Analysis of Multiple Reinforcement Learning Algorithms

For this report I will be analyzing the effectiveness of multiple reinforcement learning algorithms on sample mazes provided by Carnegie Mellon University for its MS in Information Technology.

1.0 Algorithms Selected

For this paper I'll be experimenting with different reinforcement learning algorithms, listed below. For the algorithms, we will be using the Bellman Update Equation.

1.1 Value Iteration

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

Img 1.1.1 Procedure for Value Iteration (CMU)

For value iteration, on a high level we follow a procedure in which we initialize our state space values randomly (or start at 0). Then for each state, we calculate the reward or penalty, or value, *v*, and take the smallest (for penalty) or largest (for reward) transition, which will be our policy for our state. We continue running through this algorithm until we converge to a policy that is good enough. Transitions can be stochastic, so we also incorporate that into our reward/penalty calculation. Value iteration is supposed to converge quickly because it only cares about the optimal policy for each state.

Value iteration guarantees to converge to the optimal value function.

1.2 Policy Iteration

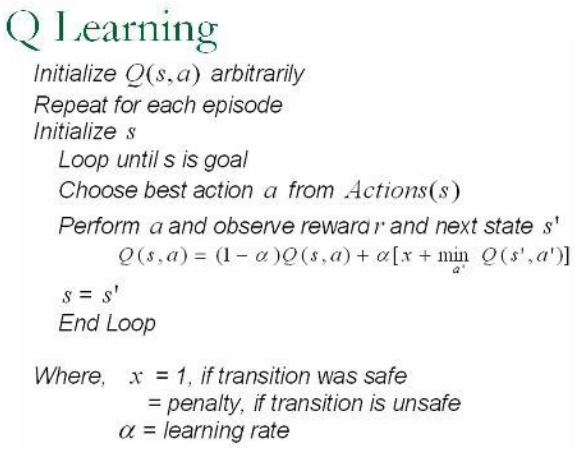
Policy Iteration
initialise $\pi(s)$ arbitararily
loop until policy good enough
  loop until $V(s)$ has converged
    loop for $s \in S$
      $V_{t+1}(s) = Path\_Cost + \sum_{s' \in S}(P(s'|s,\pi(s)) \cdot x_{ss'})$
    end loop
  end loop
  loop for $s \in S$
    $\pi_{t+1}(s) = \arg\min_{a \in A}\{\sum_{s' \in S}(P(s'|s,a) \cdot x_{ss'})\}$
  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

Img 1.2.1 Procedure for Policy Iteration (CMU)

For policy iteration, we follow a similar procedure to value iteration. We start by initializing an arbitrary policy for each state in our state space. Then we calculate the optimal policy at that state (i.e. the best transition to the next state from the current state) and update this policy. We continue this process until convergence.

Policy iteration is also optimal. It may be faster in some circumstances to Value Iteration, we'll test this out when we see the experiments.

1.3 Q-Learning



Img 1.3.1 Procedure for Q Learning (CMU)

For my favorite Reinforcement Learning algorithm, I chose Q Learning. This is a model free learning algorithm where the agent interacts with its environment and updates its internal mapping of "Q" values. The Q values are a mix of the Bellman Update Equation and an equation for optimizing the explore exploit tradeoff. The agent will essentially perform an action based on the best possible action available to it, with some chance that it will go explore somewhere else. As time progresses, the Q values will be updated and the agent will weight successful paths more than others eventually coming to a policy. Since we can technically do this infinitely, we want to limit the number of times the agent does this, limiting the episodes it runs essentially.

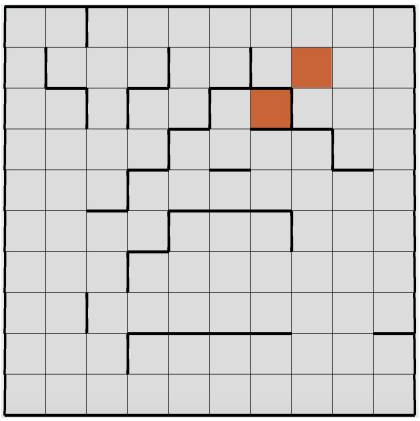
This algorithm is best for large state spaces where it's impractical to use value iteration or policy iteration to find the all states-optimal path.

2.0 Description of Problems

Here I will describe the problems I will be solving using the reinforcement algorithms. I liked these mazes because they highlight obstacles, much like those in house plans or such. I especially think they're well suited as high level introductions to planning mazes. I think the coolest application of this would be creating agents for assisted living, where a robot would need to go to a different room to pick up some item(s) and possibly return.

2.1 Small Maze (100 states)

For this problem I'm analyzing the following maze. Take a second to look at and study it.



