Conference Paper Title

Taylor Abraham
Systems and Computer Engineering
Carleton University
Ottawa, Canada
email address or ORCID

David Bascelli
Systems and Computer Engineering
Carleton University
Ottawa, Canada
email address or ORCID

Joshua Fryer
Systems and Computer Engineering
Carleton University
Ottawa, Canada
joshfryer@cmail.carleton.ca

Zein Hajj-Ali
Systems and Computer Engineering
Carleton University
Ottawa, Canada
email address or ORCID

***Abstract*—A team of Robocup soccer-playing agents was implemented using the Jason framework[1], creating a team of five Jason agents which communicate with each other and execute plans; these agents perceived and acted upon the virtual soccer field by interfacing with Krislet entities. The team demonstrated cooperative behaviour in scoring on the opposing net and preventing scoring on the friendly net.**

I. Introduction

Our goal was to design and implement a multi-agent system of agents that communicated and exhibit coordination to (reasonably) successfully play soccer in the Robocup soccer simulation. Jason was chosen as the framework because it extends the already-useful AgentSpeak language's basis for belief-desire-intention (BDI) programming with inter-agent communication and a shared abstract environment to help manage the agents' percepts and execute their actions.

II. Design

A. BDI Architecture

Our agents use a Belief-Desire-Intention (BDI) design; they hold beliefs (things they know about the world and themselves, such as their position relative to objects, or messages they have been given by other agents), desires or goals (a collection of beliefs they wish to hold and maintain, such as having kicked the ball towards the opposing net), and intentions or plans to reach those goals given their current set of beliefs.

B. Goals and Planning

We designed two types of player agent, to fulfill different roles: offensive players ("strikers"), and defensive players ("defenders"). Their decision processes are as follows:

1) Strikers: The first striker to see the ball will take on the role of primary attacker, and will attempt to kick the ball into the net. It notifies the other strikers of this, and they take on an assisting role and run towards the opposing net. When the attacking striker kicks, it notifies all strikers and the distance calculation and role assignment happens again. Figure 2 is a sequence diagram illustrating the interplay and communication of the striker agents

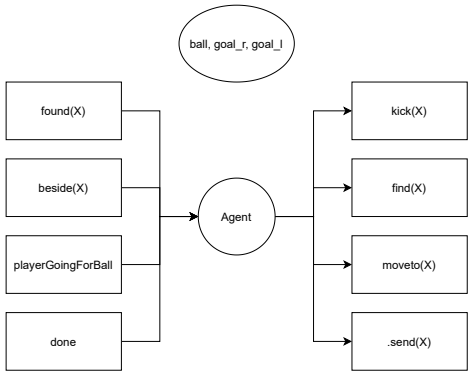

Fig. 1. Some key percepts and actions used in our agents

2) Defender: Each defender looks for its own net and runs towards it. Once there, they then defend the net by searching for the ball and, if the ball is within a certain distance of them, approaching the ball and kicking it away towards the opposing net.

III. Implementation

Three discrete components are involved in controlling agents: the Jason framework, the Java environment, and the Krislet player.

A. Jason

The abstract player behaviours were first formalized in flowcharts (Figure 3, Figure 4), and then adapted into AgentSpeak plans, which define agent actions based on their beliefs and the context they can observe. The Jason framework provides the cognitive model for executing these plans; the agent is fed percepts about the soccer simulation (such as the location of the ball, the opposing goal, etc.) and adds appropriate beliefs to its knowledge base.

