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.

The goal of this report is to design, implement and evaluate three diferent virtual agents which 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.

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), 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.

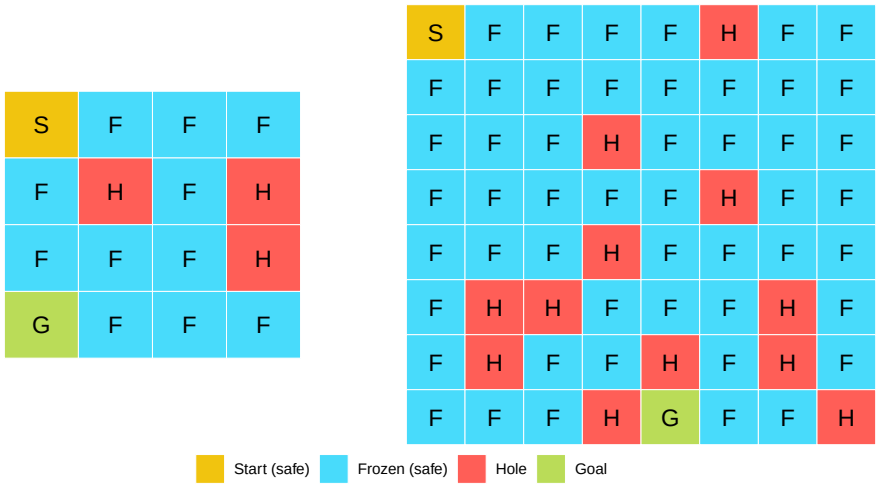


Figure 1: Loch Lomond Frozen Lake Grid Scenarios, 4x4 (left) and 8x8 (right)

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.

In the stochastic environments (which will be the case for the random and reinforcement learning agents), 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. Our goal state (G) has reward of *1.0*, the obstacles (H) have a reward of *-0.05*, and the rest of non terminal states have a reward of *0*.

2.2 Task Environment and 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

Further explanation of the task environment is found below:
• **Random Agent**: No knowledge about the current state nor state-space in general.
• **Simple Agent**: Perfect information about the current state and thus available actions in that state. Full knowledge of the state-space in general as well as full knowledge of the rewards, goals and obstacles.
• **Reinforcement Learning Agent**: Perfect information about the current state and thus available actions in that state; no prior knowledge about the state-space in general.

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:

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

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 (PAGE 93):
• $g(n)$ 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.

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[1], we make use of the update Bellman equation:

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

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. 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

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.

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

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

For our reinforcement learning agent, before running our evaluation phase, we run a "training phase", which helps to find the optimal policy for 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).

Additionally, a custom mapper (`EnvMDP` within `helpers.py` file) was implemented in order to map the enviroment from `Open AI Gym` to a `Grid MDP`. The idea was to get an optimal policy π as well as a utilities table through *Policy Iteration* and *Value Iteration* and compare it to the results obtained by our agent. It is important to note that the results obtained from the *Policy Iteration* and *Value Iteration* algorithms are completely independent from our reinforcement learning agent, and they are only provided as a measure of evaluation.

4.4 Environment Modifications (Additional Section)

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. Even though a Markov Decision Process was out of the scope of this project, an inhouse mapper from `Open AI Gym` environment to `Grid MDP` with the only

purpose of doing a sanity check between the final U and policy from the Reinforcement Learning agent and the ones that a *Policy Iteration* and *Value Iteration* algorithm would provide.

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.

5 Evaluation

Every agent produces different evaluation files that will be created inside the `out` folder. Each agent was evaluated over 10000 episodes and their reward/mean reward was calculated. For the reinforcement learning agent, there was a previous learning phase where the agent learns the best policy through the Q-value algorithm.

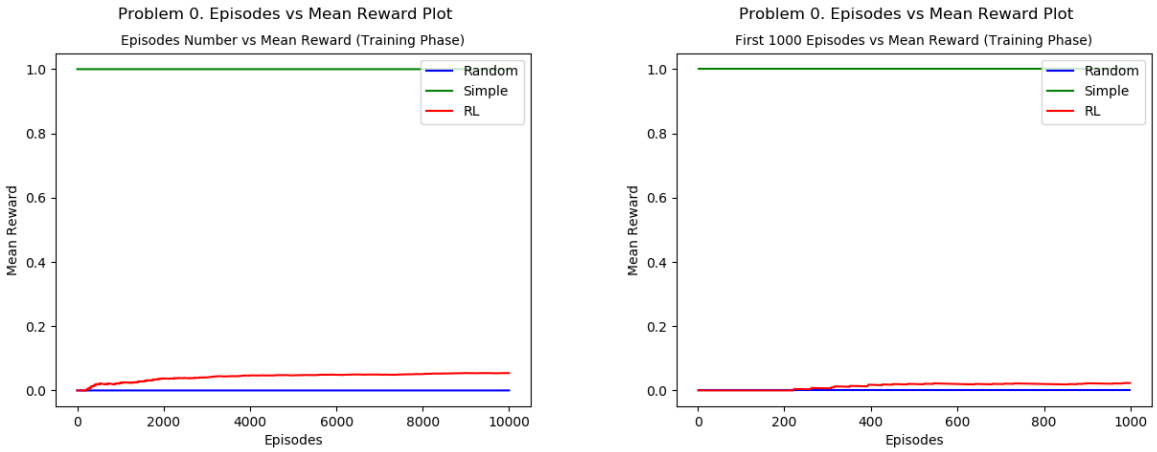
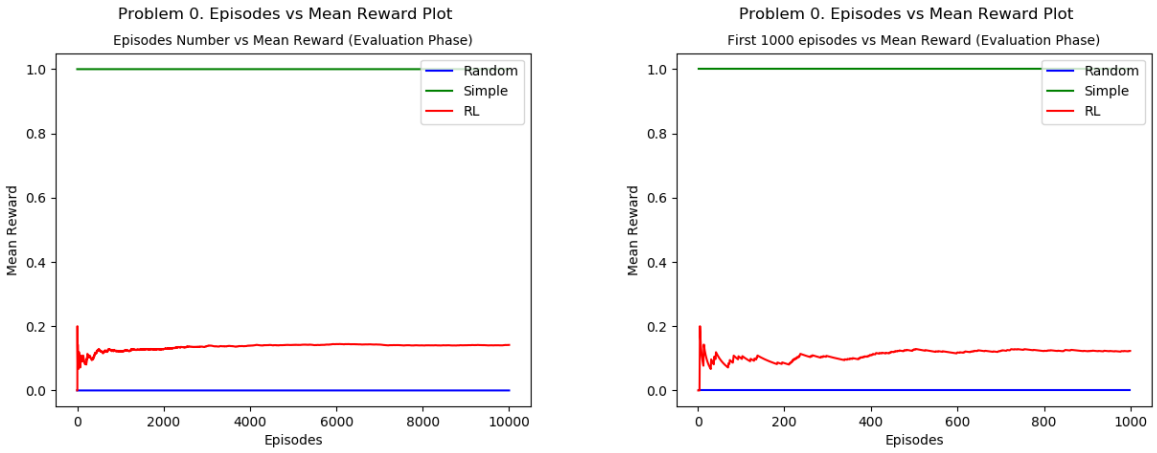


