Most materials in this chapter are based on materials from http://aima.cs.berkeley.edu/ and http://ai.berkeley.edu

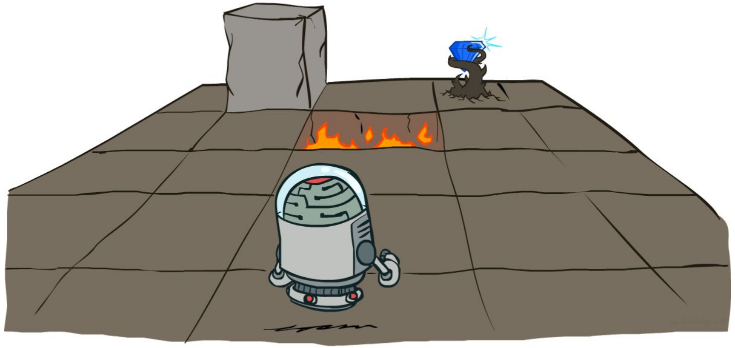
2. Markov Decision Processes

COMP 7404
Computational Intelligence and Machine Learning

Dirk Schnieders

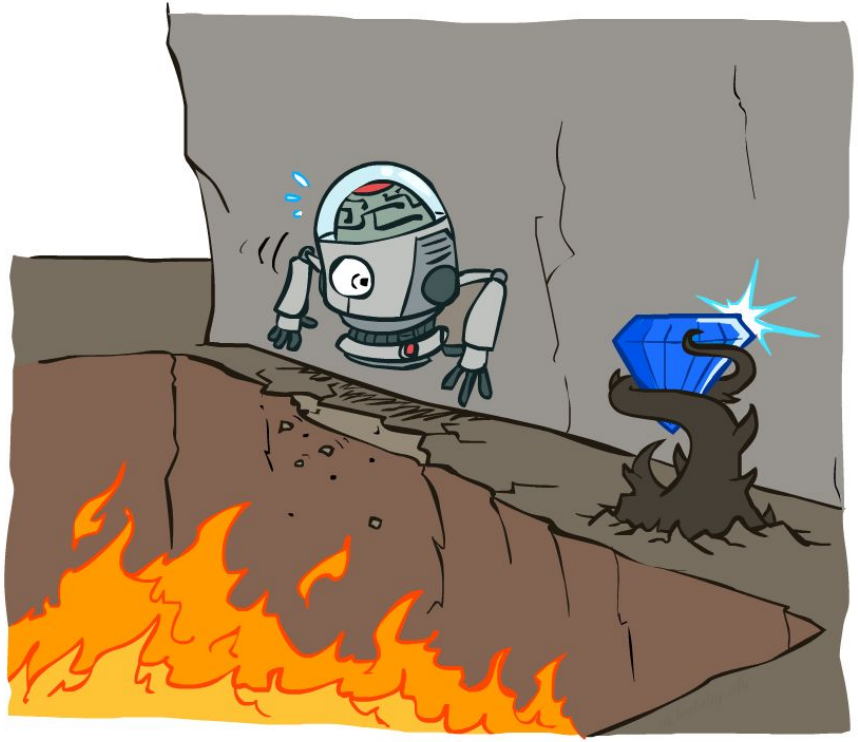
Sequential Decision Problem

- In a sequential decision problem the agent's utility depends on a sequence of decisions
- Sequential decision problems incorporate utilities, uncertainty and sensing
- Optimal behavior balances the risks and rewards of acting in an uncertain environment



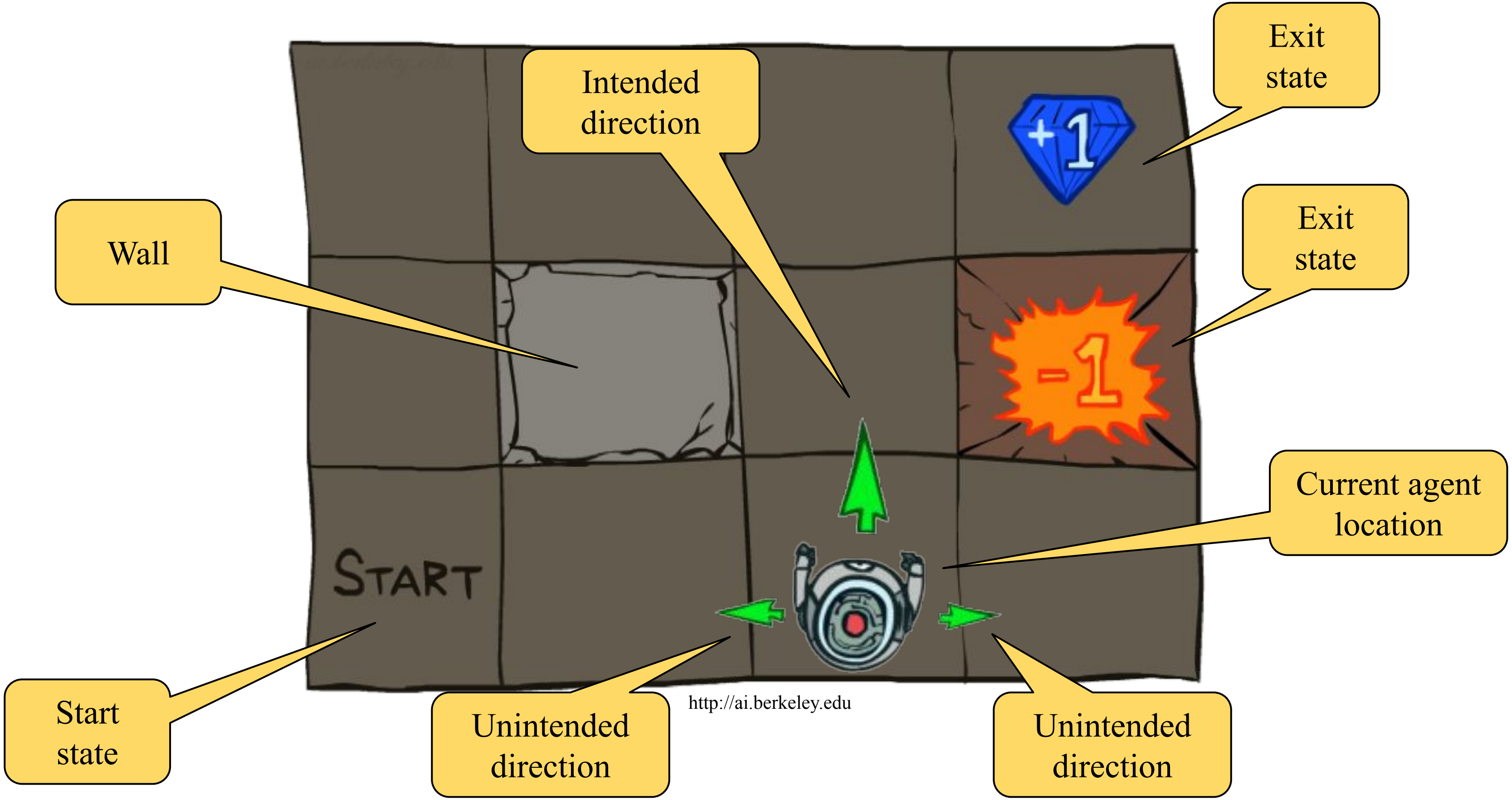
http://ai.berkeley.edu

Non-Deterministic Search



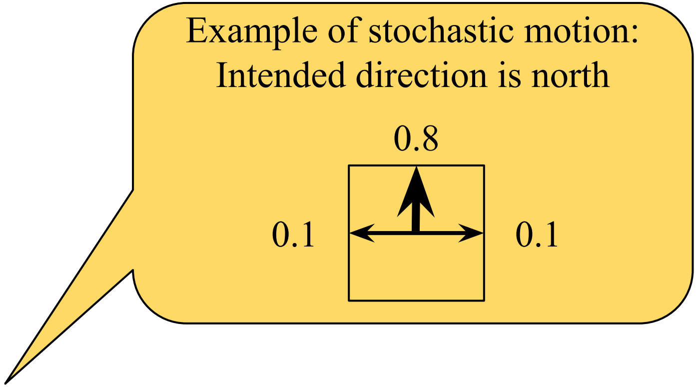
http://ai.berkeley.edu

Running Example: Grid World

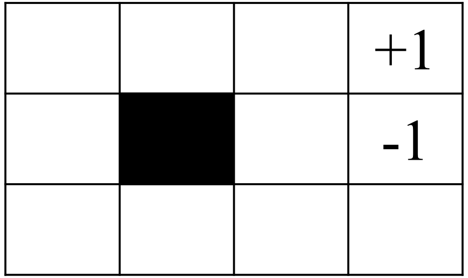


http://ai.berkeley.edu

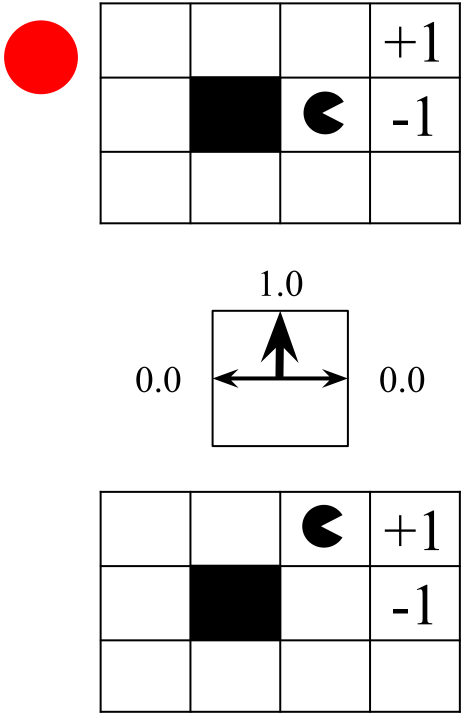
Grid World

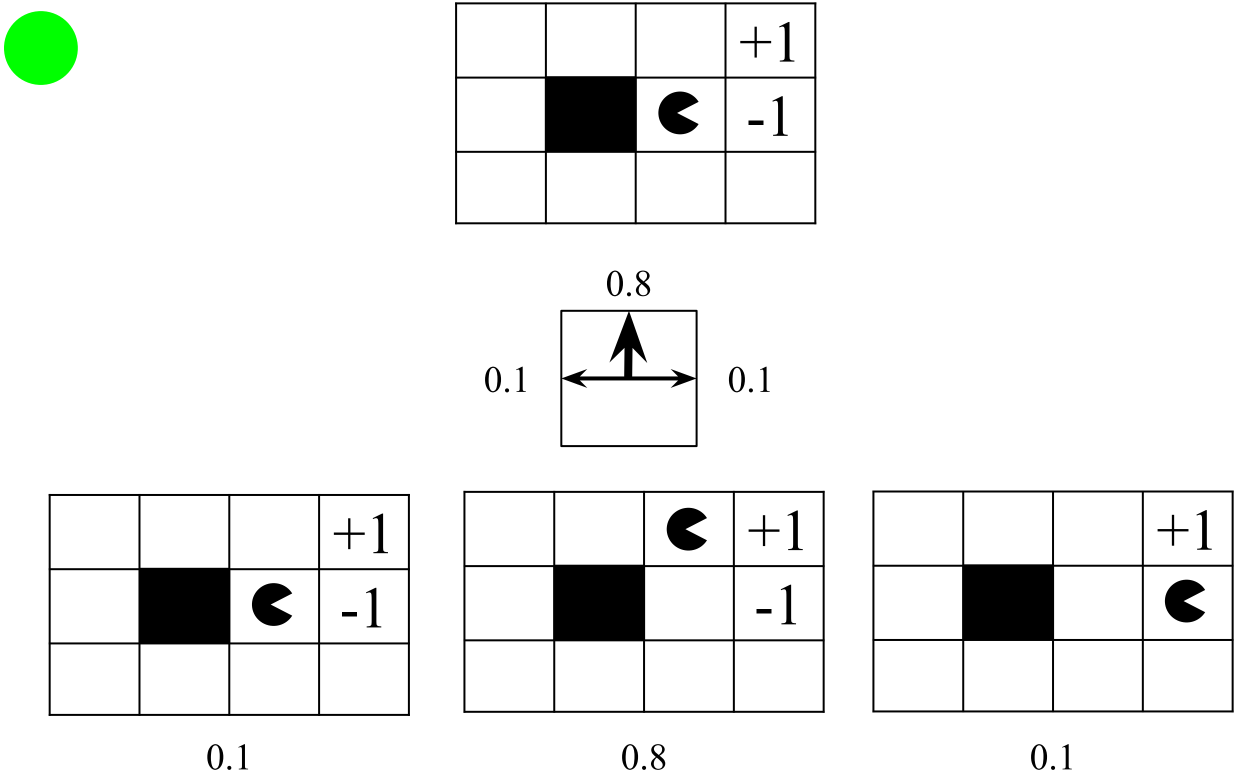


- Maze-like problem
  - Agent lives in a grid
  - Walls block the agent's path
- Unreliable actions
  - Each action achieves the intended effect 80% of the time
  - 10% of the time the action moves the agent at right angles (i.e., 90°) to the intended direction
  - If the agent bumps into a wall it stays in the same square
- The agent receives rewards for each action
  - Small "living" rewards (can be negative)
  - Big rewards come at the end (good or bad)
- Goal states
  - +1 and -1 are goal states
  - Only action available: exit action
- Goal: maximize sum of rewards



