National University of Singapore
School of Computing

Semester 2, AY2021-22

CS4246/CS5446 AI Planning and Decision Making

Issued: 30 Jan 2022 **Due: 21 Feb 2022 at 23:59**

Assignment 1

Information

- This assignment is worth 10 marks out of 100. The score you obtain will be scaled to 10.
- This assignment should be submitted individually.

On collaboration & information source

- You are allowed to discuss solution ideas on the forums. However, you *must write up the solutions independently*.
- It is considered as plagiarism if the solution write-up is highly similar to other students write-ups or to other sources.
- If you obtained the solution through research, e.g. through the web, state your source in the answer rationale in the quiz.

Submission

- The written part of the assignment should be submitted via the corresponding LumiNUS quiz.
- Things to note:
  - You will have only one attempt at the quiz on LumiNUS.
  - There is no time limit for the attempt. However, the quiz closes on the designated deadline.
  - Use the *Rationale* field in the LumiNUS quiz to show your working, or paste the well-formatted answer, or both!
- For the programming part, please **upload the zip to the aiVLE evaluation server**.
- Additionally, the zip file containing your solution to the programming assignment should be submitted on LumiNUS
- ***Late submission:***
  - Deadline for the **written part** is fixed, no late submissions are allowed.
  - For the **programming component**, you will incur a late penalty of 50% of your score for late submissions.
  - Hard deadline for programming component: 25 Feb 2022 @ 23:59. No submission will be accepted after that.

Written questions (12 marks)

1. **[8 marks] Classical planning**

   Sokoban is a puzzle game in which the player pushes boxes around in a warehouse, trying to get them to the target positions. The player can either move to an adjacent blank position (i.e., no wall or box is present in the target location) in the four cardinal directions, or push the adjacent box to the opposite position (E.g, if the player is on the right side of the box, he can only push the box to its left position). An example of the puzzle is given in the figure below.

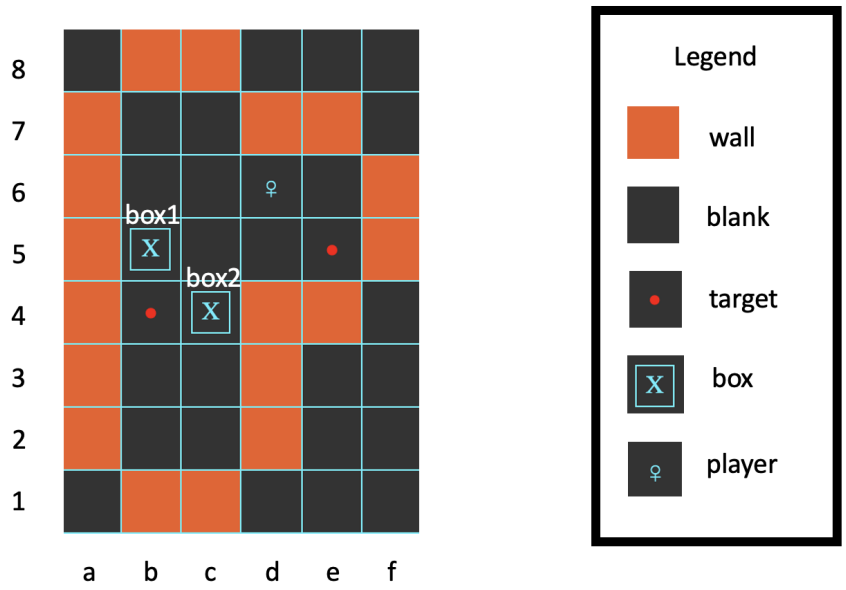

Figure 1: Sokoban

We want to model this puzzle in PDDL. You are allowed to use only the following predicates.