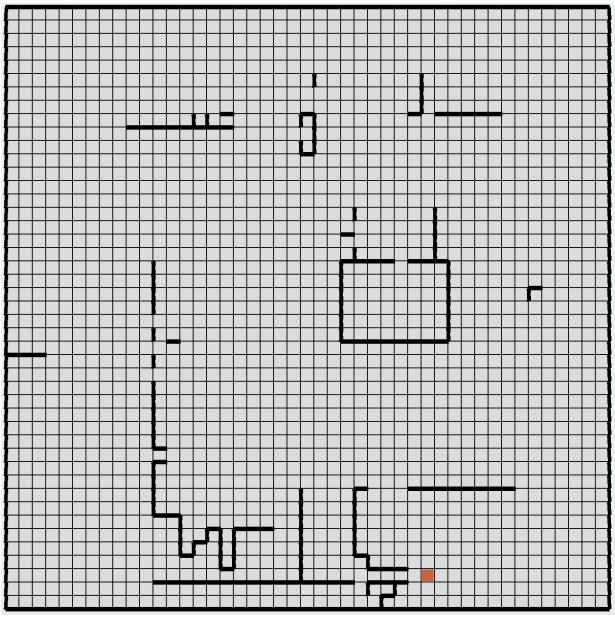
Img 2.1.1: Small maze with 2 goal states, and walls (100 states)

As we can see, this is an interesting problem because we have two close goal states, but they are separated by a number of walls. For the bottom goal state, we see that it's really only accessible by a corridor, but this might influence the agents to separate their policies between the bottom and top parts of the maze.

The initialization point for our agent will matter because it may end up in one of the two different goal states. I also find it interesting because if we increase the noise of our environment, then the policies could change where it would not be optimal to go to the bottom goal state because of how closed off it is.

2.2 Big Maze (2025 states)

For this problem I'll be analyzing the following maze. Take a second to look at and study it.



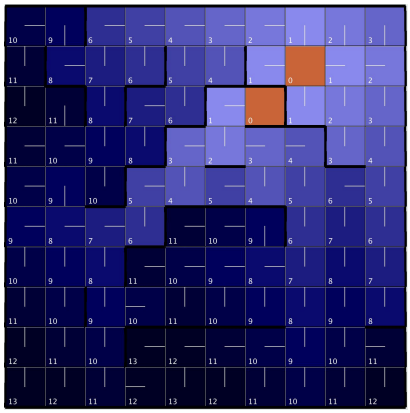
Img 2.2.1: Large maze with 1 goal state and many walls and corridors (2025 states)

This is an interesting maze because it has a number of corridors where there is only one fast option to get to the goal fast. It will be interesting to see how the algorithms react when say, the penalty is very high. I am also curious to see how well Value Iteration and Policy Iteration fair against Q Learning in this maze, since Q learning is faster, though not necessarily globally optimal, than value iteration and policy iteration because it won't need to explore every state.

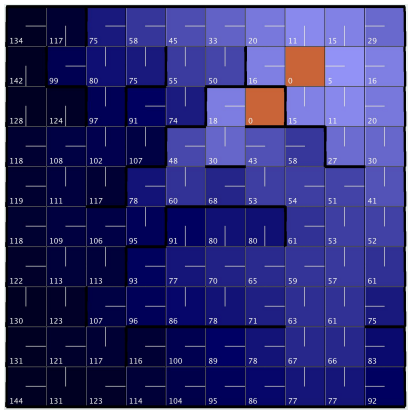
3.0 Results on small maze

3.1 Value Iteration

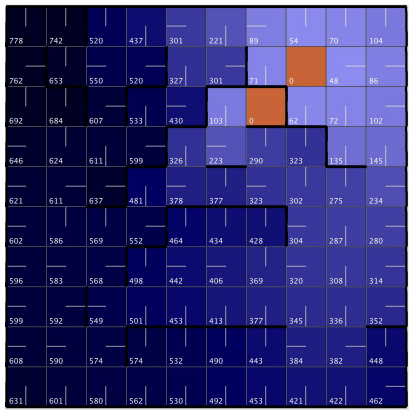
For Value iteration, I wanted to see how the algorithm faired when it had to deal with a lot of noise. The expectation is that value iteration would take longer to reach my threshold precision because of how arbitrary each state is. I also expect it to not have as clear of a path as noise increases.



Img 3.1.1: 0% noise, 14 steps



Img 3.1.2: 20% noise, 62 steps



Img 3.1.3: 100% noise, 653 steps

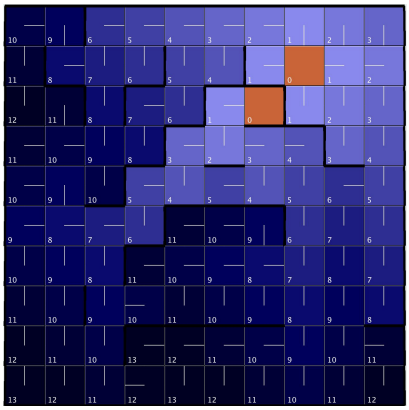
In the above photos, the numbers in the boxes represent value functions (penalty). The lower the better. In the 0% noise image, we can see there are clear paths to the goal states, especially in the corridor where the

bottom goal state is the only reasonable option. As the noise increases, we can see that even being near the bottom goal state, the value function is high and even lead away from the goal state.

