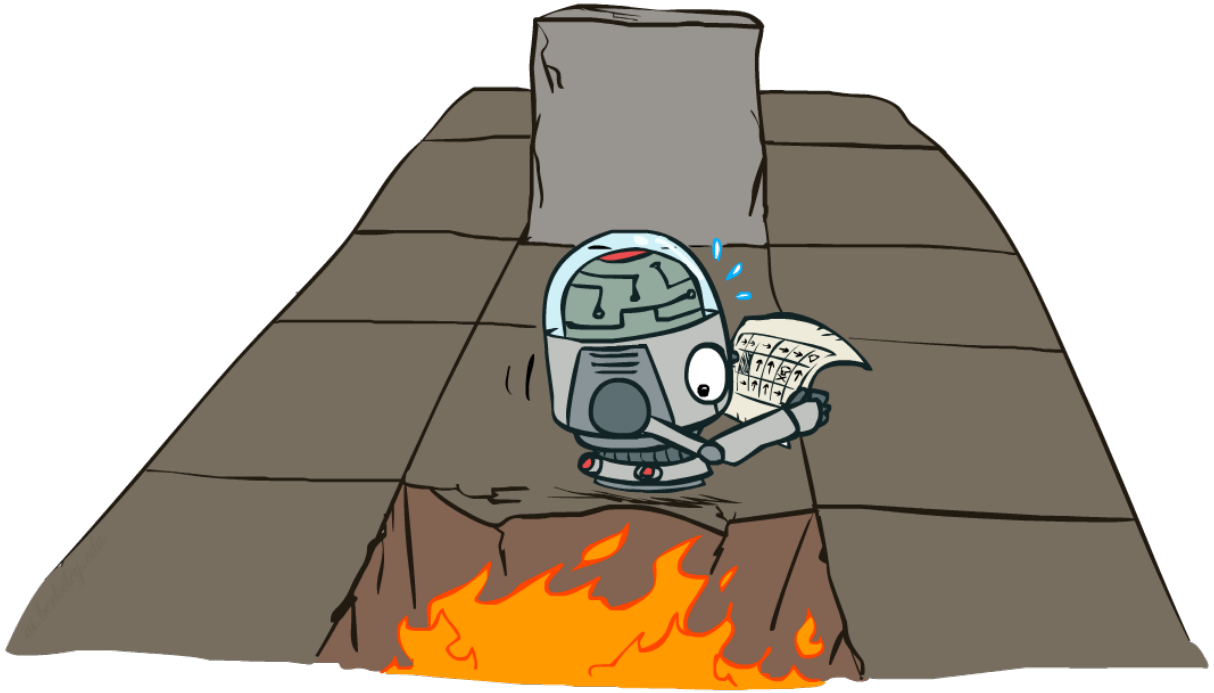
Announcements

- HW3 is due **Friday, November 10**, 11:59 PM

CS 3317: Artificial Intelligence

Markov Decision Processes II



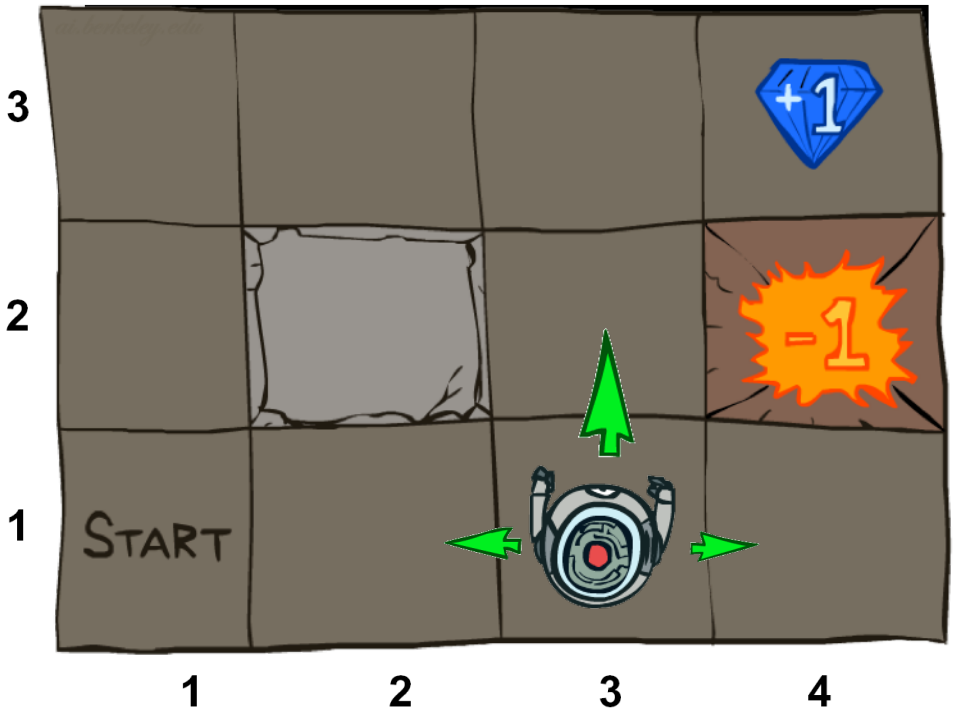
Instructors: **Cai Panpan**

Shanghai Jiao Tong University

(slides adapted from UC Berkeley CS188)

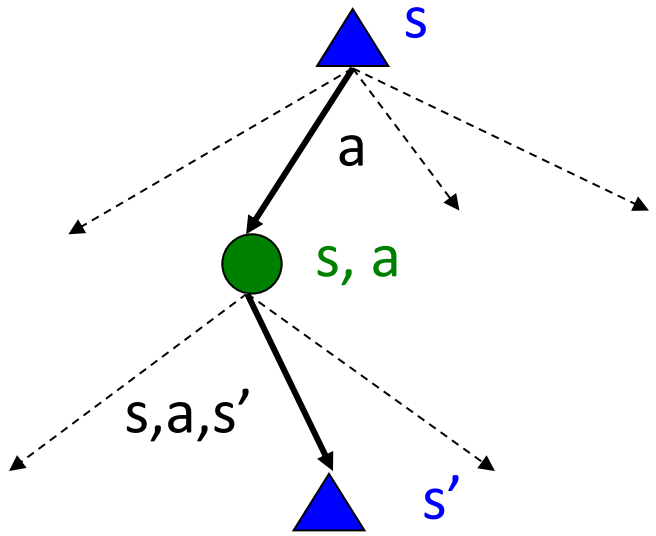
Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North
  - 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 sum of (discounted) rewards



Recap: MDPs

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



- Quantities:
  - Policy = map of states to actions
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state (max node)
  - Q-Values = expected future utility from a q-state (chance node)

Optimal Quantities

- The value (utility) of a state s:

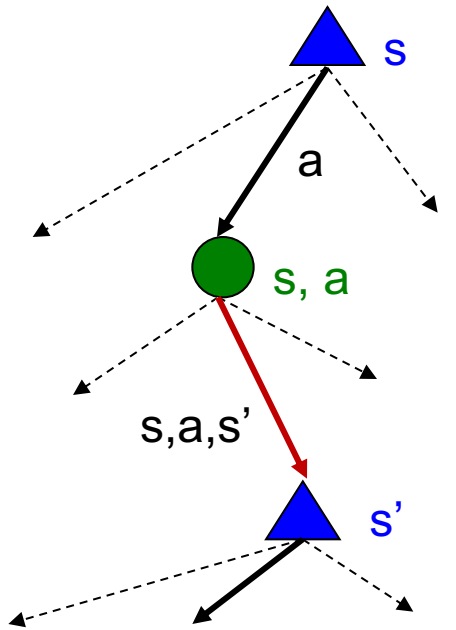
  $V^*(s)$ = expected utility starting in s and acting optimally

