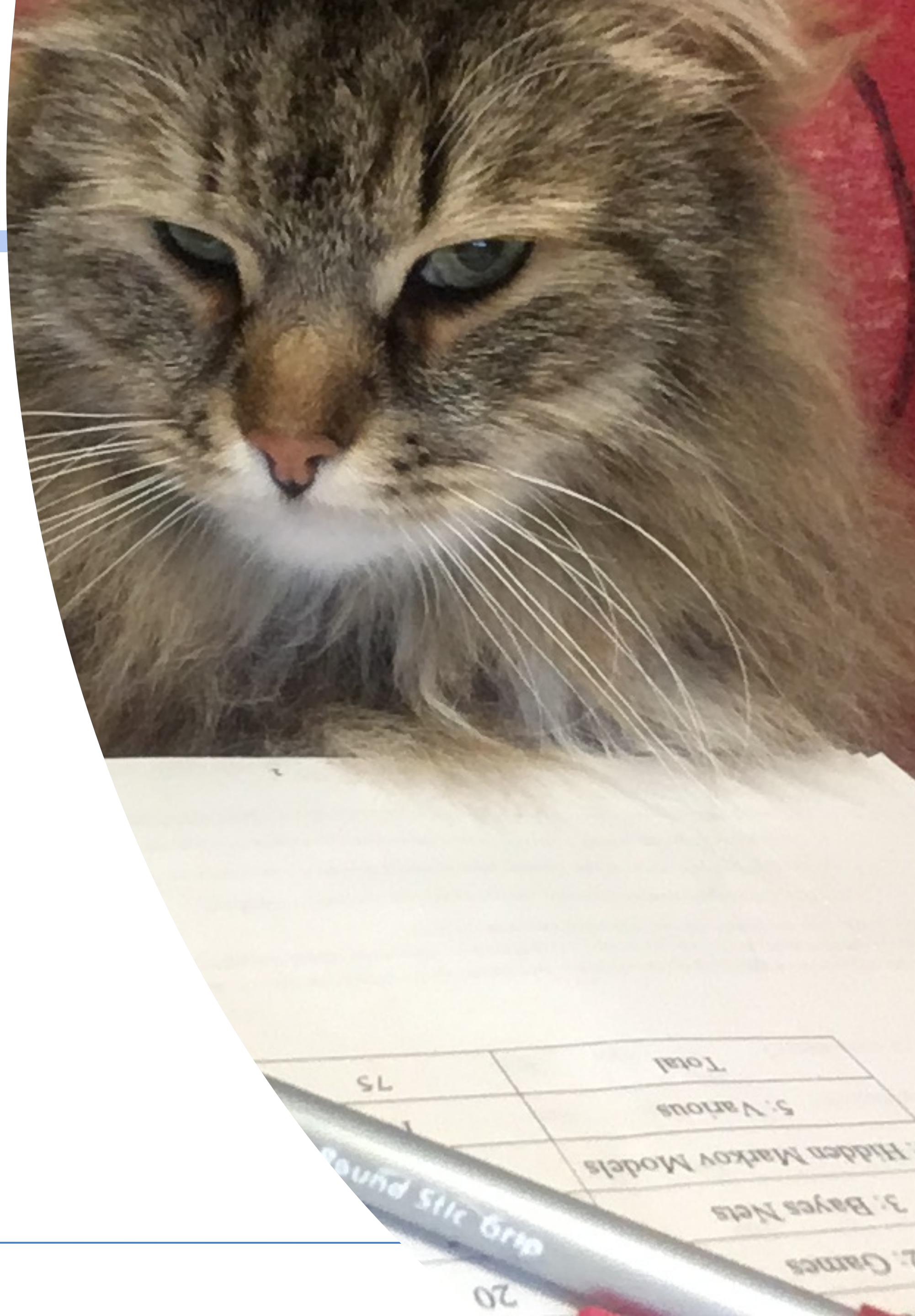
Sequential Decisions With Uncertainty: MDP (1)

With slides from Peter Abbiel, Dan Klein, and Percy Liang

Midterms Graded Posted in Canvas

- Average: 68.85 / 80 (86%)
- Curve: n/a



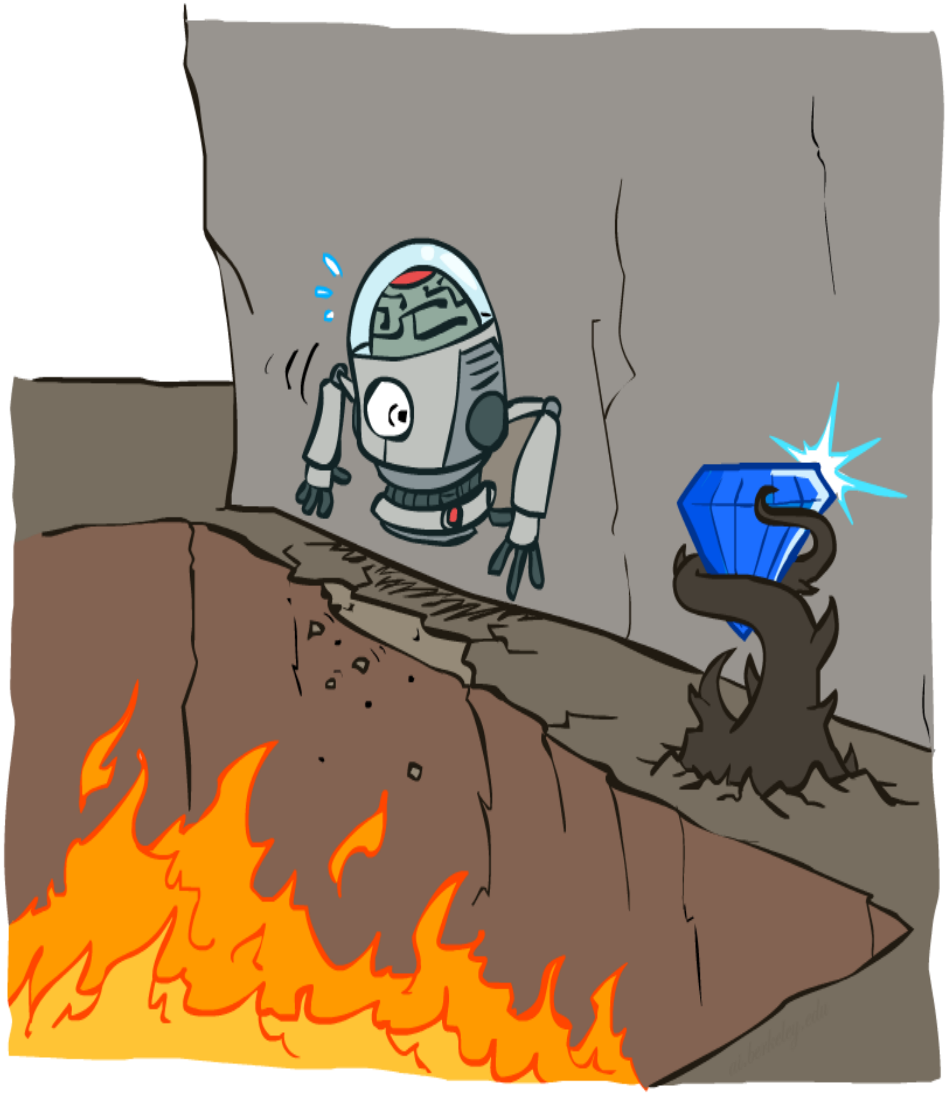
Big Picture

- AI as Planning:
  - Model of the world known (utilities, action outcomes)
  - Deterministic search: UCS, A*, MiniMax
  - Non-deterministic search → ExpectiMax
  - Inference under uncertainty: BNs, HMMs
- AI as Learning:
  - Model of world *partially* known (rewards? outcomes?)
    → Markov Decision Processes (Today)
  - Rewards, action outcomes unknown
    → Reinforcement Learning

Rough Plan (Next 3 weeks)

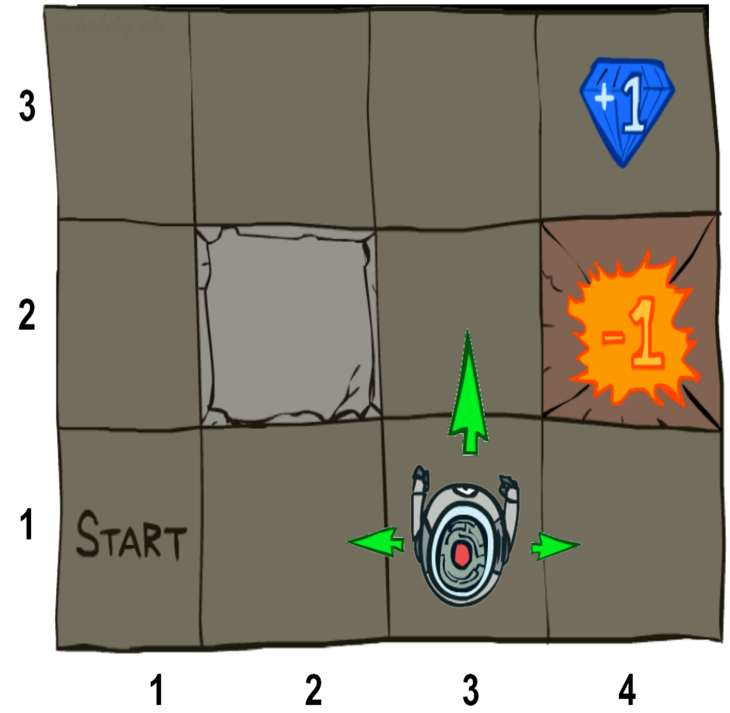
- Markov Decision Processes (MDPs)
  - MDP formalism
  - Solution: Value Iteration and Policy Iteration
- Reinforcement Learning (RL)
  - Relationship to MDPs
  - Several learning algorithms
  - RL applications to games, “real world”

Non-Deterministic **Actions**



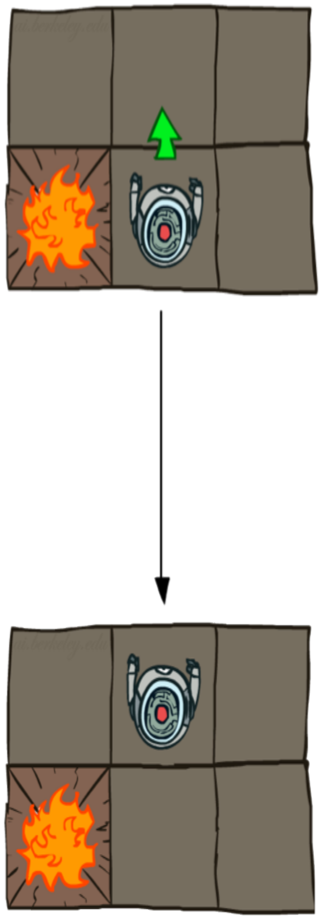
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- <u>**Noisy movement**</u>: 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
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize the <u>sum of rewards</u>

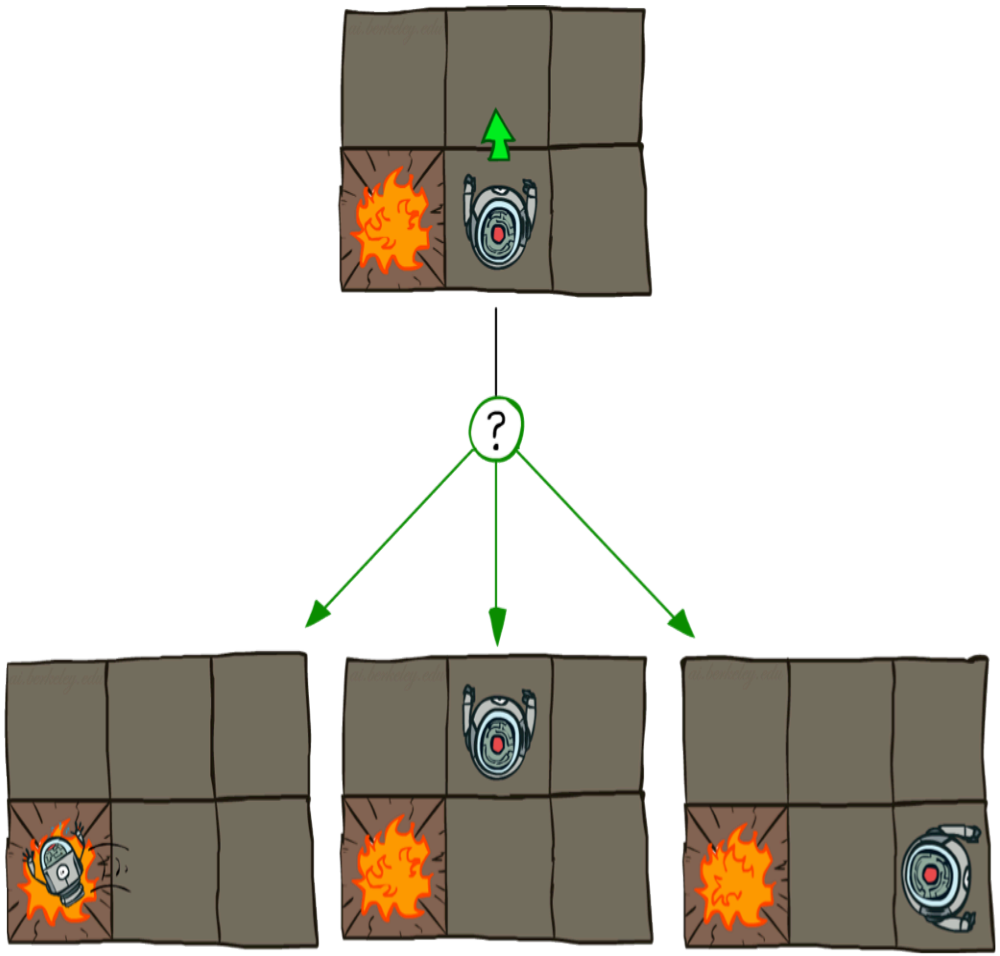


Grid World Actions

Deterministic Grid World

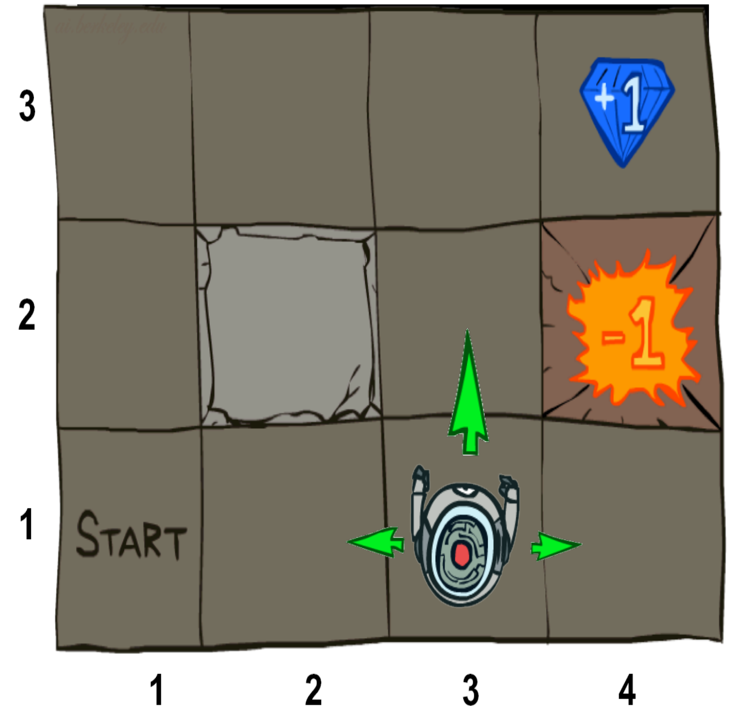


Stochastic Grid World



New Idea: Markov Decision Process (MDP)

- An MDP is defined by:
  - Set of states $s \in S$
  - Actions $a \in A$
  - Transition function $T(s, a, s')$
    - Probability that a from s leads to s', i.e., $P(s' | s, a)$
    - Also called the model or the dynamics
  - Reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - Start state (s_0)
  - Terminal state (optional)



- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll develop a better tool

What is “Markov” about MDPs?

