Tecnológico de Monterrey

Modeling of Multi-Agent Systems with Computer Graphics (Gpo 301)

Evidence 2 - Final Deliverable

Students:

Omar Michel Carmona Villalobos | A01644146
Roberto Anwar Teigeiro Aguilar | A01643651
Luis Omar Olmedo Ortiz | A01643557
Catalina Pesquet | A01763937
Andres Barrera | A01763911

Professors:

Iván Axel Dounce Nava
Carlos Johnnatan Sandoval Arrayga
Obed Nehemías Muñoz Reynoso

Guadalajara, Jalisco
November 2024

1. Problem Description

This project aims to address the complex and critical challenge of coordinated patrolling in high-risk and resource-sensitive environments, where maintaining security and operational integrity is paramount. By leveraging a multi-agent system, the simulation will integrate drones, fixed cameras, and security personnel to create a realistic and functional surveillance network. The system is designed to detect and respond to potential threats, such as unauthorized intrusions, suspicious behavior, or equipment misuse, ensuring rapid response and effective incident resolution. Through this simulation, participants will apply principles of artificial intelligence, inter-agent communication, and autonomous systems to design and implement a cohesive and efficient security framework. The project not only focuses on developing technical capabilities in programming and simulation but also aims to highlight the practical applications of such systems in enhancing safety and operational efficiency in real-world scenarios like factories, construction zones, and agricultural areas.

2. Agents Description

2.1 DronAgent

- **setup():** Initializes the drone's attributes, such as beliefs (beliefs), intention (intention), plan steps (plan_steps), and patrol mode (patrol_mode). Prepares the BDI (Belief-Desire-Intention) architecture with initial values.
- **See(Environment):** Detects a robber within a specific range (10 units). Returns the detected robber or None if no robber is within range.
- **Brf (Detected_robber):** Updates the drone's beliefs based on perceived information. If a robber is detected, the robber's position is stored in the belief system. Previous beliefs are cleared to prevent outdated information.
- **Options():** Generates desires (goals) based on current beliefs. If a robber is detected, the goal is to inspect its location.

- **Filter():**Selects the most relevant or desirable goal from the generated options. In this case, the goal is to inspect a robber.
- **Plan():** Creates a plan to reach the robber's position by generating step-by-step movements along the x and y axes from the drone's current location to the target.
- **Execute():** Executes the current plan by taking one step at a time. If the plan is completed, resets the current intention.
- **Takeoff():** Simulates the drone taking off and entering patrol mode.
- **Land():** Simulates the drone landing, exiting patrol mode after completing its tasks.
- **Step():** Implements the complete BDI cycle. It handles perception, belief revision, goal generation, planning, and execution for each simulation step. If no more tasks are available, the drone lands.

2.2 RobberAgent

- **Setup():** Initializes the robber without special attributes.
- **move_ranomly():** Generates a random movement (including staying in place) within the environment.
- **step():** Calls move_randomly() to define the robber's behavior in each simulation step.

2.3 CamaraAgent

- **setup():**Initializes the camera with a detection range larger than the drone's (20 units) and a counter for the number of alerts sent.
- **detect_robber():** Identifies if a robber is within the camera's detection range. Returns the detected robber or None if no robber is found.
- **send_alert(robber):** Sends an alert to the system with the robber's location. Increments the counter of sent alerts.

- **step():** Detects robbers within range and, if one is found, sends an alert to the system.

2.4 SecurityPersonnelAgent

- **setup():** Initializes the agent's attributes, such as whether it is actively communicating with a drone and whether the current alert has been resolved.
- **respond_to_drone_signal(drone, robber_position):** Simulates taking control of the drone after it signals for assistance. Initiates communication with the drone and proceeds to confirm the robber's presence.
- **confirm_robber(drone, robber_position):** Validates the existence of a robber at the specified location and escalates the situation by issuing a general alarm.
- **simulate_general_alarm(drone):** Issues a general alarm, indicating the resolution of the alert. Ends communication with the drone and updates the alert status.
- **step():** Represents the agent's behavior during each simulation step. Monitors for alerts from the drone and reacts by responding to signals for assistance. After handling an alert, remove it from the simulation environment.

2.5 Agent interactions

- Drones detect and track robbers, updating their beliefs and executing plans to inspect their locations.
- Cameras detect robbers and send alerts to the system.
- Security personnel respond to alerts, coordinate with drones, confirm threats, and issue general alarms if necessary.

3. Ontology

3.1 Core Classes

- **Entity (Thing):** Superclass for all agents and entities.
- **Drone (Entity):** Represents a patrolling surveillance drone.
- **Camera (Entity):** Represents a stationary surveillance camera.
- **Robber (Entity):** Represents a suspicious individual.
- **SecurityPersonnel (Entity):** Represents security personnel tasked with handling threats.
- **Place (Thing):** Represents specific areas where events occur.

