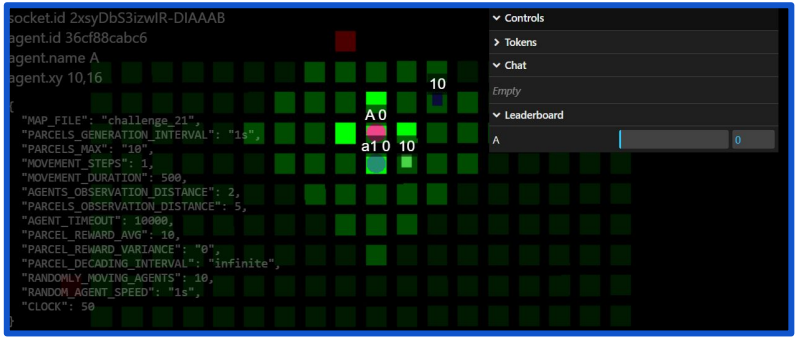
ASA in a Delivery web-based game

Davide Modolo & Davide Moletta

Briefly on the environment



Cell-based world

Blocked, walkable and delivery

Parcels spawn in walkable cells with a score

Agents can move around, pick-up and put-down parcels

THE GOAL

Objective

“An Autonomous Software Agent that plays the Deliveroo.js game on its own”

Agent based on BDI architecture

Agent should maximize the achieved points

Extend the single-agent implementation to multi-agent

Agents are able to share informations to coordinate themselves

Metrics

Pick-up score:

$$\text{agentScore} + \text{parcelReward} - \frac{(\#\text{parcelsToDeliver} + 1) \cdot \text{agentMovementSpeed} \cdot \text{distance}}{\text{parcelDecayingTime}}$$

Approximate the value a parcel will have at the moment of pick-up

Delivery score:

$$\text{agentScore} + \text{deliveryBonus} - \frac{\#\text{parcelsToDeliver} \cdot \text{agentMovementSpeed} \cdot \text{distance}}{\text{parcelDecayingTime}}$$

Bonus (growing with the current score) to reward the agent for delivery

Blind Move

Observation Distance for Parcel Selection:

- Agent avoids selecting points already in its field of view (FOV)
- Agent avoids selecting points too close to the map borders

Point Selection Process:

- Determine possible points based on distance (weighted probability)
- Randomly select a point from the list of possible points
- If the list of possible points is empty, select a random walkable point on the map

Planning

Planning Phase:

- PDDL (Planning Domain Definition Language) used for planning phase
- Agent calls the planner function to build the **problem** (**dynamic** and **static**)

API Integration with Planning Domains:

- Integration with Planning Domains through API calls
- Planning Domains API solves the problem and provides a plan (if available)

PDDL Domain

Actions: `left up down right pickup putdown`

Requirements: `:strips :typing :negative-preconditions :disjunctive-preconditions`

Typings: `a` – other agents, `p` – parcels, `c` – cells, `me` – our agent

Predicates for positions and constraints of agents, cells and parcels

PDDL Problem pt.1

STATIC part: Map Representation

Objects: cells (`c_x_y - c`)

Cells' predicates:

- `neighbourDir:` neighbouring cells in all directions
- `is-delivery:` if it's a delivery point
- `is-blocked:` if it's non-walkable

Default cell: used when no agents/parcels in sight

PDDL Problem pt.2

DYNAMIC part: non-static objects in the world

Objects: **our agent** (`me_id` - `me`), **parcels** (`p_id` - `p`), **other agents** (`a_id` - `a`)

Our agent's predicates:

- `at:` the cell our agent is in
- `holding:` the parcels our agent is carrying

Parcels' predicates:

- `in:` the cell the parcel is in
- `delivered:` if the parcel is delivered

Other agents' predicate:

- `occ:` cells occupied by other agents

Default agent/parcel: used when no agents/parcels in sight

SINGLE AGENT

Single-Agent

Agent A developed for collecting and delivering parcels in the environment

“awareness of the world (parcel and agent locations) and decision-making capabilities”

Action Flow:

1. Sense the environment using provided functions
2. Determines its position, parcel locations, and other agent locations
3. Checks available actions and selects the best one
4. If it’s satisfiable, the agent creates a PDDL plan or selects an existing plan from the *library*
5. It executes the plan
6. During the execution it continually evaluates if the current option is not the best anymore
7. If a better option is identified, it stops the current plan and performs the new intention