- Remember: “Markov” means that given the preser state, the future and the past are independent

- Markov decision processes, “Markov” means actio outcomes depend only on the current state

Andrey Markov
(1856-1922)

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

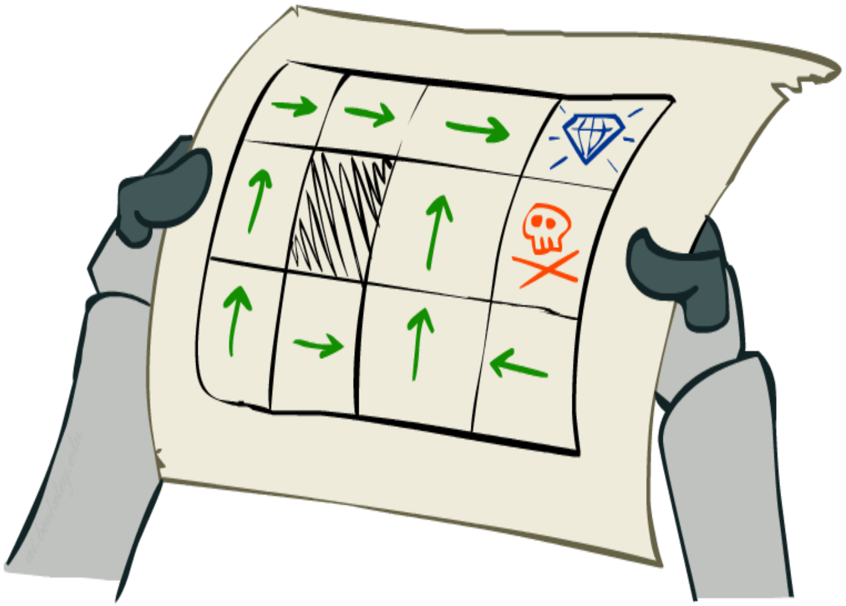
- Like (H) MMs, where the successor function only depends on current state (not the full history)

Definition: **Policy**

- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

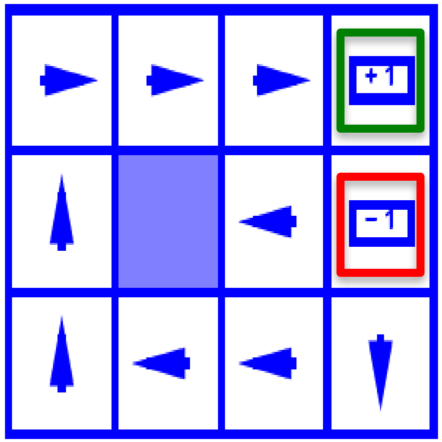
- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

- Minimax/Expectimax did NOT compute entire policies
  - **They computed the action for a single state only (and then re-planned)**

