Introduction

In this assignment, I will provide a custom map to the 3 agents: a) Reinforcement Learning agent(RL agent), Simple agent(A Star agent), and a random senseless agent for finding the goal. Additionally, the map has 8 different versions for 8X8 map and 4 different versions for a 4X4 map. And at the end I will compare and contrast all of them and provide a conclusion.

PEAS Analysis

- **Reinforcement Learning agent:**
  1. **Performance Measure:** Average reward gains per 1000 or 100 episodes, accuracy i.e goals reached/(goals reached+holes reached).
  2. **Environment**:

     Partially observable environment: Only the current state is given and the agent has to explore first (and then exploit) to know the q values of each state. Holes and the goal are not known.
     Single agent: Our RL agent is a single agent deployed on the problem.
     Stochastic Environment: The next state cannot be determined from the current state**.** Lochlomond environment is slippery, so the next state of the agent may not be intended.
     Discrete Environment: Each state is discrete and clearly defined by the environment.
     Episodic Environment: The environment is rendered in episodes. Each passing of episode is determined by either agent's success or a failure or the maximum steps allowed in that episode (whichever is quicker).
     Semi-dynamic Environment: The environment is not changing but agent's performance is, over time; however, agent's performance converges after reaching the stable state(convergence graphs in the result section).
     Known world: The agent knows there are four directions for movement.

  3. **Actuators**:

     Discrete: Four actions are given which are well defined, and there is no intermediate action between them.
     Noisy: Randomness has been added to the action, so the agent may not end up in the desired state.

4. **Sensors**: Agent only has knowledge about the current state and also the directions that are available in that state.

- **Simple Agent**:
  1. **Performance Measure:** Average reward gains per 1000 or 100 episodes, accuracy i.e goals reached/(goals reached+holes reached).

  2. **Environment**:

     Fully observable environment: All the states are known to the agent, and so the agent is fully aware of the environment.
     Single agent: Our RL agent is a single agent deployed on the problem.
     Deterministic Environment: The next state can be fully determined from the current state**.**
     Discrete Environment: Each state is discrete and clearly defined within the environment.
     Sequential Environment: The future results are dependent on the current action. The agent can take a route and hit a road-block, then it again has to trace back to a favourable state and start from there.
     Static Environment: The environment is not changing and neither is agent's performance in time.
     Known world: The agent knows there are four directions for movement.
  3. **Actuators**:

     Discrete: Four actions are given which are well defined, and there is no intermediate action between them.
     Noise-free: No randomness is added, so an action intended for a state, will take the agent to that state only.
  4. **Sensors:**

     Agent knows the complete environment using env.desc and we can make program the agent to avoid certain blocks before starting even.

- **Random Agent**:
  1. **Performance Measure:** Average reward gains per 1000 or 100 episodes, accuracy i.e goals reached/(goals reached+holes reached).

  2. **Environment**:

     Partially observable, Single agent , Non deterministic environment , Discrete Environment, Episodic Environment, Semi-dynamic Environment, Known world
  4. **Actuators**: Random actions.

5. **Sensors**: Sensors are immaterial for this agent.

Method Design

Reinforcement Learning:

I have used Q-table for my agent which learns the quality of the state over many steps taken in an episode and then followed by many episodes. The steps are taken on the utility i.e the maximum of the possible Q-values of the state. However, how quickly the agent should adopt that q-value is determined by fine-tuning the learning parameter.

From the Bellman's equation, we have :

$q*(s,a)=E[Rt+1+\gamma maxa'q*(s',a')]$ [1]

It states that the optimal Q value of a state is the expected value of the reward it gets from taking the action 'a' to reach next state summed with the next state (multiplied by a discount factor). This state s-prime has to be the most optimum one, the agent could get into by taking a certain action. This whole process is thus governed by optimum policy. The agent could always immediately adopt the new q-value since it is under the best policy; however, with stochastic environment, the agent may not end up in the intended optimum choice. Rather, the agent may take less optimum state just because of randomness. Therefore, agent has a learning parameter (alpha) to consider. If the agent doesn't have this alpha, it can update the new q-value of the state with a less optimum value which is not desirable as it will be no better than a random agent.

