Jenil Sanjaykumar Pandya

Final Project Documentation

Artificial Intelligence

1.Get your individual Jason data and successfully load it to your project. (20%)

**Output After Running “data\_loader.py”**

A screenshot of a computer

Description automatically generated

JSON Data Processing in Python  
  
• Imports the json module to handle JSON data.  
• The function load\_episodes(file\_path) reads a JSON file with episodes and returns them in a structured fashion.  
• The parameters are file\_path (str) and open(file\_path, 'r') as file.  
• Data is read from the file and sent to a Python object.  
• Processes data based on episodes, episode\_list, and step in episode.  
• Converts state to a tuple, adds action, reward, and new\_state, and updates episode\_list and episode\_list.  
• Returns a list of episodes, each expressed as a collection of tuples.  
• Example use:   
- The file path is: "C:/Udayton/AI/Final Project/sOjK115D.json".  
- Loaded episodes: print "episodes" to the console.

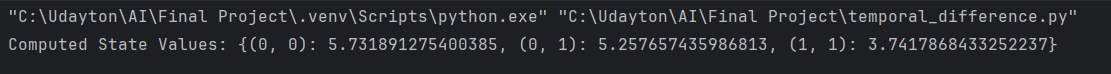
2. Successfully implement Temporal Difference (TD) and assign the state value through iterations (30%)

Imports and Functions for Machine Learning  
• Imports the random module to generate random numbers.  
• Initialize the state values. Function: Randomly assigns state values.  
• Initializes state values with an empty dictionary and loops through episodes.  
• Converts state into a tuple to ensure immutability.  
• If not found in the dictionary, assigns a random value to the uninitialized state.  
• Compute the time difference. Function: Calculates the Temporal Difference (TD) values for each state.  
• The parameters are discount factor, learning rate, and number\_iterations.  
• Example use: Provides example episodes for testing and computes the TD target and error.

**After Running “temporal\_difference.py” for seeing if all this works I’ve provided a Sample Run to the Program with State Assumptions and Action taken to get Computed State values for corresponding state and action taken.**

A screenshot of a computer

Description automatically generated



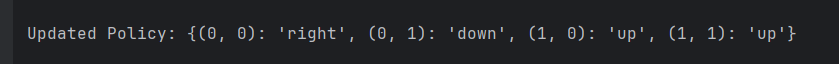
3. Successfully update the policy based on your individual data (20%)

**After Running “policy\_improvement.py” I had to do a lot of debugging for this module to work so I’ve also provided that, after Giving the script assumptions about the Stae values, the original action,**

**The output provides Calculation for each action taken at each State then Also Gives the best Action to take and it’s output and Finally Gives the Updated Policy to take.**

**A screen shot of a computer

Description automatically generated**

****

A screenshot of a computer

Description automatically generated

Imports and Setup for the Improve\_Policy Function  
• No imports; expects random and JSON imports.  
• The function seeks to determine the optimal action for each state based on current state values.  
• Parameters include state values, episodes, and gamma.  
• Steps include defining actions, initializing policies, analyzing actions, selecting the appropriate action, and debugging.  
• Usage examples: Includes sample states, actions, and episodes.  
• The Improve\_policy function determines the best actions and produces an improved policy.

4. Provide a detailed document about your implementation and result. (30%)

After Running Main.py file The Program Prints 3 Different things

1. It prints the data from **Data\_loader,py**
2. It prints the calculated state value from which episodes that it visited from **temporal\_difference.py** file
3. And At last it prints the updated policy

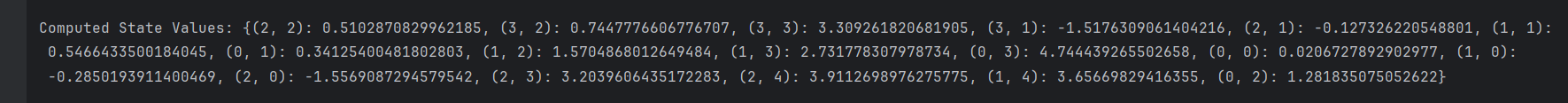
a. Snapshot of your individual dataset loaded and printed them in the console (10%)

**After Running “Main.py” FIle**

A screenshot of a computer

Description automatically generated

b. Shows the updated state values based on your individual training data. You can print them in your console and add the snapshot of the result (10%)

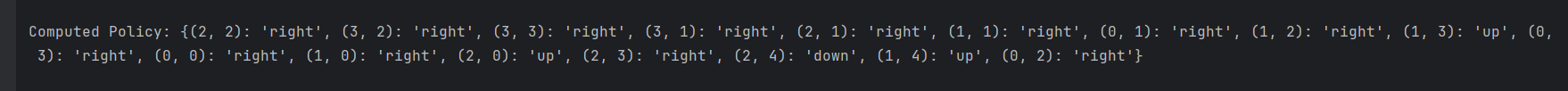


It only calculates the State Values for that which the Episodes were iterated not all of them and policy\_improvement.py does the same thing it only calculates visited states for episodic data.

A screen shot of a computer

Description automatically generated

c. Shows the updated policy based on the training data (10%)



It had similarly calculated the Policy values for each action taken for each state that it visited and then printed the final policy that it had taken. The data has been taken from data\_loader.py and the state values had been taken from temporal\_difference.py

A screenshot of a computer

Description automatically generated

Output Explanation for Script  
  
• The output includes Loaded Episodes, which is a list of episodes from the JSON file.  
• Computed State Values: State values calculated using the TD learning method.  
• Computed Policy: Improved policy based on computed state values and episodes.  
  
Module Interaction and Output Explanation.  
  
• Data\_loader.py: Loads episodes from a JSON file organized as state-action-reward transitions.  
• Temporal\_difference.py: Calculates state values depending on episodes using the TD learning technique.  
• Policy\_improvement.py: Generates improved policy from computed state values and episodes.  
  
Final Output Explanation:  
  
• Episodes: A list of state-action-reward transitions from a JSON file.  
• State Values: A dictionary of computed state values using TD learning.  
• Policy: A detailed policy that indicates the optimal course of action for each state based on the computed values.