UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SIMULAÇÃO DE FUTEBOL EM AMBIENTE WEB: VERSÃO 4.0

JOÃO HENRIQUE MAAS

BLUMENAU
2013

2013/2-10

JOÃO HENRIQUE MAAS

SIMULAÇÃO DE FUTEBOL EM AMBIENTE WEB: VERSÃO 4.0

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

Prof. Maurício Capobianco Lopes, Dr. – Orientador

BLUMENAU
2013

2013/2-10

SIMULAÇÃO DE FUTEBOL EM AMBIENTE WEB: VERSÃO 4.0

Por

JOÃO HENRIQUE MAAS

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

Presidente:	Prof. Maurício Capobianco Lopes, Dr. – Orientador, FURB
Membro:	Prof. Joyce Martins, M.Sc. – FURB
Membro:	Prof. Dalton Solano dos Reis, M.Sc. – FURB

Blumenau, 09 de Dezembro de 2013

Dedico este trabalho a todos os familiares, especialmente à minha namorada que me apoiou e incentivou para a realização deste. Também, a todo o corpo docente da FURB que me capacitaram para esta conquista.

AGRADECIMENTOS

À minha família, que mesmo longe, sempre esteve presente.

À minha mãe, por ter me ensinado o quão importante é se dedicar aos estudos.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Maurício Capobianco Lopes, por todo o auxílio prestado durante a realização deste trabalho.

Ao professor José Roque Voltolini da Silva, por ter me auxiliado no desenvolvimento da proposta deste trabalho.

Ao professor Dalton Solano dos Reis, pelas dicas e conselhos que ajudaram na realização deste.

Especialmente à minha namorada pelo apoio, empenho, carinho, atenção, amor, incentivo, cobrança e ajuda. Que acreditou na conclusão deste trabalho mesmo quando eu pensava em desistir. Muito obrigado Aline por tudo.

A alegria está na luta, na tentativa, no sofrimento envolvido. Não na vitória propriamente dita.

Mahatma Gandhi

RESUMO

Este trabalho apresenta o desenvolvimento de um planejador de movimentos baseado em campos potenciais atrativos e repulsivos, para o jogador que está com a bola em um campo de futebol. Os campos potenciais são gerados utilizando a solução numérica de problema de valor de contorno, visando evitar mínimos locais e criar movimentos suaves para o jogador. Este planejador foi aplicado ao simulador desenvolvido por Schleuss (2011), o qual foi estendido a partir dos trabalhos de Rodrigues (2008) e Schulter (2007). Melhorias de usabilidade na definição de táticas e a descrição textual da partida simulada também foram disponibilizadas no simulador.

Palavras-chave: Simulação. Futebol. Campos potenciais. Planejamento de caminhos.

ABSTRACT

This work presents the development of a motion planner based on attractive and repulsive potential fields for the player who has the ball on a football field. Potential fields are generated using the numerical solution of boundary value problem, aiming to avoid local minima and create smooth movements for the player. This planner has been applied to simulator developed by Schleuss (2011), which was extended from the work of Roberts (2008) and Schulter (2007). Usability Improvements in defining tactics and textual description of the simulated match were also available in the simulator.

Key-words: Simulation. Football. Potential fields. Path planning.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

AJAX – *Asynchronous Javascript And XML*

API – *Application Programming Interface*

CG – Computação Gráfica

FIFA – *Fédération Internationale de Football Association*

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

IA – Inteligência Artificial

IOS – *Iphone Operational System*

JS – *JavaScript*

JSON – *JavaScript Object Notation*

JSP – *Java Server Pages*

OpenGL – *Open Graphics Library*

PVC – Problema de Valor de Contorno

RF – Requisito Funcional

UC – *Use Case*

UML – *Unified Modeling Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O futebol é o esporte mais popular do mundo. Não é somente o preferido pelos jogadores em nível recreativo, mas também é o favorito dos espectadores (FRISSELLI; MANTOVANI, 1999, p. xxv). Segundo Wuolio (1981, p. 10), são várias as razões para o futebol ser considerado o rei dos esportes: seus requisitos básicos são simples e não muito numerosos, proporciona uma atividade física bastante variada, favorece o desenvolvimento social do indivíduo através da necessidade de colaboração, permite ações individuais de grande habilidade, é o tipo do esporte com diferentes funções possibilitando a escolha de uma delas e é de fácil organização. Devido a estas razões, atrai com facilidade inúmeros espectadores e praticantes.

O futebol também tornou-se popular nos jogos eletrônicos. Existem vários formatos de jogos eletrônicos, como controle de jogadores, gerência de times, simulação de partidas, entre outros. Sendo assim, a busca por tornar os jogos eletrônicos mais parecidos com os jogos de futebol do mundo real, tornou-se um desafio para a computação em diversas áreas, como Computação Gráfica (CG), Inteligência Artificial (IA), entre outras. Muitas ferramentas vêm sendo criadas ao longo do tempo para atender este desafio.

Dentre estas ferramentas, cita-se a de Schulter (2007), que criou um simulador de futebol que permite criar jogadores, formar um time, montar táticas e definir estratégias. Rodrigues (2008) acrescentou a este simulador o recurso de disponibilizá-lo em ambiente *web* para que jogadores em diferentes locais pudessem competir, melhorando significativamente o grau de entretenimento da aplicação. Porém, a simulação das ações dos jogadores ainda era muito previsível, ou seja, os jogadores tendiam a concentrar as jogadas no centro do campo.

Schleuss (2011) buscou resolver o problema de previsibilidade dos jogadores disponibilizando um motor de IA para a tomada de decisões dos jogadores. Contudo, em função da dificuldade de configurar esta inteligência, a implementação padrão dos jogadores criados no simulador continua sendo randômica, ou seja, a mesma utilizada por Schulter (2007) e Rodrigues (2008). Se os usuários do simulador desejarem um jogo mais realista, terão que obter um conhecimento mais aprofundado sobre técnicas de aprendizado com redes neurais. O último simulador apresentado em Schleuss (2011) também não detalha com facilidade o uso desta técnica, dificultando o acesso da mesma por leigos.

Assim sendo, modificou-se a ferramenta implementada por Schleuss (2011), trocando a implementação padrão da inteligência dos jogadores de randômica para a inteligência baseada em planejamento de caminhos por campos potenciais. Esta técnica consiste na

definição de um ambiente (campo de futebol) onde estão localizados: o agente autônomo (jogador com a bola), um objetivo (o gol adversário) e os obstáculos (demais jogadores). Com isso pode-se calcular, com base nas forças atrativas geradas pelo objetivo e nas forças repulsivas geradas pelos obstáculos, as direções que o agente deve seguir. O objetivo do uso de campos potenciais foi obter uma simulação com a movimentação mais real dos jogadores, não sendo necessário que o usuário conheça técnicas de aprendizado. O desenvolvimento do trabalho pretendia que as jogadas dentro da partida ficassem menos repetitivas, evitando o excesso de jogadores próximos à bola, e que as jogadas não fossem concentradas pelo centro do campo. Porém isto não foi possível através da técnica utilizada, os motivos são explicados nos resultados deste trabalho.

Também melhorou-se a usabilidade da aplicação na definição de táticas do simulador, de modo a definir a posição dos jogadores em campo através da utilização do ponteiro do *mouse*. Anteriormente as táticas eram definidas configurando as coordenadas x e y de cada jogador de forma manual. Outra melhoria de usabilidade desenvolvida foi a criação da descrição textual da partida em tempo real, com a qual melhorou-se o entendimento do usuário sobre o que está acontecendo na partida.

1.1 OBJETIVOS DO TRABALHO

O objetivo do trabalho é disponibilizar o simulador de futebol de Schleuss (2011) com a introdução de campos potenciais para o controle do fluxo do jogador com a bola.

Os objetivos específicos do trabalho são:

a) disponibilizar a implementação padrão da tomada de decisão dos jogadores criados no simulador, baseado no planejamento de caminhos com campos potenciais;
b) possibilitar ao usuário configurar suas táticas através do *mouse*, selecionando o jogador que deseja posicionar e clicando no campo de jogo na posição desejada;
c) disponibilizar a descrição textual do jogo em tempo real.

1.2 ESTRUTURA DO TRABALHO

Este trabalho é composto de quatro capítulos. O primeiro capítulo é este no qual foram apresentados a contextualização e os objetivos.

O segundo capítulo contém a fundamentação teórica necessária para a compreensão dos temas abordados na implementação.

O terceiro capítulo contempla a descrição do desenvolvimento deste trabalho. Nele são descritos os casos de uso e os diagramas de classe e atividade. No terceiro capítulo também são apresentados detalhes da implementação e são discutidos os resultados obtidos

Por último, o quarto capítulo expõe as considerações finais e conclusões, finalizando com sugestões de extensões para projetos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a revisão bibliográfica do presente estudo. Na seção 2.1 é definido o que é futebol e são discutidos alguns simuladores deste esporte. Na seção 2.2 é explorado o assunto sobre planejamento de caminhos. Na seção 2.3 é explorado o assunto sobre campos potenciais. Na seção 2.4 são descritos seis trabalhos correlatos.

2.1 FUTEBOL

O futebol é um esporte jogado por dois times com no máximo 11 jogadores cada, sendo que um jogador de cada time deve ser o goleiro. O campo de jogo é retangular e possui duas traves, uma em cada um dos lados menores do campo. A partida tem o tempo de 90 minutos, sendo divida em dois tempos de 45 minutos cada. O objetivo do jogo é colocar a bola dentro do gol adversário, o maior número de vezes possível, respeitando uma série de regras, para que a disputa seja justa. Ao final dos 90 minutos o time que tiver feito mais gols vence a partida. Em caso de quantidade de gols iguais, é declarado um empate e diferentes ações podem ser tomadas dependendo do regulamento adotado pela competição que a partida pertence (FIFA, 2012).

O futebol pode também ser simulado. Cita-se como exemplo de simulador de futebol o trabalho de Schulter (2007).

> O usuário comanda o seu time da seguinte forma: ele possui um time com um plantel de jogadores. Tendo isto, o usuário pode definir os onze (11) jogadores que participarão da partida, o posicionamento de cada jogador conforme os seus atributos e a forma que o time de uma maneira geral vai se comportar durante a simulação, como por exemplo se vai jogar de maneira defensiva, ofensiva, jogadas pelas laterais ou contra ataque. Esta configuração para a partida é denominada de tática. (SCHULTER, 2007, p. 20).

Assim, como Schulter (2007), Managerzone Football (MANAGERZONE, 2007), Hattrick (HATTRICK, 2007) e GameGol (GAMEGOL, 2007), também são exemplos de simuladores de futebol.

O simulador Managerzone Football é um *game multiplayer online* que permite a milhares de jogadores interagirem uns com os outros. O usuário pode gerenciar um time de futebol ou um de *hockey* no gelo. Entre as opções de interação disponíveis ao usuário estão: compra e venda de jogadores, treinamento de jogadores visando melhorar o desempenho do time, desafio a outros times (amistosos) e participação em campeonatos. Para simulação das partidas, o usuário tem uma lista das próximas partidas que vai participar. Para cada partida é necessário definir até duas horas antes do início da partida a tática que será utilizada. O usuário é responsável também pela administração do clube, pois ele pode vir a falência. Para isso ele deve gerenciar as receitas e despesas do seu clube (SCHULTER, 2007, p. 24).

O simulador Hattrick (HATTRICK, 2007) se auto intitula como o jogo de futebol *online* mais popular do mundo, com cerca de novecentos e sessenta mil (960.000) usuários. A cada semana mais de um milhão (1.000.000) de partidas são simuladas. Tem muitas características em comum com o Managerzone, como transferências, treinamentos de jogadores e escolha de táticas. Como diferencial possui conteúdos exclusivos e para acessá-los é necessário pagar (RODRIGUES, 2008, p. 27).

O GameGol é um investimento brasileiro de setecentos mil reais (R$ 700.000,00) e aproximadamente sete meses de desenvolvimento por uma equipe de dezesseis (16) pessoas, iniciado em dezembro de 2005. Três meses após seu lançamento já possuía em torno de trinta e cinco mil (35.000) usuários. As funcionalidades do simulador são idênticas às dos simuladores citados anteriormente (O2GAMES, 2006).

2.2 PLANEJAMENTO DE CAMINHOS

O problema básico a ser resolvido com planejamento de caminhos é calcular uma rota livre de obstáculos para que um agente autônomo consiga, a partir de um ponto, alcançar seu objetivo, desviando de obstáculos estáticos e dinâmicos. A primeira vista, este é um problema puramente geométrico e simples, se considerar que o agente tem um baixo grau de liberdade de movimentação. Porém, ao considerar agentes com alto grau de liberdade, o problema torna-se computacionalmente complexo (LATOMBE, 1999).

Desde os anos 70, várias abordagens de planejamento de caminhos estão sendo desenvolvidas na área da robótica e muitas destas são aplicáveis ao planejamento de caminhos nos jogos (NIEUWENHUISEN; KAMPHUIS; OVERMARS, 2006).

O planejamento deve considerar diversos fatores, tais como, os obstáculos fixos e móveis existentes na cena, outros agentes que possam interferir no caminho escolhido ou mesmo alterações de posição do objetivo. Logo, deve ter como resultado um movimento convincente para manter a sensação de naturalidade que o usuário tem ao assisti-lo (FISCHER, 2008, p. 13).

Os métodos clássicos para planejamento de caminhos são baseados em três principais abordagens (DAPPER, 2007, p. 17-18):

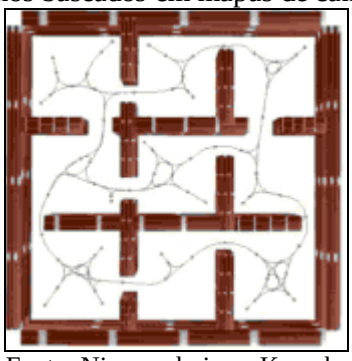
a) mapas de caminhos: representam os espaços livres do mapa, através de segmentos interligados que formam um grafo. Sobre este grafo são utilizados algoritmos de busca para definir um caminho livre até o objetivo;
b) decomposição celular: consiste em dividir o espaço livre em regiões mais simples, chamadas células, de forma que o caminho entre duas células seja possível. Estas

células são numeradas e é criado um grafo ligando as células adjacentes para ser possível definir um caminho livre até o objetivo;

c) campos potenciais: neste método o agente é tratado como uma partícula no espaço que é atraído por seu objetivo e empurrado por seus obstáculos. Neste método o mapa é subdividido em uma matriz.

Na Figura 1 é mostrado um exemplo de problema que é resolvido através do método de mapa de caminhos. Este cenário contém várias salas. Qualquer sala pode ser acessada de qualquer ponto, seguindo um dos caminhos do mapa.

Figura 1 - Planejamento de caminhos baseados em mapas de caminho - mapa de caminhos da cena



Fonte: Nieuwenhuisen, Kamphuis e Overmars (2006).

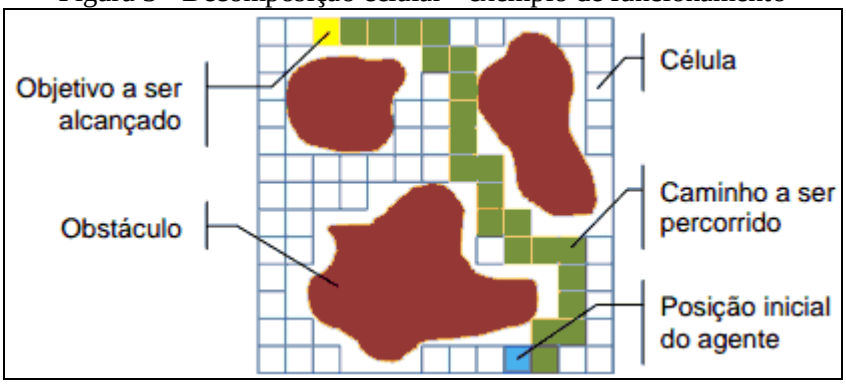
Na Figura 2, um caminho foi escolhido para se navegar entre duas salas no ambiente.

Figura 2 - Planejamento de caminhos baseados em mapas de caminho - escolha de um caminho para alcançar um objetivo

Fonte: Nieuwenhuisen, Kamphuis e Overmars (2006).