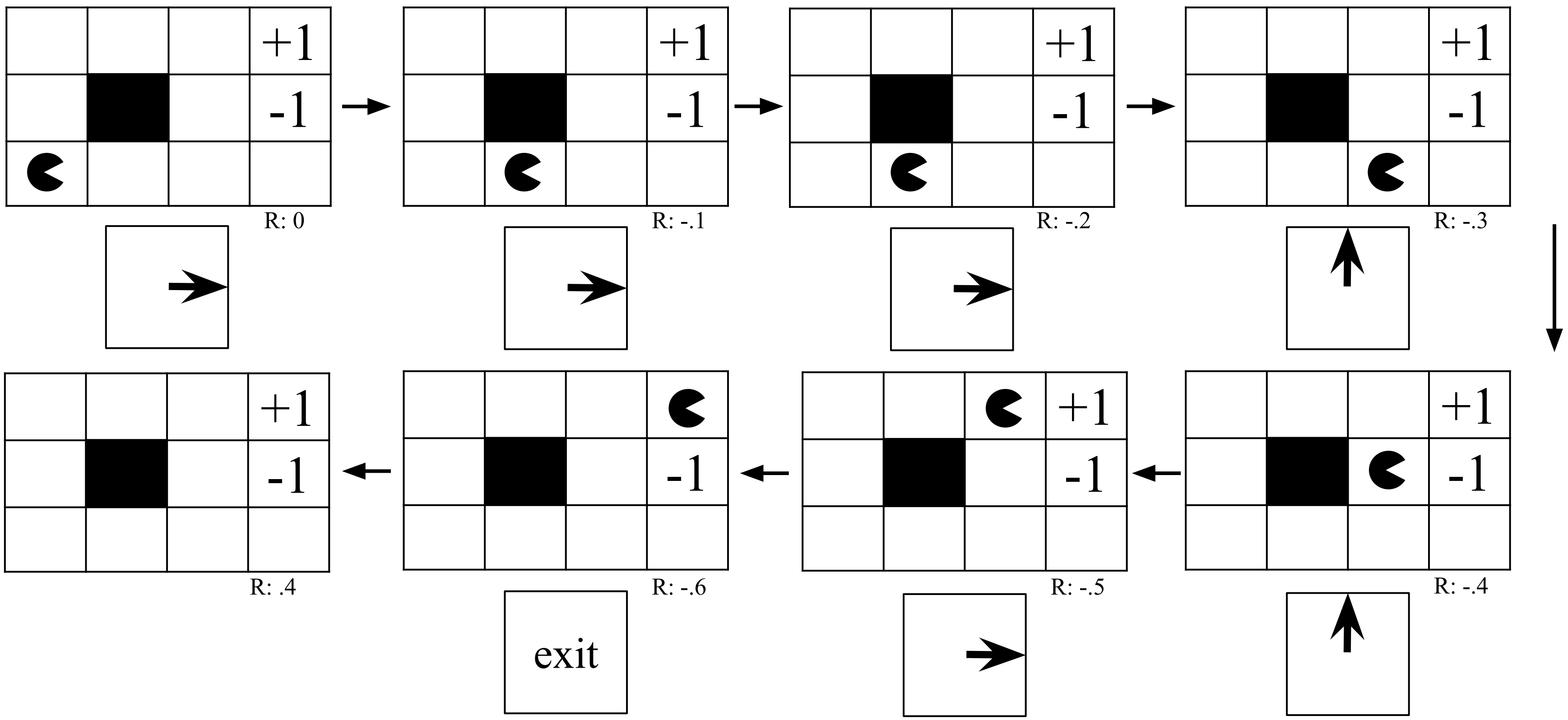
Deterministic vs. stochastic motion





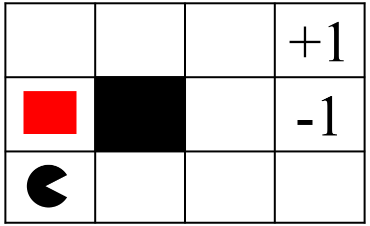
Reward (R): -0.1 per action

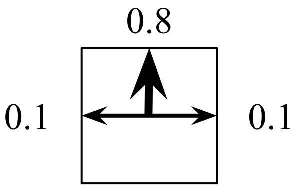
Example episode in grid world



Quiz - North North

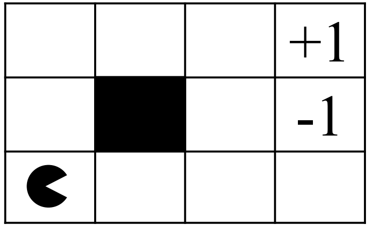
- Consider the fixed action sequence [North, North]
    - What is the probability of reaching the highlighted state from the start state with this action sequence?





Quiz - East East

- Which squares can be reached from the start state by the action sequence [East, East] and with what probabilities?



Transition model

- The transition model T(s,a,s’) describes the outcome of each action in each state
- The outcome is stochastic and we write P(s’|s,a) to denote the probability of reaching state s’ if action a is done in state s
- We assume that transitions are Markovian
    - I.e., the probability of reaching s’ from s depends only on s and not on the history of earlier states
    - The Markov property is named after the Russian mathematician Andrey Markov

source

Andrey Markov
(1856-1922)

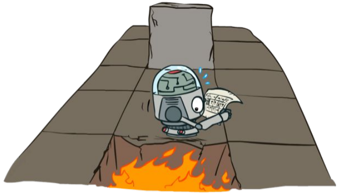
Markov Decision Processes

- A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a Markov decision process (MDP)
- It is defined by
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s,a,s')
    - Probability that action a from s leads to s', i.e., P(s'|s,a)
  - A reward function R(s,a,s')
  - A start state s_0
  - A terminal state (optional)

What is the solution to an MDP?

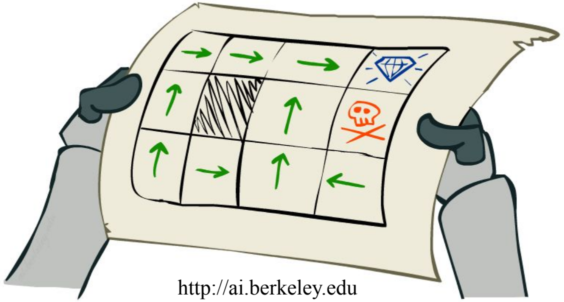
A fixed action sequence?

Policy



http://ai.berkeley.edu

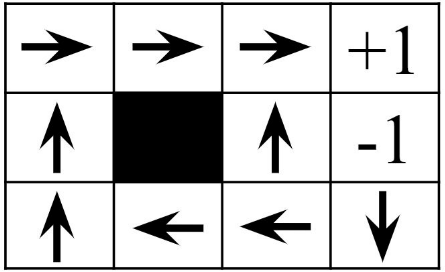
- How to solve MDPs?
  - A fixed action sequence won't solve the problem because the agent might end up in a state other than the goal
  - A solution must specify what the agent should do for any state that the agent might reach
  - A solution of this kind is called a policy π
    - $\pi(s)$ is the action recommended by the policy π for the state s



http://ai.berkeley.edu

Optimal Policy

- Each time a given policy is executed, the stochastic nature of the environment may lead to a different environment history
- The quality of a policy is therefore measured by the expected utility of the possible environment histories generated by that policy
- An optimal policy π^* is a policy that yields the highest expected utility
- Example: π^* for Grid World with $R(s) = -0.04$ ($\forall$ nonterminal s)



Optimal Policy - Example

→	→	→	+1
↑		←	-1
↑	←	←	↓

R(s) = -0.01

→	→	→	+1
↑		↑	-1
↑	←	←	←

R(s) = -0.03

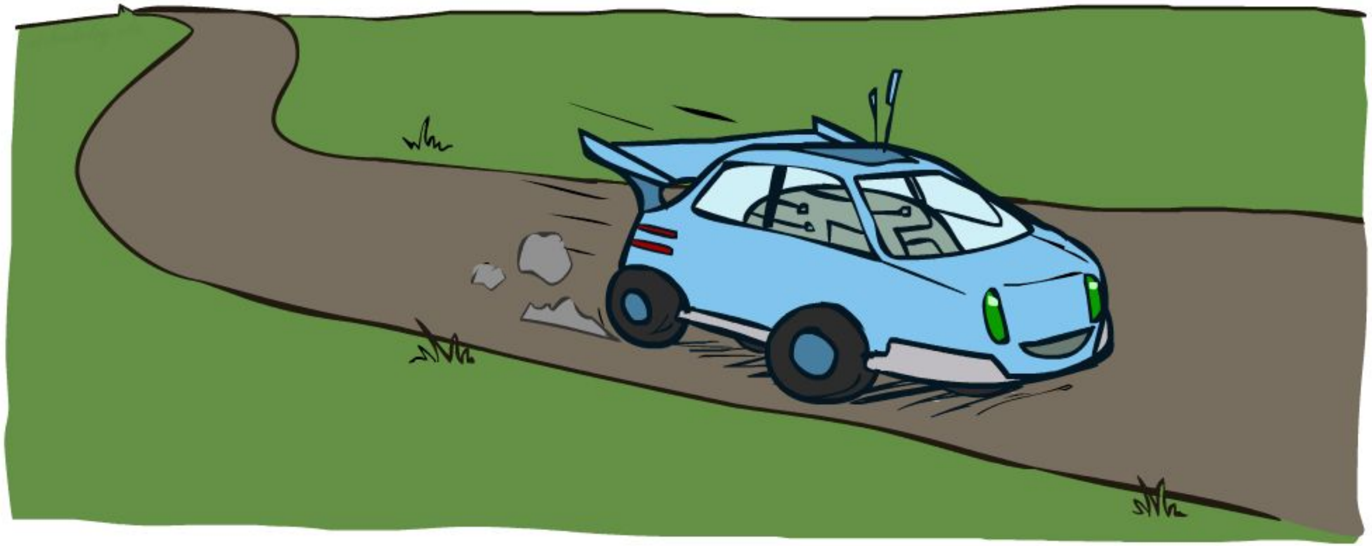
→	→	→	+1
↑		↑	-1
↑	→	↑	←

R(s) = -0.4

→	→	→	+1
↑		→	-1
→	→	→	↑

R(s) = -2.0

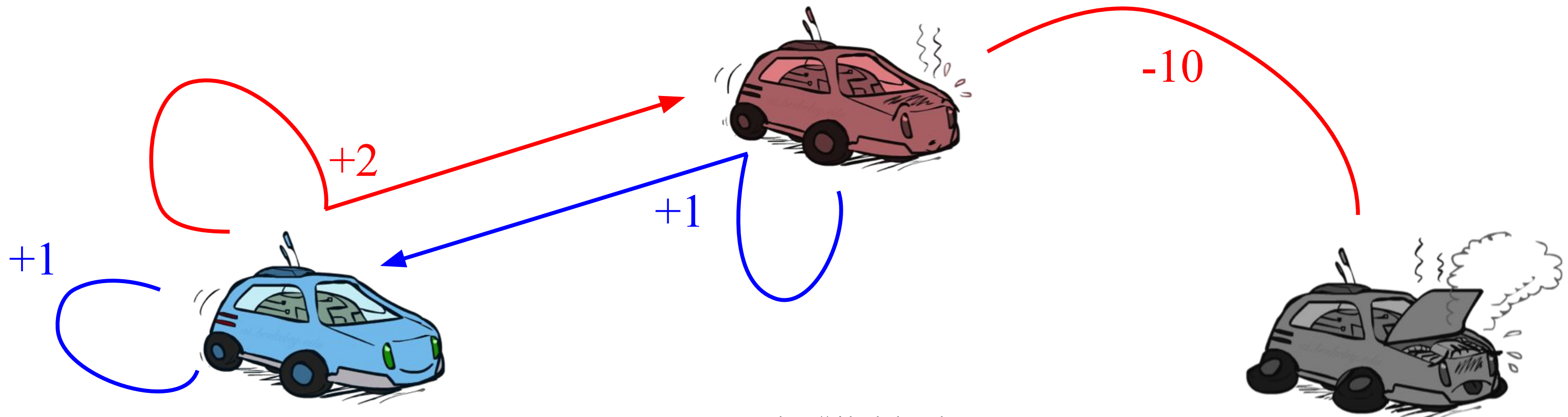
Example: Racing Car



http://ai.berkeley.edu

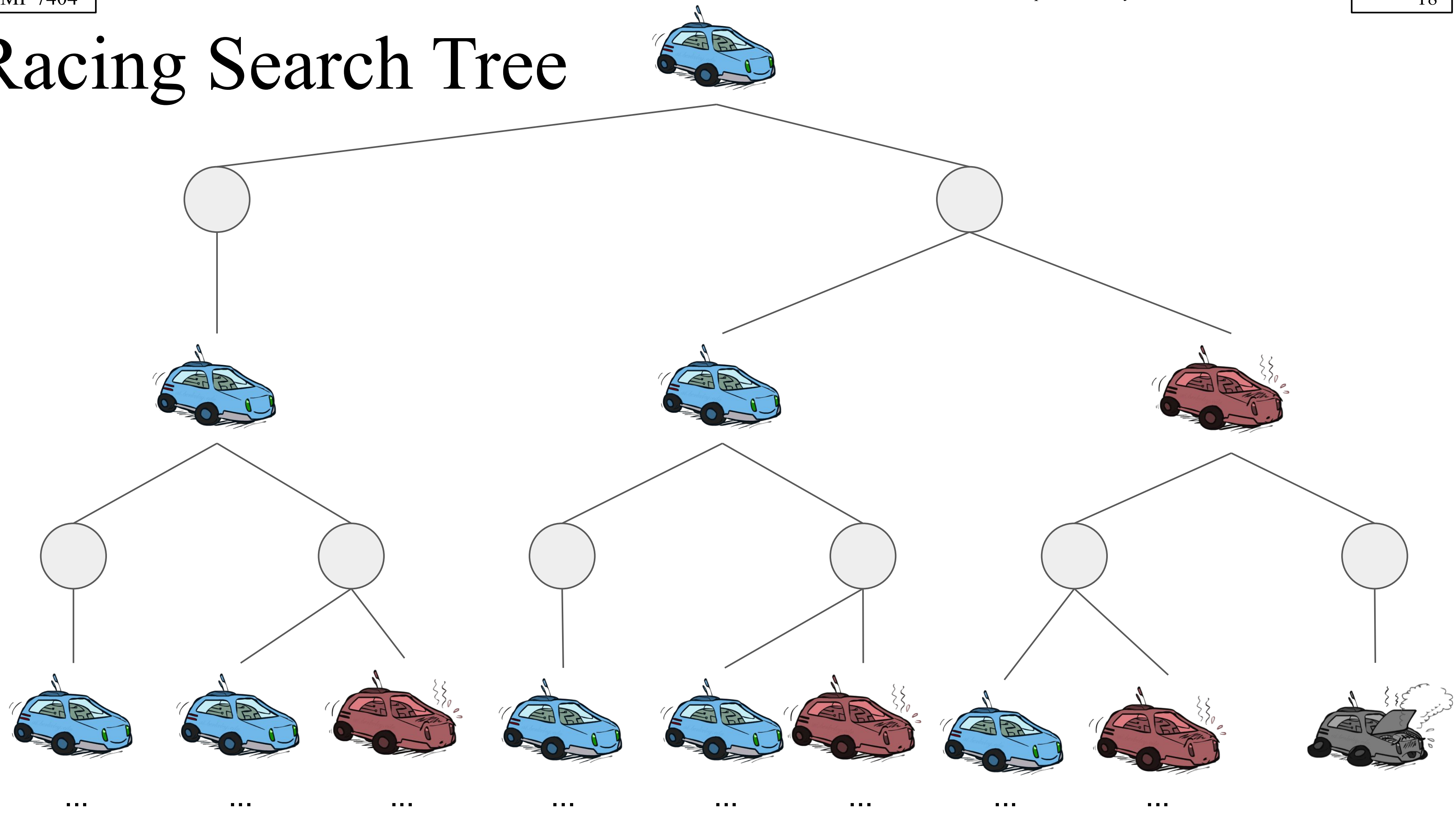
Example: Racing Car

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: Slow (+1), Fast (+2)



