PEAS Analysis

The task was to design, implement, evaluate and document three virtual agents that will navigate across the FrozenLake environment. The agents controls the movement of a character in the frozen lake depicted by a grid. Some tiles of the grid are walkable, and others lead to the agent falling into a hole. The agent must navigate its way through the grid to reach the goal. In the case of this agent, the PEAS analysis is as follows:

- Performance Measure: The number of episodes and iterations in that episode that the agent uses to get to the goal.
- Environment: The environment is the frozen lake which is grid-world with a starting position (S), obstacles (H) and a final goal (G).
- Actuators: The agent can move up, down, left or right to interact and move through the environment.
- Sensors: The agent will obtain a reward of 1.0 if it reaches the goal state, -0.01 if it falls down a hole and 0.0 for any other state.

Design

Senseless/Random agent
This agent randomly navigates itself through the environment. This agent does not have any sensors and it can take any action in any state until it falls down the hole or reaches the goal.

Simple Agent
This agent uses the A* star search algorithm to navigate through the environment. A* algorithm is one of the best and popular technique used in path-finding and graph traversal. It does this by creating an undirected graph representation of the state space and from the starting node, it tries to find the shortest path to the goal node having the smallest cost. What the algorithm does is, at each step, it picks the node according to a value 'f' which is equal to sum of 'g' and 'h'. At each step, it picks the node having the lowest 'f' value and processes it. This is given by the equation:
$f(n) = g(n) + h(n)$

Here, n is the previous node on the path. g(n) is the cost of the path from the start node to n and h(n) is a heuristic that estimates the cost of the cheapest path from n to the goal. The agent terminates when the path it chooses to extend is a path from start to goal or if there are no more paths to be extended.

Reinforcement Learning Agent
This agent uses tabular Q-learning to navigate through the environment. There is no prior knowledge about the state-space in general and for this reason, Q-learning is called a model-free method. In Q-Learning, the agent learns an action-utility function, or Q-function, giving the expected utility of taking an action in a given state. It then trains the Q-table to help guide the agent as to what to do in each state.

Implementation

Senseless/Random
The implementation is almost identical to the demo provided in the starter code with just additional lines to print relevant output to textfiles.

Simple Agent
The A* search algorithm used by the simple agent is implemented in the following methods obtained from Lab 3:
def my_best_first_graph_search(problem, f, initial_node_colors):
def my_astar_search(initial_node_colors , problem, h=None):

Reinforcement Learning Agent
Every time (t) the agent select an action (a) and observe the reward (r), the Q table is updated using the learning policy. The agent explores the states using the following function:
def action_utility(state , epsilon , env, Q):

The Q-table is updated using the learn function given below:
def learn(state , new_state , reward , action , Q, gamma, lr rate):
which implements the following equation:
$Q^{man}(s_t , a_t) = Q(s_t , a_t) + \alpha * (r_t + \gamma * max(Q(s_{t+1}, at)) - Q(s_t , a_t))$ to update the policy table.
Here, r is the reward received when moving from the state st to the state s_{t+1} , and α is the learning rate($0 < \alpha < 1$).

Evaluation

Random Agent

The maximum number of episodes was set to 5000 with a maximum of 500 iterations per episode. The following table shows the number of time the random agent reached the goal for each problem id:

Problem ID	0	1	2	3	4	5	6	7
Success Count	0	0	24	3	2	5	0	49

As we can see, there is no consistency in the success counts and the random agent even fails to reach the goal on some problem id's.

Simple Agent

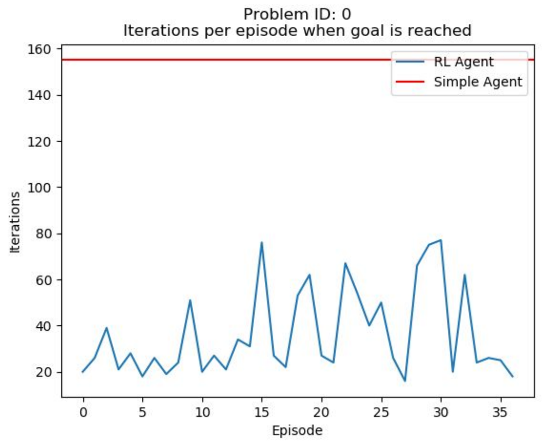
The simple agent knows the environment beforehand. So it will always reach the goal. The following table gives the number of iterations it takes for the agent to reach the goal on each problem id:

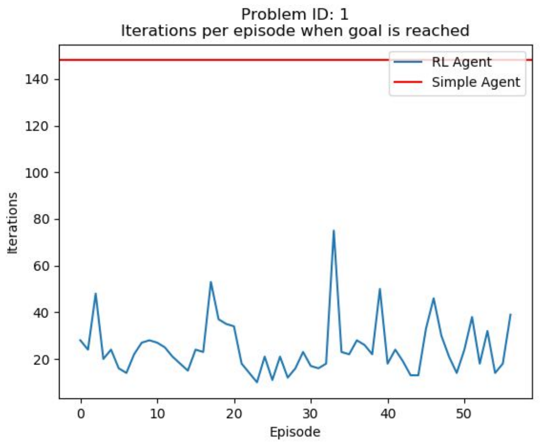
Problem ID	0	1	2	3	4	5	6	7
Iterations	155	148	141	146	146	152	146	157

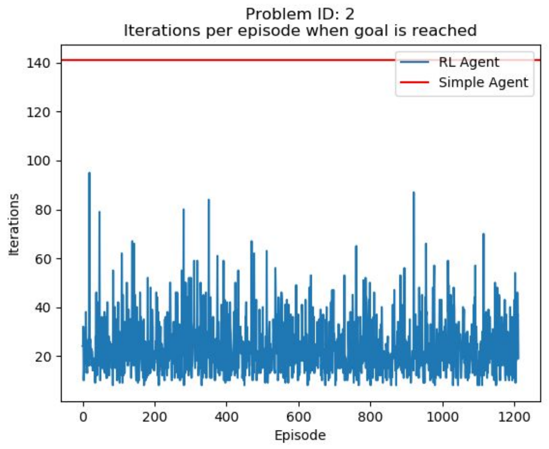
We see that the number of iterations is almost consistent and the agent takes almost the same amount of steps to reach the goal on every problem id.

