Autonomous Software Agents project

Bozzo Francesco, Izzo Federico

UNIVERSITÀ DI TRENTO

2023-07-11

Contents

1 Agent

In the field of Computer Science, the term “agent” refers to an individual situated within an environment and capable of autonomously and flexibly taking actions to achieve its design objectives. Unlike traditional algorithms, agents do not require explicit definition of every edge case. Instead, a well-defined agent possesses a reasoning component that enables it to make decisions even in unknown situations. The flexibility of an agent can be assessed along two primary dimensions:

- Reactive: This dimension measures the delay required for the agent to respond to changes in the environment.
- Proactive: This dimension gauges the agent’s ability to take proactive action to maximize future goals.

Communication models between agents and their environments vary depending on the specific characteristics of the environment and the agent itself. Generally, an agent perceives observations from the environment through sensors and carries out actions on the environment using actuators.

Autonomy is a fundamental characteristic of an agent. The internal decision-making process of an agent, often referred to as its “brain”, should be capable of handling decisions with or without collected information. Furthermore, it should be able to adapt and evolve in response to potential changes in requirements.

Agents can be designed to solve tasks or goals. Task-oriented agents focus on accomplishing smaller objectives that contribute to the achievement of a larger final goal. On the other hand, goal-oriented agents receive a specific goal and autonomously determine a list of tasks necessary to fulfill the assigned goal.

1.1 Multi-agent system

A multi-agent system, as implied by its name, refers to a collection of agents situated within the same environment. Interactions within such a system can be broadly categorized into two types: cooperative and competitive. Throughout this project, we will delve into both of these interaction modes at various stages of development.

In a competitive system, multiple agents act in opposition to one another, where the overarching goal can be divided into sub-goals focused on maximizing personal rewards while minimizing opponents’ gains. Within this scenario, the objective function may be shared among enemy agents, and it is also plausible to have multiple functions where each agent interferes with enemies solely to achieve its own goals.

On the other hand, a cooperative system consists of numerous agents working together to maximize a shared reward. In such systems, each agent must possess the capability to cooperate, coordinate, and engage in negotiation to the greatest extent possible. Cooperative systems can be further classified into two main categories:

- Simple (reciprocal) cooperation: This form of cooperation occurs when the benefits derived from collaboration outweigh the costs associated with the actions taken. It is considered the simplest type of cooperation as it leads to increased fitness for both the helper and the helped parties.
- Altruistic cooperation: In this case, the cost incurred by the individuals or species offering assistance surpasses the advantages gained. This approach is often regarded as more challenging since it cannot be readily explained by a purely “genetic-centric” perspective.

An essential aspect to consider when implementing a cooperative system is the communication mechanism. It should prioritize speed, reliability, and minimize delays as much as possible.

1.2 Architecture

There are several architectural options available for constructing an agent with the ability to operate within a specific environment. For our purposes, we have chosen to adopt the BDI architecture outlined in the following pseudocode.

Algorithm 1 Agent control loop

```
procedure AgentControLoop
    $B \leftarrow B_0$                                          ▷ Belief set initialization
    $I \leftarrow I_0$                                          ▷ Intention set initialization
    while true do
        perceive $\rho$
        $B \leftarrow$ update$(B, \rho)$                        ▷ Belief set update
        $D \leftarrow$ options$(B)$                             ▷ Desires computation
        $I \leftarrow$ filters$(B, D, I)$                       ▷ Intention update
        $\pi \leftarrow$ plan$(B, I)$                           ▷ Plan computation
        while not (empty$(\pi)$ or succ$(I, B)$ or impos$(I, B)$) do
            $\alpha \leftarrow hd(\pi)$                         ▷ Get next action
            execute$(\alpha)$                                   ▷ Execute action
            $\pi \leftarrow$ tail$(\pi)$                        ▷ Remove executed action
            perceive $\rho$
            $B \leftarrow$ update$(B, \rho)$                    ▷ Belief set update
            if reconsider(I, B) then
                $D \leftarrow$ options$(B)$                     ▷ Desires computation
                $I \leftarrow$ filters$(B, D, I)$               ▷ Intention update
            end if
            if not sound$(\pi, I, B)$ then
                $\pi \leftarrow$ plan$(B, I)$                   ▷ Plan computation
            end if
        end while
    end while
end procedure
```

After each information exchange with the environment, the belief set, denoted as B, undergoes an update. Subsequently, the desire set, referred to as D, is computed based on the updated belief set and then filtered to generate the intention set, denoted as I. The intention set, in conjunction with the belief set, is utilized to formulate a comprehensive plan, denoted as π, which aims to fulfill all the intentions. This plan is then executed incrementally, action by action, represented as α. Whenever the agent receives new information from the environment, the belief set, intentions set, and desires set are updated accordingly. Following this update, the agent can decide whether to generate a new plan or adhere to the existing one.