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

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s

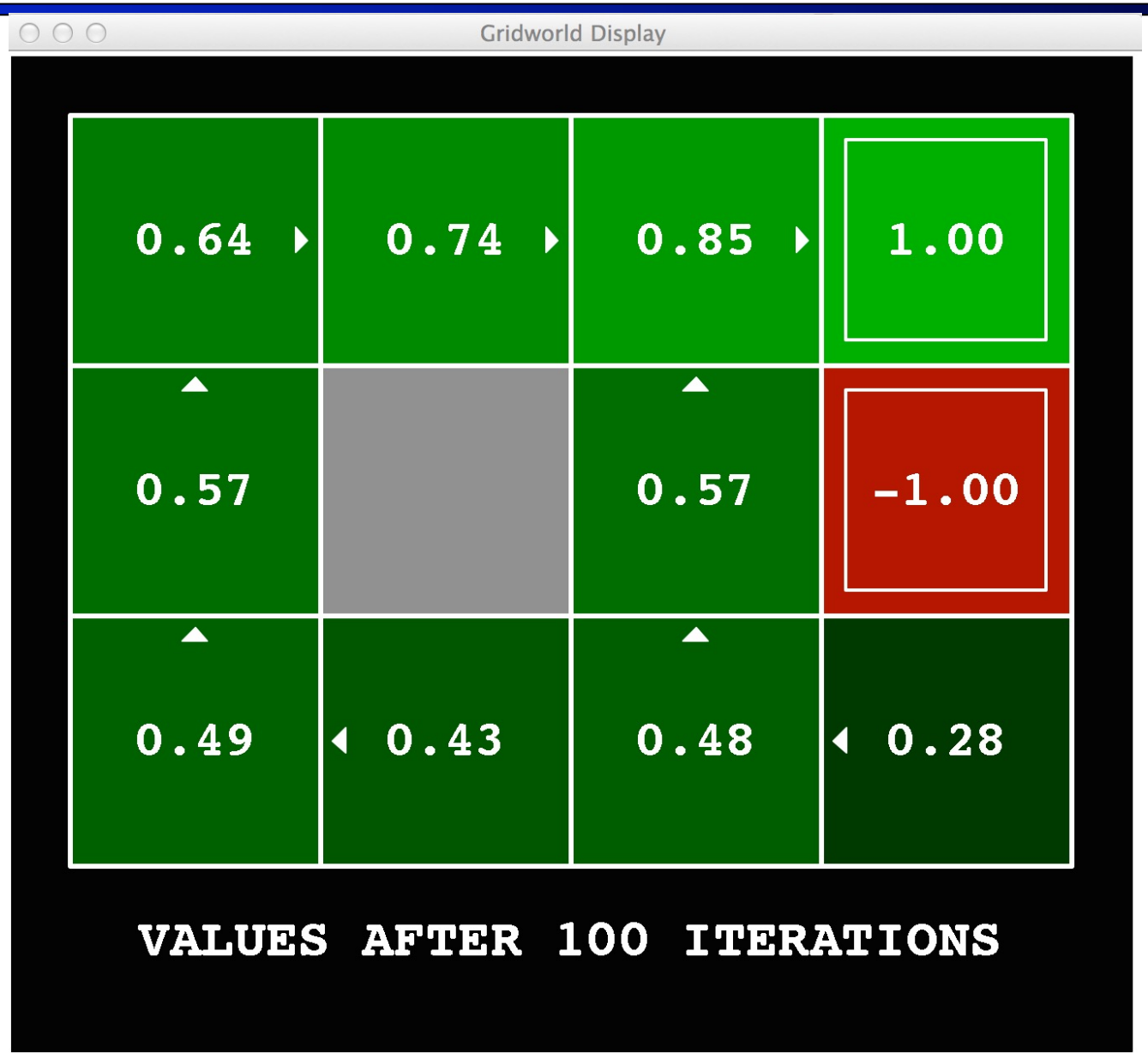


s is a *state*

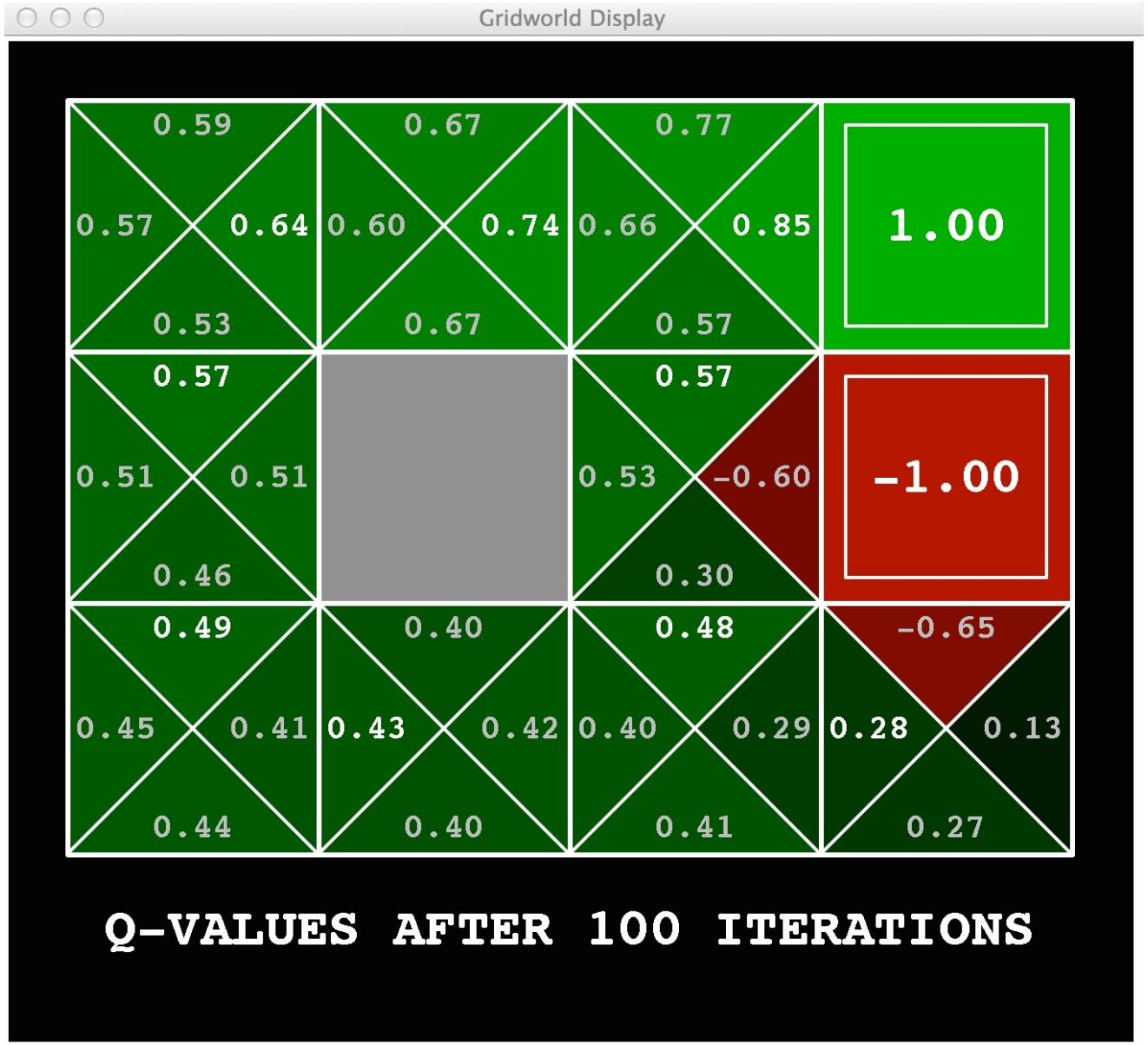
(s, a) is a *q-state*

(s,a,s') is a *transition*

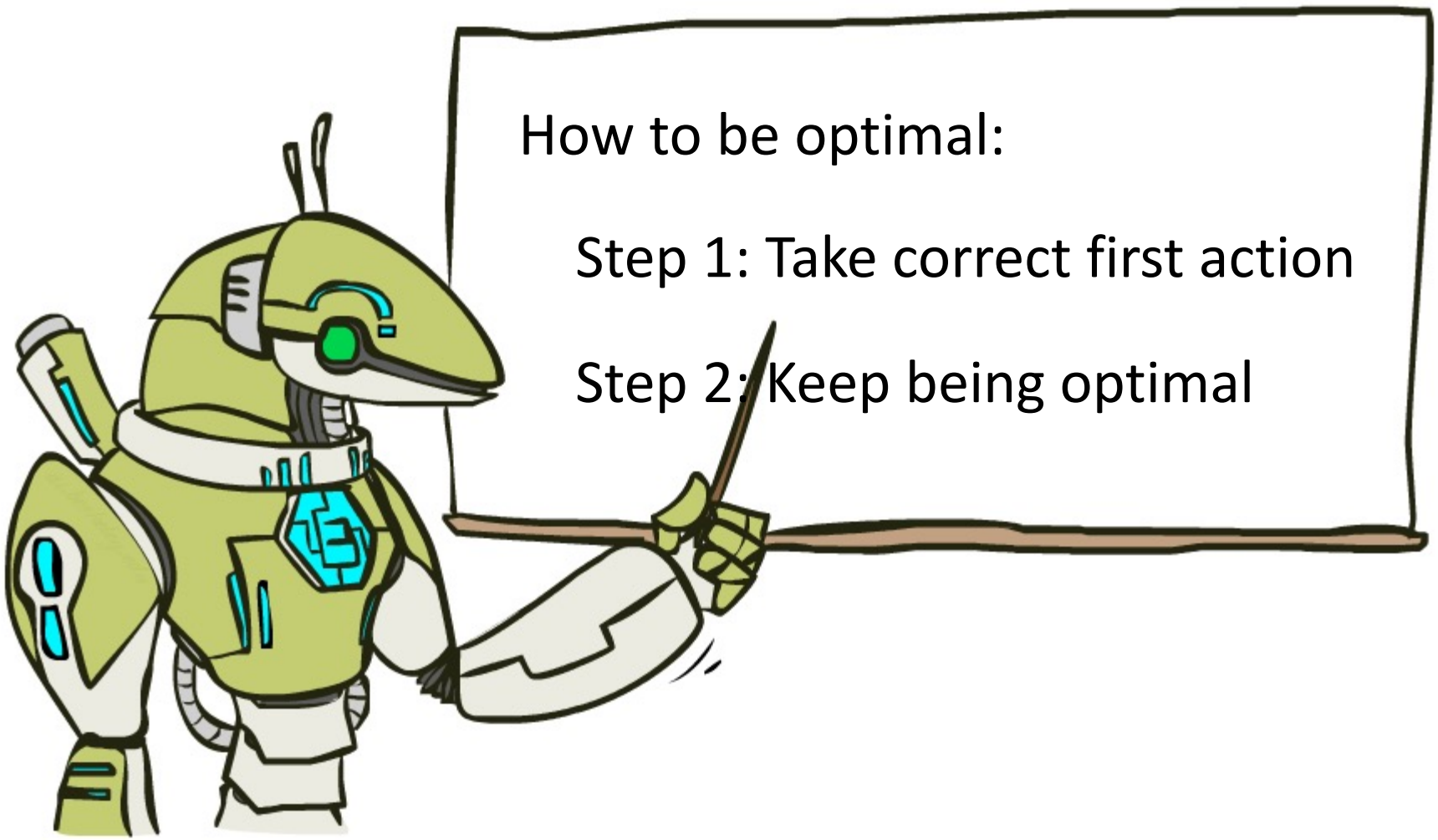
Gridworld Values V*



Gridworld: Q*


Gridworld Display
0.59
0.57
0.64
0.53
0.67
0.60
0.74
0.67
0.77
0.66
0.85
0.57
1.00
0.57
0.51
0.51
0.46
0.57
0.53
-0.60
0.30
-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
Q-VALUES AFTER 100 ITERATIONS

The Bellman Equations



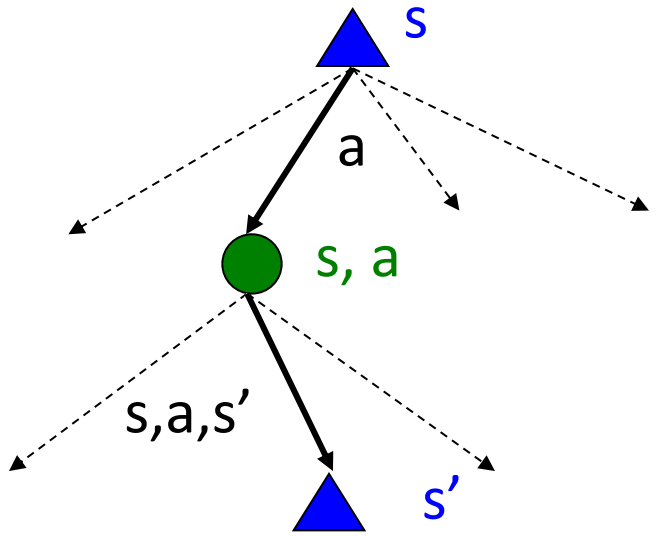
The Bellman Equations

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

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

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

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

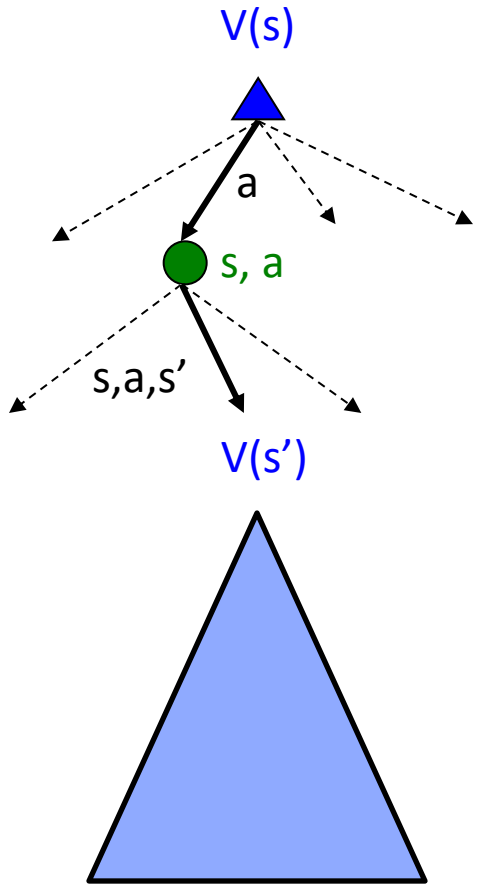


- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

Value Iteration

- Bellman equations characterize the optimal values:

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



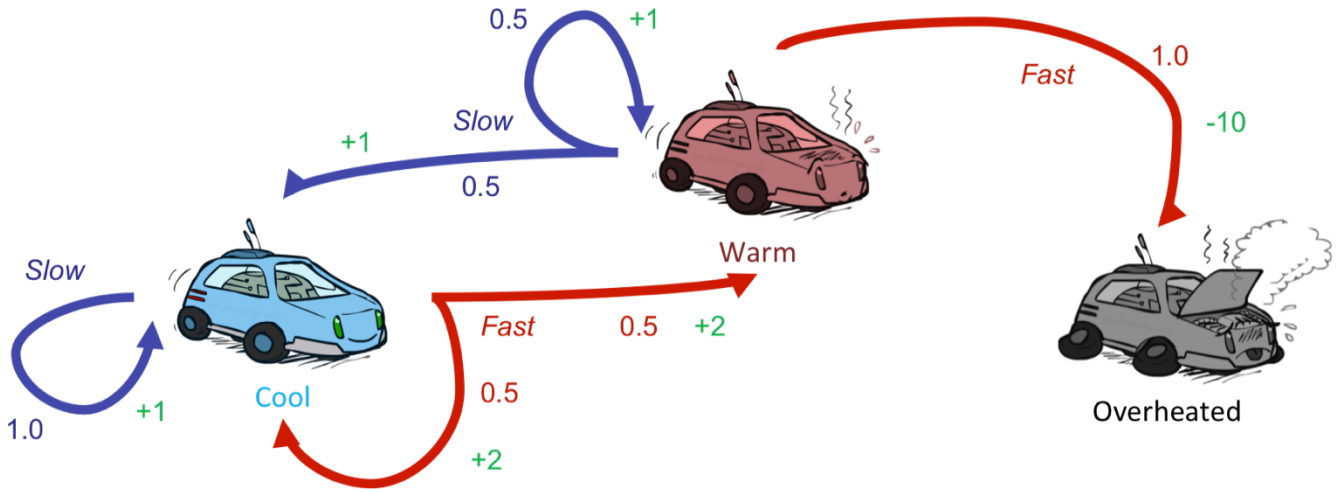
- Value iteration computes them:

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

- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values

Example: Value Iteration

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

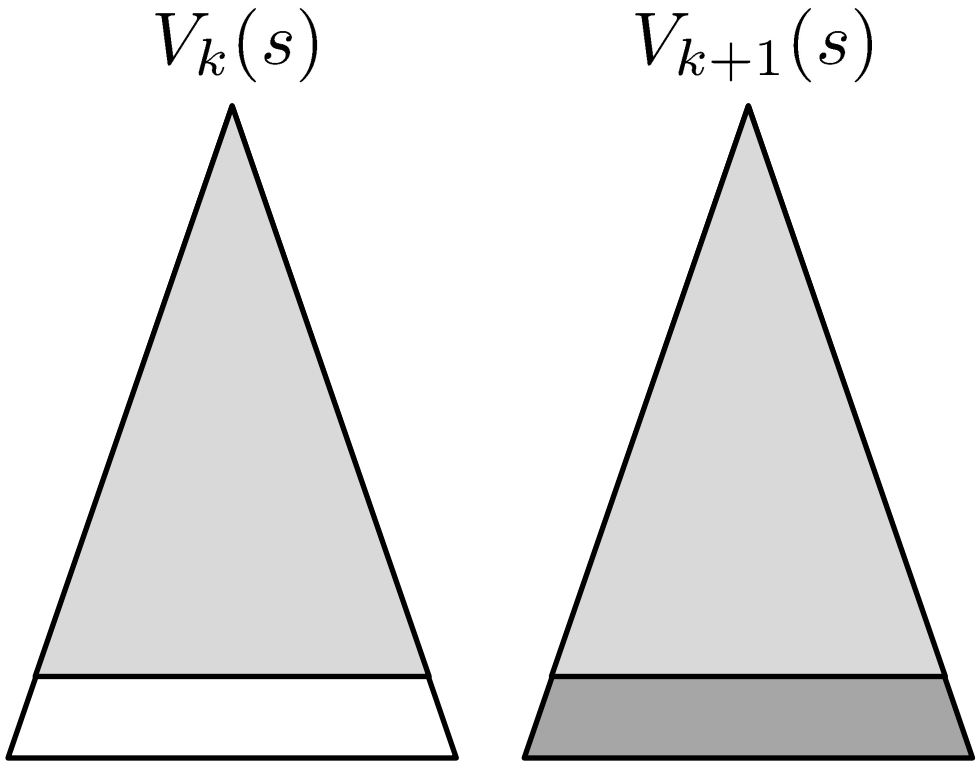


Assume no discount!

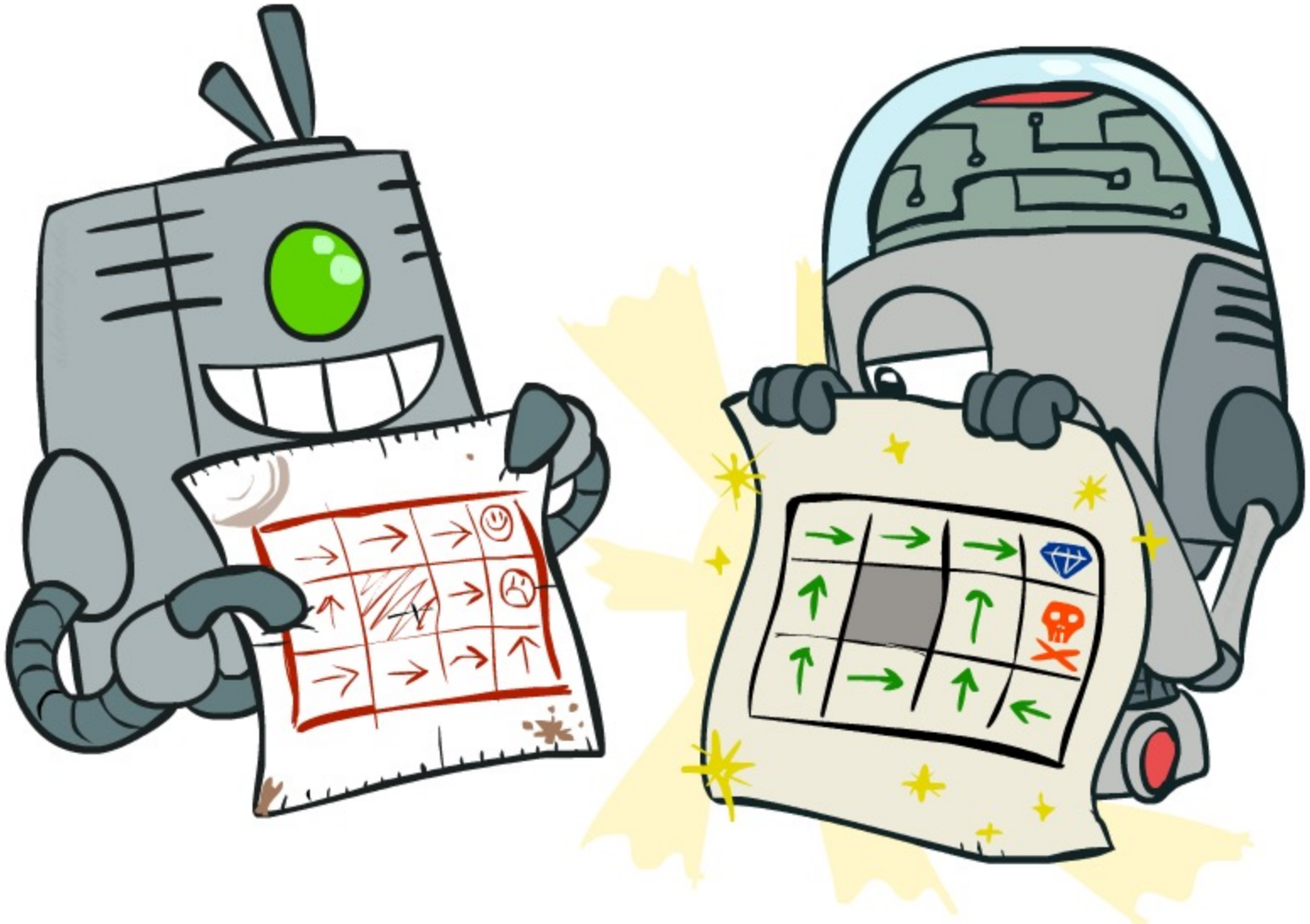
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Convergence*

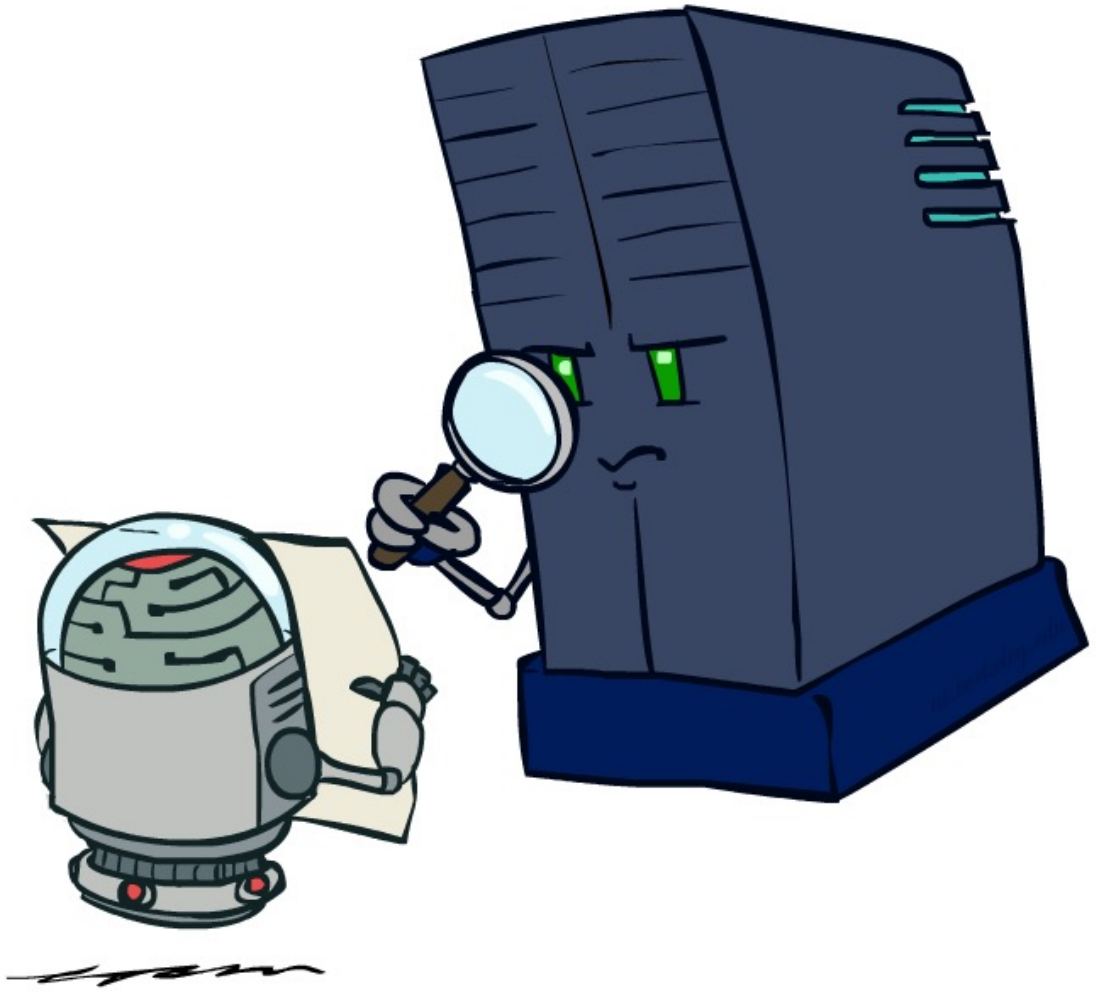
- How do we know the V_k vectors are going to converge?
- Case 1: If the tree has maximum depth M, then V_M holds the actual untruncated values
- Case 2: If the discount is less than 1
  - Sketch: For any state V_k and V_{k+1} can be viewed as depth k+1 expectimax results in nearly identical search trees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



