UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

CHARLES ROBERTO GESSER

SISTEMA PARA CONTROLE DE ROBOCUP UTILIZANDO A LINGUAGEM JAVA E O FRAMEWORK JSBACH

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2017

CHARLES ROBERTO GESSER

SISTEMA PARA CONTROLE DE ROBOCUP UTILIZANDO A LINGUAGEM JAVA E O FRAMEWORK JSBACH

Trabalho de Conclusão de Curso, apresentado ao Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, campus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Andreia Scariot Beulke

PATO BRANCO
2017

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

TERMO DE APROVAÇÃO

SISTEMA PARA CONTROLE DE ROBOCUP UTILIZANDO A LINGUAGEM JAVA E O FRAMEWORK JSBACH

por

CHARLES ROBERTO GESSER

Este trabalho de conclusão de curso foi apresentado em 01 de março de 2018, como requisito parcial para a obtenção do título de Especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Andreia Scariot Beulke (Orientadora), Robison Cris Brito e Beatriz Terezinha Borsoi, membros da banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Andreia Scariot Beulke
Prof. Orientador (UTFPR)

_____
Robison Cris Brito
Banca (UTFPR)

_____
Beatriz Terezinha Borsoi
Banca (UTFPR)

_____
Robison Cris Brito
Coordenador da IV Especialização em Tecnologia Java

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

GESSER, Charles Roberto. 2017. Sistema para controle de RoboCup utilizando a linguagem Java e o framework JSBach. 26f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Neste trabalho foi utilizado o ambiente RoboCup para simular a troca de mensagens entre equipes de futebol. A aplicação desenvolvida foi um cliente que se comunica com o servidor e permite que sejam enviadas mensagens para os jogadores que são agentes de software. Assim, é possível trocar informações entre os jogadores e suas jogadas, utilizando o ambiente RoboCup que se conecta com o servidor por meio do protocolo UDP/IP usando *socket*. As ferramentas utilizadas para isso foram a SoccerMonitor e SoccerServer que permitem a visualização do jogo no servidor e a conexão entre as equipes. Para o desenvolvimento do trabalho foi utilizado a linguagem Java e o *framework* JSBach que facilita o processo de execução no ambiente RoboCup.

Palavras-chave: RoboCup. Framework JsBach. Linguagem Java.

ABSTRACT

GESSER, Charles Roberto. 2017. Sistema para controle de RoboCup utilizando a linguagem Java e o framework JSBach. 26f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

In this work the RoboCup environment was used to simulate the exchange of messages between soccer players. The application developed was a client that communicates with a server and allows messages to be sent to the players that are software agents. Thus, it is possible to exchange information between players and their movements using the RoboCup environment that connects to the server through the UDP/IP protocol using socket. The tools used were SoccerServer and SoccerMonitor that allow the visualization of the game on the server and the connection between the teams. Java language and the JSBach framework were used to develop the software. JSBach facilitates the execution of the software in the RoboCup environment.

Keywords: RoboCup. JsBach Framework. Java Language.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

IP	*Internet Protocol*
JVM	*Java Virtual Machine*
LISP	*List Processing*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UDP	*User Datagram Protocol*

SUMÁRIO

1 INTRODUÇÃO

A necessidade de comunicação entre as pessoas por meio do compartilhamento de informações e uso de recursos é cada vez mais evidente em diversas áreas, como, por exemplo, entretenimento, comércio, indústria, entre outros. Isso é possível pelo desenvolvimento da tecnologia que tem se apresentado de diversas maneiras em cada uma dessas áreas.

As redes de computadores permitem a comunicação e o compartilhamento de recursos e informações por meio de protocolos que estão organizados em camadas que atendem às funções específicas. Os protocolos de rede atendem as necessidades de ambientes heterogêneos garantindo a confiabilidade e a eficiência na troca de informações (COSTA, 2008). Dentre os protocolos mais conhecidos estão o *Transmission Control Protocol/Internet Protocol* (TCP/IP) e o *User Datagram Protocol* (UDP). Esses modelos são responsáveis por enviar pacotes a um determinado endereço *Internet Protocol* (IP) por meio da rede (MOSHARRAF; FOROUZAN, 2013).

Aliada à expansão da comunicação em decorrência das redes de computadores e dos seus meios como a Internet estão as necessidades e os avanços em tecnologias de automação. São periféricos e equipamentos que se propõem a cada vez mais realizar atividades para e pelos seres humanos. É a robótica que já começa a estar presente nos mais diversos campos de atuação humana: do entretenimento, a exploração espacial, passando pela medicina e pela automação de chão de fábrica.

Uma forma de implementação de robótica em software é o uso de simuladores como os aplicativos de jogos de futebol. A liga RoboCup, por exemplo, tem como objetivo simular jogos entre duas equipes distintas. Para que as equipes estejam centradas nas jogadas e para dar mais autonomia aos seus jogadores, a RoboCup disponibiliza ferramentas como SoccerServer e SoccerMonitor para a implementação de software.

Esse trabalho consiste em usar *socket* em Java para exemplificar a troca de informações entre jogadores e suas jogadas, usando o ambiente RoboCup que tem como regra conectar com o servidor usando o protocolo UDP. Assim, o objetivo deste trabalho é criar um cliente via *socket* para comunicar-se com o SoccerServer.

