Autonomous Software Agents Project Report

Leonardo Bottona - 258395
leonardo.bottona@studenti.unitn.it

Gabriele Masciulli - 258394
gabriele.masciulli@studenti.unitn.it

September 5, 2025

1 Introduction

1.1 Problem Statement

This project addresses the theoretical and methodological challenges of implementing a multi-agent system for autonomous delivery coordination, where agents must simultaneously compete for resources (parcels), while potentially cooperating to reach team objectives (score maximization). The relevant aspects of the project can be summarized into three critical points: spatial reasoning under uncertainty, temporal planning with dynamic constraints, and strategic decision-making in the presence of adversarial agents.

2 System architecture

Our system implements a BDI (belief–desire–intention) architecture that addresses the fundamental challenge of bounded rationality in artificial agents by providing a structured approach to intention management. The architecture's strength lies in its ability to balance deliberation with reactivity, enabling agents to pursue long-term goals while responding appropriately to context changes. More specifically, the system architecture follows a layered approach that separates concerns, while maintaining tight integration between components.

There are four main layers:

1. **Perception Layer**: Responsible for environmental sensing and belief updating.
2. **Reasoning Layer**: Implements the BDI reasoning cycle and intention management.
3. **Planning Layer**: Chooses between PDDL and local planning through our custom local path finder.
4. **Action Layer**: Handles low-level action execution and environmental interaction.

2.1 Belief Set

The **Belief Set** serves as the central knowledge repository for the autonomous agent, maintaining a comprehensive world model that integrates information from both the perceptions retrieved by the agent itself and those communicated by its teammate in the co-op scenario. Its responsibility is to maintain a consistent and query-efficient world model that other decision-making modules (desire generation, intention selection) can rely upon.

The Belief Set maintains and updates the following information:

Agent State Management: The system tracks the current agent's position, movement status, and carried parcels, alongside teammate information when operating in multi-agent scenarios. This includes real-time monitoring of agent coordinates (when sensed) and movement direction detection.

Environment Representation: A complete grid-based map representation stores the spatial layout of the environment. The system automatically identifies and marks parcel generator tiles and delivery zones as strategic locations during map initialization, providing the foundation for spatial reasoning, path planning and map partitioning. The longest traversable path on the map is also precomputed (*longestPathLength*) to calibrate memory-expiration policies and estimate worst-case travel times.

Parcel Management: A dynamic registry of all known parcels is kept, implementing intelligent parcel life-cycle management with reward decay calculations. Parcels are either classified as actively sensed or outdated (last known state), with automatic cleanup of expired parcels based on a lazy deletion approach, triggered whenever parcels are accessed. To optimize queries, active parcels are indexed by their coordinates, allowing **O(1) spatial lookups** for checking whether the agent is standing on a pickup opportunity.

Multi-Agent Awareness: Other agents' states are maintained in a dedicated map structure. Each occupied position is stored together with a timestamp, enabling the system to automatically discard stale occupancy data after a period proportional to the longest path traversal time. This ensures a balance between accurate collision prediction and memory efficiency.

Strategic Intelligence: Advanced features include Voronoi-based territorial partitioning for multi-agent coordination. These capabilities enable sophisticated coordination strategies and resource allocation decisions.

The Belief Set employs lazy evaluation and efficient data structures to minimize computational overhead while providing O(1) lookup times for critical spatial queries. The component automatically handles information decay, agent disappearance, and state inconsistencies, ensuring robust operation in dynamic environments with partial observability.

3 Methodology

Explain your approach and design decisions.

3.1 Intentions

At the core of out agent architecture lies the process of **intention generation**, which determines the agent's course of action. The Agent continuously evaluates its current situation, the parcels available in the environment, and its own state (e.g., whether it is carrying parcels, its current position with respect to other agents and delivery zones). Each option corresponds to a potential **desire**, which may or may not become an actual intention depending on its computed **utility score** given by the current context.

The agent distinguishes between three primary types of intentions i.e., pickup, delivery and exploration. In addition to these primary desires, the agent also accounts for two **special-case intentions** that override the normal option generation process i.e. immediate pickup and delivery.

3.1.1 Pickup

The desire to collect a parcel from the map. Pickup intentions are generated when parcels are available in the environment and not already carried by another agent. In cooperative mode, an agent only considers parcels within its assigned partition of the map (or in special hallway

scenarios where cooperation requires one agent to handle parcels outside its zone). Pickup intentions are evaluated by estimating the utility of retrieving a parcel, taking into account its reward, the distance to the parcel, and the agent's current carrying load.

