Ashwin Rathie
Professor Byron Boots
CS 4641 – Machine Learning

Analysis of Markov Decision Process and Reinforcement Learning

In this analysis, two Markov Decision Processes (MDP) will be used to model a real-life problem. Three reinforcement learning algorithms – policy iteration, value iteration, and Q leaning – will be applied in an attempt to find the optimal strategy. The performances of the algorithms will be evaluated and compared based on the amount of iterations needed, computational cost/runtime, how resistant it is to randomness, and how effective the final strategy is.

The MDPs

What is a Markov Decision Process? An MDP is a mathematical construct that allows us to depict an agent navigating within a world. This world can be described by the following variables:

- S: a set of states that exist in the world that an agent can "be on"
- A: a set of actions that the agent can take which can move it to other states
- $T(s,a,s') = Pr(s' | s, a)$: A "transition model" that represents that probability that an action will actually be executed successfully (this represents the randomness of the world)
- R(s): The reward of being on a state
- $\pi(s)$: The "policy" that the agent will follow to decide which actions to take

Because MDPs are excellent at modeling situations in which states are clear and discrete (like physical locations) and time efficiency is a factor, this analysis will explore the problem of autonomous robots in factories delivering parts to locations where said parts are needed, such as a car manufacturing plant using said robots to deliver requested tools to assembly line stations. I find this to be a fascinating application of reinforcement learning because of the great utility and benefit it could viably have to industrial processes. In manufacturing, efficiency is money, and an effective algorithm for parts-delivering robots could save companies great amounts of money or increase their production output.

A factory floor can be represented as a two-dimensional grid and barriers. This makes it useful to represent the problem with the GridWorld MDP, which is an array of states that can include walls and goal states.

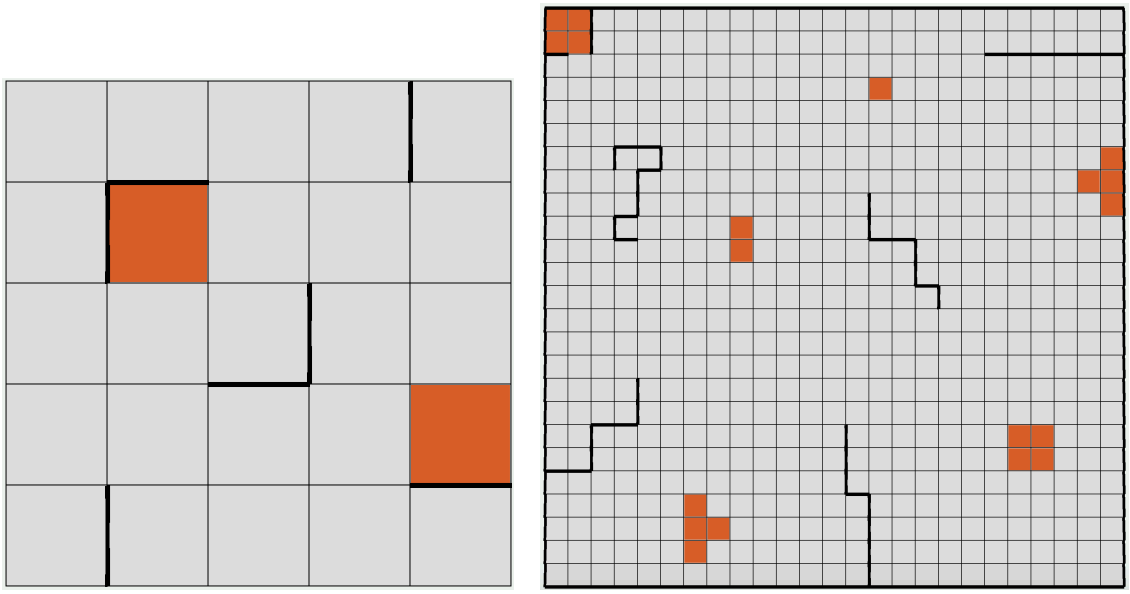
The reward will reflect the fact that factories also contain sensitive equipment and products, represented by walls. Collisions with these must be minimized, so taking an action that would result in hitting a wall will induce a 50-point penalty.

That transition model will be reflected using a parameter called PJOG. The probability that the action that the agent chooses to take is actually executed is equal to (1 – PJOG). Randomness can occur within a factory due to people walking around (restricting a certain action), temporary restricted areas, etc.