Policy Methods

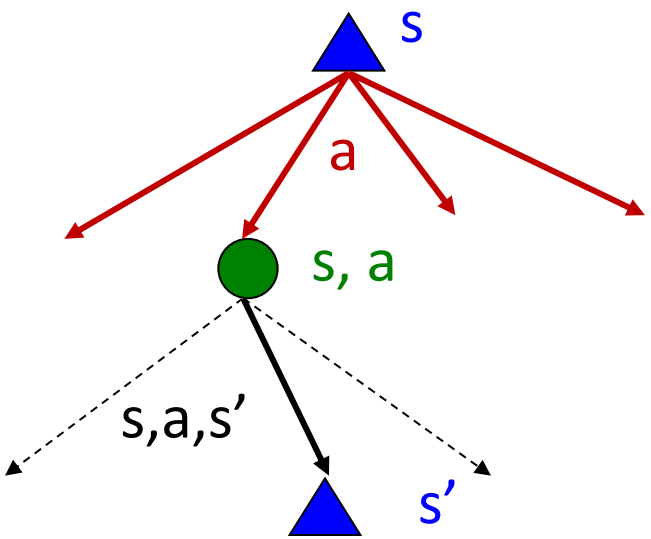


Policy Evaluation

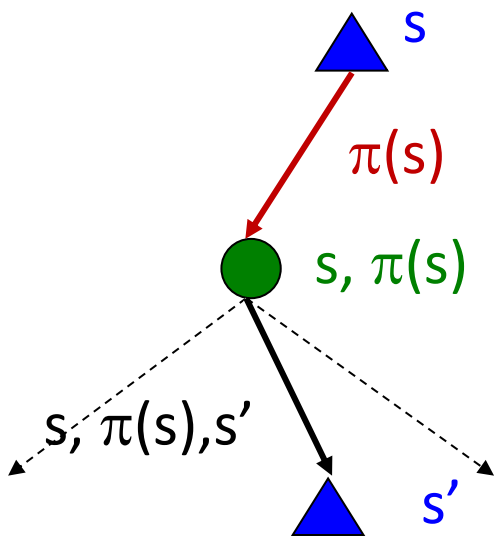


Fixed Policies

Do the optimal action



Do what π says to do



- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy π(s), then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

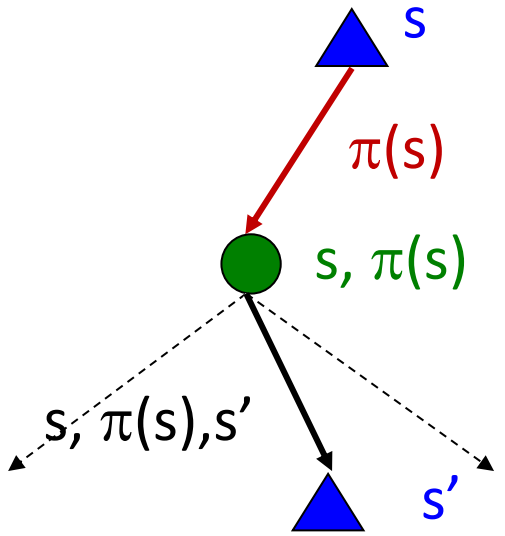
Evaluating Policy Values

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

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

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

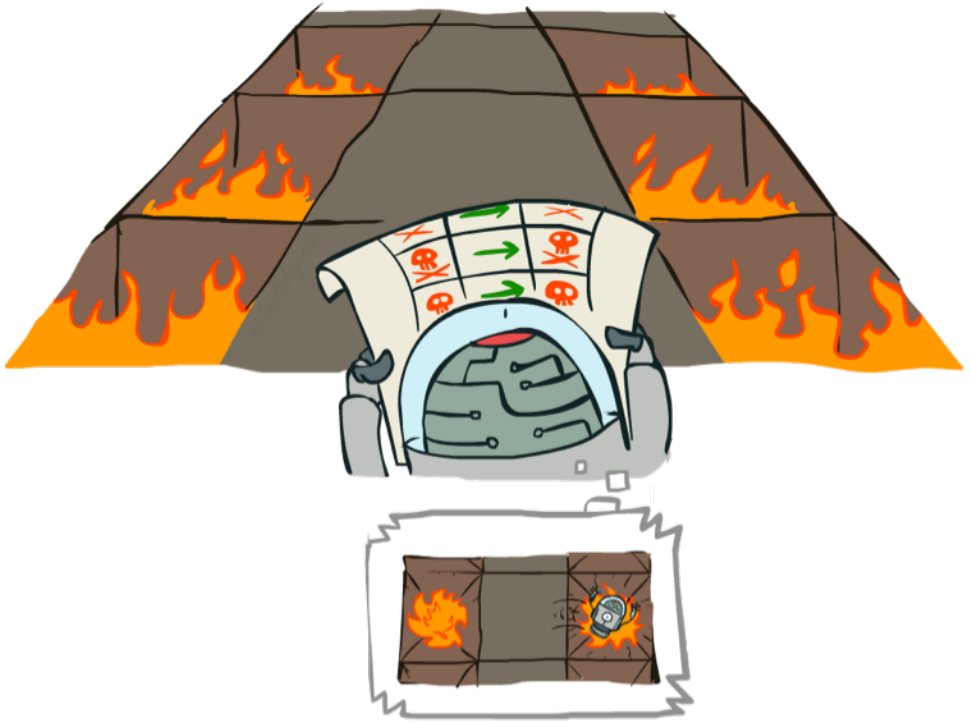
- Recursive relation (one-step look-ahead / Bellman equation):



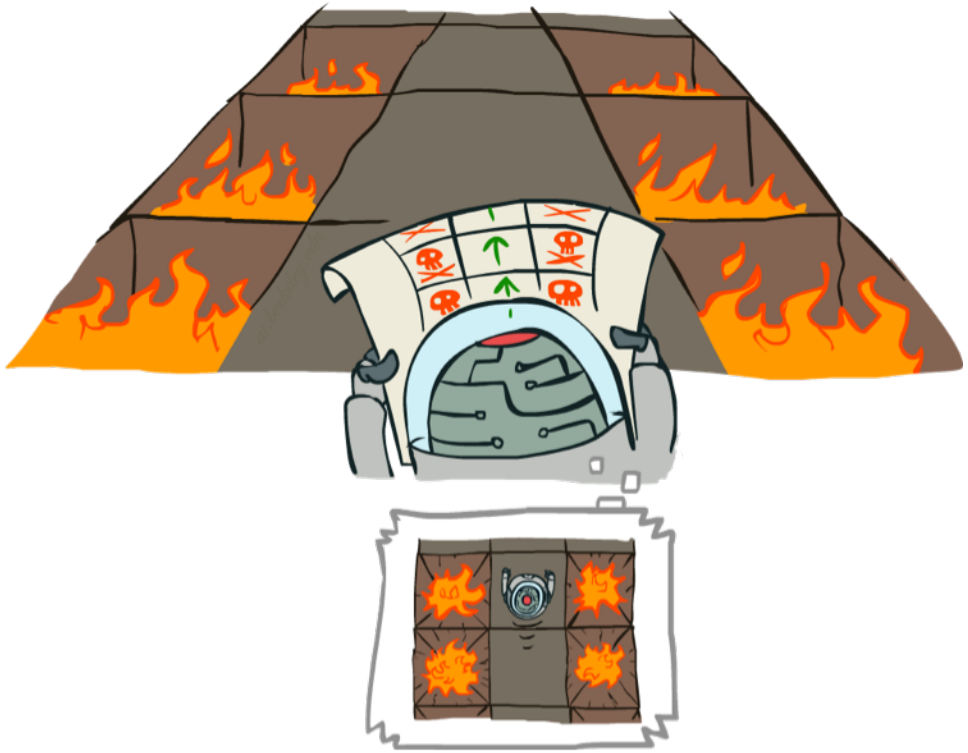
$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

Example: Policy Evaluation

Always Go Right



Always Go Forward



Example: Policy Evaluation

Always Go Right

Always Go Forward

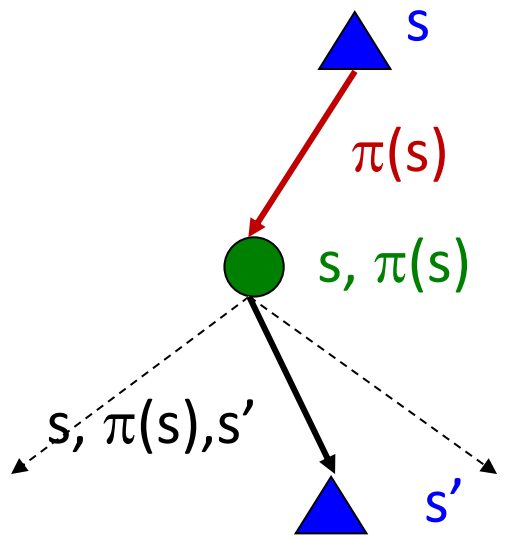
Policy Evaluation

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

- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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

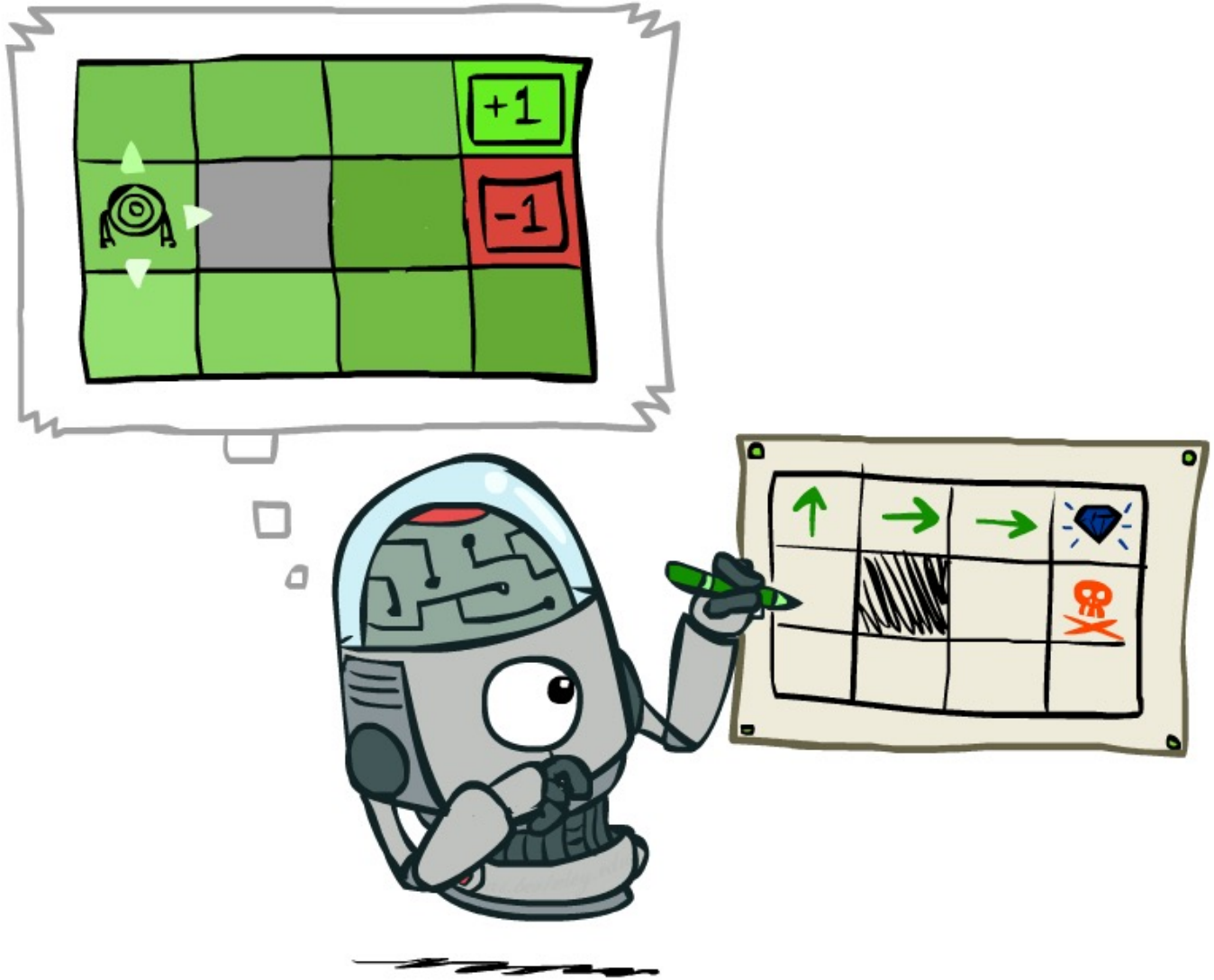
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