Figure 2: Evaluation Plots for Problem 0 (Training Phase)



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. We saw an important improvement

7 Appendices

7.1 Appendix A: Evaluation Plots for 4x4 Grids

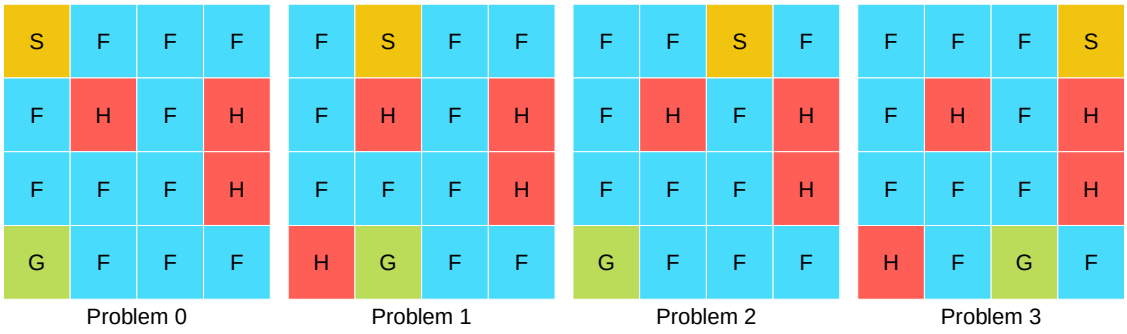


Figure 3: Loch Lomond Frozen Lake 4x4 grids

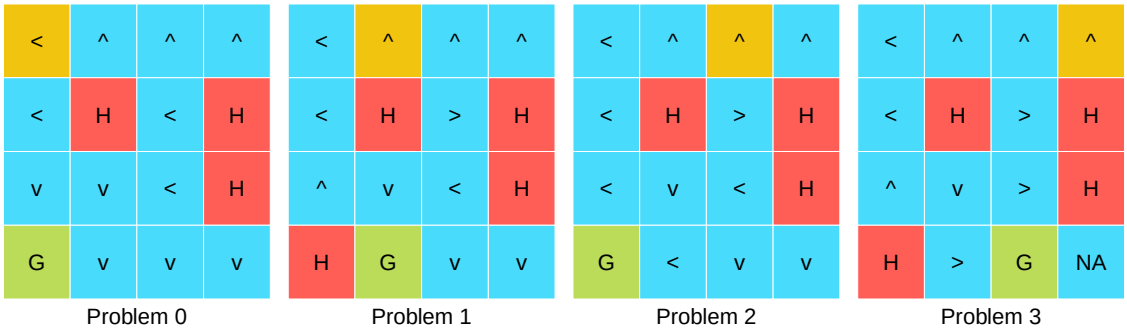


Figure 4: Policy π found by RL agent for 4x4 grids

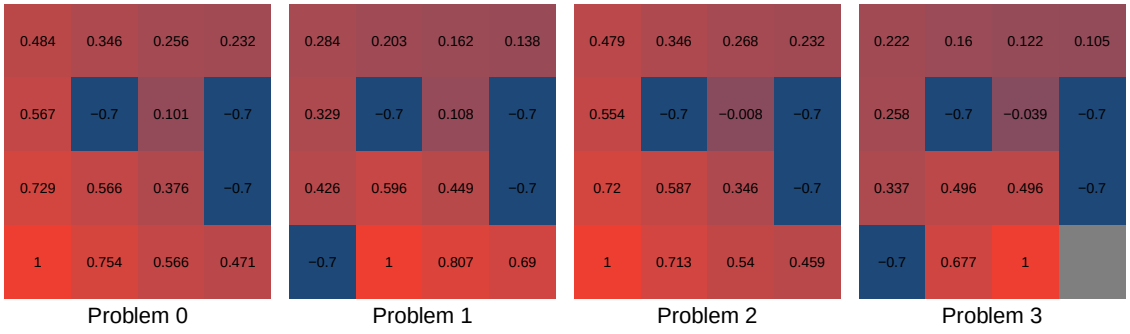
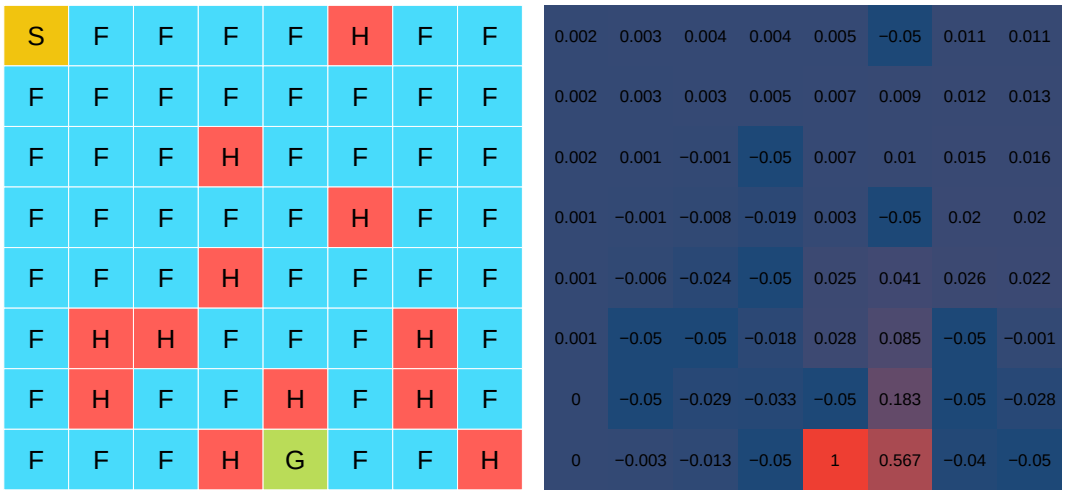


