Autonomous Software Agents Project Report

Sartore Alessandro
256152
alessandro.sartore-1@studenti.unitn.it

Missagia Fabio
256141
fabio.missagia@studenti.unitn.it

Team: Descanta Bauchi

Index

Abstract

This project focuses on the development of an autonomous software agent designed to play the Deliveroo simulation game, with the goal of maximizing the score by efficiently collecting and delivering parcels. The agent is designed using a BDI (Belief-Desire-Intention) architecture, allowing it to perceive the environment, update its internal beliefs, define important goals, and select suitable actions. Initially, the work was focused on the development of a single agent setup, who managed modules for planning, exploring, and managing knowledge. In the second phase, the system was extended to support two agents, introducing communication and distributed coordination strategies. The final solution was tested on various maps, showing strong adaptability, effective cooperation and optimized pathfinding even in dynamic and competitive environments. Our approach was successful as our agents ultimately placed second in the final challenge.

1 Introduction

The *Deliveroo simulation game* presents a complex real-time environment in which autonomous agents must collect and deliver parcels to maximize their final score. The game operates on a grid-based map, where parcels appear randomly, decay in value over time, and must be delivered to fixed destination tiles. Agents must operate under dynamic constraints, balancing **exploration**, **exploitation**, and **coordination**, especially in multi-agent scenarios where no central controller is present.

Motivation and Approach

Our approach is grounded in the **Belief–Desire–Intention (BDI)** architecture, a model well-suited for environments that require continuous reasoning and adaptability. This architecture structures agent cognition into three distinct but interconnected components: beliefs represent the agent's understanding of the world, desires correspond to its high-level objectives, and intentions guide its committed actions. The BDI paradigm provides both **flexibility** and **reactivity**, enabling agents to adapt their behavior to an ever-changing context.

Project Development Phases

The process of its development took place in two stages. The first stage aimed at the development of a strong single-agent system. The agent was created to move in the environment intelligently, with strategies based on real-time perception and goal-based planning. It used personalized exploration policies, path planning, and recovery strategies to deal with changing obstacles and inaccessible goals.

The second phase extended the architecture to support a **multi-agent setup**, where agents coordinate without any form of centralized supervision. Communication protocols were introduced to share intentions and avoid conflicts. The agents can perform a handshake at the beginning of the process, enabling one to exchange intents and self-assign spatial zones. In particular, on maps with constrained layouts, such as long corridors, we implemented specialized roles such as `RUNNER` and `CARRIER` to enhance cooperative behavior and imp qrove efficiency.

Structure of the Report

This report details the **system architecture**, including the internal design of the agent modules and the decision-making flow. It explains the option-intention cycle that governs real-time behavior, the coordination strategies adopted in multi-agent settings, and the recovery mechanisms implemented to ensure robustness. We conclude by presenting an evaluation of the performance of our system on various maps and discuss its strengths and potential improvements. Our solution ultimately placed second in the final competition, validating the **effectiveness and adaptability** of our approach.

2 System Design

The agent system is structured around the **Belief–Desire–Intention (BDI)** paradigm, which provides a robust framework for decision-making in uncertain and dynamic environments. This architectural model promotes modularity, reasoning clarity, and behavioral flexibility, making it an ideal foundation for building both single and multi-agent solutions in the Deliveroo simulation environment.

2.1 BDI Architecture Overview

At the core of the system lies the continuous loop of perception, deliberation, planning, and execution. The agent continuously updates its **beliefs** based on environmental feedback, derives new **desires** depending on its current situation and internal goals, and selects **intentions** to commit to, which are then translated into executable actions.

This reasoning loop ensures that agents remain reactive to changes and proactive in pursuing the most rewarding strategies. By maintaining a strict separation between perception, goal generation, and action planning, the BDI model allows the agent to operate with both autonomy and adaptability.

Figure 1 illustrates the high-level interaction between the main components of the architecture, showing how sensory input is transformed into rational behavior through successive layers of processing.

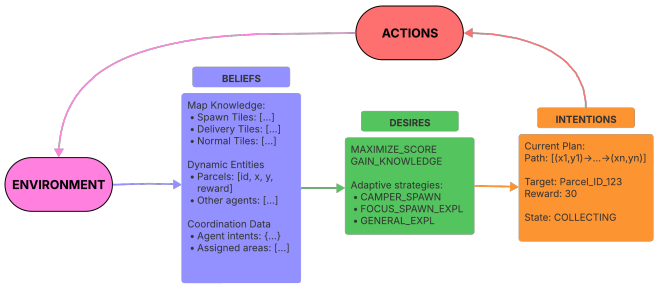

Figure 1: High-level overview of the BDI architecture used in our agent system.

