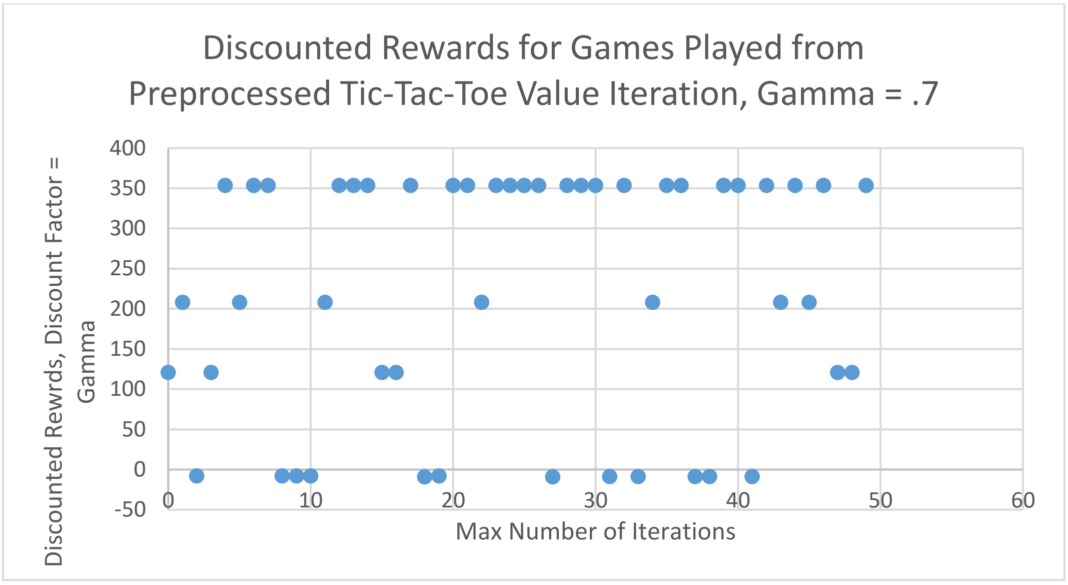
<u>Tic-Tac-Toe:</u> The game of Tic-Tac-Toe is a 3 by 3 grid where each player, represented as either an X or an O, attempts to get 3 of the same in a row, a column, or a diagonal. Such a game could potentially be represented by 3^9 states, but most of those states may be impossible, as either X or O would have won by then. Analysis from Burlap showed 5683 different states are possible for a single player, given that the game ends immediately when either a tie occurs or a win condition is met by either players, which means that Tic-Tac-Toe can be a significantly hard problem for a computer to learn from compared to other problems.

My use of Tic-Tac-Toe created a random opponent, such that the opponent's behavior was not as important to consider; this means that certain scenarios from a traditional game of Tic-Tac-Toe, such as an opponent always attempting to win, did not have to be considered, and that many of the states, such as when the random enemy player is about to win, did not necessarily result in new states. Future tests could attempt to use another agent which attempts to maximize its own value, but this type of random opponent allows for a system where an agent does not have to deal with implications of a zero-sum game, while at the same time attempting to deal with a state which changes over time. More specifically, the game is essentially 1 player, as the random opponent can simply be thought of as another portion of the environment. The game of Tic-Tac-Toe is interesting, as it provides key insights into strategies and forethought which keeps the most number of possibilities for winning available while still ensuring that an opponent is blocked off from a win. Even when the opponent is a random agent, there is always a risk that the opponent will stumble to a correct strategy.

The reward function was set so that any state that was not a finished game was -4, any game that was a finished game with the opponent winning was a -1000, a finished game state with the agent winning was rewarded 1000, and a finished game state with no winners was awarded a 0. To simplify processing, all wins by the agent were considered the same state, all wins by the opponent were considered the same state, and all "catscratches" (ties) were considered the same state. The random opponent is 'O' and therefore always second, while the agent is 'X' and is therefore always first.