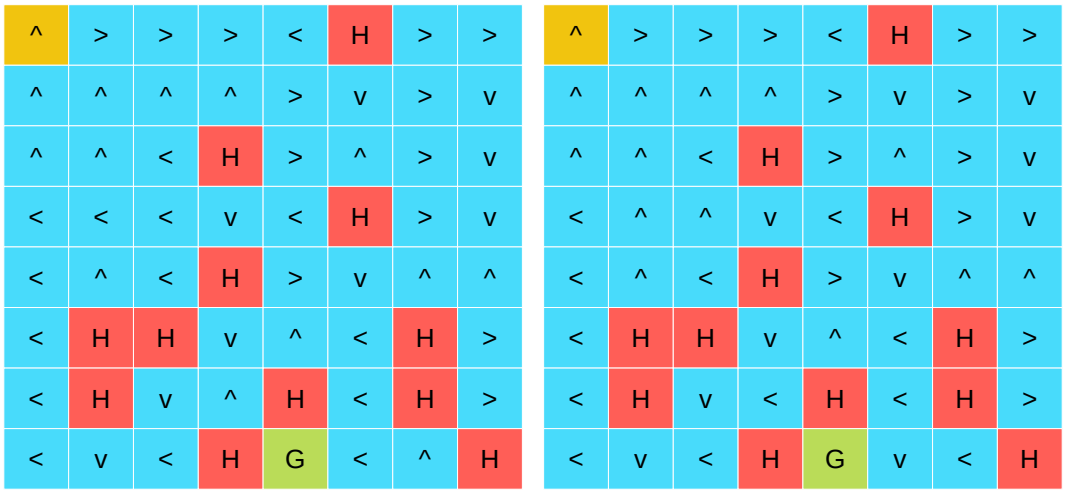
Figure 5: Utilities of the states, given policy π in the 4x4 grids (Problem 0 to 3)

7.2 Appendix B: Evaluation Plots for 8x8 Grids



8x8 Grid Problem 0 8x8 Problem 0 − Utilities Table

Figure 6: My caption here



Problem 0 Problem 0.

Figure 7: Policy π found by our agent (left) and by policy iteration algorithm (right) for problem 0

7.3 Appendix C: Evaluation of agents

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

7.4 Appendix C: Evaluation of Reinforcement Learning Agent

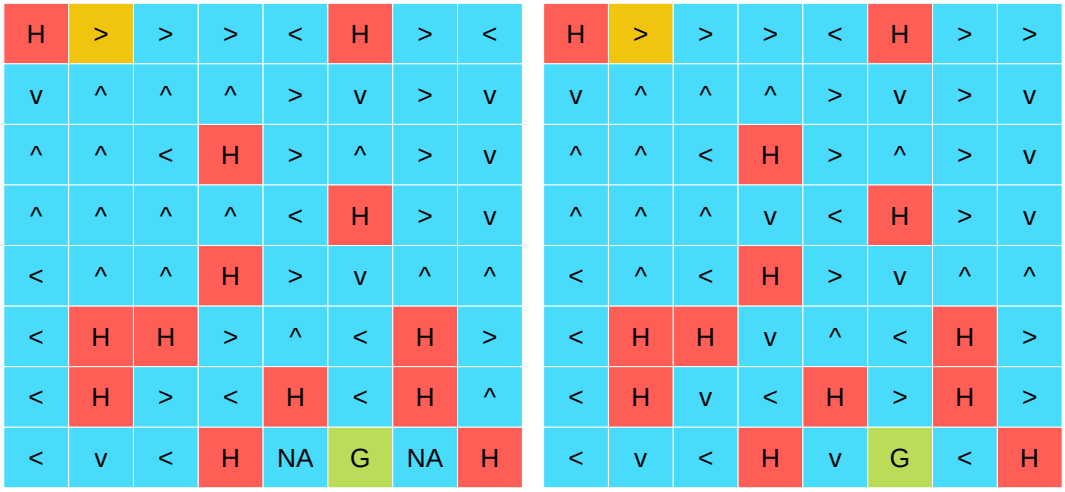


Figure 8: Policy π found by our agent (left) and by policy iteration algorithm (right) for problem 1

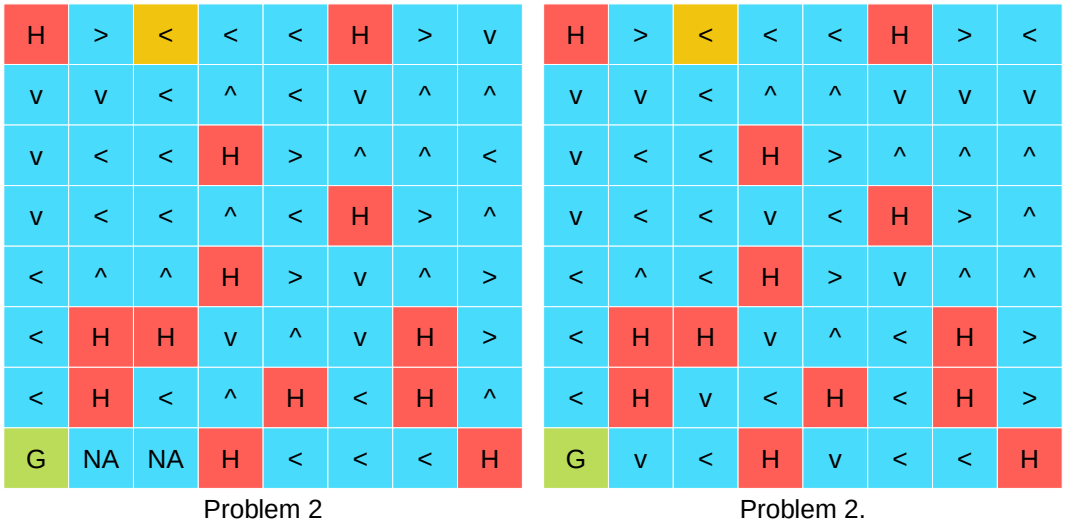


Figure 9: Policy π found by our agent (left) and by policy iteration algorithm (right) for problem 2