2.2 Module Breakdown

To implement the BDI loop, the system is organized into modular components, each responsible for a specific cognitive function.

`beliefs.js` – Knowledge Representation

This module is responsible for maintaining the agent's internal model of the world. It combines static map data such as walkable, spawn, and delivery tiles, with dynamic information like current parcel locations, agent positions, and observed rewards. Additionally, it clusters spawn tiles into regions to support area-based reasoning, and tracks teammates' intentions to avoid conflicts in multi-agent settings. This world model is continuously updated during execution, allowing the agent to reason with up-to-date and structured information.

`desires.js` – Goal Hierarchy

The desire module defines high-level exploration and delivery goals based on the characteristics of the map. Depending on the density and distribution of spawn tiles, the agent dynamically selects among several exploration strategies. These include **camping** in low-density scenarios, **focused roaming** in high-density regions, or **broad exploration** when spawn tiles are evenly distributed.

`planner.js` – Action Planning

The planner represents the agent's reasoning core, where current beliefs are transformed into concrete intentions and actions. It selects targets, evaluates parcels using dynamic utility metrics, and implements fallback mechanisms in case of blocked or unreachable goals. Recovery procedures, such as temporary banning of problematic parcels or stuck detection, ensure that the agent can respond effectively to dynamic failures and uncertainties.

`pathfinder.js` – Pathfinding

Pathfinding is implemented through a customized **A* algorithm**, specifically optimized for real-time, multi-agent grid navigation. The algorithm integrates belief updates to avoid collisions with other agents, supports efficient detour calculation, and includes BFS-based fallbacks for robustness. Additionally, the module incorporates an optional **PDDL planner** with timeout-based fallback to A*, enabling more sophisticated path planning when computational time allows while maintaining real-time responsiveness. By prioritizing both responsiveness and computational efficiency, this module enables agents to make quick and safe movement decisions.

`delivery.js` – Delivery Strategy

This module focuses on maximizing reward through intelligent parcel delivery. It considers both the shortest delivery paths and nearby opportunities for additional pickups during delivery. The agent evaluates trade-offs between detours and rewards using a greedy but robust scoring system. Timeout-based recalculation ensures resilience to unexpected blockages by other agents or environmental changes.

`area_manager.js` – **Area Assignment**

This module is responsible for dividing the map into distinct zones and assigning them to individual agents to minimize overlap and reduce redundant exploration.

`handover_coordinator.js` – **Coordination in Constrained Maps**

In maps with highly constrained topologies, such as long corridors, coordinated communication between agents becomes essential. This module facilitates parcel handovers and movement synchronization between agents, ensuring smooth operation and avoiding bottlenecks.

Additional minor modules support auxiliary tasks such as best parcel selection, exploration management and configuration handling. These are omitted here for brevity, as they do not contribute directly to the core decision-making logic.

3 Single-Agent Strategies

The initial phase of our development focused on designing a single autonomous agent capable of effectively navigating and operating within a wide variety of map configurations. The core idea was to create a system that could dynamically adjust its behavior depending on the spatial structure of the environment, the distribution of parcels, and the actions of competing agents. This adaptability was achieved through extensive testing against other agents in the first challenge, allowing us to adapt the system to different conditions.

3.1 Exploration Logic

Efficient exploration plays a fundamental role in maximizing parcel pickups, especially in maps where few parcels are present. To address different spawn tile distributions across maps, the agent employs an **adaptive exploration strategy** that automatically selects one of three distinct modes, based on a density analysis of spawn tiles relative to the map size:

- **CAMPER SPAWN** – Used in maps with very few spawn tiles. The agent positions itself directly on a spawn tile and waits for parcels to appear, maximizing early pickup probability with minimal movement.
- **FOCUS SPAWN EXPLORATION** – Applied when spawn tiles form dense clusters. The agent actively navigates within those regions to increase parcel visibility and collection opportunities.
- **GENERAL EXPLORATION** – Employed in balanced maps with uniform spawn distribution. The agent explores broadly across the environment to ensure maximum coverage.

The strategy selection is driven by a threshold-based analysis: the ratio between the number of spawn tiles and the total number of walkable tiles determines the most suitable mode.

