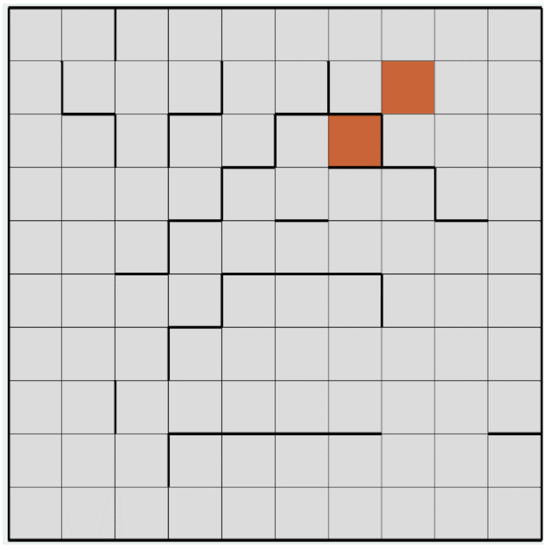
Robert Bradshaw
4/23/2017
CS 4641

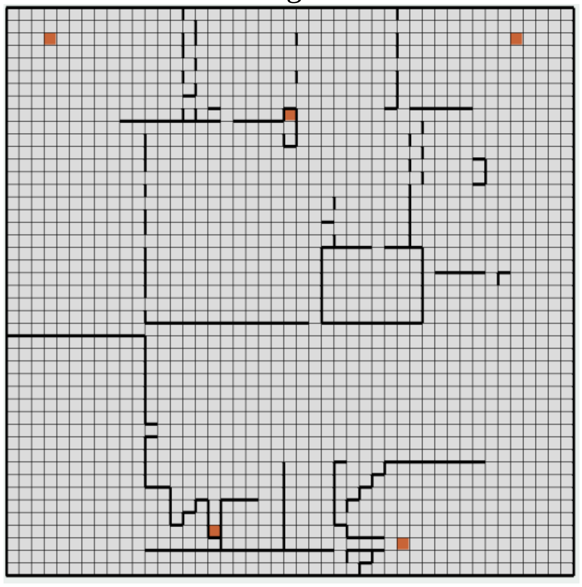
Assignment 4: MDP/RL

For this assignment, I'm using RL_sim -- a Java executable created by Carnegie Mellon University associates Rohit Kelkar and Vivek Mehta. The two problems I decided to solve are both maze problems that I stole from a Github fork of this project by a previous Georgia Tech student in CS 4641. I will refer to the first maze as 7Medium and the second one as Big2.

7Medium:



Big2:



<u>**Problem Description**</u>

So, as we all know, Markov Decision Processes have some immutable properties:

- Agents in some arbitrary state S can take an action A to make it to another state S' as defined by the transition function T(S,A,S').
- Agents receive a reward as defined by some "laws of the universe" when moving from state S to state S'.
- Agents do not maintain knowledge of any previous actions when trying to decide what next state to transition to.
- Discount factors (gamma) are applied to later rewards in MDP in order to "discourage" flip floppy behavior in the agent as it converges to a solution – this makes the policy become asymptotically more greedy over time.
- MDPs attempt to find some policy π(s) that maximizes the discounted sum of rewards (i.e. long-term utility).

The transition function ***T(S, A, S')*** permits the agent to move in any direction that isn't the opposite of the selected action. Touching any wall results in a -50 reward. For both mazes, staying in the same space allots a -1 reward (to incentivize the agent to keep moving in instances where it may determine not moving is the best way to maximize its reward). 7Medium has two goals that are pretty close distance-wise for the agent to work towards on a 10x10 grid ($10^2 = 100$ total states). Big2 has 5 goals that are sparsely distributed across a 45x45 grid ($45^2 = 2025$ total states). I believe that both of these mazes are **interesting** problems because they can effectively help elucidate both the strengths and weaknesses of these approaches to MDPs when applied to both slightly trivial and complex problems (plus watching the iterations is occur in real time one these grids is mesmerizing). The PJOG parameter determines the chance of lossy transmission of actions to the agent (e.g. it interprets a signal to go left as a signal to go right).

