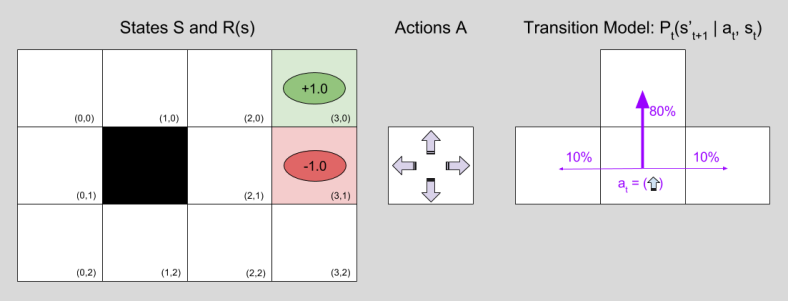
Assignment 4: Markov Decision Processes (MDP)  
CS 7641  
Connor Beveridge – cbeveridge3 – 902451314

**Introduction**

In order to analyze comparisons between Value Iteration (VI), Policy Iteration (PI), and Q-learning (QL), these algorithms are run on two MDP problems. The first of these has a small number of states and is a so-called grid-world problem. The second of these has many states and is a classic-control problem.

**Canonical Grid World (Redesigned as Mountain Climbing Problem)**

For the first MDP problem to analyze (which is the small number of states one), I have chosen the Canonical Grid World used to teach most students about value and policy iteration. This is interesting to me in this case because I have expanded it to a much greater size than the 3x4 world always used as shown in **Figure 1** as I am sure the reader is familiar. It is also is interesting because I have always enjoyed solving mazes and using search algorithms to do so, and so in this case, I would like to see how reinforcement algorithms do on them. This problem is not quite a maze problem but is actually a more interesting version as it has multiple goals, pitfalls, and walls which can be run into as there is a transition matrix. In this analysis in which I

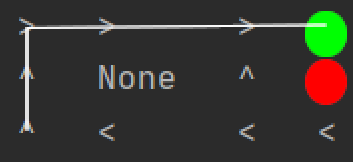
  
**Figure 1 | Canonical Grid World Model**

have expanded the this model into a 6x8 version and a 12x16 version. I consider this redesign a mountain climbing problem because I can imagine the rewards and punishments that I have designed model the terminal rewards of finding and very visually pleasing place to camp higher up the mountain, the punishments for falling into dangerous pitfalls along step mountain cliffs, and the smaller pains of moving from grid to grid. These movement pains grow more as one approaches the top of the mountain as the slope gets steeper and terrain gets rougher, highest mountain peak is near the at the top right of the grid/map for all 3 grid sizes. The code for VI and PI is taken from the code repository for Russel and Norvig’s book, *Artificial Intelligence: A Modern Approach.*

Convergence is defined for VI in the standard way. When for every full iteration over the values of all the states, if the maximum change in any value from the previous iteration is less than .001 \* (1 – discount)/discount, it is considered to have converged. For PI, if for a full iteration of policy evaluation, no policies are changed, it is considered to have converged. QL convergence is defined similar to VI. If after 25 consecutive runs, the values of all states have not changed greater than .001 from the previous run, it is considered converged. Value here is the best q-value for each state. For Q-learning, the learner transitions between states until a terminal is reached on each ‘episode’, and the starting location for each episode is random.

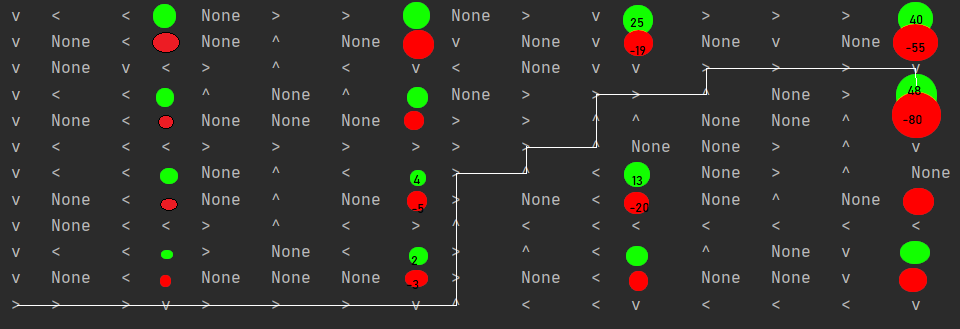
1. **Initial Policy Results on Tiny Grid world**