Pickup utility. Let p be a candidate parcel at position (x_p, y_p) with current reward r_p. Let the agent be at position (x_a, y_a), carrying N parcels whose total reward is R_c. We denote by τ the per-move travel time (`MOVEMENT_DURATION`) and by Δ the decay interval in milliseconds (`PARCEL_DECADING_INTERVAL`). Let $\lceil \cdot \rceil$ be the ceiling function. We assume shortest paths are used.

Travel times. Let c_{pickup} be the path cost (in moves) from (x_a, y_a) to (x_p, y_p), and c_{delivery} the path cost from (x_p, y_p) to the closest delivery zone. Then

$$T_{\text{pickup}} \;=\; c_{\text{pickup}}\,\tau, \qquad T_{\text{delivery}} \;=\; c_{\text{delivery}}\,\tau, \qquad T_{\text{tot}} \;=\; T_{\text{pickup}} + T_{\text{delivery}}.$$

Decay counts. Rewards decay in discrete ticks every Δ milliseconds. The number of decay ticks that elapse while traveling is

$$D_{\text{pickup}} \;=\; \left\lceil \frac{T_{\text{pickup}}}{\Delta} \right\rceil, \qquad D_{\text{delivery}} \;=\; \left\lceil \frac{T_{\text{delivery}}}{\Delta} \right\rceil.$$

Carried parcels' final reward. While moving to p the agent carries N parcels, and after picking up p it carries $(N{+}1)$ until delivery. Thus the carried-total reward after all decay is

$$R_c^{\text{final}} \;=\; \max\Big\{0,\; R_c \;-\; D_{\text{pickup}} \cdot N \;-\; D_{\text{delivery}} \cdot (N{+}1)\Big\}.$$

Target parcel's final reward. Let $\theta(p) \geq 0$ be the *threat penalty* assigned to parcel p (due to other agents nearby the considered parcel) 3.1.1. The reward of p at pickup time is

$$r_p^{\text{at-pickup}} \;=\; r_p \;-\; D_{\text{pickup}} \;-\; \theta(p),$$

and after carrying it to delivery together with the other parcels,

$$R_p^{\text{final}} \;=\; \max\Big\{0,\; r_p^{\text{at-pickup}} \;-\; D_{\text{delivery}} \cdot (N{+}1)\Big\}.$$

Threath penalty. The threat assessment component evaluates competitive pressure from other agents when deciding which parcels to pick up. This mechanism prevents agents from engaging in useless competition for resources that are likely to be claimed by closer or better-positioned competitors.

The threat calculation combines proximity-based and directional factors to assess the likelihood that another agent will reach a target parcel first. For a given parcel, the total threat T is computed by aggregating individual agent threats:

$$T = \sum_{j \neq i} T_j$$

where T_j represents the threat posed by agent j. Each agent's threat contribution consists of a base proximity component and an optional directional bonus:

$$T_j = \frac{\alpha}{d_j^2} \times \begin{cases} \beta & \text{if agent } j \text{ is stationary} \\ \beta + \gamma \cdot \max(0, \frac{\vec{v_j} \cdot \vec{u_j}}{|\vec{u_j}|}) & \text{if agent } j \text{ is moving} \end{cases}$$

where d_j is the path finding distance from agent j to the parcel, $\vec{v_j}$ is agent j. This approach promotes efficient resource allocation and reduces wasted effort in multi-agent scenarios. The movement vector, $\vec{u_j}$ is the vector from agent j to the parcel, and α, β, γ are tunable parameters controlling threat intensity.

The inverse square distance relationship ensures that nearby agents dominate the threat assessment, while the directional component captures immediate competitive intent when agents move toward the target. The final threat value is incorporated into the utility calculation as a penalty term, effectively reducing the attractiveness of highly contested parcels and guiding agents toward less competitive opportunities. This approach reduces wasted effort in multi-agent scenarios.