2 Deliveroo

The Deliveroo environment serves as the designated playground for developing our BDI agents. It encompasses a delivery-based game, where agents navigate on a two-dimensional grid. The objective of the game is to collect parcels dispersed throughout the map and swiftly deliver them to designated delivery zones. Each parcel possesses an assigned reward value, which may decay over time, and is granted to the agent responsible for its successful delivery.

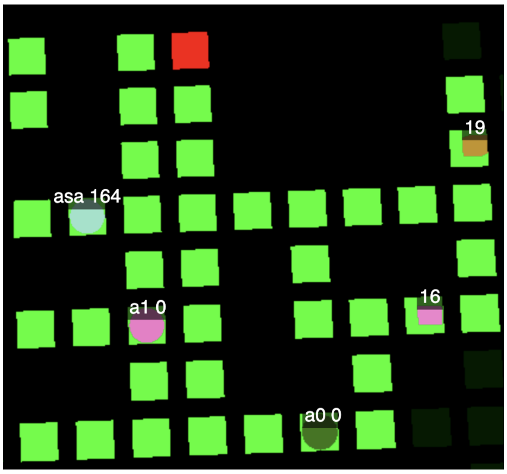


Figure 1: An example of a Deliveroo map.

The game can be played with either a single agent or multiple agents. In the latter case, agents are solid entities capable of obstructing each other's movement on the road.

Agents typically have limited perception of the world, restricting their awareness to other agents and parcels within a specified radius. Additionally, agents have the ability to carry and deliver multiple parcels simultaneously.

Deliveroo has been designed to offer diverse game scenarios through the manipulation of various parameters, including:

- Parcel: Generation interval, reward decay, and reward value distribution.
- Player: Number of steps and movement speed.
- External agent: Quantity and movement speed.
- Sensing: Radius of parcel and agent perception.
- Map: Definition of the grid board and classification of tiles as walkable, non-walkable, or delivery zones.
- Game clock.

Specifically, this examination project comprises two distinct deliverables, each with a different solution strategy:

- Single-agent: The agent operates independently, aiming to maximize its individual score.
- Multi-agent: The agent engages in communication with other agents to exchange information and collectively optimize the overall score of the group.

3 Single agent implementation

Considering the intricate nature of the environment, a multitude of tailored solutions have been implemented to address various complexities, difficulties, and challenges encountered.

3.1 Manhattan distance

The Manhattan distance is a mathematical function that offers a rapid estimation of the distance between two points. It is calculated by summing the absolute differences between the respective x-coordinates and y-coordinates of the two points:

$$d_{ab} = |b_x - a_x| + |b_y - a_y|$$

While this function may not be fully suitable for the Deliveroo game due to the presence of non-walkable tiles on the maps, it serves as a reasonable and computationally efficient heuristic for approximating distances in certain cases.

3.2 Problem parameters estimation

As detailed in Section 2, various parameters can be adjusted during the initialization of the game. While certain parameters are directly communicated to the agent, others remain obscure and can only be estimated. To enhance the accuracy of estimating player rewards, we have implemented a learning mechanism aimed at approximating the player's speed and the decay of parcel rewards over time. This mechanism enables us to build a more precise estimation for optimizing player rewards.

3.2.1 Player speed

During the agent's movement within the map, it maintains a record of its position over time. This historical positional data, along with corresponding timestamps, allows the agent to estimate its own speed using the following algorithm:

Algorithm 2 Player speed estimation

```
procedure UPDATEMAINPLAYERSPEEDESTIMATION($\mathcal{D}, s, \phi$)
    deltas ← []                                   ▷ Initialize an empty array of deltas
    for each timestamp $t_i \in \mathcal{D}$ do
        $deltas.append(t_i - t_{i-1})$            ▷ Delta of two consecutive timestamps
    end for
    $c \leftarrow s * (1 - \phi)$                 ▷ Current speed contribution
    $n \leftarrow$ avg($deltas$) $* \phi$         ▷ New speed contribution
    $s \leftarrow c + n$                          ▷ New estimation
    return $c$
end procedure
```

Here, ϕ represents the learning rate hyper-parameter, which can be utilized to control the influence of the most recent measured instantaneous speed in relation to its historical value.

3.2.2 Parcel decay

Apart from estimating the agent's speed, our agents are also capable of estimating the decay of parcel rewards over time. Similar to the player speed estimation, the parcel decay is calculated based on differences in timestamps. Each timestamp is associated with a sensed reward update of a visible parcel, allowing us to estimate the decay of the parcel rewards as time progresses.