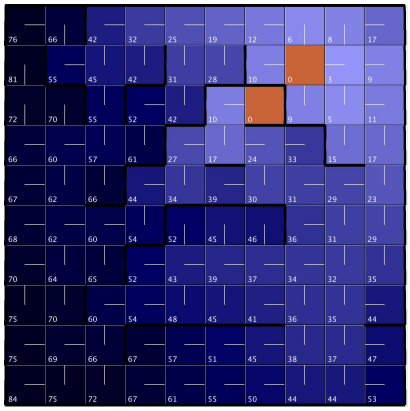
We can see that the noise also drastically increased the number of iterations required to reach convergence. This is because of the discount factor associated with the Bellman update. Future values are discounted more, and the likelihood of getting to the goal state is reduced.

3.2 Policy Iteration

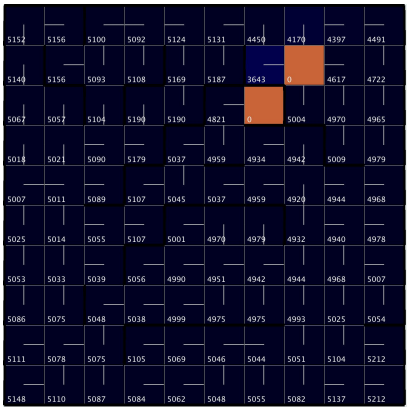
For policy iteration, the expectation is that it will converge faster than value iteration, and that it will likely have a similar policy to value iteration.



Img 3.2.1: 0% noise, 13 steps



Img 3.2.2: 20% noise, 6 steps



Img 3.2.3: 100% noise, 8 steps

After running the algorithm, we can see that indeed, value iteration and policy iteration produced the same policies, except for the high noise one. They are likely different because I set the precision too high for value iteration to fully converge to the policy iteration solution.

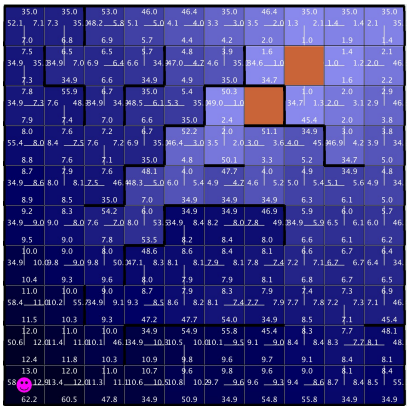
The two algorithms, as stated before converge to the optimal policy. Value iteration just takes longer and more iterations, while policy iteration builds the optimal policy and can take less time than value iteration.

3.3 Q Learning (100 cycles/episodes):

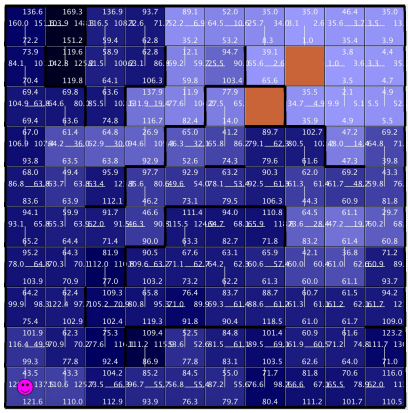
For Q Learning I tuned a number of different parameters to see how they would affect the final result.

3.3.1: Noise

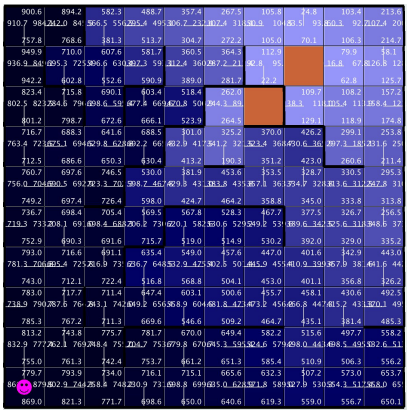
I would expect that as the agent is dropped into a more and more stochastic environment, it'll have a more difficult time finding a good policy, much like value iteration and policy iteration. I think the difference will be that since previous episodes build off each other, eventually the agent can find a policy, even if it's not the optimal one.



Img 3.3.01: 0% noise



Img 3.3.02: 20% noise



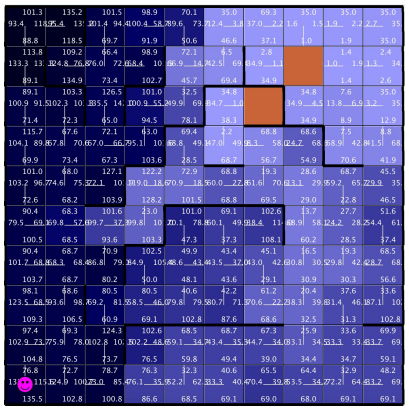
Img 3.3.03: 100% noise

We can see that the policy created for 0% noise is similar to 0% noise ones for value iteration and policy iteration. Interestingly the agent found the corners to be optimal and some specific regions on the right side

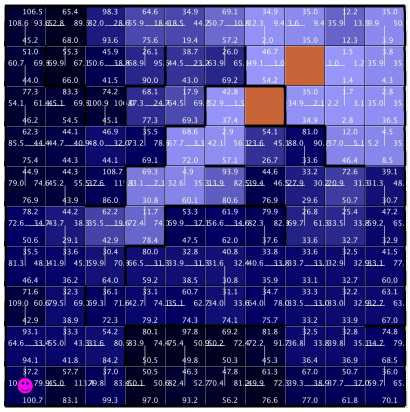
with the 20% noise environment. This is probably because of the agent starting at those points and finding a path from there quickly. The last noise graph is similar to that of value iteration and policy iteration. This means the agent was able to still hone in and see the states that were near the goal states were optimal.