The final pickup utility balances the final total reward against the total time:

$$U_{\text{pickup}}(p) = \begin{cases} \dfrac{R_c^{\text{final}} + R_p^{\text{final}}}{T_{\text{tot}}}, & \text{if both paths to pickup and delivery zone exist,} \\ -\infty, & \text{if } (x_a, y_a) \to (x_p, y_p) \text{ or } (x_p, y_p) \to \text{Delivery Zone is unreachable.} \end{cases}$$

3.1.2 Delivery

The desire to bring carried parcels to a delivery zone. Delivery intentions are triggered whenever the agent is already carrying at least one parcel. If a delivery zone is accessible from where the agent is at any given time, the agent considers the potential utility of completing a delivery based on the accumulated reward and the number of parcels it is holding. In multi-agent mode, if a direct delivery zone is not reachable or available, the agent may instead generate a delivery option towards a teammate, effectively handing over the parcels for completion of the task to the other agent.

Delivery utility. Similarly to the parcel utility computation, the delivery utility is computed as the ratio between the final reward of the parcels being carried and the total travel time, assuming a path exists:

$$U_{\text{delivery}} = \begin{cases} \dfrac{R_c^{\text{final}}}{T_{\text{del}}}, & \text{if } (x_a, y_a) \to \text{Delivery Zone exists,} \\ -\infty, & \text{otherwise.} \end{cases}$$

3.1.3 Exploration

The desire to search the environment in the absence of high-utility pickup or delivery options. Exploration intentions are generated when the agent has no current intention and no suitable parcels or deliveries are available. In this case, the agent assigns a neutral utility value to exploration and navigates either towards a generator tile in its assigned area or to a random position in the map with the idea of uncovering new opportunities (i.e. sensing new parcels). Although exploration does not generate immediate reward, it increases the likelihood of discovering new parcels, thus feeding into future high-utility options.

3.1.4 Immediate Pickup

At any given time, if the agent happens to be located on top of a parcel during its journey, it immediately triggers a pickup intention with highest priority. This ensures that no valuable parcel is overlooked simply because the agent was on its way to another goal.

3.1.5 Immediate Delivery

Similarly to the immediate pickup desire, if the agent is carrying parcels and finds itself on a delivery zone, it immediately triggers a delivery intention. This guarantees that carried parcels are offloaded at the earliest opportunity, maximizing reward, even when other intentions might seem more profitable.

By combining these mechanisms, the agent can implement both a robust and flexible decision-making strategy. High-priority, special intentions safeguard against missed opportunities, while utility-based evaluation of pickup and delivery options balances long-term reward maximization with immediate awareness i.e., greediness.

3.2 Path Finding

3.2.1 A*

To enable the agent to move efficiently within the environment, we employ the **A*** (A-star) pathfinding algorithm. The environment is represented as a **grid**, where each position corresponds to a tile that may either be traversable or blocked by another agent. The agent's knowledge of the world is continuously updated through its **belief set**, which accounts for the presence of other agents occupying certain tiles. This allows the pathfinding process to adapt dynamically to changes in the environment.

The A* algorithm operates by combining two key ideas:

- the *cost of the path so far* (i.e., the distance traveled from the starting position),
- an *estimate of the remaining distance* to the goal, provided by a heuristic function (Manhattan Distance in our case).

By summing these two components, the agent not only reaches its destination but does so in a way that balances **optimality** (shortest possible path) with **performance** (avoiding unnecessary exploration).

When the algorithm identifies a valid path, it outputs it as a sequence of directional moves guiding the agent from its starting point to the target location. If no feasible path exists—due to obstacles or occupied positions—the system returns failure, allowing higher-level decision-making processes to adjust accordingly.

3.2.2 PDDL

The planning component of our system is based on PDDL (Planning Domain Definition Language), which provides a formal way to describe the environment, agents, and possible actions. In our domain file, we define two types: `agent` and `tile`, along with predicates to represent agent positions, tile adjacency, and tile availability.

Initially, our approach involved creating a separate action for each possible move direction. However, we realized that a single, parameterized `move` action was sufficient to capture all movement possibilities. This simplification leverages the `adjacent` predicate, allowing the planner to generate valid moves in any direction based on the current map configuration.

The `move` action is defined as follows:

Listing 1: PDDL action schema for moving an agent

```
(:action move
  :parameters (?a - agent ?from - tile ?to - tile)
  :precondition (and
    (at ?a ?from)
    (adjacent ?from ?to)
    (free ?to))
  :effect (and
    (not (at ?a ?from))
    (at ?a ?to)
    (free ?from)
    (not (free ?to))))
```

This design makes the planning process more efficient and scalable, as the solver can handle any movement scenario using a single action schema. The problem file further specifies the map layout through tile adjacency relationships, tile occupancy, and initial agent positions, enabling the planner to compute optimal paths and actions for any `GO_TO` plan.

