**PROJECT 2**

**LEARNING PATHS FROM FEEDBACK USING Q-LEARNING**

**Suchismitha Vedala**

**PSID:1470929**

**INDEX**

1. BACKGROUND
   1. Reinforcement Learning
   2. Q Learning
   3. Sarsa Learning
2. PROBLEM STATEMENT
3. IDEALOGY
4. POLICIES
   1. Prandom
   2. PExploit
   3. Pgreedy
5. AGENT BEHAVIOUR IN PICK/DROP
6. EXPERIMENTS
7. IMPLEMENTATION
8. AGENT BEHAVIOUR IN PICK/DROP
9. EXPERIMENTS
10. PERFORMANCE MEASURE
11. RESULTS
12. CONCLUSION
13. REFERENCES

**1.BACKGROUND:**

**1.1 Reinforcement Learning:**

Reinforcement Learning is learning by interacting with an environment. An RL agent learns from the consequences of its actions, rather than from being explicitly taught and it selects its actions on basis of its past experiences (exploitation) and also by new choices (exploration), which is essentially trial and error learning. In other words, reinforcement learning is learning by trial and error method, primarily through rewards and punishments, At each step, the action of the agent is towards attaining more rewards rather than punishments. Thus, in RL an agent learns through all the actions that maximizes its performance

**1.2 Q Learning:**

Q learning is a reinforcement learning that learns from the expected values and the utility functions if made the right action. A key idea behind Q-learning is that an agent learns about how good actions are under perfect behavior *without* having to behave perfectly.

Algorithm Pseudocode:

1. Initialize the Q-values table, **Q(s, a)**.
2. Observe the current state, **s**.
3. Choose an action, **a**, for that state based on one of the action selection policies.
4. Take the action, and observe the reward, **r**, as well as the new state, **s'**.
5. Update the Q-value for the state using the observed reward and the maximum reward possible for the next state.
6. Set the state to the new state, and repeat the process until a terminal state is reached.

**1.3 SARSA Learning:**

State-Action-Reward-State-Action also known as SARSA Learning is reinforcement learning where the agent goes to a next state and performs an action before updating its utility values.

Algorithm Pseudocode:

1. Initialize the Q-values table, **Q(s, a)**.
2. Observe the current state, **s**.
3. Choose an action, **a**, for that state based on one of the action selection policies.
4. Take the action, and observe the reward, **r**, as well as the new state, **s'**.
5. Choose an action **a'** for that state **s’** based on the action selection policy
6. Update the Q-value for the state using the observed reward and the reward possible for the next state with the next action

Set the state to the new state and action to new action, and repeat the process until a terminal state is reached

**2. PROBLEM STATEMENT:**

Using Q Learning and SARSA learning, we analyze an agent’s performance in a grid world. The given problem does not have a goal state. It learns iteratively until the maximum number of steps is reached.

**3. IDEALOGY:**

An agent is referred to as a moving robot. The agent moves along the 5x5 grid.

An 5x5 grid is initialized with all the Q values as 0.

The Q\_table consists of 25 states. Each with 4 actions, i.e 100 keys

A agent in a state can either move east, west, north and south. An agent in each state can have block or no block.

So actually, there are 200 keys in Q\_table represented as Q(state, action, hasblock)

* Hasblock=1 if the agent carries a block
* Hasblock =0 if the agent does not carry a block

Every time, an agent is in the grid world, the agent will learn using the Q\_values of it’s state and the next state.

The action is chosen based on three policies described later in the document.

The Q\_values are computed for the SARSA and Q\_Learning based on their respective formula.

**4. POLICIES:**

**4.1PRandom:**

This policy returns a random action[east,west,north,south] from the list of actions every time.

**4.2 PExploit:**

This policy chooses a seed randomly.

If the seed is less than epsilon, then we choose an action randomly from the list of actions.

If the seed is greater than epsilon, then we choose an action greedily,

**4.3 PGreedy :**

This policy chooses an action that maximizes the utility value.

For a given state and hasblock, we observe all the Q\_values.

We then choose the action with the maximum Q\_value.

