# Agent-Oriented Software Engineering 2019/2020 Project

Software Engineering Group, University of Trento

May 2020

## Objectives

The main objective of the project consists in solving two assignments related to the knowledge acquired during the lab sessions. The first assignment requires to design a PDDL domain and two PDDL problems. The second assignment concerns the design of a multi-agent system capable of managing a fully autonomous warehouse. Further details are given in the following sections.

## 1 First assignment

The n-puzzle is a sliding puzzle that consists of a frame of numbered square tiles in random order with one tile missing. The object of the puzzle is to place the tiles in order by making sliding moves that use the empty space.

(a) Example of a 8-puzzle solved.

(b) Example of a 15-puzzle.

1. Design the PDDL domain for solving problems concerning the famous n-puzzle board game.
2. In order to test the correctness of the PDDL domain designed before, implement the two PDDL problems shown in fig. 2a and fig. 2b.

Start

| | | |
|---|---|---|
| 2 | 6 | 1 |
|  | 7 | 8 |
| 3 | 5 | 4 |

Goal

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 |  |

(a) 8-puzzle problem.

Start

| | | | |
|---|---|---|---|
| 15 | 2 | 1 | 12 |
| 8 | 5 | 6 | 11 |
| 4 | 9 | 10 | 7 |
| 3 | 13 | 14 |  |

Goal

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 |  |

(b) 15-puzzle problem.

## 2 Second assignment

The goal of this assignment consists of designing the management of an autonomous warehouse, where drones and robots are employed to retrieve, move and deliver packages (represented as boxes, in our case).

- The designed Multi-Agent System must be completely autonomous, hence there should not be any need for run-time human intervention.
- The implementation has to follow an agent-oriented approach. The delivered knowledge base should work equally well on both your personal test-cases and in our own scenarios; no hard-coded solutions!
- The C`#` scripts can be modified at will, provided that the points above are respected.
- The project is **individual**, changing the names of the variables of someone else project will not make it an original work!

You will find the framework of the project @ `https://bit.ly/AOSE19-20`

## 2.1 Specifications

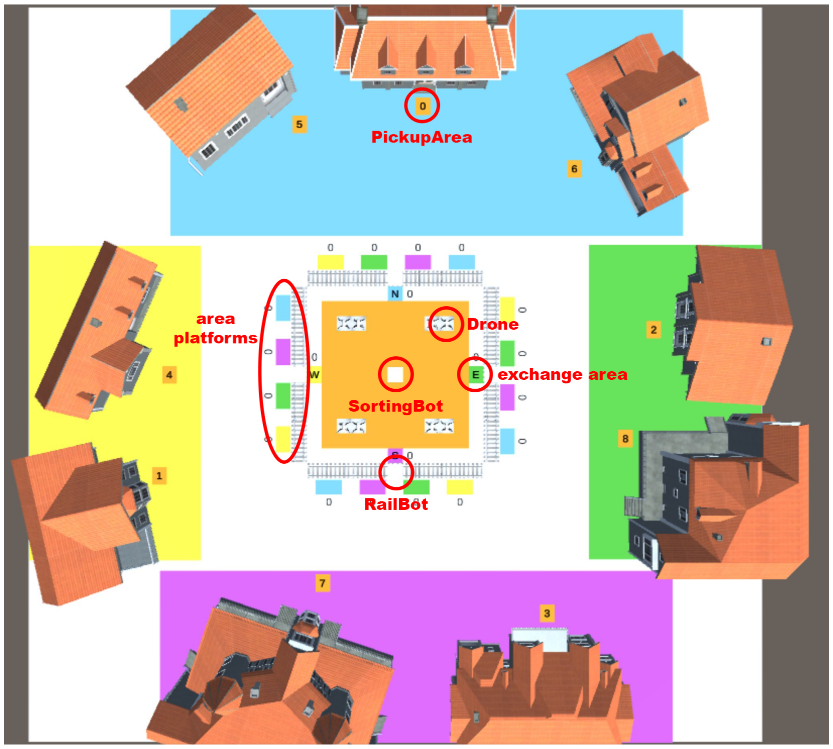


Figure 3: Representation of our warehouse.

### 2.1.1 The environment

Figure 3 shows the representation of our warehouse. Every area is distinguished by a different color (i.e., north, south, east, west). In case you are color-blinded, you can infer which color corresponds to which area by simply looking at the label on the corresponding exchange area (N, S, E, W).

