

Markov Decision Processes

Robert Platt
Northeastern University

Some images and slides are used from:

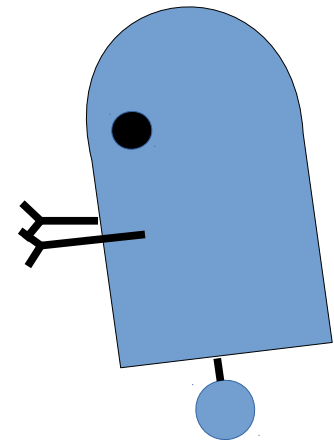
1. CS188 UC Berkeley
2. AIMA
3. Chris Amato
4. Stacy Marsella

Stochastic domains

So far, we have studied search

Can use search to solve simple planning problems,
e.g. robot planning using A*

But only in deterministic domains...

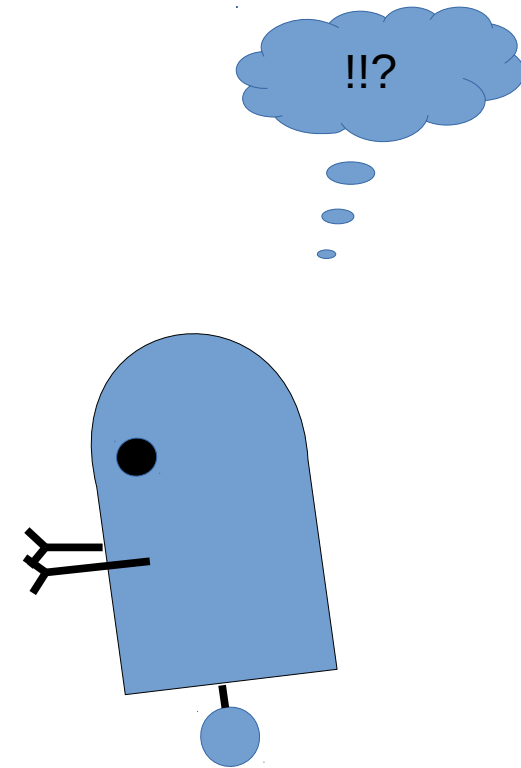


Stochastic domains

So far, we have studied search

Can use search to solve simple planning problems,
e.g. robot planning using A*

A* doesn't work so well in stochastic environments...

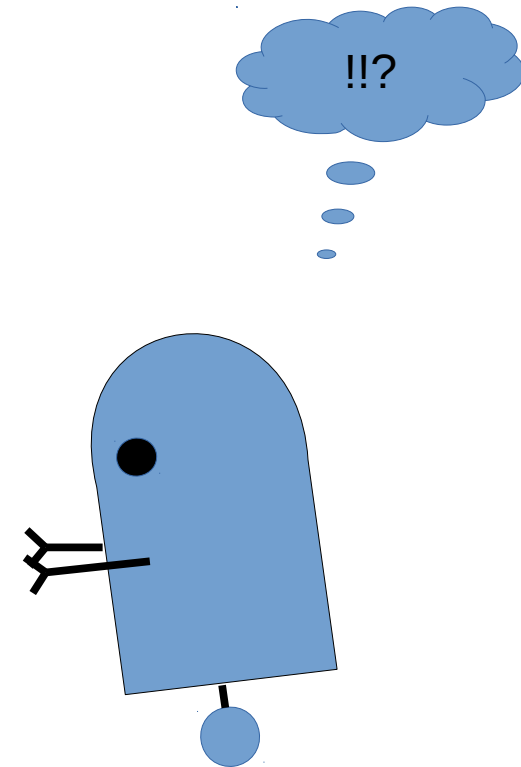


Stochastic domains

So far, we have studied search

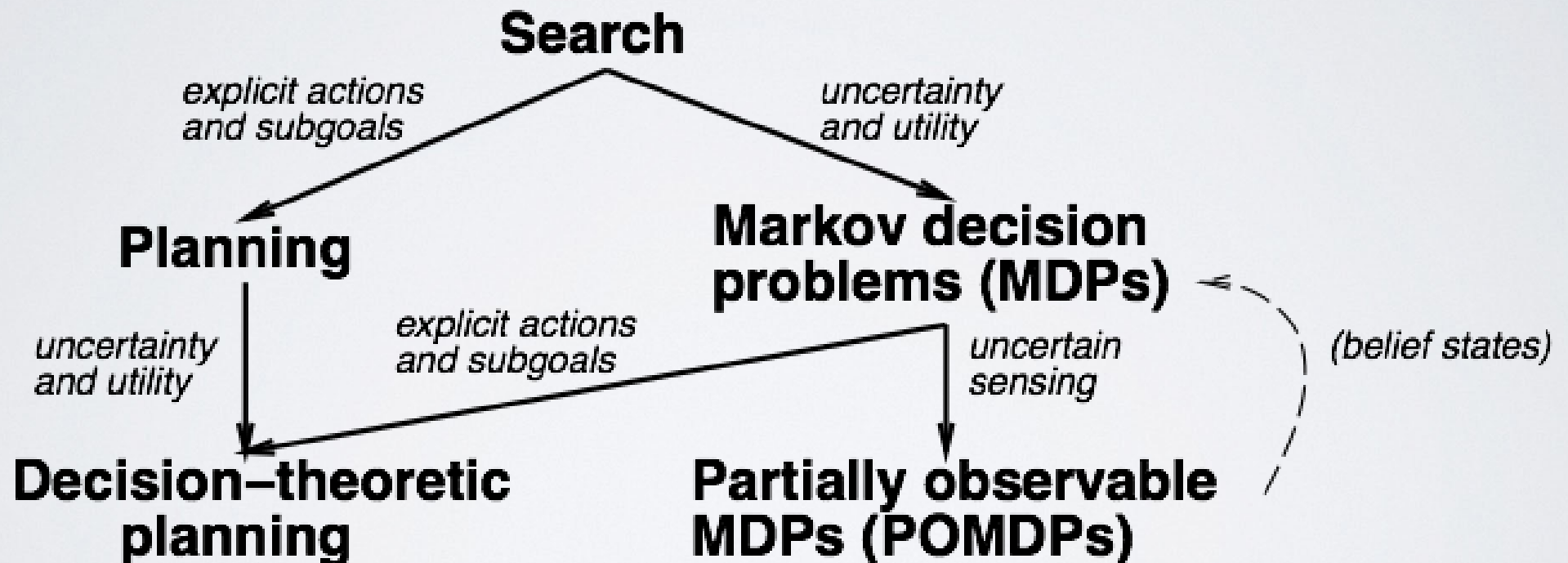
Can use search to solve simple planning problems,
e.g. robot planning using A*

A* doesn't work so well in stochastic environments...



We are going to introduce a new framework for encoding problems
w/ stochastic dynamics: the Markov Decision Process (MDP)

SEQUENTIAL DECISION- MAKING

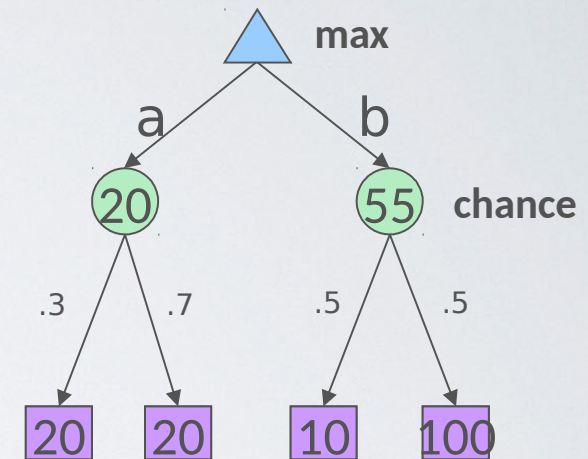


MAKING DECISIONS UNDER UNCERTAINTY

- Rational decision making requires reasoning about one's *uncertainty* and *objectives*
- Previous section focused on uncertainty
- This section will discuss how to make rational decisions based on a *probabilistic model* and *utility function*
- Last class, we focused on single step decisions, now we will consider sequential decision problems

REVIEW: EXPECTIMAX

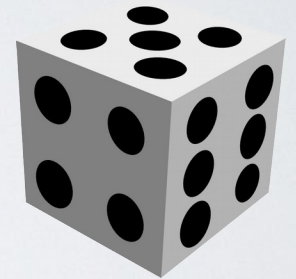
- What if we don't know the outcome of actions?
 - Actions can fail
 - when a robot moves, it's wheels might slip
 - Opponents may be uncertain



- **Expectimax search:** maximize average score
 - MAX nodes choose action that maximizes outcome
 - Chance nodes model an outcome (a value) that is uncertain
 - Use **expected utilities**
 - weighted average (expectation) of children

REVIEW: PROBABILITY AND EXPECTED UTILITY

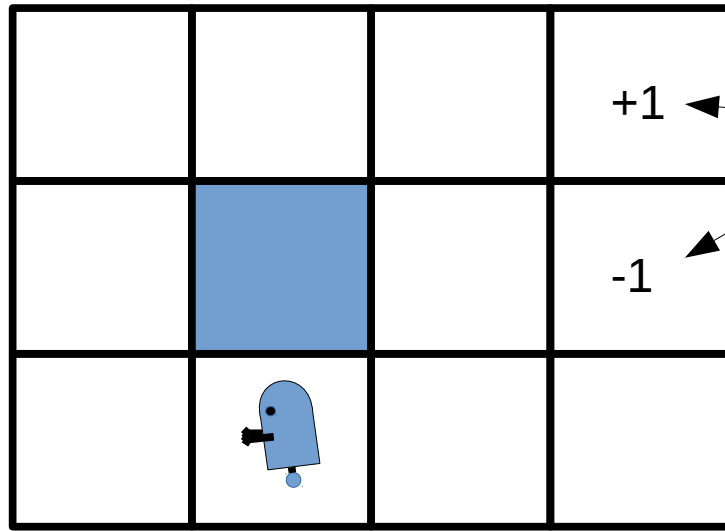
- $EU = \sum \text{probability}(\text{outcome}) * \text{value}(\text{outcome})$
- Expected utility is the probability-weighted average of all possible values
 - I.e., each possible value is multiplied by its probability of occurring and the resulting products are summed
- What is the expected value of rolling a six-sided die if you threw the die MANY times?
- $(1/6 * 1) + (1/6 * 2) + (1/6 * 3) + (1/6 * 4) + (1/6 * 5) + (1/6 * 6) = \mathbf{3.5}$