To get an initial feel for what these algorithms (VI, PI, and QL) are doing, I first run them on the smallest and largest grid. The policy results for the smallest grid can be seen in **figure 2**. In this initial run, all three algorithms have a discount factor of 1. QL additionally has a learning rate and exploration rate decay of .012 (1/85) and these rates start at 1 and decay to .1 depending on this decay factor. All 3 algorithms produced this same policy map in which the rewards are +1 and -1 for the terminal states colored and -0.04 for the step cost. The optimal policy from the starting position at the origin is shown with a white line. The fact that all the algorithms produced the same map suggests it is indeed the optimal policy.

  
**Figure 2 | Policy Results on Smallest grid for all 3 Algorithms.** The ‘None’ state is a wall. Colored circles are terminal states with positive and negative reward of 1.

1. **Discount Factor Analysis**

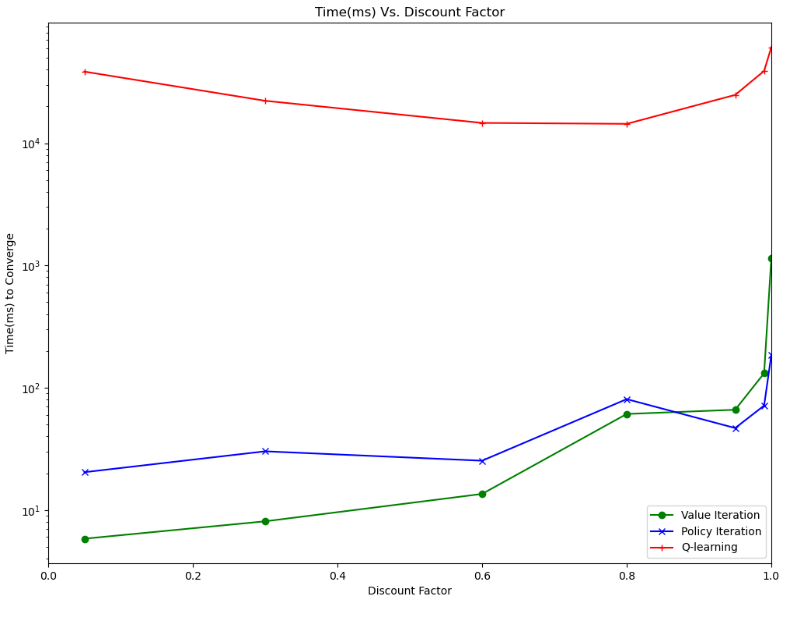
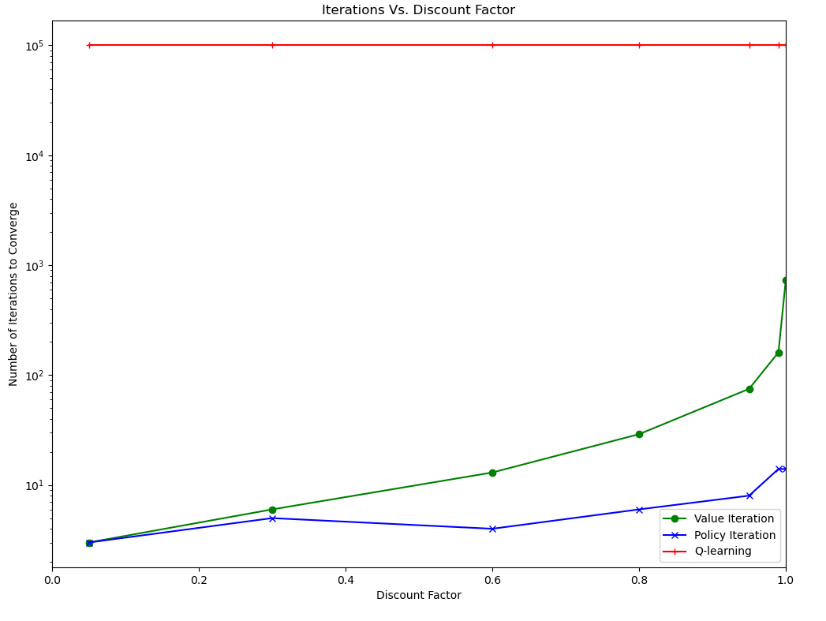
The large grid world of 192 states (counting non-transversable ‘None’ wall states) is analyzed with respect to the discount factor. **Figure 3** shows the results of VI and PI algorithms on the large grid for a discount factor of 1. QL produced an identical optimal policy along the line shown but had a few differencing policies in other places in the grid. You can see that is found a good path to reach the best reward of +48 from the origin in the bottom left, even with all of the other rewards available. Several off the policies even move directly away from positive terminal states in order to try and reach the +48.



