Autonomous Software Agents Report

Marina Segala
248447
marina.segala@studenti.unitn.it

Pietro Bologna
248715
pietro.bologna@studenti.unitn.it

Abstract

The goal of the project was to develop an autonomous software agent designed to play the Deliveroo.js game, maximizing points by collecting and delivering parcels. Using a Belief-Desire-Intention (BDI) architecture, agents sense the environment, manage beliefs, activate goals, select plans and execute actions to achieve objectives. The first part is done based on a single agent's capabilities. Then the project was extended to working in a team with a second agent, enabling communication and cooperative strategies.

1 Introduction

This report contains the key components of our autonomous agents, designed to play the Deliveroo game. The goal of the game is to earn points by collecting and delivering parcels to designated delivery points. To achieve this, we've implemented a Belief-Desire-Intention (BDI) architecture, that allows the agent to perceive the environment, establish goals, select the best plans and execute actions. This architecture allows the agent to continuously revise and improve its plans to effectively achieve its objectives.

The project is structured into two main parts:

- **Single Agent:** a single autonomous agent is developed, capable of interpreting and processing information from its environment. The agent is able to perceive the environment, update its beliefs, form its intention, and take actions.
- **Multi-Agent:** the code is expanded to include a second autonomous agent, enabling the agents to share information and coordinate their actions. In this case, the basic requirements include a defined game strategy and coordination mechanism. Specifically, agents must exchange information about the environment and their mental states to collaborate effectively and develop plans.

Finally, we assess and validate the performance of the agents by running a series of simulations at various levels, each with its own set of challenges and characteristics. This report will explore the logic, strategies, and structure involved in the development of both the single-agent and multi-agent systems.

2 Structures

Two main structures are used to collect information about the agent itself and the environment: `Me` and `Map`. In this way, a simple structure that encodes all relevant knowledge is created. Using these two classes, the object `myAgent` is created with `IntentionRevions`.

2.1 Me

The class `Me` represents the agent's state, capturing all relevant information about its current status. More specifically, it is formed by:

- `id`: id of the agent;
- `name`: name of the agent;
- `x` and `y`: coordinates of its position, updated with the `onYou` asynchronous function;
- `score`: the agent's current score;
- `numParticelsCarried`: number of particles that the agent has picked up;
- `particelsCarried`: boolean flag for saying if the agent is carrying something;
- `previus_position`: coordinates of the agent's previous position;
- `map_particles`: classic Map structure for saving the particles' position;
- `master`: boolean flag, for understanding the role of the agent. In particular, if it is `true`, the agent is the *MASTER*; otherwise, the agent can be a *SLAVE*, a *enemy* or we are in the *Single Agent Scenario*
- `friendId`: id of the other agent that collaborates with;
- `currentIntention`: the current intention the agent is executing.

2.2 Map

The other important class created is `Map`. It contains a representation of the important information present in the environment. This structure allows the agent to process and update information about its surroundings. The elements that compose the `Map` are:

- `width`: width of the map provided by the server
- `height`: height of the map provided by the server
- `map`: list of coordinates that can be reached by the agent. They can be delivery points or simple tiles;
- `deliverPoints`: list of coordinates of delivery points
- `agent_beliefset`: used for PDDL problem, it contains information about the existence and relationships between tiles

Besides functions used for saving or printing values, in `Map` there are also:

- `getAnotherDir(x, y)`: it returns the possible move that can be done starting from a specific point
- `update_beliefset()`: it updates the `agent_beliefset`, based on the current state of the map

3 Belief of the agent

The knowledge that the agent has is derived from the environment. Its belief is created, with asynchronous refining and updating, every time the agent encounters new stimuli. In this way, the knowledge is always the most recent one. The main observations that our agent can detect include

- Map
- Particle' sensing
- Agents' sensing

3.1 Particle' sensing

The detection of particles is done with an asynchronous function `onParcelsSensing`. For each parcel found, if it is available for collection, its position (x and y coordinates) is saved, as its id.

Every time a new particle is perceived, the list of possible movements and options the agent can do is updated: they are ordered by the distance between them and the agent itself.

3.2 Agents' sensing

TODO Coordinate with other agents by sharing parcel information.

4 Planning and Intention Loop

Deciding the best plan and action to implement was an important aspect of our implementation.

4.1 Intention Loop

The *IntentionLoop* is responsible for continuously processing the intentions of the agent: at each iteration, if at least one intention is present in a queue, it selects the last intention and it checks if it is still valid. In this case, it tries to achieve the intention selected. In any case, after all these operations, the intention is removed from the queue and the time of the last move is saved as the current timestamp. Saving the time of the last move is important to understand for how long our agent has been standing still: after 5 seconds, the agents will move randomly on the map. *MovingRandom* is also used in case there are no intentions to achieve.

In general, for the creation of the intentions queue the best option available is taken into account, trying to find the nearest particle to pick up or the closest delivery point where to deposit it. More specifically, these are the steps followed for the creation of the queue:

- making sure to choose the best intention to achieve **TODO** RIFORMULARE QUESTA FRASE
  - the action `pick_up` of nearest particle has to be found
  - if the agent is carrying some particles (`me.particelsCarried` = True) and the closest delivery point has a lower distance than the nearest particle, the action `put_down` is created
  - the action of our intention is not defined, the action `go_to` is created (based on a random move)

- if the intention is not already present in the queue, it is added
- if the intention is different from the previous one (that we called `old_intention`) and its action corresponds to a `pick_up`, the . **TODO**- NON SO COME SCRIVERLO A PAROLE
- the queue is sorted according to the lower distance from the agent, as last step

BSF - PDDL

4.2 PDDL

5 Communication

6 Results

7 Conclusion and future improvements