DIFFERENT APPROACH IN SEQUENTIAL DECISION MAKING

- In deterministic planning, our agents generated entire plans
 - Entire sequence of actions from start to goals
 - Under assumption environment was deterministic, actions were reliable
- In Expectimax, chance nodes model nondeterminism
 - But agent only determined best next action with a bounded horizon
- Now we consider agents who use a “Policy”
 - A strategy that determines what action to take in any state
 - Assuming unreliable action outcomes & infinite horizons

Markov Decision Process (MDP): grid world example



Rewards:

- agent gets these rewards in these cells
- goal of agent is to maximize reward

Actions: left, right, up, down

- take one action per time step
- actions are stochastic: only go in intended direction 80% of the time

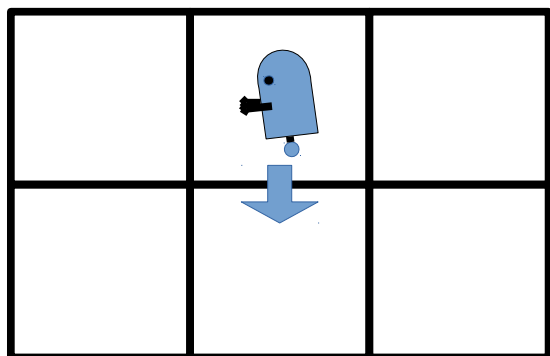
States:

- each cell is a state

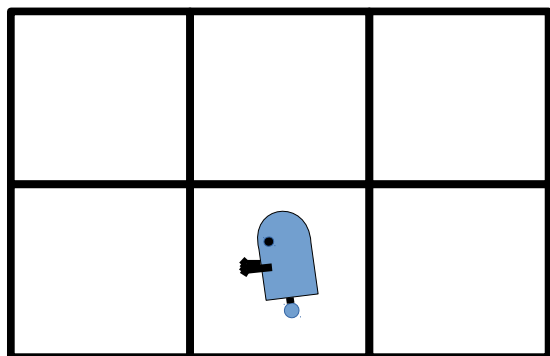
Markov Decision Process (MDP)

Deterministic

– same action always has same outcome

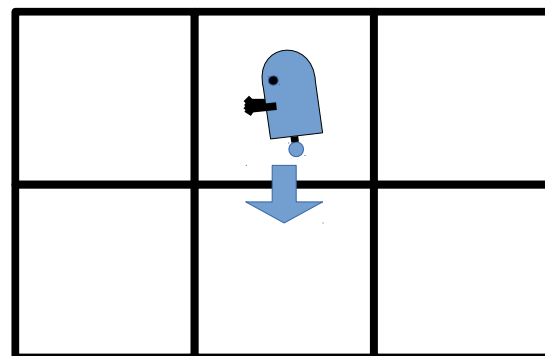


1.0



Stochastic

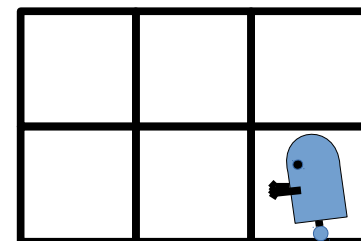
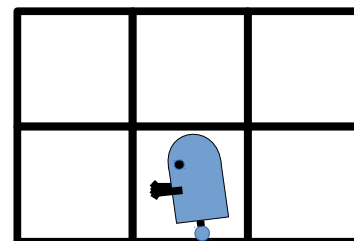
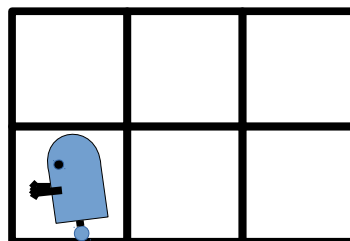
– same action could have different outcomes



0.1

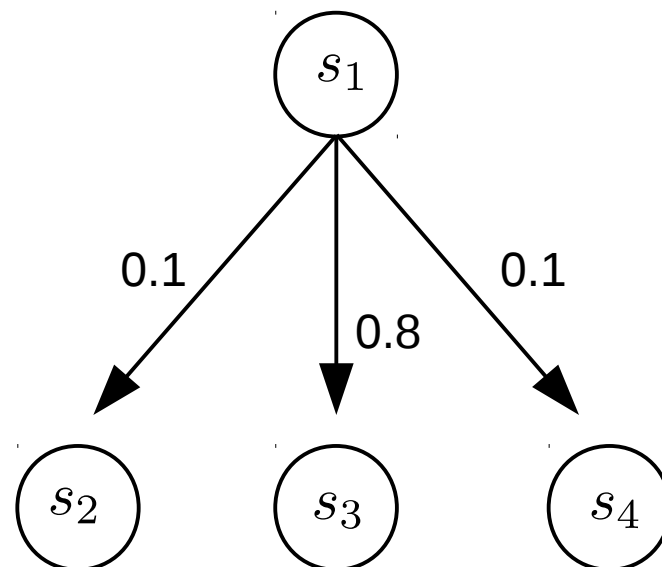
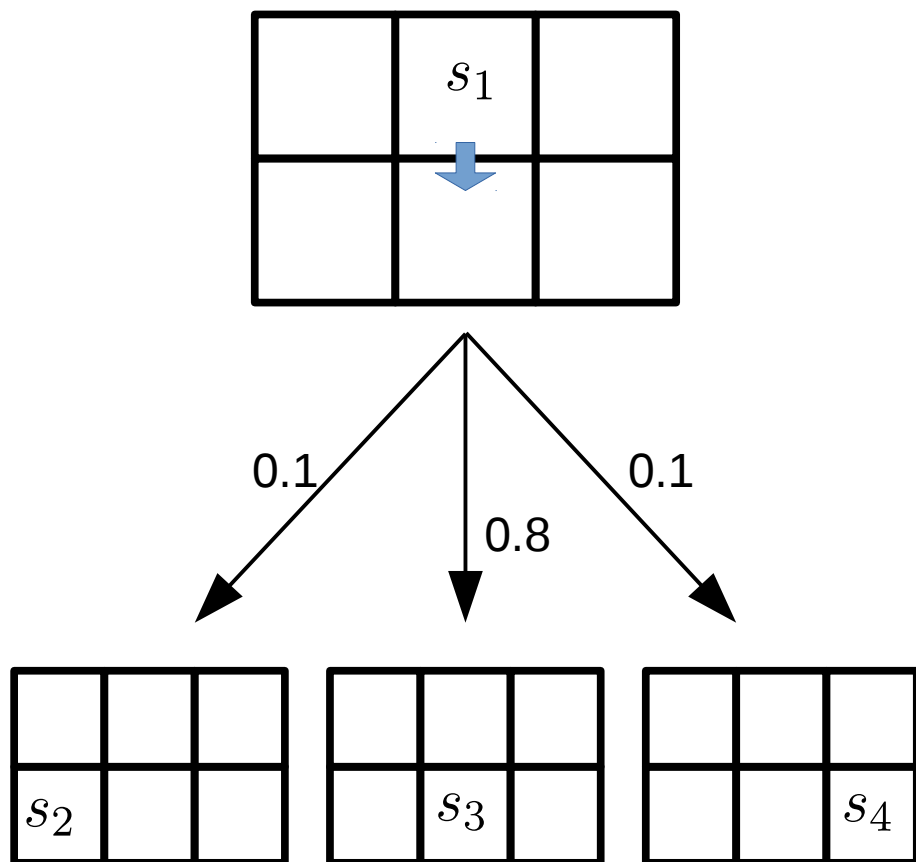
0.8

0.1



Markov Decision Process (MDP)

Same action could have different outcomes:



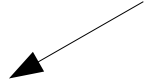
Transition function at s_1 :

s'	$T(s,a,s')$
s_2	0.1
s_3	0.8
s_4	0.1

Markov Decision Process (MDP)

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process)
defines a stochastic control problem:

$$M = (S, A, T, R)$$


State set: $s \in S$

Action Set: $a \in A$

Transition function: $T : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$

Reward function: $R : S \times A \rightarrow \mathbb{R}_{\geq 0}$

Markov Decision Process (MDP)

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process)
defines a stochastic control problem:

$$M = (S, A, T, R)$$

State set: $s \in S$

Action Set: $a \in A$

Transition function: $T : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$

Reward function: $R : S \times A \rightarrow \mathbb{R}_{\geq 0}$

Probability of going from s to s'
when executing action a

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

Markov Decision Process (MDP)

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process) defines a stochastic control problem:

$$M = (S, A, T, R)$$

State set: $s \in S$

Action Set: $a \in A$

Transition function: $T : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$

Reward function: $R : S \times A \rightarrow \mathbb{R}_{\geq 0}$

Probability of going from s to s' when executing action a

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

But, what is the objective?

Markov Decision Process (MDP)

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process) defines a stochastic control problem:

$$M = (S, A, T, R)$$

State set: $s \in S$

Action Set: $a \in A$

Transition function: $T : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$

Reward function: $R : S \times A \rightarrow \mathbb{R}_{\geq 0}$

Probability of going from s to s' when executing action a

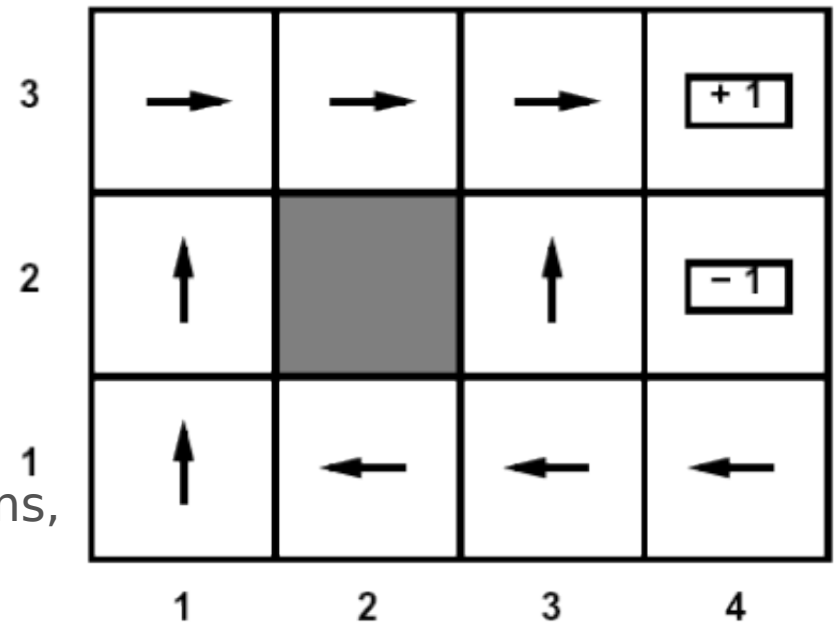
$$\sum_{s' \in S} T(s, a, s') = 1$$

Objective: calculate a strategy for acting so as to maximize the future rewards.