**Figure 3 | Policy results for VI. Some reward values are shown. Summarized Optimal policy is shown.**

It moves against walls and in places where it may have to come back to where it is to minimize the probability of falling into a terminal state on accident.

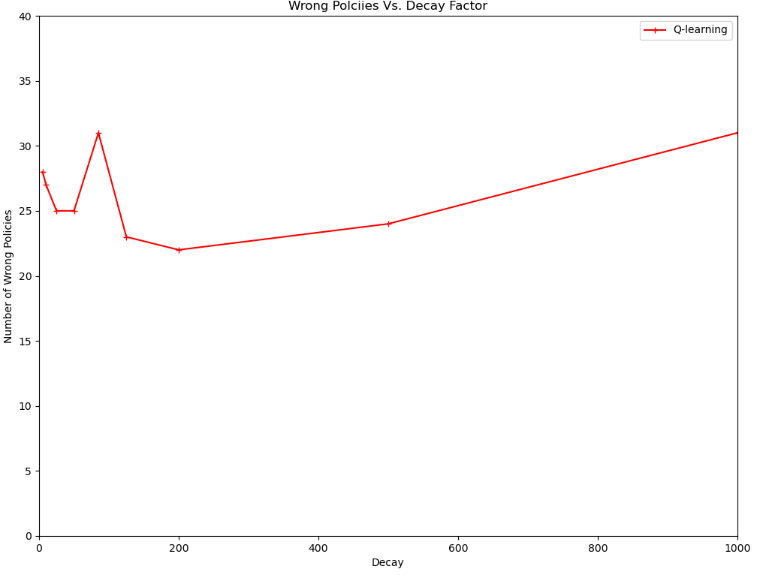
Figure 4 shows convergence of the 4 algorithms with varying discount factors with respect to time and number of iterations on a logarithmic scale. Iteration for q-learning are the number of episodes. You can see that Q-learning takes a significant longer time to converge with the convergence method chosen. This is because it is better suited for when the transition and rewards matrices are not known, and it in a way must learn them itself. PI seems to do better over VI time-wise when a large discount factor is used. This may be because the convergence criteria becomes perfectly strict when the discount factor is 1.0 for VI.

**Figure 4 | Iterations and Time (ms) vs. discount factor.**

1. **Decay Analysis**

A discount factor of .9 is chosen to explore how varying decay rates affect the QL learning. In reference to the problem, it can be imagined that the mountain climber never knows what might go wrong and he dies on the mountain. The decay rate determines how slowly the algorithm will go from full exploration and learning to a minimum of 10% exploration and learning potential. Decay rates from .05 to .001 are analyzed. A lower decay rate means you will explore more. For each decay size, the QL algorithm is run on the large grid, and converge requires that the exploration and learning rates have reached their min of 0.1 in addition to the convergence criteria in the previous section. Given an iteration limit of 250,000 iterations, convergence still did not occur. You can see in **figure 5** that decay rate didn’t directly predict how close the policies came to the optimal policy given by value iteration. This may because some of the states aren’t being reached as often as they need to be. However, the policies given are still not bad, and would suffice if we didn’t have the reward and transition matrices needed for value iteration.

