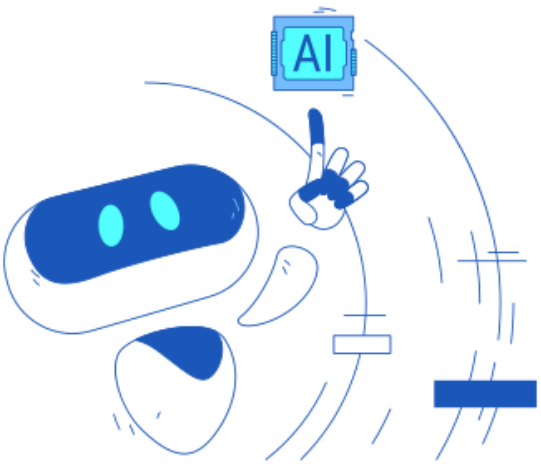
UNIVERSITÀ DI TRENTO

Department of Information Engineering and Computer Science

Master degree in Computer Science - Artificial Intelligence
FINAL REPORT

Autonomous Software Agents



INDEX:

Filippo Grilli
Petr Sabel
Autonomix
Academic year 2023/2024

INTRODUCTION

Nowadays, the ability to automate a process and make a practice autonomous is indispensable, especially in the workplace. Thanks to these automation techniques, companies save time, money, and resources, making it crucial to develop autonomous systems.

In the realm of artificial intelligence, an agent is defined as any entity capable of perceiving its environment through sensors and interacting with it via actuators. However, this broad definition isn't particularly practical. More intriguing is the concept of a rational agent (or a system of agents) designed to maximize its expected performance measure based on its available knowledge and resources. Moreover, intelligent agents must be both rational and autonomous, meaning they should operate independently of external control and adapt to compensate for incomplete or inaccurate prior knowledge about the world.

Building on these principles, this project aimed to develop an autonomous and rational (multi-)agent system to play the game of Deliveroo. This task is inspired by the real-world challenge faced by courier services, where packages need to be delivered to various locations efficiently and promptly. Specifically, the game is set on a two-dimensional map where parcels are generated randomly, each with a reward value that may decrease over time. The agents' goal is to collect and deliver these parcels to the designated locations to maximize their total rewards.

The Autonomous Software Agent course, therefore, consisted of two parts:

- Development of a single autonomous agent;
- Development of a team composed of two autonomous agents.

The project objective is to develop agents that can navigate within an environment to capture the highest number of rewards and deliver them to the designated tiles in the shortest time possible.

SINGLE AUTONOMOUS AGENT

Before starting the development phase of the autonomous software that will participate in the first challenge, we define the BDI (Belief-Desire-Intention) model:

BELIEF
We based the software on the knowledge provided to the agent, which included information about the agent itself, the environment (the game map), and other agents.

Among this data, the "decay_time" of the parcel rewards is significant, as it indicates the rate at which a parcel loses value. Consequently, we implemented this function that allows us to track the value of parcels we have seen, even if they are no longer in view:

```JavaScript
setInterval(() => {
                // Decreases parcels' rewards
                for (let parcel of this.parcels.values()) {
                    parcel.reward -= 1;
                }
                for (let parcel of this.carry) {
                    parcel.reward -= 1;
                }

                // Removes expired parcels
                this.parcels.forEach((p, id, _) => {
                    if (p.reward < 1) {
                        this.remove_parcel(id)
                    }
                })
                this.carry.filter((p) => p.reward > 0);

            }, decay_time)
```

DESIRE
We categorized the desires that the agent aims to achieve into three groups:

- Pickup: The primary goal of the agent is to collect parcels located within the map;
- Deliver: After collecting the parcels on the map, to actually achieve a result and give meaning to its actions, an agent must deliver the parcels to the appropriate areas specified on the map;
- Explore: Indicates what should be done when our agent has nothing better to do; this way, it will be possible to obtain new information, updating its beliefs.
  However, explore does not involve random moves but must have a precise structure and objective. Initially, “explore” directed the agent to move towards one of the points on the map with the highest tile density (the most visible tiles). After challenge 1, we modified this algorithm to navigate the agent to move directly to one of the spawnable cells on the map, increasing the chances of finding a newly spawned parcel.