A Figura 3 exemplifica a execução de um planejamento de caminhos utilizando decomposição celular. Nela, o agente inicia o trajeto na célula azul (posição inicial do agente), e irá percorrer o caminho formado pelas células verdes, até alcançar seu objetivo, na célula amarela. As regiões vermelhas representam obstáculos no cenário.

Figura 3 - Decomposição celular - exemplo de funcionamento



Fonte: adaptado de Fischer (2008, p. 14).

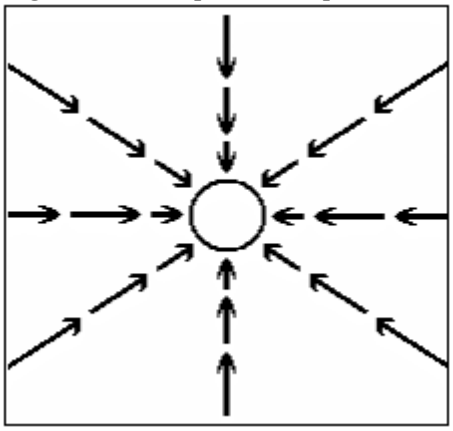
2.3 CAMPOS POTENCIAIS

Os campos potenciais têm a finalidade de guiar os agentes autônomos até seus objetivos através de caminhos livres de obstáculos. Com esta técnica é gerado um campo potencial ou campo de força que se expande por todo o ambiente e é criado a partir de forças atrativas, geradas pelos objetivos, e forças repulsivas, geradas pelos obstáculos (KHATIB, 1980 apud FERRARI, 2011, p. 13). Segundo Dapper (2007, p. 28), "isto nem sempre é bem sucedido, pois em algumas configurações do ambiente o agente pode ficar preso em um mínimo local".

Os campos potenciais são classificados em sete grupos (GOODRICH, 2004):

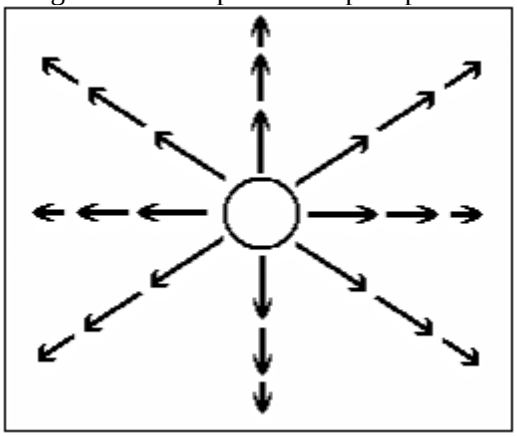
a) atrativos: são gerados pelos objetivos e têm a função de atrair os agentes autônomos. Um exemplo pode ser visto na Figura 4;
b) repulsivos: são gerados pelos obstáculos e têm a função de repelir os agentes autônomos (Figura 5);
c) perpendiculares: são gerados pelos obstáculos e paredes e fazem com que os agentes autônomos não se aproximem (Figura 6);
d) tangenciais: são uma espécie de espiral potencial ao redor do obstáculo a partir do centro do mesmo, podendo ser em sentido horário ou anti-horário (Figura 7);
e) uniformes: têm a função de permitir que os agentes autônomos possam se mover perto das paredes, que geralmente são interpretadas como obstáculos (Figura 8);
f) randômicos: são gerados de forma aleatória e servem para ajudar os agentes autônomos a não ficarem presos em locais mínimos;
g) *avoid past*: gera um campo potencial repulsivo no caminho onde o agente autônomo já passou. Serve para auxiliar na saída de mínimos locais. Isso faz com que, conforme o agente se mova pelo mínimo local, o potencial repulsivo vá aumentando de modo a repelir o agente deste local.

Figura 4 - Exemplo de campo atrativo



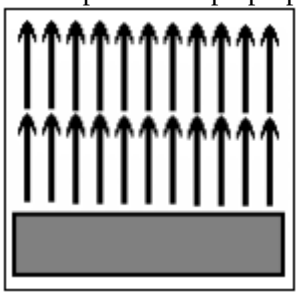
Fonte: Mafra (2004, p. 18).

Figura 5 - Exemplo de campo repulsivo



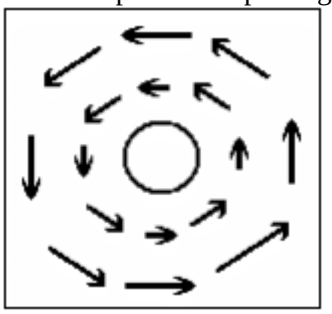
Fonte: Mafra (2004, p. 18).

Figura 6 - Exemplo de campo perpendicular



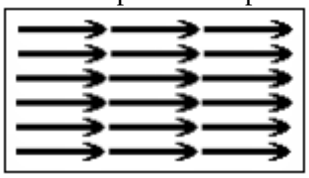
Fonte: Mafra (2004, p. 18).

Figura 7 - Exemplo de campo tangencial



Fonte: Mafra (2004, p. 18).

Figura 8 – Exemplo de campo uniforme



Fonte: Mafra (2004, p. 18).

A técnica de campos potencias consiste em discretizar o espaço em uma matriz. A célula onde está localizado o objetivo recebe o valor potencial mais baixo, as células que representam os obstáculos recebem o potencial mais alto e as células livres recebem um potencial mediano. Após aplicar a técnica de relaxamento da matriz, é possível navegar da célula onde encontra-se o agente até a célula objetivo, sempre buscando a célula adjacente de menor potencial (FISCHER, 2008, p. 16).

Uma variação desta técnica consiste em extrair uma parte menor do mapa e aplicar campos potenciais nesta parte. Esta variação é chamada de mapa local. As principais vantagens em se utilizar campos potenciais localmente são o ganho de desempenho e a

redução do consumo de memória, sem afetar significativamente o resultado (FISCHER, 2008, p. 16).

Em relação a planejadores baseados em mapas de caminho e decomposição celular pode-se citar as seguintes vantagens dos campos potenciais (FERRARI, 2011, p. 18):

a) na técnica baseada em mapas de caminho existe um grande esforço computacional inicial para que sejam calculados todos os possíveis caminhos até o objetivo. Caso existam obstáculos dinâmicos, este esforço inicial precisa ser repetido a cada movimento do agente. Isto não ocorre em campos potenciais, já que o único esforço realizado é o relaxamento da grade que, quando utilizado o método local, é matematicamente mais simples;
b) na técnica baseada em decomposição celular é necessário um grande processamento para a execução do algoritmos de pesquisa em células. Já em campos potenciais o caminho é encontrando baseando-se apenas na navegação da célula de maior potencial para a célula de menor potencial da matriz gerada sobre o mapa.

Vários métodos que utilizam campos potencias são muito eficientes, porém costumam sofrer com o problema de minímos locais. Desta forma devem ser utilizadas técnicas para escapar destas regiões, ou funções que não possuam minímos locais (DAPPER, 2007, p. 18).

2.3.1 Problema dos mínimos locais

Um exemplo de minímo local é gerado por um obstáculo côncavo e simétrico que atrai o agente autônomo pelo menor potencial, fazendo com que o agente fique preso em uma região onde as forças de atração e repulsão são iguais (DAPPER, 2007, p. 18-19).

Portanto, é necessário obter campos potenciais que sejam livres de mínimos locais. Para Dapper (2007) a melhor forma de encontrar a solução deste problema é através de soluções numéricas de uma equação diferencial parcial apropriada com condições de contorno convenientes, ou seja, um Problema de Valor de Contorno (PVC). Entre as várias formas de resolver o problema de minímos locais também podem ser citados os campos potenciais rândomicos e os campos *avoid past*.

2.3.2 Problema de valor de contorno

A primeira proposta para planejamento de movimento utilizando PVC foi feita por Connolly, Burns e Weiss (1990, p. 2102-2103) com base na utilização de funções harmônicas.

Por definição, uma função é considerada harmônica em um domínio $\Omega \subset \Re^n$, quando esta satisfaz a equação de Laplace (Quadro 1).

Quadro 1 – Equação de Laplace

$$\nabla^2 p(r) = \sum_{i=1}^{n} \frac{\partial^2 p(r)}{\partial x_i^2} = 0$$

onde ∇^2 representa o gradiente, p(r) representa o valor da célula calculada e x_i representa a soma dos potenciais das células adjacentes

Fonte: Connolly, Burns e Weiss (1990, p. 2102).

A equação de Laplace é importante em muitas áreas da física, como eletromagnetismo, gravitação e dinâmica de fluídos, pois descreve campos potenciais usados em cada uma delas. Aplicando-a sobre uma matriz obtêm-se os potenciais de cada célula. Se, em seguida, forem calculadas as linhas de fluxo, isto define o caminho para o qual uma partícula, carga elétrica ou massa pode mover-se, considerando-se que o fluxo se dá de uma célula com potencial maior para uma com potencial menor (DAPPER, 2007, p. 28).

A abordagem utilizada para resolver numericamente um PVC é considerar que o espaço da solução está discretizado em uma grade regular com células do mesmo tamanho. Cada célula (*i, j*) está associada a uma região quadrada do ambiente real, de tamanho unitário, e armazena um valor potencial $\rho^{h}_{i,j}$ no instante ^{h}. As condições de contorno de Dirichlet atribuem às células que contêm obstáculos um alto valor de potencial, enquanto as células que contêm o objetivo armazenam um baixo valor de potencial. O alto valor de potencial evita que o agente vá em direção aos obstáculos enquanto o baixo valor de potencial gera uma força de atração que atrai os agentes. O potencial das células livres também deve ser calculado (FISCHER, 2008, p. 18-19).

Para calcular o potencial das células livres é necessário atribuir ao valor potencial de cada célula, uma média dos potenciais das células adjacentes, durante uma quantidade ^{h} de iterações. Neste processo, as células com obstáculos ou objetivos permanecem inalteradas (FISCHER, 2008, p. 19).

Segundo Fischer (2008), o modo como as células são calculadas tem impacto direto no desempenho e na qualidade dos resultados obtidos. Fischer (2008) destaca que o método de Gauss-Seidel gera curvas suaves, com um pequeno número de iterações, o que permite a utilização de resultados parciais de modo mais eficiente.

2.3.3 Método de Gauss-Seidel

O método de Gauss-Seidel é um método iterativo para resolução de sistemas de equações lineares. Ele consiste em substituir o valor do potencial de cada célula livre pela média simples de seus vizinhos, com o diferencial que os vizinhos pertencem a iterações diferentes (DAPPER, 2007, p. 29). No Quadro 2 é mostrada a equação do método Gauss-Seidel.

Quadro 2 – Equação Gauss-Seidel

$$p_{x_i,y_j}^{k+1} = \frac{1}{4}(p_{x_{i+1},y_j}^{k} + p_{x_{i-1},y_j}^{k+1} + p_{x_i,y_{j+1}}^{k} + p_{x_i,y_{j-1}}^{k+1})$$

- *k* representa a iteração atual de relaxamento;
- *k+1* representa a próxima iteração de relaxamento.

Fonte: Dapper (2007, p.29).

A equação do Quadro 2 fornece um algoritmo apropriado para ser utilizado em máquinas monoprocessadas. Pode ser utilizada uma matriz e percorrê-la sequencialmente, substituindo cada célula pela média simples das células adjacentes. A equação prevê a utilização de dois vizinhos atualizados (k+1) e dois vizinhos pertencentes a iteração atual (k). Assim, um valor mais atual, o produzido na iteração k+1, é propagado, fazendo com que o resultado final seja obtido mais rapidamente. Uma forma de acelerar a obtenção do resultado é alternar o sentido em que a matriz é percorrida a cada iteração (DAPPER, 2007, p. 30).

2.3.4 Movimento do agente

Após o campo potencial ter sido calculado, é possível obter o gradiente descendente de cada célula. O gradiente descendente será utilizado para orientar o agente virtual em seu movimento. No caso bidimensional, para uma célula (*i*,*j*), o seu gradiente descendente é definido pelo vetor do Quadro 3 (FISCHER, 2008, p. 21).

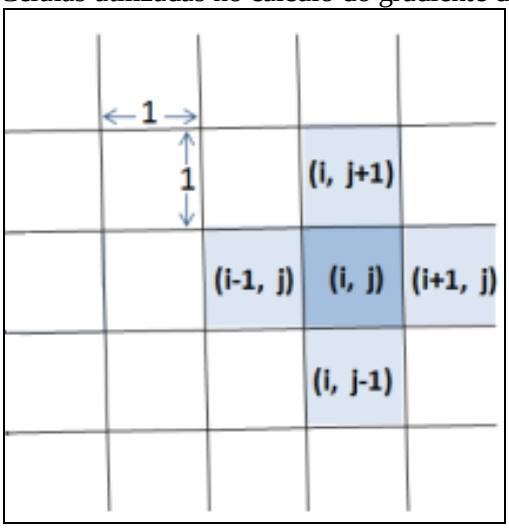
Quadro 3 – Vetor do gradiente descendente

$$(\nabla p)_{i,j} = \left(\frac{p_{i+1,j} - p_{i-1,j}}{2}, \frac{p_{i,j+1} - p_{i,j-1}}{2}\right)$$

Fonte: Fischer (2008, p. 21).

A Figura 9 mostra as células utilizadas no cálculo.

Figura 9 – Células utilizadas no cálculo do gradiente descendente



Fonte: adaptado de Fischer (2008, p. 21).

Com o resultado da equação apresentada no Quadro 3 será definida a direção que o agente deve seguir para alcançar seu objetivo, a partir da célula (*i*,*j*) (FISCHER, 2008, p. 21).

2.4 TRABALHOS CORRELATOS

Nesta seção são apresentados os trabalhos correlatos ao estudo realizado, classificados em dois tipos: os que tratam da evolução da ferramenta que está sendo estendida (Schulter, 2007; Rodrigues, 2008; Schleuss, 2011) e os que tratam do tema de campos potenciais (Mafra, 2004; Dapper, 2007; Ferrari, 2011).

Em Schulter (2007) é apresentado um simulador de futebol para o ambiente *desktop*, onde é possível criar times cadastrando jogadores, táticas e estratégias. Com os times criados é possível iniciar partidas entre dois times. Para que seja possível dois usuários competirem entre si, é necessário que os dois utilizem o mesmo computador, gerando o inconveniente de que um usuário conhece a estratégia do outro. A movimentação dos jogadores em campo é feita de forma randômica, tornando a simulação distante de uma partida real.

O trabalho de Rodrigues (2008) é uma extensão ao trabalho de Schulter (2007), que disponibilizou o simulador em ambiente *web* utilizando tecnologias da plataforma Java. O simulador manteve os mesmos requisitos funcionais do trabalho anterior, somando o recurso de *replay* das partidas e permitindo acesso pela *internet*. Para que o acesso pela *internet* seja possível, é necessário que *plugins* sejam instalados no navegador do usuário. O trabalho realizado propôs-se a melhorar a usabilidade e acessibilidade do simulador e não avançou em termos de simulação mais realista.

Schleuss (2011) estendeu o trabalho de Rodrigues (2008) trocando as tecnologias proprietárias da plataforma Java por recursos padrões implementados pelos navegadores

atuais, com o intuito de permitir que o simulador seja acessado por diversas plataformas, desde *desktop* até dispositivos móveis. Um recurso baseado em técnicas de aprendizado para alterar a inteligência dos jogadores também foi criado, sendo possível tornar a movimentação dos mesmos um pouco mais semelhante com o mundo real. Porém devido às dificuldades de configuração desta, a inteligência randômica, desenvolvida pelos trabalhos anteriores, foi mantida como padrão. Outra dificuldade da ferramenta desenvolvida é a definição do posicionamento dos jogadores no campo através da digitação de números para as coordenadas x e y, que torna improdutiva e complicada a definição de uma tática. A Figura 10 ilustra a definição de tática no simulador *web* versão 3.0 com os campos para definir a posição x e y de cada jogador.

Figura 10 - Tela obtida a partir do simulador web de Schleuss (2011)

Em Dapper (2007) é descrito um método de planejamento de caminhos, que permite a definição de comportamentos diferenciados para os vários agentes do ambiente de simulação. Os caminhos gerados são suaves e consideram obstáculos dinâmicos, representados pelos outros agentes em movimento. Mesmo seguindo na direção do objetivo e desviando dos obstáculos, o agente leva em consideração suas características individuais. Este método é

baseado em campos potenciais, gerados a partir de soluções numéricas para problemas de valor de contorno, que possuem a vantagem de não gerar mínimos locais.

