Artificial Intelligence 5M - Loch Lomond Lake

Mayra A. Valdes Ibarra - 2419105v

1 Introduction

You and your friends were tossing around a frisbee at Loch Lomond when you made a wild throw that left the frisbee out in the middle of the lake. It is your job now to navigate across the lake and retrieve the disc without stepping into the holes where the ice has melted. The ice is slippery, so you won't always move in the direction you intend.

The goal of this report is to design, implement and evaluate three different virtual agents that are able to navigate across the Loch Lomond Frozen Lake grid and retrieve the frisbee disc. Three different agents are analyzed: a senseless/random agent, a simple agent and a reinforcement learning agent. These agents are evaluated over eight different 8x8 grid environments.

2 Analysis

The Loch Lomond Frozen Lake environment is a customized Open AI Gym environment derived from FrozenLake (https://gym.openai.com/envs/#toy_text). It consists grid-world with a starting position (S), frozen surface (F), obstacles (holes) (H) and a final goal (G), where the frisbee disc is located. The task for our agents will be to get to get from S to G.

Our random agent is be used as a naive baseline, while the simple agent is used as an ideal baseline to find the optimal path under ideal circumstances.

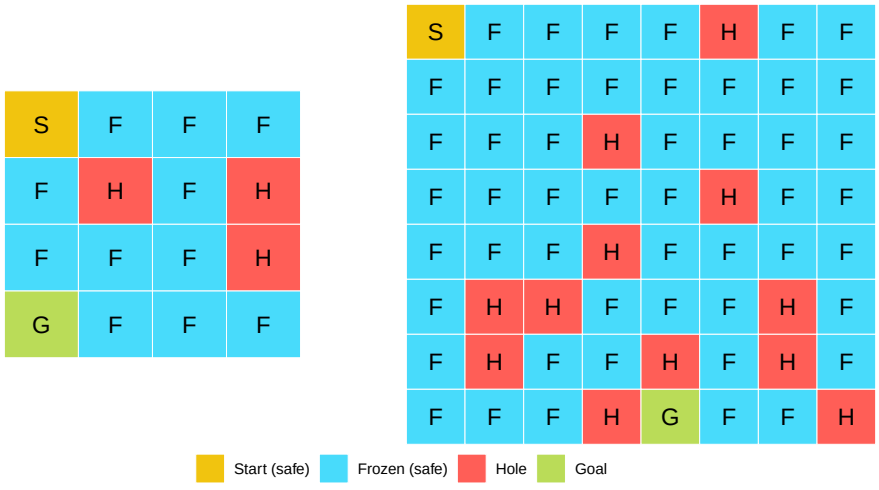


Figure 1: Loch Lomond Frozen Lake Grid for Problem 0, 4x4 (left) and 8x8 (right)

2.1 PEAS Description

• **Performance measure**: Mean rewards obtained each episode, shortest path from the starting position (S) to the frisbee location (G). Rewards are obtained by reaching the goal (G).
• **Environment**: 4x4 and 8x8 grids, as observed in Figure 1.
• **Actions**: Left (0), Down (1), Right (2), Up (3).
• **Sensors**: Grid coordinates knowledge of current state.

2.2 Task Environments and their Characteristics

Table 1: Task environment and their characteristics for our three different agents.

Task Environment	Observable	Deterministic	Episodic	Discrete	Known
Random Agent	Partially observable	Stochastic	Episodic	Discrete	Unknown
Simple Agent	Fully observable	Deterministic	Episodic	Discrete	Known
Reinforcement Agent	Partially observable	Stochastic	Sequential	Discrete	Known

In the stochastic environments, the "intended" action occurs with probability `0.333`, but with probability `0.666` the agent moves at right angles to the intented direction. A collision with a wall will result in non movement. For all our agents, the goal state (G) has reward of `1.0` and the rest of non terminal states have a reward of `0`. For the random and simple environments the reward of the obstacles (H) is `0`, while the reward of the reinforcement learning agent was set to `-0.05`.

3 Methodology

3.1 Random/Senseless Agent

This agent does not have a specific algorithm as its nature is to behave randomly.

3.2 Simple Agent

Our simple agent implements an $A^*search$ algorithm, using the following formula[1]:

$$f(n) = g(n) + h(n) \tag{1}$$

An $A^*search$ algorithm evaluates the cost to reach a node $g(n)$, and uses a heuristic $h(n)$ function to evaluate the cost to get from the node to the goal.

In our frozen like grid, we define our $g(n)$ and $h(n)$ as follows:
• $g(n)$ will represent the geodesic distance between the node and the starting point.
• $h(n)$ will represent the euclidean distance between the node and the goal.

[1]Russell, Stuart J, and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, N.J: Prentice Hall, 1995. Print. (p.93)

3.3 Reinforcement Learning Agent

Given our specifications, we decided to use an Q-learning agent, given that we needed a **model-free** method for solving the problem. Using a temporal-difference approach, we make use of the update Bellman equation[2]:

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma \cdot max_{a'} Q(s',a') - Q(s,a)) \tag{2}$$

Where:
- s denotes the previous state
- a denotes the previous action
- s' denotes the current state
- a' denotes any possible action of the current state
- $Q(s,a)$ denotes the value of doing action a in state s
- γ is the discount factor, which was implemented with a value of 0.95
- α is the learning rate, which was implemented with a value of $60/(59 + N[s,a])$, where $N[s,a]$ is the number of times an action has been tried in state s
- $R(s)$ denotes the reward for the state

3.3.1 Action selection

We make use of a greedy agent in the limit of infinite exploration, or GLIE[3]. This means that our agent chooses a random action a fraction $1/t$ of the time. We specified t as 0.075. Selection of the optimal action given a state is defined as:

$$a = argmax_{a'}(Q(s',a') + noise) \tag{3}$$

Where *noise* is a random number in the interval [0.0, 1.0) divided by number of episodes. The reason of adding a noise is explained in the implementation section below.

Additionally, we get a utility table through the Q-values table. The relation between Q-values and the utility is as follows:

$$U(s) = max_a Q(s,a)$$

4 Implementation

Our agents can be called through the `run_random.py {problem_id}`, `run_simple.py {problem_id}` and `run_rl.py {problem_id}` scripts, where `problem_id` takes a user input value of an integer between 0 and 7. It also has support for comma separated values, e.g. `run_rl.py 0,1,2,3`. The code implementation uses an object-oriented paradigm, where the agents inherit functionality from a `MyAbstractAIAgent` base agent.

[2]Russell, Stuart J, and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, N.J: Prentice Hall, 1995. Print. (p.844)

[3]Russell, Stuart J, and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, N.J: Prentice Hall, 1995. Print. (p.840)

4.1 Senseless Agent

The implementation of our senseless is pretty simple. Basically we make use of the function `env.action_space.sample()` in order to pull a random action every step.

4.2 Simple Agent

In our implementation we make use of the AIMA Toolbox (https://github.com/aimacode/aima-python) code, making use of the `astar_search` function. A mapping from our environment (`env.desc`) was used in order to create a `UndirectedGraph` and `GraphProblem` classes, inherited from the AIMA Toolbox as well.

Internally, `GraphProblem` is the class that defines the heuristic function, and `astar_search` makes use of a priority queue in order to minimize $f(n)$.

Once the `astar_search` finds the shortest way from S to G, we make use of the solution in order to create a policy π that will be used by our agent. Since simple agent is deterministic, making use of this policy will guarantee reaching the goal with the shortest possible path.

4.3 Reinforcement Learning Agent

Our reinforcement learning agent has two phases, training and evaluation. Before running evaluation phase, we run a "training phase", where the agent finds the optimal policy π. Policy π is then used during evaluation phase. Both phases are evaluated/analyzed below. Different values for our discount factor, learning rate, action selection, and reward hole value were tested. The final implementation maximizes the performance of our agent.

As noted before, our action selection has a formula $a = argmax_{a'}(Q(s', a') + noise)$. The reason behind adding the *noise* was that we encoutered a pattern where the left action was the preferred action chosen by our agent since left action is defined by `0`, being the first option for the agent to pick it through the `argmax` function as well (first index in Python arrays is 0).

5 Evaluation

Evaluation can be called through the `run_eval.py {problem_id}` script, where `problem_id` has the same properties as mentioned previously for running single agents.

Every agent produces different evaluation files that will be created inside the `out` folder. The specs for evaluation are as follows:
- **Total episodes**: 10,000
- **Training phase episodes**: 10,000 *(for reinforcement agent only)*
- **Max iterations per episode**: 1,000

5.1 Evaluation Tables

Table 2: Total of successes by agent

			Reinforcement Learning Agent	
Problem ID	Random Agent	Simple Agent	Training Phase	Evaluation Phase
0	1	10000	547	1375
1	2	10000	765	2145
2	49	10000	5417	8312
3	4	10000	4660	8194
4	4	10000	4128	8025
5	9	10000	3224	7579
6	NA	10000	3065	5788
7	109	10000	5234	8047

Table 3: Average iterations per episode where the agent reached the goal

			Reinforcement Learning Agent	
Problem ID	Random Agent	Simple Agent	Training Phase	Evaluation Phase
0	23.00	13	93.87	123.65
1	19.00	11	80.82	116.16
2	22.82	9	54.18	62.06
3	27.75	10	69.40	75.80
4	36.00	9	75.10	83.68
5	42.89	9	83.58	94.32
6	NA	11	100.93	117.51
7	23.40	7	44.89	51.12

Table 4: Total of failures by agent

		Reinforcement Learning Agent	
Problem ID	Random Agent	Training Phase	Evaluation Phase
0	9999	9453	8625
1	9998	9235	7855
2	9951	4583	1688
3	9996	5340	1806
4	9996	5872	1975
5	9991	6776	2421
6	10000	6935	4212
7	9891	4766	1953

Note:
Simple agent is not added in this table as it never failed.

Table 5: Average iterations per episode where the agent failed (reached a hole)

		Reinforcement Learning Agent	
Problem ID	Random Agent	Training Phase	Evaluation Phase
0	27.93	76.96	113.49
1	8.89	59.59	105.59
2	11.42	40.66	46.03
3	10.67	44.33	48.73
4	7.94	45.75	55.16
5	24.32	50.66	41.80
6	10.49	53.99	63.80
7	14.50	42.18	57.26

Note:
Simple agent is not added in this table as it never failed.

We can observe in tables above how our reinforcement learning agent tends to stay longer in the environment in the episodes where it reached a hole (comparing the training/evaluation phase to our naive baseline).

Table 6: Mean rewards per episode

			Reinforcement Learning Agent	
Problem ID	Random Agent	Simple Agent	Training Phase	Evaluation Phase
0	0.0001	1	0.0547	0.1375
1	0.0002	1	0.0765	0.2145
2	0.0049	1	0.5417	0.8312
3	0.0004	1	0.4660	0.8194
4	0.0004	1	0.4128	0.8025
5	0.0009	1	0.3224	0.7579
6	0.0000	1	0.3065	0.5788
7	0.0109	1	0.5234	0.8047

We can observe in table above how our reinforcement learning agent reaches more than 80% success rate in some of the problems.

5.2 Evaluation Plots

Plots below show our three different agents performance in problem 0. We show both the training and evaluating phase.