We will be analyzing the performance of a myriad of approaches on these grid problems: value iteration, policy iteration, and policy sweeping. The parameters of each algorithm are assumed to be the defaults in the RL_sim JAR unless otherwise stated. .

Data

Iterations Table:

	Value Iteration	Policy Iteration
7 Medium (PJOG 0.1)	42	6
7 Medium (PJOG 0.3)	84	7
7 Medium (PJOG 0.5)	181	5
Big Maze (PJOG 0.1)	66	22
Big Maze (PJOG 0.3)	154	16
Big Maze (PJOG 0.5)	464	12

Time Table (ms)

	Value Iteration	Policy Iteration
7 Medium (PJOG 0.1)	7	48
7 Medium (PJOG 0.3)	20	27
7 Medium (PJOG 0.5)	45	52
Big Maze (PJOG 0.1)	1198	8988
Big Maze (PJOG 0.3)	2701	11245
Big Maze (PJOG 0.5)	10424	12884

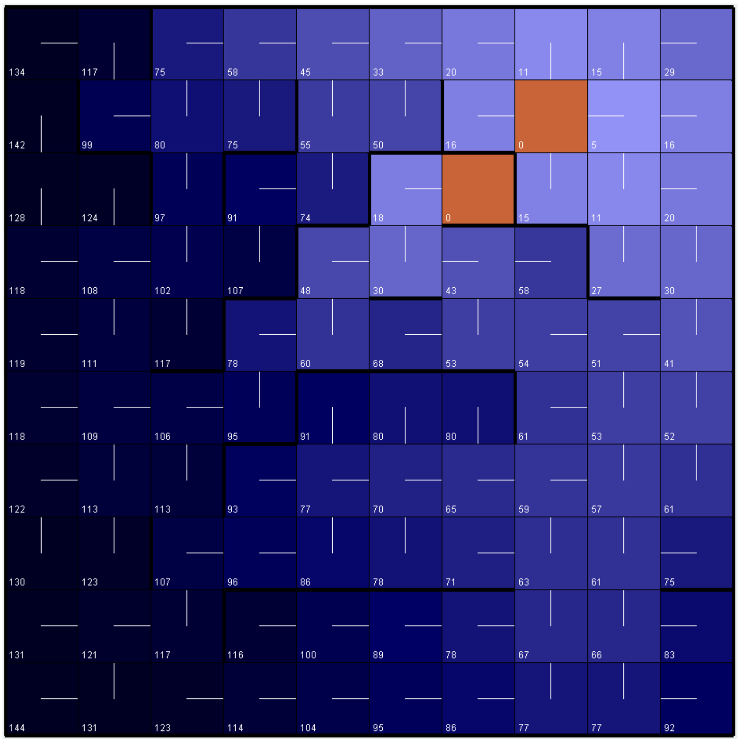
Analysis

Both value iteration and policy iteration employ the Bellman equation to compute the utilizes of states. Value Iteration initializes some random value function and then iteratively improves the value function until it finds the optimal value function. Policy Iteration is a modified version of Value Iteration that includes a policy evaluation and improvement step added to the end of the algorithm instead of outputting the argmax π(s) like in Value Iteration. As such, it makes sense that policy iteration results in non-negligible increases in run time for both the 7Medium and Big2 mazes. I found an interesting sweetspot in 7Medium when adjusting the PJOG parameter, however. An increase in PJOG from 0.1 to 0.3 applied to 7Medium results in a decrease in overall run time, but a paradoxical increase in the number of overall iterations in Policy Iteration.