- Efficiency: $O(S^2)$ per iteration

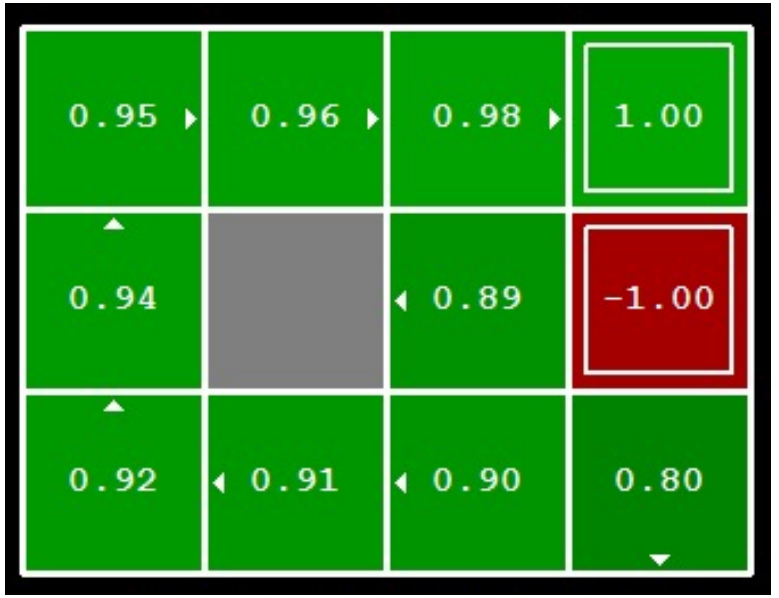
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

Policy Extraction



Computing Actions from Values

- Let's imagine we have the optimal values V*(s)
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)

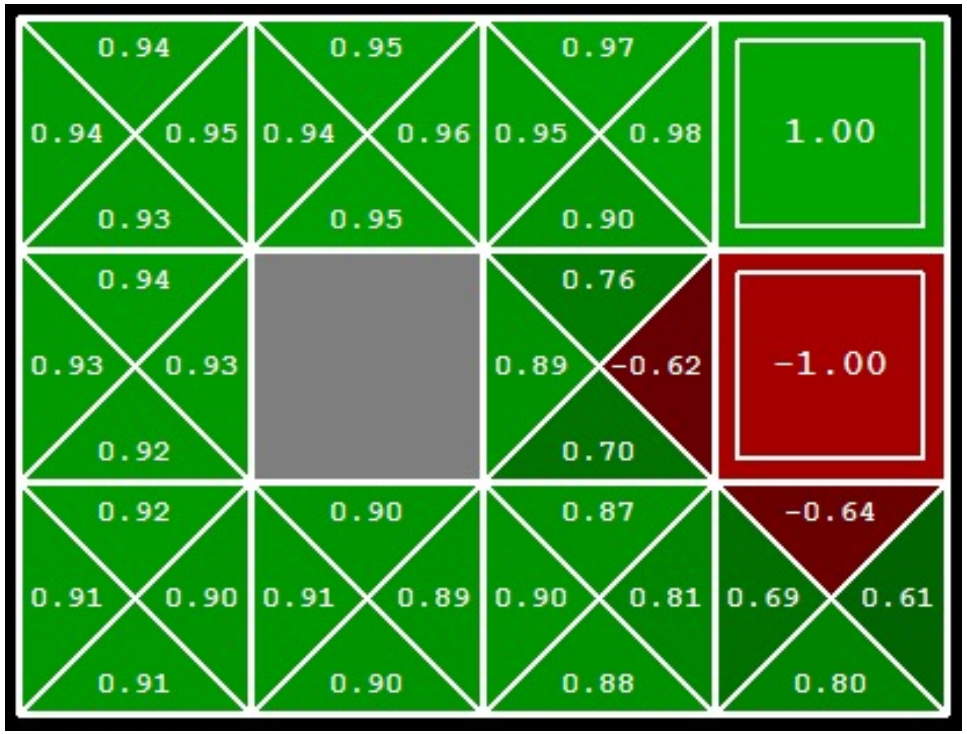


$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

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

Computing Actions from Q-Values

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

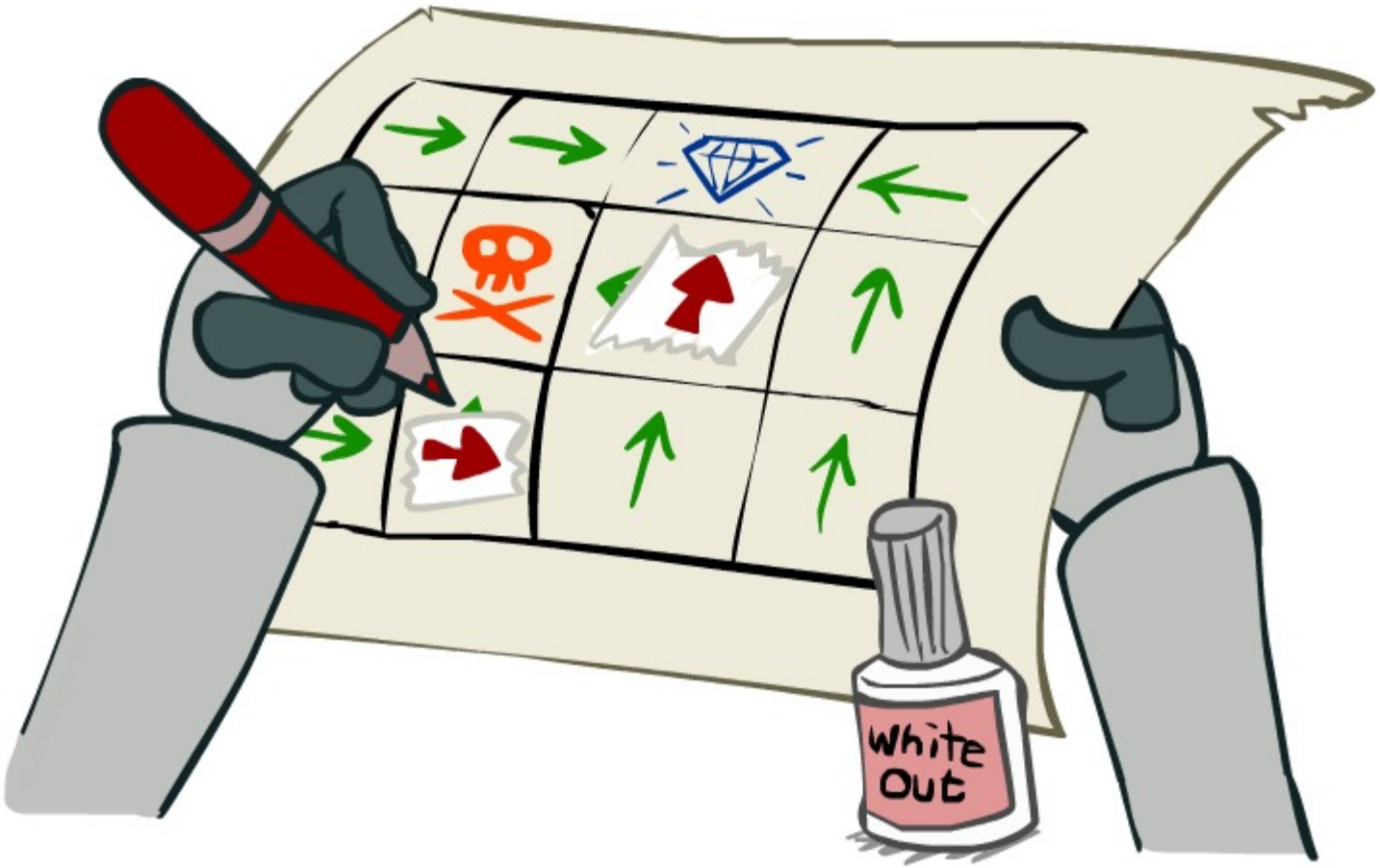


- How should we act?
  - Completely trivial to decide!

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

- Important lesson: actions are easier to select from q-values than values!

