Intelligent Soccer Playing Agents using Jason

Chase Scott
Carleton University
101092194

Diankun Sha
Carleton University
101289268

Abinandhan Selvaraj
Carleton University
101278603

***Abstract*—We introduce a BDI model tailored for RoboCup teams through the utilization of the Jason BDI framework. This approach is aimed at defining agents by employing belief and desire sets to guide intention selection. Our focus lies in outlining a software architecture using Jason for this purpose. The model is structured to create two distinct agent roles – keeper and striker - by formulating agent speak code that are unique for each role. The framework facilitates the reasoning process based on beliefs and desires, resulting in varied agent behaviors even when faced with identical perceptions. Future work involves the handling of more percept information to create more complex behaviors between agents.**

Keywords—Jason, BDI, Krislet, RoboCup, Agent.

I. INTRODUCTION

In this course, we have explored a range of concepts, methodologies, and techniques essential for constructing intelligent agents. Our project centers on the development of agents within the RoboCup simulation environment, utilizing the Jason BDI (Belief, Desire, and Intention) framework in conjunction with the Krislet client. These agents act as clients to the RoboCup server, sending commands to the server. The server synchronizes and reflects their actions in the virtual environment.

Our approach harnesses the agents' perceptions within the RoboCup world as foundational beliefs, processed and managed by Jason. By employing the BDI framework, coupled with Jason's advanced AgentSpeak Language, we have crafted sophisticated plans that guide our agents' actions and decision-making processes. Jason serves as the cornerstone for reasoning, enabling our agents to respond dynamically to the simulation environment.

The forthcoming sections detail various aspects of our project:

Section II: Delve into the architecture of our implementation, elucidating how we have integrated Jason with Krislet.

Section III: Discuss the specific percepts and actions utilized in our agents' programming.

Section IV: Focus on the development of two key agent roles – the striker and goalkeeper – and the strategies underlying their plans.

Section V: Present the results of our implementation, and how to run our code.

Section VI: Reflect on our findings, discussing both the achievements and potential avenues for future research and development.

Section VII : List of references that have informed and supported our project.

This project not only demonstrates the practical application of theoretical knowledge gained in the course but also contributes to the evolving field of autonomous agent development in simulated environments.

II. ARCHITECTURE

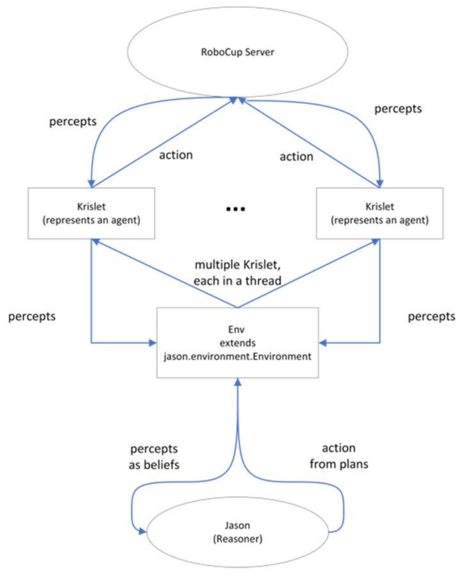
RoboCup Server
percepts
action
action
percepts
Krislet (represents an agent)
...
Krislet (represents an agent)
multiple Krislet, each in a thread
percepts
percepts
Env extends jason.environment.Environment
percepts as beliefs
action from plans
Jason (Reasoner)

Fig.1. Architecture

The design of the high-level architecture is shown above. In order to integrate Jason with the RoboCup server, the customized class, Env, extends the Jason Environment class and overrides the init and executeAction functions. In the init function, instances of Krislet are created as their own distinct thread and connections are established with the Robocup server. These threads are then managed by this environment class, which takes in percept information, passes it to the reasoner, and returns actions for each agent to perform. In the executeAction function, different actions are defined and handled, which are detailed in the following section.

The Env class also has a custom method which receives perceptions (percepts) from agents. It first clears the percepts of the specific agent it received the percept from, then updates the percepts in Jason with the custom percept representation. This representation is done in the Brain.java class, which takes in information of the surrounding and constructs these custom String representations for the reasoner to use. The percepts received by the Env are then treated as beliefs of agents and are used for reasoning.

III. Perecepts and actions

For the formulation of the project, we defined many custom percepts for the agents, along with actions defined in Krislet that can be taken.

A. *Percepts*

Our custom generated percepts can be seen in the following table:

TABLE I
Explanation of perceptions

Perception	Meaning
ball(D, A)	This perception indicates that the ball can be seen at a certain D (distance) and A (angle) from the player.
ball(lost)	This perception indicates that any perception of the ball has been lost.
goal(S, D, A)	This perception indicates that a net has been seen. Since there are two nets, S is included, which indicates which side the net belongs to. It also indicates the D (distance) and A (angle) from the player,
goal_S(lost)	This perception indicates that any perception of the goal S has been lost.

B. *Actions*

Available actions that the agent can take can be seen in the following table:

TABLE II
Explanation of actions

Perception	Meaning
turn(<angle>)	This action turns the agent by the provided angle.
dash(<power>)	This action causes the agent to dash forward at the provided power.
kick(<power>, <angle>)	This action causes the player to kick the ball at the desired power and angle, provided there is a ball close enough to kick.

