Università di Genova

June 2024

Jason Multi-Agent System for City Safety: Agents Cooperating to Arrest Criminals

Multi-Agent Systems Course Final Project
Creative Project Classified as Hard

Giulia Benvenuto - 4678610

INTRODUCTION

Project Overview

The project utilizes Jason, an interpreter for AgentSpeak, to develop a Multi-Agent System that simulates a city environment.

In this environment, four types of agents are implemented, each exhibiting a different behavior. Police, civilian, and clue agents cooperate to detect and arrest criminal agents to ensure safety in the city.

PROBLEM STATEMENT

City Safety Problem

Safety is a key issue in city areas, affecting the happiness of the residents. Thus, it's reasonable to think that civilians living in the city will collaborate with police to enhance their personal safety and contribute to the overall security of the environment.

Why a multi-agent system is suited for this problem

MAS aligns well with the City Safety Problem due to its ability to manage complex interactions among multiple agents who cooperatively work toward the common goal of ensuring city safety by arresting criminals.

DEVELOPMENT FRAMEWORK (1)

Jason

Jason is an interpreter for an extended version of AgentSpeak and is crucial in this project for defining and managing the behavior of the agents: police, civilian, clue, and criminals.

It provides a framework for these agents to interact dynamically within the simulated city environment, allowing them to manage their beliefs, goals, and actions based on specific perceptions and objectives.

DEVELOPMENT FRAMEWORK (2)

Java

Java supports Jason, enhancing its capability to integrate and extend functionalities within the multi-agent system.

This integration is essential for adding dynamic percepts to agents as they navigate the simulated city environment, and supports the development of core system components (CityEnvironment, CityModel and CityView java classes) that manage the environment, model interactions, and visualize agent movements.

ENVIRONMENT (1)

Simulated City Environment

The City Environment in the MAS serves as the dynamic stage where agents perform their roles and interact. It is implemented through a structured architecture that involves the use of three Java classes that exted Jason classes:

- CityEnvironment.java → extends Jason.environment.Environment
- CityModel.java → extends GridWorldModel
- CityView.java → extends GridWorldView

ENVIRONMENT (2)

CityEnvironment.java

Instantiate and manage the city simulation environment where agents interact and navigate. Essential functions of this class include initializing the simulation environment and handling the movement of police agents.

CityModel.java

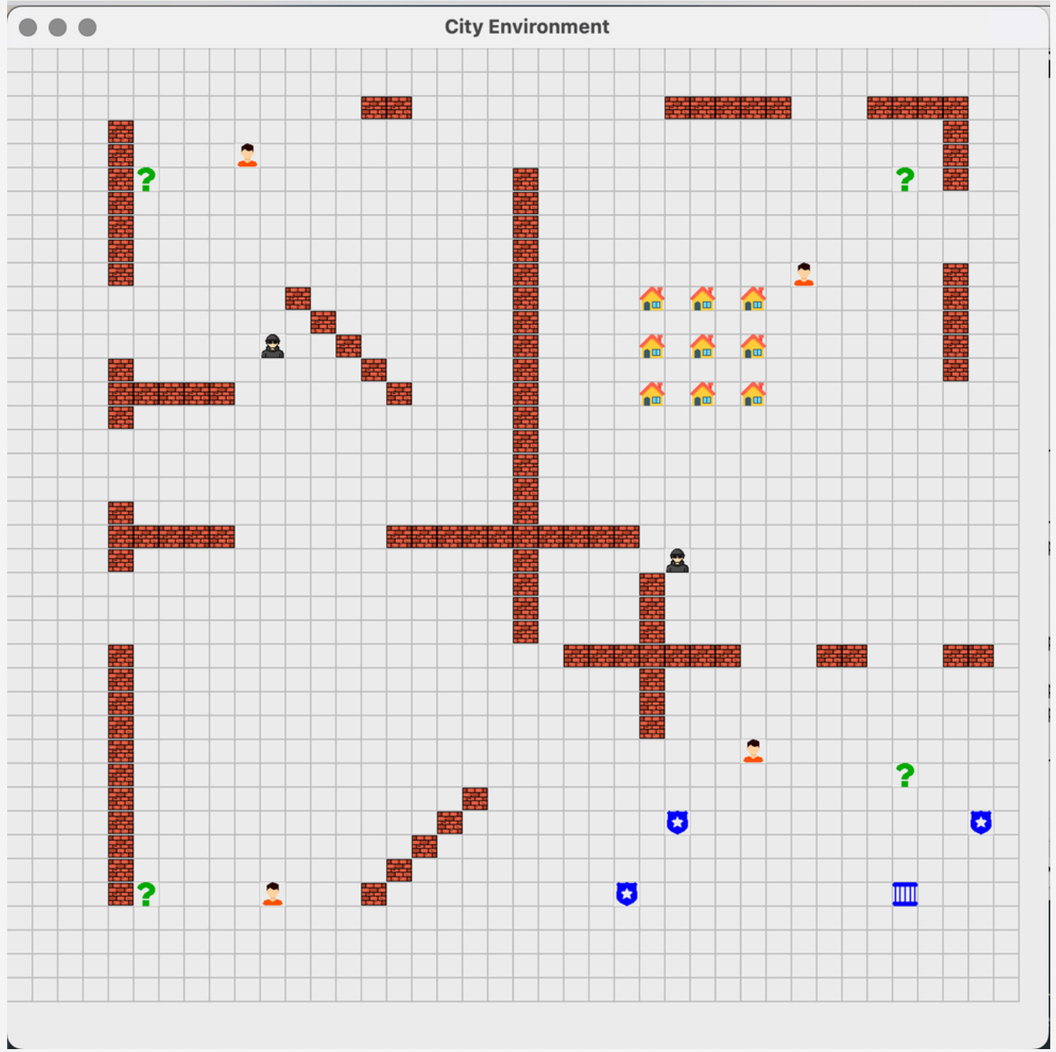
Responsible for creating and managing the 40x40 city grid, it displays both dynamic and static entities within the environment (Obstacles, Jail and Agents).