Two GridWorld MDPs will be used to model the factory floor problem. One that is relatively small (5 x 5, 25 states) and one that is much larger (25 x 25, 625 states).

In this analysis, we will be assuming that there is no concrete upper limit on how long the robot has to deliver the parts, meaning that we are assuming infinite horizons is in effect. Additionally, we will assume utility of sequences, meaning there will be a logical basis for comparing one sequence of states to another. Specifically, a sequence of states that is shorter will be preferred over a longer one. This will be implemented through discounting (in value and policy iteration that use the Bellman Equation) and through the epsilon greedy strategy for Q learning. Overall, this analysis will assume stationary preferences.

The GridWorlds



SmallFactory BigFactory

Value Iteration – Intro

The logic behind value iteration is essentially to choose the action that will result in the greatest outcome. However, we cannot determine the "best outcome" by the immediate reward R because it could jeopardize greater *long-term* return. One of the properties of MDPs makes choosing the best outcome a challenge: the agent is only making decisions on the information it has about the current state. However, if we could give a utility value to every state representing how being in that state could affect long term benefit, then we can just say that the optimal policy is the one that takes the action resulting in the highest utility. Then, value iteration essentially becomes a greedy algorithm.
The utility of a state can be described as the immediate reward R(s) summed with the discounted reward if the agent made only the optimal decisions following this one. The following steps

1. Initialize an arbitrary utility to every state
2. For every state, determine a new utility using the utility of its neighbors
3. Update the utility U for every state using the Bellman equation:

$$U(s)_{t+1} = R(s) + \gamma max_a \sum T(s, a, s')U_t(s')$$

4. Repeat steps 2 and 3 until convergence

- *Note: Value iteration may never "converge", so convergence is defined here as the algorithm reaches a point at which the changes in utility are smaller than a predetermined precision threshold (.005 in this analysis)

The pseudocode for the implementation of the GridWorld for this analysis is:

initialise $V(s)$ arbitararily
loop until policy good enough
  loop for $s \in S$
    $V_{t+1}(s) = \min_{a \in A}\{1 + \sum_{s'=S}(P(s'|s,a) \cdot x_{ss'})\}$
    $\pi_{t+1}(s) = \arg\min_{a \in A}\{\sum_{s'=S}(P(s'|s,a) \cdot x_{ss'})\}$
    where,
      $P(s'|s,a)$ = Prob. of transition from s to s' after action a.

      $x_{ss'} = V_t(s')$, if transtion from s to s' is safe
        $= Penalty + V_t(s)$, if transtion from s to s' is not safe

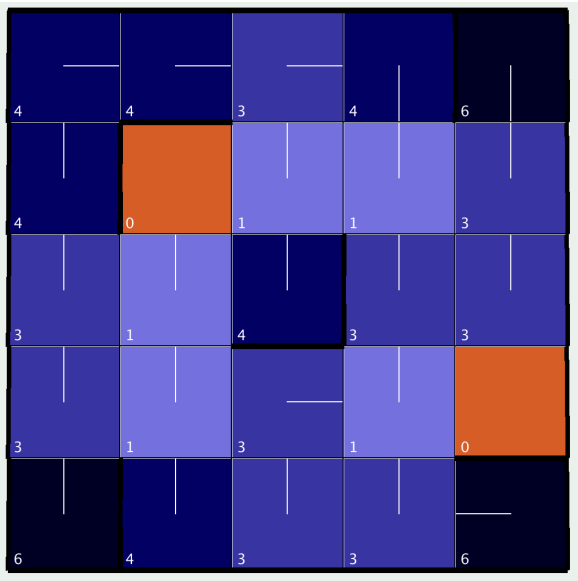
  end loop
end loop

Value Iteration – Results of SmallFactory

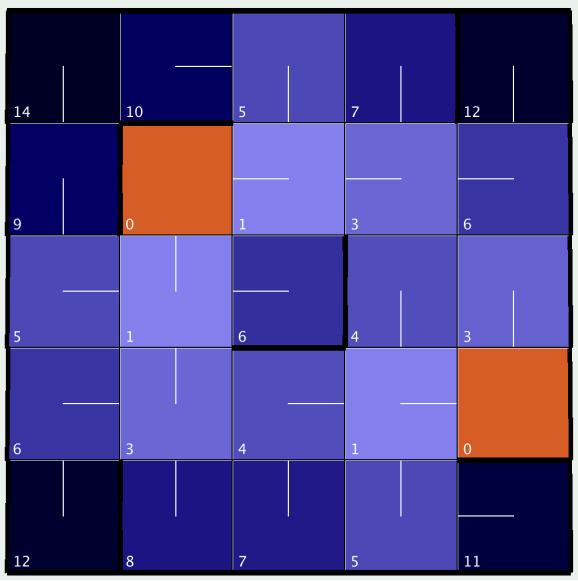
Varying PJOG Table: How resistant to randomness is it?