– we will calculate a *policy* that will tell us how to act

What is a policy?

- We want an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$
 - A policy gives an action for each state
 - An optimal policy is one that maximizes expected utility if followed
- For Deterministic single-agent search problems, derived an optimal **plan**, or sequence of actions, from start to a goal
- For Expectimax, didn't compute entire policies
 - It computed the action for a single state only
 - Over a limited horizon
 - Final rewards only



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

What is a policy?

A policy tells the agent what action to execute as a function of state:

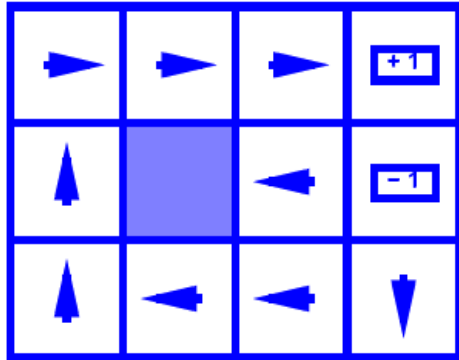
Deterministic policy: $\pi(s) : S \rightarrow A$

- agent always executes the same action from a given state

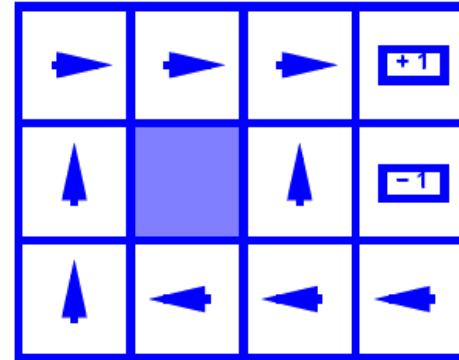
Stochastic policy: $\pi(s, a) : S \times A \rightarrow \mathbb{R}$

- agent selects an action to execute by drawing from a *probability distribution* encoded by the policy ...

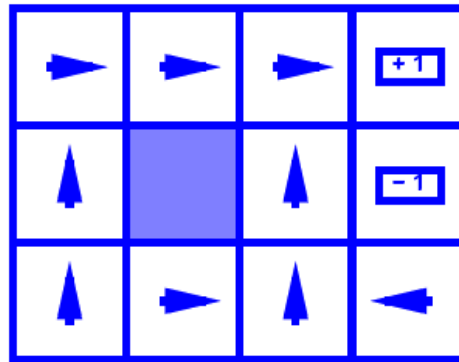
Examples of optimal policies



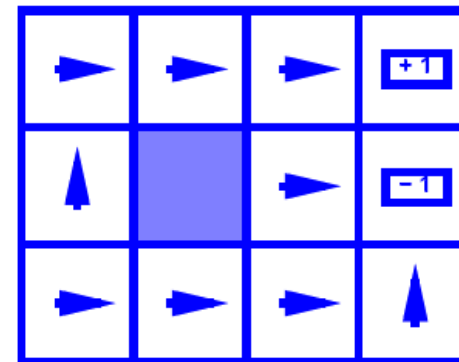
$$R(s) = -0.01$$



$$R(s) = -0.03$$



$$R(s) = -0.4$$



$$R(s) = -2.0$$

Markov?

- “Markovian Property”
 - Given the present state, the future and the past are independent
- For Markov decision processes, “Markov” means action outcomes depend only on the current state

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

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

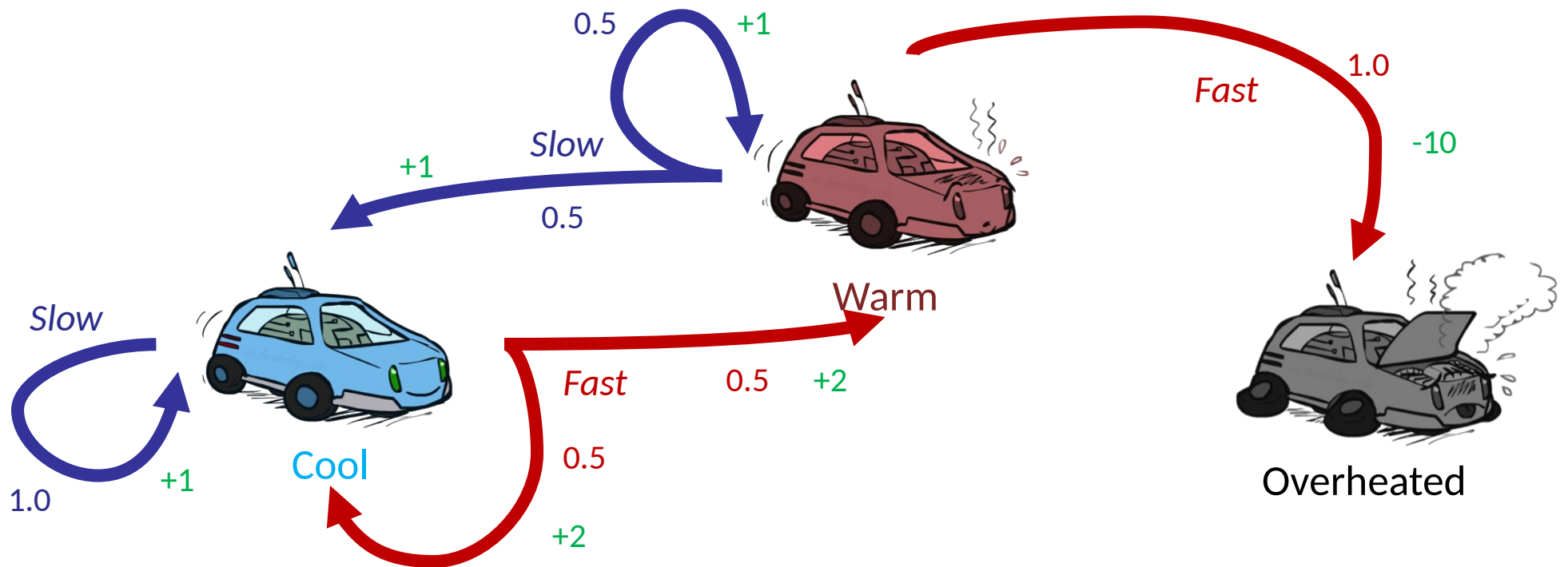
- This is just like search, where the successor function could only depend on the current state (not the history)



Andrey Markov
(1856-1922)

Another example of an MDP

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



Objective: maximize expected future reward

$R_t \equiv$ Expected future reward starting at time t

$$= \mathbb{E} \sum_{i=0}^{\infty} r_{t+i}$$

Objective: maximize expected future reward

$R_t \equiv$ Expected future reward starting at time t

$$= \mathbb{E} \sum_{i=0}^{\infty} r_{t+i}$$



What's wrong w/ this?

Objective: maximize expected future reward

$R_t \equiv$ Expected future reward starting at time t

$$= \mathbb{E} \sum_{i=0}^{\infty} r_{t+i}$$

What's wrong w/ this?

Two viable alternatives:

1. maximize expected future reward *over the next T timesteps* (finite horizon):

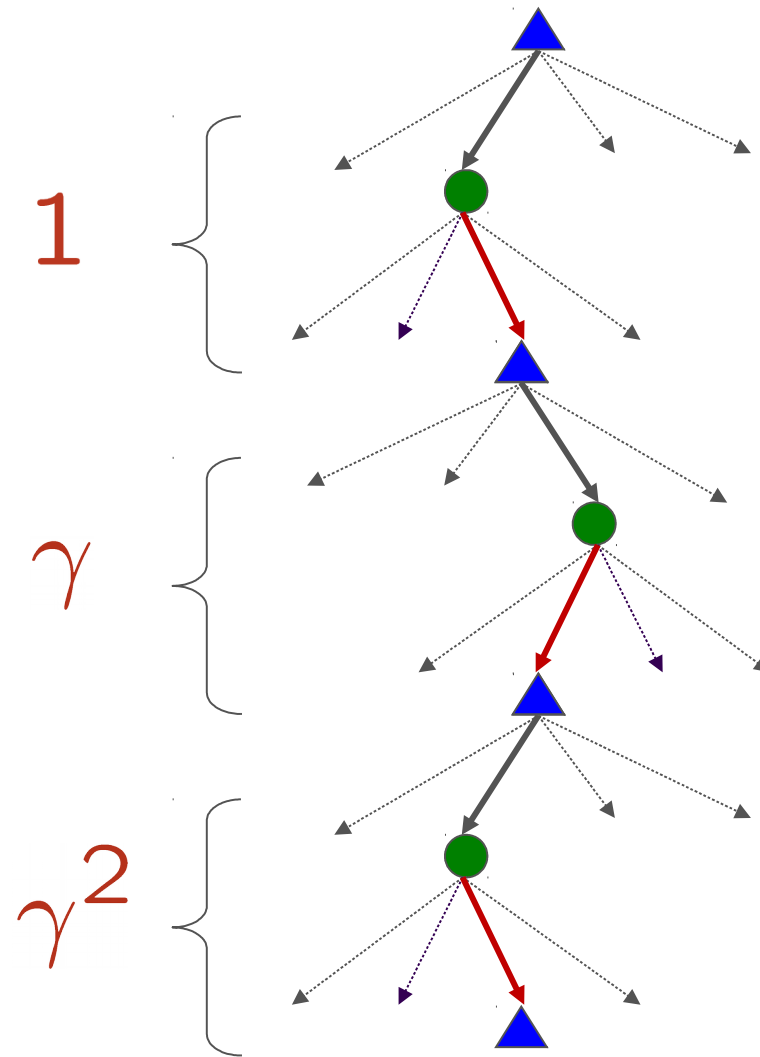
$$R_t = \mathbb{E} \sum_{i=0}^T r_{t+i}$$

2. maximize expected *discounted* future rewards:

$$R_t = \mathbb{E} \sum_{i=0}^{\infty} \gamma^i r_{t+i} = \mathbb{E} [r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots]$$

Discount factor (usually around 0.9): $0 < \gamma < 1$

Discounting



STATIONARY PREFERENCES

- Theorem: if we assume **stationary preferences**:

$$[a_1, a_2, \dots] \succ [b_1, b_2, \dots]$$

$$\Updownarrow$$

$$[r, a_1, a_2, \dots] \succ [r, b_1, b_2, \dots]$$

- Then: there are only two ways to define utilities

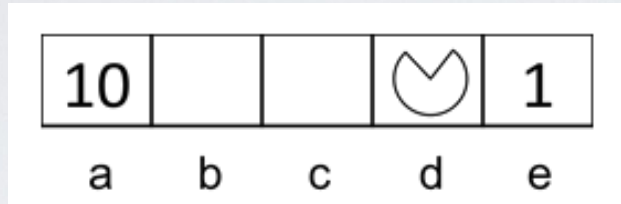
$$U([r_0, r_1, r_2, \dots]) = r_0 + r_1 + r_2 + \dots$$

- Additive utility: $U([r_0, r_1, r_2, \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \dots$

- Discounted utility:

QUIZ: DISCOUNTING

- Given:



- Actins: East, West, and Exit (only available in exit states a, e)
- Transitions: deterministic
- Quiz 1: For $\gamma = 1$, what is the optimal policy?
- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?
- Quiz 3: For which γ are West and East equally good when in state d?