3.3.2 Epsilon (explore, exploit tradeoff)

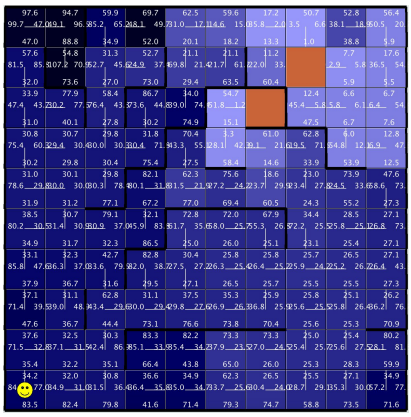
For the explore, exploit trade off, I would imagine that the agent would hit a “peak” where the tradeoff is balanced. If the epsilon is less than that value (bias to exploit), then the agent will have a policy to get to the goal state, but it might be inefficient. If the epsilon is greater, then I think the agent would take a longer time to converge, or not find an optimal policy yet.



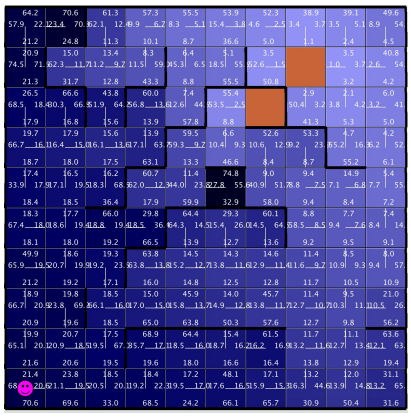
Img 3.3.04: 0 epsilon



Img 3.3.05: 0.1 epsilon



Img 3.3.06: 0.5 epsilon

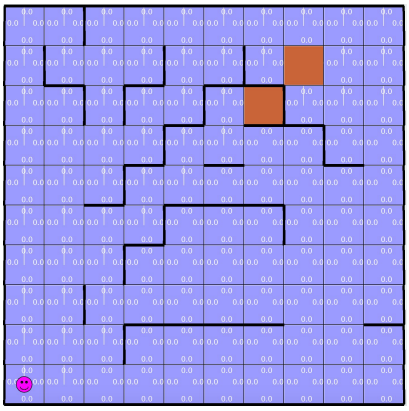


Img 3.3.07: 1 epsilon

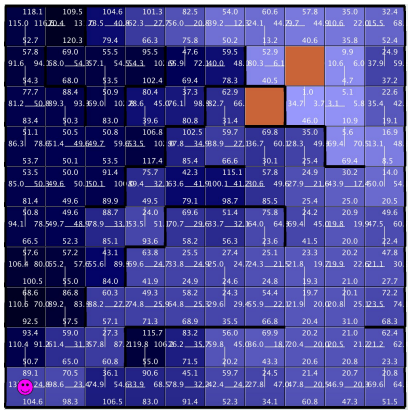
In the images, we can see that in the first one with 0 epsilon, the agent seems to have found a path that goes around the bottom of the maze, which is an inefficient route. With an epsilon of 0.1, the agent made a very strong path towards the bottom goal state. In the following states where exploration was preferred, the agent found paths, but didn’t seem to have enough time to always converge to a good solution. Note the value below the bottom goal state points away from the closest goal state.

3.3.3 Learning rate (alpha)

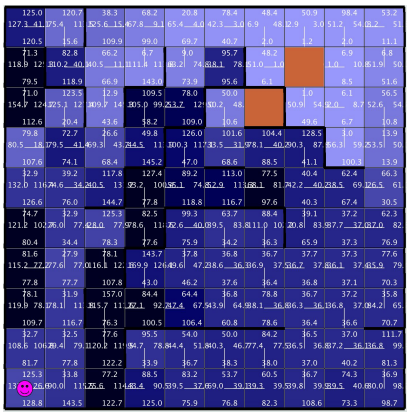
When we change the learning rate, we change the propensity for the agent to drop old knowledge and take new knowledge. Essentially what that would mean is how much bias we have towards values newly learned, vs old values. Hence, we expect if it’s 0, then we wouldn’t learn anything and just keep the same randomly initialized Q, while if it’s 1 it’ll essentially turn into value iteration for one episode as it would weight its previous observations less.