Em Ferrari (2011) é apresentada uma biblioteca de planejamento de caminhos baseada em campos potenciais para o *Iphone Operational System* (IOS) versão 4. Estes campos potenciais são livres de mínimos locais através das soluções numéricas para problemas de valor de contorno. A biblioteca permite que vários objetivos sejam definidos em sequência, porém tem limitação de manipular um único agente por simulação de movimentação.

Em Mafra (2004) os campos potenciais são utilizados para definir a formação de um time de futebol de robôs, com base na definição da área de atuação de cada agente. A área de atuação do agente é configurada como um campo de atração, evitando que o robô distancie-se muito da sua área de atuação no campo. Cada robô tem configurações de comportamento. No trabalho foram criados dois comportamentos: simples e composto. O comportamento simples é configurado para que o time apenas posicione-se em campo. Já o comportamento composto é configurado para que o time posicione-se para jogar contra o outro time.

Uma comparação entre os trabalhos citados é apresentada no Quadro 4.

Quadro 4 – Comparação dos trabalhos apresentados

Característica / Trabalho	**Schulter (2007)**	**Rodrigues (2008)**	**Schleuss (2011)**	**Dapper (2007)**	**Ferrari (2011)**	**Mafra (2004)**
livre de mínimos locais	Não se aplica	Não se aplica	Não se aplica	Sim	Sim	Não
suporte a múltiplos agentes em uma única simulação	Sim	Sim	Sim	Sim	Não	Sim
suporte a desvio de obstáculos móveis	Não	Não	Não	Sim	Não se aplica	Sim
movimentação próxima ao mundo real	Não	Não	Não	Sim	Não	Não

De acordo com o Quadro 4, em relação ao item livre de mínimos locais, os trabalhos de Schulter (2007), Rodrigues (2008) e Schleuss (2011) não se aplicam, pois não utilizam campos potenciais, sendo assim, não têm o problema de mínimos locais. Mafra (2004) não tratou os mínimos locais em seu trabalho.

No item suporte a múltiplos agentes em uma única simulação, apenas o trabalho de Ferrari (2011) não atende o requisito, pois foi implementado para o IOS4 e o processador

utilizado não respondia a contento quando utilizado para mais de um agente, em função da grande quantidade de cálculos executados.

O item suporte a desvio de obstáculos móveis não é atendido pelos trabalhos de Schulter (2007), Rodrigues (2008) e Schleuss (2011), pois os algoritmos de movimentação dos jogadores baseiam-se unicamente no jogador com a bola seguir para o gol e os jogadores sem a bola irem em direção a ela. Sendo assim, o jogador com a bola não tenta desviar de obstáculos entre ele e seu objetivo. O trabalho de Ferrari (2011) não se aplica a este requisito, pois a simulação da movimentação do agente não possui obstáculos móveis.

No item movimentação próxima ao mundo real, apenas o trabalho de Dapper (2007) atende o requisito, pois o agente leva em consideração suas características individuais para traçar seu caminho. Os demais trabalhos utilizam a mesma fórmula para calcular o caminho de todos os agentes da simulação, fazendo com que tenham movimentos semelhantes.

3 DESENVOLVIMENTO

Este capítulo detalha as etapas de desenvolvimento das novas funcionalidades do Simulador. São apresentados os requisitos, a especificação e a implementação das mesmas, mencionando as técnicas e ferramentas utilizadas. Também é apresentada a operacionalidade das funções e os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os Requisitos Funcionais (RF) da aplicação proposta são:

a) ter como implementação padrão para os jogadores que estiverem com a bola comportamentos orientados com base em campos potenciais (RF01);
b) permitir que o usuário defina o posicionamento dos jogadores com o ponteiro do *mouse*, na definição de táticas (RF02);
c) permitir a visualização dos eventos da partida em modo texto em tempo real (RF03).

Os requisitos desenvolvidos por Schleuss (2011) foram mantidos. Os requisitos definidos para este trabalho visaram apenas melhorias para o usuário, evitando alterações de comportamento nas funcionalidades existentes.

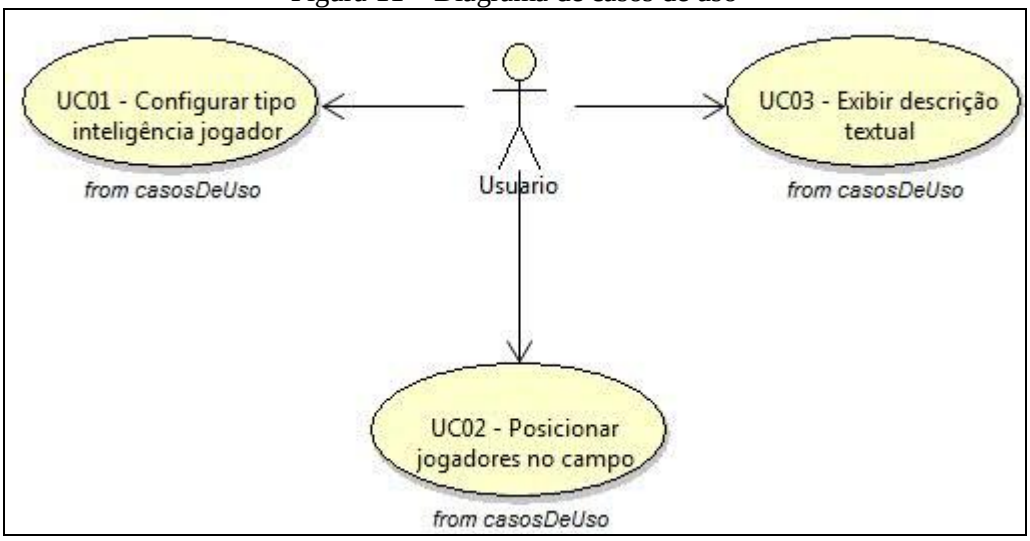
3.2 ESPECIFICAÇÃO

A especificação do presente trabalho foi desenvolvida com base em alguns diagramas da *Unified Modeling Language* (UML). A ferramenta Jakaré foi utilizada para a criação dos diagramas de casos de uso, classes e atividades.

3.2.1 Diagrama de casos de uso

Como este trabalho é uma extensão de Schleuss (2011), na Figura 11 são apresentados somente os casos de uso implementados por este trabalho.

Figura 11 – Diagrama de casos de uso



Os casos de uso visualizados na Figura 11 são:

a) `configurar tipo inteligência jogador`: permite ao usuário selecionar uma inteligência para o jogador do time (Quadro 5);

b) `posicionar jogadores no campo`: permite ao usuário posicionar os jogadores em campo através de cliques do *mouse* (Quadro 6);

c) `exibir descrição textual`: permite ao usuário visualizar em forma de texto os eventos da partida durante sua simulação (Quadro 7).

Quadro 5 – Detalhamento do UC01

Caso de uso	**UC01 – Configurar tipo inteligência Jogador**
Pré-condições	O usuário possui uma equipe criada com jogadores cadastrados.
Cenário principal	1. O usuário abre a tela de edição de jogador, clicando sobre o nome; 2. O usuário clica no botão “Avançado”; 3. O usuário escolhe entre uma das 3 opções de inteligência para o jogador (Inteligência Campos Potenciais(PVC); Inteligência Randômica e Inteligência Avançada (Neural)); 4. O usuário clica em alterar.
Pós-condições	O jogador fica configurado com a inteligência selecionada.

Quadro 6 – Detalhamento do UC02

Caso de uso	**UC02 – Posicionar jogadores no campo**
Pré-condições	O usuário possui uma equipe criada e está cadastrando/editando uma tática.
Cenário principal	1. O usuário clica sobre o campo x ou o campo y do jogador que deseja posicionar em campo; 2. O usuário clica na posição do campo em que deseja posicionar o jogador.
Pós-condições	Os campos x e y do jogador que foi posicionado ficam preenchidos e o foco passa para o campo x do próximo jogador.

Quadro 7 – Detalhamento do UC03

Caso de uso	**UC03 – Exibir descrição textual**
Pré-condições	O usuário deve ter iniciado uma partida.
Cenário principal	1. O usuário consegue visualizar a descrição das finalizações, gols, passes e desarmes que estão ocorrendo na partida.

No Quadro 8 é representado o relacionamento entre os requisitos e os casos de uso que os atendem.

Quadro 8 – Relação entre requisitos e casos de uso

Casos de uso / Requisitos	UC01	UC02	UC03
RF01	X		
RF02		X	
RF03			X

3.2.2 Diagrama de classes

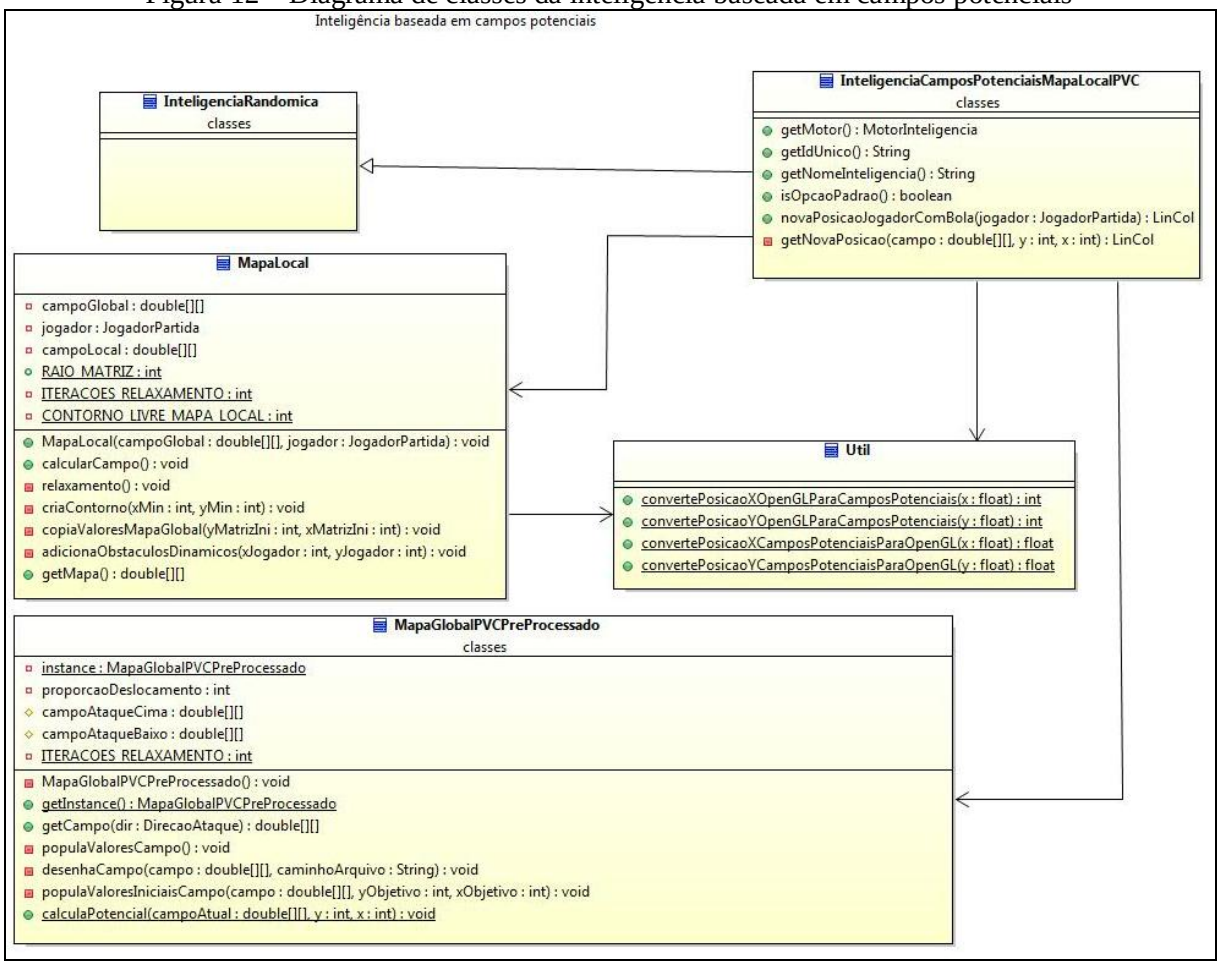
Nesta seção são apresentados os diagramas de classes das novas funcionalidades desenvolvidas. Apenas métodos e atributos utilizados por este trabalho estão ilustrados nos diagramas.

A Figura 12 apresenta as classes responsáveis pela nova inteligência implementada. Na sequência é feita uma análise sobre a funcionalidade de cada uma delas.

A classe `MapaGlobalPVCPreProcessado` é a classe que armazena os mapas globais do campo de futebol. O campo potencial é representado por uma matriz do tipo `double`.

Existem dois campos potenciais na classe: um em que o objetivo é o gol superior e outro em que o objetivo é o gol inferior do campo de jogo. Esses campos potenciais ficam armazenados nos atributos `campoAtaqueCima` e `campoAtaqueBaixo`, respectivamente.

Figura 12 – Diagrama de classes da inteligência baseada em campos potenciais



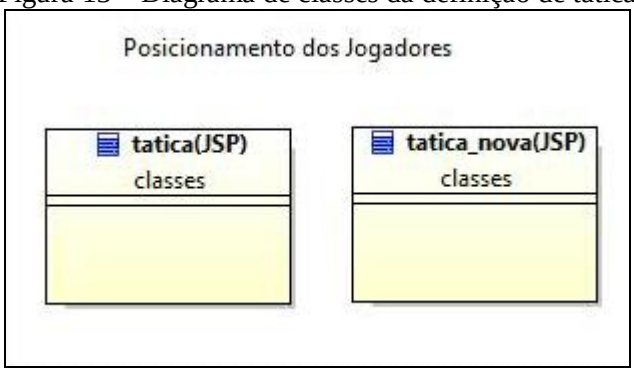
A classe `InteligenciaRandomica` representa a inteligência de um jogador baseada na tomada de decisões através da geração de números randômicos. Esta inteligência foi criada por Schulter (2007) e atualizada por Rodrigues (2008). Segundo Schleuss (2011) "outros tipos de inteligência que sejam desenvolvidos podem estender a inteligência randômica, fazendo assim com que algumas ações comuns não precisem ser implementadas".

A inteligência do jogador baseada em campos potenciais é representada pela classe `InteligenciaCamposPotenciaisMapaLocalPVC`. Esta classe implementa os métodos de identificação da inteligencia (`getMotor`, `getIdUnico`, `isOpcaoPadrao` e `getNomeInteligencia`) e a rotina de movimentação do jogador (`novaPosicaoJogadorComBola`), aproveitando os comportamentos da inteligência base (`InteligenciaRandomica`).

A classe `MapaLocal` é a responsável por efetuar os cálculos referentes ao mapa local do agente (`MapaLocal`), assim como fornecer o mapa local calculado (`getMapa`) para que o agente possa tomar a decisão de onde ir.

A Figura 13 apresenta as classes responsáveis pelo posicionamento dos jogadores em campo na definição de táticas.

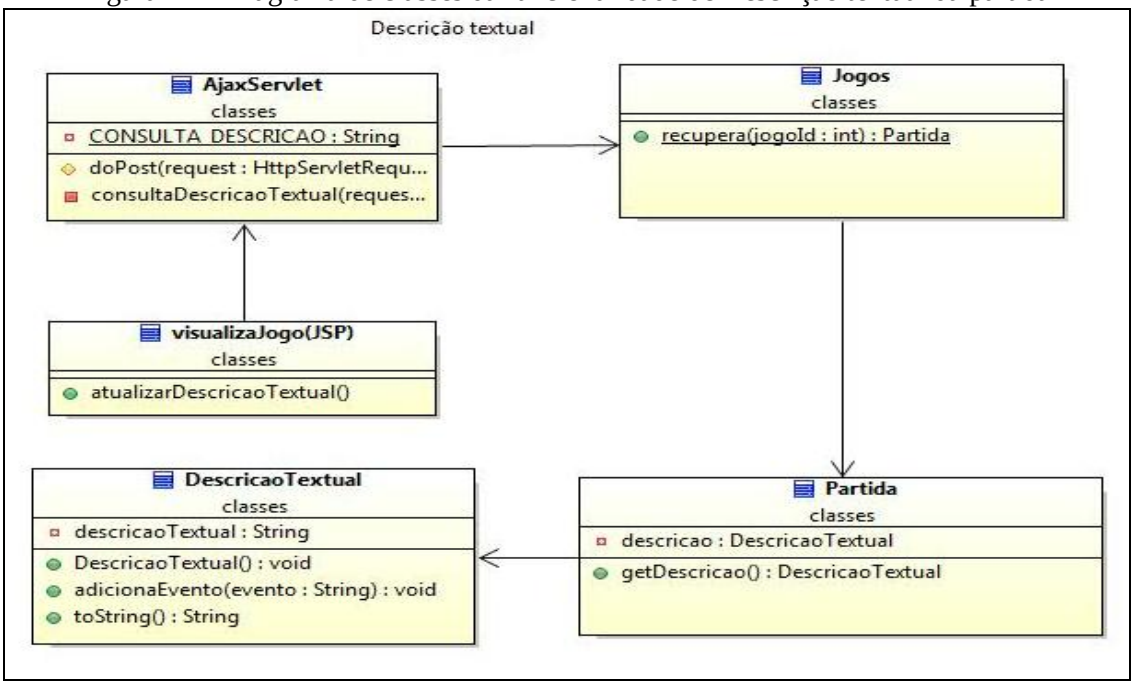
Figura 13 – Diagrama de classes da definição de tática



As classes `tatica` e `tatica_nova` são responsáveis por editar e criar táticas, respectivamente. Nelas foram criadas as funções que posicionam os jogadores com o ponteiro do *mouse*.