<u>Value Iteration</u>

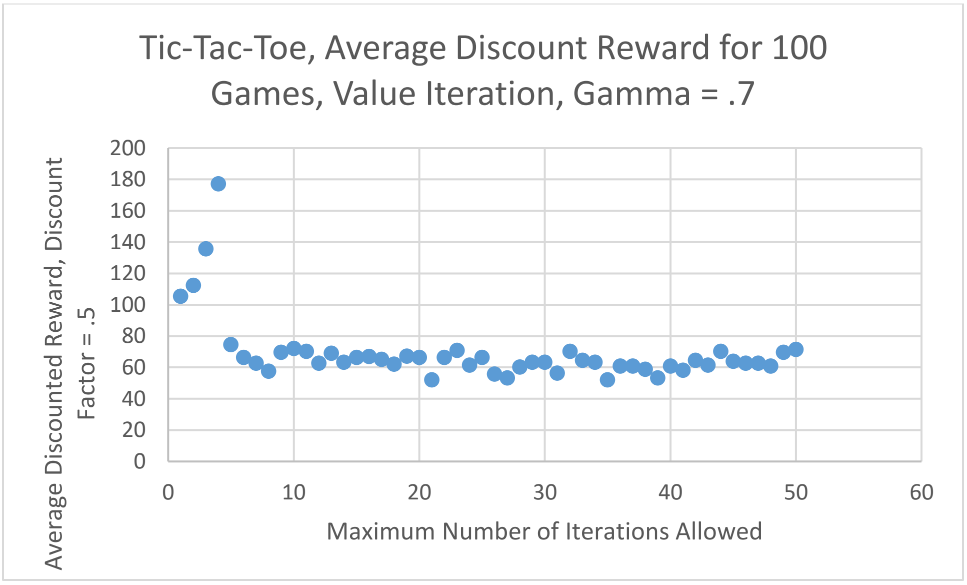
Analysis with value iteration showed that with a 3 by 3 grid for Tic-Tac-Toe, the most common version of Tic-Tac-Toe, the game has approximately 5,500 states. Key results showed that Tic-Tac-Toe was a fundamentally significant problem, because, even with so many states, a 3 by 3 grid allows for a random opponent to not just occasionally but often stumble onto a proper sequence of actions to prevent a win for the agent.



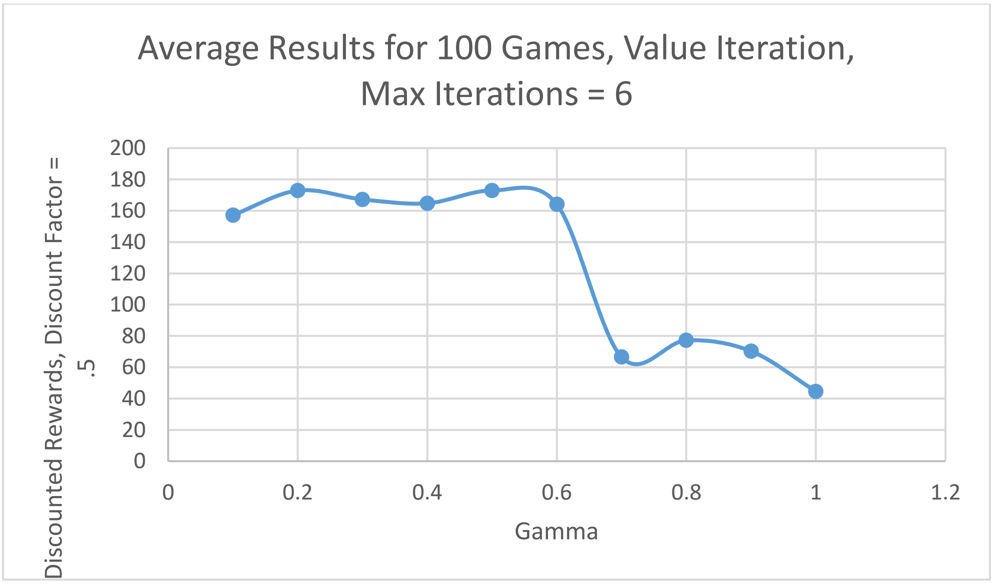
I attempted to use the max number of iterations for value iteration in order to control how the system would converge. However, there is no pattern throughout; this suggests that the system converges quickly, and that max number of iterations does not control how quickly the system finds an answer. Because the value iteration algorithm was set to end when iterations changed only by .01 and as the reward function was in the range of thousands, Tic-Tac-Toe may be simpler than thought.

Interestingly, the agent does not always win, or even win quickly. While the most instances of the agent are above 100, occasionally, the RNG opponent can occasionally delay the inevitable. It was an initial worry that the -4 reward of each non-terminating state would be high enough to drive the agent towards a tie, but a tie only occurs 12 times out of 50, while winning sequences are found the other 38 times. It may be the case that the opponent occasionally stumbles to a tied strategy. As the max number of iterations allowed increases, the amount of ties decrease, though it is hard to see if this is clearly due to the algorithm, or simply because the random opponent did not find the correct strategy during this time.

