TOWARDS THE IMPLEMENTATION OF A HANDBALL PLAYER AGENT FRAMEWORK

João Jacob[†], Rosaldo J. F. Rossetti[‡], António Coelho[†], Rui Rodrigues

[†] INESC TEC – INESC Technology and Science
[‡] Artificial Intelligence and Computer Science Laboratory (LIACC)
Department of Informatics Engineering (DEI)
Faculty of Engineering, University of Porto (FEUP)
Rua Dr. Roberto Frias S/N, 4200-465, Porto – PORTUGAL
{joao.jacob, rossetti, acoelho, rui.rodrigues}@fe.up.pt

ABSTRACT

Sports simulation can help to assess the performance of strategies and players in a sand-boxed environment. Ultimately it can lead to improved real-life performance of actual teams if it is able to provide useful information to the team coach or manager. Many tools are now available for many different sports, most notably Soccer Server Simulator, a soccer simulator, that has already attracted many researchers from the artificial intelligence community into developing intelligent agents (soccer players) for them to form teams to play against each other in simulated matches. This paper presents a methodology for developing agents for testing and further developing the Handball Sport Simulator, which is based on the Soccer Server Simulator. The main contribution of this work is to provide a basic, expandable, agent architecture, specifically capable of playing at a Handball Sport Simulator server, while at the same time testing what features and sport's rules of the Handball Sport Simulator have been implemented correctly.

Keywords: workstation design, work measurement, ergonomics, decision support system

1. INTRODUCTION

Agent-based models as means of portraying complex environments such as sport games are not new. These simulators consist of several agents cooperating and competing towards common or opposite goals. Information exchange, coordinating tasks or competing for resources are appropriate scenarios to be modeled through the multi-agent metaphor. This is also verifiable in real-life situations, thus making agents a valuable addition for simulators that attempt to simulate those scenarios, being sports simulators very complete, as cooperation and coordination are needed as well as the competition for a goal.

Several of these simulators exist, for many sports. One of the most widely known is the Soccer Server Simulator, which even has its own category in the RoboCup competition (Chen 2002). The goal of this simulator is to be a tool useful for the development and testing of artificial intelligence techniques, as it serves as a test-bed for teams of agents to compete with the same resources, implying that one team can only defeat another if it is tactically superior. As this simulator is agent based (each agent represents one player in a game), it allows for everyone interested in developing an agent to do so in any programming language, provided that it respects the client-server protocol that the Soccer Server Simulator requires.

However, there are still other sports that lack the availability of an easy-to-use agent-based simulator. In order to bridge this gap, a Handball Server Simulator (Santiago 2011) was recently developed, being based on the Soccer Server Simulator's core engine. This allowed the Handball Server Simulator to retain most of the added value of the original Soccer Server Simulator, such as the ease of use and the ease of developing agents for it. However, being such a new simulator, there are currently no agents developed specifically for it, and as such, it is difficult to assert if the simulator is missing features (such as game rules) or if those already present were incorrectly implemented.

The purpose of this work is to provide a framework for the development of simple cooperative handball agents capable of testing the implemented features of the handball server. It is not the focus of this work the development of a highly robust and competitive handball agent. The agent can be later improved upon, as it is further tested with the simulator, while the simulator is also updated taking these tests into consideration.

The remaining part of this paper is organized as follows; the Methodology section, describes the designed model, how it was implemented and the limitation of certain approaches. The Results section provides some results of the testing of the prototype agent in the Handball Server Simulator, and finally the Conclusions section, summarizes the achieved goals and criticizing the obtained results whereas also providing some future goals for the rest of this project.

2. RELATED WORK

Multi agent simulation is a technique used for simulation in several scenarios with different purposes (Murakami 2003). One possible scenario for multi agent simulation to be used is team sports. The most notable simulator would be the Soccer Server Simulator that is used in the RoboCup competition (Chen 2002). There are practically no multi-agent sports simulators similar to the Soccer Server Simulator for other sports. In order to fill this gap the Handball Server Simulator was

developed (Santiago 2011). It provides a solution for simulating handball matches in a similar way to that of the Soccer Server Simulator.