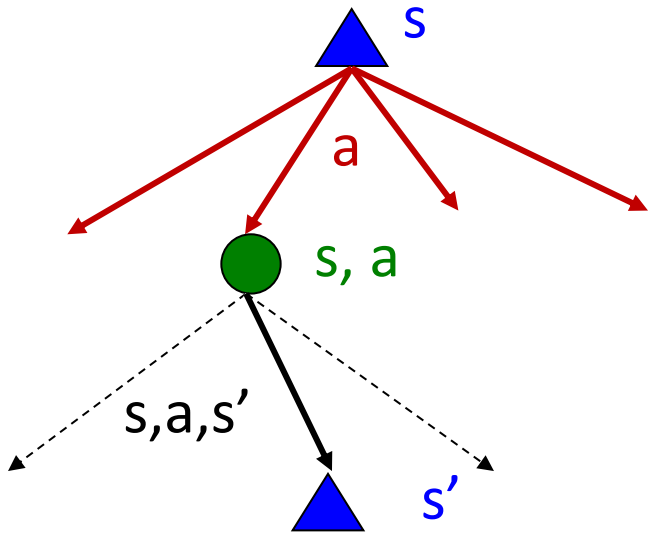
Policy Iteration



Problems with Value Iteration

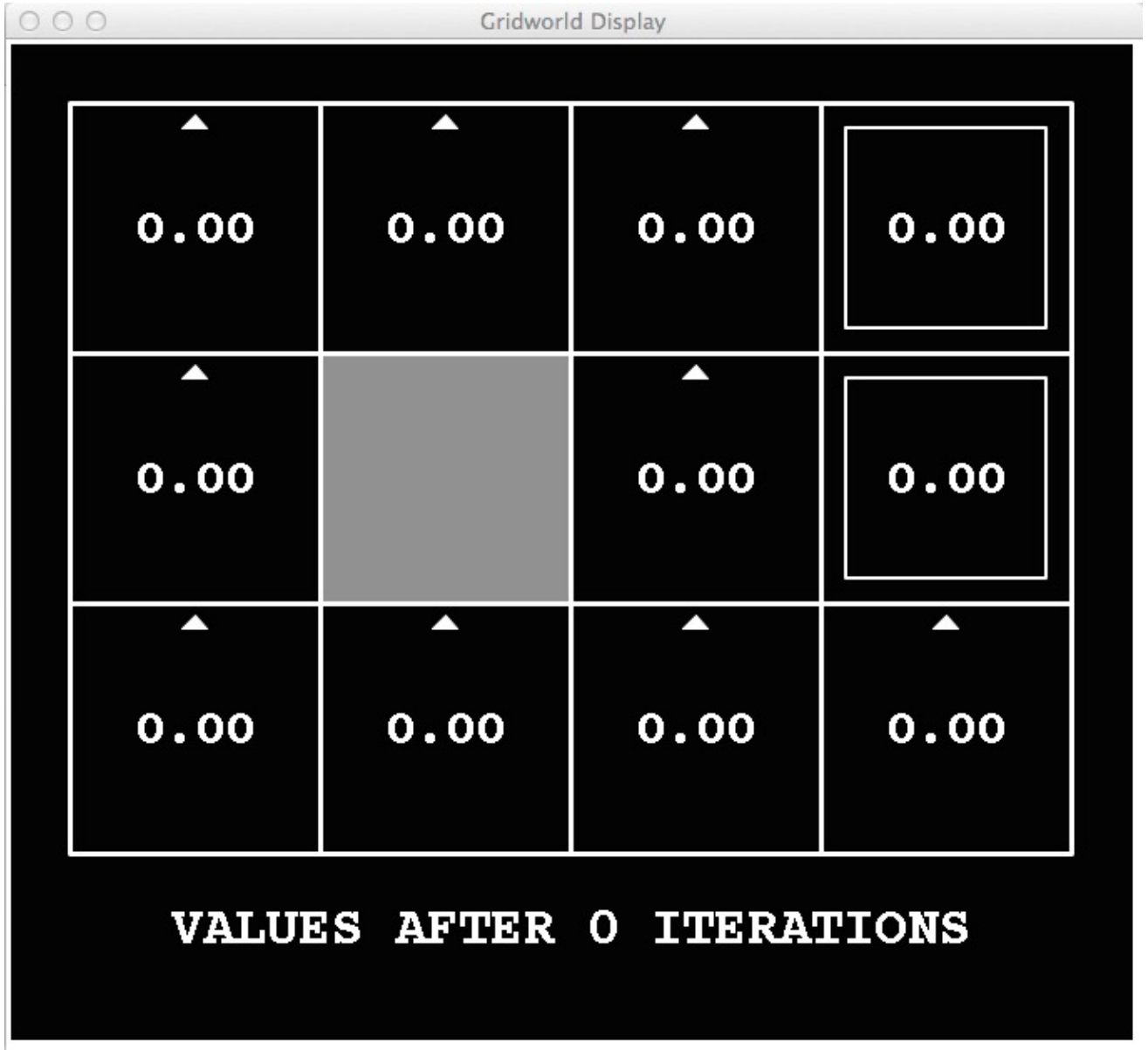
- Value iteration repeats the Bellman updates:

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



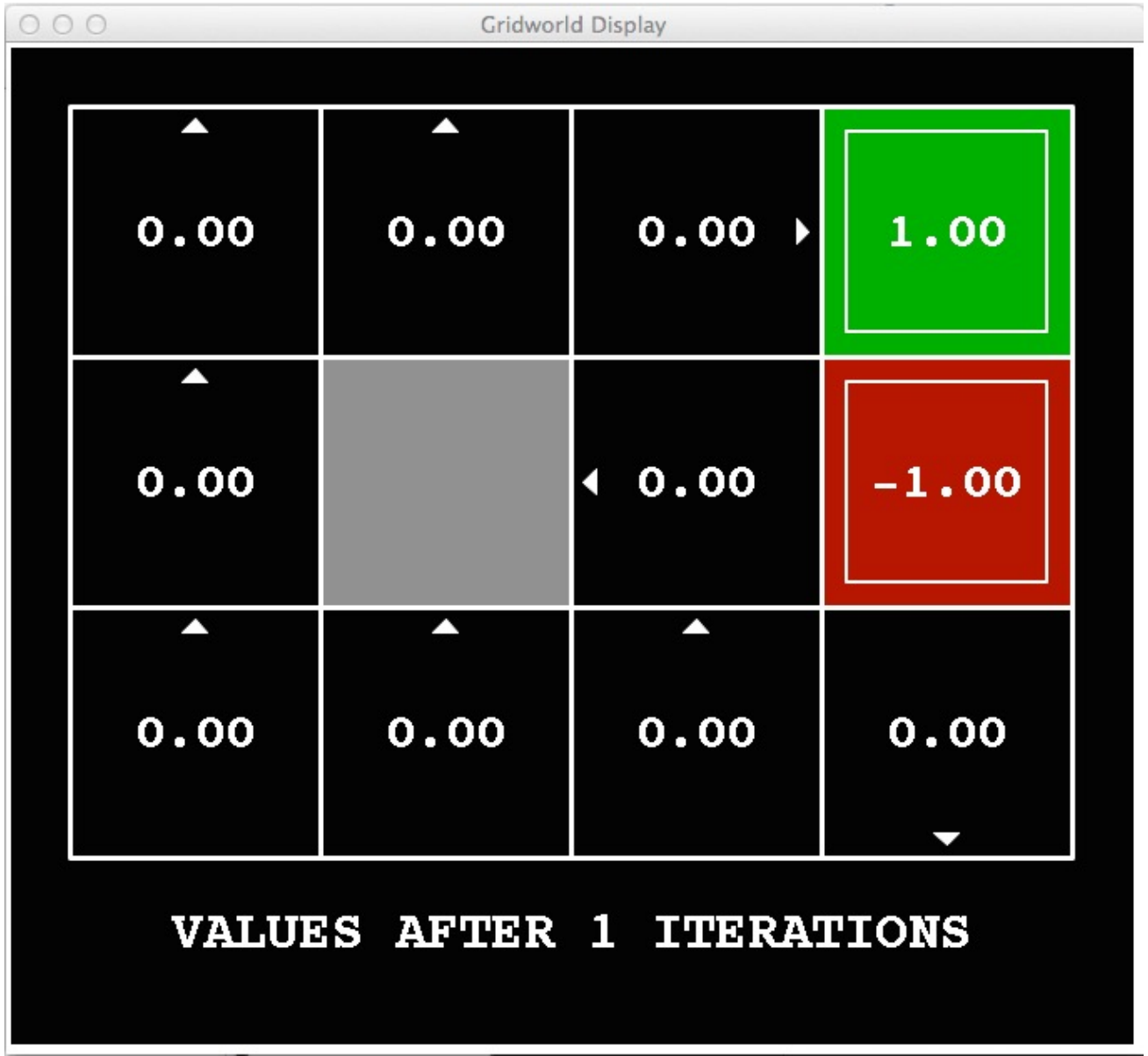
- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

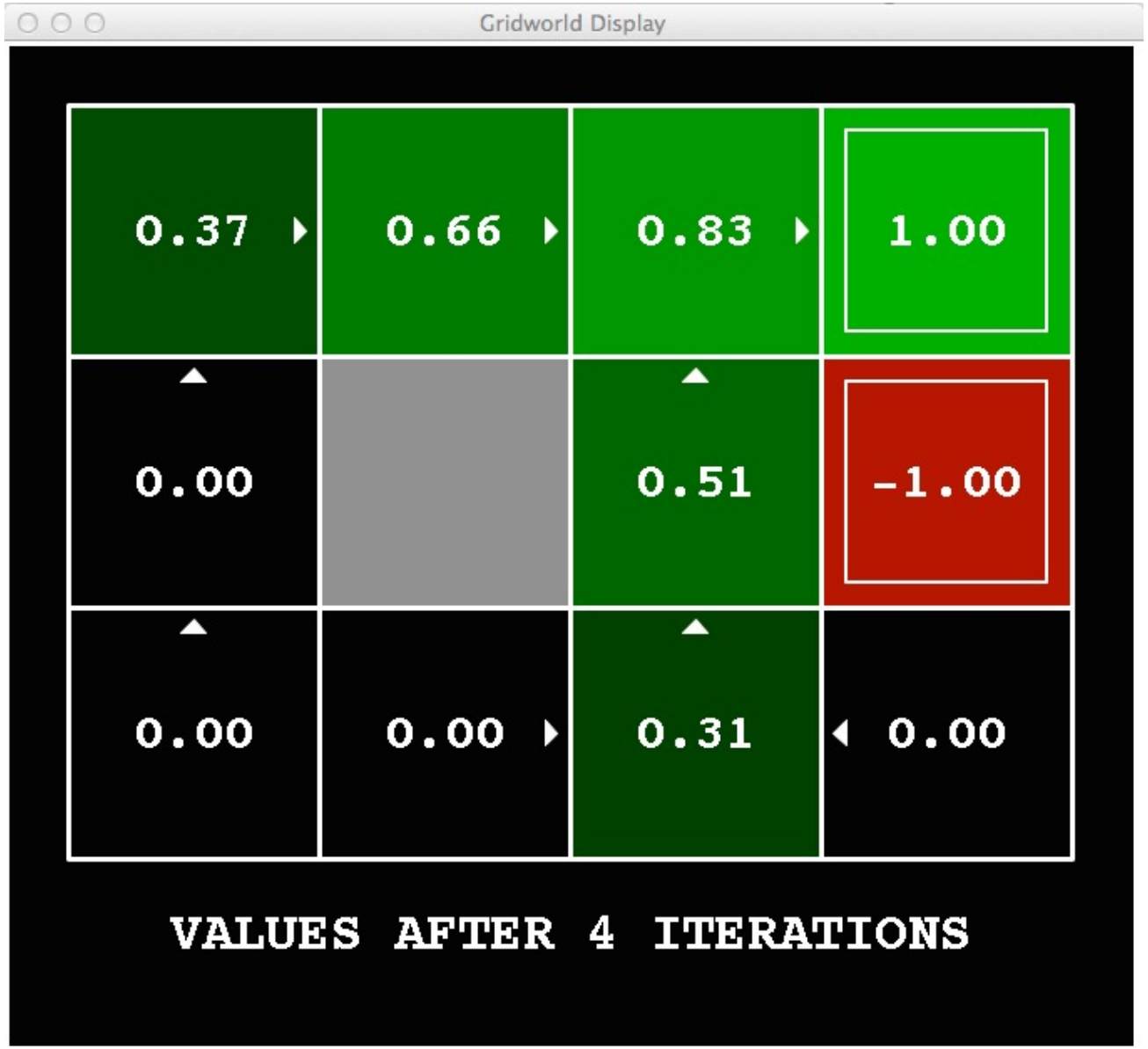
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

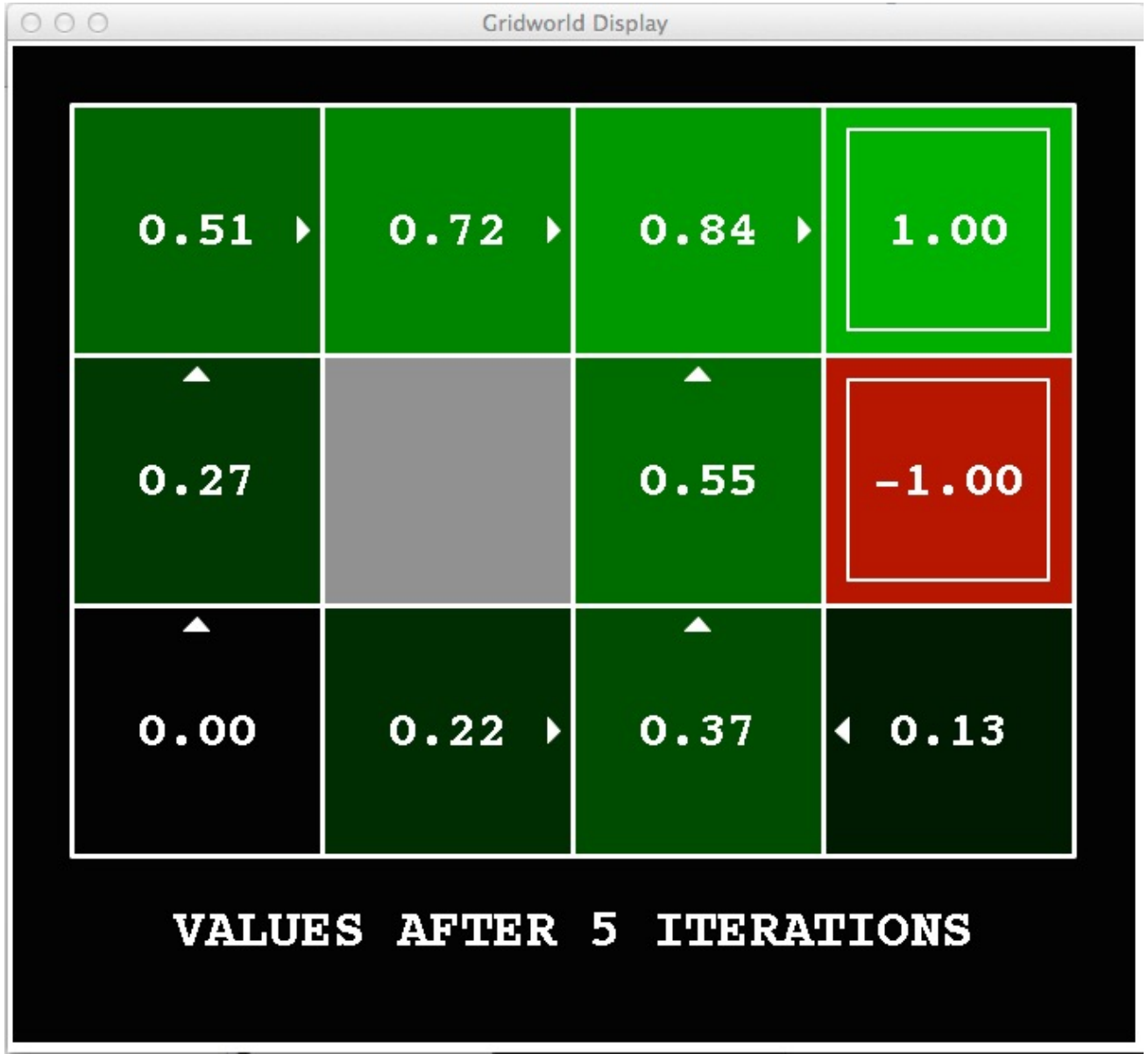
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



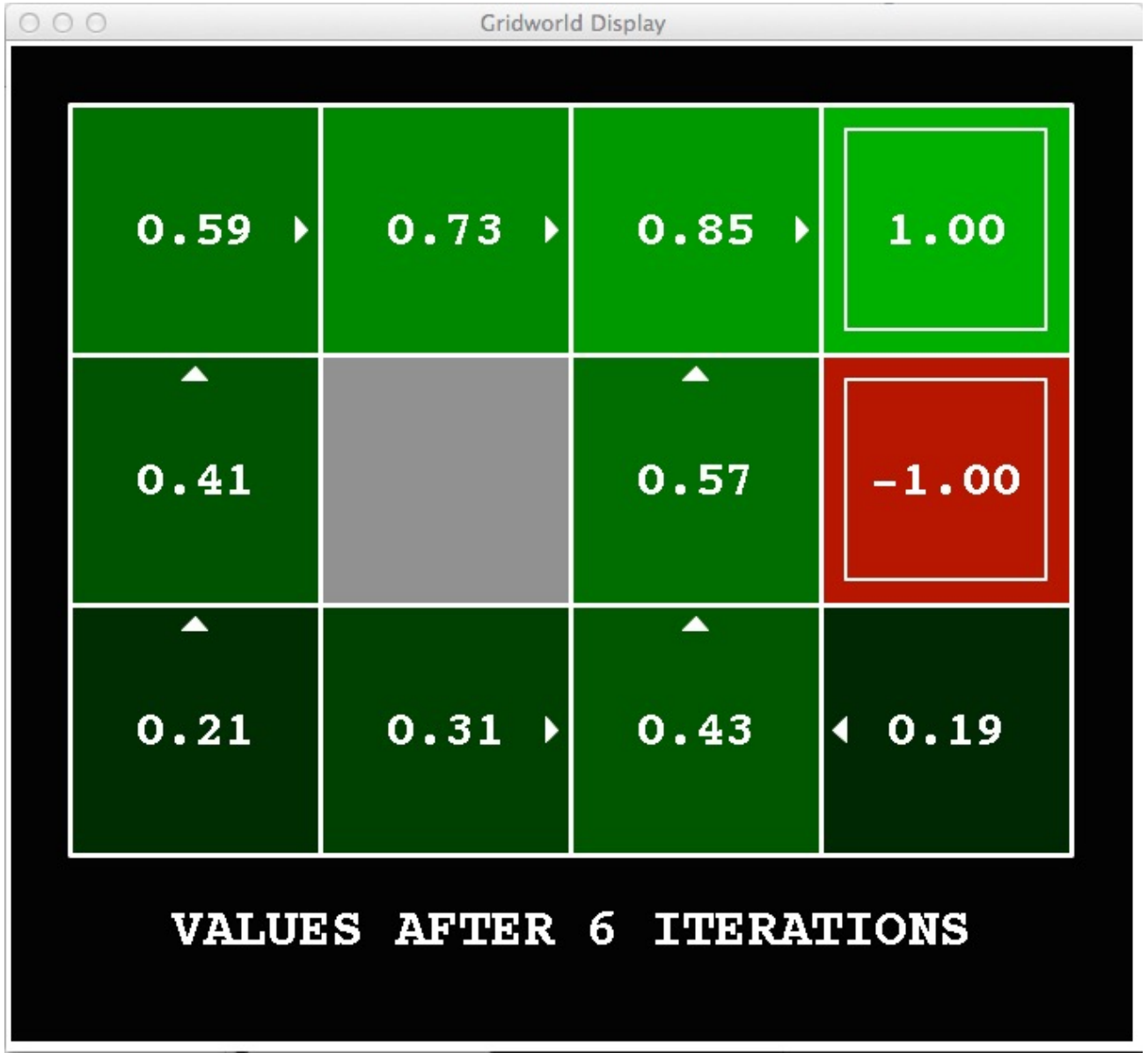
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



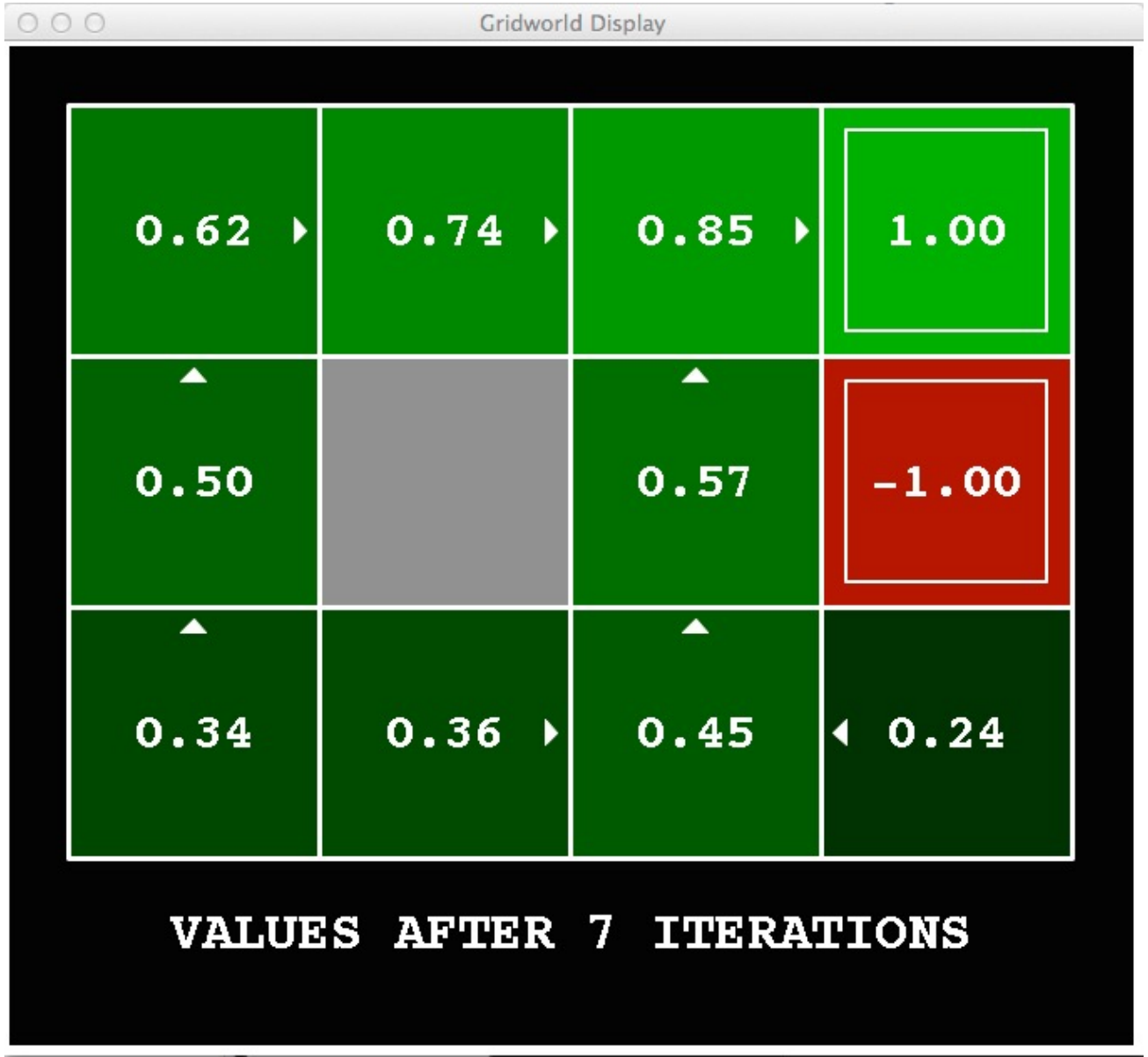
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