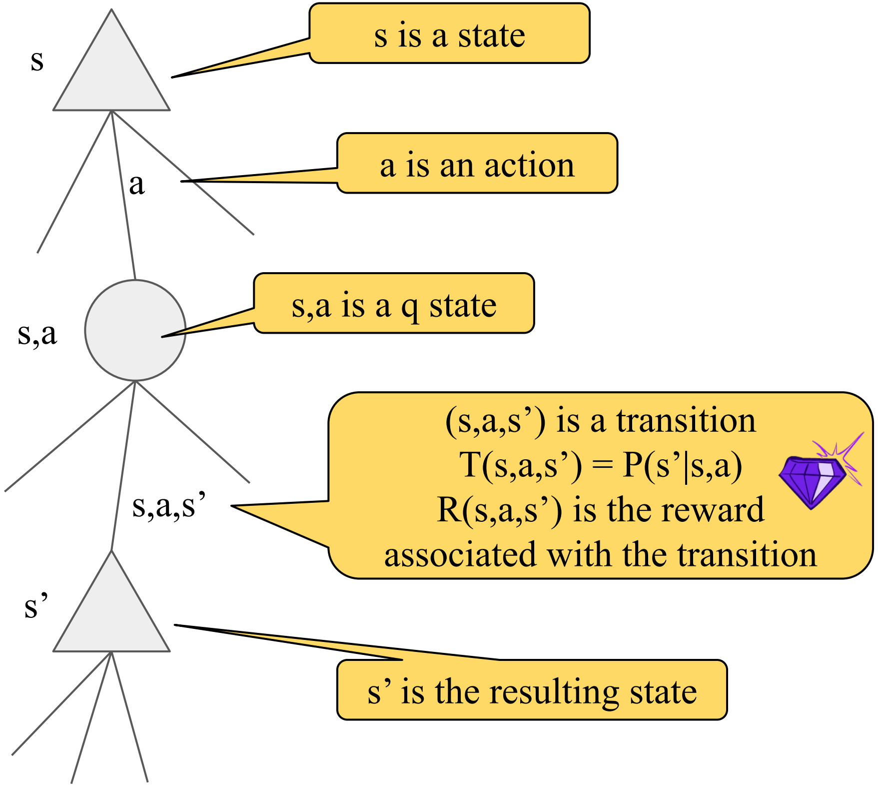
http://ai.berkeley.edu

Racing Search Tree



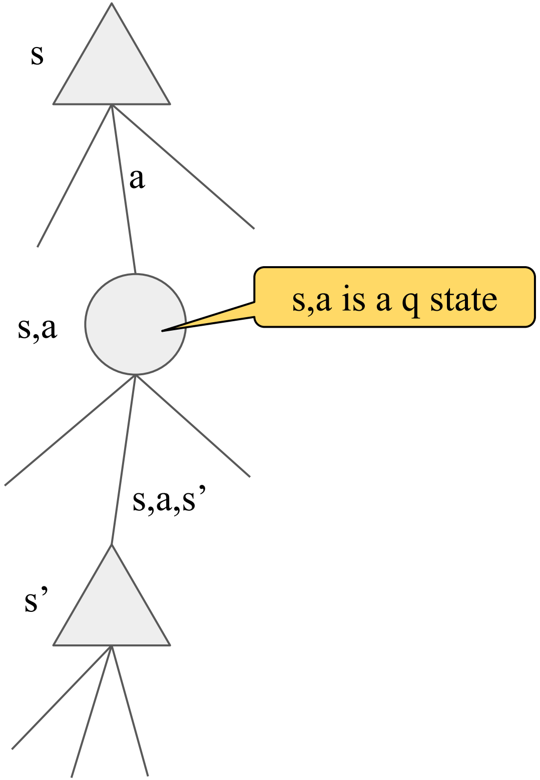
MDP Search Tree

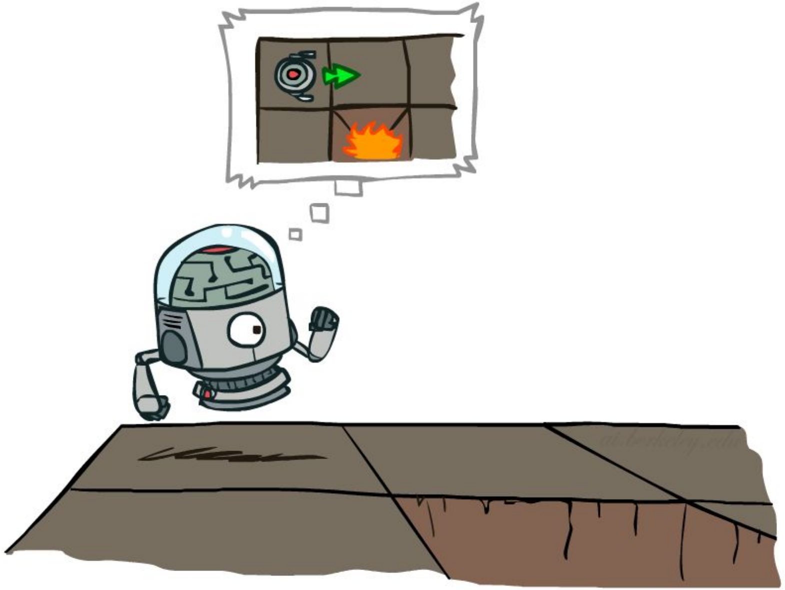
- Each MDP state projects an expectimax-like search tree



Q state

- The agent has committed to the action but has not done it yet





http://ai.berkeley.edu

Quiz - Notation

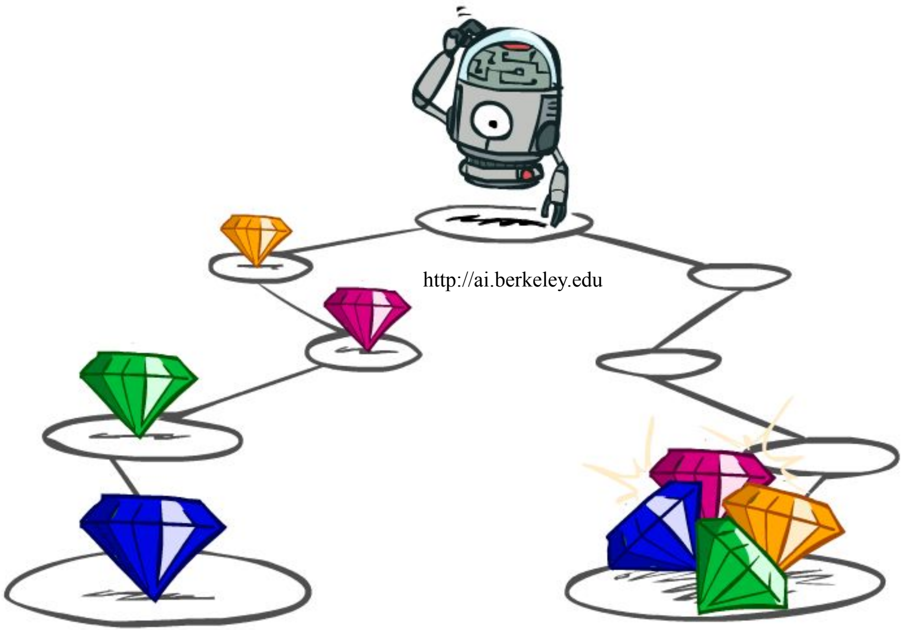
10				1
a	b	c	d	e

- Consider a grid world MDP as shown above
- The available actions in each state are to move to the neighboring grid squares
- East and west actions are successful 80% of the time
- When not successful, the agent stays in place
- From state a, there is also an exit action available, which results in going to the terminal state and collecting a reward of 10
  - Similarly, in state e, the reward for the exit action is 1
- Exit actions are successful 100% of the time

Find the following quantities

1. T(c,East,d)
2. T(c,East,e)
3. T(c,East,c)
4. T(c,East,b)
5. T(c,East,a)
6. T(c,East,terminal state)
7. T(a,Exit,terminal state)
8. T(a,East,b)
9. T(a,East,a)
10. R(a,East,a)
11. R(a,Exit,terminal state)
12. R(c,East,d)

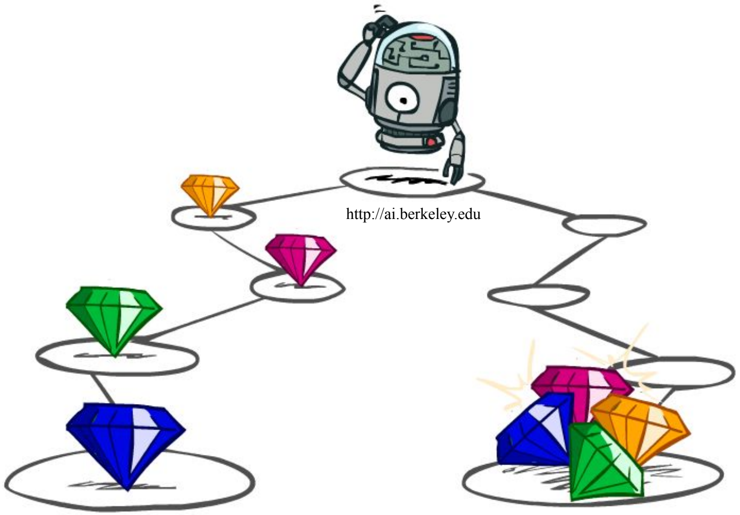
Utility of State Sequences



http://ai.berkeley.edu

Utility of State Sequences

- How to calculate the utility of state sequences?
  - What preferences should an agent have over reward sequences?
- More or less?
  - [1, 2, 2] or [2, 3, 4]
- Now or later?
  - [0, 0, 1] or [1, 0, 0]



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

http://ai.berkeley.edu

1

Worth Now

γ

Worth one step from now

γ^2

Worth two step from now

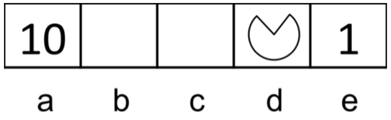
$1 > \gamma > 0$

Quiz - [1,2,3] vs. [3,2,1]

- Consider the sequences [1,2,3] and [3,2,1]
- Let $\gamma = 0.5$, which sequence has a higher utility?

Quiz - Optimal Action 1

- Consider the same grid world MDP as in one of the previous quiz
- Let actions always be successful
- Let the discount factor be $\gamma = 0.1$
  - Q1: What is the optimal action in the state d ?
- Now let the discount factor be $\gamma = 0.9999$
  - Q2: What is the optimal action in the state d ?



Quiz - Optimal Action 2

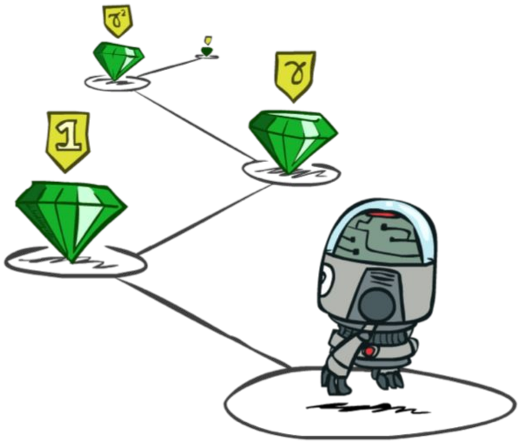
- Consider the following 101 x 3 world

+50	-1	-1	-1	...	-1	-1	-1	$\boxed{-1}$
Start				...				
-50	+1	+1	+1	...	+1	+1	+1	$\boxed{+1}$

- In the start state, the agent has a choice of two deterministic actions up or down but in the other states the agent has one deterministic action
- Assuming a discounted reward function, for what values of γ will the agent choose up ?
- Compute the utility of each action as a function of γ

Stationary Preferences

- We assume an agent's preferences between state sequences are stationary
- If two state sequences begin with the same state r, then the two sequences should be preference-ordered the same way as the sequences without r



http://ai.berkeley.edu

$$[r, s_1, s_2, \ldots] \succ [r, s_1', s_2', \ldots]$$
$$\Updownarrow$$
$$[s_1, s_2, \ldots] \succ [s_1', s_2', \ldots]$$

Stationary Preferences

- Stationarity has strong consequences
- It turns out that there are just two coherent ways to assign utilities to sequences
- Additive rewards

$$U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

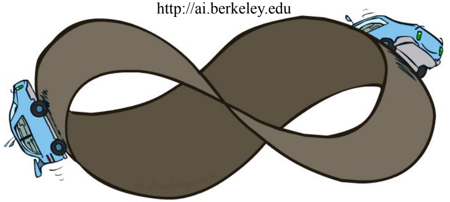
- Discounted rewards

$$U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

Quiz - Stationary Preference

- Suppose that we define the utility of a state sequence to be the maximum reward obtained in any state in the sequence
- Does this utility function result in stationary preference between state sequences?

Infinity Utilities?

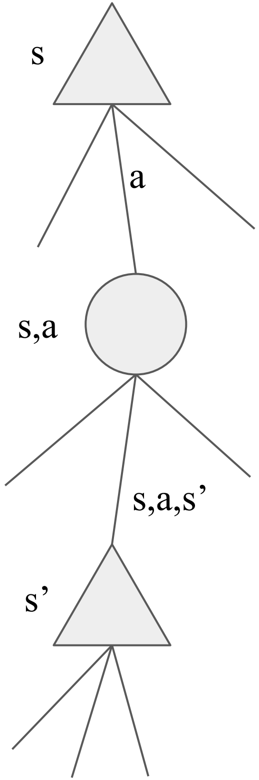


- What if the environment does not contain a terminal state or if the agent never reaches one?
- Utilities with additive undiscounted rewards will be infinite
  - E.g., Race game
    - The smart race car will never overheat
- With discounted rewards, the utility of an infinite sequence is finite

$$U([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = R_{max}/(1-\gamma)$$

Optimal Quantities

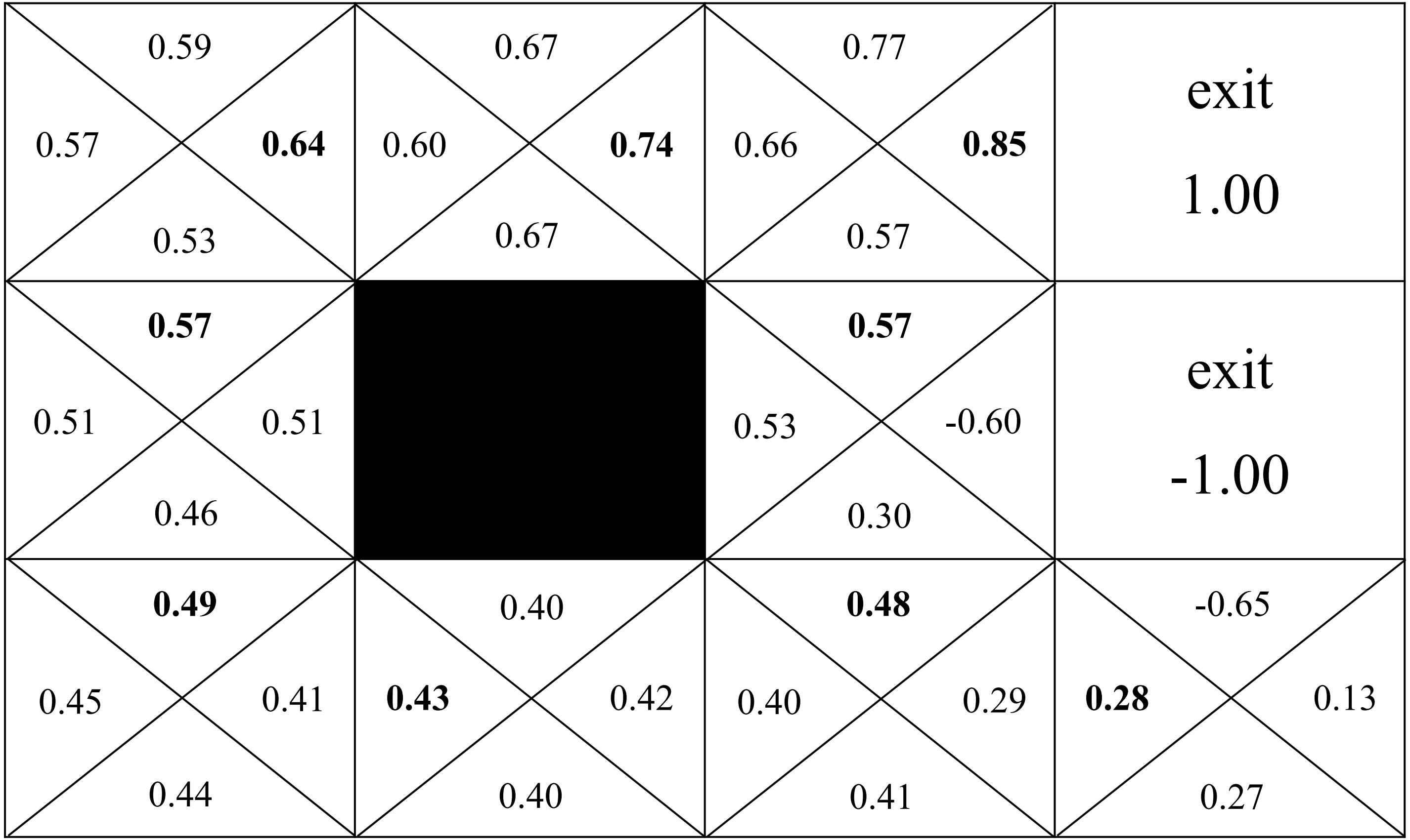
- $V^*(s)$ is the value (utility) of a state s
  - Expected utility starting in s and acting optimally
- $Q^*(s,a)$ is the value (utility) of a q-state (s,a)
  - Expected utility for having taken action a from state s and thereafter acting optimally
- $\pi^*(s)$ is the optimal policy for state s
  - I.e., optimal action from state s



Example - π^* and V^*

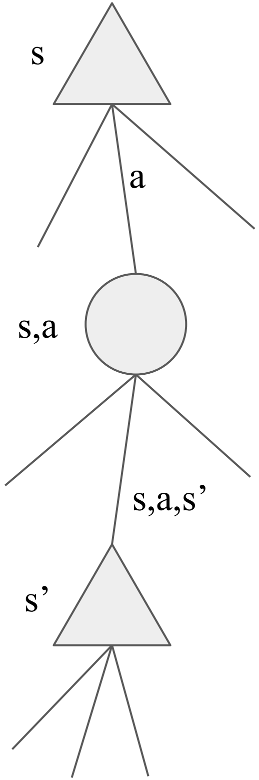
→ 0.64	→ 0.74	→ 0.85	exit 1.00
↑ 0.57		↑ 0.57	exit -1.00
↑ 0.49	← 0.43	↑ 0.48	← 0.28

Example - Q*


0.59
0.57
0.64
0.53
0.67
0.60
0.74
0.67
0.77
0.66
0.85
0.57
exit
1.00
0.57
0.51
0.51
0.46
0.57
0.53
-0.60
0.30
exit
-1.00
0.49
0.45
0.41
0.44
0.40
0.43
0.42
0.40
0.48
0.40
0.29
0.41
-0.65
0.28
0.13
0.27

Bellman Equation

- How to compute the value of a state?
  - Average sum of discounted action
  - Very similar to expectimax
  - Recursive definition

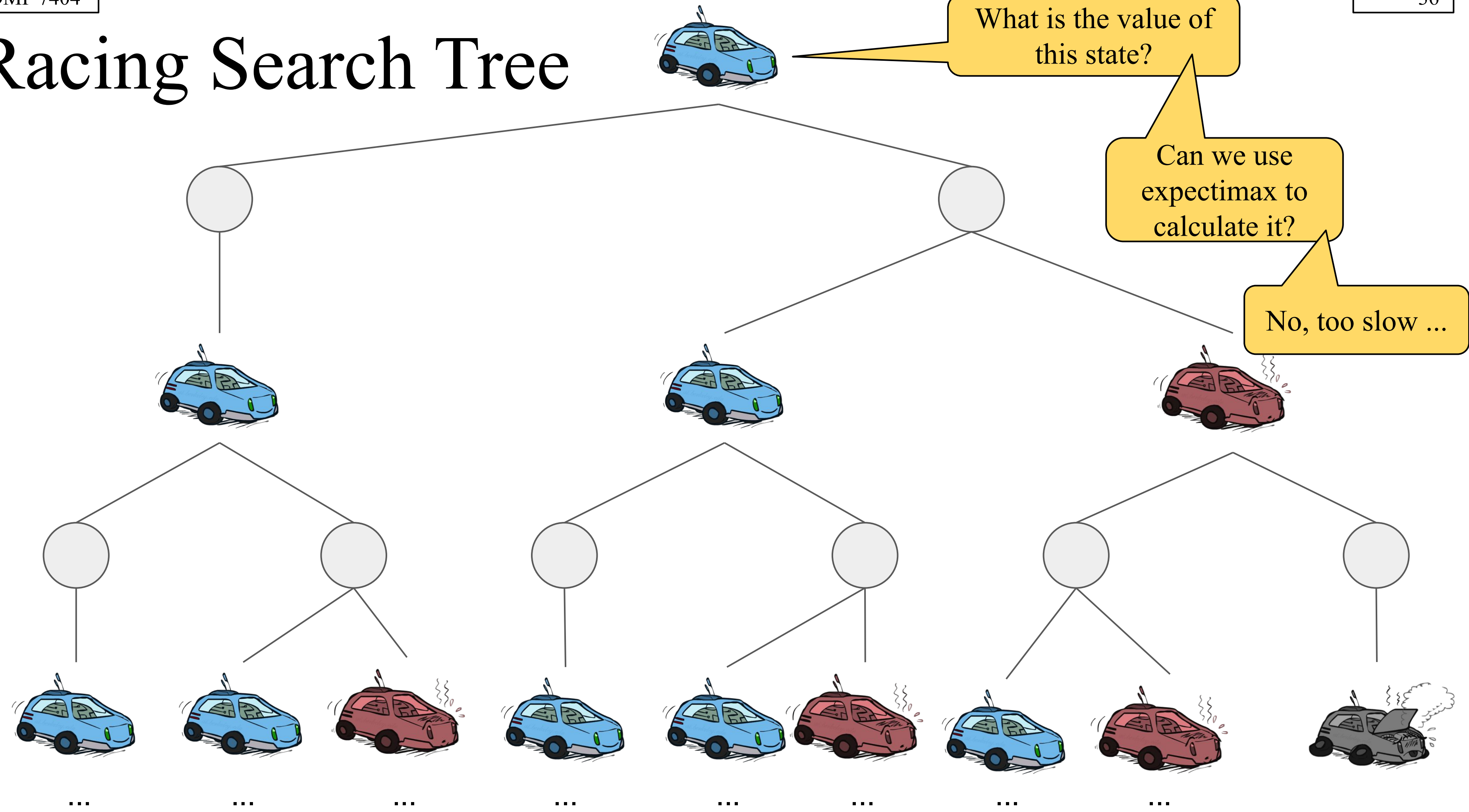


$$V^*(s) = \max_a Q^*(s,a)$$

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

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

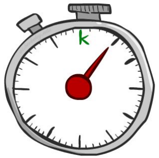
Racing Search Tree



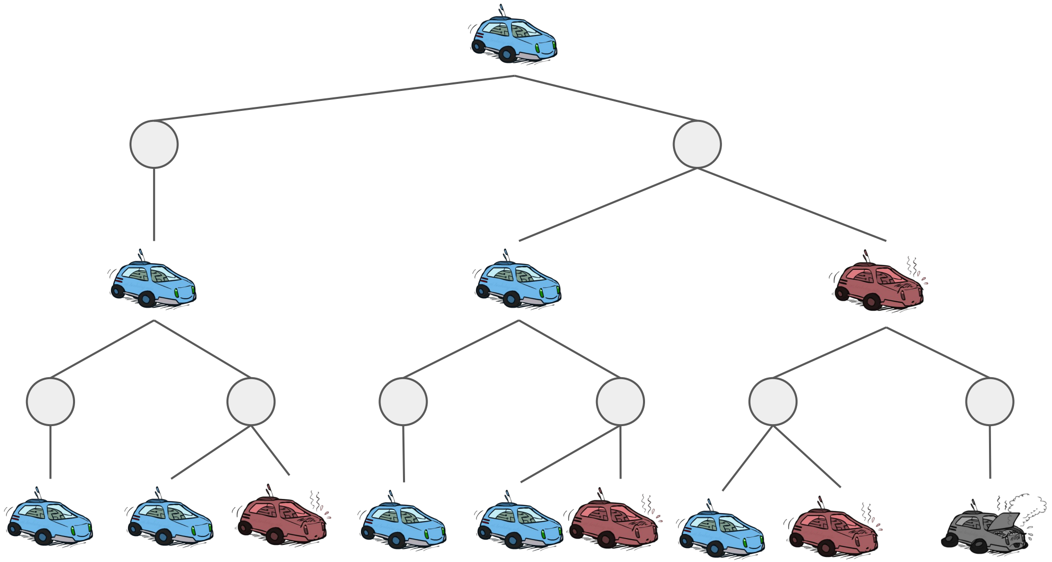
Racing Search Tree

- Problems
  - States are repeated (even at same depth)
  - 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 $\gamma < 1$
- Solution
  - Only compute needed quantities once
  - Do a depth-limited computation, but with increasing depths until change is small

Time-Limited Values



- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps (k more rewards)
- Equivalently, it’s what a depth-k expectimax would give from s



$V_2($  $)$

Example - V_0

0.00	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00	0.00	0.00

$R(s) = 0.00$
$\gamma = 0.99$

Example - V_1

0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

$R(s) = 0.00$

$\gamma = 0.99$

Example - V_2

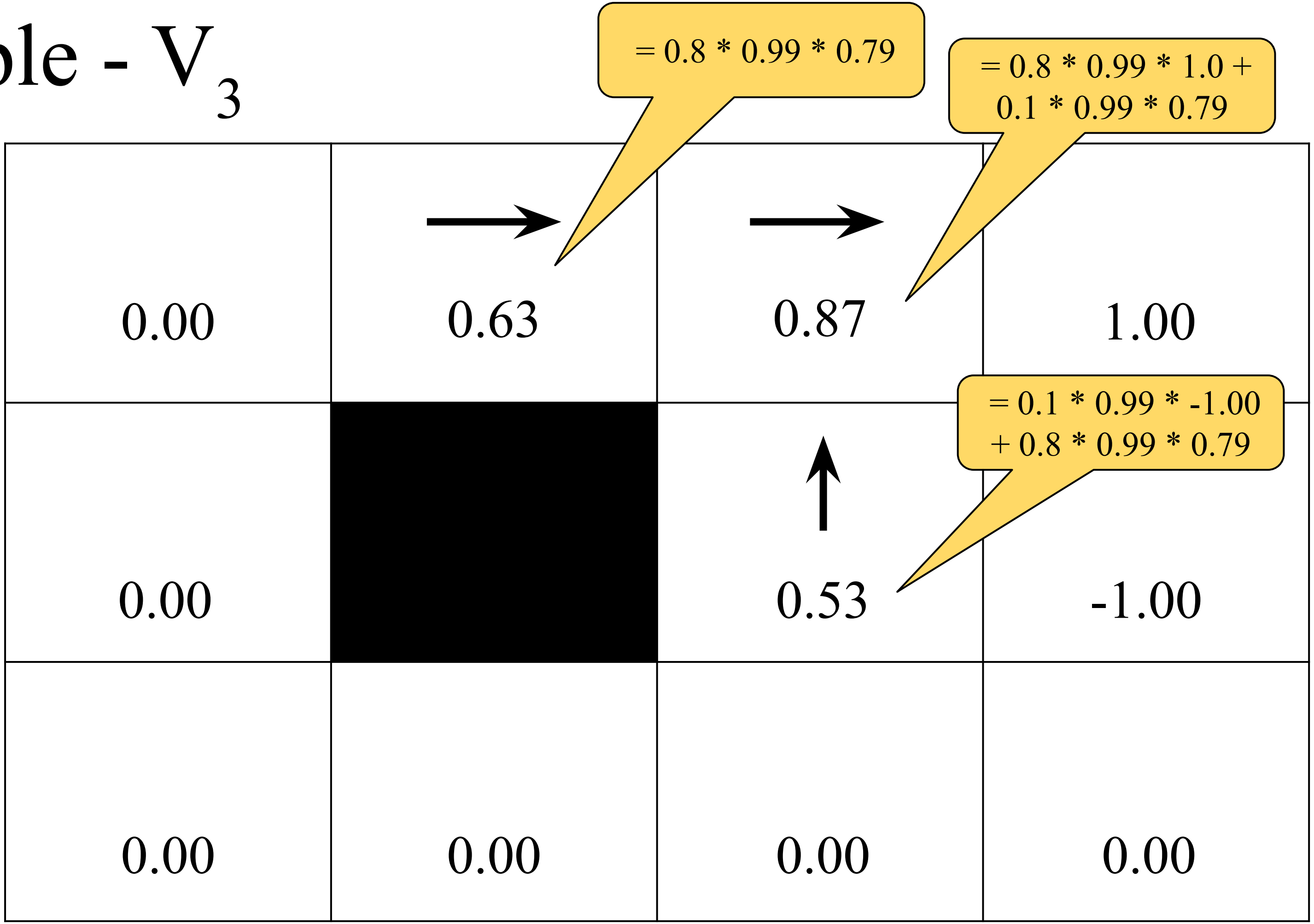
$= 0.8 * 0.99 * 1.00$

0.00	0.00	0.79	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

$R(s) = 0.00$

$\gamma = 0.99$

Example - V_3



$R(s) = 0.00$
$\gamma = 0.99$

Example - V_4

→ 0.50	→ 0.81	→ 0.93	1.00
0.00		↑ 0.64	-1.00
0.00	0.00	↑ 0.42	0.00

$R(s) = 0.00$
$\gamma = 0.99$

Example - V_5

→ 0.69	→ 0.90	→ 0.95	1.00
↑ 0.39		↑ 0.70	-1.00
0.00	→ 0.33	↑ 0.51	← 0.23

$R(s) = 0.00$
$\gamma = 0.99$

Example - V_6

→ 0.82	→ 0.93	→ 0.96	1.00
↑ 0.63	[redacted]	↑ 0.72	-1.00
↑ 0.34	→ 0.47	↑ 0.61	← 0.33

