Aatmay S. Talati

Machine Learning (CS 4641)

Project 4: Markov Decision Process (MDP) and Reinforcement Learning

Date: April 22, 2018 (Spring 2018)

Analysis of MDPs and Reinforcement Learning

Markov Decision Process (MDP): A Gentle Introduction

MDPs give a scientific structure to modeling decision making in circumstances where results are partly random and halfway under the control of a decision Maker.

A Markov Decision Process (MDP) model contains:

- A set of possible world states *S*
- A set of possible actions *A*
- A real valued reward function *R(s,a)*
- A description *T* of each action's effects in each state.

Reinforcement Learning: A Gentle Introduction

Reinforcement learning is an important type of Machine Learning where an agent learns how to behave in an environment by performing actions and seeing and analyzing the results [2]. In the absence of existing training data, the agent learns from experience. It collects the training examples

through trial-and-error as it attempts its task, with the goal of maximizing long-term reward [3].

Introduction:

In my analysis report, I'm planning on performing value iteration, policy iteration, and Q-Learning on small and large Markov Decision Processes. For the same, I've gotten familiar with the Reinforcement Learning Simulator [4]. I've used it to conduct this evaluation of Model-based and Model-Free reinforcement learning methods. For the robot to move autonomously, I've used these two provided:

2_medium_simple.maze

7_medium_complex_multigoal.maze

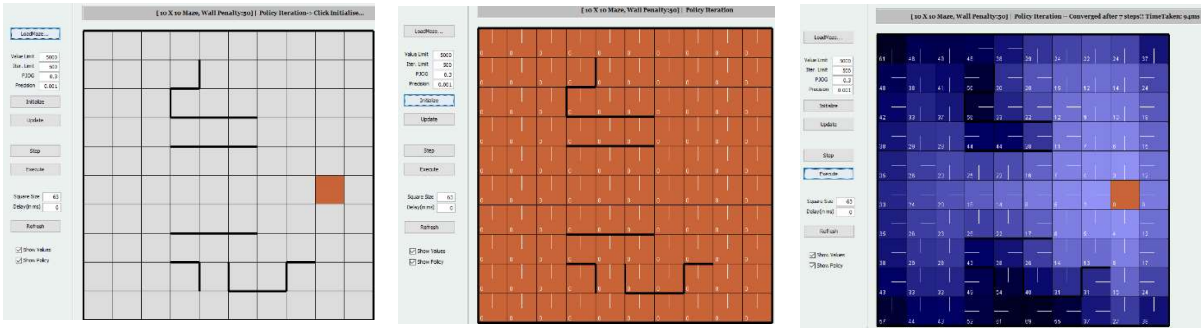
These will be used to predict the best and optimal path to the goal, given a fact of having an equal path cost without any rewards for reaching the state. In any case, if a robot hit the wall, we will have negative feedback/consequences to reach the given goal, thus with the help of the reinforcement learning the agent will learn about moving forward.

Getting Familiar with Simulator:

As a part of the using simulator, we must download and extract it, and run a command (given in README file) in Linux Terminal or in Windows PowerShell into current working directory (RL_sim).

- Rewards are displayed as orange cells
- State-action values displayed in relative directions.
- ‘Best’ policies are displayed as lines in relative directions of the actions in the Reinforcement Learning Simulator.
- PJOG is the percentage of the robot allowed to miss-step

As a first step, we just load the Maze by clicking on an appropriate method. Then we click on “Initialize” and “Execute” respectively.



Notice a general trend here,

$$((Color \uparrow \Rightarrow Solution\ gets \uparrow Optimal) \Rightarrow Time\ to\ get\ solution \uparrow)$$

Policy Iteration:

The essential idea behind policy iteration is this**:**

Start with a random policy:

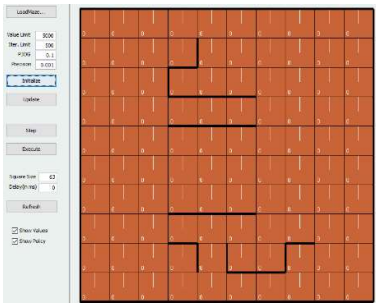
And continuously updates the policy by updating the utility of each state according to the current policy:

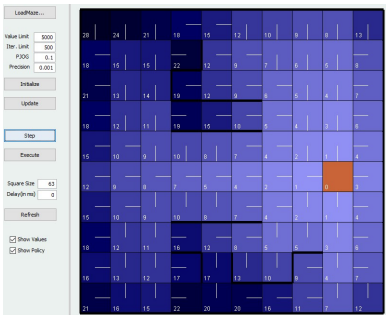
Until there is no change in the policy. [6]

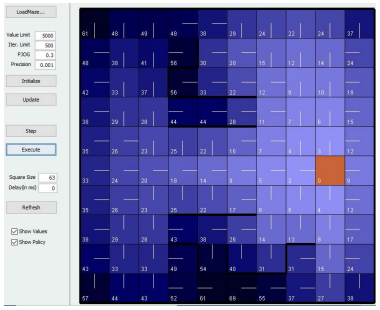
This algorithm works directly with the policy rather than getting an optimal value from it. At each step of the policy it’s pretty much guaranteed that it will update the policy with keeping in mind for the optimal solution.

Maze: 2_medium_simple

PJOG	Precision	Steps	Wall Penalty	Time Taken	Value Limit (Default)	Iteration Limit (Default)
0.1	0.001	6	50	82 ms	5000	500
0.3	0.001	7	50	76 ms	5000	500
0.5	0.001	5	50	54 ms	5000	500
0.7	0.001	4	50	165 ms	5000	500

1) 

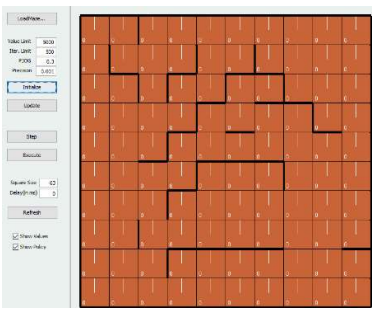
2) 

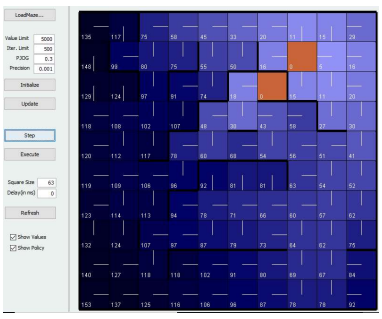
3) 

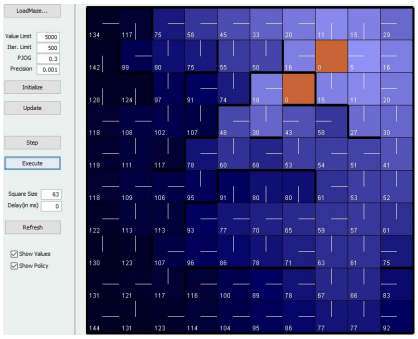
1) shows the initial step with 0.3 PJOG
2) shows at approx. 50% of the steps
3) shows at the 100% of the steps

Maze: 7_medium_complex_multigoal

PJOG	Precision	Steps	Wall Penalty	Time Taken	Value Limit (Default)	Iteration Limit (Default)
0.1	0.001	6	50	82 ms	5000	500
0.3	0.001	7	50	50 ms	5000	500
0.5	0.001	5	50	82 ms	5000	500
0.7	0.001	6	50	234 ms	5000	500

1) 

2) 

3. 

1) shows the initial step with 0.3 PJOG
2) shows at approx. 50% of the steps
3) shows at the 100% of the steps

Value Iterations:

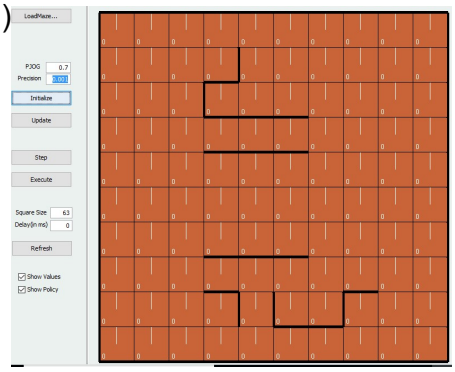
The essential idea behind value iteration is this:

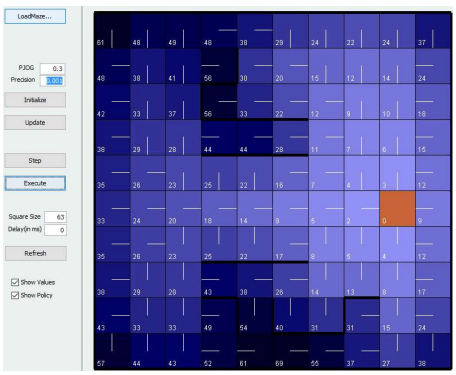
if we knew the true value of each state, our decision would be simple:

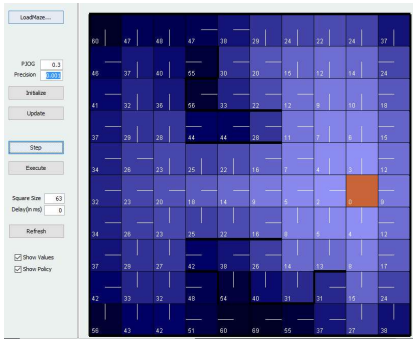
Of course, keeping in mind that it should always choose the action that maximizes expected utility. But we don't initially know a state's true value; we only know its immediate reward. Let's look at the simple pseudocode for better explanation/understanding [5].

Maze: 2_medium_simple

PJOG	Precision	Steps	Wall Penalty	Time Taken
0.1	0.001	28	50	8 ms
0.3	0.001	61	50	15 ms
0.5	0.001	164	50	55 ms
0.7	0.001	931	50	304 ms

1) 

2) 

3) 

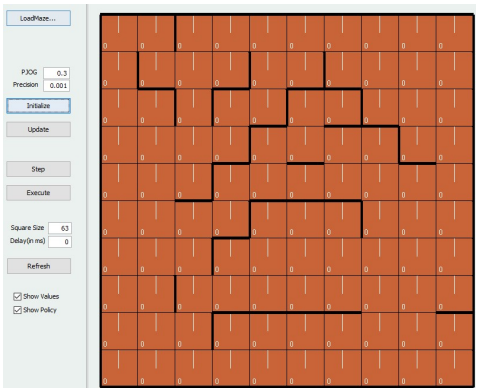
1) shows the initial step with 0.3 PJOG

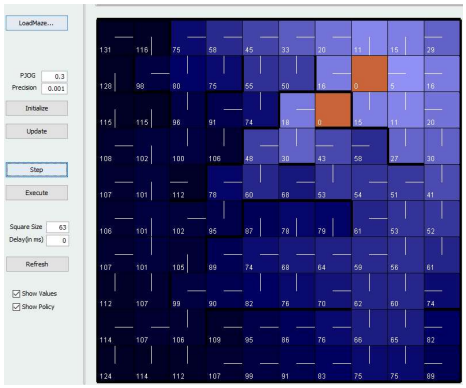
2) shows at approx. 50% of the steps

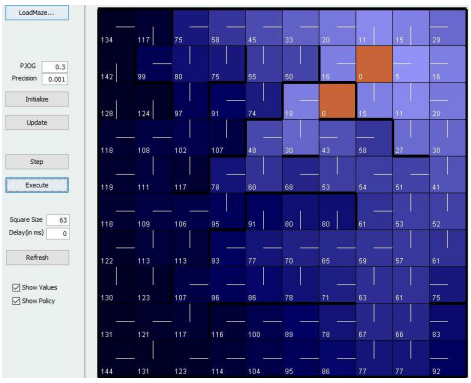
3)shows at finished number of the steps

Maze: 7_medium_complex_multigoal

PJOG	Precision	Steps	Wall Penalty	Time Taken
0.1	0.001	42	50	18 ms
0.3	0.001	84	50	38 ms
0.5	0.001	181	50	81 ms
0.7	0.001	1365	50	672 ms

1) 

2) 

3) 

1) shows the initial step with 0.3 PJOG

2) shows at approx. 50% number of steps

3) shows at 100% number of the steps

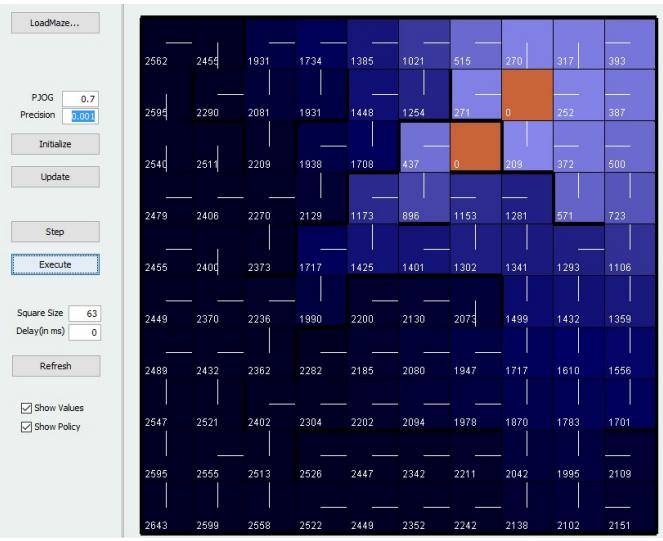
So as exemplified above, we ran the algorithm via using different values of PJOG varying from 0 to 1. We can see a clear trend that:

$$((PJOG \uparrow \Rightarrow Steps \uparrow) \Rightarrow Time\ Taken \uparrow)$$

Here,

PJOG of 0.1, 0.3, 0.5 and 0.7 $\Rightarrow$ there are 10%, 30%, 50% and 70% chances respectively for the agent makes move against its will to any of the adjacent states.

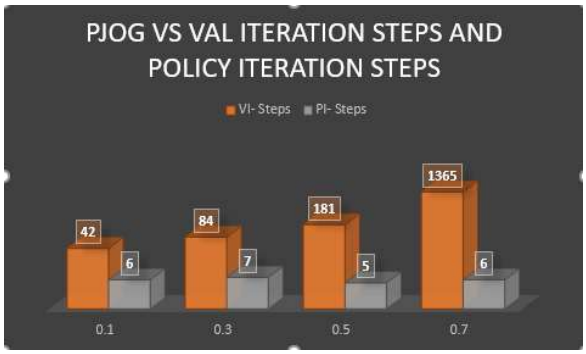
Precision (0.001) makes sure that algorithms converges.



During the execution of the algorithm, we could clearly see that it could choose to go via lengthier route, than being near the walls.

Policy Iterations vs Value Iterations:

By looking at the graphs from Policy Iterations and Value iterations at the PJOG of 0.3 and precision of 0.001, we can notice that we have the same output in the both cases. That is due to the fact that Policy iteration it keeps retraining the highest value policy, whereas in Value Iteration, it continuously changes till it converges to the highest value policy.



So, one thing we could notice that, in terms of the policy iteration, the number of steps gradually increases, whereas in the value iterations, it fluctuates.

Also, we could see similar pattern for the graph of the PJOG vs time in policy iterations and Value iterations.

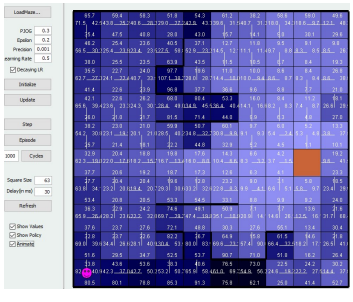
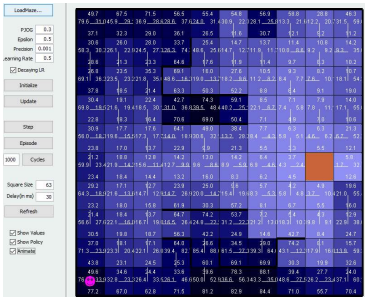
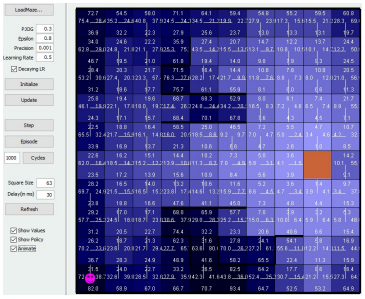
Q Learning:

This reinforcement learning technique does not require a model of the environment. It can handle problems with stochastic transitions and rewards, without requiring adaptations. For any finite MDPs, Q-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable. Q-learning can identify an optimal action-selection policy for any given finite MDP [7].

