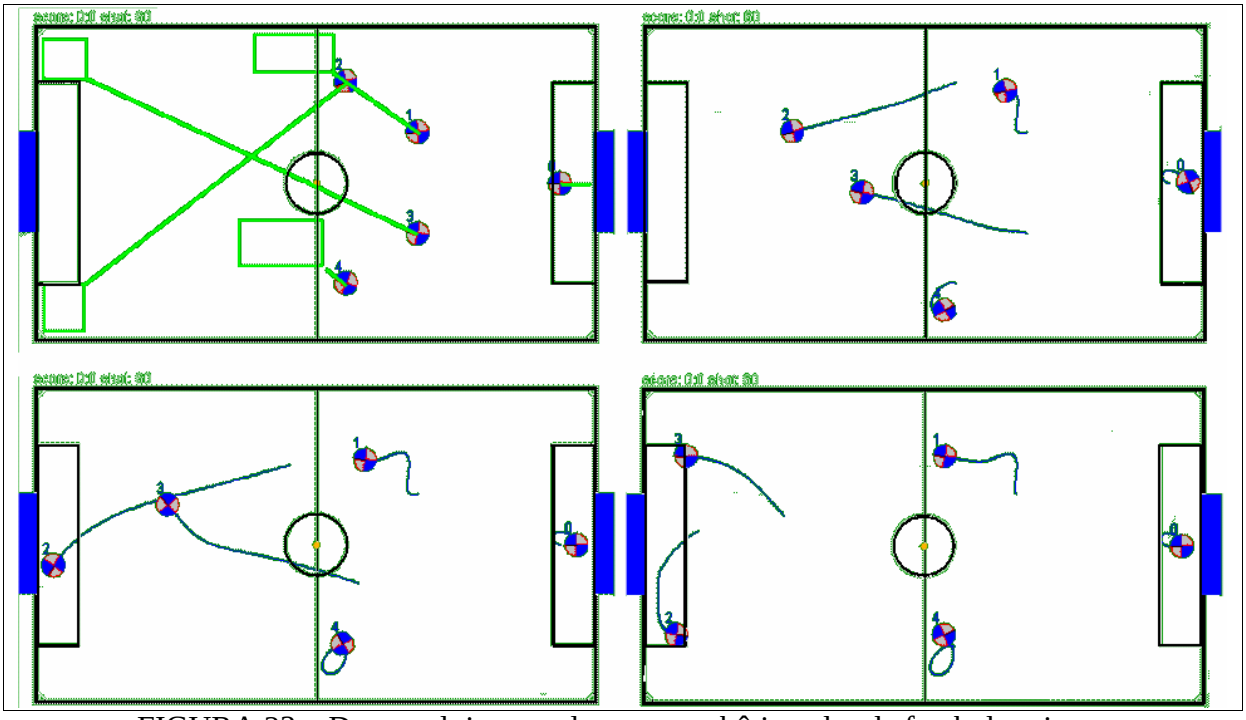


FIGURA 23 – Desenvolvimento do agente robô jogador de futebol no jogo.

3.2.3.2 Comportamento composto

Nessa seção será apresentada a segunda classe do agente robô jogador de futebol implementado. Essa classe foi desenvolvida com base na classe SchemaDemo.java para exemplificar a utilização da classe v_RectangularAttraction_v em um time. Os nomes que se encontram em inglês são de comportamentos existentes na classe referência.

Na Figura 24, é mostrado o digrama da classe ComportamentoComposto que implementa o agente jogador de futebol. A classe é estendida, assim como a classe ComportamentoSimples, da classe ControlSystemSS.

Foram implementados dois métodos, *configure()* e *takeStep()*, que serão explicados em seguida os pontos que diferenciam da classe ComportoSimples.

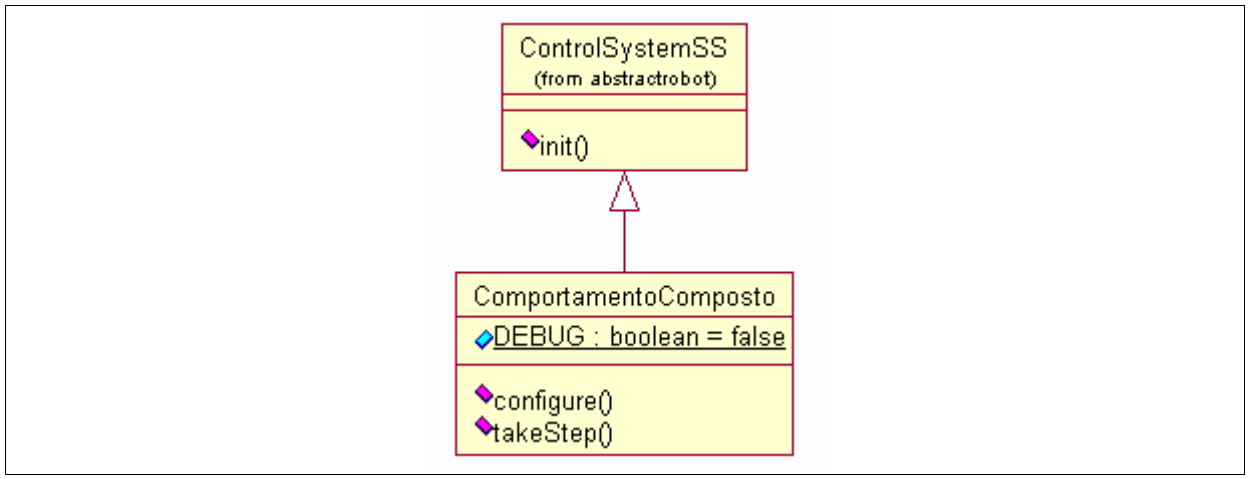



FIGURA 24 – Modelagem da classe ComportamentoComposto, com representação da classe estendida ControlSystemSS.