In each iteration, when the agents visits the state again, it attempts to minimize the difference between the old q-value of the state and the newly learnt q-value. This is the loss and also the objective of the agent which will lead to maximum rewards.

$q_*(s,a)-q(s,a)=loss$ [1]

And here comes the learning rate alpha**:**

q-new (s,a)= (1-alpha) *old_q +alpha*(newly learned q)

Alpha determines how quickly the agent forgets the old q-value and adopts the new q-value. The newly learned q-value is being governed by Bellman's equation as stated above i.e the maximum of the possible choices.

Also, a crucial piece in the Reinforcement learning is the dilemma between exploration and exploitation i.e. should the agent chase after just current reward count or freely explore the environment to know other possible and perhaps better states? One of the solutions: Greedy-Epsilon method which helps to solve this puzzle.

I have used exponential exploration decay which decays over episodes. [1] Lesser the exploration, more the exploitation with the help of Q-table.

Simple Agent: Simple agent follows a predetermined map with the location of the goal and the avoidable blocks in the path. At any location on the map, the agent is aware of its distance from the goal and its parent node i.e. from where it came. Initially, all the nodes f values are calculated by this formula:

f=g+h where f is the total cost of the node, g is the distance from the start node to the current node and h is the estimated distance from the current node to the goal node. I have used AIMA toolkit provided in the course to complete this assignment [2].

After calculating all the f values of all the nodes, and ignoring the road-blocks, the agent starts from the starting point. After that:

It examines which of the child nodes have the lowest f value and that child node becomes the next node. Whenever the agent steps on a node, it adds that node to the explored list. This list will help to trace the path back from goal to start. Also, all the expanded/children nodes are added to a frontier list and if the child node is already there, its f value and parent are re-evaluated if found lesser than the previous path. Following this procedure, the agent reaches the goal.

Once, the agent finds the goal, it will always reach the goal provided that the environment is static otherwise we will need intelligent agents.

Random Agent: Our random agent is senseless i.e. it doesn't take any input through its sensors and randomly acts on the environment.

Implementation of Agents

Reinforcement Learning agent:

qLearningAgent(): It takes all the required parameters.

Epsilon-greedy- Initial exploration rate is 1 and decays in every episode according to :

np.exp(-1*decay_rate*episodes) [2] . I am generating a random number from a uniform distribution between 0,1 and if this number is less than exploration rate, agent will explore or else it will exploit.

Testing phase: When the episode reaches 80% of the total episodes, learning rate becomes 0 and testing phase starts. Agent moves on the learnt Q-table.

Goal count and hole count under testing phase give the accuracy which is goal_count/(goal_count+hole_count).

Number of rewards is then summed up and every 1000(for 8X8 and 500 for 4X4) episode, it is averaged. The function returns q_table (used for printing policy) , rewards_per_episode(used for reward vs episode plot), accuracy of the testing phase and state_df as the utility of each state per episode.

policy(): This function takes a q table and its respective environment and prints the policy accordingly.

Simple Agent: [3]

env2statespace() is creating the maze for us by taking the Lochlomond environment. It is returning the locations of every state, the possible directions an agent can move from any state, initial state and final state. also, the holes are directly ignored i.e. the agent doesn't take holes into consideration.

my_astar_search_graph(): It calls the my_best_first_graph_search_for_vis() with the f cost and the maze problem parameters.