Instead of manually writing static problem files, we dynamically generate the PDDL problem based on the current state of the environment (i.e. based on the information from the belief set) allowing for dynamic creation of objects, initial conditions, and goals, reflecting real-time changes in the map and agent positions.

By automating the generation of the PDDL problem, agents can adapt to new delivery tasks, obstacles, or map updates without manual intervention. This flexibility is crucial for simulating realistic scenarios and testing the robustness of the planning system. The code constructs the list of tiles, agents, adjacency relations, and sets the initial and goal states, ensuring that the planner always receives an accurate representation of the environment.

3.3 Agent Cooperation

In cooperative mode, agents establish a secure and synchronized communication channel before sharing information. This is achieved through a lightweight, three-step handshake protocol followed by structured messaging. The cooperation protocol ensures that only teammates belonging to the same team key exchange information and that all messages can be associated with a valid session.

3.3.1 Handshake

The handshake is inspired by standard three-way handshake patterns in distributed systems. Its goal is to create a session identifier shared between two teammates, which acts as a reference for all subsequent communication.

1. **Discovery.** Agents broadcast a `HELLO` message containing their `teamId`, `agentId`, and a timestamp. This allows team members in the environment to detect each other. A simple tie-breaking rule ensures determinism: the agent with the lower ID initiates the handshake.

2. **Initialization.** The initiator generates a fresh *nonce* (a random UUID) and sends a `HANDSHAKE_INIT` message containing the team key, the nonce, and its identity.

3. **Acknowledgment.** The partner responds with a `HANDSHAKE_ACK` message. This echoes back the received nonce, provides a newly generated *sessionId*, and confirms the shared team key.

4. **Confirmation.** The initiator verifies the echoed nonce and, if valid, sends a final `HANDSHAKE_CONFIRM` message containing the sessionId. At this point, both agents store the sessionId and mark each other as trusted teammates.

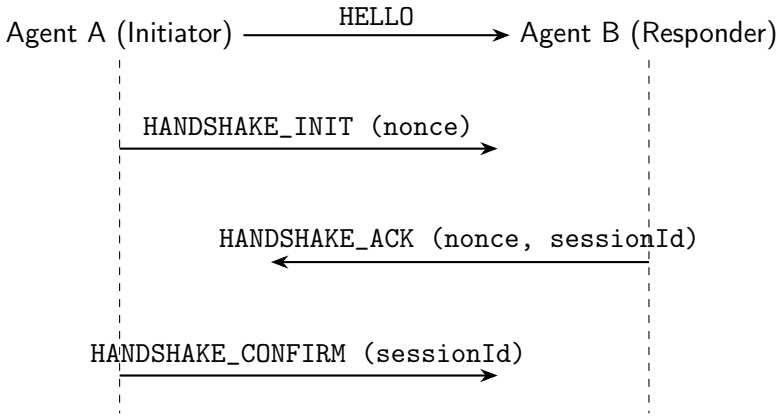

Figure 1: Three-way handshake protocol between teammates.

3.3.2 Messaging

Once the handshake is complete, all communication between teammates is associated with the negotiated sessionId. The following message types are supported:

- **Parcels Sensed (`PARCELS_SENSED`):** Shares the list of parcels detected by the agent's sensors. This helps synchronize knowledge about task opportunities in the environment.
- **Agents Sensed (`AGENTS_SENSED`):** Broadcasts the positions or states of other non-teammate agents. This allows teammates to reason about competition and avoid redundant actions.
- **My Info (`MY_INFO`):** Communicates the current state of the agent its position.
- **Map Partitioning (`MAP_PARTITIONING`):** Distributes a partition of the environment map. This is useful for dividing exploration or task allocation among teammates without redundancy.

3.4 Map partitioning

In cooperative mode, agents must coordinate to divide the environment efficiently while avoiding redundant effort and resource conflicts. This coordination is achieved through a deterministic *map partitioning* mechanism that combines territorial division with utility-based decision making.

Once the handshake is complete, the agent which initiated the handshake becomes the partitioning coordinator. In this way, only one agent performs the computation, while the other simply adopts the result, thereby avoiding conflicts or inconsistencies.

