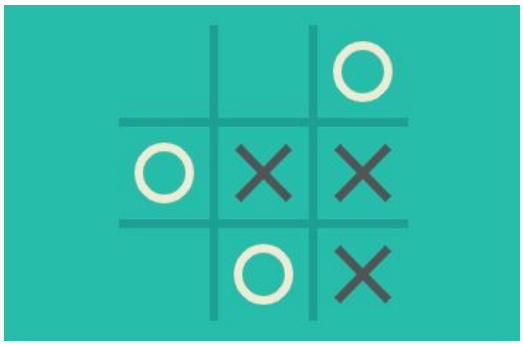
Homework Programming Assignment 5: Reinforcement Learning

Due Time: November 18, 2019, 3:00PM



Introduction

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. This homework programming assignment consists of 2 parts. In the first part, you are tasked with solving a simple MDP using value iteration algorithm. In the second part, you are asked to train a tic-tac-toe player using Q-learning.

Please use ***Python 3.6*** *or* ***Python 3.7*** to implement your homework assignment.

1. Homework Description

1.1 Task 1: 2D Maze (30 pts)

Task Description

Your agent is trapped in a 2D Maze, and your are helping your agent to determine the policy to get out of the maze.

There are obstacles in the maze that your agent should avoid. If your agent crashes into an obstacle, 100 Health Points of your agent will be deducted (HP -100). Each time your agent moves, 1 HP will also be deducted (HP -1). When your agent arrives at the destination, your agent will receive 100 HP (HP +100). *(Hint: think of HP as reward)*

However, there is also uncertainty in the agent's navigation. The agent will go in the correct direction 70% of the time (10% in each other direction, including borders).

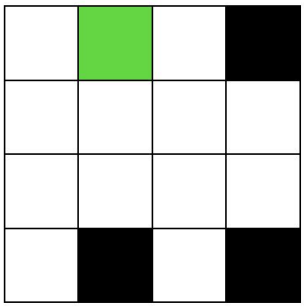
The goal is to compute the best policy given the maze using ***value iteration***.

Example

Coordinate System

0,0	1,0	2,0	3,0
0,1	1,1	2,1	3,1
0,2	1,2	2,2	3,2
0,3	1,3	2,3	3,3

Maze



HP (implied in the inputs)

-1	99	-1	-101
-1	-1	-1	-1
-1	-1	-1	-1
-1	-101	-1	-101

Utilities (State-Value)

82.42382	99.0	66.80476	-67.78149
69.08451	77.65930	61.36781	37.03885
55.01046	50.54507	45.53575	23.85361
33.20321	-71.41770	13.12707	-103.41539

Policy

>	.	<	x
^	^	<	<
^	^	^	<
^	x	^	x

- Policy:
  - The lowercase letter `x` represents an obstacle
  - Period `.` represents destination
  - `^ v < >` represents NORTH, SOUTH, WEST, EAST respectively
- Utilities here are just for demonstration, do NOT round in your algorithm
- It is guaranteed that there is NO tie in all test cases

Value Iteration

Recall the bellman equation to calculate the expected utility starting in `s` and acting optimally:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s') \ .$$

And the value iteration algorithm:

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
rewards $R(s)$, discount γ
ϵ, the maximum error allowed in the utility of any state
local variables: U, U', vectors of utilities for states in S, initially zero
δ, the maximum change in the utility of any state in an iteration

repeat
$U \leftarrow U'$; $\delta \leftarrow 0$
for each state s **in** S **do**
$U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \epsilon(1-\gamma)/\gamma$
return U

Parameter Selection

Please use exact same values

- Discount factor gamma = `0.9`
- Maximum error allowed epsilon = `0.1`
- Initial conditions `U0` = `0.0`

Input and Output Format

The input file will be formatted as follows:

```
grid_size
number_of_obstacles
followed by number_of_obstacles lines of x, y #location of obstacles
x, y #location of destination
```

For the above mentioned example, the input is:

```
4
3
1,3
3,3
3,0
1,0
```

And your output should be :

```
>.<x
^^<<
^^^<
^x^x
```

NO blank line at the end of the file, `\n` for line break

Grade
You are given blackbox51.txt, blackbox52.txt for development, and your code will only be graded with hidden test cases blackbox53.txt and blackbox54.txt

The program should be run in the following way:

```
python3 GridMDP.py input_file output_file
=> output_file
```

For example, when we grade with hidden blackbox53, we will run:

```
python3 GridMDP.py blackbox53.txt blackbox53-policy.txt
=> blackbox53-policy.txt
```

The score will be calculated as:

Correctness (Exact Same)	blackbox53 (10pts)	blackbox54 (20pts)
Correct	10	20
Wrong	0	0

