

Autonomous Software Agents

Martina Panini Avanzolini Elia
A.A. 2024/2025



Develop an autonomous software agent(s) that is able to play [Deliveroo.js](https://www.deliveroo.js.com/) game, in which it gains points by collecting and retrieving parcels.



Implement a Belief-Desire-Intentions (BDI) architecture in Javascript.



Implement single agent architecture



Use PDDL to make agent design a plan



Add another agent and communication to allow cooperation

Belief:

Represent agent's perception of its environment. Agent perceives:

- Itself
- Parcels
- Other Agents
- Map and its updating

Desire:

Goals that agent want to reach.

Intentions:

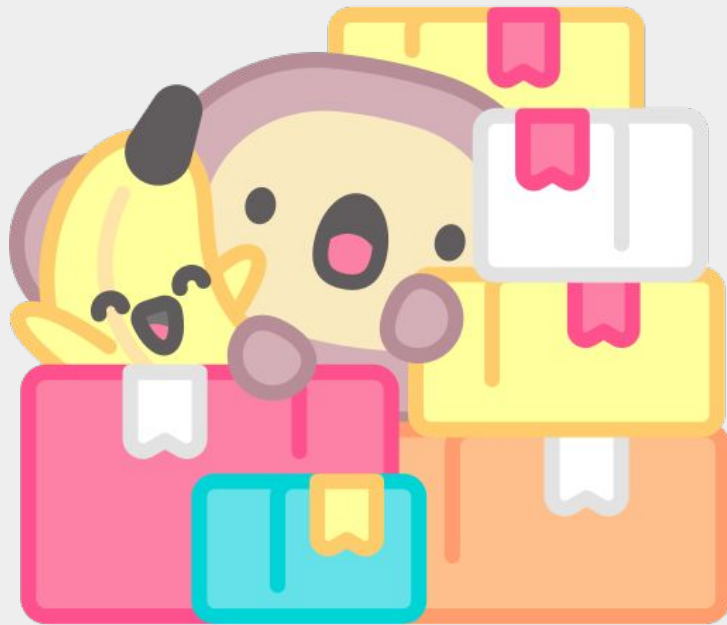
Selected move to achieve current goals, after evaluate all possible options.

Parcels

Handled by the asynchronous function *onParcelsSensing*, triggered every time a parcel is detected.

Register its position (x,y) and reward.

Control to ensure that only parcels within observation distance are perceived.

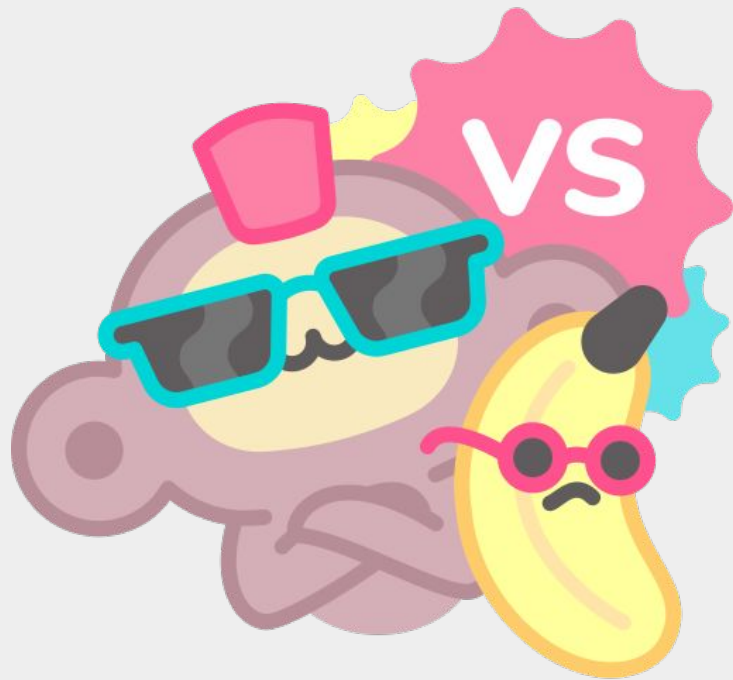


Agents

The asynchronous function *onAgentsSensing* keeps track of other agents' position.