São relacionados a seguir os esquemas perceptivos que a classe ComportamentoComposto utiliza além dos já utilizados na classe ComportamentoSimples no método *configure()*.

Os esquemas perceptivos são:

a) PS_BALL: percebe onde se encontra a bola;

b) PS_SWEET_SPOT: percebe o ponto entre o gol adversário e a bola. Uma ilustração desse esquema pode ser vista na Figura 25. O ponto escuro próximo ao centro do campo seria o local adequado que o robô estivesse para chutar a bola em direção ao gol adversário;

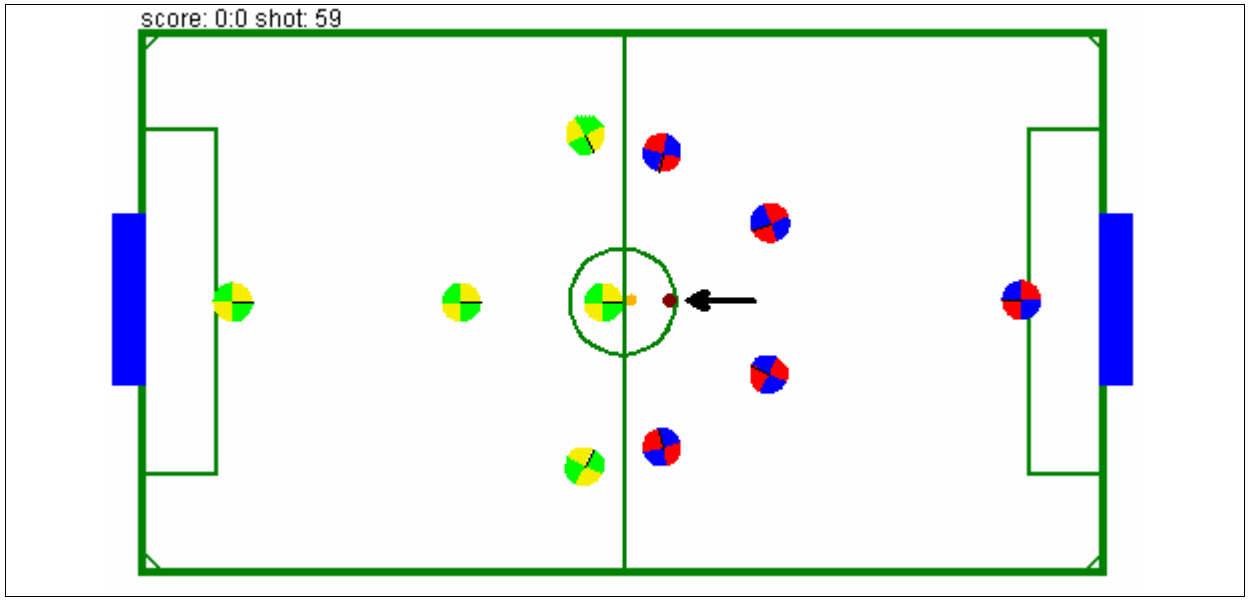



FIGURA 25 – Ilustração da percepção de PS_SWEET_SPOT.

c) PS_OUR_GOAL: percebe onde se localiza o gol do próprio time;

d) PS_THEIR_GOAL: percebe onde se localiza o gol do adversário;
e) PS_HALFWAY: percebe ponto entre o próprio gol e a bola. Uma ilustração desse esquema pode ser vista na Figura 26. O ponto escuro próximo a metade do campo do time azul/vermelho é exemplo do ponto determinado pela percepção PS_HALFWAY.

