MARKOV DECISION PROCESSES

Scott O'Hara

Metrowest Developers Machine Learning Group

08/28/2018

REFERENCES

The material for this talk is drawn from the slides, notes and lectures from several offerings of the CS181 course at Harvard University:

- *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Sarah Finney, Spring 2009
- *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Prof. David C Brooks, Spring 2011
- *CS181 – Machine Learning*, Prof. Ryan P. Adams, Spring 2014. https://github.com/wihl/cs181-spring2014
- *CS181 – Machine Learning*, Prof. David Parkes, Spring 2017. https://harvard-ml-courses.github.io/cs181-web-2017/

OVERVIEW

1. **Introduction**
   - Types of Machine Learning
   - Decision Theory
2. **Markov Decision Processes**
   - Definitions
   - Examples
3. **MDP Solutions**
   - finite horizon techniques
     - expectimax
     - value iteration
   - Infinite horizon techniques
     - value iteration
     - policy iteration

TYPES OF MACHINE LEARNING

There are (at least) 3 broad categories of machine learning problems:

Supervised Learning

$\boldsymbol{Data = \{(x_1, y_1), \ldots, (x_n, y_n)\}}$

e.g., linear regression, decision trees, SVMs

Unsupervised Learning

$\boldsymbol{Data = \{x_1, , \ldots, x_n\}}$

e.g., K-means, HAC, Gaussian mixture models

Reinforcement Learning

$\boldsymbol{Data = \{s_1, a_1, r_1, s_2, a_2, r_2 \ldots\}}$

an agent learns to act in an uncertain environment by training on data that are sequences of **state, action, reward.**

DECISION THEORY

The three components of the decision theoretic framework MDPs:

Probability

Use probability to model uncertainty about the domain.

Utility

Use utility to model an agent's objectives.

Decision Policy

The goal is to design a decision policy, describing how the agent should act in all possible states in order to maximize its expected utility.

UNCERTAINTY MODELED WITH PROBABILITY

- The agent may not know the current state of the world
- The effects of the agent's action might be unpredictable.
- Things happen that are outside the agent's control.
- The agent may be uncertain about the correct model of the world.

"HAPPINESS" MODELED WITH UTILITY

1. Utility is a real number.

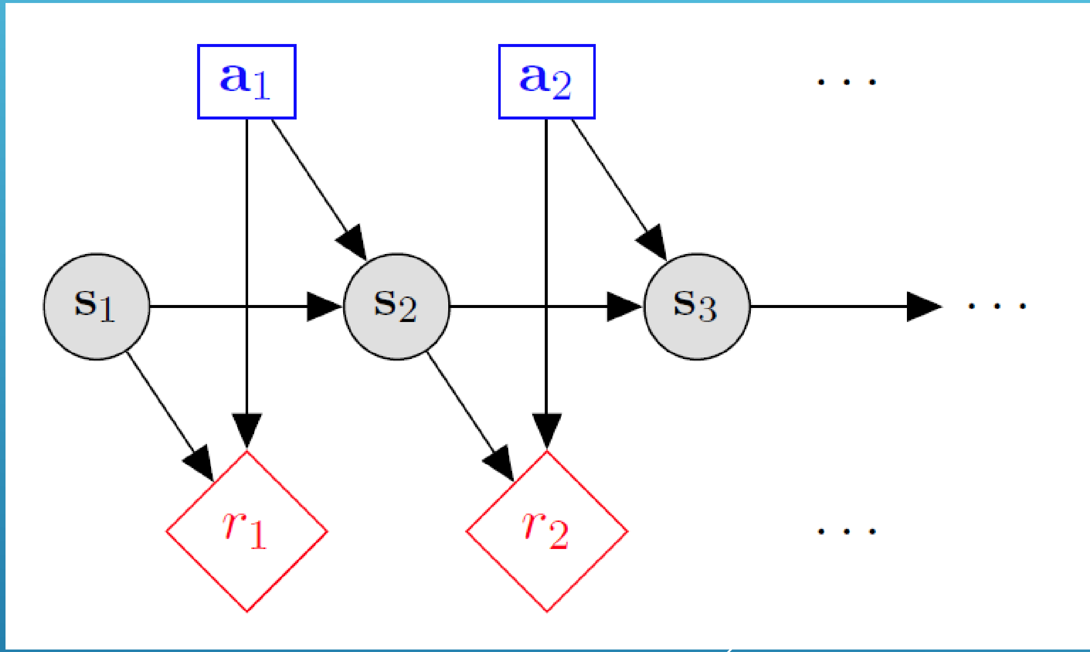
2. The higher the utility, the "happier" your robot is.

3. Utility is based on the assumptions of **Utility Theory**, which if obeyed, make you rational.

4. For example, if you prefer reward A to reward B, and you prefer reward B to reward C, then you should prefer reward A to reward C.

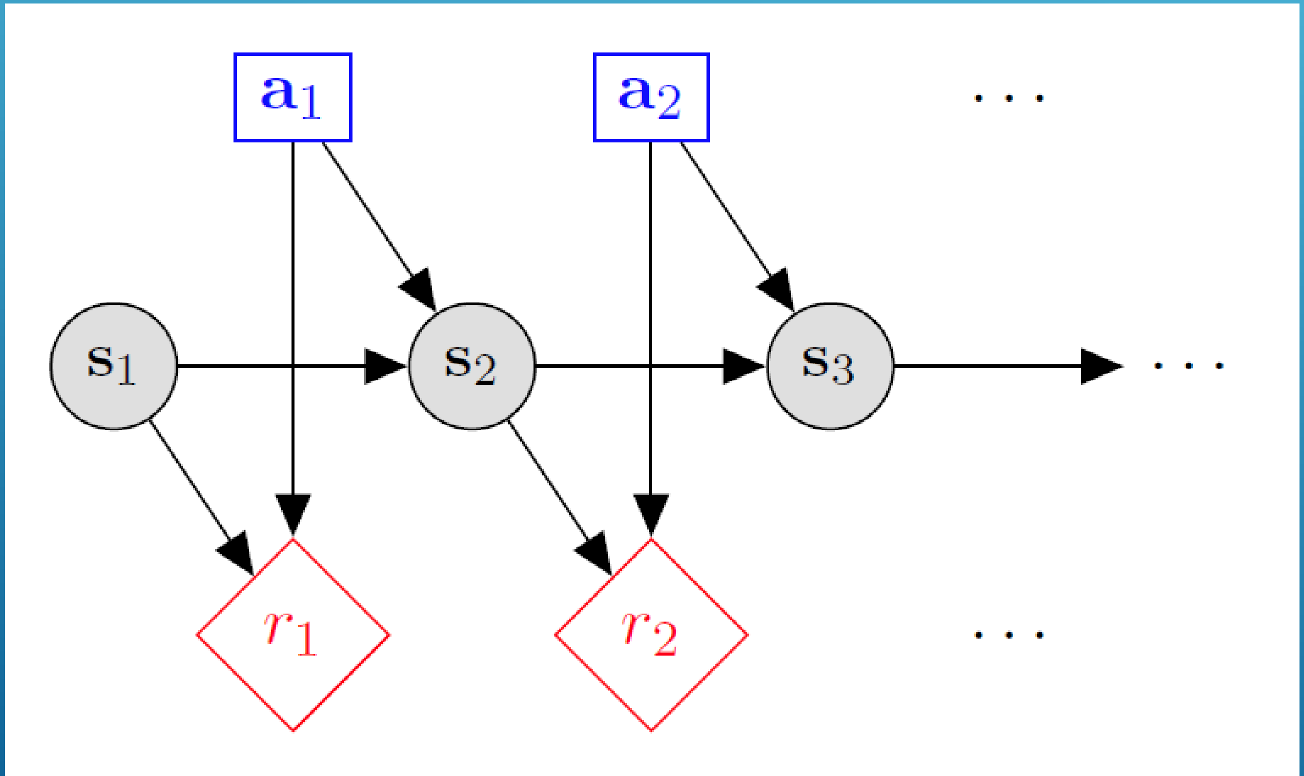
MARKOV DECISION PROCESSES

- **States:** $s_1, \ldots, s_n$
- **Actions:** $a_1, \ldots, a_m$
- **Reward Function:** $\mathrm{r}(s, a) \in R$
- **Transition model:** $\mathrm{p}(s'|s, a)$

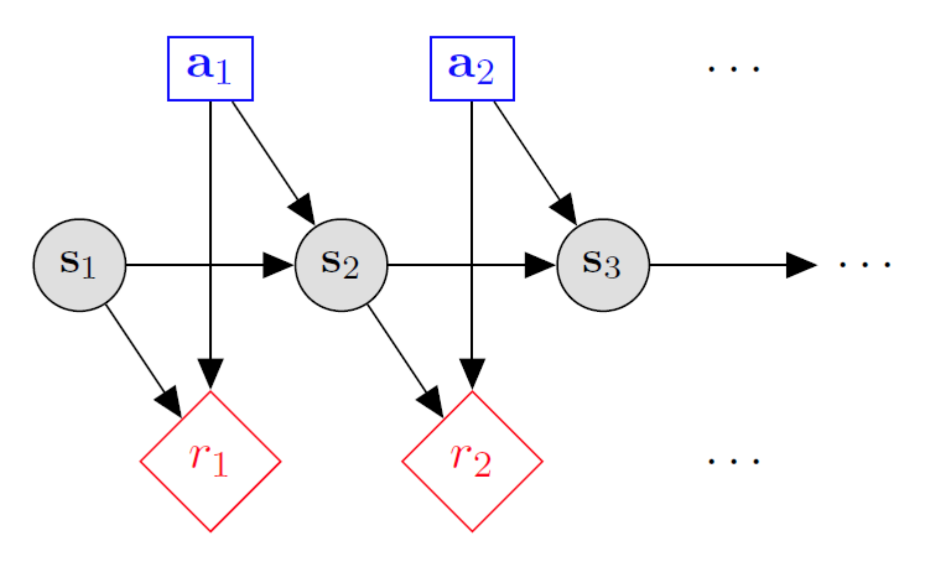


MARKOV DECISION PROCESSES

GOAL: find a **policy π** that tells you what action to take in each state. We want to find 'rewarding' policies.

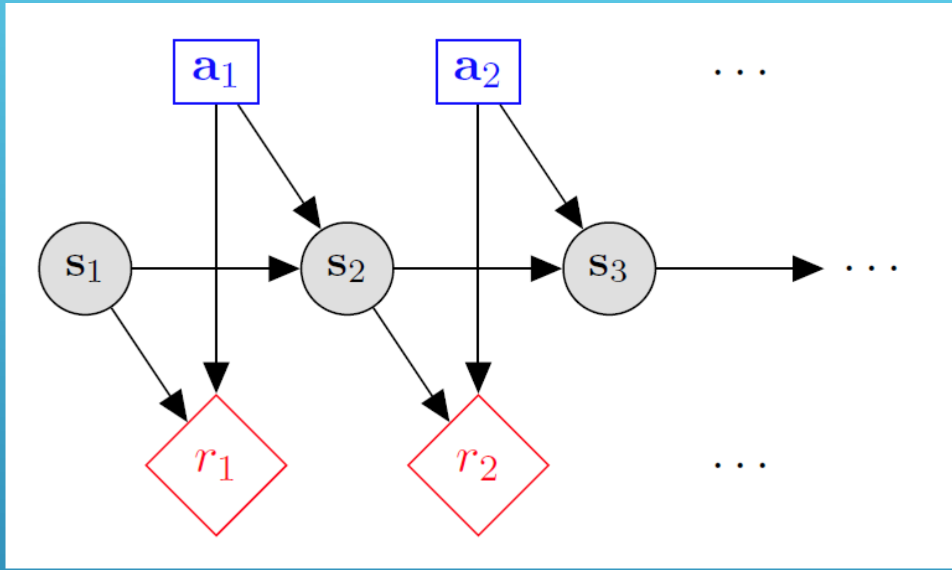


APPLICATION 1: ROBOTS



- **States:** physical location, objects in environment
- **Actions:** move, pick-up, drop, …
- **Reward Function:** +1 if pick up dirty clothes, -1 if break dish
- **Transition model:** describe actuators and uncertain environment.

APPLICATION 2: GAME OF GO

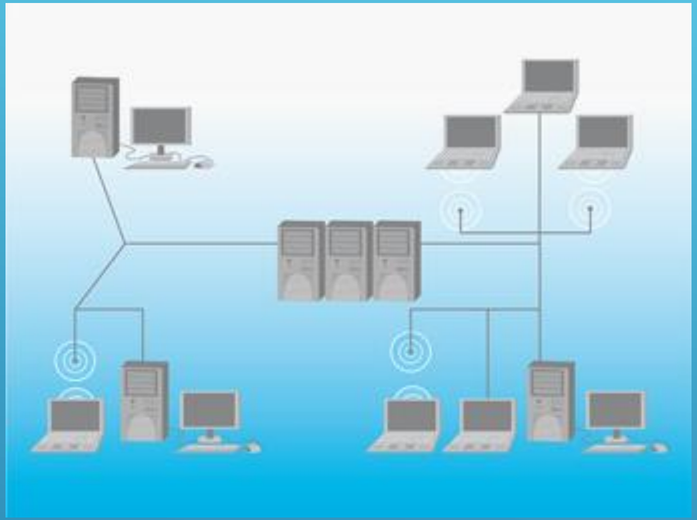


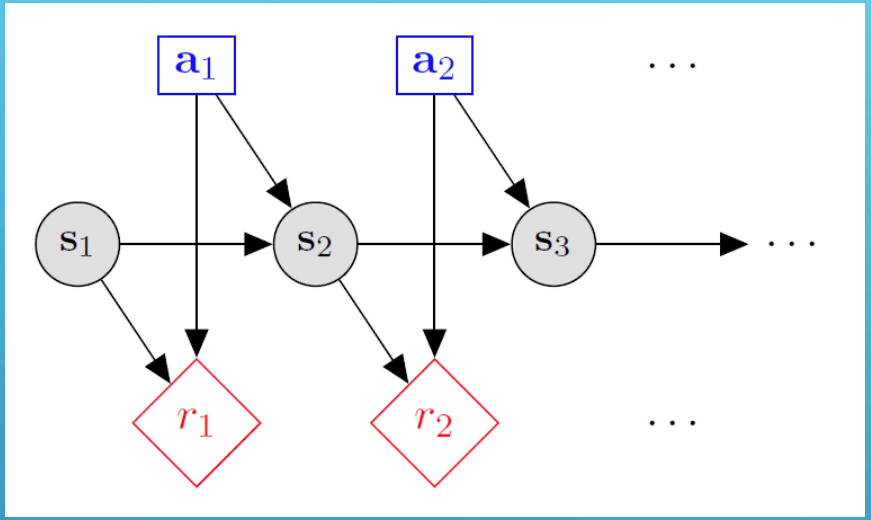
- **States:** board position
- **Actions:** place a piece
- **Reward Function:** +1 if win the game, 0 if draw, -1 if lose.
- **Transition model:** rules of the game, response of other player.