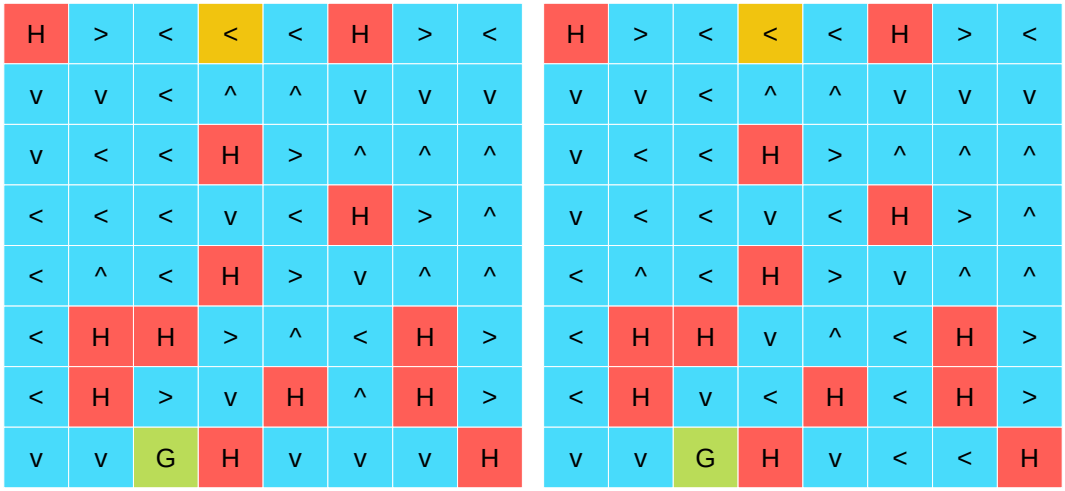


Figure 10: Policy π found by our agent (left) and by policy iteration algorithm (right) for problem 3

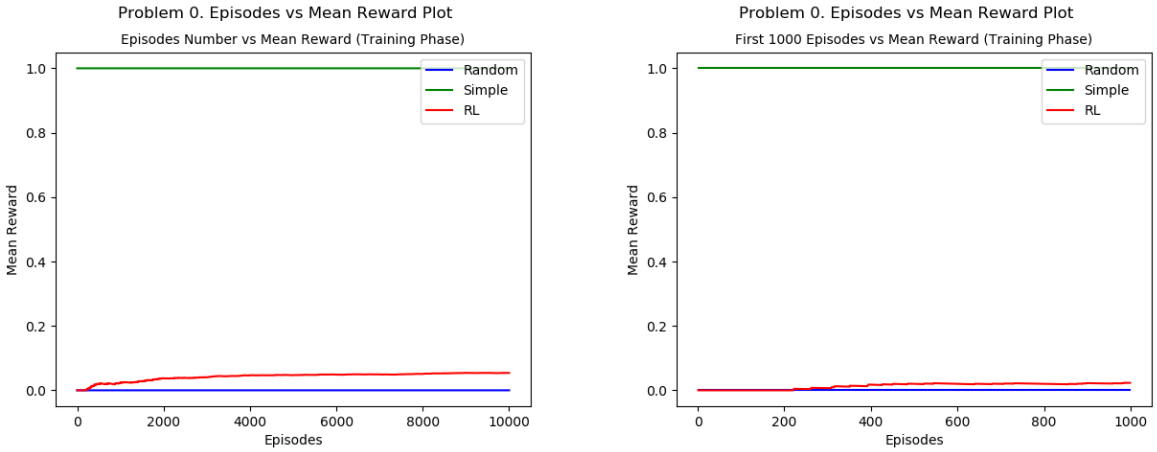


Figure 11: Evaluation Plots for Problem 0 (Training Phase)

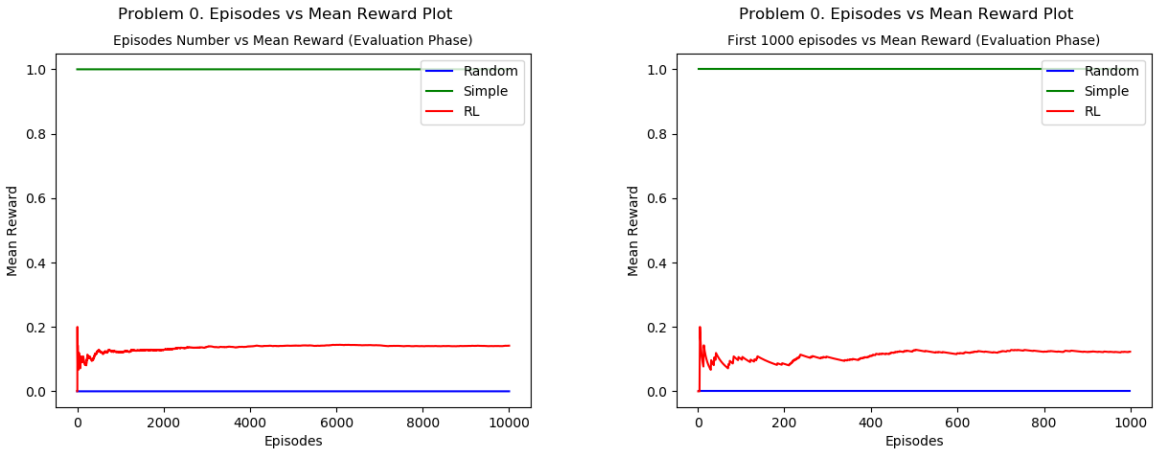


Figure 12: Evaluation Plots for Problem 0 (Evaluation Phase)

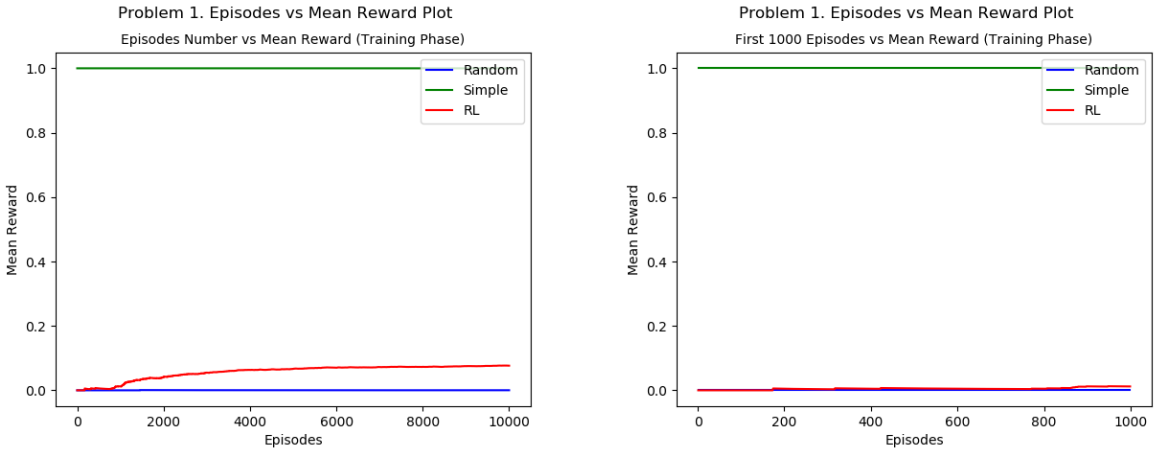


Figure 13: Evaluation Plots for Problem 1 (Training Phase)

