SOLUÇÕES DESENVOLVIDAS PELA EQUIPE IBOTS/UFT 2013

ALEXANDRE T. R. DA SILVA[1], LUIS A. V. DE CARVALHO[2], CARLOS A. S. P. RODRIGUES[1], NILO DA S. MARQUES JR[1], RUBEN A. NASCIMENTO[1], SÁVIO S. DIAS[1], MARCOS R. S. MATOS[1]

1. *Laboratório de Hardware e Robótica, Ciência da Computação, Universidade Federal do Tocantins*
Caixa Postal 266, 77001-090, Av. NS 15, ALCNO 14, Bloco IV, 109 Norte, Palmas-TO
E-mails: arossini@uft.edu.br, krezeberto@gmail.com, nilobohn@gmail.com, rubenanapu@hotmail.com, diasssavio@gmail.com, mraylanptr@gmail.com

2. *Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia, UFRJ*
Caixa Postal: 68511, 21941-972, Cid. Universitária, Centro de Tecnologia, Sala 319H, Rio de Janeiro-RJ
E-mail: LuisAlfredo@ufrj.br

Abstract— This work presents a debugging tool for the RoboCup Simulation 2D category. This tool has graphics capabilities, monitoring values of variables and game statistics collection. This tool is able to pause, fast forward an rewind a game at runtime, presenting the world model's information of each agent (players and coach) selected by the user. This way, two modules were implemented: the iBotsDebuggerLib, a library to collect and send data agents and game statistics; and the monitor module, the iBotsDebugger, responsible for graphical interaction with the user. After implementation, validation tests were performed, comparing the developed tool with pre-existing tools, and the results demonstrated the using feasibility of iBotsDebugger.

Keywords— debugger, robot soccer, simulation 2D, multiagent systems.

Resumo— Este trabalho apresenta uma ferramenta de depuração de código fonte para a categoria *RoboCup Simulation 2D*. Essa ferramenta possui recursos gráficos, monitoramento de valores de varáveis e coleta de estatísticas do jogo. Essa ferramenta é capaz de pausar, avançar e retroceder uma partida em tempo de execução, apresentando ainda informações do modelo de mundo de cada um dos agentes (jogadores e técnico) selecionadas pelo usuário. Para tanto, dois módulos foram implementados: o *iBotsDebuggerLib*, uma biblioteca responsável pela coleta e envio dos dados dos agentes e estatísticas do jogo; e o módulo monitor, *iBotsDebugger*, responsável pela interação gráfica com o usuário. Após a implementação, testes de validação foram executados, comparando a ferramenta desenvolvida com ferramentas pré-existentes, e os resultados obtidos demonstraram a viabilidade do uso do *iBotsDebugger*.

Palavras-chave— depurador, futebol de robôs, simulação 2D, sistemas multiagentes.

1 Introdução

Este artigo descreve as soluções implementadas pela equipe iBots/UFT na categoria *RoboCup Simulation 2D*. A equipe é baseada no Agent2D 3.1.1, que depende da biblioteca librcsc - a versão utilizada foi 4.1.0. A equipe iBots/UFT identificou ao longo do seu desenvolvimento que nos últimos anos a sua maior dificuldade era a não existência de um depurador que atendesse suas necessidades. Para suprir essa lacuna, foi desenvolvido o depurador *iBotsDebugger* para ser usado em conjunto com o Agent2D.

As ferramentas disponibilizadas pela *RoboCup* desempenham os papéis de simulação e visualização de jogos da categoria *Robocup Simulation 2D*, porém são limitadas quanto a depuração e em recursos para testes e validação de soluções.

Um time é composto por 12 agentes (11 jogadores e um técnico), cada agente é um processo diferente em execução que recebe informações limitadas e com ruído, exceção feita ao treinador por ser um agente privilegiado e, por isso, recebe informações precisas sobre todo o campo. Desse modo, cada jogador tem sua própria visão de mundo. Como o monitor recebe apenas as informações precisas recebidas pelo servidor, não é possível visualizar graficamente o modelo de mundo de cada agente utilizando as atuais ferramentas cedidas pela *RoboCup*. A incerteza sobre o modelo de mundo dos jogadores pode levar, eventualmente, a situações de reimplementações desnecessárias, em que a solução desenvolvida está correta apesar de aparentar estar errada, ou ainda levar a acreditar que um método esteja funcionando quando na verdade não está.

