Assignment #4: Stochastic Robot Navigation (Q-Learning)
CSCI 364 Fall 2021 Oberlin College

Background

In HW3, we considered a grid world example where an agent moves around in a 3x3 grid and puts out wildfires. Grid worlds are a common type of problem that we use for the initial testing and evaluation of our decision-making algorithms because they are small enough to conceptualize but complex enough to include a variety of states/actions/rewards/etc.

In this homework assignment, we consider another grid world example where a robot agent moves around from a start state S (in the bottom left corner) and attempts to reach a goal state G (in the top right corner). The agent earns a small cost for each movement (-0.1), a positive reward (+5) for reaching the goal G, and a negative reward (-5) for accidentally falling into a pit P right below the goal. Unfortunately, the robot's motors are not perfect, meaning that sometimes it ends up in a different location from where it intended to go. Thus, the next state transitions are stochastic, meaning we cannot simply solve this problem using Search algorithms (e.g., A* Search). A summary of the world is given in Chapter 17 of the (optional) textbook.

														G
														P
		X	**X**			**X**								
		X	**X**			**X**								
S														

Figure 1: The Robot's Grid World
S = Start Location G = Goal Location P = Pit Location X = Obstacle

Assignment

Your assignment is to write a program in **Python or Java** that uses Q-Learning to enable the robot to learn how to navigate from its starting location in the bottom left corner to the goal location in the top right corner.

You can get started on the assignment by following this link:
https://classroom.github.com/a/V1Ac39yO

Please call your program either learner.py (in Python) or Learner.java (in Java), so that it is executed with either:

```
python3 learner.py <alphaValue> <epsilonValue>
```

or

```
java Learner <alphaValue> <epsilonValue>
```

where `<alphaValue>` is a decimal number between 0 and 1 to use for the learning rate α in the Q-Learning update rule and `<epsilonValue>` is a decimal number between 0 and 1 to use for the exploration rate ϵ in the ϵ-greedy exploration-exploitation algorithm. **Note**: if the user passes in a value of 0 for `<alphaValue>`, then your agent should instead use $\alpha = 1 \,/\, n(s, a)$ where $n(s, a)$ tracks the number of times the agent has thus far chosen action a in state s. For this assignment, fix $\gamma = 0.99$.

I've provided you with a `Grid` class that represents the grid above. The only methods you should need to use from this class are:

1) `generateStartState()`, which returns the starting state for the robot
2) `generateNextState(state, action)`, which takes in the current `state` and a chosen `action` as parameters and returns a random next state for the agent
3) `generateReward(state, action)`, which also takes in the current `state` and a chosen `action` as parameters and returns a reward for the agent (to use for reinforcing that action choice)

Also, you will want to use the `ABSORBING_STATE` in `Grid` to know when your agent has finished an episode – once this is the **current state**, your agent is done! The possible actions the agent can take are represented by the strings `"up"`, `"down"`, `"left"`, and `"right"` in the `actions` instance variable of the `Grid` class. You should use ϵ-greedy to choose your actions.

An episode in this environment starts in the start state and continues as the agent keeps choosing actions until the current state is the `Grid.ABSORBING_STATE` (which occurs after the agent either reaches the goal or the pit). For each episode, you should keep track of the cumulative reward earned by the agent for that episode:

$$\sum_{t=0}^{H} r_t = r_0 + r_1 + r_2 + \cdots + r_H$$

where it takes *H* steps to reach `ABSORBING_STATE` (*H* might vary from episode to episode). You should run the agent for 100 total episodes each time your program is executed. You should not reset the Q-Table [nor the $n(s, a)$ values] between these 100 episodes so that the agent can remember what it has learned in previous episodes to help it do better in the current episode.

Experiments

After your program is implemented, you will conduct two sets of experiments to see how the learning rate α and exploration rate ϵ affect the agent's behavior and performance.

For the first experiment, use an exploration rate $\epsilon = 0.10$. Then, use five different learning rates $\alpha = 0.50, 0.25, 0$ [for the decaying learning rate $1 \,/\, n(s, a)$], and two values of your choosing. Record the cumulative rewards earned by the agent across all 100 episodes for each learning rate α and plot them on a line chart where the x-axis is the episode number, the y-axis is the

cumulative rewards and you have five lines (one for each α value). You can use any program you want to create these line charts; some options include Excel, R, and Python with matplotlib. Please save the line chart as an image file and upload it to your GitHub repository.

For the second experiment, use the best learning rate from your first experiment. Then, use five different exploration rates $\epsilon = 0.01, 0.05, 0.10$ and two of your choice. Once again, record the cumulative rewards earned by the agent across all 100 episodes for each exploration rate ϵ and plot them on a line chart where the x-axis is the episode number, the y-axis is the cumulative rewards and you have five lines (one for each ϵ value). Again, save the line chart as an image file and upload it to your GitHub repository.

Within a README file, you should include:

1. A paragraph (5-10 sentences) comparing the results in your line chart from the first experiment. What trends did you observe about the agent's learning? How did the agent's performance change as you varied the learning rate α? Which learning rate did you find led to the best performance? Please make sure to list the five learning rates compared in your experiment.
2. A paragraph (5-10 sentences) comparing the results in your line chart from the second experiment. What trends did you observe about the agent's learning? How did the agent's performance change as you varied the exploration rate ϵ? Which exploration rate did you find led to the best performance? Please make sure to list the five exploration rates compared in your experiment.
3. Based on your results from both experiments, what advice would you offer someone new to reinforcement learning about how to choose appropriate values for α and ϵ?
4. A short paragraph describing your experience during the assignment (what did you enjoy, what was difficult, etc.)
5. An estimation of how much time you spent on the assignment, and
6. An affirmation that you adhered to the honor code

Please remember to commit your solution to your repository on GitHub. You do not need to wait until you are done with the assignment; it is good practice to do so not only after each coding session, but maybe after hitting important milestones or solving bugs during a coding session. ***Make sure to document your code***, explaining how you implemented the data structures and algorithms for Q-Learning.

Honor Code

Each student is allowed to complete this assignment with a single partner or alone. Students are encouraged to collaborate with one another to discuss the abstract design and processes of their implementations. For example, please feel free to discuss the pseudocode for the Q-Learning and ϵ-greedy algorithms to help each other work through issues understanding exactly how the algorithms work. At the same, since this is a pair-programming assignment, no code can be shared between groups, nor can students look at code from someone other than your partner.

Grading Rubric

Your solution and README will be graded based on the following rubric:

Program flow: /10 points
Q-Learning implementation: /25 points
ϵ-greedy implementation: /10 points
Correct learned behaviors: /10 points
Line charts from your experiments: /15 points
Answers to README questions: /25 points
Appropriate code documentation: /5 points

By appropriate code documentation, I mean including a header comment at the top of each file explaining what the file provides, as well as at least one comment per function explaining the purpose of the function.