2 MATERIAIS E PROCEDIMENTOS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas no desenvolvimento do aplicativo.

Ferramenta / Tecnologia	Versão	Disponível em	Aplicação
Soccer Monitor	15.0.0	https://sourceforge.net/projects/sserver/files/rcssmonitor/	Visualização do jogo no servidor
Soccer Server	15.3.0	https://sourceforge.net/projects/sserver/files/rcssserver/	Servidor de troca de mensagens
JSBach			*Framework* para auxílio ao desenvolvimento RoboCup
Java	1.8	http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html	Linguagem de programação

Quadro 1 - Tecnologias e ferramentas utilizadas na modelagem e na implementação do aplicativo

2.1.1 Soccer Monitor

A ferramenta Soccer Monitor é utilizada para visualização do jogo no servidor. Uma ferramenta com essa finalidade é necessária porque o servidor não dispõe de interface gráfica. O Monitor recebe as informações vindas do servidor e desenha na tela a interação entre os times. Com essa ferramenta é possível observar o que está acontecendo no campo e como os agentes estão interagindo. A Figura 1 apresenta a interface gráfica do Soccer Monitor.

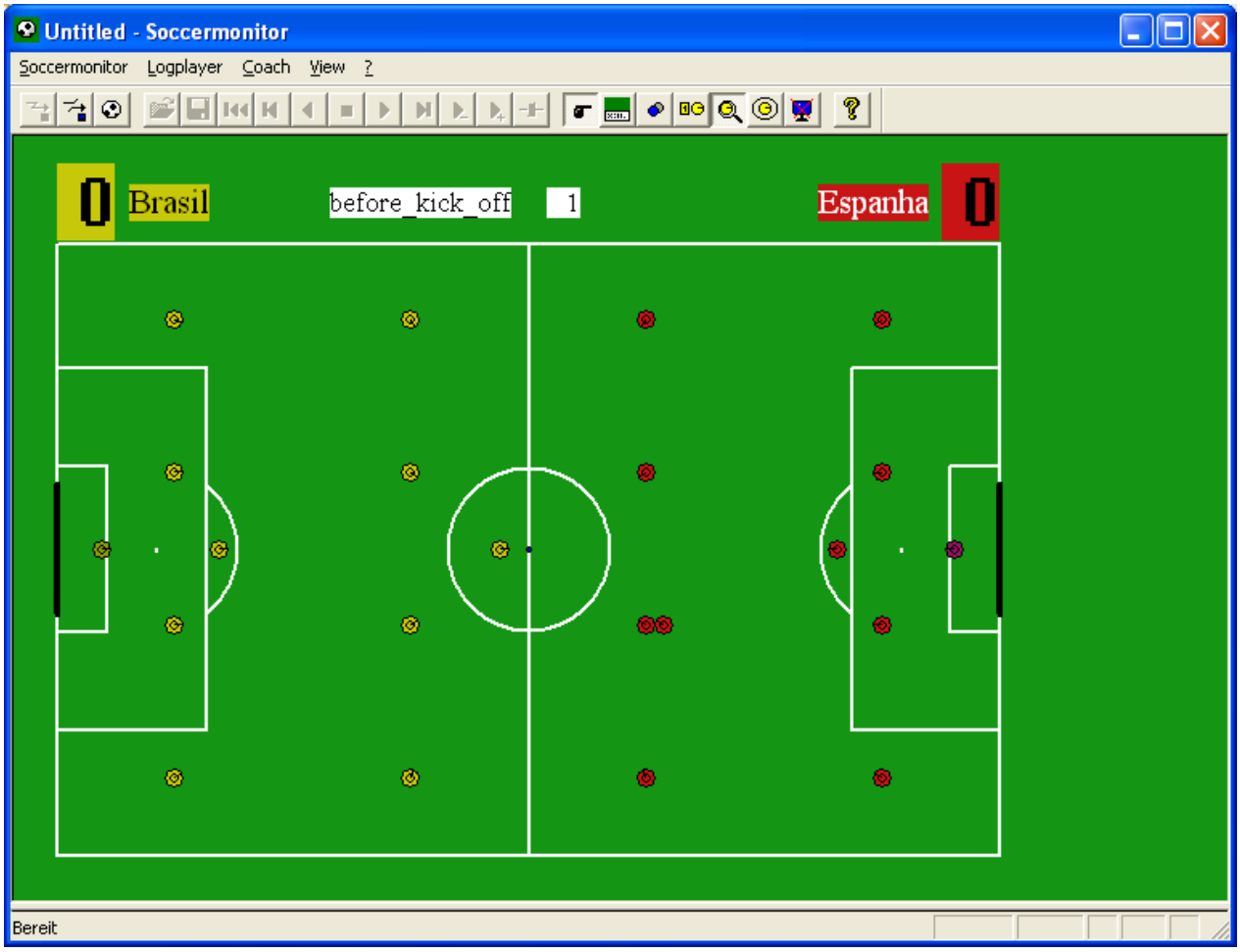

Figura 1 - Tela do RoboCup Monitor

2.1.2 Soccer Server

A ferramenta Soccer Server é um sistema que permite a conexão de duas equipes em um jogo virtual de futebol. A conexão é cliente-servidor via protocolo UDP/IP usando *socket*. A Figura 2 mostra a imagem da tela do servidor Server Soccer.