This further strengthens the conclusion that Q-learning is not at all worth using when the transition and reward matrices are known, but has shown itself to provide solutions that are ‘good enough’ without too much computational time when they are not known.

**Figure 5 | Decay rates and Policy Results.** Number of diffe****rent state policies compared to VI are shown. Decay rate is 1/Decay shown. So 200 = 0.05. Higher decay as quantized in the graph have slower decay of exploration and learning rates.

**Classic Control Problem, Cartpole: inverted pendulum problem**

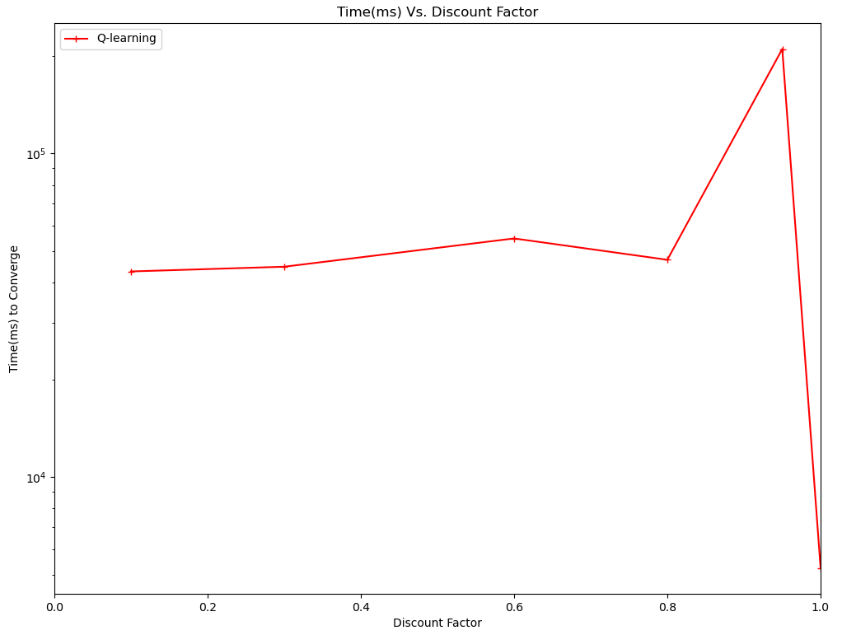
This mdp problem has a large number of states which can be exponentially higher and still provide more information. This is because it as a continuous space problem which is transformed into a discrete space problem so that it can be handled by these algorithms.

This problem is interesting as compared to the previous problem because it is much more applicable to real life robotics, and is the first robotic-like, non-grid world mdp I have analyzed. This problem is characterized as an inverted pendulum. There exists a cart with an attached pole as you can see in **Figure 9**. The goal is to move to the left or right at each time step so that the pole remains balanced upright for at least 200 timesteps while staying within the boundaries. Failure is characterized by the robot exiting its confined space or the angle of the pole growing greater than 50 degrees or moving too fast. A +1 or -1 reward is given at each timestep given simply based on whether the agent has / has not yet dropped the pole. Due to the nature of this problem in this case, the robot does a perfect job just keeping it not fallen over for 200 timesteps, so full Q-table converge is not necessary. Instead, the algorithm is considered converged when it can keep the pole balanced for the full 200 timesteps for 50 episodes in a row, so 10,000 consecutive times steps of successful balancing. OpenAI gym is used to acquire the problem environment, however the Q learning algorithm is within the agent provided.

This problem in the real world has an infinite amount of states. In order for this problem to be suitable for q-learning, it must be discretized into ‘buckets’, or value ranges for each dimension so that we can have a finite number of states. There are four dimensions to this problem robot position and velocity, and pole angle and velocity. The robot position doesn’t ultimately matter in deciding how to keep the pole upright so only one bucket is needed for that dimension. To start, for cart velocity, 6 dimensions are used. 12 are used for pole angle and 24 are used for pole angular velocity. This comes to 1728 states.

The minimum learning rate which is decayed to is .1. The minimum exploration rate is however .05. This is lowered to allow the agent to converge as it is learning with live episodes while not checking for Q-table convergence but instead checking for continued success in balancing the pole. Not only does it not have to run additional tests to check for accuracy in the real world, but the .05 exploration rate simulates action noise and state noise that may occur in the real world when the agent is being used to balance the pole. This could for instance be vibrations on the agents surface.

Unfortunately, I was not able to get VI or PI to work with this problem because I had trouble getting the transition model, but I have attempted to make up for this with extra analysis on QL.

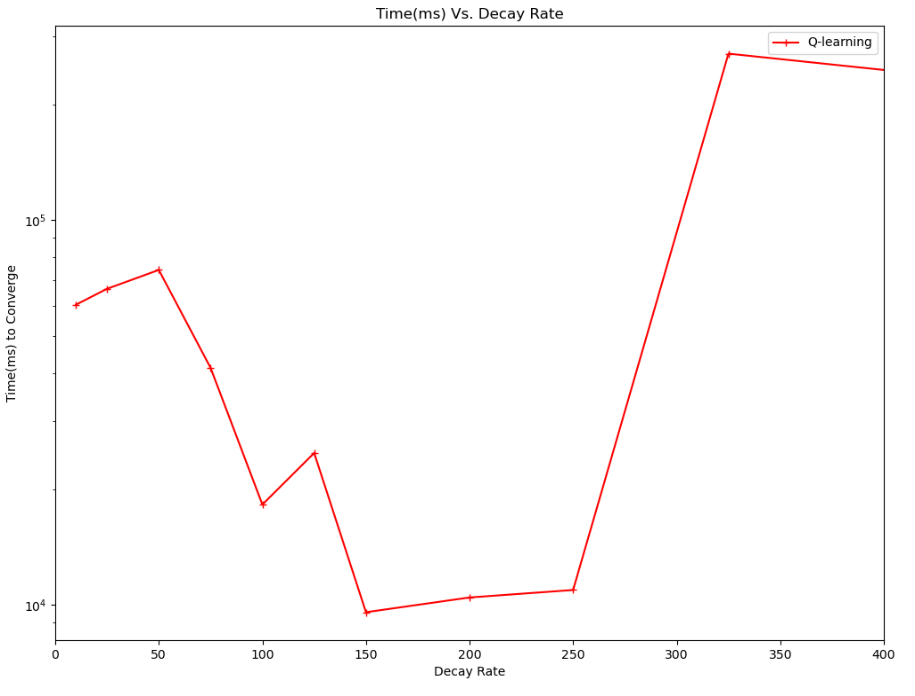
1. **Discount Rate**

**FIGURE 6** | Discount Factor Vs. Training time.

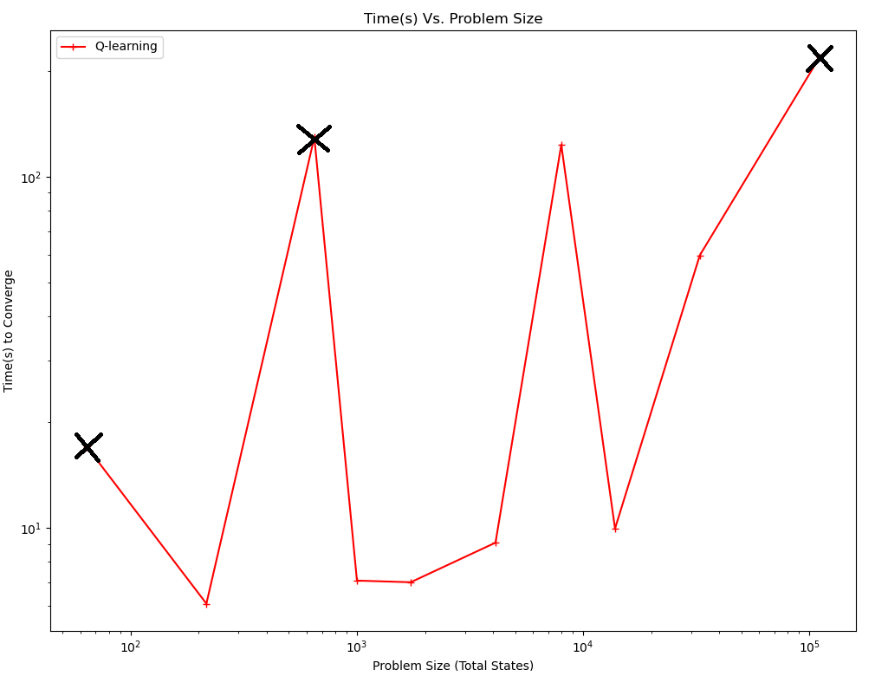
This problem is first analyzed with respect to the discount rate. The same decay rate and bucket sizes are used as previously mentioned. Figure 6 shows the training time needed for varying discount factors. Training time needed appeared to increase strongly with increasing discount size and then drop incredibly when the discount factor is 1. This may indicate that the algorithm is getting more information from states many time steps in the future if a certain action leads to stable states reoccurring. It seems that the discount rate of .95 may have taken so long to converge because that rate is enough to create big changes in the Q-table but less than one so it doesn’t have as stable an optimal mapping having every state only look so far into the future, compared to as when the discount is 1 an looks all the way into the future. This is likely why the discount factor of one was so much faster because it did use all the subsequent (time) states. A discount factor of one will be used for the next analysis.

1. **Decay Rate**

As mentioned, the decay controls the rate of decay for the learning (1.0 - 0.1) and exploration rates (1.0 - 0.01). Decay rates of 5 – 5000 will be analyzed. A discount factor of one is used and again convergence is tested during training and determined when the agent succeeds in balancing the pole for 200 timesteps for 50 episodes in a row. Figure 7 shows a general U-shape result with both low and high decay rates taking more time. Once the decay exceeded 250 (goes below .004), exploration occurred to much and convergence did not occur within 30,000 episodes. When decay was under 75 (rate above .013), exploration and learning were reduced to soon, leading to longer times. This indicates that when exploration and learning are reduced

**Figure 7 | Decay Rate and Learning time with Q-learning**  
  
early and quickly in learning, a poorer policy is converged upon which takes the agent further time to learn its way out of with a now low exploration and learning rate. It appears that when the exploration and learning rates stay high for longer and decays slower, the agent heavily explores states that shouldn’t be commonly approached in the first place such as when the pole and cart are moving very fast in opposite directions. The agent explores these undesirable states when it should focus on more likely states that will help it relative to the starting state. It also sometimes develops policies that lead to these undesirable states because it selects them out of exploration. For the next analysis, a decay rate of 150 is chosen.

1. Problem Size

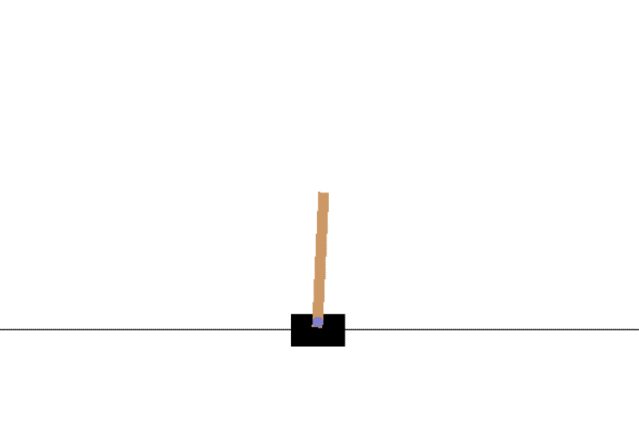
****In these previous analyses, their exists 1728 states. The cart’s speed and the pole’s position and angular velocity are taken from the continuous world and placed into 6, 12, and 24 bucket’s respectively. In this analysis using optimal hyperparameters from the previous sections, this bucketing is increased and decreased by the same ratio for each bucket. These state space sizes used are thus (2x4x8=64, 3x6x12=216, 4x8x16=648, 5x10x20=1000, 6x12x24=1728, 8x16x32=4096, 10x20x40=8000, 12x24x48=13824, 16x32x64=32768, 24x48x96=110592)