CityView.java

Designed to visualize the City Environment, this class displays agents and obstacles on the grid. It manages and updates the visual representation in response to changes within the model.

ENVIRONMENT (3)

City Environment Result



Wall Obstacle

House Obstacle

Jail

One criminal in jail

Two criminals in jail

Police Agent

Police Agent Escorting

Civilian Agent

Clue Agent

Criminal Agent

AGENTS (1)

Behavior of Police Agents

- Three police agents.
- Complex and dynamic position.
- Navigate the city following the shortest path computed with the A* algorithm.
- At each step, look around the 8 neighboring cells.
- If they find a civilian agent take the clue position (X, Y) from it and next move there.
- If they find a clue agent collect the X or Y of a criminal.
- If both X and Y of a criminal has been found, they move there.
- If they find a criminal, arrests him, escort him to the jail and broadcast other agents that a criminal has been arrested.

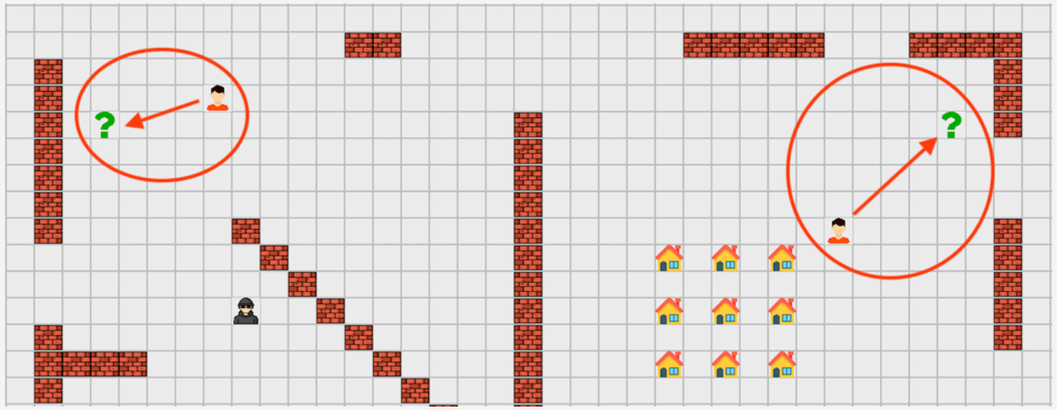
 **Police Agent Icon**

 **Police Agent Icon while Escorting**

AGENTS (2)

Behavior of Civilian Agents

- Four civilian agents.
- Static position.
- If they are found by a police agent, communicate to him the position (X, Y) of their closest clue.