UTILITIES OVER TIME: FINITE OR INFINITE HORIZON?

- If there is fixed time, N , after which nothing can happen, what should an agent do?
 - E.g., if $N=3$, Bot must head directly for +1 state
 - If $N = 100$, can take safe route
- So with finite horizon, optimal action changes over time
 - Optimal policy is nonstationary
 - (π^* depends on time left)

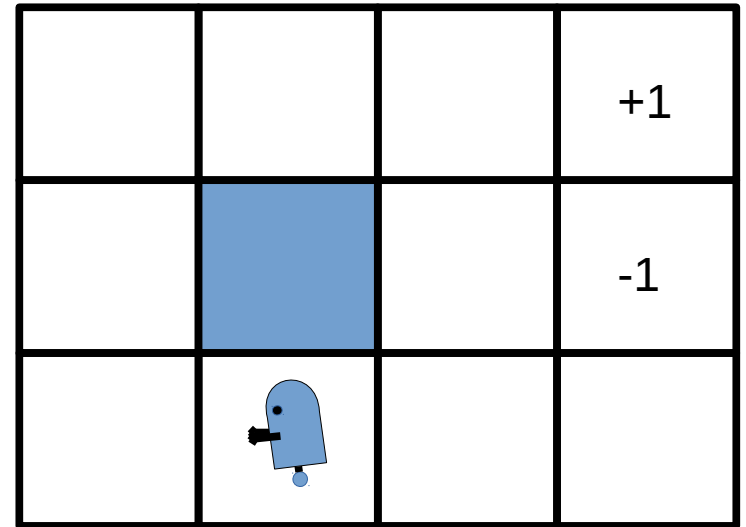
Choosing a reward function

A few possibilities:

- all reward on goal
- negative reward everywhere except terminal states
- gradually increasing reward as you approach the goal

In general:

- reward can be whatever you want



Value functions

Value function

$V(s) \equiv$ Expected discounted reward if agent acts optimally starting in state s .

$Q(s, a) \equiv$ Expected discounted reward if agent acts optimally after taking action a from state s .

Action value function

Game plan:

1. calculate the optimal value function
2. calculate optimal policy from optimal value function

Grid world optimal value function

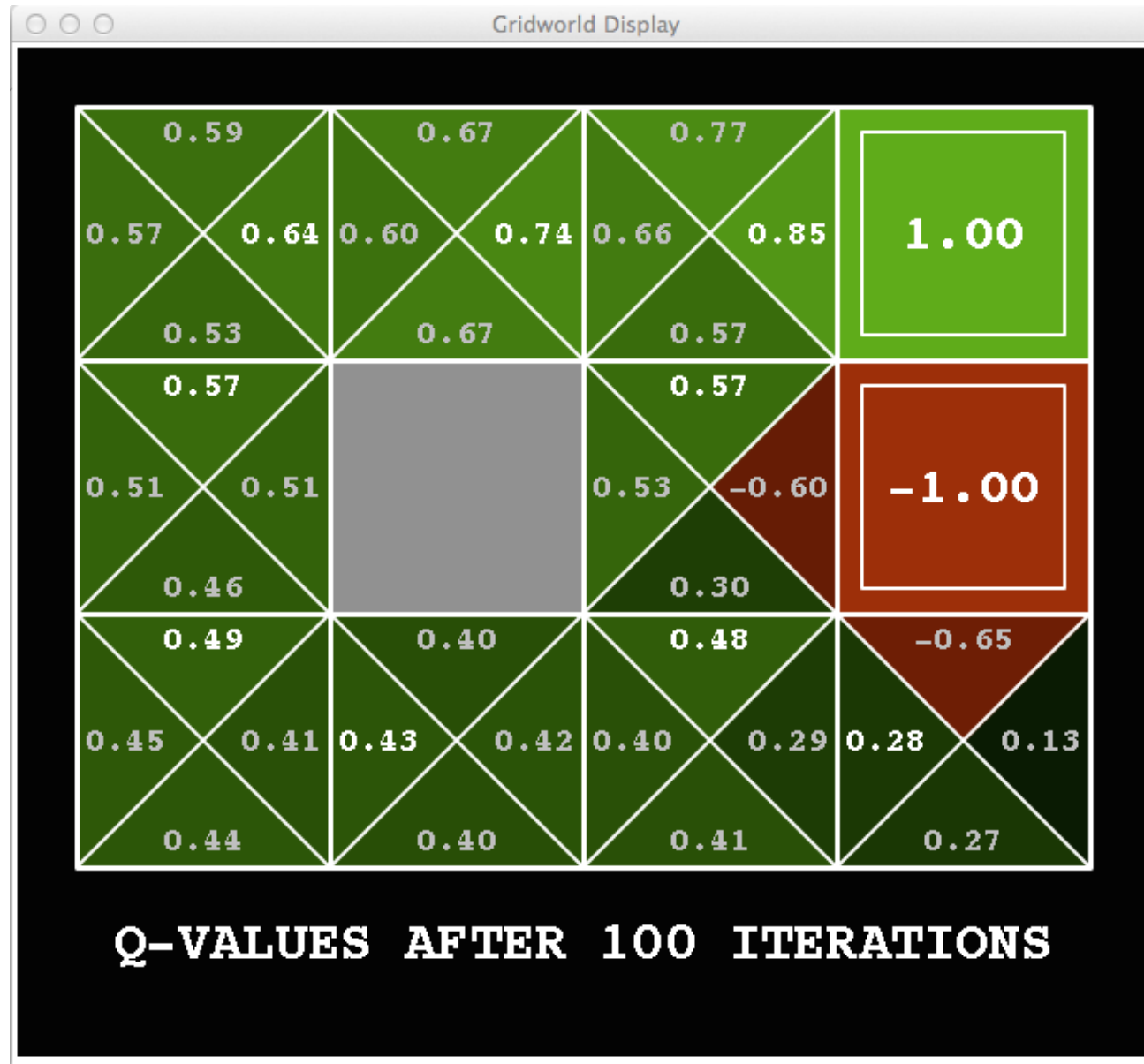


Noise = 0.2

Discount = 0.9

Living reward = 0

Grid world optimal action-value function



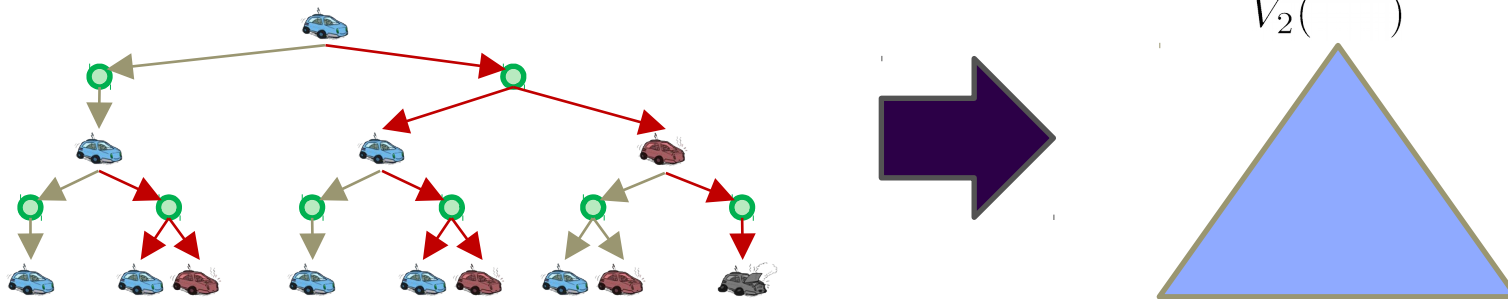
Noise = 0.2

Discount = 0.9

Living reward = 0

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



Value iteration

Value iteration calculates the time-limited value function, V_i :

