COMP3211 2021-22 Spring Semester Project on Multi-Agent Path Finding
(For A Team of Two)

Date assigned: Thursday, Mar 31.

Due time: 23:59 on Tuesday, May 10.

How to submit it: Submit your work as a zip file - see below for details.

Penalties on late submission: 20% off each day (anytime after the due time is considered late by one day)

This document is for those who choose to do the project as a team of two students. The tasks are exactly the same as the ones for the individual project except that you need to consider three agents.

1 Project description

Imagine in a warehouse, a fleet of robots are designed to pick up packages from certain locations and deliver them to certain ports. They should deliver the packages as efficiently as possible, and at the same time, must not bump into each other or obstacles (no collision).

In this project, you are going to design environment specific controllers for three robots. There are three environments called `small`, `medium` and `large` - see the appendix. Each environment comes with three fixed goal positions, one for each robot.

For each environment, you need to come up with 3 policies (π_1, π_2, π_3), one for each robot. A policy for a robot is a function from states to actions, with the following definitions of states and actions:

- A state is a tuple (l_1, l_2, l_3), where l_i is the current location of robot i.
- The actions are: $up, down, right, left, nil$, with their usual meaning.

We will test your policies on some random initial states, and measure the quality of them by the sum of the numbers of actions the robots take to arrive at their respective goal locations.

2 Project details

This project comes with a simulator and a code skeleton.

2.1 File Description

1. `game.py`: provides the class `Env` which takes as input a map and a set of goals, and the class `Game` which takes as input a set of initial positions, a group of agents and an instantiated environment.
2. `base.py`: provides several helper functions as well as the `BaseAgent` class which restricts the inputs of an agent to only include a name and an instantiated environment.
3. `run.py`: parses the given commandline. You can also see how the agent will be tested in the main function.
4. `animator.py`: visualizes the whole process, you do not have look into this file.
5. `map/`: provides three maps named `small.map`, `medium.map`, `large.map`. You agent will only be tested on these three given maps (i.e., no hidden map will be used for evaluation). Each map file is also associated with a set of goals and a constant `MAX_NUM_STEP` representing the maximum number of steps allowed in this map.
6. `agent.py`: a simple demo agent.

2.2 Environment Set-up

1. Check if anaconda has been installed on your PC,

   ```
   conda -V
   ```

2. Create a virtual environment for this project with python 3.8 and activate the environment,

   ```
   conda create -n mapf python=3.8
   conda activate mapf
   ```

3. Install dependency packages,

   ```
   pip install -r requirements.txt
   ```

2.3 Run the Code

1. Activate the environment,

   ```
   conda activate mapf
   ```

2. You are given a simple agent in `agent.py`, to run it,

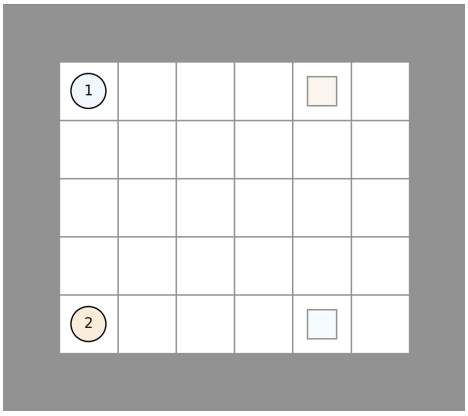
   ```
   python run.py --agents p1 p2 --map empty --goals 5_5 1_5 --vis
   ```

This command launched two agents named 'p1' and 'p2' on the map named 'empty' and goals are specified as (5,5) and (1,5) for each, respectively. The `--vis` option turns on the visualizer. Note that the visualizer for large maps is quite slow, you may want to turn off the `--vis` option for large maps.

3. Specify the initial positions for each agent as input,

   ```
   Specify an initial position for agent p1: 1 1
   Specify an initial position for agent p2: 5 1
   ```

   If the code runs successfully, then you will see an animation window like this as well as the whole path finding history printed in the terminal.



   Note that you only need to specify the map and goals once but then different initial positions are allowed to input as many times as you like, which supports what we decribed as *policies* above. If you want to terminate the program, just input `n` as an invalid coordinate.

   ```
   Specify an initial position for agent p1: n
   ```

4. For further detailed usage,

   ```
   python run.py -h
   ```

2.4 Tasks

Your main task is to implement `MyAgent` class in `agent.py`. In particular, you need to implement a `MyAgent.get_action()` function. This function is basically your controller: it will be called to decide which action to do in each state. In order for the robot to run in realtime, a hard constraint is that ***every call to this function will need to return in 1 second***. Otherwise the *nil* action will be selected for all agents. This function will have

access to information about which environment the robots are in so that your controller can be environment specific.

You are allowed to 1) implement any other helper functions, which can be global functions or member functions; 2) import any other packages. This will enable you to design your controller in any way you want. For example, you can build a database of state-action pairs. Then your `MyAgent.get_action()` can simply perform query to this database. However, this is unlikely to be effective for the large map as the number of states is very large. Another approach is to train a machine learning model for each map, and then use this model in your controller.

2.5 Evaluation

For the evaluation of each map, we will instantiate a team of your agents once and then test your agents with 10/20/30 (for maps of different sizes) randomly generated sets of initial positions. For each set of initial positions, you will have a time limit of `MAX_NUM_STEP`. Let *total_num_steps* be the sum of the numbers of actions the robots take to arrive their respective goal locations, your score in that round will be

$$\begin{cases} 100 - 100 \cdot \frac{total\_num\_steps}{\texttt{MAX\_NUM\_STEP}} & \text{, all agents accomplish their jobs within the time limit} \\ -50 & \text{, any collision happens} \\ 0 & \text{, otherwise} \end{cases}$$

Your score for that particular map will be the average of scores over all rounds. Finally, your total score will be

$$15\% \cdot avg\_score(small\_map) + 25\% \cdot avg\_score(medium\_map) + 40\% \cdot avg\_score(large\_map)$$

Since this is a team project, you will need to consider three-agent cases, that is, your submitted agent will be tested using the following command,

```
python run.py --agents p1 p2 p3 --map {map} --goals {g1} {g2} {g3} --eval
```

3 Submission

You must submit a report of your project (`report.pdf`, one column, at most 3 pages). It must describe the methods that you used to design your controller. Since this is a team project, please describe the role of each member.

Before submission, please re-write the dependency file if you have used additional packages (remember to switch to the project directory first),

```
pip install pipreqs
pipreqs --force ./
```

Then arrange your files as follows in a folder named as studentIDs of both team members and submit the whole zip file (e.g., 123456_135246.zip).

```
{studentID1}_{studentID2}/
    |- report.pdf
    |- agent.py
    |- requirements.txt
    |- extra/
           |- other_useful_file1
           |- ...
```

You can put any of your trained model files or helper functions under the `extra/` folder. 20% of your grade will be based on your report and 80% from the total score that your controllers get in evaluation as described above.

4 Enquiries

- Enquiries before deadline: Fengming ZHU (fzhuae@connect.ust.hk)
- Grading:
  - Zhili CHEN (zchenei@connect.ust.hk),
  - Chenglin WANG (chenglin.wang@connect.ust.hk)

5 Appendix - maps

A small map (Figure 1, size: 8×8):

A medium-sized map (Figure 2, size: 18×18):

A large map (Figure 3, part of Berlin, size: 256×256):