Img 3.3.08: 0 learning



Img 3.3.09: 0.7 learning

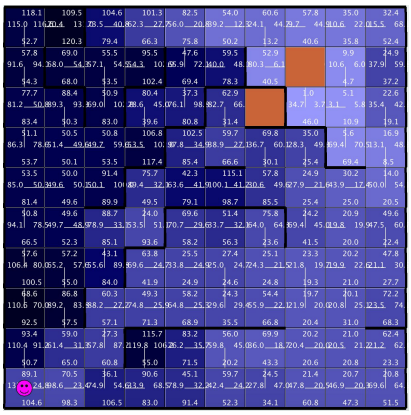


Img 3.3.10: 1 learning

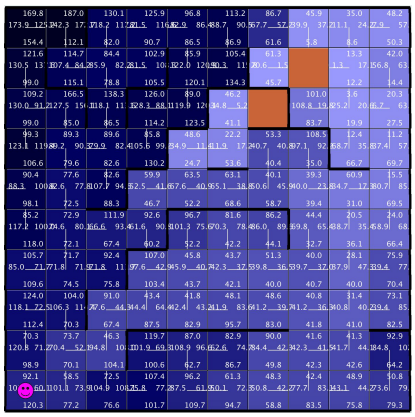
As expected, with 0 learning, our agent didn't acquire any new knowledge and kept the same 0 weights as before. For 0.7 learning, it was able to find a decent strategy by going through the bottom portion of the maze, but completely missing the more optimal path going through the corridor above it. For the agent with learning rate 1, it bypassed the second goal and went for the top goal, which is likely due to it not new knowledge over that of previous episodes.

3.3.4 Decay

For decay, we should expect to see values that are near the goal be less valuable the further away we get. If we have no decay, then we will likely end up concentrating on one of the two goals, likely the closer one because that gets us our reward faster.



Img 3.3.11: Decay on



Img 3.3.12: Decay off

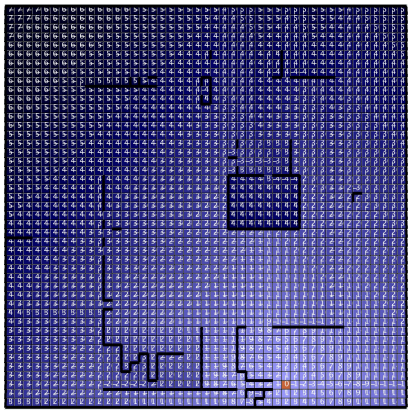
In the above images we can see that with decay on, the values lead towards the top goal more, this is probably because it is easier to reach than the bottom goal. For the no decay graph, we see that the values near the goals are highly valued. Since there is no decay, these values are at their maximum and aren't as "normalized" as decayed values.

4.0 Results on second maze

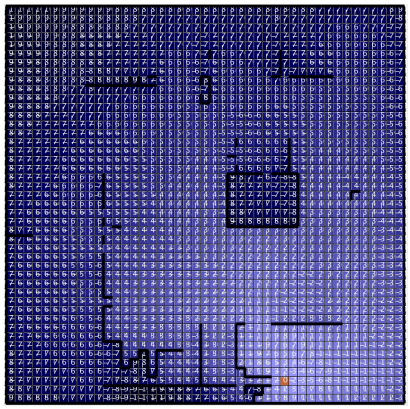
For these mazes, the graphs are really zoomed out, so I would recommend just looking at the color gradient to get an idea of the policy.

4.1 Value Iteration

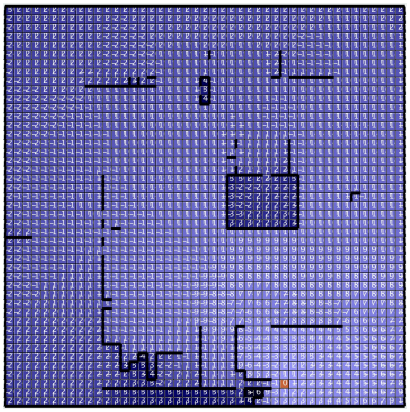
For a larger state space than before, I would expect that the algorithm will take longer in general to run than the smaller maze from before. Along with this, as we increase the noise, since there is only one goal, most of the values would be weighted similarly unless they're in some restricted portion of the maze. I'd expect that the algorithm would dislike going into enclaves.



Img 4.1.1: 0% noise, 74 steps



Img 4.1.2: 20% noise 136 steps



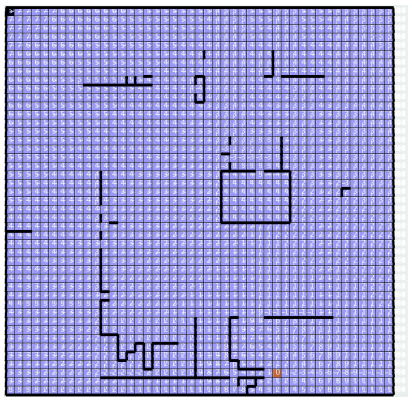
Img 4.1.3: 100% noise 960 steps

We can see that with no noise there is a clear gradient emerging where the optimal policy points to getting to the goal state. The gradient darkens for all the graphs when it's near the middle box section and bottom corridors (except 0% noise). This is because getting trapped in that box would make it difficult to get out of, especially as the noise increases. For the 0% noise algorithm, there is no trouble with the bottom corridor though, because it is able to make deterministic moves and not experience problems with getting through the tiny gaps to reach the goal.

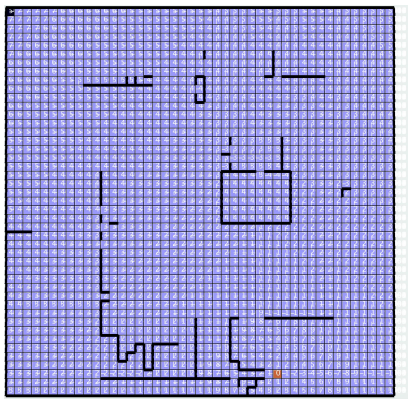
Also as anticipated, in the 100% noise graph, the values around the graph are generally the same because of how stochastic the noise makes the value function.