Although there are no other agents developed for the Handball Server Simulator, since the basis of the simulator is the Soccer Server Simulator, it is possible to adapt existing ones. Considering that most of the principles used in the development and testing of agents for the Soccer Server Simulator also apply for the Handball counterpart, an existing soccer agent can be adapted easily with minor changes in the client-server protocol and expected changes in its behavior, as it will now be playing handball and should perform accordingly. Having this in mind we opted to adapt an existing agent architecture, designed for the Soccer Server Simulator, into a Handball Server Simulator player.

Zang and associates discuss on of how agent architectures can be developed specifically for the usage in the Robocup soccer simulator (Zhang 2000). They explain how agents interact with the simulation environment, capturing information from it, building representations of that information, deliberating and acting based on those representation and the information exchanged among them. They also provide an overview of the architecture of some teams which take part in that competition. The CMUnited-99 (Reis 2001) (a former champion of the competition) was identified as having a behavior-based architecture, a common architecture for agents, as they take actions depending on the actual world state. Other architectures were identified, such as the layered architecture. This architecture is similar to that of the architecture used in this work, where there is a clear separation between layers of basic abilities or actions, client-server commands, interpretation of received information and higher level abilities, such as spatial reasoning and ponderation. This architecture can be found in teams such as RobologKoblenz99, among others (Certo 2007).

Candea and associates noted the importance of coordination in multi-agent simulations such as the Robo Cup simulator (Candea 2001). They mention how communication based coordination can help improve team performance and self-organization in a more reliable way. Other forms of coordination, as well an autonomous coordination or heterogeneous agents' coordination are referred. However, communication based coordination as a means of distributed coordination was found to be the preferred approach in their work, as it eases the burden of coordination (agents do not have to rely on "coach" like tactics) and ensures homogeneity. The issues identified, such as the dynamic assignment of roles and team strategies are important ones. These issues and the importance of roles in this context have also been identified elsewhere (Åberg 1998).

Positioning the agents of a team correctly is also an issue, since the environment they are at is very dynamic. Akiyama proposed a solution for finding suitable positions for each agent, based on the status of the environment (Akiyama 2008). The method requires training data to be collected and has several advantages such as being fast, scalable, accurate and reproducible. A similar approach was that of Reis and his associates, that based the positioning and coordination of a team on predefined tactics and formations (Reis 2001). This allowed the team to dynamically change the gameplay, position and roles of its players in different situations of the game. They considered a "High level decision module" that was responsible for outputting adequate actions taking into account formations, roles, state of the world and the game's situation. The adaptability of this solution provided very positive results with the FC Portugal team (using this approach) winning most matches against several of the RoboCup teams.

Genetic algorithms have also been used for the training and performance evaluation of RoboCup simulator teams (Aronsson 2003). However, they are hindered by the usual limitation of this type of algorithms, such as overfitting, limited search space, expensive computation and premature convergence. Even so, it was proven that it is possible for software robots to learn the rules of simulated soccer.

3. METHODOLOGIC APPROACH

The aim of this work is to provide a framework for the creation of handball player agents for the Handball Server Simulator while also testing it via the agents developed with said framework.

3.1. Specifications of a Handball Player Agent

Although there are some similarities between the Soccer Server and the Handball Server, as they both simulate a team sport, played with a ball, two opposite goals on a rectangular field, since the rules for each sport are different, there has to be differences between them.

As such, it is expected that a player agent developed for the Handball Server Simulator will differ from one aimed at the Soccer Server Simulator. Similar to a soccer player agent, however, a handball player agent will attempt to accomplish its goals while ensuring that the game's rules are respected. In light of this, it is possible to adapt a soccer player agent's code into a handball one as they differ (in terms of their own logic) in rules to respect (which influences their available actions).

