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).

Assignment due date: Friday, December 1, 2023

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.

Components

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

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 (and any packages they depend on). 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.

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

Submit the implementation as a single PY file.

Technical Document

Your technical document should answer any questions posed below. Ensure your answers are clear and concise.

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.

Submit the technical document as a single PDF file.

Implementation

Consider a simplified version of the famous video game Frogger. In Frogger, the frog tries to get from a starting square to a goal square without getting hit by a car or other obstacle. In this assignment, we will use temporal difference Q-learning to learn the optimal policy for the frog to achieve as much reward as possible.

In this environment, we will assume there are eight squares. One square at the top of the environment is a goal square and one square at the bottom of the environment is the start square. There are six other squares.

	Square G (goal)	
Square A	Square B	Square C
Square D	Square E	Square F
	Square S (start)	

Assume that this is a fully observable environment. That is, the frog knows for all times its position and which squares have cars. Assume that this is a discrete time environment. At each time, the frog agent may take one action to move to a vertically adjacent square. Furthermore, at each time, with some probability, a car may appear or disappear from one or more squares A, B, C, D, E, or F.

The state of the environment comprises which squares have cars in them and the position of the frog. We use the following string representation for environment states:

$$P_{Frog}Car_ACar_BCar_CCar_DCar_ECar_F$$

Where P_{Frog} indicates which square the frog is in.
Where Car_i is a boolean variable indicating if there is a car in the square i.

As an example, the representation $E010001$ represents the frog is in square E and there are cars in squares B and F.

The frog can take the following actions, with the following string representations:
"N": do not move
"U": move up
"D": move down

The frog receives large positive reward and the trial ends if the frog is in the goal square. The frog receives large negative reward and the trial ends if the frog is in the same square as a car. The frog receives small negative reward for all other states. The reward r associated with a state s is:
r(s) = +100 if $P_{Frog} == G$
r(s) = -100 if $(P_{Frog} == B \text{ and } Car_B == 1) \text{ or } (P_{Frog} == E \text{ and } Car_E == 1)$
r(s) = -1 otherwise

Use the following parameters:
Gamma = 0.90 (discount factor)
Alpha = 0.10 (learning rate)

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

A few important notes for your implementation:

1. Use all the provided trials (which were generated under a fixed random policy) to learn the Q-function.
2. Iterate over all the trials multiple times until convergence is reached.

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 directory containing CSV files with trials 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 ith 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 trials in the directory 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 a string representation of the optimal action (according to the Q-function learned from the trials in the directory 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:

```
S000000,N
S000000,U
E001000,N
E000000,U
B100000,U
G000010,-
```

Example input to “qvalue” function:

```
S000000
U
```

Example output from “qvalue” function:

```
62.171
```

Example input to “policy” function:

```
E000000
```

Example output from “policy” function:

```
U
```

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]

The implementation uses temporal difference q-learning to update the Q-value for the agent.

The “td_qlearning” _class implements three functions: “__init__” _(i.e. the constructor), “qvalue”, and “policy”. The primary computation of the Q-values occurs in the “__init__” _function using the temporal difference q-learning update equation. The “qvalue” _function acts as a getter for the Q-values (which were updated in the constructor). The “policy” _function acts as a getter for the policy, selecting the action for the given state that maximizes the Q-value (which were updated in the constructor).

We use a dictionary of dictionaries to store the Q-values. The keys for the outer dictionary are the states; the values for the outer dictionary are the inner dictionaries. The keys for the inner dictionaries are the possible actions; the values for the inner dictionaries are the q-values. We also maintain a dictionary of all the possible actions for each state.

We have separate helper functions to: return a list of all possible actions in a given state, return a list of all possible states the environment could be in, compute the rewards for a given state. We also implement two helper functions to handle checking for Q-value convergence: one that deep copies a dictionary of Q-values, and one that computes if two dictionaries containing Q-values are equal to within some threshold.

2. What type of agent have you implemented (simple reflex agent, model-based reflex agent, goal-based agent, or utility-based agent)? [3 marks]

The agent is a utility-based agent. It aims to maximize the long-term rewards it receives from the environment.

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?

Fully observable; the agent has access to the entire environment.
Single agent; the frog agent (cars are treated as part of the environment).
Stochastic; cars appear randomly in the environment.
Sequential; the environment updates with the new position of the car and frog.
Static; the environment doesn't change while the agent is thinking.
Discrete; the agents can only be in discrete squares and move in discrete ways.
Known; the outcome of all actions is known for the frog.

4. 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]

The Q-value should be equal to the reward associated with the state. Because the state-action pair has not been encountered, the Q-value is not updated. Thus, the Q-value will maintain the initial value, which is the reward associated with the state.

5. 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]

There may be several reasons for not finding the optimal Q-value:

i. The state-action pair has never been visited or hardly visited. This results in the estimate of the Q-value not converging to the true expected long-term rewards from a state-action pair.
ii. The learning rate is not changing throughout the updating process. Thus, the Q-value is changing when we take a new action. The Q-value may oscillate within a small range.

6. Suppose we used a large state space approximation for learning Q-values for this task. Suggest a set of basis functions (or features) that could be used for the large state space approximation and explain why they are useful features. [5 marks]

Given the specific structure of the environment and its state representation, we can consider the following features.

Frog's Position:
Encode the current position of the frog as a feature. This ensures that the agent considers its location in the state space, which is essential for making optimal decisions.

Car Presence in Each Square:
Include boolean variables for each square to indicate the presence or absence of cars. This information is vital for the frog to avoid collisions with cars and navigate safely to the goal.

Action direction:
Use an indicator variable of -1 for action "down", 0 for action "do not move", and +1 for action "up". This could incentivize the frog to move in a direction toward or away from the goal.

Considering these features, the Q-function can be updated using the temporal difference Q-learning algorithm with the provided parameters (Gamma = 0.90, Alpha = 0.10). The rewards and penalties associated with different states and actions, as defined in the question, guide the learning process toward the optimal policy for the frog to reach the goal while avoiding collisions with cars.

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]

We can introduce a dynamic exploration bonus based on uncertainty in the Q-values. This approach aims to prioritize actions that the agent is less certain about, encouraging exploration in regions of the state space where the agent lacks confidence. One way to achieve this is by incorporating an exploration bonus term in the action selection process.

During each trial, the agent calculates the uncertainty or variance in its Q-value estimates for each action in a given state. Actions with higher uncertainty are then given a higher probability of being selected. This ensures that the agent is more likely to explore actions for which it has less confidence in the Q-value estimation.

This strategy allows for adaptive exploration, where the agent can be more explorative in unfamiliar or uncertain regions of the state space while still favoring exploitation in areas where it has more reliable knowledge. By introducing a dynamic exploration bonus based on Q-value uncertainty, we can enhance the adaptability of the agent's exploration strategy, potentially leading to more efficient learning and improved performance across diverse state-action pairs during trials.

Grading

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