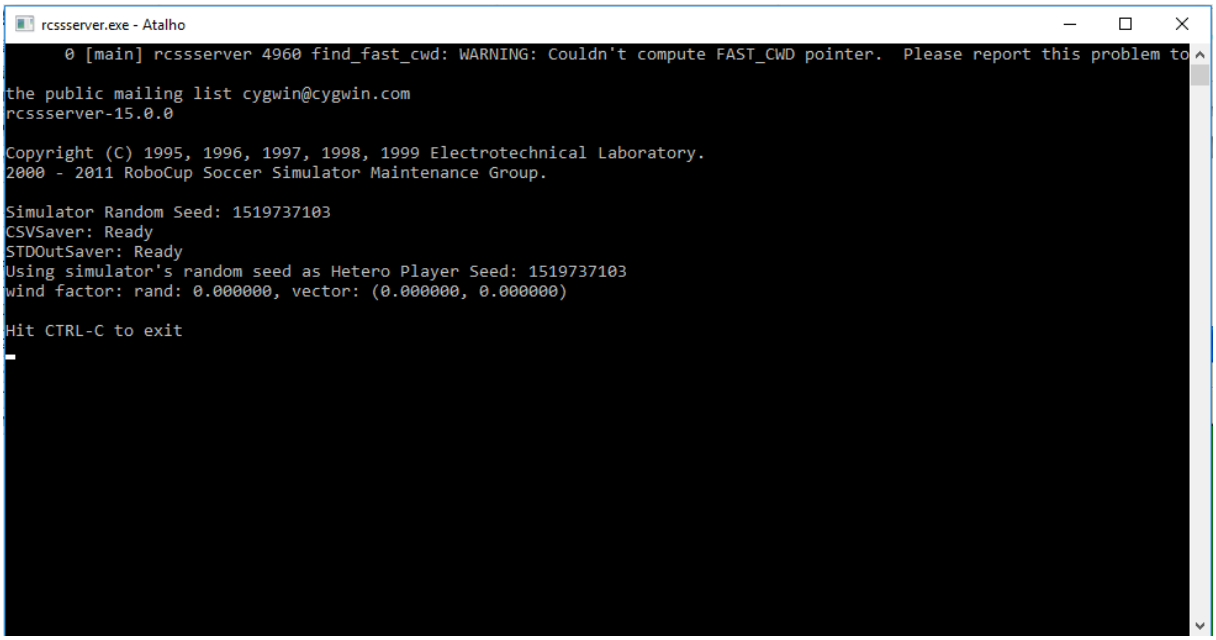

Figura 2 – Tela do Soccer Server

Depois de conectado ao servidor usando a porta 6000, cada jogador recebe outra porta para que seja efetuada a sua comunicação. Cada equipe pode conectar 12 clientes, 10 jogadores, 1 goleiro e um técnico. Os jogadores enviam pedidos ao servidor em relação às ações que querem executar (chutar, movimentar, virar, etc.) o servidor recebe os pedidos, executa e atualiza o monitor. Além disso, o servidor envia, no padrão LISP (*LIST PROCESSING*), informações sensoriais como o que o jogador está vendo ou sua resistência a cada 120 ms como mostra na Figura 3.

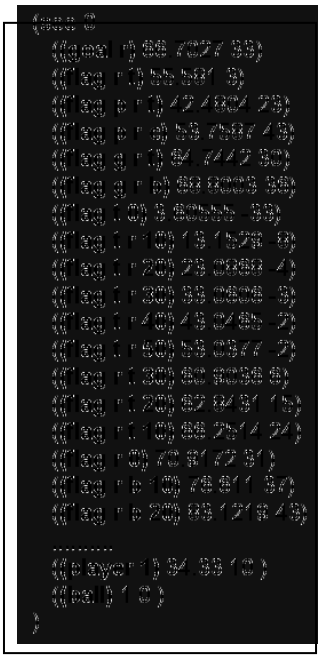

Figura 3 – Modelo LISP

2.1.3 JSBash

Os comandos disponibilizados pelo RoboCup, tanto para executar ações quanto para recuperar as percepções, são limitados, não é possível executar de forma direta comandos como “chutar para o gol”, ou então saber “se existe adversário a frente”. Para facilitar o processo de execução no ambiente RoboCup, foi utilizado um *framework* português chamado JSBach. Essa ferramenta foi desenvolvida em Java e implementa as seguintes ações:

a) **vira_para_bola** - vira o jogador na direção da bola;
b) **vai_para_bola** - avança em direção a bola;
c) **alivia_bola** - chuta a bola para um ponto intermédio do meio do campo adversário;
d) **vira_para_posicao_base** - vira na direção da sua posição de base do jogador. A posição base é o local onde o jogador deve se encontrar em determinada situação (ataque, meio de campo, defesa);
e) **vai_para_posicao_base** - avança para a sua posição de base;
f) **procura_bola** - procura a bola girando 360 graus;
g) **acompanha_bola** - acompanha a bola seguindo o seu movimento apenas rodando sobre si próprio;
h) **passa_bola** - passa a bola ao colega mais perto que está a sua frente;
i) **avanca_com_bola** - chuta a bola um pouco para a frente;
j) **centra_bola** - chuta a bola para um ponto central da grande área do adversário;
k) **remata_bola** - chuta a bola para a baliza adversária;
l) **vai_para_campo** - vai para dentro do campo fazendo um *dash*.