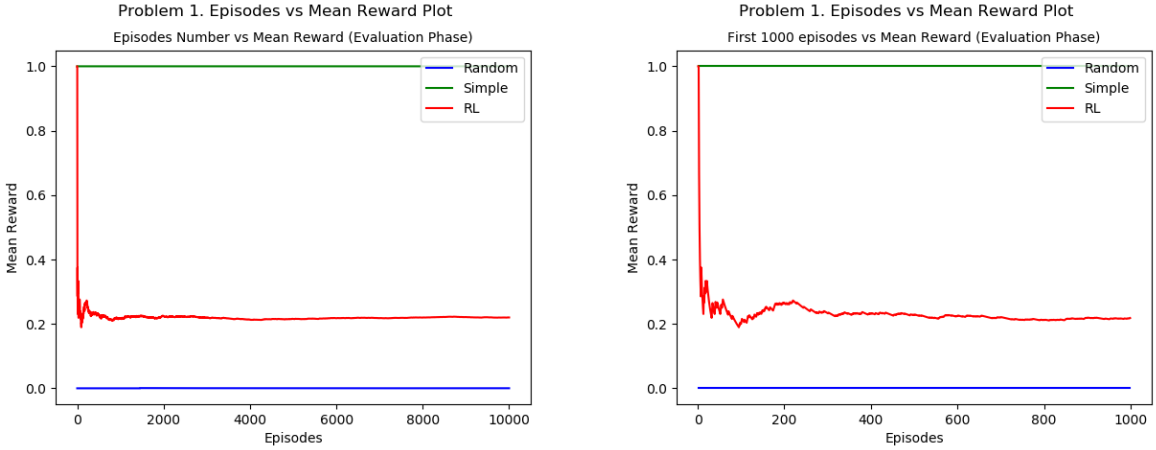


Figure 14: Evaluation Plots for Problem 1 (Evaluation Phase)

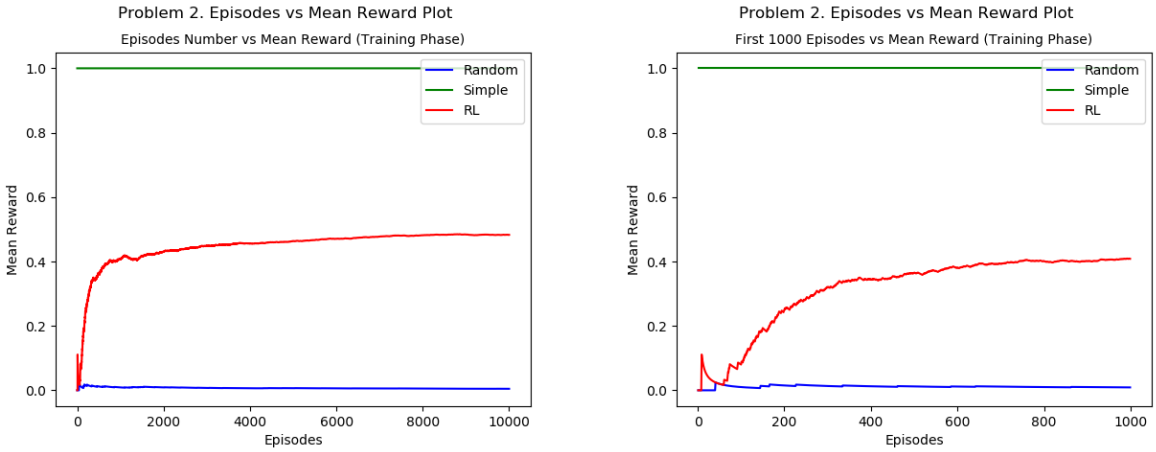


Figure 15: Evaluation Plots for Problem 2 (Training Phase)

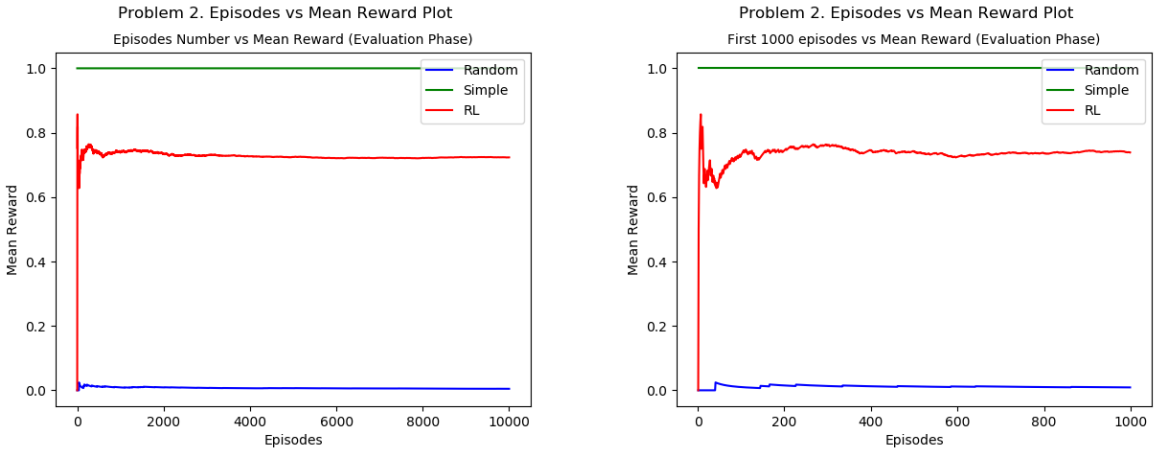


Figure 16: Evaluation Plots for Problem 2 (Evaluation Phase)