3.2 Relationships (Object Properties and Data Properties)

- **is_in_place (ObjectProperty):** Links entities to specific places.
  - **Domain:** Entity
  - **Range:** Place

- **at_position (DataProperty, FunctionalProperty):** Specifies the position **of a place as a string.**
  - **Domain:** Place
  - **Range:** str

- **detects_suspicion (ObjectProperty):** Links cameras or drones to robbers they detect.
  - **Domain:** Camera, Drone
  - **Range:** Robber

- **alerts (ObjectProperty):** Links cameras to drones or security personnel when an alert is issued.
    - **Domain:** Camera
    - **Range:** Drone, SecurityPersonnel

- **patrolled_by (ObjectProperty):** Links a place to a drone patrolling it.
    - **Domain:** Place
    - **Range:** Drone

- **status (DataProperty, FunctionalProperty):** Indicates the current status of an entity as a string.
    - **Domain:** Entity
    - **Range:** str

3.3 DroneAgent Class

Represents the drone with a BDI (Belief-Desire-Intention) architecture.

3.3.1 Attributes

- **beliefs:** Holds current knowledge about the robber's position.
- **intention:** Current goal, such as verifying an alert.
- **plan_steps:** Steps the drone takes to achieve its goal.
- **patrol_mode:** Indicates if the drone is patrolling.
- **landing_position:** The fixed landing location for the drone.

3.4 CameraAgent Class

Represents stationary cameras detecting robbers within their range.

3.4.1 Attributes

- **detection_range:** The range within which the camera can detect robbers.
- **alerts_sent:** Counter for the number of alerts issued by the camera.

3.5 Robber Class

Represents a suspicious entity moving randomly within the grid.

3.5.1 Attributes

- **agent_type:** Unique identifier for visualization.

3.6 SecurityPersonnelAgent Class

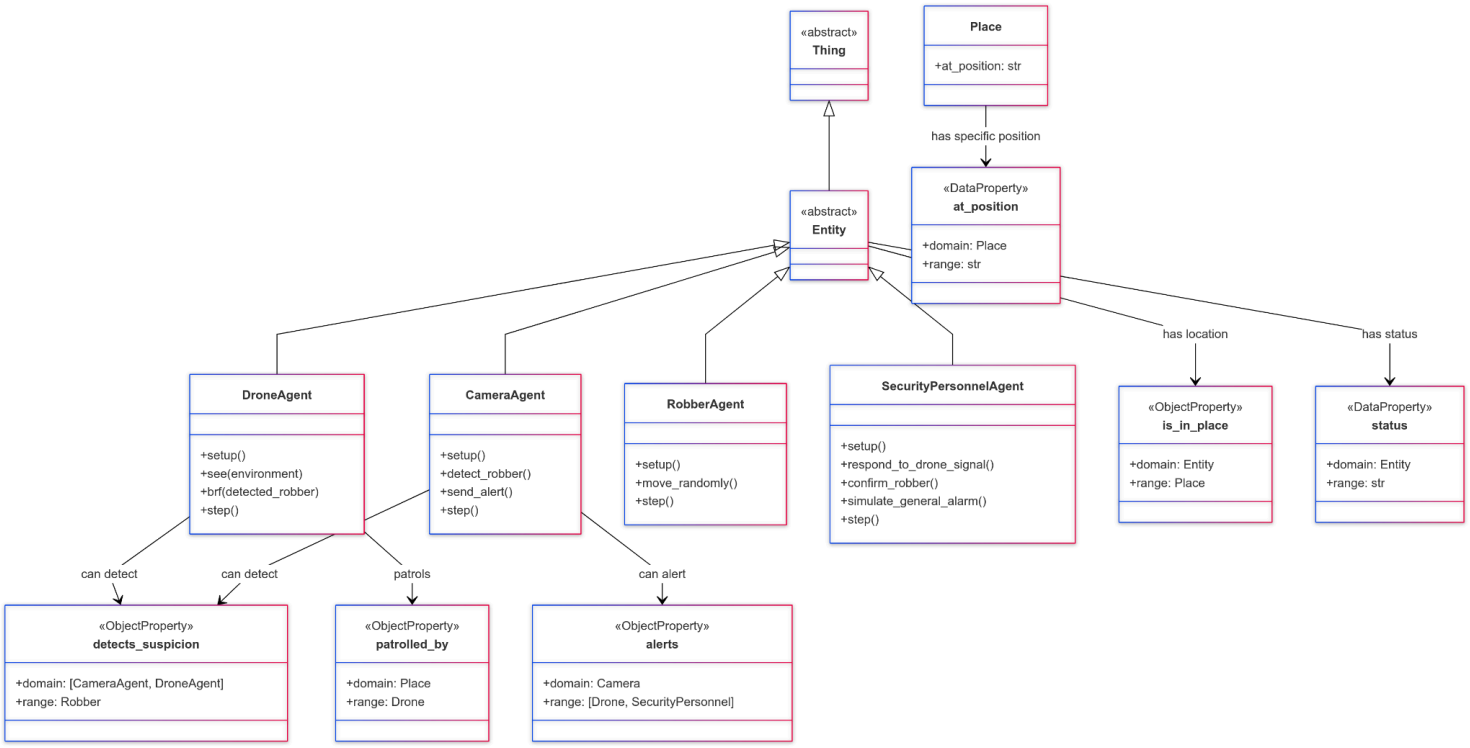
Represents security personnel managing alerts and drone control.