ALPHAGO VS. LEE SEDOL

- AlphaGo (DeepMind) defeated Lee Sedol, 4-1 in March 2016, the top Go player in the world
- AlphaGo combines Monte-Carlo tree search with deep neural nets (trained by supervised learning), with reinforcement learning.
- Learns both a `policy network' (which action to play in which state) and a `value network' (estimate of value of an action under self-play).

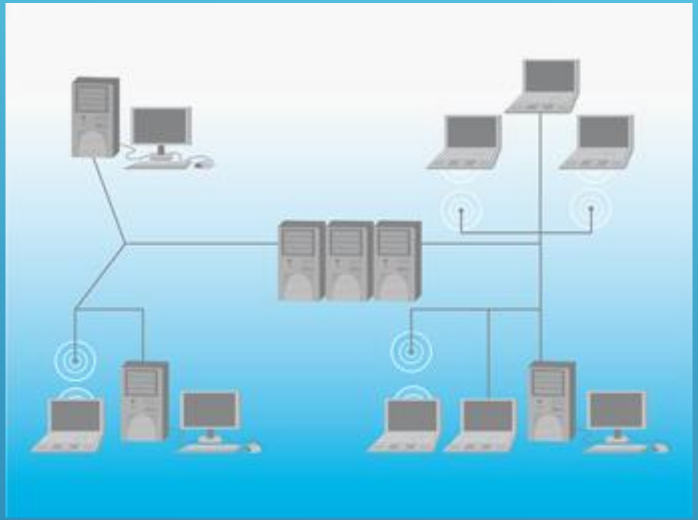
APPLICATION 3: EMAIL ROUTING

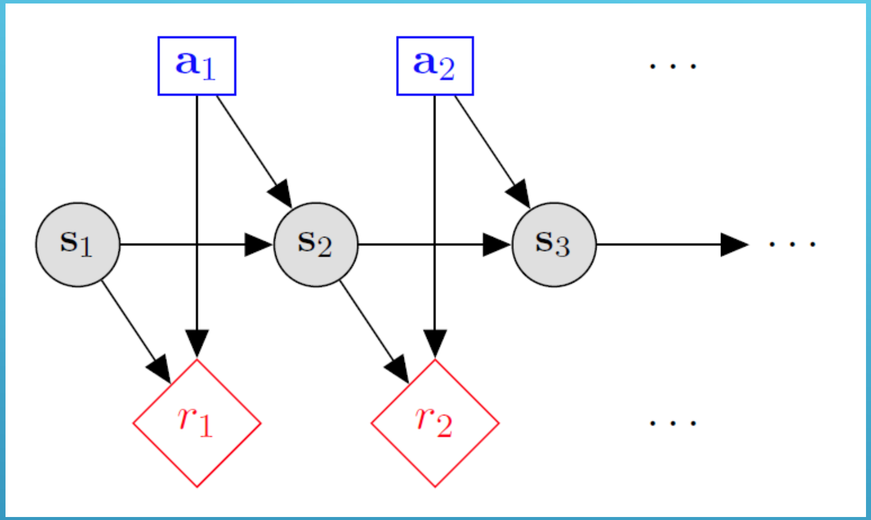




- **States:**
  - up/down for each server
  - current location and goal of each message.
- **Actions:** choose path of servers for each message.
- **Reward Function:** +1 for each message delivered to goal.
- **Transition model:** describe network of servers.

APPLICATION 3: TRANSITION MODEL





P(s’|s, a) depends on:

- probability that each server fails given current load
- probability that new messages enter the queue
- probability that each message completes hop, given the state of the servers

SCOPE OF MDP APPLICABILITY

The Markov Decision Process is a general probabilistic framework, and can be applied in many different scenarios.

Planning ← this talk

- Full access to the MDP, compute an optimal policy.
- "How do I act in a known world?"

Policy Evaluation ← this talk

- Full access to the MDP, compute the `value' of a fixed policy.
- "How will this plan perform under uncertainty?"

Reinforcement Learning (later)

- Limited access to the MDP.
- "Can I learn to act in an uncertain world?"

DIFFERENT OBJECTIVE CRITERIA

- Sequence of $\boldsymbol{s_1, a_1, r_1, s_2, a_2, r_2}$ ... ; discrete time ***t***
- **Finite horizon**, $T \geq 1$ steps

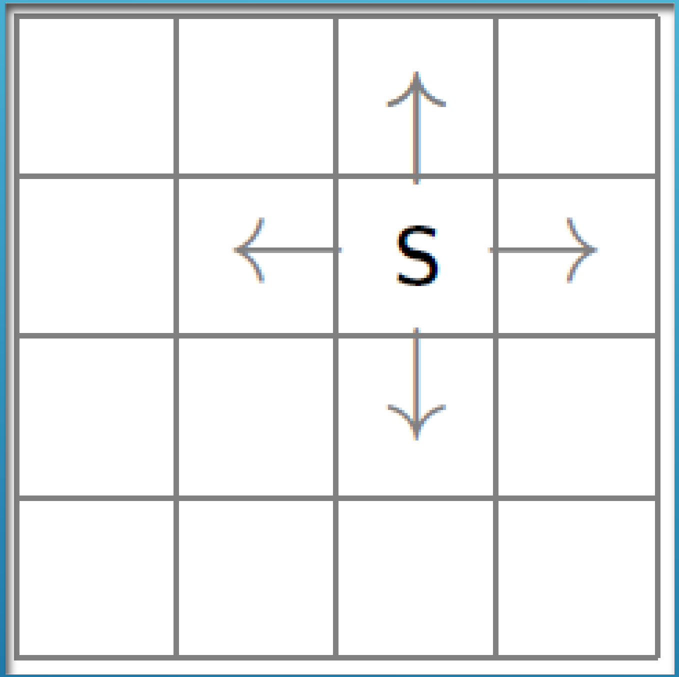
$$utility = \sum_{t=1}^{T} \mathrm{r}(s_t, a_t)$$

- **Infinite horizon**, discount factor $\gamma \, \epsilon \, (0, 1]$

$$utility = \mathrm{r}(s_1, a_1) + \gamma \cdot \mathrm{r}(s_{2t}, a_2) + \gamma^2 \cdot \mathrm{r}(s_3, a_3) + \cdots$$

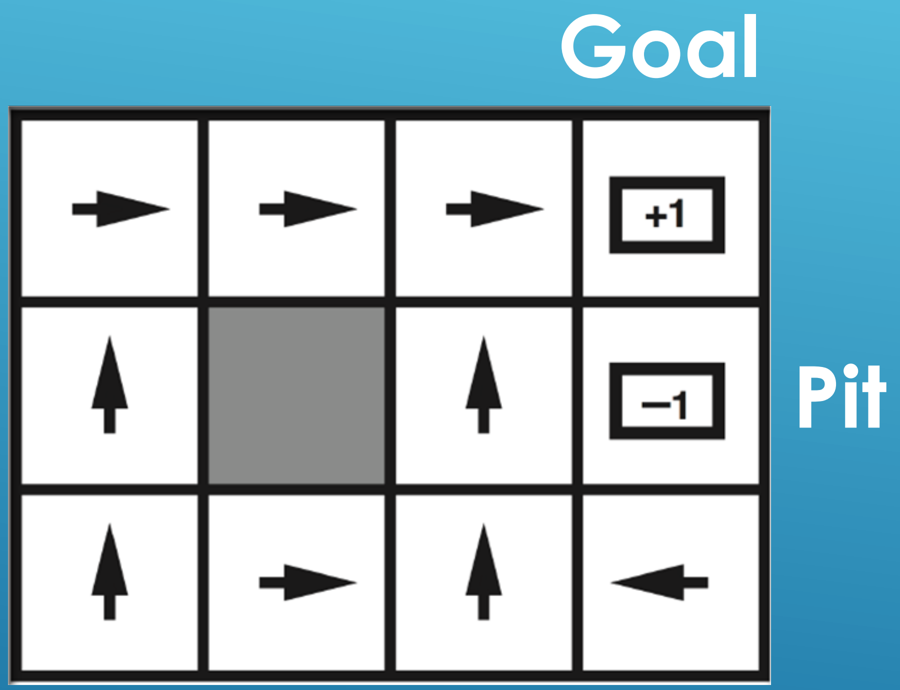
$$utility = \sum_{t=1}^{T} \gamma^{t-1} \cdot \mathrm{r}(s_t, a_t)$$

OPTIMAL POLICY EXAMPLES: GRIDWORLD



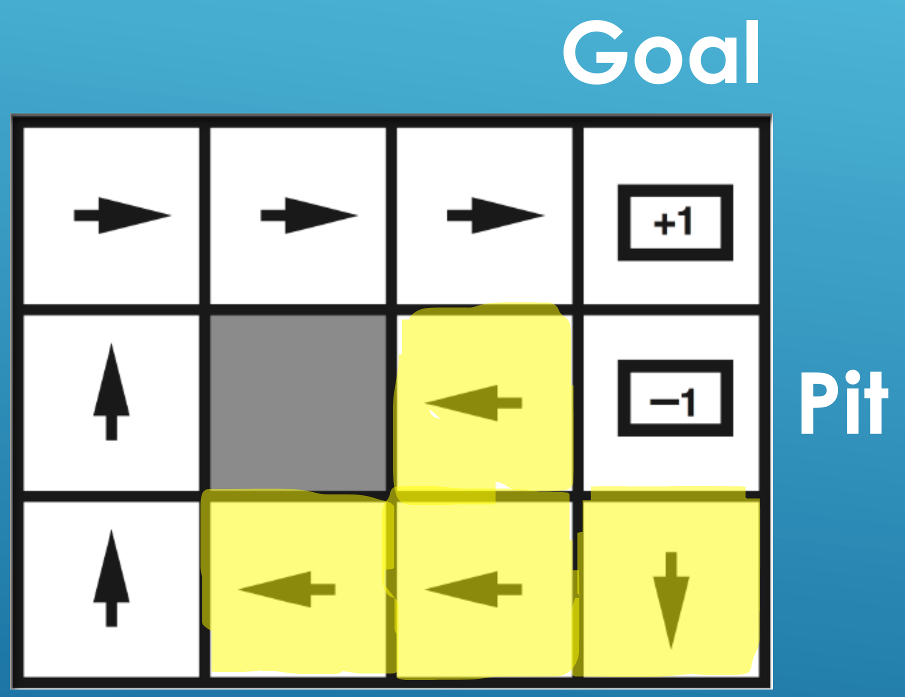
S	Location on the grid (x_1, y_1)
A	Local movements $\leftarrow, \rightarrow, \uparrow, \downarrow$
$\mathbf{r}(\boldsymbol{s}, \boldsymbol{a})$	Reward function, e.g., make it to a goal, don't fall into a pit.
$\boldsymbol{p}(\boldsymbol{s}' \vert \boldsymbol{s}, \boldsymbol{a})$	Transition model e.g., d deterministic or slippages.

OPTIMAL POLICY: PERFECT ACTUATOR



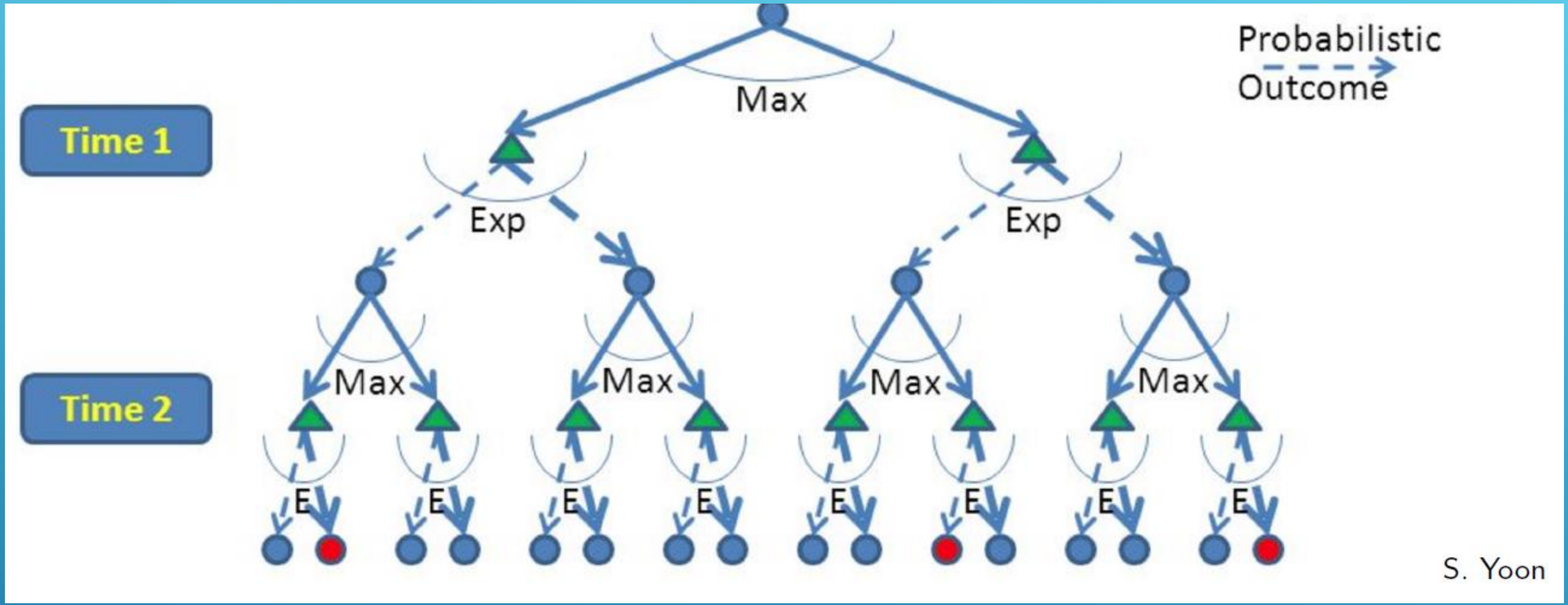
- $\mathrm{r}(goal, a) = +1$ and stop
- $\mathrm{r}(pit, a) = -1$ and stop
- $\mathrm{r}(s, a) = -0.04$ everywhere else.
- Bounce off obstacles
- Perfect actuator

