Collaboration policy for homework: This homework is to be done by yourself. Please do not collaborate with others, look for answers online, or post homework problems or solutions to websites or discord servers. You may discuss the homework concepts at a high level with other students. **We encourage you to come to office hours for help.**

Submit the following files to Canvas:

`hw3_solution.pdf` - PDF including your program output, and also answers questions in problems 2 and 3.

`mdp_vi.py`

`mdp_pi.py`

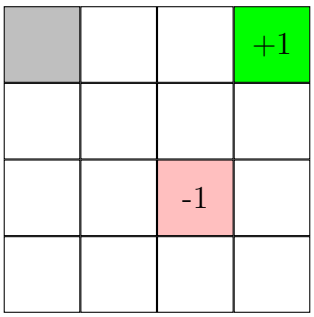
`bandit_agent.py`

`README.txt` - If necessary, extra instructions for using your code, any assumptions you are making

Submit your non-programming answers as a pdf. You may use any method to create your pdf that you want, it does not need to be Latex. However, if you handwrite your answers, we reserve the right to take off points if we cannot read your handwriting.

Note: This homework asks you to write Python code "from scratch". What I mean by this is that I expect you to not import any machine-learning libraries. Basic libraries like `sys`, `collections`, `random` are okay.

For problems 1 and 2, consider an environment similar to Grid world (from the book and lecture) that is fully observable, with grid locations, with a nondeterministic transition model and two terminal states. The possible actions from each location are Up, $Left$, $Down$, and $Right$. Running into a wall results in the agent stopping in the location where it hit the wall. There is one location $[1, 4]$ that is blocked off. The two terminal states have values $+1$ and -1. The transition model is as follows: There is a 80% chance that the agent properly executes the action that it chooses (that is, it moves 1 space in the direction chosen.) There is a 10% chance that it moves perpendicular to the direction chosen. Suppose that there is a constant reward value r that applies for each nonterminal state that we reach.



Problem 1. (35 points)[Programming] Use the Value iteration algorithm to compute utilities U and an optimal policy π^*. Write python code (from scratch) to implement the algorithm in a file named `mdp_vi.py` and run your code for the three different reward values:

$r = -2$, $r = -0.2$, and $r = -0.01$. Use a discount factor $\gamma = 0.9$. The code should take as input a command line argument that specifies r, and output the states, their utilities, and the policy for each state.

In your pdf, include examples of running your program and its outputs for the requested values of r.

An example of running the code might look like:

```
$ python3 mdp_vi.py -0.5
State pi* U
(1, 1) U 0.1234
(1, 2) L 0.2345
(1, 3) D 0.3456
...
```

(Note: These are not the correct policy choices or utility values for $r = -0.5$.)

Problem 2. (35 points)[Programming]

(a) (30 points) Use the Policy Iteration algorithm to compute utilities U and an optimal policy π^*. Write python code (from scratch) to implement the algorithm in a file name `mdp_pi.py` and run your code for the three different reward values: $r = -2$, $r = -0.2$, and $r = -0.01$. Use a discount factor $\gamma = 0.9$. The code should take as input a command line argument that specifies r, and output the states, their utilities, and the policy for each state.

In your pdf, include examples of running your program and its outputs for the requested values of r.

(b) (5 points) In your pdf, answer the question: How did you handle policy evaluation? (As mentioned in lecture and in the book, there are multiple approaches. This problem is small enough that multiple choices will work, but let us know which one you went with.)

An example of running the code might look like:

```
$ python3 mdp_pi.py -0.5
State pi* U
(1, 1) U 0.1234
(1, 2) L 0.2345
(1, 3) D 0.3456
...
```

(Note: These are not the correct policy choices or utility values for $r = -0.5$.)

Problem 3. (30 points)[Programming] Download my `bandit.py` module that defines a class for an n-armed bandit.

(a) (20 points) Using this module, write an agent (either ϵ-greedy, UCB, or Thompson sampling) that will attempt to minimize regret. For simplicity, use a hard-coded setup with 5 arms of the bandit. Call your module `bandit_agent.py`. Have it take as input a command-line argument that specifies the number of iterations to run for and a random seed to initialize the bandit object's random number generators. Your agent can assume that the bandit arms do have a gaussian distribution.

After n iterations, have the program output the total reward and estimated regret achieved by the agent.

In your pdf, include examples of running your program and its outputs for $n = 100$ and $n = 1000$ for the same random seed.

An example of running the code might look like:

```
$ python3 bandit_agent.py 1000 5678
With seed 5678, after 1000 iterations, total reward is 105342.52, estimated
regret is 52642.25.
```

(Note: These are not necessarily the correct results for 1000 iterations of seed 5678.)

(b) (10 points) In your writeup, answer the following questions:

- Which agent type did you use?
- If ϵ-greedy, what is your function for ϵ as iterations increase?
- If UCB, what $g(n)$ function did you use?
- If Thompson sampling, how exactly are you generating your samples?
- How would you have changed things if you did not know that the bandit arms have a gaussian distribution for their rewards?