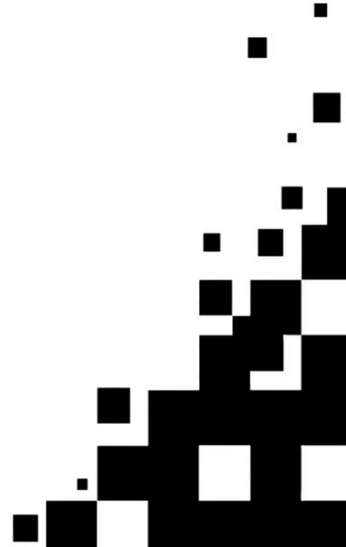
UNIVERSITÀ
DI TRENTO

Università Degli Studi di Trento
Artificial Intelligence Systems

Designing a BDI agent for Deliveroo.js

Autonomus Software Agents Project



Project Partner

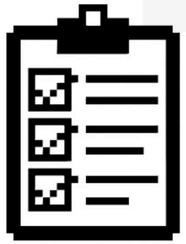
Sorrentino Francesco

Richichi Andrea

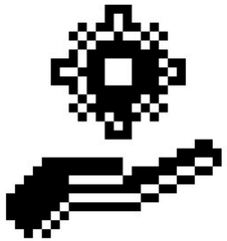
Salas Ichocan Rodrigo

Introduction

Develop BDI-based agents for Deliveroo.js capable of planning, adapting, and cooperating in dynamic environments.



- Designed a single-agent system based on Belief-Desire-Intention logic.
- Developed cooperative behaviors for a two-agent team with real-time communication.
- Integrated PDDL planning to solve complex delivery scenarios in constrained maps.



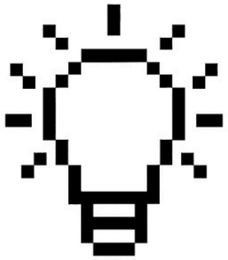
Deliveroo.js offers a controlled environment for testing decision-making and coordination in autonomous multi-agent systems.

Single Agent Design

BDI (Belief–Desire–Intention) agents update beliefs, select intentions from desires, and plan actions.

The model balances reactivity and goal pursuit in dynamic, partially observable environments.

Belief Revision



Intent Revision

Modular Planning

Belief Revision

The `Belief` class mirrors the game's callback structure, enabling continuous updates of map, agent, and parcel knowledge.
Spatial awareness is handled by `DeliverooMap`, which supports both local reasoning and PDDL integration via `mapBeliefSet`.

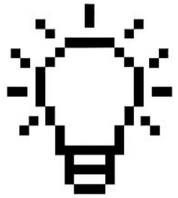
```
config = {}; // Configuration data of the game
me = {}; // Info about the agent
time = {}; // Time inside the game
map = new DeliverooMap(); // Belief regarding the game map
agentBelief = new Map(); // Knowledge about agents' states
parcelBelief = new Map(); // Knowledge about parcels' states
```

Agent & Environment Belief State

```
width; height;
 map = []; // Map that gets updated with agents movement
 originalMap = [];
 deliveryTiles = []; // List of delivery tiles
 spawnTiles = []; // List of spawn tiles
 deliveryMap = []; // List of movement to reach the nearest delivery
 spawnType = ""; // Type of map, based on spawn tiles disposition
 mapBeliefSet = new Beliefset(); // Used for PDDL
 POD = 0; // Parcel observation distance
```

DeliverooMap structure

Intent Revision

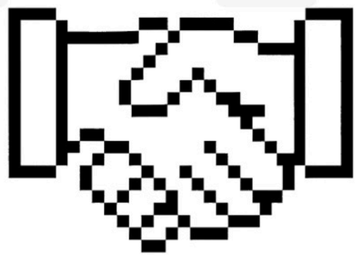


Intention revision continuously selects the most relevant action for the agent to pursue.

1. Both `OptionHandling` and `IntentionRevision.loop` run concurrently throughout the game.
2. Options are generated with a `type` (PickUp, Delivery, Idle) and a computed `priority`.
3. Only the highest-priority option is kept and evaluated for validity.
4. If valid (e.g. parcel still exists, agent carries items), it is turned into a plan.
5. Obsolete options are removed; higher-priority new options can interrupt the current plan.