OPTIMAL POLICY: IMPERFECT ACTUATOR



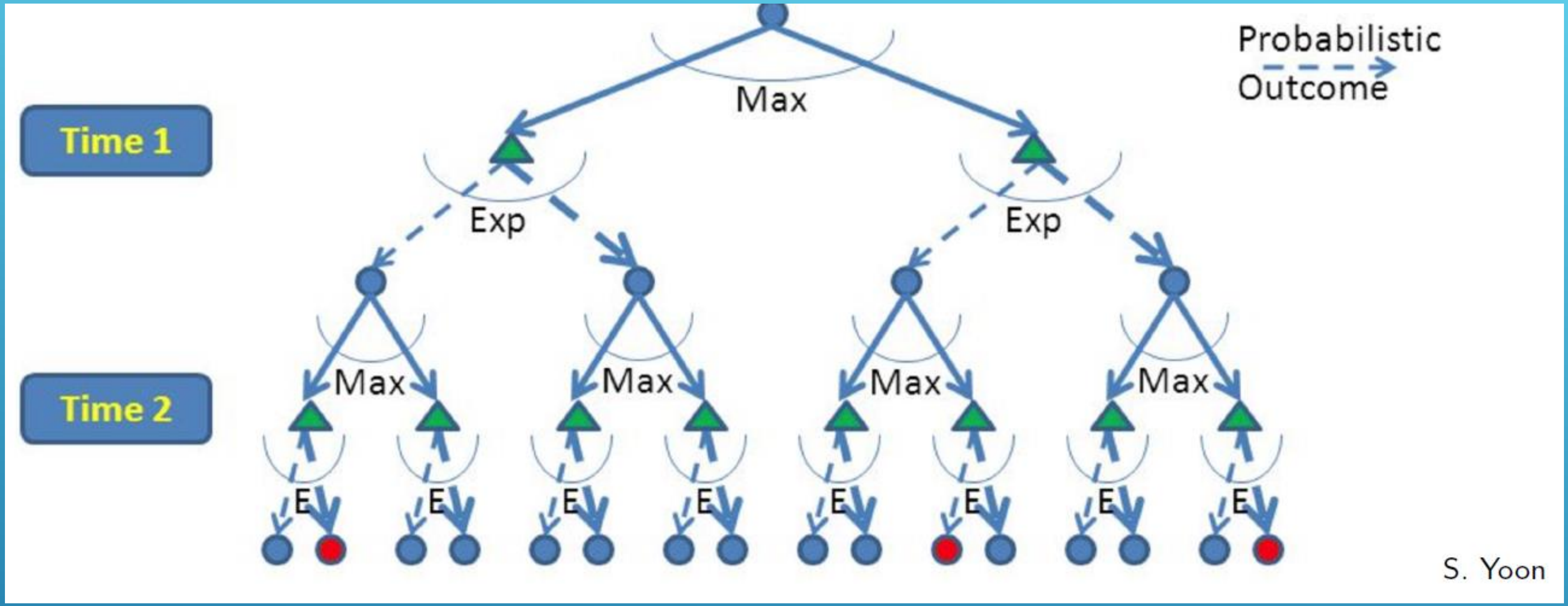
- $\mathrm{r}(goal, a) = +1$ and stop
- $\mathrm{r}(pit, a) = -1$ and stop
- $\mathrm{r}(s, a) = -0.04$ everywhere else.
- Bounce off obstacles
- Imperfect actuator:
  - 0.8 probability of going straight
  - 0.1 probability of moving right
  - 0.1 probability of moving left

FINITE HORIZON : EXPECTIMAX



- Build out a look-ahead tree to the decision horizon; max over actions, exp over next states.
- Solve from the leaves, backing-up the expectimax values.
- Problem: computation is exponential in horizon.
- May expand the same subtree multiple times.

EXPECTIMAX: A GAME AGAINST NATURE



- Like a game except opponent is probabilistic
- Strongly related minmax used in game theory
- Nodes where you move: S (△)
- Nodes where nature moves: <S,A> (◯)

POLICY AND VALUES

- Policy: action to take at each state
- Value of state (node): expected total reward
  - Depends on policy

NOTATION

- π: S→A (policy)
- π^*: optimal policy
- $\pi^*(i)$: best action in state i
- $V^\pi(i)$: value of node i, assuming policy π
- $Q^\pi(i,a)$: value of nature's node $\langle i,a \rangle$ assuming policy π

BASIC EQUATIONS

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

$$V(i) = Q(i,\pi^*(i))$$

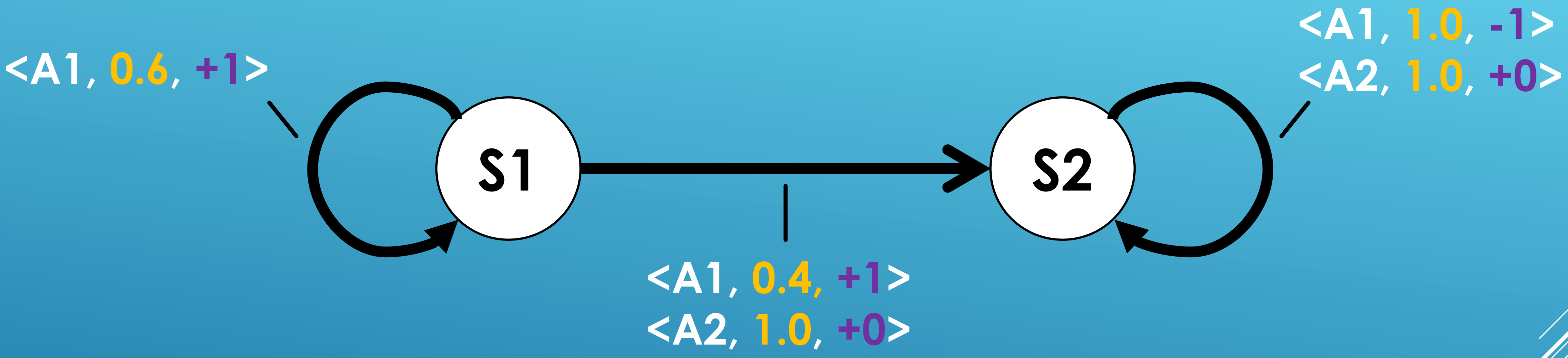
$$Q(i,a) = R(i,a) + \sum_j T_{ij}^a V(j)$$

EXPECTIMAX ALGORITHM

Algorithm 1 Expectimax Search

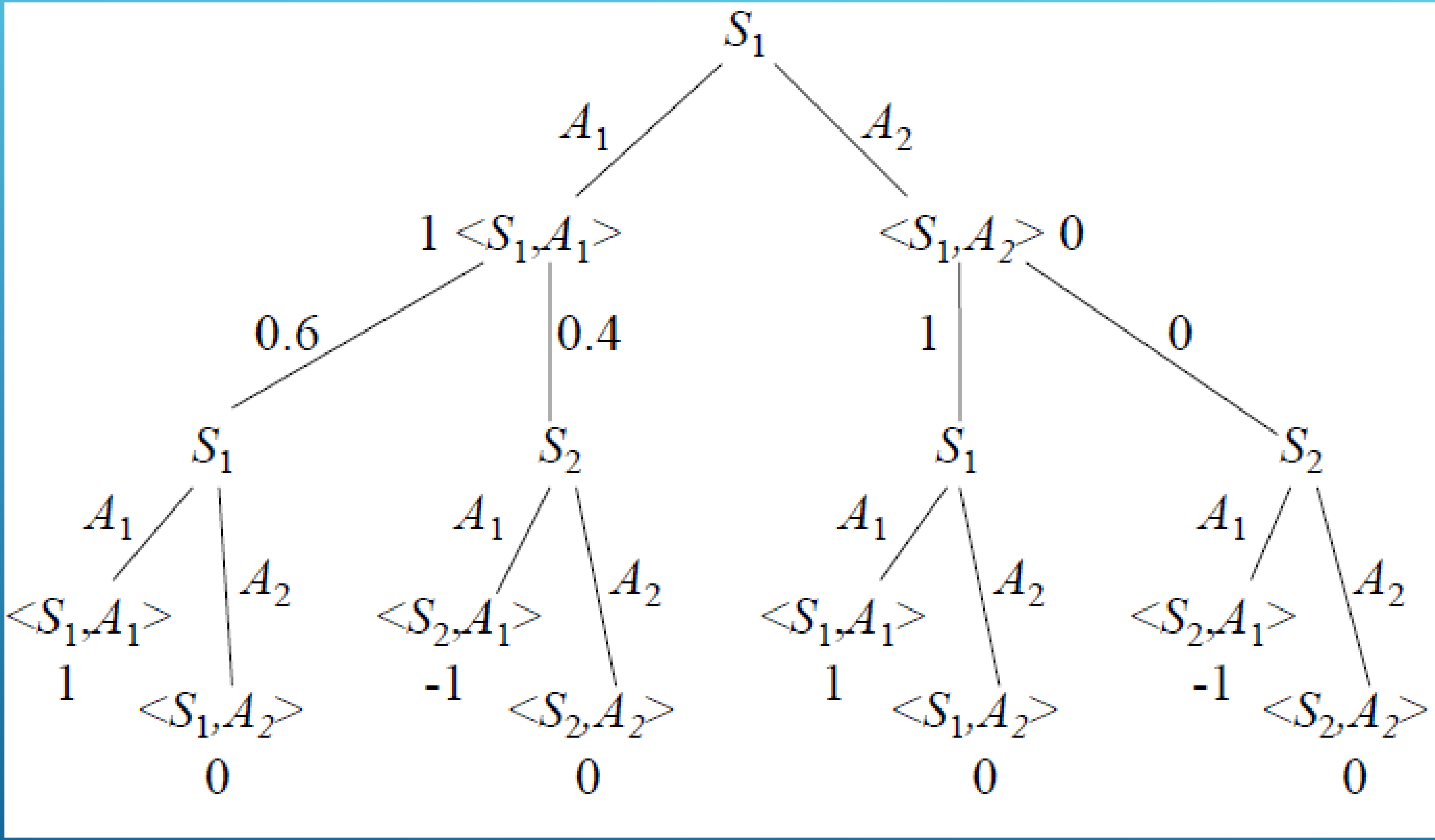
function EXPECTIMAX(s) ▷ Takes a state as an input.
  if s is terminal **then**
    Return 0
  else
    for $a \in \mathcal{A}$ **do** ▷ Look at all possible actions.
      $Q(s,a) \leftarrow R(s,a) + \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\, \text{EXPECTIMAX}(s')$ ▷ Compute expected value.
    end for
    $\pi^*(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s,a)$ ▷ Optimal policy is value-maximizing action.
    Return $Q(s, \pi^*(s))$
  end if
end function