INTENTION
Regarding the intentions, we decided to manage them through formulas that calculate scores, providing a value to all the intentions that the agent might desire to pursue.

To evaluate each intention, it is necessary to know the plan for the desired goal in order to determine the number of movements required to reach a certain tile.

We demand that there is no intention with a negative score. If a score were to be negative, we decided to set it randomly between 0 and 1. This is because otherwise, “explore” desire would always be executed, but following “explore” might not always be the best choice compared to following a plan with a negative score.

Below are the formulas that calculate the score for the different desires:

- pickup:

```JavaScript
score = parcel.reward - new_plan.length * agent.move_cost / agent.time_to_decay;
```

Initially, this formula was adjusted if there were other agents nearby the parcel, to discourage the pickup of parcels that are close to other agents.

```JavaScript
// Evaluates if there is an agent closer than me
       const enemy_gap = detect_agents(parcel.x, parcel.y, agent)
       if (enemy_gap > 0) {
              score /= Math.pow(2, enemy_gap);
       }
```

However, we decided to stop using it after completing challenge 1 because we had observed a scenario where parcel spawn tiles were all close together in a single area of the map. Consequently, all opponents trying to collect parcels found themselves in that area, and instead of attempting to collect the parcels our agent moved away.
Furthermore we decided to be more aggressive ignoring some other checks that we set before the first challenge. For example, we do not check anymore if the plan is still available because it can be blocked just for an instant.

- deliver:

```JavaScript
const DELIVERY_WEIGHT: number = 0.3;

score = reward / DELIVERY_WEIGHT;
```

We decided to use a constant variable declared by us because, in the first challenge, using a more complex formula based on the "decay_time" of the parcel reward led to several issues, including the low frequency of deliveries. Furthermore, utilizing planning and exploiting the "decay_time" proves inaccurate since the agent may stop multiple times to recalculate the best route.

- explore: score is a random value between 0 and 1.

Other implementation details:

PLANNING
Initially, for participation in challenge 1, we used the A* algorithm, a shortest path search algorithm, to choose the best path to the destinations selected by the desires.
Then, we added a possibility to change the planner from the A* to a PDDL one.
One additional feature that we have implemented is the use of cache to reduce the planning times for frequent plans because one drawback of the PDDL solver is that it is much slower that A*.

In the PDDL domain file we specified possible actions, meanwhile in the problem file we describe the current situation and the goal the agent wants to achieve.

It is possible to use a distinct approach based on the utility estimation of tiles, like in Reinforcement Learning, but we choose the planner solution to make the decisions of the agent more predictable and explainable.
Our strategy surely affects the agent's behavior and, unfortunately, decreases the performance compared to other groups.

REACTIVE BEHAVIOUR
A feature we implemented that makes the program more efficient is allowing the agent to collect or deliver parcels reactively. This occurs when the agent already has a specified plan but encounters the opportunity to collect or deliver a parcel along the way without needing to recalculate the plan.

The following situations can occur:

- When the agent passes over a parcel, in this case the agent will perform the pickup action reactively;
- When the agent passes on a delivery tile, the agent reactively will perform a delivery action.

It would still be possible to increase the distance of the reactive behavior (i.e, to one or more adjacent tiles).

TEAM OF AUTONOMOUS AGENT

Differently from the previous case now the agents are in two and should collaborate to reach better results.
To get the current solution we started from the code of the single autonomous agent implementation and added some necessary functions that permit agents to:

- communicate;
- share information;
- collaborate.