Agents check sensed opponents and avoid conflicting paths before each move. Some collisions still occur due to timing mismatches.

Favor fast movement and accept minor penalties for better overall performance.



Map

The agent maintains an internal map that updates continuously every time new data arrives.

Keep track of every tile and categorize it (wall, delivery or spawn point) to help agent's path finding.

Reconstructs its internal model to ensure an accurate, up-to-date understanding of the environment.

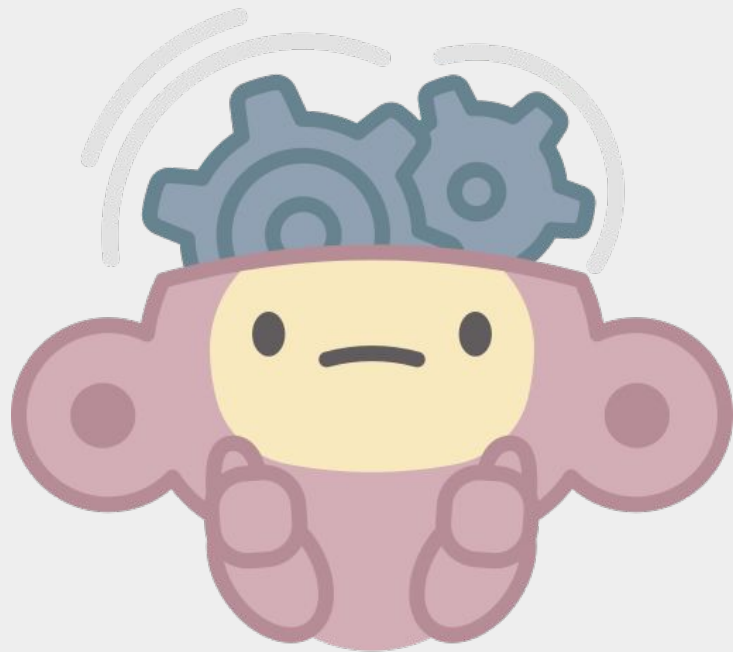


Options

Core of the decision-making process of the agent.

Generate actions (*pick_up*, *go_delivery*) and assign utility to each one.

Pursues the action with the highest utility. If no valid options are available, the agent explores the environment.



Options (more) & Utility

For the two actions (*pick_up*, *go_delivery*) utility is computed with two different formulas.



Parcel Delivery Evaluation

When carrying parcels, the agent computes delivery utility by:

- Considering remaining reward after decay over delivery time
- Reducing utility if a teammate is targeting the same delivery
- Sharing delivery intention to avoid conflict and improve team efficiency



Pick Up Evaluation

Even if the agent is carrying parcels, the evaluation is based on:

- Estimating potential reward
Subtracting time-based decay (pickup + delivery time).
- Ignoring parcels targeted by the teammate (considered discarded)
- Sharing its pickup intention for coordination

Intentions

Intentions

Handled through a dedicated loop implemented in the *IntentionRevision* class. Ensuring dynamically adaptiveness to agent behaviour.

Success

Continuously monitors its intention queue and, upon finding an intention, selects and executes a matching plan from its library to achieve the goal, removing the intention if execution succeeds.

Intention Revision

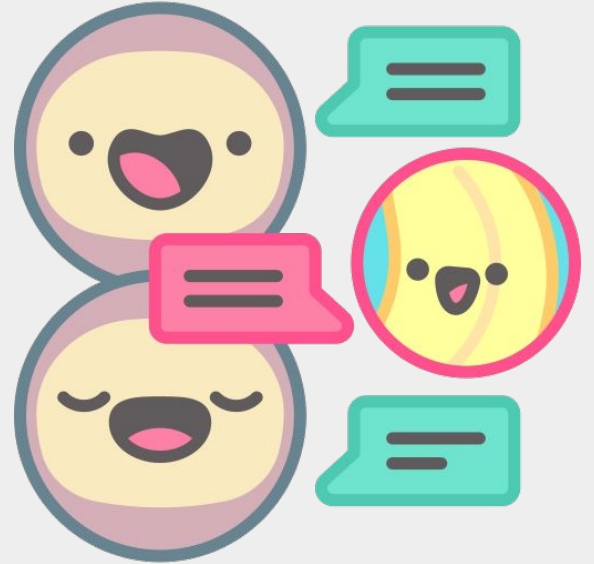
The *IntentionRevisionReplace* subclass enables the agent to replace its current intention with a higher-utility one, allowing it to prioritize more beneficial actions over rigid execution.

Failure

If an intention fails due to a missing or failed plan, it is still removed, the failed option is excluded, blocked if related to a parcel, and the system updates its available options to stay adaptive.



Planning and communication



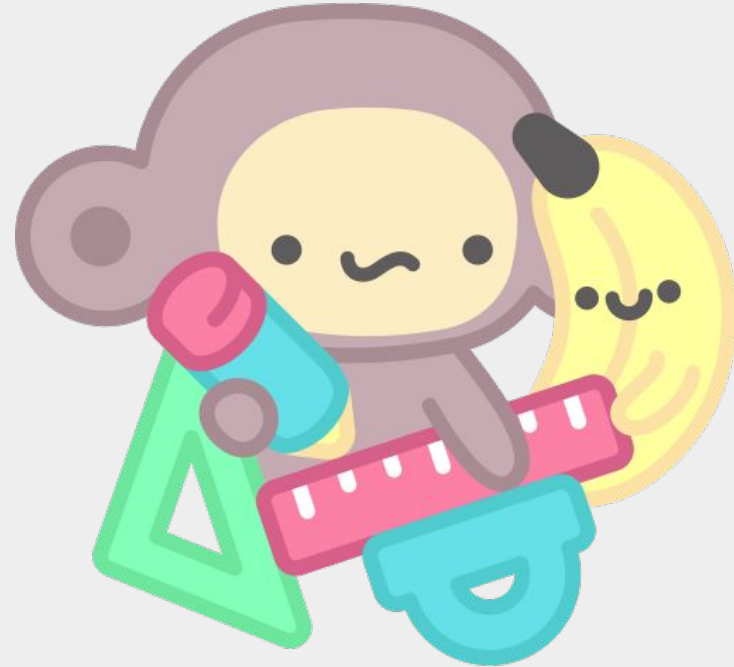
Planning

The planning phase relies on plans linked to predicates that specify how intentions are fulfilled.

Goals are broken down into smaller subgoals through hierarchical planning.

A* algorithm is used to find the shortest path while dynamically avoiding obstacles.

The plan is reactive, replanning if the path is blocked or the agent is interrupted.



Planning: Classes for exploration



Random Move

Most basic movement strategy. The agent selects the next available tile at random, resulting in minimal coverage and limited exploration efficiency.



SmartExplore

Uses a heuristic to guide the agent toward less-visited or promising areas. Improves coverage and balances exploration efficiency by avoiding redundant movement.



ExploreSpawnTiles

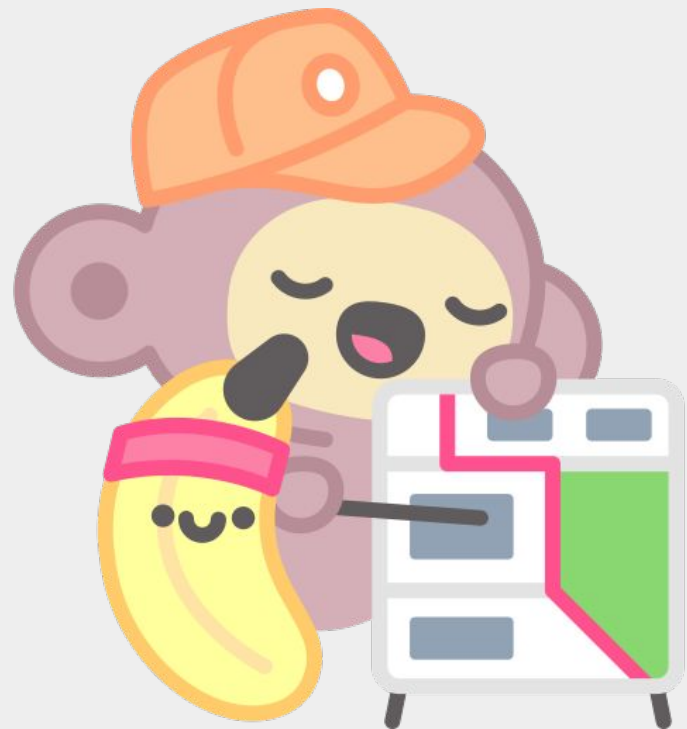
Directs the agent through known spawn locations, ensuring they are regularly checked. It is particularly effective on maps with few and scattered spawn tiles.