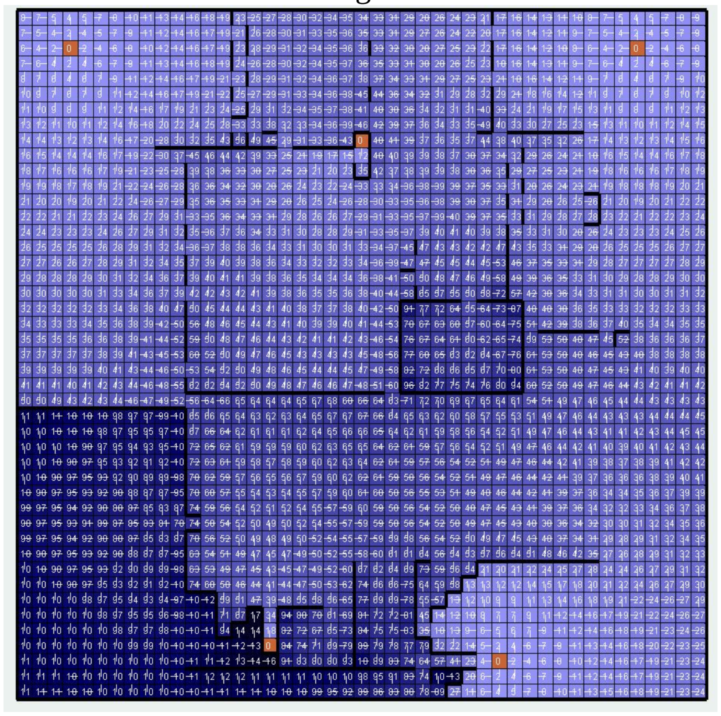
Increasing PJOG w.r.t Value Iteration on both mazes results in a substantial increase in the number of iterations needed for the algorithm to converge. However, with policy iteration (excluding that paradoxical sweet spot of the 0.3 PJOG applied to 7Medium) the number of iterations required for the algorithm to converge is reduced almost linearly. This is because the overall throughput of Policy Iteration algorithm increases due to the policy evaluation step as a result of an increase in the entropic nature of each action taken by the agent with a rise in PJOG.

Here’s what the converged mazes look like (policy iteration and value iteration both converged to the same answers):

7 Medium

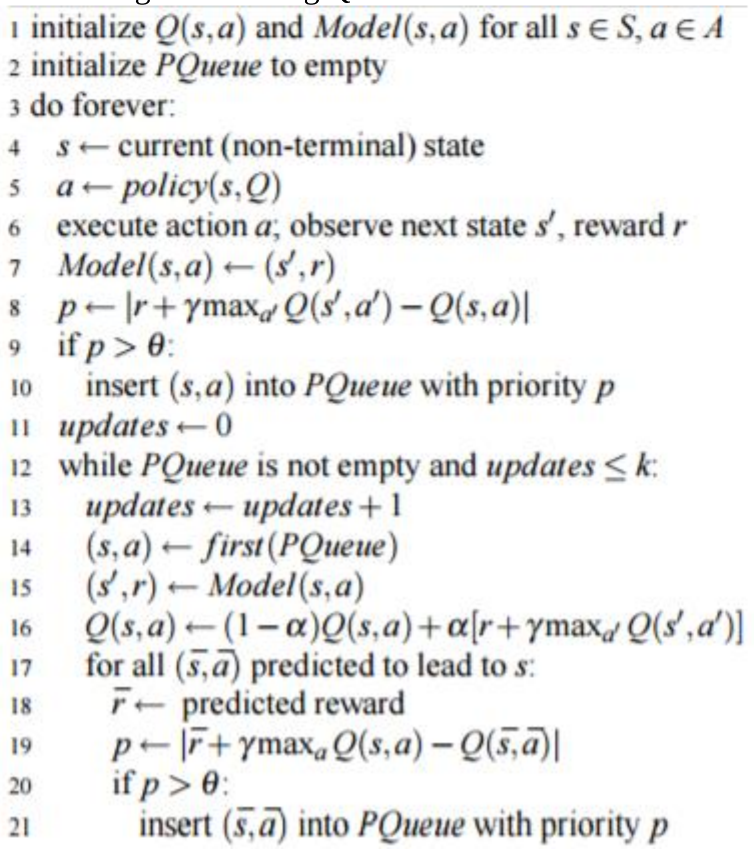


Big2:



Now let’s take a look at the performance of our RL algorithm (prioritized sweeping) on these mazes.