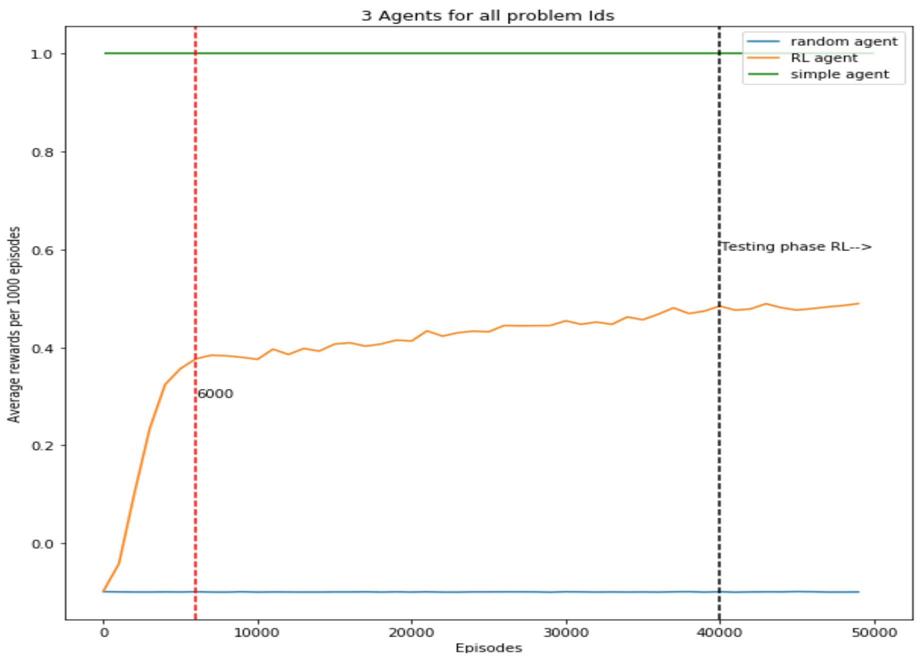
my_best_first_graph_search_for_vis(): This defines the standard algorithm of A star where how a node is checked, it child nodes are checked, etc. have been stated in the Method Design section.

simpleAgent() This method is driving the agent into solving the A star problem. It takes node locations, possible directions and call the my_astar_search_graph() which returns the number of iterations it took to reach the goal and the solution path.

Random Agent:

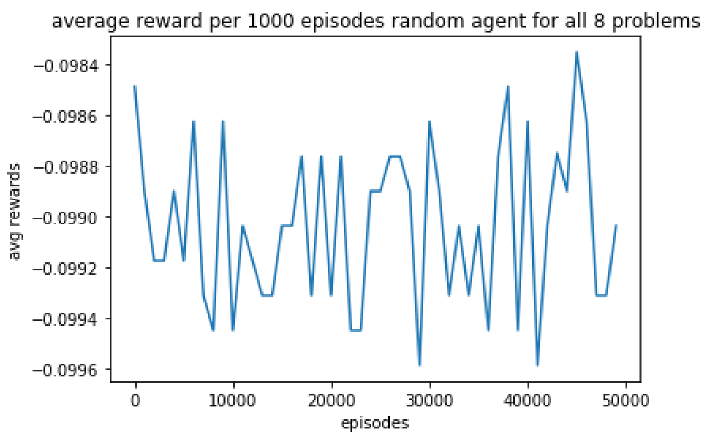
This agent doesn't take any input and randomly samples actions. And returns q_table_3, accuracy_3, reward_per_episode_3, state_df_3,env variables just like the RL agent. Another difference is the accuracy is measured throughout the max episodes unlike the RL agent where the accuracy is measured for the testing phase.

Experiments and Evaluation



8X8-base:

- The plot shows the average number of rewards gained per 1000 episodes by all the 3 agents, averaged over all the 8 problems.
- The Convergence point achieved at episode 6000 approximately for RL agent (knee-point can be found out using KneeLocator class from kneed package).
- The testing phase is after 40,000 episodes(80% of the max_episodes).
- The RL agent curve continues to grow after the convergence point but slowly. This states that the overall trend is increasing for RL agent. We can also say that the agent continues to gain rewards if we look at all 8 problem IDs simultaneously.
- We can see that the simple agent, when reaches the goal, it always yields the maximum reward which is 1 in this case.
- Random agent also seems constant, but below is the detailed version of it:



- As expected, it moves randomly.