Among the modifications that we introduces one of the most important was regarding the planning:

PLANNING:
The multi-agent planner uses a more complex domain, which describes actions available to both agents and adds a possibility of collaboration between them, and a more detailed problem description, with the addition of information about the both agents.
That permitted agents to perform a completely new action, that is the “exchange”.

```
Unset
(:action exchange
        :parameters (?me ?me_pos ?f ?f_pos)
        :precondition (and
            (me ?me)
            (carry ?me)
            (ally ?f)
            (tile ?me_pos) (tile ?f_pos)
            (at ?me ?me_pos) (at ?f ?f_pos)
            (or
                (left ?me_pos ?f_pos)
                (right ?me_pos ?f_pos)
                (down ?me_pos ?f_pos)
                (up ?me_pos ?f_pos)
            )
        )
        :effect (and
            (not (carry ?me))
            (withparcel ?me_pos)
            (exchanged)
        ))
```

After the second challenge we noted there is a huge problem with computing plans for big maps, because planner took a lot of time so we decided to change the PDDL solver planner algorithm to speed up.
This was achieved by modifying the PDDL client library, in particular, we added a new parser for *forbidden iterative-topk* planner.

However, the planner is slow, especially on large maps due to the numerous paths it has to calculate to decide how to act. To improve this situation, we introduced a function that estimates the costs of various tasks and chooses the most convenient intention, then calculates the best path using the planner.

EXCHANGE ACTION
We need to use this action when we have a particular situation in which

- only one agent can actually deliver parcels;
- while the other can only get them from the spawners but not deliver because it is hindered and needs help from the other agent.

The idea to solve this problem is the following:

1) The first agent takes a parcel and moves it to some far position;
2) The second agent moves to that position, claims the parcel and delivers it.

The PDDL action present above permits the agent to put down a parcel but requires that its collaborator stays on an adjacent tile.

To organize such behavior of the agents we need a communication between them in order to synchronize their actions.
If the actions are not synchronized the agents may collide and the entire plan will fail.

COMMUNICATION:
The communication between the two agents is initially established through message exchange so that the two agents can become "friends." This happens as soon as the agents connect to the server and occurs as follows: the first agent sends a message in the game chat, which the second agent recognizes by verifying the name and content of the message using a key string embedded within it. At this point, the agent who received the message responds in the game chat to establish a trust relationship. Now, the two agents recognize each other because they know each other's identifiers, so any messages sent from now on will be private.

Communication between the two agents is used to share essential information such as details about the environment (the location of various objects on the map, such as Parcels and Agents), information needed to coordinate their activities, and information necessary to develop multi-agent plans.

SYNCHRONIZATION

After establishing a connection between agents one of them may compute a collaborative plan and share it with the other.
To execute that plan initially the first agent sends it to the other and asks to verify that it is a valid plan.
After a verification both agents start to execute one step at the time, while one agent is executing an action the other waits for it to finish.
In that way, there is no possibility that two agents interrupt each other and in the case of failure an agent informs the other to drop the plan.

CONCLUSION

During the course of this project, we successfully developed and implemented autonomous systems to execute the process of collecting parcels on a map, and faced several challenges that allowed us to understand our mistakes and replace some functions with more precise ones. The main objective was to increase the efficiency and accuracy of the operations performed by the agent.

We have overall achieved the desired goals, demonstrating that our autonomous systems are capable of finding parcels, collecting them, and delivering them correctly.

The most problematic part was about the PDDL planner that we tried to mitigate in different manners.

The resulting implementation tends to make decisions explainable in order to reduce risks associated with agent activity because we think in the future similar solutions could be integrated into the human environment. This constraint influences the behavior of the agent and so also its performance.

POSSIBLE IMPROVEMENTS:

- The possibility of adding additional agents to the team could be implemented;
- Speed up the PDDL planner, optionally using some heuristic function;
- Execute multi-plans in parallel if possible;
- Better functions to evaluate intention scores.