3.2. Handball Server Simulator communication protocol

The Handball Server Simulator, like the Soccer Server counterpart is part of a client-server architecture consisting of a game (the server) that is disputed by two teams, formed by several players (the clients). The server will provide information regarding the game's state to all of the agents so that the agents may deliberate upon it and act by sending to the server individual actions. Each agent receives both common (such as the game's score or what the other players are saying) and unique information (such as what is in its

field of view) from the server. Thanks to this approach, both servers allow for clients to be implemented in diverse ways, as long as they respect the server's protocol.

Prior to designing and developing the agent's behaviors it was deemed necessary the creation of an architecture for its basic modules. Since the Handball Server Simulator was based on the protocol of the Soccer Server Simulator, searching for a released simple Soccer Server Simulator agent's code that could serve as a basis for this work. From that basic code, we could change the needed protocol parts and agent behaviors.

Since the communication between the Handball Server Simulator and the clients (agents) is made via sockets (UDP/IP), these clients may be implemented in any language and run on any device as long as they comply with the server's protocol.

Determining what the protocol was, relied on consulting the Soccer Server Simulator's protocol (Chen 2002) as well as the source code for the Handball Server Simulator. The handball simulator has many common aspects with the Soccer Server Simulator's protocol as it can be seen in the table 1 and table 2.

Table 1: Comparison between Soccer Server Protocol and Handball Protocol (Client to Server)

Soccer Server	Handball Server	Behaviour
Init	Init	Registers the player in the server
Kick	Throw	Allows a ball that is in possession by the player to be kicked or thrown with varying force and angle
-	Step	The agent takes one step with the ball in its hand without releasing it (moving without dribbling)
Catch	Catch	The agent catches a ball that is in his range of action
Change_View	Change_View	Allows the agent to change its viewing angle. A wider angle allows for more information to be seen and a narrower one for the information seen to be of higher quality
Say	Say	The agent can share information with all agents present in the field
Move	Move	Places the agent instantly at a given position of the field (only available when the game hasn't started)
Sense_Body	Sense_Body	Asks for a sense_body command to be sent back from the server
Turn	Turn	Allows the agent to rotate its whole body a variable number of degrees
Turn_Neck	Turn_Neck	Allows the agent to only turn his head a variable number of degrees while keeping the rest of the body in the same orientation
Dash	Dash	The agent is capable of running at a certain direction and intensity

Table 2: Comparison between Soccer Server Protocol and Handball Protocol (Server to Client)

Soccer Server	Handball Server	Behaviour
Hear	Hear	Agent is capable of receiving all messages that were transmited to the server using the Say command from any agent
See	See	Agent captures visual information relative to his position and orientation, as the presence of other players, the ball, goals or game lines
Sense_Body	Sense_Body	Provides some information of the current situation of the agent such as current speed and stamina

As the above two tables show, there is not much difference from the handball server to the soccer server in terms of protocol. The two major differences are the Step command, that does not exist in the Soccer Server and the Throw command that behaves exactly like the Kick counterpart in the Soccer Server. The Step command is, as the name suggests, a step to be performed with the ball in hand (assuring that the ball is not lost, unlike dribbling with the ball in the Soccer Server). However, as the rules of Handball imply, only 3 steps may be taken with the ball in possession.

4. A FRAMEWORK TO DEVELOP HANDBALL PLAYER AGENTS

Since the goal of this work was to create a basis for the development of Handball Server Simulator agents, there was no need to choose a Soccer Server Simulator agent with complex behavior code. As such, the chosen base code was of the krislet-0.2 (Langner,2011) agent as it was a simple soccer agent that was already capable of processing many soccer server commands, being some of those commands also present in the handball server.

This agent was chosen instead of other candidates (such as Agent2D) as it contained nearly no logic for soccer or other sport-related notions, such as strategy. Since the focus of this work is mainly to create agents that test what features the handball server has working,

a complex agent isn't needed. The portion of code that was responsible for the agent's connection to the server was mostly left unaltered as the Handball Server Simulator's protocol is similar to the Soccer Server Simulator's in that aspect, and only values such as the port of the running server and team names were changed. Other elements, concerning the client-server protocol, that were present in the original agent's code were replaced or omitted in the final code.