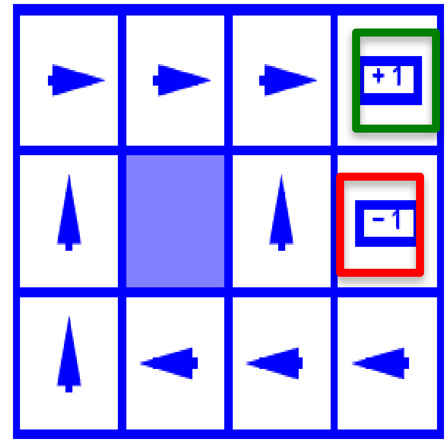


Optimal policy when $R(s, a, s') = -0.03$ for all non-terminals s

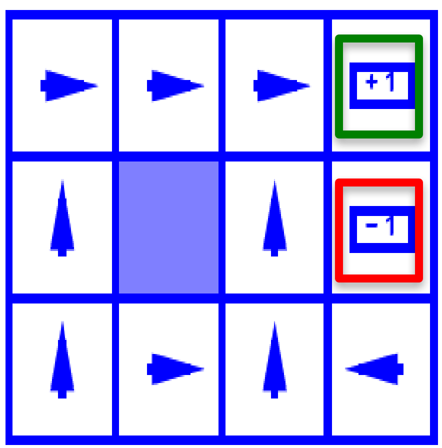
Optimal Policies



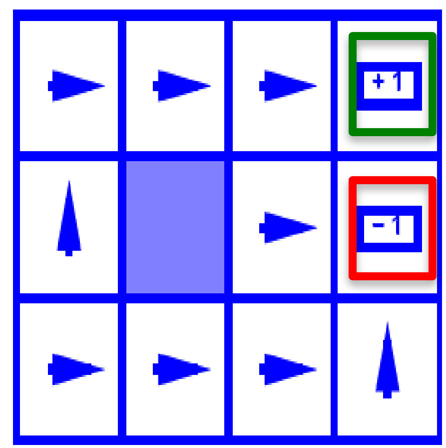
R(s) = -0.01



R(s) = -0.03

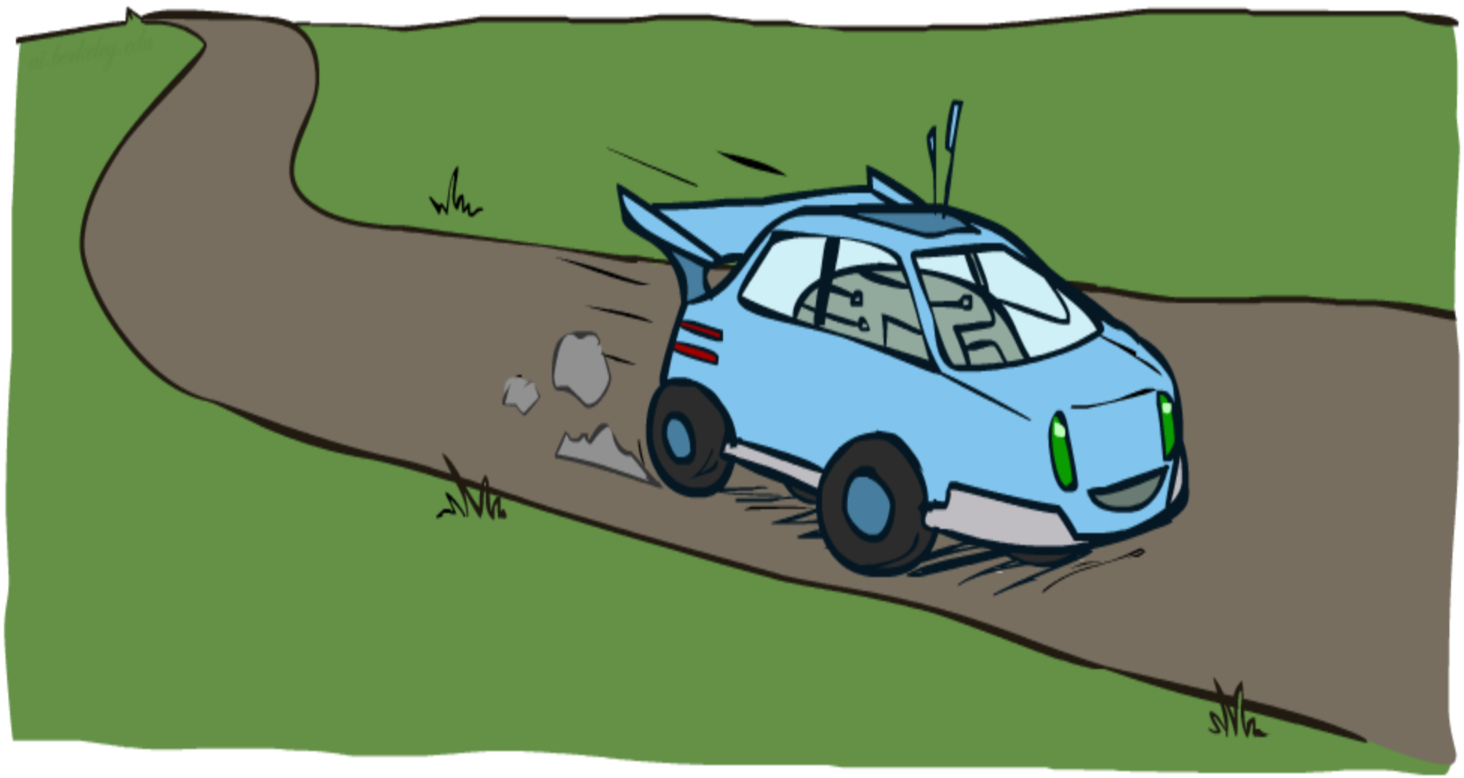


R(s) = -0.4



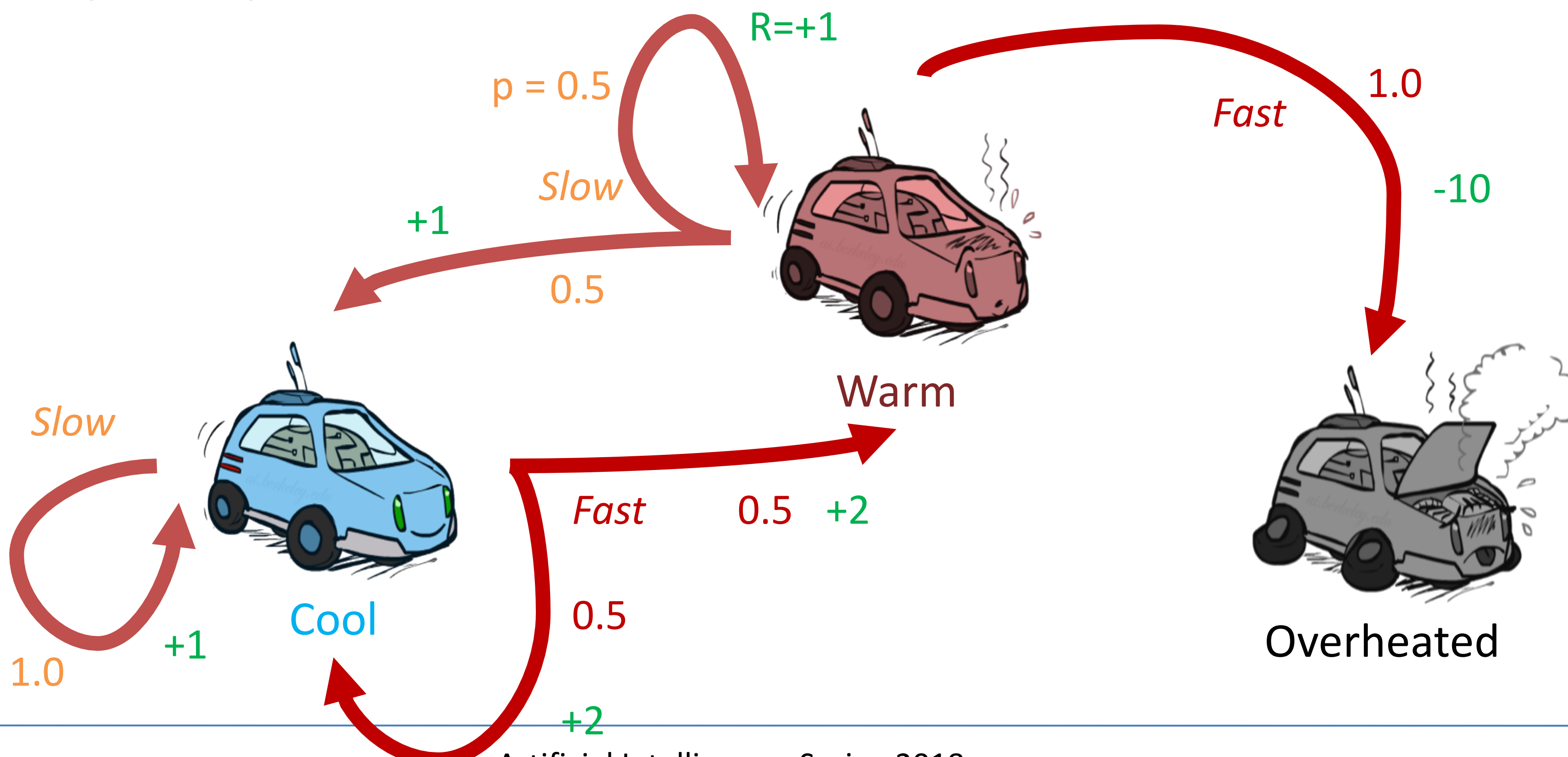
R(s) = -2.0

Example: Racing

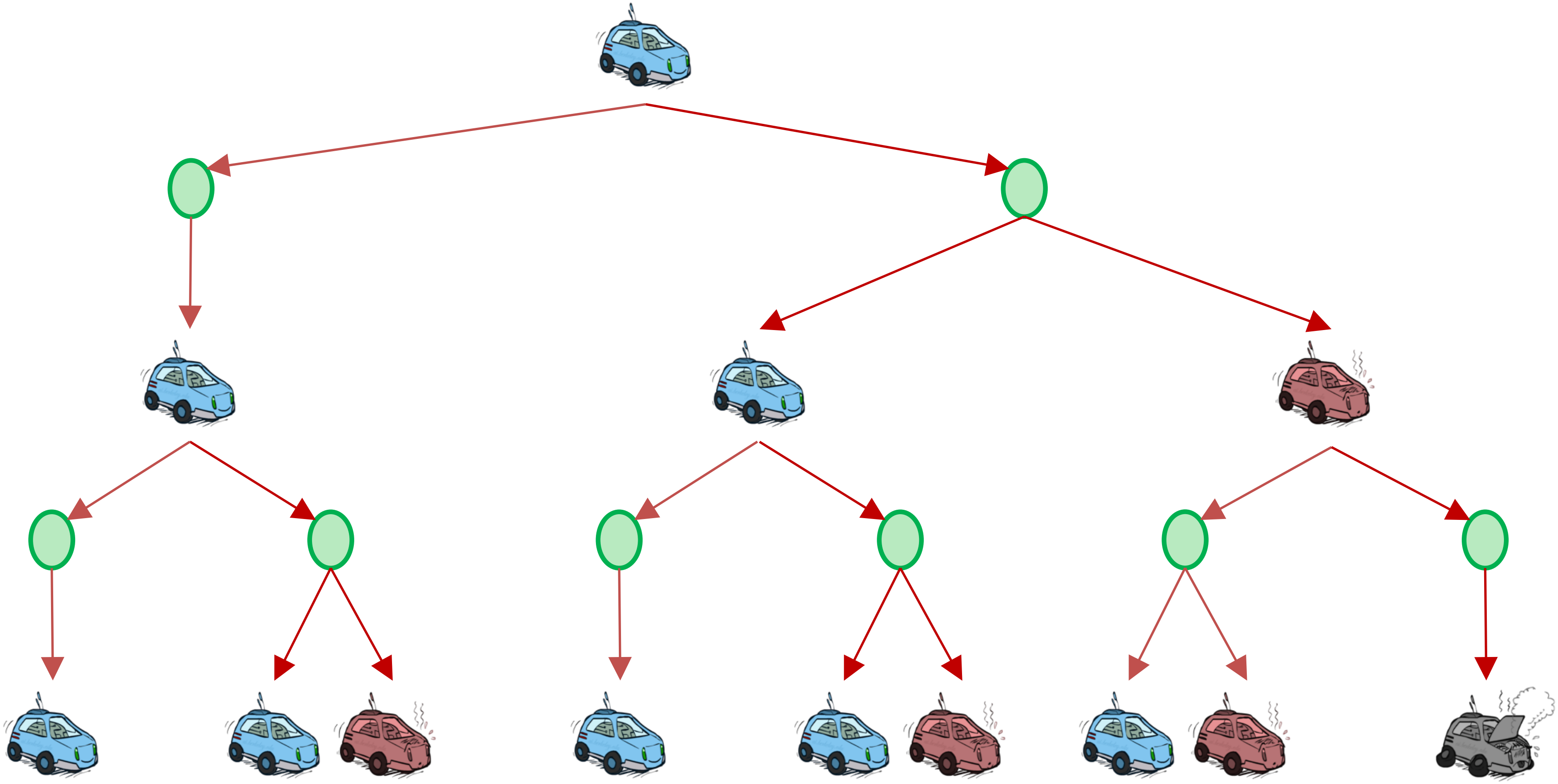


Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



Racing Search Tree (infinite depth ☹)



... potentially infinite depth …

Quit/Stay Game

For each round $r = 1, 2, \ldots$

- You choose stay or quit.
- If quit, you get $\$10$ and we end the game.
- If stay, you get $\$4$ and then I roll a 6-sided dice.
  - If the dice results in 1 or 2, we end the game.
  - Otherwise, continue to the next round.

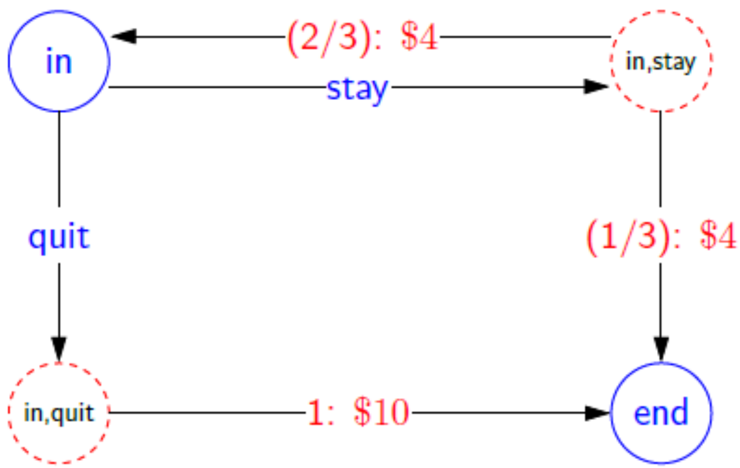
Start Stay Quit

Dice: Rewards: 0

MDP **DIAGRAM** for Quit/Stay Game

For each round $r = 1, 2, \ldots$

- You choose stay or quit.
- If quit, you get $\$10$ and we end the game.
- If stay, you get $\$4$ and then I roll a 6-sided dice.
  - If the dice results in 1 or 2, we end the game.
  - Otherwise, continue to the next round.



Game Transition Probabilities

Example: transition probabilities

s	a	s'	$T(s,a,s')$
in	quit	end	1
in	stay	in	2/3
in	stay	end	1/3

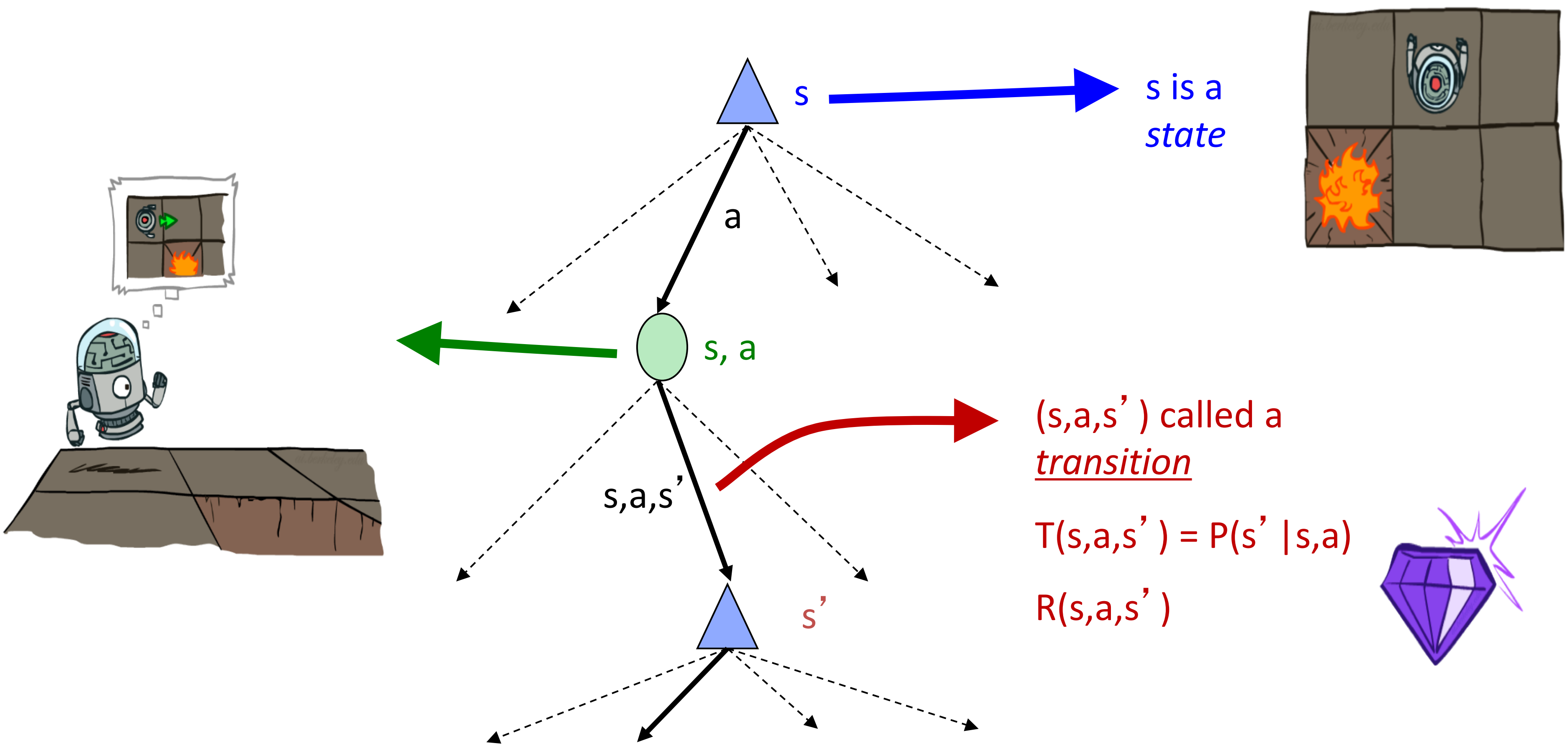
For each state s and action a:

$$\sum_{s' \in \text{States}} T(s,a,s') = 1$$

Successors: s' such that $T(s,a,s') > 0$

MDP Search Trees

- Each MDP state projects an expectimax-like search tree



Utilities of Sequences

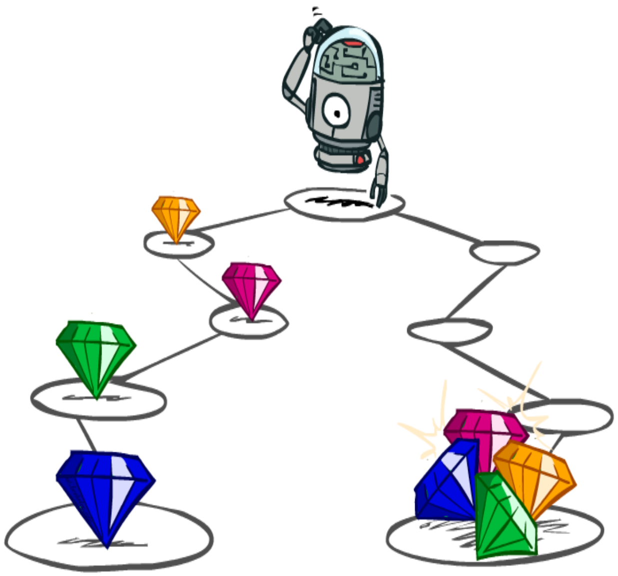
- What preferences should an agent have over reward sequences?

[1, 2, 2] or [2, 3, 4]

- More or less?

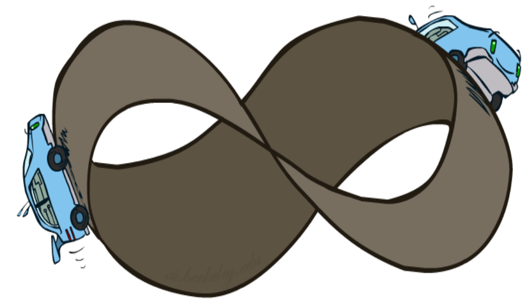
[0, 0, 1] or [1, 0, 0]

- Now or later?



Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?



- Solutions:
  1. Finite horizon: (similar to depth-limited search)
     - Terminate episodes after a fixed T steps (e.g. life)
     - Gives nonstationary policies (π depends on time left)
  2. Discounting: use $0 < \gamma < 1$

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

     - Smaller γ → smaller "horizon" – shorter term focus
  3. Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

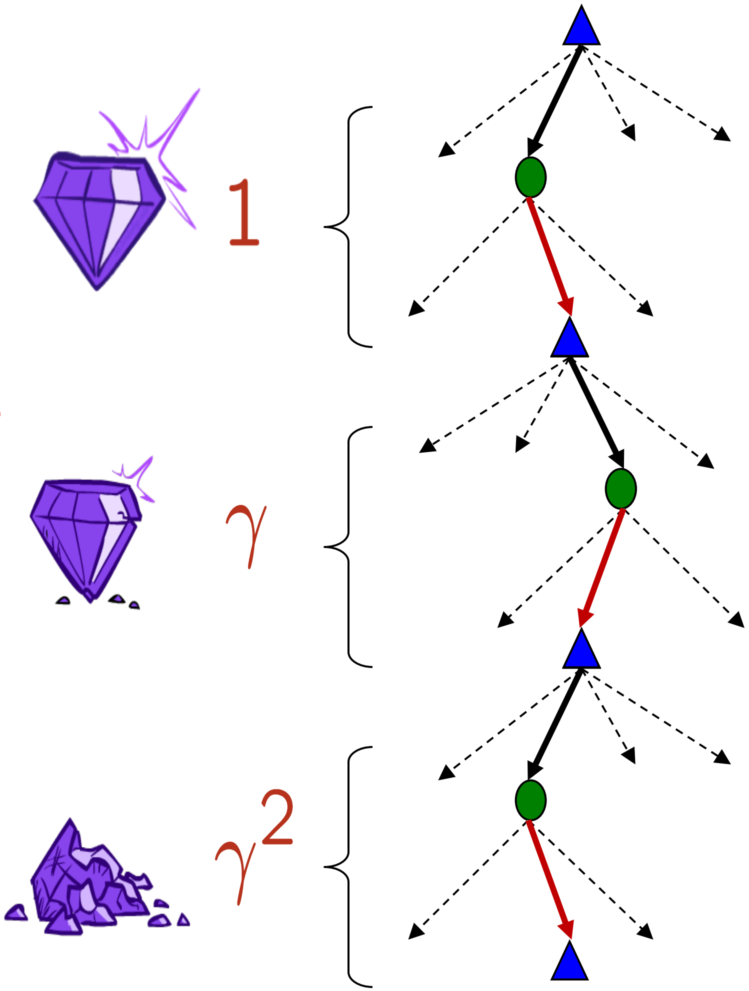
Worth Next Step

γ^2

Worth In Two Steps

Discounting

- How to discount?
  - Each time we descend a level, we multiply by the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our 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])

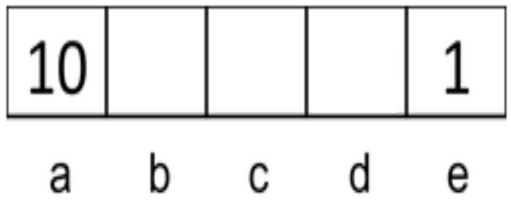


Detour: Temporal/Delay Discounting

- What would you rather have?
  - A. $100 today
  - B. $150 a year from now
- What about:
  - A. $100 in 12 months
  - B. $110 in 13 months
- Humans temporally discount values of rewards
  - https://en.wikipedia.org/wiki/Temporal_discounting
- Delayed gratification: https://www.youtube.com/watch?v=QX_oy9614HQ

Quiz: Discounting

- Given:



  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic (no noise, for now)

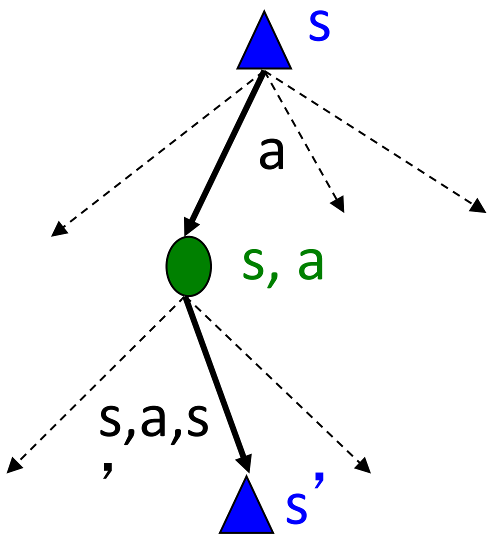
- P 1: For $\gamma = 1$, what is the optimal policy?