**5. AGENT BEHAVIOUR IN PICK/DROP**

* There are 4 pickup cells and 2 drop cells in our grid world.
* Each pickup state can contain maximum 4 blocks and drop state can contain maximum 8 blocks
* Initially, the pickup is maximized to 4 while the drop-off is initialized to 0
* When the agent reaches a pickup­­ block and the agent does not have a block, it will pick.
* When the agent reaches a drop state and the agent does have a block, it will drop the block.
* When the agent picks, its maximum value is reduced while when agent drops, its value is incremented.
* When the agent is In pick/drop state , the agent learns as follows:

The next state will be the current state

The hasblock of next state will be inverse of the hasblock of current state.

**6.EXPERIMENTS:**

Three experiments are supposed to run according to the Problem Statement.

**Experiment 1:**

For 3000 steps, the agent learns through the policy PRandom using Q learning formula

For 3000 steps, the agent learns through the policy PGreedy using Q learning formula

**Experiment 2:**

For 200 steps, the agent learns through the policy PRandom.

For the next 5800 steps, the agent learns through the policy PExploit using Q learning.

**Experiment 1:**

For 200 steps, the agent learns through the policy PRandom using SARSA Learning

For the next 5800 steps, the agent learns through the policy PExploit using SARSA Learning

**7.IMPLEMENTATION:**

* The project is implemented in Python.
* 25 states are created , each of the form [I,j] where i<5 and j<5.
* For each state, and each action and for each value of hasblock , we initialize Q values to be 0.
* For all our experiments, alpha=0.3 and discount=0.3
* We start our program at (1,5) and the agent has no block.
* At each step, we assign a reward to the agent based on its action.
* If pick or drop

Reward=12

Else if agent moves

Reward=-1

* At each step, next state and action is decided based on the policies.
* Once we compute these values, we update the Q\_table with the following formula:

Q:learning

Q(state, action, hasblock) =(1-alpha)\* Q(state, action, hasblock) +

             alpha [ reward + discount\*maxa’ Q(state’, action, hasblock )]

SARSA:learning

Q(state, action, hasblock) =(1-alpha)\* Q(state, action, hasblock) +

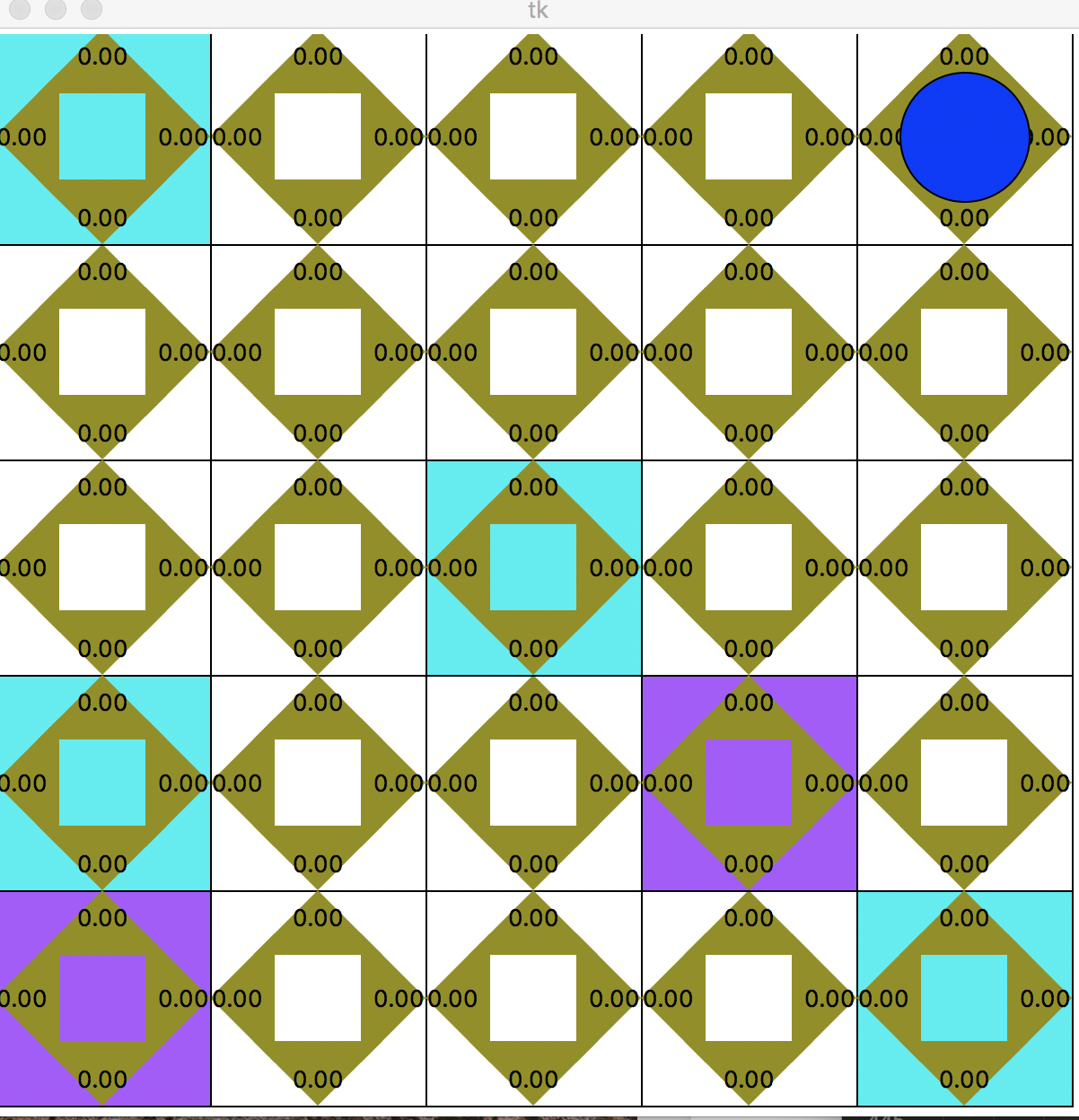
             alpha [ reward + discount\* Q(state’, action’, hasblock )]

