UNIVERSITÀ
DI TRENTO

AUTONOMOUS SOFTWARE AGENTS

DELIVEROO

Department of Information Engineering and Computer Science

Jonathan Fin - Matteo Bando

UNIVERSITÀ DI TRENTO

SINGLE AGENT

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

STEPS

5

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

STEPS

5

CLIENT SETUP

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

AGENT INJECTION

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

AGENT INJECTION

EVENT-DRIVEN POPULATION

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

AGENT INJECTION

EVENT-DRIVEN POPULATION

READINESS CHECK

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

AGENT INJECTION

EVENT-DRIVEN POPULATION

READINESS CHECK

CLIENT SETUP

UNIVERSITÀ DI TRENTO

DeliverooClient instantiation —WHAT→ Just a **wrapper** around **DeliverooApi**

CLIENT SETUP

UNIVERSITÀ DI TRENTO

DeliverooClient instantiation

WHAT → Just a **wrapper** around **DeliverooApi**

WHY → To **centralize** connection parameters and **decouple** the code from the external library

CLIENT SETUP

UNIVERSITÀ DI TRENTO

DeliverooClient instantiation

WHAT → Just a **wrapper** around **DeliverooApi**

WHY → To **centralize** connection parameters and **decouple** the code from the external library

AFTER INSTANTIATION → Exposes **methods** to register callback for all incoming **events**:

- **onYou** ⟶ agent identity and initial position
- **onTile** ⟶ map tiles
- **onMap** ⟶ map load
- **onConfig** ⟶ server configuration parameters
- **onParcelsSensing** ⟶ visible parcels
- **onAgentsSensing** ⟶ other visible agents

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

AGENT INJECTION

EVENT-DRIVEN POPULATION

READINESS CHECK

MODEL INSTANTIATION

UNIVERSITÀ DI TRENTO

The agent’s internal knowledge is organized into
5 core **components**

MODEL INSTANTIATION

UNIVERSITÀ DI TRENTO

The agent's internal knowledge is organized into

5 core **components**

→ **Me** holds the agent's identifier, current coordinates, and status

MODEL INSTANTIATION

UNIVERSITÀ DI TRENTO

The agent's internal knowledge is organized into

5 core **components**

- **Me** holds the agent's identifier, current coordinates, and status
- **ParcelsStore** tracks known parcels on the map

MODEL INSTANTIATION

UNIVERSITÀ DI TRENTO

The agent's internal knowledge is organized into

5 core **components**

- **Me** holds the agent's identifier, current coordinates, and status
- **ParcelsStore** tracks known parcels on the map
- **MapStore** maintains the grid layout, tile types, computed distance matrix, and sparsity metrics

MODEL INSTANTIATION

UNIVERSITÀ DI TRENTO

The agent's internal knowledge is organized into

5 core **components**

- **Me** holds the agent's identifier, current coordinates, and status
- **ParcelsStore** tracks known parcels on the map
- **MapStore** maintains the grid layout, tile types, computed distance matrix, and sparsity metrics
- **AgentStore** records sensed positions and timestamps of other agents

MODEL INSTANTIATION

UNIVERSITÀ DI TRENTO

The agent's internal knowledge is organized into

5 core **components**

- **Me** holds the agent's identifier, current coordinates, and status
- **ParcelsStore** tracks known parcels on the map
- **MapStore** maintains the grid layout, tile types, computed distance matrix, and sparsity metrics
- **AgentStore** records sensed positions and timestamps of other agents
- **ServerConfig** caches global simulation parameters (clock interval, map dimensions, sensing radius, carrying capacity)

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

AGENT INJECTION

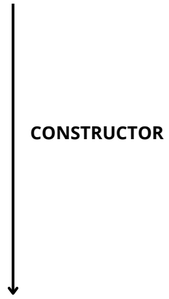
EVENT-DRIVEN POPULATION

READINESS CHECK

AGENT INJECTION

UNIVERSITÀ DI TRENTO

We create an **Agent instance** and wire it to the client and all five stores



main.js

```
const agent = new Agent(client, me, parcels, mapStore, agentStore, serverConfig);
```

