UNIVERSITY OF TRENTO
Department of Information Engineering and Computer Science

Autonomous Software Agents

Project Presentation

Bonetto Stefano
217179
stefano.bonetto@studenti.unitn.it

Roman Simone
217181
simone.roman@studenti.unitn.it

Objective

Goal: create an autonomous software agent(s) designed to play the Deliveroo.js game.

Our program is structured into two main workflows:

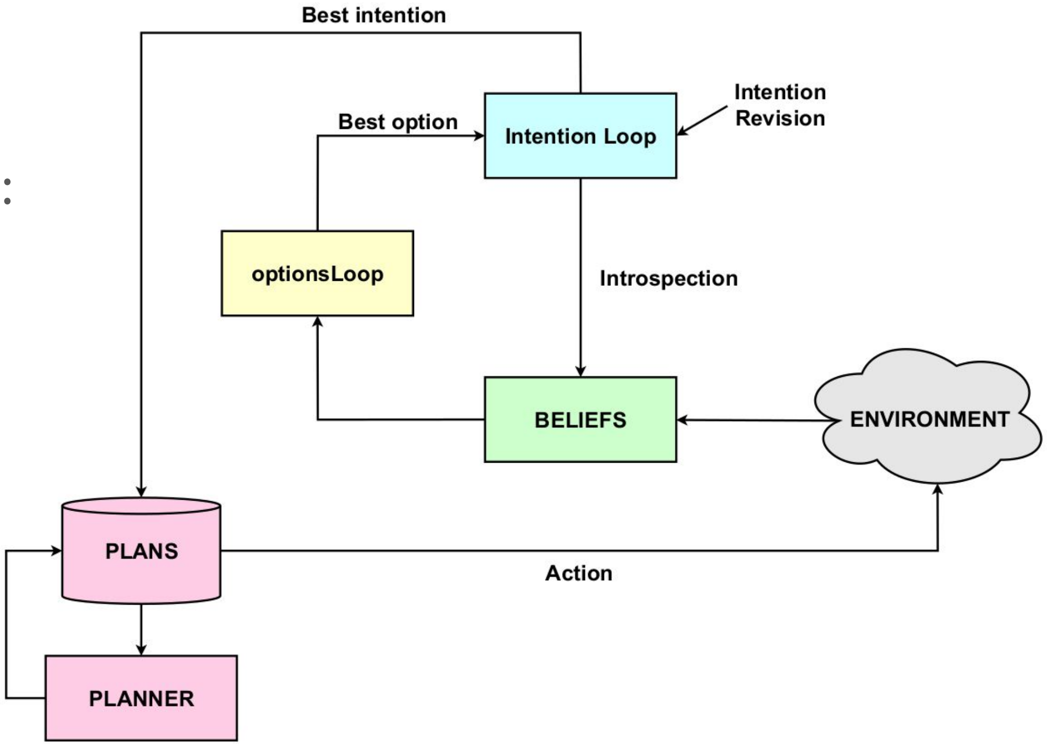
- **Single-agent**: one agent in the environment.
- **Multi-agent**: two collaborative agents cooperate in the same environment.

To do both of them, we’ve build a **BDI** architecture.

Belief-Desire-Intention

There are **4 main steps/entities**:

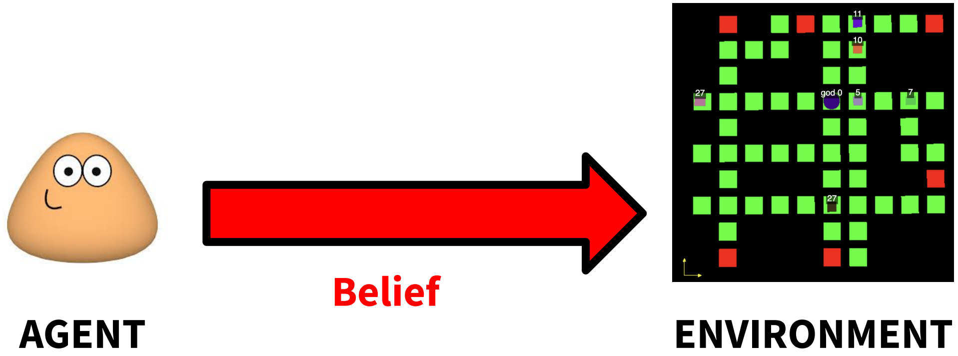
- **Belief**
- **OptionLoop**
- **IntentionLoop**
- **Plans**



Beliefs

Agent perceive different entities from the environment and build its belief based on:

- **Configuration parameters**
- **Map**
- **Parcels**
- **Agents**



Map & Configuration

At the very beginning of the game the agent receive:

- **Configuration of the level**: information and parameters about the level.
- **Map**: the environment via JSON file, we convert it into a more readable matrix.

```
{
  "MAP_FILE": "default_map",
  "PARCELS_GENERATION_INTERVAL": "2s",
  "PARCELS_MAX": "5",
  "MOVEMENT_STEPS": 1,
  "MOVEMENT_DURATION": 500,
  "AGENTS_OBSERVATION_DISTANCE": "infinite",
  "PARCELS_OBSERVATION_DISTANCE": "infinite",
  "AGENT_TIMEOUT": 10000,
  "PARCEL_REWARD_AVG": 30,
  "PARCEL_REWARD_VARIANCE": 10,
  "PARCEL_DECADING_INTERVAL": "1s",
  "RANDOMLY_MOVING_AGENTS": 0,
  "RANDOM_AGENT_SPEED": "2s",
  "CLOCK": 50,
  "BROADCAST_LOGS": false
}
```

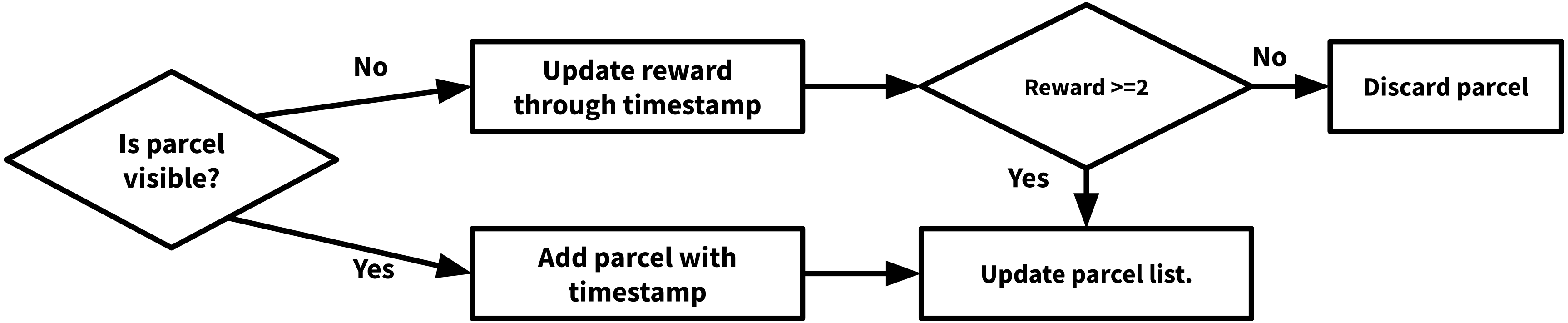
```
{ x: 0, y: 2, delivery: false, parcelSpawner: true },
{ x: 0, y: 4, delivery: false, parcelSpawner: true },
{ x: 0, y: 6, delivery: false, parcelSpawner: true },
{ x: 1, y: 0, delivery: true, parcelSpawner: false },
{ x: 1, y: 1, delivery: false, parcelSpawner: true },
{ x: 1, y: 2, delivery: false, parcelSpawner: true },
{ x: 1, y: 3, delivery: false, parclSpawner: true },
{ x: 1, y: 4, delivery: false, parcelSpawner: true },
{ x: 1, y: 5, delivery: false, parcelSpawner: true },
{ x: 1, y: 6, delivery: false, parcelSpawner: true },
...
```

0	2	0	3	2	3	3	3	3	2
0	3	3	3	0	3	3	0	0	0
0	3	0	0	0	3	3	0	0	0
3	3	3	3	3	3	3	3	3	3
0	3	0	0	0	3	3	0	3	0
3	3	3	3	3	3	3	0	3	3
0	3	0	0	0	3	3	0	0	2
3	3	3	3	3	3	3	3	3	3
0	3	0	0	0	3	3	0	0	0
0	2	0	0	0	2	3	0	0	0

Parcels’ sensing

The **main idea** is to have updated **list of parcel** sensed during gameplay:

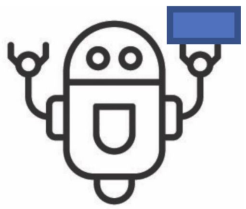
- **previously seen**: update reward using timestamp
- **seen**: add/update with updated info **Reward >= 2**

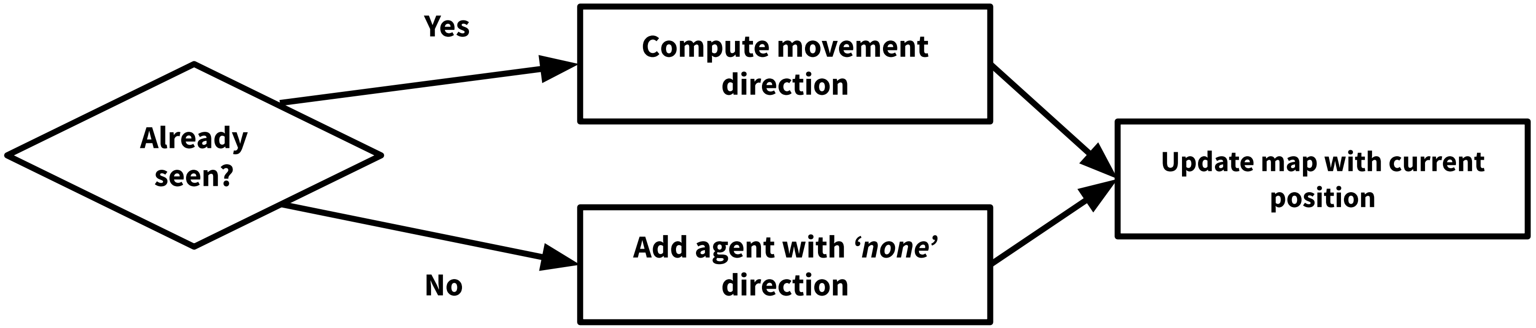


Agents’ sensing

The **main idea** is to register in a list **agents sensed** during gameplay:

- **Already seen**: update
- **New**: add

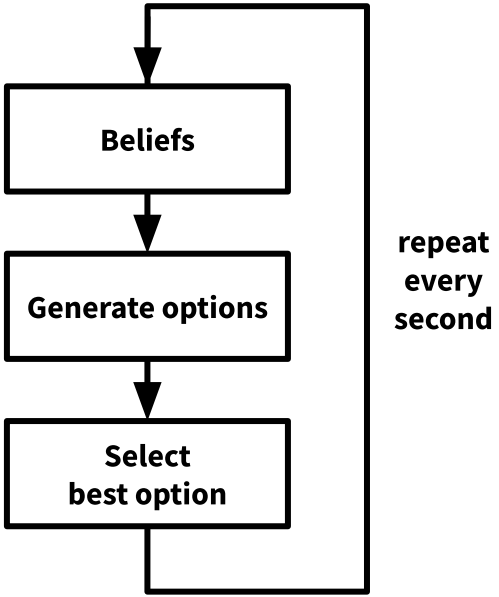




optionsLoop

In the **optionLoop**, we do two things:

1. Generate various **options** based on the belief set (`go_pick_up`, `go_put_down` and `go_random`)
2. Choose `best_option`

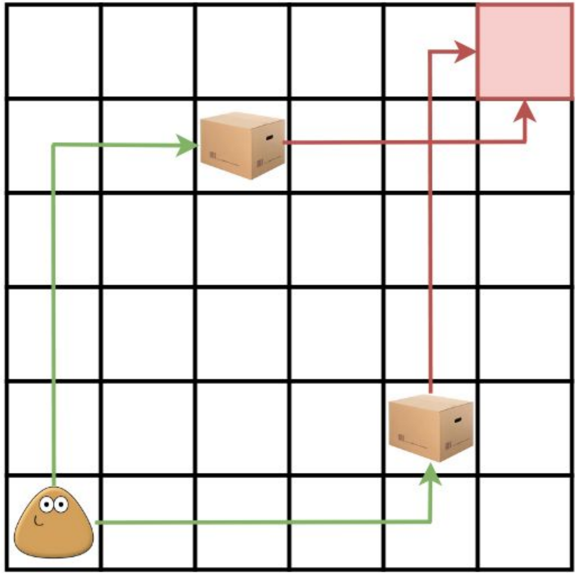


go_pick_up

For each sensed parcel, we calculate the **utility value** computing these predicted values:

1. **reward in mind** when we’ll reach parcel
2. **total reward** when we’ll pick up parcel
3. **final reward** when we’ll reach delivery

To update reward we use the `decade_frequency`.

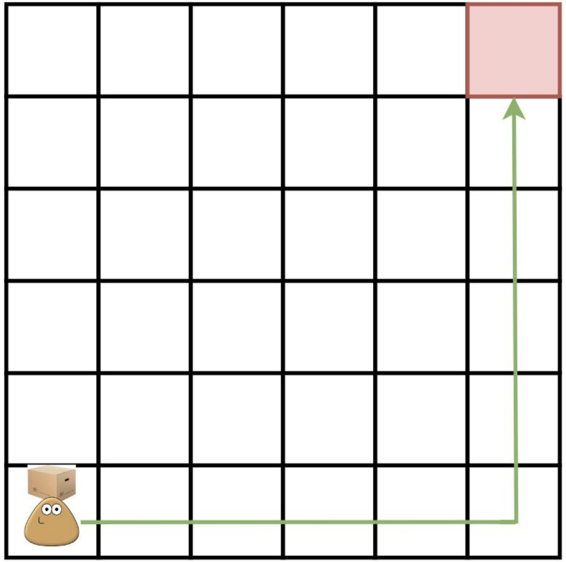


`go_put_down`

Similarly as for `go_pick_up` utility we calculate an **utility** based on reward obtained at end.

The behaviour depends on a level parameter value (`parcel_decading_interval`):

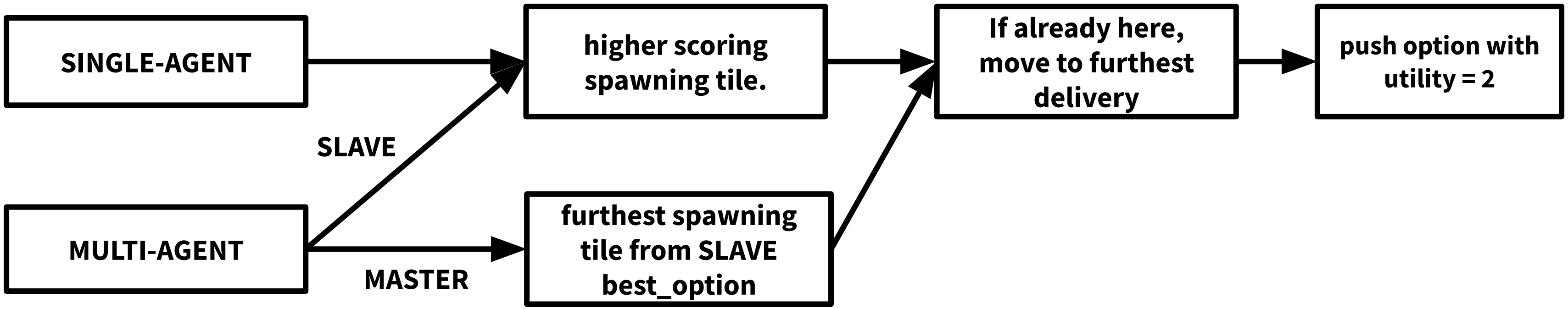
- **if is infinite** we obligate agent to deliver periodically when we exceeds a threshold.
- **otherwise** we normally calculate utility.



`go_random`

We have two different behavior:

- **Single Agent**: we direct the agent to high-scoring spawn zones
- **Multi Agent**: we prioritize **exploration** (agents distant from each other)
    - **SLAVE** as in Single Agent case, while the
    - **MASTER** go in furthest spawning tile from the **SLAVE**'s **best_option** coordinates.

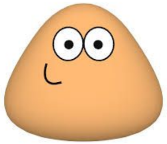


Best option

The choose of best option change in different **scenario**:

Single Agent

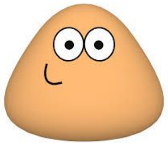
Simply choose option with highest utility

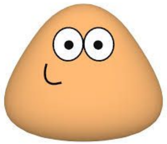


Multi Agent

`MASTER` choose best option **for both** agent, in three step:

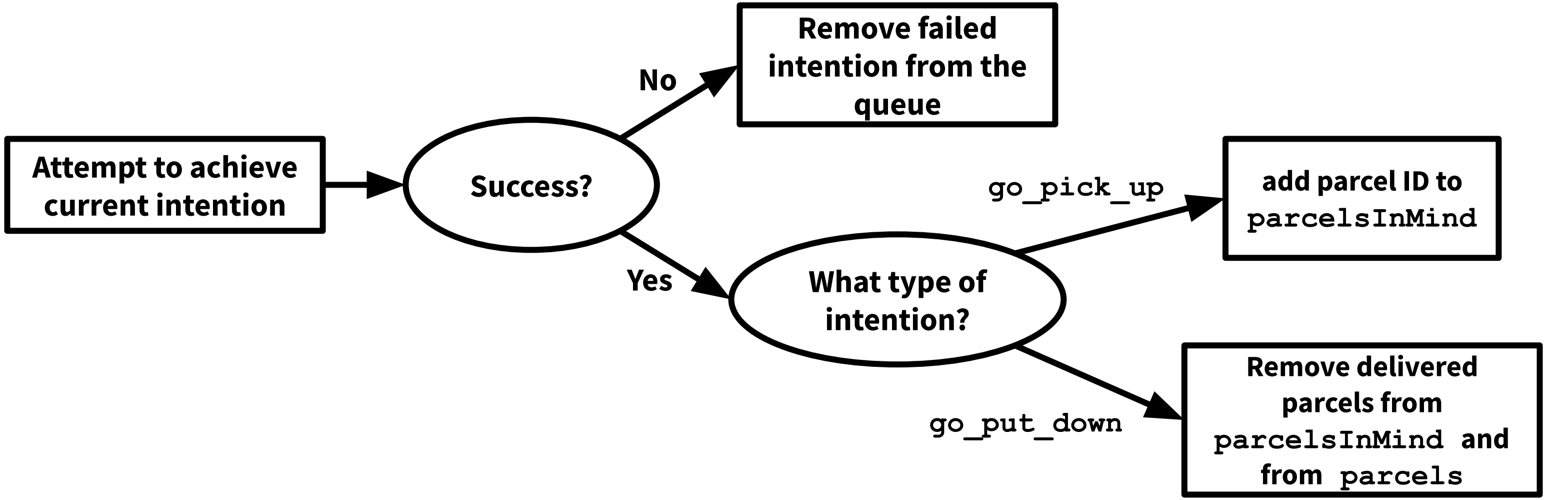
1. Unification of options
2. Best option selection
3. Conflict resolution





Intention loop

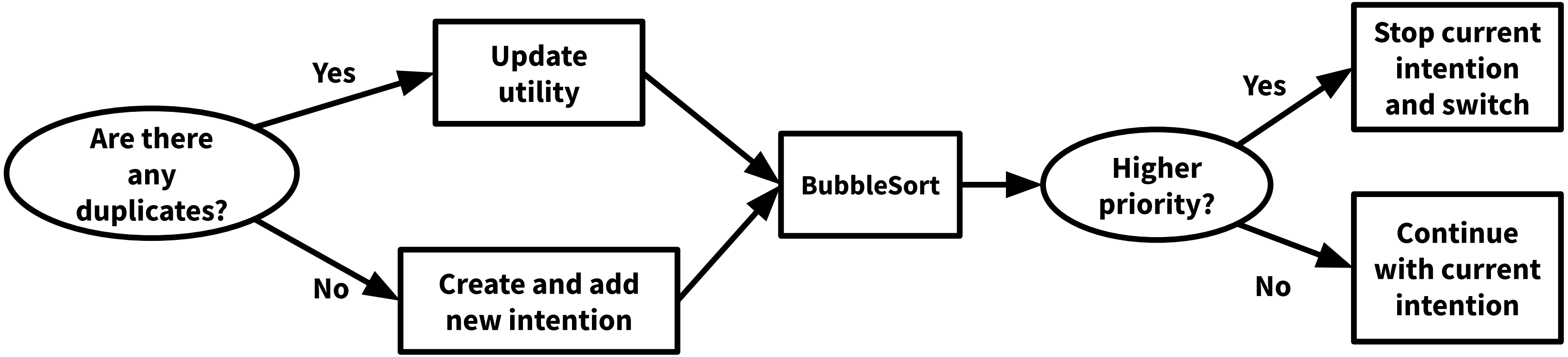
The **intention loop** is responsible for **continuously processing** the agent's intentions.
Agent try to achieve the current top intention in the queue. If have **successful** achieve it, based on the intention itself it performs some additional steps.



Intention Revision

After each `best_option` is chosen, intention revision step is performed.

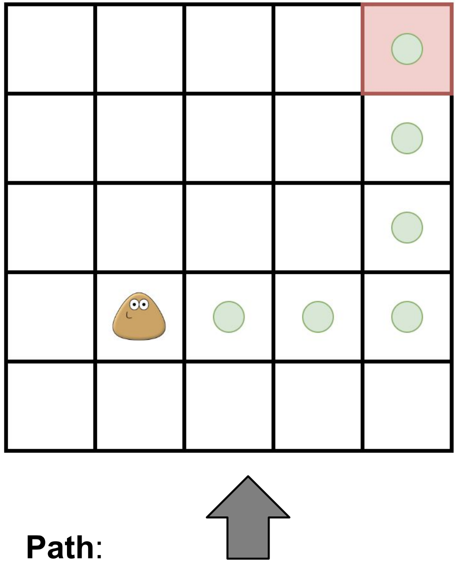
Based on the utility values, we determine whether we should stop our current plan and switch or continue with our current course of action.



Planning

To calculate **path** our agent use:

- **BFS**: for fast task (ex. for utilities)
- **PDDL**: for moving in the map
  - **Domain**: static
  - **Problem**: dynamic based on map updated



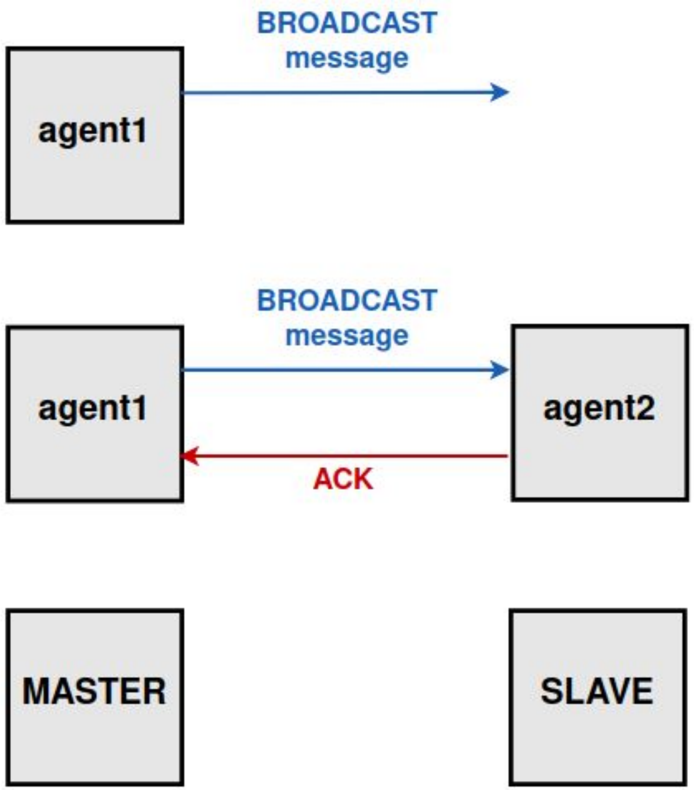
Path:
```
[
  { x: 2, y: 1 }, { x: 3, y: 1 },
  { x: 4, y: 1 }, { x: 4, y: 2 },
  { x: 4, y: 3 }, { x: 4, y: 4 },
]
```

Communication - handshake

1) The first agent to log in sends a broadcast message to all agents in the game

2) When **`agent_2`** login, he is able to decode **`agent_1`** broadcast message, and send back to him an ACK.

3) Once **`agent_1`** receive the ACK, the roles are instantiated as shown in the figure.



Communication - Belief/options sharing

Agents **continuously exchange information** to keep update of each other's information.

Notice that the master holds two instances of **`AgentData`**:

- **`MyData`** for his data
- **`CollaboratorData`** for the SLAVE's ones