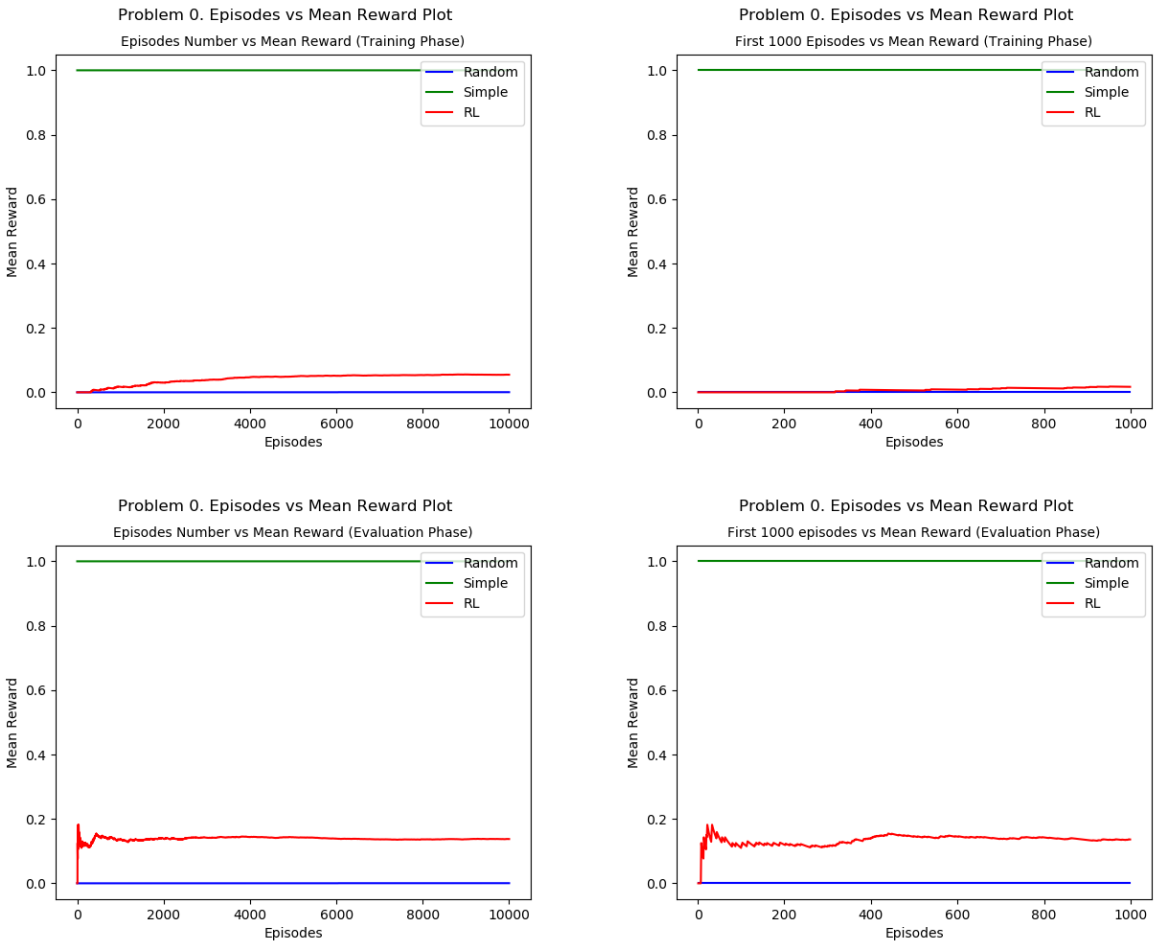


Figure 2: Evaluation Plots for Problem 0. The full list of evaluation plots can be found in the appendix D.

5.3 Evaluating our Reinforcement Learning Agent

We can observe in figure below the grid environment for problem 0, as well as the best policy found by our agent after 10,000 episodes. From our mean reward plot, we can see how the mean reward is increasing at a fast rate during the training phase. During the evaluation phase, we can see that it rapidly converges to the mean reward average of `~0.15` before reaching the 2000 episodes.

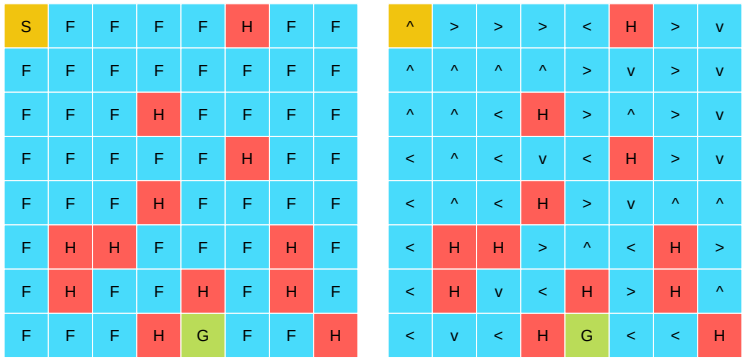


Figure 3: Problem 0 Grid (left) and best policy (right). The full list of grids and best policies can be found in the appendix B.

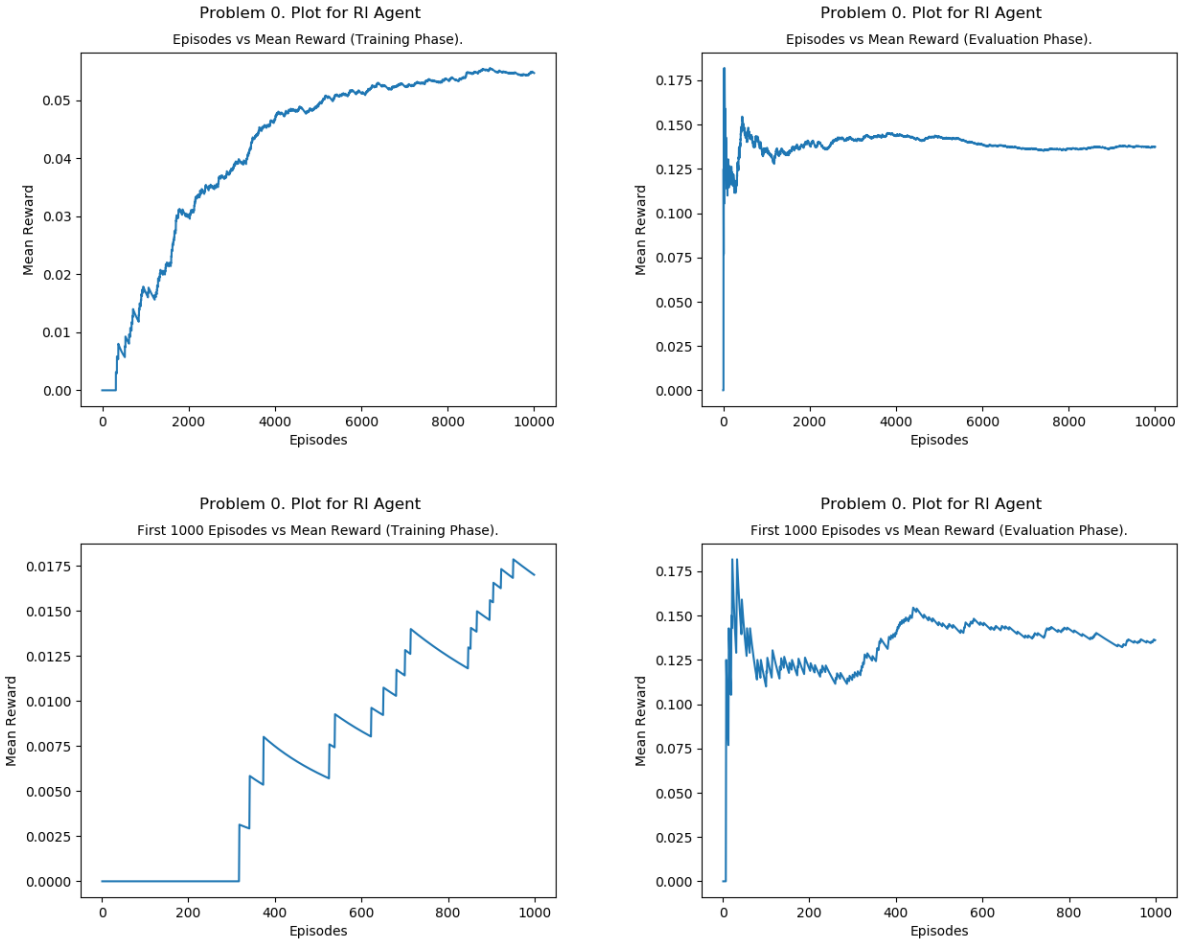


Figure 4: Evaluation plots for Problem 0. The full list of grids and best policies can be found in the appendix D.

Figure below show the utility table that was got from our reinforcement learning agent.

Figure 5: Problem 0 Utility table

6 Conclusions

Three agents were analyzed for the Loch Lomond Frozen Lake grid. Reinforcement Learning agent was compared to a random/senseless agent as well as a deterministic agent. Our Q-learning agent converges to a best policy in most of the environments in less than 10,000 episodes. However, we noticed that some of the policies are incomplete after this training period, suggesting that more episodes are needed in order to reach convergence. We saw how our model-free agent was able to obtain over 80% success rate in some of the training grid environments. Even though the amount of mean iterations is considerably higher for the q-learning agent than the simple agent, it is important to remember that simple agent is deterministic, while our reinforcement learning agent is stochastic.

Active reinforcement learning using a Q-learning algorithm is an interesting method for solving model-free problems. Further algorithms such as neural networks could be used in order to find a best policy and compare results to our Q-learning algorithm.

7 Disclaimer: Environment Modifications

In order to add additional flexibility and generic support to the agents, slight changes were made to the `uofgsocsai.py` file, the one containing the main `LochLomondEnv` class. The changes mentioned below were approved as long as justification was provided. The changes and justifications are as follows:

• Parameters `map_name_base`, `reward` and `path_cost` were added to the `LochLomondEnv` constructor. The default values are `8x8-base`, `1.0` and `0` respectively. The default values do not alter the functionality from the original file provided.

• Attributes `is_stochastic`, `reward_hole`, `reward` and `path_cost` were added to the `LochLomondEnv` class.

The reason of the changes for the constructor was to add flexibility to be able to test different scenarios without the need to modify the file every time a different variant was analyzed. The attributes were added to the class in order to be able to access them via the object (e.g. `env.path_cost`) and create a Markov Decision Process out of it. Markov Decision Processes were used in unit testing to compare the performance of our agent and the values generated by *Policy Iteration* and *Value Iteration* algorithms.

Finally, the way to assign en environment grid was changed from `MAPS_BASE[map_name_base]` to `copy.deepcopy(MAPS_BASE)[map_name_base]`, with the only purpose of being able to instantiate the `LochLomondEnv` more than once in a single run (e.g. `python run_rl.py 1,2,3,4,5,6,7`), which runs all the variants in a single run.

There may be other better ways of accomplishing the same without code changes, but due to the current lack of experience/knowledge in Python programming, time did not permit to find better ways for it.