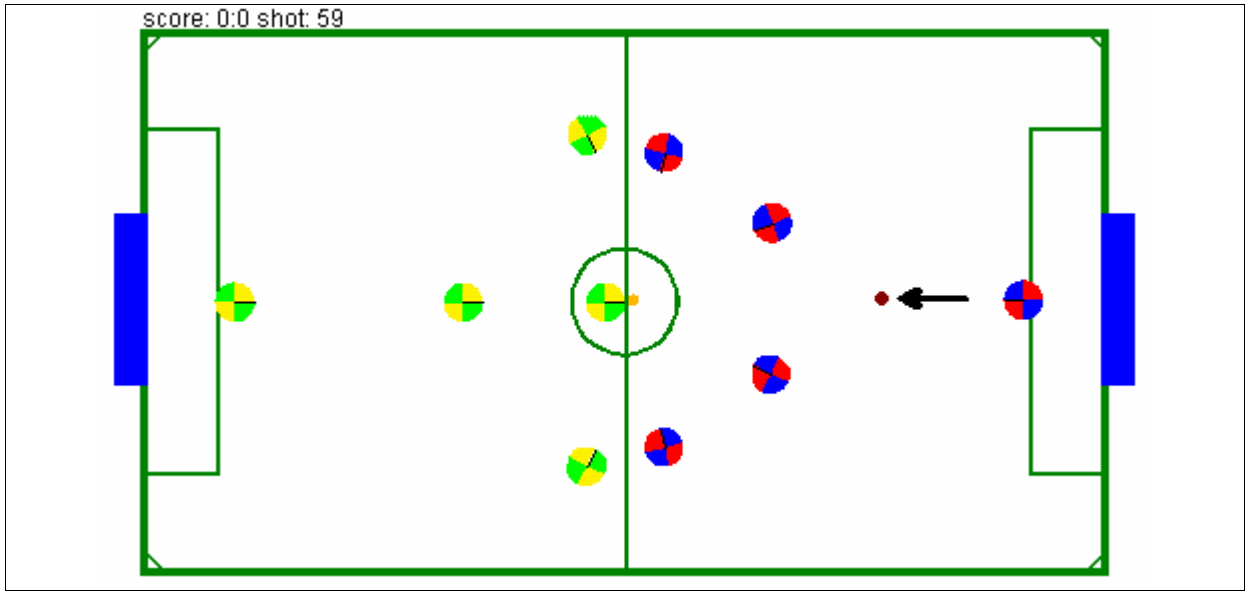



FIGURA 26 – Ilustração da percepção de PS_HALFWAY.

Esses esquemas perceptivos são utilizados para compor os seguintes esquemas motores:

a) MS_MOVE_TO_SWEET_SPOT: cria uma força que atrai o robô para local apontado pelo esquema perceptivo OS_SWEET_SPOT;
b) MS_MOVE_TO_HALFWAY: cria uma força que atrai o robô para o ponto mediano entre a bola e o gol próprio. O robô irá tangenciando em direção a esse ponto a fim de interceptar a bola;
c) MS_SWIRL_BALL: cria uma força que atrai o robô para o ponto entre os locais percebidos por PS_BALL e PS_HALFWAY. Essa força fará o robô se movimentar tangencialmente em direção ao ponto de atração. Esse tangenciamento é feito com a finalidade principal de interceptar a bola.

Como o time possui cinco agentes robôs jogadores de futebol, fora criadas as áreas de atuação de cada agente. Como pode ser observado na Figura 27, existem áreas com regiões em comum entre os agentes. Essas áreas em comum servem para que os agentes interajam na busca do objetivo, facilitando a cobertura de todo campo. A formação implementada no

agente da classe ComportamentoComposto visa um time que atua mais no ataque do que na defesa.

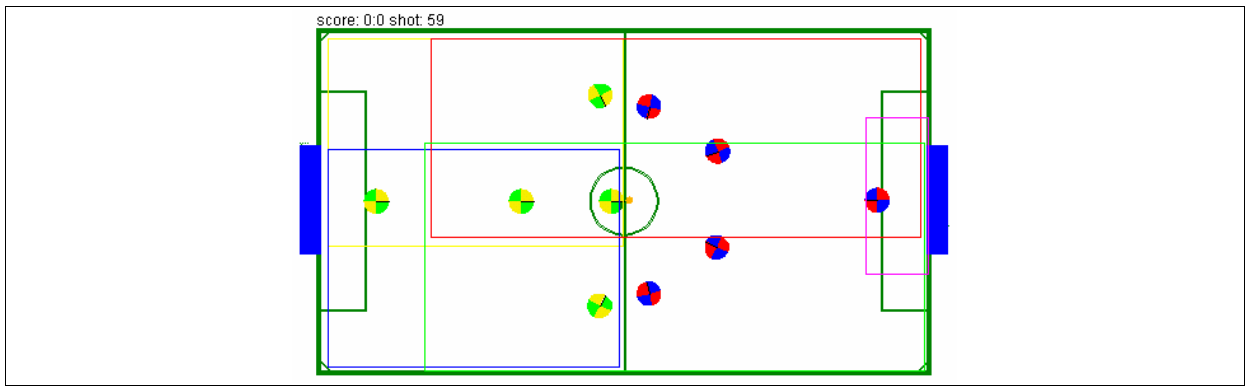



FIGURA 27 – Demonstração das atrações retangulares existentes na classe ComportamentoComposto.

A Figura 28 mostra a área de atuação lateral direita. A área de atuação está destacada em tom cinza.

