

FC PORTUGAL 3D SIMULATION TEAM: ARCHITECTURE, LOW-LEVEL SKILLS AND TEAM BEHAVIOUR OPTIMIZED FOR THE NEW ROBOCUP 3D SIMULATOR

Hugo Marques¹, Nuno Lau¹ and Luís Paulo Reis²

¹*Departamento de Electrónica e Telecomunicações – IEETA,
Universidade de Aveiro,
Campus Universitário,
3810-193 AVEIRO PORTUGAL.*

²*Faculdade de Engenharia da Universidade do Porto - LIACC,
Rua Dr. Roberto Frias, s/n
4200-465 PORTO, PORTUGAL.*

Abstract: This paper aims to describe the agents built by FC Portugal team in the scope of the recent RoboCup Simulation 3D competition. The new 3D simulator released for the event is in its 0.2 version and the documentation about it is almost inexistent. Thus, this paper, besides describing FC Portugal 3D team approach, also describes the major features of rcserver3d simulator.

Keywords: Artificial Intelligence, Multi Agent Systems, Simulated Robotic Soccer, Physical Models, FC Portugal.

1. INTRODUCTION

RoboCup is an international project that aims to contribute to the research on multi agent systems and intelligent robotics (Kitano *et al* 1995). It includes several types of competitions like robotic search and rescuing or soccer playing. Each of these is made on simulated environments or using real robots.

The soccer simulation competition aspires to make a set of agents to play a soccer game, by making them behave as intelligent as possible in order to score more goals than the other team. The first world cup of this competition was in 1997 in Nagoya. Besides the simulation competition this tournament also includes a Medium Size, Small Size and Legged Leagues of Robotic Soccer.

The Simulation competition is based on socserver simulator (Chen *et al* 2002). This simulator establishes a virtual 2D field and virtual robotic players' capabilities. The agents act as clients of the socserver, receiving sensations from the server and sending actions to the server. Each agent controls one distinct player. The simulation is quite realistic including limited vision angle, low-bandwidth player communication, stamina control, noise in every action and sensation, etc. A coach agent is allowed a noiseless vision of the game. This agent may send instructions to other agents in certain moments, like during game pauses.

In 2001 a new contest was added – the coach competition. This competition aimed to improve the strategic part of the game by giving to an agent the ability to advice a team trying to improve its performance.

In 2004 the 3D soccer simulation competition was born. Exactly with the same goals as the 2D soccer simulation, this new competition adds the third dimension to the game seeking to make it more realistic. The 3D simulator goes at its 0.2 version. It is still very incomplete and the documentation about it is almost inexistent. One of the goals of this paper is precisely begin to gather information concerning the server in order to give the agents more accurate and intelligent behaviours.

This paper is organized as follows. Section 2 gives an overview of SPADES which is a generic platform for agent-based simulations used by the 3D Simulator Server. Section 3 gives the basic information about the 3D Simulator Server. Section 4 is related with the agent made by the FC Portugal team. Section 5 is the tests and results section. Section 6 gives an idea of the work that is planned to do in the future. Finally, section 7 is the conclusions section.

2. SPADES

The simulation server is implemented above a platform called SPADES (System for Parallel Agent Discrete Agent Simulation). SPADES is a middleware system for agent-based distributed simulation (Riley 2003). It aims to provide a generic platform to run in multi-computer systems. It implements the basic structure to allow the interaction between agents and a simulated world in such a way that the users do not have to worry about sockets, addresses, etc.

SPADES main features are:

- Agent based execution - support to implement sensations, thinking and actions.
- Distributed processing – support to run the agents-application on many computers.
- Results unaffected by network delays or load variations among the machines – SPADES ensure that the events are processed in the appropriate order.
- Agents can be programmed independently from the programming language – the agents can be programmed in any language once it provides methods to write/read to/from PIPEs.
- Actions do not need to be synchronized in the domain – the actions of the agents can take effect at varying times during the simulation.