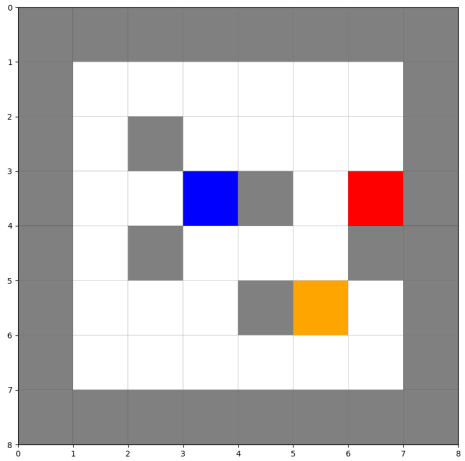


Figure 1: $goal_1(orange) = (5, 5), goal_2(blue) = (3, 3), goal_3(red) = (3, 6)$

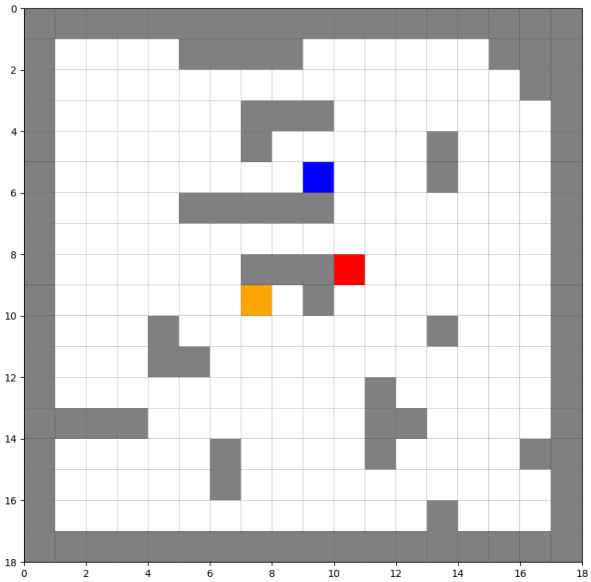


Figure 2: $goal_1(orange) = (9, 7), goal_2(blue) = (5, 9), goal_3(red) = (8, 10)$

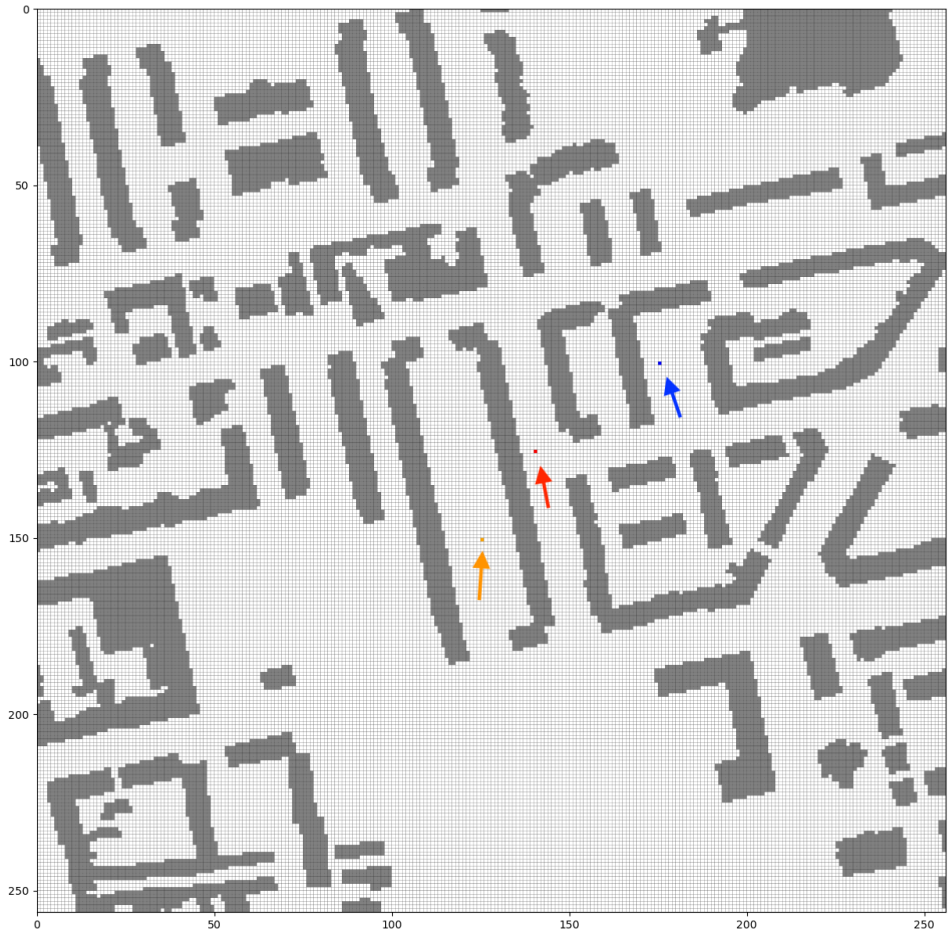


Figure 3: $goal_1(orange) = (150, 125), goal_2(blue) = (100, 175), goal_3(red) = (125, 140)$