To avoid repetitive behavior, the exploration component also includes tile visitation tracking, which discourages agents from returning too frequently to recently visited zones, thereby promoting more diverse exploration patterns.

3.2 Parcel Selection and Targeting

The decision of which parcel to pursue is governed by a custom utility function that balances **reward value**, **decay over time**, and **distance**. Each candidate parcel is evaluated through the following scoring metric:

$$\text{efficiency} = \frac{\text{reward} \times \text{timeFactor}}{\text{distance} + 1}$$

where:

- `reward` is the current value of the parcel.
- `timeFactor` is the ratio between current reward and original reward, accounting for decay.
- `distance` is the Manhattan distance from the agent to the parcel.

This formula favors parcels that are both highly valuable and close, while penalizing those that may have decayed or are located far away. The use of the `+1` in the denominator avoids division by zero and slightly prefers nearby parcels.

To prevent inefficient behavior, the system incorporates a **ban mechanism** that temporarily excludes certain parcels or tiles from consideration. This mechanism is activated in the following situations:

- The agent attempts to reach a parcel but repeatedly fails, typically due to another agent blocking access.
- The parcel disappears or becomes unreachable due to environmental changes or conflicts.

Once a ban is triggered, the corresponding target is ignored for a fixed number of turns, allowing the agent to redirect its attention to more viable alternatives.

In addition, a **stuck detection and recovery system** monitors the agent's position across turns. If the agent remains on the same tile for an extended period, indicating that it may be blocked or trapped, the system clears current intentions and replans from scratch.

Together, these single-agent strategies provide a strong foundation for effective performance in isolation. The modularity and reactivity of the system also pave the way for seamless extension to multi-agent coordination, as discussed in the following sections.

4 Multi-Agent Extension

While the single-agent implementation demonstrated strong autonomous behavior, the competitive nature of the simulation called for effective **multi-agent cooperation**. To extend the architecture without introducing a centralized controller, we developed a decentralized coordination framework that allows agents to share information, divide responsibilities, and adapt their strategies dynamically during runtime.

4.1 Coordination between agents

Multi-agent cooperation begins with each agent broadcasting a special `HANDSHAKE` message that includes its unique identifier and current position. When an agent receives a `HANDSHAKE` broadcast from another agent, it stores the received `ID` and recognizes the sender as a teammate.

Once this initial contact is established, the agents switch to **direct communication**, exchanging information about their intended spawn area coverage. Such peer-to-peer interaction allows the agents to organize effectively without conflicts or duplicating efforts.

To organize spatial responsibilities, the map is first processed using a **BFS-based clustering algorithm** that groups adjacent spawn tiles into coherent regions. These clusters are then **sorted by size**, with larger areas assigned first. Each agent selects its patrol region based on this ordering, ensuring that larger and more critical areas are covered first.

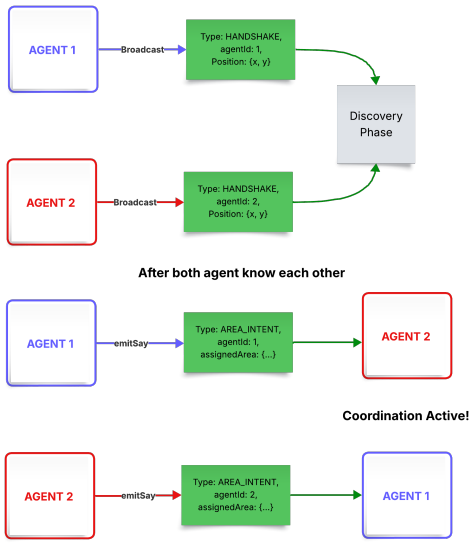

Figure 2: Handshake protocol: agents broadcast their identity and position, establish peer recognition, and proceed with direct coordination.

In edge cases where the map contains only a single spawn area the clustering algorithm identifies this limitation. In such situations, the map is partitioned **either vertically or horizontally**, depending on the most balanced division. This ensures that each agent still operates in a distinct region, maintaining coordination even when the map topology is highly constrained.

4.2 Specialization in Linear Maps

Although the general coordination strategy performs well across most map types, certain topologies like long corridor-like maps, introduce unique challenges. In such cases, agents frequently interfere with each other due to the lack of maneuvering space and bottlenecks in parcel pickup and delivery paths.

To address this, we implemented a role-based strategy that assigns specialized responsibilities to agents. The agent nearest to the spawn tile takes on the role of `RUNNER`, focusing exclusively on collecting parcels from the spawn and delivering them to a shared midpoint. The other agent assumes the role of `CARRIER`, collecting parcels from this midpoint and completing their delivery.