**Figure 8 | Problem Size Vs. Time Analysis**

**In Figure 8** you can see the results of running QL with the different bucket sizes. Note that the smallest and largest instances with X’s on them did not produce a successful pole balancer within 30,000 episodes. 64 states unsurprisingly were not enough to afford creating proper policies for a continuous environment, it simply did not have the expressiveness required, and I doubt it would have succeeded if given more time. Surprisingly, 216 states were enough to successfully model the act of balancing for this purpose, and it allowed for the quickest learning. For similar real-world applications, I would fear that 216 would not be expressive enough, but it makes sense why it converged so fast, because it had much fewer states to search. 648 states also did not find successful policy solution, thus confirming that although 216 worked out good, this range is still too risky overall due to likely lack of expressiveness. For the rest of the problem sizes up to 32,000 states the training time varied by about a factor of 10. It is not clear if this variation is due to the way the continuous values are being bucketed, or if it is just due to random chance on how the state space is being initially explored. This uncertainty is an important topic for future investigation.

Up until the 32,000 number, the benefits of expressiveness provided by a larger state space countered the increased time to search this space. Past this approximate 32,000 mark, the increasing number of states did not help choose better policies overall and required much more processing time to explore and fill a Q-table of such size. Based on this analysis, I would in the future choose around 1000-4000 states for similar problems to balance discretion expressiveness needs. Too much expressiveness results in needless processing time wasted.

1. **Optimal Learner: Simulated Policies**

The optimal (in the sense of ‘good enough’ learning speed) learner given these analyses with a discount factor of 1, decay of 150, and bucket size of (1,6,12,24) is learned and its policy map is tested on a simulated environment (continuous). While this final policy was learned very quickly and it did a marvelous job of balancing the pole within the space confines for at least 200 timesteps, the animated simulation reveals it does not do a very good job at balancing the pole in hopes of keeping it up for longer than 200 seconds, acting poorly from the perspective of general balancing. This is fine as it does not matter as 200 timesteps is all that is needed, but shows how the reward distribution really does make the learner only care about the 200 seconds it is given and not about being the best balancer overall. For instance in the state shown in Figure 9, the agent goes left even though a better general balancer would have gone right, but this doesn’t matter given only 200 seconds because the robot can recover fine even though it will be moving off the allowable space given greater than 200 seconds. To create better balancer, rewards could be given based on how vertical (or other metric of balance) the pole is at any timestep instead of just the fact that the pole has not yet fallen.

**Figure 9 | Simulation of Optimal Policy – Poor balancing still is successful overall**

**Conclusion and Improvements**

This biggest takeaway for me in this analysis is that Q-learning is an essential tool for real life application of value-based learning. In robotics and even simulation, as found here, it is often very difficult to acquire the complete transition and reward matrices. Q-learning took a much longer time to learn and even then, it did not learn the optimal policy with iteration limits. However, VI and PI would not work at all without these matrices as I came to find. Thus, the importance of Q-learning in this circumstance is revealed but if you have these reward and transition matrices or can acquire them easily, the it is a waste of time as VI and PI converge quickly and to optimal policies.

Some obvious improvements for the sake of this analysis would be to find a way to do VI and PI on the cart pole problem. I would do this in the future by running monte-carlo like simulations until I had almost all the transition and reward matrices filled. For the small remaining amount, it would probably be worth it to just pick a random action in that state. I would also create a grid generator for the first problem and expand the analysis to look at much larger grid sizes as well. If I had a lot of time I would have run the Q-learner on much larger state spaces with the cart problem to best approximate the continuous nature of real life. This would also provide clearer insight into how learning time increases with this increase in state space size.

I have learned in this analysis that to have an optimal learner for a given task, the rewards must match the goals. For instance in this case, the goal was just to balance the pole ‘good enough’ for 200 timesteps, and the quickest learner did this well but did not actually balance well in general. Thus, more general goals must be defined in terms of rewards properly.