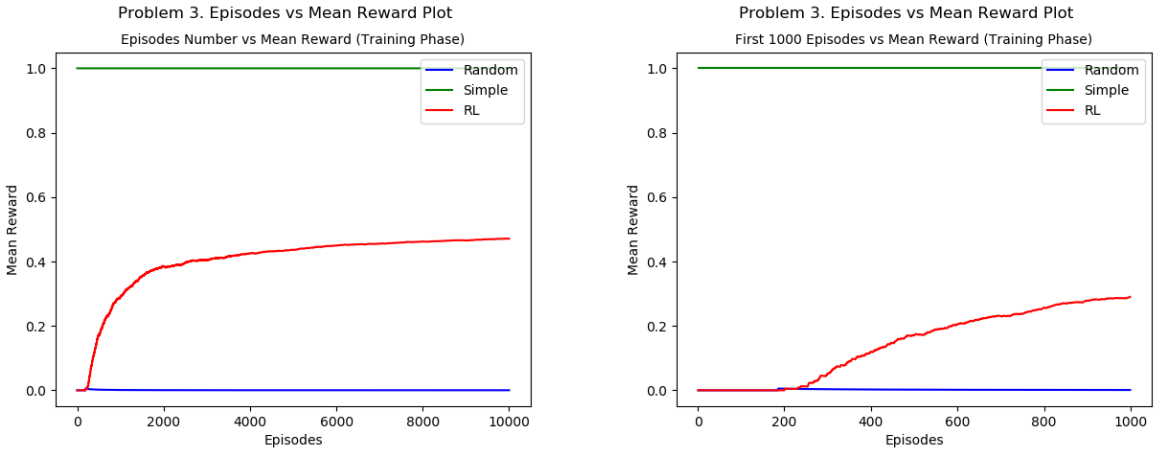


Figure 17: Evaluation Plots for Problem 3 (Training Phase)

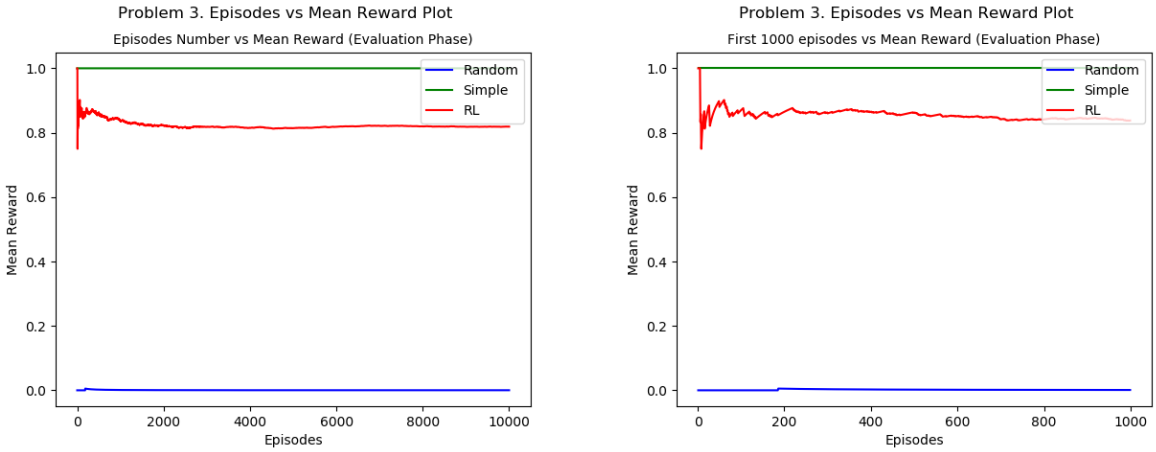


Figure 18: Evaluation Plots for Problem 3 (Evaluation Phase)

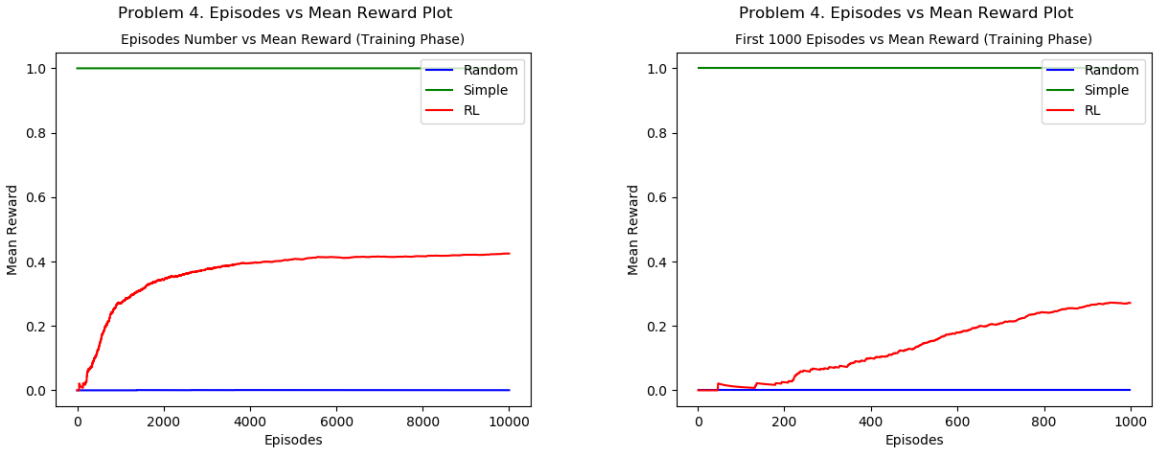


Figure 19: Evaluation Plots for Problem 4 (Training Phase)

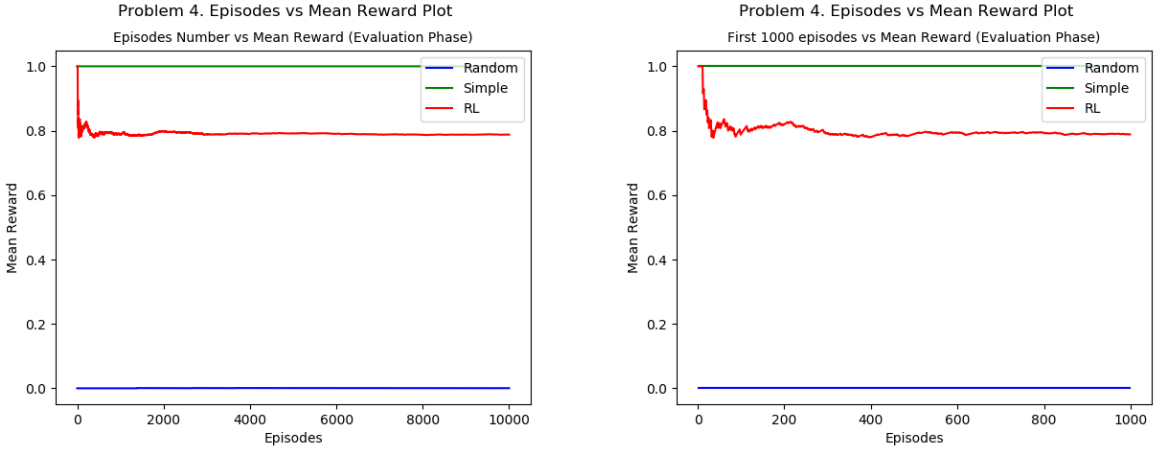


Figure 20: Evaluation Plots for Problem 4 (Evaluation Phase)