2-STATE FINITE HORIZON EXAMPLE

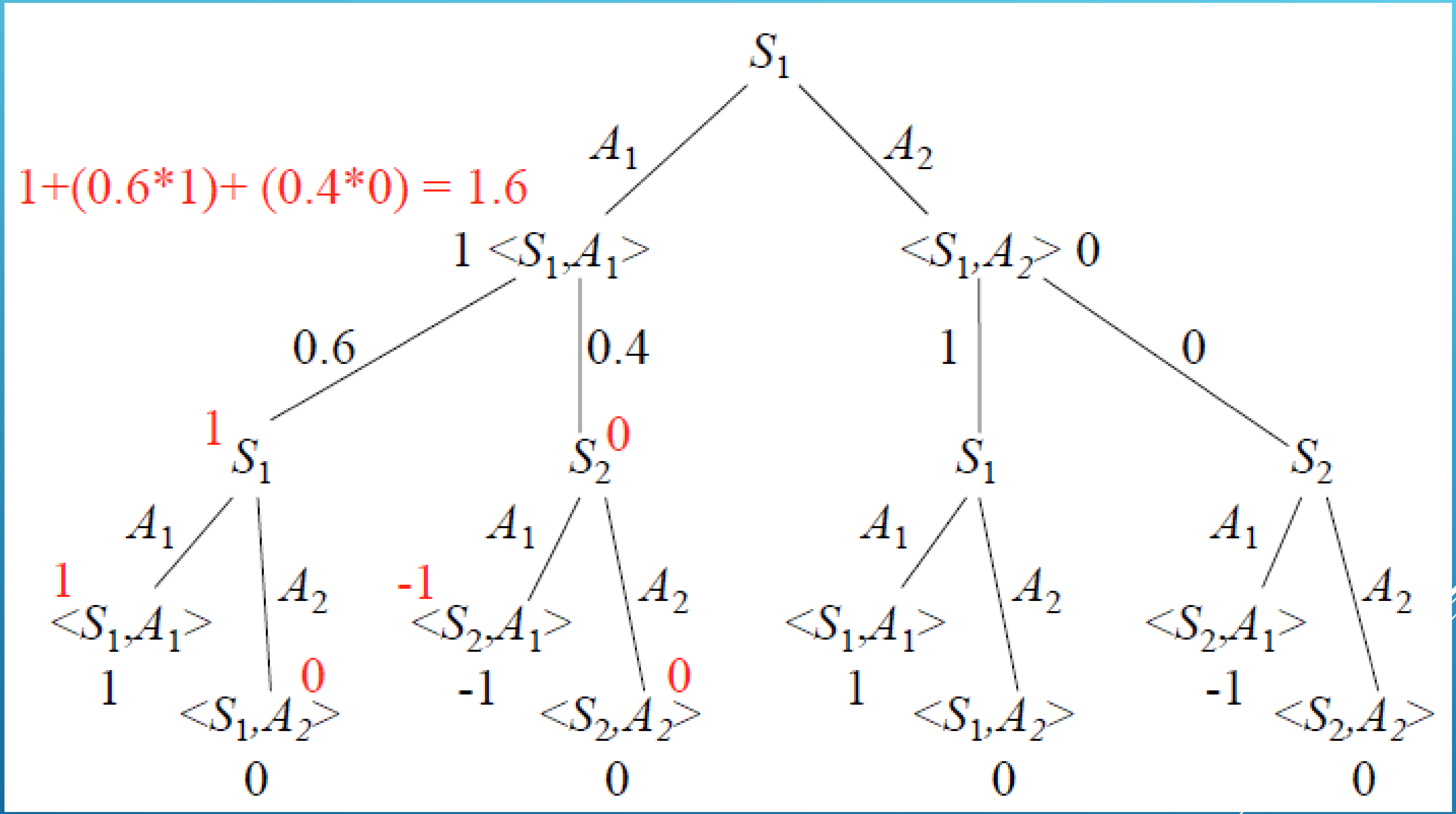


States: S1, S2
Actions: A1,A2
Edges: <action, probability, value>

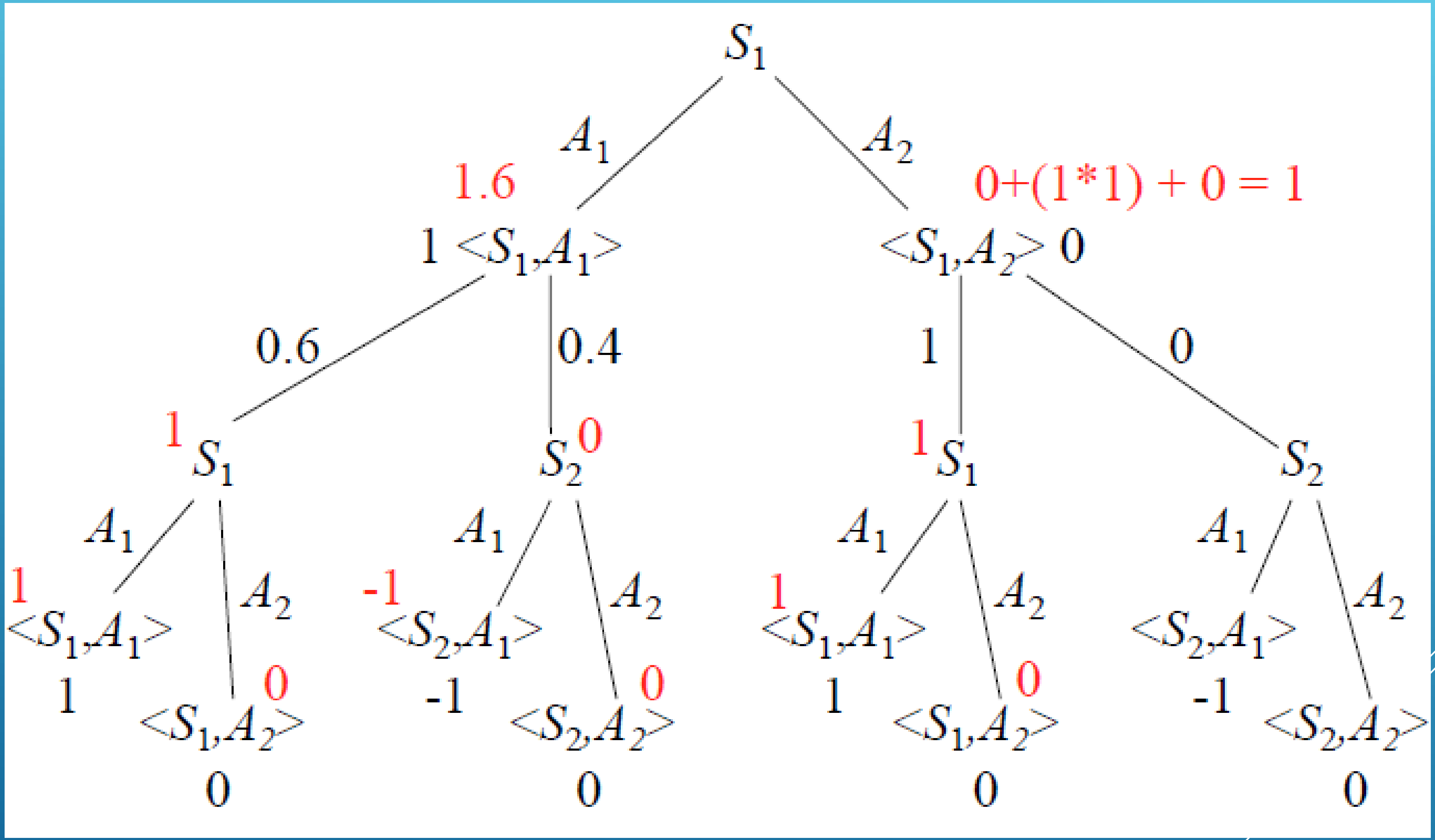
EXPECTIMAX: GAME TREE EXAMPLE (1)



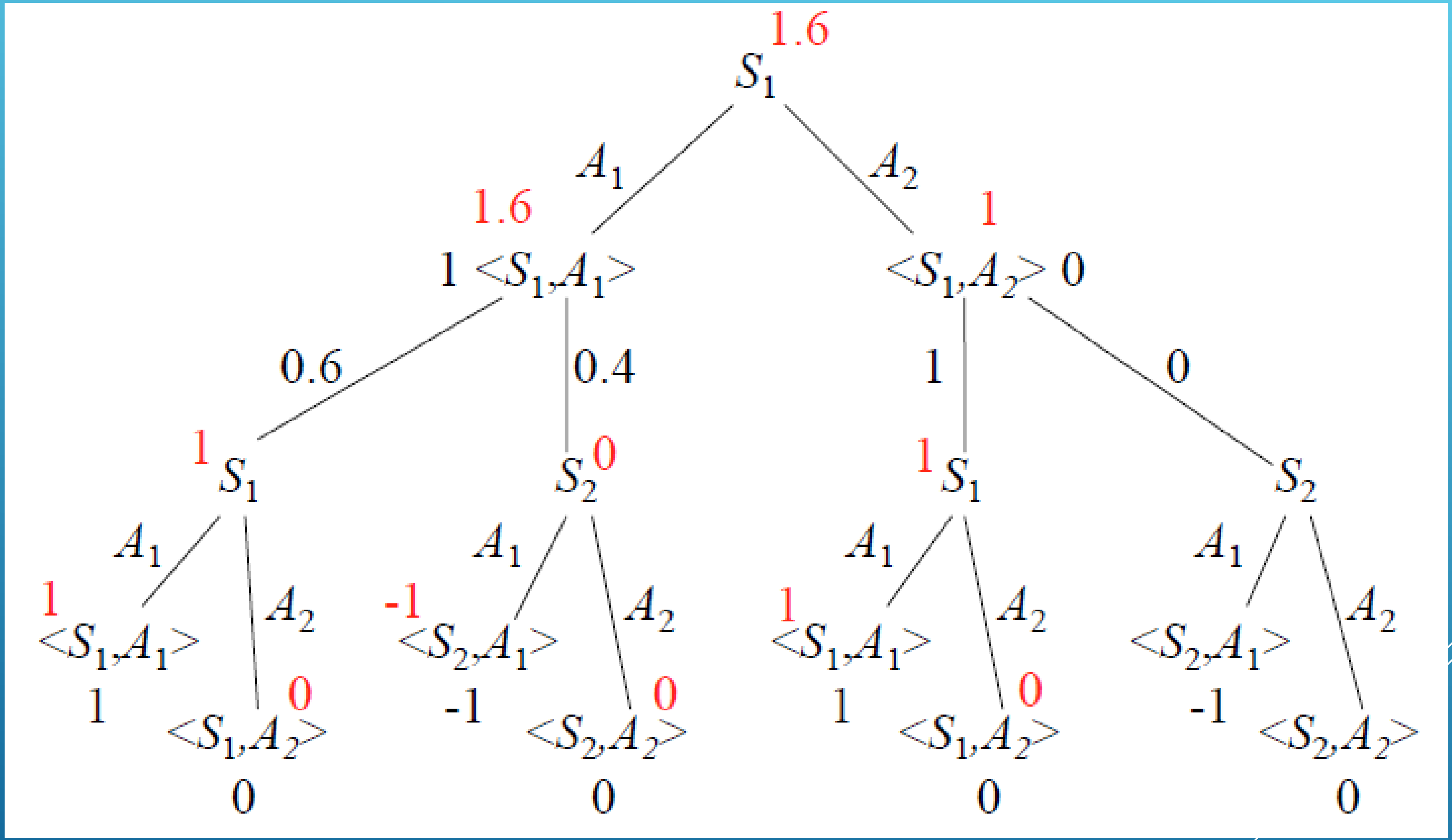
EXPECTIMAX: GAME TREE EXAMPLE (2)



EXPECTIMAX: GAME TREE EXAMPLE (3)



EXPECTIMAX: GAME TREE EXAMPLE (4)



COMPLEXITY OF EXPECTIMAX

- Linear in size of tree
- Exponential in horizon
- Base of the exponent: (# of actions) * (# of transitions)
- $O\left((m \cdot l)^h\right)$
  - h = horizon
  - m = number of actions
  - l = maximum # of non-zero transitions from state

AVOID EXPONENTIAL BLOWUP WITH DYNAMIC PROGRAMMING

- The same state can be reached by many paths
- To solve a large problem, solve smaller subproblems
- Must be able to combine subproblem solutions effectively to solve larger problems.
- For Expectimax, the solution is Value iteration

VALUE ITERATION

- Break up problem by number of steps to go
- Given optimal policy for k-1 steps to go, compute Q-values for k steps to go
- Base case value with no timesteps to go

K-STEPS-TO-GO NOTATION

- $V_k(i)$: value of state i
- $\pi^*_k(i)$: optimal policy for state i
- $Q_k(i,a)$: value of taking action a in state i

- Subscript denotes k steps to go
- Each assumes optimal future choices

BASIC EQUATIONS

- Compute Q-values from values on next timestep:

$$\pi_k^*(i) = \arg\max_a Q_k(i,a)$$

$$V_k(i) = Q_k(i, \pi^*(i))$$

$$Q_k(i,a) = R(i,a) + \sum_j T_{ij}^a V_{k-1}(j)$$

- Need a base case:

$$V_0(i) = 0$$

VALUE ITERATION ALGORITHM

Algorithm 2 Value Iteration

function VALUEITERATION(T) ▷ Takes a horizon as input.
  for $s \in \mathcal{S}$ **do** ▷ Loop over each possible ending state.
    $V_0(s) \leftarrow 0$ ▷ Horizon states have no value.
  end for
  for $k \leftarrow 1 \ldots T$ **do** ▷ Loop backwards over time.
    for $s \in \mathcal{S}$ **do** ▷ Loop over possible states with k steps to go.
      for $a \in \mathcal{A}$ **do** ▷ Loop over possible actions.
        $Q_k(s,a) \leftarrow R(s,a) + \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_{k-1}(s')$ ▷ Compute Q-function for k.
      end for
      $\pi_k^\star(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q_k(s,a)$ ▷ Find best action with k to go in state s.
      $V_k(s) \leftarrow Q_k(s, \pi_k^\star(s))$ ▷ Compute value for state s with k steps to go.
    end for
  end for
end function

VALUE ITERATION EXAMPLE (1)

k	$Q_k(s_1,a_1)$	$Q_k(s_1,a_2)$	$Q_k(s_2,a_1)$	$Q_k(s_2,a_2)$	$\pi^*_k(s_1)$	$\pi^*_k(s_2)$	$V_k(s_1)$	$V_k(s_2)$
0							0	0
1	1+0.6*0+0.4*0=1							
2								
3								
4								

VALUE ITERATION EXAMPLE (2)

k	$Q_k(s_1,a_1)$	$Q_k(s_1,a_2)$	$Q_k(s_2,a_1)$	$Q_k(s_2,a_2)$	$\pi^*_k(s_1)$	$\pi^*_k(s_2)$	$V_k(s_1)$	$V_k(s_2)$
0							0	0
1	1	0	-1	0	a_1	a_2	1	0
2								
3								
4								

VALUE ITERATION EXAMPLE (3)

k	$Q_k(s_1,a_1)$	$Q_k(s_1,a_2)$	$Q_k(s_2,a_1)$	$Q_k(s_2,a_2)$	$\pi^*_k(s_1)$	$\pi^*_k(s_2)$	$V_k(s_1)$	$V_k(s_2)$
0							0	0
1	1	0	-1	0	a_1	a_2	1	0
2	$1+0.6*1+0.4*0=1.6$							
3								
4								

VALUE ITERATION EXAMPLE (4)

k	$Q_k(s_1,a_1)$	$Q_k(s_1,a_2)$	$Q_k(s_2,a_1)$	$Q_k(s_2,a_2)$	$\pi^*_k(s_1)$	$\pi^*_k(s_2)$	$V_k(s_1)$	$V_k(s_2)$
0							0	0
1	1	0	-1	0	a_1	a_2	1	0
2	1.6	1	-0.4	0	a_1	a_2	1.6	0
3								
4								

VALUE ITERATION EXAMPLE (5)

k	$Q_k(s_1,a_1)$	$Q_k(s_1,a_2)$	$Q_k(s_2,a_1)$	$Q_k(s_2,a_2)$	$\pi^*_k(s_1)$	$\pi^*_k(s_2)$	$V_k(s_1)$	$V_k(s_2)$
0							0	0
1	1	0	-1	0	a_1	a_2	1	0
2	1.6	1	-0.4	0	a_1	a_2	1.6	0
3	1.96	1.6	-0.04	0	a_1	a_2	1.96	0
4	2.176	1.96	0.176	0	a_1	a_1	2.176	0.176

HORIZON EFFECT

- **h = 1**: greedy; only consider immediate reward
- **h is small:** only consider short term rewards, no long-term planning
- **h is large:** willing to sacrifice short-term gain for long-term reward

COMPLEXITY OF VALUE ITERATION

- h = horizon, m actions, n states
- l = max number of non-zero outgoing transitions
- # of Q-values per time step: m*n
- How long to compute Q-value? Need to sum over possible next states: O(l)
- Total cost: O(mnlh)

COMPARISON

- Value iteration is better when states can be reached by multiple paths
- Expectimax is better when each state is reachable only one way and many states cannot be reached at all

INFINITE HORIZON

- Expectimax and finite horizon value iteration rely on a finite horizon since both algorithms assume a base case.

- We would like to look infinitely far into the future.