- P 2: For $\gamma = 0.1$, what is the optimal policy?

- P 3: For which γ are West and East equally good when in state d?

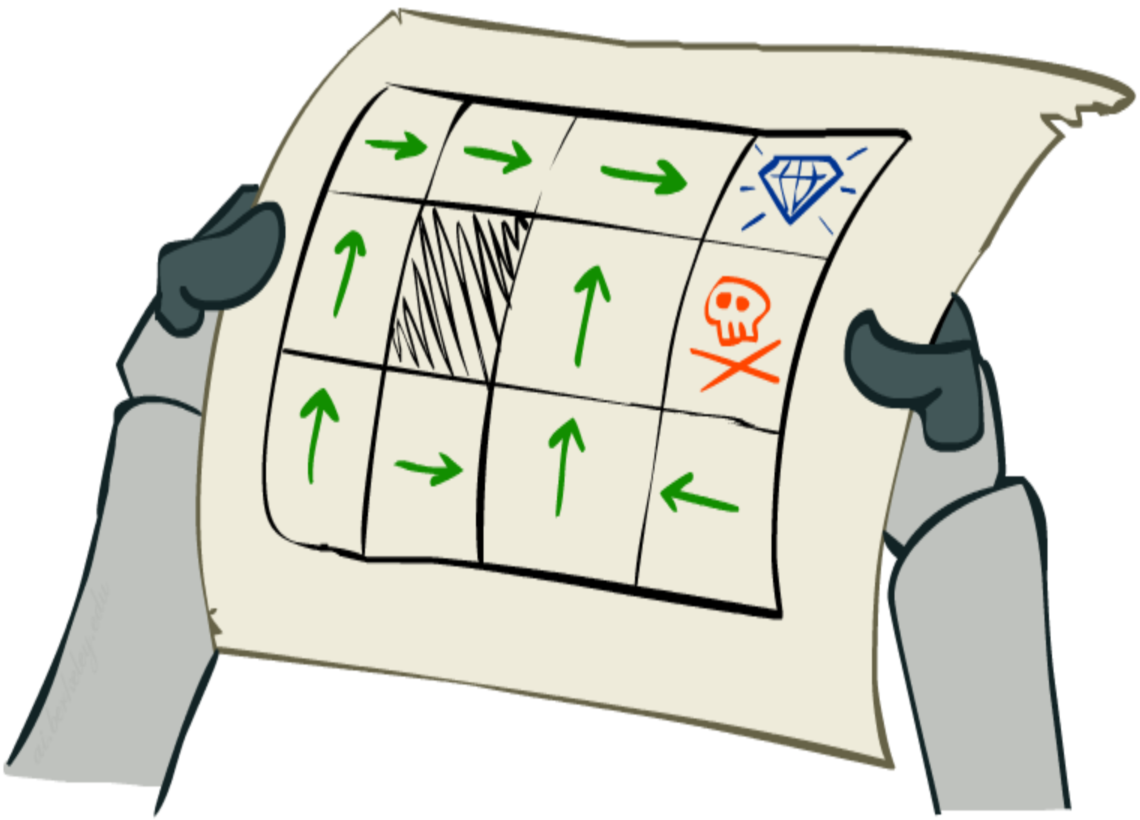
Recap: Defining MDPs

- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

Solving MDPs



Definitions: Optimal Quantities

- The value (utility) of a state s:

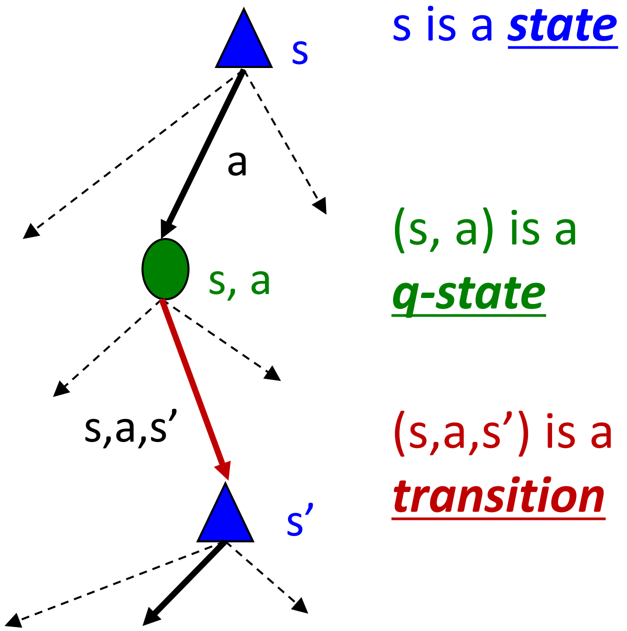
  V^*(s) = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  Q^*(s,a) = expected total **utility** if:
  -- from ***s***, take action ***a***
  -- and act *optimally* after

- The optimal policy:

  π^*(s) = optimal action from state s



Important! Bellman Equations

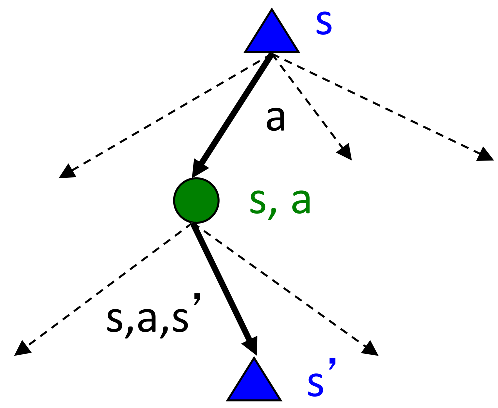
Richard Bellman
These are my equations!

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

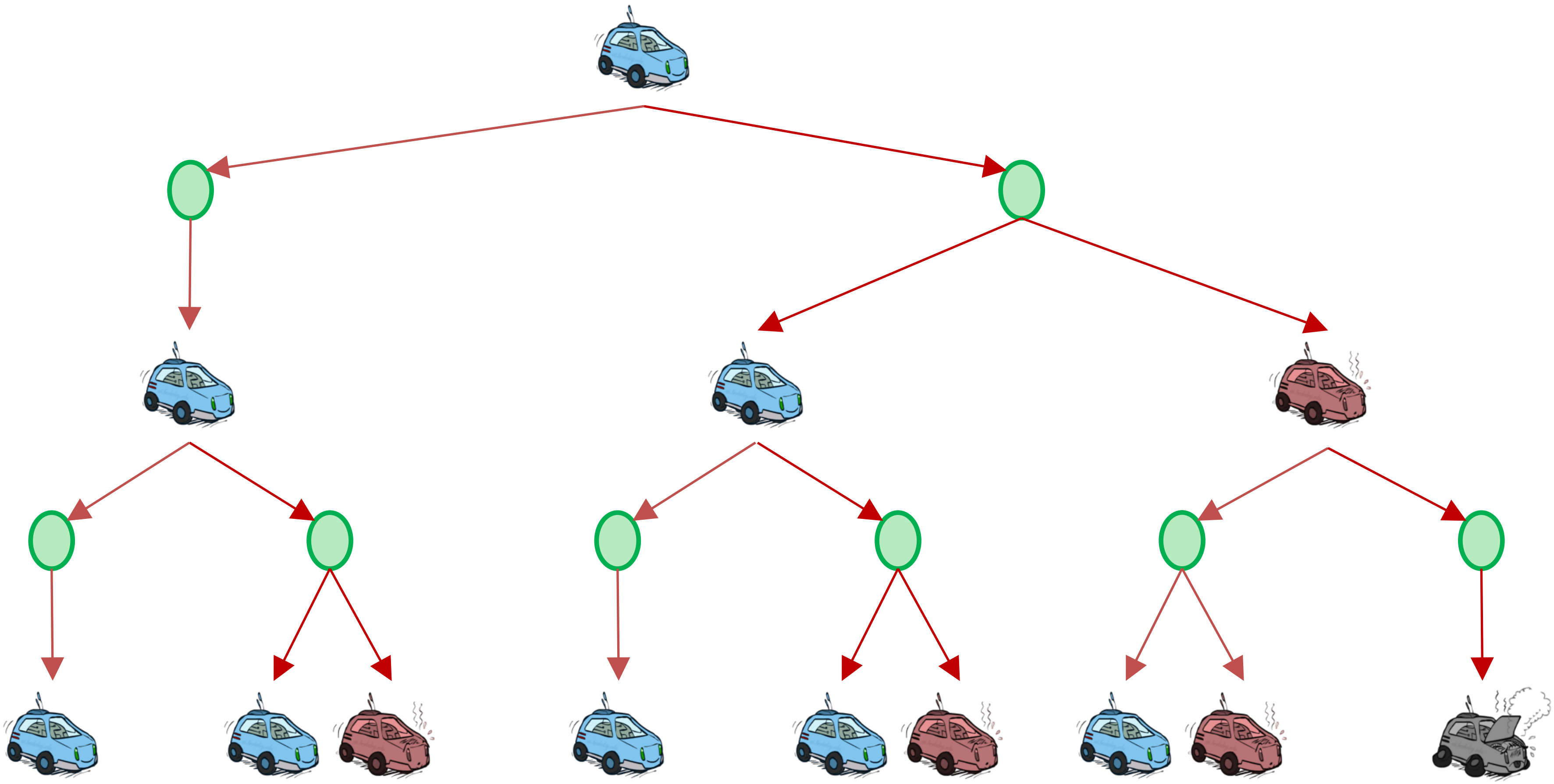
$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

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

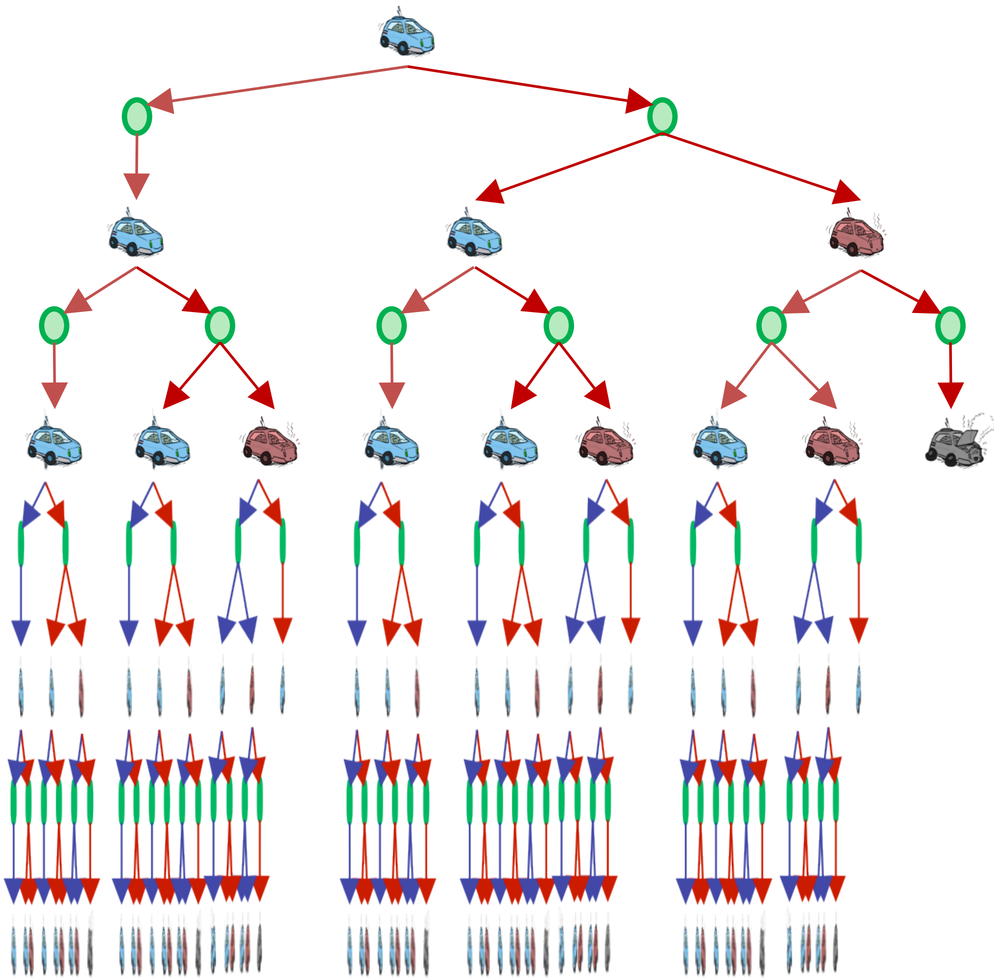


- These are the Bellman equations, and they characterize the **optimal values recurrence**