Prioritized sweeping is sort of like a corollary from Q learning. It works by computing multiple updates per timeslice of the algorithm, and prioritizing the order in which updates are completed by the magnitude of the change of resulting Q values. It looks sort of like this:

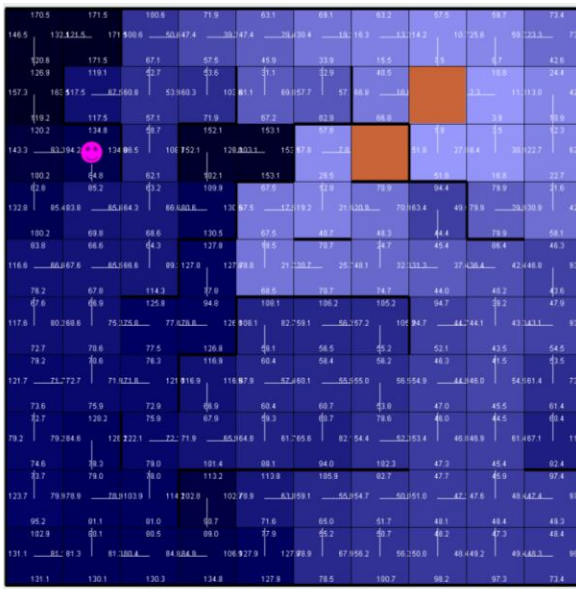


```
initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in S, a \in A$
initialize $PQueue$ to empty
do forever:
  $s \leftarrow$ current (non-terminal) state
  $a \leftarrow policy(s,Q)$
  execute action $a$; observe next state $s'$, reward $r$
  $Model(s,a) \leftarrow (s',r)$
  $p \leftarrow |r + \gamma \max_{a'} Q(s',a') - Q(s,a)|$
  if $p > \theta$:
    insert $(s,a)$ into $PQueue$ with priority $p$
  $updates \leftarrow 0$
  while $PQueue$ is not empty and $updates \leq k$:
    $updates \leftarrow updates + 1$
    $(s,a) \leftarrow first(PQueue)$
    $(s',r) \leftarrow Model(s,a)$
    $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$
    for all $(\bar{s},\bar{a})$ predicted to lead to $s$:
      $\bar{r} \leftarrow$ predicted reward
      $p \leftarrow |\bar{r} + \gamma \max_a Q(s,a) - Q(\bar{s},\bar{a})|$
      if $p > \theta$:
        insert $(\bar{s},\bar{a})$ into $PQueue$ with priority $p$
```

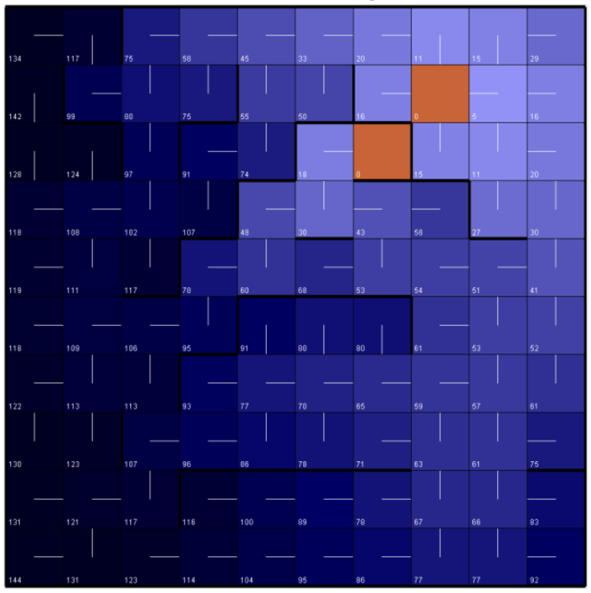
RL algorithms don’t necessarily converge like Policy Iteration and Value Iteration since they lack full observability of a problem’s model. As such, a sweet spot between exploration and exploitation must be found for each problem RL is applied to.

