Homework 2 - Stochastic Task

Introduction

In this exercise, you continue running the fleet from HW1. However, this time the Marines movement and the place for each treasure is not deterministic.

Task

The environment is like the one in HW1: The same grid world, with the same pirate ships, treasures and marines. Key differences include:

- The marines do not move exactly as their path states (more on that below).
- The execution has a limited number of turns.
- The goal is to collect as many points as possible (more on that below).
- When encountering the marines, the ship pays a fee (more on that in the 'points' section)

Pirate ships:

Now, the input of the pirate ships is a little different than in HW1:

- The "location" parameter is the location of the ship.
- The "capacity" parameter is how much capacity the ship has. Once the ship collects a treasure, it is lowered by 1 and when it deposits treasure, the capacity returns to 2. When the capacity is 0, the ship cannot collect anymore treasure!

```
"pirate_ships": {'pirate_ship_1': {"location": (2, 0),
                                   "capacity": 2}
                 },
```

The location of each ship in the first turn is always the base but can change for next turns (using the "sail" atomic action)

The Stochastic Marines

Each marine has a certain path, the same way as in HW1. However, in this exercise they do not have to precisely follow it: Each marine can stay, move forward, or move backwards in the path.
For example, it the path is $A \to B \to C \to D$ and the marine is in tile B, he can move to tile A, C or stay in tile B. The probabilities are the same for each direction.
If the marine was in tile A, he can move to tile B or stay in tile A. The probabilities are also the same for each direction.
The input for the marines is:

```
    "marine_ships": {'marine_1': {"index": 0,
                                  "path": [(1, 1), (1, 2), (2, 2), (2, 1)]}},
```

The "path" argument is the same path as was in HW1.
The "index" indicates where the marine is on his path.
For example: if we are at (1,2) (i.e index = 1), then the probability to move to (1, 1) is 1/3 and the probability to move to (2, 2) is also 1/3, and the probability to stay in (1, 2) is 1/3 as well.
If the pirate is at (1, 1), then the probability to move to (1, 2) is 1/2 and the probability to stay in (1,1) is also 1/2 .
Notice:

- The path is not necessarily simple so multiple indices can have the same location. The index can increase, decrease, or stay the same as explained above.

The stochastic Treasures

As you have probably realized, we are dealing with special and magical treasures. As such, they can change their location at random at each turn. For example, let “treasure_1” be a treasure with the following parameters:

```
{'treasure_1': {"location": (0, 2),
                "possible_locations": ((0, 2), (1, 2), (3, 2)),
                "prob_change_location": 0.1}
},
```

Here, the treasure’s name is “treasure_1”, its location is (0, 2), each turn there is a 10% chance that the treasure will draw a location, uniformly at random, from the “possible_locations”. Notice, the new location **can** be the location he is currently at and thus, will not move.
Each location in the "possible_locations" must be an island.
Important: if a pirate collects a treasure in the same turn it changes its location, the treasure is still collected.

Actions:

The action syntax and rules remain as they were in HW1, with 2 new actions:

1. **Reset**: It resets the places of treasure, pirate ships and marines to the initial state. The action does not reset the number of turns or the points accumulated. The syntax is simple – “reset” (Just a string, not a tuple containing the string).
2. **Terminate**: This action terminates the game. This may be of use if resetting yields negative expected results. After you terminate the game, the game is over and no more actions can be made. The syntax is simple – “terminate” (Just a string, not a tuple containing the string).

The rest of the actions stay the same – a tuple of "atomic actions". Notice that when the action is Reset or Terminate – it resets or terminates the entire game, i.e. not for one ship alone.

Points:

In this exercise, your goal is to achieve maximum points. Points are given for the following:

- Successfully retrieving a treasure to base: 4 points.
- Resetting the environment: -2 points.
- Encountering a marine. -1 points for each ship that encounters a marine each turn. For example, if 2 ships encounter marines in some turn t, then you get -2 points. Applies for ships with and without treasures.

The input

The input is presented as follows:

```
{
    "optimal": True,
    "map": [
        ['S', 'S', 'I', 'S'],
        ['S', 'S', 'I', 'S'],
        ['B', 'S', 'S', 'S'],
        ['S', 'S', 'I', 'S']
    ],
    "pirate_ships": {'pirate_ship_1': {"location": (0, 1),
                                       "capacity": 2}
                     },
    "treasures": {'treasure_1': {"location": (0, 2),
                                 "possible_locations": ((0, 2), (1, 2), (3, 2)),
                                 "prob_change_location": 0.1}
                  },
    "marine_ships": {'marine_1': {"index": 0,
                                  "path": [(1, 1), (1, 2), (2, 2), (2, 1)]}},
    "turns to go": 100
},
```

- "optimal" indicates that the agent must act optimally.
- "map", "pirate_ships" and "marine_ships" are the same as in HW1.
- "treasures" is as described above.
- "turns_to_go" – the number of turns the execution lasts.

The task

Code-wise, your task is to implement two agents (PirateAgent and OptimalPirateAgent). Each agent shall have (at least) two functions:

- __init__(self, inital) – A constructor.
- act(self, state) – A function that returns an action given a state.

Bonus:

You can implement an additional agent – InfinitePirateAgent that optimally solves problems with infinite turns to go. If implemented optimally, 15 points bonus.
In infinitePirateAgent you will also receive γ in the __init__ (as seen in the code given to you).
Additionally, you need to implement the value(self, state) function which returns $V^*(state)$

Some important notes:

- As you could have guessed, the OptimalPirateAgent is required to solve the problem optimally. The algorithms you learned in class could be useful here.
- The PirateAgent doesn't have to be optimal, yet we expect it to gain strictly positive scores.
- As compared to the OptimalPirateAgent, the PirateAgent will need to handle larger problems.
- Timeouts for initiating an agent and choosing an action are 300 seconds and 0.1 seconds, respectively.
- The InfinitePirateAgent needs to return the **optimal** action and V^* for each state given an infinite number of turns left. If solved using value iterations, use $\epsilon = 10^{-2}$.

Code handout

Code that you receive has 5 files:

1. ex2.py - The only file you should modify, which implements your agent.
2. check.py - The file that implements the environment, the file that you should run.

3. utils.py - The file that contains some utility functions. You may use the contents of this file as you see fit.
4. inputs.py + additional_inputs.py: self-explanatory.

Note: We do not provide any means to check whether the solution your code provided is correct, so it is your responsibility to validate your solutions.

Submission and grading

You are to submit only the file named **ex2_ID1_ID2.py** as a python file (no zip, rar, etc.). We will run check.py with our inputs and your ex2_ID1_ID2.py. The check is fully automated, so it is important to be careful with the names of functions and classes.

The grades will be assigned as follows:

- 60% - If your PirateAgent finishes solving the test inputs with a strictly positive score.
- 25% - If your OptimaPirateAgent solves all given problems optimally.
- 20% - Competitive Part: Grading on the relative performance of the algorithm, based on points.
- Additional 15 points (bonus) – InfinitePirateAgent as mentioned above.
- There is a possibility to earn bonus points by reporting bugs in the checking code. The bonus will be distributed to the first reporter.
- The submission is due on 10.3.2024
- Submission in pairs/singles only
- Write your ID numbers as strings in the appropriate field ('ids' in ex2.py). If you submit alone, leave only one string.
- The name of the submitted file should be "ex2_ID1_ID2.py" where ID1 and ID2 are your IDs. If you submit alone, the name should be ex2_ID.py

Important notes:

- You are free to add your functions and classes as needed. Keep them in the ex2.py file.
- We encourage you to double-check the syntax of the actions as the check is automated. More inputs will be released a week after the release of the exercise.