Na Figura 14 é apresentado o diagrama de classes referente à descrição textual da partida. Em seguida são descritas as classes e suas funcionalidades.

Figura 14 – Diagrama de classes da funcionalidade de Descrição textual da partida



A renderização da partida no navegador é de responsabilidade da classe `visualizaJogo`, classe alterada para executar a atualização e exibição da descrição textual. A

descrição é atualizada automaticamente pelo método `atualizarDescricaoTextual` através de requisições *HyperText Transfer Protocol* (HTTP) para a classe `AjaxServlet`.

A classe `AjaxServlet` tem a função de receber requisições HTTP e retornar as informações solicitadas com base nos parâmetros da requisição. Para atender a requisição de descrição textual foi criado o método `consultaDescricaoTextual` que utilizada a classe `Jogos`, através do método `recupera` já existente na versão anterior, para obter a partida que está sendo exibida no navegador. Uma partida é representada pela classe `Partida`, que armazena sua descrição textual no atributo `descricao`, criado para atender esta funcionalidade. A descrição textual é representada pela classe `DescricaoTextual`, classe criada para armazenar os eventos da partida e fornecer um texto com a descrição de cada um deles.

3.2.3 Diagrama de atividade

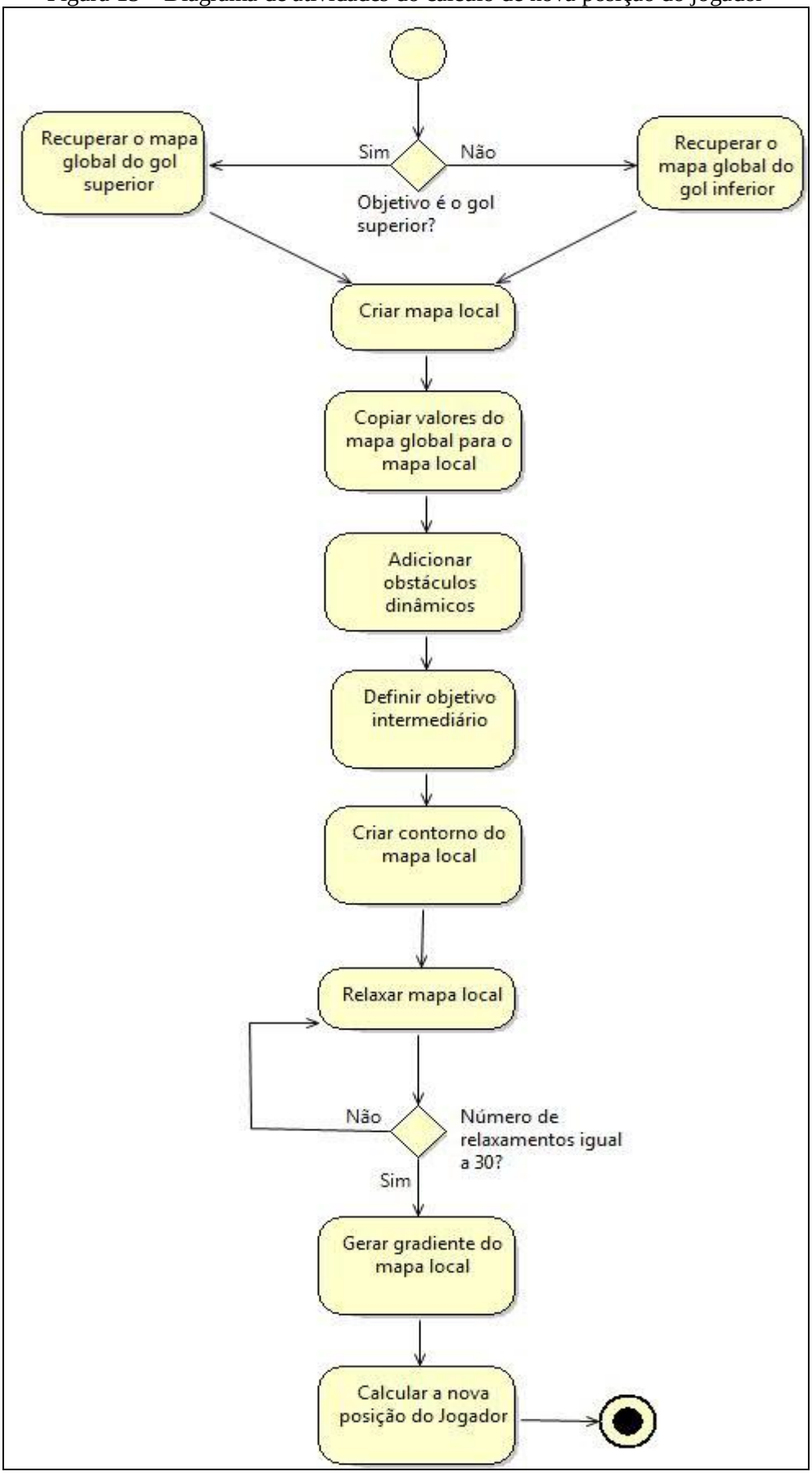
Nesta seção é apresentado o diagrama de atividades do método `novaPosicaoJogadorComBola` da classe `InteligenciaCamposPotenciaisMapaLocalPVC`, representada na Figura 12.

A Figura 15 detalha as etapas para o jogador decidir a direção de seu próximo passo. A primeira atividade para definir a nova posição do jogador no campo é escolher o mapa global que leva ao objetivo do jogador. Em seguida o mapa local é criado, sendo que o centro do mapa é a posição do jogador que está movendo-se.

Logo após, são copiados os valores de potencial do mapa global para o mapa local, os obstáculos dinâmicos (jogadores) são posicionados, é definido o objetivo intermediário (menor potencial do contorno do mapa local) e é criado o contorno do mapa. As células que fazem parte do contorno recebem potencial um, exceto a célula do objetivo intermediário e suas vizinhas, que recebem potencial de objetivo que é zero.

Com o mapa criado e configurado, acontece o laço de relaxamento do mapa local. São executados 30 (trinta) execuções de relaxamento. Após o mapa local estar relaxado, é gerado o gradiente do campo local e a nova posição do jogador é calculada.

Figura 15 – Diagrama de atividades do cálculo de nova posição do jogador



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Nesta nova versão foi mantida a ferramenta Maven (APACHE SOFTWARE FOUNDATION, 2011) que auxilia no gerenciamento e automação de projetos em Java através de arquivos de configuração simples baseado no formato *eXtensible Markup Language* (XML). O *plugin* para utilizar o servidor Jetty foi atualizado para a versão 8.1.10.v20130312, com o objetivo de manter a compatibilidade da comunicação via *WebSockets* com as versões mais recentes dos navegadores *Web*.

Para o desenvolvimento das novas funcionalidades de interface foram utilizadas as linguagens *HyperText Markup Language* (HTML), *Asynchronous Javascript And XML* (AJAX), *JavaScript* (JS) e *Java Server Pages* (JSP). Para realizar os testes do *site* foram utilizados os navegadores Internet Explorer versão 10.0 (MICROSOFT CORPORATION, 2012) e Google Chrome versão 30.0 (GOOGLE, 2013).

3.3.2 Código desenvolvido

A seguir são apresentadas três seções que detalham as implementações desenvolvidas para atender os requisitos propostos. Como este trabalho é uma continuação, informações detalhadas referentes aos itens desenvolvidos por Schulter (2007), Rodrigues (2008) e Schleuss (2011) não foram incluídas. Uma documentação completa sobre o funcionamento do simulador pode ser consultada nos trabalhos citados anteriormente.

3.3.2.1 Posicionamento dos jogadores pelo ponteiro do *mouse*

O desenvolvimento iniciou-se pelo recurso de posicionar jogadores através do ponteiro do *mouse*. Para a criação desta funcionalidade foram utilizadas funções JSP e JS. O Quadro 9 apresenta o código desenvolvido para fornecer esta funcionalidade ao usuário.

A lógica do posicionamento foi associada ao evento `click` da imagem do campo de futebol, presente na página de definição de tática. Na linha 19 é possível verificar como o evento é associado à função que gera a coordenada `x` e `y` em relação ao campo e atribui ao jogador selecionado.

Para definir as coordenadas `x` e `y` do clique dado no campo de futebol, o primeiro passo é encontrar a posição do clique relativa à imagem do campo. Nas linhas 23 e 24 são

recuperadas as coordenadas da página HTML e é descontada a distância entre a borda da imagem do campo e a coordenada (0,0) da página HTML, somada ao tamanho da borda do campo em *pixels*.

Quadro 9 – Código JS/JSP da funcionalidade de posicionar jogadores em campo com o *mouse*

```
19 $("#dropField").click(function (){
20     if(codJogadorSelecionado != null) {
21         var img = document.getElementById('dropField');
22         var bordaCampo = 33; //borda do campo
23         var x = (event.clientX - (img.getClientRects()[0].left + 1 + bordaCampo));
24         var y = (event.clientY - (img.getClientRects()[0].top + bordaCampo));
25         var campoAltura = img.offsetHeight - (2 * bordaCampo);
26         var campoLargura = img.offsetWidth - (2 * bordaCampo);
27         var escalaY = (1400/campoAltura);
28         var escalaX = (940/campoLargura);
29         y = Math.round(700 - (y*escalaY));
30         x = Math.round(-470 + (x*escalaX));
31         var edtX = document.getElementById('x' + codJogadorSelecionado);
32         var edtY = document.getElementById('y' + codJogadorSelecionado);
33         edtX.value = x;
34         edtY.value = y;
35         //alert('Posição clicada: X: ' + x +' Y: ' + y);
36     }
37 });
38 <%
39 List<JogadorPartida> jogadores1 = equipePartida.getListaJogadores();
40 if(jogadores1!=null) {
41     for(JogadorPartida jog1 : jogadores1) {
42 %>
43 $("#" + "x<%=jog1.getCdJogador() %>").focus(function (){
44     codJogadorSelecionado = "<%=jog1.getCdJogador() %>";
45 });
46 $("#" + "y<%=jog1.getCdJogador() %>").focus(function (){
47     codJogadorSelecionado = "<%=jog1.getCdJogador() %>";
48 });
```

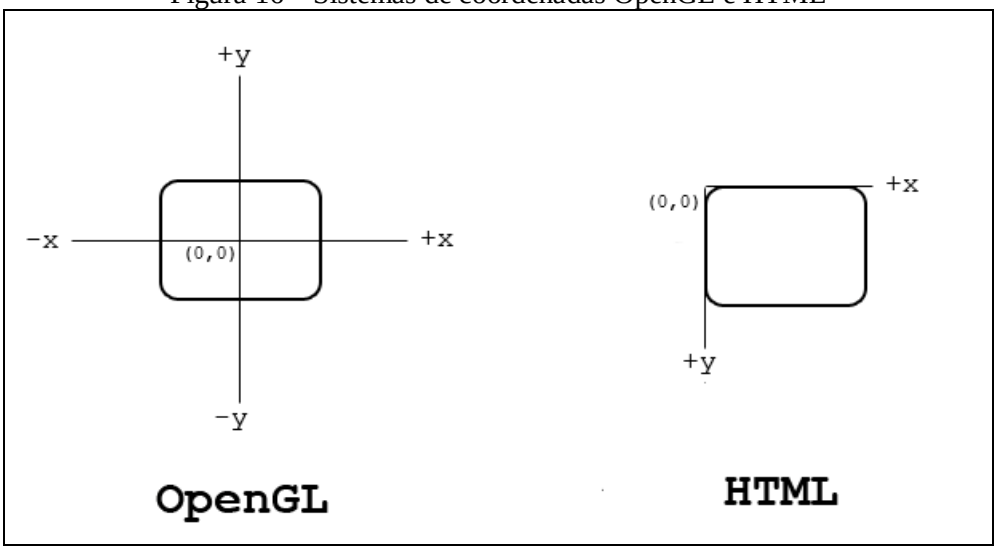
Após calcular as coordenadas `x` e `y` do clique na imagem, é necessário gerar uma proporção de quanto vale cada *pixel* da imagem em relação às dimensões do campo do jogo. Esta lógica está implementada nas linhas 25, 26, 27 e 28.

Como o simulador necessita que a posição do jogador seja informada com base nas coordenadas do *OPEN Graphics Library* (OpenGL) e o sistema de coordenadas do HTML calcula suas coordenadas de forma diferente (**Erro! Fonte de referência não encontrada.**), é necessário a conversão descrita nas linhas 29 e 30. Os valores 700 para a coordenada y e -470 para a coordenada x são utilizados devido às dimensões do campo que são fixas. Nas linhas 31 à 34 o valor da coordenada selecionada é atribuído aos campos texto da tela.

Entre as linhas 39 e 48 estão as funções responsáveis por detectar qual jogador foi selecionado pelo usuário e armazenar o código deste. Para isto, é associado ao evento `focus` dos campos `x` e `y` de cada jogador, uma função que armazena o código que corresponde ao

jogador selecionado, para que no momento em que ocorrer um clique no campo de jogo, a posição seja atribuída ao jogador.

Figura 16 – Sistemas de coordenadas OpenGL e HTML



3.3.2.2 Descrição textual

Para atender a nova funcionalidade de descrição textual da partida, foram necessárias alterações na interface e no servidor da aplicação. O leiaute da página de visualização de partidas foi alterado, sendo adicionado um campo de texto. O código exibido entre as linhas 90 e 92 do Quadro 10 mostram esta alteração que tem o objetivo de exibir a descrição textual da partida. O restante do código exibido no quadro é referente aos itens que já existiam.

Quadro 10 – Código HTML de visualização da partida

```
73 <section id="content" class="body">
74     <table style="width: 100%;">
75         <% if( "true".equals(isReplay) ) { %>
76         <tr>
77             <td>
78                 <div style="float: left; clear: both; font-weight: bold; margin-top: 10px; margin-bottom: 10px; margin-left: 8px; width: 100%;">
79                     <div id="slider" style="top: -7px;"></div>
80                 </div>
81             </td>
82         </tr>
83         <% } %>
84         <tr>
85             <td style="vertical-align: top; text-align: center;">
86                 <canvas id="futebolField" width="0" height="0">
87                     Seu browser não possui suporte ao Canvas.
88                 </canvas>
89             </td>
90             <td style="vertical-align: top; text-align: right;">
91                 <textarea readonly id="descricao" style="widht:250px;height:550px"><%=partida.getDescricao().toString()%></textarea>
92             </td>
93         </tr>
94         <% if( "true".equals(isReplay) ) { %>
95         <tr>
96             <td>
97                 <span style="float: left; clear: both; font-weight: bold; margin-top: 10px; margin-left: 8px; width: 100%; text-align: center;">
98                     <button id="beginning">Para o inicio</button>
99                     <button id="rewind">Para traz</button>
100                    <button id="reverse">Reverte</button>
101                    <button id="play">Continua</button>
102                    <button id="forward">Para frente</button>
103                    <button id="end">Para o fim</button>
104                </span>
105            </td>
106        </tr>
107        <% } %>
108    </table>
109 </section>
```

A atualização da descrição textual é realizada automaticamente a cada um segundo. Para isso foi desenvolvida uma função JS conforme Quadro 11. Nas linhas 50 e 51 é

executada uma requisição AJAX para o servidor, solicitando a descrição textual da partida que está sendo visualizada pelo usuário. O retorno da solicitação tem o formato *JavaScript Object Notation* (JSON). Para tratá-lo é utilizada a função `getJSON` (linha 50) da API jQuery[1] que converte o texto JSON para objetos em memória. Após obter o valor da descrição textual (linha 54), o texto da descrição é obtido através do seu identificador `descricao` (linha 55) e o texto do campo é substituído pela descrição mais atual fornecida pelo servidor de aplicação (linha 56). Na linha 60 a execução da função `atualizarDescricaoTextual` é agendada para ser executada novamente em um segundo através da função `setTimeout`.