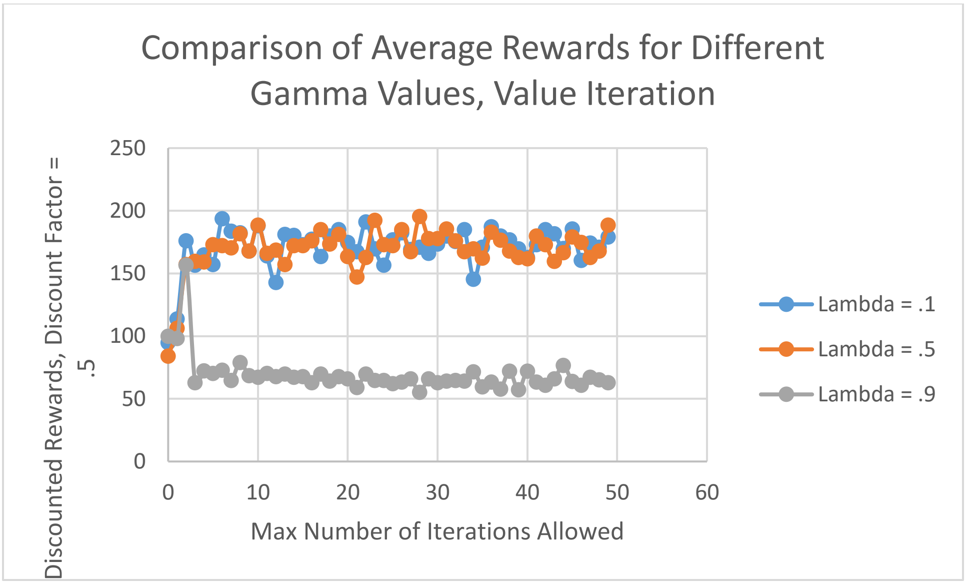
To attempt to control for randomness, the same value iteration followed by testing was used, but the policy generated was followed for 100 times. The original data was discounted according to the gamma for that specific value iteration instance, but, in order to compare discounted rewards from different gammas together, .5 was chosen as the actual discount factor, in order to standardize the discounted rewards.



Analysis for the average discounted reward shows a convergence to a policy after only 5 iterations. Interestingly, it appears that the policy is significantly better for when the policy is forced to converge after 5 iterations. This may be due to the agent taking increased risks in order to skew its discounted rewards to the quickest games after only a few moves, even if the games allow the random opponent to occasionally interrupt and tie or even win a game. For value iteration, this would make sense, as 5 iterations may not be enough for the rewards from the terminal states to have propagated through all the states; it may be the case that only the most superficially quick paths to victory in Tic-Tac-Toe are followed.



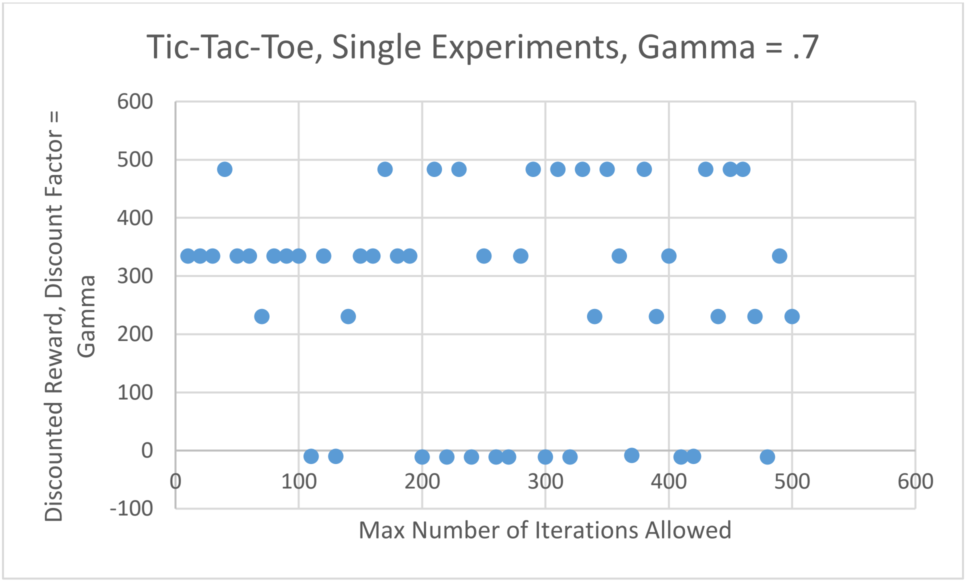
Results suggest that the reward for high gamma converge quickly to the safe, drawn out games which leads to the lower discounted rewards, while lower gamma value iteration policies cannot converge after only 6 iterations. In addition, the higher gammas make a weight such that terminal state rewards can be delayed for longer, while lower gammas severely pressure an agent to quickly find an end goal before the reward has been discounted so much as to be entirely useless.