3.6.1 Attributes

- **in_communication:** Tracks active communication with the drone.
- **alert_handled:** Tracks if an alert has been addressed.

4. Agent classes and protocol diagram

4.1 Agent classes and ontology diagram



4.1.1 Diagram description:

The diagram represents the class hierarchy and relationships within the security system, modeling the different agents and their interactions.

1. **Entity Hierarchy:**

- The diagram starts with the abstract Thing class, which is the parent class for all entities in the system.
- The Entity class is also abstract and serves as the base class for the specific agent classes.

2. **Agent Classes:**
    - **Drone:** Represents the autonomous surveillance drone agent. It has methods for setup, perception (seeing the environment), belief revision, and execution of the main logic.
    - **Camera:** Represents the stationary surveillance camera agent. It has methods for setup, detecting robbers, and sending alerts.
    - **Robber:** Represents the suspicious individual (robber) agent. It has methods for setup and randomly moving within the environment.
    - **SecurityPersonnel:** Represents the security personnel agent responsible for responding to the drone's alerts, confirming the robber's presence, and issuing a general alarm.
    - **Place:** Represents a specific location or position in the environment, with the at_position data property storing the position.

3. **Inheritance Relationships:**

    All the specific agent classes (Drone, Camera, Robber, SecurityPersonnel) inherit from the abstract Entity class.

4. **Object Properties (Relationships):**
    - **is_in_place:** An object property that links an Entity (such as the Drone, Camera, Robber, or SecurityPersonnel) to a Place.
    - **detects_suspicion:** An object property that represents the ability of a Camera or Drone to detect a Robber.
    - **alerts:** An object property that models the ability of a Camera to alert a Drone or SecurityPersonnel.

- **patrolled_by:** An object property that associates a Place with the Drone agent responsible for patrolling that area.

5. **Data Properties:**
    - **at_position:** A data property that stores the specific position of a Place.
    - **status:** A data property that represents the status of an Entity.

6. **Relationships:**
    - The diagram shows the relationships between the different classes, indicating the capabilities and interactions between the agents.
    - For example, the Drone and Camera can detects_suspicion of a Robber, the Camera can alerts the Drone or SecurityPersonnel, and the Drone patrols the Place.
    - The Entity has a is_in_place relationship with the Place, and the Place has an at_position data property.

The diagram provides an overview of the class hierarchy, properties, and relationships within the security system, serving as a valuable reference for understanding the structure and interactions between the different agents that were created.

5.0. Steps to do for the final delivery:

- Finish the documentation of the evidence, for Friday 29th of 2024, to be developed by Luis Omar Olmedo Ortiz and Omar Michel Carmona Villalobos.
- Take enough screenshots of the wolf in the environment made to train our agents so it can have almost 100% precision. This is due on Wednesday 27th by Michel.
- Train the agent so it can identify the objects in the environment made in unity. This task is due on Wednesday 27th by Roberto.
- Complete the scenario in unity giving more details to the environment. This is due for Wednesday 27th by Andres.
- Finish the final details and improve the codes for the agents made previously. This task is due on Thursday 28th by Catalina.
- Prepare the final details for the presentation. This is for Thursday 28th by everybody.

5. Utility and success of agents

6. Analysis and Reflections

Luis Olmedo: Why did you select the multi-agent model used? What were the variables that were taken into account at the time of making the decision? What is the interaction of these variables with respect to the simulation result? Why did you select the graphical design presented? What are the advantages you find in the final solution presented? What are the disadvantages that exist in the solution presented? What modifications could you make to reduce or eliminate the disadvantages mentioned?

Michel Carmona:

Roberto Teigeiro:

Catalina Pesquet:

Andrés Barrera: