National University of Singapore
School of Computing

Semester 1, AY2022-23

CS4246/CS5446 AI Planning and Decision Making

Issued: 18 Aug 2022 Due: 2 Sep 2022@23:59

Assignment 1

General Guidelines

Please complete the Assignment in a **2-person TEAM**.

- While not encouraged, you may form a team with someone from a different Tutorial Group.

On collaboration

- You are encouraged to discuss solution ideas. However, each team *must write up the solutions independently*. It is considered plagiarism if the solution write-up is highly similar to other teams' write-ups or to other sources.

Homework Problems Submission

- Submission will take place through **LumiNUS Quiz**.
- Only **ONE** person from each team has to post the answers. Grading will be done team-wise.
- No late submissions for the homework assignment solutions are allowed.

Programming Problems Submission

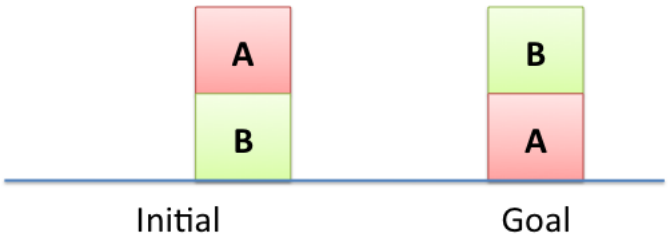
- **IMPORTANT!** Please write the following on the top of your solution files for EACH problem in the assignment:
  - *Name, metric number and Tutorial Group* of **all team members** as they appear on LumiNUS
  - Collaborators (write **None** if no collaborators)
  - Sources, if any, or if you obtained the solutions through research, e.g. through the web.
- **Upload the code of your programming assignment to the aiVLE evaluation server**, by following the instructions here. Please keep a copy of your submission zip file for safety.

- Please name your files, using ONE of the members' information according to this format: `A1_<Member1-TutorialGroup>-<Member1-name>-Task1` and `A1_<Member1-TutorialGroup>-<Member1-name>-Task2`.
  E.g., `A1_TG6-joycexiaomei.tan-Task1.zip` and `A1_TG6-joycexiaomei.tan-Task2.zip`
- A late score penalty of 20% will be incurred for late programming assignment solutions submissions.

1 Homework Assignment (LumiNUS Quiz)

1.1 Homework Problem 1: The Blocks World Reloaded

(15 marks) Consider a blocks-world as shown in the figure below. The objective of this problem, called *swap-blocks*, is to swap the stacking order of the two blocks, A and B on the Table (represented by the horizontal line).



a. Express the blocks-world problem as a planning problem in the Planning Domain Definition Language (PDDL). You may use the following assumptions, and/or make **additional** assumptions as necessary:
   - There are only four objects in the world – Block A, Block B and the Table.
   - Every block is on top of another object, i.e., On(x, y), where x is a block and y is another object.
   - You can re-arrange a block by moving it to (the top of) another object, i.e., Move(x, *fr*, *to*), where x is a block, and *fr*, *to* are other objects in the world.
   - You can move a block only when it is not stacked upon, i.e., it is "clear" on the top, or Clear(x), where x is a block.
   - You **cannot** move the Table, but you can always move something to or from the Table, i.e., Clear(Table) is always assumed to be True (interpreted as: there is always "clear space" on the Table).

b. Show the search tree for planning by forward search.

c. Show the search tree for planning by backward search.

d. Show a valid plan for the problem.

1.2 Homework Problem 2: Programming as Planning

(10 marks) [RN4e 11] Some of the operations in standard programming languages can be modeled as actions that change the state of the world. For example, the assignment operation changes the contents of a memory location, and the print operation changes the state of the output stream. A program consisting of these operations can also be considered as a plan, whose goal is given by the specification of the program. Therefore, planning algorithms can be used to construct programs that achieve a given specification.

a. Write an action schema for the assignment operator (assigning the value of one variable to another). Remember that the original value will be overwritten!

b. Show how object creation can be used by a planner to produce a plan for exchanging the values of two variables by using a temporary variable.

2 Programming Assignment (aiVLE submission)

In this assignment, we will learn to generate the PDDL description files which will be used to solve 2 different planning problems. Before proceeding further, follow the instructions below to complete the setup required for this programming assignment.

Installation Instructions[1]

- Setup docker on your machine, instructions for which can be found here.
- Pull the docker that we have already setup for you with all the required dependencies. You can follow the instructions here to do so.
- After pulling the docker image, test your your installation by running
  `docker run -it --rm -v $PWD:/workspace cs4246/base python test_installation.py`
  for Linux/Mac or
  `docker run -it --rm -v \%(cd)\%:/workspace cs4246/base`
  `python test_installation.py` for Windows (file given along with this assignment) on the docker container.

Programming Assignment Submission Instructions

Please follow the instructions listed here. Task specific submission instructions are as follows:

a. For Task 1 : You have been provided with a .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.

b. For Task 2 : Modify the `__init__.py` as required and make a separate submission. You can add code out of "FILL ME" block and even edit some code in the template if needed.

Note :

- Please do not print anything to the console, as it might interfere with the grading script.

Getting started

We will be using the gym_grid_environment to simulate the solution obtained on feeding the PDDL files generated to a planner. These dependencies have been installed in the docker image "cs4246/base" which you have downloaded.

Read through the introduction and example from Pellierd/pddl4j tutorial to understand the PDDL description format. You can look at sample problem and domain files for further understanding. If you want to, you can also feed any problem and domain PDDL files to a planner by running the following command on the docker container (using the `docker run ...` command):

[1]The files to be installed are very large. Please be mindful if you intend to use metered data connections. Alternatively, you can do the installation if/when you are in SoC.

```
/fast_downward/fast-downward.py  domain.pddl problem.pddl  --search "lazy_greedy([ff()], preferred=[ff()])"
```

Note that in this PDDL format, specifying negative literals in preconditions is allowed.

For you to get used to the PDDL format, we have prepared a sample python script to generate PDDL files for the Air Cargo problem. You can run the file `pddl_cargo_example.py` (using the `docker run ...` command)] to see the PDDL files generated `cargodomain.pddl`, `cargoproblem.pddl` and also relevant portions of the code that generates it. Specifically, it will be helpful to look at functions :

1. `generateDomainPDDLFile()`,
2. `generateProblemPDDLFile()`
3. `generateInitString()`
4. `generateGoalString()`

To complete this assignment, you need to know some details about the environment : how a state is represented, the actions the agent can take, and few other small intricacies. We have prepared an IPython Notebook file on Google Colab here to walk you through all such details. It would be helpful for you to go through it before working on the tasks. While this may seem like a lot to understand for one homework assignment, we will be using the same environment for the other homeworks and for the project!

You are now ready to begin solving the two tasks which are listed below!
We use the PDDL description format mentioned here.

2.1 Programming problem 1: Parking Task

(15 marks) You are employed as a valet at a parking lot. You have to drive the car given to you and park it at a parking spot assigned to the car (identified as goal state in the environment). But, you also have to plan your way from your current position to the spot while wasting the least fuel possible (your tip depends on it!).

By virtue of being a Computing student, you decide to put your *planning* skills to use. You first retrieve an old python script written for a similar task by the TAs of CS4246/CS5446, parts of which have been lost. The script used to generate the PDDL files and fed them to the fast-downward solver to generate a solution for this planning problem. It also runs the plan on the simulator to see if it does the job.

The script `python __init__.py` is missing 3 code snippets (marked with "FILL ME" in the file) .

1. Code which generates the action schemas of the three actions available namely, UP, DOWN and FORWARD. These action schemas should reflect the 3 actions available in the environment simulator. The agent's speed range in the environment is restricted to [-1, -1] (negative speed moves towards the left, see the environment for more details).

2. Code which generates the initial state condition.

3. Code which generates the goal description.

You can test your code with the test configurations given in the script by running `python __init__.py parking N` (present in the .zip file) , where N is an int value between 0 to 5, on the docker (using the `docker run ...` command). **It might be helpful to look at the generated PDDL files for debugging.**

2.2 Programming problem 2: Crossing the road

(10 marks) A customer forgets directions to the parking lot, and ends up on the other side of the road. The customer is not skilled enough to cross a busy multi-lane road with **moving cars**, where the speed of cars can differ across lanes, but cars in the same lane move with the same speed. To reach the entrance of the parking lot (identified as goal state in the environment), he calls you up and asks you to drive the car from his spot (the initial state) to the destination. Unlike the previous task, the agent's speed range in the environment is restricted to [-3, -1].

You decide to modify the script you wrote in Task 1 to handle this situation. You can test your code by running `python __init__.py crossing 0` in command with docker

Hint: Instead of modeling the parking lot/road as a 2 dimensional grid, it might be useful to include time as an additional dimension.