8 References

• [1], [2], [3] Russell, Stuart J, and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, N.J: Prentice Hall, 1995. Print. (p.93, p.844 and p.840)

9 Appendices

9.1 Appendix A: Evaluation Plots for Loch Lomond Frozen Lake 4x4 Grids

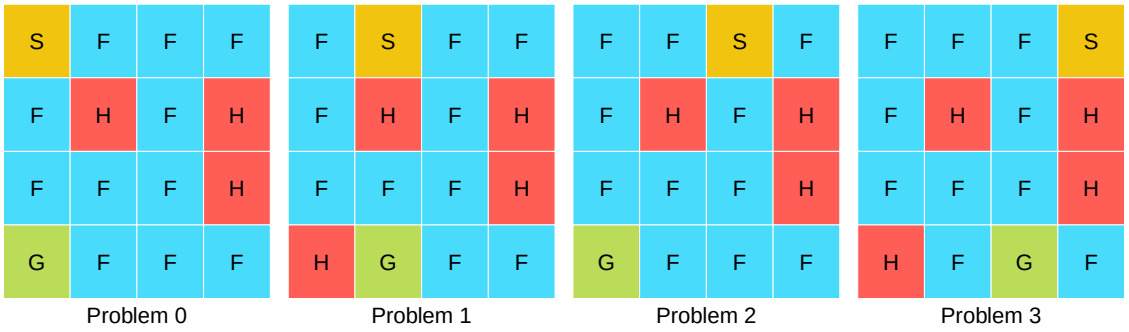


Figure 6: Loch Lomond Frozen Lake 4x4 grids

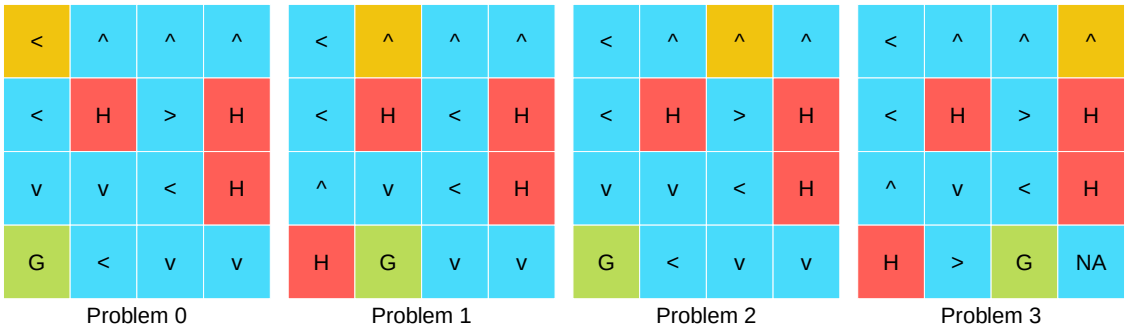


Figure 7: Policy π found by RL agent for 4x4 grids

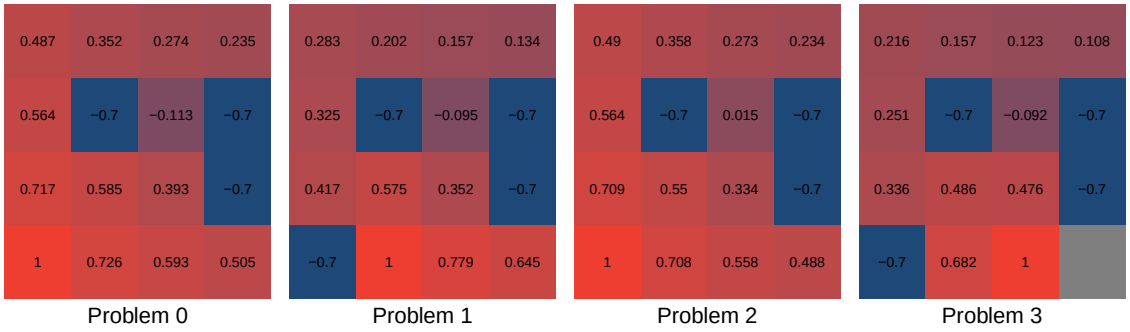


Figure 8: Utilities of the states, given policy π in the 4x4 grids (Problem 0 to 3)

9.2 Appendix B: Evaluation Plots for Loch Lomond Frozen Lake 8x8 Grids

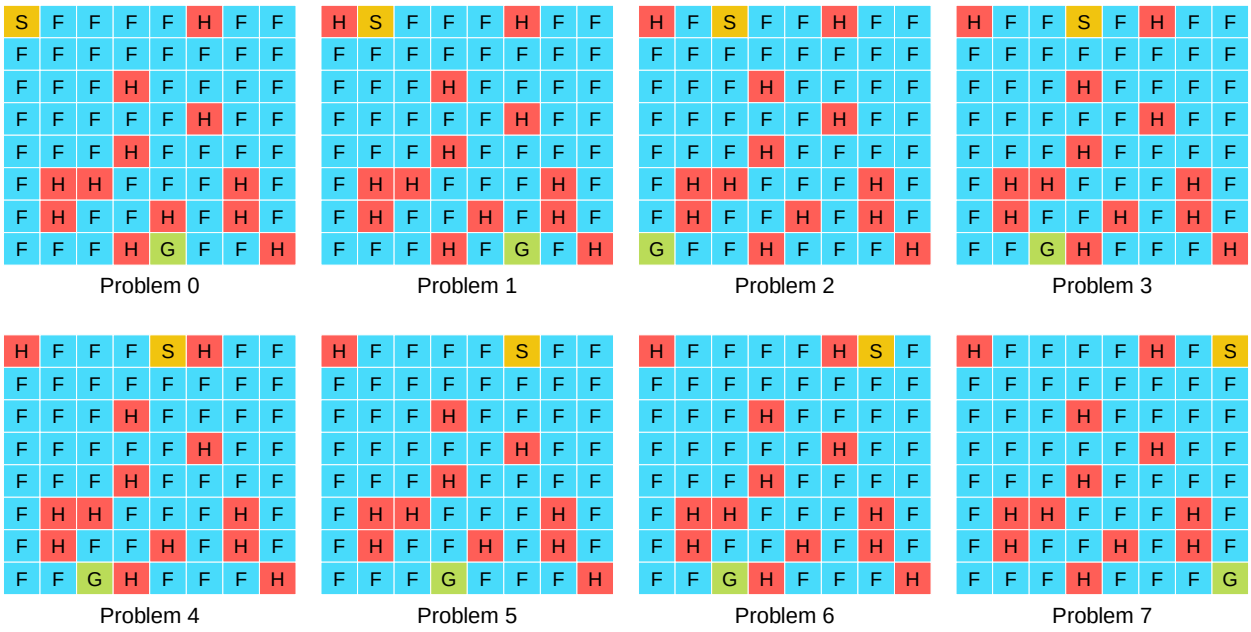


Figure 9: Loch Lomond Frozen Lake 8x8 grids

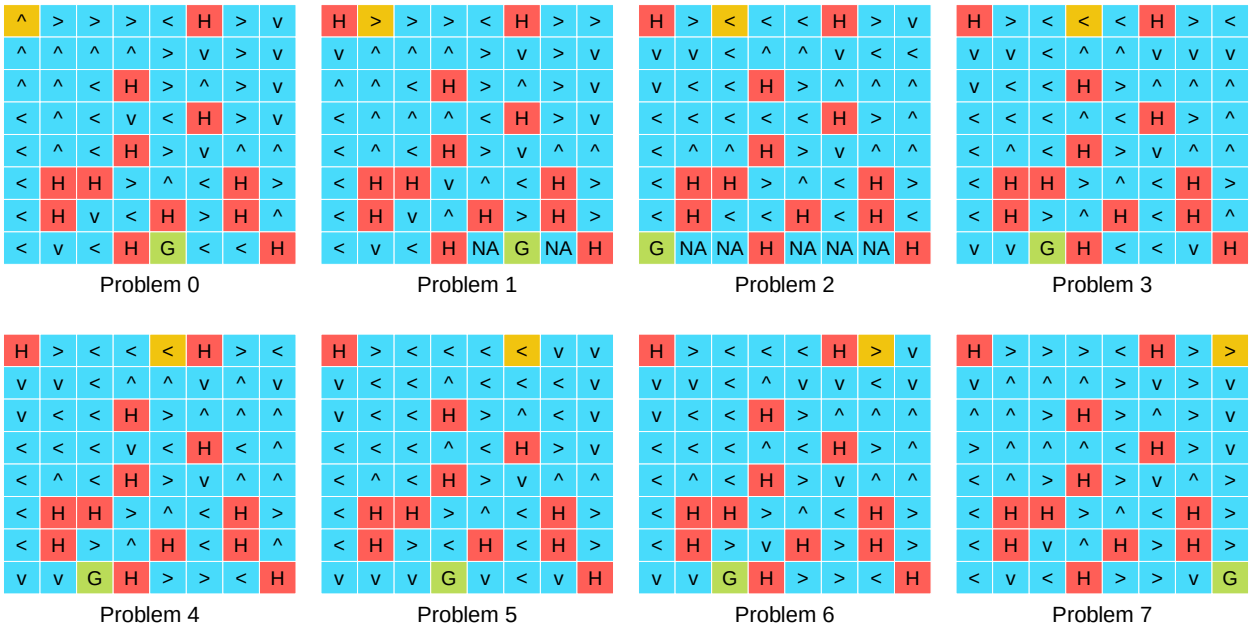
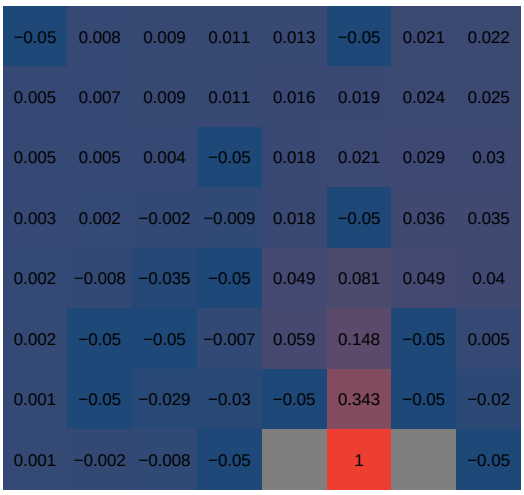


