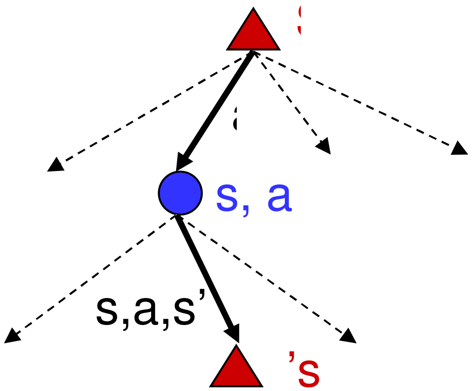
# Recap: MDPs

- Markov decision processes:
  - States S
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount $\gamma$ )
  - Start state $s_0$




- Quantities:
  - Policy = map of states to actions
  - Episode = one run of an MDP
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state
  - Q-Values = expected future utility from a q-state

# Utilities of Sequences

- What utility does a sequence of rewards have?

- Formally, we generally assume stationary preferences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r_0', r_1', r_2', \ldots]$$
$$\Leftrightarrow$$
$$[r_0, r_1, r_2, \ldots] \succ [r_0', r_1', r_2', \ldots]$$

- Theorem: only two ways to define stationary utilities
  - Additive utility:

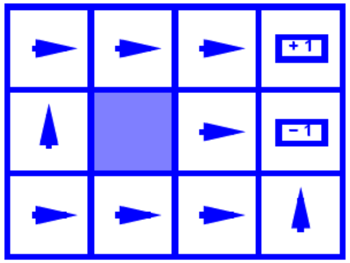
$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

  - Discounted utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$$

# Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards



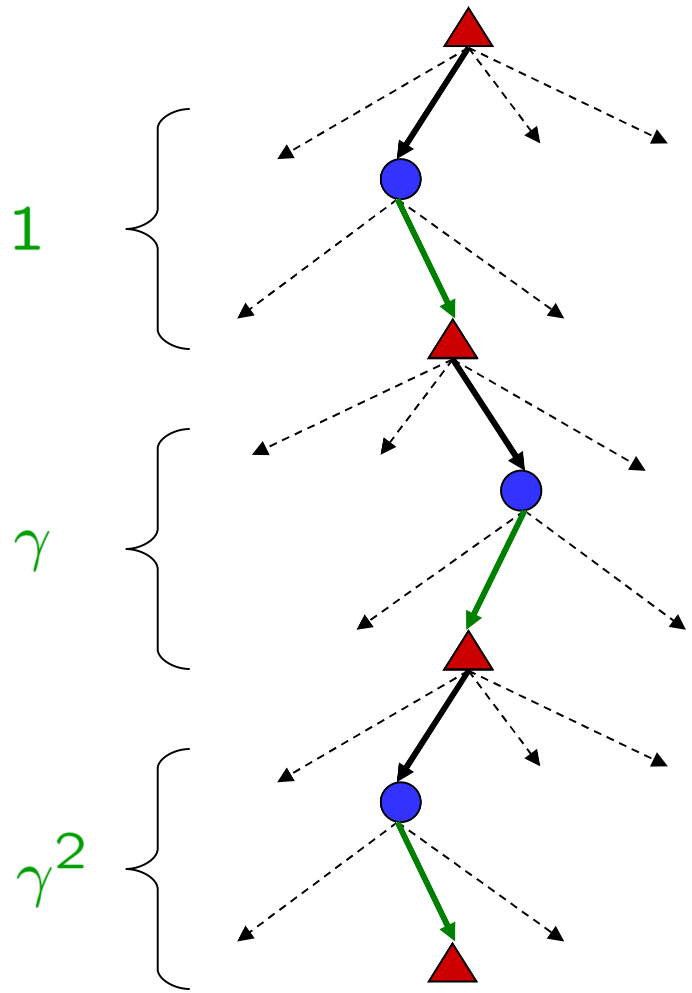

- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies ($\pi$ depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “done” for High-Low)
  - Discounting: for $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller $\gamma$ means smaller “horizon” – shorter term focus