$R(s) = 0.00$
$\gamma = 0.99$

Example - V_7

→ 0.88	→ 0.94	→ 0.96	1.00
↑ 0.77		↑ 0.73	-1.00
↑ 0.58	→ 0.58	↑ 0.65	← 0.42

$R(s) = 0.00$

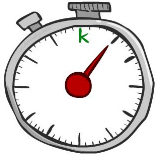
$\gamma = 0.99$

Example - V_{100}

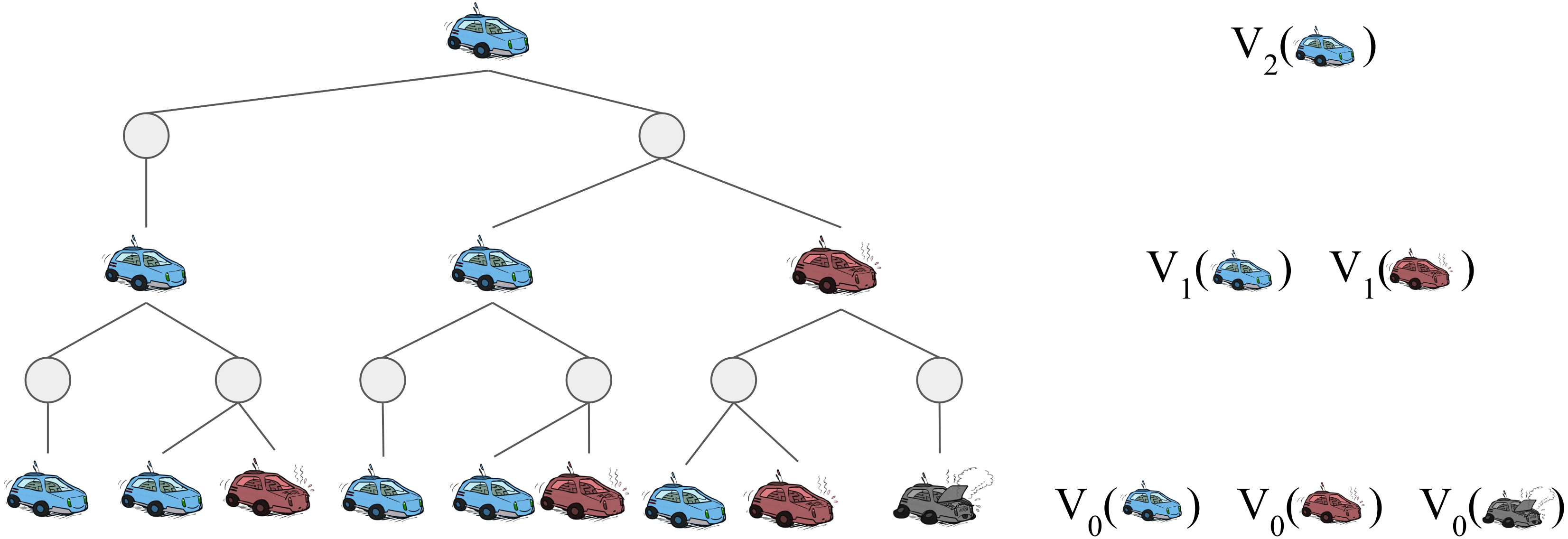
→ 0.95	→ 0.97	→ 0.98	1.00
↑ 0.94		← 0.89	-1.00
↑ 0.93	← 0.92	← 0.90	↓ 0.82

$R(s) = 0.00$
$\gamma = 0.99$

Computing Time-Limited Values

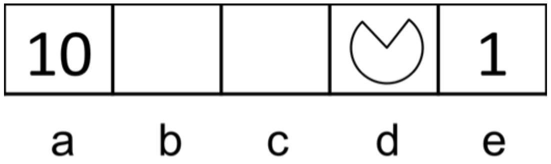


- We can save a lot of computation
- Example:
  - At every layer we have to compute at most 3 time limited values

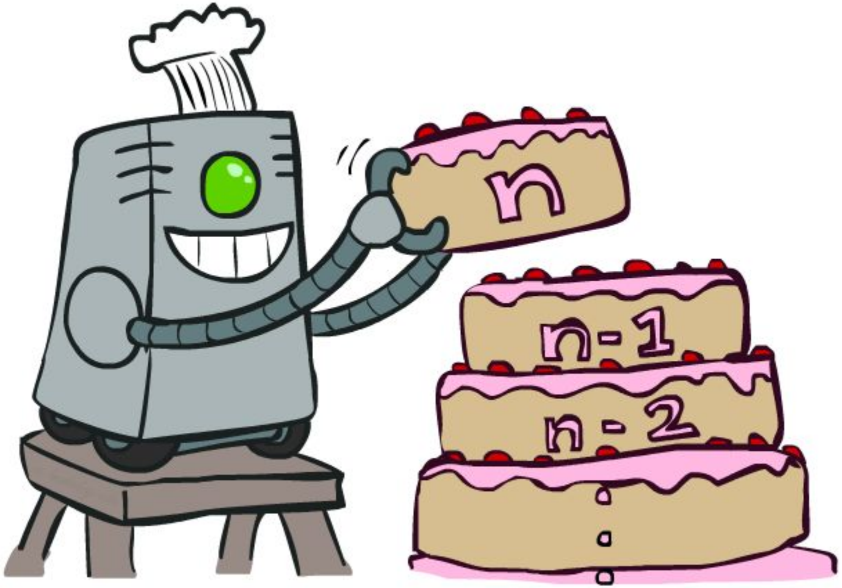


Quiz - Time-Limited Values

- Consider the same grid world MDP as in the previous quiz
  - Actions are successful 100% of the time
  - $\gamma = 1$
- Determine in the following quantities
  - $V_0(d)$ 0
  - $V_1(d)$ 0
  - $V_2(d)$ 1
  - $V_3(d)$ 1
  - $V_4(d)$ 10
  - $V_5(d)$ 10



Value Iteration



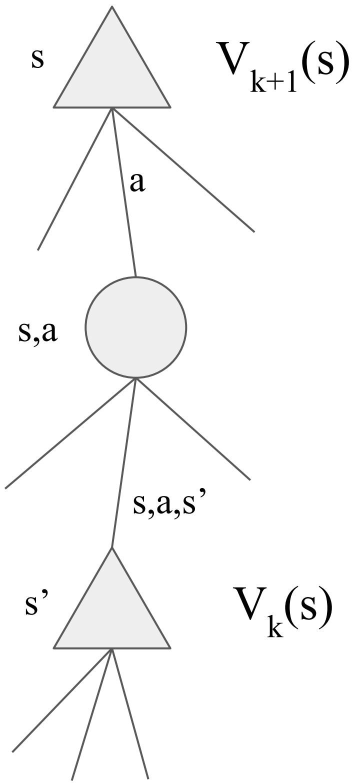
http://ai.berkeley.edu

Value Iteration

- Start with $V_0 = 0$
  - No time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one round of expectimax

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

- Repeat until convergence
- Complexity
  - $O(S^2A)$
- Theorem
  - Converges to unique optimal values



Quiz - Value Iteration

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

$V_0($$) = 0$

$V_1($$) =$ max(1.0*[1+1*0], 1/2[2+1*0]+1/2[2+1*0])=2

$V_2($$) =$ max(1.0*[1+1*2], 1/2[2+1*2]+1/2[2+1*1])=3.5

$V_0($$) = 0$

$V_1($$) =$ max(1.0*[-10+1*0] ,1/2[1+1*0]+1/2[1+1*0])=1

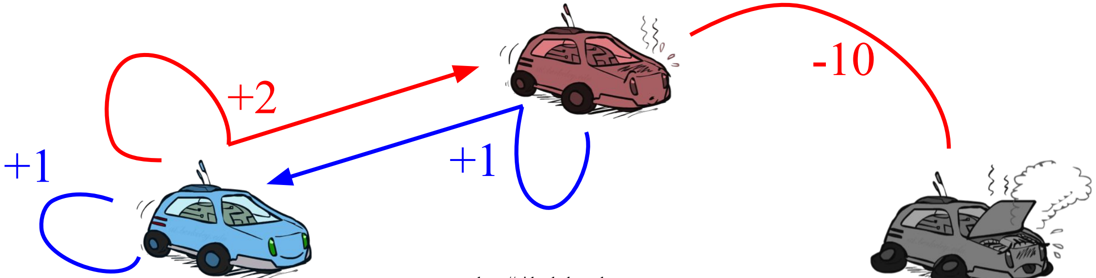
$V_2($$) =$ max(1.0*[-10+1*0], 1/2[1+1*2]+1/2[1+1*1])=3.5

$V_0($$) = 0$

$V_1($$) = 0$

$V_2($$) = 0$

$\gamma = 1.0$



http://ai.berkeley.edu

Quiz - V_∞

- Consider the same grid world as in the previous quiz (where east and west actions are successful 100% of the time)
- $\gamma = 0.2$
- Determine the following quantities
  - $V^*(a) = V_\infty(a)$
  - $V^*(b) = V_\infty(b)$
  - $V^*(c) = V_\infty(c)$
  - $V^*(d) = V_\infty(d)$
  - $V^*(e) = V_\infty(e)$

	v0	v1	v2	v3
a	0	10	10	10
b	0	0	10	10
c	0	0	0	10
d	0	0	1	0
e	0	1	1	0

不对

a	b	c	d	e
10				1

Quiz - Convergence

- Consider the following example where γ = 1.0 and R(s) = -0.04
  - Show that the value at (3,3) has converged

3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

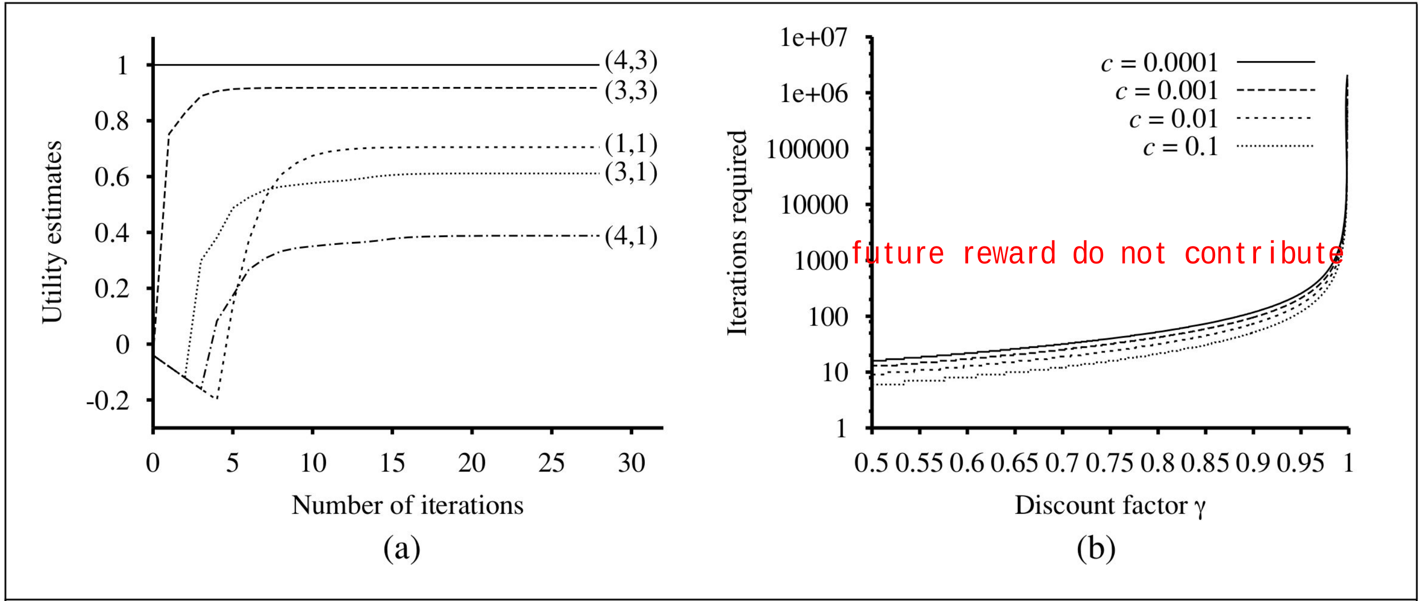
```
Vk+1(3,3)=max( 0.8*[-0.04+1*1]
               +0.1*[-0.04+1*0.660]
               +0.1*[-0.04+1*0.918]
,

                              )
```

```
find the max value in 4 directions
```

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

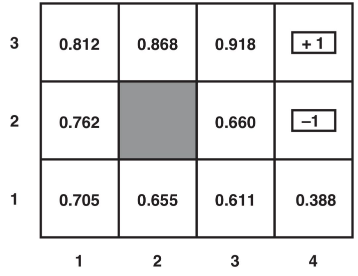
Evolution of Utilities / # Iterations vs. Gamma

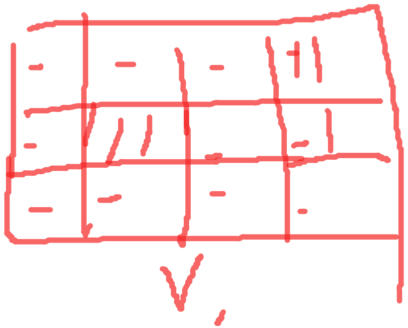
going down because at first -1 is always use



future reward do not contribute

Figure 17.5 (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations k required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of c, as a function of the discount factor γ.