Algorithm 3 Get parcel decay estimation

```
procedure getParcelDecayEstimation($\mathcal{D}$)
    deltas ← []                                      ▷ Initialize an empty array of deltas
    for each timestamp $t_i \in \mathcal{D}$ do
        $deltas.append(t_i - t_{i-1})$               ▷ Delta of two consecutive timestamps
    end for
    return deltas
end procedure
```

Algorithm 4 Parcels decay estimation

```
procedure updateParcelsDecayEstimation($\mathcal{P}, d, \phi_2$)
    deltas ← []                                      ▷ Initialize an empty array of deltas
    for each parcel $p_i \in \mathcal{P}$ do
        $deltas.concat(getParcelDecayEst(p.timestamps))$   ▷
    end for
    $c \leftarrow d * (1 - \phi_2)$                  ▷ Current decay contribution
    $n \leftarrow$ avg$(deltas) * \phi_2$            ▷ New decay contribution
    $d \leftarrow c + n$                             ▷ New estimation
    return $d$
end procedure
```

Here, ϕ_2 represents the learning rate, which controls the contribution of past estimations relative to the current estimated parcel decay. It allows us to regulate the influence of previous estimations when updating and refining the estimation of the parcel decay over time.

3.3 Probabilisitic model

Within the environment, multiple competitive agents coexist, and their effectiveness in picking up parcels significantly impacts the value of each individual parcel. To address this, we have developed a penalty value based on a probabilistic model that takes into account the potential plans of other competing agents.

The underlying concept behind this probabilistic model can be summarized as follows: “If there is a parcel available and I am the closest agent to it, I have a higher probability of reaching and acquiring it faster than any other agents. Consequently, this parcel should be given more weight and consideration, even if its assigned value is lower than that of other parcels located further away.” This assertion can be formulated more formally as follows:

$$\text{penalty probability} = \frac{\sum_{a \in \mathcal{A}} \frac{d_{max} - d_{pa}}{d_{max}}}{|A|}$$

Here, we denote $\mathcal{A}$ as the set of opponent agents, d_{max} as the maximum distance between the parcel and the collective group comprising opponent agents, the main player, and cooperative agents. Additionally, d_{pa} represents the distance between the parcel and an opponent agent.

3.4 Potential parcel score

The process of parcel selection plays a crucial role in defining an effective agent. To accurately estimate the potential reward gain of a parcel, our agents consider various elements and metrics. Formally, the final reward for a parcel is computed as follows:

$$r_f = r - \left(d_{ap} * \frac{s_a}{decay}\right) - \left(d_{min} * \frac{s_a}{decay}\right) - r * \text{penalty probability}$$

Here, d_{ap} represents the distance between the agent and the parcel, s_a denotes the estimated speed of the agent, d_{min} represents the minimum distance between the parcel and the nearest delivery zone, and penalty probability corresponds to the probability calculated using the probabilistic model discussed in Section 3.3.

The resulting formula takes into consideration multiple factors, including:

- The reward that the agent expends to approach the parcel and deliver it to the nearest delivery zone, which represents the minimum cost associated with delivering that particular parcel.
- An approximate estimation of other agents’ intentions based on their distances from the parcels, incorporating a probabilistic model.

3.5 Distances cache

In order to optimize computational efficiency and enhance the accuracy of reward estimation, a cache is maintained to store distances between tiles throughout the map. Whenever a plan is generated, the distance between the starting point and any other tile along the path is stored in the cache. However, it should be noted that this approach does not guarantee the shortest route between two tiles. As a result, cache entries are updated whenever a smaller distance value is discovered. This caching mechanism acts as a form of learning, gradually improving over time.

Throughout the codebase, the cache is utilized in numerous instances. In the event of a cache miss, the agent resorts to using the Manhattan distance as a fallback measure. By employing this caching strategy, the goal is to strike a balance between computation efficiency and accurate reward estimation.

3.6 Replan

As Deliveroo is a dynamic game that involves simultaneous actions from multiple agents, we have implemented a mechanism to replan the actions of the current agent if it fails to execute a move within a specific time frame. This functionality allows agents to prevent getting stuck in narrow or crowded areas of the map. By triggering a replanning process when necessary, agents can adapt their actions and navigate through challenging situations more effectively.

4 Multi agent implementation

Building upon the foundation of the single agent implementation described in Section 3, we have also developed agents that can collaborate to achieve a shared objective. The communication protocol utilized in our implementation is built upon a library that provides various endpoints to facilitate the handling of different types of messages. These endpoints include "say" for regular communication, "shout" for broadcasting messages to all agents, "ask" for querying other agents, and "broadcast" for widespread information dissemination. By leveraging these endpoints, agents are able to engage in effective communication and exchange relevant information during the game.

Let's now explore three distinct multiagent implementations:

1. Simple information sharing agents that operate in separate areas of the map.
2. Leader-election based agents where one agent generates and communicates multiple plans to every other agent upon request, referred as "plan communication" approach.
3. Leader-election based agents where one agent generates a single multiagent plan and communicates it in a syncronized way, action by action, to the each agent responsible for its execution, referred as "action dispatch" approach.

4.1 Information sharing agents

During the game, teams have the ability to share sensed data from the environment among their members. For simplicity, our agents utilize broadcast messages as the means of sharing information, making them visible to all connected agents in the game.

This allows each agent to construct its own plan based on the information sensed by all other agents across the map. Additionally, agents communicate their intended parcel pickups and the corresponding plans via broadcast messages. This mechanism helps reduce collisions among agents on the map and prevents unnecessary detours to pick up parcels that have already been claimed.

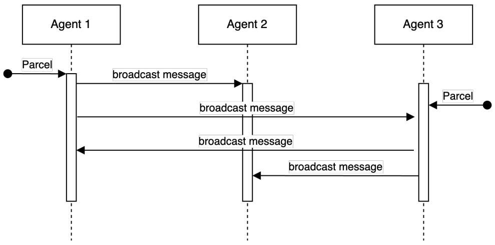


Figure 2: Information sharing.

By sharing information, agents are able to compute more refined plans that take into account hidden map locations. This multi-agent implementation follows a distributed model that can scale effectively with a large number of agents. Moreover, since there is no central computing unit, agents can be added or removed at any time.

It is important to note that this approach relies on broadcast communication, which means that any malicious agent could potentially understand the communication protocol and inject false information. To address this issue, we have considered implementing an encryption mechanism based on an initial exchange of keys. However, considering the nature of the course, we opted for a simpler implementation, focusing on other important implementation details.

4.2 Leader-members agents

The leader negotiation process plays a pivotal role in the multi-agent architecture, whereby one agent is designated as the leader responsible for computing plans for all other agents.

4.2.1 Leader negotiation

The process of electing a leader is a well-known problem in computer science, but for the purpose of our project, we opted for a simple solution due to our focus on other aspects. The negotiation for the leader role is facilitated through two types of messages: “ask-for-leader” and “leader”.

Upon connecting to the game and receiving their initial position, each agent broadcasts an “ask-for-leader” message, inquiring if a leader has already been elected. This message serves as a request for information regarding the existence of a leader. Conversely, the “leader” message is used by the agent who has been elected as the leader to communicate its identity.

Following the transmission of the “ask-for-leader” message, a timeout period of 2.5 seconds is set. If no response from an existing leader is received within this timeframe, it indicates that no leader has been elected yet. In such a case, the agent who sent the “ask-for-leader” message assumes the role of the leader and broadcasts a message to inform other agents of its newly elected leader status.

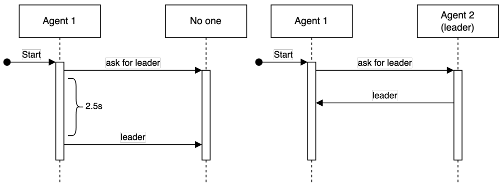


Figure 3: Leader negotation.

It is important to note that the simple implementation described above may introduce rare scenarios where multiple agents connect at exactly the same time, potentially leading to a race condition and the presence of multiple active leaders.

This enhanced model implementation can be considered an extension of the approach discussed in Section 4.1. In addition to sharing information about non-visible areas, this model allows for more comprehensive decision-making by leveraging knowledge derived from the computation of all plans. However, it comes with certain drawbacks. Firstly, it introduces a single point of failure, as all plans are generated by a single node. Secondly, the scalability of the system is limited when dealing with a large number of nodes. On the positive side, the system offers enhanced security, as plan communication occurs through point-to-point communication channels that cannot be accessed by malicious agents.

4.2.2 Plan communication

As our second multiagent approach, the plan communication system operates in a straightforward manner. When an agent does not have a plan, it initiates a request for a new plan by sending an ask-for-plan message. This request is implemented using a point-to-point ask primitive, which ensures that the request is directed specifically to the leader. Upon receiving the request, the leader begins the process of computing the plan. Once the computation is complete, the leader sends the list of actions comprising the plan back to the original agent using another point-to-point communication. This ensures that the plan is securely and efficiently transmitted between the leader and the requesting agent. Once the plan has been completed by the agent, it will then ask another plan back to the leader.

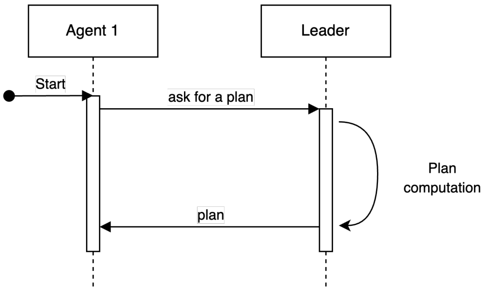
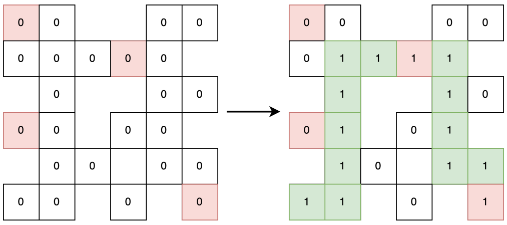


Figure 4: Plan communication.