Figure 10: Policy π found by our reinforcement learning agent

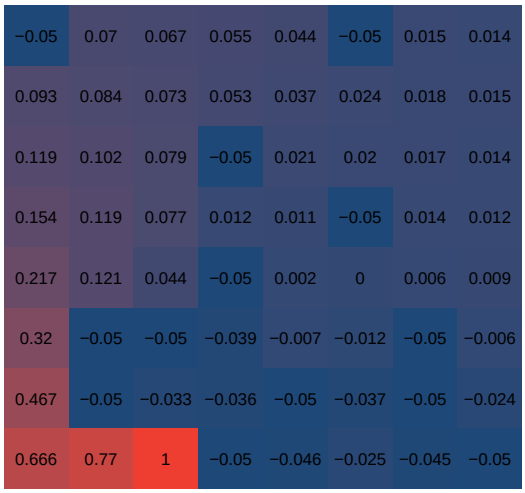
Problem 0



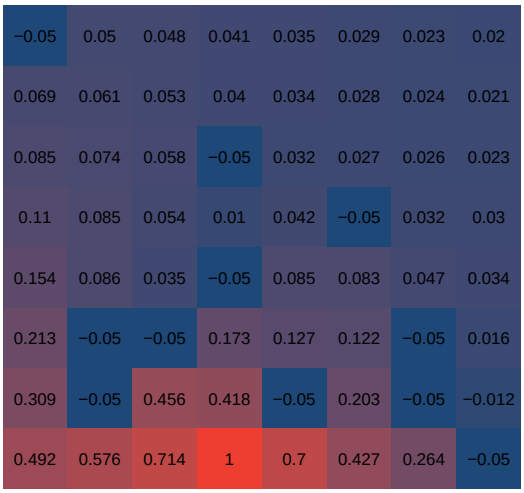
Problem 1

Problem 2

Problem 3



Problem 4



Problem 5

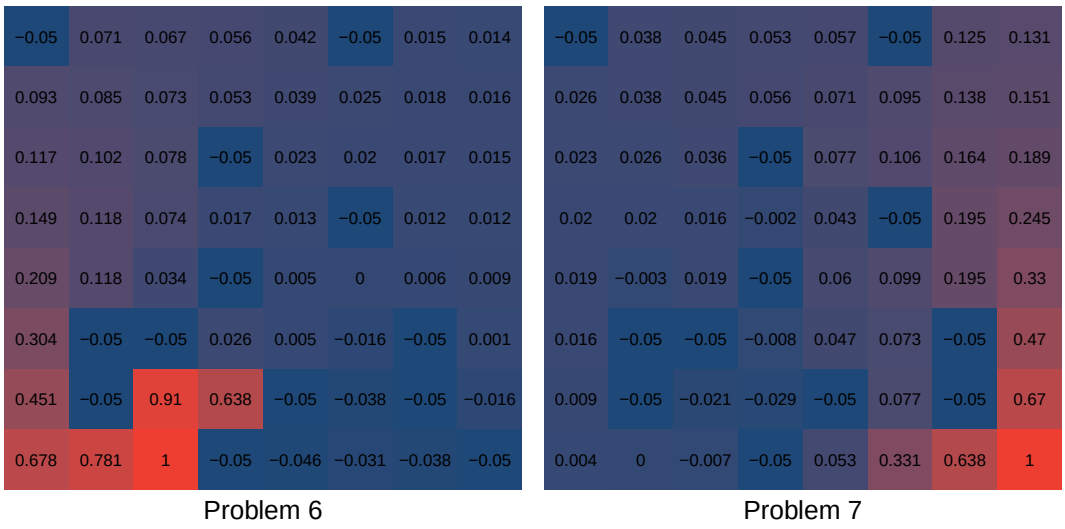
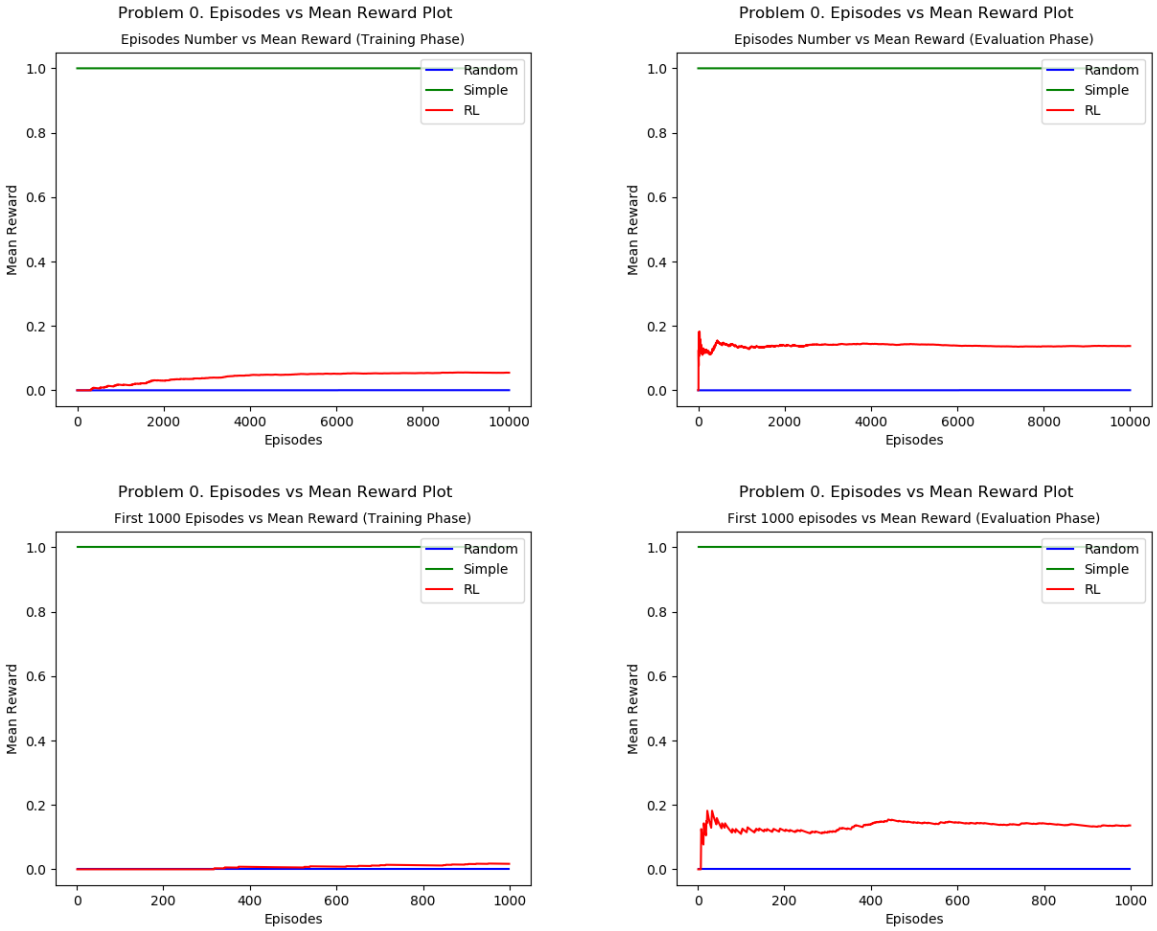
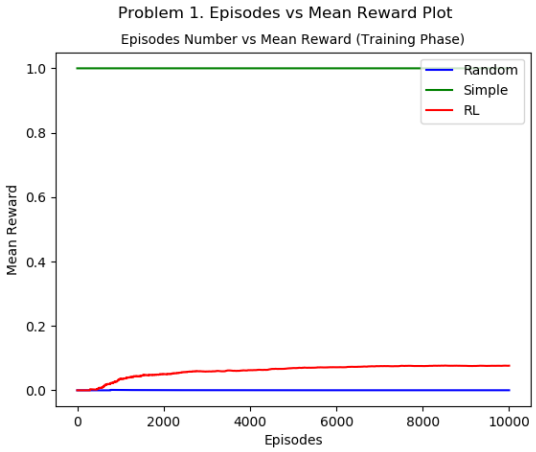


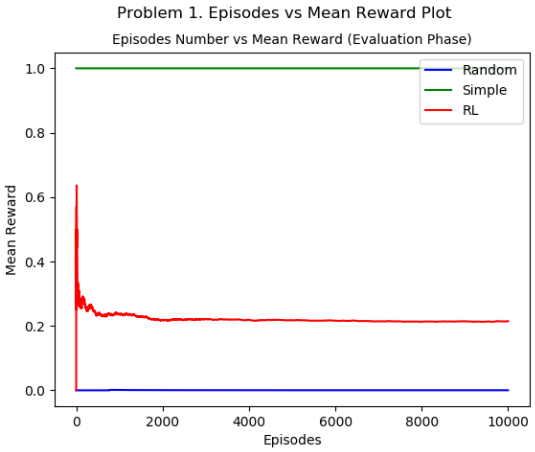
Figure 11: Utility tables our reinforcement learning agent

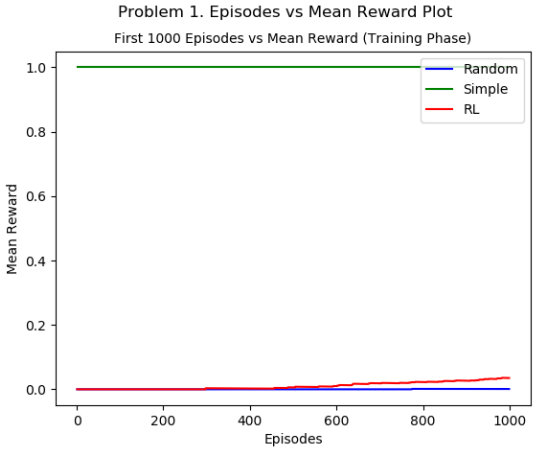
9.3 Appendix C: Evaluation of agents

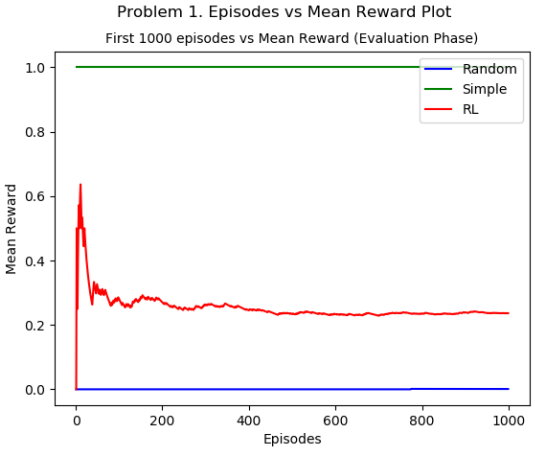
The plots with the mean reward vs episodes number are presented below.

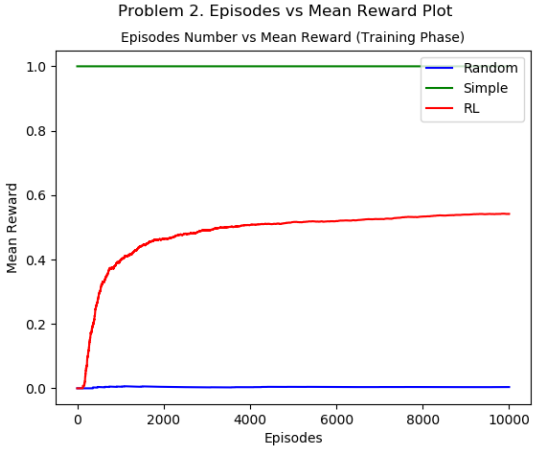


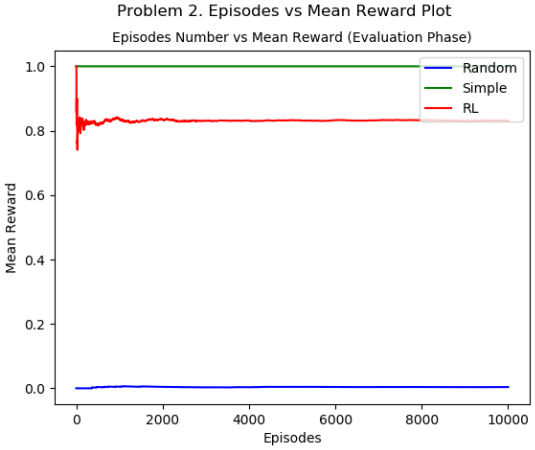

Problem 1. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

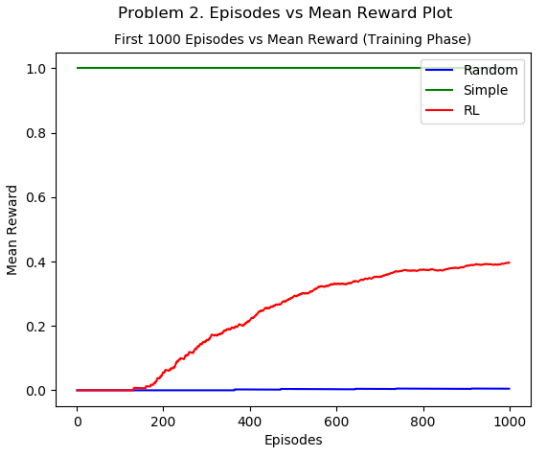

Problem 1. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

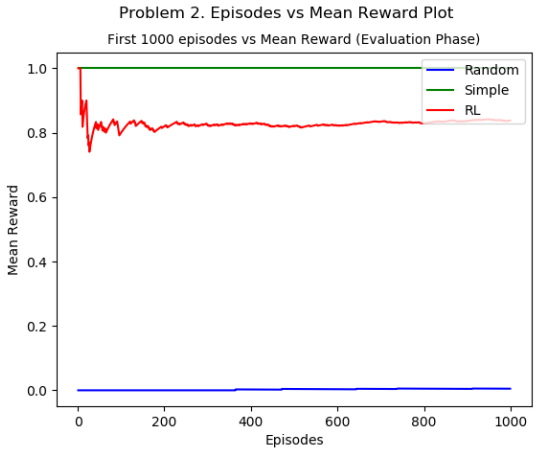

Problem 1. Episodes vs Mean Reward Plot
First 1000 Episodes vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