The designated coordinator computes the Voronoi-based division of the map into regions, assigning each agent a distinct area in which to operate. The resulting partitioning is then shared with the teammate, ensuring both agents maintain a consistent understanding of territorial boundaries.

Importantly, the partitioning is not static. It is updated whenever significant events occur in the environment, such as the *pickup* or *delivery* of a parcel, or a new *sensing* update that provides fresh information about the map.

3.4.1 Voronoi-Based Territorial Assignment

The spatial coordination problem can be formalized as an optimization problem over the space of possible territorial assignments. Given a set of agents $A = \{a_1, a_2, \ldots, a_n\}$ (in our case the two teammate agents) and a set of resource locations $R = \{r_1, r_2, \ldots, r_m\}$ (in our case the parcel generator locations), the objective is to find a partitioning function $\pi : R \to A$ that optimizes task assignments.

The Voronoi diagram provides a natural solution to this partitioning problem. For each agent of the team a_i (in our scenario only two agents are being considered) located at position p_i, the Voronoi cell V_i is defined as:

$$V_i = \big\{(x, y) \in \text{Map} \ : \ d\big((x, y), p_i\big) \leq d\big((x, y), p_j\big), \ \ \forall j \neq i\big\},$$

where $d(\cdot, \cdot)$ denotes the shortest-path distance in the environment map.

This partitioning minimizes the total travel distance while ensuring that each resource is assigned to exactly one agent. The dynamic nature of the environment requires continuous recomputation as agents move and new resources appear.

3.4.2 Dynamic Rebalancing

The partitioning **rebalancing** mechanism ensures that the division of labor remains fair and efficient once the initial partitioning is computed. The system first defines a target capacity for each agent, corresponding to the number of generators that the agent should ideally manage. Agents that exceed this capacity are marked as overloaded, while those below capacity are underloaded.

During rebalancing, generators are iteratively reassigned from overloaded to underloaded agents. The choice of reassignment is guided by minimizing the additional travel cost: for each candidate generator, the algorithm evaluates the distance difference between the current and the potential new agent, selecting the transfer that introduces the smallest penalty. This process continues until all agents operate within their assigned capacity, resulting in a balanced and spatially efficient partitioning of parcel generators.

4 Experiments and Results

Our agent system has been evaluated across multiple scenarios to assess its effectiveness in both single-agent and multi-agent configurations. Testing was conducted using standardized benchmark scenarios with controlled parameters to ensure consistent evaluation conditions.

4.1 Testing Methodology

All tests were executed using scenarios from the 25c series with the following standardized parameters:

- **Duration**: 2 minutes per scenario
- **Competition**: 4 random agents as opponents
- **Enemy Movement Speed**: 1 second
- **Evaluation Metric**: Total reward points accumulated

The test suite comprised 17 distinct scenarios divided into single-agent and multi-agent configurations, each designed to evaluate different aspects of agent performance including pathfinding efficiency, resource prioritization, and coordination capabilities.

4.2 Single-Agent Performance

The single-agent configuration evaluated the system's fundamental capabilities in competitive environments without teammate coordination. Performance varied significantly across scenarios, demonstrating the system's adaptability to different environmental conditions. From an empirical perspective, these are the results:

Scenario	**Score (pts)**
25c1_1	400
25c1_2	1204
25c1_3	816
25c1_4	2211
25c1_5	2677
25c1_6	577
25c1_7	669
25c1_8	543
25c1_9	3320
Average	**1268.6**

Table 1: Single-agent performance across test scenarios

While evaluating these scenarios, we observed several behaviors that highlight the strong theoretical foundation upon which our agent based its decisions:

- Its ability to re-evaluate its intentions in a reactive manner consistent with its beliefs and the fast-changing environment.
- Its ability to avoid paths that might lead to collisions with enemy agents, thus demonstrating the robust implementation of our pathfinder.

4.3 Multi-Agent Performance

The multi-agent configuration tested the cooperative coordination system, including Voronoi-based map partitioning and dynamic rebalancing mechanisms. Results demonstrate the system's ability to coordinate effectively while maintaining individual agent performance.