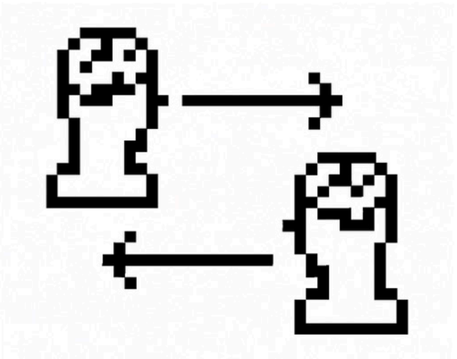
Team of Agents

Extend the BDI model to a multi-agent setting, enabling coordinated decision-making and collaboration through real-time communication protocols.



Handshake

Agents connect and identify each other before collaborating

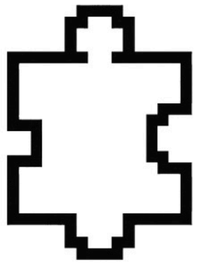


Belief & Option Sharing

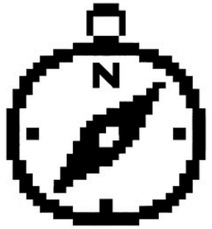
Agents exchange state info and coordinate planned actions.

PDDL Extensions

Use PDDL (Planning Domain Definition Language) to formally describe environments and actions, enabling agents to generate flexible, goal-driven plans in complex tasks.
The three components below will be explored in the following slides.



Domain Formalization



Single Agent Pathfinding

Alleyway Coordination

1. Domain Formalization

The environment is described using PDDL objects, predicates, and logical actions.

- **Objects**

  Represent entities in the world, such as tile, agent, and parcel.
  These are the references used in predicates and actions.

- **Predicates**

  Define relationships and properties like position, connectivity, and occupancy.
  Examples: `at`, `occupied`, `right`, `left`, `agent`.

- **Actions**

  Specify agent capabilities using logical preconditions and effects.
  Examples: `move`, `pickup`, `putdown`.

2. Single Agent Pathfinding

PDDL is used to plan paths between tiles instead of hardcoding movement logic.

- **Goal definition**

  The agent defines a goal tile (e.g. where a parcel is or should be delivered).

- **Problem file generation**

  Based on current beliefs (e.g. tile connections, agent position), a PDDL problem file is created.

- **Planner invocation**

  An external planner (e.g. FastDownward) returns a valid action sequence.

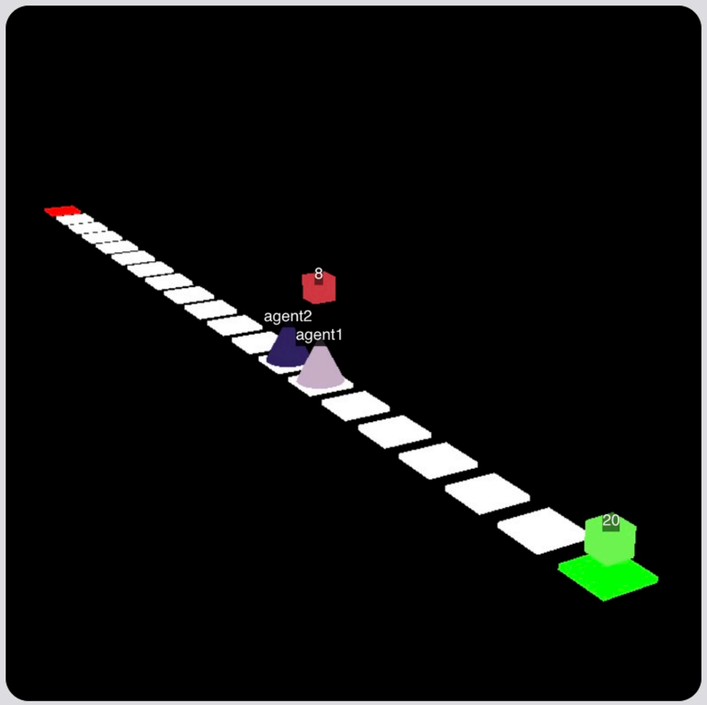
- **Plan execution**

  The agent executes the actions step-by-step, checking for interruptions.

3. Alleyway Coordination

The collector agent builds a PDDL problem including both agents and the parcel, defining initial positions and a delivery goal. The plan is executed step-by-step via synchronized agent communication.