Discarded Options

The Agent maintains a **memory** of "discarded" options

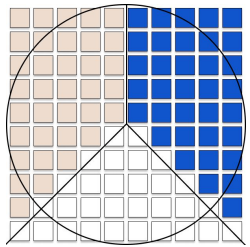
Discarded options are stored for potential future use when they may become optimal

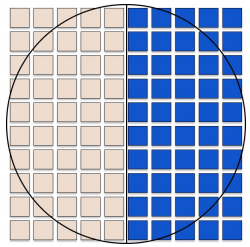
Memory is updated constantly and outdated options are dropped to avoid unachievable intentions

MULTI AGENT

Slicing Function

1. Compute the slice borders
2. For each cell, determine to which slice it belongs
3. Compute ‘problematic’ cells
4. Add those cells (or entire borders) to neighbouring slices





Message Parsing

For multi agent implementation we created a message parsing-based **information sharing**, messages are defined as follows:

```
messageType$information
```

Types of messages are:

1. **i**: agent ids
2. **p**: parcels
3. **a**: other agents
4. **e:** expected score for concurrency
5. **s**: stop the current plan

Multi-Agent 1

Agent B1 & C1 are extensions of **agent A** developed for sharing environment's information

"awareness of the world, decision-making capabilities and belief sharing"

Action Flow (in addition to the ones of Agent A):

1. Each agent computes its own slice using the Pizza slicing algorithm
2. During belief revision, if an agent senses a parcel and/or an agent outside its own slice, it sends a message to the colleague
   a. Parcels messages: `p$p1Id.pX.pY.pCourier.pReward_p2Id…`
   b. Agents messages: `a$a1Id.aX.aY_a2Id…`
3. The colleague listen for messages and upon receipt it extract the information
4. Parcels and agents are then added to the beliefs

Multi-Agent 2

Agent B2 & C2 are extensions of **agent A** developed for sharing mental states

“awareness of the world, decision-making and cooperation with mental states sharing”

Action Flow (in addition to the ones of Agent A):

1. Exchange their ids (`i$...`)
2. When an agent wants to achieve an intention it queries its colleague (`e$...`)
3. The colleague receives the message and checks if it wants to achieve the same intention
4. If so, it calculates its score and replies sending that information to the first agent
5. The first agent then compares the values and determines who won the concurrency
6. If the first agent loses it stops its own intention, otherwise it tells the other to stop (`s$`)

RESULTS

Results

Single agent results

SA	Run 1	Run 2	Run 3	Run 4	Run 5	avg
C21	290	310	260	300	360	304
C22	319	251	288	355	346	311.8
C23	2637	2398	2589	2613	2343	2516
C24	990	1138	855	909	1439	1066.2

Multi agent slicing results

MAS	Run 1	Run 2	Run 3	Run 4	Run 5	avg
C31	497+955	785+746	706+824	820+976	927+903	1627.8
C32	0+0	0+0	0+0	0+0	0+0	0
C33	708+594	545+750	813+997	949+807	527+898	1517.6

Multi agent results

MAM	Run 1	Run 2	Run 3	Run 4	Run 5	avg
C31	830+657	679+677	715+713	643+969	1122+883	1577.6
C32	1131+1247	854+999	361+226	961+442	997+975	1638.6
C33	589+417	691+421	215+712	424+700	332+498	999.8

Single-Agent Challenges

It behaved **as expected** and similarly to the ones seen during the challenge lectures

	Ch 21	Ch 22	Ch 23	Ch 24
Avg	304	312	2516	1066
Class Avg	197	270	1422	261

Multi-Agent Challenges 1

They **failed** in the challenge 32:

- basic-level communication not enough to discriminate the best move
- single move every couple of minutes
- zero points achieved

In the other two challenges they behaved **correctly**:

slicing function very general ► more points with a challenge-based customized function

	Ch 31	Ch 32	Ch 33
Avg	747+880	0+0	708+809

Multi-Agent Challenges 2

In all three challenges they behaved **as expected**:

- Challenge 32: sometimes slow start for `weightedBlindMove`
- Challenge 33: either good or bad runs (random spawn points and max #parcels = 6)

Very general approach ► fine in “classic” maps, not the best for the challenges’ maps

	Ch 31	Ch 32	Ch 33
Avg	798+780	861+778	450+550

CONCLUSIONS

Conclusions

Successfully developed autonomous agents for the Deliveroo.js game

Analyzed strengths and weaknesses of implemented approaches

The project provided insights and learning experiences

Thank you

Davide Modolo & Davide Moletta