4.2 Policy Iteration

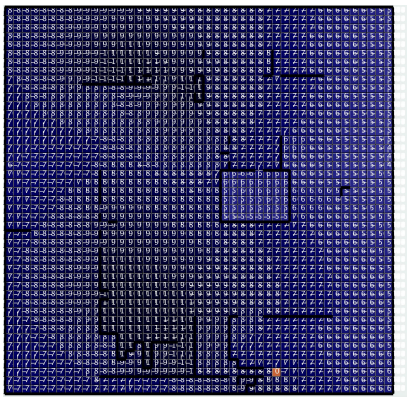
For policy iteration, as mentioned in the previous maze, we should expect to see the same policies, but with the expectation that policy iteration converges faster.



Img 4.2.1: 0 noise 17 steps



Img 4.2.2: 20% noise 17 steps



Img 4.2.3: 100% noise 139 steps

It might not be visible, but the policies for the first two images is the same as value iteration. The policy for the last image might be different because of the ending precision limit set.

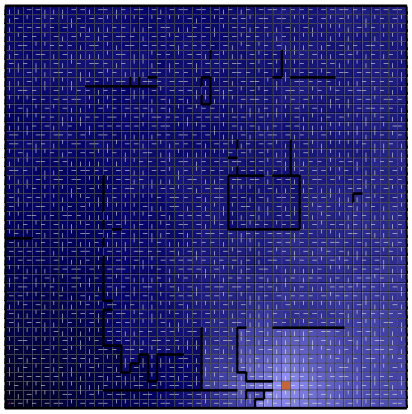
Policy iteration converged significantly faster than value iteration, taking nearly 80% fewer steps. This is because policy iteration updates a policy until convergence, which is cheaper than calculating many values and finding the optimal policy in that.

4.3 Q-Learning (100 cycles/episodes):

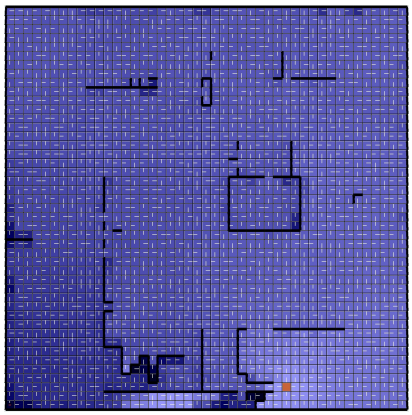
The agent will be run for 100 episodes for each experiment.

4.3.1 Noise

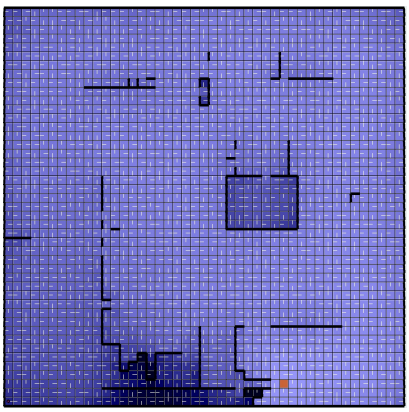
Much like the noise for the smaller maze, I'd anticipate that the noise would normalize the value function, minus the enclaves.