4.2.3 Traffic penalty

In this communication model, the leader serves as the central compute node responsible for generating plans for all agents. This central position grants the leader extensive knowledge about the future movements of other agents. To enhance the computation of potential parcel scores described in Section 3.4, we have introduced an additional penalty that accounts for traffic considerations and aims to create plans that evenly distribute agents across the entire map.

To facilitate this, a traffic map is maintained on the leader. This map is a copy of the original map, and it is updated every time a plan is generated. For each tile included in a plan, the corresponding position on the traffic map is incremented by 1. When an agent requests a new plan, the previously computed plan for that agent is used to decrement the corresponding positions on the traffic map. This approach allows for the consideration of traffic patterns and encourages agents to choose paths that minimize congestion and evenly distribute their movements throughout the map.

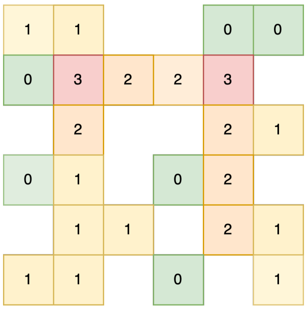


0	0			0	0
0	0	0	0	0	
	0			0	0
0	0		0	0	
	0	0		0	0
0	0		0		0

0	0			0	0
0	1	1	1	1	
	1			1	0
0	1		0	1	
	1	0		1	1
1	1		0		1

Figure 5: Empty traffic map to traffic map with one plan.

In Figure 5 it is presented an empty traffic map (on the left) with the respective delivery zones, after the computation of a plan the traffic map is updated (on the right). This process is applied for every generated plan and the final result is presented in Figure 6, it is clear that some tiles are more trafficate than others.



1	1			0	0
0	3	2	2	3	
	2			2	1
0	1		0	2	
	1	1		2	1
1	1		0		1

Figure 6: Traffic map with multiple plans.

Using the traffic map, it is possible to select a parcel and analyze its neighboring areas to determine if it is a congested region. This analysis helps us determine whether it is advisable to proceed with taking that parcel.

The logic behind the traffic penalty is summarized in the following pseudocode:

Algorithm 5 Traffic penalty

```
procedure TRAFFICPENALTY($M, p$)
    $m \leftarrow$ max($M$)                                  ▷ Obtain the maximum traffic in the current traffic map
    $t \leftarrow 0$                                          ▷ Initialize neighbourhood traffic
    $ns \leftarrow$ getNeighbours($p$)                        ▷ Get parcel neighbour tiles
    for each neighbour $n_i \in ns$ do
        $traffic \leftarrow traffic + M[p.x][p.y]$            ▷ Update neighbourhood traffic
    end for
    $t \leftarrow t/$len($ns$)                                ▷ Average neighbourhood traffic
    $p \leftarrow$ min($t/m$)                                 ▷ Obtain a probability from average traffic
    return $2 * p.reward * p$
end procedure
```

4.2.4 Action dispatch

Unlike the plan communication system, we have also developed a third multiagent solution whoch consists of action dispatch approach between the leader and simple agents. In this scenario, when the leader receives an "ask-for-leader" message, it communicates that it is the current leader and stores the identifier of the requesting agent. The requesting agent will then be considered an active player when generating the next plan.

As explained in Section 5.2, in this case, the leader generates a multiagent plan. The leader sends one action at a time to the agent responsible for executing it and waits for an acknowledgement message confirming the action execution by that agent. This process continues until the leader exhausts all remaining actions in the plan. At that point, a new plan is generated.

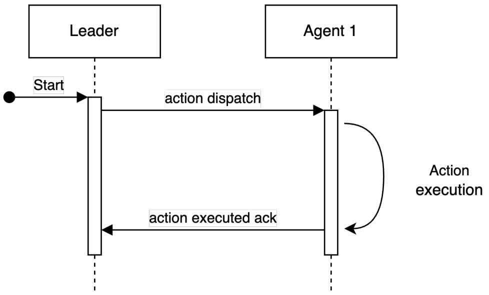


Figure 7: Action dispatch.

5 PDDL

Throughout the project, we developed two PDDL planning approaches:

- PDDL planning one agent at a time: This approach involves a single-agent pathfinder that picks up all requested parcels and delivers them collectively. It is used in the single agent approach, distributed multiagent, and plan communication multiagent scenarios.

- PDDL planning more agents at a time: This approach employs a sophisticated multiagent planner capable of handling multiple agents simultaneously. It is utilized in the action dispatch multiagent approach.