Example: Racing Search Tree

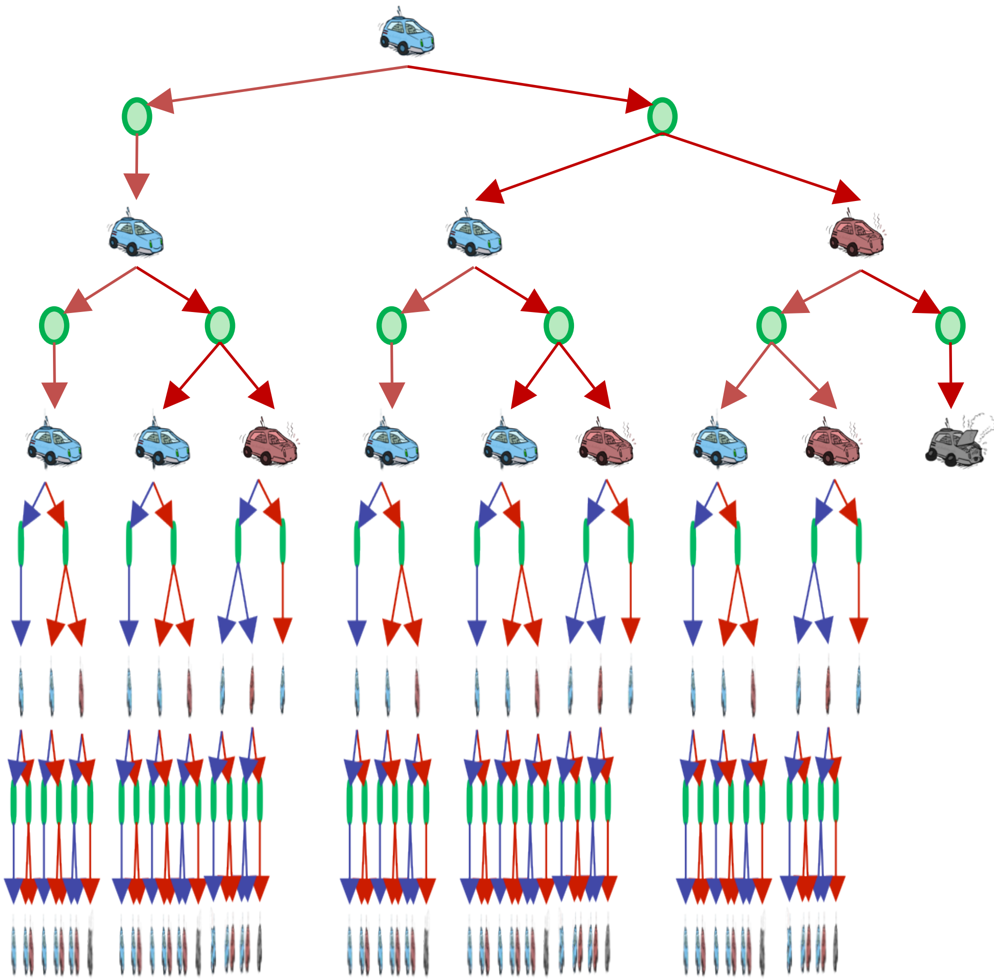


Racing Search Tree

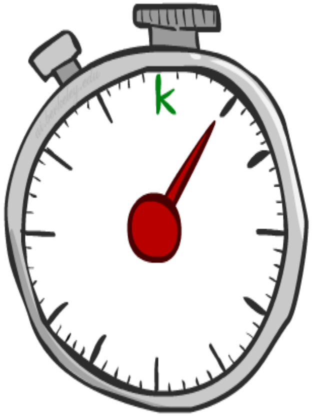


Racing Search Tree

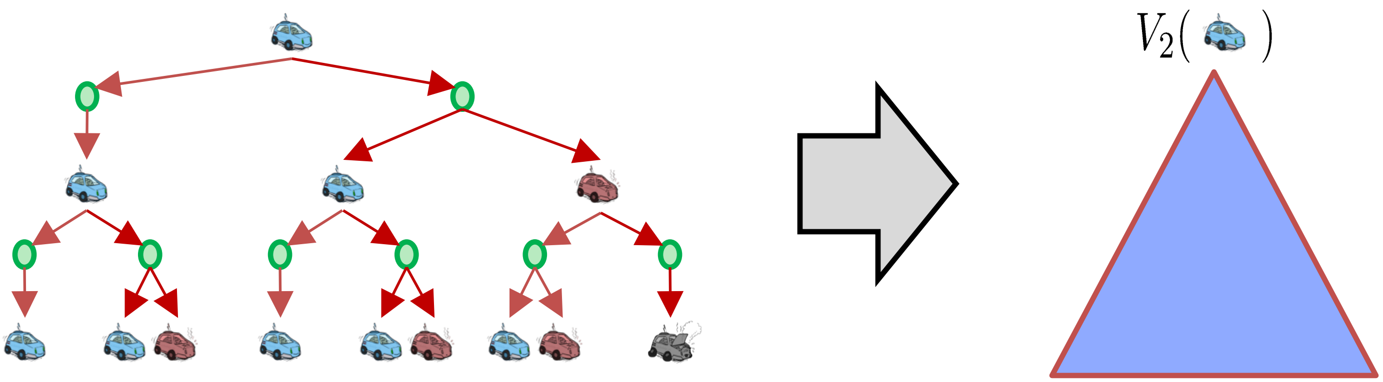
- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter if $\gamma < 1$



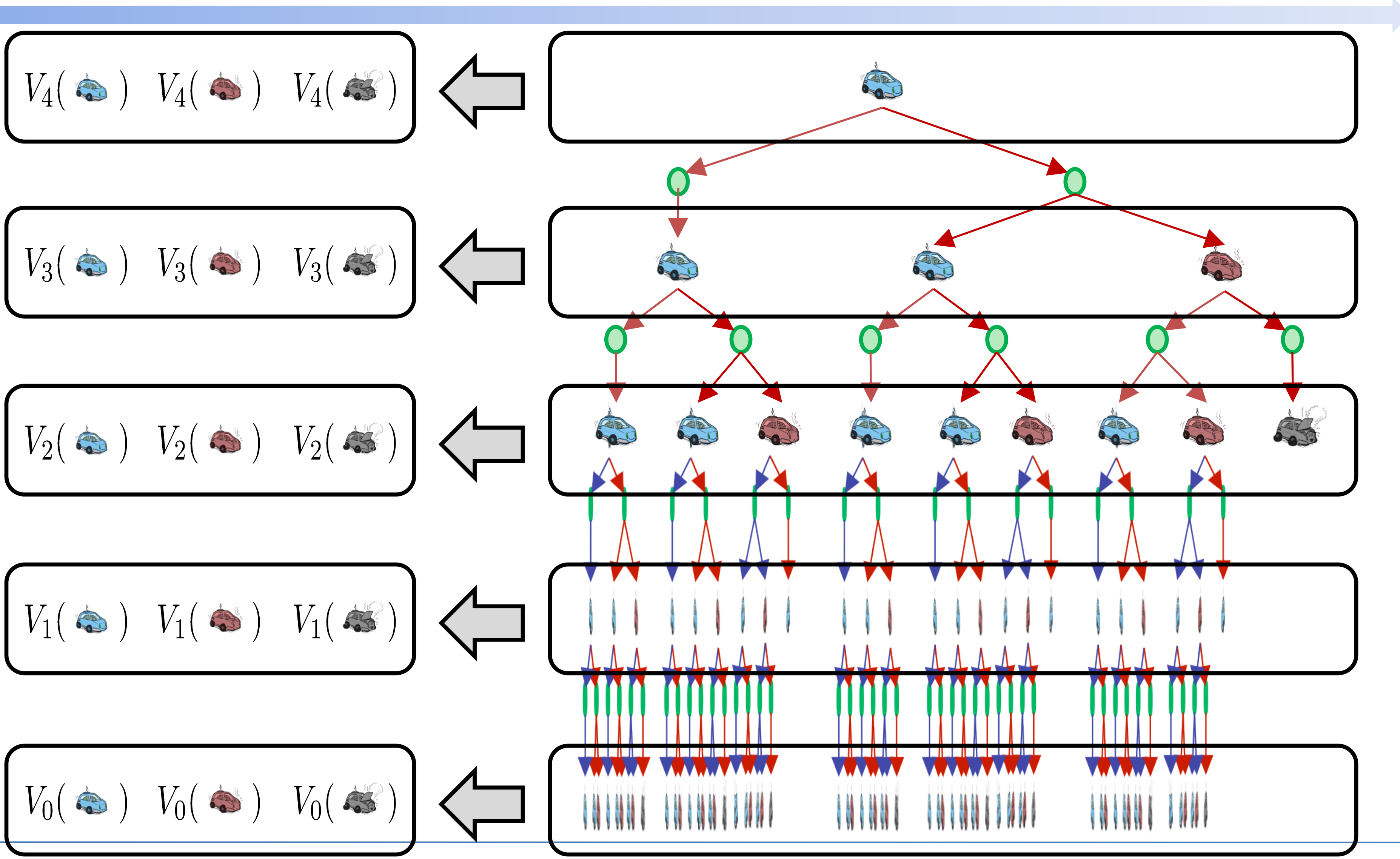
Idea: Time-Limited Values



- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s



Computing Time-Limited Values

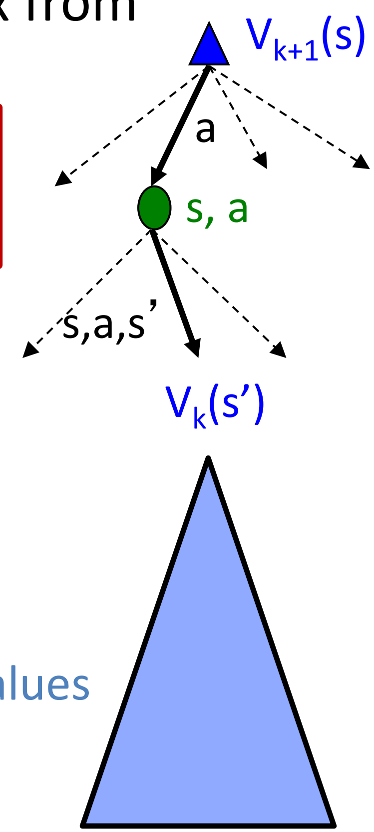


Solution: Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

- Given vector of $V_k(s)$ values, do one ply of ExpectiMax from each state:

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



- Repeat until convergence

- Complexity of each iteration: $O(S^2A)$

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Value Iteration: Algorithm form

Algorithm: value iteration [Bellman, 1957]

Initialize $V_{\text{opt}}^{(0)}(s) \leftarrow 0$ for all states s.

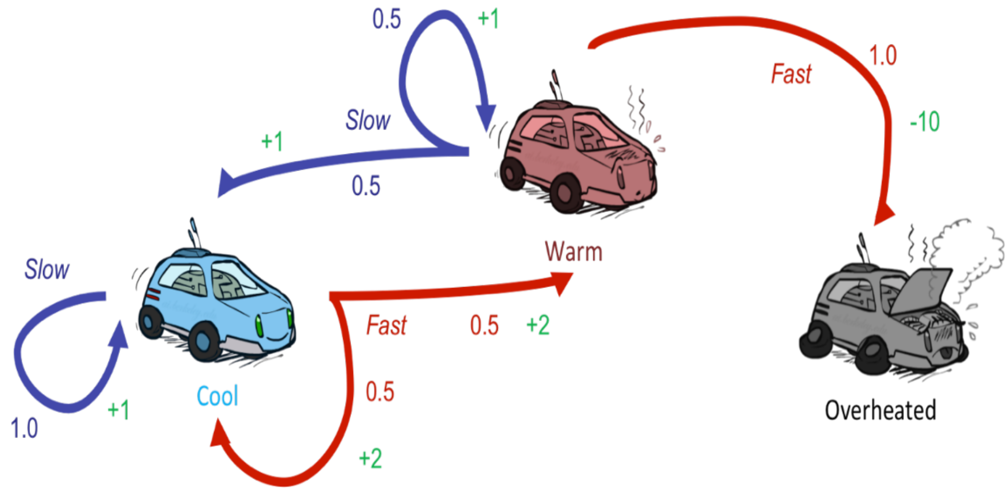
For iteration $t = 1, \ldots, t_{\text{VI}}$:

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}_{Q_{\text{opt}}^{(t-1)}(s,a)}$$

Example: Value Iteration for Race Car

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

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

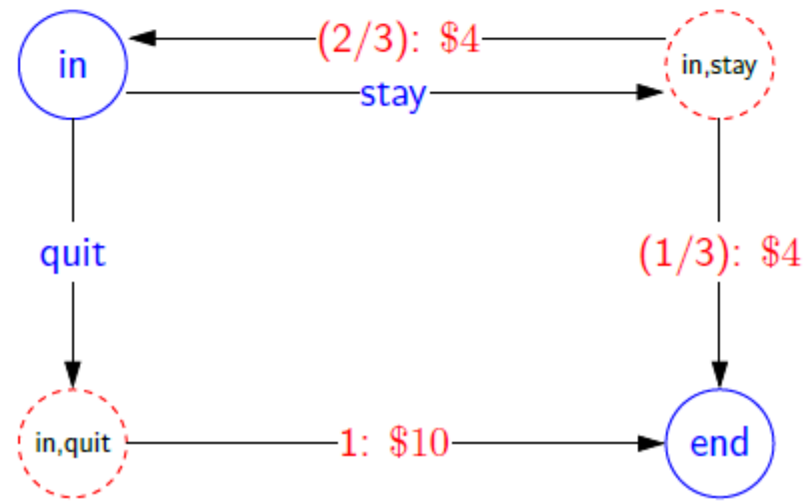
https://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value Iteration for Stay/Quit Game

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	0.00	(t=0 iterations)
$\pi_{\text{opt}}(s)$	-	-	

