**Markov Decision Processes**

For this assignment we were asked to implement Markov Decision Process on a set of problem, one simple with a small number of states and a harder one with a larger number of states. The MDP problems were solved using the following algorithms:

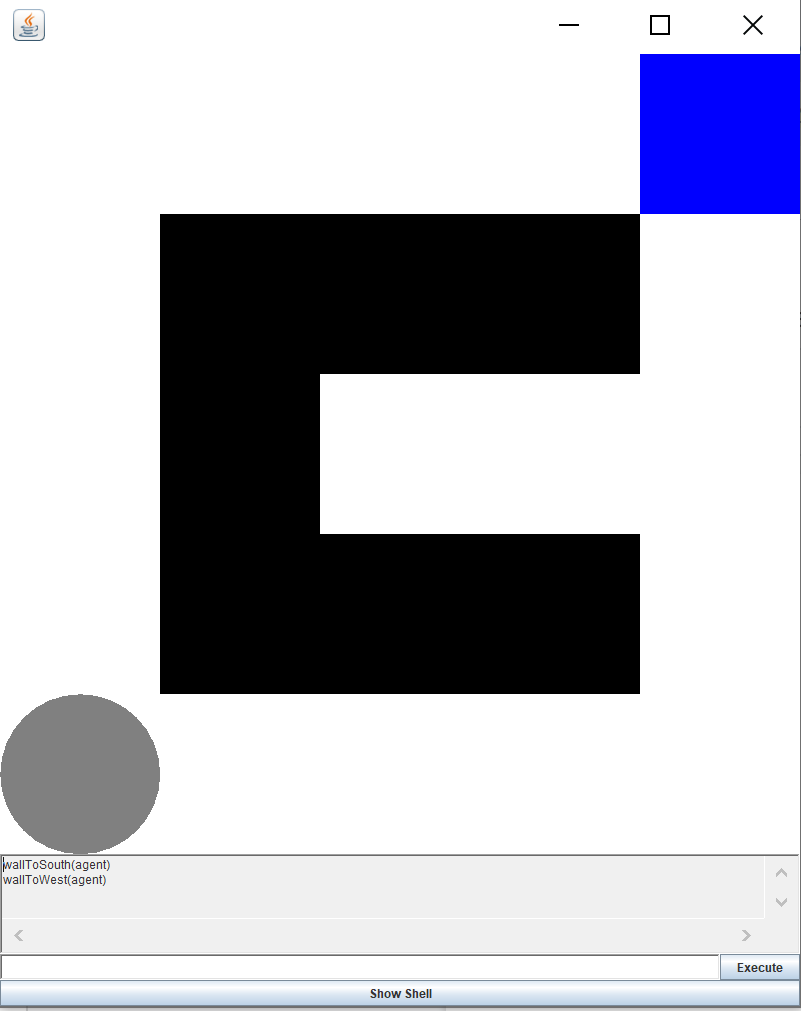
- Value Iteration

- Policy Iteration

- Q-Learning as our Reinforcement Learning Algorithm

The algorithm were implemented using the Brown-UMBC Reinforcement Learning and Planning (BURLAP) Java code library.

**Simple problem with a small number of states**



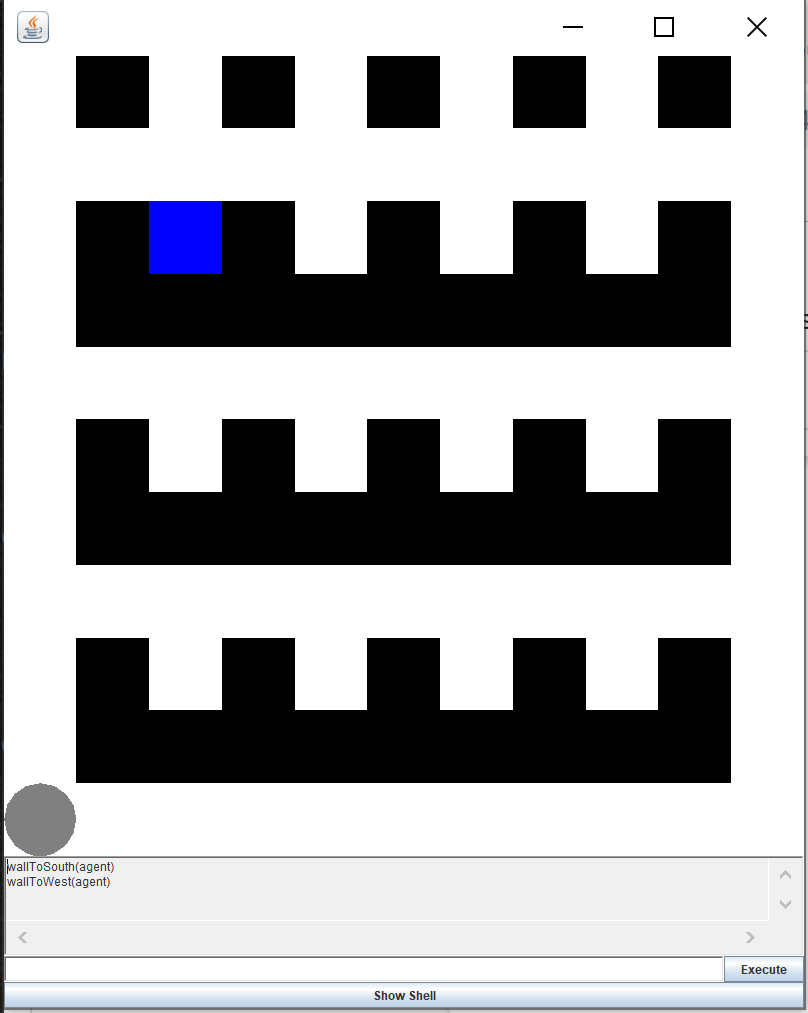
In this problem the agent (the grey sphere) must get to the goal state in blue (blue square). It is an almost trivial problem, excepted for the inlet where the agent may get stuck if the probability to succeed in the next transition state is low.

In my small number of states problem, I have chosen a 5x5 grid with 18 possible states where the goal state is located at (4;4).

**Larger problem with many states**

In this problem the agent (the grey sphere) must get to the goal state in blue (blue square). The problem attempts to model an agent trying to find a book in a library bookshelf.

In my large number of states problem, I have chosen an 11x11 grid with 74 possible states where the goal state is located at (2;8).



**Solving the Markov Decision Process using Value Iteration and Policy Iteration**

**Value Iteration**

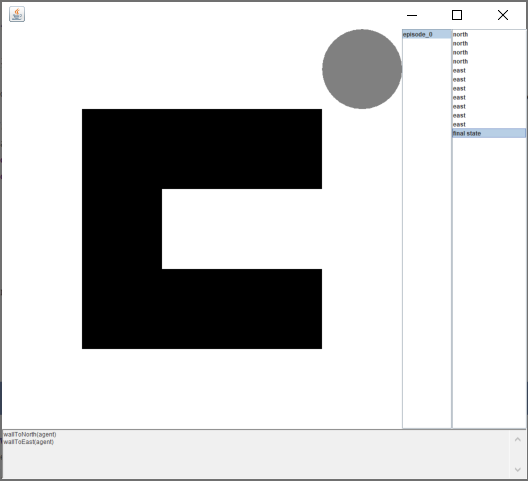
Using Value Iteration we can compute the optimal Markov Decision Process policy and its value. As such, Value Iteration starts at the end state and works in a backward motion to estimate the value-function, V(s). This function represents the wellbeing of an agent at a specific position or state. V(s) is iteratively computed to approximate a best estimation. Q(s,a) and V(s) are updated through each iterations until they converge. (Alzantot 2017)

Thus, the Value-Iteration algorithm improves the value-function V(s) iteratively until the value-function converges.

**Simple Problem:**

In this part of the assignment I have applied the 80% stochasticity and used 0.99 for discount factor with number of iterations is set to 30. Thus the problem is solved using 12 states.

Run time 104 milliseconds.



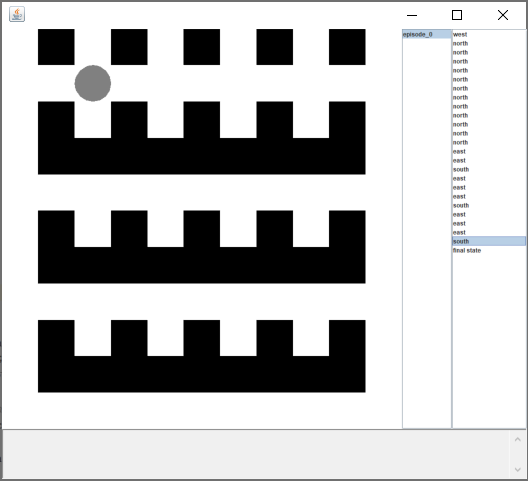
If we increase the number of iterations to 300 of course it increases the runtime to 868.