Quiz - $V_{k+1}(B)$

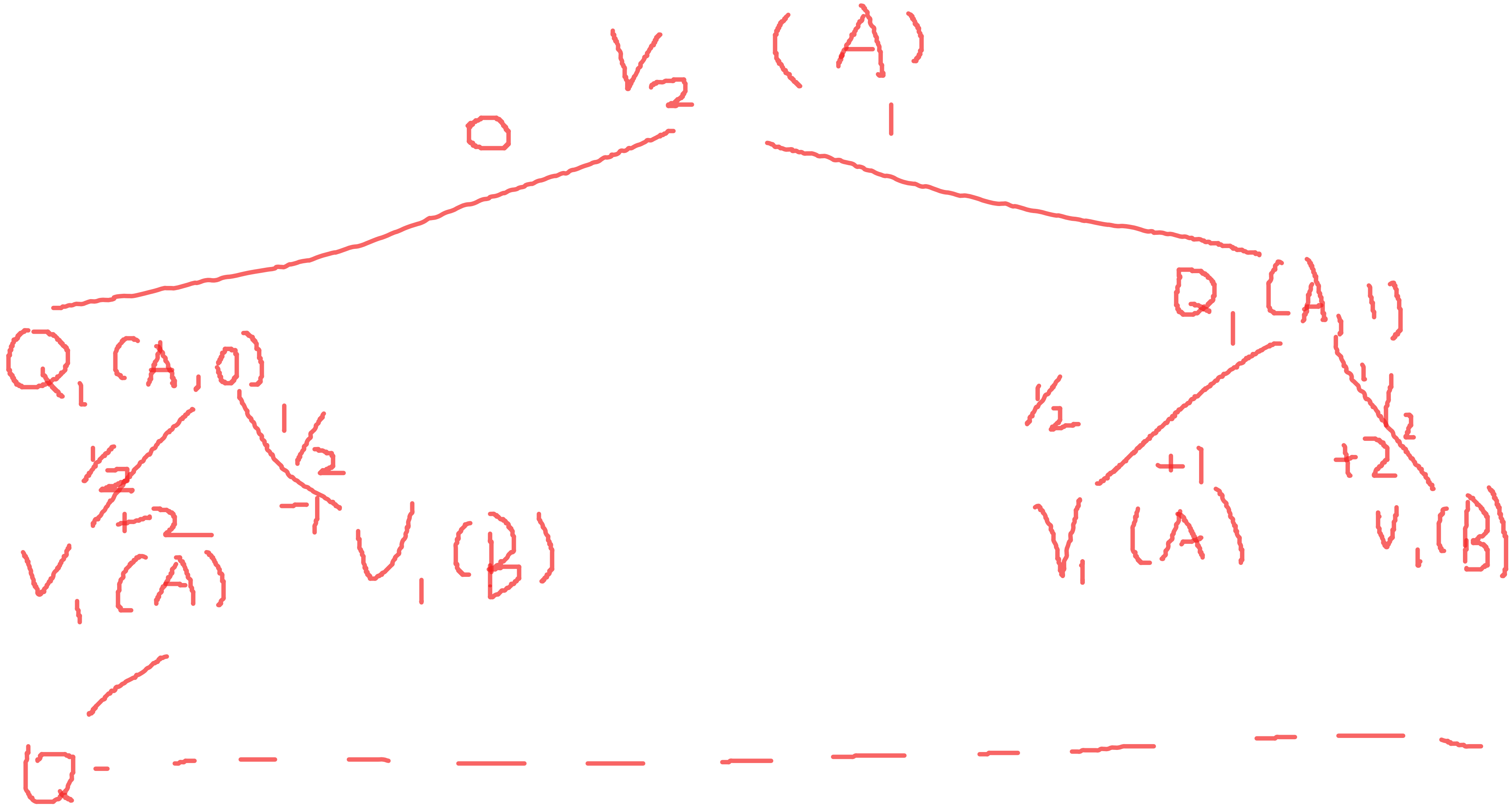
- Consider the following transition and reward functions for an MDP with $\gamma = 0.5$
- Suppose that after iteration k of value iteration we end up with the following values for V_k
  - $V_k(A) = 1.7$, $V_k(B) = 1.82$, $V_k(C) = 1.22$
  - What is $V_{k+1}(B)$?
- Now, suppose that we ran value iteration to completion and found the following value function
  - $V^*(A) = 2.208$, $V^*(B) = 2.416$, $V^*(C) = 1.766$
- What is $Q^*(B, CW)$?
- What is $Q^*(B, CCW)$?
- What is the optimal action from state B ?

s	a	s'	T(s,a,s')	R(s,a,s')
A	CW	B	1.0	1.0
A	CCW	B	0.4	-2.0
A	CCW	C	0.6	-1.0
B	CW	A	0.6	1.0
B	CW	C	0.4	2.0
B	CCW	A	0.2	1.0
B	CCW	C	0.8	-1.0
C	CW	A	0.6	-2.0
C	CW	B	0.4	1.0
C	CCW	A	0.4	0.0
C	CCW	B	0.6	1.0

Quiz

- Consider the transition and reward function shown in the table for an MDP that has two states and two actions
- Let $\gamma = 1.0$
- Determine the following
  - $V_0(A)$, $V_0(B)$ 0 0
  - $Q_1(A,0)$, $Q_1(A,1)$, $Q_1(B,0)$, $Q_1(B,1)$
  - $V_1(A)$, $V_1(B)$
  - $Q_2(A,0)$, $Q_2(A,1)$
  - $V_2(A)$

s	a	s'	T(s,a,s')	R(s,a,s')
A	0	A	0.5	2
A	0	B	0.5	-1
A	1	A	0.5	1
A	1	B	0.5	2
B	0	A	0.0	-2
B	0	B	1.0	-1
B	1	A	0.1	-3
B	1	B	0.9	-1



V2 (A)
0
1
Q1 (A, 0)
Q1 (A, 1)
1/2
1/2
+2
-1
V1 (A)
V1 (B)
1/2
1/2
+1
+2
V1 (A)
V1 (B)
Q

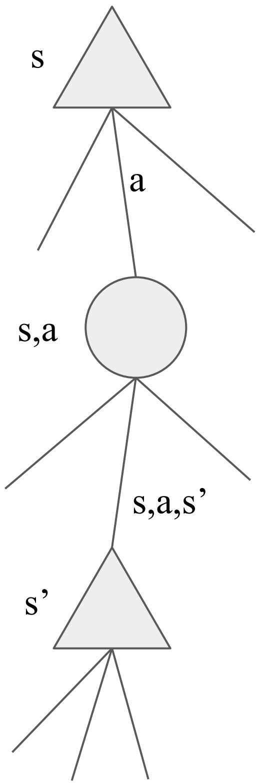
Bellman Equation vs. Value Iteration

- Bellman equations characterize the optimal values

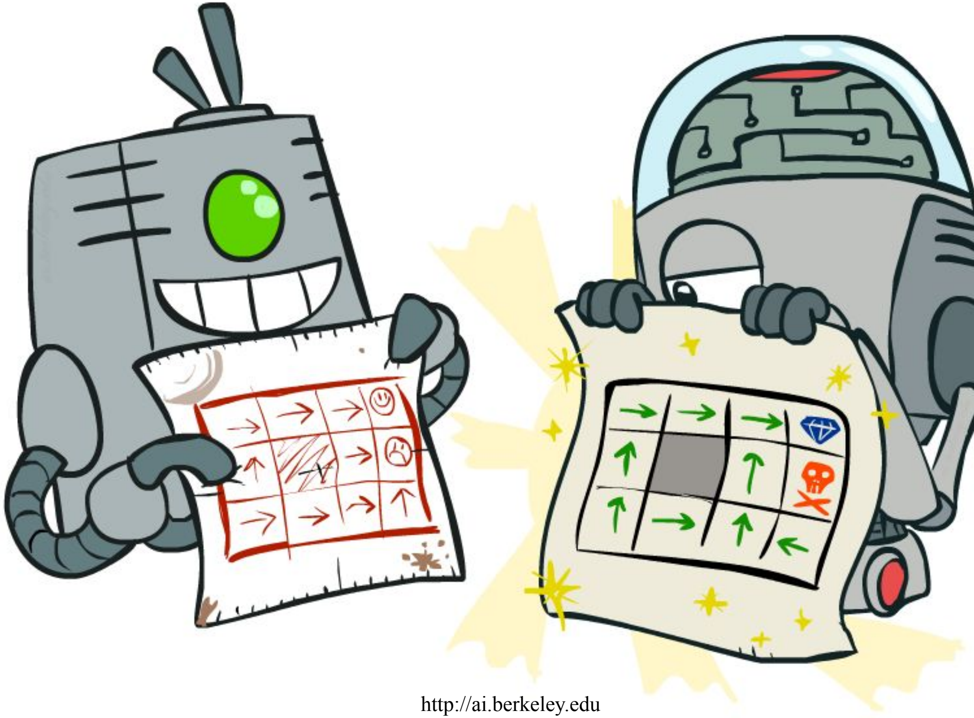
$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- Value iteration computes them

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

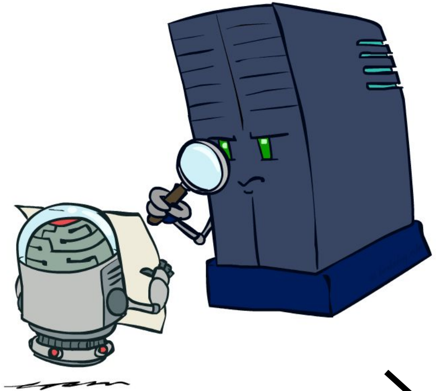


Policy Methods

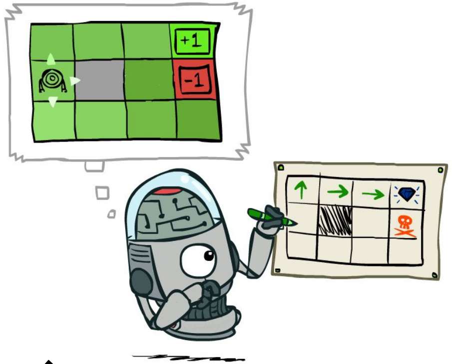


http://ai.berkeley.edu

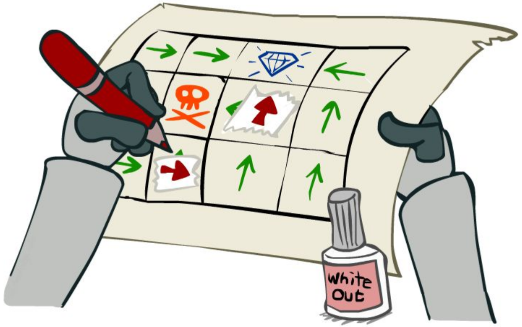
Policy Evaluation



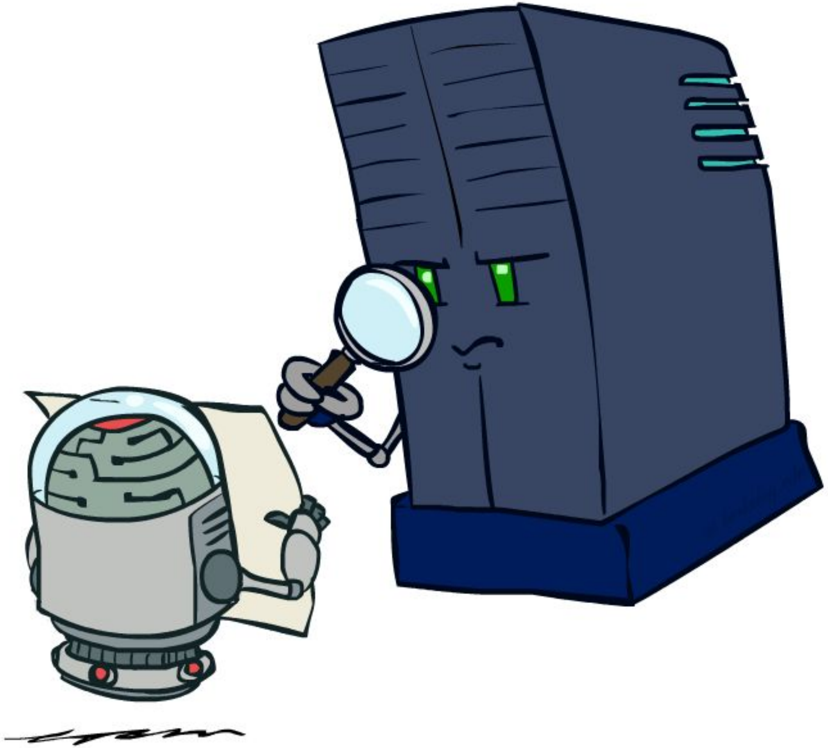
Policy Extraction



Policy Iteration



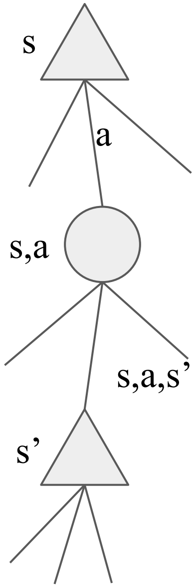
Policy Evaluation



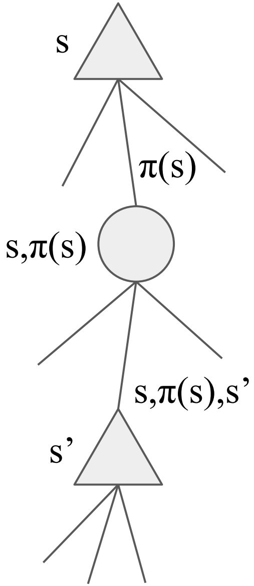
Policy Evaluation

- We would like to determine how good a given a policy π is
  - How well will I perform if I follow π ?