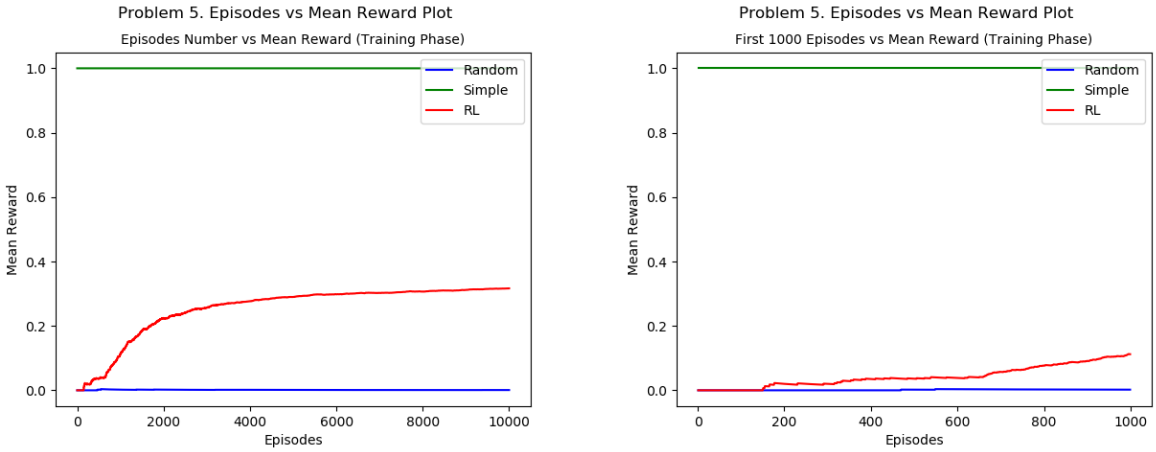


Figure 21: Evaluation Plots for Problem 5 (Training Phase)

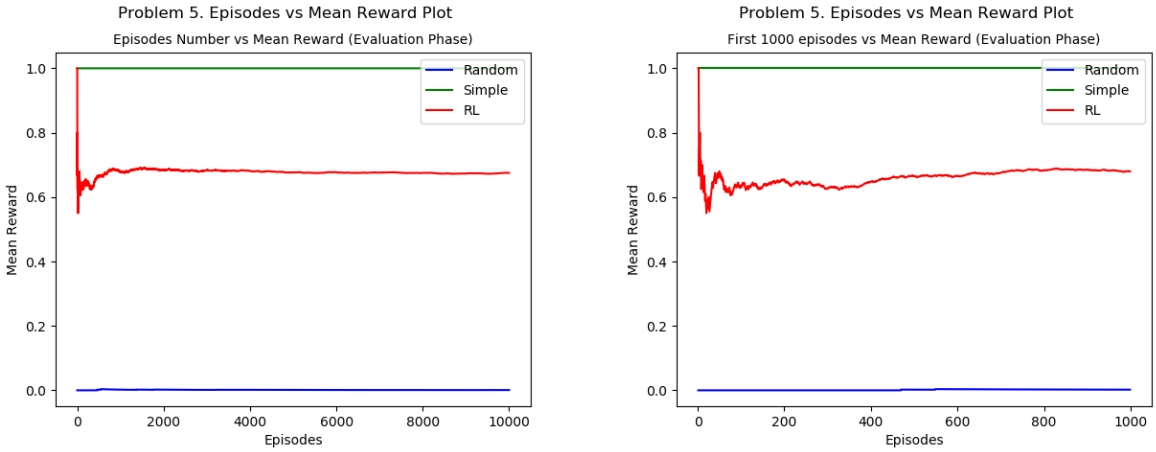


Figure 22: Evaluation Plots for Problem 5 (Evaluation Phase)

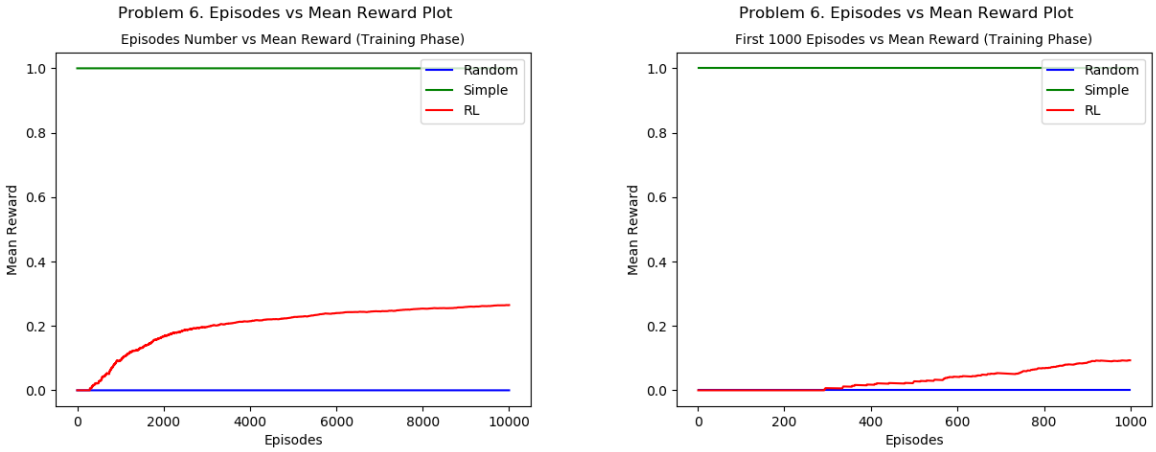


Figure 23: Evaluation Plots for Problem 6 (Training Phase)