- Comparison of the accuracy in the **testing phase** of RL agent:

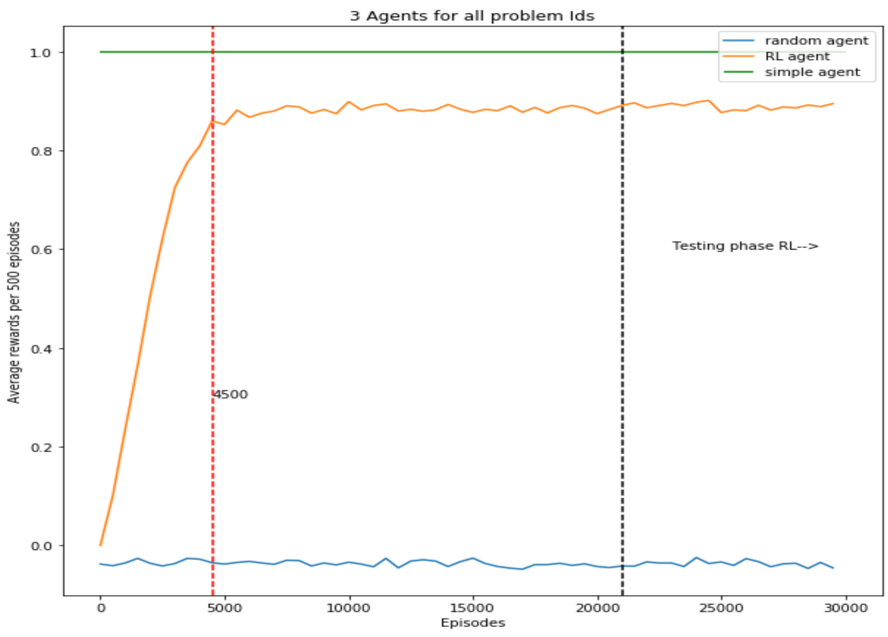
	random agent	RL agent	simple agent
0	0.012001	0.227101	100
1	0.008000	17.468531	100
2	0.428000	83.355858	100
3	0.062000	63.128717	100
4	0.038000	61.734529	100
5	0.064000	78.180655	100
6	0.002000	58.822286	100
7	0.054000	60.327903	100

- This shows the accuracy i.e goal attained /(goal attained+hole attained) (where goal attained is +1.0 reward while hole attained is -0.1 reward) **in the testing phase of RL agent.**
- The RL agent on problem ID 0 and 1, does poorly. If we check the policy of id 1 with hole reward= -0.1:

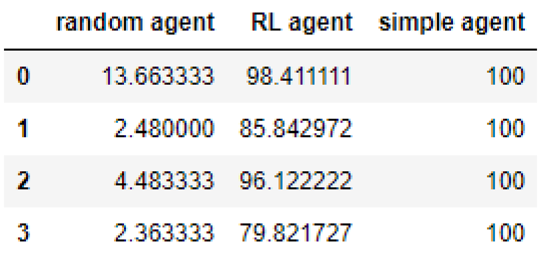
```
[['Hole' '>' '>' '>' '<' 'Hole' '>' '^']
 ['down' '^' '^' '^' '^' 'down' '<' 'Hole']
 ['^' '^' '<' 'Hole' '>' '^' '>' 'down']
 ['<' '^' '<' 'down' '<' 'Hole' '>' 'down']
 ['<' '^' '^' 'Hole' '>' 'down' '^' '^']
 ['<' 'Hole' 'Hole' '>' '^' '<' 'Hole' '>']
 ['<' 'Hole' 'down' '^' 'Hole' '>' 'Hole' '>']
 ['<' 'down' '<' 'Hole' 'null' 'Goal' 'null' 'Hole']]
```

- The S is at (0,1), there is a hole just next to. Agent keeps falling into that hole if the hole reward is 0.
- We can see that the RL agent performs best on problem ID 2 with 83.355 accuracy**.** And the Simple agent is always 100% accurate, since it will reach the goal always**.**

4X4:



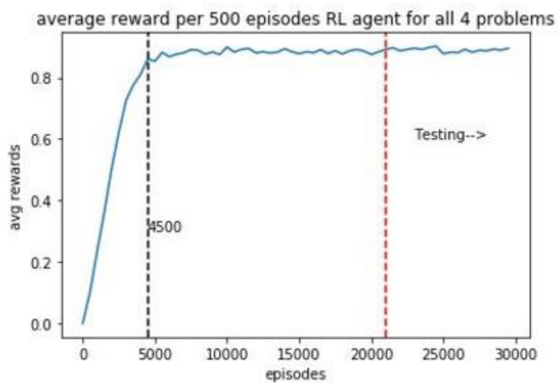
- The plot shows the average number of rewards gained per 500 episodes by all the 3 agents, averaged over all the 4 problems.
- We can see that the RL agent is converged denoting the max reward averaged across 4 problem IDs have been achieved.
- The convergence point for RL agent is 4500 approximately.
- Testing phase for RL agent begins after 70% of max_episodes which is 21,000 in this case.
- A few fluctuations observed for random agent which can be explained by the fact that it takes random actions.
- Comparison of the accuracy in the **testing phase** of RL agent:

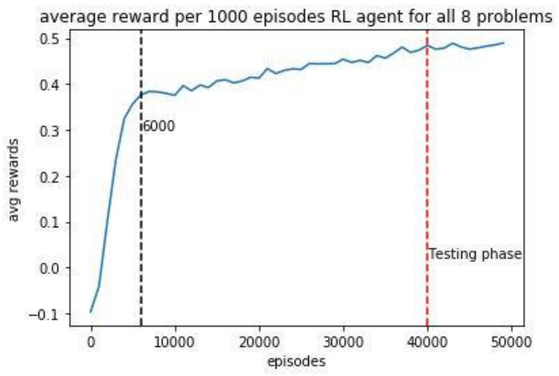


	random agent	RL agent	simple agent
0	13.663333	98.411111	100
1	2.480000	85.842972	100
2	4.483333	96.122222	100
3	2.363333	79.821727	100

-
- We can see that in case of 4X4 where we see that the RL agent does best on problem ID. 0 and coincidently random agent does the best on this one too.

Comparison between 4X4 and 8X8:





- Overall, although slowly, 8X8 RL agent is continuing to achieve rewards. While 4X4 RL agent has converged.
- 8X8 RL agent does best on problem ID 2 while 4X4 RL agent does best on problem ID 0 (if we check the above tables again).
- The convergence of 8X8 RL agent is achieved at episode 6000 approximately while for 4X4 RL agent, it is 4500 episodes.
- 4X4 RL agent does better than the 8X8 RL agent which perhaps needs more episodes to reach a more stable state.
- The average number of iterations required by 4X4 Simple agent to attain goal node is 24 while 8X8 takes 89 iterations.

Conclusion

Reinforcement learning agent learns about the environment and finds out the optimum policy which ensures max reward is achieved. The world can be random, but the RL agent optimizes itself against all odds. We have seen that our 8X8 RL agent reaches a knee-point ;however, continues to earn the average rewards. While the 4X4 RL agent with lesser episodes, converges. This proves that the size of the environment adds to the complexity of the problem and thus more time is needed.

Also, problem Id 0 and 1 for 8X8 map, has holes either too close to the start or goal. If the reward for falling into the hole is just 0, the agent is not going to learn the policy at all. It will continue to fall. In order to prevent that, we need to give negative rewards when it falls through a hole. However, if the hole reward is too big, agent will take more iteration to find its way to the goal because then its objective will be more to prevent from falling into a hole.

References:

1) ***What do Reinforcement Learning Algorithms Learn - Optimal Policies*** [online] Available at < https://deeplizard.com/learn/video/rP4oEpQbDm4 >

2) ***Exploration rate decay*** [online] Avalable at < https://deeplizard.com/learn/video/HGeI30uATws>

2) Dr. Byorn Sand Jensen, University of Glasgow.