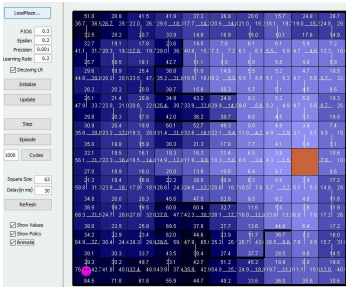
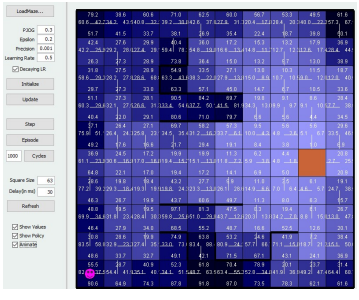
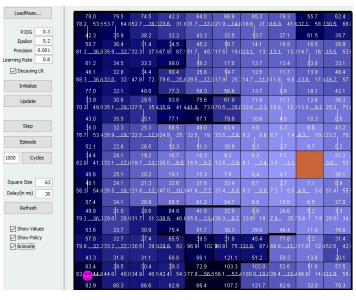
After conducting a little research on the web, I found out that Q-Learning is like Simulated Annealing in terms of knowingly making bad conclusions to not to get stuck into local maxima. CMU emulator was efficient enough for us to feed in that value. This value is known is 0 < “Epsilon” < 1 value in the simulator. Just like this, there’s 0 < “Learning Rate” < 1 value which shows the changes into the existing policy.

To demonstrate Q-Learning I would like to take couple of the scenarios for the small maze. In first scenario, we will keep the learning rate constant to 0.5 and we will have three different values of Epsilon: 0.2, 0.5 and 0.8. In the second scenario, we will keep epsilon to 0.2 and we will keep varying the values of learning rate to 0.2, 0.5 and 0.8 respectively. For the big mazes, let's do first scenario with Epsilon = learning rate = 0.5 and second scenario of epsilon = 0.2, and learning rate of 0.5. In all cases, we will keep PJOG and number of cycles to the default settings, which are 0.3, and 1000 respectively.

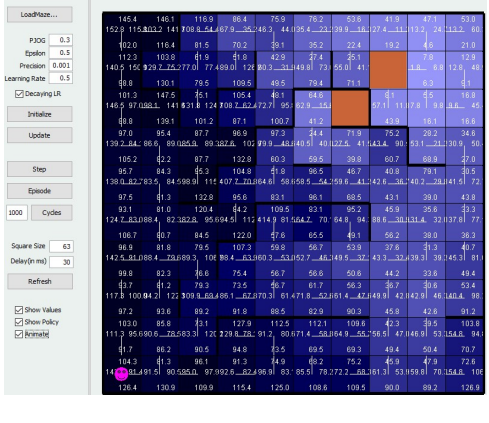
First Scenario for Small Maze

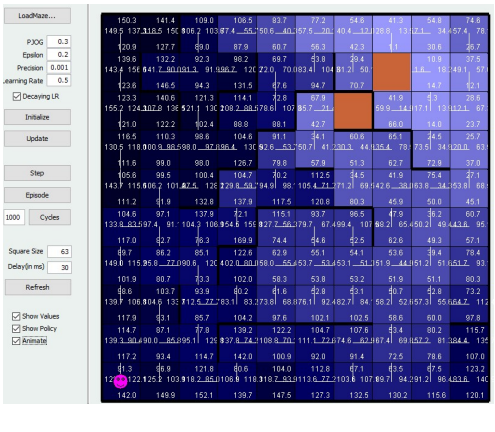
Second Scenario for Small Maze

First Scenario for Big Maze



Second Scenario for Big Maze



For the small mazes we could see that:

$$The\ Learning\ Rate\ \uparrow \Rightarrow number\ of\ less\ optimal\ actions\ \downarrow$$

However, in terms of the big mazes, it has performed the worst.

Conclusion:

- With an absolute certainty I can say that I've learned many of the concepts of reinforcement learning algorithms specifically policy iterations, value iterations and Q-Learning.
- Generally speaking, policy iterations and value iterations converged to have similar solutions for small and big mazes, whereas the Q Learning performed pretty poorly on the big mazes.
- Also, value iterations work much faster than policy iterations, but value iterations necessitates to have two things:
    - convergence will need large number of iterations
    - full knowledge of the state space and the model.
- But, Q-Learning does not require any prior knowledge of the state space or the model. Which leads to agent towards having less optimal solutions, but via gotten feedback it can improve to have an optimal solution in the future.
- However, with the help Q-Learning finding the balance between exploration and exploration of the correct and optimal solution is the key to effectively finding solutions to MDPs.

References:

[1]https://www.cs.cmu.edu/~katef/DeepRLControlCourse/lectures/lecture2_mdps.pdf

[2]https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419

[3]https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265

[4]http://www.cs.cmu.edu/~awm/rlsim/

[5]https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node19

[6]https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node20.html

[7] https://en.wikipedia.org/wiki/Q-learning