Quadro 11 – Código responsável por atualizar a descrição textual da partida em exibição

```
48 function atualizarDescricaoTextual() {
49
50     $.getJSON("<%=JSPUtil.getHost()%>/AjaxServlet?cdPartida="+<%=partida.getId()%>+
51         "&acao=consultaDescricaoTextual&format=json&jsoncallback=?",
52         function(data){
53             $.each(data.items, function(i,item){
54                 var descricao = item.value;
55                 var textArea = document.getElementById('descricao');
56                 textArea.value = descricao;
57             });
58         });
59
60     setTimeout("atualizarDescricaoTextual()",1000);
61 }
```

Para dar suporte às requisições de descrição textual, o servidor da aplicação foi alterado iniciando pela classe `AjaxServlet`. Confome Quadro 12, o método `doPost` foi alterado para considerar a ação `CONSULTA_DESCRICAO`.

Quadro 12 – Processamento das requisições para a classe AjaxServlet

```
70 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
71
72     // Recupera a ação
73     String action = "";
74     try {
75         action = String.valueOf(request.getParameter("acao"));
76     } catch (Exception e) {
77         LOG.error("Erro ao recuperar ação.", e);
78     }
79
80     // Valida o login no momento do cadastro
81     if (action.equals(VALIDA_LOGIN)) {
82         validaLogin(request, response, VALIDA_LOGIN);
83     }
84
85     // Valida o login no momento da edição do usuário
86     if (action.equals(VALIDA_LOGIN_NOVO)) {
87         validaLogin(request, response, VALIDA_LOGIN_NOVO);
88     }
89
90     // Consulta as estratégias de uma equipe
91     if (action.equals(CONSULTA_ESTRATEGIAS)) {
92         consultaEstrategias(request, response);
93     }
94
95     if (action.equals(CONSULTA_DESCRICAO)) {
96         consultaDescricaoTextual(request, response);
97     }
98 }
```

[1] jQuery é uma biblioteca JS *cross-browser* desenvolvida para simplificar a criação de *scripts client side* que interagem com o HTML (JQUERY, 2007).

Sempre que a descrição textual da partida for requisitada, o método `consultaDescricaoTextual` será o responsável por efetuar o processamento da requisição. Conforme o Quadro 13, a primeira etapa do processo é recuperar o identificador da partida através do parâmetro `cdPartida`, executado na linha 107.

Quadro 13 – Processamento da requisição de descrição textual da partida

```
106 private void consultaDescricaoTextual(HttpServletRequest request, HttpServletResponse response) {
107     String cdPartida = request.getParameter("cdPartida");
108     int cd = -1;
109     try {
110         cd = Integer.parseInt(cdPartida);
111     } catch (Exception e) {
112         LOG.error("Erro ao comsultar descrição textual da Partida.", e);
113     }
114     final Partida p = Jogos.recupera(cd);
115     JSONArray descricoes = new JSONArray();
116     JSONObject descricaoTextual = new JSONObject();
117     if (p != null) {
118         descricaoTextual.put("value", p.getDescricao().toString());
119         descricoes.add(descricaoTextual);
120     }
121     try {
122         // escreve o objeto JSON
123         String jsoncallback = request.getParameter("jsoncallback");
124         String corrigida = jsoncallback + "({ \"items\": " + descricoes.toString() + " })";
125         response.setContentType("text/javascript");
126         response.setCharacterEncoding("ISO-8859-1");
127         response.getWriter().write(corrigida);
128         response.getWriter().flush();
129         response.getWriter().close();
130     } catch (Exception e) {
131         LOG.error("Erro ao comsultar descrição textual.", e);
132     }
133 }
```

Conhecendo o código da partida, é possível recupera-lá através do método `recupera` da classe `Jogos`. Esta funcionalidade é um legado dos trabalhos anteriores e é muito útil para o processamento das requisições vindas do navegador, sendo que a parte de interface armazena apenas o código da partida que está sendo visualizada. Com os dados da partida, entre as linhas 115 e 120 a descrição textual é recuperada através do método `getDescricao` da classe `Partida` e é criada a definição da descrição no formato JSON. Já entre as linhas 121 e 131 a resposta da requisição é enviada para o navegador.

Para que a partida tenha uma descrição textual, é necessário que os eventos que ocorrem na partida sejam adicionados a essa descrição. Para atender esse requisito foi criada a classe `DescricaoTextual` conforme Quadro 14. Seu principal método é `adicionaEvento`, que tem a função de adicionar a descrição do novo evento aos eventos ocorridos anteriormente.

Para isso, como a descrição é um texto e o último evento notificado deve ser o primeiro a ser apresentado ao usuário, sempre que uma nova notificação é recebida, a nova descrição passa a ser composta pelo último evento, duas quebras de linha e a descrição do anterior.

Quadro 14 – Classe de controle da descrição textual

```
1  package br.com.futebol.util;
2
3  public class DescricaoTextual {
4
5      private String descricaoTextual;
6
7      public DescricaoTextual() {
8          this.descricaoTextual = "";
9      }
10
11     public synchronized void adicionaEvento(String evento) {
12         evento = evento.concat("\n");
13         evento = evento.concat("\n");
14         evento = evento.concat(this.descricaoTextual);
15         this.descricaoTextual = evento;
16     }
17
18     @Override
19     public String toString() {
20         return this.descricaoTextual;
21     }
22 }
```

A arquitetura construída pelos trabalhos anteriores facilita que o objeto da classe `Partida` seja acessado de todas as rotinas que envolvem ações de jogadores. Logo, para criar uma nova notificação basta acessar o objeto da partida e executar a chamada `partida.getDescricao().adicionaEvento("eventoASerNotificado")`.

3.3.2.3 Inteligência baseada em campos potenciais

A principal implementação do presente estudo é o desenvolvimento da inteligência do jogador com a bola através de campos potenciais. O primeiro passo para que seja possível a utilização dos campos potenciais é o processamento dos mapas globais. Como este é um passo que pode ser pré-processado, por motivos de performance o cálculo dos campos potenciais globais é realizado durante a inicialização do servidor de aplicação (ver linha 43 no Quadro 15). Estes cálculos são feitos logo após o registro da inteligência baseada em campos potenciais (linha 42).

Quadro 15 – Rotina de inicialização do servidor de aplicação

```
30 public class InicializaSistema extends HttpServlet {
31
32     private static final long serialVersionUID = 1L;
33     private static final Logger LOG = Logger.getLogger(InicializaSistema.class);
34
35     @Override
36     public final void init(ServletConfig config) throws ServletException {
37         // Registra o provedor de criptografia
38         Security.addProvider(new BouncyCastleProvider());
39
40         // Registra os tipos de inteligencia
41         ProvedorInteligencia.registraProvedor(new InteligenciaRandomica());
42         ProvedorInteligencia.registraProvedor(new InteligenciaCamposPotenciaisMapaLocalPVC());
43         MapaGlobalPVCPreProcessado.getInstance();
44         ProvedorInteligencia.registraProvedor(new InteligenciaRPROP());
45         // Carrega as propriedades do sistema
46         this.carregaConfiguracaoLog(config);
47         this.carregaConfiguracaoBanco(config);
48         this.carregaConfiguracaoGeral(config);
49     }
```

Como a implementação proposta por este estudo permite a utilização do mesmo mapa global para todas as partidas, na classe `MapaGlobalPVCPreProcessado` foi utilizado o padrão de projeto *Singleton*, que garante a existência de apenas uma instância da classe implementada. No Quadro 16 encontra-se a implementação do padrão *Singleton* na classe `MapaGlobalPVCPreProcessado`.

Quadro 16 – Implementação do Singleton dos mapas globais

```
15     private static MapaGlobalPVCPreProcessado instance;
16     private int proporcaoDeslocamento;
17
18     private MapaGlobalPVCPreProcessado() {
19         this.proporcaoDeslocamento = -1;
20     }
21
22     public static MapaGlobalPVCPreProcessado getInstance() {
23         if (instance == null) {
24             instance = new MapaGlobalPVCPreProcessado();
25             instance.populaValoresCampo();
26         }
27         return instance;
28     }
```

O método `populaValoresCampo` é o responsável por criar e calcular os campos potenciais para os dois mapas da classe: um dos mapas leva ao gol superior e o outro leva ao gol inferior. Estes são armazenados respectivamente nos atributos `campoAtaqueCima` e `campoAtaqueBaixo`, que são matrizes do tipo `double`. O Quadro 17 apresenta a parte inicial da lógica do método.

Quadro 17 – Inicio da criação e cálculo de campos potenciais

```
46    private void populaValoresCampo() {
47        if (this.proporcaoDeslocamento == -1) {
48            this.proporcaoDeslocamento = Math.round(Constantes.DESLOCAMENTO_JOGADOR * Constantes.OPENGL_CONVERSION);
49        }
50        double[][] campo = new double[(int) (Constantes.CY_MAX_CAMPO * 2) / this.proporcaoDeslocamento]
51                [(int) (Constantes.CX_MAX_CAMPO * 2) / this.proporcaoDeslocamento];
52        int yObjetivo = 0;
53        int xObjetivo = (int) Constantes.CX_MAX_CAMPO / this.proporcaoDeslocamento;
54        populaValoresIniciaisCampo(campo, yObjetivo, xObjetivo);
55        ...
```

A primeira etapa para a gerar os campos potenciais é calcular a proporção de deslocamento do jogador no mapa, de modo que cada célula da grade seja equivalente a um movimento do jogador. Em seguida, a matriz com as dimensões do campo é criada dividindo a altura e largura do campo pela proporção de deslocamento. A definição das coordenadas do objetivo variam de acordo com o gol que será o objetivo. Quando for o gol superior, o valor da coordenada y será 0 (zero), caso contrário será o tamanho da matriz que representa o campo menos um. Como o gol sempre está localizado no centro do eixo x, a coordenada x é definida pela largura da matriz dividida por dois. O último passo da inicialização é configurar o peso inicial das células.

Na técnica aprimorada por Fischer (2008), durante a inicialização do campo potencial as células que possuem obstáculos recebem valor de potencial um, o objetivo recebe potencial zero e as células livres recebem um potencial próximo a um. No Quadro 18 está ilustrado o código do método `populaValoresIniciaisCampo` da classe `MapaGlobalPVCPreProcessado` responsável por inicializar os valores do mapa.

Quadro 18 – Código do método que inicializa os valores do campo potencial

```
308    private void populaValoresIniciaisCampo(double[][] campo, int yObjetivo, int xObjetivo) {
309        for (int y = 0; y < campo.length; y++) {
310            for (int x = 0; x < campo[0].length; x++) {
311                if (y == 0 || y == campo.length - 1) {
312                    if (y != yObjetivo || (x < xObjetivo - 4 || x > xObjetivo + 4)) {
313                        campo[y][x] = Constantes.PESO_OBSTACULO;
314                    }
315                } else if (x == 0 || x == campo[y].length - 1) {
316                    campo[y][x] = Constantes.PESO_OBSTACULO;
317                } else {
318                    campo[y][x] = Constantes.PESO_CELULA_LIVRE;
319                }
320            }
321        }
322    }
```

No cenário do campo de futebol, as células consideradas obstáculos no mapa global são todo o contorno do campo, exceto a célula objetivo e suas próximas quatro células vizinhas para a direita e esquerda, totalizando nove células objetivo. As demais células do campo são células livres.

Após incializar o valor de todas as células, é realizado o relaxamento da matriz, que consiste em percorrer as células e substituir seu valor de potencial pela média do potencial das

vizinhas, através do método `calculaPotencial`. No Quadro 19 é apresentada a lógica de relaxamento da matriz.

Quadro 19 – Relaxamento do mapa global

```
53          ...
54          populaValoresIniciaisCampo(campo, yObjetivo, xObjetivo);
55          for (int k = 0; k < ITERACOES_RELAXAMENTO; k++) {
56              switch (k % 4) {
57                  case 0:
58                      //percorre matriz a partir do canto inferior direito
59                      for (int y = campo.length - 1; y >= 0; y--) {
60                          for (int x = 0; x < campo[0].length; x++) {
61                              calculaPotencial(campo, y, x);
62                          }
63                      }
64                      break;
65                  case 1:
66                      //percorre matriz a partir do canto inferior esquerdo
67                      for (int y = campo.length - 1; y >= 0; y--) {
68                          for (int x = campo[0].length - 1; x >= 0; x--) {
69                              calculaPotencial(campo, y, x);
70                          }
71                      }
72                      break;
73                  case 2:
74                      //percorre matriz a partir do canto superior direito
75                      for (int y = 0; y < campo.length; y++) {
76                          for (int x = 0; x < campo[0].length; x++) {
77                              calculaPotencial(campo, y, x);
78                          }
79                      }
80                      break;
81                  case 3:
82                      //percorre matriz a partir do canto superior esquerdo
83                      for (int y = 0; y < campo.length; y++) {
84                          for (int x = campo[0].length - 1; x >= 0; x--) {
85                              calculaPotencial(campo, y, x);
86                          }
87                      }
88                      break;
89              }
90          }
91          ...
```

A principal característica desta implementação é a troca na direção em que a matriz é percorrida a cada iteração de relaxamento. Este comportamento é importante para que sejam necessárias menos iterações para ocorrer a convergência da matriz, ou seja, que de qualquer célula do mapa seja possível alcançar o objetivo caminhando sempre para a célula de menor potencial.

O valor da constante `ITERACOES_RELAXAMENTO` para o mapa global é 2500 (dois mil e quinhentos). Entre os valores testados esta foi a menor quantidade de iterações que conseguiu convergir a matriz do campo de futebol. Os testes para obter o valor de iterações necessárias foram feitos através da impressão dos potenciais do mapa global e verificação visual dos

valores gerados, a fim de garantir que os potenciais aumentem conforme as células estão mais distantes do objetivo.

O método `calculaPotencial` é um método público e estático utilizado para calcular o potencial das células do mapa global e local. Conforme pode ser visto no Quadro 20, o método não altera o potencial das células do tipo objetivo e obstáculo.

Quadro 20 – Método que calcula o potencial de uma célula através da média das vizinhas

```
318     public static void calculaPotencial(double[][] campoAtual, int y, int x) {
319         double v = campoAtual[y][x];
320         double y1x = 0;
321         double y_1x = 0;
322         double yx1 = 0;
323         double yx_1 = 0;
324         if (v != Constantes.PESO_OBSTACULO && v != 0) {
325             //cima
326             if (y - 1 >= 0) {
327                 y_1x = campoAtual[y - 1][x];
328             }
329             //Baixo
330             if (y + 1 < campoAtual.length) {
331                 y1x = campoAtual[y + 1][x];
332             }
333             //esquerda
334             if (x - 1 >= 0) {
335                 yx_1 = campoAtual[y][x - 1];
336             }
337             //direita
338             if (x + 1 < campoAtual[0].length) {
339                 yx1 = campoAtual[y][x + 1];
340             }
341             double valor = (y1x + y_1x + yx1 + yx_1) / 4;
342             campoAtual[y][x] = valor;
343         }
344     }
```

Os passos descritos acima (criação, inicialização e relaxamento do mapa global) são executados duas vezes, a fim de gerar os mapas para os dois objetivos do jogo: o gol superior e o inferior. Após o fim desta etapa de pré-processamento, o Simulador está pronto para executar partidas com jogadores que tenham sua inteligência baseada em campos potenciais.

O próximo passo da implementação da inteligência foi o desenvolvimento da classe responsável pelo controle do jogador. Conforme sugerido por Schleuss (2011), a classe `InteligenciaRandomica` foi utilizada como base para a nova inteligência criada e apenas os métodos de identificação da inteligência e o método `novaPosicaoJogadorComBola` foram sobre-escritos (Quadro 21).