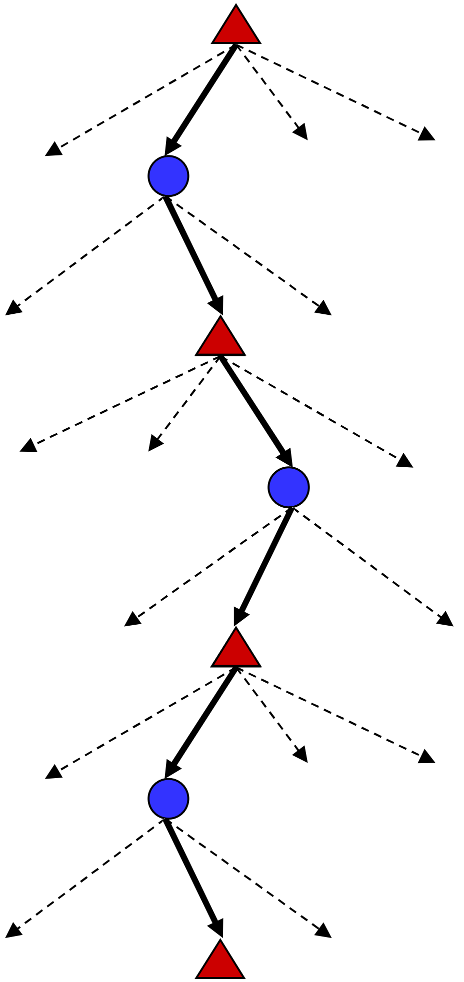
# Discounting

- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



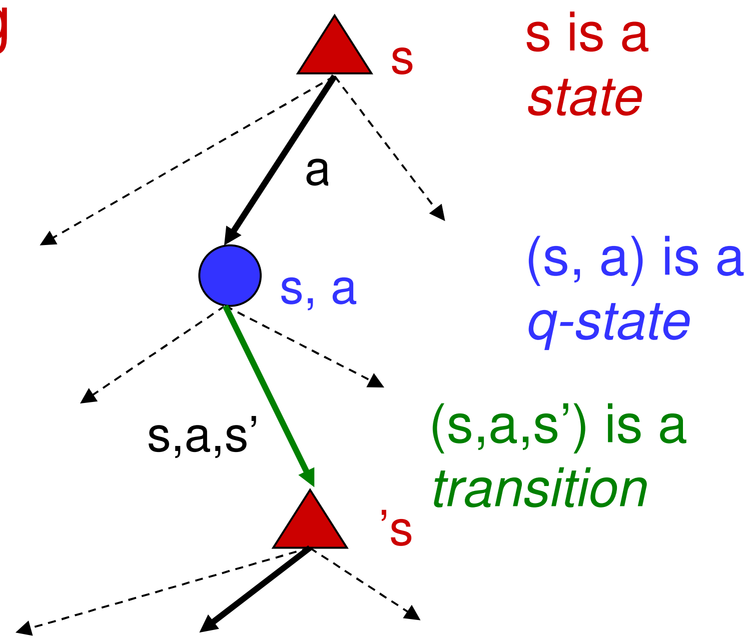
# Why Not Search Trees?

- Why not solve with expectimax?
- Problems:
  - This tree is usually infinite (why?)
  - Same states appear over and over (why?)
  - We would search once per state (why?)
- Idea: Value iteration
  - Compute optimal values for all states all at once using successive approximations
  - Will be a bottom-up dynamic program similar in cost to memoization
  - Do all planning offline, no replanning needed!



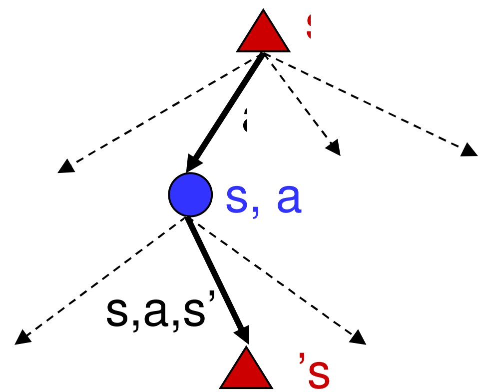
# Optimal Utilities

- The utility of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally
- The utility of a q-state (s,a):
  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
  $\pi^*(s)$ = optimal action from state s



# Bellman Equations

- Definition of utility leads to a simple one-step lookahead relationship amongst optimal utility values:




Total optimal rewards = maximize over choice of (first action plus optimal future)

- Formally:

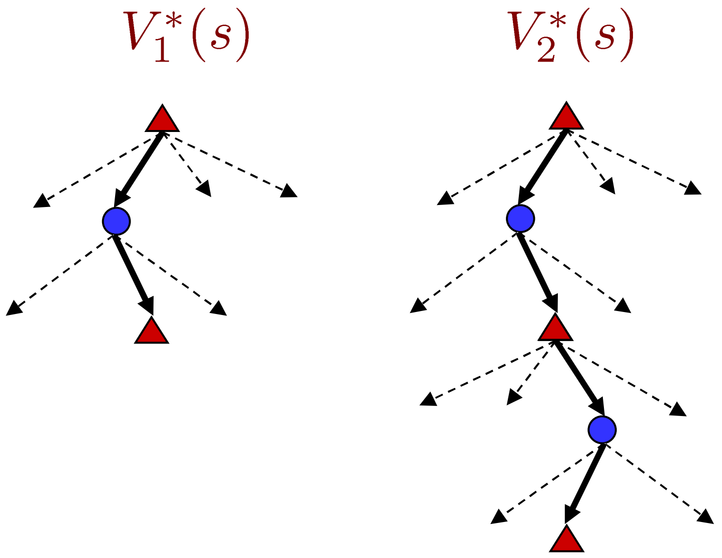
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

# Value Estimates

- Calculate estimates $V_k^*(s)$
  - Not the optimal value of s!
  - The optimal value considering only next k time steps (k rewards)
  - What you'd get with depth-k expectimax
  - As k $\rightarrow \infty$, it approaches the optimal value
- Almost solution: recursion (i.e. expectimax)
- Correct solution: dynamic programming



# Value Iteration

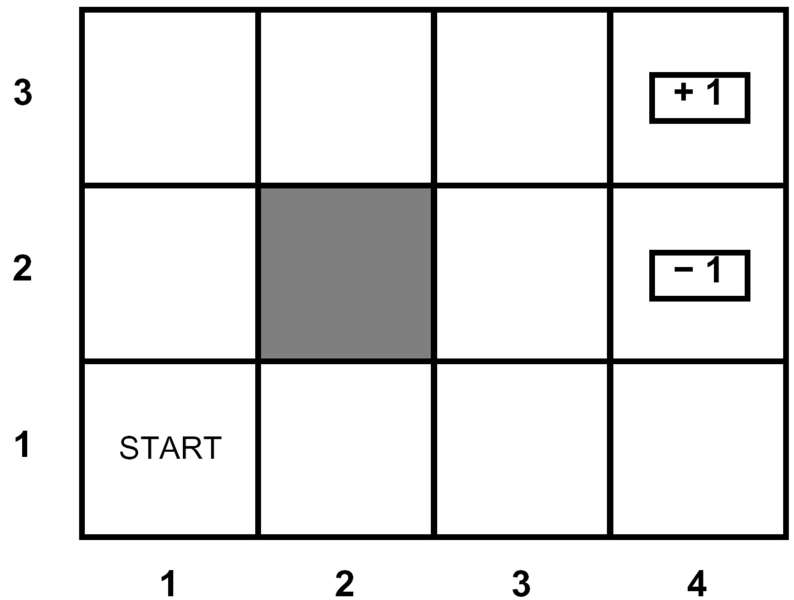
- Idea:
  - Start with $V_0^*(s) = 0$ for all s, which we know is right (why?)
  - Given $V_i^*$, calculate the values for all states for depth i+1:

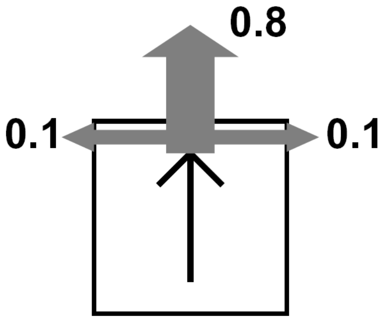
$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

  - Throw out old vector $V_i^*$
  - Repeat until convergence
  - This is called a value update or Bellman update
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

# Grid World

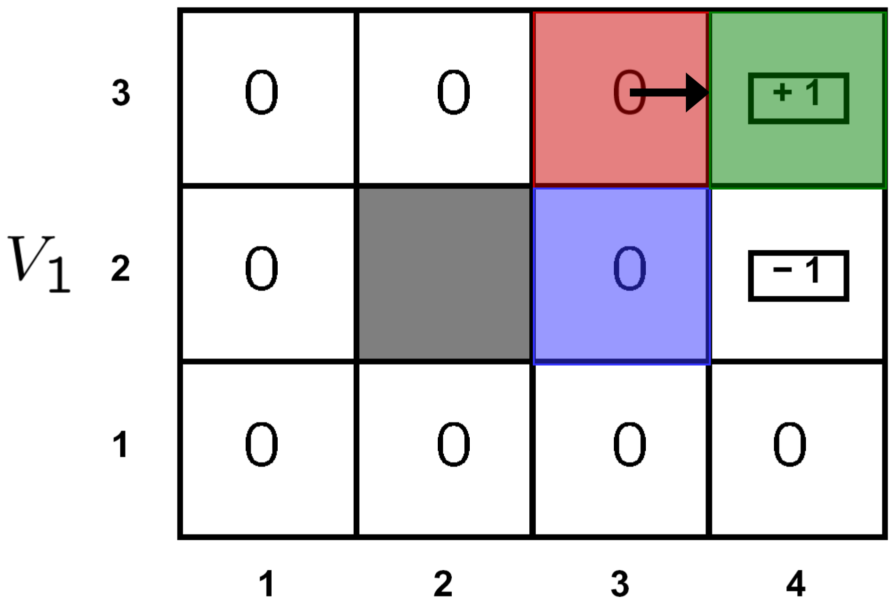
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North
    (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards*

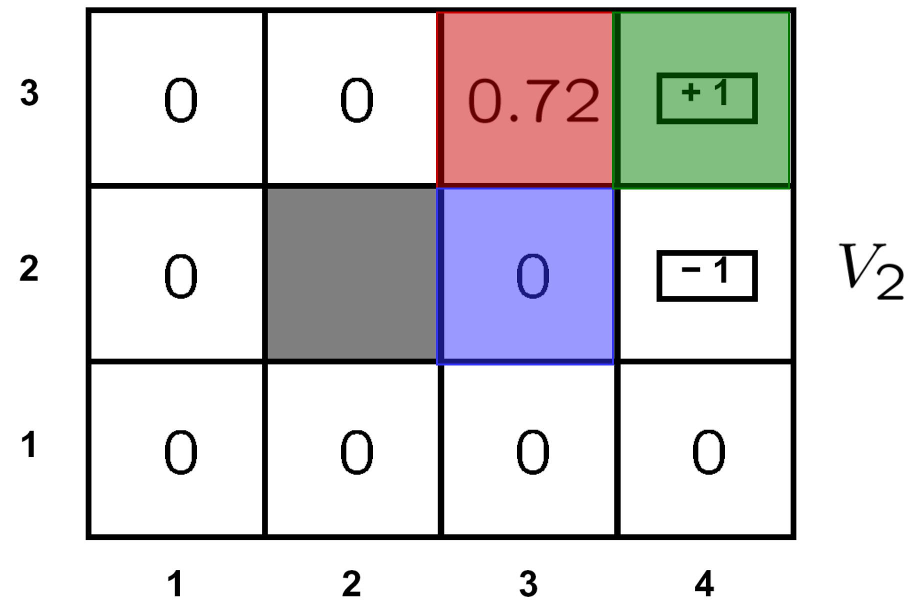


Example: $\gamma$ =0.9, living reward=0, noise=0.2

# Example: Bellman Updates




$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_i(s') \right]$$

$$V_2(\langle 3,3 \rangle) = \sum_{s'} T(\langle 3,3 \rangle, \text{right}, s') \left[ R(\langle 3,3 \rangle) + 0.9\, V_1(s') \right]$$

*max happens for a=right, other actions not shown*

$$= 0.9\,[0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$

# Example: Value Iteration

$V_2$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 3 | 0 | 0 | 0.72 | +1 |
| 2 | 0 | | 0 | −1 |
| 1 | 0 | 0 | 0 | 0 |

$V_3$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 3 | 0 | 0.52 | 0.78 | +1 |
| 2 | 0 | | 0.43 | −1 |
| 1 | 0 | 0 | 0 | 0 |

- Information propagates outward from terminal states and eventually all states have correct value estimates

# Convergence*

- Define the max-norm: $||U|| = \max_s |U(s)|$

- Theorem: For any two approximations U and V

$$||U^{t+1} - V^{t+1}|| \leq \gamma \, ||U^t - V^t||$$

  - I.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true U and value iteration converges to a unique, stable, optimal solution

- Theorem:

$$||U^{t+1} - U^t|| < \epsilon, \Rightarrow ||U^{t+1} - U|| < 2\epsilon\gamma/(1-\gamma)$$

  - I.e. once the change in our approximation is small, it must also be close to correct

# Practice: Computing Actions

- Which action should we chose from state s:
  - Given optimal values V?

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

  - Given optimal q-values Q?

$$\arg\max_a Q^*(s,a)$$

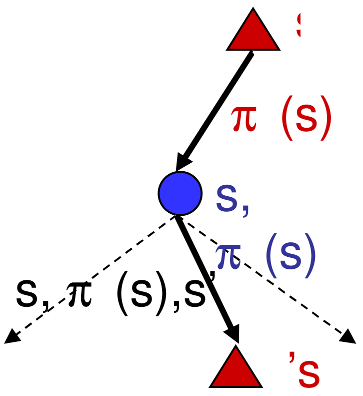
  - Lesson: actions are easier to select from Q's!

# Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- Define the utility of a state s, under a fixed policy π :

  $V^{\pi}(s)$ = expected total discounted rewards (return) starting in s and following π




- Recursive relation (one-step look-ahead / Bellman equation):

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

# Policy Evaluation

- How do we calculate the V's for a fixed policy?

- Idea one: turn recursive equations into updates

$$V_0^\pi(s) = 0$$

$$V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^\pi(s')]$$

- Idea two: it's just a linear system, solve with Matlab (or whatever)

# Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge faster under some conditions

# Policy Iteration

- Policy evaluation: with fixed current policy $\pi$ , find values with simplified Bellman updates:
  - Iterate until values converge

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

# Comparison

- Both VI and PI compute the same thing (optimal values for all states)
- In value iteration:
  - Every pass (or "backup") updates both utilities (explicitly, based on current utilities) and policy (implicitly, based on current utilities)
  - Tracking the policy isn't necessary; we take the max

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_i(s') \right]$$

- In policy iteration:
  - Several passes to update utilities with fixed policy
  - After policy is evaluated, a new policy is chosen
- Both are dynamic programs for solving MDPs

# Asynchronous Value Iteration*

- In value iteration, we update every state in each iteration
- Actually, *any* sequences of Bellman updates will converge if every state is visited infinitely often
- In fact, we can update the policy as seldom or often as we like, and we will still converge
- Idea: Update states whose value we expect to change: If $|V_{i+1}(s) - V_i(s)|$ is large then update predecessors of s