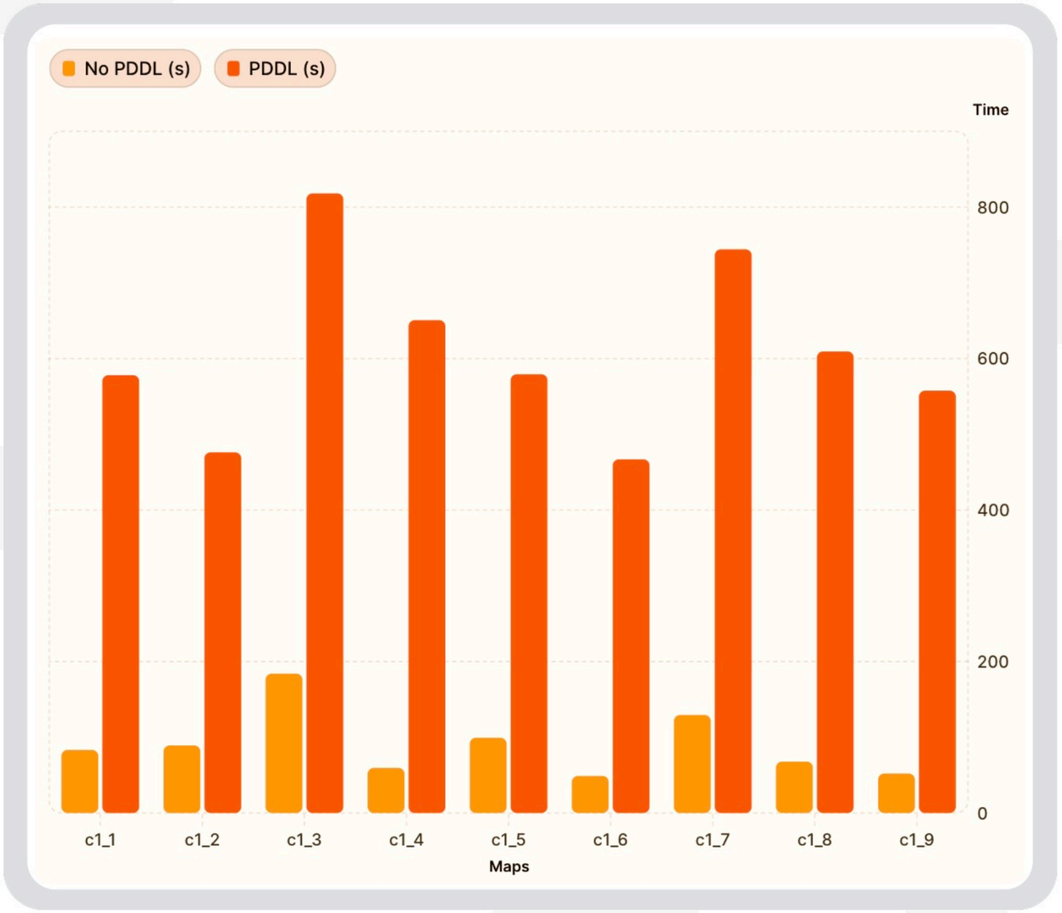
```
// Get objects of the problem
const objectList = [...myBelief.map.mapBeliefSet.objects, myAgentName, friendAgentName,parcelName];
const objects = objectList.join(' ');


 // Set positions on the map
 const initState = myBelief.map.mapBeliefSet.toPddlString()
+ ` (agent ${myAgentName})`
+ ` (at ${myAgentName} ${myCurrentTile})`
+ ` (agent ${friendAgentName})`
+ ` (at ${friendAgentName} ${friendCurrentTile})`
+ ` (parcel ${parcelName})`
+ ` (at ${parcelName} ${parcelPos})`;

// Construct target goal predicate
 const goal = `and (at ${parcelName} ${targetPos})`;
```