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}$$



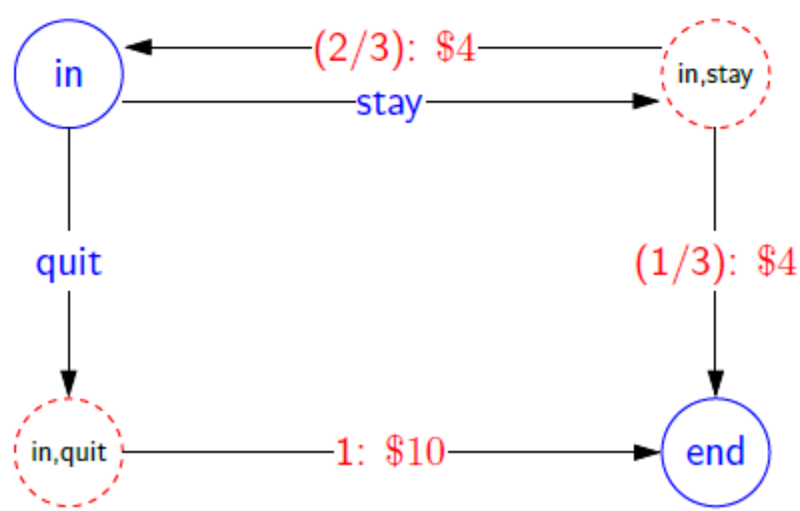
Value Iteration Example: k=1

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	0.00	(t=0 iterations)
$\pi_{\text{opt}}(s)$	-	-	

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	10.00	(t=1 iterations)
$\pi_{\text{opt}}(s)$	-	**quit**	

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}$$



Calculation: on board

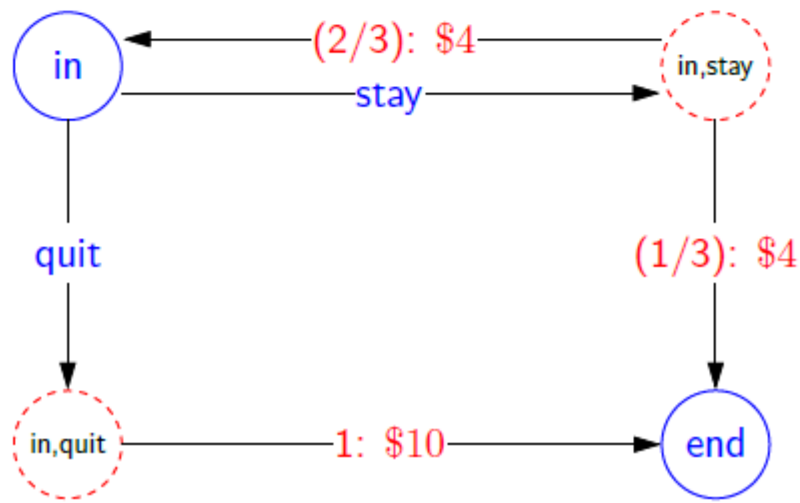
Value Iteration Example: k=2

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	10.00	(t=1)
$\pi_{\text{opt}}(s)$	-	quit	

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	10.67	(t=2)
$\pi_{\text{opt}}(s)$	-	stay	

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]$$



Calculation: on board

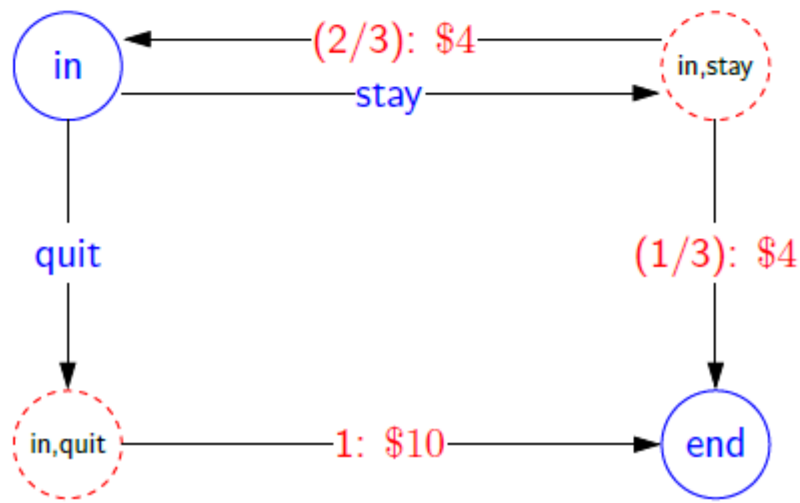
Value Iteration Example: k=3

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	10.67	(t=2)
$\pi_{\text{opt}}(s)$	-	stay	

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	**11.11**	(t=3)
$\pi_{\text{opt}}(s)$	-	stay	

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}$$



Calculation: on board

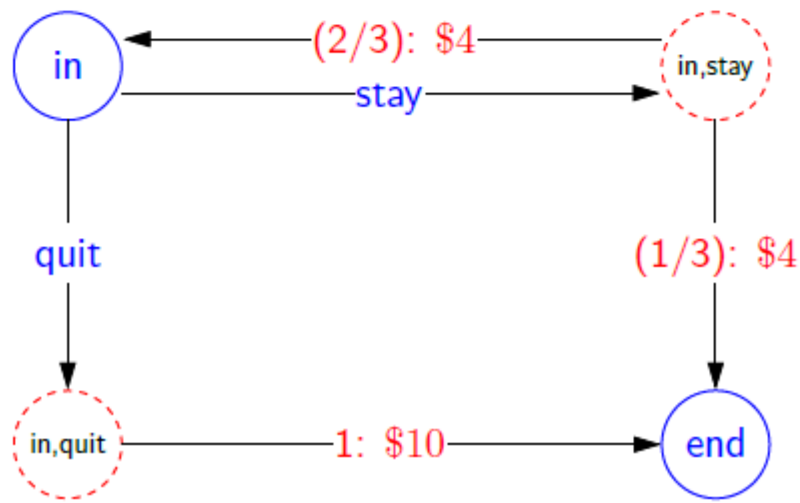
Value Iteration Example: k=100

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	11.9998	(t=99)
$\pi_{\text{opt}}(s)$	-	stay	

s	end	in	
$V_{\text{opt}}^{(t)}$	0.00	**12**	(t=100)
$\pi_{\text{opt}}(s)$	-	stay	

For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]$$



Calculation: on board

MDP for Grid World

Set of states $s \in S$: positions

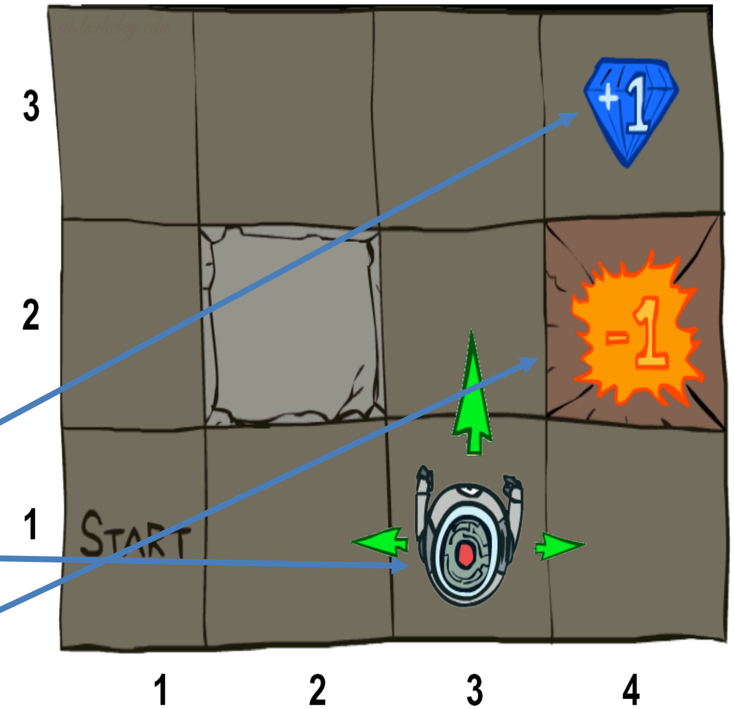
Actions $a \in A$: N, S, E, W

Transition function $T(s, a, s')$: given
"noise": probability of not following ***a***

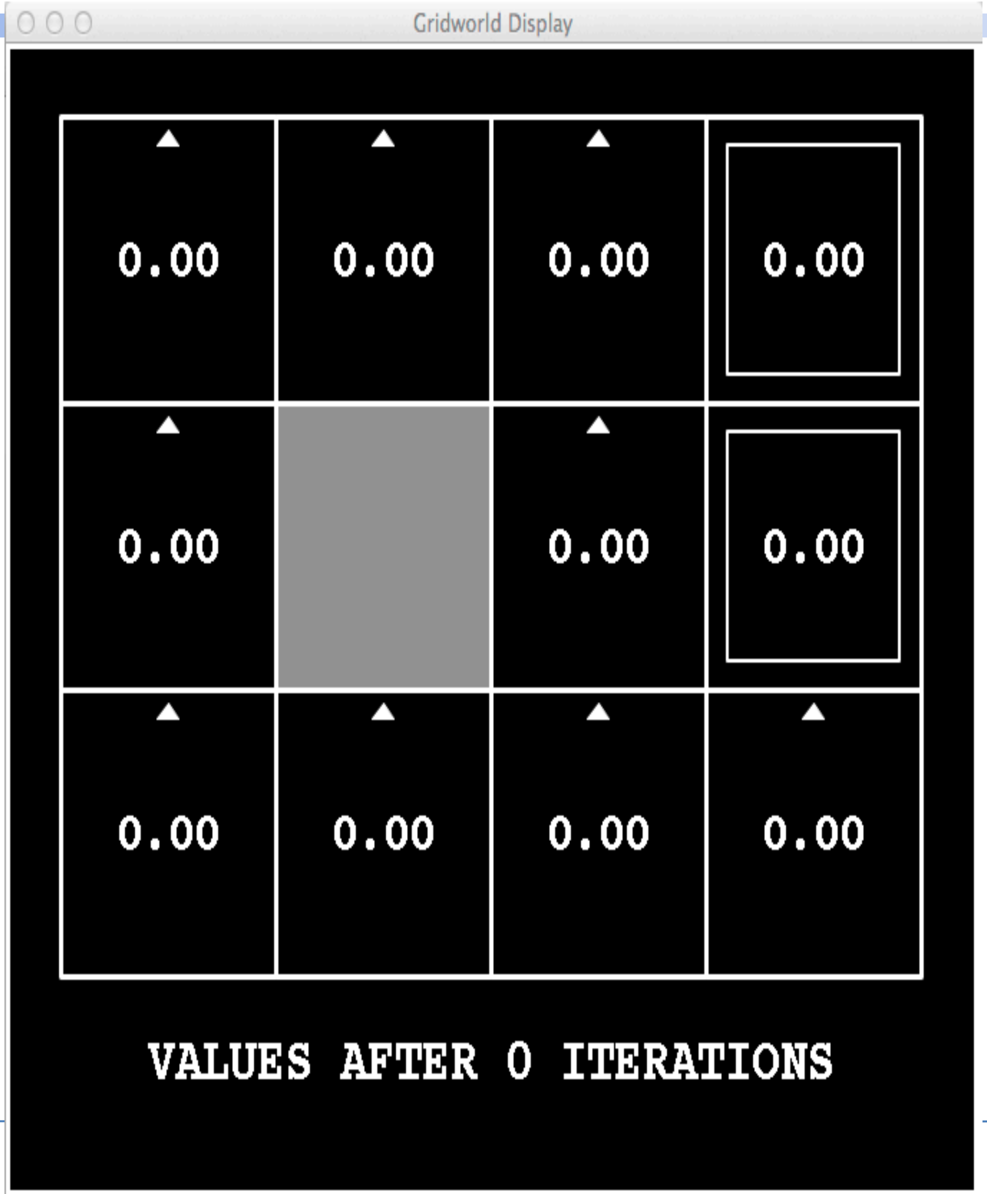
Reward function $R(s, a, s')$: given

Start state (s_0)