**8.VISUALIZATION**

The agent movement along the grid world is visualized using tkinter.

* A 5x5 grid world is created .
* The pick up cells are represented with cyan and the drop cells are represented with purple.
* The agent is represented with a blue circle.
* Each state consists of 4 triangles with text in each triangle representing its Q\_value.
* The text in the triangle keeps changing as the agent learns along the path.

Fig 1: Visualization Grid



**9.PERFORMANCE MEASURES**

Two Performance Measures are taken into account:

* Bank\_account: The sum of reward at each step is considered as the bank account. The Bank account is the total reward.
* Number of operators: The number of operators needed to reach terminal stage. That is, every time an agent picks a block, or drops a block, or moves to a new state, the operator count is incremented. When the drop off of both the states reach the maximum value and all the pickup states have been decremented to 0, then the terminal state is reached in our problem.

This can happen many number of times. So an array is considered with the number of operators needed to reach a terminal state each time during the experiment.

**10. RESULTS**

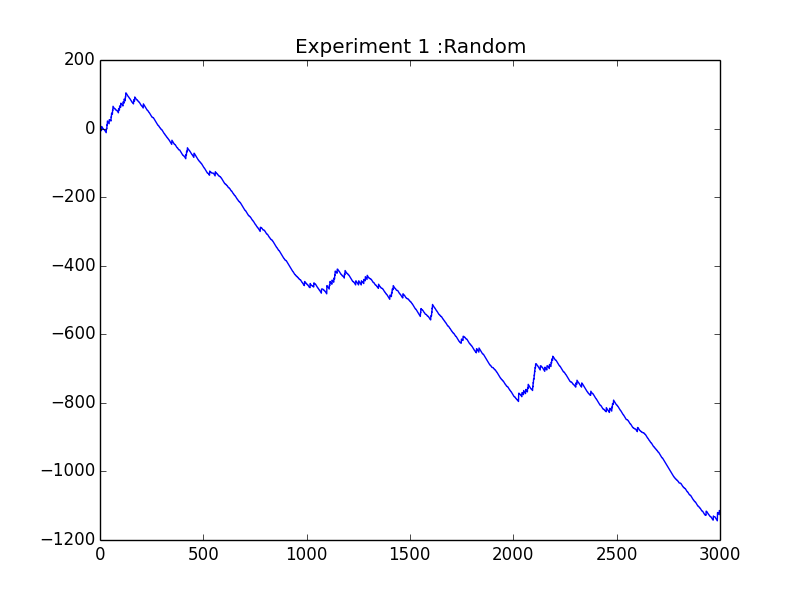
Below is the performance measure of Bank account for three experiments during three runs.