PJOG	ITERATIONS TO CONVERGE	RUNTIME (MS)
.1	11	2
.2	17	3
.3	25	2
.5	56	7

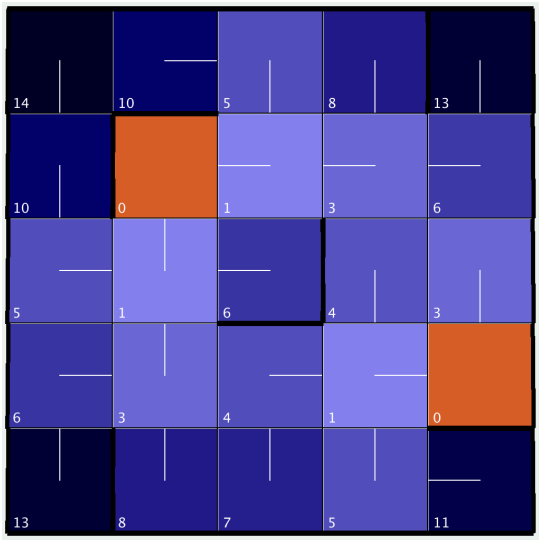
Note: PJOG = .1 for the following images



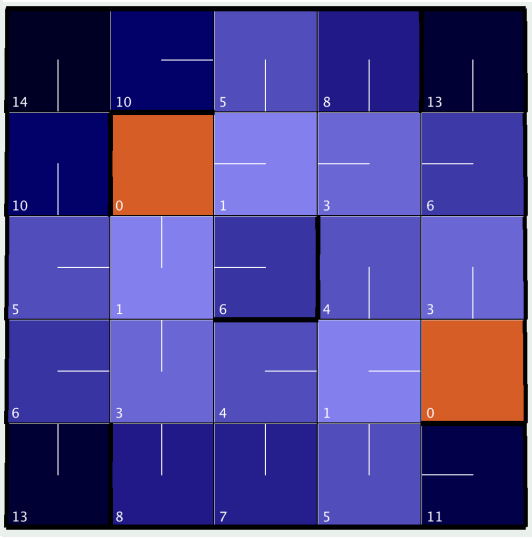
Iteration = 1



Iteration = 5



Iteration = 8



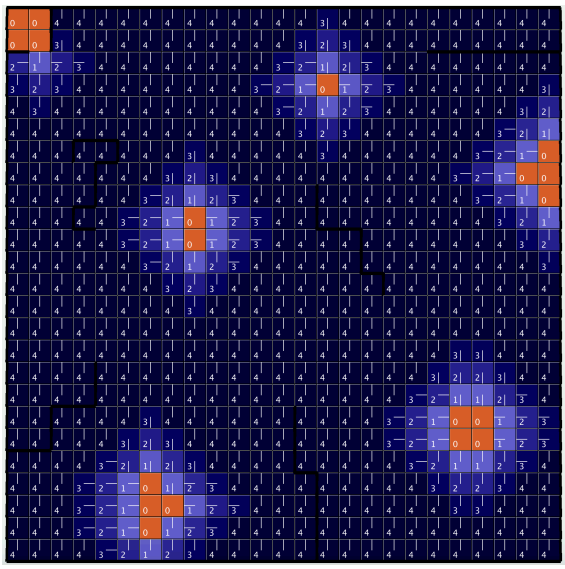
Iteration = 11, Converged (2 ms)

Value Iteration – Results of BigFactory

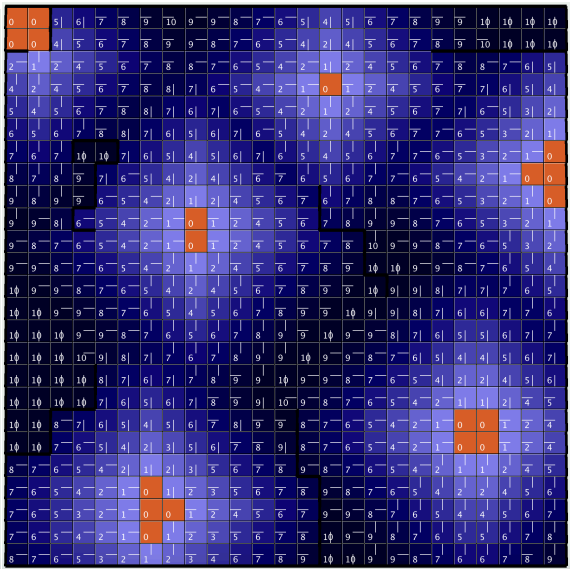
Varying PJOG Table: How resistant to randomness is it?