[1]http://jason.sourceforge.net/wp/

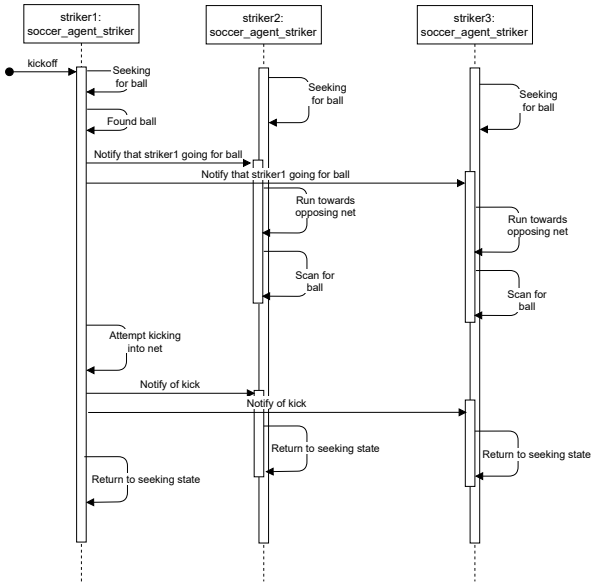

Fig. 2. Communication between striker-type agents

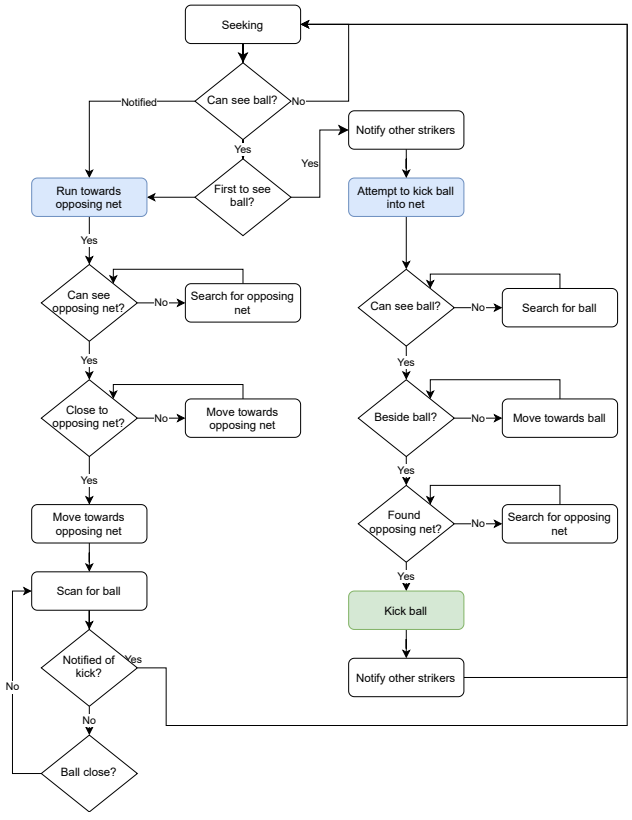

Fig. 3. Flowchart for striker-type agents

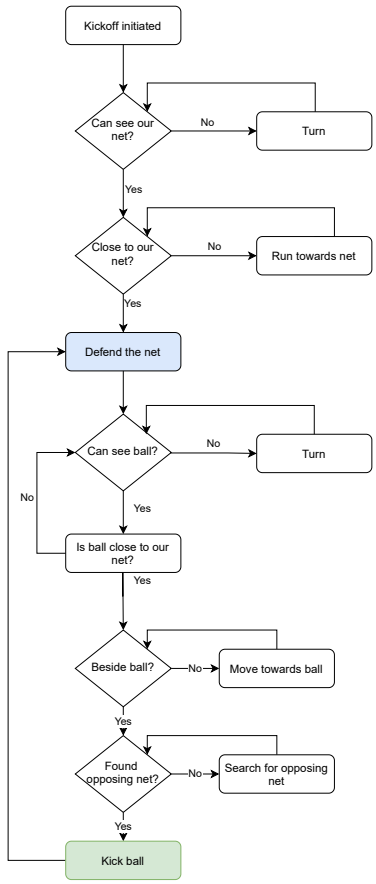

Fig. 4. Flowchart for defender-type agents

B. Environment

The environment is a Java class that handles the interface between the Robocup soccer environment server and the Jason framework. At program startup it initializes all Krislet objects and has callbacks to map Krislet memory objects to Jason percepts.