IV. Plans in ASL

Our agents have two specified roles: striker and goalie. Our 4 forwards all use the striker.asl file, which defines their attacking behavior, and our goalie uses the goalie.asl file, which defines their defensive behaviour. They are programmed using Jason's agent speak language, and operate under the closed-world assumption. This implies that if a belief is not explicitly stated in our knowledge base, it can be assumed to not be satisfied.

A. *Goalie*

Our goalie has an initial belief: out_of_net. This drives its initial reasoning, wherein it assumes it is out of its net and attempts to return to it to cover potential shots. Using percept information, it is able to reason using the provided plans to first get vision of its own net, and then dash towards the center. Here is an example of it receiving a percept, reasoning about beliefs related to its world including if it is facing the goal, and if it believes it is out of the net, and then calling the related plan to move forward:

+goal(S, D, A) : facing(goal) & out_of_net <- !move_forward.

Once it reaches the net, it simply remains in position to attempt to block incoming shots.

B. *Striker*

Our strikers follow a very similar behavior to the original Krislet agents. They take percept information regarding the ball and the opposing net and attempt to get to the ball and kick it towards the opposing team's net. It first attempts to find the ball, once it does, it runs towards the ball until it reaches it. It then looks for the opponent's goal and once found, kicks it towards it. Here we see a variation of the look_for_goal plan, where if we believe we are facing the goal, we kick towards it at full power:

+!look_for_goal : facing(goal) <- kick(500, 0).

After this plan has concluded, it then repeats for the remainder of the game.

V. Results

A. *Run Code*

To run the code, first download and extract the zip file called robocup. This contains all of the relevant code for the project. Then, start up the related robocup server and monitor to allow the agents to be spawned in. Finally, navigate to the directory "/robocup" and simply type "jason robocup.mas2j" to run the code. This uses Jason's new CLI to begin running the agents. Ensure that the necessary path variables have been set for your system, and you are using a shell tool like GitBash if issues are encountered.

B. *Results and findings*

When running the code, we see similar behavior as to what is described in our agent speak code. Our designated goalie agent immediately returns to the net and stays there, while our strikers attempt to reach the ball and shoot towards the enemy net. Testing of the programs against the original Krislet yields marginal improvements, as the goalie position manages to block a good number of incoming shots, with similar production from offense. We can see an example position here:

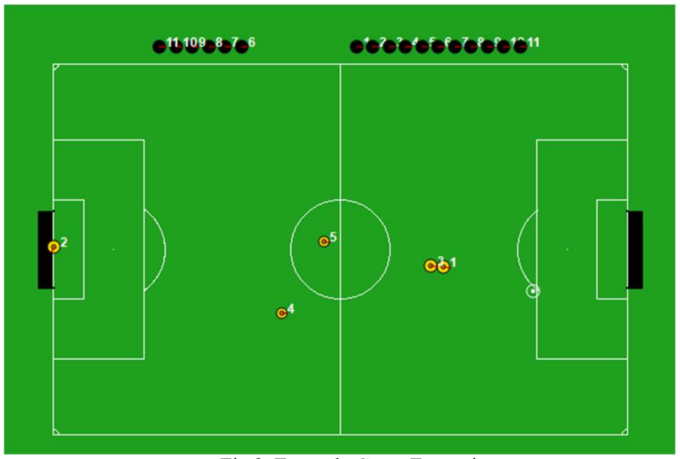

Fig.2. Example Game Formation

VI. Discussion

The implementation of agents using the Jason BDI framework within the RoboCup simulation environment demonstrates a promising approach to defining distinct agent roles through beliefs and desires, fostering collaboration among them. The agents display basic behaviors, but it shows the potential of the Jason BDI framework in facilitating the creation of sophisticated agent systems for complex environments like RoboCup.

There's a lot of room for improvement. For instance, Krislet's hear() function could be used for agent communication. Jason's .tell action can also enable agents to share messages. These features could improve teamwork among agents and lead to more advanced behaviors.

Three selection functions in the TS class in Jason could also be overridden to give priority to some goals. This would make agents more committed to some goals. The default three selection functions just poll the queue, so the behaviours of agents heavily depend on the plan order and event order.

The goalie also could be improved with behaviour to allow it to clear the ball, and to come out and challenge attackers. Additionally, all agents could benefit from increased percept information, allowing them to take more factors about their environment into account when making decisions.

VII. References

[1] Autonomous Systems Sistemi Autonomi, "Programming Intentional Agents in AgentSpeak(L) & Jason," [Online]. Available: https://core.ac.uk/download/pdf/17198968.pdf. [Accessed: 2-Dec-2023].

[2] J. Wiley & Sons Ltd., "Programming Multi-Agent Systems in AgentSpeak Using Jason," Oct. 2007. [Online]. Available: https://jason.sourceforge.net/jBook/SlidesJason.pdf. [Accessed: 2-Dec-2023].

[3] "Multiagent systems," [Online]. Available: https://ktiml.mff.cuni.cz/~pilat/en/multiagent-systems/. [Accessed: 3-Dec-2023].

[4] "jason-lang/jason," GitHub. [Online]. Available: https://github.com/jason-lang/jason. [Accessed: 7-Dec-2023].