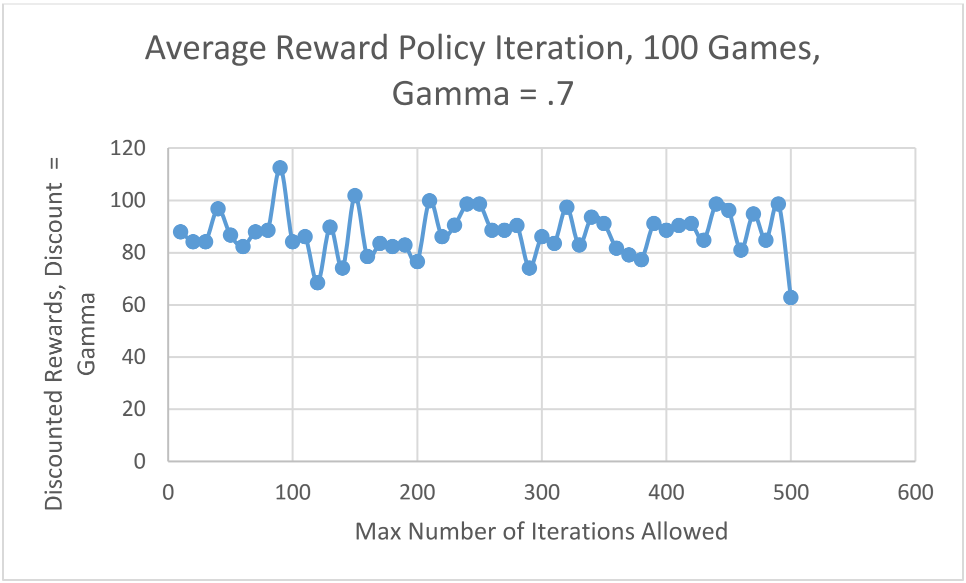
Interestingly, the threshold for weighing solutions solved in a short amount of time versus solutions which are drawn out but more likely to avoid ties and accidental opponent wins appears to be somewhat high in this case.

<u>Policy Iteration</u>

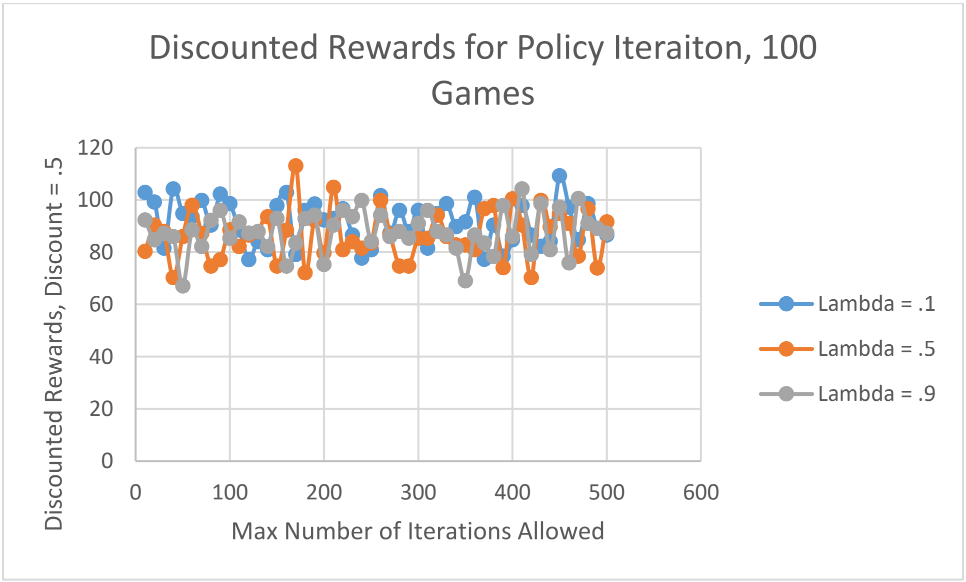
Single experiment results with policy iteration with Tic-Tac-Toe are similar to the single experiment results with value iteration, in that the random opponent clearly affects how the results work out.