Besides the client-server protocol, the original architecture of the agent was mostly left intact. However, there were some changes worth noticing.

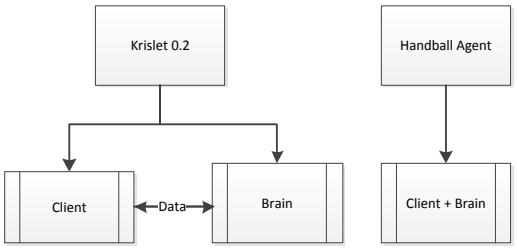

Figure 1: Diagram representing original architecture (left) versus the final architecture (right)

As it can be seen in figure 1, above, the original architecture was multi-threaded. This feature served the purpose of running the "Brain" thread (responsible for the decision making process) concomitantly with the "Client" thread (responsible for the client-server communication and parsing of the data), allowing for the agent to never miss a server communication, even while performing the needed calculations prior to sending a command to the server. Although, in theory, this can be considered a useful feature there were some issues to be addressed:

Synchronism: If the "Brain" thread was processing a received command, the "Client" thread should not update that received command in the event of receiving a server communication, as it might lead to mixing old information with new one. This means that new information is stored in a stack and the Brain will access the next item in that stack whenever it is done processing the current one. This can lead to the client's notion of the game's current state to be severely out of synchronization with the server's.

Wasted cycles: If the "Brain" thread finishes computing a command and has not received a new command to process, it will continuously verify if there are new requests to process. Even though the authors of the original code included a *sleep* call to help reduce the number of wasted cycles, this means that there is a possibility that the server and client are not synchronized, as the server may send new information to the client while the "Brain" thread is sleeping, wasting several milliseconds in this state.

All these issues pointed that a multi-threaded agent was difficult to synchronize and as such would not add any value to the solution, as it led mostly to wasted processing cycles. There is no gain in using separate threads for the retrieval and sending of commands and the processing of said commands if these tasks are to be executed sequentially. So, the architecture was changed to a single threaded one. This was done easily, and meant only changing the "Brain" thread's infinite cycle to a simple function that would be called each time whenever the "Client" thread would receive the server's commands. Part of the "Brain's" thread was left intact, the part that dealt with initializations and preprocessing, as we felt it would not interfere with synchronization issues when the agent started processing game instructions.

Table 3: Comparison between the original code's protocol and the final code's protocol

Original Code Protocol	**Final Code Protocol**
Init	Init (altered to include default values)
Shoot	Throw (operates the same way as shoot)
Say (unused in Brain class)	Say (altered to be used with objects of the Message class)
Dash	Dash
Turn	Turn
Catch(unused in Brain class)	Catch
Move	Move
See	See
Sense_Body(unused in Brain class)	Sense_Body
Hear(unused in Brain class)	Hear

As the above table 3 depicts, the original code's protocol did implement many commands (the Client class had each command implemented individually as a function). However, many of these commands were not used for the original agent behavior.

Most notably, the original soccer agent's code did not make use of the "say" command for inter-agent communication. This is a much needed feature for developing a cooperative team of agents for any team sport. In order to fill this blank, a "Message" class was created that allows the agent to send and receive coordination-specific content.

Instantiating a handball player and results

After designing the final architecture of the agent, it was necessary to develop a prototype handball agent. This agent should make use of all the previous mentioned features of the developed architecture. The developed agent implements a simple behavior, as depicted in figure 2.