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

AGENT INJECTION

EVENT-DRIVEN POPULATION

READINESS CHECK

EVENT-DRIVEN POPULATION

UNIVERSITÀ DI TRENTO

onYou → Sets the agent's unique ID and coordinates

onTile → Build the map in MapStore

onMap → Build the map in MapStore

onConfig → Populates ServerConfig with the parameters

onParcelsSensing → Refresh dynamic information about parcels and other agents

onAgentsSensing → Refresh dynamic information about parcels and other agents

INITIAL CONFIGURATION

UNIVERSITÀ DI TRENTO

5 STEPS

CLIENT SETUP

MODEL INSTANTIATION

AGENT INJECTION

EVENT-DRIVEN POPULATION

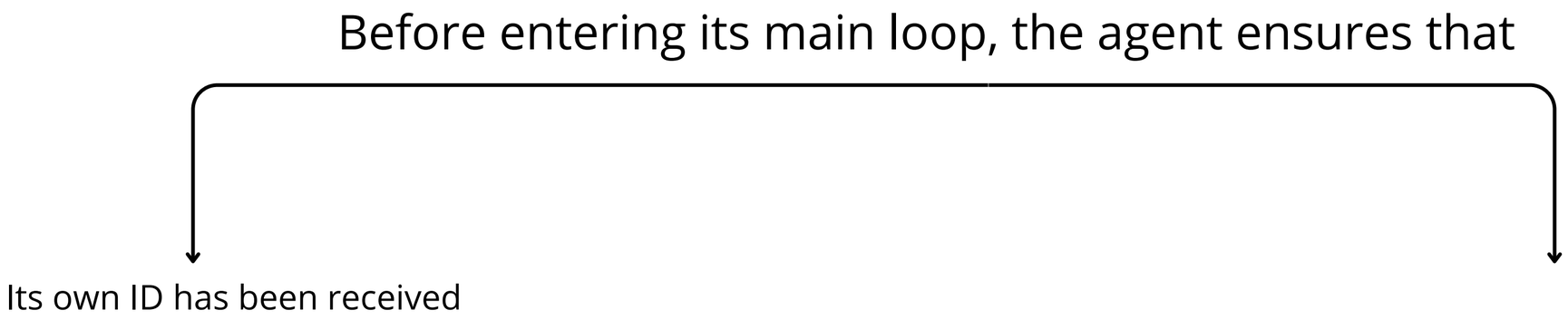
READINESS CHECK

READINESS CHECK

UNIVERSITÀ DI TRENTO

Before entering its main loop, the agent ensures that

READINESS CHECK



READINESS CHECK

UNIVERSITÀ DI TRENTO

Before entering its main loop, the agent ensures that

- Its own ID has been received
- The map has been fully loaded

READINESS CHECK

UNIVERSITÀ DI TRENTO

Before entering its main loop, the agent ensures that

- Its own ID has been received
- The map has been fully loaded

main.js

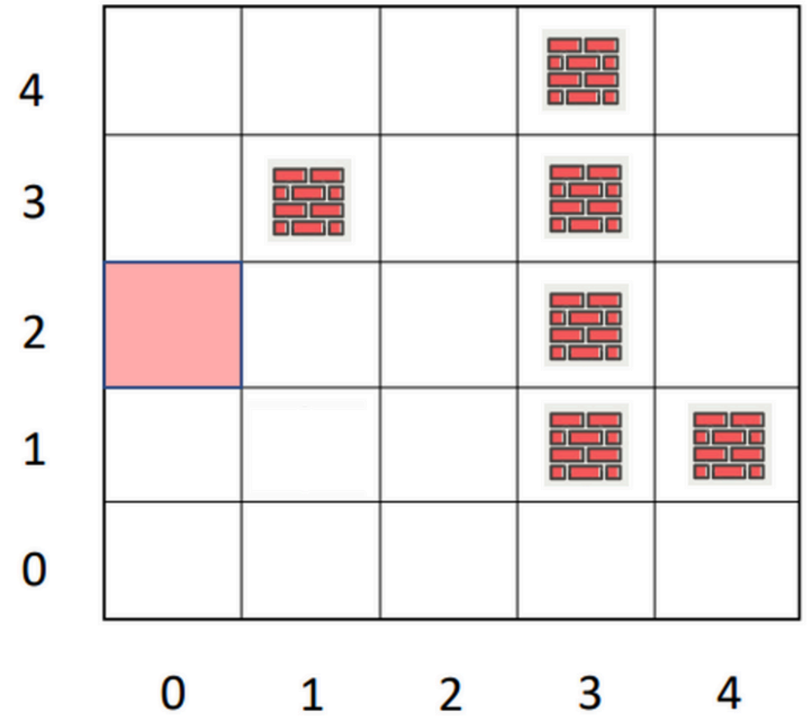
```
agent.updateBeliefs();

agent.generateDesires();

agent.filterIntentions();

agent.act();
```

MAP MEMORIZATION

UNIVERSITÀ DI TRENTO

After onMap finished receiving updates → run **Floyd-Warshall** algorithm

→ stores in memory a matrix which contains the **distances between all pairs of tiles**



PARCELS SENSING

UNIVERSITÀ DI TRENTO

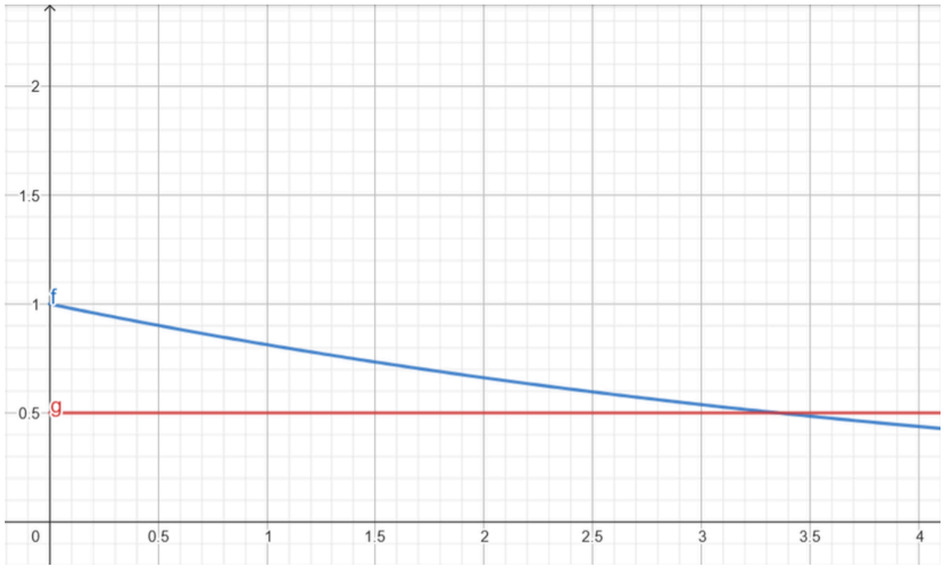
Sensed parcels are added to the memory or updated
→ When added, also the **distance from the nearest base** is calculated

PARCELS SENSING

UNIVERSITÀ DI TRENTO

Sensed parcels are added to the memory or updated
→ When added, also the **distance from the nearest base** is calculated