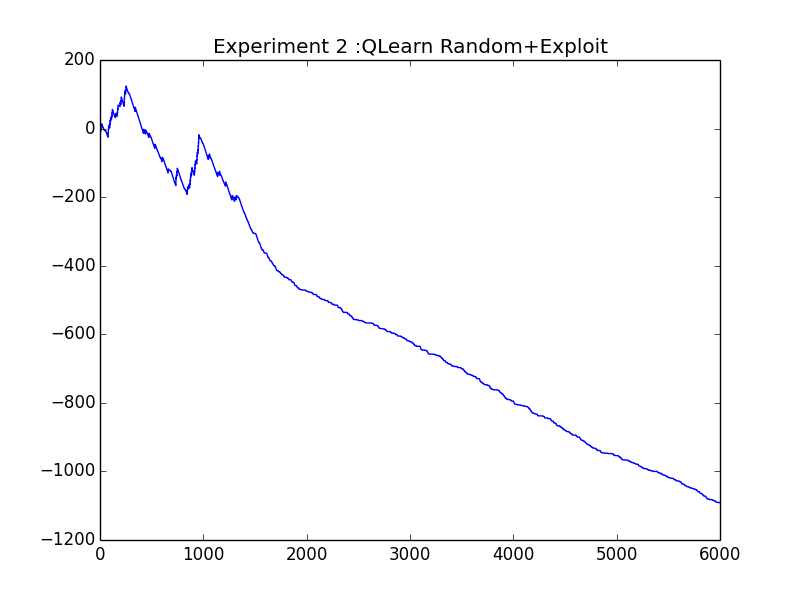
Fig 2: Bank account Values

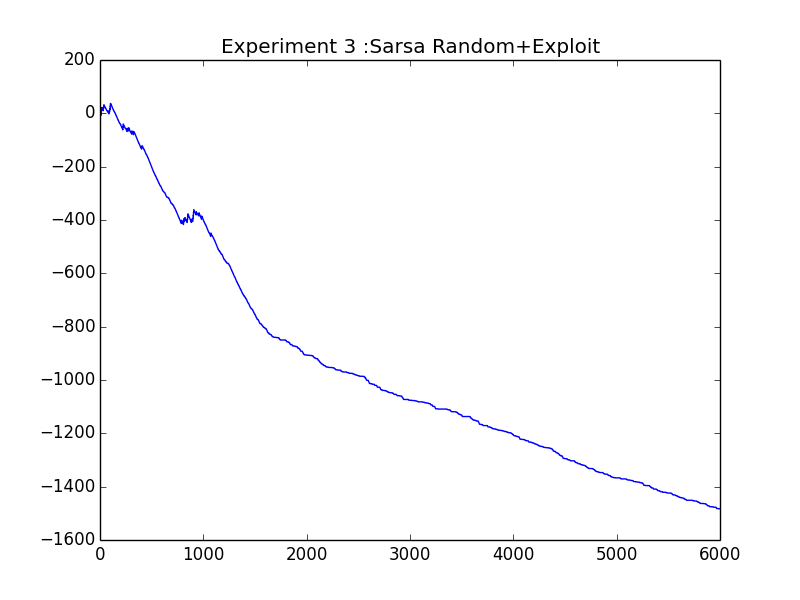
|  |  |  |  |
| --- | --- | --- | --- |
|  | **RUN 1** | **RUN 2** | **RUN 3** |
| **Experiment 1** | Random : -728  Greedy : -260 | Random : -968  Greedy : 6 | Random : -1308  Greedy : 20.0 |
| **Experiment 2** | -782 | -315 | -1049 |
| **Experiment 3** | -2463 | -2555 | -1438 |

**Fig3: Number of operators**

|  |  |  |
| --- | --- | --- |
|  | **RUN 1** | **RUN 2** |
| **Experiment 1** | 2 | 7 |
| **Experiment 2** | 9 | 4 |
| **Experiment 3** | 5 | 10 |

****



****

**The Q\_tables are stored in csv files in the path where the source code is present.**

**11.CONCLUSION:**

The Project focuses on learning new paths using Reinforcement Learning. The project can be further extended in learning new agents with other policies.

**12.REFERENCES:**

1. <https://www.youtube.com/watch?v=1XRahNzA5bE&t=10s>
2. <https://www.youtube.com/watch?v=XrxgdpduWOU>
3. <https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/>
4. <https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0>
5. <http://archive2.cra.org/Activities/craw_archive/dmp/awards/2004/Coggan/FinalReport.pdf>