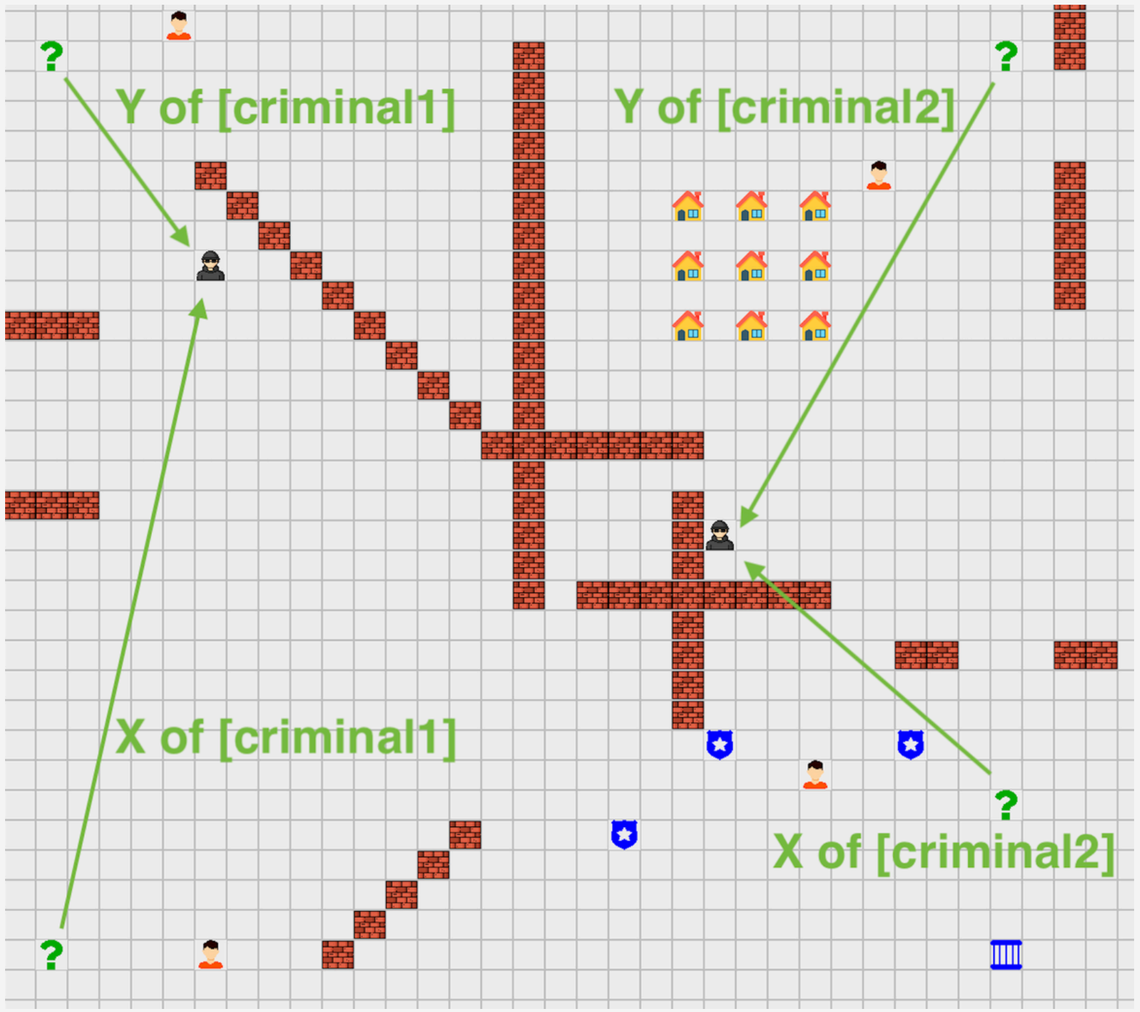
- If they receive a broadcast message from a police agent about the arrest of a criminal they respond with a message thanking him.

Civilian Agent Icon

AGENTS (3)

Behavior of Clue Agents

- Four clue agents.
- Static position.
- If they are found by a police agent, communicate to him the known X or Y coordinate and the ID of the criminal.



Clue Agent Icon

AGENTS (4)

Behavior of Criminal Agents

- Two criminal agents.
- Static position.
- They hide in the city and upon being discovered, they receive a message from the police agent declaring that they are under arrest.
- When arrested by a police agent, they are removed from the grid as they are escorted to jail.

Criminal Agent Icon

INTERNAL ACTIONS (1)

What are Internal Actions?

Internal Actions are used in the project as a building block for agents to perform operations that go beyond basic logical reasoning.

To develop an Internal Action, one must extend a java class with the “DefaultInternalAction” class provided by Jason.

This approach allows the implementation of specific agent behaviors.

INTERNAL ACTIONS (2)

Important Internal Actions in the Project

1) FindPath.java

Internal Action that handles the logic of finding a path between two locations (x, y) in the city grid. Used by police agents in the “explore” goal.

2) Escorting.java

Internal Action that handles the escorting logic for the police agents.
It activates the escorting state to true when a police agent arrest a criminal. This change triggers an update to the police agent’s icon.

INTERNAL ACTIONS (3)

Important Internal Actions in the Project

3) EnterJail.java

Internal Action that handles the process for a police agent to enter the jail upon reaching it while escorting a criminal. It involves removing the police agent's icon from the grid, signifying that the agent is inside the jail with the criminal.

4) Arrested.java

Internal Action that handles the logic to arrest a criminal which involves removing the criminal's icon from the grid when a police agent finds and arrest him.

UTILITY AND HELPER CLASSES (1)

Important Helper Classes in the Project

1) AgentIdMapper.java

This utility class is fundamental to map the IDs of agents from Java to Jason. This mapping is essential because the ID numbering systems differ between the two environments: in Java, agent IDs range from 0 to 12, whereas in Jason, the IDs span from 0 to one less than the total number of instances of that agent type.

2) AgentPercept.java

This utility class is used in the project to add and update the percepts of the agents, so it acts as the intermediary layer between the agent representation in Java and Jason.

UTILITY AND HELPER CLASSES (2)

Important Helper Classes in the Project

3) LookAround.java

This utility class is used by police agents at each step to detect the presence of other agents in the eight neighboring cells around their current position.

Each time an agent or the jail is found in the neighboring cells, a new percept is added to the police agent who discovered it.

Thank You