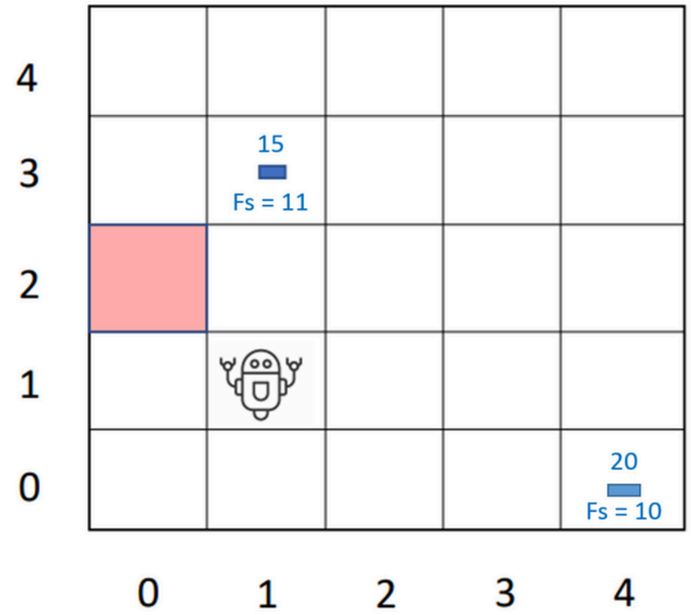
Non-sensed parcels are updated based on time passed
→ The **reward** is **decreased**
→ The **existing probability** is calculated based also on the **number of agents** present in the game



PICKUP DESIRE

UNIVERSITÀ DI TRENTO

The agent calculates a possible **score for each parcel**, using the *parcel reward* and subtracting the *points* that will be lost in the meantime



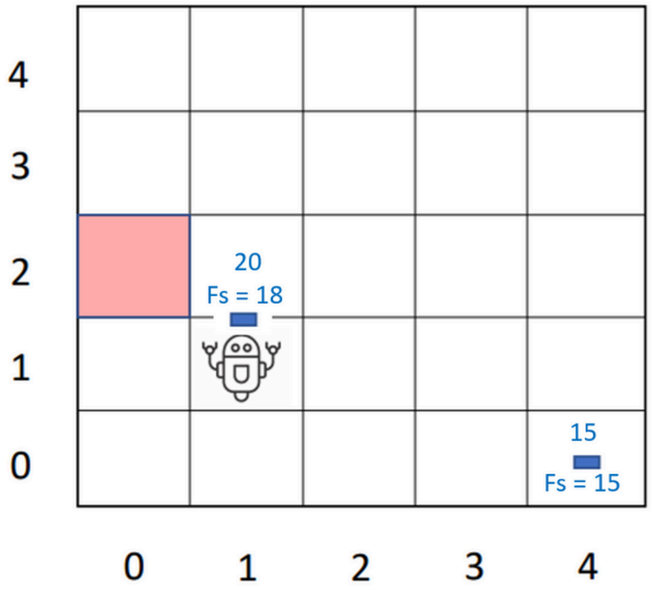
misc.js

```
const distanceToParcel = mapStore.distance(startPos, goalPos);

    // Total reward = sum of all carried parcels + this parcel's reward (or group of parcels)
    const totalReward = carriedValue + reward;

    const totalDistance = distanceToParcel + baseDistance;

    const totalParcels = carriedCount + pickupCount;

    // Potential reward: total reward - temporal cost to bring the parcel to the base
    return totalReward - (totalDistance * totalParcels * clockPenalty / configuration.parcels_decaying_interval);
```

DEPOSIT DESIRE

UNIVERSITÀ DI TRENTO

If the agent is carrying at least 1 parcel → it calculates a possible **score for depositing the parcels**



- **18 points** if the agent goes to deposit immediately
- **15 points** if the agent goes to pick up the parcel and the deposit (each parcel will lose 10 points from the original score)

EXPLORE DESIRE

UNIVERSITÀ DI TRENTO

The agent always has a desire to explore, with very **low score**

If **no other desire is possible** → the agent explores the map

INTENTION SELECTION

UNIVERSITÀ DI TRENTO

The agent **sorts** the desires **by score**, and then **picks** the one with **the highest** score

If the agent sees another agent (from another team), it performs the following checks:

misc.js

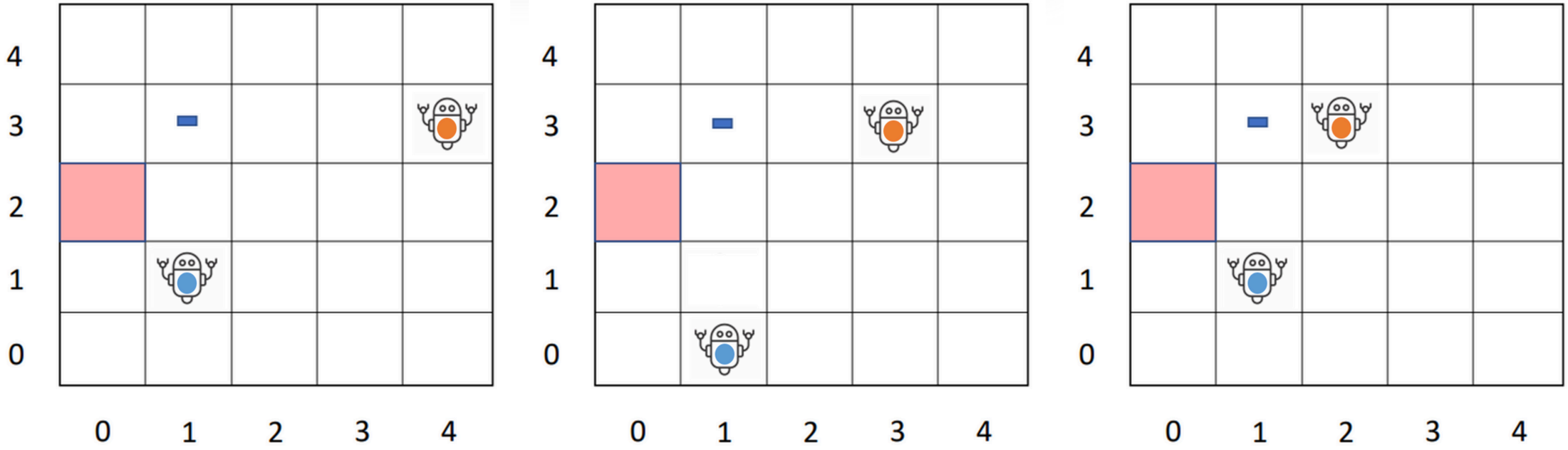
```
if a senses another agent:
    if agent is closer than a:
        if agent is 1 tile away from parcel OR going towards the parcel:
            discard parcel and pick next intention
```