5.1 PDDL planning one agent at a time

The initial PDDL approach we implemented involves a straightforward implementation of an agent. This agent is capable of starting from a given position, collecting a specified list of parcels, and delivering them to designated delivery zones in the map. Due to the limitations of PDDL, it is not possible to prioritize parcels based on their reward and distance. As a result, this approach heavily relies on the agent's intentions when filtering parcels, as explained in Section 3.4.

Here are the details of the predicates, actions, and problem initialization used in our basic PDDL model:

Predicates:

- "at" defines the position of the agent and parcels.
- "can-move" determines whether it is possible to move between two tiles.
- "carrying" indicates whether the agent is carrying a specific parcel.
- "delivery" identifies whether a position on the map is a delivery zone or not.
- "delivered" signifies that all the given parcels have been delivered. It always represents the goal of the plan.

Actions:

- "move": the agent can move from one tile to a neighboring tile.
- "pickup": the agent can collect a parcel present on the current tile.
- "putdown": the agent can place all the carried parcels that have been asked to collect onto the current tile.

Problem initialization:

- A list of all walkable tiles.
- A list of available moves between walkable tiles.
- A list of tiles designated as delivery zones.
- A list of all the parcels that the agent must deliver in order to complete the plan.

Goal: deliver all the listed parcels.

5.2 PDDL planning more agents at a time

The main drawback of our initial PDDL approach is the lack of collaboration among multiple agents to solve a shared problem. As discussed in the Section 6, certain problems cannot be solved unless the two agents collaborate with each other. To address this issue, we decided to develop a more intricate belief model to be sent to the PDDL Online Planner, utilizing complex PDDL constructs such as typings and forall/when clauses.

Here are the details of the types, predicates, actions, and problem initialization used in our enhanced model:

Types:

- “entity” and “position” are subclasses of “object”.
- “agent” and “parcel” are subclasses of “entity”, as both can be assigned positions on the map.

Predicates:

- “at” defines the position of an entity on the map.
- “can-move” determines whether it is possible to move between two tiles.
- “carrying” indicates whether an agent is carrying a specific parcel.
- “delivery” identifies whether a position on the map is a delivery zone or not.
- “delivered” signifies whether a parcel has been delivered.
- “blocked” is used to block potential agent movements to tiles already occupied by other agents.

Actions:

- “move”: an agent can move from one tile to a neighboring tile that is not blocked.
- “pickup”: an agent can collect parcels present on the current tile if they are not already being carried by any agent.
- “putdown”: an agent can place all the parcels it is carrying onto the current tile.
- “deliver”: an agent can deliver all the parcels it is carrying if the current tile is a delivery zone.

Problem initialization:

- A list of all walkable tiles.
- A list of available moves between walkable tiles.
- A list of tiles designated as delivery zones.
- A list of all agents participating in the shared plan.
- A list of all the parcels that must be delivered, including both those to pick up and the ones already carried.
- A list of blocked tiles occupied by agents.

Goal: delivered all the listed parcels.

The final PDDL plan involves a multiagent approach, where multiple agents collaborate to solve a single problem. To implement this approach using the leader-election paradigm, we chose to have the leader send one action at a time (similar to a single-action plan) to the agent responsible for executing the action according to the planned schedule. As described in 4.2.4, this mechanism also necessitates an acknowledgment messaging system from the agent performing the action to the team leader.

6 Benchmarking

During the benchmarking phase of the project, six different maps were used to evaluate the proposed solutions. Three maps were specifically designed for the single-agent implementation, while the other three maps were used for the multi-agent implementation.

Each test was conducted by running the agent or agents for a total duration of 5 minutes. It is important to note that there were no strict requirements for result reproducibility (such as random seed initialization), which means that the results may slightly vary across different runs, but they should generally remain within a similar range.

The purpose of the benchmarking phase was to assess the performance and effectiveness of the implemented solutions under realistic conditions and evaluate how well they performed in terms of various metrics such as score, efficiency, and robustness.

6.1 Single-agent

The benchmarking phase for the single-agent implementation included three specific challenges:

1. *challenge_21.js*: This challenge featured a full square map with numerous enemy agents moving randomly. The map had a limited number of delivery zones, and there was no decay in parcel rewards.
2. *challenge_22.js*: In this challenge, a more complex map was used with large roads and no enemy agents. Parcels in this scenario had very low rewards, and the agent's movement speed was also significantly slow.
3. *challenge_23.js*: The most complex scenario for the single-agent implementation involved a map with small roads and a high number of enemy agents. Parcels in this challenge had very high rewards, and the agent's movement speed was set to be very high.

These challenges were designed to test the single-agent implementation's performance and efficiency under different conditions, including variations in map structure, enemy agent presence, parcel rewards, and agent speed.