- $\mathbf{At}(x, y)$: subject x (a player or a box) is at position y. E.g, $\mathbf{At}(p, d6)$.
- $\mathbf{Left}(y1, y2)$: position $y1$ is on the left side of position $y2$. E.g, $\mathbf{Left}(d6, e6)$.
- $\mathbf{Right}(y1, y2)$: position $y1$ is on the right side of position $y2$. E.g, $\mathbf{Right}(d6, c6)$.
- $\mathbf{Up}(y1, y2)$: position $y1$ is above position $y2$. E.g, $\mathbf{Up}(d6, d5)$.
- $\mathbf{Down}(y1, y2)$: position $y1$ is below position $y2$. E.g, $\mathbf{Down}(d6, d7)$.
- $\mathbf{Blank}(y)$: position y is blank. E.g, $\mathbf{Blank}(d5)$.
- $\mathbf{Player}(x)$: subject x is a player. E.g, $\mathbf{Player}(p)$.
- $\mathbf{Box}(x)$: subject x is a box. E.g, $\mathbf{Box}(box1)$.
- $\mathbf{Position}(y)$: position y is a position. E.g, $\mathbf{Position}(d6)$.

a. (2 marks) Write the possible goal states the puzzle given in the Figure 1 in PDDL. Assume that the initial state has been properly stated.

b. (4 marks) Write the action schema for the following actions:

- $\mathbf{MoveLeft}(p, y1, y2)$:
  move the player p from position $y1$ to $y2$, where, $y2$ is a blank position to the left of $y1$.
- $\mathbf{PushLeft}(p, b, y1, y2, y3)$:
  Player p (at position $y1$) pushes the box b on his left (at position $y2$), to position $y3$, which is a blank position to the left of $y2$. After the push succeeds, both the player and box positions are updated appropriately.

c. (2 marks) Assume you have enumerated the proper action schema for moving left, right, up, down and pushing the box left, right, up, down. What would be a valid plan for the problem? (Try to keep the plan length to less than 25 actions).
For your convenience, you may omit the variables, and you may use $\mathbf{l}, \mathbf{r}, \mathbf{u}, \mathbf{d}, \mathbf{L}, \mathbf{R}, \mathbf{U}, \mathbf{D}$ to represent $\mathbf{MoveLeft}, \mathbf{MoveRight}, \mathbf{MoveUp}, \mathbf{MoveDown}, \mathbf{PushLeft}, \mathbf{PushRight}, \mathbf{PushUp}, \mathbf{PushDown}$ respectively.

2. **[2 marks] Hierarchical planning**

   If a sequence of primitive actions does not appear in any of the refinement methods of a hierarchical task network (HTN) planner, it will never appear as a solution for any planning problem solved by the planner. State true or false; explain your answer in the rationale field.

3. **[2 marks] Decision theory**

   Alice's bike breaks down with a rattling sound on the weekend. She calls up the dealer to check if it can be serviced. The dealer says due to the rattling sound, there is a 30% chance that the gear mechanism is broken, in which case, it costs $\$200$ to repair, there is a 60% chance that the transmission chain is broken and it costs $\$80$ to repair; there is a 10% chance that the entire transmission assembly is broken, in which case it costs $\$400$ to repair. He also says that there is a new-launch offer for another bike: what is originally priced at $\$400$ is now available at $\$200$. Now (choose the most appropriate answer):
   (Show your working in the rationale field.)

   a. Alice should get the bike repaired

   b. Alice should buy the new bike

   c. Alice should be indifferent between the two choices

   d. Alice should wonder what to do

Programming Assignment (8 marks)

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.

Getting started

We will be using the gym_grid_environment (https://github.com/cs4246/gym-grid-driving) 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):

```
/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()`,

[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.

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.

Problem statement

1. **Parking Task :**
   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, 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) .

   (a) 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).

   (b) Code which generates the initial state condition.

   (c) 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. **Crossing the road :**
   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-line 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.

Submission Instructions

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

(a) For Task (a) : 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 (b) : Modify the `__init__.py` as required and make a separate submission.

Note :

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