INTENTION SELECTION

UNIVERSITÀ DI TRENTO

misc.js

```
if a senses another agent:
    if agent is closer than a:
        if agent is 1 tile away from parcel OR going towards the parcel:
            discard parcel and pick next intention
```



ACTIONS

UNIVERSITÀ DI TRENTO

All actions are preceded by a planning phase, calculating the path to the destination with the **A* algorithm**

Explore → the agent selects a **random**, reachable **spawn tile** as its destination

If the agent doesn't find anything in its walk, it performs another exploration

If the map is **sparse**, the agent waits a few seconds before picking another spawn tile

If the agent finds **another agent blocking its path**, it **waits** a predetermined time, to see if the other agent will move, and then, if the agent is still there, it temporarly removes that tile from the map and **recalculates the path** to the destination

The time **can also be 0**, which is seen to fit the best in a many-agents environment

UNIVERSITÀ DI TRENTO

MULTI AGENT

LAUNCH

UNIVERSITÀ DI TRENTO

The 2 agents are launched by the same script, passing to them the same pointer for the main memory

The memory is **shared**, so each agent can insert and read from the same location, without the need for message exchange

mainMulti.js

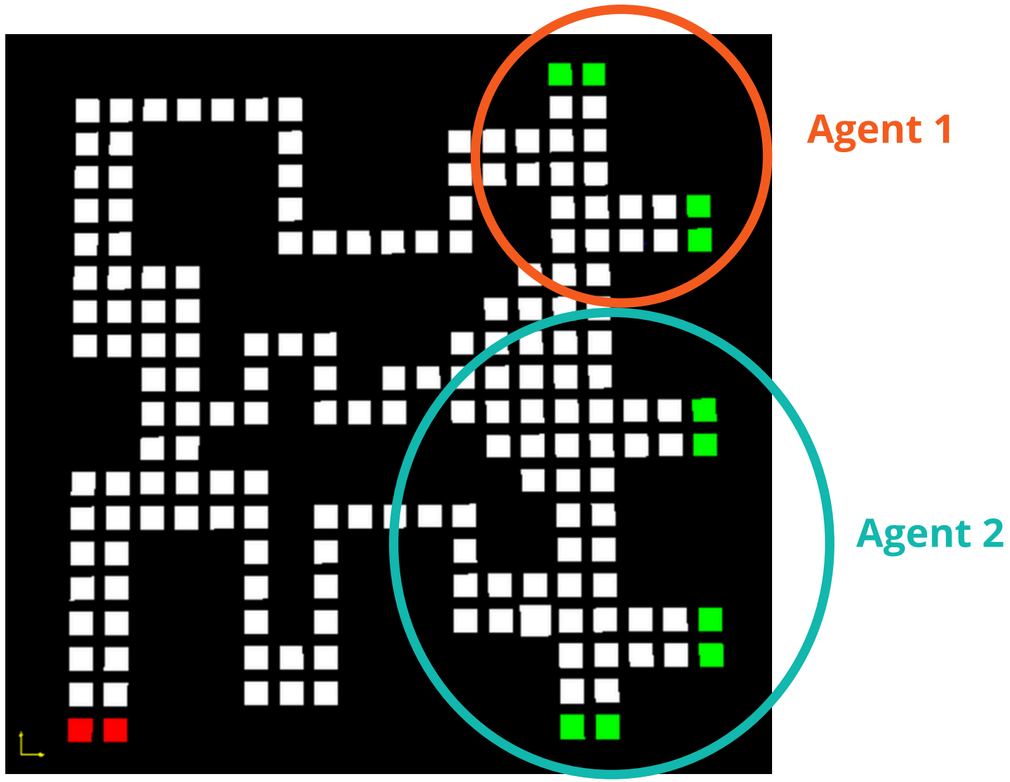
```
const agent1 = new MultiAgent(client1, me1, me2, parcels, mapStore, agentStore,communication, serverConfig, true);
const agent2 = new MultiAgent(client2, me2, me1, parcels, mapStore, agentStore, communication, serverConfig, false);
```

MAP DIVISION

UNIVERSITÀ DI TRENTO

The **spawn tiles** are **split** into 2 groups, one for each agent

→ Duration the **exploration**, the agent picks one of the tiles assigned to itself

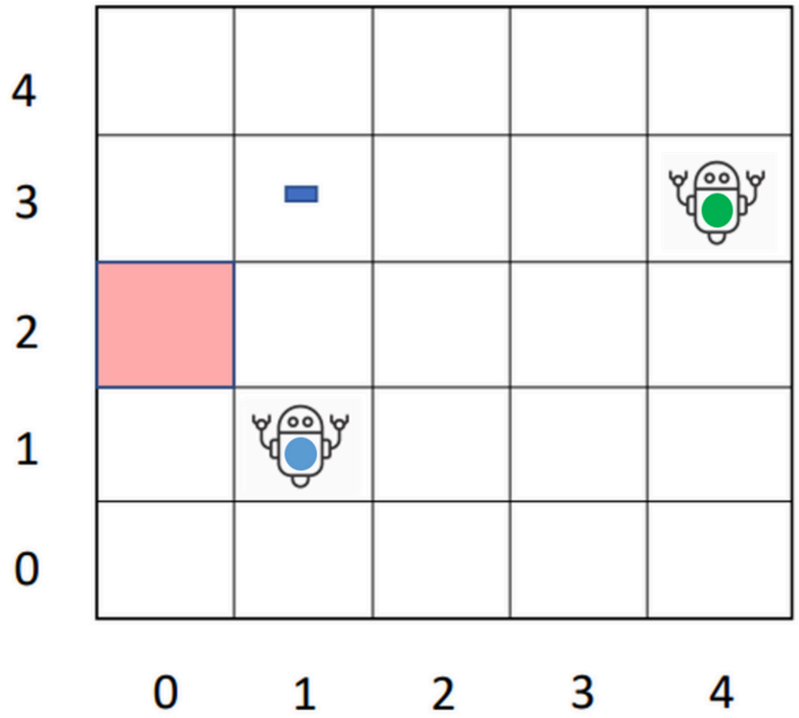


COORDINATION - PARCEL PICKUP

UNIVERSITÀ DI TRENTO

Each agent calculates the score for itself and for its teammate

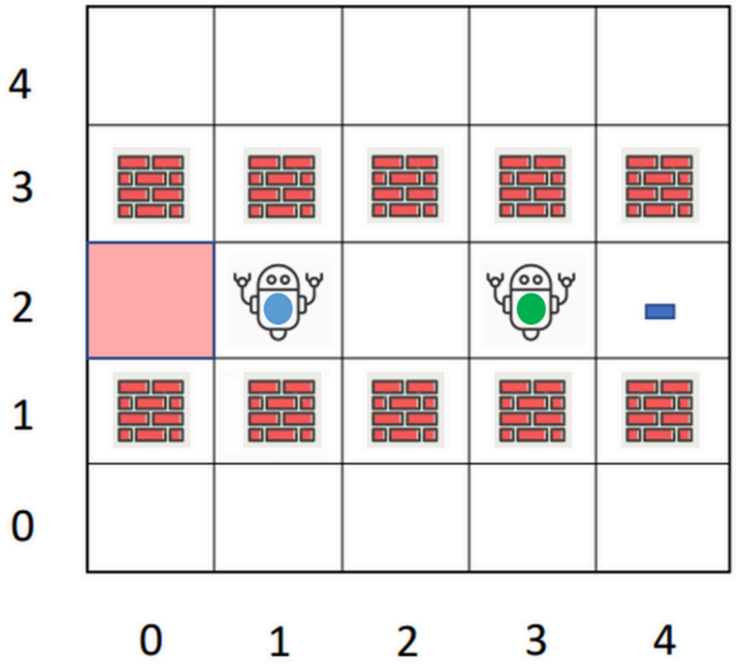
→ A desire is generated only if its own **score is higher** than it teammate's

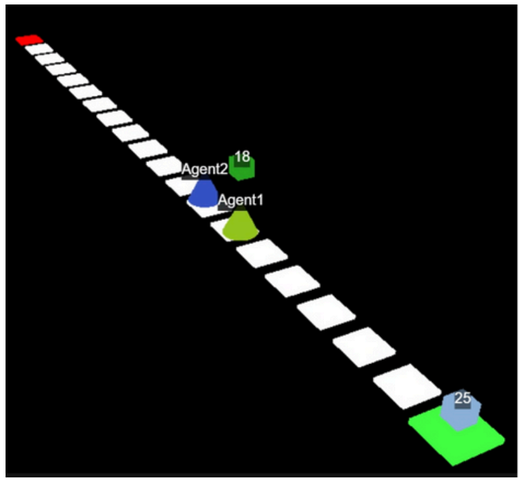


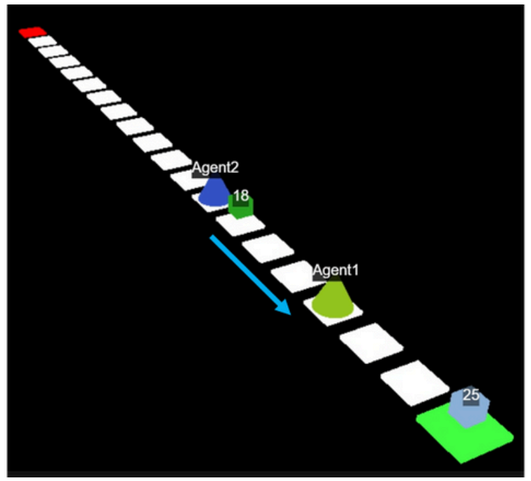
COORDINATION - PARCEL EXCHANGE

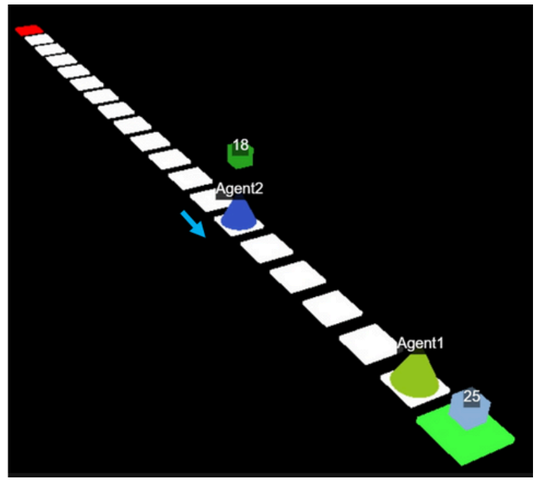
UNIVERSITÀ DI TRENTO

If an agent is carrying some parcels and detects its teammate in a neighboring tile
→ it **deposits** the parcels and **moves away**, **telling the other agent** that he deposited them