Scenario	**Agent 1 (pts)**	**Agent 2 (pts)**	**Total (pts)**
25c2_1	597	551	1148
25c2_2	296	284	580
25c2_3	1018	1077	2095
25c2_4	1055	919	1974
25c2_5	366	467	833
25c2_6	596	881	1477
25c2_7	614	1608	2222
25c2_hallway	0	889	889
Average	**567.8**	**834.5**	**1402.3**

Table 2: Multi-agent cooperative performance across test scenarios

The multi-agent results reveal interesting coordination dynamics, for example:

- In general, we observed how sharing beliefs between the two agents proved to be effective in cases where the sensing capabilities of one agent were not sufficient to generate valuable intentions. Hence, relying on sensed information from the other agent would generate much more rewarding intentions.

- From map 25c2_7, we observed how powerful our map partitioning logic was. In fact, the agents split the parcel generator tiles in an effective way, avoiding overlaps. Also, the existence of just one delivery zone (constrained resources) was well addressed without any conflicts.
- As we previously mentioned in Section 3.1.2, in the hallway scenario, simply allowing one agent to deliver the carried parcels to a non-delivery tile near its teammate when no delivery zones were available to it proved effective, as it implied handing over the task to the teammate. This behavior allowed us to obtain consistent results (reward maximization) even in hostile environments.

5 Conclusion

The experimental evaluation of our system highlighted several strengths in the architectural and algorithmic choices, but also revealed certain limitations that constrained overall performance in specific settings. In what follows, we discuss the aspects that worked well, and those that did not.

Strengths

Pathfinding efficiency. The adoption of the A* algorithm with Manhattan distance as the heuristic function significantly reduced path computation times compared to uninformed search methods such as BFS or Dijkstra's algorithm. The integration of collision avoidance -achieved by dynamically factoring other agents' positions into the pathfinding process- further reduced the number of collision penalties incurred during navigation. Crucially, the frequent regeneration of options, triggered by continuous belief updates, allowed the pathfinder to operate on up-to-date information and made it a robust component in a dynamic and partially observable environment.

Utility-based decision making. The utility computation framework proved to be the most impactful element of the system. By explicitly incorporating reward decay, carrying capacity, and travel time, the utility function captured the essential trade-offs of the task environment. The introduction of a *threat penalty*, which adjusted parcel attractiveness based on the proximity and direction of competitor agents, prevented wasteful competition for contested parcels. This ensured that agents made context-sensitive decisions, closely aligned with the underlying problem structure.

Greedy opportunism. The inclusion of immediate pickup and delivery mechanisms introduced a beneficial layer of opportunism. These greedy intentions allowed agents to exploit unexpected opportunities in a highly dynamic, partially observable environment without undermining long-term reward maximization. In practice, these mechanisms prevented agents from missing parcels or delivery opportunities that happened to fall directly within their path, thereby improving responsiveness and adaptability.

Communication and coordination. The communication layer proved highly effective in cooperative scenarios. The sharing of parcel and opponent beliefs ensured that both teammates had access to a broader and more accurate world model than they could have maintained individually. The addition of map partitioning, combined with dynamic rebalancing, minimized redundant effort, avoided conflicting actions, and maximized team reward by ensuring coverage of unexplored or less contested areas. This mechanism, coupled with belief sharing, produced consistently higher team-level performance than independent operation.

Limitations

Planning with PDDL. While theoretically sound, the use of PDDL for online path planning revealed itself as a major bottleneck in practice. The limitation stemmed not from the domain model itself, but from reliance on the online solver provided by the planning library. Solver calls introduced latencies in the order of 1000–3000 milliseconds per path, which is incompatible with the demands of a highly dynamic environment where hundreds of path computations may be required each second. As a result, computed paths frequently arrived too late to be useful: by the time a solution was returned, the environment had changed, making the path obsolete. This degraded agent reliability and predictability undermined the overall responsiveness of the system. Consequently, while PDDL remains a promising tool for offline reasoning and scenario analysis, it proved unsuitable as the primary online planning mechanism within this setting.

In summary, the combination of A* pathfinding, utility-based decision making, opportunistic intentions, and cooperative communication formed a robust and adaptive framework for autonomous agents in dynamic delivery environments. However, the reliance on an online PDDL solver highlighted the importance of aligning planning tools with the temporal constraints of the domain. Future work should investigate hybrid approaches where PDDL is reserved for higher-level strategic reasoning, while local pathfinding and decision-making continue to rely on lightweight, real-time methods.