PJOG	ITERATIONS TO CONVERGE	RUNTIME (MS)
.1	27	60
.2	36	76
.3	48	100
.5	105	227

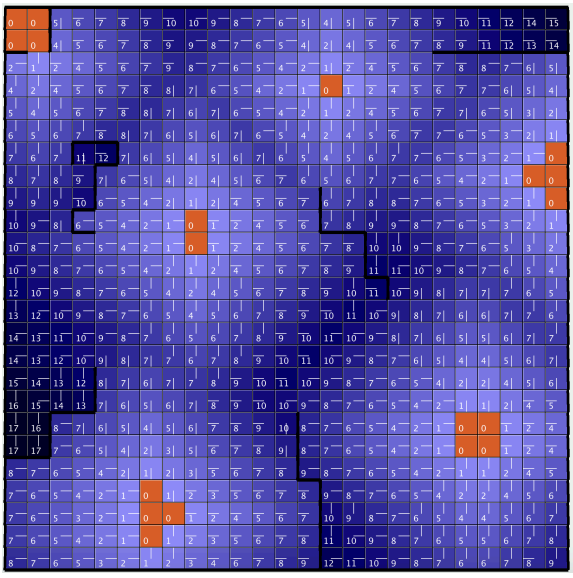
Note: PJOG = .1 for the following images



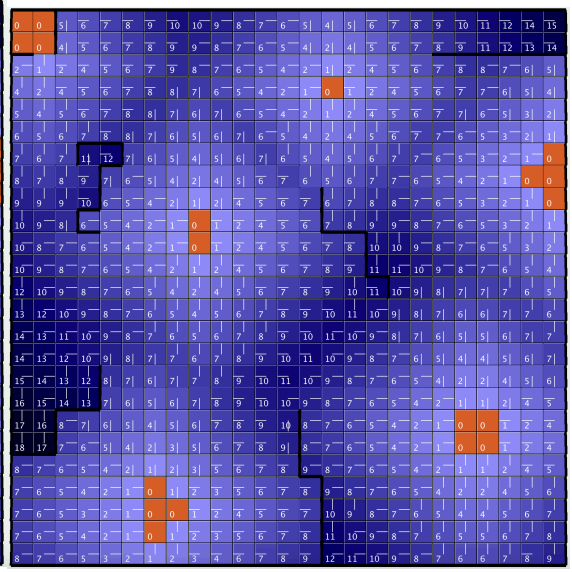
Iteration = 4



Iteration = 10



Iteration = 18



Iteration = 24, Converged (60 ms)

Value Iteration – Analysis

The value iteration algorithm did converge to an optimal policy within reasonable iterations and runtime. In both the small and large GridWorlds, the optimal policy appears to attempt to miss the walls at all cost, which is the behavior we wanted. This, of course, is due to the negative reward we have associated with the robot bumping to into sensitive equipment and product. We also notice that as randomness increases, so do iterations and runtime. This doesn't come as a surprise given the fact that an agent must take a non-optimal route when the previous optimal action is missed, an event that occurs more often when PJOG is increased.

Policy Iteration – Intro

As seen above, value iteration can be an effective method. However, it is known that it is susceptible to long convergence times (especially if the desired precision is very high). Additionally, it only indirectly finds the optimal policy by stopping when the changes in utility between iterations are within a certain tolerance.