For each area, there are 4 different platforms. Each platform has the same color of the finishing or starting area, depending whether the box is in the retrieving or delivering phase respectively. For instance, a box that should be delivered from address 4 to address 2, will be transferred by the Drone on the green platform of the west area (because the box has to arrive in the east area, which is green) and then the RailBot of the east area will place the box – given by the SortingBot – on the yellow platform of the east area (because the box has arrived from the west area, which is yellow). For more clarity, you can find a demo showing the expected behavior in the *didatticaonline* (moodle) platform.

The numbers near each platform simply show the number of boxes being there at each instant of time.

### 2.1.2 The MAS

The MAS is characterized by 1 model of artifact:

- Box
  It represents the model of the boxes that will be instantiated at run-time;

and by 5 models of agent:

- PickupArea
  These areas are full-fledged agents: they have plans, they can add/remove desires and beliefs from other agents, etc. They have mainly 3 roles:

  1. represent the post-office box of the house,
  2. deciding which drone must take care of a box it needs to send,
  3. recall a box sent to it (actually destroying it).

  As can be seen in the `PickupArea.cs` code, at the very start of the simulation to each PickupArea a belief is added stating to which area it belongs (i.e., north, south, east, west). This information could be useful to correctly design the knowledge bases of the other agents.

- GameManager
  The GameManager is the agent responsible for the creation of the boxes. It also triggers the PickupArea representing the address of the sender. We have already provided you an example of implementation for the knowledge base of such an agent, but feel free to edit it in order to make it suit your design choices.

- Drone
  Drones' main responsibility is to retrieve boxes from the PickupAreas and deliver them to the corresponding platform, and vice versa, as explained in section 2.1.1;

- RailBot
  Each RailBot can only move along its own rail. Its main duty is to either

  – transfer the box from a platform to the exchange area and delegate the management of the box to the SortingBot (if the box must be delivered to a different area), or
  – leave the box where it is and ask a drone to deliver to the correct address (if box's destination address is in the same area as the starting address);

- SortingBot
  The SortingBot is responsible for the sorting of the boxes from an exchange area to another one. It is invoked by the RailBot of the starting area, and it delegates the management of the box to the RailBot of the destination area.

Drones, RailBots and the SortingBot may not have enough energy to do more than one travel, so they have to recharge after each of them (i.e., have to go back to a charging station before starting a new task). The position of the charging stations is shown by the circles in the image below.

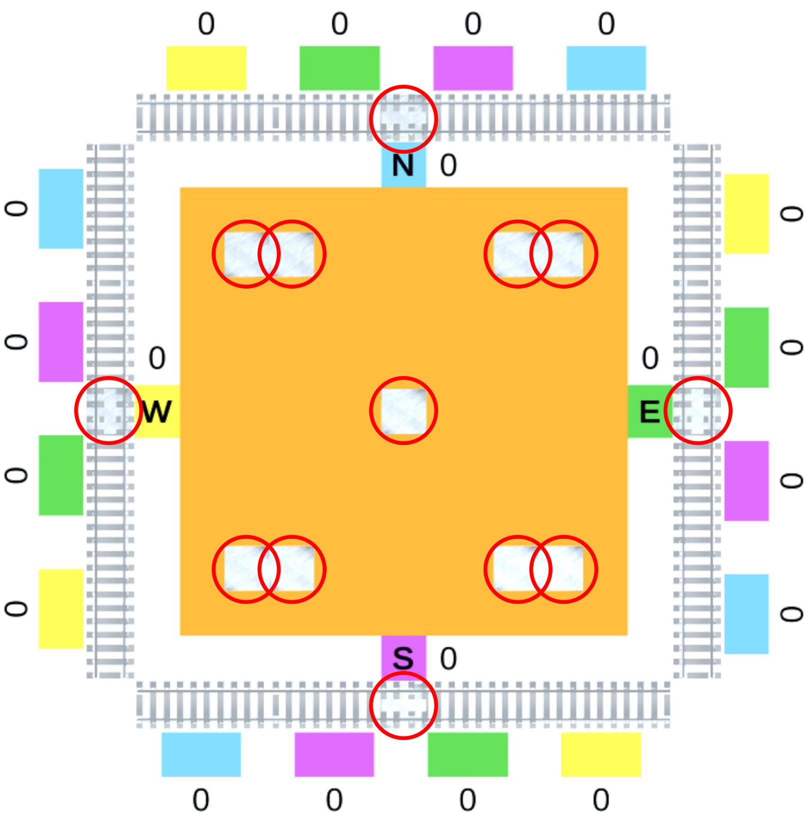


Figure 4: Position of the charging stations.