Even at high number of iterations, the games remain extremely variable. Not much information can be gleamed from here, though it is interesting to note that at lower iterations, cat-scratches are much less likely, though it may simply be due to random chance.

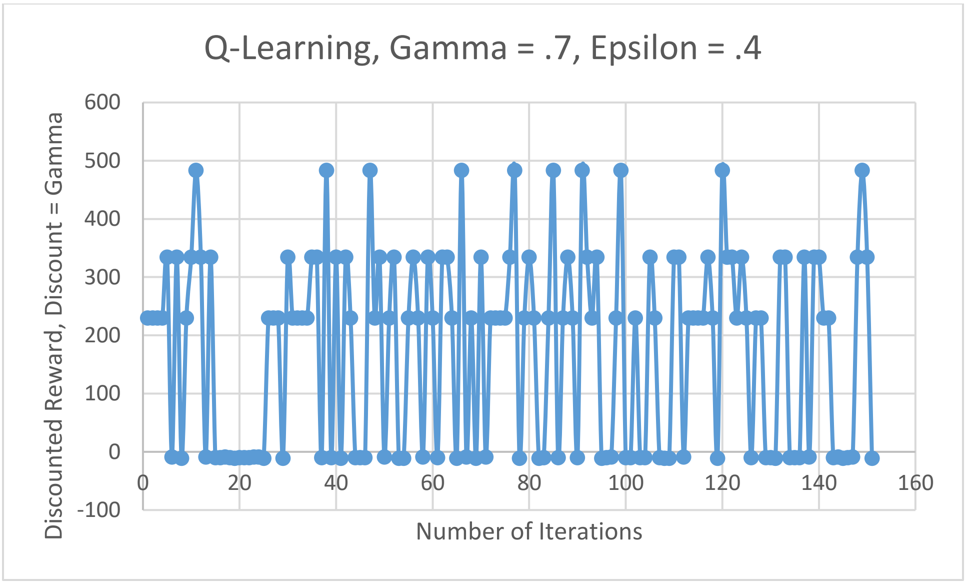


Comparison to value iteration shows that the policies, for moderately high gamma values, the gagen converge relatively quickly to the answer which allows for lowest risk wins.



Taken over a wide range of gamma values, in fact, the agent appears to never choose a high-risk strategy which allows for quicker games at the expense of higher chances of ties. In comparison to value iteration, policy iteration may simply be able to converge more quickly; it may even be the case that for low gamma instances of value iteration discussed earlier in the paper that the games may converge but that 50 iterations were not nearly enough.

Q-Learning



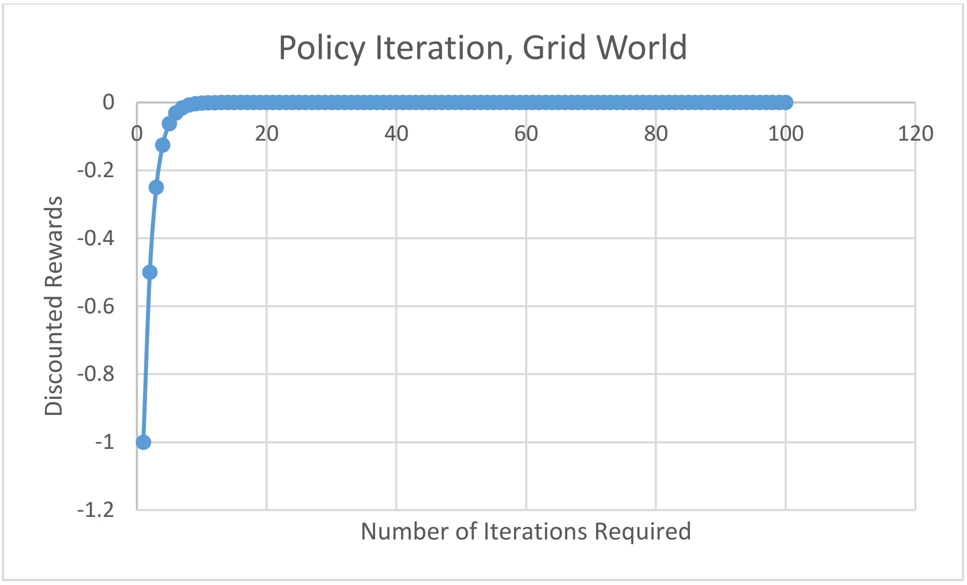
Analysis of Q-learning suggests, similar to policy iteration and value iteration, that the highly random nature of the opponent can allow for significant amounts of change. Interestingly, Q-Learning did not have a single instance of the opponent winning, which is extremely interesting. This may be because once the agent found the single winning state, it would attempt to return to it over time.