- To do this, we consider discounting future
  - we may never get there
  - the equations don't blow up

BASIC EQUATIONS FOR INFINITE HORIZON VALUE ITERATION

$$\pi_k^*(i) = \arg\max_a Q_k(i,a)$$

$$V_k(i) = Q_k(i,\pi^*(i))$$

$$Q_k(i,a) = R(i,a) + \gamma \sum_j T_{ij}^a V_{k-1}(j)$$

$$V_0(i) = 0$$

VALUE ITERATION (FINITE HORIZON)

Algorithm 2 Value Iteration

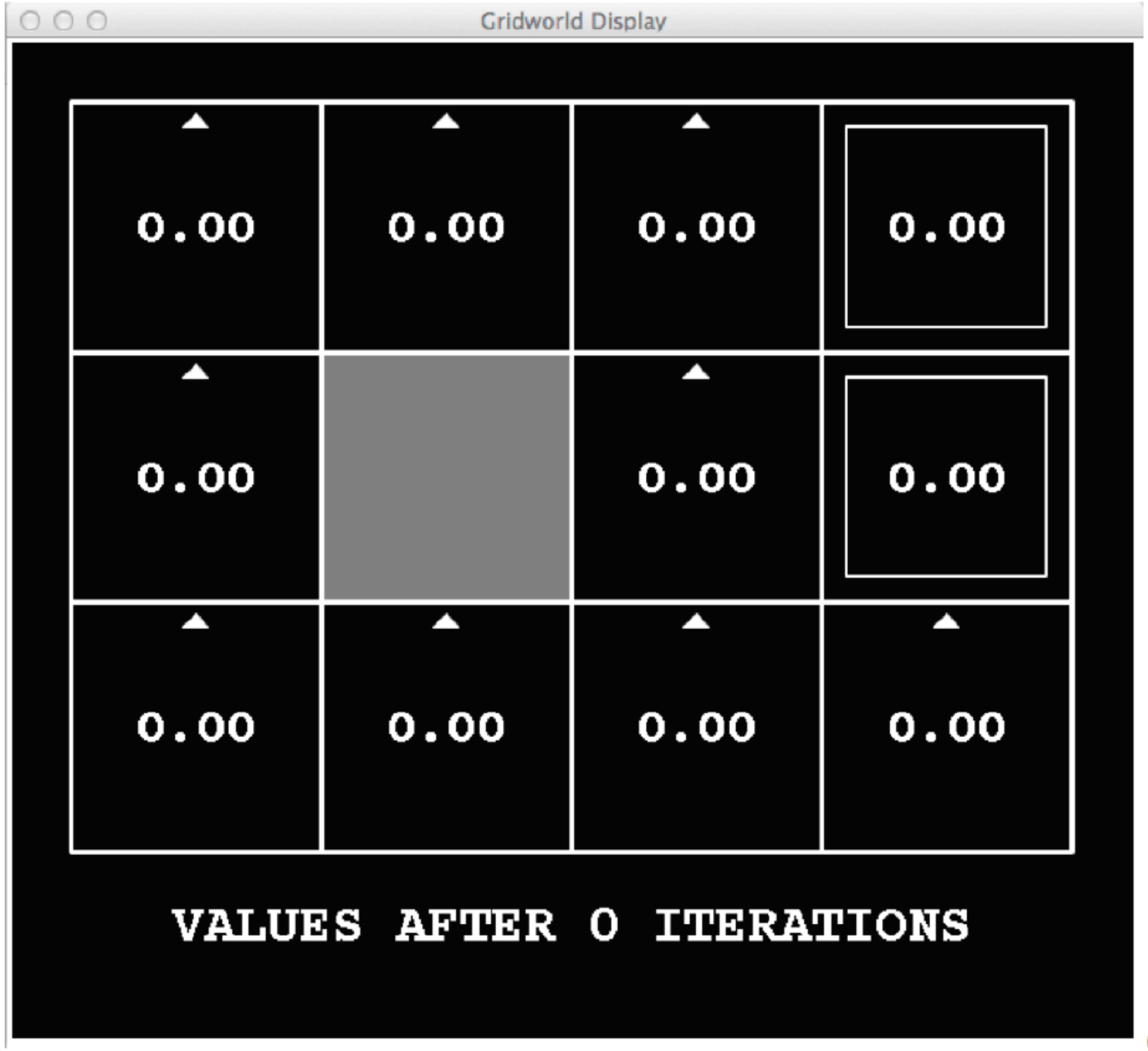
function VALUEITERATION(T) ▷ Takes a horizon as input.
  for $s \in \mathcal{S}$ **do** ▷ Loop over each possible ending state.
    $V_0(s) \leftarrow 0$ ▷ Horizon states have no value.
  end for
  for $k \leftarrow 1 \ldots T$ **do** ▷ Loop backwards over time.
    for $s \in \mathcal{S}$ **do** ▷ Loop over possible states with k steps to go.
      for $a \in \mathcal{A}$ **do** ▷ Loop over possible actions.
        $Q_k(s,a) \leftarrow R(s,a) + \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_{k-1}(s')$ ▷ Compute Q-function for k.
      end for
      $\pi_k^\star(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q_k(s,a)$ ▷ Find best action with k to go in state s.
      $V_k(s) \leftarrow Q_k(s, \pi_k^\star(s))$ ▷ Compute value for state s with k steps to go.
    end for
  end for
end function

VALUE ITERATION (INFINITE HORIZON)

Algorithm 1 Infinite Horizon Value Iteration

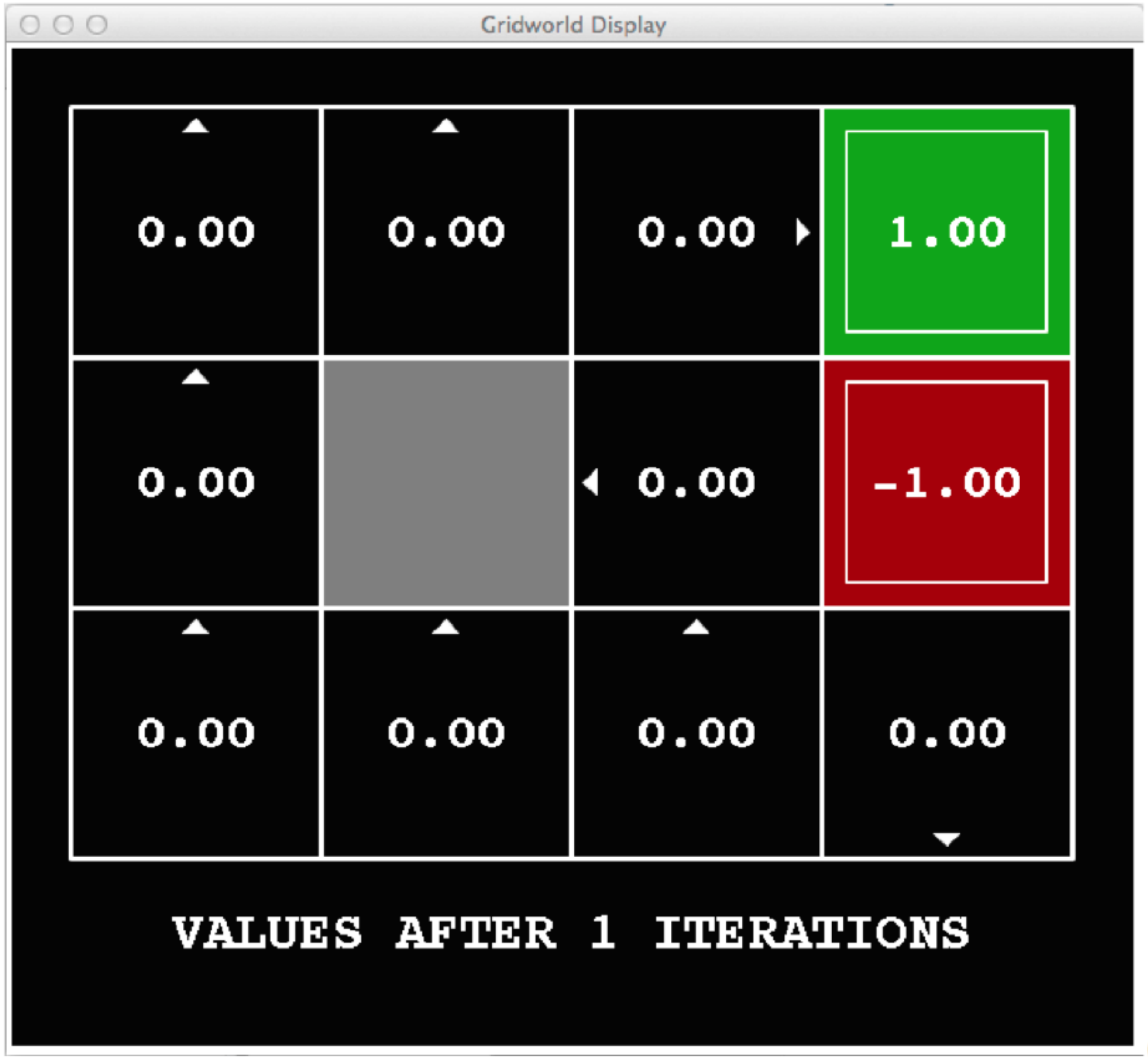
function ValueIteration(γ) ▷ Takes a discount factor as input.
    for $s \in \mathcal{S}$ **do** ▷ Loop over each possible ending state.
        $V(s) \leftarrow 0$ ▷ Initialize states with zero value.
    end for
    repeat
        $V_{\text{old}}(\cdot) \leftarrow V(\cdot)$ ▷ Store off old value function.
        for $s \in \mathcal{S}$ **do** ▷ Loop over all the states again.
            for $a \in \mathcal{A}$ **do** ▷ Loop over possible actions.
                $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V_{\text{old}}(s')$ ▷ Compute Q-function.
            end for
            $\pi^\star(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s,a)$ ▷ Find best action from state s.
            $V(s) \leftarrow Q(s, \pi^\star(s))$ ▷ Update value for state s.
        end for
    until $|V(s) - V_{\text{old}}(s)| < \epsilon, \ \forall s \in \mathcal{S}$ ▷ Loop until convergence for some small ϵ.
    Return $\pi^\star(\cdot)$
end function

VALUE ITERATION IN GRIDWORLD (0)



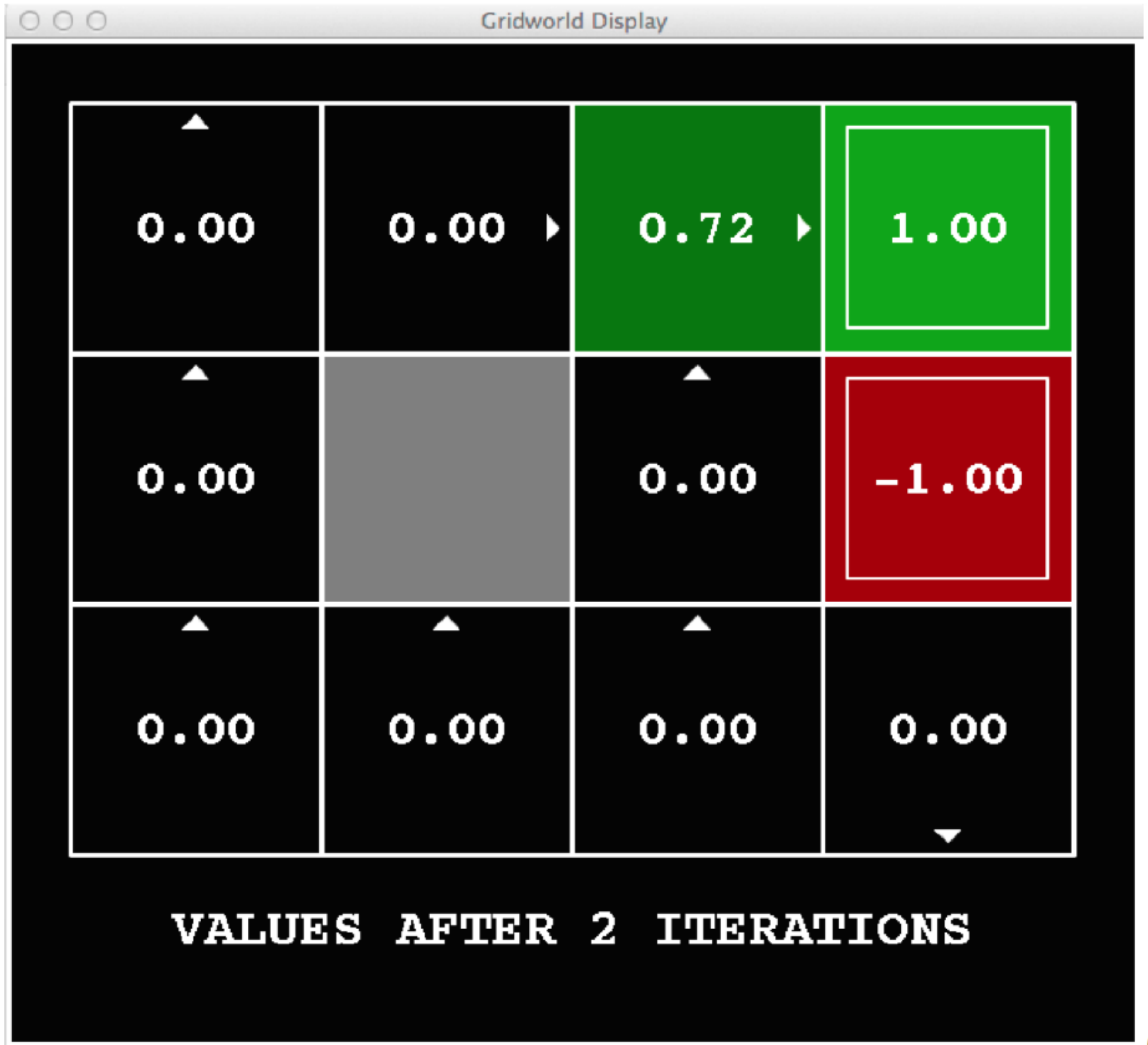
(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (1)



(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (2)

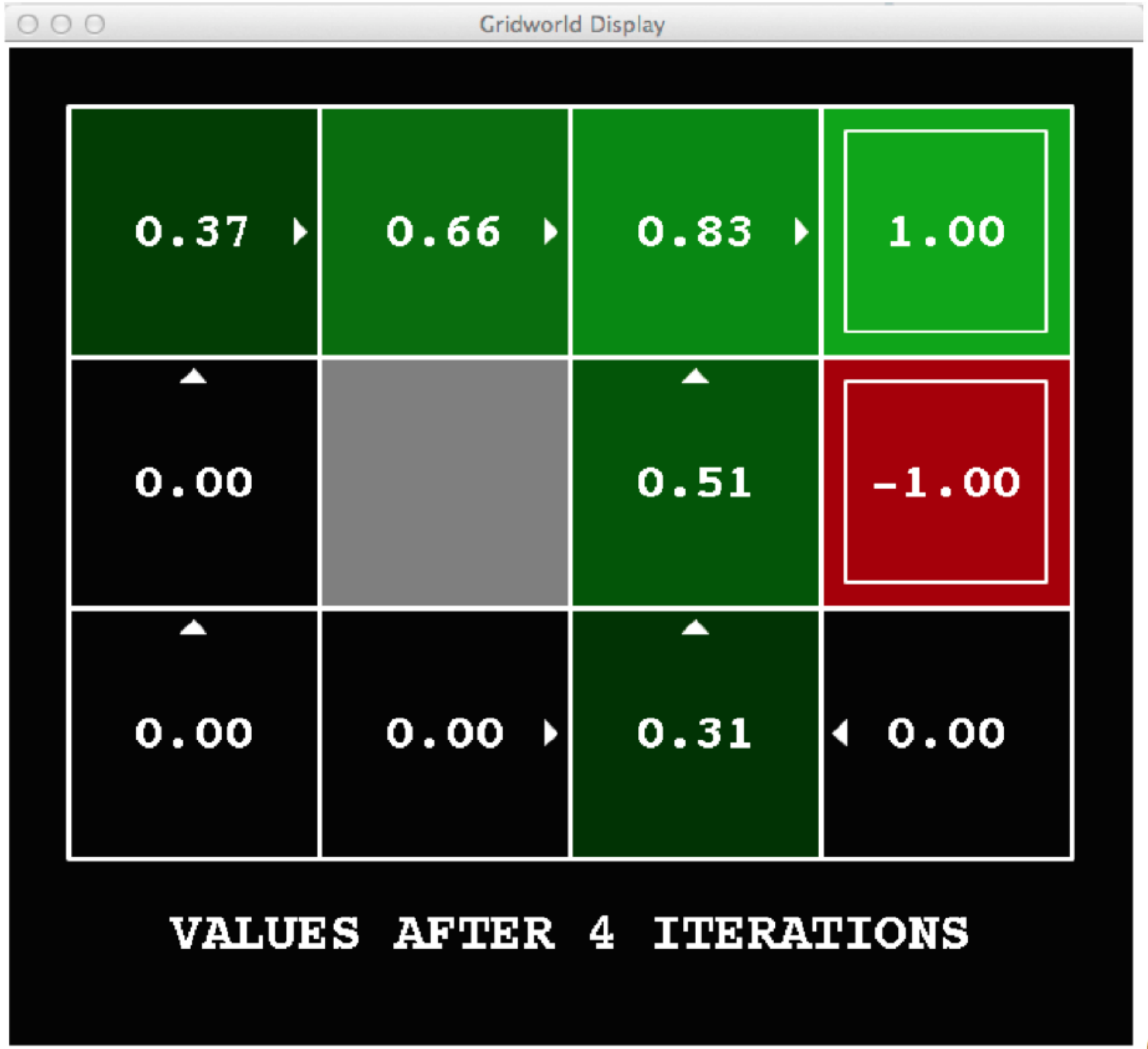


(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (3)

