Homework 2 - Probabilistic Problem

Introduction

In this exercise, you continue to lead the drone delivery company, but this time your clients exhibit a non-deterministic behavior.

Environment

The environment is similar to the one in Exercise 1: the same grid, with the same packages and clients. Key differences include:

- Clients don't have a predetermined path, but rather choose their move probabilistically.
- The agent is not required to output a whole plan but rather given a state output an action
- The execution runs for a limited, pre-determined number of turns
- The goal of the exercise is to collect as many points as possible (more on that in the "Points" section)

Clients' behavior

For each client, you are given two parameters governing its behavior: the starting location and the probability to move in each direction (up, down, left, right, or stay in place). The clients can move only to adjacent tiles. If a client is located near the edge of the grid, the probability of movement beyond the edge is distributed between other directions proportionally. For example, if a client is on the upper edge of the grid (thus can't move up), and has probabilities [0.5, 0.2, 0.05, 0.1, 0.15] – it will move according to probabilities [0, 0.4, 0.1, 0.2, 0.3].

As in the previous exercise, every turn, first the drones execute their commands, and only after that do clients move. I.e. if you have a client on tile X that is about to go to a tile Y, you have to do the delivery on tile X.

Actions

The actions' syntax and rules remain as they were in Exercise 1, with two exceptions:

- Drones can now move diagonally on the grid (still only to the adjacent tiles)
- There are two new global (non-atomic) actions. You may do those instead of sending an action as a tuple of atomic actions.
    - Reset: this action does not consist of any atomic actions, and it resets the places of packages, drones, and clients to the initial state. The action does not reset the

number of turns or the points accumulated so far. It is advisable to use this action when all the packages are delivered so you could accumulate more points, but this may not be the only use. The syntax is simple – “reset”

    ○ Terminate: this action does not consist of any atomic actions and terminates the execution before the turns run out. This may be of use if resetting yields negative expected results. The syntax is simple – “terminate”

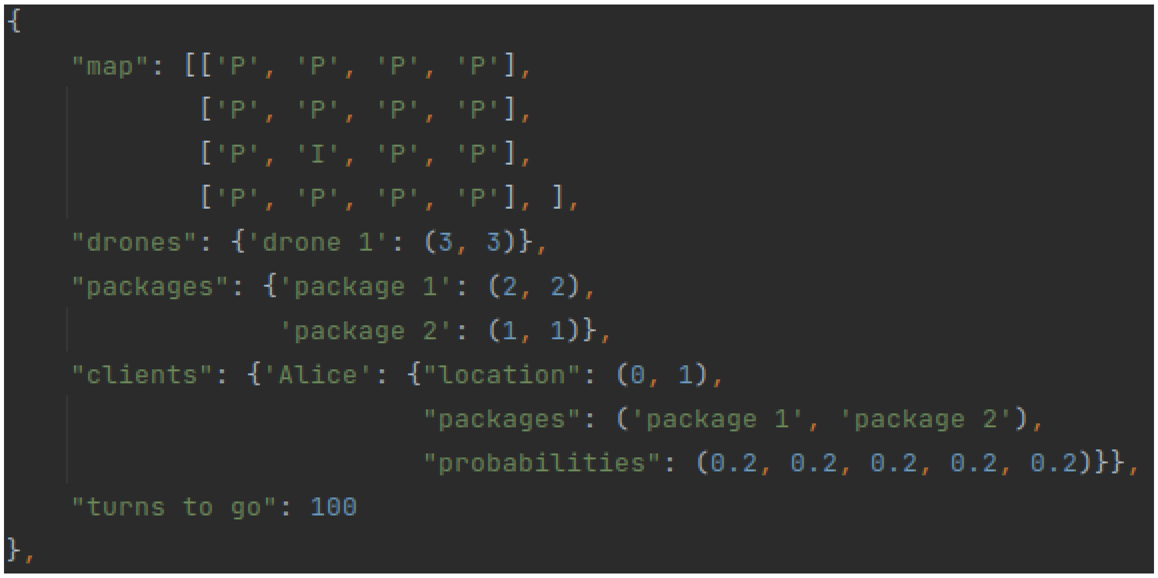
Points

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

- Successful delivery of a package: 10 points
- Resetting the environment: -15 points

The input

The input is presented as follows:



```
{
    "map": [['P', 'P', 'P', 'P'],
            ['P', 'P', 'P', 'P'],
            ['P', 'I', 'P', 'P'],
            ['P', 'P', 'P', 'P'], ],
    "drones": {'drone 1': (3, 3)},
    "packages": {'package 1': (2, 2),
                 'package 2': (1, 1)},
    "clients": {'Alice': {"location": (0, 1),
                          "packages": ('package 1', 'package 2'),
                          "probabilities": (0.2, 0.2, 0.2, 0.2, 0.2)}},
    "turns to go": 100
},
```

Where:

- “map” is a grid - exactly as in the first exercise. ‘P’ means terrain passable for drones, while ‘I’ means impassable. In the example, tile (2, 1) is impassable for drones.
- “drones” contains the names of the drones in the problem along with their initial position.
- “packages” contains the names of packages with their starting locations.
- “clients“ contains the starting location for each client, packages it requires, and probabilities to move to each direction (up, down, left, right, stay in place).
- “turns to go” is the number of turns the execution lasts

The task

Code-wise, your task is to implement an agent (`class DroneAgent`) that has 2 functions:

- `__init__(self, initial)` – a constructor
- `act(self, state)` - a function that returns a global action given a state

The timeout for both functions is 5 seconds

Code handout

Code that you receive has 3 files:

1. ex2.py - the only file that you should modify, 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

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.py as a python file (no zip, rar, etc.). We will run check.py with our own inputs, and your ex2.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:

- 70% - if your code finishes solving the test inputs with a strictly positive score
- 30% - Grading on the relative performance of the algorithm, based on points
- 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 the 2.1.22, at 23:59
- Submission in pairs/singles only.
- Write your ID numbers in the appropriate field ('ids' in ex2.py) as strings. If you submit alone, leave only one string.
- The name of the submitted file should be "ex2.py". Do not change it.

Important notes

- You are free to add your own functions and classes as needed, keep them in the ex2.py file
- We encourage you to start by implementing a working system without a heuristic and add the heuristic later.
- 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

- You may use any package that appears in the standard library and the Anaconda package list, however, the exercise is built in a way that most packages will be useless. You may also use the aima3 package which accompanies the coursebook.
- We encourage you not to optimize your code prematurely. Simpler solutions tend to work best.