Artificial Intelligence (H) Assessed Exercise Report

Ruairidh Macgregor 2250079m

1 Introduction

The aim of this investigation was to evaluate the performance of three different agents in a computer simulated environment in which the agent has to navigate successfully through a frosty lake to reach the goal without falling in a hole. The three type of agents used were: a senseless/random agent that makes non-deterministic decisions at each state, a simple agent that has informed knowledge of the problem space so can use efficient searching algorithms to navigate to the goal and an reinforcement learning agent that learns a (potentially) optimal policy after running a number of episodes in the environment.

1.0.1 PEAS Analysis

A PEAS analysis on all three agents is as follows.

- **Performance**
  - Navigate to the goal state without reaching any bad states and in as few steps as possible.
- **Environment**
  - Frozen lake. Each state in the environment can either be a frozen surface (safe) or a hole (bad). The properties of the environment are as follows:
    - **Partially observable** for Senseless and Reinforcement Learning Agents and **Fully Observable** for the Simple Agent
    - **Single agent** in all cases
    - **Deterministic** for the simple agent and **stochastic** for the senseless and reinforcement learning agents.
    - **Sequential** in all cases
    - **Static Environment** in all cases
    - **Discrete** in all cases
    - **Unknown** for the senseless and reinforcement learning agents. **Known** for the simple agent.

- **Actuators**
  - Physical Movement in all cases
- **Sensors**
  - **None** for the senseless agent, **oracle** for the simple agent and **perfect information about the current state** for the reinforcement learning agent.

2 Method and Design

The environment itself was made using the Open-AI Gym[1]. It was instantiated with a given problem ID n ($n \in \{0..7\}$), which affects the difficulty of the problem; a Boolean flag indicating if the environment was stochastic or not and the reward an agent will receive if the agent falls in a hole. If any of the three agents reach a hole, then the current trial is ended, the environment is reset and the agent tries again. The AIMA Toolbox[2] was also used, more specifically for the algorithmic side of the investigation. For each agent, the agent executed a number of episodes of the problem and in each episode, the agent was allowed a maximum number of iterations. The NumPy[3], SciPy[4] and matplotlib[5] libraries were also made use of, NumPy for efficiency purposes and matplotlib and SciPy for plotting results.

2.0.1 Random/Senseless Agent

To implement the senseless agent, the environment was set with a problem ID in the range specified above, a stochastic environment and the reward for falling in a hole was set to 0.0. For each episode, and for each iteration within the episode, the agent has no sensors and does not know where it is or what the reward is, so the agent takes an action chosen at random from the possible actions available to the agent and moves into a new state.

2.0.2 Simple Agent

The simple agent was given knowledge of the problem space before hand. This being the entire map of the environment and a heuristic $h(s)$ for each state s in the environment. $h(s)$ returns the straight line distance from s to the goal state. In order to maximize the agents performance measure, the choice of searching algorithm was obvious in this case - **A* search**[6]. Since the environment was not stochastic, and the agent was given knowledge of the environment beforehand. We know A* search to be complete and optimal[6]. The agent was run once in each problem ID as the results should not change.

2.0.3 Reinforcement Learning Agent

The choice of reinforcement learning algorithm used in this investigation was **Tabular Q-Learning**. Tabular Q-learning was chosen as the number of Q-values is not so large as to exhaust the state of the agent itself. The search space consists of only 64 states and in each state (apart from the terminal state) there is 4 actions to take (left, down, right, up). The choice for a Q-learning agent over other learning agents, is because a transition model of the environment does not have to be learned by the agent as the Q-value of being in a particular state s is directly related to s's neighbours.

The agent maintains a table of Q values Q, a state-action frequency table N, where in both cases the table is index by being in state s and taking action a. Other data that is provided to the agent is an exploration limit N_e, a learning rate α, a discount factor γ, a "bonus reward" R^+ before the agent has used up its exploration limit in s and taking a and the exploration function is defined as:

$$f(u,n) = \begin{cases} R^+ & \text{if } n < Ne \\ \text{u} & \text{otherwise} \end{cases}$$

Where the value that will be passed in as u is $Q[s,a]$, n is $N[s,a]$. Upon each iteration in the episode, the agent will observe its current state s and the reward r for being in s. It decides its next action to take by selecting the action a' that has the highest $f(u,n)$ with the parameters listed above and in doing so, also updates the Q value for being in s and taking action a.