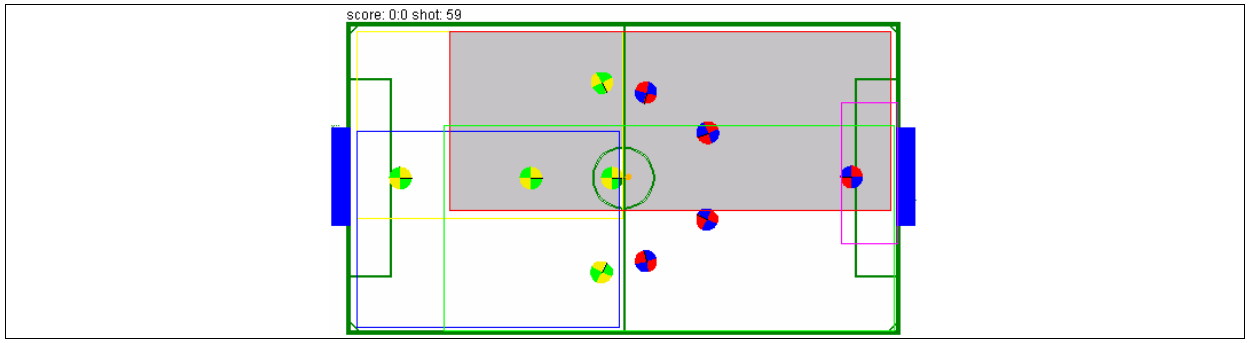



FIGURA 28 – Atração retangular lateral direita existente na classe ComportamentoComposto.

A Figura 29 mostra a área de atuação lateral esquerda. A área de atuação está destacada em tom cinza.

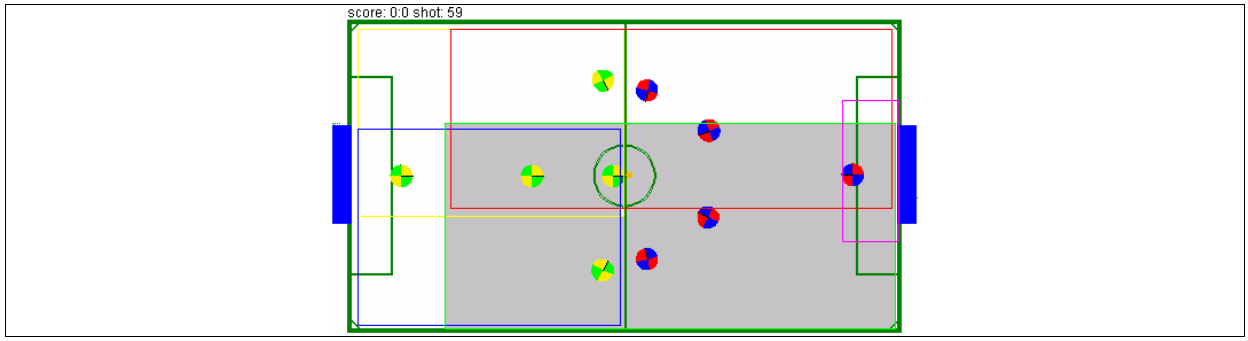



FIGURA 29 – Atração retangular lateral esquerda existente na classe ComportamentoComposto.

A Figura 30 mostra a área de atuação ataque direita. A área de atuação está destacada em tom cinza.

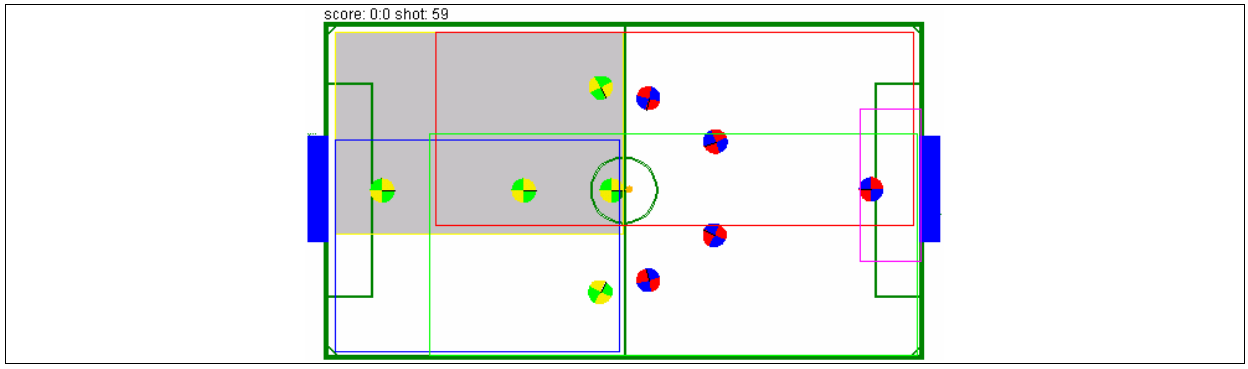



FIGURA 30 – Atração retangular ataque direito existente na classe ComportamentoComposto.

A Figura 31 mostra a área de atuação lateral esquerda. A área de atuação está destacada em tom cinza.