Terminal states: +1, -1 rewards



Value Iteration for GridWorld, k=0

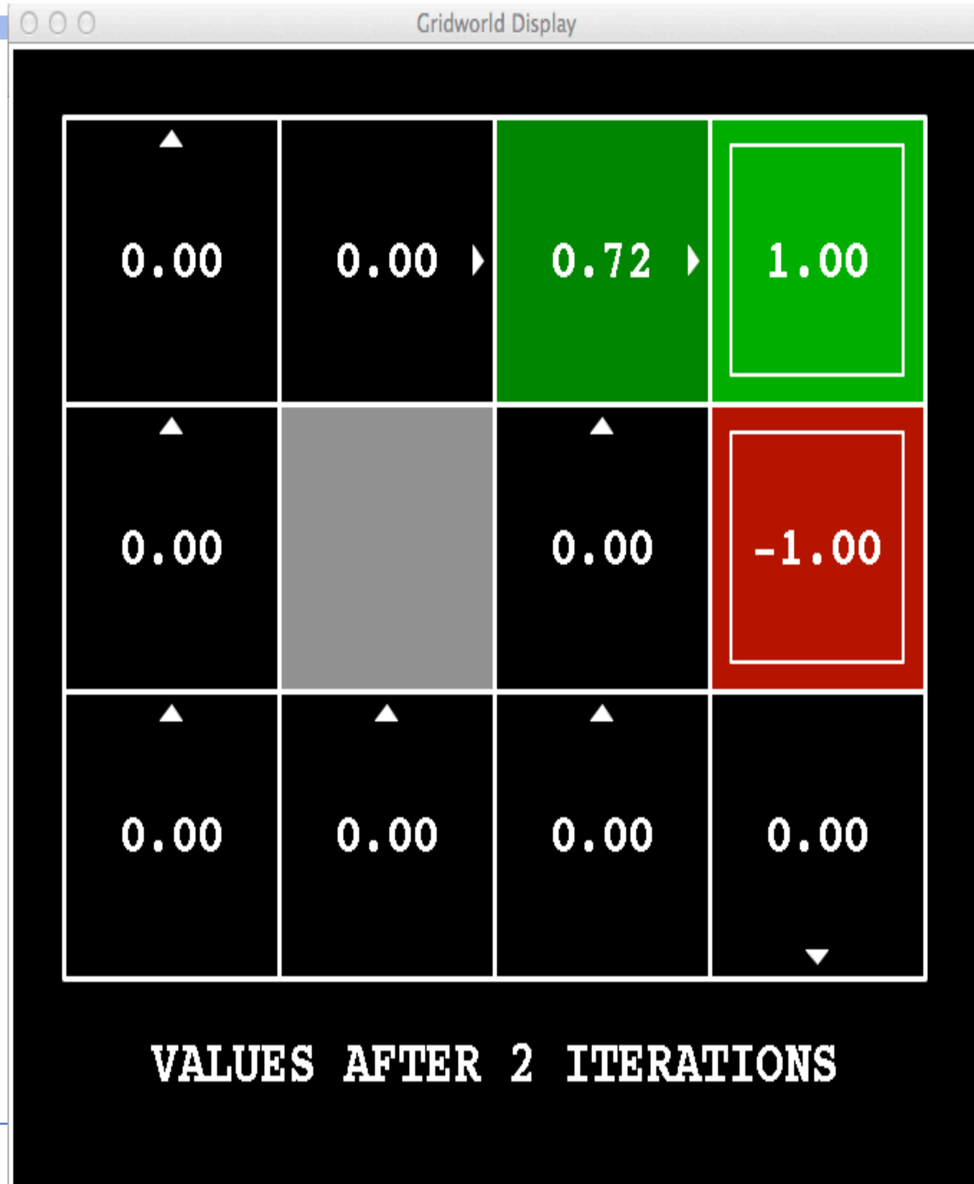


Noise = 0.2
Discount = 0.9
Living reward = 0

k=1

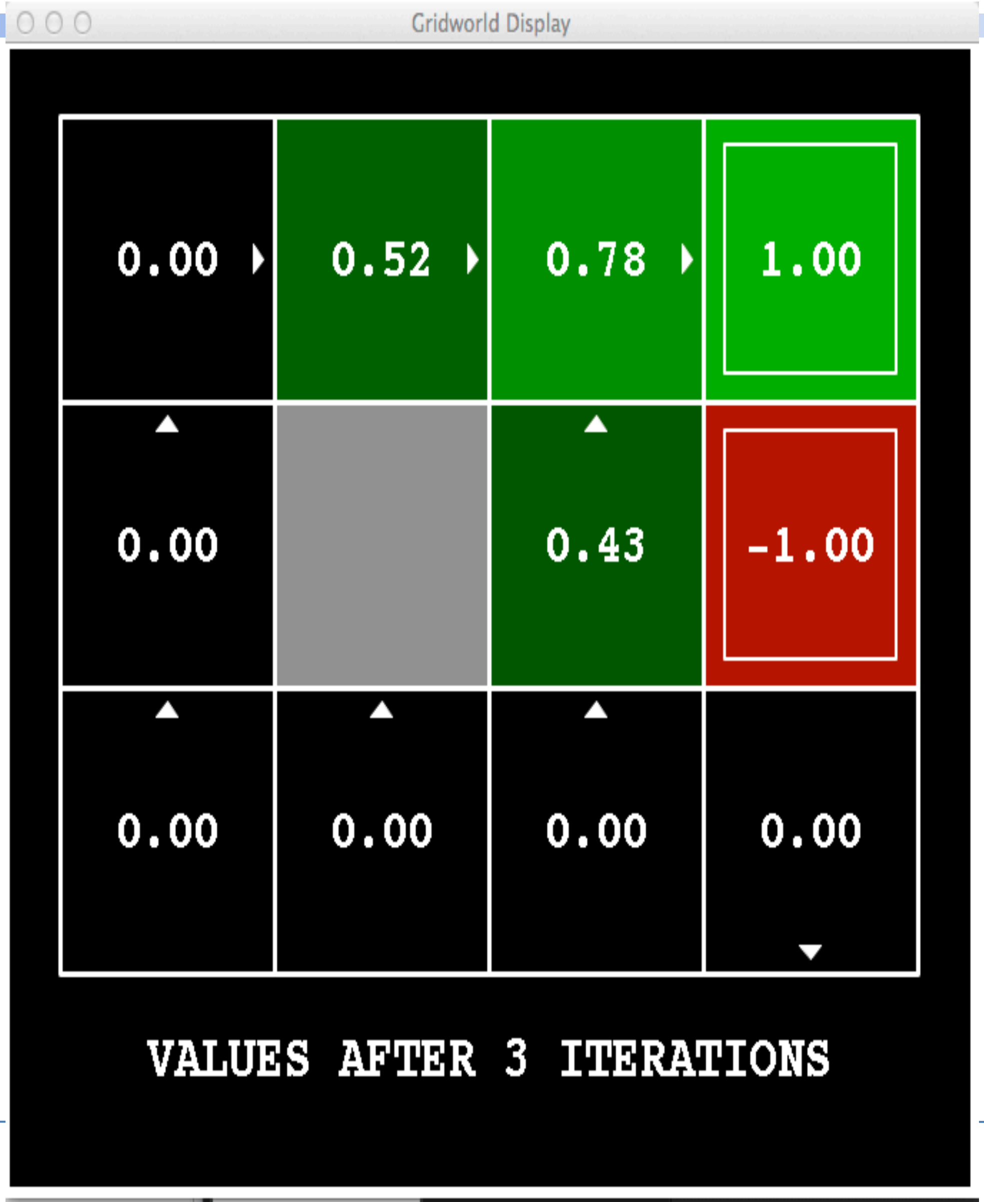
Noise = 0.2
Discount = 0.9
Living reward = 0

k=2



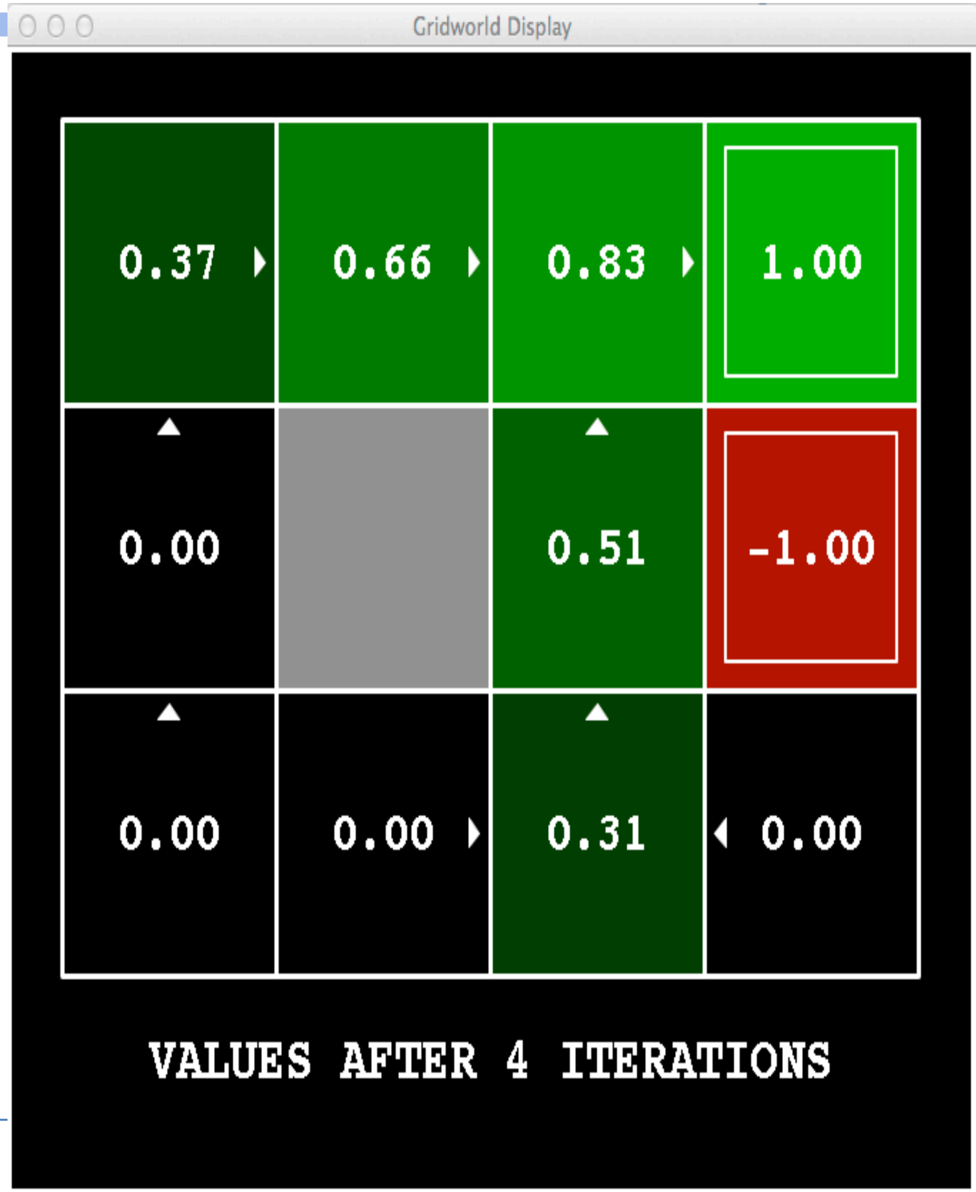
Noise = 0.2
Discount = 0.9
Living reward = 0

k=3



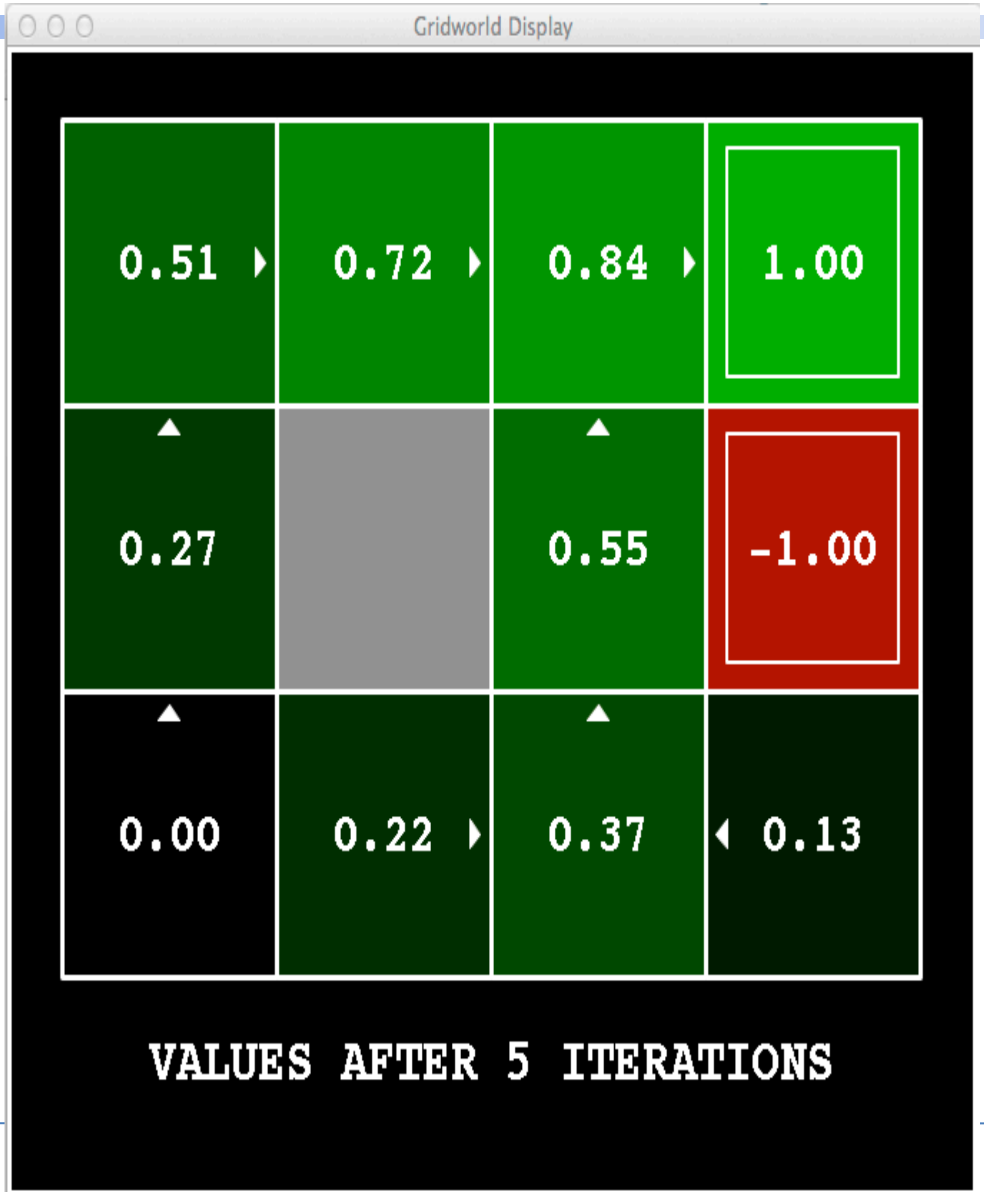
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