Appendix:

8X8 problem Id: 0

```
Accuracy is overall goals achieved wrt total loss at holes AT TESTING PHASE: 0.22710068130204392

The time it took to train the agent in a 8X8 maze:

00:05:21
The accuracy at testing phase is : 0.22710068130204392
problem_id: 0, Agent Trained. Below is the policy:

[['<' '<' '<' '<' '<' 'Hole' '>' '^']
 ['<' '^' '^' '^' '^' 'down' '<' 'Hole']
 ['<' '<' '<' 'Hole' '>' '^' '>' 'down']
 ['<' '<' '^' 'down' '<' 'Hole' '>' 'down']
 ['<' '^' '^' 'Hole' '>' 'down' '^' '^']
 ['<' 'Hole' 'Hole' '>' '^' '<' 'Hole' '>']
 ['<' 'Hole' 'down' '<' 'Hole' '>' 'Hole' 'down']
 ['<' 'down' '<' 'Hole' 'Goal' 'down' 'down' 'Hole']]
```

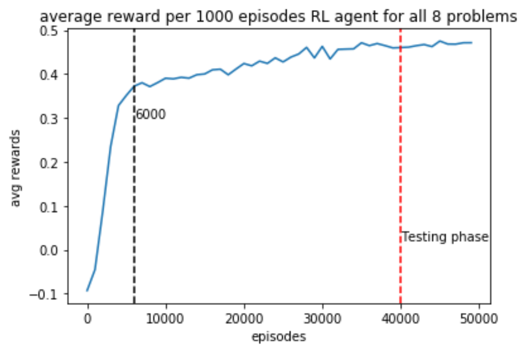
Overall performance of the RL agent on 8X8 map:

Visualizing the average of rewards per 1000 episodes for RL agent accross 8 problems:

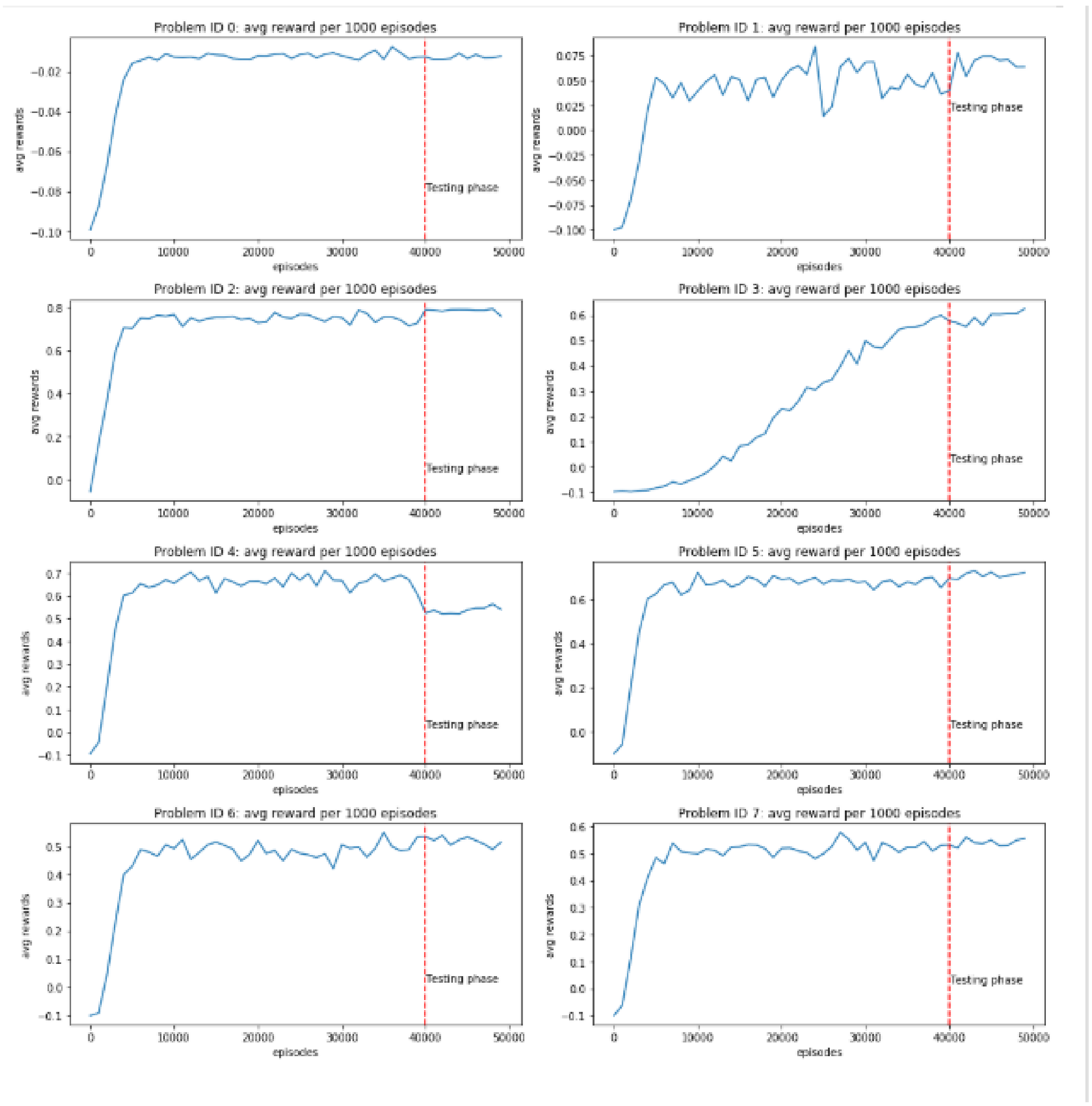
```
avg_reward_rl=np.mean(reward_df.values,1) #this will be used to produce the final comparison graph



plt.plot(list(range(0,max_episodes,1000)),avg_reward_rl)
plt.title("average reward per 1000 episodes RL agent for all 8 problems")
plt.xlabel("episodes")
plt.ylabel("avg rewards")
plt.text(40000,0.02,"Testing phase")
plt.axvline(testing_phase,color='r', linestyle='--')

converge_point = KneeLocator(range(0,max_episodes,1000), avg_reward_rl, direction='increasing') #knee locator
print("The point of convergence for the RL agent is ",converge_point.knee)
plt.axvline(converge_point.knee,color='k',linestyle='--')
plt.annotate('6000',xy=(converge_point.knee,0.3),xytext=(converge_point.knee,0.3))
```

```
The point of convergence for the RL agent is  6000
```

```
Text(6000,0.3,'6000')
```



