2Jason4JSON: A Multi-Agent Team for RoboCup

Taylor Abraham
Systems and Computer Engineering
Carleton University
Ottawa, Canada
taylorabraham@cmail.carleton.ca

David Bascelli
Systems and Computer Engineering
Carleton University
Ottawa, Canada
davidbascelli@cmail.carleton.ca

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
zeinhajjali@cmail.carleton.ca

***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 communicate and exhibited 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 takes on the role of primary attacker, and attempts 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 that it has done so, and the role assignment happens again. Figure 2 is a sequence diagram illustrating the interplay and communication of the striker agents.

2) Defender: Each defender looks for their own net and runs towards it. Once they are near the net, they then defend the net by searching for the ball and, if the ball is within a certain distance of them, approaches the ball and kicks it away towards the opposing net.

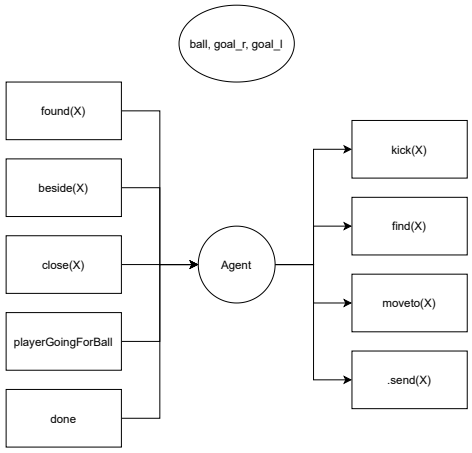

Fig. 1. Some key percepts and actions used in our agents

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/

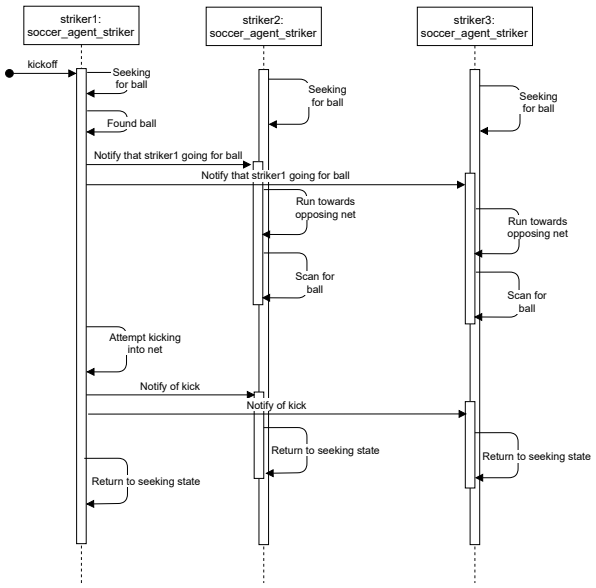

Fig. 2. Communication between striker-type agents

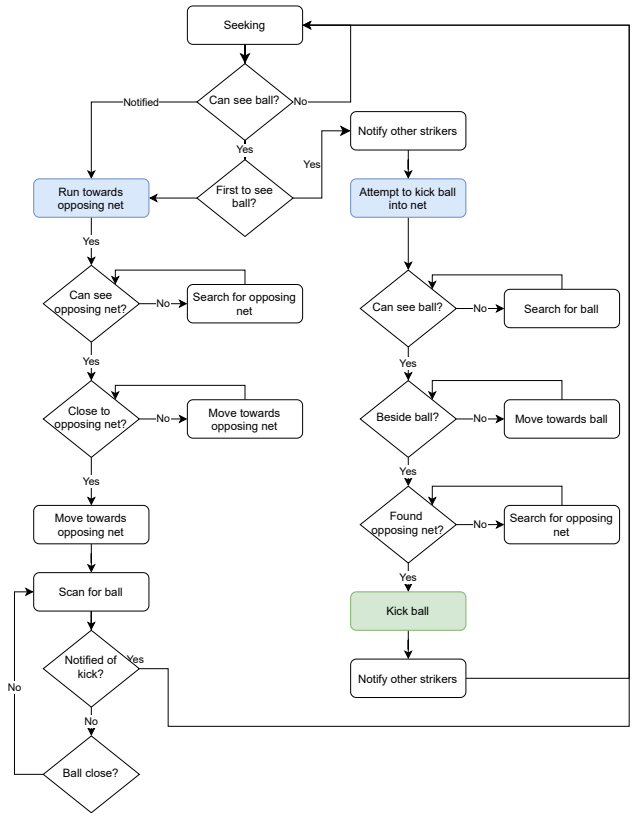

Fig. 3. Flowchart for striker-type agents

B. Environment

The environment is a Java class that serves as 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.

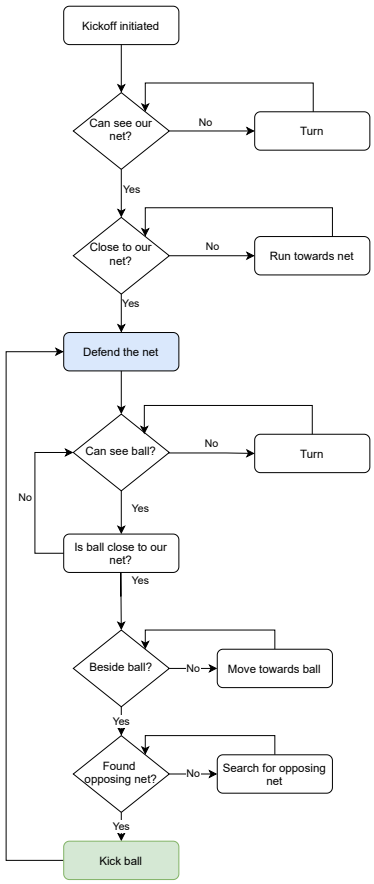

Fig. 4. Flowchart for defender-type agents

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

Basic functionality was first tested with the team playing standalone. One attacker successfully moved to the ball and others to the opposing net to support. Defenders moved to their own goal and waited for the ball to approach. After basic functionality was confirmed, a test game was played against a team of five default Krislet agents. Our team was reliably defeated by the default agents. While our solution was a good proof of concept, a few flaws became evident. Our Jason-controlled agents turn very slowly and have difficulty finding the ball. Attackers were able to keep the ball within their vision as they pursued it, but defenders struggled to find the ball

before the default agent had already kicked it into the net. In addition, the most important task was to run after the ball and kick it into the target net but, due to the random nature of the movement speeds, with five default agents running after the ball, at least one of them usually arrived first, vastly outweighing the advantage of having agents open at the net. Finally, our agents only know how to kick the ball towards the opposing net, and do not account for opposing agents in the way; this leads to frequent interceptions by the opponents.

V. Conclusion

We implemented a team of agents that successfully coordinated to achieve their joint goal of scoring on the opposing side and preventing a goal on their net. Further development of this system would make the strikers properly communicate their distance to each other and have the closest agent move to the ball. Additionally the strikers should properly pass to other strikers rather than kick the ball directly at an opponent.

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