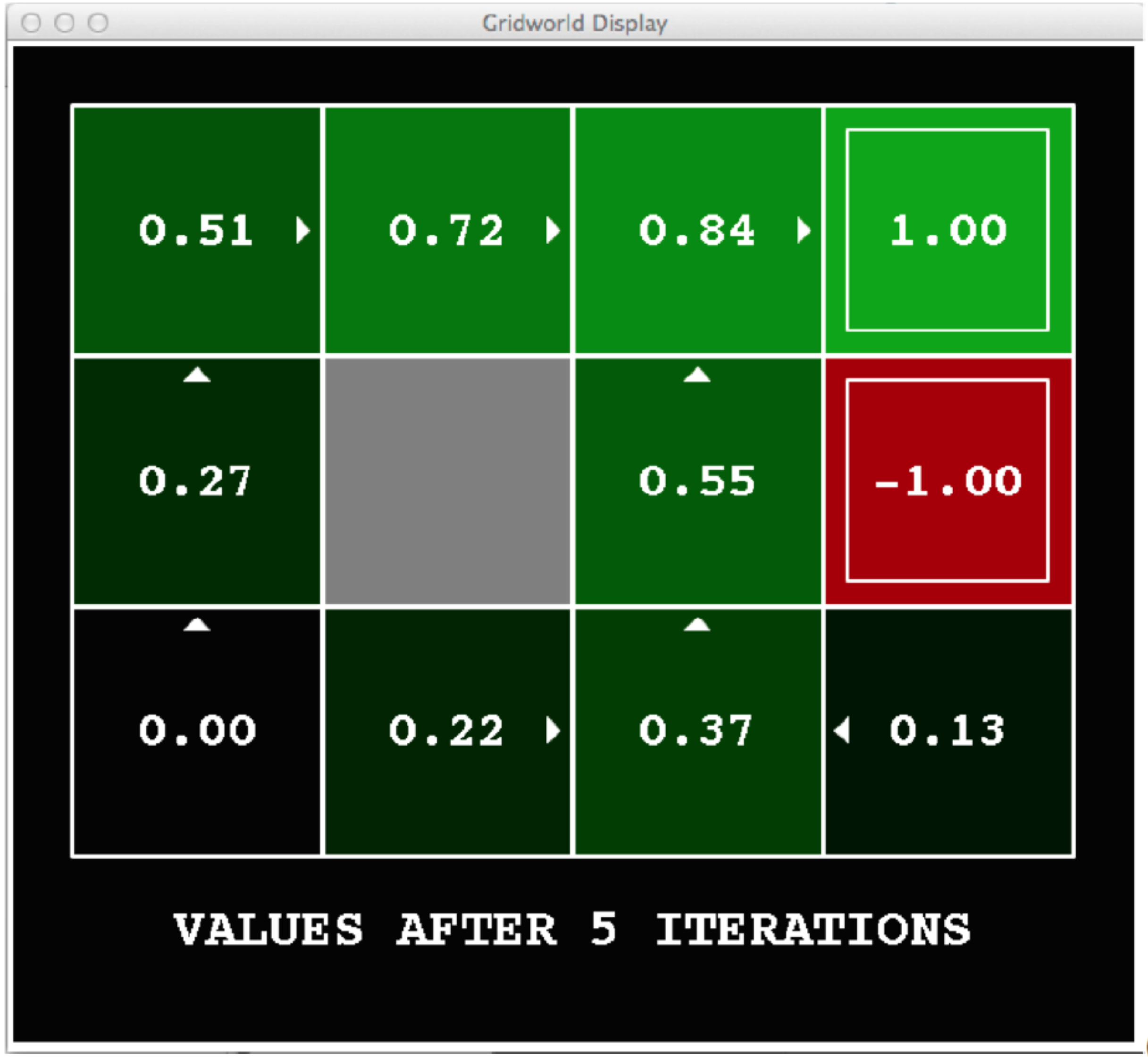
(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (4)



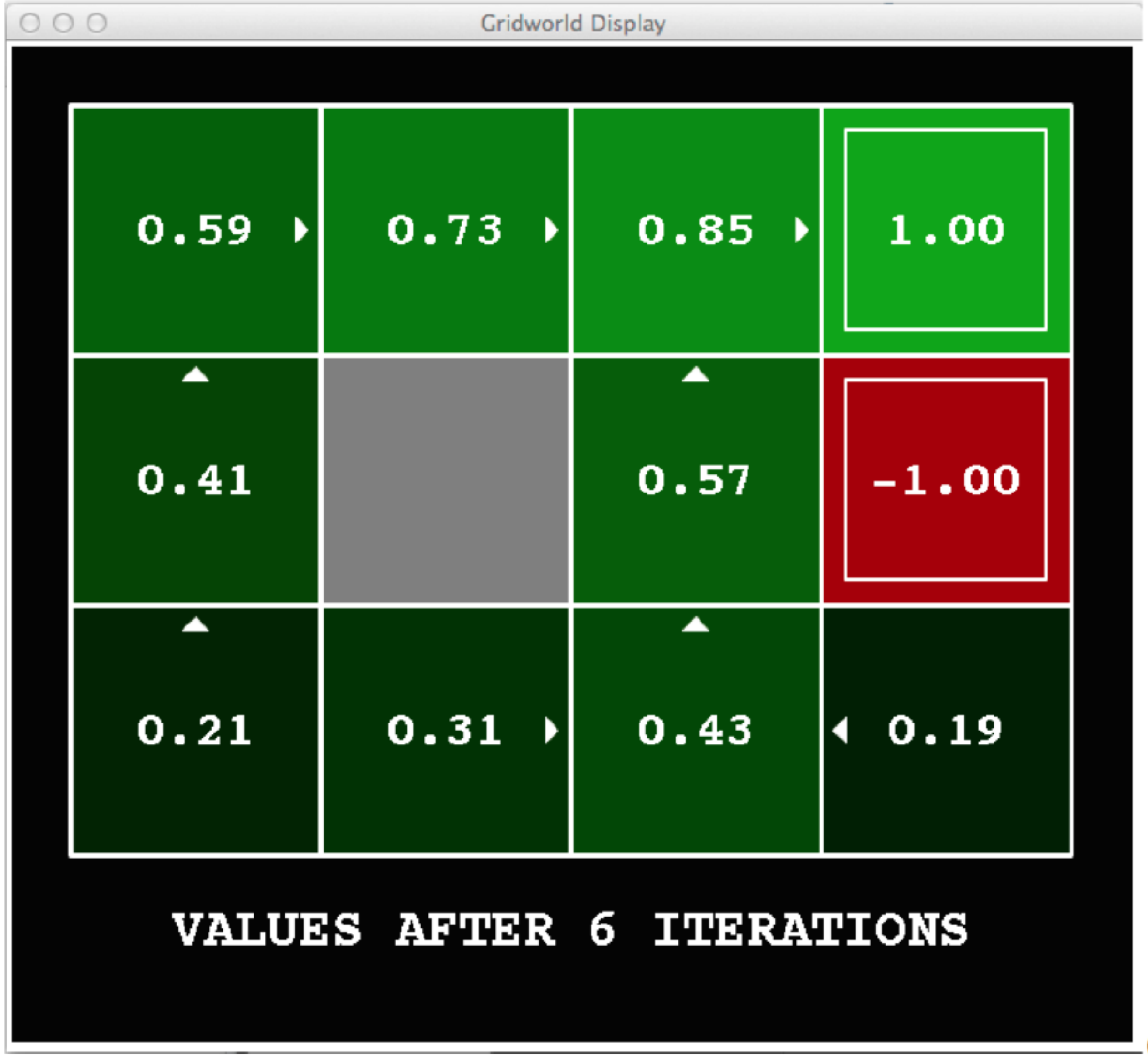
(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (5)



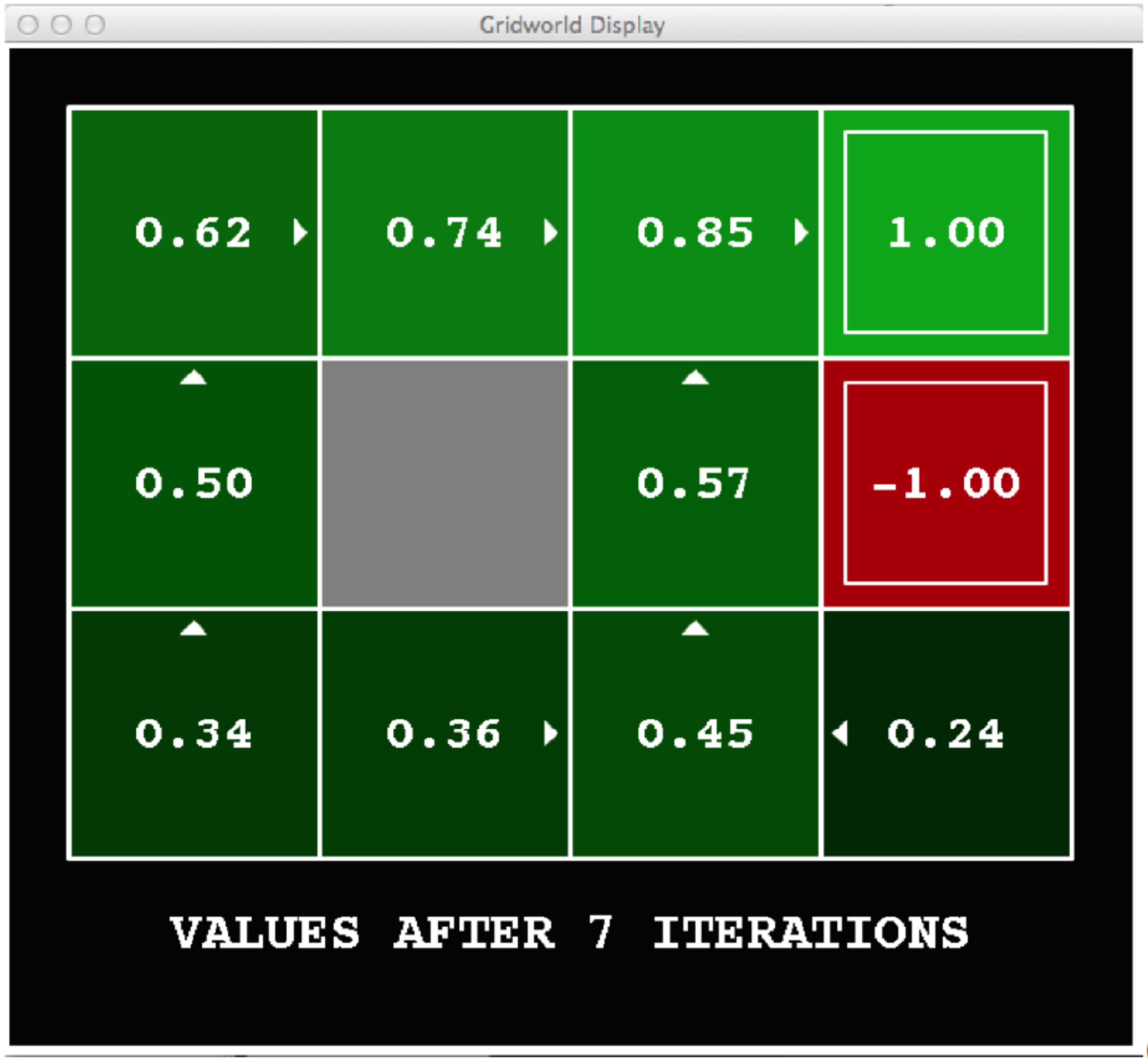
(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (6)