Atualmente, para analisar valores de variáveis na categoria *RoboCup Simulation 2D*, pode-se usar a escrita em arquivo de log para posterior leitura "manual" ou exibição através de aplicativos desenvolvidos pela própria equipe. Porém, esses dois métodos não se integram com a janela do visualizador padrão, o que acaba dificultando o acompanhamento ciclo a ciclo. Além disso, vale ressaltar que diferentes agentes têm valores de variáveis distintos, o que resulta na necessidade de análise de 12 arquivos diferentes de log por partida.

A ferramenta visual *iBotsDebugger* objetiva proporcionar o desenvolvimento de soluções com maior agilidade e permitir a validação das soluções

em testes, algo que atualmente é muito dispendioso de ser feito. Para a categoria *RoboCup Simulation 2D*, a existência de uma ferramenta como essa não traria benefícios apenas às equipes já estabelecidas. O depurador pode auxiliar no processo de aprendizagem de códigos fontes disponibilizados por outras equipes e, como consequência, espera-se popularizar ainda mais o simulador 2D da *RoboCup* como plataforma de testes para diversos problemas (estudo de técnicas de inteligência artificial, robótica cooperativa e interação de sistemas multiagente) e, com isso, estimular a criação de novas equipes.

2 Propósito

Em termos computacionais, um depurador (do inglês *debugger*) é um programa utilizado para testar outros programas, sendo um dos seus objetivos o auxílio no processo de desenvolvimento através do acompanhamento detalhado de cada componente do código fonte. Porém, no futebol de robôs simulados, esse conceito deve ser adaptado uma vez que se trabalha com o conceito de sistemas multiagentes. No domínio de sistemas multiagentes os principais pontos que dificultam o processo de depuração são (Lacerda Jr, 2004): a não previsibilidade do comportamento de uma sociedade de agentes; a dependência entre os agentes na execução das funcionalidades do sistema; a grande variedade de dados a serem analisados durante a depuração (regras disparadas, mensagens trocadas, modelos de mundo e a própria informação processada pelo sistema).

Em relação à depuração de sistemas multiagentes no futebol de robôs, algumas dificuldades a mais são observadas. Essas dificuldades estão relacionadas ao fato do ambiente ser executado em tempo real, onde o desempenho dos agentes é vital para a execução das tarefas e, consequentemente, não pode ser afetado pelo processo de depuração. Há ainda a introdução intencional de ruído pelo servidor nas ações e percepções do agentes, o que gera um nível a mais de informação a ser depurado. Esses fatos demonstram a complexidade envolvida no processo de construção de um depurador voltado ao futebol de robôs simulados.

Atualmente, as ferramentas disponibilizadas pela *RoboCup* são completas quanto à simulação e visualização de jogos da categoria de simulação 2D, porém são limitadas quanto às opções que facilitariam o processo de teste e validação de novas abordagens. Por exemplo, a ferramenta de visualização padrão para as competições, o *rcssmonitor*, não apresenta comandos que permitam pausar, retroceder ou avançar o jogo e ao mesmo tempo verificar o conteúdo de variáveis dos agentes. Também não coleta estatísticas das equipes, algo importante no processo de validação de soluções.

Na literatura, alguns trabalhos a respeito de ferramentas gráficas de depuração podem ser encontrados. Em Akiyama et al. (2012), algumas informações a respeito do depurador desenvolvido pela equipe Helios podem ser encontradas. Esse depurador é uma ferramenta de código aberto, nomeado de soccerwindow2. Além da funcionalidade de depuração, essa ferramenta possui ainda interface de exibição de uma partida em tempo real similar ao rcssmonitor. Assim como ocorre com o Agent2D, que é disponibilizado por essa mesma equipe, seu pleno uso é limitado.

