Assignment 3

Introduction

- Download the code for this assignment and then unzip the archive.
- This assignment uses Python 3. Do not use python 2.
    - We have tested the assignment with Python 3.11.4.
- You can work on the assignment using your favourite python editor. We recommend VSCode.
- Post any questions or issues with this assignment to our discussion forum.

This assignment uses auto-grading for all problems. For the first question, we rely on the random number generator generating the same random sequence. Same as in Assignment 2. This time, we will be using the random.choices() function as follows.

```
import random
from collections import Counter
seed = 2
if seed!=-1:
    random.seed(seed, version=1)
n = 0.1 #noise
a = 'N' #intended action
d = {'N':['N', 'E', 'W'], 'E':['E', 'S', 'N'], 'S':['S', 'W', 'E'], 'W':['W', 'N', 'S']}
l = []
for _ in range(100000):
    l += random.choices(population=d[a], weights=[1 - n*2, n, n])[0]
print(Counter(l).keys()) # equals to list(set(words))
print(Counter(l).values()) # counts the elements' frequency
print(l[:5])
```

This will give the following output.

```
dict_keys(['W', 'N', 'E'])
dict_values([10025, 80059, 9916])
['W', 'W', 'N', 'N', 'E']
```

Note that we obtain ~80% intended actions and 10% unintended actions here. Make sure that you understand the output and that you can reproduce it on your machine before proceeding.

Problem 1: MDP Episode

In this part of the assignment, we will play an episode in an MDP by following a given policy. Consider the first test case of problem 1 (available in the file `test_cases/p1/1.prob`).

```
seed: 2
noise: 0.1
livingReward: -0.05
grid:
    _    _    _    1
    _    #    _   -1
    S    _    _    _
policy:
    E    E    E exit
    N    #    N exit
```

```
    N    W    N    W
```

The first part of this file specifies an MDP. `S` is the start state with four available actions (`N`, `E`, `S`, `W`), `_` is an ordinary state with the same four available actions and `1`,`-1` are states where the only available action is `exit` and the reward is `1` and `-1` respectively. The reward for action in other states is `-0.05`. `#` is a wall.
Actions are not deterministic in this environment. In this case with `noise = 0.1`, we are successfully acting 80% of the time, and 20% of the time we will act perpendicular to the intended direction with equal probability, i.e. 10%, for each unintended direction. If the agent attempts to move into a wall, the agent will stay in the same position. Note that this MDP is identical to the example we covered extensively in class.

The second part of this file specifies the policy to be executed.

As usual, your first task is to implement the parsing of this grid MDP in the function `read_grid_mdp_problem_p1(file_path)` of the file `parse.py`. You may use any appropriate data structure.

Next, you should implement running the episode in the function `play_episode(problem)` in the file `p1.py`.

Below is the expected output. Note that we always use exactly 5 characters for the output of a single grid and that the last line does not contain a new line.

```
Start state:
    _    _    _    1
    _    #    _   -1
    P    _    _    _
Cumulative reward sum: 0.0
-----------------------------------------
Taking action: W (intended: N)
Reward received: -0.05
New state:
    _    _    _    1
    _    #    _   -1
    P    _    _    _
Cumulative reward sum: -0.05
-----------------------------------------
Taking action: W (intended: N)
Reward received: -0.05
New state:
    _    _    _    1
    _    #    _   -1
    P    _    _    _
Cumulative reward sum: -0.1
-----------------------------------------
Taking action: N (intended: N)
Reward received: -0.05
New state:
    _    _    _    1
    P    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.15
-----------------------------------------
Taking action: N (intended: N)
```

```
Reward received: -0.05
New state:
    P    _    _    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.2
-----------------------------------------
Taking action: S (intended: E)
Reward received: -0.05
New state:
    _    _    _    1
    P    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.25
-----------------------------------------
Taking action: N (intended: N)
Reward received: -0.05
New state:
    P    _    _    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.3
-----------------------------------------
Taking action: E (intended: E)
Reward received: -0.05
New state:
    _    P    _    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.35
-----------------------------------------
Taking action: E (intended: E)
Reward received: -0.05
New state:
    _    _    P    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.4
-----------------------------------------
Taking action: E (intended: E)
Reward received: -0.05
New state:
    _    _    _    P
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: -0.45
-----------------------------------------
Taking action: exit (intended: exit)
Reward received: 1.0
New state:
    _    _    _    1
    _    #    _   -1
    S    _    _    _
Cumulative reward sum: 0.55
```

As you can see, in this question we don't use any discount factor. We will introduce that in the next question. You can also try some of the other test cases such as `test_cases/p1/8.prob`.

```
seed: 42
noise: 0.2
livingReward: -1
```

```
grid:
    #   10    #
 -100    _ -100
 -100    _ -100
 -100    _ -100
 -100    _ -100
 -100    S -100
    #    1    #
policy:
    # exit    #
 exit    N exit
 exit    N exit
 exit    N exit
 exit    N exit
 exit    N exit
    # exit    #
```

With correct implementation, you should be able to pass all test cases.

```
(base) scdirk@Dirks-Air a3 % python p1.py -8
Grading Problem 1 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
----------> Test case 8 PASSED <----------
```

Problem 2: Policy Evaluation

In problem 2 you will implement policy evaluation as follows

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

This time we have discounting and we also introduce a new variable for the number of iterations. Here is the first test case.

```
discount: 0.9
noise: 0.1
livingReward: 0
iterations: 10
grid:
  -10  100  -10
  -10    _  -10
  -10    _  -10
  -10    S  -10
policy:
  exit exit exit
  exit    N exit
  exit    N exit
  exit    N exit
```

Note that there is no randomness involved this time and that we use discounting.

As usual, your first task is to implement the parsing of this grid MDP in the function `read_grid_mdp_problem_p2(file_path)` of the file `parse.py`. You may use any appropriate data structure.

Next, you implement value iteration for policy evaluation as discussed in class. Your `policy_evaluation(problem)` function in `p2.py` should return the evolution of values as follows.

```
V^pi_k=0
|   0.00||   0.00||   0.00|
|   0.00||   0.00||   0.00|
|   0.00||   0.00||   0.00|
|   0.00||   0.00||   0.00|
V^pi_k=1
| -10.00|| 100.00|| -10.00|
| -10.00||   0.00|| -10.00|
| -10.00||   0.00|| -10.00|
| -10.00||   0.00|| -10.00|
V^pi_k=2
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  -1.80|| -10.00|
| -10.00||  -1.80|| -10.00|
V^pi_k=3
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  -3.10|| -10.00|
V^pi_k=4
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=5
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=6
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=7
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=8
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
V^pi_k=9
| -10.00|| 100.00|| -10.00|
| -10.00||  70.20|| -10.00|
| -10.00||  48.74|| -10.00|
| -10.00||  33.30|| -10.00|
```

This example should look familiar. We have covered it in chapter 2 of our lecture slides.

Hint: The output of an individual floating point value `v` was done as follows

```
return_value += '|{:7.2f}|'.format(v)
```

Finally, check the correctness of your implementation via

```
(base) scdirk@Dirks-Air a3 % python p2.py -7
Grading Problem 2 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
```

Problem 3: Value Iteration

Now it is time to implement value iteration.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

This time, the provided problems do not include policies:

```
discount: 1
noise: 0.1
livingReward: -0.1
iterations: 10
grid:
    _    _    _    1
    _    #    _   -1
    S    _    _    _
```

As usual, load the problem definition in the function `read_grid_mdp_problem_p3(file_path)` of the file `parse.py`.

Next, implement value_iteration(problem) in p3.py such that it returns the following string for the first test case. Note that this is still a non-deterministic environment as before with the same stochastic motion.

```
V_k=0
|   0.00||   0.00||   0.00||   0.00|
|   0.00|| ##### ||   0.00||   0.00|
|   0.00||   0.00||   0.00||   0.00|
V_k=1
|  -0.10||  -0.10||  -0.10||   1.00|
|  -0.10|| ##### ||  -0.10||  -1.00|
|  -0.10||  -0.10||  -0.10||  -0.10|
pi_k=1
| N || N || N || x |
| N || # || N || x |
| N || N || N || N |
V_k=2
|  -0.20||  -0.20||   0.68||   1.00|
```

```
|  -0.20|| ##### ||  -0.20||  -1.00|
|  -0.20||  -0.20||  -0.20||  -0.20|
pi_k=2
| N || N || E || x |
| N || # || W || x |
| N || N || N || S |
V_k=3
|  -0.30||   0.40||   0.75||   1.00|
|  -0.30|| ##### ||   0.32||  -1.00|
|  -0.30||  -0.30||  -0.30||  -0.30|
pi_k=3
| N || E || E || x |
| N || # || N || x |
| N || N || N || S |
V_k=4
|   0.16||   0.58||   0.81||   1.00|
|  -0.40|| ##### ||   0.43||  -1.00|
|  -0.40||  -0.40||   0.10||  -0.40|
pi_k=4
| E || E || E || x |
| N || # || N || x |
| N || N || N || S |
V_k=5
|   0.34||   0.66||   0.82||   1.00|
|  -0.05|| ##### ||   0.49||  -1.00|
|  -0.50||  -0.10||   0.16||  -0.16|
pi_k=5
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
V_k=6
|   0.46||   0.69||   0.83||   1.00|
|   0.16|| ##### ||   0.51||  -1.00|
|  -0.20||   0.01||   0.26||  -0.08|
pi_k=6
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
V_k=7
|   0.52||   0.70||   0.83||   1.00|
|   0.30|| ##### ||   0.52||  -1.00|
|   0.01||   0.11||   0.30||   0.00|
pi_k=7
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
V_k=8
|   0.54||   0.71||   0.83||   1.00|
|   0.37|| ##### ||   0.52||  -1.00|
|   0.15||   0.16||   0.32||   0.04|
pi_k=8
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
V_k=9
|   0.56||   0.71||   0.84||   1.00|
|   0.41|| ##### ||   0.52||  -1.00|
|   0.23||   0.19||   0.34||   0.06|
pi_k=9
| E || E || E || x |
| N || # || N || x |
| N || E || N || W |
```

Once you are done. Check if you can pass all test cases as follows.

```
(base) scdirk@Dirks-Air a3 % python p3.py -4
Grading Problem 3 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
```

Problem 4: Q-Value TD Learning

In the final problem of this assignment you will implement temporal difference learning of Q values using the following equation from our lecture.

$$sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$
$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha)\,[sample]$$

To force exploration you will implement epsilon greedy with decay. You will have to determine a suitable starting value for the learning rate and epsilon. Also you will need a way to adjust the learning rate so that all q values will converge. You will also have to determine when convergence has occurred.

As usual, load the problem definition in the function `read_grid_mdp_problem_p4(file_path)` of the file `parse.py` and then implement the `td_learning(problem)` function in `p4.py`.

There are a number of example test cases provided. The first testcase comes with a solution so that you understand the output requirements. For the other testcases there are no solutions and you check if convergence has occurred! We will manually mark you code based on additional test cases with various values for the discount and reward.

As in the previous assignments, you will have to write a report. For this question, your report should include the converged policy and the converged Q values for all states and test cases. You should use the following format for the converged Q values. We recommend using a debug flag to enable the output of converged Q values.

```
|N 0.826  E 0.852  S 0.796  W 0.818||N 0.864  E 0.894  S 0.863  W 0.830||N 0.904  E 0.932  S 0.707  W 0.847||x 1.000                          |
|N 0.815  E 0.784  S 0.751  W 0.784|| ################################ ||N 0.680  E-0.669  S 0.494  W 0.679||x-1.000                          |
|N 0.772  E 0.716  S 0.738  W 0.747||N 0.705  E 0.673  S 0.705  W 0.735||N 0.637  E 0.487  S 0.648  W 0.696||N-0.717  E 0.289  S 0.467  W 0.473|
```

Congratulations you have completed this assignment.

Short Written Report

Write a short (at most four A4 pages) written report in PDF format explaining which problem were completed and where you have struggled. Make sure everything is well documented and the code is well organized and easy to read. A portion of the marks will be allocated to well-maintained documented and easy to read code.

In the report, you should explain the various design choices that you have made regarding state representation and evaluation functions etc. What was the impact of your design choices?

Finally, write down the approximate number of hours you have spent per questions for this assignment.

Submission

To submit your assignment to Moodle, `*.zip` the following files ONLY:

- `p1.py`
- `p2.py`
- `p3.py`
- `p4.py`
- `parse.py`
- `report.pdf`

Do not zip any other files. Use the `*.zip` file format. Name the file `UID.zip`, where UID is your 10 digit university number. Make sure that you have submitted the correct files and named all files correctly. We will deduct up to 5% for files with incorrectly file names. We will not allow late submissions. Submission of incorrect files may result in 0 marks.

Check that you have submitted the correct files before the deadline.