Grid World: The grid world domain is an 11 by 11 world where a player must navigate towards a terminating state with a high reward, despite partially controllable actions which cause the player to occasionally move in an incorrect manner. The grid world problem is an abstraction used frequently to explain MDP problems; however, grid world is surprisingly similar to many real world activities, such as self-driving cars and other robots which must correct themselves based on stochastic movements.

More specifically, Grid World is formulated so that an agent can be in one of 4 rooms; in addition, each room has small entrances to adjacent rooms.

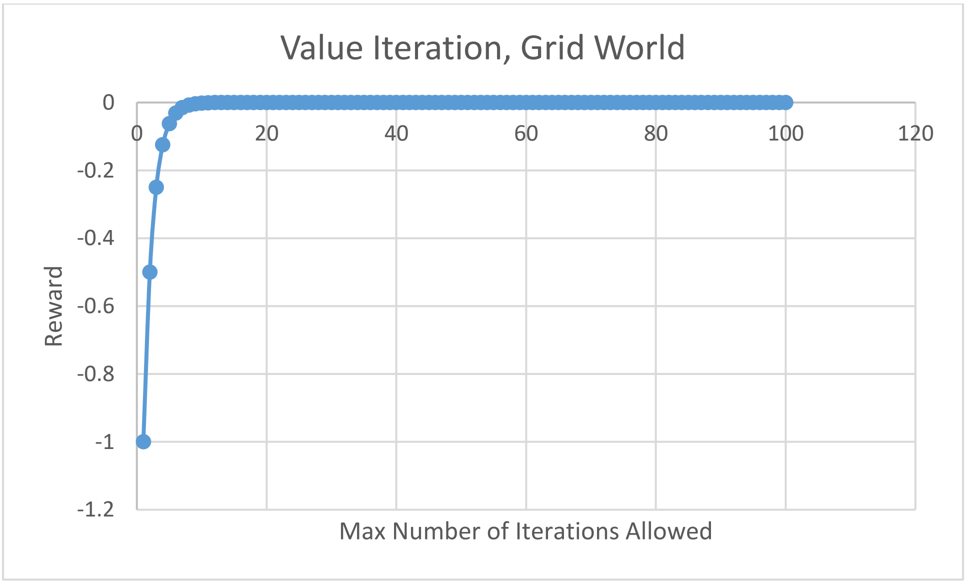
Grid world provides only 104 different states and while the Tic-Tac-Toe example beforehand did not include any real opponent, only a random opponent, grid world has only the agent itself attempting navigate. Even though the Tic-Tac-Toe opponent was random, the grid space of 3 by 3 was so small that occasionally the opponent would choose its most optimal set of actions, forcing the agent to consider that. In contrast, Grid World is truly a single agent domain to consider and significantly easier to consider.