We will be qualitatively assessing how well Policy Sweeping does as the number of cycles the algorithm is permitted and the parameter epsilon increases by comparing the image of the grid when policy sweeping is completed to the image of the optimal policy side by side.

7 Medium Policy Sweeping (100 Cycles, Epsilon = 0.1)

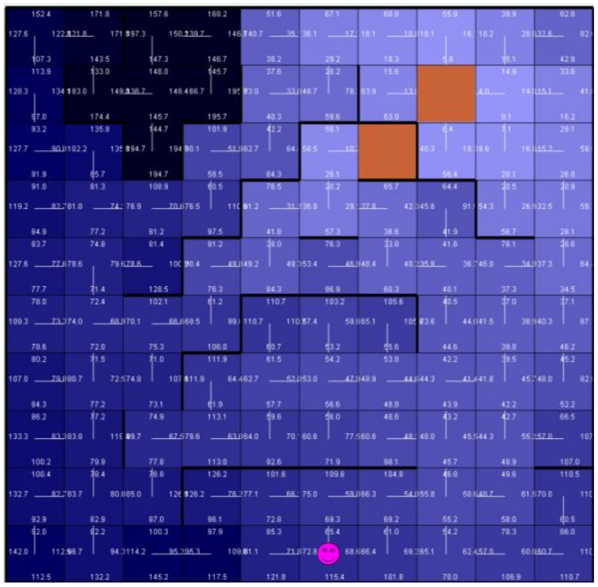


7 Medium (Converged)

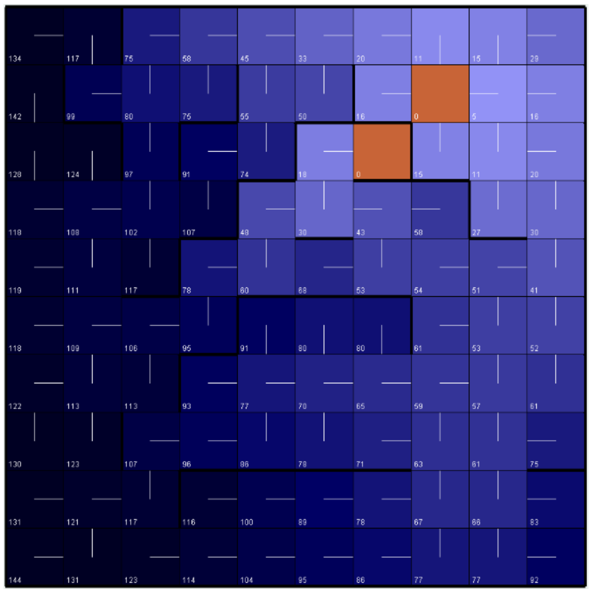


As you can see above, after 100 Cycles (which roughly tough around 7 minutes of computation time), Prioritized Sweeping looks very similar to the optimal solution we found using Value/Policy iteration near the goal states, but does not look similar around the edges of the grid.