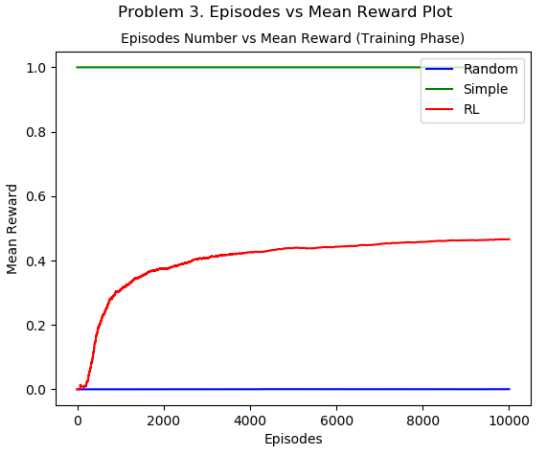

Problem 1. Episodes vs Mean Reward Plot
First 1000 episodes vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

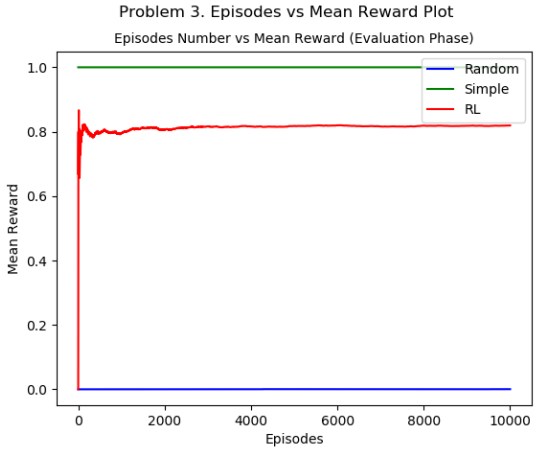

Problem 2. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

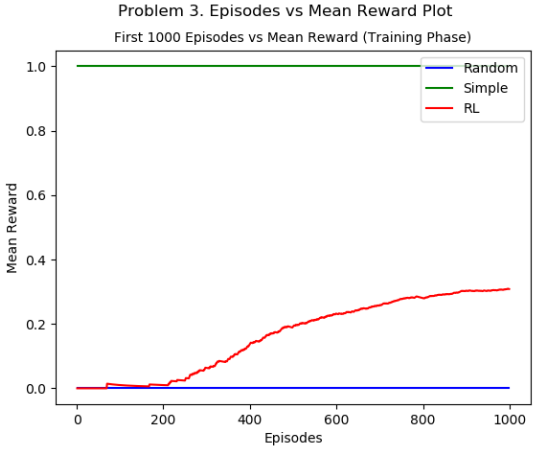

Problem 2. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

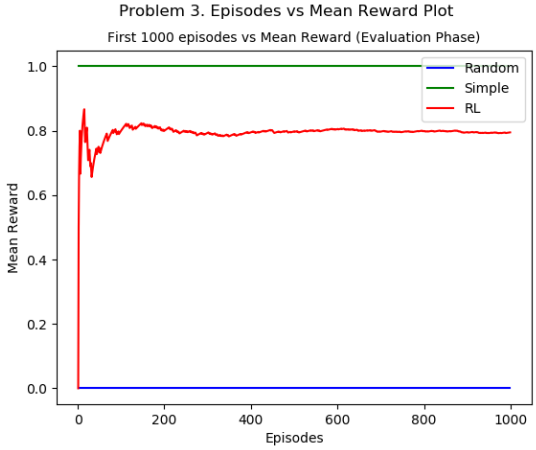

Problem 2. Episodes vs Mean Reward Plot
First 1000 Episodes vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

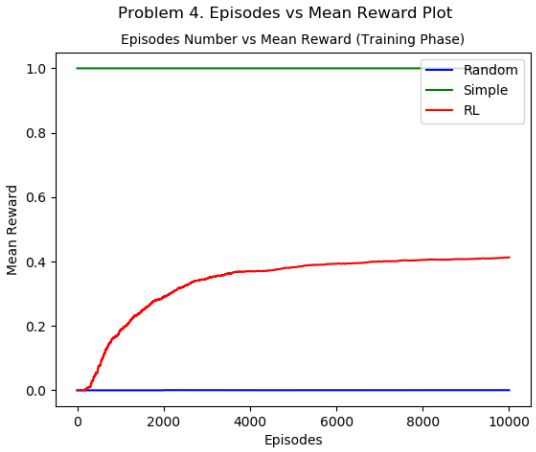

Problem 2. Episodes vs Mean Reward Plot
First 1000 episodes vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

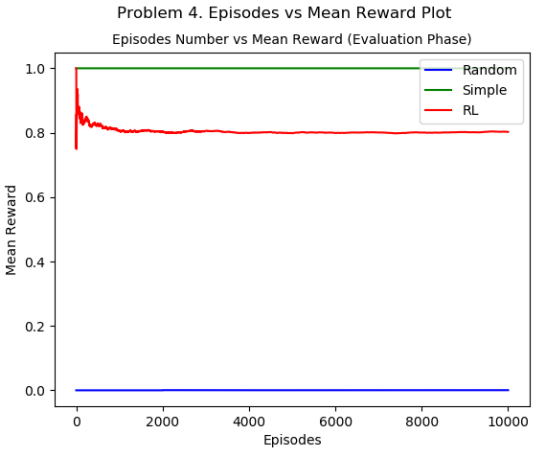

Problem 3. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

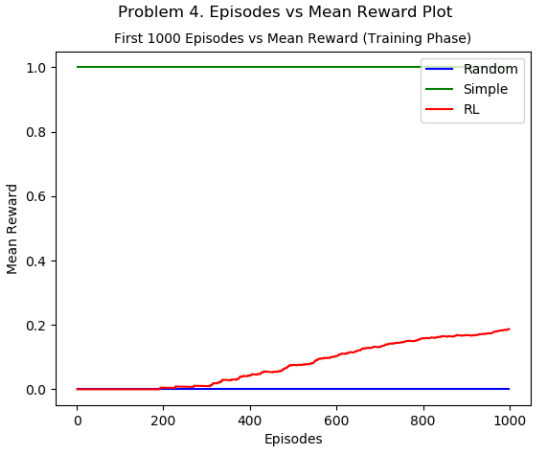

Problem 3. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

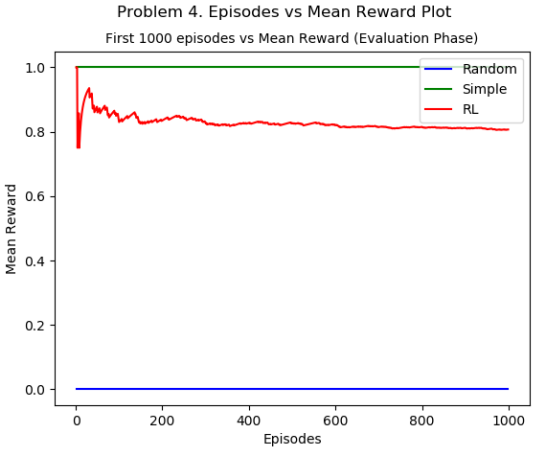

Problem 3. Episodes vs Mean Reward Plot
First 1000 Episodes vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

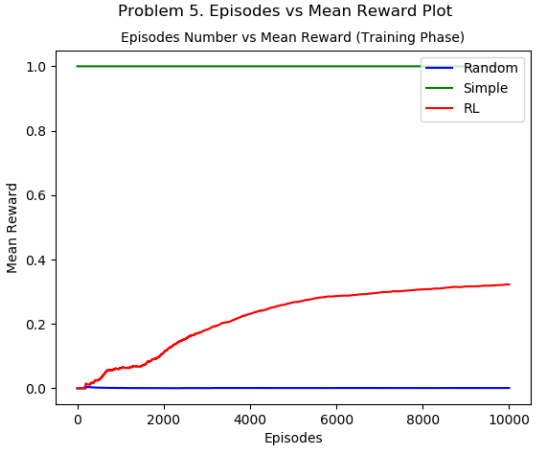

Problem 3. Episodes vs Mean Reward Plot
First 1000 episodes vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

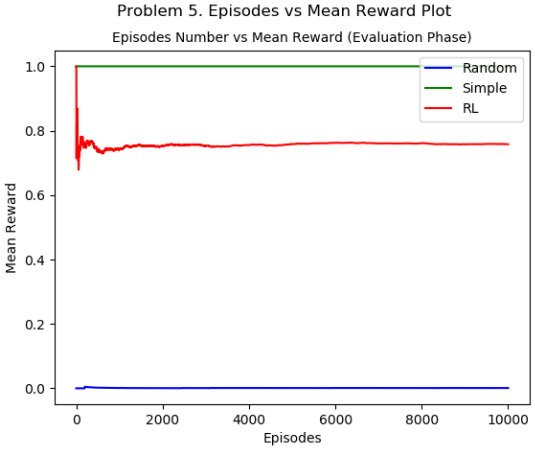

Problem 4. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

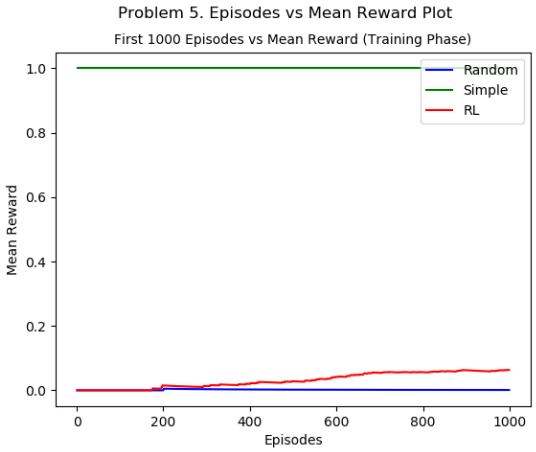

Problem 4. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

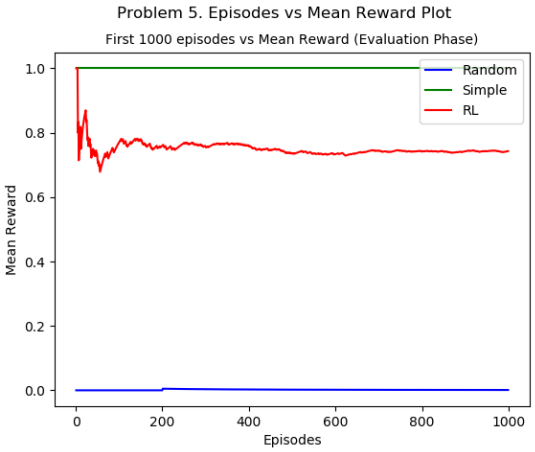

Problem 4. Episodes vs Mean Reward Plot
First 1000 Episodes vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

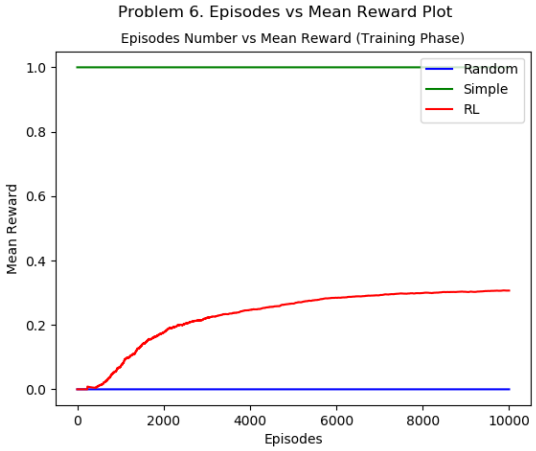

Problem 4. Episodes vs Mean Reward Plot
First 1000 episodes vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