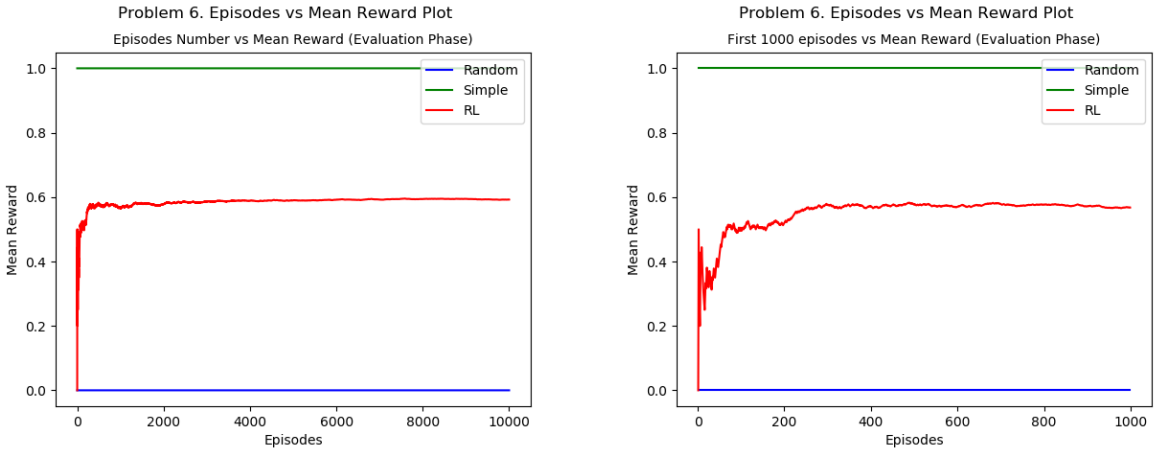


Figure 24: Evaluation Plots for Problem 6 (Evaluation Phase)

Figure 25: Evaluation Plots for Problem 7 (Training Phase)

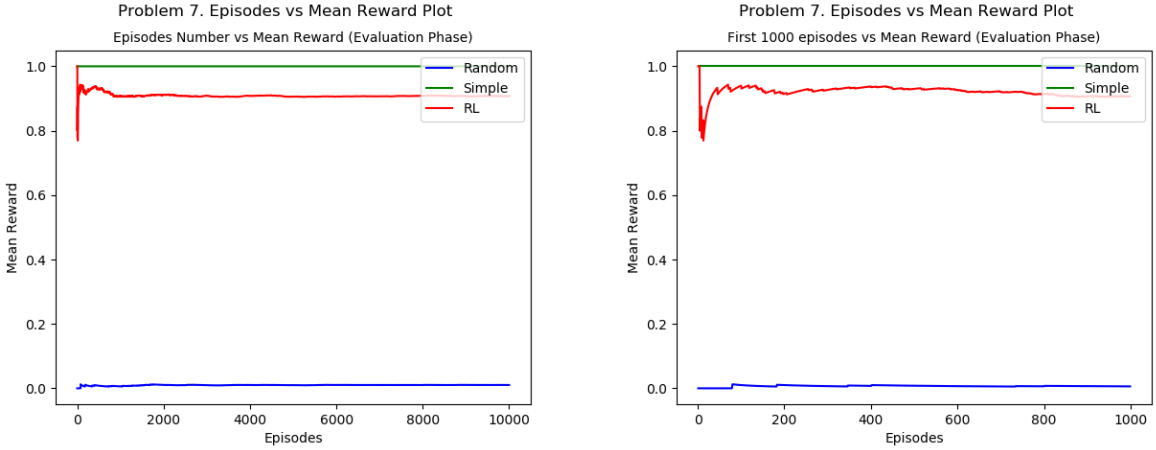


Figure 26: Evaluation Plots for Problem 7 (Evaluation Phase)

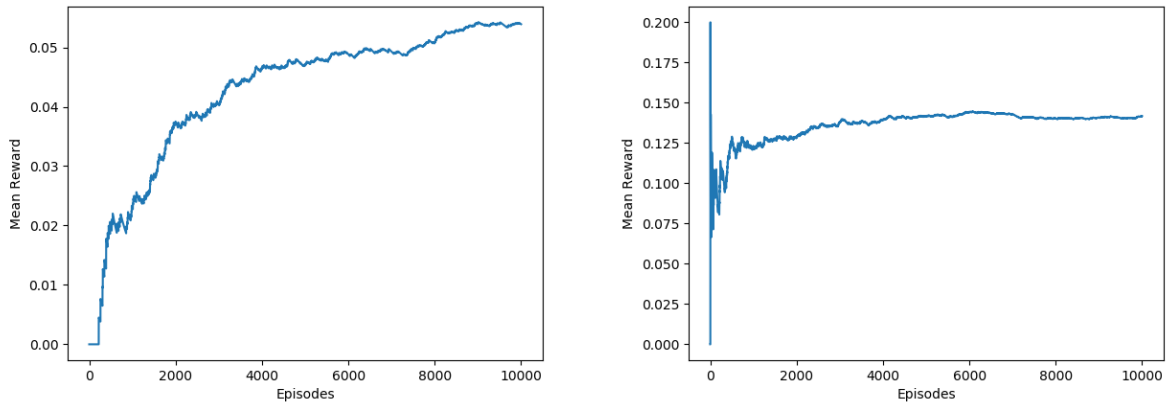


Figure 27: Evaluation Plots for Problem 0, Reinforcement Learning Agent (Evaluation Phase)

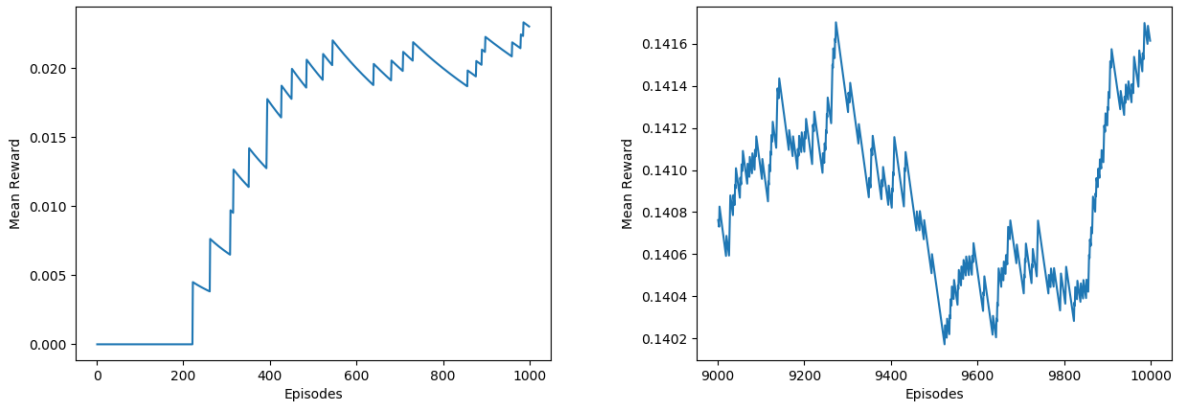


Figure 28: Evaluation Plots for Problem 0, Reinforcement Learning Agent (Evaluation Phase)