Policy Iteration



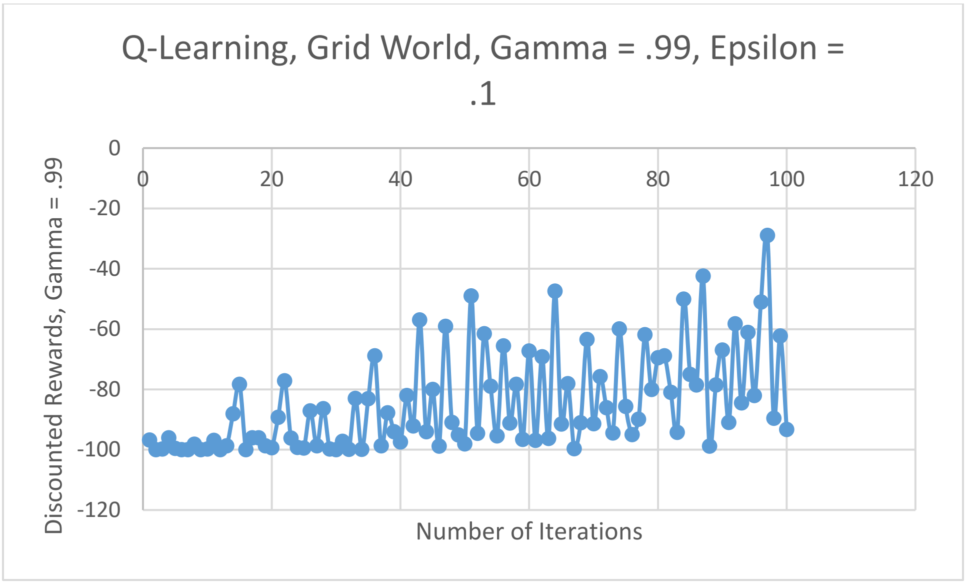
Policy iteration discounted rewards for Grid World is vastly different than the policy iteration discounted rewards seen for Tic-Tac-Toe. It appeared that in the Tic-Tac-Toe MDP, the correct policy was stumbled onto after only a few iterations, while, for Grid World, the number of actions to take can result in significantly higher chances for failure.

Value Iteration

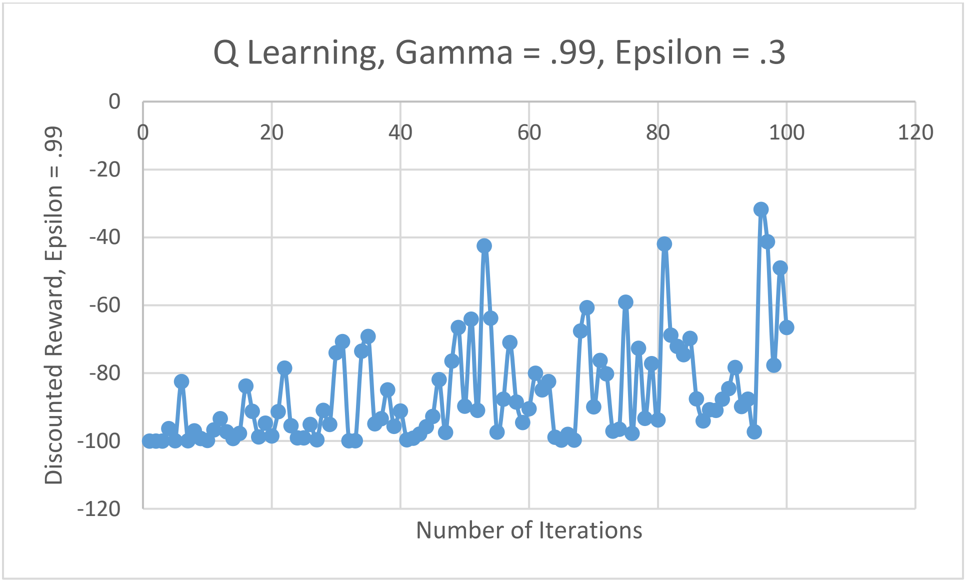


Similar to policy iteration, Grid World value iteration shows a quick convergence to the optimal solution. The small domain space of Grid World must allow any algorithm that is given the transition model to quickly iterate to the correct solution.