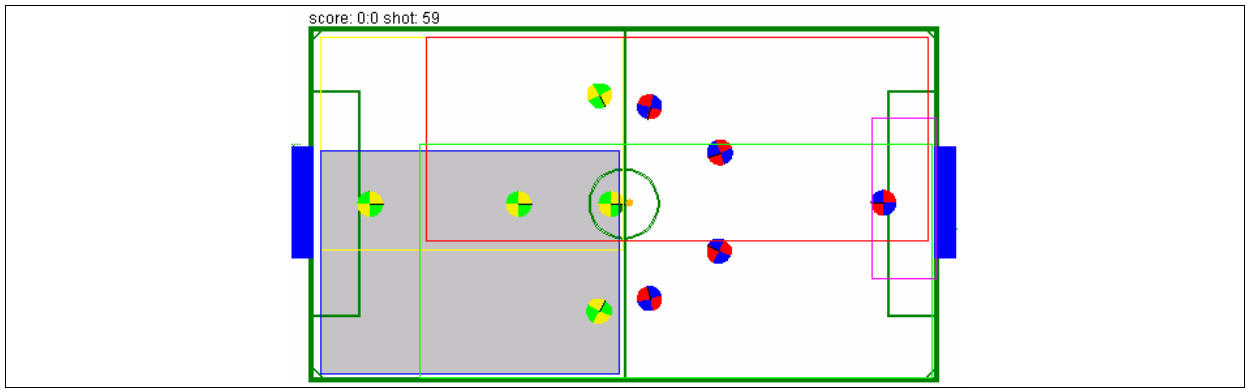



FIGURA 31 – Atração retangular ataque esquerdo existente na classe ComportamentoComposto.

A Figura 28 mostra a área de atuação do goleiro. A área de atuação está destacada em tom cinza.

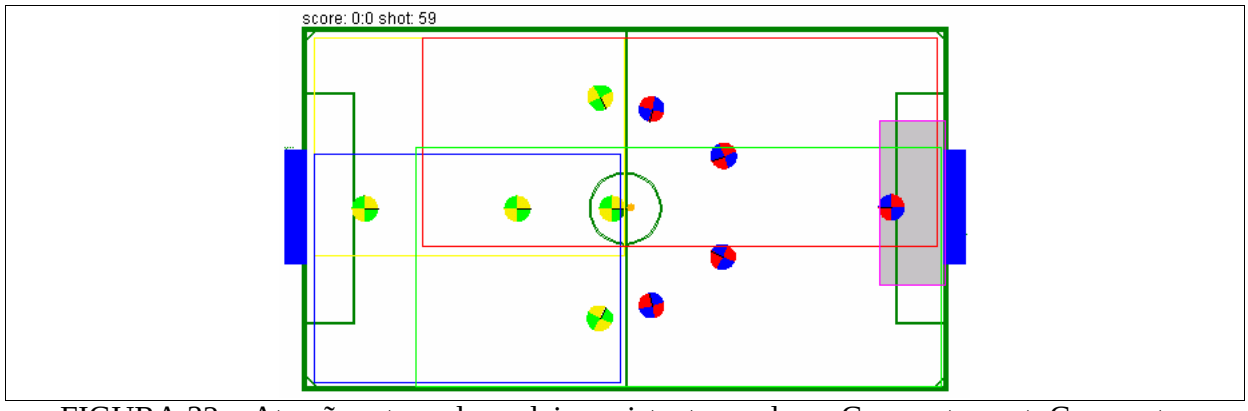



FIGURA 32 – Atração retangular goleiro existente na classe ComportamentoComposto.

A fim de tornar a configuração da formação mais fácil, o comportamento dos agentes jogadores, com exceção do agente jogador goleiro; são parecidos mudando apenas a área de atuação. A área de atuação é definida pela classe v_RectangularAttraction_v.

O agente jogador goleiro tem o comportamento resultante a seguinte construção de comportamentos simples:

a) área de atuação (MS_MOVE_TO_RAREA0): a área definida compreende a região do gol conforme demonstrado anteriormente na Figura 32;
b) MS_MOVE_TO_BALL: comportamento que atrai o robô até a bola, afim de afastar a bola do gol;
c) MS_MOVE_TO_SWEET_SPOT: comportamento que faz o robô se posicionar de forma que, quando chutar a bola, chute em direção ao gol adversário.

A Figura 33 ilustra o comportamento acima e o Quadro 25 exibe o código da implementação do comportamento.