With Policy Iteration, we can find the optimal policy directly rather than say the optimal policy is based on assigned value. Instead of iterating over states and assigning utility to them, we will iterate over policies. The algorithm is as follows:

1. Create an initial policy from a permutation of all actions for all states in the MDP
2. Loop through the following until convergence (no actions in the policy change between iterations):
    a. Calculate the utility for each state using the policy at hand
    b. Update the utilities
    c. Assign the new optimal actions to the states, thereby changing the policy

The pseudocode for the implementation follows:

initialise $\pi(s)$ arbitararily
loop until policy good enough
  loop until $V(s)$ has converged
    loop for $s \in S$
      $V_{t+1}(s) = Path\_Cost + \sum_{s' \in S}(P(s'|s, \pi(s)) \cdot x_{ss'})$
    end loop
  end loop
  loop for $s \in S$
    $\pi_{t+1}(s) = \arg\min_{a \in A}\{\sum_{s' \in S}(P(s'|s, a) \cdot x_{ss'})\}$
  end loop
  where,
    $P(s'|s, a)$ = Prob. of transition from s to s' after action a
    $x_{ss'} = V_t(s')$, if transtion from s to s' is safe
      $= Penalty + V_t(s)$, if transtion from s to s' is not safe
end loop

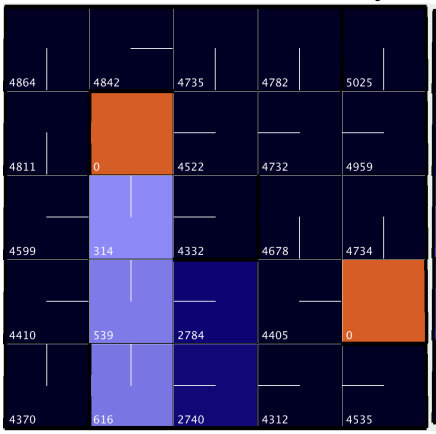
Like value iteration, policy iteration uses the Bellman equation and derives its algorithms from it. However, the calculations for policy iteration boil down to a linear system of equations, rather than nonlinear, making computations less intensive in many cases. Because of the nature of the policy iteration algorithm, it is guaranteed to truly converge, unlike value iteration or Q Learning!

Policy Iteration – Results of SmallFactory

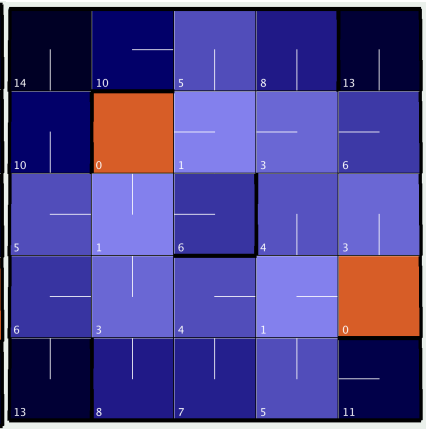
Varying PJOG Table: How resistant to randomness is it?

PJOG	ITERATIONS TO CONVERGE	RUNTIME (MS)
.1	4	5
.2	3	6
.3	4	6
.5	4	10

Note: PJOG = .1 for the following images



Iteration = 1



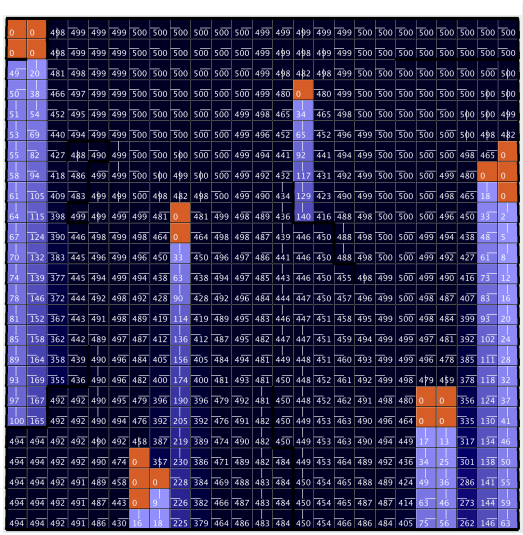
Iteration = 4, Converged (5 ms)

Policy Iteration – Results of BigFactory

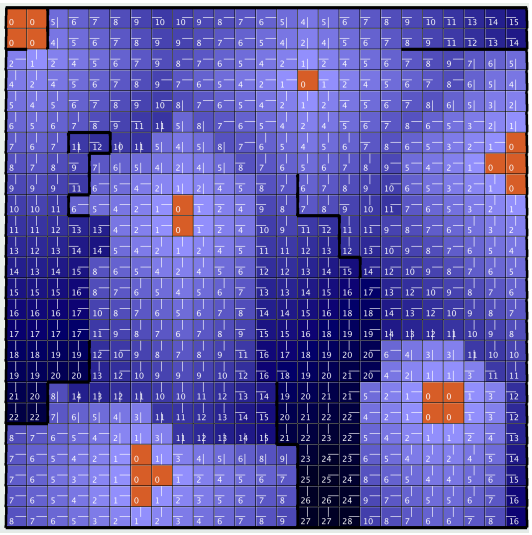
Varying PJOG Table: How resistant to randomness is it?