E estão disponíveis as seguintes percepções:

a) **ve_bola** - *boolean*; *true* se o agente vê a bola;
b) **bola_zona_accao** - *boolean*; *true* se a bola está próxima ao jogador;
c) **bola_dir** - int; devolve a direção da bola em relação ao jogador;
d) **alinhado_bola** - *boolean*; *true* se o jogador está alinhado com a bola;
e) **bola_perto** - *boolean*; *true* se a bola está a menos de 1,5 metros;
f) **posicao_base** - *boolean*; *true* se o jogador está a menos de 4 metros da sua posição de base;
g) **alinhado_posicao_base** - *boolean*; *true* se o jogador está alinhado com a sua posição de base;

h) **adversario_frente** - *boolean*; *true* se o jogador está vendo um adversário na sua frente;

i) **colega_frente** - *boolean*; *true* se o jogador está vendo um colega na sua frente;

j) **alinhado_zona_accao** - *boolean*; *true* se o jogador está alinhado com a sua zona de ação;

k) **estado_equipa** - int; devolve o estado da equipe, se elaa se encontra no ataque(0), na defesa(1) ou no meio de campo(2);

l) **funcao_jogador** - int; devolve a função do jogador dependendo do estado da equipe, a função pode ser defender(0), atacar(1) ou roubar a bola(2);

m) **zona_accao** - *boolean*; *true* se o jogador está na sua zona de ação;

n) **ve_objectos** - *boolean*; *true* se o jogador está vendo pelo menos um objeto do campo;

o) **fora_do_campo** - *boolean*; *true* se o jogador está fora das quatro linhas do campo.

Essa ferramenta permitiu que o foco do trabalho ficasse em conexão UDP com o servidor.

2.1.4 Java

Java é uma linguagem de programação orientada a objetos criada em 1995 por uma equipe de programadores chefiada por James Gosling na empresa Sun Microsystems. Esta linguagem é compilada para um *bytecode* e interpretada pela *Java Virtual Machine* (JVM), isso permite que seja criado um código e aproveitado em diferentes sistemas operacionais (DEITEL; DEITEL, 2010).

A linguagem Java é utilizada em aplicações *desktop*, *mobile*, sistemas embarcados e *web*.

2.1.5 Arquitetura e Funcionamento do RoboCup

Na categoria simulação, os comandos da arquitetura *RoboCup* são executados por meio do envio de mensagens para o servidor. Essas mensagens devem respeitar um protocolo de comunicação (ROBOCUP, 2017). Tanto as mensagens enviadas ao servidor, como as recebidas são no formato texto.

Um agente (transmissor) pode se comunicar com outros (receptores), enviando ao servidor a *String "say"* e a mensagem desejada. O servidor, por sua vez, transmitirá uma mensagem para todos os agentes próximos do agente transmissor. Essa mensagem será formada pela *String "hear"*, concatenada com a mensagem.

Além do comando "*say*", utilizado para a comunicação, os agentes podem enviar outros comandos para o servidor:

a) ***catch***: utilizado pelo goleiro para segurar a bola;

b) ***change_view***: altera o modo de visualização do jogador, o jogador pode visualizar o campo de maneira mais ampla, porém com poucos detalhes, ou olhar para um ponto específico com bastantes detalhes;

c) ***dash***: faz com que o jogador se movimente para frente, é passado por parâmetro se o jogador irá andar ou correr;

d) ***move***: movimenta o jogador para uma posição específica dentro do campo, esse comando só é valido se o jogo ainda não começou;

e) ***sense_body***: o jogador que enviou essa mensagem recebe do servidor informações como: modo de visualização, se está cansado ou não, a velocidade em que está se movimentando, posição no campo, entre outros;

f) ***score***: o servidor irá retornar o placar do jogo;

g) ***turn***: faz o jogador girar sobre seu próprio eixo;

h) ***turn_neck***: faz com que o jogador gire apenas a cabeça, a ponto de visualizar outro ponto do campo.

2.2 PROCEDIMENTOS E CONFIGURAÇÕES

Para o jogador localizar-se no campo são utilizadas marcas de localização (*flags*). Essas *flags* são apresentadas na Figura 4.

