# Robot Autonomy Scribe Notes

April 3, 2017

Andy Tracy - atracy@andrew.cmu.edu
Che-Yen Lu - cheyenl@andrew.cmu.edu
Matt Lauer - mlauer@andrew.cmu.edu

## Reinforcement Learning

There are various type of learning:

- Supervised Learning
  - Characterized by input data that is organized into categories by users prior to the learning process
- Unsupervised Learning
  - In this paradigm, the learning algorithm categorizes the data automatically
- Reinforcement
  - In reinforcement learning, the system learns a policy for how to interpret input data. It does this by applying a reward function to new experiences and finding the policy that maximizes the total reward.

Within reinforcement learning, there are different approaches:

- Learn a model of the environment
- Learn a value function that can be used to calculate a policy
- Learn a policy directly

### Learning a Value Function (Q-Learning)

In the absence of a model of the environment, reinforcement learning can still be applied by directly learning a value function (Q function). This is done by iteratively applying actions to states and observing the resulting rewards.

The basic algorithm for learning a value function is:

- Input experience
- Learn a state transition or Q function
- Calculate the ideal policy
- Apply the policy to new experiences
- Iterate and update the policy

In order to update the policy on each iteration, one can apply a learning rate. This means that the updated value function is determined by the value from prior iterations, a discount factor, and the newly estimated value:

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)$$

Source: [https://en.wikipedia.org/wiki/Q-learning](https://en.wikipedia.org/wiki/Q-learning)

The discount factor allows the algorithm to weight past estimates more or less than current estimates.

## Policy Search

A common approach for learning a policy is to use an existing policy and optimize around it. This assumes that the policy can be parameterized in such a way to make this approach effective.

## Exploration vs Exploitation

When an autonomous agent has an existing policy, they need to decide when they will follow the policy (exploitation) and when they will deviate from the policy (exploration). This is a necessary aspect to consider because no policy will be completely relevant in all environments and situations. When a policy does not apply to a new, unknown environment, exploration is required.

There are different ways to decide how to implement exploration:

- A greedy approach
  - For each planning step, take a random action with a fixed probability
- Use an optimistic initialization of the parameters
- Use a set of confidence bounds
  - When a state is visited more frequently, the certainty of the value function at that state increases; this encourages exploration into less visited states to increase the confidence level

## Problems with Reinforcement Learning

- The curse of dimensionality. As your dimensions grow, you state action space grows exponentially. This can make it essentially impossible to map out the values associated with each state in a reasonable time.
- The model of the robot or environment may be inaccurate or may not exist

- Exploration is difficult
    - How does one make it safe, inexpensive, and reproducible?
- The reward function can be hard to define
- There are a lot of parameters to learn
    - For each state, there are parameters for every possible action

## Where is Reinforcement Learning Used?

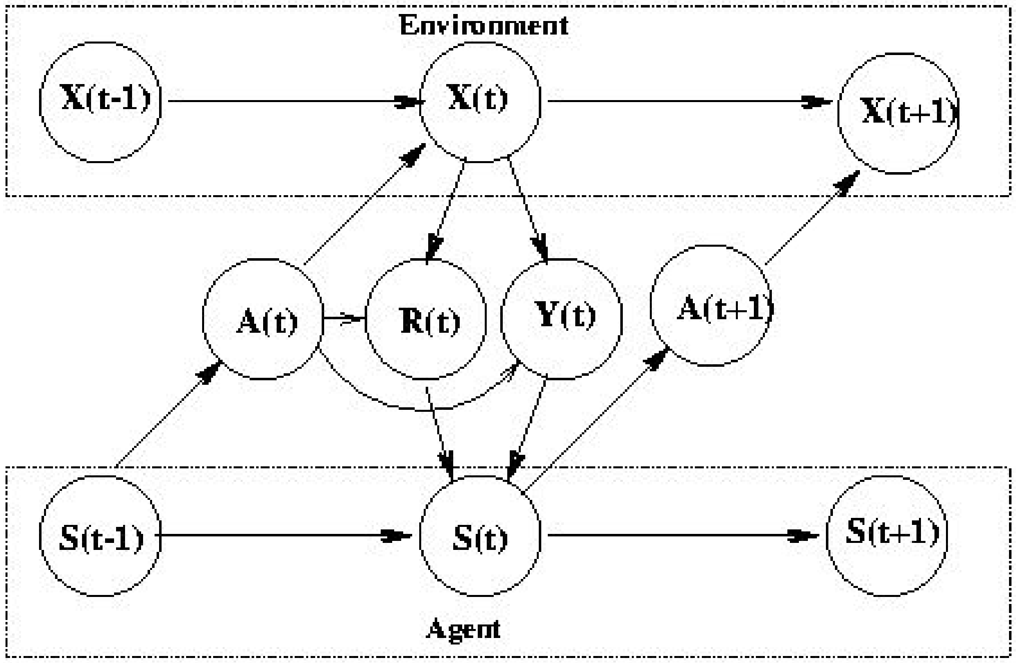
A few examples from CMU include an autonomous helicopter, a robot that balances poles, and a robot that learned to play the ball in a cup game.

In general, reinforcement is not widely used.

# Markov Decision Process

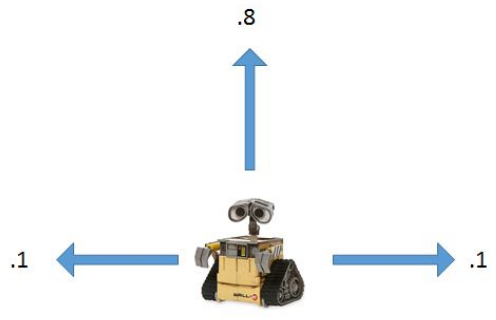
Markov Decision Process provides a mathematical framework for decision making, which is useful for solving optimization problems. An MDP is a discrete time stochastic control process. At each time step, the process is in some state $s$ and the decision maker may choose any action $a$ that is available in state $s$. The process responds at the next time step by randomly moving into a new state $s'$, and giving the decision maker a corresponding reward $R_a(s, s')$ .

Basically, MDP is an extension of Markov Chain (MC), and it also follows Markov property: the effects of an action taken in a state depend only on that state and not on the prior history. The difference between MDP and MC is the addition of actions and rewards. Conversely, if only one action exists for each state and all rewards are the same, a MDP reduces to a MC.

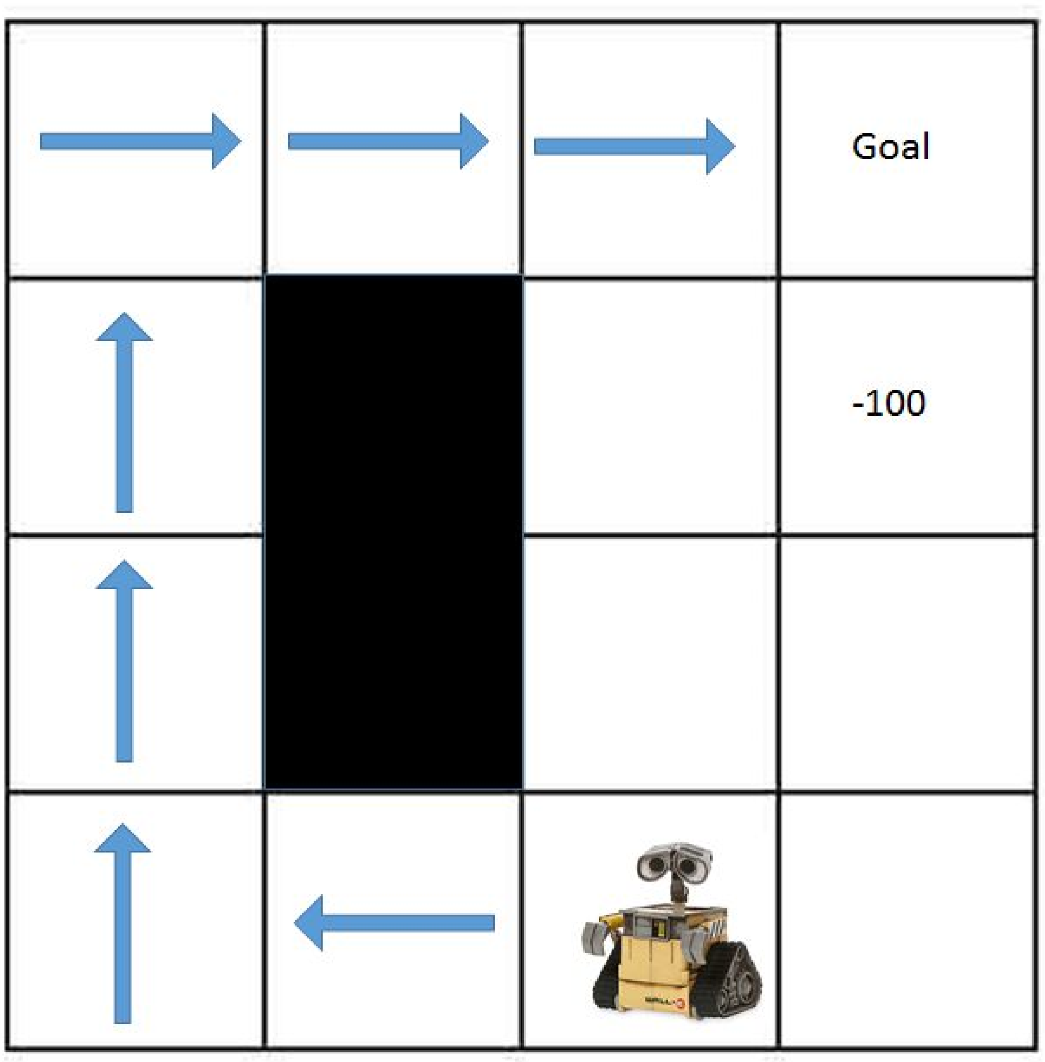


Consider this world, where the value in each box is the reward for that state (and the goal state is in the top right corner):

| | | | |
|---|---|---|---|
| -1 | -1 | -1 | 10 |
| -1 | ■ | -1 | -100 |
| -1 | ■ | -1 | -1 |
| -1 | -1 | -1 | -1 |



The robot will want to take a short path to reach the goal, because every intervening state has a negative reward and it wants to avoid negative rewards. That said, there is also a small chance that the robot will not go in the intended direction. The previous image of the robot shows that there is only an .8 chance of going the direction of the given action. So, when the MDP is evaluated, the robot will attempt to avoid the risky decision of going near the -100 reward and instead follow this path:



## Definition

There are five symbols in MDP

$S$ : is a finite set of states

$A$ : is a finite set of actions

$P_a(s, s')$ : probability that action $a$ in state $s$ at time $t$ will lead to state $s'$ at time $t+1$

$R_a(s, s')$ : is the immediate reward (or expected immediate reward) received after the transition

$\gamma$ : is the discount factor, which represents the difference in importance between future rewards and present rewards

## Algorithms

In the original MDP, we need to optimize the policy $\pi(s)$ and immediate reward function $V(s)$:

$$\pi(s) := \arg\max_a \left\{ \sum_{s'} P_a(s, s') \left( R_a(s, s') + \gamma V(s') \right) \right\}$$

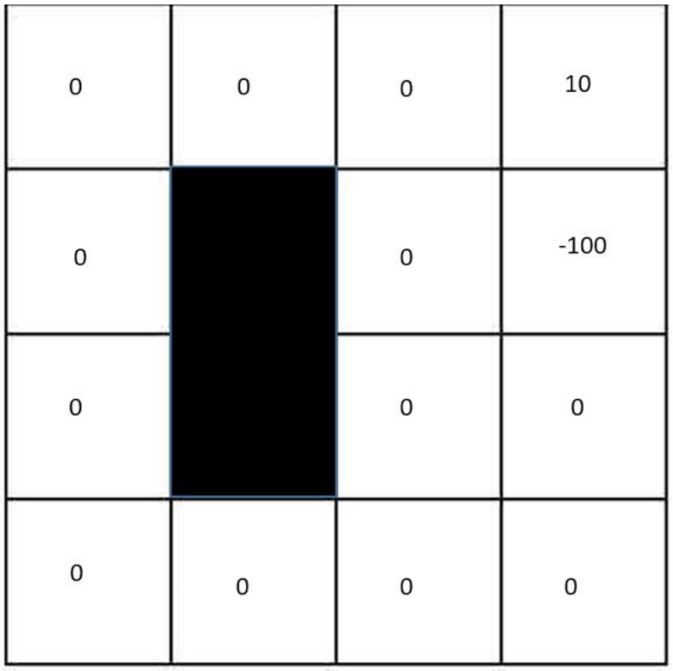
$$V(s) := \sum_{s'} P_{\pi(s)}(s, s') \left( R_{\pi(s)}(s, s') + \gamma V(s') \right)$$

However, in class, we use one of the MDP evaluation methods, which is value iteration. Instead of optimizing solution by two steps, there is only one calculation for reward functions. Policy function is merged into reward function. The algorithm has the following steps, which are repeated in some order for all the states until no further changes take place:

```
loop till V converge
    loop over all $s \in S$
        loop over all $a \in A$
            $Q(s,a) = V_t(s) + \gamma \Sigma_{s' \in S} P_a(s,s') V_t(s')$ # $V$ from last iteration
        end
        $V_{t+1}(s) = max_a[Q(s,a)]$    #new $V$
    end
end
```

Note: $R(s,a)$ *is used instead of* $V(s)$ *in class, but I think they are the same*

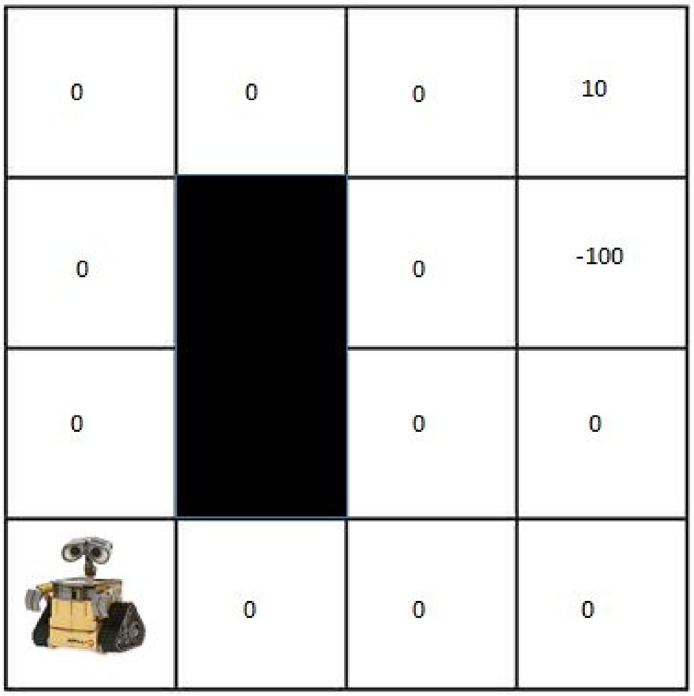
The first step of this method is to initialize V(s). For our example, we initialize V(s) to look like this:

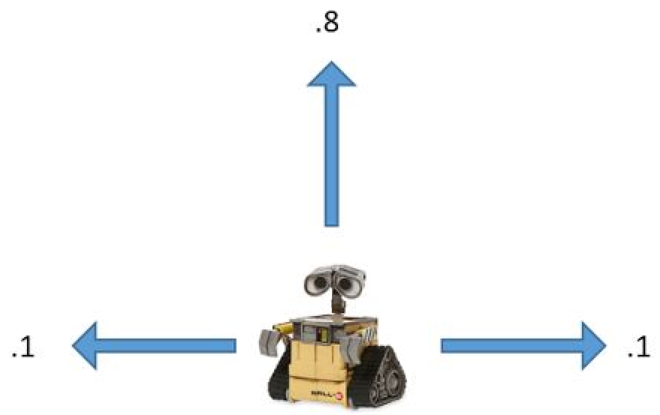


The next step is to iterate over all actions and states in the environment. Here are some examples of how we can evaluate different states and actions using the Bellman equation:

$$Q(s, a) = R(s,a) + \gamma(P(s,a,s')*V(s'))$$

For s = (3,3) and we have a world that looks like this:

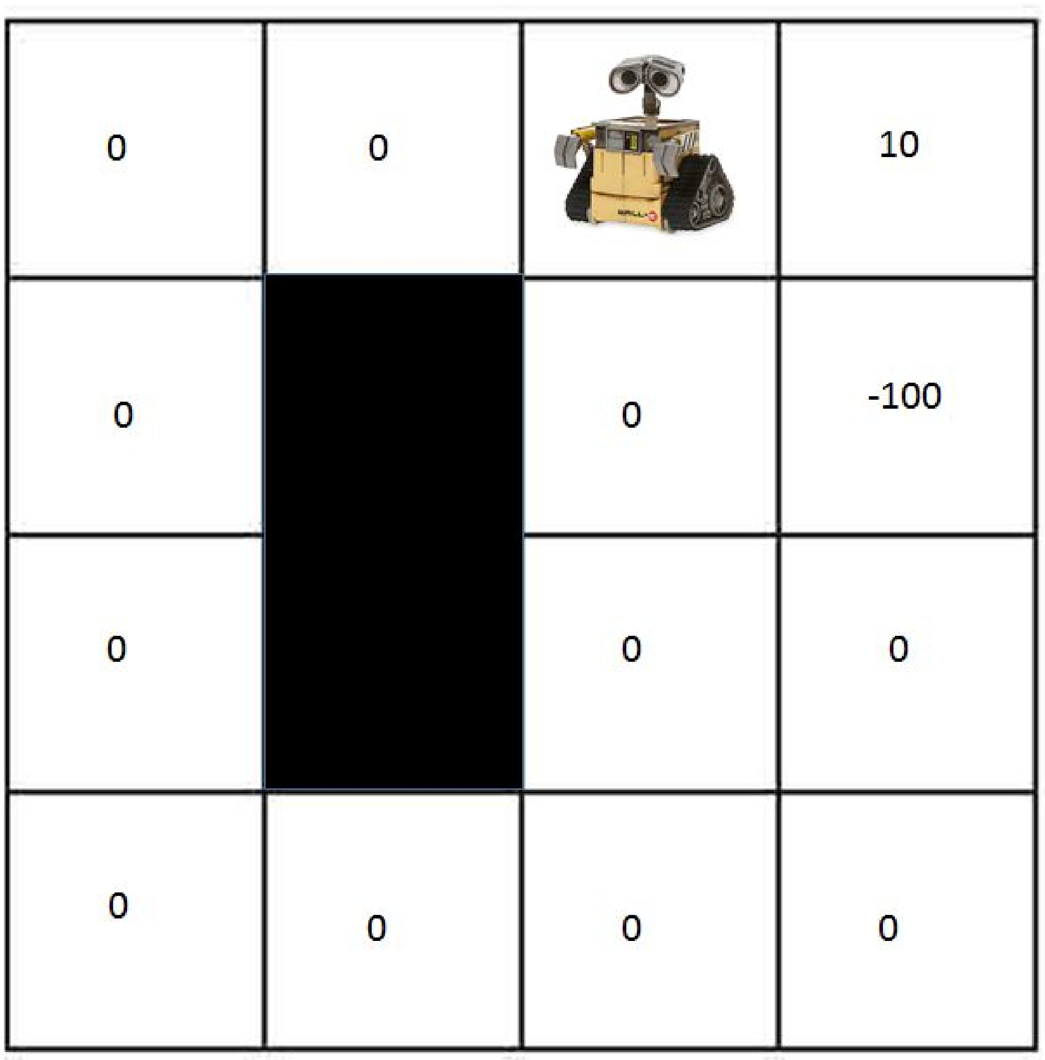




It is important to note that there is some chance our robot does not move in the intended direction. This will cause us to evaluate every possible resultant state for each action and weight the state by the probability of its occurrence due to the action.

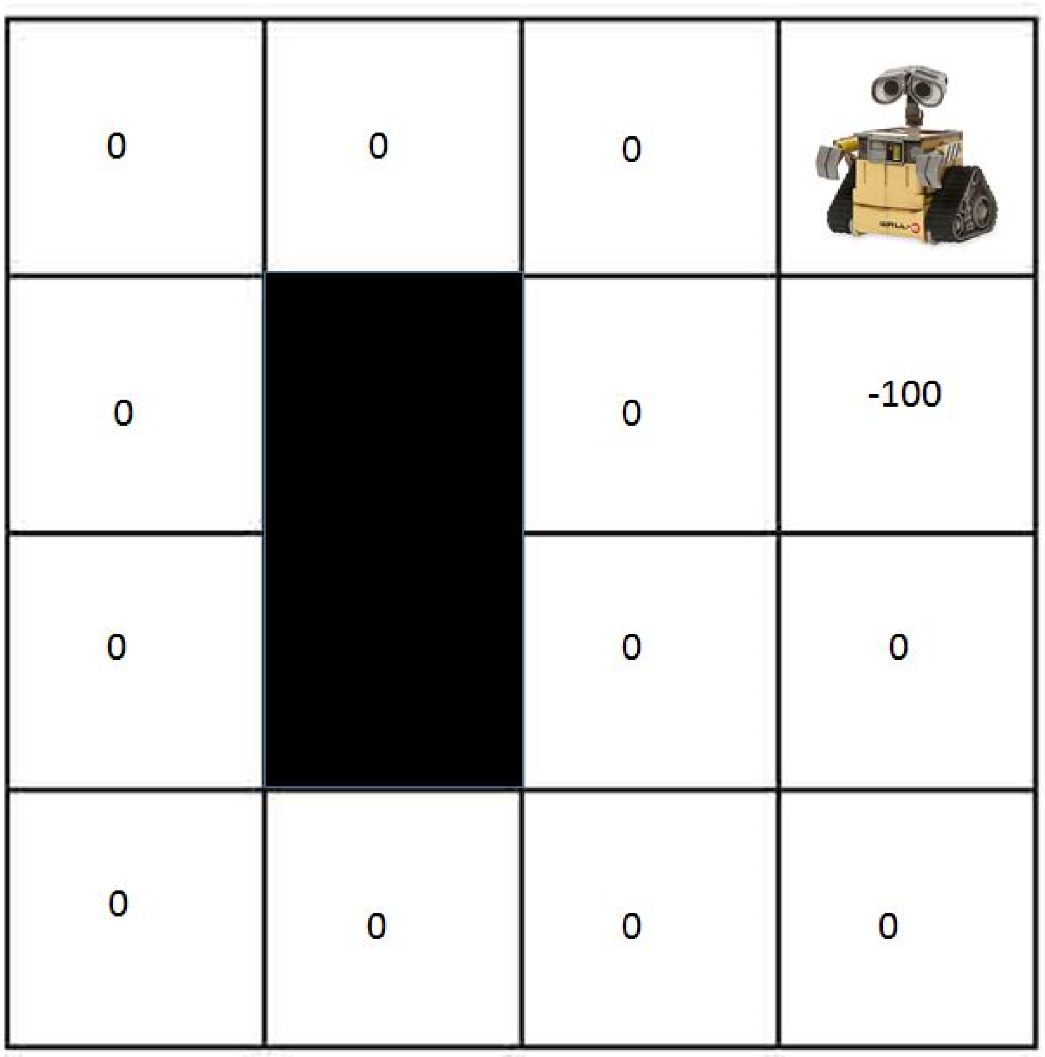
$Q(s, a) = R(s,a) + \gamma(P(s,a,s')*V(s'))$
$Q((3,3),upward) = -1 +.9(.8*0 + .1*0 + .1*0)$
$Q = -1$
Since all nearby states have a V of zero (we just initialized), Q is equal to the reward at the current state.
You can verify that for any action in this state (left, right, down), there will be a Q of -1.
This means that when we select a V for this state it will be -1, since $V(s) = \max_a(Q(s,a))$

For s = (1,0) we have a world that looks like this:



$Q(s, a) = R(s,a) + \gamma(P(s,a,s')*V(s'))$
$Q((1,0),upward) = -1 +.9(.8*0 + .1*0 + .1*10)$
$Q = -.1$
In this case, a nearby reward makes us evaluate the state action pair higher than the previous example.
$Q((1,0),right) = -1 +.9(.8*10 + .1*0 + .1*0)$
$Q = 6.2$
Since we actually try to move to the reward, our Q value is very high.
$V(s) = \max_a(Q(s,a)) = 6.2$ (you can find the other Q values to verify this)

For s = (0,0) we have a world that looks like this:



$Q(s, a) = R(s,a) + \gamma(P(s,a,s')*V(s'))$
$Q((0,0),upward) = 10 + .9(.8*10 + .1*0 + .1*0)$
$Q = 17.2$

$Q((1,0),right) = 10 + .9(.8*10 + .1*-100 + .1*0)$
$Q = 8.2$
Even, just a small chance of a very bad reward can have a big effect.
$V(s) = \max_a(Q(s,a)) = 17.2$ (you can find the other Q values to verify this)