PDDL

In single-agent scenario agent uses a PDDL planner to compute plans toward a target. By modeling the world with predicates and actions.

Unfortunately, PDDL is not ideal in multi-agent scenario due to its lack of reactivity in handling fast revision.

We chose to use the PDDL planner only in the single-agent setting,



Communication (in brief)

Agents are aware of the teammate's ID during the startup.

Agents share messages about their beliefs, allowing them to inform the teammate about currently visible parcels and other agents.

Sharing the intentions between agents consent to avoid redundant and useless efforts, optimizing task allocation.



Communication: Messages Details

The primary coordination mechanism revolves around two specific message types:



delivery_intention

When an agent plans a delivery, it sends a message with the target coordinates. The teammate records this and applies a soft penalty to that zone, aiming to avoid overlap while accounting for agent movement over time.



pickup_intentions

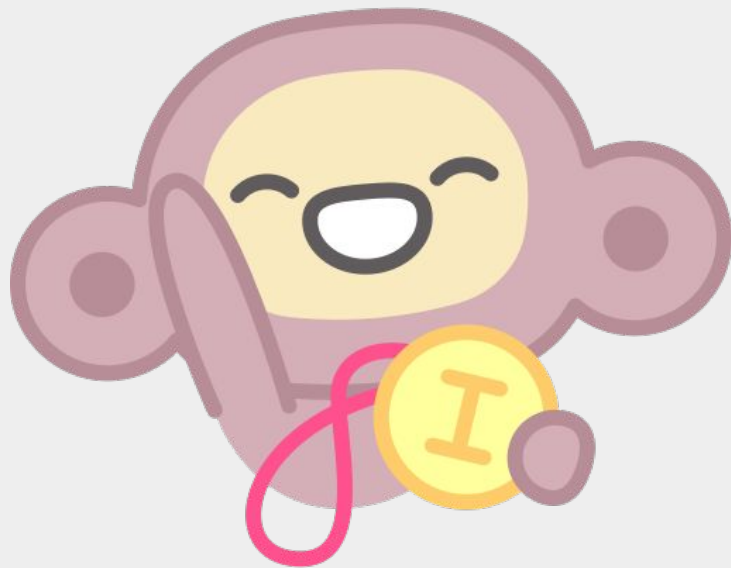
When an agent decides to pick up parcels, it sends a message listing the parcel IDs to its teammate. The teammate updates a local set to track parcels already targeted. This coordination prevents duplicate efforts and enables efficient task distribution.

Results of the challenges


- 🏆 Challenge one: 4th placement
- 🏆 Challenge two: 3rd placement


Consistent scoring across rounds, showing the robustness and adaptability of our approach to agent strategy, coordination, and decision-making.


Encouraging performances confirm that our design decisions are effective and well-suited for competitive, real-world multi-agent environments.




Conclusion

 We design an agent using BDI architecture, enabling to play Deliveroo game both in single and multi-agent version, thanks to cooperation.

 Utility function plays a key role in balancing all the factors that can influence the choice for the best option, especially with two agents.

 The architecture proved flexible and reactive, with dynamic plan selection and exploration behaviors enabling adaptation to changing environments.

 Some simplifications such as basic collision avoidance led to occasional penalties, showing a trade-off between decision speed and control accuracy.

Thanks for your attention

Any Questions?

Avanzolini Elia
Martina Panini