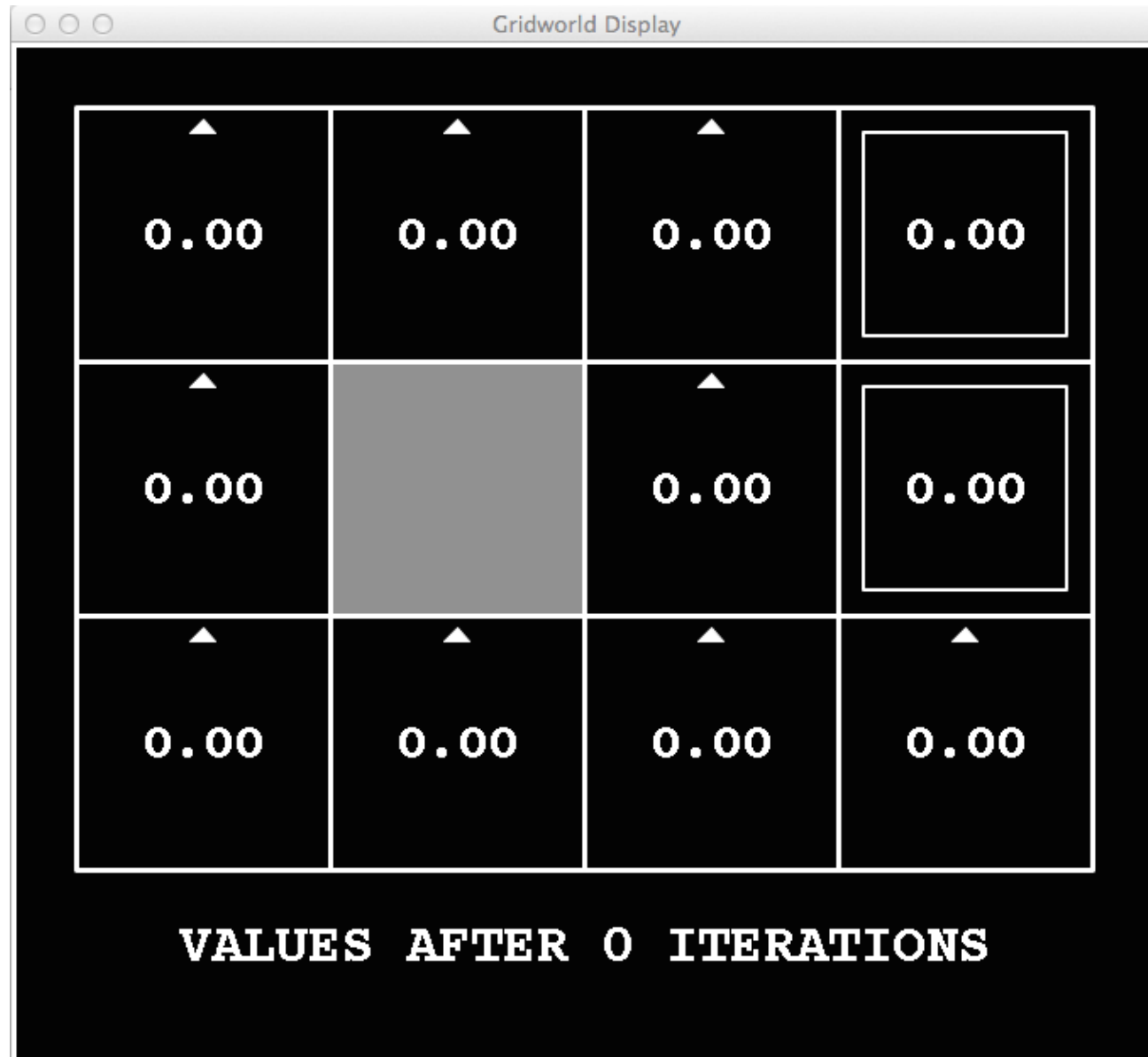
Value Iteration

Input: MDP = (S, A, T, r)

Output: value function, V

1. let $\forall s \in S, V_0(s) = V_{init}$
2. for $i=0$ to infinity
3. for all $s \in S$
4.
$$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') [r(s, a) + \gamma V_i(s')]$$
5. if V converged, then break