	Chal. 21	Chal. 22	Chal. 23
Prob model	210	no agents	2288
No prob model	270	186	2818

6.1.1 Challenge 21

Based on the investigation and analysis conducted after the 5-minute benchmarking in challenge 21, it was discovered that the main issue lies in the parcel decay estimation mechanism. The conservative design of the mechanism, intended to prevent excessive greediness, is causing a decrease in parcel rewards even when there is no actual decay specified in the challenge.

Additionally, the probabilistic model implementation, which assumes that enemy agents tend to collect parcels in close proximity to their positions, is not applicable in challenge 21 where enemy agents move randomly. This mismatch between the model assumption and the actual behavior of enemy agents results in missing out on valuable parcels.

These findings suggest that the conservative approach and the probabilistic model, as currently implemented, are not suitable for challenge 21. Adjustments or alternative strategies may be required to improve performance and better adapt to the specific characteristics of this challenge.

6.1.2 Challenge 22

Based on the provided information, the low results obtained in challenge 22 can be attributed to the configuration and characteristics of the challenge itself. The parcels in this challenge are initially spawned with a value around

10, and the decay rate is fast. The single-agent implementation based on PDDL relies on various heuristics to estimate the value of parcels, which is used to compute desires and select the most favorable parcels to take.

However, due to the nature of the implementation, the estimation of parcel value is performed before computing the plan, which means that the length of the plan and the time required to complete it are not known in advance. As a result, taken parcels may reach zero value before they can be delivered. To address this issue, attempts were made to discount parcels further by considering the proximity to the closest delivery zone. However, these adjustments did not yield the desired improvements, and the estimation remained overly conservative.

The challenges posed by fast decaying parcels and the dynamic nature of estimating their value based on uncertain plan lengths can be complex and require careful consideration. It may be necessary to explore alternative strategies or refine the existing heuristics to improve the performance of the single-agent implementation in challenge 22.

6.1.3 Challenge 23

Based on the information provided, it seems that the results obtained in challenge 23 are relatively better compared to the previous challenges. This can be attributed to several factors:

1. Higher Parcel Rewards: The challenge is designed in such a way that the parcel rewards are higher compared to the previous challenges. This means that even if the agent encounters some delays or inefficiencies in its plan, the overall rewards obtained from delivering parcels are still significant.
2. Probabilistic Model: The probabilistic model takes into account the distance between parcels and opponent agents to estimate the likelihood of successfully delivering a parcel. In a scenario with smaller roads and many enemy agents, this model could help the agent make more informed decisions about which parcels to prioritize and avoid potential collisions or conflicts.
3. Parcel Decay Estimation: The conservative estimation of parcel decay may also play a role in the agent's success in challenge 23. By being cautious and considering the potential decay of parcels, the agent is more likely to prioritize parcels with higher rewards and deliver them before their values decrease significantly.

It is important to note that the qualitative analysis conducted during the 5-minute run is crucial in understanding the agent's performance in this specific environment. The combination of higher parcel rewards, the utilization of the probabilistic model, and the conservative estimation of parcel decay contribute to the agent's improved results in challenge 23.

6.2 Multi-agent

The three maps used for the benchmark phase of the multi-agent implementation provide different scenarios to evaluate the performance of the agents. Let's take a closer look at each challenge:

1. *challenge_31.js*: This map features a multi-linear layout with a single perpendicular hallway. There are no enemy agents present, and there are many parcels with high rewards. This scenario tests the agents' ability to efficiently collect parcels and deliver them to the appropriate delivery zones. The absence of enemy agents allows the agents to focus solely on maximizing their rewards without the added complexity of avoiding conflicts.

2. *challenge_32.js*: In this map, the multi-linear layout is maintained, but the perpendicular hallway is removed. Instead, the two agents need to collaborate and coordinate their actions to score points. This challenge emphasizes the importance of teamwork and communication between the agents. They must work together to collect parcels and deliver them effectively, taking into account the limitations imposed by the modified map layout.
3. *challenge_33.js*: This challenge is a modified version of challenge_31.js, where the map is divided into two different blocks. This division introduces additional complexity and requires the agents to navigate between the blocks to collect and deliver parcels. It tests the agents' ability to plan and coordinate their movements efficiently, considering the divided nature of the map.

By evaluating the agents' performance on these three challenges, it is possible to assess their ability to adapt to different map layouts, collaborate with other agents, and make strategic decisions based on the specific game conditions.

	Chal. 31	Chal. 32	Chal. 33
Information sharing	786	0	0
Plan communication	323	0	0
Action dispatch	873	1058	387

6.2.1 Challenge 31

In the context of challenge 31, which features a large map with a high number of parcels and an infinite visibility radius, efficiency issues were encountered due to the large number of nodes in the graph used during the PDDL algorithm. To mitigate this problem, an additional hyper-parameter was introduced to limit the number of parcels taken during a plan. By constraining the number of parcels considered in the planning process, the complexity of the PDDL computation can be significantly reduced.

