# Homework 2 – Logical Inference

## Introduction

We continue in the world of Harry Potter. This time, Harry heard a rumor that one of the deathly hallows is hidden in one of the Gringotts bank vaults, and since such a tool can be useful in winning the war, he intends to break in and find the important artifact – **without** dying or getting caught in the way. However, the task will not be so easy, since Gringotts is very secretive about the structure of the vaults and is also known to use dragons and traps to stop any intruders.

Hermione Granger, a powerful wizard friend of Harry, offered Harry some help in the shape of a spell cloak cast on him. The spell cloak allows Harry to discover ahead of time where the traps are. When Harry stands next to a trap, he gets a strong smell of sulfur.

To achieve this most efficiently, you must make use of the algorithms shown in class, specifically using the logical inference methods.

### Environment

The environment is represented as a rectangular grid, with details provided in the "Input" section as a dictionary. Notice that in this task you will not get the entire map in the initial condition and will have to discover it while searching for the deathly hallow. Each cell within this grid symbolizes a different region in the world. Some cells might contain a vault, a dragon, or a trap (or any combination of these, but never dragon and vault).

The core objective is to navigate strategically to find the deathly hallow without Harry getting killed or caught.

Each action in this environment occurs in discrete turns, altering the state of the environment and the current position of Harry.

**Note:**

- The only thing that changes its location during this game is Harry. The vaults, traps and dragons do not change their positions.

### Actions

You assume control of Harry, which can perform various actions to navigate the world, destroy traps and avoid dragons:

1. **Move**: Harry can move up to one tile vertically or horizontally on the grid (he cannot move diagonally). The action's syntax is ("move", (row_idx, col_idx)). For example, if you want to move Harry to a tile (0,2), the correct syntax for this action is ("move", (0, 2)).
2. **Destroy trap:** Harry can destroy a trap if he is in a tile that is directly next to the trap. The syntax for this action is ("destroy", (row_idx,col_idx)). For example, if you want Harry to destroy a trap that is in (0,2), the correct syntax for this action is ("destroy", (0, 2)).
3. **Wait**: Harry can choose to wait, which does not change anything about its state or position. The syntax for this action is ("wait",).

4. **Collect:** Once Harry reaches a vault, he can try to collect the deathly hallow that lies within. The syntax for this action is (“collect”,). However, the deathly hallow might be in another vault – in this case the game continues. If the deathly hallow in indeed in the vault, the hallow is collected and the game stops.

Notice that each action is a tuple, even if its size is 1.

Each of the above actions is performed in turns or timestamps. After the action is performed, you are given an observation of the environment. The observations are used to expand your knowledge base about the environment, which you can (and should) use to guide your next actions.

**Note:**

- Ensure Harry does not step over the edge of the map.
- Ensure Harry does not step into a tile with a trap or a dragon, both will result in the game being lost.

## Observations

As stated above, after every action you perform you receive an observation about the environment. Harry can observe the following two things:

1. **Vault –** If Harry stands one square away from a vault (not diagonally), he can see that indeed there is a vault there (and exactly where it is). The syntax for this observation is (“vault”, (row_idx,col_idx)). For example, if Harry moves to a tile (0,2) and there is a vault in (0, 3), you will get (“vault”, (0, 3)).
2. **Dragon –** If Harry stands one square away from a dragon (not diagonally), he can see that indeed there is a dragon there (and exactly where it is). The syntax for this observation is (“dragon”, (row_idx,col_idx)). For example, if Harry moves to a tile (0,2) and there is a dragon in (0, 3), you will get (“dragon”, (0, 3)).
3. **Sulfur –** If Harry stands next to a trap, he can smell sulfur (but does not know where the trap is). The syntax for this observation is (“sulfur”,).

Notice that each observation is a tuple, even if its size is 1.

After every action you perform, you will receive a **list** of observations. The list may be empty.

## Goal

The overall goal is to collect the deathly hallow without Harry getting caught in a trap or stepping into a dragon tile. You are required to finish this task in at most 5 + 1.5 * circumference_of_map turns (regardless of actual time it takes you to make the calculations).

## Additional clarification

1. **Null actions**: Trying to destroy a trap that does not exist is a valid action and will result in no change in the board. Similarly, trying to collect the deathly hallow from a place that does not have it (a vault or any other tile) will result in no change in the board.

## Input and the task

You are required to implement the class “GringottsController”, which has the following two methods:

1. **Init –** initialize the controller. The input for this action will be three variables:
    a. **Map_shape –** the dimensions of the map.
    b. **Harry_loc –** the starting location of Harry.
    c. **Initial_observations –** the observations you receive from the starting location, in the format stated above.
2. **Get_next_action –** in this method you should implement the policy for choosing the next action. This method will be repeatedly called from the checker and is the only way you can progress in the game. The method should only return legal action, otherwise the code will fail. Every time this method is called you will receive observations from your current location. The input will be a single variable:
    a. **Observations –** the observations from your current location, in the format stated above.

Full example of an input for both methods (this example appears in the file ‘inputs.py’ you received):

```
EXAMPLE_INIT_INPUT = (  # This one corresponds to the first input example
    (3, 3),
    (0, 0),
    []
)

EXAMPLE_OBSERVATIONS = [  # This is the observation for the 4th input example when standing at (2, 1)
    ('vault', (2, 2)),
    ('dragon', (1, 1)),
    ('sulfur',)
]
```

## Evaluating your solution

Having implemented all the methods above, you may launch the checker.py file to check your results over the inputs. Your code should finish the init method in 60 seconds and the get_next_action method in 5 seconds every time they are called[1]. We encourage you to create more input and learn from them how to improve your code.

## Output

You may encounter one of the following outputs for every one of the inputs:

- A bug – self-explanatory.
- Timeout message – your code takes too long to run.
- Illegal action message – self-explanatory.
- A message stating how fast your code has solved the input (in number of turns).

## Code handout

Code that you receive has 4 files:

[1] The code will be checked on an i7 intel processor with 16GB of RAM. To ensure you stand in the time limits you can also check the time your code takes using the time library. The time given is significantly bigger than the time necessary to make most of the computations we tried on this exercise.

1. ex2.py – the main file that you should modify (has the controller class), implements the specific problem.
2. check.py – the file that includes some wrappers and inputs and the checker of your controller, 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 – the file that contains some examples of inputs. If you add inputs, it should be here.

**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 ex2.py as 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:

- 85 points – Solving all the non-competitive problems under 60 seconds plus five seconds per action.
- 25 points – competitive part. Solving extra inputs in the smallest possible number of turns (not necessarily optimal, just in less turns than others) for as many problems as possible. This 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. You should aim for a small number of turns even at the expense of longer time for every turn, but should not take more than the allowed amount of time.
- 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 15.1, 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.