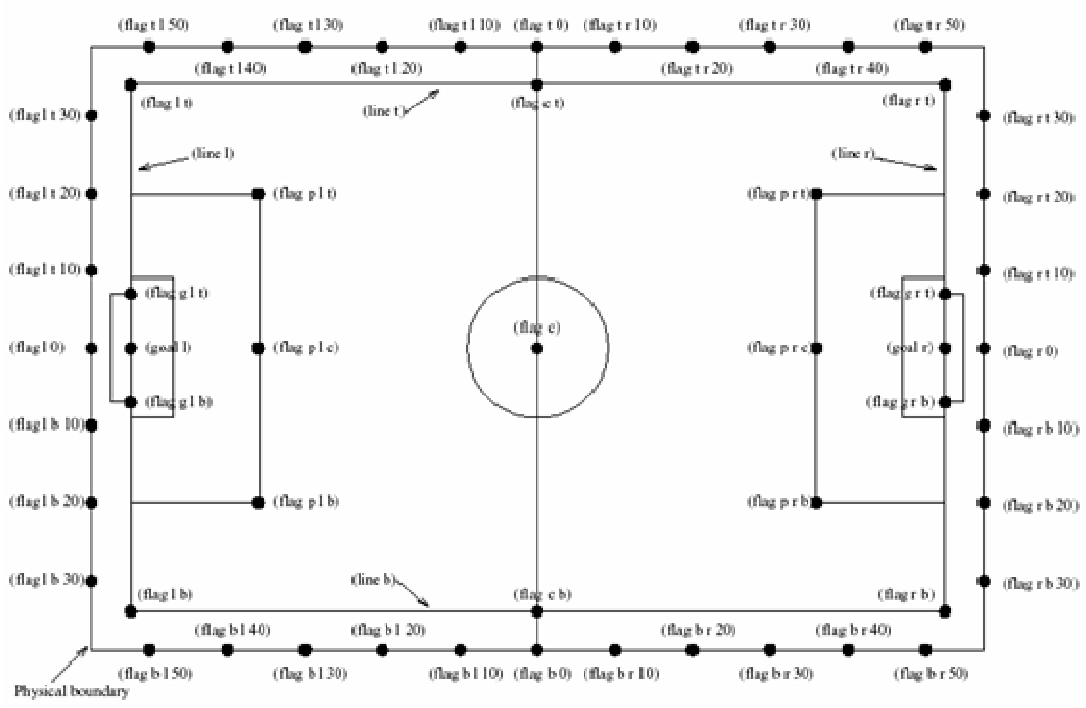

Figura 4 - Campo de futebol, marcado por *flags*

Como pode ser observado na Figura 4, o campo é formado por várias *flags*, entre elas se encontram a *flag* de centro (c), *flags* do lado direito (r), do lado esquerdo (l), inferiores (b), superiores (t), *flags* de pênalti (p) e *flags* de gol (g). Essas *flags* podem ser combinadas, exemplo a *flag* prb representa a marca de pênalti (p) inferior (b) do lado direito (r).

O jogador recebe constantemente mensagens do servidor, que inicia com *see* e apresenta todas as *flags* que o jogador está visualizando, bem como a distância dessa *flag* e o deslocamento (em graus) em relação ao jogador.

Além das *flags* estáticas no campo, o jogador também recebe informações sobre a posição dos outros jogadores (*player*) e a posição da bola (*ball*). Isso ocorre se elas estiverem no campo de visão do jogador. Como se tratam de objetos dinâmicos, além da distância e do deslocamento, são enviadas a velocidade do objeto e a direção do movimento (em graus).

A Figura 5 apresenta um modelo de mensagem enviado pelo servidor ao jogador. As mensagens são estruturadas no padrão LISP e deve ser processada pelo jogador.

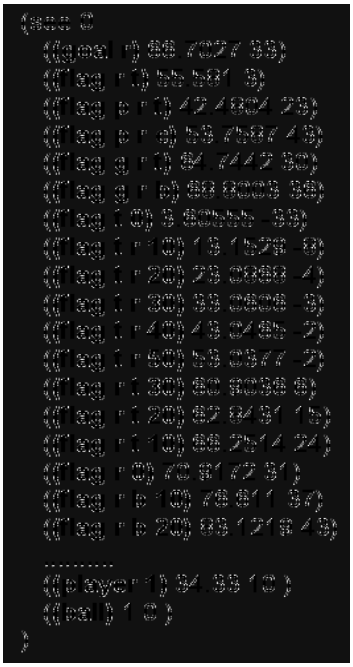

Figura 5 - Exemplo de mensagem enviada pelo servidor ao jogador

3 RESULTADOS

Este capítulo apresenta o resultado da realização do trabalho. O capítulo inicia com a descrição do escopo, seguido da apresentação e implementação do sistema.

3.1 ESCOPO DO SISTEMA

Na Figura 6, que mostra a estrutura do projeto desenvolvido, pode-se observar o time A e o time B que enviam informações ao *soccer server* por meio de *socket* utilizando o protocolo UDP. O servidor as recebe e executa o comando e, em seguida, atualiza o *soccer monitor* que tem a função de mostrar o posicionamento e a movimentação dos jogadores. O servidor envia informações a cada 150 ms aos jogadores usando o padrão LISP, para que sejam tomadas as decisões necessárias e prosseguir a jogada ou mudar a estratégia.