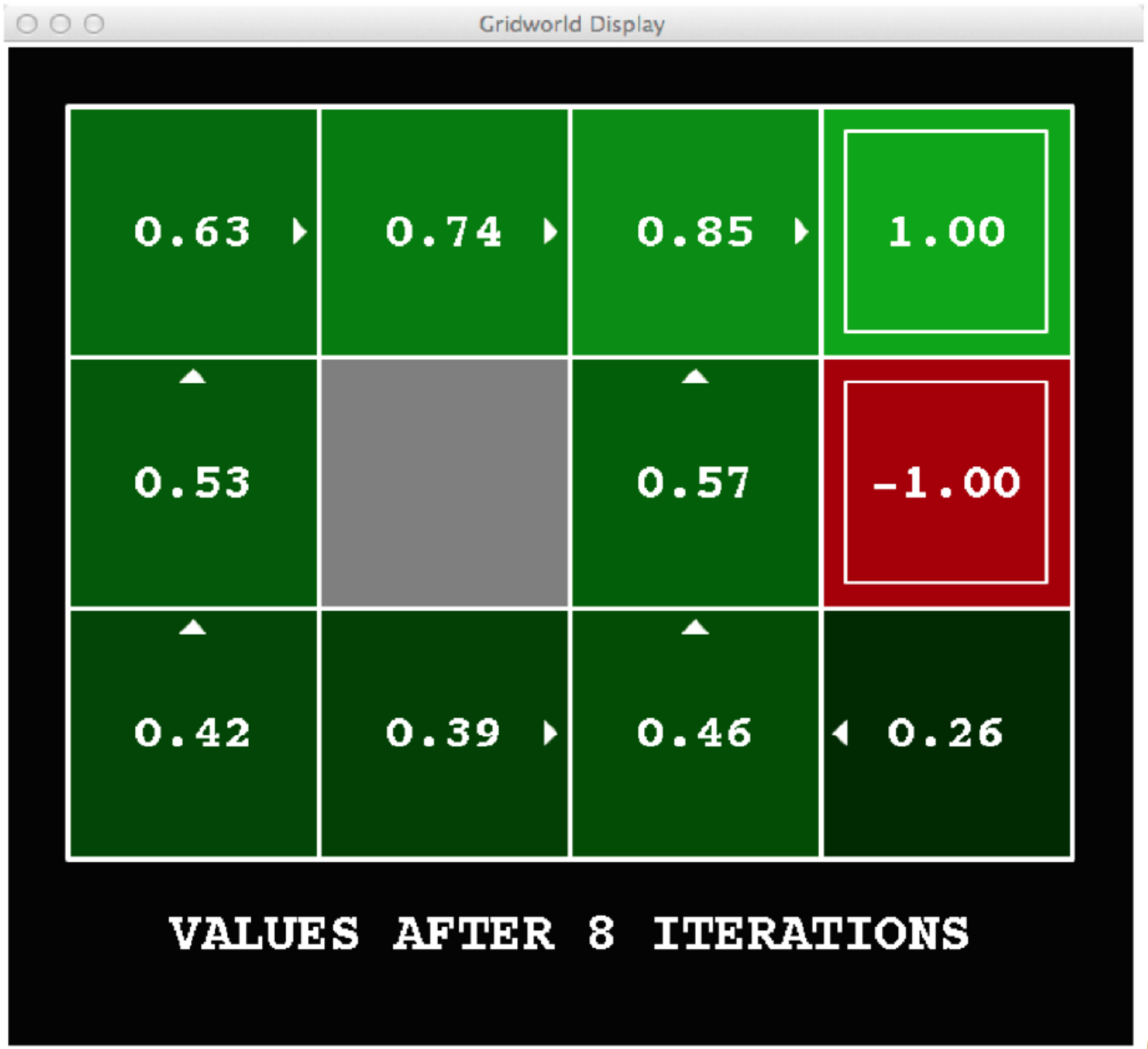
(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (7)



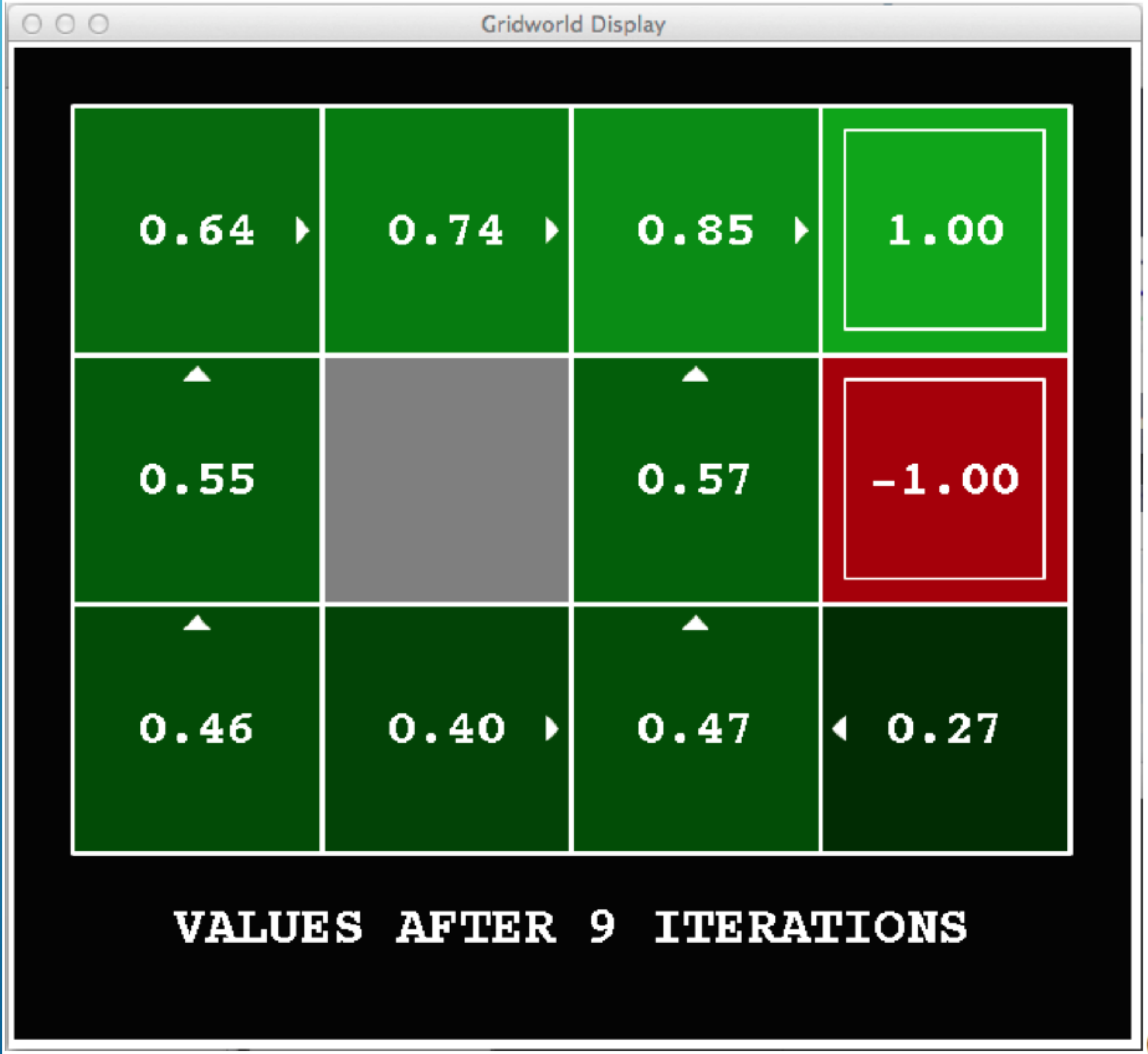
(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (8)



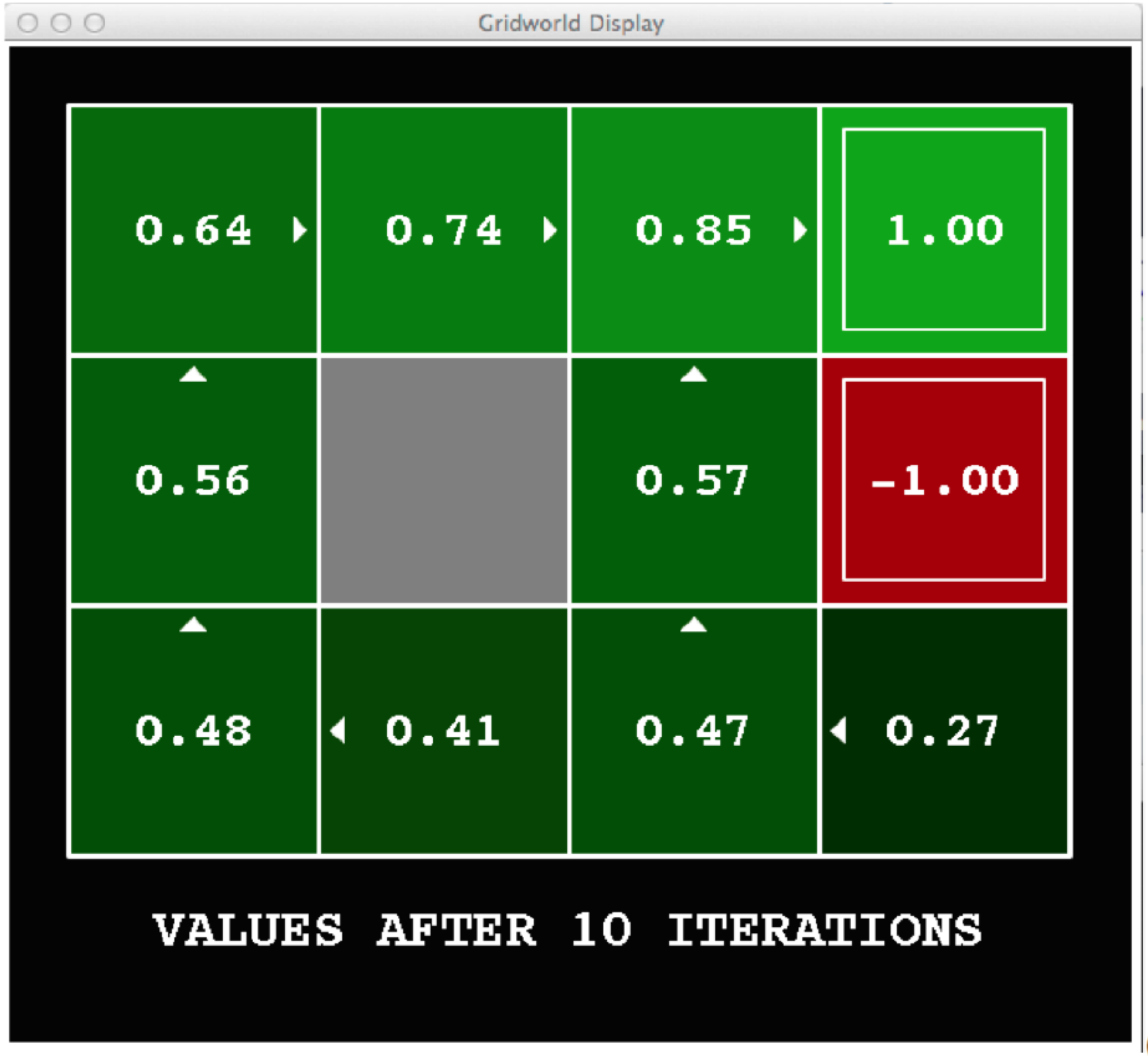
(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (9)



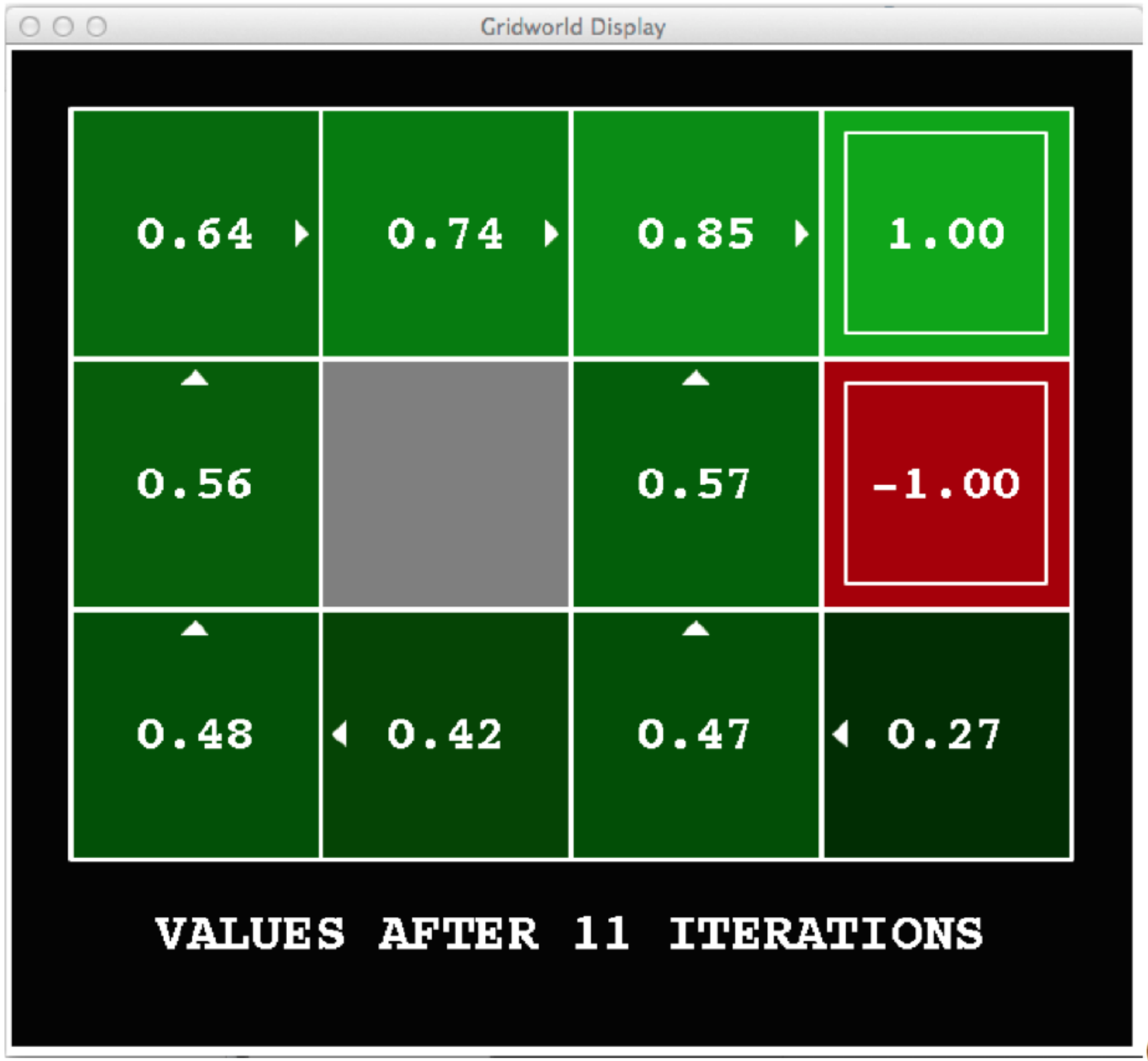
(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (10)



(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (11)