$$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma \ \text{ argmax }_{a'} Q[s',a'] - Q[s,a])$$

3 Experiments and Results

Statistics collected from experiments:

- Maximum/Minimum mean reward from all episodes
- Mean reward collected per episode
- Covariance of the mean vector containing the mean rewards
- Goal loss
- Maximum/Minimum number of iterations to reach the goal
- Utility Loss

Extra parameters:

REWARD_HOLE_RANDOM = 0.0
REWARD_HOLE_SIMPLE = 0.0

REWARD_HOLE_Q $= -1.0$
MAX_EPISODES $= 50000$
MAX_ITERS_PER_EPISODE $= 1000$
$\alpha = 0.8$
$\gamma = 0.9$
$R^+ = 0.1$
$N_e = 1000$

3.0.1 Maximum/Minimum Mean Reward From All Episodes

Random Agent:

Problem ID	0	1	2	3	4	5	6	7
Max	0.06	0.08	0.09	0.06	0.08	0.10	0.05	0.13
Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Q Learning:

Problem ID	0	1	2	3	4	5	6	7
Max	0.00	0.063	0.09	0.08	0.07	0.08	0.063	0.11
Min	-0.14	-0.33	-0.25	-0.25	-0.33	-0.20	-0.33	-0.25

3.0.2 Mean Reward Per Episode

(Turn over for Graphs)

3.0.3 Covariance of the Mean Rewards

Random Agent:

Problem ID	0	1	2	3	4	5	6	7
Covariance	9.20×10^{-8}	4.90×10^{-7}	1.12×10^{-5}	9.65×10^{-7}	7.49×10^{-7}	1.47×10^{-6}	6.84×10^{-8}	3.13×10^{-5}

Q Learning:

Problem ID	0	1	2	3	4	5	6	7
Covariance	0.00012	0.0021	0.0023	0.0022	0.0058	0.0011	0.0021	0.0018

3.0.4 Goal Loss

Random Agent:

Problem ID	0	1	2	3	4	5	6	7
Num Goal	5	10	218	37	27	44	2	546

Q-Learning:

Problem ID	0	1	2	3	4	5	6	7
Num Goal	0	3676	14059	17413	13730	24154	1951	27157

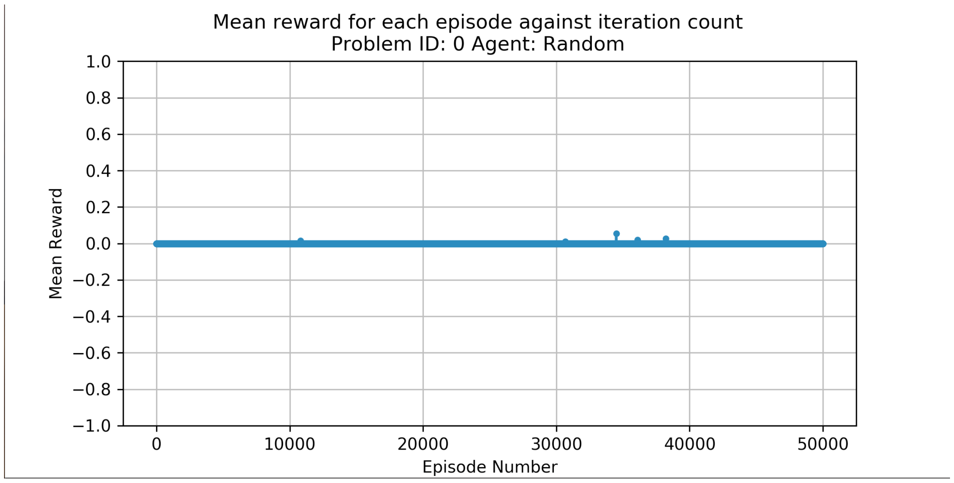


Figure 1: Problem ID = 0, Random Agent. Shows the agent rarely ever reaches the goal and a reward of +1.0.

