**Project 3**

**Artificial Intelligence**

Spring 2024

*[Solutions to this assignment must be submitted via CANVAS prior to midnight on the due date which is 19 April 2024. Submissions up to one day late will be penalized 10% and a further 10% will be applied for the next day late. Submissions will not be accepted if more than two days later than the due date.]*

This project may be undertaken in pairs or individually. If working in a pair, state the ***names*** of the two people undertaking the project and the ***contributions*** that each has made. Only ONE submission should be made per group.

**Purpose**: To gain a thorough understanding of the working of a robot that is implemented as a reinforcement learning agent. The robot needs to navigate in a grid that contains obstacles. The robot can start in any position along the grid that is either not a position occupied by an obstacle or is not the destination. Both obstacles and the destination position are fixed and indicated in the grid below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| O1 |  |  | D | O2 |
|  |  |  |  |  |
|  |  | HH |  |  |
|  | S |  |  |  |
| O3 |  |  |  | O4 |

5

4

3

2

1

An example starting position for the robot

Hazard area with negative reward

1 2 3 4 5

**Figure 1**

**Environment Description:** The environment in which the robot navigates is a 5 by 5 grid. Obstacles are represented by O1, O2, O3, O4 and fixed in position as given in Figure 1. When the robot *collides with an obstacle it will need to return to the grid position that it occupied before the collision*. An episode consists of the robot moving from its starting position and then navigating to the destination point D which is fixed between episodes. The starting position of the robot varies randomly between episodes and can take any values (i.e., grid position) as long as it does not coincide with an obstacle position or the destination position.

There is an exit reward with positive value R1 (exact value specified later) associated with the destination D. The hazard position H is associated with a negative reward of magnitude R2 (exact value also specified later). Each state (other than the exit state, hazard, and obstacle states) have a live-in reward of value r.

The navigation rule (from any position on the grid) is as follows. A maximum of four actions, Up, Down, Left and Right are possible from any given position. Navigation in the intended direction occurs with 90% probability and movement in unintended directions (taken as a sum) adding up to 10%. Note that movements never take place opposite (at 180 degrees) to the intended direction. The discount factor

The rest of the parameters are as follows:

R1 lies in the range [10, 50, 100]

R2 lies in the range [-5, -50, -500]

r takes one of two values -5 and -1, with default value of -5.

Jot down any questions/doubts that you may have and feel free to ask me questions in class or in person. Together with your partner, ***work out a strategy before you start coding the solution in Python.*** Note that this project, unlike the previous two projects, has only one milestone and hence we expect that you on your own will carry on the good practice of designing a solution before implementing it.

Given the limited timeframe for the project, some simplifications have been applied.

Your task in this project is to implement the following requirements. The project has only ONE milestone (one submission) which has the following requirements. See [Tutorial 5 for Project 3](https://docs.google.com/document/d/1MklZmOudpjY6Kjwt2aQDyWFhTCrwAlvI/edit?usp=sharing&ouid=105068164970584867093&rtpof=true&sd=true) for a discussion on setting rewards and transition probabilities.

# Requirements

Your Python code in Colab should meet the following requirements. All the outputs required in R1-R4 must be displayed by your Python program. The outputs should also be included in the pdf report file that you hand in.

T1. Your task is to use the first value for each of R1, R2 and r (i.e., 10, -5, -5) parameters and generate the policy (let’s call it P1) by application of the Value Iteration (Bellman) algorithm. The policy P1 can be visualized in one of two ways: (a) Draw it with arrows showing the path in the same style as discussed in the lectures or if you prefer using (b) Populate each cell in the grid with numeric values produced after running the Value Iteration algorithm. **(30 marks)**

T2. This task requires no programming but requires you to reflect on the policy P1 you produced in T1 above. To evaluate a policy, two criteria are applied: (1): The *number of paths* from the starting position (which can be anywhere on the grid that satisfies the two constraints mentioned earlier) that lead to the destination (2): what *fraction of paths* that lead to the destination have the shortest possible path? **(10 marks)**

T3: This task also does not require programming but running further experiments on the code that you developed in T1. Run 3 further experiments as follows:

1. Use the following combination for the reward parameters: R1=50, R2=-50 and r=-5. Produce the new policy P2 and compare it with the policy you generated in T1 above. For the comparison make use of the two criteria mentioned in T2 above. **(5 marks)**
2. Use a combination of R1=100, R2=-500, r=-5 and produce a new policy P3. Compare it with the two policies you generated in T1 and T3 part 1 above. **(5 marks)**
3. Take the combination of R1 and R2 that yielded the best policy (from the ones generated in T1, T3 part 1 and T3 part 2 and now use r=-1. Produce the new policy P4 and compare with the current best policy (chosen from P1, P2 and P3). On the basis of the experimentation that you have done, which value of r out of -1 and -5 is best? How do you justify your answer?

**(5 marks)**

T4. Only applicable to those enrolled in 5210.

One limitation of the system implemented is that it is restricted to a single agent (robot). In practice many agents may be required to navigate at the same time while avoiding collisions with just walls but also colliding with each other. An example of this is an automated warehouse which uses multiple robots to service different customer orders in parallel with each other. The challenge here is to cope with obstacles that are mobile. The Bellman Value Iteration algorithm only accommodates a static environment.

Suggest a suitable modification (without an actual implementation) to Value Iteration that will enable it to be applied to this type of environment. Hint: Positions that are free of fixed obstacles now may be occupied by robots with a certain probability. Assume that you have a formula to compute this probability. How will this help with the Value Iteration algorithm? Your explanation needs to be at most 2 paragraphs in length but must be convincing. You may use pseudo code to present your answer if you so wish.

**(15 marks)**

Notes:

Produce ONE pdf document that contains Python code. If you submit an image version of your pdf code file, it will not be graded.

Also submit a separate pdf (report dpf) that contains answers to the questions. Do NOT bury answers to questions as comments in your code.

Also submit a publicly accessible link to your Colab code file.

End of project specification