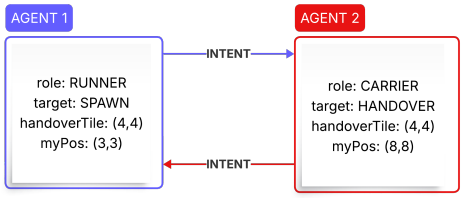

Figure 3: `RUNNER`/`CARRIER` strategy in linear maps: the `RUNNER` focuses on parcel collection and handover; the `CARRIER` completes the delivery. Coordination is maintained via periodic position updates.

This division of labor reduces round-trip time for both agents and ensures that movement through narrow areas is minimized. The agents synchronize their behavior by sharing their current positions and the coordinates of the handover point. In our implementation, this midpoint is dynamically defined during the handshake phase or determined heuristically based on spawn and delivery tile locations.

5 Decision-Making: Option–Intention Loop

The agent's cognitive core is built upon the **Option–Intention Loop**, a continuous cycle that transforms environmental perception into rational and goal-directed action. This loop integrates multiple components including beliefs, intention selection, pathfinding, and execution, into a unified decision process that adapts in real-time to a dynamic, multi-agent environment.

As illustrated in Figure 4, the loop begins with the acquisition of beliefs from the environment, followed by option generation, deliberation, and intention execution.

5.1 Option Generation

At each iteration, the agent evaluates the current world state and produces a set of candidate options. These options represent feasible goals and are categorized into:

- **Delivery Options**, activated when the agent carries parcels and can reach a delivery point. Additional detour opportunities are also considered if beneficial.
- **Collection Options**, evaluated based on parcel visibility and desirability. Each parcel is scored using a custom utility metric, as discussed in Section 3.2.
- **Exploration Options**, triggered when no collection or delivery is optimal. The agent selects target areas according to its active exploration strategy (e.g., CAMPER, FOCUS, or GENERAL).

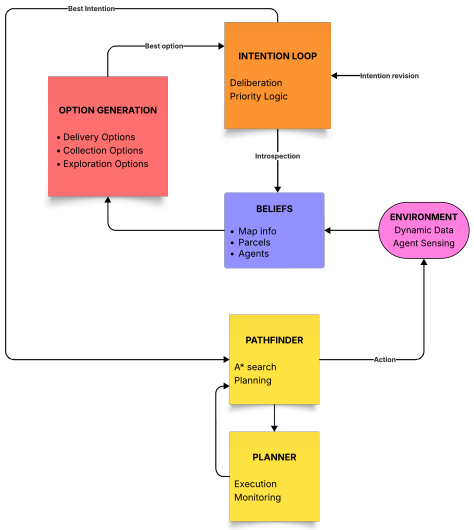

Figure 4: Conceptual overview of the Option–Intention Loop. Each component interacts with others in a continuous cycle of sensing, reasoning, planning, and acting.

For delivery routes that may intersect with parcels, the following score is used to evaluate detour-worthiness:

$$\text{detourScore} = \frac{\text{reward}}{\text{addedSteps} + 1}$$

This promotes short, high-value diversions during delivery when the opportunity cost is low.

5.2 Deliberation and Prioritization

Once options are generated, the agent selects the best intention through a hierarchical prioritization scheme:

1. **Deliver**: Highest priority. Ensures that carried parcels are delivered before their reward decays.
2. **Pick-up**: Next in priority, if valuable and accessible parcels are available.
3. **Explore**: Lowest priority, used to maintain awareness and discover new opportunities.

5.3 Execution and Monitoring

After an intention is chosen, the agent generates a concrete plan using an **A* pathfinding algorithm** with Manhattan distance heuristic:

$$f(n) = g(n) + h(n) \quad \text{where} \quad h(n) = |x_1 - x_2| + |y_1 - y_2|$$

Fallbacks are applied if the target is unreachable, using breadth-first search (BFS) to find nearby alternatives.

Throughout execution, the agent monitors its state and reacts to disruptions using three key mechanisms. These include a **Ban System** that temporarily excludes unreachable parcels or tiles from future decisions; **Timeout Detection**, which triggers replanning if no progress is made within a certain time window; and **Stuck Recovery**, which resets the current intention and restarts the decision cycle when the agent remains immobile for several turns.