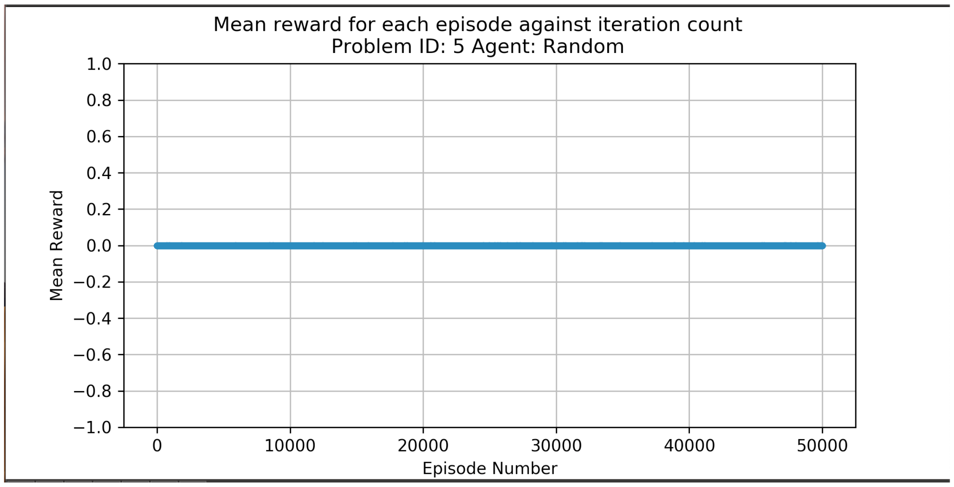


Figure 2: Problem ID = 3, Random Agent. Same as above

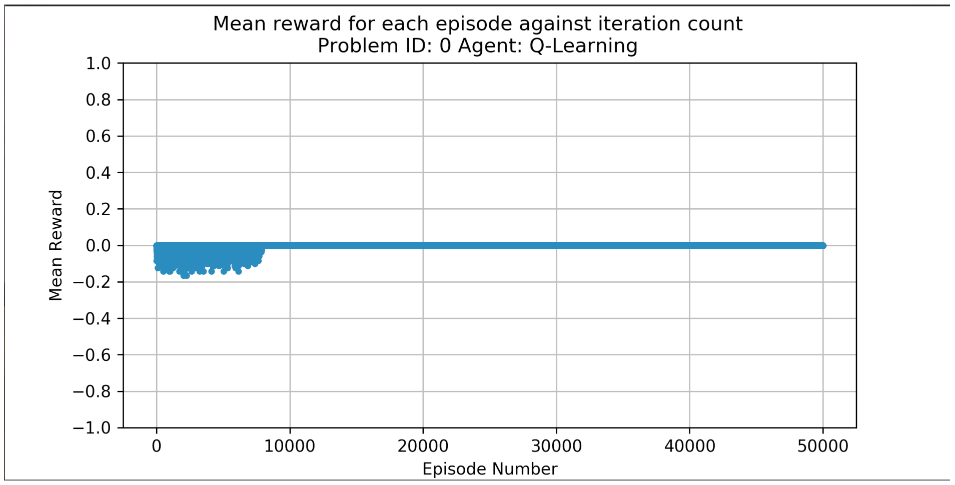


Figure 3: Problem ID = 0, Q-learner. Shows signs the agent is learning better actions to take by the increasing reward and then what seems to be converging to a fixed policy.

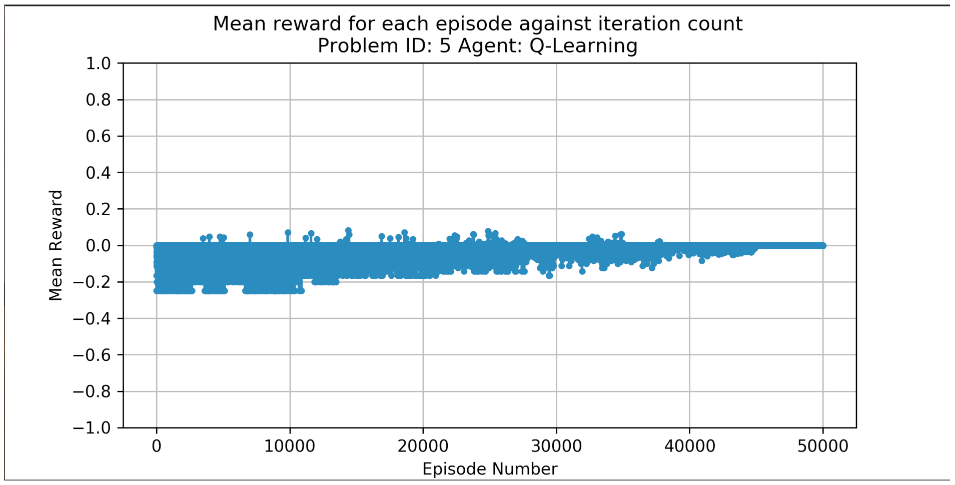


Figure 4: Problem ID = 5, Q-learner. Shows similar features as above, but there is a lot higher variance within this graph and a slower convergence time to reach a fixed policy (optimal or not).