Quadro 21 – Métodos da classe InteligenciaRandomica sobre-escritos pela classe `InteligenciaCamposPotenciaisMapaLocalPVC`

```
14  public MotorInteligencia getMotor() {
15      return null;
16  }
17
18  @Override
19  public String getIdUnico() {
20      return "TIPO_INTELIGENCIA_CAMPOS_POTENCIAIS_V1";
21  }
22
23  @Override
24  public String getNomeInteligencia() {
25      return "Inteligência Campos potenciais (Básica)";
26  }
27
28  @Override
29  public boolean isOpcaoPadrao() {
30      return true;
31  }
32
33  @Override
34  public LinCol novaPosicaoJogadorComBola(JogadorPartida jogador) {
35      DirecaoAtaque dir;
36      if (jogador.multLadoCampo() > 0) {
37          dir = DirecaoAtaque.CIMA;
38      } else {
39          dir = DirecaoAtaque.BAIXO;
40      }
41      double campoGlobal[][] = MapaGlobalPVCPreProcessado.getInstance().getCampo(dir);
42      MapaLocal mapaLocal = new MapaLocal(campoGlobal, jogador);
43      mapaLocal.calcularCampo();
44      LinCol novaPosicao = getNovaPosicao(mapaLocal.getMapa(), MapaLocal.RAIO_MATRIZ, MapaLocal.RAIO_MATRIZ);
45      float diffY = (novaPosicao.getY() - 8) * 2;
46      float diffX = (novaPosicao.getX() - 8) * 2;
47      int xJogador = Util.convertePosicaoXOpenGLParaCamposPotenciais(jogador.getPosicaoAtual().getX());
48      int yJogador = Util.convertePosicaoYOpenGLParaCamposPotenciais(jogador.getPosicaoAtual().getY());
49      novaPosicao.setY(Util.convertePosicaoYCamposPotenciaisParaOpenGL(yJogador + diffY));
50      novaPosicao.setX(Util.convertePosicaoXCamposPotenciaisParaOpenGL(xJogador + diffX));
51      return novaPosicao;
52  }
```

Nesta versão, a inteligência baseada em campos potenciais passou a ser padrão nos times criados no simulador. Para isso, o método `isOpcaoPadrao` foi implementado para retornar `true`.

O método `novaPosicaoJogadorComBola` inicialmente verifica qual dos gols é o objetivo do jogador que está com a bola e recupera o respectivo mapa global. Após obter o mapa global é criada uma instância da classe `MapaLocal` que irá efetuar os cálculos do mapa local do jogador, considerando os obstáculos dinâmicos (outros jogadores). Com o mapa local devidamente calculado, o jogador verifica para qual célula deve se mover através do método `getNovaPosicao` da classe `InteligenciaCamposPotenciaisMapaLocalPVC` e realiza a conversão da posição na matriz para a posição correspondente no OpenGL (linhas 45 a 50 no Quadro 21).

Devido ao fato da arquitetura desenvolvida pelos trabalhos anteriores exigir que a posição dos jogadores em campo seja correspondente com coordenadas do OpenGL, que podem ser representadas por valores negativos tendo em vista que a coordenada (0,0) fica no centro do campo, surgiu a necessidade de criar uma classe utilitária para converter a posição do jogador em OpenGL para a matriz de campos potenciais, para facilitar a manipulação de

listas do Java e também converter a posição na matriz em posição do OpenGL para atualizar a nova posição do jogador no simulador. O Quadro 22 apresenta o código responsável pelas conversões.

Quadro 22 – Classe de conversão de coordenadas entre OpenGL e campos potenciais

```
5  public class Util {
6
7      public static int convertePosicaoXOpenGLParaCamposPotenciais(float x) {
8          float xReturn = x * Constantes.OPENGL_CONVERSION;
9          xReturn += Constantes.CX_MAX_CAMPO;
10         xReturn = xReturn / Constantes.DESLOCAMENTO_JOGADOR_CP;
11         return (int) xReturn;
12     }
13
14     public static int convertePosicaoYOpenGLParaCamposPotenciais(float y) {
15         float yReturn = y * Constantes.OPENGL_CONVERSION;
16         yReturn -= Constantes.CY_MAX_CAMPO;
17         yReturn = -yReturn;
18         yReturn = yReturn / Constantes.DESLOCAMENTO_JOGADOR_CP;
19         return (int) yReturn;
20     }
21
22     public static float convertePosicaoXCamposPotenciaisParaOpenGL(float x) {
23         float xReturn = x * Constantes.DESLOCAMENTO_JOGADOR_CP;
24         xReturn = xReturn - Constantes.CX_MAX_CAMPO;
25         xReturn = xReturn / Constantes.OPENGL_CONVERSION;
26         return xReturn;
27     }
28
29     public static float convertePosicaoYCamposPotenciaisParaOpenGL(float y) {
30         float yReturn = y * Constantes.DESLOCAMENTO_JOGADOR_CP;
31         yReturn = -yReturn;
32         yReturn += Constantes.CY_MAX_CAMPO;
33         yReturn = yReturn / Constantes.OPENGL_CONVERSION;
34         return yReturn;
35     }
36 }
```

Na conversão de OpenGL para matriz de potenciais o primeiro passo é multiplicar a posição passada como parâmetro por mil, pois as coordenadas do simulador no OpenGL estão entre 0.7 e -0.7 para o eixo y e 0.47 e -0.47 para o eixo x. Para a coordenada y é necessário descontar o tamanho da metade do comprimento do campo e multiplicar esse valor por um negativo, pois a posição 0 da matriz é equivalente à posição 0.7 do OpenGL. Já para a posição `x` é necessário apenas somar a metade da largura do campo, pois a posição 0 da matriz é equivalente a posição -0.47 do OpenGL. A divisão pelo deslocamento do jogador é realizada para os dois eixos, pois quando a matriz é criada ela é reduzida por este deslocamento. A conversão de coordenada da matriz para o OpenGL é realizada através do processo matemático inverso ao explicado anteriormente.

A principal classe da inteligência implementada é a classe `MapaLocal`. Nela são efetuados todos os cálculos necessários para que o jogador com a bola desvie dos obstáculos

dinâmicos. Para que seja possível criar uma instância de `MapaLocal` é preciso um mapa global e um jogador, conforme Quadro 23.

Quadro 23 – Construtor da classe MapaLocal

```
20      public MapaLocal(double[][] campoGlobal, JogadorPartida jogador) {
21          this.campoGlobal = campoGlobal;
22          this.jogador = jogador;
23      }
```

Estas informações são importantes para a criação do mapa local pois a posição do jogador será o centro do mapa e os valores do mapa global que estão ao redor do jogador serão utilizados para inicializar os valores potenciais de cada célula local. O Quadro 24 apresenta a lógica do método `calcularCampo` responsável por criar, inicializar os potenciais, posicionar os obstáculos dinâmicos e relaxar o mapa local.

Quadro 24 – Método que cria e calcula o mapa local

```
17      private double campoLocal[][];
18      public static final int RAIO_MATRIZ = 8;
19      public void calcularCampo() {
20          int xJogador = Util.convertePosicaoXOpenGLParaCamposPotenciais(jogador.getPosicaoAtual().getX());
21          int yJogador = Util.convertePosicaoYOpenGLParaCamposPotenciais(jogador.getPosicaoAtual().getY());
22          int yMatrizIni = yJogador - RAIO_MATRIZ;
23          int xMatrizIni = xJogador - RAIO_MATRIZ;
24          //A Matriz tem o tamanho de 2 raios da matriz somando a posição do jogador
25          campoLocal = new double[RAIO_MATRIZ * 2 + 1][RAIO_MATRIZ * 2 + 1];
26          copiaValoresMapaGlobal(yMatrizIni, xMatrizIni);
27          adicionaObstaculosDinamicos(xJogador, yJogador);
28          //Define objetivo
29          double minValue = Double.MAX_VALUE;
30          int xMin = -1;
31          int yMin = -1;
32          for (int y = 0; y < campoLocal.length; y++) {
33              for (int x = 0; x < campoLocal[y].length; x++) {
34                  if (minValue >= campoLocal[y][x]) {
35                      minValue = campoLocal[y][x];
36                      yMin = y;
37                      xMin = x;
38                  }
39              }
40          }
41          criaContorno(xMin, yMin);
42          relaxamento(xMin, yMin);
43      }
```

A seguir é descrita a implementação do diagrama de atividade de criação do mapa local (Figura 15). O primeiro passo para criação do mapa local é converter a posição do jogador em OpenGL para a posição correspondente no mapa global. Este processo é realizado pelos métodos da classe `Util`. Tendo a posição do jogador em relação ao mapa global, é possível encontrar a coordenada inicial do mapa local. Isto é feito descontando o `RAIO_MATRIZ` da posição do jogador conforme as linhas 22 e 23 (Quadro 24). O valor do raio da matriz foi configurado como oito para que o mapa local gerado seja uma matriz de 17x17, pois estas são as dimensões das matrizes utilizadas por Dapper (2007), que apresentaram resultados satisfatórios de qualidade do movimento e performace.

Após criar o mapa local é necessário copiar os valores do mapa global. O Quadro 25 apresenta a lógica da cópia do valores.

Quadro 25 – Cópia dos potenciais do mapa global para o local

```
163    private void copiaValoresMapaGlobal(int yMatrizIni, int xMatrizIni) {
164        for (int y = 0; y < campoLocal.length; y++) {
165            for (int x = 0; x < campoLocal[y].length; x++) {
166                int yGlobal = yMatrizIni + y;
167                int xGlobal = xMatrizIni + x;
168                if ((0 <= yGlobal && yGlobal < campoGlobal.length) && (0 <= xGlobal && xGlobal < campoGlobal[yGlobal].length)) {
169                    campoLocal[y][x] = campoGlobal[yGlobal][xGlobal];
170                } else {
171                    //Se a posição do mapa local não existe no mapa global, logo é um obstáculo
172                    campoLocal[y][x] = Constantes.PESO_OBSTACULO;
173                }
174            }
175        }
176    }
```

O método percorre toda a matriz local e preenche as células com os potenciais correspondentes do mapa global. Conhecendo x e y iniciais referente ao mapa global, basta ir incrementando esses valores com o número do laço de repetição. Um ponto importante da lógica é verificar se a coordenada gerada está dentro dos limites do mapa global. Caso não esteja deve ser atribuído o potencial de obstáculo para a célula do mapa local. Isto deve ser feito pois se o jogador estiver próximo das laterais ou da linha de fundo, as áreas fora do campo devem repelir o jogador.

O próximo passo para gerar o mapa local é o posicionamento dos obstáculos dinâmicos (jogadores). Isto é feito atribuindo valor de potencial um para as células correspondentes à posição de cada obstáculo. Conforme descrito na técnica estudada, não podem ser posicionados obstáculos dinâmicos nas duas primeiras e nas duas últimas linhas e colunas do mapa local. Deste modo, a fim de garantir que seja possível gerar um campo potencial para o objetivo intermediário, os obstáculos que estão nestas áreas são omitidos do mapa local. Esta lógica está ilustrada no Quadro 26.

Quadro 26 – Método que adiciona os obstáculos dinâmicos ao mapa local

```
178    private static final int CONTORNO_LIVRE_MAPA_LOCAL = 2;
179    private void adicionaObstaculosDinamicos(int xJogador, int yJogador) {
180        for (LinCol linCol : jogador.getSimulador().getCampo().getPosicoes().values()) {
181            int xBase = Util.convertePosicaoXOpenGLParaCamposPotenciais(linCol.getX());
182            int yBase = Util.convertePosicaoYOpenGLParaCamposPotenciais(linCol.getY());
183            if ((xJogador - (RAIO_MATRIZ - CONTORNO_LIVRE_MAPA_LOCAL) <= xBase && xBase <= xJogador + (RAIO_MATRIZ - CONTORNO_LIVRE_MAPA_LOCAL))
184                &&
185                (yJogador - (RAIO_MATRIZ - CONTORNO_LIVRE_MAPA_LOCAL) <= yBase && yBase <= yJogador + (RAIO_MATRIZ - CONTORNO_LIVRE_MAPA_LOCAL))) {
186                if (xJogador != xBase && yJogador != yBase) {
187                    int diffY = yBase - yJogador;
188                    int diffX = xBase - xJogador;
189                    int posYLocal = RAIO_MATRIZ + diffY;
190                    int posXLocal = RAIO_MATRIZ + diffX;
191                    campoLocal[posYLocal][posXLocal] = Constantes.PESO_OBSTACULO;
192                }
193            }
194        }
195    }
```

Tendo o mapa local com os valores iniciais atribuídos e os obstáculos posicionados, é realizada a busca pelo objetivo intermediário do jogador, ou seja, a célula do mapa local que

possui menor potencial. As coordenada dessa célula são passadas como parâmetro para o método `criaContorno` que é o responsável por atribuir potencial de obstáculo para todas as células da primeira e última linha e coluna do mapa local, exceto a célula objetivo e suas células vizinhas que recebem potencial de objetivo (zero). O Quadro 27 apresenta a implementação da lógica descrita acima.

Quadro 27 – Método que cria o contorno do mapa local

```
133    private void criaContorno(int xMin, int yMin) {
134        if (yMin == 0 || yMin == campoLocal.length - 1) {
135            //se o Y é o objetivo então tem que variar o X
136            for (int y = 0; y < campoLocal.length; y++) {
137                for (int x = 0; x < campoLocal[y].length; x++) {
138                    if ((y == 0 || y == campoLocal.length - 1 || x == 0 || x == campoLocal[y].length - 1)) {
139                        if (y != yMin || x < xMin - 1 || x > xMin + 1) {
140                            campoLocal[y][x] = Constantes.PESO_OBSTACULO;
141                        } else {
142                            campoLocal[y][x] = 0;
143                        }
144                    }
145                }
146            }
147        } else {
148            //se o X é o objetivo então tem que variar o Y
149            for (int y = 0; y < campoLocal.length; y++) {
150                for (int x = 0; x < campoLocal[y].length; x++) {
151                    if ((y == 0 || y == campoLocal.length - 1 || x == 0 || x == campoLocal[y].length - 1)) {
152                        if ((x != xMin || y < yMin - 1 || y > yMin + 1)) {
153                            campoLocal[y][x] = Constantes.PESO_OBSTACULO;
154                        } else {
155                            campoLocal[y][x] = 0;
156                        }
157                    }
158                }
159            }
160        }
161    }
```

O último passo para que o mapa local possa ser utilizado pela inteligência desenvolvida por este estudo é o relaxamento da matriz executado pelo método `relaxamento` que é exibido no Quadro 28.

O processo de relaxamento do mapa local é praticamente idêntico ao do mapa global, sendo que a única diferença é a quantidade de iterações de relaxamento. Como o mapa local é menor que o mapa global, são necessárias menos iterações para a convergência da matriz. No caso desta implementação, a quantidade de iterações foi configurada em 30, que, segundo Dapper (2007, p. 41), é um valor padrão para gerar trajetórias em mapas de até 17x17.

Quadro 28 – Relaxamento do mapa local

```
47      private static final int ITERACOES_RELAXAMENTO = 30;
48      private void relaxamento() {
49          for (int k = 0; k < ITERACOES_RELAXAMENTO; k++) {
50              switch (k % 4) {
51                  case 0:
52                      //percorre matriz a partir do canto superior direito
53                      for (int y = 0; y < campoLocal.length; y++) {
54                          for (int x = 0; x < campoLocal[y].length; x++) {
55                              MapaGlobalPVCPreProcessado.calculaPotencial(campoLocal, y, x);
56                          }
57                      }
58                      break;
59                  case 1:
60                      //percorre matriz a partir do canto superior esquerdo
61                      for (int y = 0; y < campoLocal.length; y++) {
62                          for (int x = campoLocal[y].length - 1; x >= 0; x--) {
63                              MapaGlobalPVCPreProcessado.calculaPotencial(campoLocal, y, x);
64                          }
65                      }
66                      break;
67                  case 2:
68                      //percorre matriz a partir do canto inferior direito
69                      for (int y = campoLocal.length - 1; y >= 0; y--) {
70                          for (int x = 0; x < campoLocal[y].length; x++) {
71                              MapaGlobalPVCPreProcessado.calculaPotencial(campoLocal, y, x);
72                          }
73                      }
74                      break;
75                  case 3:
76                      //percorre matriz a partir do canto inferior esquerdo
77                      for (int y = campoLocal.length - 1; y >= 0; y--) {
78                          for (int x = campoLocal[y].length - 1; x >= 0; x--) {
79                              MapaGlobalPVCPreProcessado.calculaPotencial(campoLocal, y, x);
80                          }
81                      }
82                      break;
83              }
84          }
85      }
```

Após o relaxamento, o mapa local está pronto para que o jogador utilize este para decidir seu próximo passo, sendo esta a implementação que permite que o jogador se movimente utilizado campos potenciais.