Noise = 0.2
Discount = 0.9
Living reward = 0

k=6

Noise = 0.2
Discount = 0.9
Living reward = 0

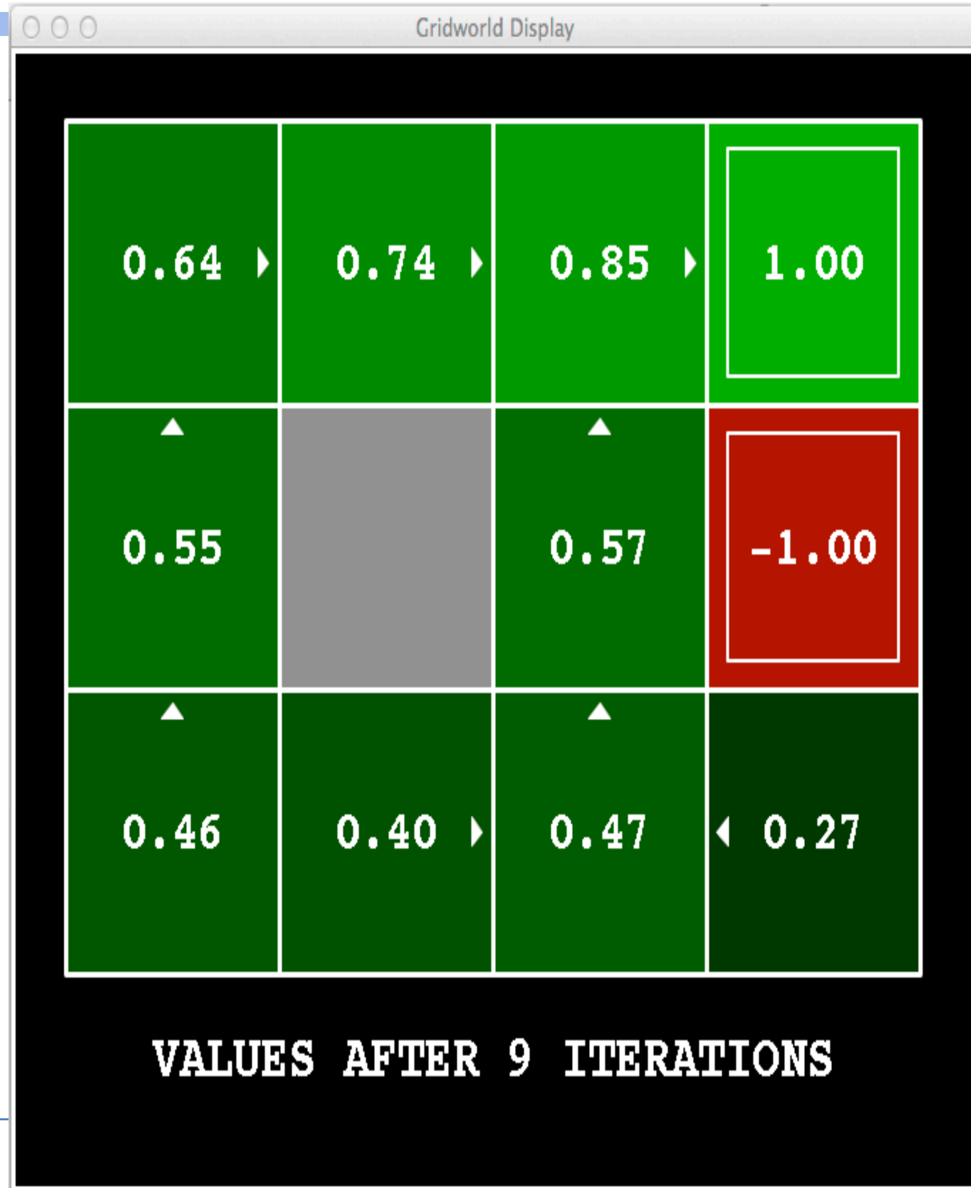
k=7

Noise = 0.2
Discount = 0.9
Living reward = 0

k=8

Noise = 0.2
Discount = 0.9
Living reward = 0

k=9

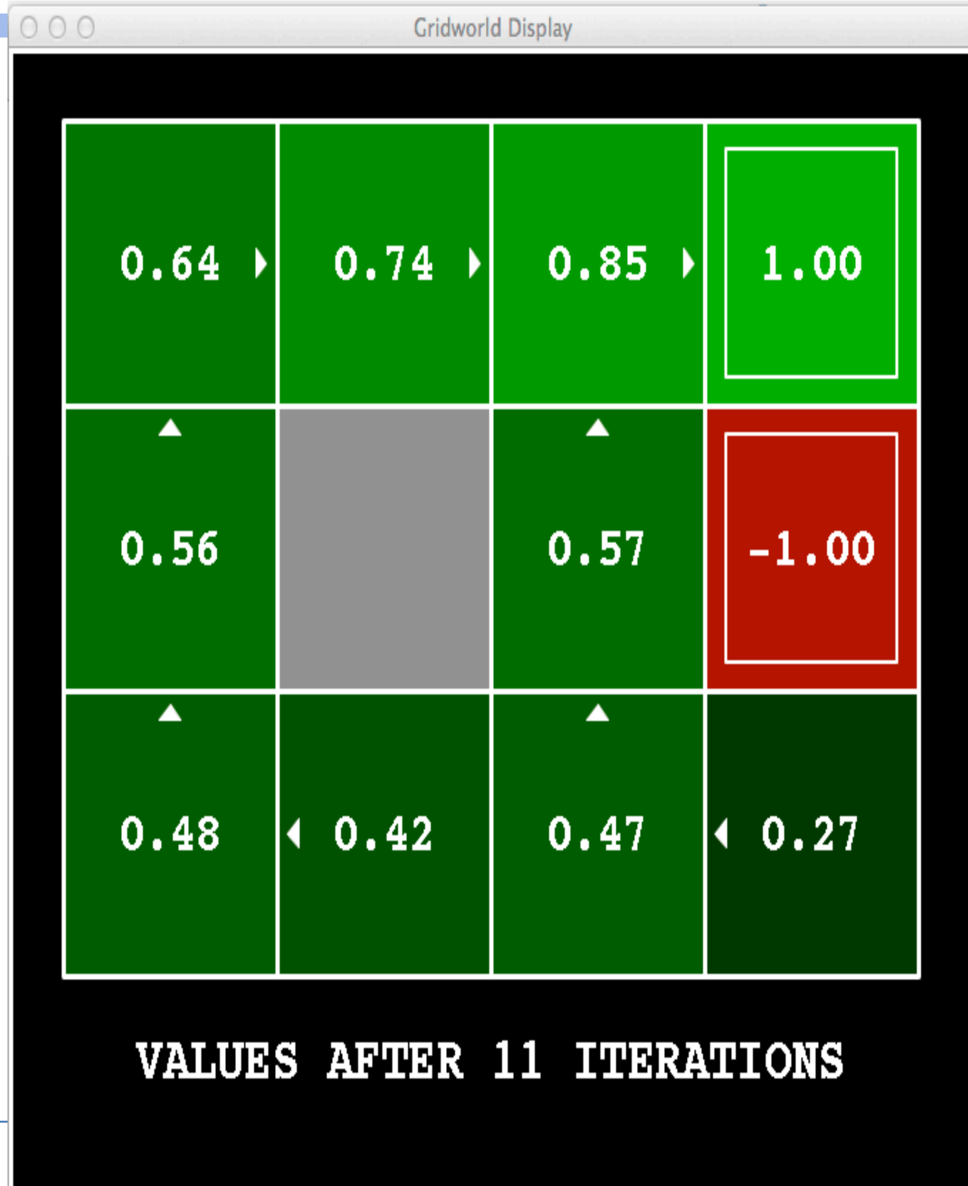


Noise = 0.2
Discount = 0.9
Living reward = 0

k=10

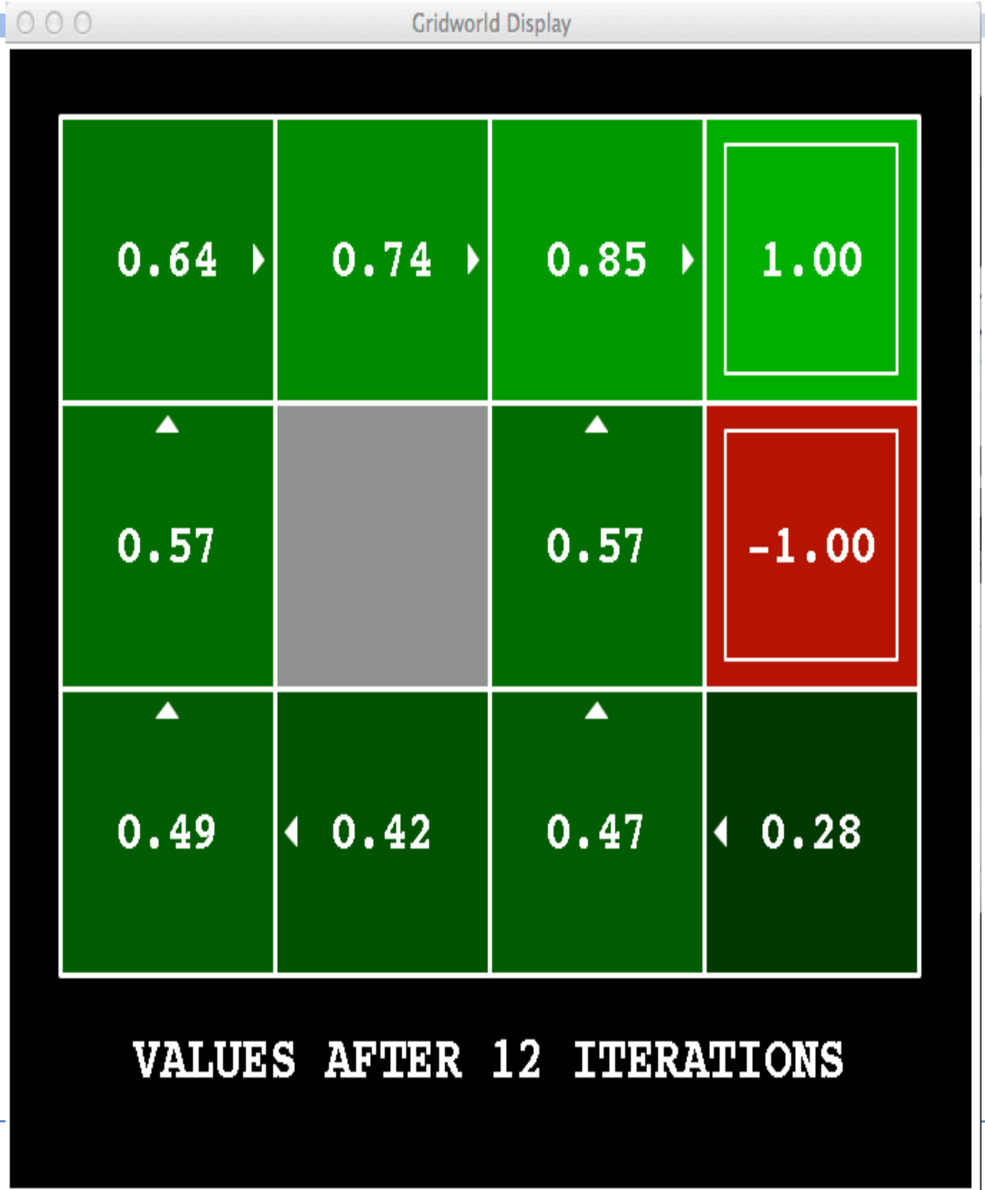
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



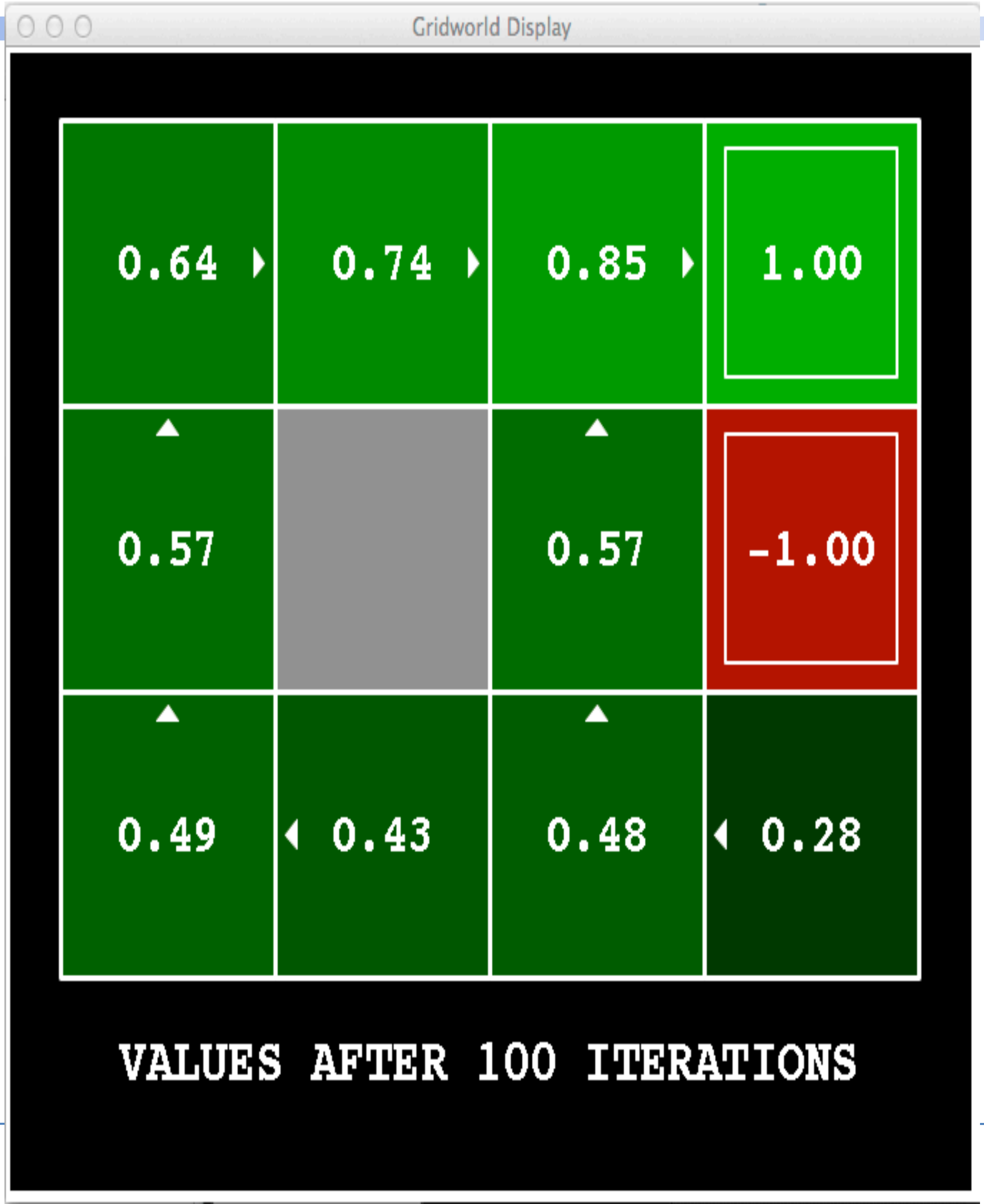
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

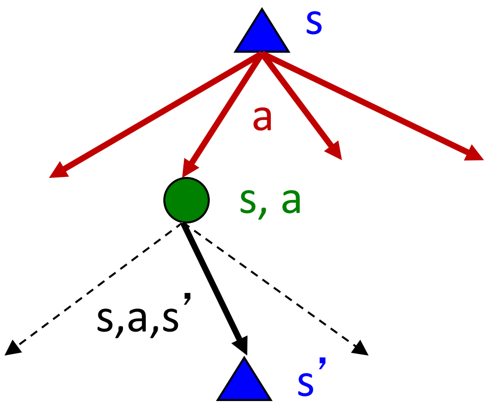


Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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



- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- **Problem 3: How to Read out Policy from Values?**
- **Problem 4: The policy often converges long before the values**