7 Medium (1000 Cycles, Epsilon = 0.1)

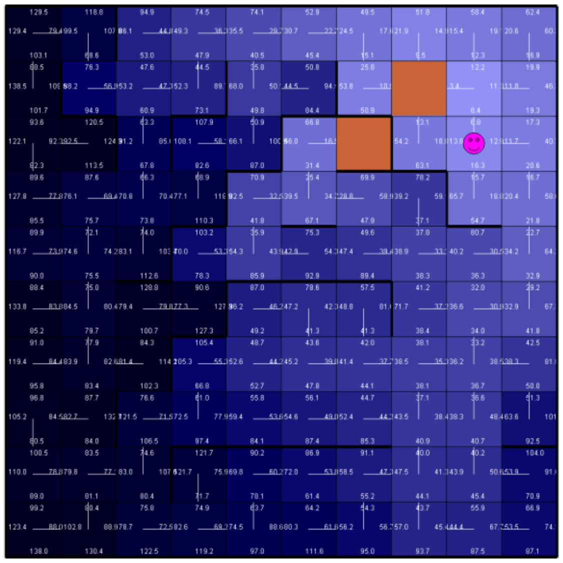


7 Medium (Converged)

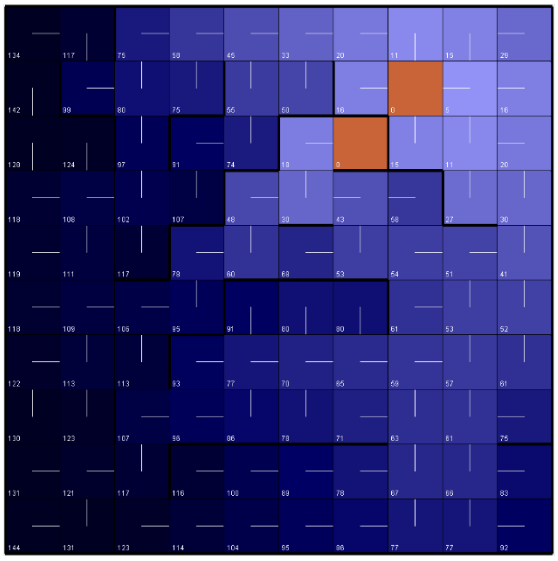


Increasing the number of cycles didn't really impact the accuracy of Priority Sweeping. It seems as though with a default Epsilon of 0.1 the algorithm tends to do well in very small sections of the grid, but it tends to ignore the rest of the state space. I cranked Epsilon up to 0.3 (to increase the algorithm's propensity to explore more of the state space) and checked out how policy sweeping did on 7 Medium once more.

7 Medium (1000~ Cycles, Epsilon = 0.3)



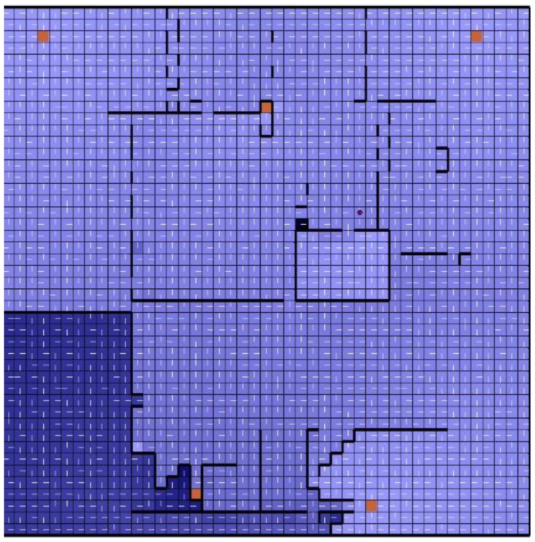
7 Medium (Converged)



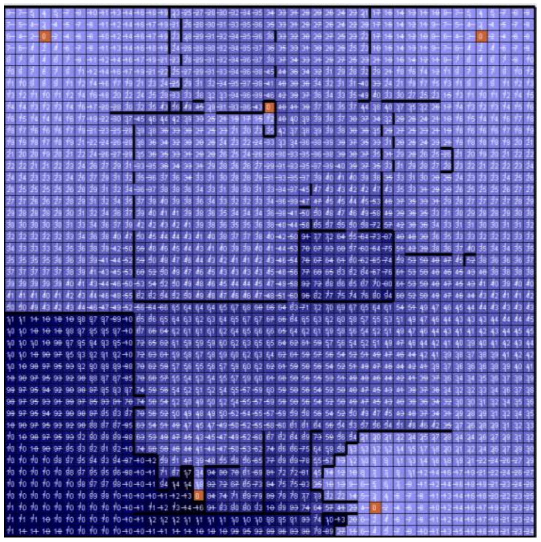
Not too shabby if I do say so myself! Increasing epsilon was great for this initially stubborn agent. Although at 1000 cycles this algorithm took over half an hour to complete.