Chelberg et al. (2000) implementaram um depurador visual tridimensional chamado 3D-VDBM. De acordo com os autores, o principal objetivo desse trabalho foi o desenvolvimento de uma arquitetura genérica para depuradores voltados para aplicações de tempo real, mais especificamente para os problemas da *RoboCup*. Essa arquitetura então inclui uma linha de comunicação do cliente para um observador que irá coletar e agrupar as informações do servidor e clientes. Além da posição normal de todos os objetos, dados adicionais como *stamina* e informação auditiva são encaminhados em tempo real para vários 3D-VDBM, de modo a inspecionar vários clientes ao mesmo tempo.

Reis et al. (2007) descrevem algumas soluções desenvolvidas pela equipe FC Portugal, entre elas a utilização de um depurador visual a fim de analisar o comportamento dos agentes. Shi et al. (2008) também relatam nas inovações incorporadas à equipe WrightEagle2008 de futebol de robôs simulados o desenvolvimento de um depurador, que tem como característica modificar o modo de comunicação entre os agentes e o servidor, através de um *buffer* comum.

3 Métodos

Para o desenvolvimento do depurador *iBotsDebugger* foi necessário dividir o problema em duas partes. A primeira parte diz respeito à uma biblioteca de classes para serem chamadas no código fonte a ser testado (*iBotsDebuggerLib*), a fim de coletar dados para serem exibidos através da interface. Já a segunda parte refere-se à interface de um monitor propriamente dito (*iBotsMonitor*). O monitor é importante para que o usuário consiga visualizar graficamente as informações dos agentes de um time.

Esse monitor por sua vez recebe informações, mensagens em XML, através de *sockets* UDP. A mensagem XML é formada a partir de dados coletados do código fase do agente e estruturada pela biblioteca *iBotsDebuggerLib*. A figura 1 exibe esse processo.

Figura 1. Fluxo de funcionamento do depurador *iBotsDebugger*

A ideia inicial era utilizar o TCP TCP (*Transmissiont Control Protocol*) no processo de comunicação entre o monitor desenvolvido e a biblioteca por ser um protocolo robusto e confiável. O TCP é um protocolo orientado à ligação que utiliza confirmação positiva e retransmissão para controlar o fluxo e confirmar a entrega dos dados. A confirmação positiva e retransmissão PAR (*Positive Acknowledge and Retransmission*) tem a função de fornecer fiabilidade, ou seja, com o pacote acionado, a origem aciona um temporizador e aguarda a confirmação antes de enviar o próximo pacote da sessão. Se o temporizador expirar antes de a origem receber uma confirmação, a origem retransmite o pacote e reinicia o temporizador. Esse mecanismo garante a não ocorrência da perda de pacotes e que estes pacotes sejam recebidos na mesma ordem de envio. Porém, o TCP não demonstrou desempenho suficiente para manipular a quantidade de dados trafegados por exigir a confirmação de recebimento. Além disso, a utilização de mecanismos de sincronização (semáforo) no TCP pode resultar em um fila de mensagens a serem armazenadas. Enquanto o jogador aguarda para enviar sua mensagem, ele permanece sem executar suas ações e fica parado em campo. Conforme o volume de dados na fila de mensagens vai aumentando, mais jogadores param no decorrer de uma partida. Devido a esses fatores, o uso do TCP tornou-se inviável. Visando solucionar o problema, foi adotado o protocolo UDP (*User Datagram Protocol*).

O UDP é um protocolo de transporte não orientado, o qual troca datagramas sem confirmação ou entrega garantida. O protocolo UDP procura oferecer às aplicações um serviço de entrega de pacotes básico, que consiste apenas em colocar a mensagem recebida da camada de aplicação dentro de um segmento (pacote UDP) e utiliza os números de portas para identificar as aplicações. Portanto, as características oferecidas pelo UDP são praticamente as mesmas que o próprio IP oferece, com a diferença que o pacote UDP utiliza números de porta para multiplexar os pacotes entre as aplicações. Apesar de não garantir fiabilidade como ocorre no TCP, quando se trabalha com problemas onde a velocidade de transmissão e recepção são pontos críticos a serem considerados, como aplicações em tempo real, a baixa sobrecarga do UDP torna-se altamente interessante. A utilização do protocolo UDP no depurador visual deste trabalho não se mostrou prejudicial para a execução do time, nenhum agente foi impedido de executar suas ações durante todo o processo de teste e, por isso, não comprometeu o desempenho da equipe.