UNIVERSITÀ DI TRENTO

PDDL

DOMAIN FILE

UNIVERSITÀ DI TRENTO

domain.pddl

```
(define (domain deliveroo)
  (:requirements :strips :typing)
  (:types
    agent
    parcel
    tile
    base
  )

  (:predicates
    ;; agent ?a is on tile ?t
    (at ?a - agent ?t - tile)

    ;; parcel ?p is on tile ?t (not yet picked up)
    (parcel-at ?p - parcel ?t - tile)

    ;; agent ?a is carrying parcel ?p
    (carrying ?a - agent ?p - parcel)

    ;; base ?b is located on tile ?t
    (base-at ?b - base ?t - tile)

    ;; parcel ?p has been delivered
    (delivered ?p - parcel)

    ;; adjacency relation between tiles
    (adjacent ?t1 - tile ?t2 - tile)
  )
```

It **describes** the immutable physics and rules of the Deliveroo world

---

PREDICATES

- **(at ?a ?t)** - agent a is located on tile t.
- **(parcel-at ?p ?t)** - parcel p is lying on tile t.
- **(carrying ?a ?p)** - agent a is currently hold ing parcel p.
- **(base-at ?b ?t)** - base b occupies tile t.
- **(delivered ?p)** - parcel p has been delivered and it is no longer in play.
- **(adjacent ?t1 ?t2)** - tiles t1 and t2 share an edge (undirected)

DOMAIN FILE

UNIVERSITÀ DI TRENTO

domain.pddl

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

  (:action pickup
    :parameters (?a - agent ?p - parcel ?t - tile)
    :precondition (and
      (at ?a ?t)
      (parcel-at ?p ?t)
    )
    :effect (and
      (not (parcel-at ?p ?t))
      (carrying ?a ?p)
    )
  )

  (:action deposit
    :parameters (?a - agent ?p - parcel ?b - base ?t - tile)
    :precondition (and
      (at ?a ?t)
      (base-at ?b ?t)
      (carrying ?a ?p)
    )
    :effect (and
      (not (carrying ?a ?p))
      (delivered ?p)
    )
  )
)
```

It **describes** the immutable physics and rules of the Deliveroo world

---

ACTIONS

- **move(?a, ?from, ?to)** - The agent leaves the source tile and appears on the destination tile
- **pickup(?a, ?p, ?t)** - The parcel is removed from the ground and held by the agent
- **deposit(?a, ?p, ?b, ?t)** - The parcel is dropped and marked as delivered

PROBLEM FILE

UNIVERSITÀ DI TRENTO

problem.pddl

```
(define (problem deliveroo_problem)

  (:domain deliveroo)

  (:objects
    agent_0d4ea4                          - agent
    p22100 p22102 p22104                  - parcel
    base_1_0 base_1_9  base_9_9           - base
    t_8_2 [ ... ] t_9_6 t_9_9             - tile
  )

  (:init
    [ ... ]
    (adjacent t_9_9 t_8_9)
    [ ... ]
  )

  (:goal (and
    (delivered p22100)
    [ ... ]
  ))
)
```

It encapsulates one concrete **snapshot** of the game world that the planner must reason about

---

(:objects) declares every logical object that currently exist

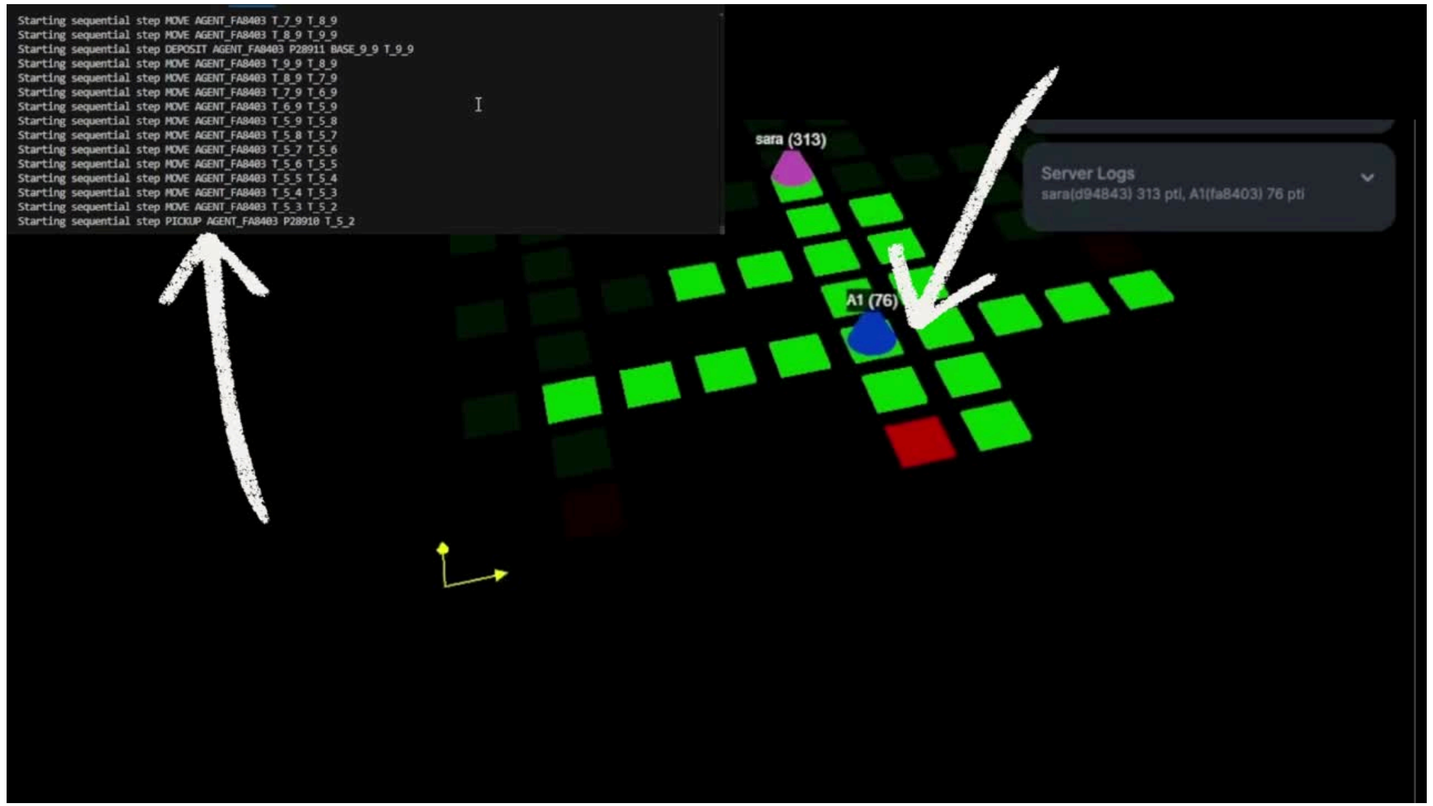
(:init) lists the ground facts that are true right now

(:goal) a conjunctive goal that requires every currently known parcel to be delivered

LIMITATIONS

UNIVERSITÀ DI TRENTO

Relying on an external PDDL solver brings two main challenges: **network and solver latency**, and **world drift** caused by the evolving game state



UNIVERSITÀ
DI TRENTO

DOCUMENTATION

JSDOC + GITHUB PAGES

UNIVERSITÀ DI TRENTO

JSDoc

movement.js

