ASSIGNMENT #3 – COMP 3106 ARTIFICIAL INTELLIGENCE

The assignment is an opportunity to demonstrate your knowledge on reinforcement learning and practice applying it to a problem.

The assignment may be completed individually, or it may be completed in small groups of two or three students. The expectations will not depend on group size (i.e. same expectations for all group sizes).

Components

The assignment should contain two components: an implementation and a technical document.

Implementation

The implementation should provide a complete solution in Python 3 to the problem outlined below.

The implementation will be graded both on correctness and use of good programming practices.

Technical Document

Your technical document should answer any questions posed below. Use these questions to elaborate on your implementation. After reading the technical document, the reader should understand how your implementation works. The technical document will be graded both on content and on presentation quality.

If the assignment was completed in a small group of students, the technical document must include a statement of contributions. This statement should identify: (1) whether each group member made significant contribution, (2) whether each group member made an approximately equal contribution, and (3) exactly which aspects of the assignment each group member contributed to.

Logistics

Assignment due date: Friday, December 2, 2022

Assignments are to be submitted electronically through Brightspace. It is your responsibility to ensure that your assignment is submitted properly. Copying of assignments is NOT allowed. Discussion of assignment work with others is acceptable but each individual or small group are expected to do the work themselves

Implementation

Programming language: Python 3

You may use the Python Standard Library (https://docs.python.org/3/library/). You may also use the NumPy, Pandas, and SciPy packages. Use of any additional packages requires approval of the instructor.

You must implement your code yourself. Do not copy-and-paste code from other sources, but you may use any pseudo-code we wrote in class as a basis for your implementation. Your implementation must follow the outlined specifications. Implementations which do not follow the specifications may receive a grade of zero. Please make sure your code is readable, as it will also be assessed for correctness. You do not need to prove correctness of your implementation.

You may be provided with a set of examples to test your implementation. Note that the provided examples do not necessarily represent a complete set of test cases. Your implementation may be evaluated on a different set of test cases.

Technical Document

Your technical document must answer all questions posed below. Ensure you answers are clear and concise. Submit the technical document as a single PDF file.

Implementation

Consider a cat and mouse game with six squares, as illustrated below. In this assignment, we will use passive temporal difference Q-learning to learn the optimal policy for the mouse to achieve as much reward as possible.

Square A	Square B	Square C
Square D	Square E	Square F

Assume that this is a fully observable environment. That is, the mouse knows for all times its position and the cat's position. Assume that this is a discrete time environment. At each time, the mouse agent may take one action. Furthermore, at each time, with some probability, the cat may move to a horizontally or vertically adjacent square.

The state of the environment comprises the positions of the mouse and the cat. We use the following string representation for environment states:
$P_M P_C$
Where P_M is the square the mouse is in
Where P_C is the square the cat is in

The mouse can take the following actions, with the following string representations:
"N": do not move
"L": move left
"R": move right
"U": move up
"D": move down

If the cat and mouse occupy the same square, the mouse receives large negative reward. If the cat and mouse occupy different squares, the mouse receives positive reward. Square B is the mouse's "home" square. The cat cannot occupy Square B, but the mouse receives negative reward in Square B. The reward r associated with a state s is:
$r(s) = -10$ if $P_M == P_C$
$r(s) = +1$ if $P_M != P_C$ and $P_M != B$
$r(s) = -1$ if $P_M == B$

Use the following parameters:
Gamma = 0.95 (discount factor)
Alpha = 0.05 (learning rate)

Initially estimate the Q-function as:
$Q(s, a) = 0$

A few important notes for your implementation:

1. Use the provided trials (which were generated under a fixed random policy) to learn the Q-function.
2. Consider a single iteration of temporal difference Q-learning (i.e. do not iterate through the trials multiple times; update the Q-value for a state-action pair once for every time it occurs in the trials).
3. Update the Q-values for the state-action pairs in the order they appear in the trials.
4. The trials are cut off after an arbitrary number of steps.

Your implementation must contain a file named "assignment3.py" with a class named "td_qlearning".

The “td_qlearning” class should have three member functions: “__init__”, “qvalue”, and “policy”.

The function “__init__” is a constructor that should take one input argument (in addition to “self”). The input argument is the full path to a CSV file containing a trial through the state space. Each CSV file will contain two columns, the first with a string representation of the state and the second with a string representation of the action taken in that state. The i^{th} row of the CSV file indicates the state-action pair at time i. You may assume only valid actions are taken in each state in the trial.

The function “qvalue” should take two input arguments (in addition to “self”). The first input argument is a string representation of a state. The second input argument is the string representation of an action. The function should return the Q-value associated with that state-action pair (according to the Q-function learned from the trial in the file passed to the __init__ function).

The function “policy” should take one input argument (in addition to “self”). The input argument is a string representation of a state. The function should return the optimal action (according to the Q-function learned from the trial in the file passed to the __init__ function). In the case of a tie (i.e. multiple actions are equally optimal), the function may return any one of the equally optimal actions.

The “qvalue” and “policy” functions will be called after the constructor is called. They may be called multiple times and in any order. I recommend computing the Q-function within the “__init__” function (store it in a member variable) and implement the “qvalue” and “policy” functions as getters.

Attached are example inputs and corresponding example outputs. Note that your functions should not write anything to file. These examples are provided in separate files for convenience.

Example trial CSV file:
BE,L
AE,N
AD,R
BA,R
CA,N
CD,N
CE,N
CF,L
BC,D
EC,N

Example input to “qvalue” function:
CD
N

Example output from “qvalue” function:
0.05

Example input to “policy” function:
AD

Example output from “policy” function:
R

Attached is skeleton code indicating the format your implementation should take.

Grading

The implementation will be worth 60 marks.

40 marks will be allocated to correctness on a series of test cases, with consideration to both the Q-function and the policy. These test cases will be run automatically by calling your implementation from another Python script. To facilitate this, please ensure your implementation adheres exactly to the specifications.

20 marks will be allocated to human-based review of code.

Technical Document

Please answer the following questions in the technical document. Explain why your answers are correct.

1. Briefly describe how your implementation works. Include information on any important algorithms, design decisions, data structures, etc. used in your implementation. [10 marks]
2. What type of agent have you implemented (simple reflex agent, model-based reflex agent, goal-based agent, or utility-based agent)? [3 marks]
3. Is the task environment: [7 marks]
    a. Fully or partially observable?
    b. Single or multiple agent?
    c. Deterministic or stochastic?
    d. Episodic or sequential?
    e. Static or dynamic?
    f. Discrete or continuous?
    g. Known or unknown?
4. Describe, at a high-level in words, the optimal strategy for the mouse (note that your implemented agent might not actually learn the optimal strategy through temporal difference Q-learning). [5 marks]
5. Suppose there is a state-action pair that is never encountered during a trial through state space. What should the Q-value be for this state-action pair? [5 marks]
6. For some cases (even with a long trial through state space), the optimal Q-value for a particular state-action pair will not be found. Explain why this might be the case. [5 marks]
7. In the test cases provided, the trials through state space were simulated using a random policy. Describe a different strategy to simulate trials and compare it to using a random policy. [5 marks]

Grading

The technical document will be worth 40 marks, allocated as described above.