Do the optimal action



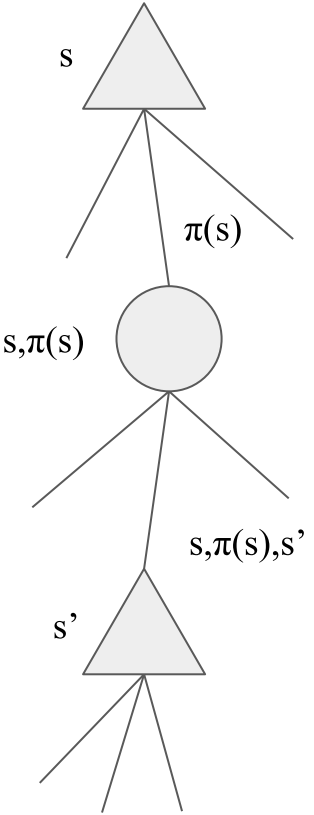
Do what π says



Utilities for a Fixed Policy

- We compute the utility of a state s under a fixed (generally non-optimal) policy
  - $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

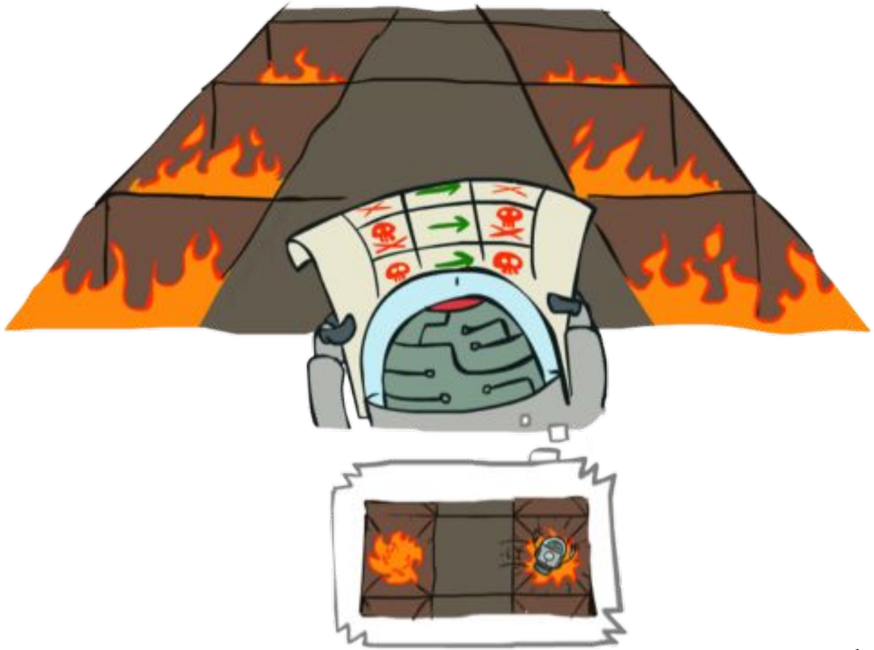
$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

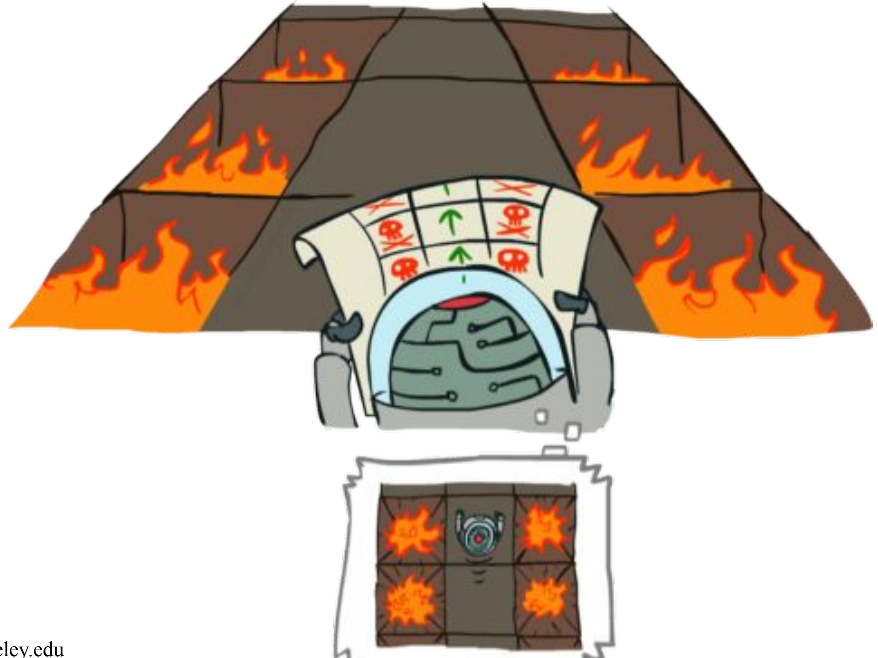


New Example

-10	100	-10
-10		-10
-10		-10
-10		-10

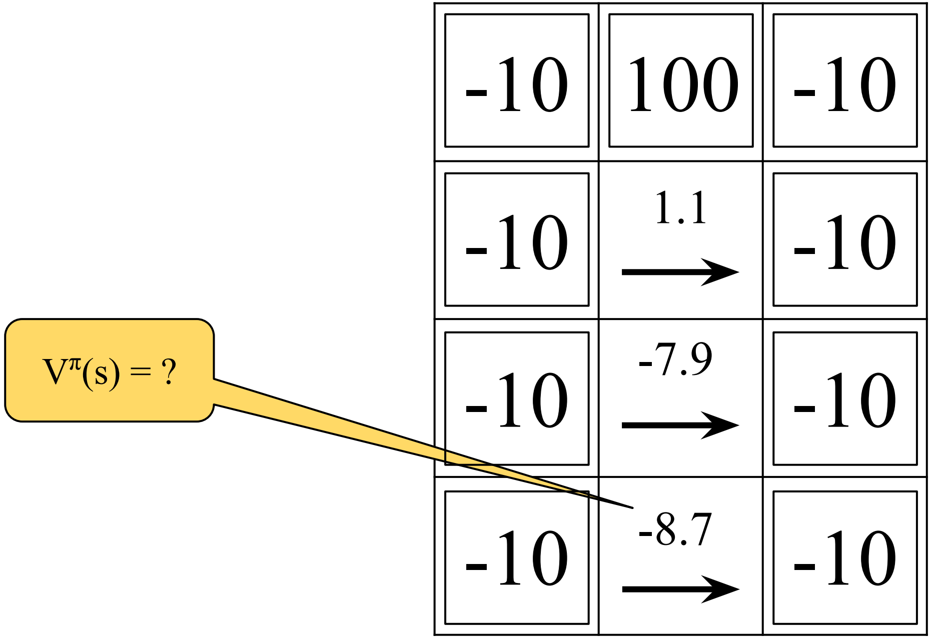
Example: Policy Evaluation





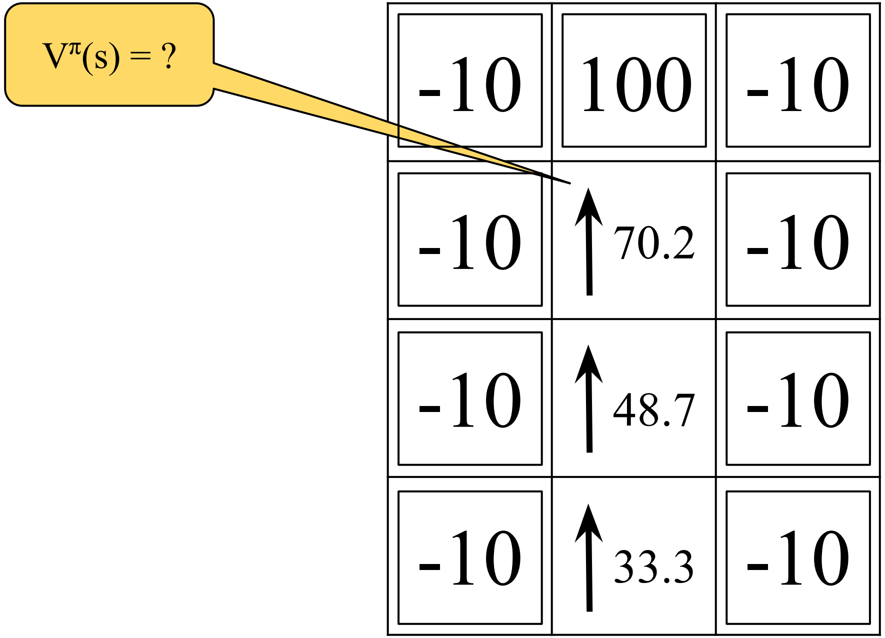
http://ai.berkeley.edu

Quiz - Go East



$R(s) = 0.00$
$\gamma = 0.90$

Quiz - Go North



$R(s) = 0.00$

$\gamma = 0.90$

Example: Policy Evaluation

-10	100	-10
-10	→ 1.1	-10
-10	→ -7.9	-10
-10	→ -8.7	-10

-10	100	-10
-10	↑ 70.2	-10
-10	↑ 48.7	-10
-10	↑ 33.3	-10

R(s) = 0.00
$\gamma = 0.90$

Policy Evaluation

- How do we calculate the V's for a fixed policy π ?
  - Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$

    - Efficiency: $O(S^2)$ per iteration
  - Idea 2: Without the max, the Bellman equations are just a linear system
    - Use a linear system solver

Quiz - Policy Evaluation

a	b	c	d	e
10				1

- Consider the same grid world as in the previous quiz, where east and west actions are successful 100% of the time and $\gamma = 1$
- Consider the policy π

$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$

$V^{\pi}(s) = 0$

  - Evaluate the values for all states

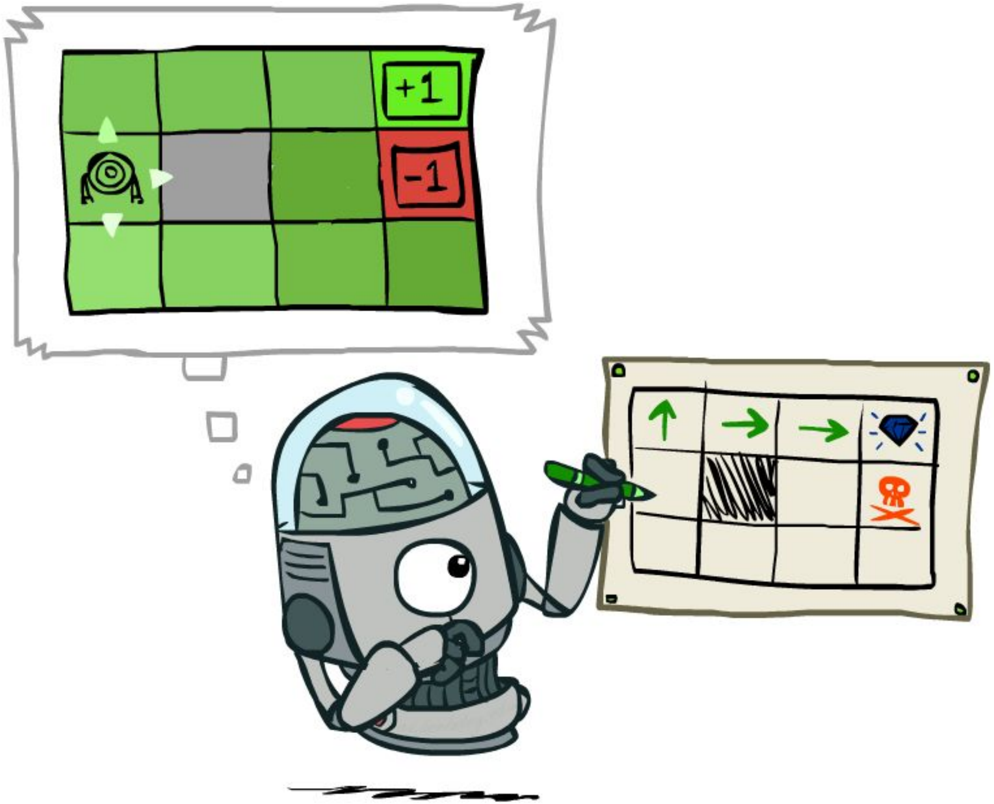
- Consider the policy π'

$\leftarrow$ $\leftarrow$

exit	$\leftarrow$	$\leftarrow$	$\rightarrow$	exit
10	10	10	1	1

  - Evaluate the values for all states

Policy Extraction



http://ai.berkeley.edu

Policy Extraction

- Let's imagine we have the optimal values $V^*(s)$

0.95	0.97	0.98	+1
0.94		0.89	-1
0.93	0.92	0.90	0.82

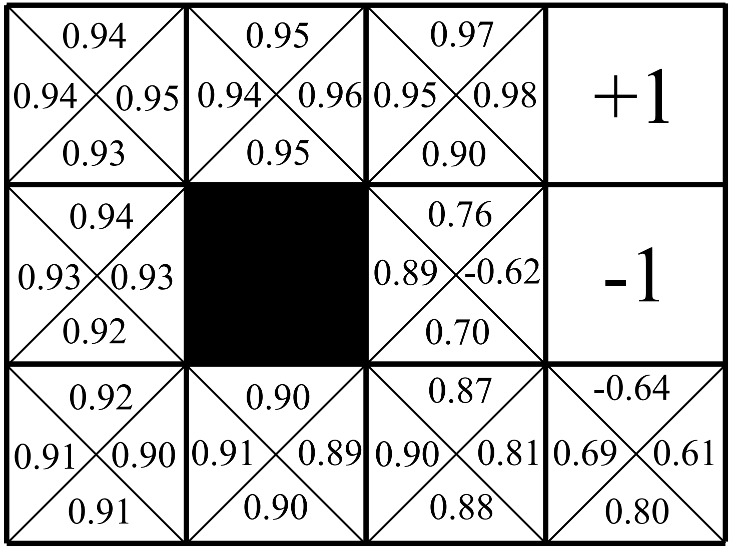
- How should we act?
- We need to do a mini-expectimax (one step)

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

- This is called policy extraction, since it gets the policy implied by the values

Policy Extraction

- Let's imagine we have the optimal q-values



- How should we act?

- Super easy:

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

- Actions are easier to select from q-values than values

Can we improve the runtime of value iteration?