Value iteration example

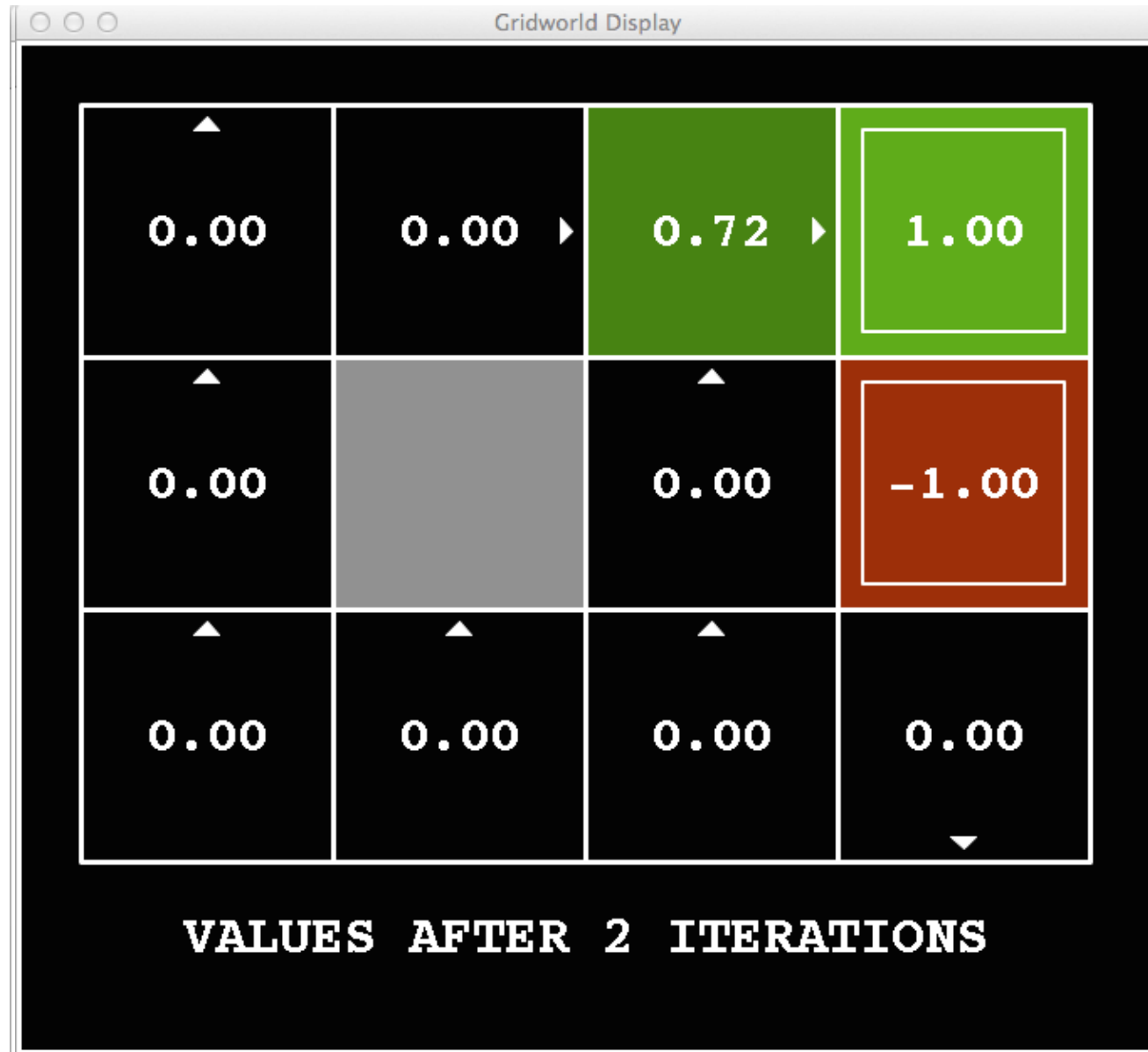


Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration example



Value iteration example



Value iteration example



Value iteration example



Value iteration example



Value iteration example



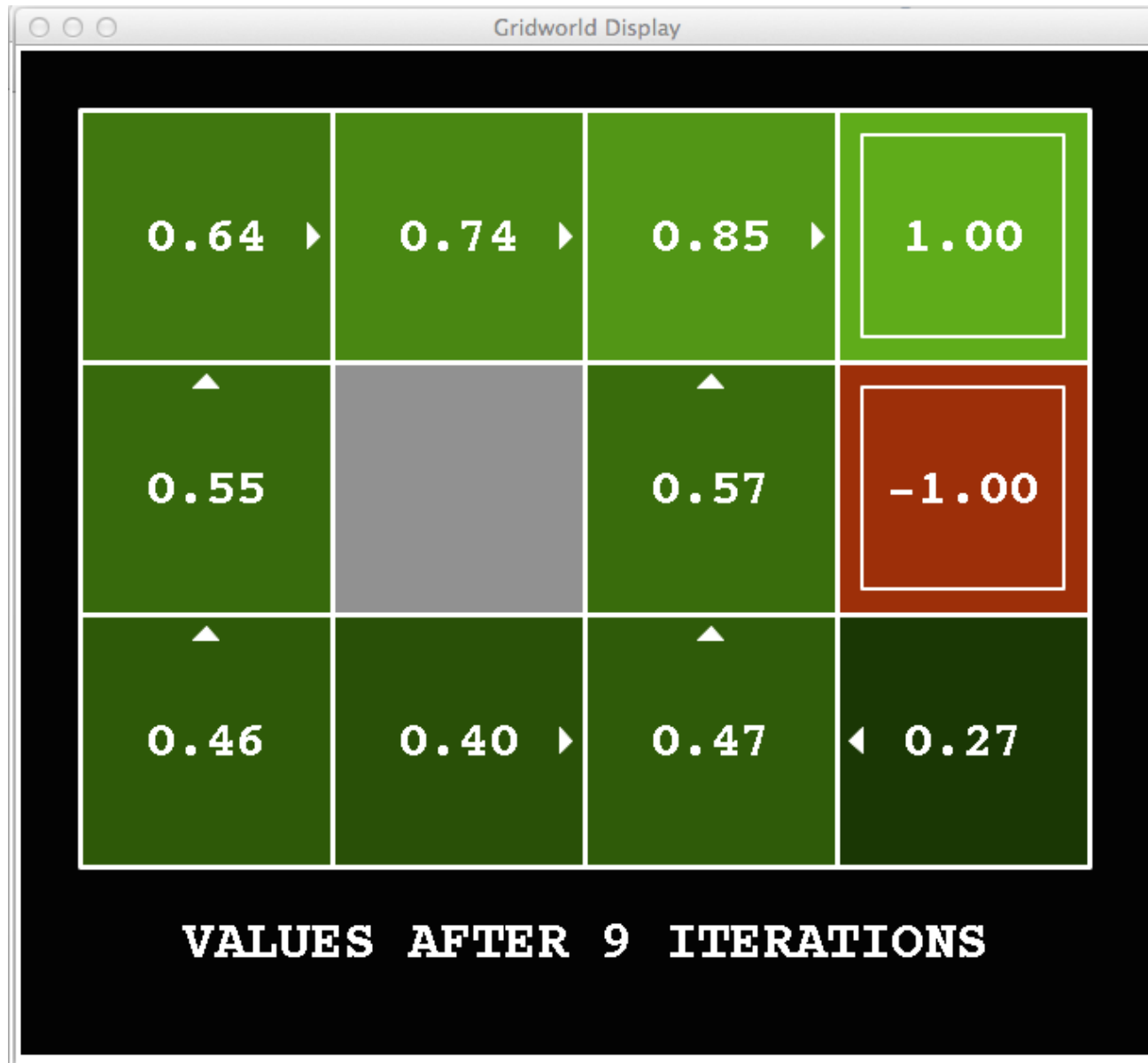
Value iteration example



Value iteration example



Value iteration example



Value iteration example



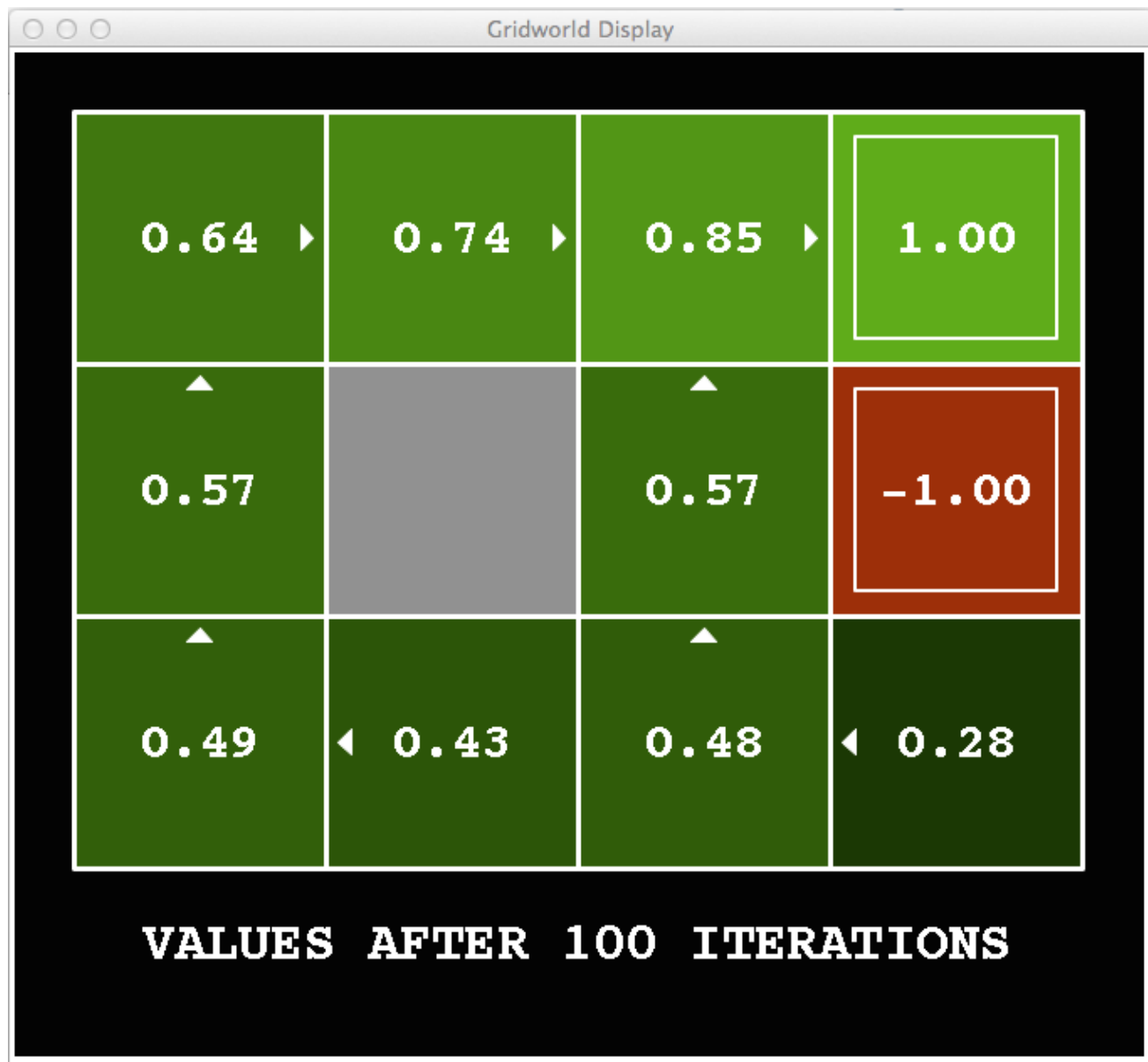
Value iteration example



Value iteration example



Value iteration example



Value iteration

Value Iteration

Input: MDP = (S, A, T, r)

Output: value function, V

1. let $\forall s \in S, V_0(s) = V_{init}$

2. for i=0 to infinity

3. for all $s \in S$

4.
$$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') [r(s, a) + \gamma V_i(s')]$$

5. if V converged, then break

Value iteration

Value Iteration

Input: MDP=(S, A, T, r)

Output: value function, V

1. let $\forall s \in S, V_0(s) = V_{init}$

2. for $i=0$ to infinity

3. for all $s \in S$

4.
$$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') [r(s, a) + \gamma V_i(s')]$$

5. if V converged, then break

Let's look at this eqn more closely...

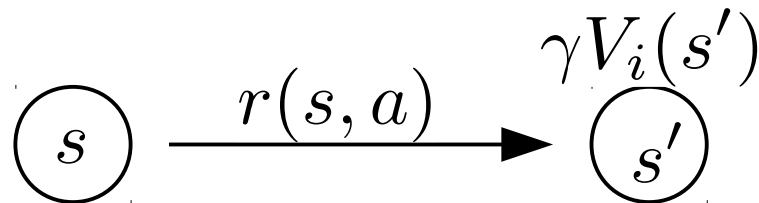
Value iteration

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

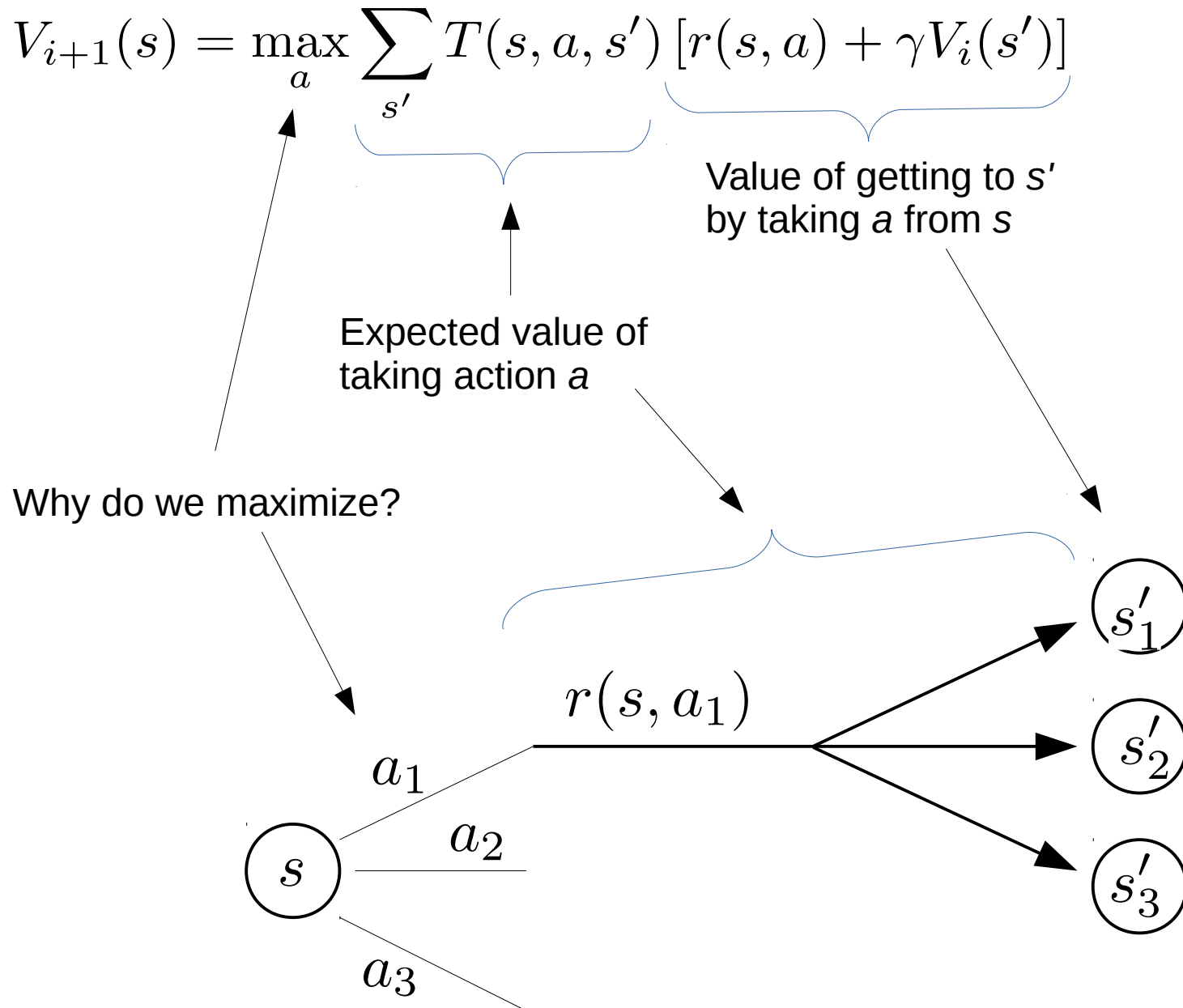
Value of getting to s' by taking a from s : $r(s, a) + \gamma V_i(s')$

reward obtained on this time step

discounted value of being at s'



Value iteration



Value iteration

Value of s at k timesteps to go: $V_k(s)$

Value iteration:

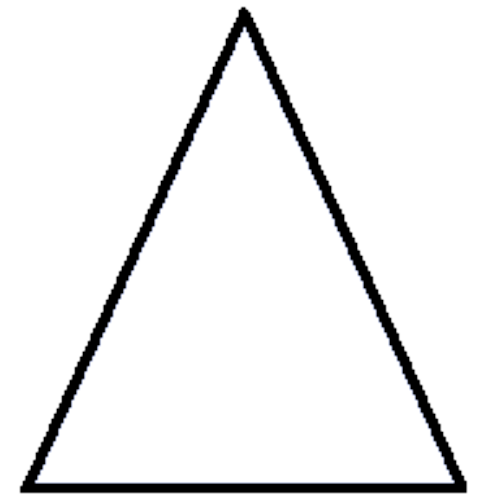
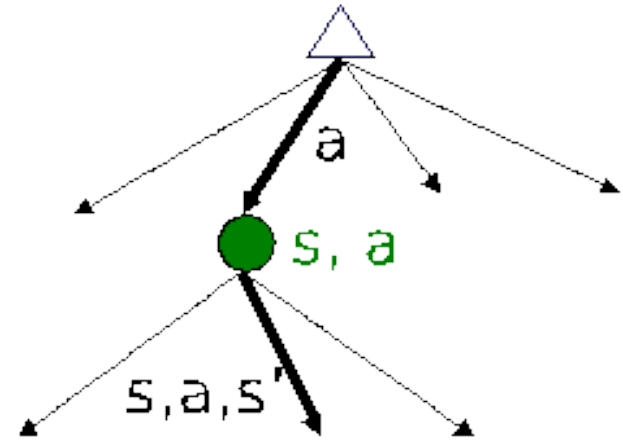
1. initialize $V_0(s) = 0$

2. $V_1(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_0(s')]$

3. $V_2(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_1(s')]$

4.

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



Value iteration



V_2

3.5	2.5	0
-----	-----	---

$$S = 1.0 [1 + V_1(c)]$$

$$F = .5 [2 + V_1(c)] + .5 [2 + V_1(w)]$$

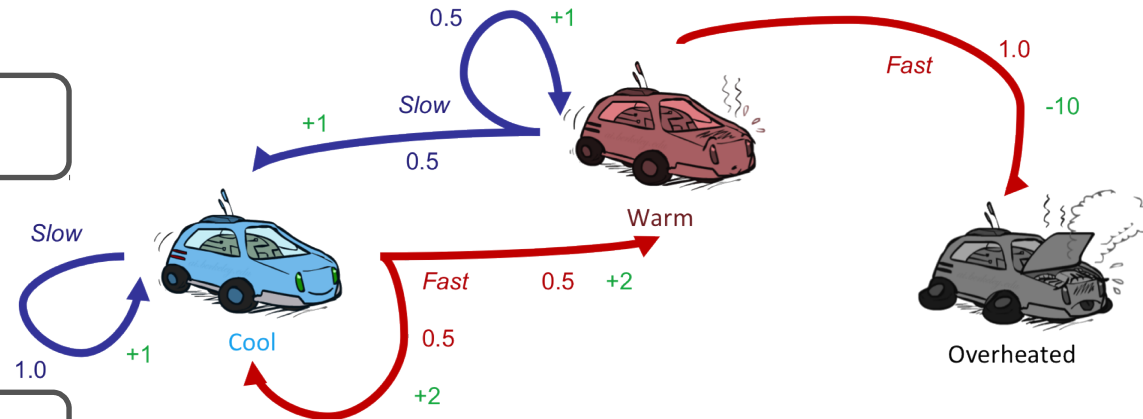
V_1

2	1	0
---	---	---

$$S = 1.0 [1 + V_0(c)]$$

$$F = .5 [2 + V_0(c)] + .5 [2 + V_0(w)]$$

0	0	0
---	---	---



Assume no discount!

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

Value iteration

Value Iteration

Input: MDP = (S, A, T, r)

Output: value function, V

1. let $\forall s \in S, V_0(s) = V_{init}$

2. for i=0 to infinity

3. for all $s \in S$

4.
$$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') [r(s, a) + \gamma V_i(s')]$$

5. if V converged, then break

How do we know that this converges?

How do we know that this converges to the optimal value function?

Convergence

- How do we know the V_k vectors are going to converge?
- Case 1: If the tree has maximum depth M , then V_M holds the actual untruncated values
- Case 2: If the discount is less than 1
 - Sketch: For any state V_k and V_{k+1} can be viewed as depth $k+1$ expectimax results in nearly identical search trees
 - The last layer is at most all R_{MAX} and at least R_{MIN}
 - But everything is discounted by γ^k that far out
 - So V_k and V_{k+1} are at most $\gamma^k \max |R_{MAX} - R_{MIN}|$ different
 - So as k increases, the values converge

Optimality

At convergence, this property must hold (why?)



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

What does this equation tell us about optimality of value iteration?

– we denote the *optimal* value function as: V^*

Bellman Equation

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

- With this equation, Bellman introduced dynamic programming in 1953
- Will be the focus of the next few lectures



Richard Bellman
1920 –1984

Gauss-Siedel Value Iteration

Value Iteration

Input: MDP = (S, A, T, r)

Output: value function, V

1. let $\forall s \in S, V_1(s) = V_{init}$


2. for i=1 to infinity

3. for all $s \in S$

4.
$$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') [r(s, a) + \gamma V_i(s')]$$

5. if V converged, then break

Regular value iteration maintains two V arrays: old V and new V



Gauss-Siedel maintains only one V matrix.

- each update is immediately applied
- can lead to faster convergence

Computing a policy from the value function

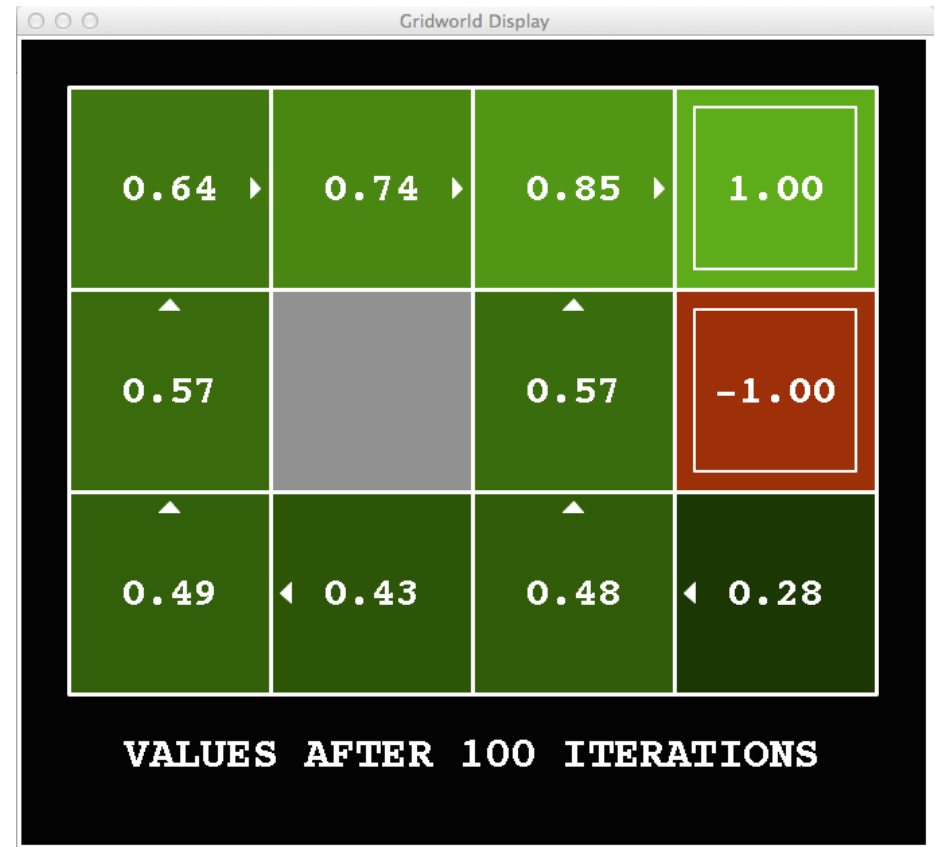
Notice these little arrows



The arrows denote a policy
– how do we calculate it?

Computing a policy from the value function

Given values calculated using value iteration, do one step of expectimax:



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

The optimal policy is implied by the optimal value function...

Stochastic policies vs deterministic policies

In general, a policy is a distribution over actions: $\pi(s, a) : S \times A \rightarrow \mathbb{R}$

Here, we restrict consideration to deterministic policies: $\pi(s) : S \rightarrow A$

Problems with value iteration

Problem 1: It's slow – $O(S^2A)$ per iteration

Problem 2: The “max” at each state rarely changes

Problem 3: The policy often converges long before the values

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 (much) faster under some conditions

Policy evaluation

What if you want to calculate the value function for a given sub-optimal policy?

Answer: Policy Evaluation!

Value Iteration

Input: MDP = (S, A, T, r)

Output: value function, V

1. let $\forall s \in S, V_0(s) = V_{init}$
2. for i=0 to infinity
3. for all $s \in S$
4.
$$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') [r(s, a) + \gamma V_i(s')]$$
5. if V converged, then break

Policy evaluation

What if you want to calculate the value function for a given sub-optimal policy?

Answer: Policy Evaluation!

Policy Evaluation

Input: MDP = (S, A, T, r), π

Output: value function, V

1. let $\forall s \in S, V_0(s) = V_{init}$

2. for i=0 to infinity

3. for all $s \in S$

4.
$$V_{i+1}(s) = \sum_{s'} T(s, \pi(s), s') [r(s, \pi(s)) + \gamma V_i(s')]$$

5. if V converged, then break

Policy evaluation

What if you want to calculate the value function for a given sub-optimal policy?

Answer: Policy Evaluation!

Policy Evaluation

Input: MDP = (S, A, T, r), π

Output: value function, V

1. let $\forall s \in S, V_0(s) = V_{init}$

2. for i=0 to infinity

3. for all $s \in S$

4.
$$V_{i+1}(s) = \sum_{s'} T(s, \pi(s), s') [r(s, \pi(s)) + \gamma V_i(s')]$$

5. if V converged, then break



Notice this

Policy evaluation

What if you want to calculate the value function for a given sub-optimal policy?

Answer: Policy Evaluation!

Policy Evaluation

Input: MDP = (S, A, T, r) , π

Output: value function, V

1. let $\forall s \in S, V_0(s) = V_{init}$

2. for $i=0$ to infinity

3. for all $s \in S$

4.
$$V_{i+1}(s) = \sum_{s'} T(s, \pi(s), s') [r(s, \pi(s)) + \gamma V_i(s')]$$

5. if V converged then break

Notice this

OR: can solve for value function as the sol'n to a system of linear equations
– can't do this for value iteration because of the maxes

Policy iteration: example

Always Go Right



Always Go Forward



Modified policy iteration

Policy iteration often converges in few iterations, but each is expensive

Idea: use a few steps of value iteration (but with π fixed) starting from the value function produced the last time to produce an approximate value determination step.

Often converges much faster than pure VI or PI

Leads to much more general algorithms where Bellman value updates and Howard policy updates can be performed locally in any order

Reinforcement learning algorithms operate by performing such updates based on the observed transitions made in an initially unknown environment

Online methods

Solving for a full policy offline is expensive!

What can we do?

Online methods

Online methods compute optimal action from current state

Expand tree up to some horizon

States reachable from the current state is typically small compared to full state space

Heuristics and branch-and-bound techniques allow search space to be pruned

Monte Carlo methods provide approximate solutions

Forward search

Provides optimal action from current state s up to depth d

Recall
$$V(s) = \max_{a \in A(s)} \left[R(s, a) + \gamma \sum_{s'} T(s, a, s') V(s') \right]$$

Algorithm 4.6 Forward search

```
1: function SELECTACTION( $s, d$ )
2:   if  $d = 0$ 
3:     return (NIL, 0)
4:    $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$ 
5:   for  $a \in A(s)$ 
6:      $v \leftarrow R(s, a)$ 
7:     for  $s' \in S(s, a)$ 
8:        $(a', v') \leftarrow \text{SELECTACTION}(s', d - 1)$ 
9:        $v \leftarrow v + \gamma T(s' | s, a) v'$ 
10:    if  $v > v^*$ 
11:       $(a^*, v^*) \leftarrow (a, v)$ 
12:  return  $(a^*, v^*)$ 
```

Time complexity is $O((|S| \times |A|)^d)$

Monte Carlo evaluation

Algorithm 4.11 Monte Carlo policy evaluation

```
1: function MONTECARLOPOLICYEVALUATION( $\lambda, d$ )
2:   for  $i \leftarrow 1$  to  $n$ 
3:      $s \sim b$ 
4:      $u_i \leftarrow \text{ROLLOUT}(s, d, \pi_\lambda)$ 
5:   return  $\frac{1}{n} \sum_{i=1}^n u_i$ 
```

Algorithm 4.10 Rollout evaluation

```
1: function ROLLOUT( $s, d, \pi_0$ )
2:   if  $d = 0$ 
3:     return 0
4:    $a \sim \pi_0(s)$ 
5:    $(s', r) \sim G(s, a)$ 
6:   return  $r + \gamma \text{ROLLOUT}(s', d - 1, \pi_0)$ 
```

Estimate value of a policy by sampling from a simulator

Sparse sampling

Requires a generative model $(s', r) \sim G(s, a)$

Algorithm 4.8 Sparse sampling

```
1: function SELECTACTION( $s, d$ )
2:   if  $d = 0$ 
3:     return (NIL, 0)
4:    $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$ 
5:   for  $a \in A(s)$ 
6:      $v \leftarrow 0$ 
7:     for  $i \leftarrow 1$  to  $n$ 
8:        $(s', r) \sim G(s, a)$ 
9:        $(a', v') \leftarrow \text{SELECTACTION}(s', d - 1)$ 
10:       $v \leftarrow v + (r + \gamma v') / n$ 
11:     if  $v > v^*$ 
12:        $(a^*, v^*) \leftarrow (a, v)$ 
13:   return  $(a^*, v^*)$ 
```

Complexity? Guarantees?

Sparse sampling

Requires a generative model $(s', r) \sim G(s, a)$

Algorithm 4.8 Sparse sampling

```
1: function SELECTACTION( $s, d$ )
2:   if  $d = 0$ 
3:     return (NIL, 0)
4:    $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$ 
5:   for  $a \in A(s)$ 
6:      $v \leftarrow 0$ 
7:     for  $i \leftarrow 1$  to  $n$ 
8:        $(s', r) \sim G(s, a)$ 
9:        $(a', v') \leftarrow \text{SELECTACTION}(s', d - 1)$ 
10:       $v \leftarrow v + (r + \gamma v')/n$ 
11:     if  $v > v^*$ 
12:        $(a^*, v^*) \leftarrow (a, v)$ 
13:   return  $(a^*, v^*)$ 
```

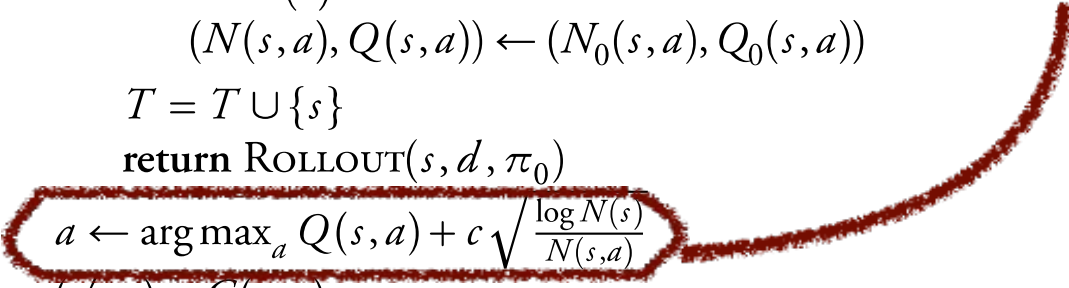
Complexity = $O((n \times |A|)^d)$, Guarantees = probabilistic

Monte Carlo tree search

Algorithm 4.9 Monte Carlo tree search

```
1: function SELECTACTION( $s, d$ )
2:   loop
3:     SIMULATE( $s, d, \pi_0$ )
4:   return  $\arg \max_a Q(s, a)$ 
5: function SIMULATE( $s, d, \pi_0$ )
6:   if  $d = 0$ 
7:     return 0
8:   if  $s \notin T$ 
9:     for  $a \in A(s)$ 
10:       $(N(s, a), Q(s, a)) \leftarrow (N_0(s, a), Q_0(s, a))$ 
11:     $T = T \cup \{s\}$ 
12:    return ROLLOUT( $s, d, \pi_0$ )
13:     $a \leftarrow \arg \max_a Q(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}}$ 
14:     $(s', r) \sim G(s, a)$ 
15:     $q \leftarrow r + \gamma \text{SIMULATE}(s', d - 1, \pi_0)$ 
16:     $N(s, a) \leftarrow N(s, a) + 1$ 
17:     $Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s, a)}{N(s, a)}$ 
18:  return  $q$ 
```

UCT (Upper Confident bounds for Trees)



UCT continued

Search (within the tree, T)

Execute action that maximizes $Q(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}}$

Update the value $Q(s, a)$ and counts $N(s)$ and $N(s, a)$

c is a exploration constant

Expansion (outside of the tree, T)

Create a new node for the state

Initialize $Q(s, a)$ and $N(s, a)$ (usually to 0) for each action

Rollout (outside of the tree, T)

Only expand once and then use a rollout policy to select actions (e.g., random policy)

Add the rewards gained during the rollout with those in the tree:

$$r + \gamma \text{ROLLOUT}(s', d - 1, \pi_0)$$

UCT continued

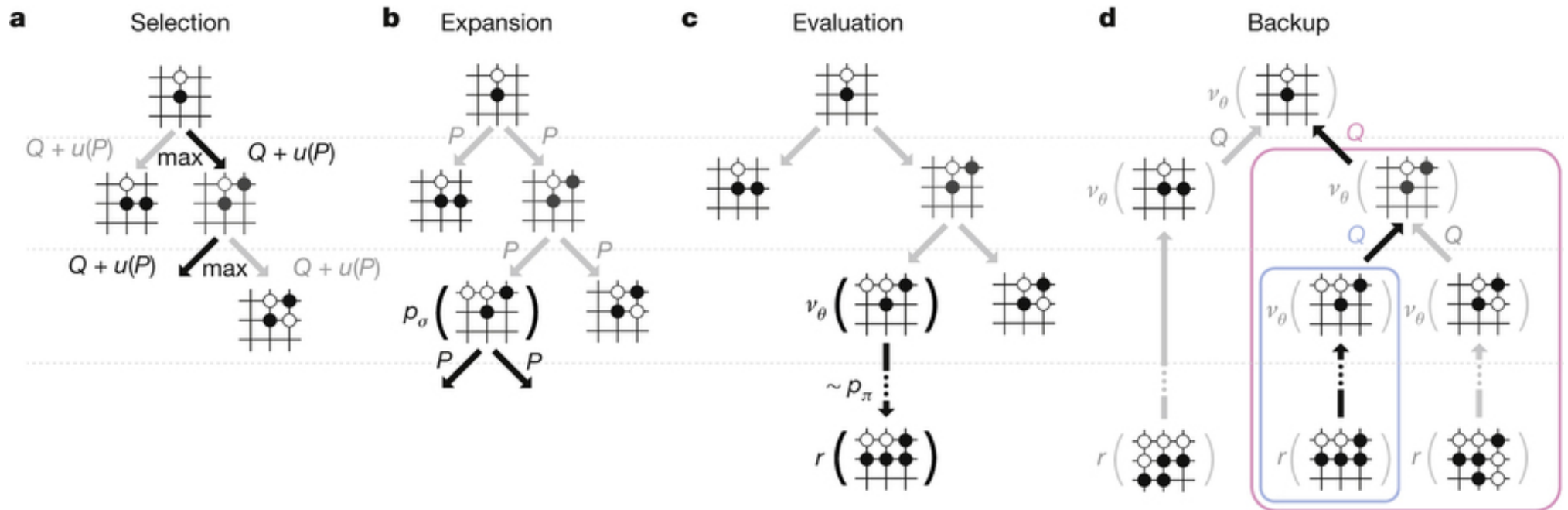
Continue UCT until some termination condition (usually a fixed number of samples)

Complexity?

Guarantees?

AlphaGo

Uses UCT with neural net to approximate opponent choices and state values



Branch and bound search

Requires a lower bound $\underline{U}(s)$ and upper bound $\bar{U}(s)$

Algorithm 4.7 Branch-and-bound search

```
1: function SELECTACTION( $s, d$ )
2:   if  $d = 0$ 
3:     return (NIL,  $\underline{U}(s)$ )
4:   ( $a^*, v^*$ )  $\leftarrow$  (NIL,  $-\infty$ )
5:   for  $a \in A(s)$ 
6:     if  $\bar{U}(s, a) < v^*$ 
7:       return ( $a^*, v^*$ )
8:      $v \leftarrow R(s, a)$ 
9:     for  $s' \in S(s, a)$ 
10:      ( $a', v'$ )  $\leftarrow$  SELECTACTION( $s', d - 1$ )
11:       $v \leftarrow v + \gamma T(s' | s, a) v'$ 
12:     if  $v > v^*$ 
13:       ( $a^*, v^*$ )  $\leftarrow$  ( $a, v$ )
14:   return ( $a^*, v^*$ )
```

Worse case complexity?