Homework 1 - Deterministic Search

Introduction

This homework will be in the world of Harry Potter. The evil lord Voldemort has placed multiple horcruxes in the world and your job is to control a group of wizards to find and destroy them – **without** any wizard dying in the way. Only after you destroyed each horcrux, Voldemort can be killed. However, the task will not be so easy since Voldemort has also placed death eaters to guard them. To achieve this most efficiently, you must make use of the search algorithms shown in class, with the first task being modeling the problem precisely.

Environment

The environment is represented as a rectangular grid, with details provided in the "Input" section as a dictionary. Each cell within this grid symbolizes a different region in the world and can have different properties:

1. **Passable (P):** A passable region – the wizard, death eaters and horcruxes can be there. It can be used by the wizards to move.
2. **Impassable (I):** An impassable region - the wizard, death eaters and horcruxes cannot be there.
3. **Lord Voldemort (V):** The location of Voldemort. A wizard that arrives there before all horcruxes has been destroyed will be killed – thus the game will end.

The core objective is to navigate strategically to destroy ALL of the horcruxes without any wizard getting killed.

Each action in this environment occurs in discrete turns, altering the state of the environment and depicting the current positions of the wizards and death eaters.

Actions

You assume control of the wizards, which can perform various actions to navigate the world, destroy horcruxes, and avoid death eaters:

1. **Move**: The wizard can move up to one tile vertically or horizontally on the grid (they cannot move diagonally), and it cannot move into an impassable tile. The action's syntax is ("move", wizard, (x, y)). For example, if you want to move the wizard "harry" to a tile (0,2), the correct syntax for this action is ("move", "harry", (0, 2)).
2. **Destroy horcrux:** A wizard can destroy a horcrux if it is in a tile that contains the horcrux. The syntax for this action is ("destroy", wizard, horcrux_index).
3. **Wait**: A wizard can choose to wait, which does not change anything about its state or position. The syntax for this action is ("wait", wizard).
4. **Kill:** Only after each of the horcruxes have been destroyed, can Harry Potter kill Voldemort. Notice, the wizard whose name is "Harry Potter" is the only one who can perform this action and only after every horcrux has been destroyed. You can assume that there will be a Harry Potter in each game. The syntax is just ("kill", "Harry Potter")

Each of the above actions is performed in turns or timestamps, allowing the environment and other entities like death eaters to react or change positions based on the chosen action.
These four actions are known as atomic actions, as they relate to a single wizard. Since you can control multiple wizards, you can command them to do things simultaneously. The syntax of a full action is a tuple of atomic actions. Each wizard must do something during any given turn, even if it is waiting.
The actions are performed synchronously.

Examples of a Valid Atomic Action:

- If you want the wizard "wizard_1" to move to a tile (1,3): ("move", "wizard_1", (1, 3))
- If you want the wizard "Ron" to destroy a horcrux in the index i (starting with 0): ("destroy", "Ron", i)
- If you want the wizard "w1" to wait: ("wait", "w1")
- If you want Harry Potter to kill Voldemort ("kill", "Harry Potter")

Note:

- Ensure that each wizard does not attempt to move into impassable tiles.

Examples of a Valid Action:
If you have one wizard: (("wait", "Harry Potter"),)
If you have 2 wizards: (("wait", "Harry Potter"),("move", "ron", (1, 2)))

Notice: the order between the wizards is not important. However, it is important that all wizards get an action, and each wizard gets only one.

Additional clarification

1. **Destroying horcruxes**: Horcruxes disappear upon destruction and cannot be destroyed multiple times. To destroy a horcrux, a wizard must be in the tile that the horcrux is present. At each turn a wizard can destroy at most one treasure. Horcrux cannot be destroyed by 2 wizards in the same turn, but doing so is legal and will result in the horcrux being destroyed and both wizards staying in place moving.
2. **Encounter with death eaters**: each wizard has a limited number of lives. If a wizard ends its turn on a tile with a death eater (after they both moved), the wizard will lose one life. However, the wizard can continue its journey – unless it reached 0 lives. In this case he dies and the game stops. The game ends in a loss even if Voldemort was killed in this turn.
3. **Death eaters Movement**: death eaters move only on passable regions. They move along a pre-determined path, which is provided in the input. They can wait or move one tile vertically or horizontally (no diagonal moves) per turn in the passable tiles. If a death eater has a path of A→B→C→D, instead of returning to point A after reaching point D, it will move to point C, and continue retracing the path in reverse, i.e., D→C→B→A, and then again A→B→C→D, and so forth. So, the path will be A→B→C→D→C→B→A→B…
4. **Wizards encountering Voldemort**: Before all horcruxes were destroyed, every wizard will die when encountering Voldemort. After they were all destroyed, only harry will not die if he encounters him. If Harry encounters him on the same turn that the last horcrux was destroyed, Harry will still die.

Goal

The overall goal is to ensure that every horcrux is destroyed and then killing Voldemort without any of the wizards dying while minimizing the number of turns.

Input and the task

As an input, you will get a dictionary that describes the initial environment.
"**map**": A list of lists of strings, where for example 'P' represents a passable tile, and 'I' represents impassable tiles.
"wizards": Contains the initial position of your wizards and the number of lives they have. The first coordinate specifies the row, counting from
"death_eaters": Details the paths for every death eater, representing their pre-determined movement pattern through the grid.

Full example of a dictionary:

```
{
    "map": [
        ['P', 'P', 'I', 'I'],
        ['P', 'P', 'P', 'P'],
        ['I', 'P', 'I', 'P'],
        ['P', 'P', 'V', 'I']
    ],
    "wizards": {"Harry Potter": ((2, 1), 1), "Hermione Granger": ((0, 0), 2)},
    "death_eaters": {'death_eater1': [(0, 1), (0, 0)]},
    "horcruxes": [(1, 3)],
},
```

This input is provided to the constructor of the class **HarryPotterProblem** as the variable initial. This variable is then used to create the root node of the search, hence you will need to transform it into your representation of the state before the **search.Problem.__init__(self, initial)** line.

Implementation Requirements:

You will need to implement the following functions in the **HarryPotterProblem** class:

1. **def actions(self, state)**:
    - Returns all available actions from a given state.
2. **def result(self, state, action)**:
    - Returns the next state, given a previous state and an action.
3. **def goal_test(self, state)**:
    - Returns 'True' if a given state is a goal, 'False' otherwise.
4. **def h(self, node)**:
    - Returns a heuristic estimate of a given node.

Note on State Representation:

You are free to choose your own representation of the state. However, it must be Hashable. We strongly suggest sticking to the built-in data types and not creating a new class for storing the state. Nevertheless, you might find unhashable data structures useful, so consider functions that transform your data structure to a hashable one and vice versa.

Evaluating your solution

Having implemented all the functions above, you may launch the A* search (already implemented in the code) by running check.py. Your code is expected to finish the task in 60 seconds.

Output

You may encounter one of the following outputs:

- A bug - self-explanatory
- (-2, -2, None) - No solution was found. This output can stem either from the insolvability of the problem or from a timeout (it took more than 60 seconds of real-time for the algorithm to finish).
- A solution of the form (number of turns, run time, list of actions taken)

Code handout

Code that you receive has 4 files:

1. ex1.py - the main file that you should modify, implements the specific problem.
2. check.py - the file that includes some wrappers and inputs, the file that you should run.
3. search.py - a file that has the implementation of the A* function.

4. 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.
Note: You may use any package that appears in the standard library, however, the exercise is built in a way that most packages will be useless. Any other packages are not allowed. Use the python 3.10 version for running the code.

Submission and grading

You are to submit only the file named ex1.py as python files (no zip, rar, etc.). We will run check.py with our own inputs, and your ex1.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:

- 85 points – Solving all the non-competitive problems under 60 seconds with the optimal solution.
- 25 points – competitive part. Computing the optimal plans (minimal number of turns) in under 60 seconds, for as many problems as possible. The grade will be relative to the other students in the class. The problems which will be tested might be bigger than in the non-competitive part.
- Notice, you can get more than 100 points in this HW. Scores above 100 will **not** be rounded down to 100.
- The submission is due on 18.12, at 23:59
- Submission in pairs/singles only.
- Write your ID numbers in the appropriate field ('ids' in ex1.py) as strings. If you submit alone, leave only one string.
- When grading, we check the validity of your solutions. Make sure not to change the names of the wizards.