PJOG	ITERATIONS TO CONVERGE	RUNTIME (MS)
.1	9	365
.2	9	395
.3	7	397
.5	6	504

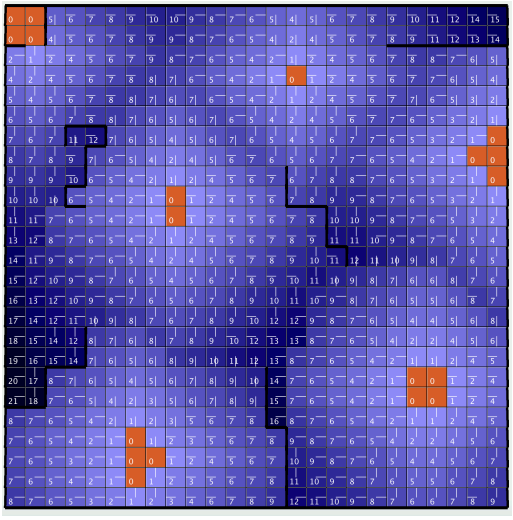
Note: PJOG = .1 for the following images



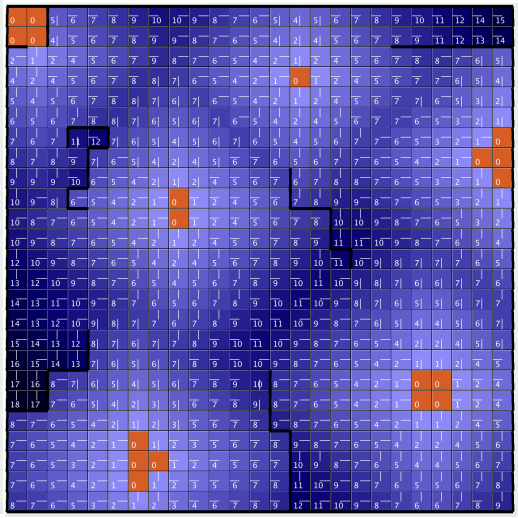
Iteration = 1



Iteration = 3



Iteration = 6



Iteration = 9, Converged (365 ms)

Policy Iteration – Analysis

Both value and policy integration ultimately converged to same optimal policy, so the accuracy appears to be even. Very interestingly, policy iteration doesn't seem to worsen at all when, the PJOG value, is increased. Not only did the number of iteration stay the same or better, but the runtime was relatively constant as well. Furthermore, the larger gridworld for policy integration required far less iterations than that of value integration. This is likely the result of the linear computation load required for policy iteration compared to the nonlinear, more complex computation for value iteration. For these reason, policy integration appears to be the superior method for this type of problem.

Q Learning – Intro

A problem with value and policy iteration is that both of them require full planning before anything is done. That is to say that the entire model must be calculated initially, and only then can our factory parts delivering robot get to work. Q Learning is a model-less reinforcement learning algorithm. Instead of determining a model using something like the Bellman equation, it instead creates an approximation of the optimal solution essentially by trial and error. It assigns values for each pair of

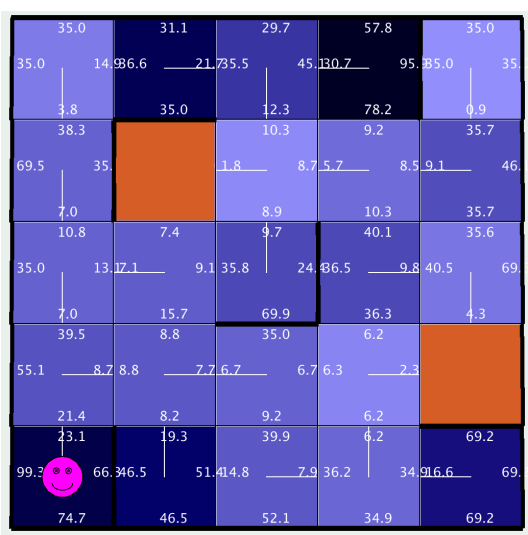
states and actions and puts it in a Q table. This value based on the expected total of future reward. After the Q table is full, the optimal policy is to simply follow the action that has the highest Q value at any state, reminiscent of the greedy algorithm feel of value iteration.
The Q values are determined using the following equation:

$$Q(s, a) = R(s) + \alpha \sum T(s, a, s')max_{a'}Q(s', a')$$

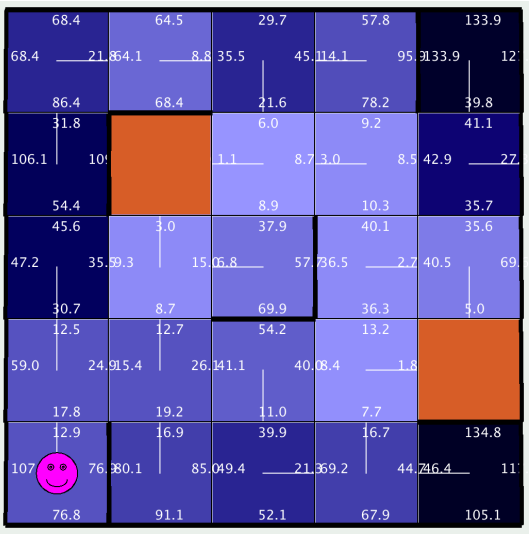
In reinforcement algorithms, the idea of Exploitation vs Exploration should always be considered. Exploitation is the idea that an agent should take the best possible action that it knows at all times. Exploration is the idea that an agent should sometimes take a different action than what may seem to be optimal. In this Q learning algorithm, the epsilon greedy strategy is employed. In the beginning, epsilon rates will be higher and the agent will explore more. As time goes on, the epsilon value will lower and the agent will begin to exploit, acting on what it "learned" in the exploitation phase.

Q Learning – Results of SmallFactory

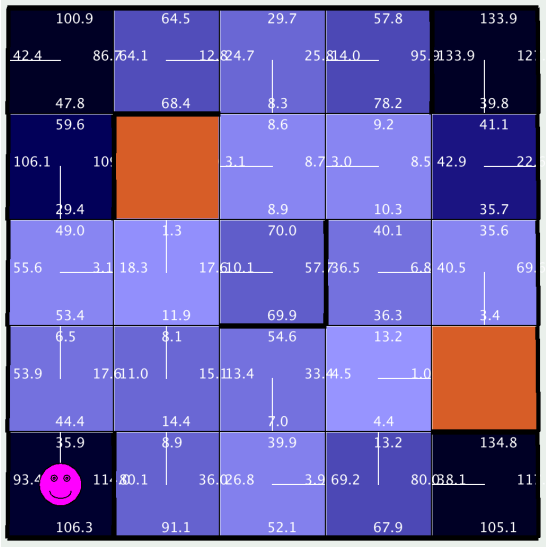
**Note: Epsilon is kept constant at .1, Learning Rate is constant at .7 and PJOG is constant at .3*