3.3.3 Operacionalidade da implementação

Esta seção mostra os recursos do simulador de futebol priorizando a apresentação das novas funcionalidades desenvolvidas no presente estudo.

Na tela inicial, o usuário tem acesso ao menu `Cadastrar` (Figura 17) onde pode informar seus dados de acesso tais como nome, *e-mail*, data de nascimento, *login* e senha para criar uma nova conta e acessar o sistema.

Figura 17 – Cadastro de usuário

Após efetuar o cadastro e se autenticar no sistema, o usuário é apresentado à capa restrita (Figura 18) onde é exibida uma mensagem de boas-vindas no topo à direita e um menu de opções à esquerda.

Figura 18 – Capa restrita

Ao efetuar o cadastro, o usuário ganha automaticamente uma equipe padrão, que pode ser conferida e editada no menu `Equipe`. Na tela mostrada na Figura 19, o usuário pode editar equipes clicando sobre o seu nome, pode cadastrar novas equipes clicando no botão `Nova Equipe` e informando um nome para a mesma ou ainda pode excluir equipes existentes selecionado-as e clicando no botão `Excluir Equipes`.

Figura 19 – Tela de gerenciamento de equipes

Ao clicar para editar uma equipe a Figura 20 é exibida. Nela o usuário pode editar jogadores clicando em `Jogadores`, editar táticas clicando em `Táticas` ou ainda editar estratégias clicando em `Estratégias`.

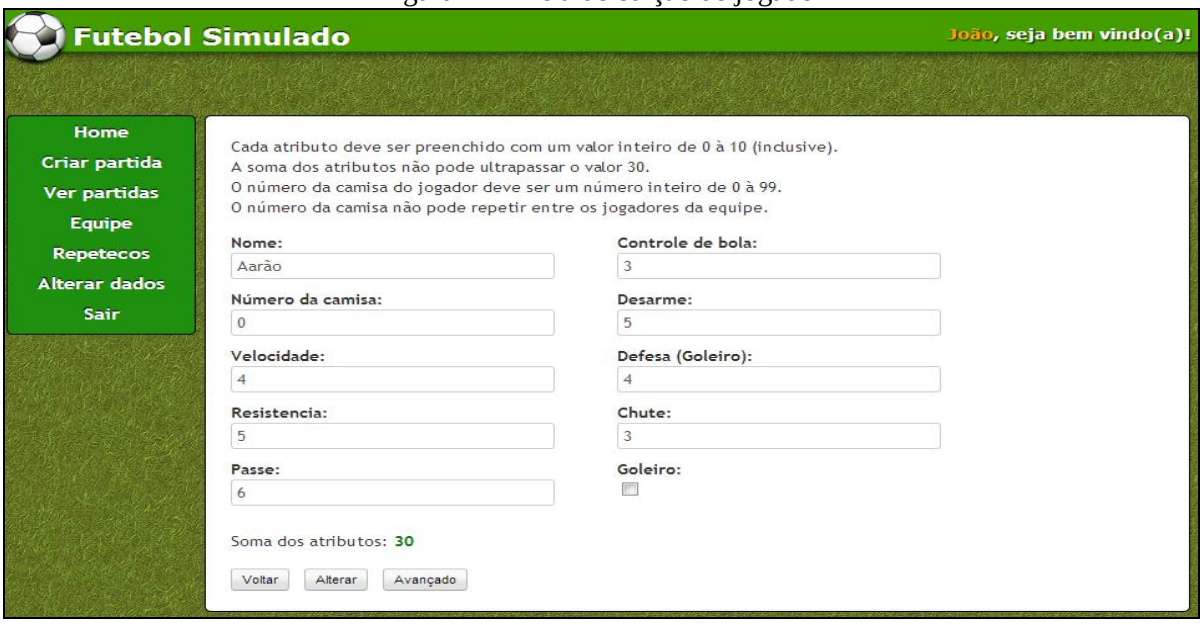
Figura 20 – Tela de edição de equipe

Os times criados no simulador têm todos os seus jogadores configurados com a inteligência baseada em campos potenciais, uma vez que esta foi definida como sendo a inteligência padrão. Caso o usuário deseje alterar essa inteligência, ele deve entrar na tela de edição de jogadores (Figura 21).

Figura 21 – Tela de edição de jogadores

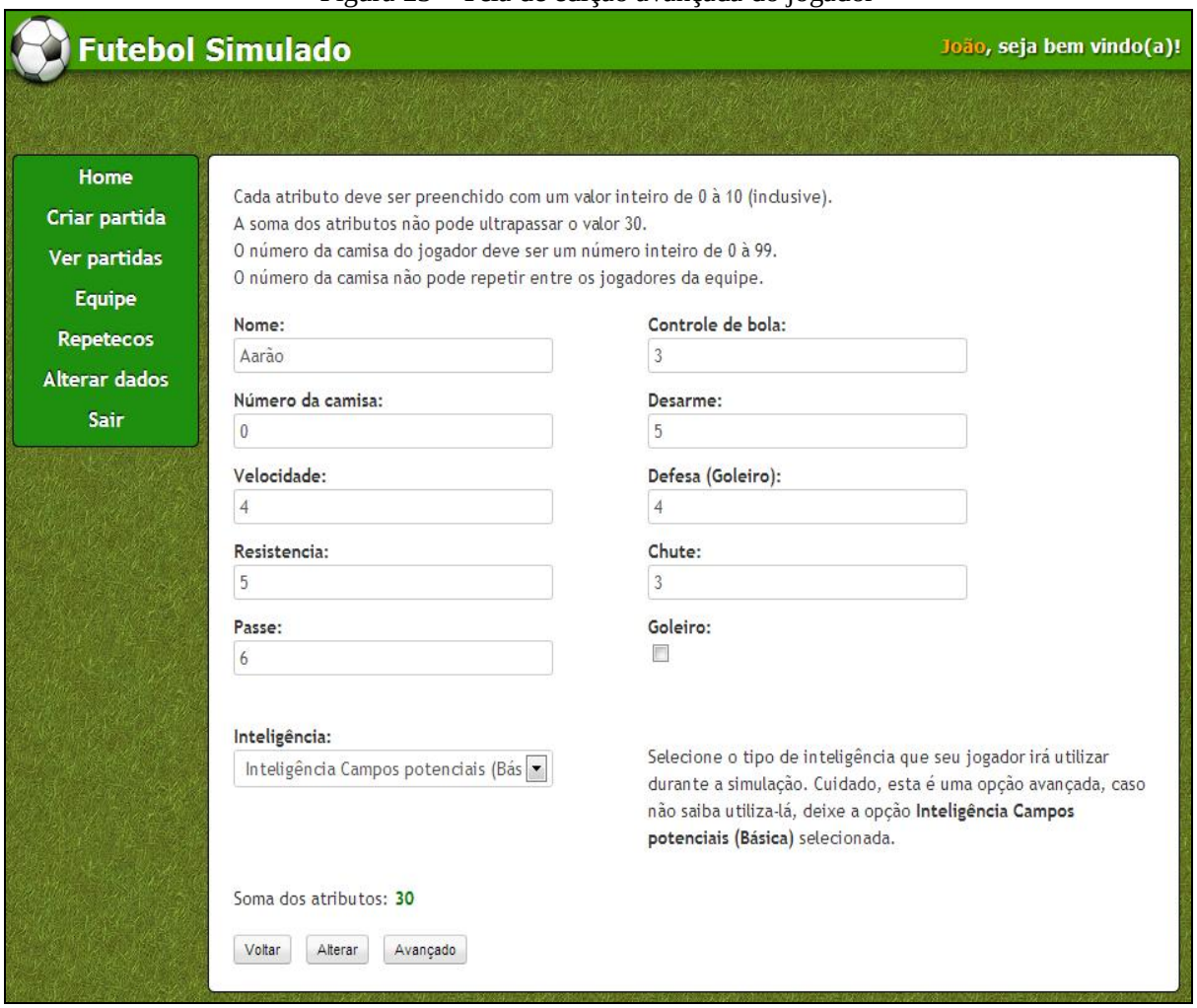
Na tela de edição de jogadores, o usuário pode editar um jogador clicando sobre o nome do mesmo, criar um jogador clicando no botão `Novo Jogador`, gerar um time clicando no botão `Gerador de Time` ou ainda excluir um ou mais jogadores, selecionando os jogadores que deseja excluir e clicando no botão `Excluir Jogadores`. Para editar a inteligência de um jogador, o usuário deve abrir a tela de edição de jogador (Figura 22).

Figura 22 – Tela de edição de jogador



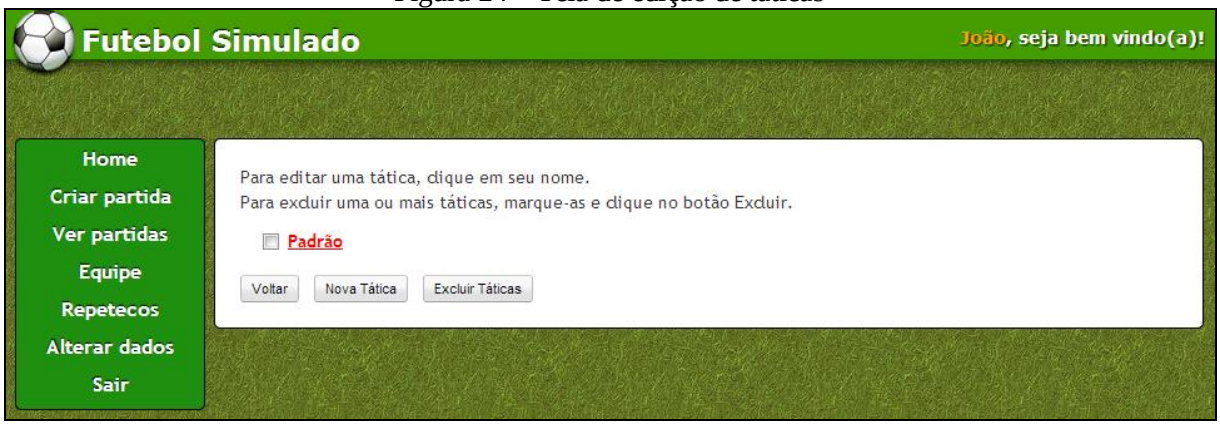
Nesta tela o usuário pode editar o nome, o número da camisa, as habilidades de cada jogador e indicar se o jogador é goleiro. A configuração de inteligência do jogador é uma opção avançada, portanto o usuário necessita clicar no botão `Avançado` para ter acesso a ela. Clicando neste botão a tela aparece conforme a Figura 23.

Figura 23 – Tela de edição avançada do jogador



Após editar as informações do jogador, o usuário pode clicar no botão `Alterar` para confirmar as alterações ou ainda cancelar as alterações clicando no botão `Voltar`. Após editar os jogadores da equipe, o usuário deve definir ao menos uma tática para seu time. A criação e edição de táticas está disponível através do menu `Táticas` da Figura 20. Ao acessar a edição de táticas a tela é apresentada conforme a Figura 24.

Figura 24 – Tela de edição de táticas



Nesta tela o usuário pode criar uma tática clicando no botão `Nova Tática`, excluir uma ou mais táticas selecionando-as e clicando no botão `Excluir Táticas` ou ainda editar uma tática clicando sobre seu nome. Ao clicar sobre o nome de uma tática é exibida a tela conforme a Figura 25.

Figura 25 – Tela de edição da tática

Nesta tela o usuário pode configurar o nome da tática, selecionar quais os 11 jogadores que farão parte dela e definir a posição de cada jogador em campo, preenchendo os campos `x` e `y` de cada jogador. Para facilitar o preenchimento das posições dos jogadores foi criado o recurso para que o usuário possa clicar no campo `x` ou `y` de um jogador e em seguida clicar sobre o campo de futebol na posição em que deseja posicioná-lo. Após finalizar a edição da

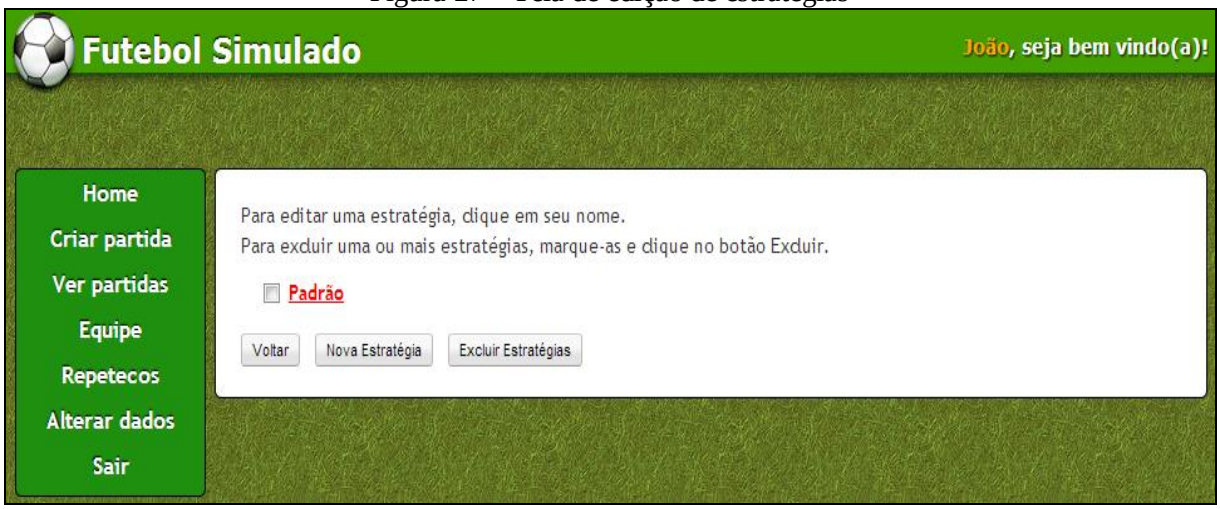
tática o usuário pode confirmar a alteração através do botão `Alterar` ou então cancelá-la através do botão `Voltar`.

Caso exista alguma inconsistência no posicionamento de algum jogador, ao confirmar a alteração é exibida uma mensagem informando ao usuário o que deve ser corrigido (Figura 26).

Figura 26 – Mensagem de inconsistência na edição de tática

Voltando à tela da equipe (Figura 17) o usuário tem a opção de definir estratégias para seu time. Ao clicar na palavra `Estratégias`, será exibida ao usuário a tela conforme a Figura 27. Nela o usuário pode criar uma nova estratégia clicando no botão `Nova Estratégia`, excluir estratégias selecionando-as e clicando no botão `Excluir Estratégias` ou ainda editar uma estratégia clicando sobre o nome desta.

Figura 27 – Tela de edição de estratégias

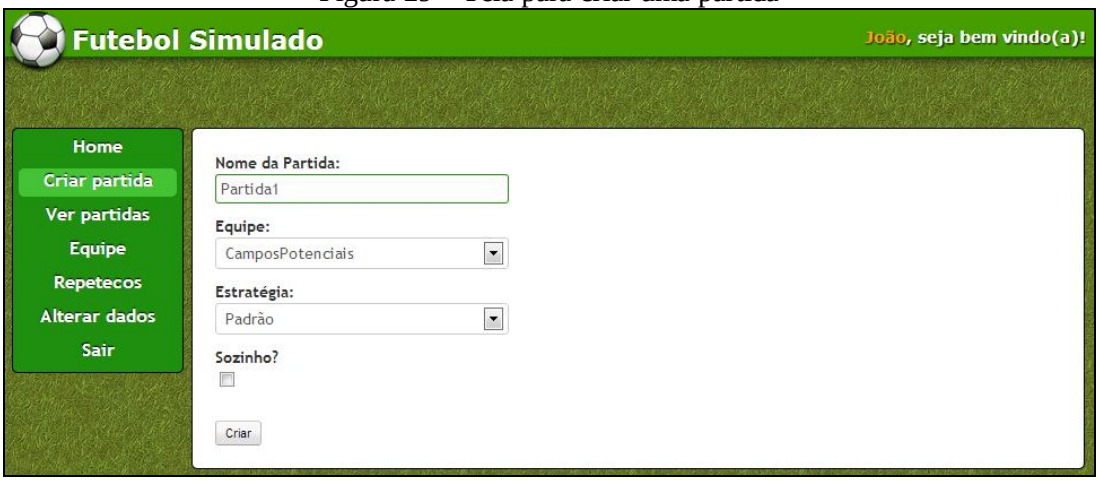


Ao criar ou editar uma estratégia, a tela apresentada na Figura 28 é exibida. Nesta tela pode-se definir uma ou mais táticas para serem utilizadas em momentos específicos do jogo, como por exemplo, cadastrar uma tática para ser executada durante o jogo enquanto o time estiver empatando ou ganhando. É possível alterar para outra tática quando o time estiver perdendo.