Hint: Linux `diff` command or python `filecmp.cmp()` function can be used to compare two files

1.2 Task 2: Tic-Tac-Toe (70 pts)

Task Description

In this part, you are asked to solve a real-life problem Tic-tac-toe. If you don't remember it, familiarize yourself with the rules (https://en.wikipedia.org/wiki/Tic-tac-toe)

You are given 6 files:

- `Board.py`: tic-tac-toe board, 3 by 3 gridx

```
[[1 1 0]
 [0 0 0]    ---->
 [0 2 2]]
```

- `RandomPlayer.pyc, SmartPlayer.pyc, PerfectPlayer.pyc` (cpython-3.6): think of them as 3 blackboxes. You do not need to know how things work inside of each player, but it may be helpful to know how they behave. See next section for more details
- `QLearner.py`: Q-Learning Player to be implemented

- `TicTacToe.py`: where all players will be called to play tic-tac-toe games and where your QLearner will be trained and tested, i.e. grading script, simply run `python3 TicTacToe.py`

Your task is to complete `QLearner.py`. In `QLearner.py`, method `move()` and `learn()` must be implemented, and the number of games `GAME_NUM` needed for training the Q-Learner must be set. Any other helper functions can be added inside the `QLearner` class.

To see how these two methods and the variable `GAME_NUM` will be used, please see `TicTacToe.py` for more details.

You are not supposed to modify `Board.py, TicTacToe.py` as well as other `Players`. The only python file you need to edit is `QLearner.py`.

Opponents

- **RandomPlayer:** moves randomly
- **SmartPlayer:** somehow better than RandomPlayer, but cannot beat PerfectPlayer
- **PerfectPlayer:** never lose

If we let these 3 players play against each other, game results will look like this:

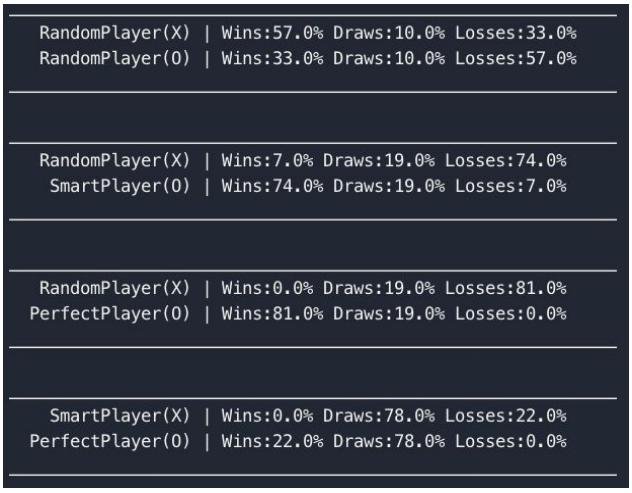


Figure 2.1

Q-Learning

Recall the formula:

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

You are encouraged to make any improvement to your Q-Learner in terms of speed and performance to get prepared for the upcoming competition.

Hint: The rewards will only be assigned for the last action taken by the agent

Parameters Selection

You are free to choose values for all parameters, i.e. reward for `WIN,DRAW,LOSE,` learning rate, discount factor and initial conditions.

Grade

This time, the grading script (`TicTacToe.py`) is given to you. After execution, the game results will be printed out. See Figure 2.2

```
~/INF552/PA5 python3 TicTacToe.py
--------------------------------------------------------------------------------
      QLearner(X) | Wins:99.0% Draws:1.0% Losses:0.0%
  RandomPlayer(O) | Wins:0.0% Draws:1.0% Losses:99.0%
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
  RandomPlayer(X) | Wins:0.0% Draws:9.4% Losses:90.6%
      QLearner(O) | Wins:90.6% Draws:9.4% Losses:0.0%
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
      QLearner(X) | Wins:43.0% Draws:57.0% Losses:0.0%
   SmartPlayer(O) | Wins:0.0% Draws:57.0% Losses:43.0%
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
   SmartPlayer(X) | Wins:0.0% Draws:74.0% Losses:26.0%
      QLearner(O) | Wins:26.0% Draws:74.0% Losses:0.0%
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
      QLearner(X) | Wins:0.0% Draws:100.0% Losses:0.0%
 PerfectPlayer(O) | Wins:0.0% Draws:100.0% Losses:0.0%
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
 PerfectPlayer(X) | Wins:0.0% Draws:100.0% Losses:0.0%
      QLearner(O) | Wins:0.0% Draws:100.0% Losses:0.0%
--------------------------------------------------------------------------------

Summary:
--------------------------------------------------------------------------------
QLearner VS  RandomPlayer |  Win/Draw Rate = 100.0%
QLearner VS   SmartPlayer |  Win/Draw Rate = 100.0%
QLearner VS PerfectPlayer |  Win/Draw Rate = 100.0%
--------------------------------------------------------------------------------

Task 2 Grade: 70 / 70
```

