National University of Singapore

CS4246/CS5446 AI Planning and Decision Making

Homework 2

Submission Deadline: **March 15, 2021 @ 23:59**

Please write the following at the top of your files for the programming assignment

- Names of both team members as it appears on LumiNUS
- Collaborators (write **None** if no collaborators)
- Source, if you obtained the solution through research, e.g. through the web.

On Collaboration

- While you may collaborate, but you *must write up the solution yourself.*
- It is okay for the solution ideas to come from discussions. However, it is considered as plagiarism if the solution write-up is highly similar to your collaborator's write-up or to other sources.

Submission

- Your solution for the programming assignment should be submitted to both LumiNUS (zip files) and **the code uploaded to aiVLE evaluation server**.
- ***Late submission:***
  - **Deadline for the written component is fixed**, no late submissions are allowed.
  - For the programming assignment, you will incur a late penalty of 20% of your score for late submissions.
  - No submission will be accepted after March 18, 2021 @ 23:59.

Programming Assignment (4 marks)

In the first homework, we learnt to solve deterministic planning problems with PDDL solvers. In this task, we will learn to solve planning problems in non-deterministic environments with two separate algorithms, Value Iteration and Monte Carlo Tree Search.

We will be using the same gym_grid_environment (https://github.com/cs4246/gym-grid-driving) to simulate the solutions. All dependencies have been installed in the docker image "cs4246/base" which you have already downloaded for Homework 1.

By now, you should be familiar with the gym_grid_environment. Go through the IPython Notebook file on Google Colab here, if you have already not done so in the previous programming assignment. You are now ready to begin solving the two tasks which are listed below.

1. **Value Iteration:** In the crossing the road task in homework 1, we assumed all the cars move with a constant speed of -1. Most of the times, this assumption does not hold. There is

inherent noise in the car speeds due to reasons like bumpy roads, uneven driving etc. This non-deterministic behaviour is not captured in the PDDL format.

For problems that are not very large, planning algorithms like Value Iteration can be used to handle non-deterministic behaviours. In this task, we model the problem as a Markov Decision Process and use Value Iteration algorithm to find the optimal policy.

We have provided the script which models the problem as an MDP. Every state in this MDP is represented as a tuple of features
$\langle \text{Agent}_x, \text{Agent}_y, \text{Car}^1_x, \text{Car}^1_y, \cdots, \text{Car}^N_x, \text{Car}^N_y, \text{isTerminal} \rangle$,
where:

- $\text{Agent}_x, \text{Agent}_y$ denote the x and y coordinates agent.
- $\text{Car}^N_x, \text{Car}^N_y$ denote the x and y coordinates of Car^N.
- isTerminal is a boolean variable denoting whether the state is terminal.

Your task is to fill up the function that implements the Value Iteration algorithm which is marked with "FILL ME".

You can test your code with the test configurations given in the script by running `python __init__.py` (present in the .zip file) on the docker (using the `docker run ...` command).

HINT : You can use Matrix multiplication to optimise the code, but refrain from using the function `np.matmul()/np.multiply()/np.dot()` as these functions interfere with the evaluation script. Instead you can use function `matmul()` defined in the same script.

2. **Monte Carlo Tree Search :**

Unfortunately, we moved to a bigger parking lot and Value Iteration is not feasible as it does not scale well with large state spaces. However, we can use Monte Carlo Tree Search (MCTS) to handle such cases.

We have provided a separate script which solves the problem with MCTS, Your task is to complete the script by filling up 2 code snippets inside the functions marked with "FILL ME".

The functions required to be filled up are,

- backpropagate() : This function should implement the backpropation step of MCTS.
- chooseBestActionNode() : Populate the list bestNodes with all children having maximum value. The value of the i^{th} child node can be calculated with the formula $\frac{v_i}{n_i} + c\sqrt{\frac{\log N}{n_i}}$. where,
  - v_i : sum of returns from the i^{th} child node
  - n_i : Number of visits of the i^{th} child node

- N : Number of visits of the current node
- c : The exploration constant. We set it to be 1.

For more details on MCTS, it might be helpful to look here. You are also encouraged to look through the rest of the code to see how the entire MCTS algorithm is implemented.

You can test your code with the test configurations given in the script by running `python __init__.py` (present in the .zip file) on the docker (using the `docker run ...` command).

Submission Instructions

Please follow the instructions listed here. You have to make 2 separate submissions as mentioned below :

1. For Task (1) : You have been provided with a vi_agent.zip file in the correct submission format. Complete the functions marked with "FILL ME" in `__init__.py`. Your submission zip file should have the exact same structure as the zip file you received.
2. For Task (2) : Modify the `__init__.py` in the mcts_agent.zip by completing the functions with "FILL ME" and make a separate submission.

Submit both the zip files to the designated folder on LumiNUS as well.
Please name your files according to this format:
`<GroupName>-Task1` and `<GroupName>-Task2`.
E.g., `CS4246Group1-Task1.zip` and `CS4246Group1-Task2.zip`

Note :

1. Please remember to set the variable SUBMISSION in `__init__.py` to False when testing locally, and to True before making a submission.
2. Please do not print anything to the console, as it might interfere with the grading script.