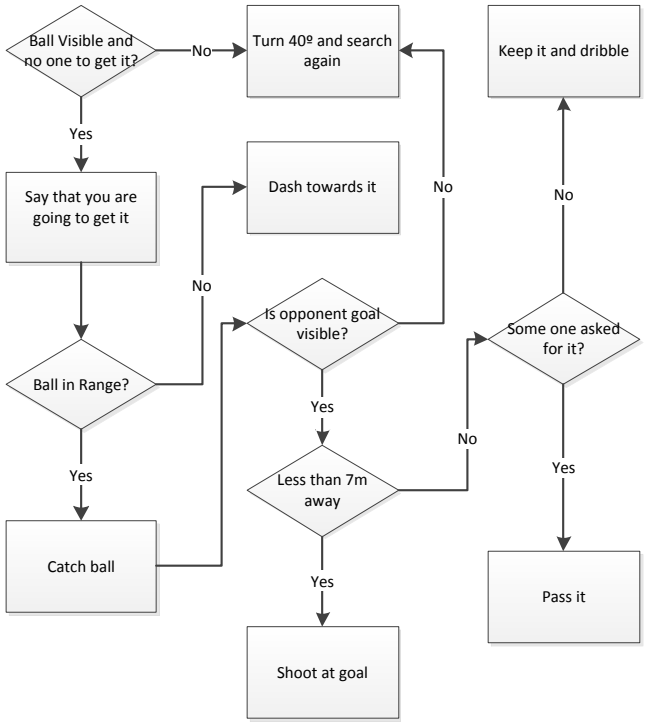

Figure 2: Agent Behavior

With this behavior, the commands present in table 2 were tested. Note that inter-agent communication and coordination was also tested and allowed for simple teamwork to be achieved. The untested commands were inherited directly from the Soccer Server Simulator's core engine and, as such, did not need any testing in order to validate them.

Table 4: Test coverage of client-server commands

Handball Server	**Testing Status**
Init	Tested
Throw	Tested
Step	Tested
Catch	Tested
Change_View	Untested
Say	Tested
Sense_Body	Tested
Turn	Tested
Turn_Neck	Tested
Dash	Tested
Score	Untested

And the following simulator objects were identified and used correctly by the handball agent.

Table 5: Simulator object identified and use in the agent's behavior

Object retrieved by "see"	**Presence in agent behavior**
Ball	Present
Flag	Present
Player	Present
Goal	Present
Line	Present

Finally, we were able to validate the following rules of the sport[10] in the Handball Server Simulator. This validation was made by checking the messages that were sent and received by the agent and the server, and by the game's referee. Additionally, most of these could be seen via the server's graphical viewer.

Table 6: Handball rules verification by the agent when connected to a server

Handball Server Testing Status	**Handball Server Testing Status**
Do not block or kick ball using feet	Server doesn't verify
Do not hold ball for more than three seconds	Tested
Do not bounce the ball, catch the ball and bounce it again	Server doesn't verify
Take no more than three steps with the ball caught	Tested
Do not enter goal area	Tested
Do not touch the ball lying inside the goal area	Server doesn't verify
Do not charge the opponents or run into a defending player	Untested
Do not engage in passive play	Server doesn't verify
Do not steal the ball from the hands of an opponent	Untested
Do not push or hold attacking player's body	Untested
Stay at least 3 meters away from the attacking player when restarting the game	Server doesn't verify
Goalkeeper cannot take the ball outside the area	Tested
Goalkeeper cannot take the ball inside the area	Tested

As table 6 suggests, some defensive features were untested, as the focus of the agent's behavior is mainly an offensive one. These untested features have to be validated by a human, (passive play, charging, pushing or holding adversaries) much like how "fouls" are handled by humans in the soccer server (Chen 2002). Other rules were not verified by the handball server,

meaning that although the agent was tested in a scenario where that rule would be infringed, the handball server did not punish the team for doing so.

5. CONCLUSIONS AND FUTURE WORK

The current work served both the purpose of creating a foundation for the development of future handball server simulator agent behaviors and of exposing the limitations of the server that result in an unrealistic representation of the sport. Although nearly all issues are due to the fact that the server has its roots in the Soccer Server Simulator, these issues were less relevant there. These issues revolve around the fact that the simulator is a 2D simulator and the players' physiognomy has been simplified. This means that at this time the server is incapable of determining if:

- The ball has been blocked or kicked using the feet
- The player is "in the air" when inside the goal area
- There was physical contact, such as a collision between opposing players
- The dribble was valid
- The defending player is closer than 3 meters to the attacking player when the game is being restarted

The Handball Server Simulator is also missing the implementation of some other handball rules, but that does not have to do with the simulators limitations. In fact, even in real life some rules are difficult to apply; for instance, the "passive play" rule is triggered if the players in possession of the ball aren't focused on attacking, solely preoccupied in maintaining the ball possession, a free throw is awarded to the opposing team.

The presented agent behavior server the purpose of proving that, currently, developing and testing handball agents is somewhat limited, as some rules of a handball game go unverified by the server. Furthermore these issues, if left unattended, can also lead to agents using strategies that would be rendered unusable in real life (such as long distance shots at the goal or passive play). Even so, these issues are acceptable due to the simulator being in its early days of development and having inherited some of these limitations from the Soccer Server Simulator. Hopefully, these foundations for the development of handball agents can help perfect the simulator, as it tested most of the rules and features already present in the Simulator.

We have presented the foundations of a handball agent, and also developed and tested a cooperative sample agent. Although the agent itself is still lacking some features, such as the capability of determining his position or the position of fellow and opposing players, these features can be easily added, thanks to the division of the "client" communication with the server and the "brain" component of the agent, responsible for the behavior and decision making of the agent. In fact, any further alteration to the agent is expected to be made only on the "Brain" component.

The next step for this project is to further develop the handball agent framework in parallel with the Handball Server Simulator. The development of a complete, cooperative and intelligent handball team of agents is the ultimate goal to be achieved. However, for that to happen, the Handball Server Simulator must see further development as well, to become more complete. Furthermore, testing new server features and developing more complex handball agent behavior will be needed and as such, the proposed handball agent framework will serve as a base for these future steps in this project.

REFERENCES

Åberg, H. (1998). *Agent Roles in RoboCup Teams*. Science And Technology , (February), 1-24.

Candea, C. (2001). *Coordination in multi-agent RoboCup teams*.Robotics and Autonomous Systems, 36(2-3), 67-86. doi:10.1016/S0921-8890(01)00137-3

Chen, M., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., Kostiadis, K., Kummeneje,J., et al. (2002). *RoboCup Soccer Server*.

H. Akiyama(2008), *Multi-agent positioning mechanism in the dynamic environment*, RoboCup 2007: Robot Soccer World Cup XI, pp. 377-384, 2008.

International Handball Federation, *Rules of the Game* , July 2010

J. Aronsson (2003), *Genetic Programming of Multi-agent Systems in the RoboCup Domain*, M.Sc thesis, Lund Institute of Technology, 2003.

J. Certo, N. Lau, L.P. Reis (2007), *A generic multi-robot coordination strategic layer*, First International Conference on Robot Communication and Coordination. Athens, Greece, 2007.

L. Reis and N. Lau (2001), *Situation based strategic positioning for coordinating a team of homogeneous agents*, Balancing Reactivity and Social Deliberation in Multi-Agent Systems, From RoboCup to Real-World Applications, 2001.

Langner, K., *Krislet 0.2* (http://www.ida.liu.se/~frehe/RoboCup/Libs/Sources/ Last accessed on September 3 , 2011).

Santiago, C. B., & Reis, L. P. (2011). *Foundations for Creating a Handball Sport Simulator*. Information Systems and Technologies CISTI, 2011

Y. Murakami, T. Ishida, T. Kawasoe, and R. Hishiyama (2003), *Scenario description for multi-agent simulation*, Proceedings of the AAMAS '03, pp. 369-376, 2003.

Y. Zhang (2005), *Multi-agent systems: the Tao of Soccer* , Tutorial presented at SFU Survey,March 8th 2005.

Zhang, B., Chen, X., Liu, G., & Cai, Q. (2000). *Agent architecture: a survey on RoboCup-99 simulator teams*. Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393),1, 194-198. Ieee. doi:10.1109/WCICA.2000.859946Y.