5.4 Planning Integration and Use of PDDL

Within the BDI architecture, the **planning phase** bridges the gap between an agent's selected intention and the sequence of actions required to achieve it. In our implementation, the `planner.js` module traditionally relied on heuristics and lightweight algorithms such as A* and BFS. However, we extended this framework by integrating a **PDDL-based planner**.

The PDDL planner is invoked immediately after the deliberation phase, once an intention has been selected. Its role is to transform high-level goals like '*delivery parcel at (x,y)*' into concrete sequences of primitive actions such as `moveUp`, `pickup`, or `putdown`. As illustrated in the Option–Intention Loop (see Figure 4), this corresponds to the transition from intention to planning.

5.4.1 Implementation Details and Agent Extension

Both single-agent case and multi-agent case were extended to support a dual-mode planning strategy:

- **Local planning (A*)**: used by default for standard actions, quick recovery, and fallback when timing is tight.
- **PDDL planning**: activated only when the corresponding flag in `utils.js` is enabled. It is particularly effective in maps with narrow corridors, as once a plan is computed, its results are cached and reused, allowing for faster execution on previously solved paths.

6 Evaluation and Results

To validate our implementation, we conducted a series of tests using the official `Deliveroo.js` simulation server. The server provides different sets of levels, one dedicated to test the agent's performance in straight corridors (`25_hallway`).

All tests were run in 5 minute gameplay sessions to ensure consistency. For planning purposes, we employed a PDDL-based planner executed via the `planning-as-a-service` repository. This tool allowed us to run the planner locally inside a Docker container.

The following tables summarize the results obtained during our testing phase. On the left, we report the scores from the single-agent runs, while on the right we present the results from multi-agent runs, broken down into scores for each individual agent.

Level	A*	PDDL
25c1-1	1623	380
25c1-2	2471	1329
25c1-3	1569	1381
25c1-4	3502	2605
25c1-5	2963	2078
25c1-6	1391	1167
25c1-7	3513	2236
25c1-8	1487	842
25c1-9	9400	5282

Table 1: Single-Agent Results

Level	Total	A1	A2
25c2-1	3235-2736	1639-1540	1596-1196
25c2-2	1931-1094	1146-638	785-456
25c2-3	2519-2140	1645-1341	874-799
25c2-4	7235-1630	4190-724	3045-906
25c2-5	4786-2841	2593-1934	2193-907
25c2-6	4402-3103	2308-1845	2094-1258
25c2-7	11022-4340	8718-2959	2304-1381
25c2_hall	2070-1296	2070-1296	0-0

Table 2: Multi-Agent Results. `A* - PDDL`

The results demonstrate the effectiveness of our agents across both single and multi-agent configurations. In single-agent mode, performance remained consistently strong across various map layouts. In multi-agent settings, the coordination mechanisms and distributed strategies allowed both agents to contribute meaningfully to the total score, with clear advantages in complex or high-density environments.

7 Conclusion and Future Work

In this project, we developed an autonomous agent system based on the Belief–Desire–Intention (BDI) architecture to address the challenges posed by the Deliveroo simulation game. Through a modular and reactive design, our agents demonstrated strong decision-making capabilities, effective exploration, and robust cooperation in both single and multi-agent settings.
The introduction of decentralized coordination strategies, combined with dynamic role assignment in linear maps, proved particularly effective in improving performance and minimizing agent interference. Our solution consistently achieved competitive results, culminating in a second-place finish in the final challenge.

Looking forward, several directions for improvement remain open. Reducing path planning latency through more efficient PDDL integration or hybrid planning methods could enhance real-time responsiveness, for example in situations where agent movement speed is low, as the increased planning time would not significantly affect reactivity.

Another challenge we observed relates to parcel spawning and agent positioning. Although camping on large spawning tile clusters might seem advantageous, it often causes agents to ignore parcels that spawn in less concentrated but still significant areas of the map. This strategy can result in missed opportunities and lower overall efficiency. A more balanced exploration strategy could help mitigate this issue.

Furthermore, there is room to leverage the data provided by the APIs in more intelligent and adaptive ways. For instance, in scenarios with high parcel spawning rates or high parcel rewards, it may be more beneficial to collect multiple parcels before delivering them, rather than delivering immediately as we currently do.

Finally, further refinement of the communication protocol and dynamic adjustment of roles based on map structure may lead to even stronger multi-agent performance. Adopting a runner-carrier strategy on non-linear maps may be an effective strategy on future experiments, provided that only our agents are present, as external agents may interfere and steal our parcels.