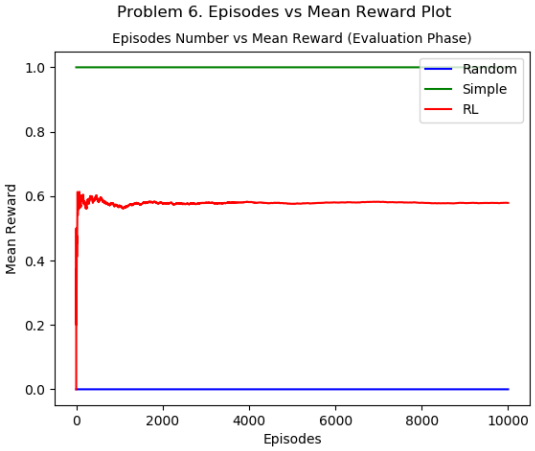

Problem 5. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

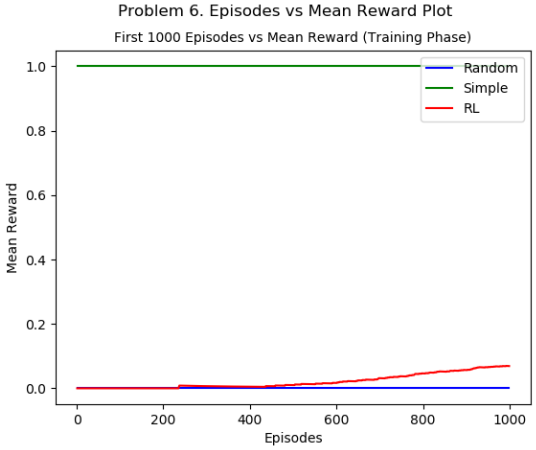

Problem 5. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

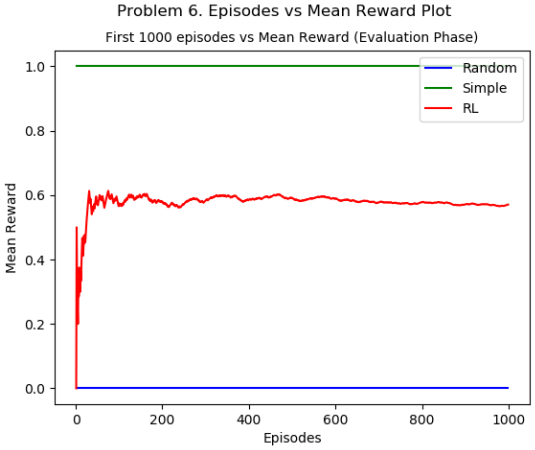

Problem 5. Episodes vs Mean Reward Plot
First 1000 Episodes vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

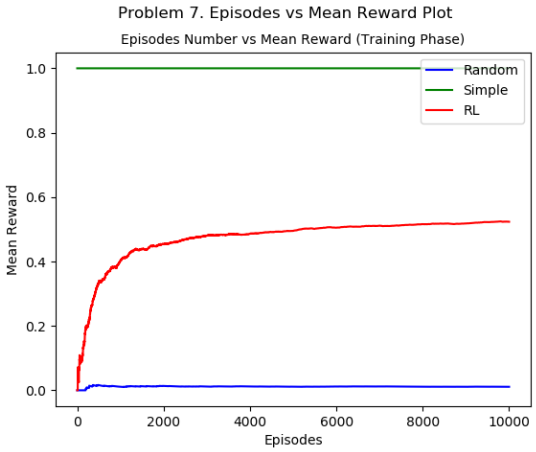

Problem 5. Episodes vs Mean Reward Plot
First 1000 episodes vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

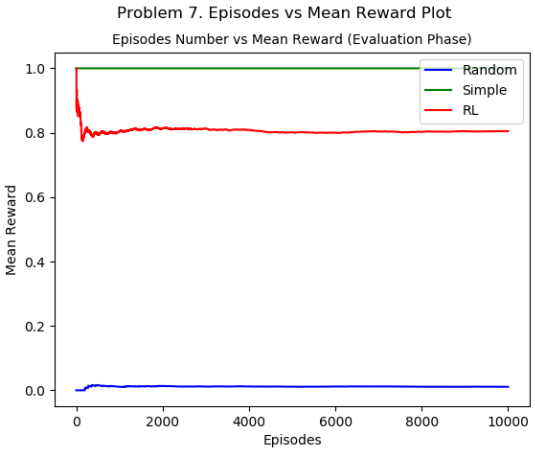

Problem 6. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes

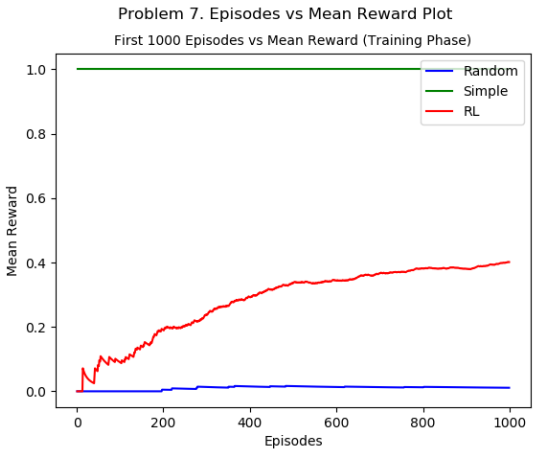

Problem 6. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes

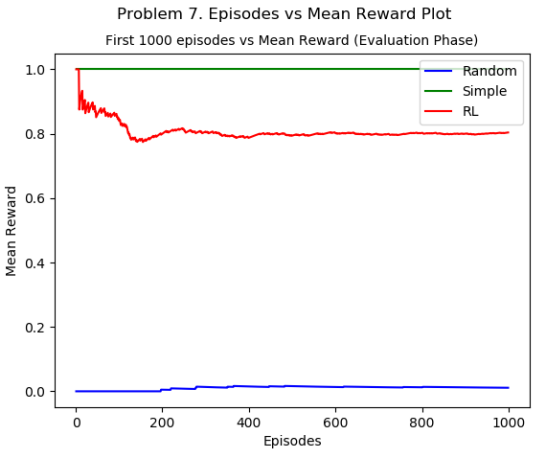

Problem 6. Episodes vs Mean Reward Plot
First 1000 Episodes vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes


Problem 6. Episodes vs Mean Reward Plot
First 1000 episodes vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes


Problem 7. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000


Problem 7. Episodes vs Mean Reward Plot
Episodes Number vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000


Problem 7. Episodes vs Mean Reward Plot
First 1000 Episodes vs Mean Reward (Training Phase)
Random
Simple
RL
Mean Reward
Episodes
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000


Problem 7. Episodes vs Mean Reward Plot
First 1000 episodes vs Mean Reward (Evaluation Phase)
Random
Simple
RL
Mean Reward
Episodes
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000

9.4 Appendix D: Evaluation of our Reinforcement Learning Agent

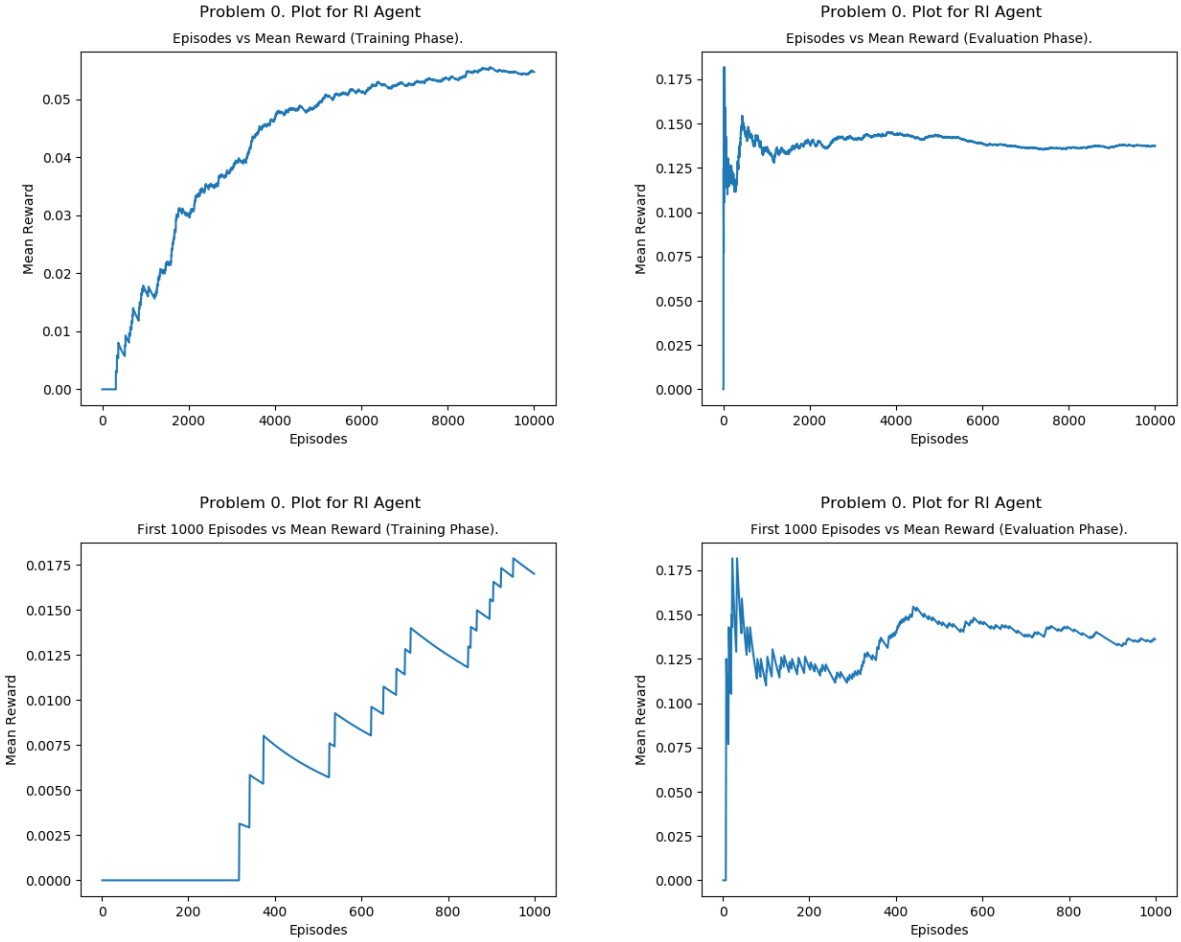
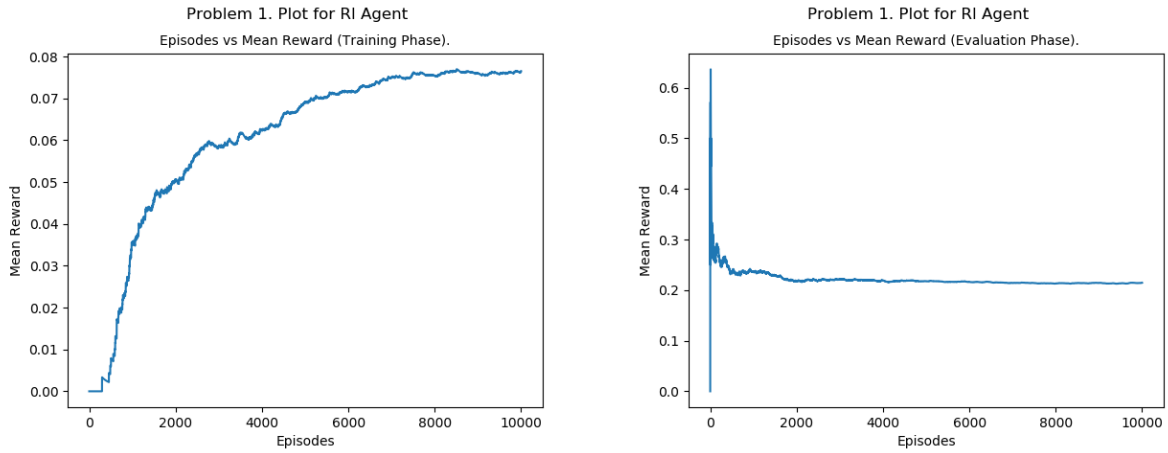


Figure 12: Evaluation Plots for Problem 0



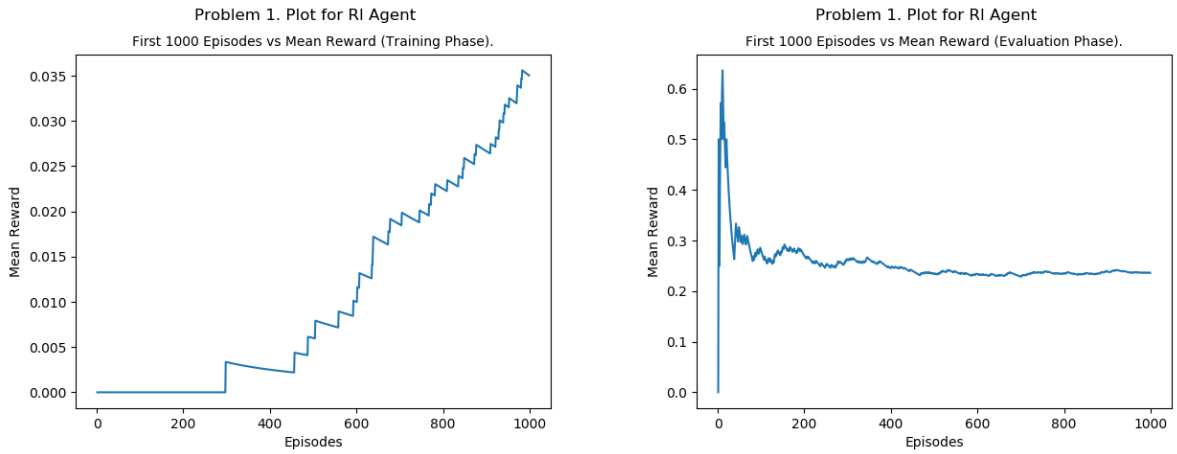


Figure 13: Evaluation Plots for Problem 1

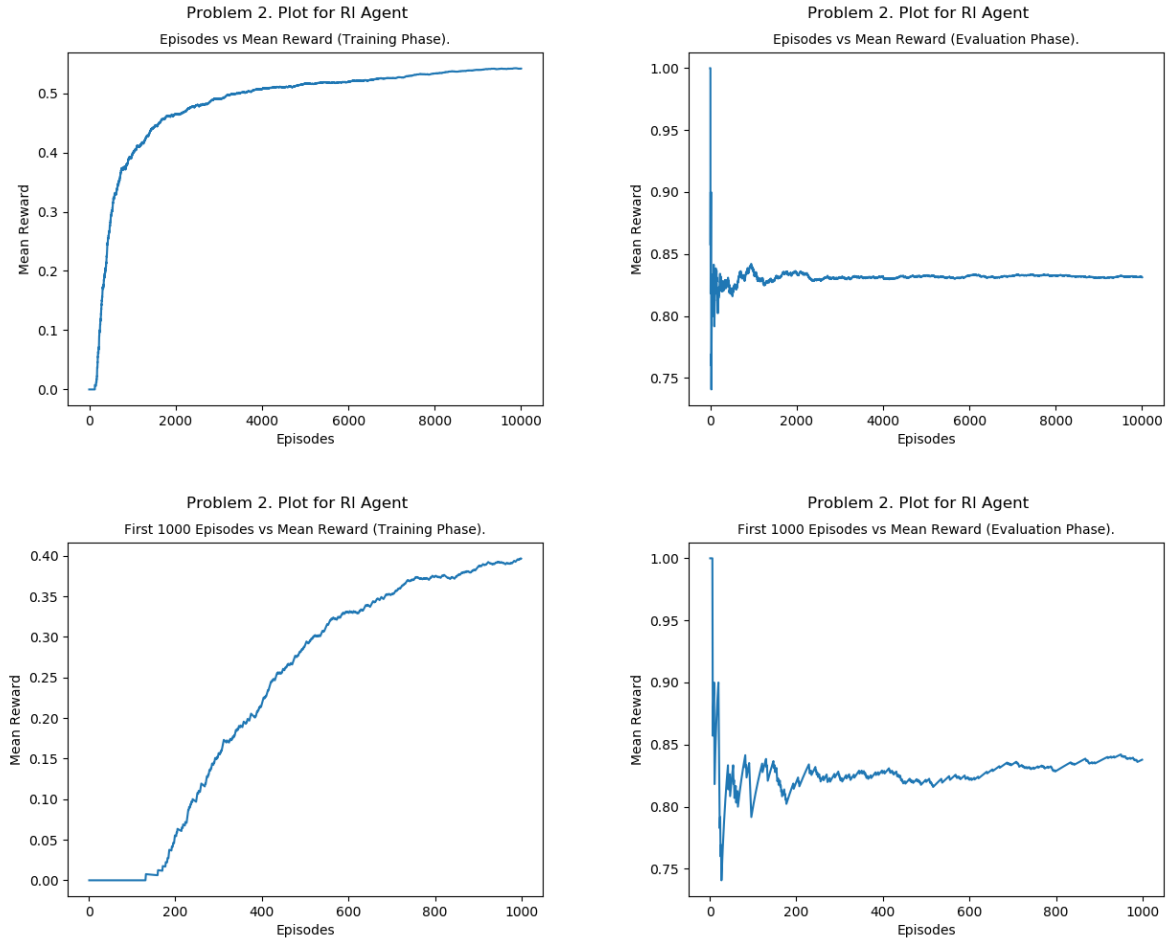


Figure 14: Evaluation Plots for Problem 2

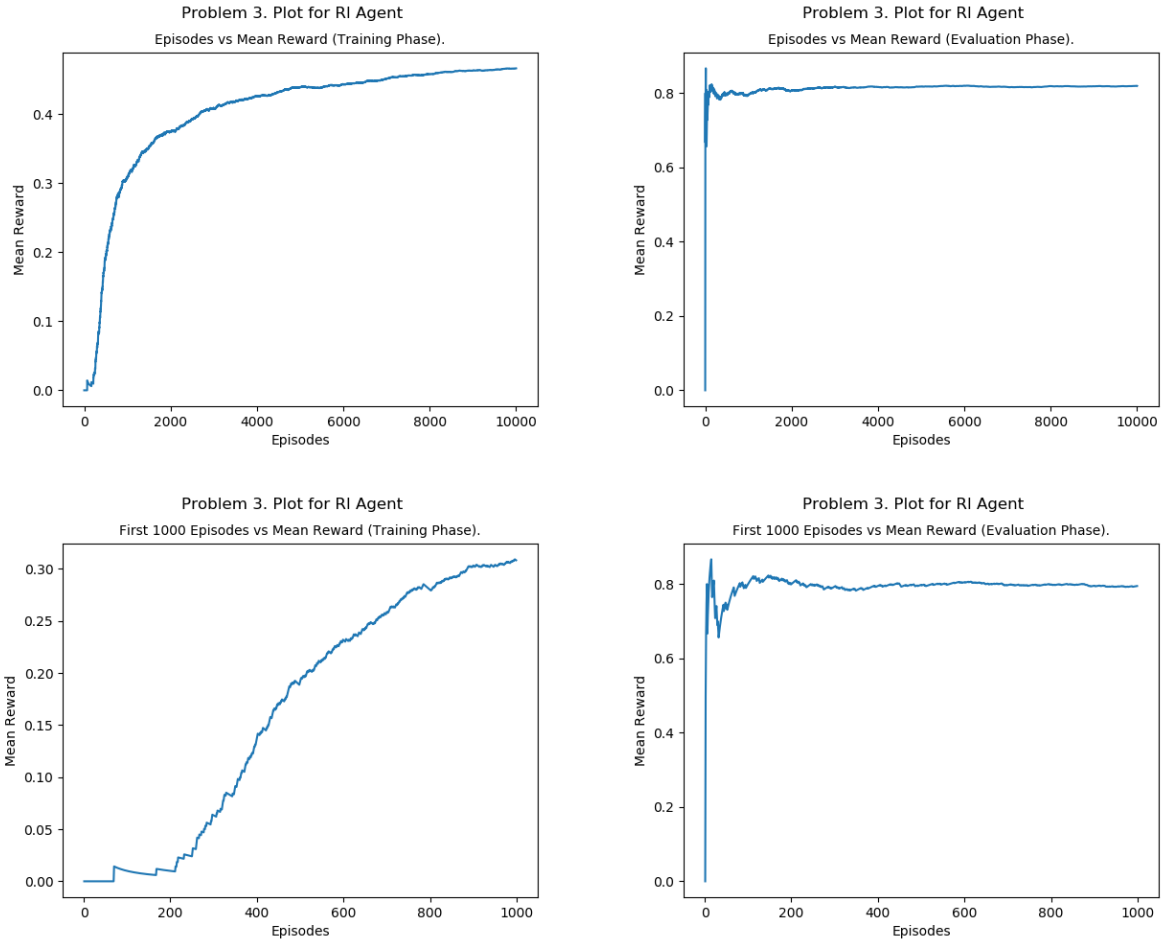
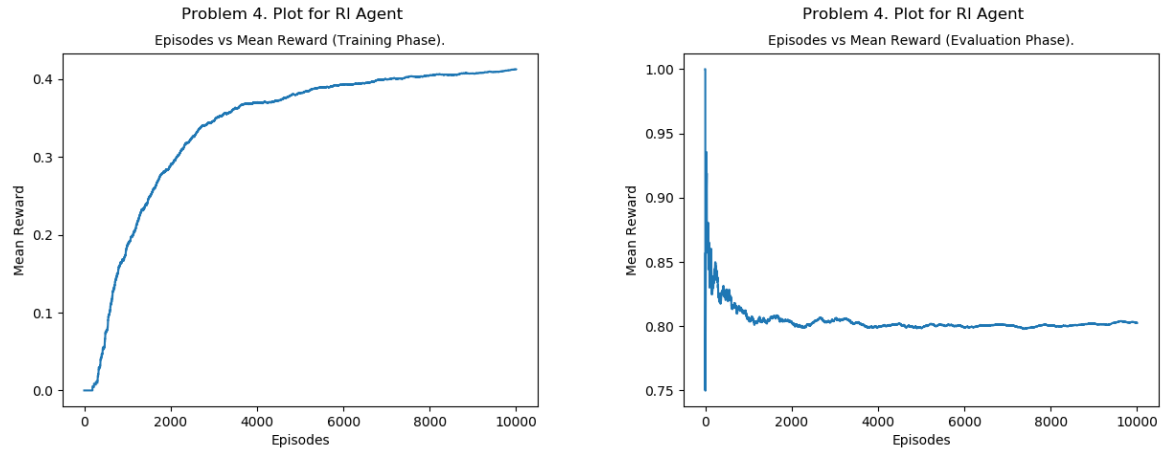