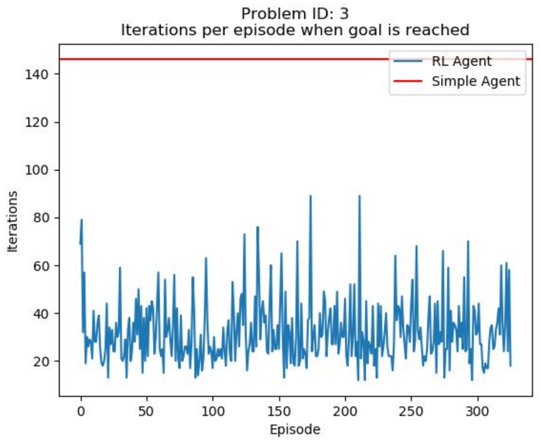
Reinforcement Learning Agent

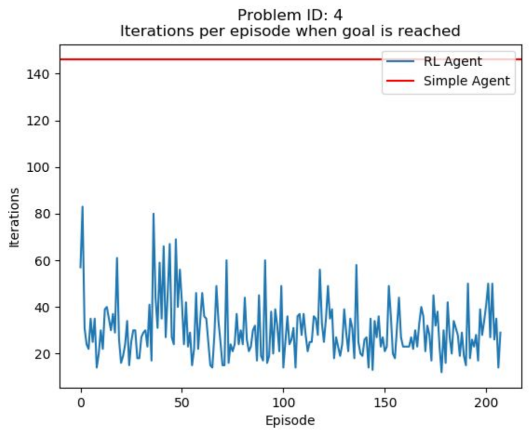
The maximum number of episodes was set to 50000 with a maximum of 500 iterations per episode. The following graphs shows comparison in number of iterations to reach the goal between the Simple and RL agent for all iterations:

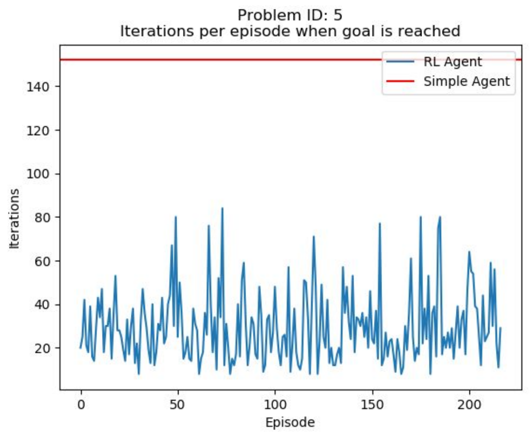


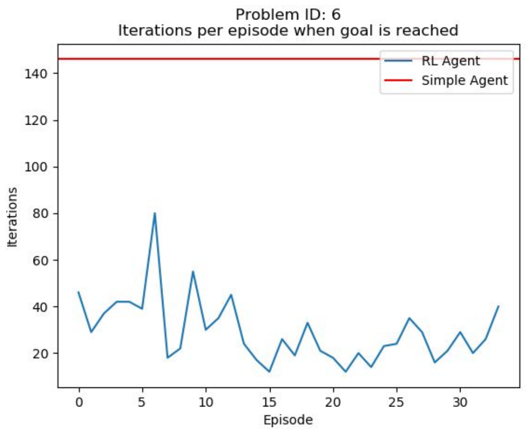


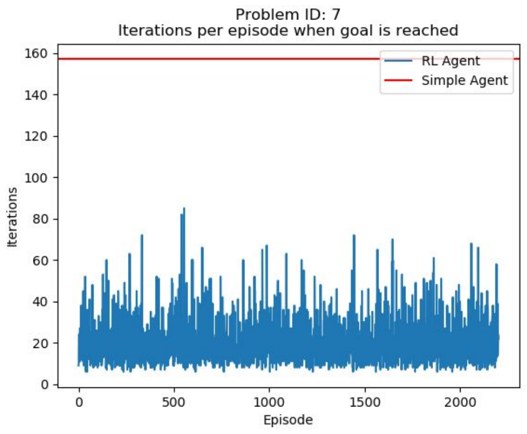












We can see from the graphs that the plot for the simple agent is almost a straight line. This is as we observed from the previous table where the simple agent takes almost the same amount of steps to reach the goal on every problem id. However, we see that the RL agent takes much less iterations to reach the goal on all problem id's

Conclusion

The simple agent takes almost the same number of iterations to reach the goal every time. This is because the agent knows its environment beforehand. Even though it reached the goal, the use of such an agent in the real world is not that useful. For example, a self driving car does not know what obstacles lie ahead of it before it reaches the destination. Hence, we use an RL agent like the one we implemented using Q-learning that learns, trains and guides the agent. With a better learning policy, the agent can improve to reach the goal in even less iterations.