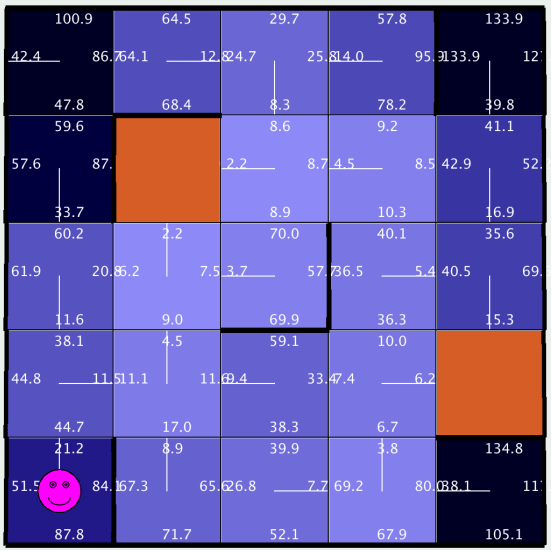
Cycles = 25



Cycles = 50



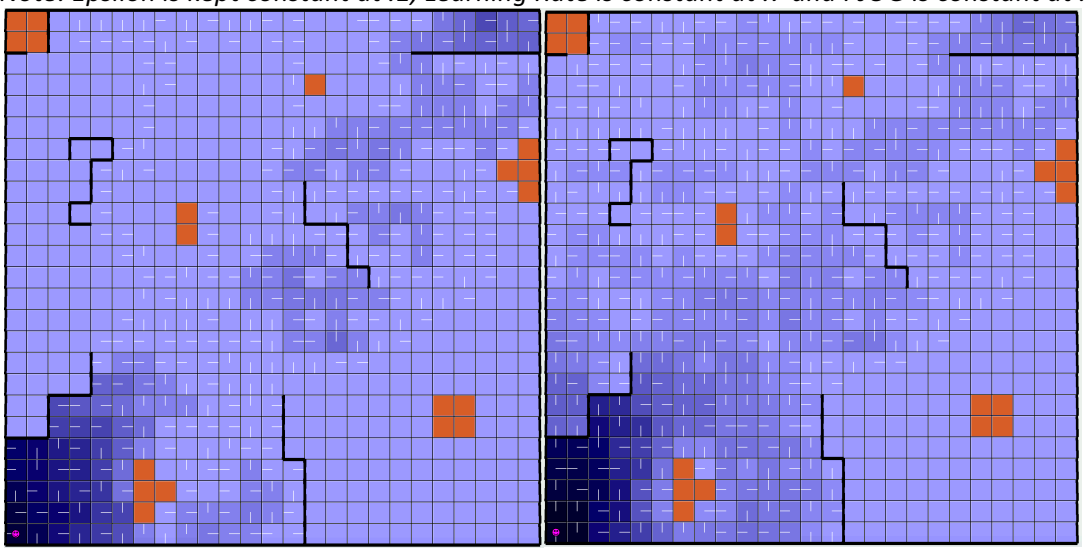
Cycles = 75



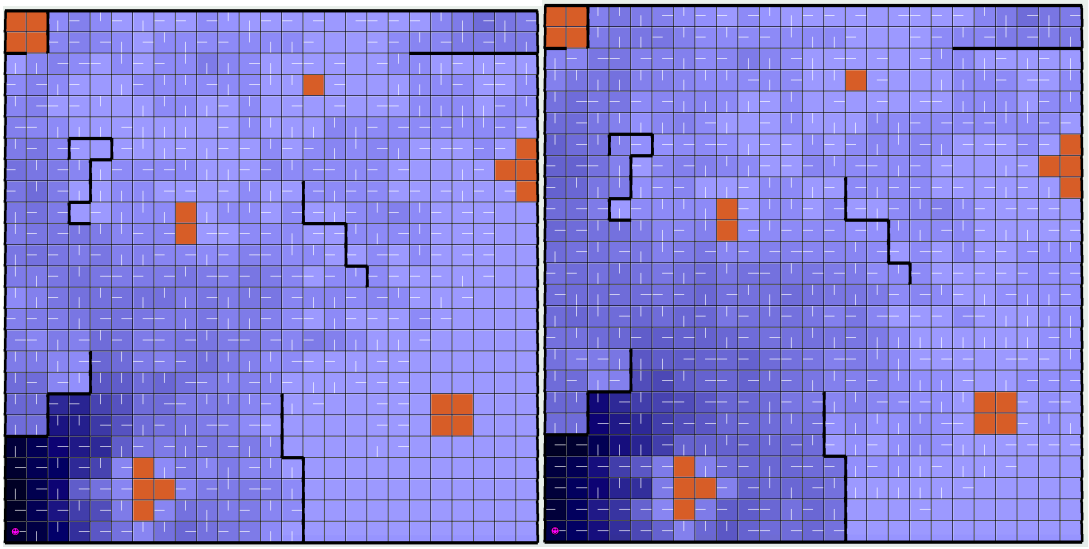
Cycles = 100

Q Learning – Results of BigFactory

**Note: Epsilon is kept constant at .1, Learning Rate is constant at .7 and PJOG is constant at .3*



Cycles = 25 Cycles = 50



Cycles = 100 Cycles = 1000

Q Learning – Analysis

While value and policy integration ultimately came to the same optimal policy, Q learning did not. Although the final policy was very similar, especially for SmallFactory, there are slight differences. This could be due to the epsilon factor that is enabling the agent to explore more in addition to having a greater PJOG. These both contribute to the agent taking non-optimal actions.

Q learning is faster in the sense that, because it is model-less, it runs faster initially. However, it must be noted that for the BigFactory MDP, it took many cycles to being to create a completed policy. Even after 1000 cycles, there are states that have yet to be visited. If the goal is to create a complete policy, value and policy integration may be beneficial when dealing with larger scale MDPs.

Works Cited:

https://www.cs.cmu.edu/~awm/rlsim/
https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/qlearning_simple.html
https://people.eecs.berkeley.edu/~pabbeel/cs287-fa12/slides/mdps-exact-methods.pdf