3.0.5 Minimum/Maximum Number of Iterations to Reach the Goal

*if N/A appears it is because the agent did not reach that goal

Problem ID	0	1	2	3	4	5	6	7
Max iters	103	43	75	68	66	137	33	86
Min iters	18	12	11	18	12	10	20	8

Simple Agent:

Problem ID	0	1	2	3	4	5	6	7
Iters	155	148	141	146	146	152	146	157

Q-Learning:

Problem ID	0	1	2	3	4	5	6	7
Max iters	N/A	1000	962	993	998	1000	994	655
Min iters	N/A	16	11	12	19	13	16	9

3.0.6 Utility Loss

The Value Iteration algorithm was used to compute the optimal policy for the underlying Markov Decision Process in the environment. (Turn over for results):
*if Q-learning U is -1000.000 it is because the agent never reached that state

4 Analysis of Results

4.0.1 Mean Rewards

As expected - the random agent's mean reward per episode stays mostly the same throughout all the episodes. This is expected as the agent is making non-deterministic decisions at each state it reaches and is not motivated by anything else such as the Q-learner, or the simple agent. Looking at the Q-learner, the agent clearly is learning what actions to take throughout the trials in order to maximize its performance measure. There is some variance in the mean rewards for the Q-learner, but this is to be expected since the environment itself is stochastic, therefore increasing the complexity of the environment as there is a transition model $P(s'|s, a)$ for each state in the environment.

4.0.2 Iterations

Looking at the tables, it is clearly noticeable that the simple agent will always find a solution to the goal. We know this to be optimal by the nature of A* search and the environment itself, is not stochastic. With regards to the random agent, the iterations to reach the goal cannot be really compared as the decision is just made at random and it is by chance, whether the agent reaches the goal or not. Over to the Q-learner, it is clear that the agent is learning throughout the episodes. The values for the maximum iterations to reach the goal is a lot higher compared to the minimum.