Figura 28 – Tela de edição da estratégia

Após o usuário ter montado uma equipe com os 11 jogadores, ter definido suas táticas e estratégias, o mesmo encontra-se apto a criar uma partida através do menu `Criar partida`, conforme Figura 29, onde deve-se informar um nome para a partida, a equipe e a estratégia a ser utilizada.

Figura 29 – Tela para criar uma partida

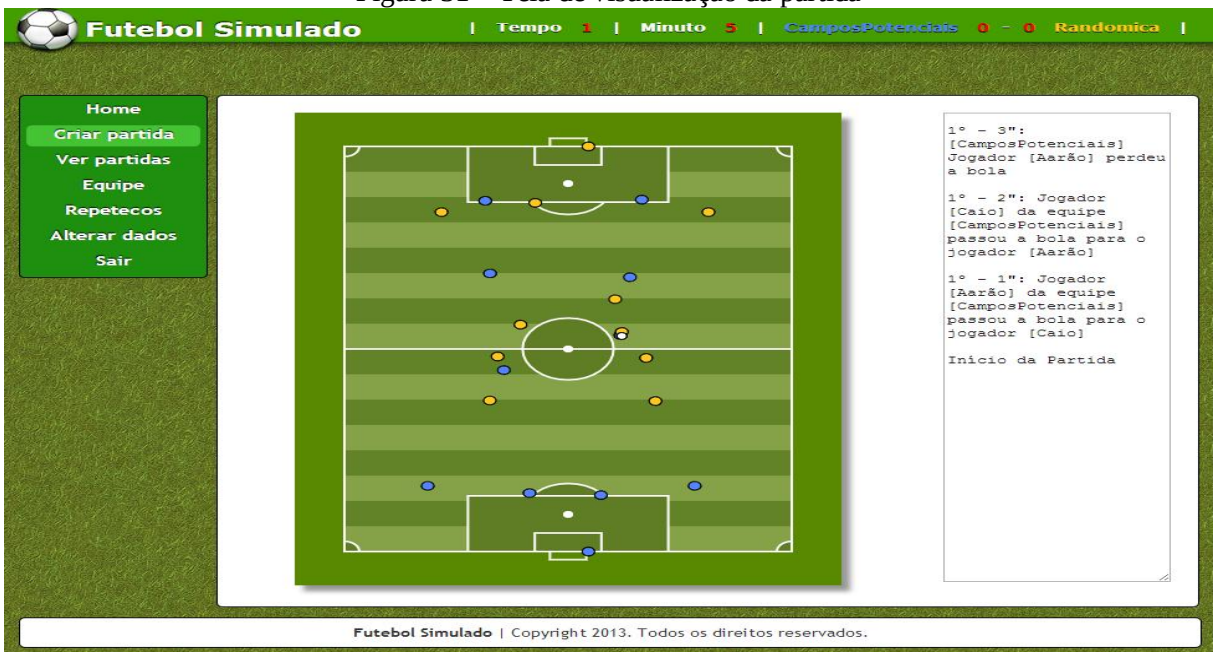


O usuário pode optar por realizar a simulação sozinho, sem a necessidade de um adversário. Para isso deve marcar o campo `Sozinho?` e será apresentada a tela conforme a Figura 30. Basta o usuário definir a equipe e a estratégia do adversário e clicar no botão `Criar`.

Figura 30 – Tela de uma partida individual

Ao iniciar a partida, será exibida a tela de visualização do jogo (Figura 31). Nela o usuário pode vizualizar a movimentação dos jogadores em campo, assim como visualizar a descrição textual dos eventos ocorridos durante o jogo, localizada no lado direito da tela.

Figura 31 – Tela de visualização da partida



Ao término da partida, no minuto 45 do segundo tempo, é exibida uma mensagem de fim de jogo para todos os espectadores (Figura 32). A partir deste momento a partida fica disponível para *replay* no menu `Repetecos` e o usuário pode visualizar partidas que estejam em execução através do menu `Ver Partidas`.

Figura 32- Tela de fim do jogo

3.4 RESULTADOS E DISCUSSÃO

No Quadro 29 encontra-se uma comparação entre os trabalhos correlatos e o trabalho desenvolvido.

Quadro 29 – Comparação os trabalhos correlatos e o desenvolvido

Característica / Trabalho	**Schulter (2007)**	**Rodrigues (2008)**	**Schleuss (2011)**	**Dapper (2007)**	**Ferrari (2011)**	**Mafra (2004)**	**Maas (2013)**
livre de mínimos locais	Não se aplica	Não se aplica	Não se aplica	Sim	Sim	Não	Sim
suporte a múltiplos agentes em uma única simulação	Sim	Sim	Sim	Sim	Não	Sim	Sim
suporte a desvio de obstáculos móveis	Não	Não	Não	Sim	Não se aplica	Sim	Sim
movimentação próxima ao mundo real	Não	Não	Não	Sim	Não	Não	Não

Conforme pode ser observado no Quadro 29, os trabalhos de Schulter (2007), Rodrigues (2008) e Schleuss (2011) não possuem suporte a desvio de obstáculos móveis, enquanto o presente trabalho possui. Em relação ao trabalho de Mafra (2004) o presente trabalho tem como diferencial ser livre de minímos locais, porém ao considerar o contexto do jogo de futebol e as opções que um jogador tem quando está cercado por adversários, esta é uma vantagem de menor relevância. A vantagem entre o presente trabalho e o de Ferrari (2009) é o suporte a múltiplos agentes em uma única simulação.

O principal objetivo do trabalho realizado era a movimentação próxima ao mundo real, que, entre os trabalhos correlatos, é suportado apenas pelo trabalho de Dapper (2007). A proposta era descentralizar a partida através dos campos potenciais, porém não foi possível obter esta característica.

Uma das causas detectadas para que o resultado esperado não tenha sido alcançado é a existência de apenas um objetivo para o jogador que está com a bola, o gol adversário. Esta característica faz com que o campo potencial gerado sempre leve o jogador para a posição zero (0) do eixo x, centralizando as ações do time. Isto já era esperado no início das pesquisas, porém existia a expectativa que, ao considerar os outros jogadores como obstáculos, a trajetória para o gol seria alterada, levando o jogador a passar por regiões do campo menos ocupadas. Porém, isso não aconteceu pois a força de repulsão dos jogadores não foi suficiente para alterar de forma significativa a trajetória do jogador com a bola e fazê-lo ocupar outros

espaços do campo. O jogador com a bola seguiu movimentando-se muito próximo aos outros jogadores, o que facilitou a aproximação dos adversários.

Outra causa para o não atendimento do objetivo proposto é o fato de que geralmente a aplicação de campos potenciais serve como solução para o problema de um agente que precisa chegar a um objetivo desviando de obstáculos estáticos ou dinâmicos. Entretanto, em um jogo de futebol o problema a ser resolvido é de um grupo de agentes que têm como objetivo chegar a um ponto de forma coletiva. Outra dificuldade é que no jogo de futebol os jogadores adversários (obstáculos dinâmicos) tem por objetivo não apenas bloquear a ação do jogador com a bola, mas também desarmá-lo, ou seja, os obstáculos não apenas disputam o espaço mas também o recurso (neste caso, a bola). Nos algoritmos estudados não foi observada a previsão de situações como esta.

Para demonstrar os resultados obtidos na implementação da inteligência utilizando campos potenciais, foram realizadas dez (10) partidas entre duas equipes idênticas, tendo como diferencial a inteligência utilizada pelos jogadores, que em uma equipe é a inteligência randômica e na outra é a baseada em campos potenciais. Os resultados das partidas são apresentados na Tabela 1.

Tabela 1 - Total dos resultados de partidas

RESULTADO PARTIDAS		
Partida	**Equipe Campos Potenciais**	**Equipe Randômica**
1	1	0
2	2	3
3	0	2
4	1	0
5	0	1
6	1	1
7	0	1
8	0	3
9	2	2
10	3	0
	10	13

Com base nos resultados demonstrados na Tabela 1, pode-se observar que o time com inteligência baseada em campos potenciais equiparou-se ao time com inteligência randômica. Das 10 partidas executadas o time randômico ganhou cinco e o time campos potenciais três. Ainda houve dois empates. No quesito quantidade de gols marcados a equipe randômica teve vantagem de apenas três gols, sendo que o placar com maior diferença de gols (três (3) a zero (0)) foi conquistado pelas duas equipes.

4 CONCLUSÕES

O objetivo principal deste trabalho que era o de disponibilizar uma inteligência baseada em campos potenciais para que os jogadores tivessem comportamentos parcialmente semelhantes aos do mundo real não foi atendido. A inteligência implementada consegue levar os jogadores ao gol adversário e consegue equilibrar partidas com equipes que utilizam inteligência randômica. Porém, as ações dos jogadores que utilizam a inteligência implementada são totalmente centralizadas. Isto ocorre porque o método utilizado para calcular os campos potenciais gera contornos muito suaves, fazendo o jogador com a bola movimentar-se muito próximo dos adversários o suficiente para que seja possível efetuar o desarme.

Os objetivos que tratavam das melhorias de usabilidade do simulador foram alcançados. A definição do posicionamento dos jogadores na edição de táticas através do ponteiro do *mouse* mostrou ser uma forma mais fácil e rápida de posicionar os jogadores em relação ao método anterior onde se definia o posicionamento digitando as coordenadas.

A descrição textual da partida facilitou o entendimento dos eventos que ocorrem durante o jogo. Principalmente nas situações em que a bola fica travada entre três ou mais jogadores, o usuário consegue perceber que estão ocorrendo os desarmes.

O estudo dos campos potencias mostrou que a técnica de mapa global e mapa local torna viável sua utilização para o simulador de futebol. Contudo, o presente estudo também concluiu que utilizar apenas dois mapas globais, um para cada gol do campo, não gera bons resultados, pois a movimentação dos jogadores fica muito centralizada. Uma possibilidade para melhorá-los seria criar um mapa global para cada jogador, ou seja, cada jogador teria um objetivo em campo. Como exemplo, podem ser citados os laterais que geralmente têm o objetivo de chegar à linha de fundo, realizando cruzamentos para a área.

Os desafios para implementar a sugestão anterior são o gerenciamento de memória para armazenar os mapas globais utilizados nas partidas em execução e o tempo necessário para efetuar o cálculo de um mapa global que, atualmente, leva uma média de 20 segundos por mapa. Considerando efetuar o cálculo dos campos potenciais no início de cada partida, a espera do usuário seria em média de sete minutos, tornando o método inviável.

4.1 EXTENSÕES

As sugestões para possíveis extensões desse trabalho são enumeradas abaixo:

a) implementar uma inteligência baseada em campos potenciais, onde cada jogador tenha o seu próprio mapa global, visando descentralizar as ações da partida;

b) revisar a inteligência do jogador com a bola não apenas em relação à movimentação, mas também quanto a passar a bola para um companheiro, chutar à gol ou driblar conforme as habilidades e as possibilidades de caminhos de cada jogador;

c) estudar outros métodos de movimentação que pemitam ao jogador com a bola se afastar dos adversários o suficiente para dificultar o desarme.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. **Maven**. [S.l.], 2011. Disponível em: <http://maven.apache.org/what-is-maven.html>. Acesso em: 07 out. 2013.

CONNOLLY, Christopher I.; BURNS, Brian J.; WEISS, Richard. Path planning using Laplace's equation. In: IEEE INTERNATION CONFERENCE ON ROBOTICS AND AUTOMATION, 7th, 1990, Cincinnati. **Proceedings…** Cincinnati: IEEE Computer Society Press, 1990. p. 2102-2106. Disponível em: <https://courses.cs.washington.edu/courses/cse599j/12sp/papers/Connolly.pdf >. Acesso em: 22 out. 2013.

DAPPER, Fábio. **Planejamento de movimento para pedestres utilizando campos potenciais**. 2007. 73 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/12219/000622643.pdf?sequence=1>. Acesso em: 7 abr. 2013.

FERRARI, Marcelo R. **Biblioteca para explorar algoritmos de planejamento de movimento utilizando campos potenciais no SDK do iOS 4**. 2011. 66 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2011/347313_1_1.pdf>. Acesso em: 8 abr. 2013.

FIFA. **Laws of the game**. Zurique, 2012. Disponível em: <http://pt.fifa.com/mm/document/footballdevelopment/refereeing/81/42/36/lawsofthegame_2012_e.pdf>. Acesso em: 07 abr. 2013.

FISCHER, Leonardo G. **Otimização de desempenho em planejadores de caminho usando campos potenciais**. 2008. 63 f. Trabalho de Graduação (Bacharelado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/16009/000681150.pdf?sequence=1>. Acesso em: 24 ago. 2013.

FRISSELI, Ariobaldo; MANTOVANI, Marcelo. **Futebol:** teoria e prática. São Paulo: Phorte, 1999.

GAMEGOL. **Simulador de futebol**. [S.1.], 2007. Disponível em: <http://www.gamegol.com.br>. Acesso em: 18 abr. 2013.

GOODRICH, Michael A. **Potential fields tutorial**. [S.1.], 2004. Disponível em: <http://students.cs.byu.edu/~cs470ta/goodrich/fall2004/lectures/Pfields.pdf >. Acesso em: 7 abr. 2013.

GOOGLE. **Google Chrome**. [S.l.], 2013. Disponível em: <http://www.google.com/chrome>. Acesso em: 08 nov. 2013.

HATTRICK. **Simulador de futebol**. [S.1.], 2007. Disponível em: <http://www.hattrick.org>. Acesso em: 18 abr. 2013.

JQUERY. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2007. Disponível em: <http://pt.wikipedia.org/wiki/JQuery>. Acesso em: 10 nov. 2013.

LATOMBE, Jean-Claude. **Motion planning:** a journey of robots, molecules, digital actors, and other artifacts. Stanford, 1999. Disponível em: <http://gamma.cs.unc.edu/courses/planning-f07/PAPERS/Motionp-Planning-Overview.pdf>. Acesso em: 7 abr. 2013.

MAFRA, Julio C. **Protótipo de formação em times de futebol de robôs utilizando robótica baseada em comportamento**. 2004. 59 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2004/306333_1_1.pdf>. Acesso em: 8 abr. 2013.

MANAGERZONE. **Game multiplayer online**. Suécia, 2007. Disponível em: <http://www.managerzone.com>. Acesso em: 18 abr. 2013.

MICROSOFT CORPORATION. **Windows Internet Explorer**. [S.l.], 2012. Disponível em: <http://windows.microsoft.com/pt-br/internet-explorer/ie-10-worldwide-languages>. Acesso em: 08 nov. 2013.

NIEUWENHUISEN, Dennis; KAMPHUIS, Arno; OVERMARS, Mark H. **High quality navigation in computer games**. Netherlands, 2006. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.933&rep=rep1&type=pdf>. Acesso em: 7 abr. 2013.

O2GAMES. **GameGol**: um manager de futebol on-line brasileiro. [S.l.], 2006. Disponível em: <http://www.programadoresdejogos.com/download/o2games_jjd2.pdf>. Acesso em: 03 nov. 2013.

RODRIGUES, Roberto R. **Simulador de futebol em ambiente web**. 2008. 80 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SCHLEUSS, Thyago. **Simulação de futebol em ambiente web:** versão 3.0. 2011. 102 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SCHULTER, Fábio. **Simulador de uma partida de futebol com robôs virtuais**. 2007. 90 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

WUOLIO, Jukka. **Futebol:** o jogo mais popular. Rio de Janeiro: Salvat, 1981.