The introduction of this hyper-parameter allows for a more efficient planning process by focusing on a subset of parcels. By selectively considering a limited number of parcels, the number of nodes in the graph used by the PDDL algorithm is reduced, leading to improved efficiency and faster plan computation.

This hyper-parameter provides a trade-off between computational complexity and the agent's ability to select the most beneficial parcels. By tuning this parameter, it is possible to strike a balance between the complexity of the planning process and the quality of the resulting plans.

6.2.2 Challenge 32

This map is the most interesting case study for the evaluation. There are two different approaches, where the agents can spawn on different rows or in the same row, they present unique challenges and require different strategies.

In the scenario where the agents spawn in the same row, the narrow road and the presence of both agents create a need for deep coordination and collaboration. One agent needs to bring parcels closer to the other agent, who can then deliver them. This coordination requires careful planning and communication between the agents to ensure efficient parcel collection and delivery, which only the action dispatch approach is able to offer among the different methods we developed.

On the other hand, when the agents spawn on different rows, they act as separate entities since they cannot block each other's paths. This reduces the need for direct coordination, but still requires effective decision-making and resource allocation between the two agents. The high parcel rewards and the large number of parcels in this scenario contribute to achieving a very high result.

Furthermore, the infinite visibility radius eliminates the necessity for map exploration, enabling agents to concentrate exclusively on optimizing their actions using the provided information. On the flip side, agents must judiciously choose the parcels to include in their desire set by filtering out unreachable ones. In our case, we implemented a Breadth-first search to identify non-reachable parcels as part of the intentions filtering process.

Overall, the evaluation of the multi-agent implementation in this challenging scenario showcases the agents' ability to coordinate their actions, strategically handle parcel collection and delivery, and take advantage of the high parcel rewards to achieve a favorable outcome.

6.2.3 Challenge 33

The modified version of challenge 31 presents a scenario that requires a less refined strategy compared to the previous challenges. There is no specific strategy needed for the exchange of parcels between agents, as each agent can independently compute its own plan and divide the parcels to pick up.

One of the notable aspects of this challenge is the configuration of the map, particularly the visibility radius. With a reduced visibility radius, the agents are required to explore the map to discover new parcels. However, the map is designed with a good distribution of parcels, allowing the agents to find a sufficient number of parcels within their visibility range. After a couple of deliveries, the need for further exploration decreases as the agents become more familiar with the map layout.

The map being divided into blocks adds an additional element to the challenge. Agents can spawn in the same block or different locations, but the overall behavior and strategies remain consistent. The agents can still compute individual plans and divide the parcels accordingly.

Overall, this modified challenge offers an interesting combination of reduced visibility, exploration requirements, and map division. It allows the agents to adapt their strategies accordingly and optimize their parcel collection and delivery processes.

7 Conclusion

The Deliveroo environment provided a suitable scenario for the development of single and multi-agent strategies, allowing for a deep understanding of various aspects of the architecture presented in Section 1.2.

Processing and storing received information in a belief set and computing desires based on this information was a crucial step in the implementation. Desires could be expressed in different ways, such as the number or location of parcels to pick, the maximum or minimum distance of the path, the maximum or minimum reward, and more. The project primarily focused on parcel-based desires due to limitations in the PDDL solver used, which did not support numbers. While this limitation does affect the solver's capabilities by simplifying the problem definition, it does have the advantage of reducing the time required to generate a new plan avoiding potential timeouts.

Desires were used to compute plans that aimed to maximize the reward function. The time required for plan computation played a dominant role, highlighting the need to strike a balance in the replanning strategy.

The utilization of PDDL solutions provided the advantage of focusing solely on defining the desires set without concerning the plan computation. However, this approach had its pros and cons. The obtained plans could be non-optimal, and the inability to introduce heuristics during plan computation limited the ability to obtain more refined instructions. Plan refinement was only possible after receiving the plan, and it required simulation on a copy of the belief set, posing additional challenges.

The results presented in Section 6 demonstrated the applicability of PDDL-based strategies for the Deliveroo environment. The focus was primarily on developing multiple solutions without delving into extensive optimizations. Future work in this area could involve:

- Developing an agent capable of selecting different strategies based on the environment and available information.
- Defining additional heuristics for desires computation to enhance decision-making.
- Exploring the adoption of a more powerful planners that can use more recent PDDL versions that supports numbers, enabling more flexible problem definition.
- Designing a strategy for plan evaluation and refinement to enhance plan quality.

These avenues of future work can contribute to further improving the effectiveness and efficiency of the agent's decision-making and planning processes in the Deliveroo environment.