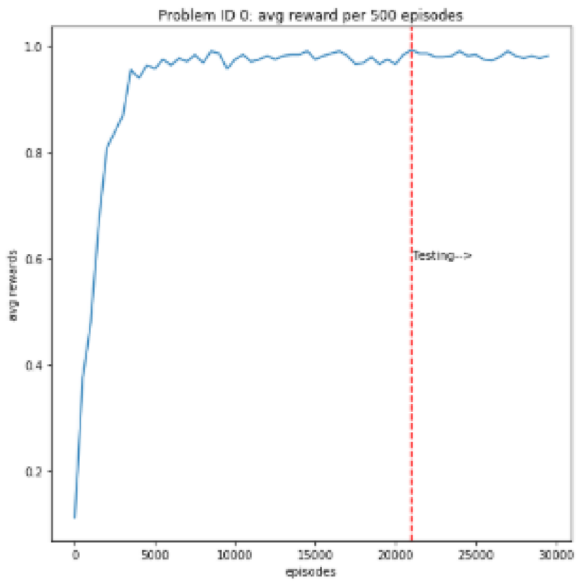
All problem IDs for 8X8 RL agent:

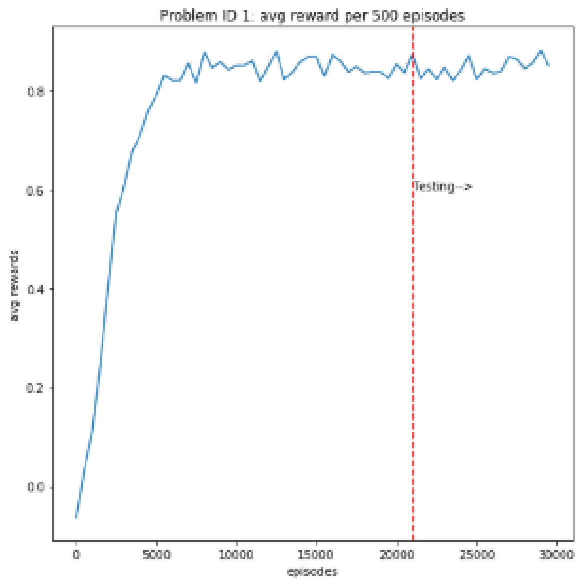


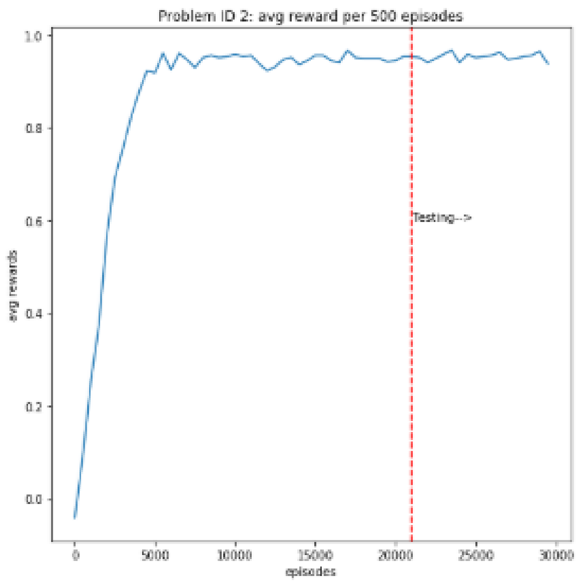
4X4 RL agent on problem ID 0:

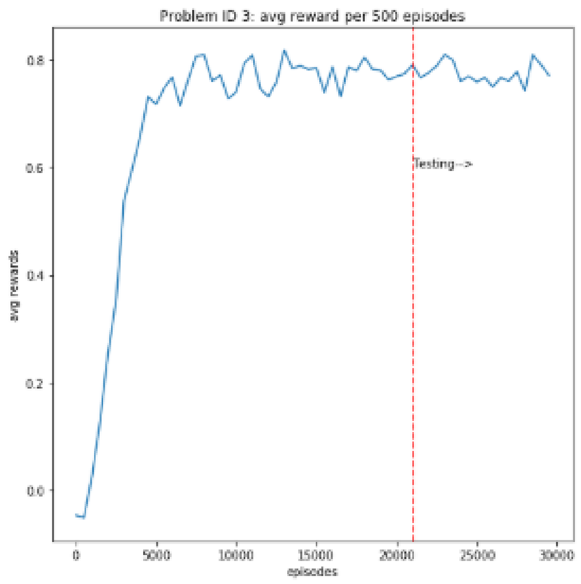
```
Accuracy is overall goals achieved wrt total loss at holes AT TESTING PHASE: 86.10184567489438

The time it took to train the agent in a 4X4 maze:

00:00:57

 The accuracy at testing phase is: 86.10184567489438
problem_id: 1, Agent Trained. Below is the policy:

[['<' '^' '^' '^']
 ['<' 'Hole' '<' 'Hole']
 ['^' 'down' '<' 'Hole']
 ['Hole' 'Goal' 'down' 'down']]
```

All problem IDs for 4X4 agent:


Problem ID 0: avg reward per 500 episodes
avg rewards
episodes
Testing-->
0.2
0.4
0.6
0.8
1.0
0
5000
10000
15000
20000
25000
30000


Problem ID 1: avg reward per 500 episodes
avg rewards
episodes
Testing-->
0.0
0.2
0.4
0.6
0.8
0
5000
10000
15000
20000
25000
30000


Problem ID 2: avg reward per 500 episodes
avg rewards
episodes
Testing-->
0.0
0.2
0.4
0.6
0.8
1.0
0
5000
10000
15000
20000
25000
30000


Problem ID 3: avg reward per 500 episodes
avg rewards
episodes
Testing-->
0.0
0.2
0.4
0.6
0.8
0
5000
10000
15000
20000
25000
30000