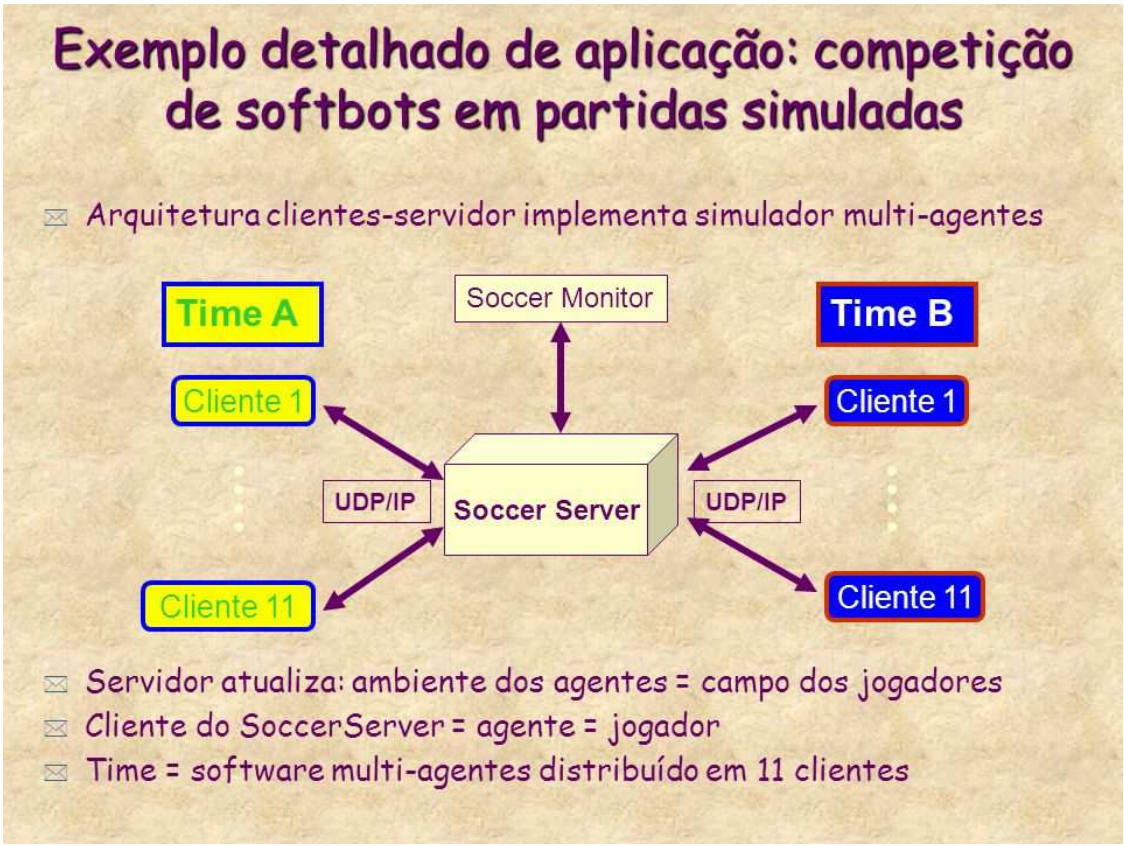

Figura 6 - Visão geral da aplicação e interação entre cliente e servidor

Destaca-se que *socket* é um ponto de comunicação entre duas máquinas a fim de enviar mensagem. E o UDP é um protocolo de comunicação simples e rápido. Contudo, esse

protocolo não fornece garantia de chegada da mensagem, pois apenas envia a mensagem e não se preocupa com a resposta de chegada.

3.2 APRESENTAÇÃO DO SISTEMA

O cliente é uma tela construída em Java usando componentes da paleta Swing. Essa tela tem por objetivo apresentar as informações do servidor e enviar comando para os jogadores. Outra função da tela é inicializar a classe JSBach01. Essa classe vai receber por parâmetro um vetor com informações de *host*, porta para se conectar e nome do time. Também possui comandos básicos dos jogadores. Essa classe que é responsável por instanciar os jogadores.

O cliente *socket* UDP mostra a comunicação com o servidor e são apresentados os comandos básicos dos jogadores e as mensagens do servidor. A Figura 7 mostra a tela desenvolvida que representa o cliente UDP.