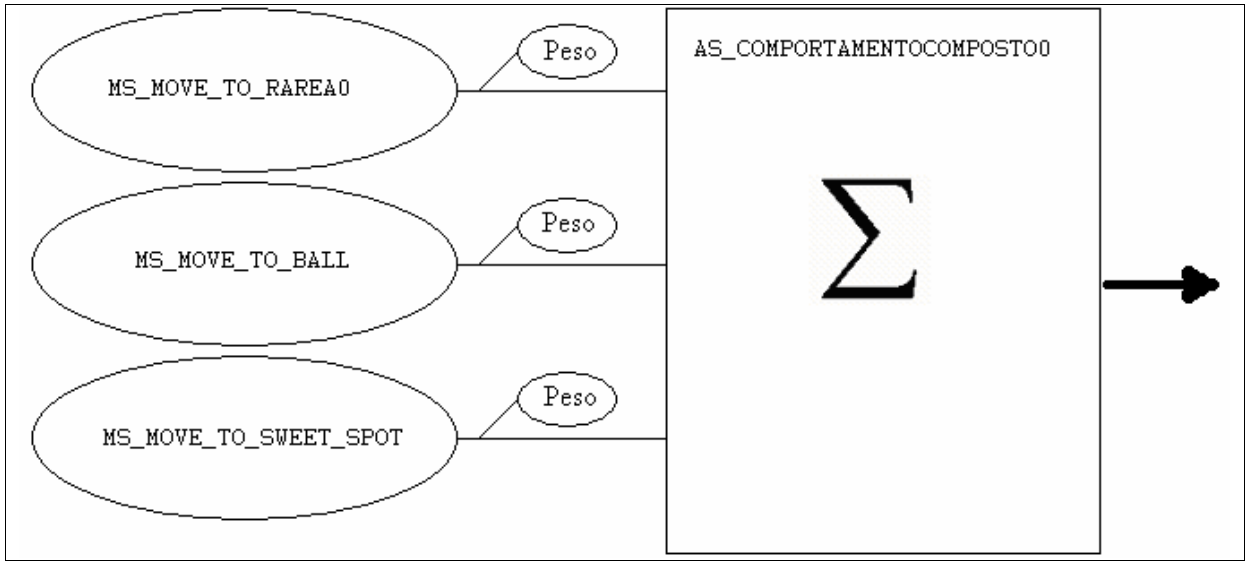



FIGURA 33 – Atração retangular goleiro.

```
133             v_StaticWeightedSum_va AS_COMPORTAMENTOCOMPOSTO0 =
134                 new v_StaticWeightedSum_va();
135             AS_COMPORTAMENTOCOMPOSTO0.weights[0]  = 7.2;
136             AS_COMPORTAMENTOCOMPOSTO0.embedded[0] = MS_MOVE_TO_RAREA0;
137
138             AS_COMPORTAMENTOCOMPOSTO0.weights[1]  = 0.9;
139             AS_COMPORTAMENTOCOMPOSTO0.embedded[1] = MS_MOVE_TO_BALL;
140
141             AS_COMPORTAMENTOCOMPOSTO0.weights[2]  = 0.7;
142             AS_COMPORTAMENTOCOMPOSTO0.embedded[2] = MS_MOVE_TO_SWEET_SPOT;
```

QUADRO 25 – Código da implementação do comportamento do agente jogador goleiro.

Os demais agentes jogadores têm como comportamento resultante o retorno da seguinte construção:

a) MS_AVOID_TEAMMATES: comportamento que faz o robô não se aproxime muito de seus parceiros. Esse comportamento evita que um agente atrapalhe o outro;
b) MS_MOVE_TO_SWEET_SPOT: comportamento que faz o robô se posicionar de forma que, quando chutar a bola, chute em direção ao gol adversário;
c) MS_MOVE_TO_BALL: comportamento que atrai o robô até a bola;
d) área de atuação (MS_MOVE_TO_RAREAn): para n igual a 1 até 4, cada comportamento define um agente cujas áreas foram ilustradas nas Figuras 28 a 31;

A Figura 34 ilustra o comportamento acima e o Quadro 26 exibe o código da implementação do comportamento.