C. Krislet

The Krislet program was provided as a demo agent to run in the Robocup simulation. This program was modified to be a stand alone agent for interfacing with the simulation, acting as a puppet for the Jason agent. Each Krislet object represents one Jason agent, and when created opens a socket to the soccer environment server and registers the player. Actions chosen by the Jason agent are executed by the Krislet object following the Command design pattern, which also feeds back percepts to the Jason agent.

IV. Results

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent auctor, urna eu suscipit convallis, nisi urna tempus risus, eget pulvinar elit nisl ac nulla. Integer arcu augue, vehicula feugiat lobortis elementum, interdum ut augue. Nunc egestas nulla eget lorem pulvinar commodo. Nunc egestas turpis efficitur aliquet euismod. In eleifend sodales commodo.

Maecenas non posuere lorem. Nullam lacus ligula, sollicitudin ac est a, congue tempor tellus. Suspendisse cursus felis fringilla, dignissim sem feugiat, pulvinar risus. Fusce nec tempus lacus. Donec ac massa at nunc pellentesque aliquet. Nam sollicitudin posuere enim, non consequat arcu cursus at.

V. Conclusion

Aenean eget hendrerit nunc, vitae dapibus risus. Sed ut ultrices quam, at facilisis lectus. Fusce dapibus condimentum consectetur. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus facilisis vulputate odio imperdiet condimentum. Cras accumsan nunc ac sagittis sodales.

VI. Appendix

```
// Agent soccer_agent_defender in project agent_soccer_team

/* Initial beliefs and rules */

/* Initial goals */

!stayback.

/* Plans */

// How to defend
// Go to friendly net
// Look for ball
// Ball is closer than a threshold, go to ball and kick away

+!defend : beside(ball) & found(goal_r) <- kick(goal_r); -amBesideGoal; !stayback.
+!defend : found(ball) & beside(ball) & not found(goal_r) <- find(goal_r); -amBesideGoal; !defend.
+!defend : found(ball) & not beside(ball) <- moveto(ball); -amBesideGoal; !defend.
+!defend : true <- find(ball); !defend.

+!stayback : found(ball) & close(ball) & amBesideGoal <- find(ball); !defend.
+!stayback : found(goal_l) & close(goal_l) <- find(ball); +amBesideGoal; !stayback.
+!stayback : found(goal_l) & not close(goal_l) <- moveto(goal_l); !stayback.
+!stayback : true <- find(goal_l); !stayback.

```

Fig. 5. AgentSpeak code for defender-type agents

```
// Agent soccer_agent_striker in project agent_soccer_team

/* Initial beliefs and rules */

/* Initial goals */

// Start with the seeking goal where our objective is to find the ball OR hear from another player
// that they are going for the ball
!seeking.

/* Plans */
-itIsSelf <- -playerGoingForBall; -done.
+playerGoingForBall : not itIsSelf <- .print("Another player is getting the ball, I will assist them").
+done[source(A)] <- -done[source(A)]; -playerGoingForBall; -itIsSelf; .print("Other agent has kicked the ball. I wil

+!seeking : playerGoingForBall & itIsSelf <- !score.
+!seeking : playerGoingForBall & not itIsSelf <- !assist.
+!seeking : not playerGoingForBall & found(ball) <- +itIsSelf; .send([soccer_agent_striker1,soccer_agent_striker2,so
+!seeking : not playerGoingForBall & not found(ball) <- find(ball); !seeking.

+!score : beside(ball) & found(goal_r) <- kick(goal_r) .send([soccer_agent_striker1,soccer_agent_striker2,soccer_age
+!score : beside(ball) & not found(goal_r) <- find(goal_r); !score.
+!score : found(ball) & not beside(ball) <- moveto(ball); !score.
+!score : true <- find(ball); !score.

+!assist : not done & close(goal_r) <- find(ball); !assist.
+!assist : not done & found(goal_r) & not close(goal_r) <- moveto(goal_r); !assist.
+!assist : not done <- find(goal_r); !assist.

```

Fig. 6. AgentSpeak code for striker-type agents