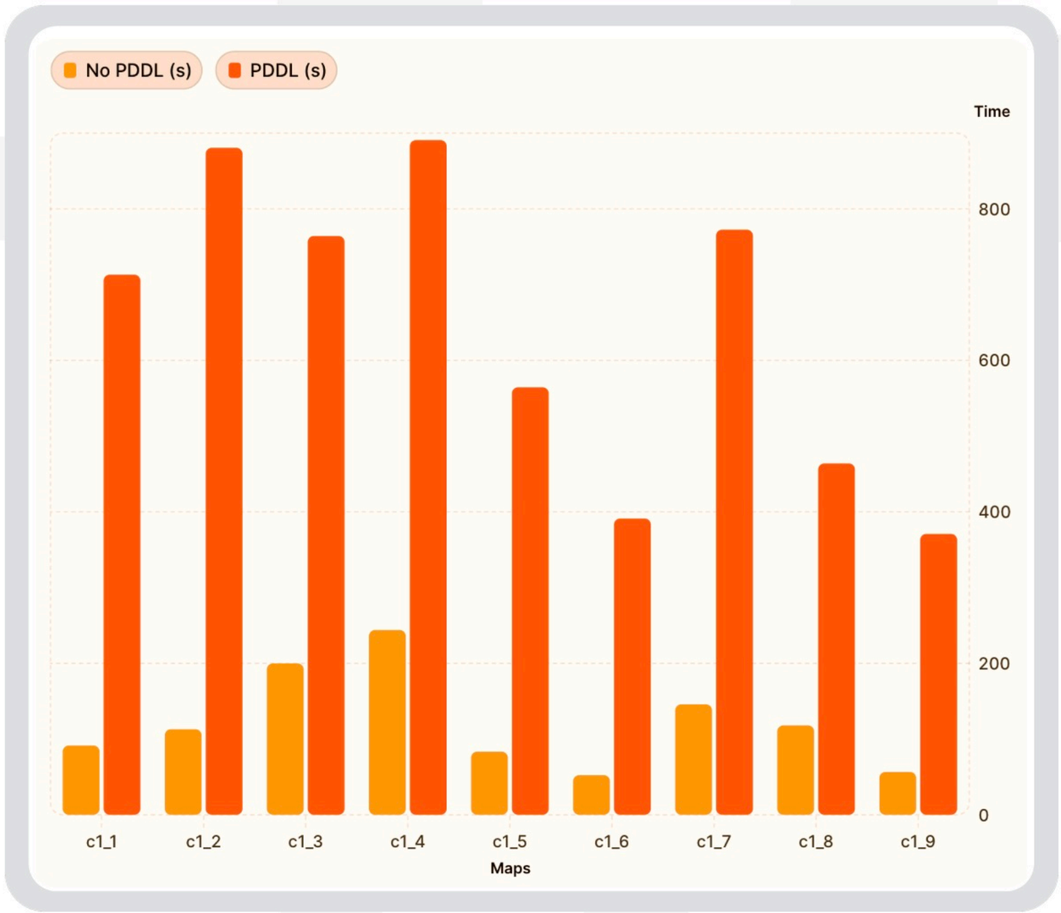
Testing

Testing was conducted across C1 and C2 maps to evaluate single-agent autonomy and multi-agent cooperation.