**Complex Problem:**

In the complex graph I have used 80% stochasticity and applied a discount factor of 0.99 for gamma with number of iterations is set to 30.

Agent reached the goal state with 25 states, where the terminal state is at grid location 2;8.

Runtime is 275 milliseconds



**Policy Iteration**

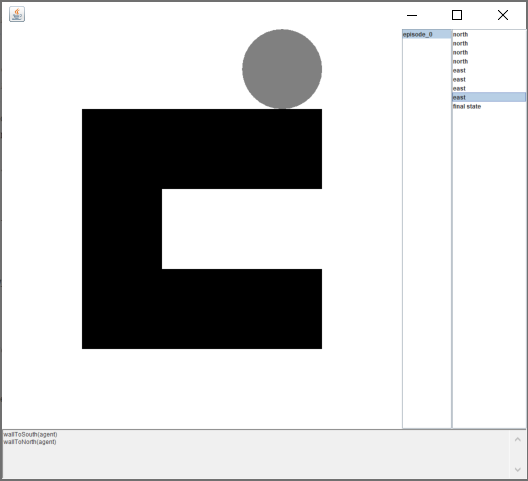
Policy-Iteration re-defines the policy iteratively and computes the value according to this new policy until the policy converges. Policy-Iteration is guaranteed to converge to an optimal policy and takes less iterations to do so. The Policy-Iteration algorithm manipulates the policy directly, rather than finding it indirectly via the optimal value function. (Kaelbling, 1996)

The value function of a policy is the expected infinite discounted reward that will be gained, at each state, by executing that policy. If the value can be improved by changing the first action taken. Then, the policy can be changed to take a new action to improve performance. Doing this improves the performance of the policy altogether. When no improvement is possible the policy is then optimal.(Alzantot, 2017).

**Simple Problem:**

Run time 88 milliseconds

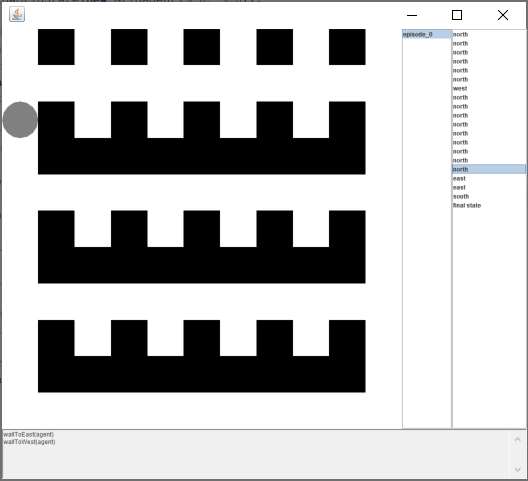
Problem is solved using 9 states.



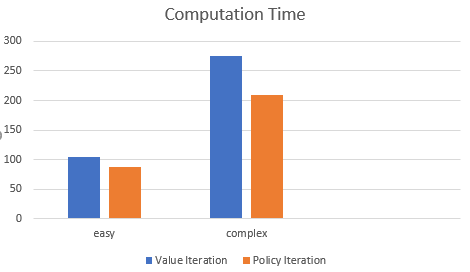
**Complex Problem:**

It took 20 states to reach the goal state.

Run time 210 milliseconds



Looking at the graph below, we are comparing the value iteration and policy iteration computation time



Based on the graph we can see that the value iteration takes longer computational time in both cases, the small number of states and large number of states. Thus the policy iteration takes less computational time.

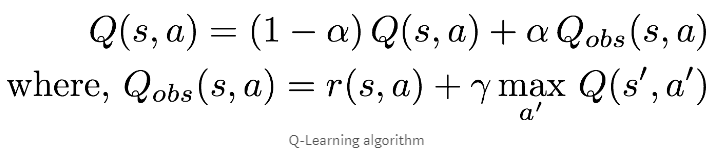
**Reinforcement Learning Algorithm**

**Q-Learning** was used as the reinforcement learning algorithm to solve both the simple and complex Markov Decision Process.

With Q-Learning the agent does not know in advance what the state transition and reward models are, instead, the agent only knows what are the possible states and actions can be.

Q-Learning once again is a model-free learning algorithm. In this model, the Q-Learning agent will learn from the environment from its history of interactions. Then approximating the environment state transition and reward models. The agent can then use value-iteration or policy-iteration to find an optimal policy.

The agent can only assess the environment’s current state. With Q-Learning the agent is learning through several iterations of interactions with the environment. With Q-Learning in this case, the agent will discover through various transitions what are the best action and worse actions. Q-Learning approximates the state-action pairs Q-function from the samples of Q(s, a) that we observe during interaction with the environment, or Time-Difference Learning. (Alzantot, 2017).



Alpha is the learning rate.

Q(s,a) table is initialized randomly.

The the agent starts to interact with the environment.

After each interaction the agent assesses the reward and states transition , r(s,a) and s’ respectively.

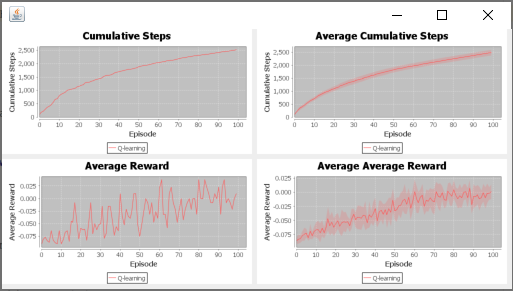
The agent then computes the observed Q-Value (Q\_obs(s,a)) then updates its estimate of Q(s,a), (Alzantot, 2017).

**Simple Problem:**

Using stochastic transition with 0.8 success rate.

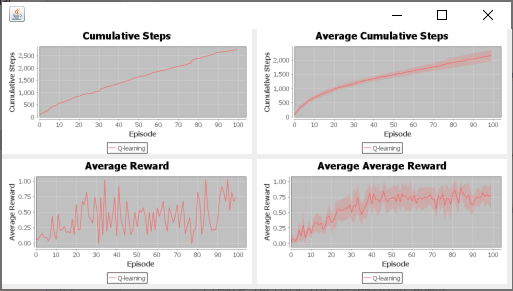
I have also set the discount rate to 0.99 and learning rate to 0.1, goal Reward is set to 1 and default is -0.1

In the graph below we can see the episodes are set to 100.



If the agent went to everywhere but the goal state then the expected reward would be -1.8 and by reaching the goal the reward would be -0.8, thus looking at the graph above Q-Learning doing poorly compared to value and policy iterations on the Simple problem with small number of states.

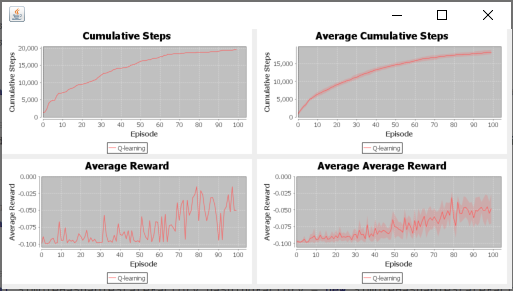
When increasing the reward at the goal state to 10 then the average reward no longer goes below zero, as can be seen in the graph below:



**Hard Problem:**

Using stochastic transition with 0.8 success rate.

I have also set the discount (gamma) rate to 0.99 and learning rate to 0.1.



If the agent went everywhere but the goal state then the expected reward would be -7.4 and by reaching the goal the reward would be -6.4, however in the graph above we can see the worse average reward is -0.1 which lets me believe that Q-Learning does learn from past states.

**Conclusion**

The value iteration in easy and hard grid case took more states then the policy iteration. Hence the number of states affects the rewards collected, runtime and the stochasticity.

The value iteration and policy iteration does a better performance compared to the Q-Learning algorithm in the number of states and runtime.

Value iteration and policy iteration does go to the same answer/goal however they take different path based on the defined algorithms.

Policy iteration seems to perform the best out of the three algorithms.

.

**Reference(s)**

Alzantot, M. (2017, July 9). *Deep Reinforcement Learning Demystified (Episode 2) — Policy Iteration, Value Iteration and Q-learning*. Retrieved from Medium.com: <https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa>

Pack Kaelbling, L. (1996, May 1). *Policy Iteration*. Retrieved from cs.cmu.edu: <https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node20.html>