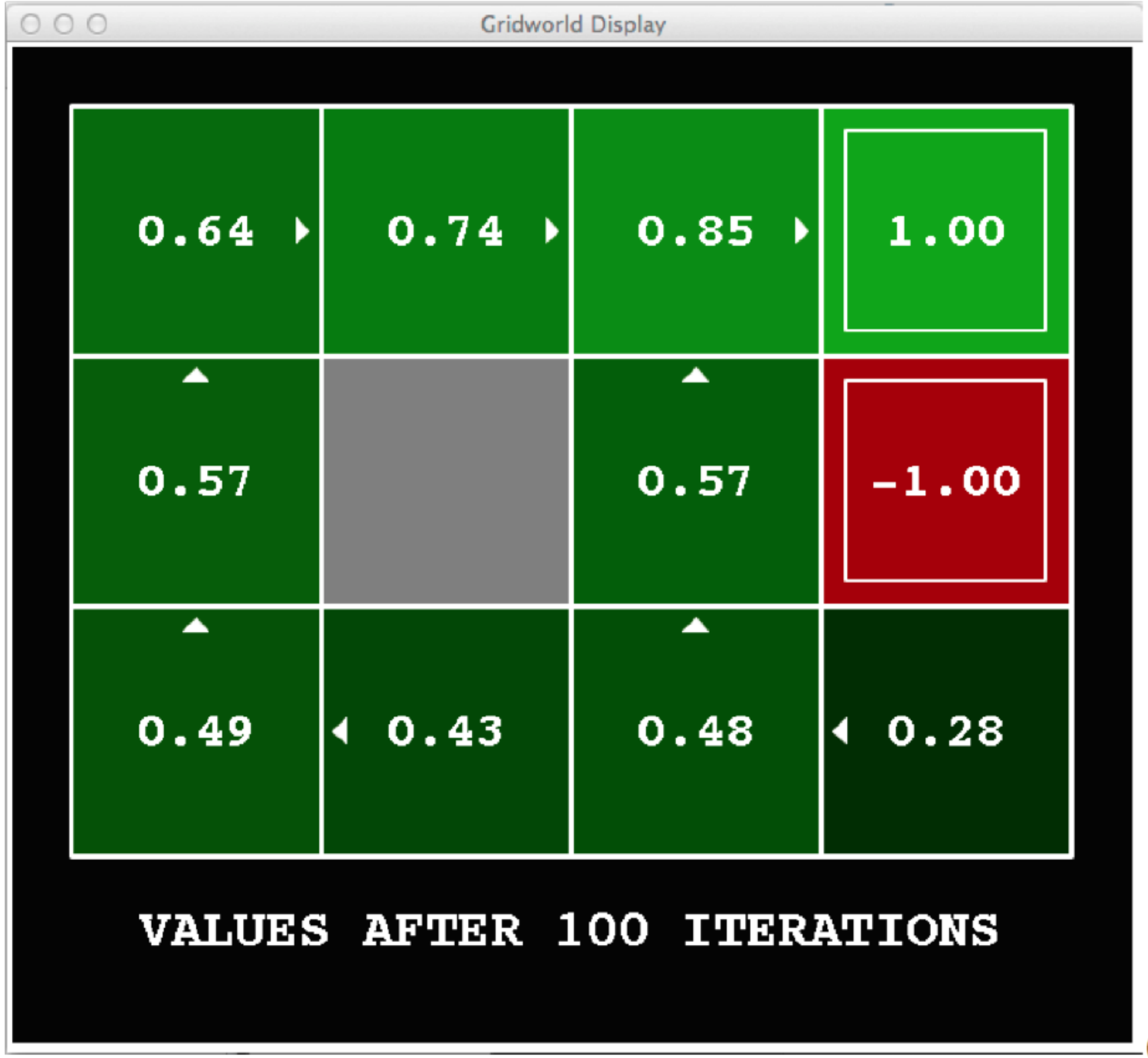


(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (12)

(D. Klein and P. Abbeel)

VALUE ITERATION IN GRIDWORLD (100)



(D. Klein and P. Abbeel)

PROBLEMS WITH VALUE ITERATION

- The ‘max’ value at each state rarely changes.
- The policy often converges long before the values converge.

Policy iteration is an alternative approach, which is still optimal and can converge much more quickly.

POLICY ITERATION

$$\pi^{(0)} \xrightarrow{E} V^{\pi^{(0)}} \xrightarrow{I} \pi^{(1)} \xrightarrow{E} V^{\pi^{(1)}} \xrightarrow{I} \pi^{(2)} \xrightarrow{E} \dots$$

Repeat (until policy converges):

- Evaluate (E) V^{π} (where π is current policy)

- Policy improvement (I):

$$\pi'(s) \leftarrow \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V^{\pi}(s') \right], \quad \forall s$$

  update policy using one-step look-ahead with V^{π} as future values

- $\pi \leftarrow \pi'$

Proof of convergence shows $V^{\pi^{(k+1)}} > V^{\pi^{(k)}}$ (if policy changes).

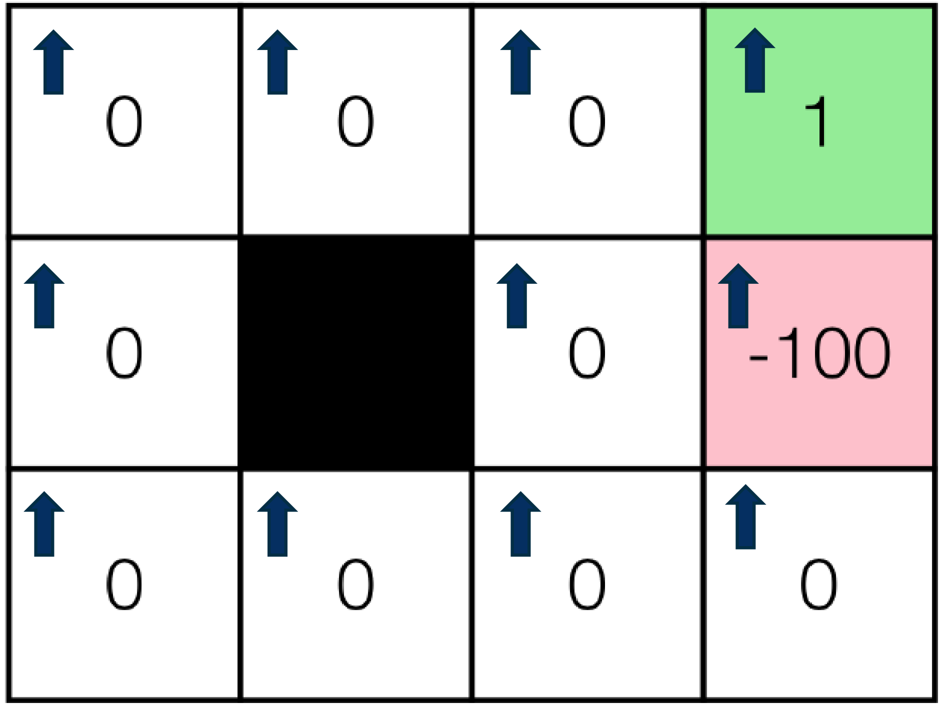
POLICY ITERATION

Algorithm 2 Policy Iteration

function POLICYITERATION(γ) ▷ Takes a discount factor as input.
  $\pi(\cdot) \leftarrow \pi_0(\cdot)$ ▷ Initialize the policy in any way.
  repeat
    $\pi_{\text{old}}(\cdot) \leftarrow \pi(\cdot)$ ▷ Store off the old policy.
    Solve system: $V(s) = R(s, \pi_{\text{old}}(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi_{\text{old}}(s)) V(s')$ for $V(s)$.
    for $s \in \mathcal{S}$ **do** ▷ Loop over all states.
      for $a \in \mathcal{A}$ **do** ▷ Loop over all actions.
        $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V(s')$ ▷ Compute Q-function.
      end for
      $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} Q(s,a)$ ▷ Update policy to be optimal for current Q.
    end for
  until $\pi(s) = \pi_{\text{old}}(s), \forall s \in \mathcal{S}$ ▷ Loop until the policy converges.
  Return $\pi(\cdot)$
end function

POLICY ITERATION EXAMPLE (0)

Example on a different grid world, initialized with $\pi(s) = \uparrow$ (all states). NOTE: don't stop in goal states in this grid world, thus MDP value can be < -100 when in -100 state.

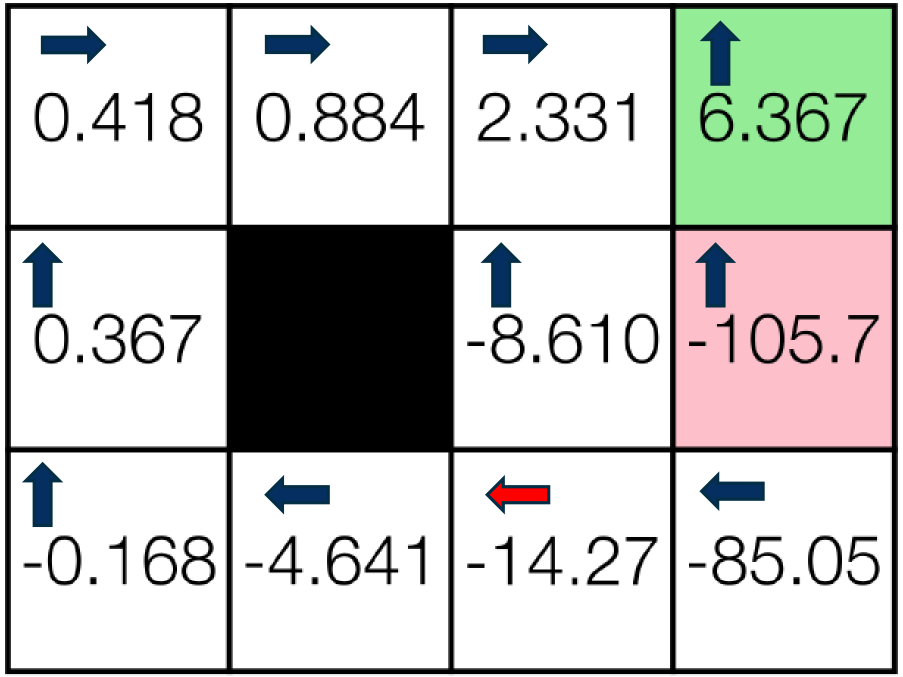


Z. Kolter

Original reward function

POLICY ITERATION EXAMPLE (1)

Example on a different grid world, initialized with $\pi(s) = \uparrow$ (all states). NOTE: don't stop in goal states in this grid world, thus MDP value can be < -100 when in -100 state.

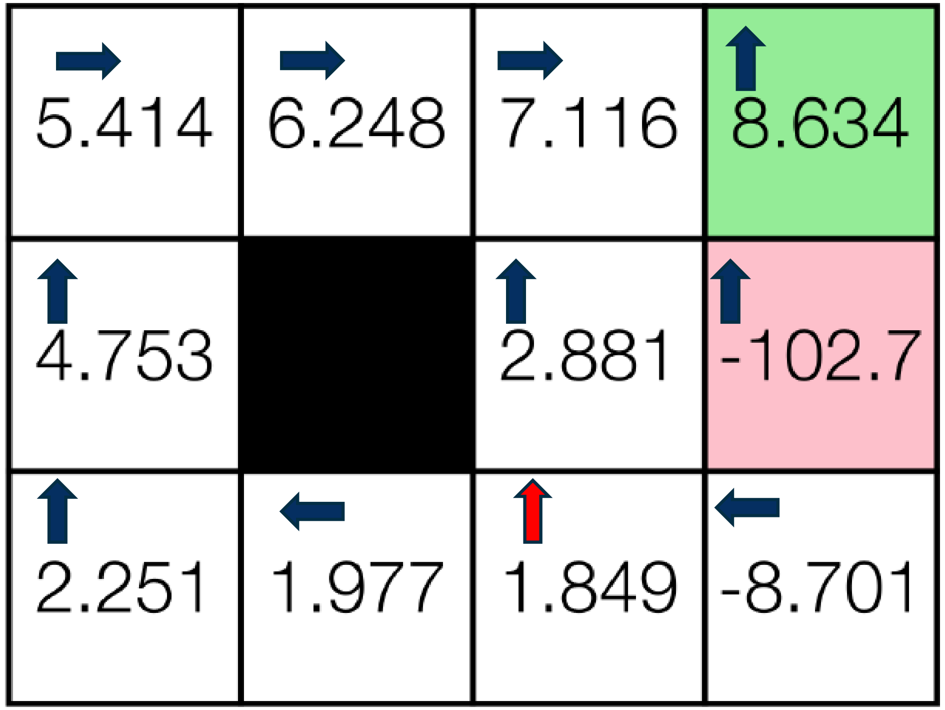


Z. Kolter

V^{π} at one iteration

POLICY ITERATION EXAMPLE (2)

Example on a different grid world, initialized with $\pi(s) = \uparrow$ (all states). NOTE: don't stop in goal states in this grid world, thus MDP value can be < -100 when in -100 state.

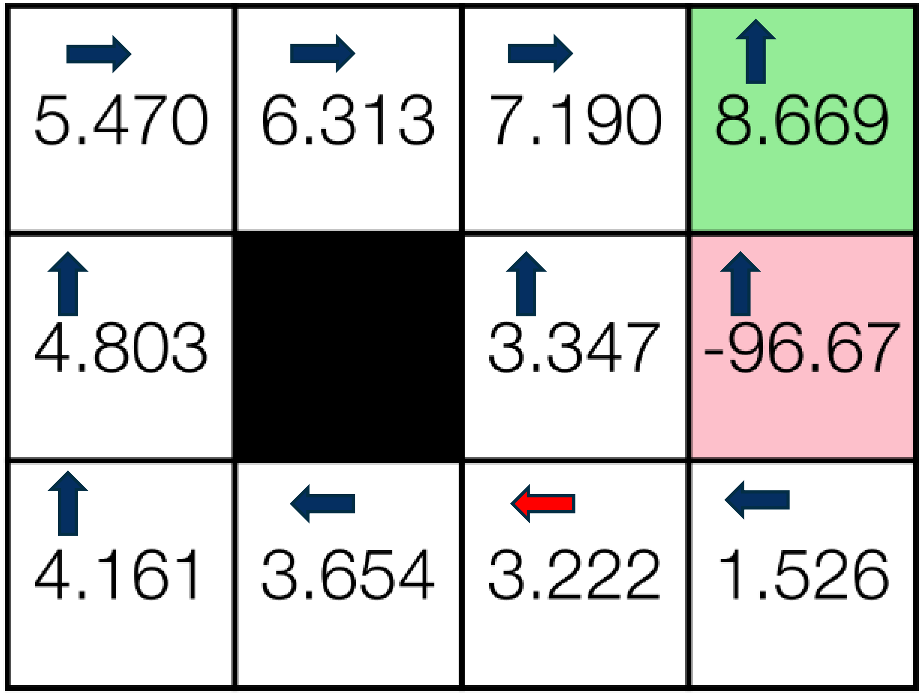


Z. Kolter

V^π at two iterations

POLICY ITERATION EXAMPLE (3)

Example on a different grid world, initialized with $\pi(s) = \uparrow$ (all states). NOTE: don't stop in goal states in this grid world, thus MDP value can be < -100 when in -100 state.



Z. Kolter

V^π at three iterations (converged!)