Img 4.3.01: 0% noise



Img 4.3.02: 20% noise

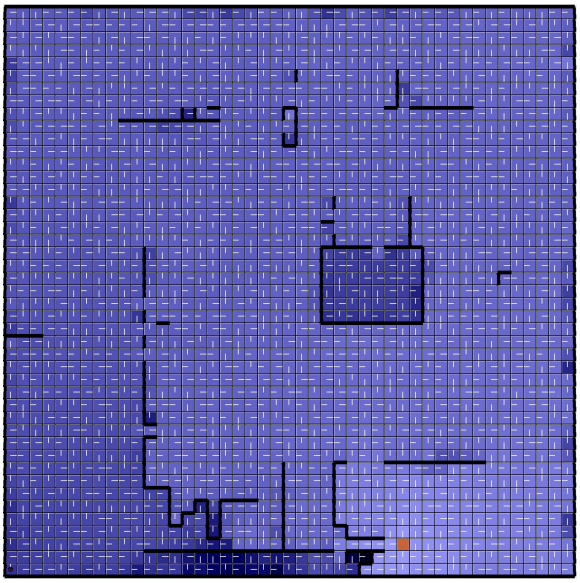


Img 4.3.03: 100% noise

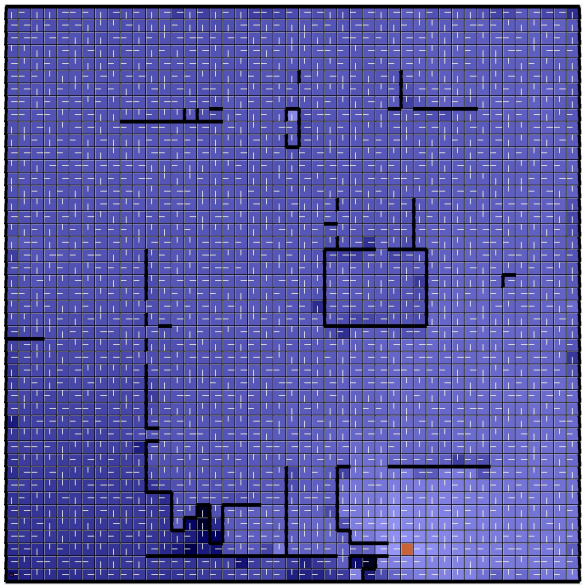
Interestingly with the increase in the size of maze, it became significantly harder for the agent to find the goal state. As we can see in any dark patch of the 0% noise model, it has a tendency to arbitrarily search (different from epsilon). The noise also exasperated it a lot and made it very hard for the agent to navigate, especially in the dark bottom corridor.

4.3.2 Epsilon (explore, exploit tradeoff)

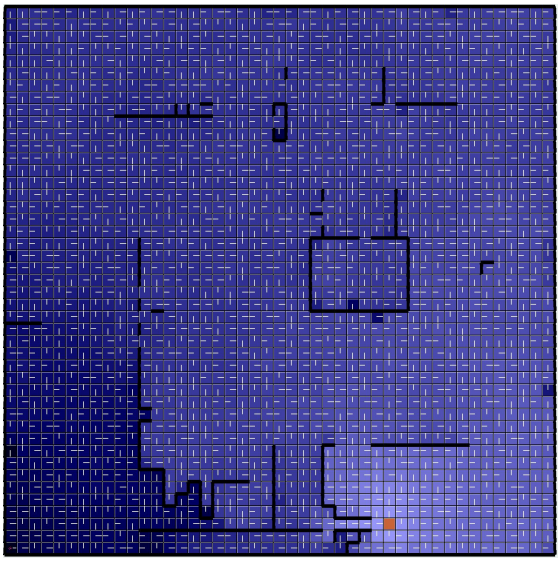
When increasing the epsilon, i'd expect the agent to have more trouble getting to the goal state than before, because it might not be able to converge fast enough to the solution. Similar to the previous maze, I'd expect there to be a peak epsilon that leads to the optimal policy.



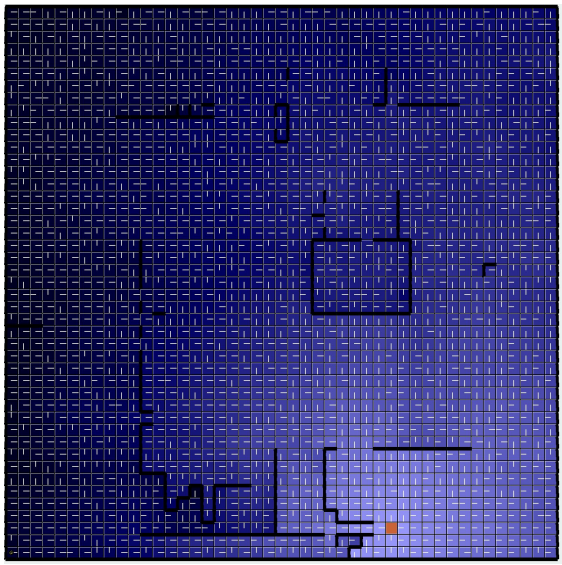
Img 4.3.04: 0 epsilon



Img 4.3.05: 0.1 epsilon



Img 4.3.06: 0.5 epsilon



Img 4.3.07: 1 epsilon

Surprisingly, the increase in epsilon generally helped the algorithm get to the goal state. In 0.5 epsilon we see that there were still some struggles where the agent would go back and forth, but it was able to figure out that going into the corridor may not be a good idea.

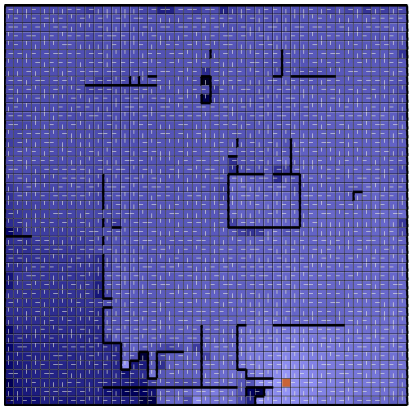
The increased epsilon gave the agent the chance to search the feature space and come up with some optimal policies. It ends up looking similar to value iteration.