```
/**
 * Moves the agent to the nearest base using smart movement.
 * @param {DeliverooClient} client - The Deliveroo client instance.
 * @param {Me} me - The current player instance.
 * @param {MapStore} mapStore - The MapStore instance containing the current map state.
 * @returns {Promise<void>} - A promise that resolves when the movement is complete.
 * @description
 * This function finds the nearest base on the map and moves the agent towards it
 * using the `smartMove` function. If a base is found, it will navigate to that base
 * efficiently, taking into account the current map state and obstacles.
 */
export async function smartMoveToNearestBase(client, me, mapStore) {
  const [base] = mapStore.nearestBase(me);
  if (base) {
    await smartMove(client, me, base, mapStore);
  }

}
```

GitHub Pages

`smartMoveToNearestBase(client, me, mapStore)` → {Promise.<void>}

This function finds the nearest base on the map and moves the agent towards it using the `smartMove` function. If a base is found, it will navigate to that base efficiently, taking into account the current map state and obstacles.

Parameters:

Name	Type	Description
client	DeliverooClient	The Deliveroo client instance.
me	Me	The current player instance.
mapStore	MapStore	The MapStore instance containing the current map state.

Source src/actions/movement.js,line 69

Returns:

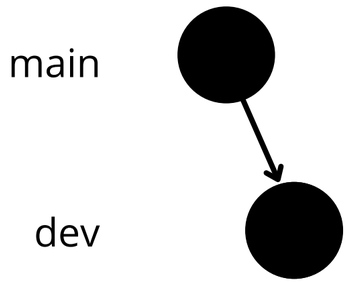
- A promise that resolves when the movement is complete.

Type: Promise.<void>

UNIVERSITÀ
DI TRENTO

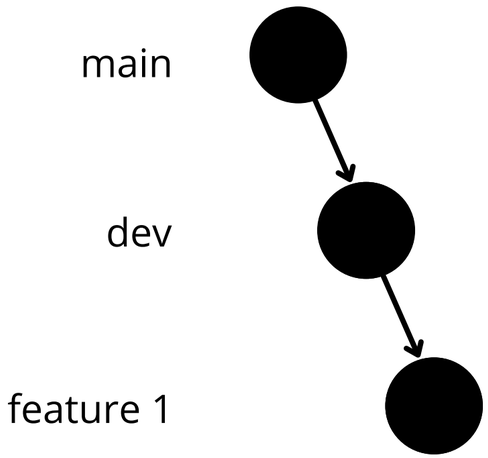
VC & PM

VERSION CONTROL



We used **Git** as VC and **GitHub** to host the project

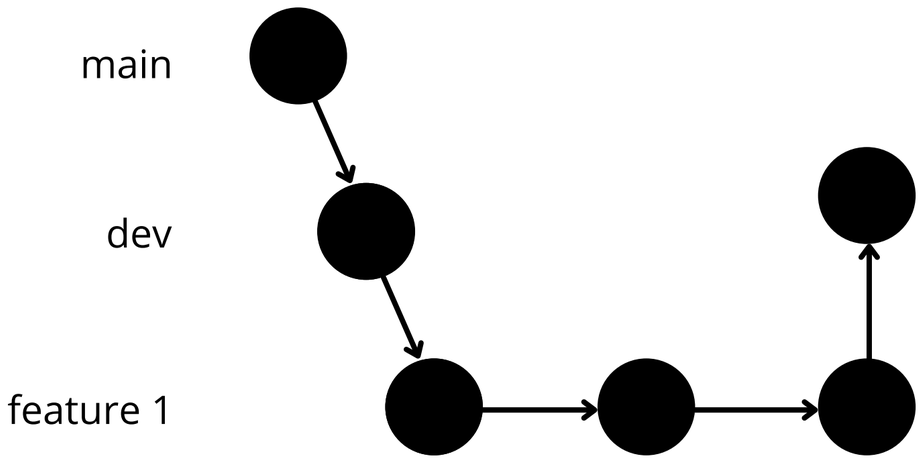
VERSION CONTROL



We used **Git** as VC and **GitHub** to host the project

Whenever we needed to add a new feature, we created a new branch from **dev** and then merged it back

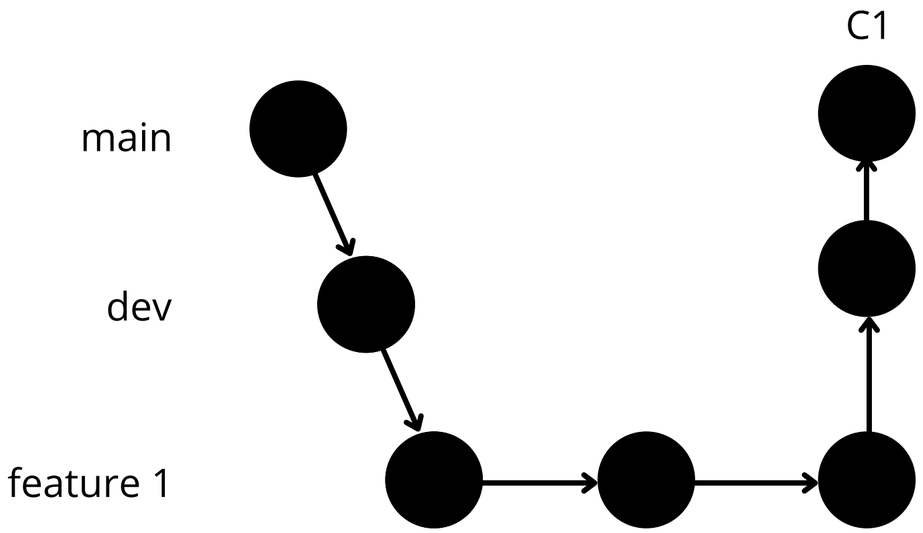
VERSION CONTROL



We used **Git** as VC and **GitHub** to host the project

Whenever we needed to add a new feature, we created a new branch from **dev** and then merged it back

GITHUB



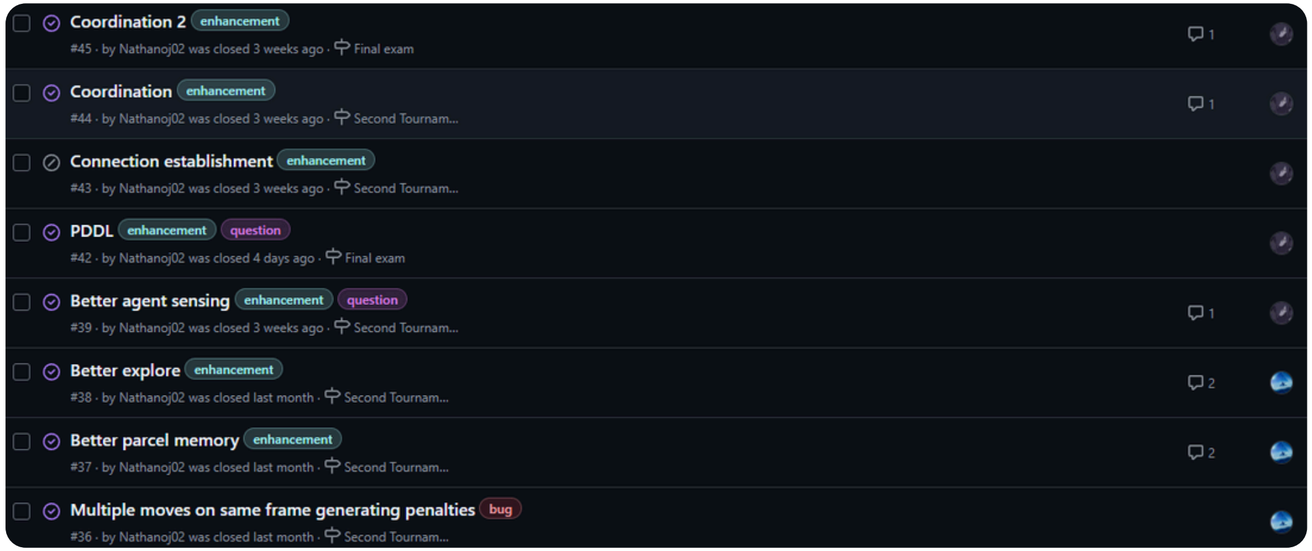
We used **Git** as VC and **GitHub** to host the project

Whenever we needed to add a new feature, we created a new branch from **dev** and then merged it back

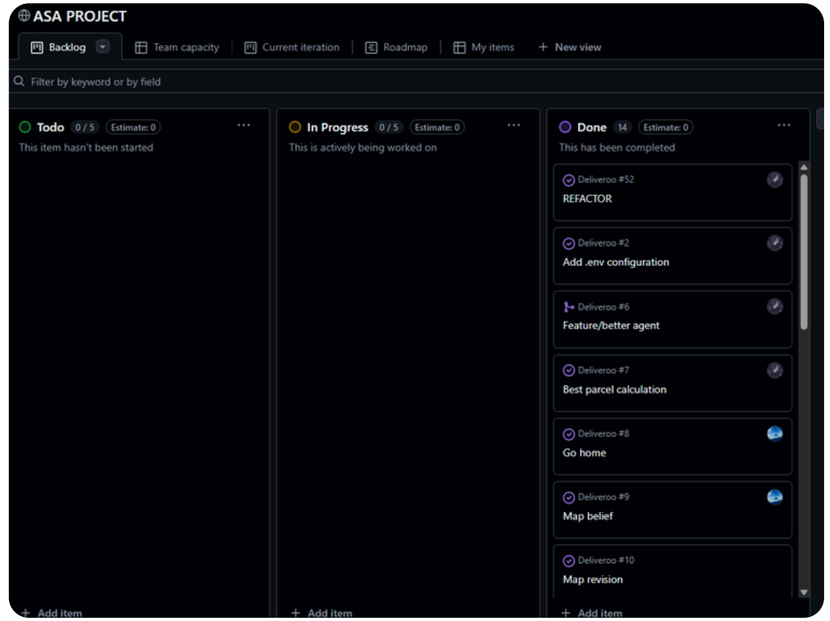
When the product was ready (for instance, **Challenge 1**), we merged everything into the **main** branch

PROJECT MANAGEMENT

Issues



Kanban Board



Thanks for your attention!