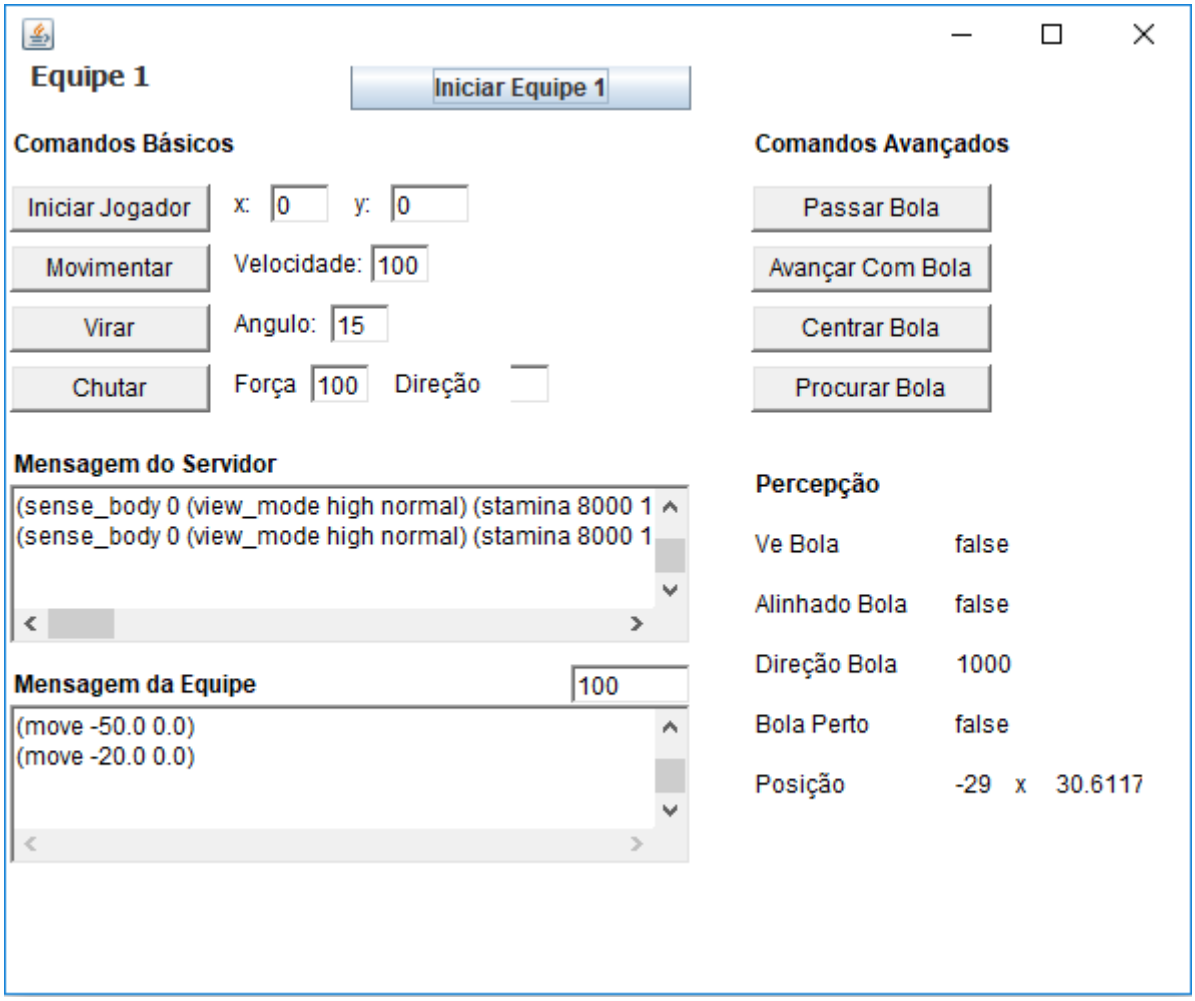

Figura 7 – Interface desenvolvida

Primeiro devemos iniciar o *Soccer Server* em seguida o *Soccer Monitor* e só então o cliente. Para iniciar o cliente, deve-se usar a porta 6000 e o servidor espera um cabeçalho padrão com 3 parâmetros, *"(init Brasil (version 4))",* onde o init é o método do servidor que registra o jogador, o segundo parâmetro é o nome do time e o terceiro parâmetro é a versão que esta sendo usada.

Os jogadores são iniciados de um a um, poderia ter 11 telas diferentes com os mesmos parâmetros que o servidor entenderia a que time o jogador pertence e o cadastraria corretamente, isso permite que cada cliente tenha sua própria porta para efetuar a comunicação com o servidor.

Depois de cadastrado o servidor envia as mensagens de posicionamento, de ângulo de visão, de movimentação etc, para cada jogador de acordo com a porta dedicada a ele quando entrou. Essa mensagem é encaminhada a cada 150 ms e cabe ao cliente como vai ser aproveitada. A partir deste momento o framework *JSBach* entra em ação. Sua função é receber parâmetros do jogador comparar com as informações dos colegas e efetuar cálculos de acordo com as necessidades.

Depois que todos os jogadores entrarem em campo a partida pode ser iniciada com o comando *KickOff*.

Para passar a bola o cliente teria que informar um chute, a força e teria que colocar o jogador no ângulo correto, com o uso do framework *JSBach* isso não é necessário, podemos apenas chamar o método *passa_bola* que todos os cálculos ficam abstraídos do cliente. Além deste método existem outros que facilitam muito com o método *procura_bola* que faz um giro de 360 graus para achar a bola.

A tela apresentada na Figura 7 se divide em 4 partes. Na primeira, estão os comandos básicos do *RoboCup* que permitem iniciar, movimentar um jogador e virar um jogador, além de chutar a bola. A segunda é a parte das mensagens que o servidor envia para o cliente e a equipe. A terceira e quarta partes são os comandos e as informações avançados disponíveis na biblioteca JSBach.

O uso desta biblioteca enriqueceu as informações disponíveis para o cliente, pois sua principal função é receber dados e transformá-los em informações úteis para a tomada de decisão ou usar funções prontas como passar, avançar, centrar ou procurar a bola. Com ela, o jogador pode saber em que parte do campo e em que direção está a bola e, ainda, se o jogador está próximo ou alinhado com a bola.

3.3 IMPLEMENTAÇÃO DO SISTEMA

As classes implementadas são apresentadas de acordo com a ordem de fluxo do cliente.

a) **ControleJogador.java:** é a classe responsável por fazer a interação do usuário com o jogador, nela são definidos os parâmetros para que a mensagem chegue no servidor, IP do servidor, porta de comunicação e nome do time. Também pode ser definida a posição inicial do jogador além de possuir alguns comandos básicos e informação do jogador.

b) **JSBach01.java:** essa classe instancia o jogador e envia os parâmetros de conexão.

c) **Jogador.java:** no construtor desta classe é aberto o *socket* para a comunicação com o servidor. O jogador estende da classe Thread, o objeto criado chama o método *run* para iniciar a comunicação. Nesta classe estão todos os métodos do servidor como *move, turn, dash* etc.

d) **Brain.java:** essa classe é responsável por disponibilizar os comandos da biblioteca JSBach.

e) **SendComand.java:** é a uma classe de interface gerada para garantir que todas as outras classes que a implementarem terão os métodos básicos do RoboCup.

f) **CalcPos.java:** essa classe é responsável por calcular as coordenadas do jogador de acordo com as *flags* disponibilizada pelo servidor.

g) **FlagInfo.java:** classe responsável por disponibilizar informações das *flags*.

h) **GoalInf.java:** classe responsável por informar posição do gol.

i) **LineInf.java:** classe responsável por informar a direção.

j) **SenseInf.java:** classe responsável por informações sensoriais.

k) **VisualInf.java:** classe responsável por analisar informações visuais e corporais do servidor.

As outras classes são usadas para gerar autonomia (Inteligência Artificial) aos jogadores ou complementas as classes citadas acima. A Figura 8 mostra a estrutura completa do cliente.

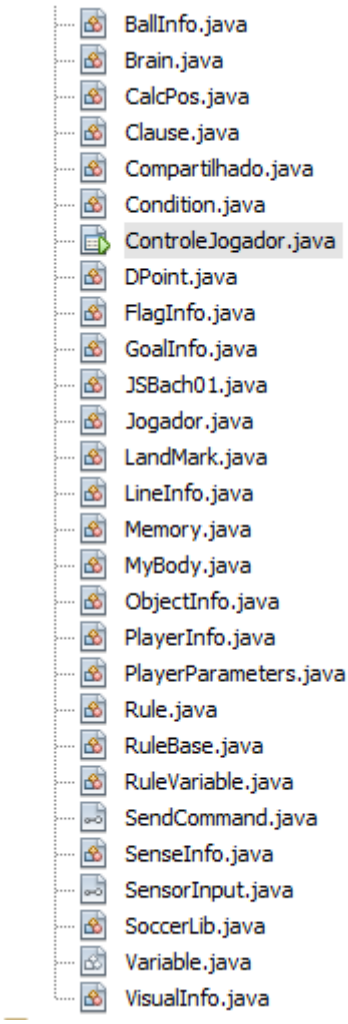

Figura 8 – Estrutura de classe do cliente

3.3.1 Funcionamento do cliente

A classe jogador estende da classe Thread para que possam ser enviados comandos e recebidos parâmetros de forma independente aos outros jogadores. Na Figura 9 pode-se observar que o método *send* é responsável por instanciar o DatagramPacket que utiliza um construtor que recebe como parâmetro um *buffer* (mensagem a ser transmitida), o tamanho do *buffer*, o IP onde se encontra o *soccer server* e a porta de comunicação que o servidor está escutando.

```
    // Esta função envia via mensagem de soquete para o servidor
    private void send(String message) {
        byte[] buffer = new byte[MSG_SIZE];
        buffer = message.getBytes();
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length, m_host, m_port);
//*****************************************************************************************
        try {
            m_socket.send(packet);
        } catch (IOException e) {
            System.err.println("socket sending error " + e);
        }
    }
```

Figura 9 – Método que envia mensagem para o servidor

A Figura 10 mostra um trecho de código no qual o cliente recebe a mensagem do servidor. Neste código também é utilizado o método DatagraPacket, porém com outro construtor, neste caso os parâmetros são apenas o buffer que receberá a mensagem e o tamanho do *buffer*, os parâmetros como IP e porta não são necessário neste momento.

```
    // Esta função aguarda nova mensagem do servidor
    private String receive() {
        byte[] buffer = new byte[MSG_SIZE];
        String aux1;
        DatagramPacket packet = new DatagramPacket(buffer, MSG_SIZE);
        try {
            m_socket.receive(packet);
        } catch (IOException e) {
            System.err.println("socket receiving error " + e);
        }
        aux1 = new String(buffer);

        return aux1;
    }
```

Figura 10 – Método que recebe a mensagem do servidor

As mensagens recebidas do servidor são analisadas e traduzidas pela biblioteca JSBash e mostradas no cliente UDP. Um exemplo de método é apresentado na Figura 11 que tem como objetivo saber onde está a bola e deslocar o jogador até ela.

Nessa função, a biblioteca busca informações de posicionamento da bola como distância e direção, armazena na variável obj1. Em seguida chama a função getXY responsável por calcular o novo posicionamento e direção a ser tomada. Logo após armazenar na variável p1 é chamado o método dashXY que informará a velocidade e a direção necessárias, armazena na variável aux1 e chama o método *dash* enviando-a como parâmetro. Esse método vai efetuar o *send* com o servidor que recebe e executa o deslocamento do jogador.

```
private void vai_para_bola(){
        ObjectInfo objl = m_memory.getObject("ball");
        if(objl != null){
                DPoint pl = SoccerLib.getXY(objl.m_distance,objl.m_direction,m_body)
                double auxl = SoccerLib.dashXY(pl.x,pl.y,m_body);
                m_debug.println("dash "+auxl);
                m_jogador.dash(auxl);
        }
}
```

Figura 11 – Método vai_para_bola

4 CONCLUSÃO

Neste trabalho foi utilizado *socket* em Java para exemplificar a troca de informações entre jogadores de futebol e suas jogadas. Para isso, foi utilizado o ambiente RoboCup que tem como objetivo simular jogos entre duas equipes distintas por meio de ferramentas como SoccerServer e SoccerMonitor. Portanto, o objetivo principal deste trabalho foi criar o cliente via *socket* para se comunicar o servidor.

Para facilitar o processo de execução no ambiente RoboCup foi utilizado o *framework* JSBach que possui ações para os jogadores, como, por exemplo, visualizar a bola, virar o jogador, avançar posição e procurar a bola. Assim, o uso desse *framework* enriqueceu as informações disponíveis para o cliente, pois sua principal função é receber dados e transformá-los em informações úteis para a tomada de decisão. Com essa ferramenta, houve um aumento significativo de produtividade no desenvolvimento de jogadas, pois ela encapsula os métodos mais utilizados dentro do ambiente RoboCup.

A linguagem Java foi utilizada para criar o cliente, que é uma interface que apresenta as informações vindas do servidor e envia comandos para os jogadores.

Com o desenvolvimento deste trabalho foi possível adquirir maior experiência e conhecimento na linguagem Java e conhecer um pouco sobre o ambiente e tecnologias possíveis para trabalhar com RoboCup. Além disso, neste trabalho, é possível visualizar, apenas, um exemplo de comunicação entre cliente e servidor, permitindo o envio de mensagens para os jogadores. Contudo, é possível desenvolver sistemas complexos que utilizam Inteligência Artificial, como Sistemas Multiagentes, por exemplo.

REFERÊNCIAS

COSTA, Daniel Gouveia. **Java em rede recursos avançados de programação**. Rio de Janeiro: Brasport, 2007.

DEITEL, Paul; DEITEL, Harvey. **Java**: como Programar. 8 ed. São Paulo: Pearson, 2010.

FOROUZAN, Behrouz A.; MOSHARRAF, Firoz. **Redes de computadores**: uma abordagem top-down. Porto Alegre: AMGH, 2013.

ROBOCUP. **The RoboCup Federation**. Disponível em: <www.RoboCup.org>. Acesso em: 23 ago. 2017.