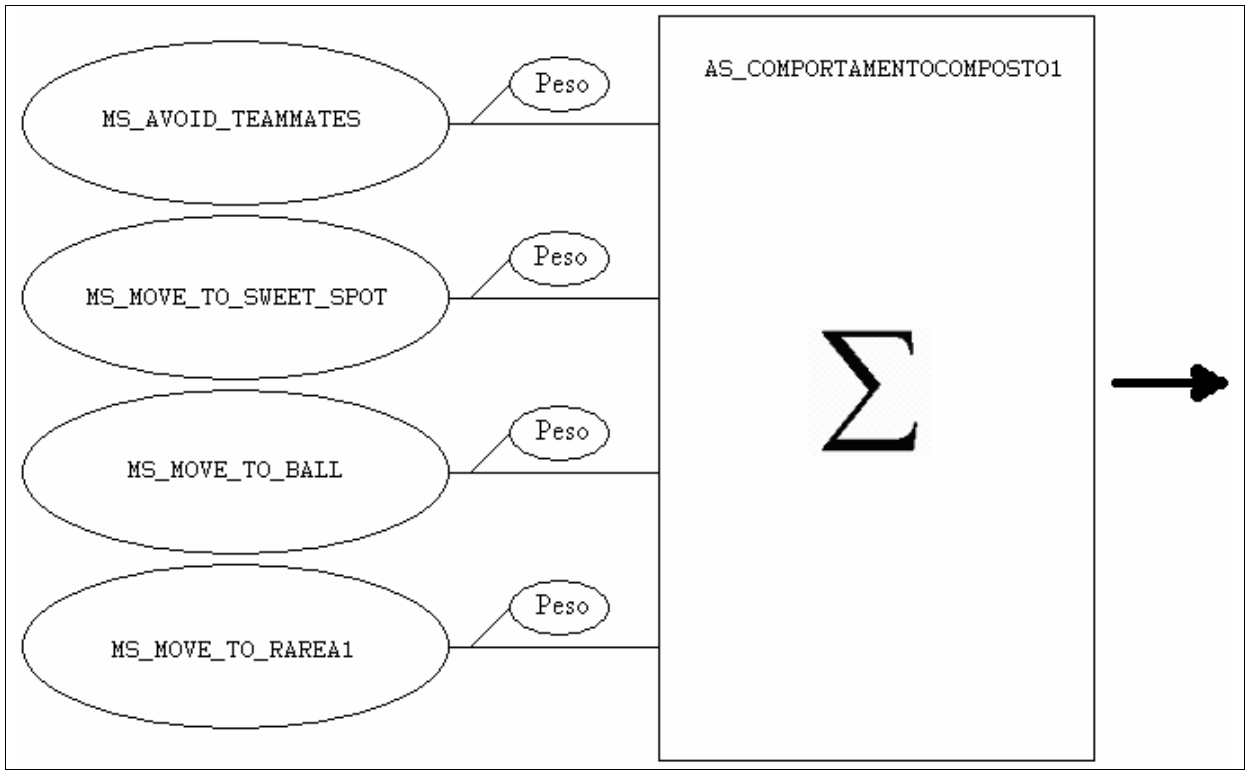


FIGURA 34 – Formatação da atração retangular dos demais jogadores.

```
157             v_StaticWeightedSum_va AS_COMPORTAMENTOCOMPOSTO1 =
158                 new v_StaticWeightedSum_va();
159             AS_COMPORTAMENTOCOMPOSTO1.weights[0]  = 0.1;
160             AS_COMPORTAMENTOCOMPOSTO1.embedded[0] = MS_AVOID_TEAMMATES;
161
162             AS_COMPORTAMENTOCOMPOSTO1.weights[1]  = 1.0;
163             AS_COMPORTAMENTOCOMPOSTO1.embedded[1] = MS_MOVE_TO_SWEET_SPOT;
164
165             AS_COMPORTAMENTOCOMPOSTO1.weights[2]  = 1.0;
166             AS_COMPORTAMENTOCOMPOSTO1.embedded[2] = MS_MOVE_TO_BALL;
167
168             AS_COMPORTAMENTOCOMPOSTO1.weights[3]  = 5.0;
169             AS_COMPORTAMENTOCOMPOSTO1.embedded[3] = MS_MOVE_TO_RAREA1;
```
QUADRO 26 – Código da implementação do comportamento dos demais agentes jogadores.

A classe ComportamentoComposto foi desenvolvida dessa forma com a finalidade de tornar os ajustes na formação mais simples, necessitando alterar apenas os pesos e as definições das áreas de atuação.

Alterando essas duas informações na classe já causa uma mudança no comportamento dos agentes robôs jogadores de futebol.

## 3.3 RESULTADOS E DISCUSSÃO DE TÉCNICAS E FERRAMENTAS UTILIZADAS

Para o desenvolvimento da classe e do agente jogador de futebol foi utilizada a linguagem Java através do Java 2 SDK, Standard Edition Versão 1.4.1; e o pacote TeamBots. O pacote TeamBots foi utilizado para simulação e para especificação dos comportamentos, da classe e dos agentes. Para especificação da classe e diagrama se seqüência e Rational Rose Enterprise Edition versão 2002.05.00.

Para efetuar-se a construção da classe e do agente jogador de futebol foi utilizado o JCreator, versão 2.0; o qual atendeu as necessidades para o desenvolvimento do protótipo. Apesar da simplicidade, o editor JCreator ofereceu uma boa plataforma para desenvolver o trabalho, sem exigir muito do processador.

O TeamBots se mostrou um ótimo ambiente para desenvolvimento de comportamentos assim como de agentes jogadores de futebol. O simulador TBSim, exige muito do processador, sendo necessário em alguns momentos dar maior prioridade para a execução dele. Contudo foi muito eficiente para testar o protótipo. TeamBots disponibiliza uma variedade de classes que fornecem as informações necessárias para a criação de novos comportamentos e desenvolvimento de agentes jogadores de futebol de robôs.

A classe v_RectangularAttraction_v facilitou o desenvolvimento de comportamentos para robôs jogadores de futebol por possibilitar definir comportamento para todo o campo, já que a área do campo é retangular assim como a área que a classe implementou a atração. Na API Clay existe a classe v_LinearAttraction_v que implementa a atração em regiões circulares, não sendo eficiente para cobrir a área do campo, já que trabalha com regiões circulares dentro de uma região retangular.

O resultado obtido através da classe desenvolvida foi satisfatório, pois possibilitou atingir o objetivo proposto, porém não com a eficiência esperada. Os robôs não possuem a movimentação esperada, ocasionando uma demora no deslocamento dos jogadores. Isso

ocorre por ter sido efetuada a definição da força com base no cálculo da distância entre o robô e ponto objetivo. Acredita-se que efetuando uma implementação que modifique a força com base no posicionamento do robô em relação a área de atuação, o desempenho do robô melhore significativamente.

A classe foi desenvolvida para um time que joga no lado leste do campo. Com isso, não é garantido o correto funcionamento em robôs jogadores de times do lado oeste.

### 3.3.1 PROBLEMAS E DIFICULDADES

Durante o desenvolvimento desse protótipo, foram encontradas dificuldades na implementação em função do desconhecimento da linguagem Java por parte do autor. Porém, através de mini-cursos e livros foi possível adquirir a base necessária para possibilitar o desenvolvimento do protótipo.

Outros dois fatores não menos importantes foram a ausência documentação referente ao pacote TeamBots e ausência de recursos para a depuração passo-a-passo. O que ocasionou um tempo elevado para entender o funcionamento do pacote e classes contidas neste.

Como a depuração passo-a-passo não era possível, foi feito uso de comandos que imprimem informações na tela para acompanhar o comportamento e seu resultado. Somente assim foi possível verificar onde existiam erros a serem corrigidos.

# 4 CONCLUSÕES

Concluí-se que os objetivos de desenvolvimento de um protótipo de formação de times de futebol de robôs utilizando a robótica baseada em comportamento foi alcançado através de:

a) desenvolvimento da classe v_RectangularAttraction_v, responsável pela implementação da área de atuação dos agentes robôs jogadores de futebol através de um campo de atração;
b) desenvolvimento de duas classes adicionais, ComportamentoSimples e ComportamentoComposto, que são, respectivamente, um time de agentes robôs jogadores de futebol com o objetivo de se posicionar no campo e um time de agentes robôs jogadores de futebol com o objetivo de se jogar contra outro time.

Através da classe v_RectangularAttraction_v é possível determinar áreas de atração que terão um desempenho melhor do que as áreas de atração definidas pela classe v_LinearAttraction_v, que determinam um campo de atração circular, pois as áreas de atração retangulares preencherão o campo por este ser retangular também.

A partir deste protótipo poderão ser feitas novas implementações que contribuam na definição de estratégias para times de robôs que jogam em ambiente simulado, com possibilidade de aplicação em times de robôs que jogam no mundo real. Como exemplo, pode ser citado campo de repulsão retangular e comunicação entre agentes robôs jogadores.

## 4.1 EXTENSÕES

Como sugestão de extensão desse trabalho, são enumeradas abaixo:

a) criar uma classe implemente repulsão retangular;
b) tornar possível a utilização da classe de atração retangular utilizável indiferente do lado do time do agente robô jogador que irá utilizá-la;
c) implementar novos comportamentos compostos que combinando áreas de atração com outros comportamentos, originados na metodologia de campos potenciais ou outra metodologia, tornando o jogador mais eficiente no alcance do objetivo.

# REFERÊNCIAS BIBLIOGRÁFICAS


ARKIN, Ronald C. **Behavior-based robotics**. Cambridge, Massachusetts: The MIT Press, 1998.

BALCH, Tucker. **TeamBotsTM**. Pittsburgh, [2000?]. Disponível em: <http://www2.cs.cmu.edu/~trb/TeamBots/index.html> ou <http://www.teambots.org>. Acesso em: 14 set. 2004.

BORDINI, Rafel Heitor; VIEIRA, Renata; MOREIRA, Álvaro Freitas. Fundamentos de sistemas multiagentes. In: Congresso da Sociedade Brasileira de Computação, 21., Jornada de Atualização em Informática (JAI), 20., 2001, Fortaleza. **Anais...** Porto Alegre: SBC, 2001.

DAVID, C. Pellejero *et al*. **FURGBOL – futebol de robôs**. Rio Grande, dez. 2001. Disponível em: <http://www.ecomp.furg.br/ecompericte/2001/Resege36.pdf>. Acesso em: 25 set. 2004.

FILHO, Elso Drigo: RUGGIERO, José Roberto. **CDF – física – ensino médio – eletromagnetismo – eletrostatica**. [S.l.], [2004?]. Disponível em: <http://webfis.df.ibilce.unesp.br/cdf/roem/ele/el/el.html>. Acesso em: 15 dez. 2004.

FURLAN, Davi. **Modelagem de objetos através da UML** – the unified modeling language. São Paulo: Makron Books, 1998. 329 p.

LCMI - LABORATÓRIO DE CONTROLE E MICROINFORMÁTICA. **UFSC team**. Florianópolis, out. 2000. Disponível em: <http://www.lcmi.ufsc.br/ufsc-team/>. Acesso em: 18 set. 2004.

KITANO, Hiroaki. **RoboCup-97:** robot soccer world cup I. Berlin; New York: Springer, 1998.

LEAL, Julio C. **Futebol:** arte e ofício. Rio de Janeiro, Rio de Janeiro: Sprint, 2000.

MURPHY, Robin R. **Introduction to AI robotics**. Cambridge, Massachusetts: The MIT Press, 2000.

ROBOCUP. **ROBOCUP official site**. [S.l.], [2004?]. Disponível em: <http://www.robocup.org>. Acesso em: 30 out. 2004.

ROQUE, Waldir L.; BARONE, Dante A. C. **Workshop on intelligent robotics**. Porto Alegre: SBC, 1998.

STONE, Peter. **Layered learning in multiagent systems:** a winning approuch to robotic soccer. Cambridge, Massachusetts: The MIT Press, 2000.

SUN MICROSYSTEMS. **The Java tutorial**. [S.l.], [2003?]. Disponível em: <http://java.sun.com/>. Acesso em: 18 set . 2004.

ULLRICH, Roberto A. **Robótica:** uma introdução, o porque dos robôs e seu papel no trabalho. Rio de Janeiro: Campus, 1987.

WEISS, Gerhard. **Multiagent systems:** a modern approach to distributed artificial intelligence. Cambridge, Massachusetts: The MIT Press, 1999.

# APÊNDICE A – Diagrama de classes das principais classes utilizadas