4.3.3 Learning rate (alpha)

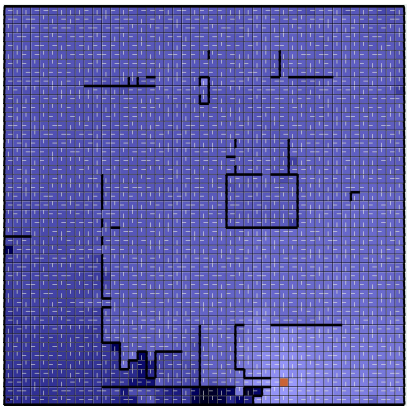
Sticking with the same assumptions as before, I'd expect to see no change to the feature space when there is 0 learning rate, and as we increase there will be a sweet spot where

Learning rate = 0 took too long to run

Img 4.3.08: 0 learning



Img 4.3.09: 0.7 learning

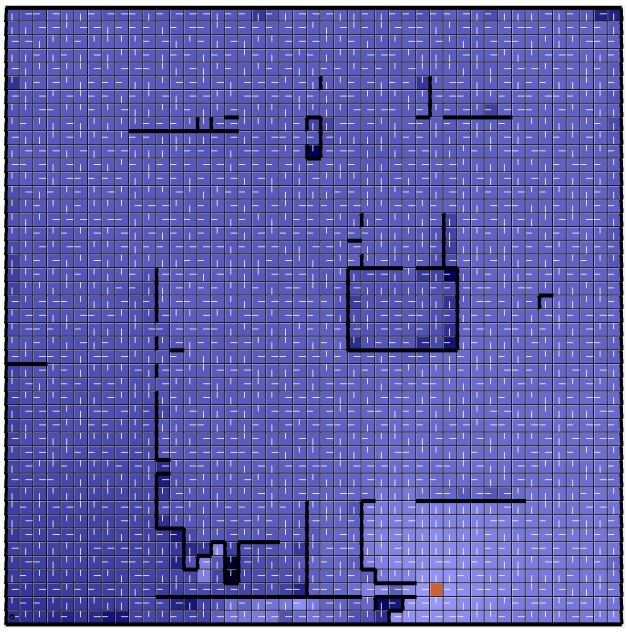


Img 4.3.10: 1 learning

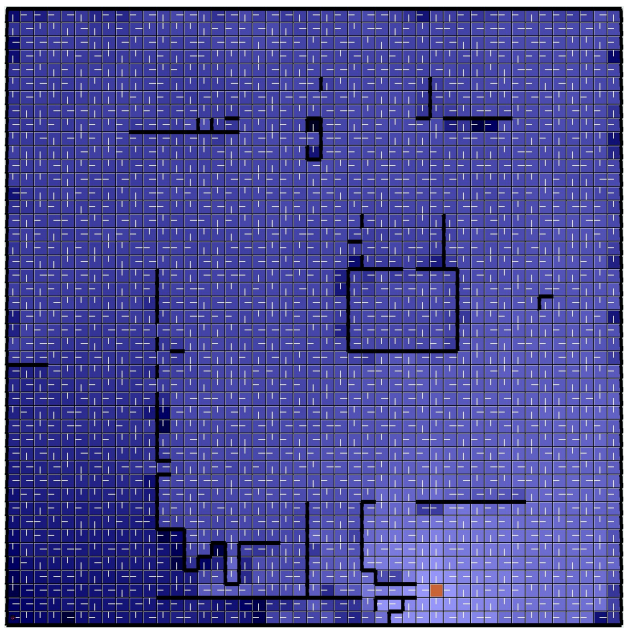
In my simulation, when the learning rate was tuned to 0, there appeared to be a bug, or it was not able to complete. However, as the learning rate increased, We can see interesting parallels. Take for example the bottom corridor, the agent is able to figure out how to exit with 0.7 learning rate, but for 1 learning rate, the agent may have calculated a different optimal action and ended up changing the policy to get away from the goal state.

4.3.4 On/Off Decay

For decay, I'd anticipate that this would have little affect on the final result, because in this case we're only working with one goal state. The places I would anticipate it would matter is when the agent is stuck, so in the middle rectangular section or such. I believe then having no decay would help the agent.



Img 4.3.11: Decay on



Img 4.3.11: Decay off

For this it looks like the decay didn't have much effect. In the middle rectangular section it also had some trouble getting out in both graphs.

5.0 Conclusion:

To conclude there were a number of different things that could affect the reinforcement learning algorithms.

State space size: The increased size of the second maze caused the algorithms to take far longer to run. It also reduced the quality of the solutions in general.

Noise: This would cause the optimal policy to take far longer to compute for value iteration. Policy iteration did better and generally computed the policy faster than value iteration. Q learning caused the agent to struggle to find the goal state and take a long time to produce solutions. They were also not good solutions.

Q Learning specifics:

Epsilon: The tradeoff generally followed a trend where it was better to explore some amount than not at all or too much. Too much exploration made the algorithm take longer, but perform well.

Learning Rate: The learning rate weighted recent actions more than older actions, which caused the agent to sometimes unlearn good old actions.

Decay: The decay had little affect on the larger state space with 1 goal, but focused the results on one goal for the smaller maze.

Overall, Policy Iteration and Value Iteration produce similar results, but policy iteration in this type of problem computed the solution faster than value iteration. Policy Iteration and Value iteration also seemed to outperform Q learning. I expected Q learning to be better at the larger problem set, but it had troubles and wasn't quite able to do well because of all the walls and blockers that threw off the agent.

In an more realistic Reinforcement Learning setting I would experiment more with Q Learning, especially because large state spaces would be very inefficient for policy iteration and value iteration. Along with that Q learning would do better in a stochastic and not fully observable environment.

Citations:

UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/index.php

Simulator: https://www.cs.cmu.edu/~awm/rlsim/