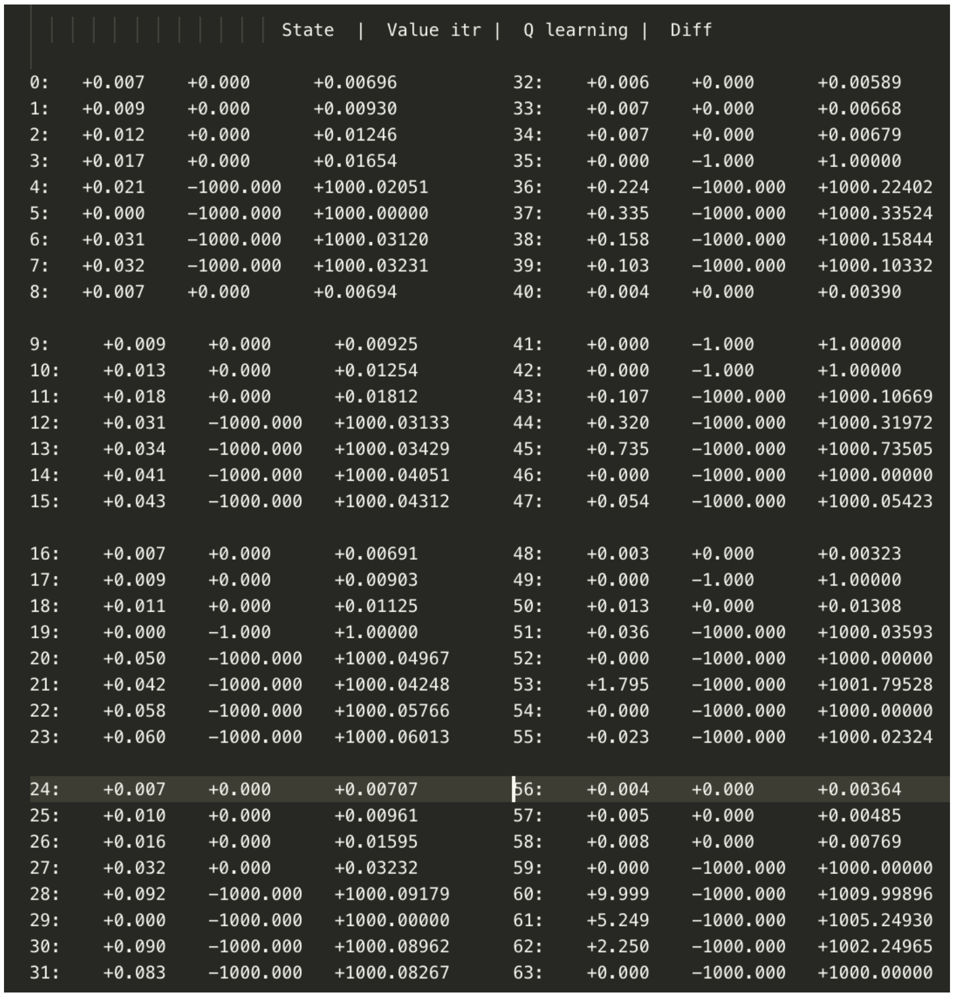
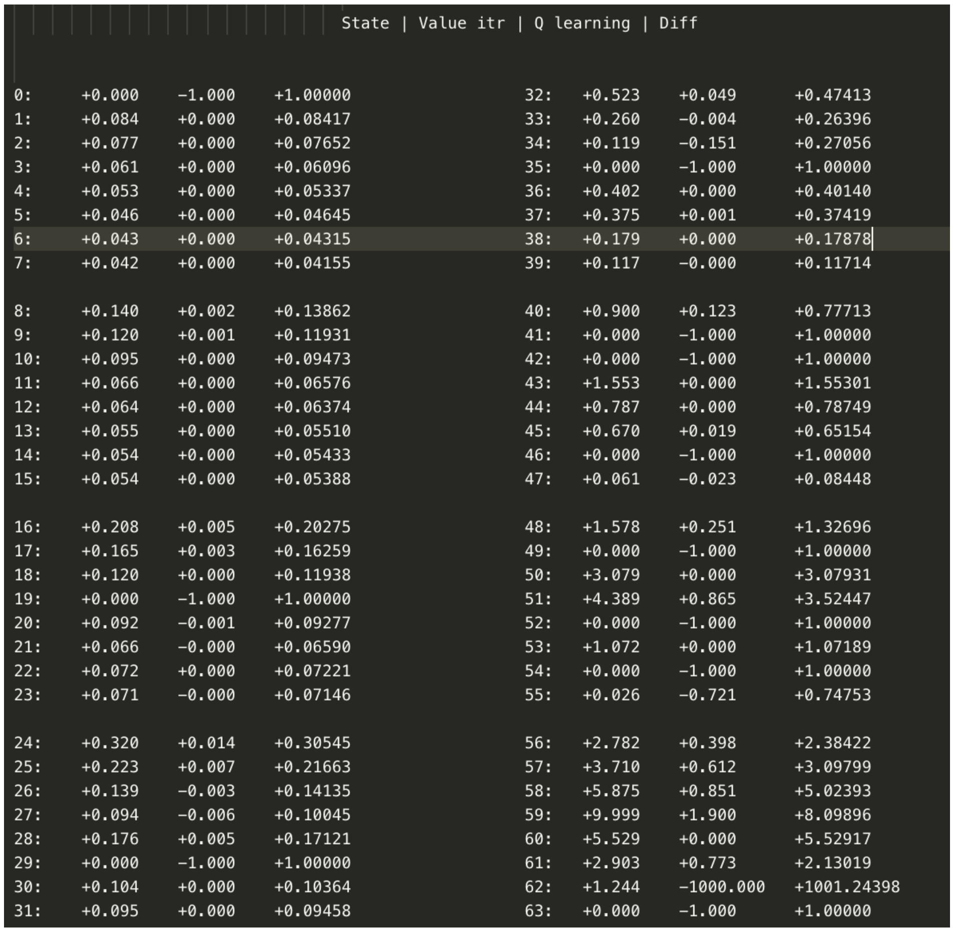

```
             State  |  Value itr |  Q learning |  Diff

0:    +0.007    +0.000      +0.00696          32:    +0.006    +0.000      +0.00589
1:    +0.009    +0.000      +0.00930          33:    +0.007    +0.000      +0.00668
2:    +0.012    +0.000      +0.01246          34:    +0.007    +0.000      +0.00679
3:    +0.017    +0.000      +0.01654          35:    +0.000    -1.000      +1.00000
4:    +0.021    -1000.000   +1000.02051       36:    +0.224    -1000.000   +1000.22402
5:    +0.000    -1000.000   +1000.00000       37:    +0.335    -1000.000   +1000.33524
6:    +0.031    -1000.000   +1000.03120       38:    +0.158    -1000.000   +1000.15844
7:    +0.032    -1000.000   +1000.03231       39:    +0.103    -1000.000   +1000.10332
8:    +0.007    +0.000      +0.00694          40:    +0.004    +0.000      +0.00390

9:      +0.009    +0.000      +0.00925        41:    +0.000    -1.000      +1.00000
10:     +0.013    +0.000      +0.01254        42:    +0.000    -1.000      +1.00000
11:     +0.018    +0.000      +0.01812        43:    +0.107    -1000.000   +1000.10669
12:     +0.031    -1000.000   +1000.03133     44:    +0.320    -1000.000   +1000.31972
13:     +0.034    -1000.000   +1000.03429     45:    +0.735    -1000.000   +1000.73505
14:     +0.041    -1000.000   +1000.04051     46:    +0.000    -1000.000   +1000.00000
15:     +0.043    -1000.000   +1000.04312     47:    +0.054    -1000.000   +1000.05423

16:     +0.007    +0.000      +0.00691        48:    +0.003    +0.000      +0.00323
17:     +0.009    +0.000      +0.00903        49:    +0.000    -1.000      +1.00000
18:     +0.011    +0.000      +0.01125        50:    +0.013    +0.000      +0.01308
19:     +0.000    -1.000      +1.00000        51:    +0.036    -1000.000   +1000.03593
20:     +0.050    -1000.000   +1000.04967     52:    +0.000    -1000.000   +1000.00000
21:     +0.042    -1000.000   +1000.04248     53:    +1.795    -1000.000   +1001.79528
22:     +0.058    -1000.000   +1000.05766     54:    +0.000    -1000.000   +1000.00000
23:     +0.060    -1000.000   +1000.06013     55:    +0.023    -1000.000   +1000.02324

24:     +0.007    +0.000      +0.00707        56:    +0.004    +0.000      +0.00364
25:     +0.010    +0.000      +0.00961        57:    +0.005    +0.000      +0.00485
26:     +0.016    +0.000      +0.01595        58:    +0.008    +0.000      +0.00769
27:     +0.032    +0.000      +0.03232        59:    +0.000    -1000.000   +1000.00000
28:     +0.092    -1000.000   +1000.09179     60:    +9.999    -1000.000   +1009.99896
29:     +0.000    -1000.000   +1000.00000     61:    +5.249    -1000.000   +1005.24930
30:     +0.090    -1000.000   +1000.08962     62:    +2.250    -1000.000   +1002.24965
31:     +0.083    -1000.000   +1000.08267     63:    +0.000    -1000.000   +1000.00000
```

Figure 5: Utility Loss, Problem ID = 0, Q-learner. From these utilities we can infer that the agent found this problem to be very difficult, as the policy learned is nowhere near optimal.



```
State | Value itr | Q learning | Diff


0:     +0.000    -1.000    +1.00000                    32:   +0.523    +0.049      +0.47413
1:     +0.084    +0.000    +0.08417                    33:   +0.260    -0.004      +0.26396
2:     +0.077    +0.000    +0.07652                    34:   +0.119    -0.151      +0.27056
3:     +0.061    +0.000    +0.06096                    35:   +0.000    -1.000      +1.00000
4:     +0.053    +0.000    +0.05337                    36:   +0.402    +0.000      +0.40140
5:     +0.046    +0.000    +0.04645                    37:   +0.375    +0.001      +0.37419
6:     +0.043    +0.000    +0.04315                    38:   +0.179    +0.000      +0.17878
7:     +0.042    +0.000    +0.04155                    39:   +0.117    -0.000      +0.11714

8:     +0.140    +0.002    +0.13862                    40:   +0.900    +0.123      +0.77713
9:     +0.120    +0.001    +0.11931                    41:   +0.000    -1.000      +1.00000
10:    +0.095    +0.000    +0.09473                    42:   +0.000    -1.000      +1.00000
11:    +0.066    +0.000    +0.06576                    43:   +1.553    +0.000      +1.55301
12:    +0.064    +0.000    +0.06374                    44:   +0.787    +0.000      +0.78749
13:    +0.055    +0.000    +0.05510                    45:   +0.670    +0.019      +0.65154
14:    +0.054    +0.000    +0.05433                    46:   +0.000    -1.000      +1.00000
15:    +0.054    +0.000    +0.05388                    47:   +0.061    -0.023      +0.08448

16:    +0.208    +0.005    +0.20275                    48:   +1.578    +0.251      +1.32696
17:    +0.165    +0.003    +0.16259                    49:   +0.000    -1.000      +1.00000
18:    +0.120    +0.000    +0.11938                    50:   +3.079    +0.000      +3.07931
19:    +0.000    -1.000    +1.00000                    51:   +4.389    +0.865      +3.52447
20:    +0.092    -0.001    +0.09277                    52:   +0.000    -1.000      +1.00000
21:    +0.066    -0.000    +0.06590                    53:   +1.072    +0.000      +1.07189
22:    +0.072    +0.000    +0.07221                    54:   +0.000    -1.000      +1.00000
23:    +0.071    -0.000    +0.07146                    55:   +0.026    -0.721      +0.74753

24:    +0.320    +0.014    +0.30545                    56:   +2.782    +0.398      +2.38422
25:    +0.223    +0.007    +0.21663                    57:   +3.710    +0.612      +3.09799
26:    +0.139    -0.003    +0.14135                    58:   +5.875    +0.851      +5.02393
27:    +0.094    -0.006    +0.10045                    59:   +9.999    +1.900      +8.09896
28:    +0.176    +0.005    +0.17121                    60:   +5.529    +0.000      +5.52917
29:    +0.000    -1.000    +1.00000                    61:   +2.903    +0.773      +2.13019
30:    +0.104    +0.000    +0.10364                    62:   +1.244    -1000.000   +1001.24398
31:    +0.095    +0.000    +0.09458                    63:   +0.000    -1.000      +1.00000
```

Figure 6: Utility Loss, Problem ID = 6, Q-learner. Signs of the agent is learning a good policy in a slightly easier problem.

4.0.3 Goal Loss

We know the simple agent to always reach the goal, due to the nature of A*-search in a deterministic environment. However, looking at the random agent, the number of times the agent reaches the goal is very low in all cases, as previously mentioned, this is to be expected. The Q-learner wasn't able to reach the goal state in all problems, with signs of reaching a (most likely local) optimal policy when trying to reach the goal as indicated from the graphs and the agent is making it to the goal at a higher rate than the random.

4.0.4 Utility Loss

Looking at the tables, the agent never learns the true maximum expected utility in each state (and hence an optimal policy). Looking at the graphs of the mean reward, we can see that the agent learns a policy which it may believe to be optimal and hence making the agent to make decisions it believes to be correct. But there is states the agent could also visit that may fully optimize its performance measure for the given environment, but is choosing not to. The number of episodes and iterations per episode is a vital factor when trying to work out the optimal policy and in this investigation may not have been high enough to allow the agent to explore the environment further and hence can be viewed as a limitation in this investigation.

5 Conclusion

In conclusion, it is clear that reinforcement learning can learn very close to optimal (if not optimal itself) policies to help the agent make rational decisions. Reinforcement learning is certainly the best approach out of the three agents in environments that are partially observable and stochastic. However, if the environment is fully deterministic, simple agents are a very good option as there is a wide range of deterministic algorithms to choose from that have good performance and are proven to be correct that you can provide the agent with.

References

[1] OpenAI Gym `https://gym.openai.com/`

[2] Artificial Intelligence: A Modern Approach Toolbox *Code implementations of the algorithms* `https://github.com/aimacode/aima-python`

[3] NumPy Python Library `http://www.numpy.org/`

[4] SciPy Python Library `https://www.scipy.org/`

[5] matplotlib Python Library `https://matplotlib.org/`

[6] Artificial Intelligence: A Modern Approach Pages 93-99 *A* search algorithm overview and proofs regarding it*