2.1 Components organization

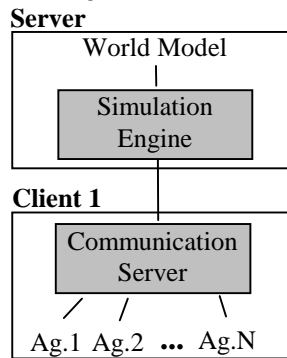


Fig.1 – SPADES Components diagram.

SPADES components are organized in a client-server architecture (Fig.1). The Simulation Engine and the Communication Server are supplied as a part of SPADES; while the Agents and the World Model are built by the user and run upon the formers. The Simulation Engine is a generic piece of software that allows creating specific world models upon it. It runs on the server side and provides the interaction and communication between the agents and their world via the Communication Server. On the same machine it must run the specifications of the World Model that has the characteristics of the environment where the agent will act. Distributed along the clients are the Agents and the Communication Server. The Communication Server must be present in all client machines end provides the communication between the agents and the Simulation Engine. It receives messages from the Agents and sends them to the server and vice-versa. The Agents also run on the client side.

2.2 Sense-Think-Act Cycle

SPADES implements what it calls the sense-think-act cycle in which each agent receives sensations and reply with actions. That means that an agent is only able to act as a reply to a sensation message. Of course that it is capable of requesting its own sensations, but the principle remains - a sensation must always precede an action. Following this, SPADES provides an action called request time

notify that returns an empty sensation (time notify) and by receiving it the agent is able to respond with actions.

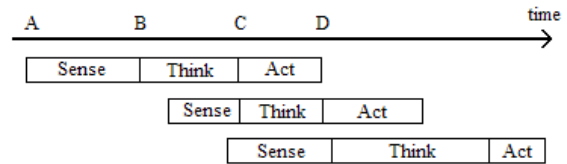


Fig.2 – SPADES Sense-Think-Act Cycle.

Fig.2 depicts the sense-think-act cycle and the time where each of its components runs. From A to B a sensation is sent to the agent. After receiving the sensation (from B to C) the agent decides which actions will be executed; then from (C to D) the actions are sent do the server.

In many agents, the sense, think and act components may be overlapped in time (like in Fig.2); there is just one restriction – the thinking cycles for one agent can not be overlapped. This constraint makes sense, since just a single processing unit is used per agent, and thus, just one sensation at time can be processed.

3. SIMULATION SERVER

As stated before the simulator runs upon the SPADES, and uses ODE to calculate the physical interactions between the objects of the world (Russell 2003). The graphical interface is implemented using OpenGL.

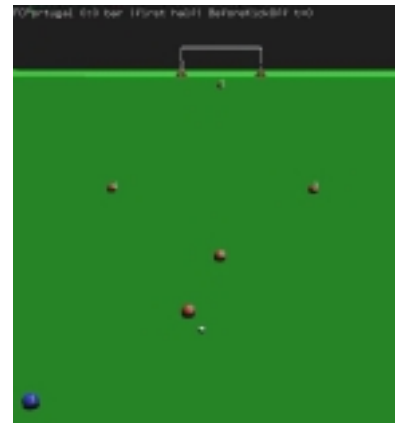


Fig.3 – Snapshot of the 3D Simulator.

The 3D simulation server (Fig.3) (RoboCup Soccer Server 3D Maintenance Group 2003) allows twenty two agents (eleven from each team) to interact with the server in order to play a simulated robotic soccer game. Each agent receives sensations about the relative position of the other players and field goals and other information concerned with the game state and conditions. At this time the information about the positioning of the objects in the world is given by an awkward omnivision that allows the agent to receive visual information in 360 degrees. The agents have the shape of a sphere. Replying each sensation an agent sends actions like *drive* or *kick*. Driving implies applying a force on the body with a given

direction and kicking implies applying a force on the ball radially to the agent. Each sensation is received on every 20 cycles of the server and each cycle takes 10 ms.

3.1 Sensations

There are already several sensations that an agent is able to receive. Every sensation starts with the character 'S' followed by two integers. The first is the cycle in which the message was sent and the second is the cycle in which the message arrived to the agent. A sensation message has the shape:

```
Stime time data
```

where *data* is the string with the information related with the sensation itself.¹

Vision. The vision sensation gives the agent the spatial arrangement of the world objects in the field. World objects are the players, the ball, the goals and flags in the corners. The vision is, on the 0.2 version, omni-directional. The position of the objects is given in polar coordinates relative to the respective agent. The coordinates are given by a distance an x^y angle – theta – and an $(x-y)^z$ angle – phi.

```
Stime time (Vision
  (Flag (id id) (pol d theta phi)) ...
  (Goal (id id) (pol d theta phi)) ...
  (Ball (pol d theta phi))
  (teamname (id id) (pol d theta phi)) ...
)
```

GameState. The game state sensation gives the agent all the information concerned with the game. It gives information about the dimensions of the field, the goals, the ball and the agents. It also gives information about the masses of the world objects and other aspects of the game like: time; play mode; agent number; and if the agent's team is the right or the left one. Here is the format of the given information:

```
Stime time (GameState
  (team side)
  (unum number)
  (FieldLength length)
  (BallMass mass)
  (playmode playmode)
  ...)
```

AgentState. The agent state gives the agent information about its internal state, namely its battery condition and temperature.

```
Stime time (AgentState
  (battery battery)
  (temp temp)
)
```

3.2 Actions

As well as sensations, several actions are implemented that allow an agent to interact with the world. Every action message starts with the 'A' character and it is followed by a string,

```
Adata
```

where *data* contains the information about the action itself.

Create. The first action that an agent must send is the create action. This action allows the server to register an agent and thus establish the communication with it. The action create as the form,

```
A(create)
```

Init. The init message allows the server to receive essential information about the agent, namely its number and its team. If the number is passed as 0, the server automatically attributes a number to the agent. The init action has the format:

```
A(init (unum number) (teamname name))
```

Beam. The beam action allows an agent to move to a given point before game kickoff. Its structure is the following:

```
A(beam x y z)
```

Drive. The drive action allows an agent to move. It applies a force vector (x, y, z) to the center of the agent's body with the maximum length of 100.0 units. It has the format:

```
A(drive x y z)
```

Kick. The kick action allows an agent to kick the ball with a given force intensity and a given direction (see 3.4 *Physics – Kick*). The action must be sent like:

```
A(kick angle force)
```

3.3 Other important communication procedures

The server establishes the communication by sending a *done* ('D') message (Riley *et al.* 2003). When the agent receives this message it should execute its initialization procedures and when it finishes them it must send an *initdone* ('I') message. After that the server starts to send sensations and the agent replying with actions. Every set of actions must finish with a *done* ('D') message.

As already stated, each sensation is received in every 20 server cycles, which means that an agent should only be able to execute actions within 20 cycles intervals. However, this is not quite so, since an agent can ask the server to receive a sensation in a given cycle by sending a *request time notify* (R) message. The format of the message is the following:

```
Rtime
```

¹ Note that one sensation message can have more than one sensation.

where *time* is the time at which the server must reply. This procedure makes the server reply with an empty sensation ('T') at the cycle given. The shape of the message received is:

Ttime

where *time*, again, is the server cycle in which the message was sent.

By receiving this sensation the agent is able to respond with action messages.

2.4 Physics

Each player has the shape of a sphere with radius $R_p = 0.22m$ and a mass of $M_p = 75kg$. The ball has also a sphere shape with a radius $R_b = 0.111m$ and a mass of $M_b = 0.4kg$.

Drive. To move an agent a force vector is applied to the centre of its sphere for 10ms. An agent is only able to move itself if it is in contact with the ground. The maximum length of vector force is 100 units. When an agent is moving it is under the influence of drag force of :

$$F_{drag} = 0.3 * Vel \quad (1)$$

with the ground that applies to it a torque and forces it to rotate (Fig.4).

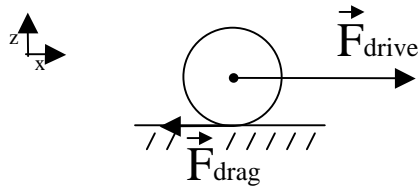


Fig.4 – The forces on the body of an agent when an Fdrive force is applied to it.

Kick. To be able to kick the ball an agent must be at a maximum distance of $R_p + R_b + 0.04$ to the ball.

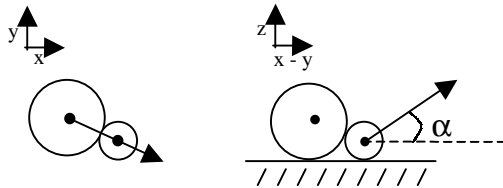


Fig.5 – Diagram of how the kick is performed.

The kick action receives two arguments – the intensity of the force applied to the ball and the angle that the force does with the x-y plane. The direction on the x-y plane is radial to the agent's body (Fig.5). When a kick is performed a force and a torque are applied to the ball.

4. FCPORTUGAL AGENT

The FC Portugal 3D Basic Agent is not very sophisticated given the short time since the release of the 0.2 version of rcssserver. Nevertheless, a simple

organization was enough to start thinking about the structure that it will have in the near future.

4.1 Agent's Architecture

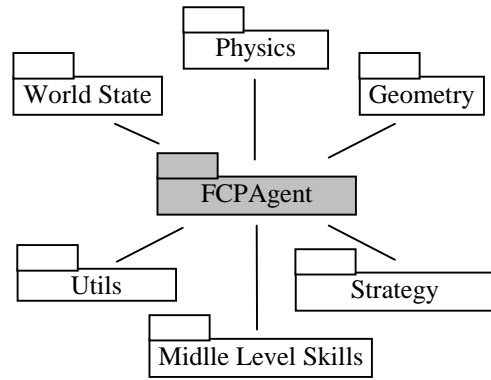


Figure 6 – Modular division of FC Portugal 3D Basic Agent.

The agent structure includes six main modules/packages that cover different parts of its functioning (Fig.6).

World State. The “World State” module (Fig.7) is used to keep the agent's world state representation up to date. Information like the position of the ball and the other players (and itself) or position of the goals, or even information like the score and time are stored in this module.

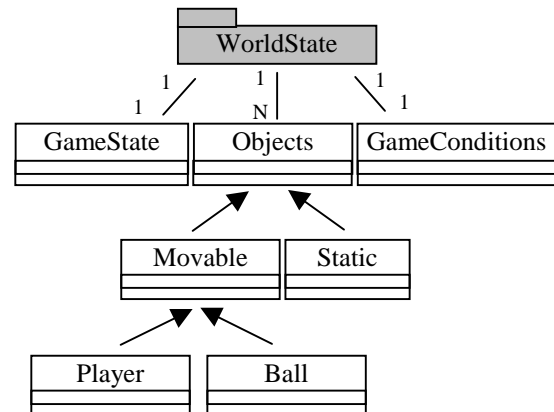


Fig.7 – FC Portugal scheme representing the state of the world.

There is already a draft of the structure that will be used. The world state class is divided into three main classes - the *GameState* class, the *GameConditions* class and *Objects* class. The former will have the information concerned with the facts of the match, that is, information like time, score, which team must start playing, etc. The second will deal with the data related with the temperature, field dimensions, etc. The later is a super class to keep information about the objects of the world. It is divided into two subclasses – *Static* and *Movable* objects. Static objects are the goals for example or the flags in the corners, while movable objects are the players and the ball.

It is desirable to use the same structure to anticipate next world states, in order to have the information about where the movable objects will be positioned in the near future.

Strategy / Positioning. The “Strategy” module applies the SBSP (Situation Based Strategic Positioning) already used by FC Portugal team on the 2D simulator (Reis and Lau 2001). Basically, the method consists in assigning the strategic position on the field to each agent given the position of the ball and the current situation. The player that runs to the ball is the one that has the closest distance from its strategic position² to the ball. Each agent is differentiated by its position (Reis, Lau and Oliveira 2001).

Low Level Skills. The “Low Level Skills” module implements procedures that allow the agent to perform basic movements like run and kick (see 3.4)

Physics. The “Physics” model is used to simulate, on the agent’s side, the calculations made by the server. It encloses methods to compute the results of physical interactions like position, velocity, acceleration of bodies or breaking distances, forces, etc.

Geometry. The “Geometry” module is used to make easier the execution of geometrical calculations. It is used to compute data concerned with distances, vectors, etc.

Utils. The “Utils” module is used to make some tasks simpler like send/receive information to/from the server, parse the messages received, compose messages to send and write log files.

4.2 Agent sketch

```
Do forever
  msg = GetMessage();
  Case msg[0]:
    'D': ProcessInitMessage(); Break;
    'S': ProcessSensation();
          Think(); Act(); Break;
    'K': ProcessThinkingMessage(); Break;
  End Case;
End Do Forever;
```

The agent main code consists in an infinite cycle in which it receives messages, processes the messages (“think”) and replies with actions.

4.4 Low Level Skills

Using the latest release of the server3D, to kick the ball to some point, an agent must be positioned behind the ball at a maximum distance of d and on the extension of the line that connects the ball with the point. This way of kicking implies a good accuracy on the position of the player since a small

error may imply to kick the ball to a completely different place.

To calculate the way to make an agent stop in a given place, the first measure to do is the maximum distance B (breaking distance) that takes for a body with mass m and velocity V_0 to stop, given a maximum force (F) that can be applied in the opposite direction of the velocity. Starting with the movement equations:

$$x = x_0 + V_0.t + 1/2.a.t^2 \quad (2)$$

and

$$V = V_0 + a.t \quad (3)$$

It must be calculated the x in (2). Given that $x_0 = 0$, since the starting position is not important because the goal is the distance and not the final point, the breaking time t and the acceleration a performed by the force F applied must be found. The acceleration can be found by³,

$$a = -F / m \quad (4)$$

and the time t can be found by (3) since the final velocity desired V is 0,

$$t = -V_0 / a \quad (5)$$

Substituting (4) and (5) in (2) one has,

$$x = 0,5.V_0^2.m / F \quad (6)$$

Finally, one has to calculate the y coordinate using the one-dimensional formula (6) since the moving is performed in two dimensions. Given this, the algorithm is:

```
Function MoveTo (myVel, myPos, myMass,
                endPoint)
  // vector from myPos to endPoint
  direction = endPoint - myPos;

  d = Length(vector);
  f = min(100, ForceLen(myPos, endPoint));
  b = min(100, BreakLen(myPos, endPoint));
  force = getVector(direction, f);
  bforce = getVector(direction, b);

  bdist = getBreakDistance(myVel,
                          myPos, myMass,
                          endPoint, b)

  if (abs(bdist.x) >= abs(vector.x))
    force.x = -bforce.x;

  if (abs(bdist.y) >= abs(vector.y))
    force.y = -bforce.y;

  return force;
End Function;
```

The force intensity applied is constrained by the distance to the end point. This is done in order to avoid applying too much force when the agent is close the ball and consequently pass the ball. The same is done to the breaking force witch depends on

² not its actual position.

³ The minus signal applied to F is because the force applied is opposite to the direction of velocity.

the distance to the ball, in order to avoid breaking with such intensity that the body, instead of break, gets farther from the ball.

4.3 Agent behaviour

```

If (gamestate = BeforeKickOff)
    pos = GetPositionSBPS(mynum);
    MoveTo (pos);
Else If (gamestate = TheirKickOff())
    DoNothing();
Else If (gamestate = OurKickOff())
    If (WhoPlaysTheBall() = mynum)
        If (InPositionToKick(mypos))
            KickToGoal();
        Else RunToKickablePosition(); EndIf;
    Else DoNothing(); EndIf;
Else If (gamestate = PlayOn)
    If (WhoPlaysTheBall() = mynum)
        If (InPositionToKick(mypos))
            KickToGoal();
        Else RunToKickablePosition(); EndIf;
    Else
        pos = GetPositionSBPS(mynumber);
        MoveTo(pos);
    EndIf;
EndIf;

```

The agent behaviour is very much tied with the SBPS originally made by FC Portugal team. Before the game starts each agent requests the position that it must occupy on the playing field given its number. The SBSP has the advantage to make easier for an agent to know where their team players are positioned without having to calculate their real position; because each agent is able to know the position of its companions simply by running the SBSP algorithm for all the team players.

If the kick-off is assigned to the opposing team the agents must do nothing until the ball is touched by one of the opposing players. If the kick-off is assigned to the FC Portugal team, the agent with the closest distance from the position given by the SBSP to the ball, starts to run to a position that allows it to kick the ball to the goal. Arriving there it kicks the ball. The other players do not do anything until the former touches the ball.

After the ball is touched by some agent of the starting team the game state is changed to “PlayOn” and the game starts. At this state each player moves to the respective position by running the SBPS algorithm. The only exception is player that has to play the ball. That player has to run to a position that allows it to kick the ball to the goal and after getting there kick it.

5. TESTS AND RESULTS

So far two main tests have been performed. On the first test just an agent was put on the field in order to check its behaviour without opponents. The goal was to make it run and kick the ball aiming to score a goal. The results were found to be very encouraging since the agent scores almost every time.

The second test was done by placing the eleven players of the FC Portugal team on the field and the two agents that came with the simulator as the opponent team. The test consisted in making the FC

Portugal players move according with the SBSP and making the team score a goal. The success rate decreases but still our team is able to score goals often. The main obstacle is the opponent agents; when they are trying to get closer the ball, they keep pushing the opposing agents to far way from it. Even so the team positioning revealed to be easily applied.

6. FUTURE WORK

From this point on the FC Portugal team wishes to use some ideas that proved to be successful on the 2D Simulation League to the 3D one namely, the SBSP, the Low Level Skills evaluators, and others like the use of intelligent communication according with the advances performed on the server.

7. CONCLUSIONS

The SBSP showed to be flexible enough to be used by the new agents with very few changes and the current knowledge about the server proved to be already reasonably good. Thus, even with the simulator server very far from being complete and ready to use, the team built so far makes FC Portugal very confident in a very good result on the competition in Lisbon 2004.

ACKNOWLEDGEMENTS

This research is supported by FCT-POSI/ROBO/43910/2002 Project – “FC Portugal New Coordination Methodologies applied to the Simulation League”.

REFERENCES

- Riley, Patrick (2003). *SPADES for Parallel Agent Discrete Event Simulation*. <http://spades-sim.sourceforge.net/>
- Chen, Mao *et al* (2002). *RoboCup Soccer Server*. <http://sserver.sourceforge.net/>
- Smith, Russell (2003). *Open Dynamics Engine v0.039 User Guide*. <http://opende.sourceforge.net/>
- RoboCup Soccer Server 3D Maintenance Group (2003). *The RoboCup 3D Soccer Simulator*. <http://sserver.sourceforge.net/>
- Kitano, Hiroaki *et al* (1995). *RoboCup: The Robot World Cup Initiative*. In *Proc. of IJCAI-95 Workshop on Entertainment and AI/Alife*.
- L.P. Reis, N. Lau, E.C. Oliveira (2001). *Situation Based Strategic Positioning for Coordinating a Team of Homogeneous Agents*. In: *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, (Markus Hannebauer, Jan Wendler, Enrico Pagello. (Ed)), LNCS 2103, pp. 175-197, Springer Verlag, Berlin.
- L.P. Reis and N. Lau (2001). *FC Portugal Team Description: RoboCup 2000 Simulation League Champion*. In: *RoboCup-2000: Robot Soccer*

World Cup IV. (Peter Stone, Tucker Balch and Gerhard Kraetzschmar. (Ed)), LNAI 2019, pp. 29-40, Springer Verlag, Berlin.