Figure 2.2

In summary section, you can see the Win/Draw rate against each opponent as well as the final grade for task 2. The task 2 score is calculated as:

Q-Learner Win+Draw Rate	**Opponents**		
	RandomPlayer (25pts)	**SmartPlayer** (25pts)	**PerfectPlayer** (20pts)
>= 95%	25	25	20
< 95% and >= 85%	15 * Rate	15 * Rate	10 * Rate
< 85%	0	0	0

If your player is not based on Q-Learning (e.g. rule-based), you will lose **ALL** points for task 2.

In your implementation, ***please do not use any existing machine learning library call***. You must implement the algorithm yourself. Please develop your code yourself and do not copy from other students or from the Internet.

Your code will be autograded for technical correctness. Please name your file correctly, or you will wreak havoc on the autograder. **The maximum running time is 3 minutes for each task.**

2. Submission:

2.1 Submit your code to Vocareum

- Submit `GridMDP.py`, `QLearner.py` on ***Vocareum***
- The program will terminate and fail if it exceeds the ***3 minutes*** time limit.
- After the submission script/grading script finishes, you can view your submission report immediately to see if your code works, while the grading report will be released after the deadline of the homework.
- You don't have to keep the page open while the scripts are running.

You do NOT need to submit your code or report to Blackboard. Your score on **Vocareum** will be your final score. The submission instructions and examples are provided in **Appendix A.**

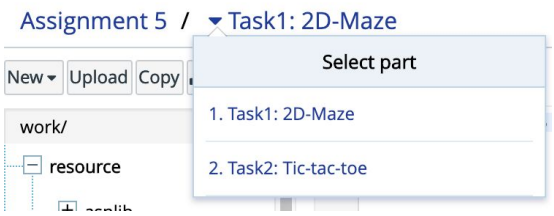
Appendix A: Vocareum Submission

This time your total score on Vocareum will be your final grade for assignment 5, so it is important for you to fully understand how to submit your work and how final grades will be generated.

There are 2 parts on Vocareum as well. Two parts are independent so that you can work on and test them separately.

Parts	Due date	Max points
Task1: 2D-Maze	Nov 18 2019 14:59:59 PST	30
Task2: Tic-tac-toe	Nov 18 2019 14:59:59 PST	70

Switch Task



Task 1

As usual, upload your `GridMDP.py` to the work directory.

After you submit the code, you will be able to see the submission report. The `submission script` will only check `blackbox51` and `blackbox52`.

Passed means you are correct on that blackbox

```
================================================
blackbox51 passed
================================================
================================================
blackbox52 passed
================================================
```

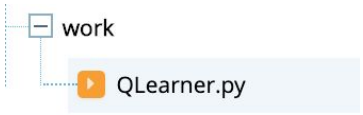
Otherwise, you will see **Failed**

```
================================================
blackbox51 failed
================================================
================================================
blackbox52 failed
================================================
```

After the due time, the `grading script` will check your code using `blackbox53` and `blackbox54` to generate score for task1.

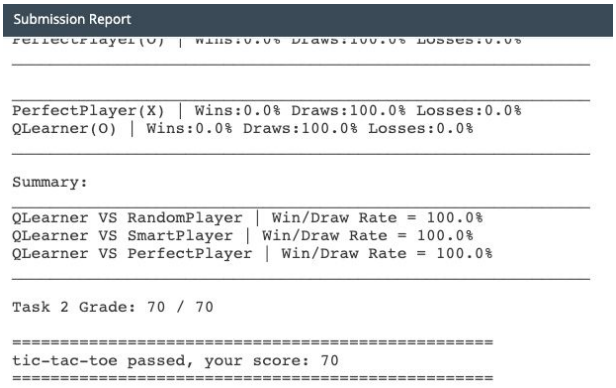
Task 2

Only `QLearner.py` needs to be submitted.



All starter code (including `Board.py`, `*Player.py` and `TicTacToe.py`) are the **SAME** on Vocareum, and you do not need to submit them.

If everything works fine, you should be able to see the submission report either in `terminal` or under `Details` tab.



It is really important for you to know that the grade shown in the submission report is **NOT** your final grade. To be fair, the `grading script` (exact **SAME** as `submission script`) will run your `Q-Learner.py` **again** to generate the final score after the deadline.

Final Grade

After grades published, you can check your grades on Vocareum and the total score here will be your final grade for this assignment.