Figure 15: Evaluation Plots for Problem 3



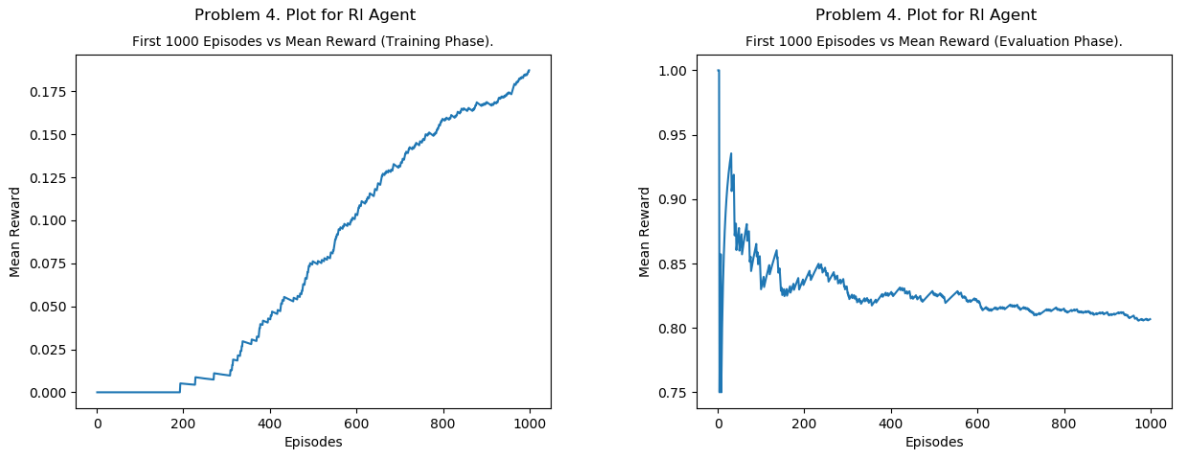


Figure 16: Evaluation Plots for Problem 4

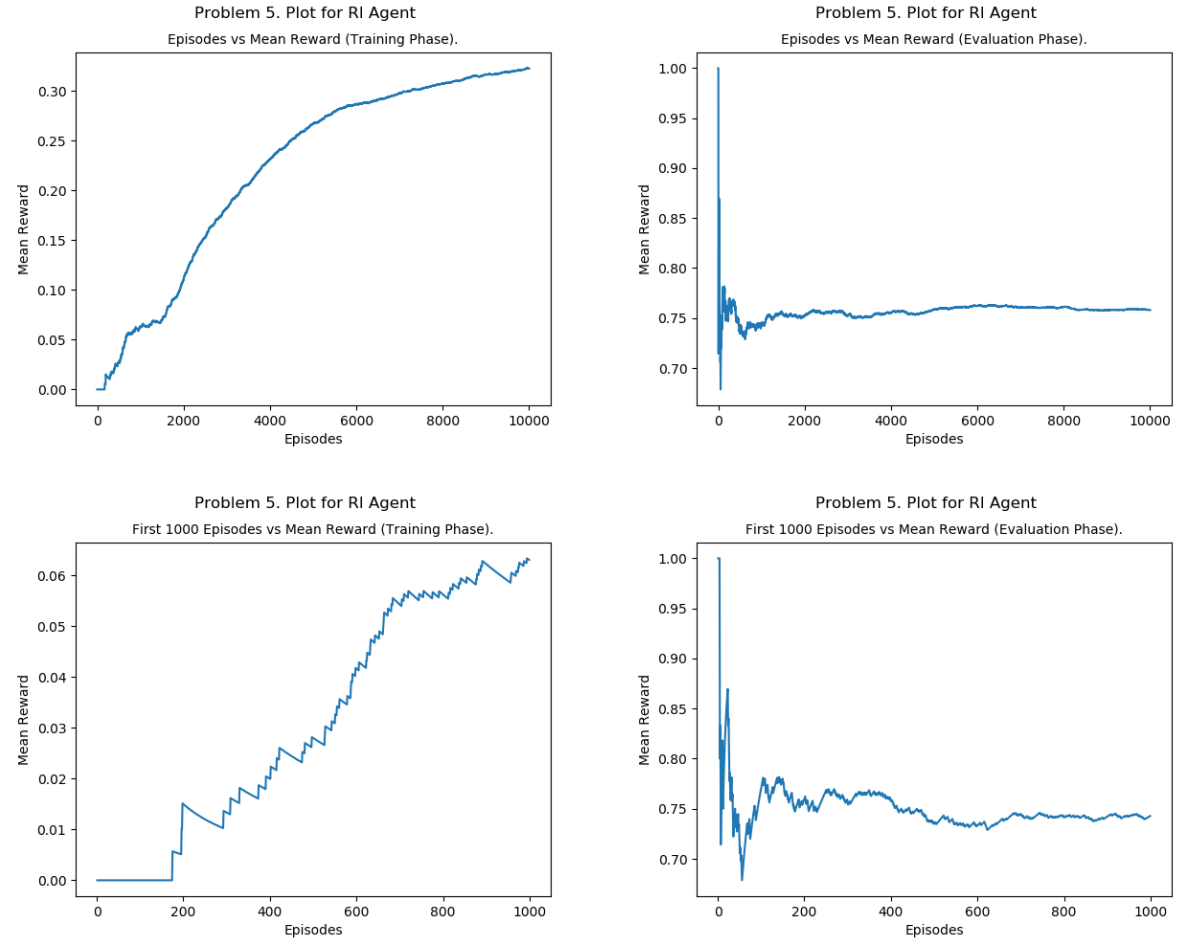


Figure 17: Evaluation Plots for Problem 5

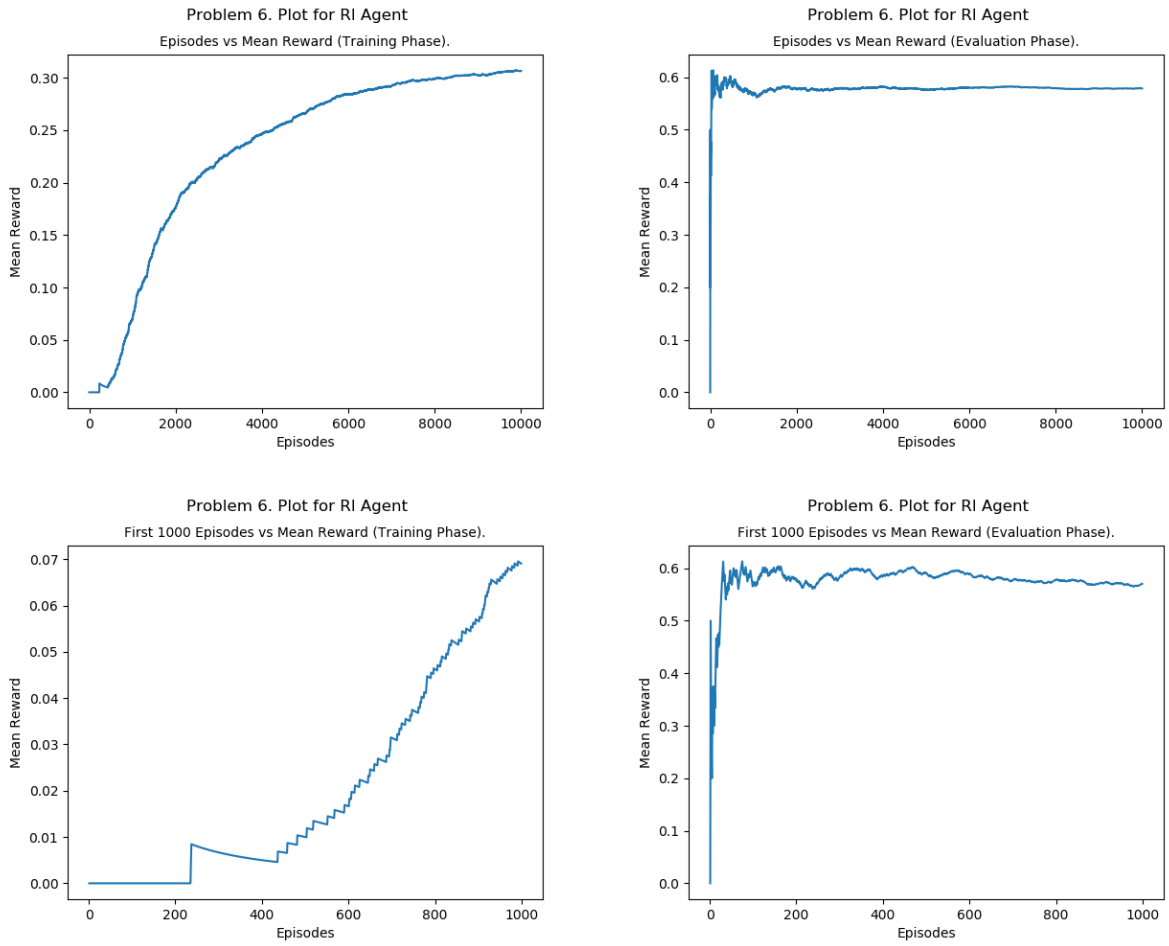
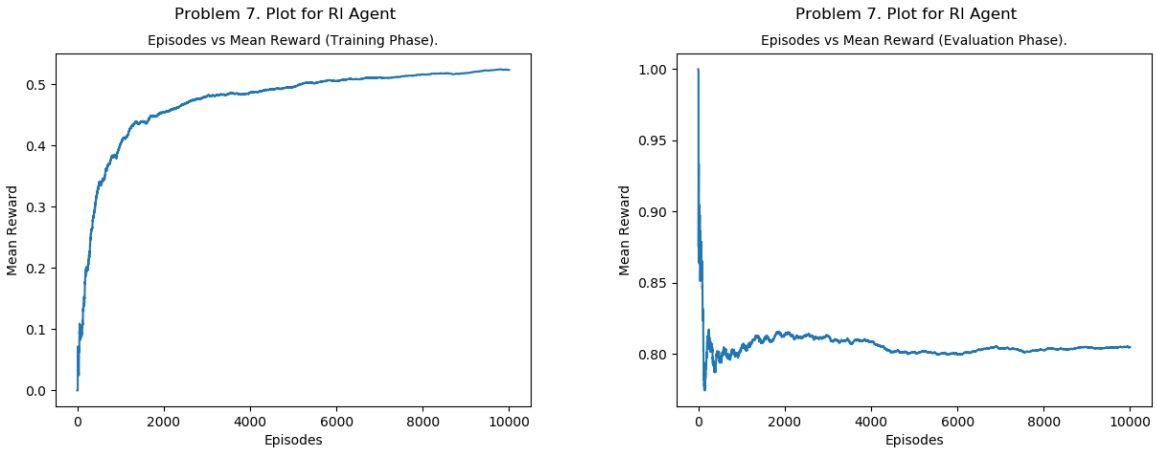


Figure 18: Evaluation Plots for Problem 6



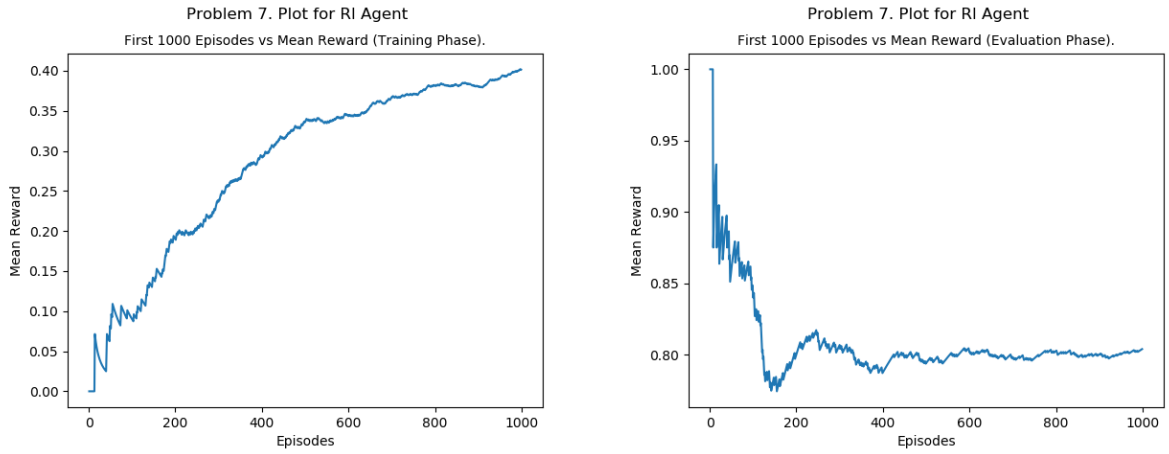


Figure 19: Evaluation Plots for Problem 4