Let's take a look at how Priority Sweeping does with a big, complex maze:

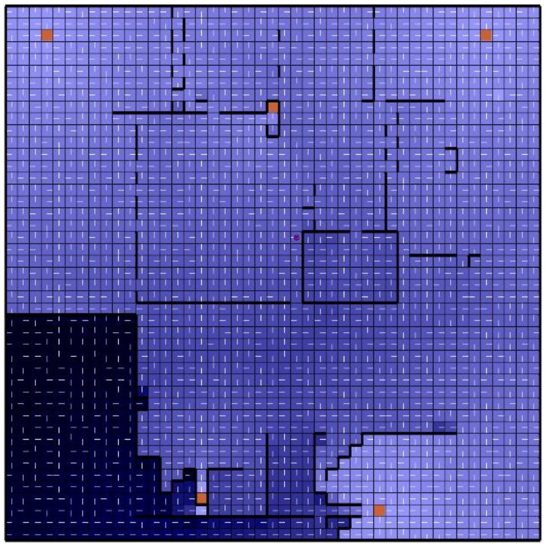
Big2 P Sweep (100 Cycles, Eps = 0.1)



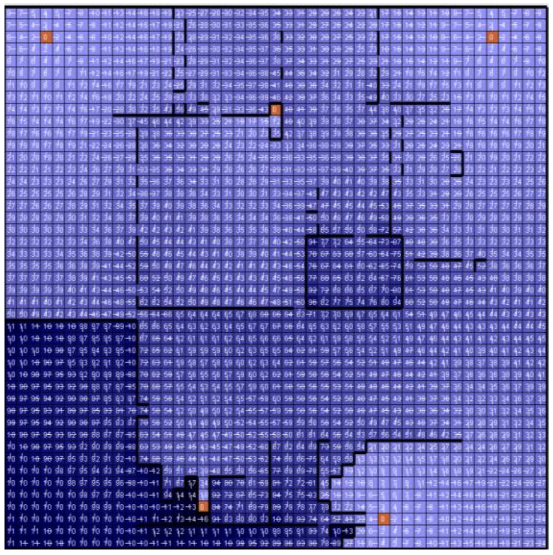
Big2 P Sweep (Converged)



Big2 P Sweep (100 Cycles, Eps = 0.3)



Big 2 P Sweep (Converged)



The sheer dimensionality of this grid compared to 7Medium means this algorithm runs forever. 100 Cycles took around half an hour to complete.

Once again, increasing the Epsilon parameter of this Reinforcement Learning algorithm allowed it to more accurately solve areas toward the center of the grid where the lower Epsilon failed to learn. As a wise weeaboo warrior Genji Shimada from Overwatch once stated, “waga tamashī wa kinkō o motomeru”. This means “my soul seeks balance”. RL algorithms must find balance between exploration and exploitation in order to effectively understand a problem. Too much of either doesn’t bode well for actually solving an issue.

References

- http://www.cs.cmu.edu/~awm/rlsim/
- https://github.com/james7132/GTCourseWork/tree/master/CS%204641%20-%20Machine%20Learning/Markov%20Decision%20Processes%20Assignment
- https://people.eecs.berkeley.edu/~pabbeel/cs287-fa12/slides/mdps-exact-methods.pdf
- https://www.quora.com/profile/Tad-zeMicheal
- http://stackoverflow.com/questions/37370015/difference-between-value-iteration-and-policy-iteration-reinforced-learning
- https://lenabayeva.files.wordpress.com/2015/03/aamasreport2011.pdf