Single Agent 24

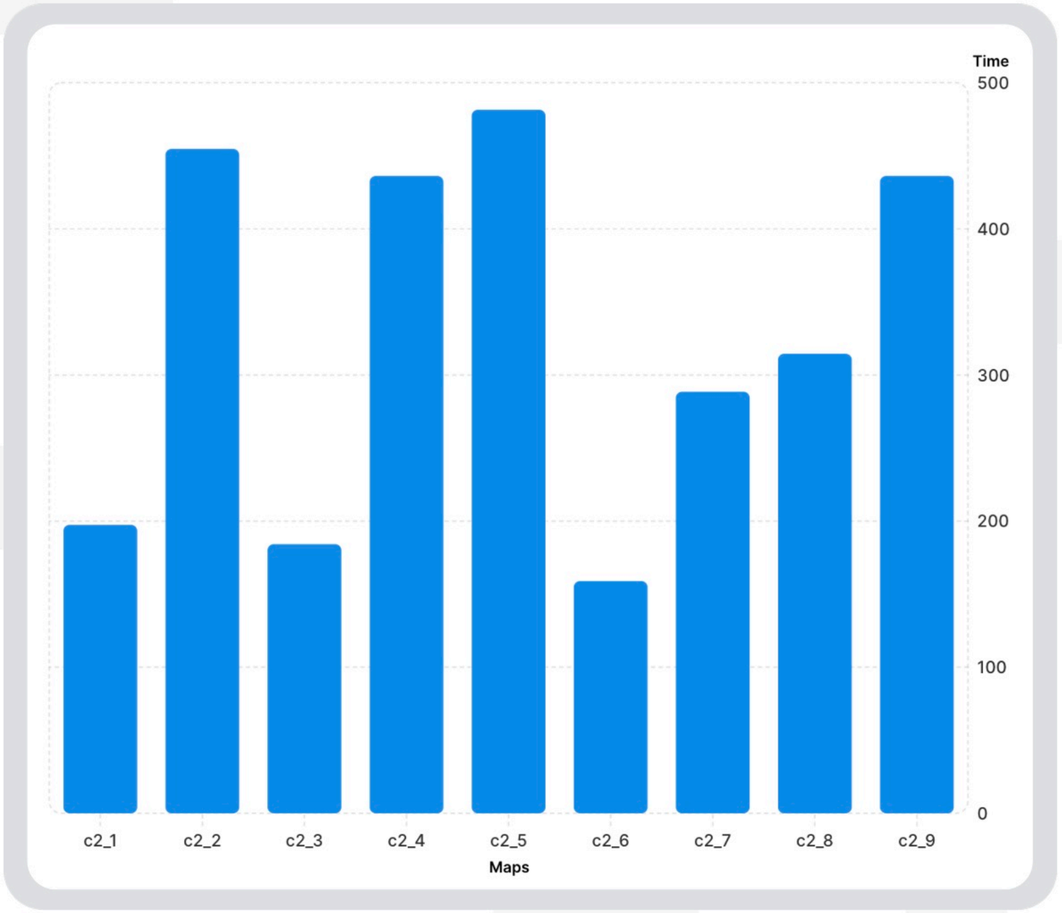


Single Agent 25

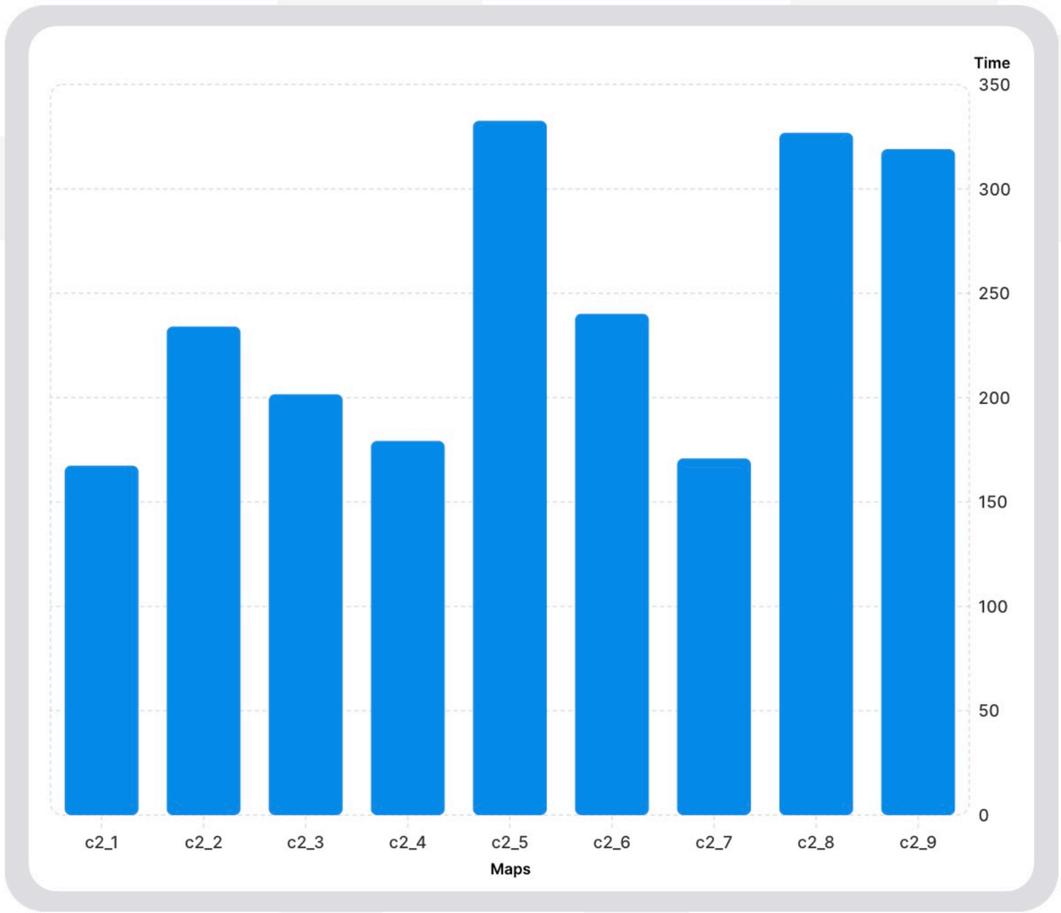


PDDL is slower due to external API-based planning.
No PDDL is faster, yet less reliable on complex maps.

Multi Agent 24



Multi Agent 25



Multi-agent performance depends heavily on coordination quality and collision management across agents

Conclusion

Designing this project offered valuable insights into **BDI agents** acting in **dynamic**, **multi-agent** environments.

Extending the system with **PDDL** enabled more formal reasoning and improved the agent’s ability to handle complex scenarios

THANKS FOR YOUR ATTENTION

Feel free to ask any questions