Value Iteration - V_0

0.00	0.00	0.00	0.00
0.00		0.00	0.00
0.00	0.00	0.00	0.00

$R(s) = -0.10$
$\gamma = 1.0$

Value Iteration - V_1

-0.10	-0.10	-0.10	1.00
-0.10		-0.10	-1.00
-0.10	-0.10	-0.10	-0.10

$R(s) = -0.10$

$\gamma = 1.0$

Value Iteration - V_2

-0.20	-0.20	→ 0.68	1.00
-0.20		-0.20	-1.00
-0.20	-0.20	-0.20	-0.20

$R(s) = -0.10$

$\gamma = 1.0$

Value Iteration - V_3

-0.30	→ 0.40	→ 0.75	1.00
-0.30		↑ 0.32	-1.00
-0.30	-0.30	-0.30	-0.30

$R(s) = -0.10$
$\gamma = 1.0$

Value Iteration - V_4

→ 0.16	→ 0.58	→ 0.81	1.00
-0.40		↑ 0.43	-1.00
-0.40	-0.40	↑ 0.10	-0.40

R(s) = -0.10
γ = 1.0

Value Iteration - V_5

→ 0.34	→ 0.66	→ 0.82	1.00
↑ -0.05		↑ 0.49	-1.00
-0.50	→ -0.10	↑ 0.16	← -0.16

R(s) = -0.10

γ = 1.0

Value Iteration - V_6

→ 0.46	→ 0.69	→ 0.83	1.00
↑ 0.16		↑ 0.51	-1.00
↑ -0.20	→ 0.01	↑ 0.26	← -0.08

$R(s) = -0.10$
$\gamma = 1.0$

Value Iteration - V_7

→ 0.52	→ 0.70	→ 0.83	1.00
↑ 0.30		↑ 0.52	-1.00
↑ 0.01	→ 0.11	↑ 0.30	← 0.00

$R(s) = -0.10$
$\gamma = 1.0$

Value Iteration - V_8

→ 0.54	→ 0.71	→ 0.83	1.00
↑ 0.37		↑ 0.52	-1.00
↑ 0.15	→ 0.16	↑ 0.32	← 0.04

$R(s) = -0.10$
$\gamma = 1.0$

Value Iteration - V_9

→ 0.56	→ 0.71	→ 0.84	1.00
↑ 0.41		↑ 0.52	-1.00
↑ 0.23	→ 0.19	↑ 0.34	← 0.06

$R(s) = -0.10$

$\gamma = 1.0$

Value Iteration - V_9

→ 0.56	→ 0.71	→ 0.84	1.00
↑ 0.41		↑ 0.52	-1.00
↑ 0.23	→ 0.19	↑ 0.34	← 0.06

$R(s) = -0.10$

$\gamma = 1.0$

Value Iteration - V_{100}

→ 0.57	→ 0.71	→ 0.84	1.00
↑ 0.44		↑ 0.52	-1.00
↑ 0.31	→ 0.22	↑ 0.35	← 0.09

$R(s) = -0.10$

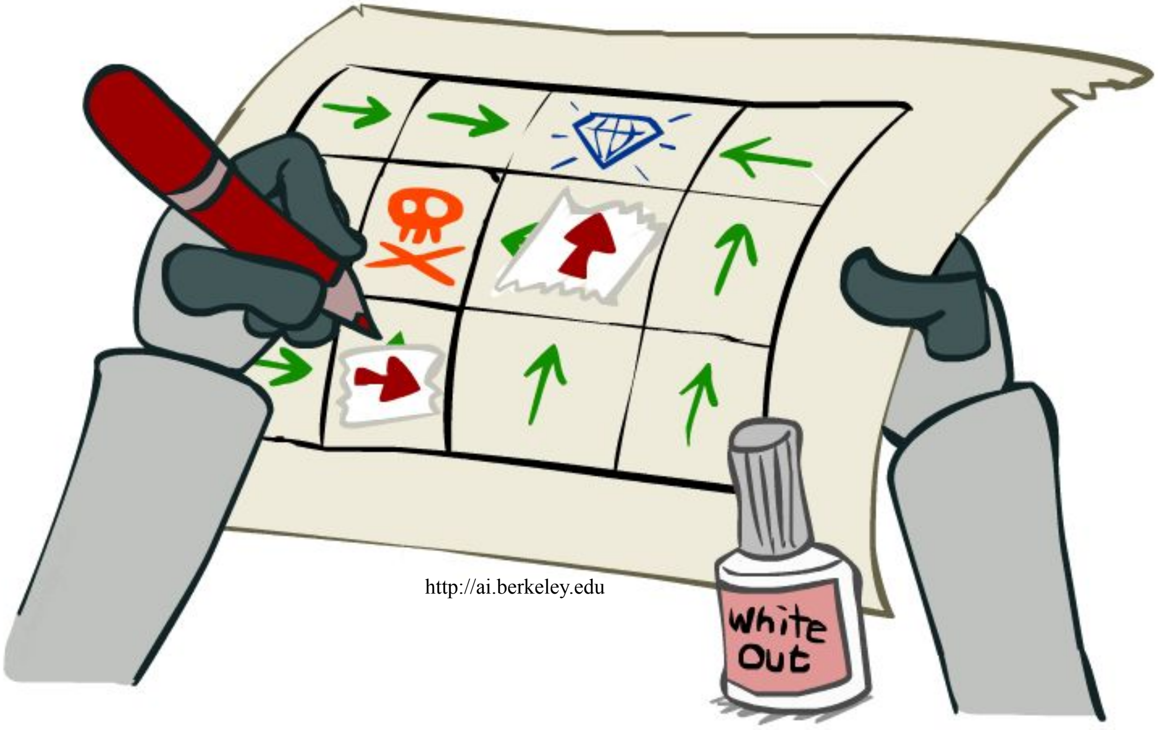
$\gamma = 1.0$

Properties of Value Iteration

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

- Slow
  - $O(S^2A)$ per iteration
- Max at each state rarely changes
  - The policy often converges long before the values

Policy Iteration



http://ai.berkeley.edu

Policy Iteration

- Policy iteration is an alternative approach for optimal values
  - Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges
- Still optimal
- Can converge (much) faster under some conditions

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation
  - Iterate until values converge

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead

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

Quiz - Policy Iteration

a	b	c	d	e
10				1

- Consider the same grid world as in the previous quiz, where east and west actions are successful 100% of the time
- $\gamma = 0.9$
- We will execute one round of policy iteration
- Consider the policy π_i shown below

exit	⟵	⟶	⟵	exit
10	9	0	0	1

- Evaluate the following quantities
  - Policy evaluation: $V^{\pi_i}(a), V^{\pi_i}(b), V^{\pi_i}(c), V^{\pi_i}(d), V^{\pi_i}(e)$
  - Policy improvement: $\pi_{i+1}(a), \pi_{i+1}(b), \pi_{i+1}(c), \pi_{i+1}(d), \pi_{i+1}(e)$

e ⟵ ⟵ ⟶ e

1+2 e ← ← ← e

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)
- Both are dynamic programs for solving MDPs

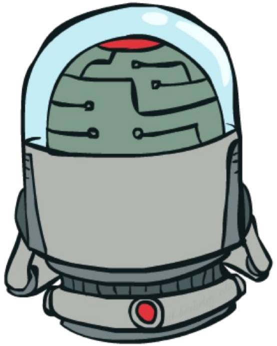
Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)
- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

MDP vs. RL

next chapter

Double-Bandits



http://ai.berkeley.edu

Double-Bandits

- An agent can play two slot machines
  - Blue, or Red

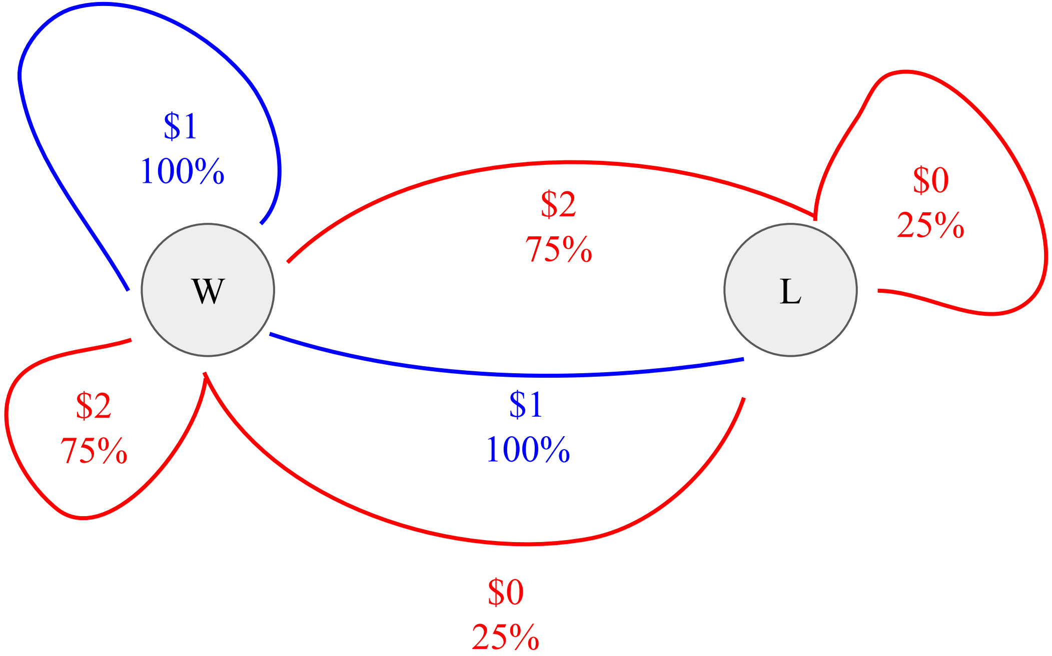
You receive $1

You receive $0 or $2, 25% and 75% of the time, respectively

- What should the agent do?

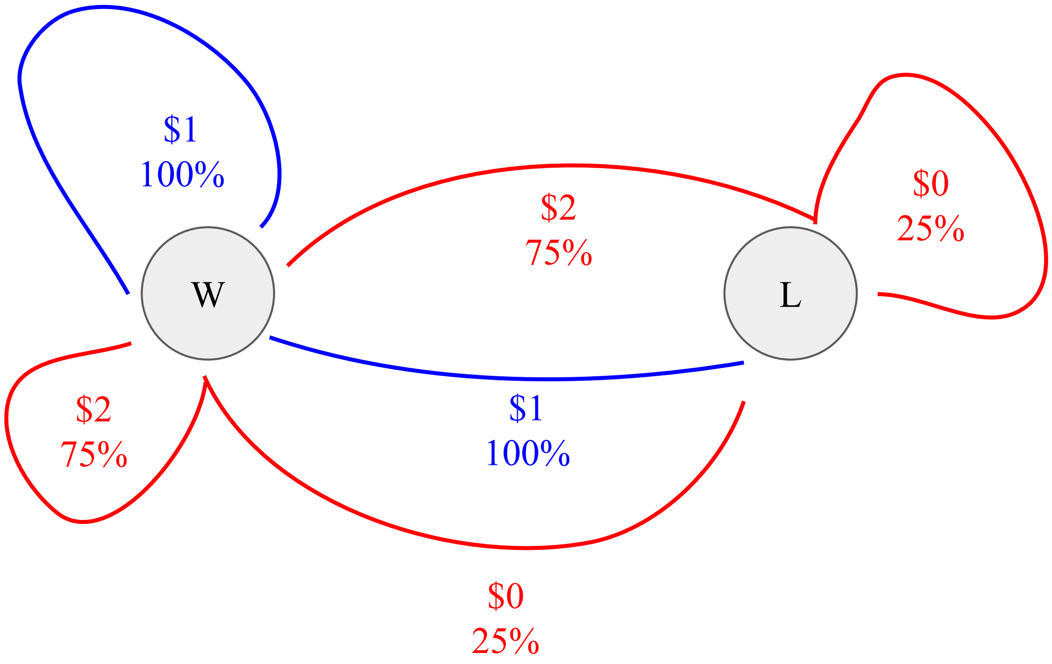
Double-Bandits MDP

- Actions: Blue and Red
- States: Win, Lose
- Assumption:
  - No discount
  - 100 time steps



Offline Planning

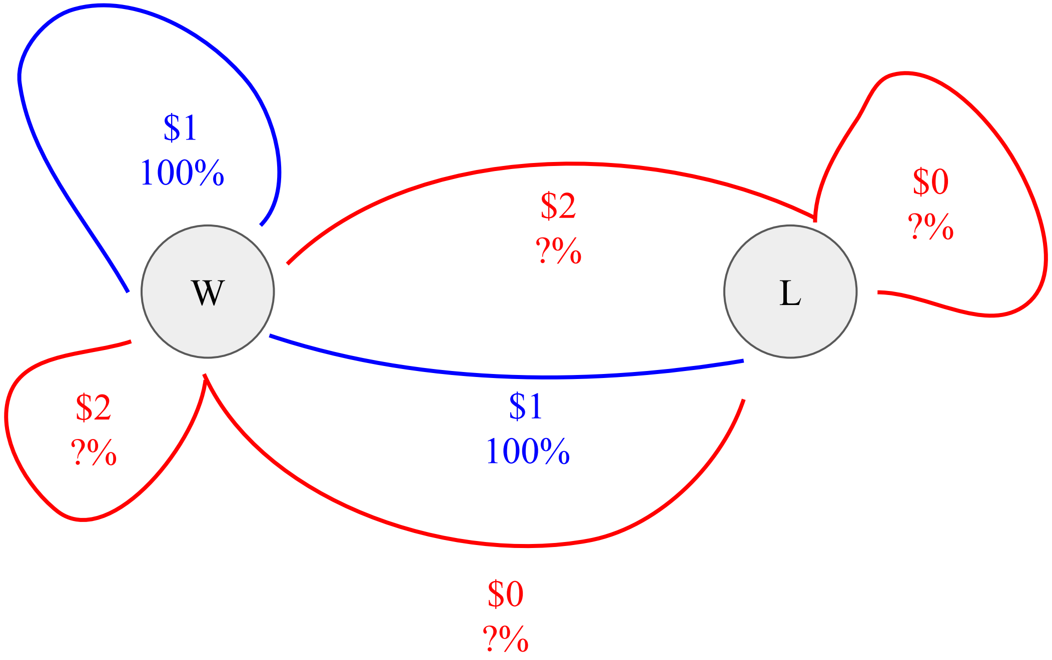
- Solving MDPs is offline planning
  - We determine all quantities through computation
  - We need to know the details of the MDP
  - We do not actually play the game



Let's Play!

$2 $0 $2 $2 $0 $0

Rules Changed!



Let’s Play!

$1

$0 $0 $0 $0 $0 $2

What Just Happened ?

- That wasn't planning, it was learning
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation !
  - You needed to actually act to figure it out
- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDPs