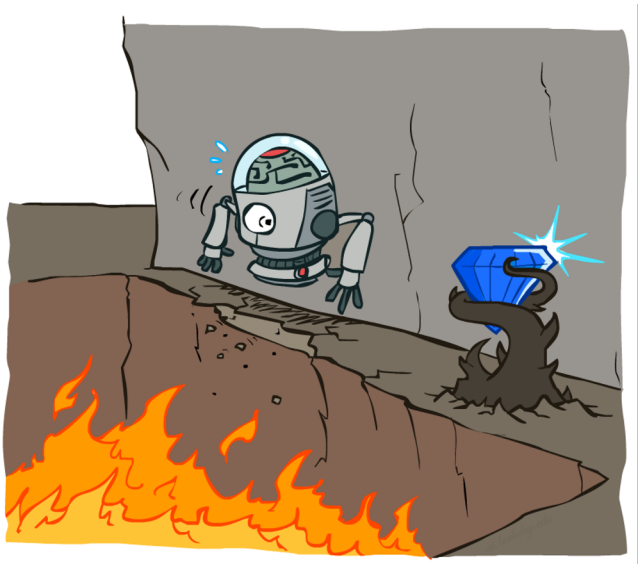
CS440 – Non-Deterministic Search



Purpose: Consider a robot that performs an action of moving forward and 80% of the time the robot succeeds with moving forward but 20% of the time the wheels slip, so 10% of the time the robot slips and moves left and 10% of the time the robot slips right.

The purpose of this module is to present how to deal with this kind of non-deterministic scenario.

Knowledge: This module will help you become familiar with the following content knowledge:

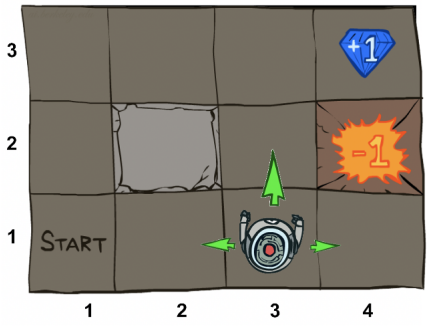
- Markov Decision Processes (MDPs)
- MDP search trees
- How to compute an optimal policy

Activity1 - Markov Decision Processes:
With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 30 minutes.

1. A **Markov Decision Process (MDP)** has six components:

   - A set of states $s \in S$
   - A set of actions $a \in A$
   - A transition function T(s,a,s') which gives the probability that a from s leads to s'
   - A reward function R(s,a,s')
   - A start state
   - Maybe a terminal state

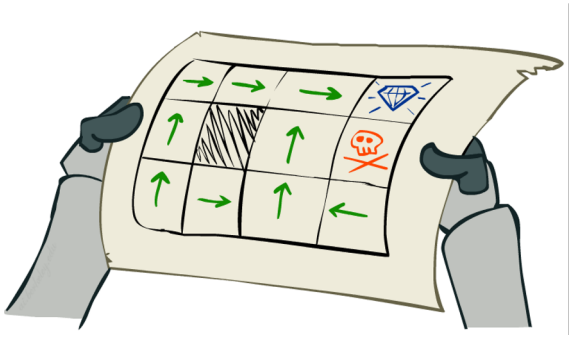
   Suppose a robot on moving forward has 80% chance of success and 10% chance of slipping to the right and 10% chance of slipping left. The robot needs to traverse a world to obtain a jewel while avoiding (hopefully) falling into a pit of lava.



   What would be T((1,3), forward, (2,3))?
   What would be T((2,3), forward, (2,4))?
   What would be T((2,3), forward, (2,3))?

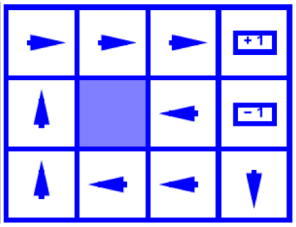
2. For MDPs, we want an optimal *policy*, $\pi^* : S \rightarrow A$

   A policy π gives an action for each state and an optimal policy is one that maximizes expected utility if followed.



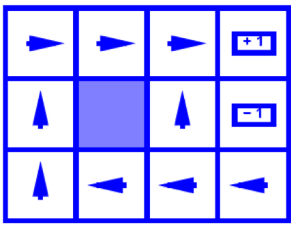
   What is the difference between a plan in a deterministic search snd a policy in a non-deterministic search?
   Given a policy, what if any computation would an agent do?

3. Suppose the "living reward" function is R(s)=-.01. This means that the agent is "rewarded" -.01 points for each move. The agent gets 1 point for getting the jewel and -1 point for dying in the pit of lava. Here is an optimal policy:



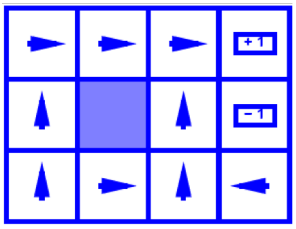
   Why is the agent moving left around the wall rather than going past the pit?
   What the heck is happening in the state right next to the pit? Why would the robot want to move forward into the wall?

Now look at the optimal policy after changing the living reward R(s)=-.03



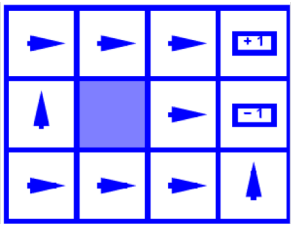
Why does the optimal policy change when the living reward is changed?

The next optimal policy is after changing R(s)=-.04



What would you expect would change in the policy if R(s)=-2? (Think about this before peeking at the resulting policy on the next page)

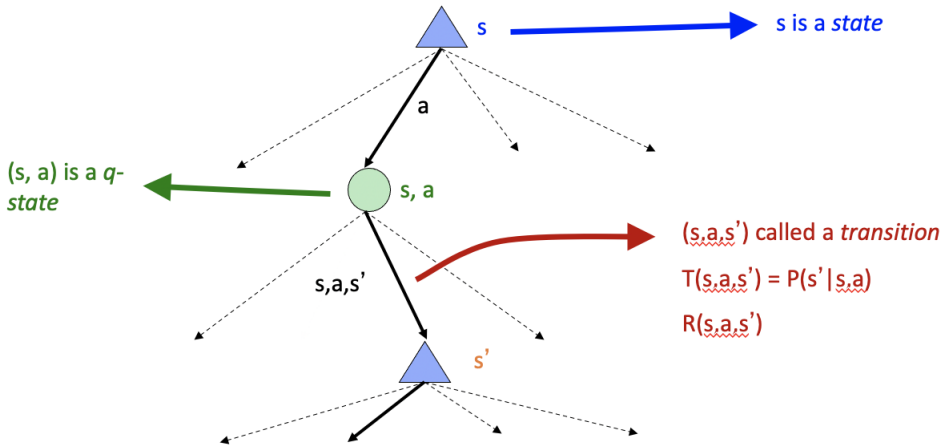
Optimal policy when R(s)=-2



Why is the robot now throwing itself into the pit?!!

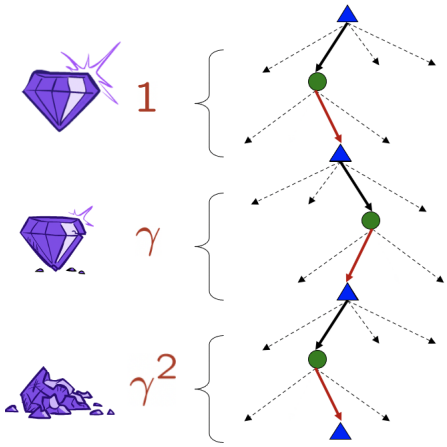
Activity2 - MDP Search Trees:
With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 20 minutes.

1. Each MDP state projects an expectimax-like tree.



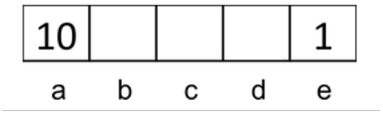
In the MDP, chance nodes represent uncertainty about what might happen based on (s,a), as opposed to being a random adversary. A Q-state (s,a) is when you were in a state s and took an action a and haven't seen which of the possible results has occurred.

Rewards in the future (deeper in the tree) matter less than rewards that are closer, therefore we are going to *discount* by a factor of γ the rewards in the future.



If the discount is 0.5 and we have [1,2,3] which is a reward of 1 followed by 2 in the next step, followed by 3 in the next step, we would compute the *utility* of that as U([1,2,3)] = 1 * 1 + .5* 2 + .25*3 = 2.75

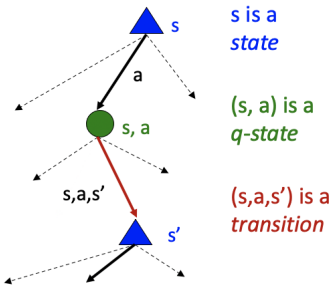
Consider the following states and rewards:



The actions are East, West, and Exit (which is only available at the exit states a and e).
The initial state is d and the transitions are deterministic here.

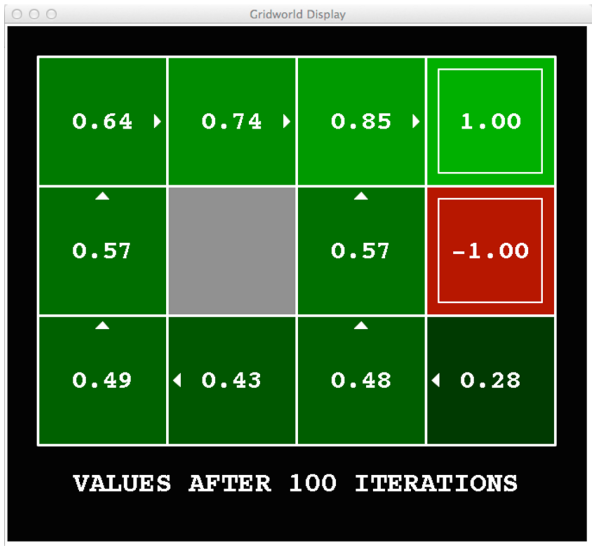
For $\gamma = 1$, what is the optimal policy and the reward?
For $\gamma = .1$, what is the optimal policy and the reward?

2. We need some notation:



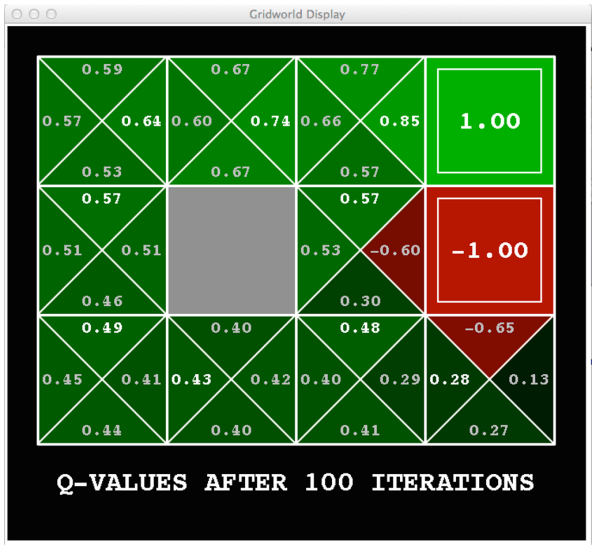
V*(s) = expected utility starting in s and acting optimally
Q*(s,a) = expected utility starting out having taken action a from state s and thereafter acting optimally
π^*(s) = optimal action from state s

This is what the V values of our gridworld might look like:



They give the expected utility of being in a state given all the things that can happen.

This is what the Q values of our gridworld might look like:



This gives the expected utility takeing each of the four actions.

Looking at the V and Q values for gridworld, how would we get the V* value of a state if we know the Q-values?

This gives us a recursive definition for V*:

$V^*(s) = \max_a Q^*(s, a)$

$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$

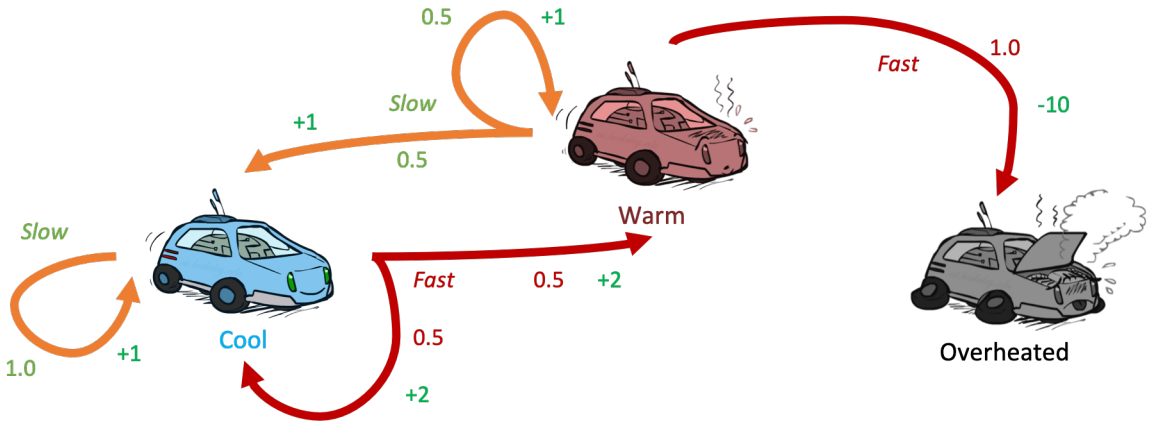
so

$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$

We need a way to compute these!!

Activity3 - Getting an optimal policy:
With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 30 minutes.

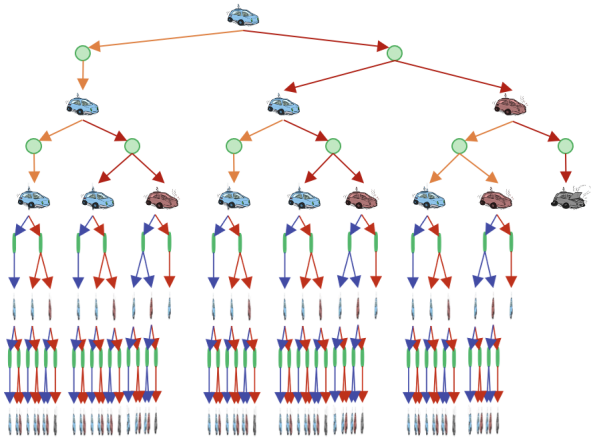
1. Consider another example where we have a robot car that wants to travel far, quickly. It has three states: Cool, Warm, Overheated and two actions: Slow, Fast. Going faster gets double reward.
   Here is a state diagram:



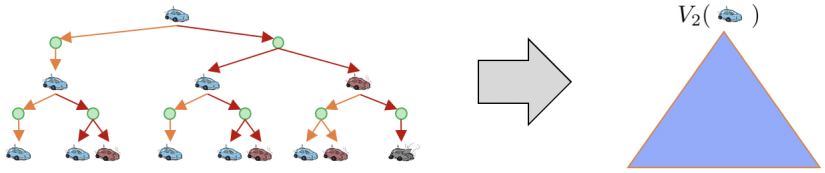
   Draw three levels of the search tree.

   What are some problems with the entire search tree and suggest some things that can be done to overcome these problems.

2. The deep parts of the tree don't matter if γ is less than 1 so we can do a *time-limited* search which only goes down k steps.

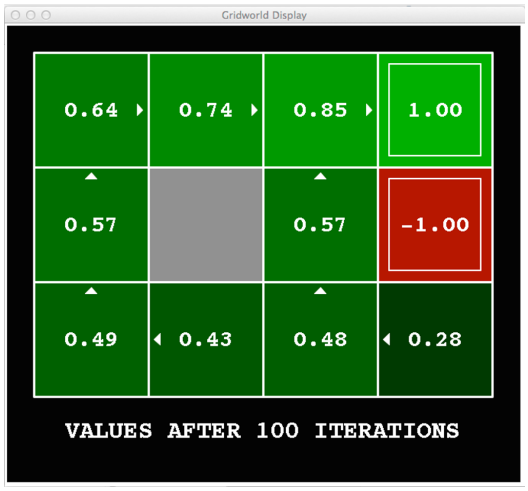


The idea is to define $V_k(s)$ to be the optimal value of s in the game ends in k steps. This is what expectimax would give from s with a depth k search.



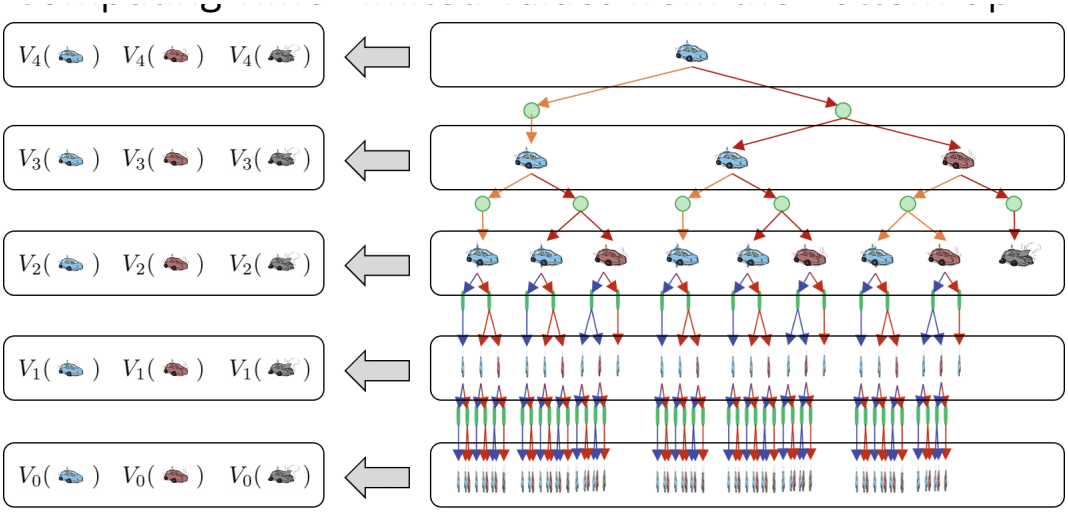
We will look at the V_k values for gridworld with a living reward of 0. Here are the values when k is 12:

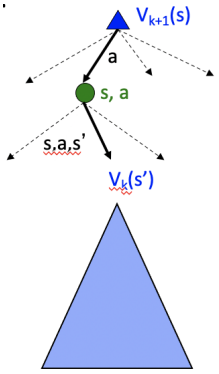
Here are the values when k is 100:



Do you see much difference? Does this reassure you that limiting the k will be okay?

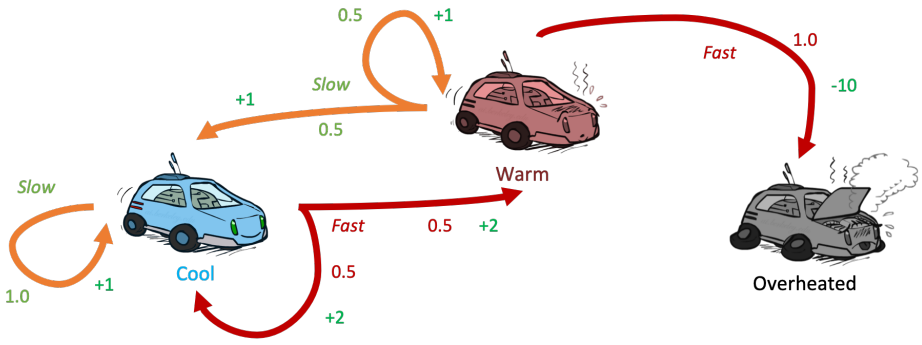
3. We will compute the V_k values from the bottom-up.





We will use the formula:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$



$V_0(Cool) = 0$ $V_0(Warm) = 0$ $V_0(Overheated) = 0$

Use the formula to compute the V_1 values

Again, compute the V_2 values.