## 2.2 Tools

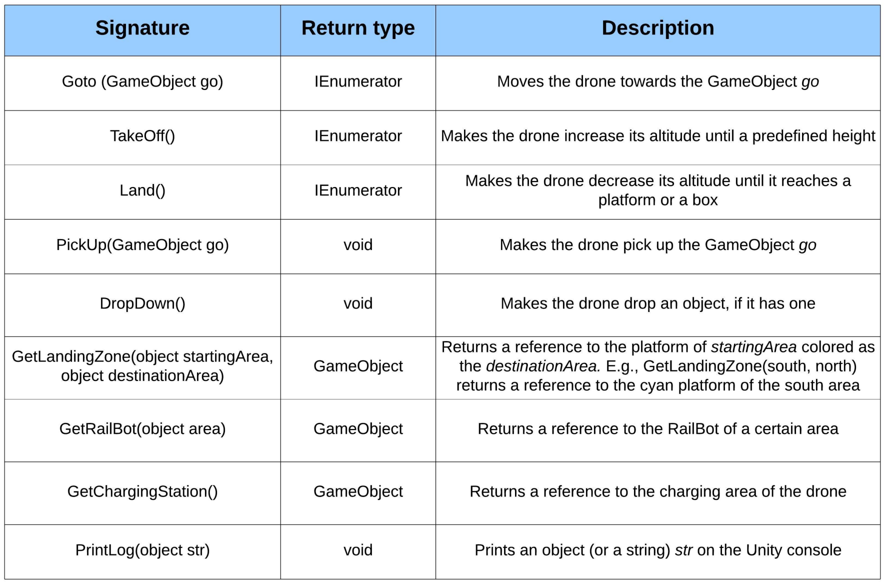
UnityProlog allows the design of BDI agents in Unity, as seen in lab classes. A detailed documentation of UnityProlog is available on the *didatticaonline* (moodle) platform.

The student, besides creating its custom C# APIs (not necessary for solving the assignment), can use different API depending from the agent type.

**IMPORTANT! The functions that return an IEnumerator type are coroutines (so should be activated by the *cr* command) while all the others are methods (hence the *act* command should be used).**

## Drone

Drones have the following capabilities:



| Signature | Return type | Description |
|---|---|---|
| Goto (GameObject go) | IEnumerator | Moves the drone towards the GameObject *go* |
| TakeOff() | IEnumerator | Makes the drone increase its altitude until a predefined height |
| Land() | IEnumerator | Makes the drone decrease its altitude until it reaches a platform or a box |
| PickUp(GameObject go) | void | Makes the drone pick up the GameObject *go* |
| DropDown() | void | Makes the drone drop an object, if it has one |
| GetLandingZone(object startingArea, object destinationArea) | GameObject | Returns a reference to the platform of *startingArea* colored as the *destinationArea.* E.g., GetLandingZone(south, north) returns a reference to the cyan platform of the south area |
| GetRailBot(object area) | GameObject | Returns a reference to the RailBot of a certain area |
| GetChargingStation() | GameObject | Returns a reference to the charging area of the drone |
| PrintLog(object str) | void | Prints an object (or a string) *str* on the Unity console |

Figure 5: Default API for the design of Drone agents.

## RailBot

RailBots have the following capabilities:

| Signature | Return type | Description |
|---|---|---|
| Goto (GameObject go) | IEnumerator | Moves the bot towards the GameObject *go* |
| PickUp(GameObject go) | void | Makes the bot pick up the GameObject *go* |
| DropDown(GameObject area) | void | Makes the bot drop an object, if it has one, on the *area* area |
| GetExchangeArea() | GameObject | Returns a reference to exchange area of the bot's rail |
| GetSortingBot() | GameObject | Returns a reference to the SortingBot |
| GetDrone() | GameObject | Returns a reference to a random Drone |
| GetArea(object areaName) | GameObject | Returns the proper platform given an *areaName* (i.e., north, south, east, west) |
| GetChargingStation() | GameObject | Returns a reference to the charging area of the bot |
| PrintLog(object str) | void | Prints an object (or a string) *str* on the Unity console |

Figure 6: Default API for the design of RailBot agents.

### SortingBot

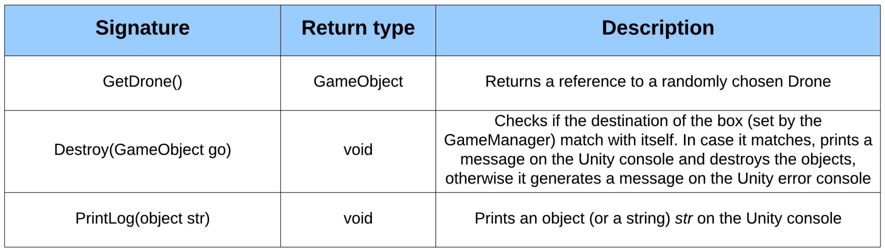
The SortingBot has the following capabilities:

| Signature | Return type | Description |
|---|---|---|
| Goto (GameObject go) | IEnumerator | Moves the bot towards the GameObject *go* |
| PickUp(GameObject go) | void | Makes the bot pick up the GameObject *go* |
| DropDown(GameObject area) | void | Makes the bot drop an object, if it has one, on the *area* area |
| GetExchangeArea(object areaName) | GameObject | Returns a reference to exchange area of the *areaName* area |
| GetRailBot(object areaName) | GameObject | Returns a reference to the RailBot of the *areaName* area |
| GetChargingStation() | GameObject | Returns a reference to the charging area of the bot |
| PrintLog(object str) | void | Prints an object (or a string) *str* on the Unity console |

Figure 7: Default API for the design of the SortingBot agent.

### PickupArea

PickupAreas have the following capabilities:

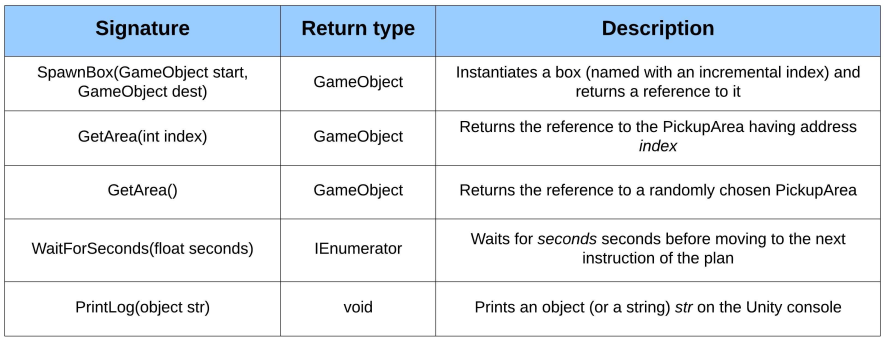


| Signature | Return type | Description |
|---|---|---|
| GetDrone() | GameObject | Returns a reference to a randomly chosen Drone |
| Destroy(GameObject go) | void | Checks if the destination of the box (set by the GameManager) match with itself. In case it matches, prints a message on the Unity console and destroys the objects, otherwise it generates a message on the Unity error console |
| PrintLog(object str) | void | Prints an object (or a string) *str* on the Unity console |

Figure 8: Default API for the design of PickupArea agents.

### GameManager

The GameManager has the following capabilities:



| Signature | Return type | Description |
|---|---|---|
| SpawnBox(GameObject start, GameObject dest) | GameObject | Instantiates a box (named with an incremental index) and returns a reference to it |
| GetArea(int index) | GameObject | Returns the reference to the PickupArea having address *index* |
| GetArea() | GameObject | Returns the reference to a randomly chosen PickupArea |
| WaitForSeconds(float seconds) | IEnumerator | Waits for *seconds* seconds before moving to the next instruction of the plan |
| PrintLog(object str) | void | Prints an object (or a string) *str* on the Unity console |

Figure 9: Default API for the design of the GameManager agent.

## 3 Deliverable

The student is required to deliver, in a compressed folder, the following:

Concerning the n-puzzle assignment

- The PDDL domain file.
- At least two PDDL problem files, representing the 8-puzzle and 15-puzzle shown in fig. 2a and fig. 2b.

Concerning the warehouse assignment

- The folder *KBs*, containing the knowledge-base of agents and artifacts.
- The folder *Scripts*, **only in case you have modified any C# script, or created a new one**.
- A brief report, where the student is asked to explain the solution and describe the encountered problems, if any, and how he/she managed to work around them.

The deliverable should be sent by e-mail to `francesco.alzetta@unitn.it`, having *"AOSE Project delivery"* as subject, by the $9^{th}$ of June.