Todo o projeto foi desenvolvido utilizando a linguagem C++, tanto por ser a linguagem utilizada no *framework* Agent2d e nas atuais ferramentas disponibilizadas pela RoboCup quanto por ser uma linguagem popular e de bom desempenho. A seguir são descritas as estruturas da interface e da biblioteca.

3.1 A biblioteca iBotsDebuggerLib

A *iBotsDebuggerLib* é a parte do depurador responsável por coletar dados dos agentes e enviá-los à interface. Ela é composta de um módulo principal, chamado de dbg, que oferece ao programador os métodos que serão adicionados ao código a ser depurado, o Código Base da aplicação. Uma vez adicionados, esses métodos coletam os dados relevantes a respeito dos agentes, bem como estados de variáveis para exibição no monitor.

Essas variáveis são enviadas para o módulo VarsModel, o qual armazena seus valores para posterior empacotamento na mensagem. Depois que o dbg finaliza suas tarefas de coleta de dados, o módulo XMLBuilder recebe as informações de posicionamento de agentes e objetos da dbg, além das informações de variáveis previamente armazenadas em VarsModel. Após recebê-las, o XMLBuilder então é capaz de construir a mensagem, em formato XML, e enviá-la para o módulo UDPClient, que se encarrega de enviar esta mensagem ao *iBotsMonitor* por meio de *socket*. A figura 2 ilustra todo esse processo.

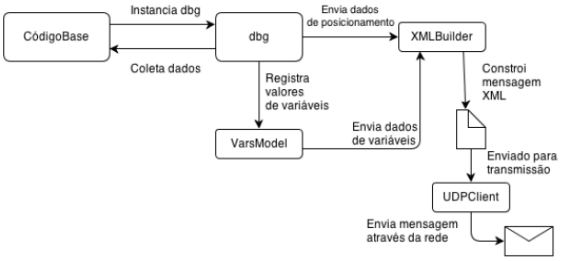

Figura 2. Esquema de funcionamento básico do depurador *iBotsDebugger*

Antes de ser enviada, a mensagem contendo as informações do modelo de mundo do agente em questão é estruturada por meio do padrão XML, que facilita sua interpretação pela interface.

A mensagem XML possui uma estrutura levemente diferenciada para os agentes do tipo jogador e treinador, algo necessário devido às diferenças entre os modelos de mundo destes agentes. Por exemplo, os agentes do tipo jogador enviam informações de posicionamento e *stamina* próprios e seu número de uniforme, informações não contidas no agente treinador.

Além dessas informações, a mensagem XML ainda possui campos comuns entre os dois tipos de mensagens, que são a posição atual da bola e a posição de todos os jogadores de ambos os times presentes em seu modelo de mundo. Outras informações também são comuns, como, por exemplo, jogadores que foram punidos com algum cartão punições com cartões, além das variáveis recebidas pelo módulo VarsModel.

Uma vez que a mensagem está pronta para ser enviada ao monitor, o módulo UDPClient se encarrega de enviar a mensagem através da rede. Visando

otimizar o processo de envio, uma biblioteca chamada SocketCC (SocketCC, s/d) foi utilizada. Essa biblioteca possui classes previamente implementadas para operar com aplicações cliente/servidor, utilizando tanto o protocolo UDP quanto o TCP. Após o envio da mensagem, o trabalho da biblioteca *iBots-DebuggerLib* é finalizado e deve ser repetido a cada ciclo.

3.2 O monitor iBotsMonitor

Quando um usuário solicita uma informação a respeito da situação de um determinado agente em um determinado ciclo, o monitor verifica se essa informação lhe foi previamente enviada pela *iBotsDebuggerLib*. O monitor se adequa para exibir graficamente todas as posições dos objetos conhecidos (jogadores do próprio time, do time adversário e bola) pelo agente (treinador ou jogador) que foi selecionado pelo usuário no ciclo requisitado, como ilustrado na figura 3. Além dos objetos, paralelamente são exibidos os valores das variáveis enviados pela *iBotsDebuggerLib* (número de uniforme, peso de alguma equação, estados de algoritmo, etc.) ou os resultados dos cálculos das estatísticas (chutes a gol, posse de bola, etc.) até o ciclo selecionado.

Figura 3. Tela principal do *iBotsMonitor*, exibindo as informações do ciclo requisitado

Um dos objetivos principais e diferencial do depurador desenvolvido consiste na possibilidade de pausar, retroceder ou avançar (até o estado atual da partida) a visualização da partida em tempo de execução.

Ao pausar a visualização da partida em execução, é possível analisar com maior precisão fatores como: sistema tático de uma equipe, valores de variáveis, estatísticas, além da visão de mundo de cada jogador no ciclo pausado. Em alguns casos a análise de um único ciclo não é conclusivo, dessa forma, os comandos de retrocesso e avanço possibilitam ao usuário a análise da imagem de um conjunto de ciclos adjacentes.

Para permitir essas funcionalidades, o *iBotsMonitor* foi estruturado de acordo com a figura 4. O monitor encaminha a mensagem recebida da *iBots-DebuggerLib* ao módulo de análise e desconstrução (Analisador da Mensagem). Esse módulo utiliza a biblioteca Libxml2 (Libxml2, 2013) para *tokenizar* a mensagem, ou seja, dividi-la de acordo com seus padrões estruturais e possibilitar que esses dados sejam armazenados em estruturas próprias (Estruturas de Armazenamento). As Estruturas de Armazenamento são um encadeamento de listas visando facilitar o acesso aleatório às informações requeridas pelo usuário. Já os Coletores de Estatísticas levam em consideração apenas as informações oriundas do agente treinador e disponibiliza informações quantitativas para a Interface com o Usuário. Por último, a Interface com o Usuário, que foi desenvolvida através da biblioteca de interface gráfica Qt (Digia, s/d), tem a função de garantir a interação do usuário com o restante do programa com uma interface amigável.

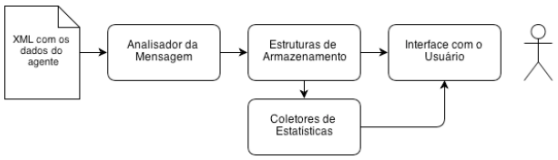

Figura 4. Esquema de funcionamento do monitor *iBotsMonitor*

3.3 Coletor de estatísticas

Dentro do *iBotsDebugger* foi implementado um módulo coletor de estatísticas. Atualmente, caso uma equipe queira coletar estatísticas da partida, ela deverá implementar métodos integrados ao código fonte dos agentes para coleta dos dados que julgue importantes. Por meio de dados estatísticos, a equipe pode obter valores quantitativos sobre o desempenho do time, algo necessário para validar soluções.

Foram propostas seis estatísticas para serem calculadas pelo *iBotsDebugger*: tempo de posse de bola das equipes; quantidade de passes errados e certos das equipes; quantidade de chutes e chutes a gol das equipes; e a estimativa do sistema tático utilizado pela equipe adversária. Essas estatísticas foram escolhidas empiricamente através da experiência adquirida no processo de desenvolvimento da equipe e por serem informações comumente utilizadas no futebol.

O processo de validação de soluções é amparado por estatísticas. Por exemplo, se foi desenvolvido um novo algoritmo de sistema defensivo, espera-se que o número de chutes ou de chutes a gol do adversário diminua ou ainda que o número de passes errados do adversário aumente. Do mesmo modo, se foi desenvolvido uma nova soluções ofensiva, espera-se que o número de chutes ou de chutes a gol da equipe aumente ou que a relação gols por chutes seja incrementada. Ainda se o algoritmo de passe foi modificado, verificar se a porcentagem de passes certos aumentou, tipo de teste que a equipe iBots/UFT necessitou e apresentou em Silva (2012).

Saber o sistema tático adotado pelo adversário subsidia parte da informação para saber o quanto a

postura do adversário é defensiva ou ofensiva. O algoritmo implementado classifica o posicionamento tático do adversário em sete classes: 541, 532, 451, 442, 433, 352 e 343.

4 Resultados

A fim de se analisar o desempenho do depurador visual *iBotsDebugger* desenvolvido neste trabalho, os seguintes testes foram realizados: posicionamento dos jogadores na tela principal do monitor; visualização das informações de variáveis recebidas por cada agente por ciclo ou conjunto desses; validação do algoritmo de estatística correspondente à estimativa do sistema tático adotado pela equipe adversária.

O primeiro ponto a ser analisado é a capacidade do visualizador de exibir as informações recebidas pelos agentes de forma precisa. Para tanto, foram executados testes visuais comparando a imagem de saída do visualizador padrão da *RoboCup*, conhecido como *iBotsMonitor*, com a imagem resultante do depurador *iBotsDebugger* para o mesmo ciclo. Para exemplificar, é possível observar na figura 5 a comparação do posicionamento dos jogadores para o ciclo 196, escolhido aleatoriamente, de uma mesma partida.

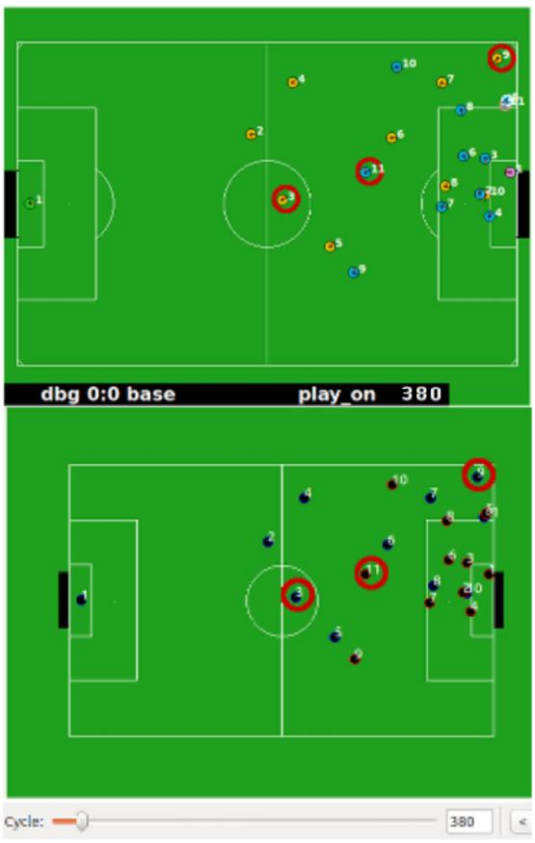

Figura 5. Exemplo de comparação de visualização do posicionamento dos jogadores entre *rcssmonitor* e *iBotsMonitor*

Os resultados demonstraram a viabilidade da utilização do monitor do *iBotsDebugger* como ferramenta de visualização de uma partida em tempo real, atingindo uma meta proposta por este trabalho.

A desigualdade existente entre os modelos de mundo do treinador e dos jogadores é ilustrada a partir das diferenças visuais apresentadas nas figuras 6, 7 e 8, que exibem o estado do modelo de mundo de três agentes (treinador, goleiro e o jogador de número seis) em um mesmo ciclo de uma mesma partida.

Figura 6. Exemplo de visualização do modelo de mundo do treinador

Figura 7. Exemplo de visualização do modelo de mundo de um jogador afastado dos demais

Figura 8. Exemplo de visualização do modelo de mundo de um jogador próximo dos demais

É possível notar que as diferenças entre o modelo de mundo do treinador (figura 6) e de um jogador mais afastado dos demais, no caso o goleiro (figura 7), são mais expressivas do que as diferenças entre o modelo de mundo do técnico e de um jogador mais próximo dos demais, no caso do jogador de número seis (figura 8). Isso ocorre devido às limitações do campo de visão simuladas pelo *rcssserver*, além da atribuição de ruídos nas informações recebidas. Vale ressaltar que nem todas as informações de posicionamento dos demais jogadores podem estar disponíveis para um agente. Um exemplo desse evento ocorre na imagem do modelo de mundo do jogador de número seis, que não exibe a posição do goleiro do time adversário por falta de informações confiáveis.

Após a constatação da viabilidade do módulo de visualização, o próximo passo foi a validação das informações de valores de variáveis selecionadas pelo usuário. Essa validação ocorreu através da comparação de dados salvos em arquivos de log oriundos do código fonte da equipe com os dados exibidos pela interface do *iBotsDebugger*. Essa funcionalidade permite um acompanhamento mais preciso em relação às informações de cada agente.

Em 100% dos casos observou-se a concordância entre os dados apresentados no log do código fonte (parte superior da figura) e dos dados exibidos pela interface *iBotsDebugger*, o que demonstra o correto funcionamento do módulo VarsModel da *iBotsDebuggerLib*.

Com os módulos de visualização e coleta/envio de dados obtendo êxito em suas tarefas, o módulo de estatísticas passou a ser testado. Para validar os algoritmos de tempo de posse de bola, chutes/chutes a gol, passes errados e certos, três humanos assistiram três partidas cada um. Os resultados da posse de bola de uma equipe obtidos pelo algoritmo e dos analisados pelos humanos foram muito próximos, coincidindo quase que em 100% dos casos; por outro lado, os algoritmos de chutes e passes ainda necessitam ser melhorados uma vez que os resultados divergiram em alguns casos.

A validação da estimativa da formação tática foi realizada também pelos três observadores humanos. Eles assistiram uma partida realizada entre a equipe iBots/UFT e uma equipe composta integralmente pelo código fonte do Agent2D. Os observadores indicaram que o sistema tático que foi mais adotado pelo Agent2D foi o 433 e em segundo lugar o 343, resultado que coincide com o do algoritmo proposto.

5 Conclusão

Este trabalho apresentou o desenvolvimento do depurador *iBotsDebugger* para a categoria *RoboCup Simulation 2D*, que deve ser executado com o framework Agent2D. Esse depurador é uma ferramenta visual que permite pausar, avançar e retroceder uma partida em tempo de execução, além de permitir o acesso às informações do modelo de mundo de cada um dos agentes, apresentando ainda estatísticas do jogo.

No processo de desenvolvimento do *iBotsDebugger*, o primeiro passo foi a implementação de uma biblioteca de classes a serem chamadas no código fonte a ser testado, a *iBotsDebuggerLib*. Essa biblioteca foi capaz de coletar os dados de todos os agentes em campo (jogadores e treinador), empacotando tais informações em formato XML e as enviando ao módulo da interface gráfica. Posteriormente, foi desenvolvida a interface gráfica (monitor) propriamente dita, chamada de *iBotsMonitor*. Esse monitor ao receber as informações dos agentes, no formato de mensagens XML, da biblioteca *iBotsDebuggerLib*, foi capaz de apresentá-las ao usuário de forma visual. O desenvolvimento dessa ferramenta visual teve como principal finalidade suprir as necessidades básicas de informação a respeito do estado de cada agente em tempo de execução.

Referências Bibliográficas

Akiyama, H., Shimora, H., Nakashima, T., Narimoto, Y., and Yamashita, K. (2012). HELIOS2012 Team Description Paper.

Chelberg, D. M., Gillen, M., Zhou, Q., and Lakshmikumar, A. (2000). 3D-VDBM: 3D visual debug monitor for RoboCup. In IASTED International Conference on Computer Graphics and Imaging, pages 14–19.

Digia (2013). Qt - Digia. Online. Disponível em: http://qt.digia.com/. Acessado em: 22 de agosto de 2013.

Lacerda Jr, L. F. B (2004). Módulo de Visualização de Dados de Depuração de Agentes Inteligentes da RoboCup. Trabalho de Graduação, UFPE.

Libxml2 (s/d). The XML C parser and toolkit of Gnome. Online. Disponível em: http://www.xmlsoft.org/. Acessado em: 22 de agosto de 2013.

Reis, L. P. (2003). Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico, Coordenação em SMA no Futebol Robótico (Páginas 299-404), PhD.

Shi, K., Bai, A., Tai, Y., and Chen, X. (2008). WrightEagle2008 2D Soccer Simulation Team Description Paper. In RoboCup International Symposium.

Silva, A. T. R., Rodrigues, C. A. d. S. P., Jardim, V. P., Nascimento, R. A., Silva, V. S., and Rodrigues, H. B. (2012). iBots 2012 - Team Description Paper. Latin American Robotics Competition.

SocketCC (s/d). SocketCC - A C++ Sockets Library. Online. Disponível em: http://www.ctie.monash.edu.au/SocketCC. Acessado em 22 de agosto de 2013.