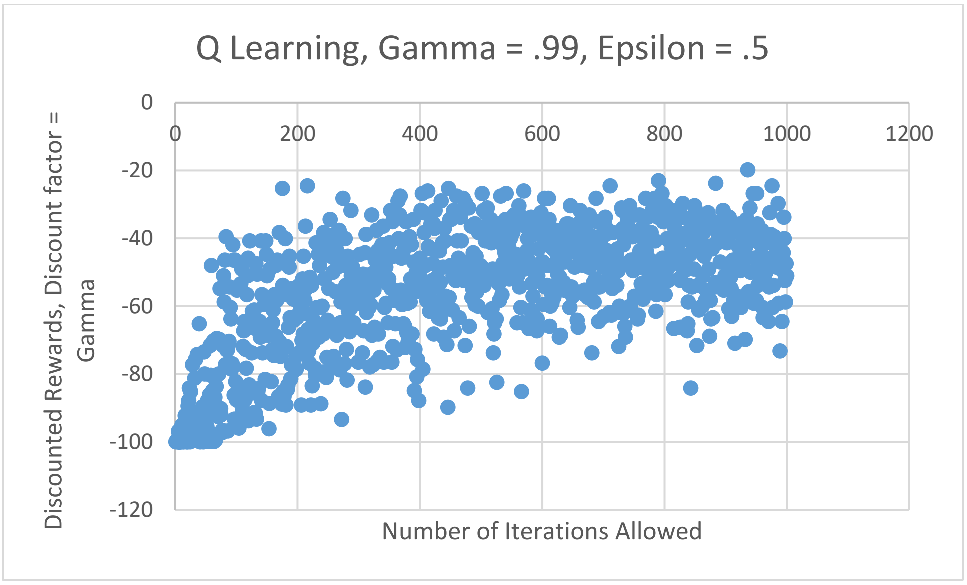
Q Learning



Q-learning for Grid World is significant worse performing than either value iteration and policy iteration, as expected. The discounted rewards vary significantly, especially as the number of iterations given to compute the solution increases. Unlike Tic-Tac-Toe, the agent cannot simply stumble to an optimal solution within a few moves. In addition, because the epsilon is extremely low, it appears that the system must take more iterations to converge.



Allowing the epsilon to be increased to .3 allows for better results but significantly more variability as well, due to the exploration of the agent. Further testing with a higher epsilon and more iterations were then done.



Unfortunately, even with 1000 iterations and significantly higher epsilon values, there does not appear to be a significant increase in rewards as compared to value iteration or policy iteration. What may be happening now is that the epsilon value may have been tuned up higher than optimal, resulting in unnecessary exploration.

Overall, the results' variability increased as epsilon increased, up to a certain point.

Conclusion:

The Grid World and Tic-Tac-Toe MDPS were significantly different. While Tic-Tac-Toe had significantly more states than Grid World, the random opponent could eventually be controlled, as picking states which limited the probability of the random opponent stumbling on a way to either win or cat-scratch(tie) the game could allow for rather complex policies to form. Specifically, the agent could be shown to find either one of two states, one which valued a long-term reward which minimized the chance of ties, or a short term reward which could allow the opponent to occasionally randomly stumble on a way to cat-scratch. The grid world problem appeared to be significantly easier to iterate over, in comparison, when given the transition model, as both policy iteration and value iteration converged to the correct answer in a relatively short amount of time.

A key point of observation is that QLearning for the Grid World problem was significantly worse than the QLearnign problem for Tic Tac Toe. Fundamentally, the agent is able to control how its world is able to progress in a game of Tic Tac Toe, as it has the first move advantage, leaving the random opponent to choose among the spots on the board left. More importantly, it is clear that the Tic Tac Toe game is symmetric, and so many states are essentially functionally equivalent. Any game of Tic Tac Toe will last for at most only 5 states, as a new state is generated every odd move for the entire game, due the agent going first. As a result of these factors, the agent only needs to find a set of moves which optimize along a few paths; it does not need to consider every path, as winning and conditions which lead to winning are symmetric. QLearning works so well here because these factors allow the agent to quickly stumble onto and improve on a winning strategy.

In contrast, the Grid World problem agent is significantly worse than the Tic Tac Toe agent. While there is some symmetry in its environment (the four rooms divide an 11 by 11 environment into 4 squares), the goal state is in only one room, meaning that different policies to the same state may not actually be functionally equivalent. As a result, the QLearner fails to fundamentally describe the situation unless if there is a significantly high epsilon value and high number of iterations given.

Further areas of research could include attempting to create a larger Tic Tac Toe game, attempting to convert the Tic Tac Toe game from a random opponent to a reasoning agent, and creating an Ultimate Tic Tac Toe version of the game, which forces players to attempt to win a large 3 by 3 game of Tic Tac Toe by winning games of Tic Tac Toe in each cell first. Formulations of the original problem could also be changed, as currently, a cat-scratch terminal state provides 0 reward; it may be the case that further decreasing the reward received in this state could lead to different results.

For Grid World, more goal states and a negative reward terminal state could be added. In addition, further testing with QLearning could have been used to try to find if a good model similar to the models found by policy iteration and value iteration could have been found. Changing the rewards of intermediate states to vary instead of being uniform could also have helped.