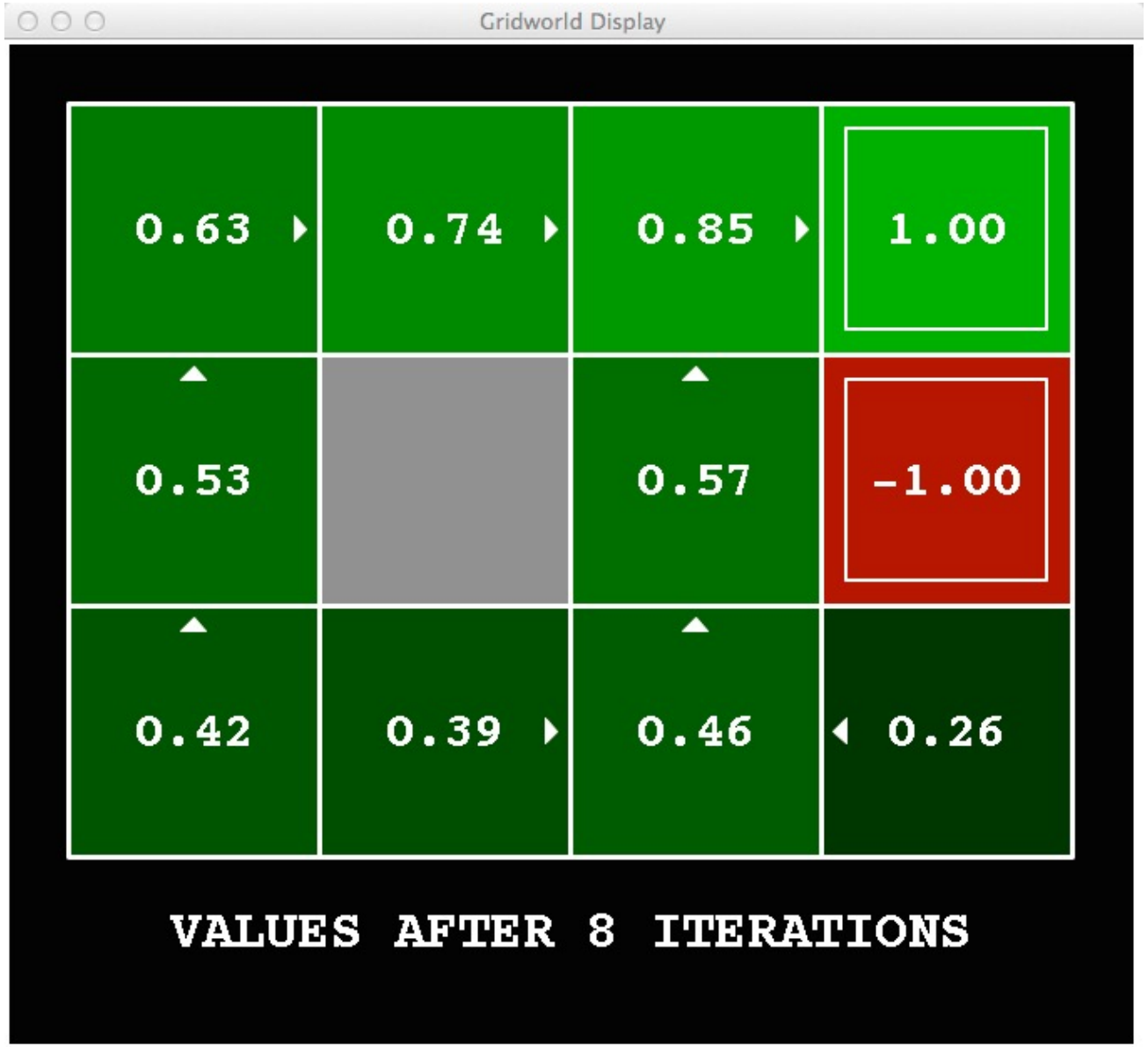
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



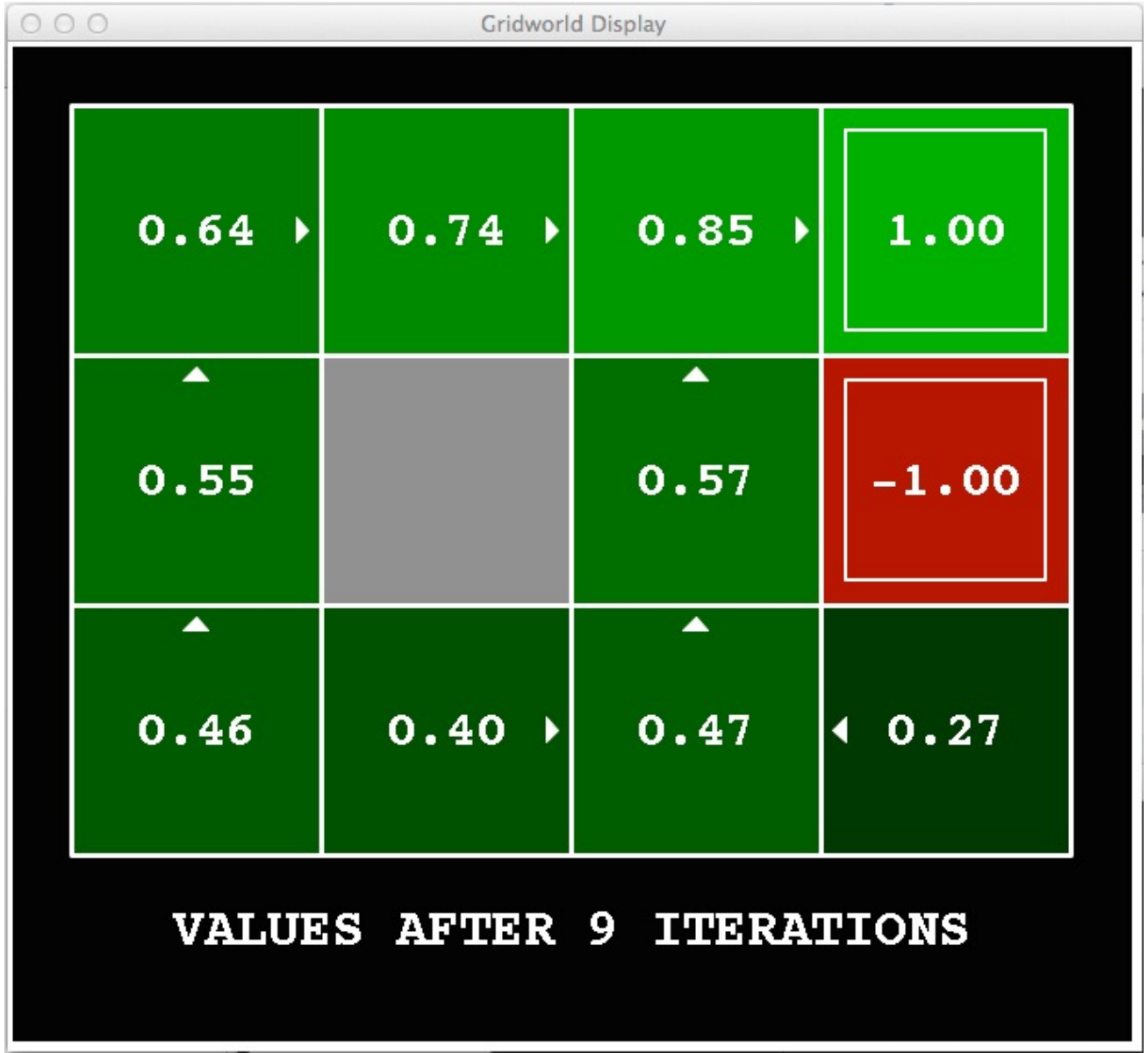
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



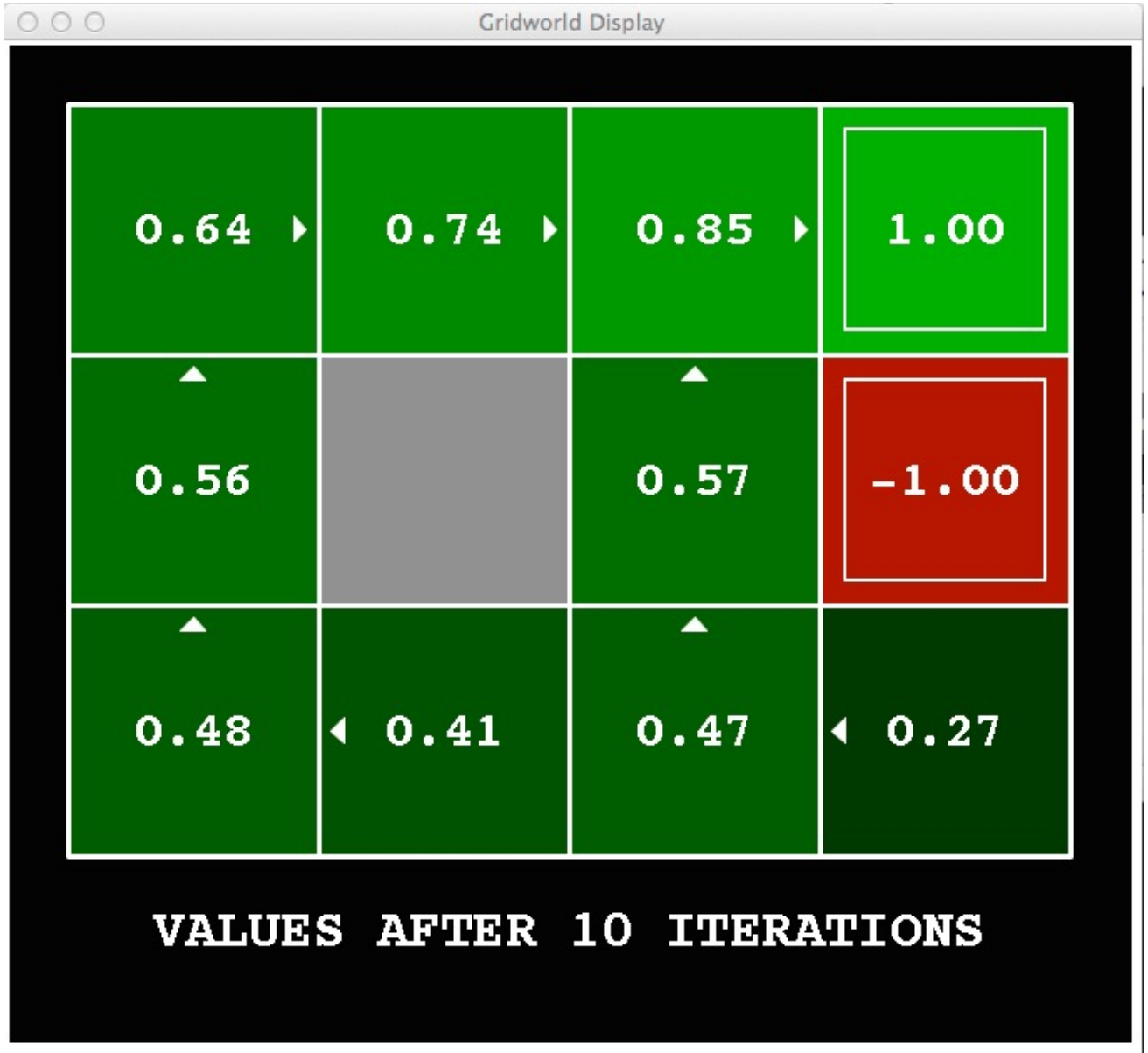
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



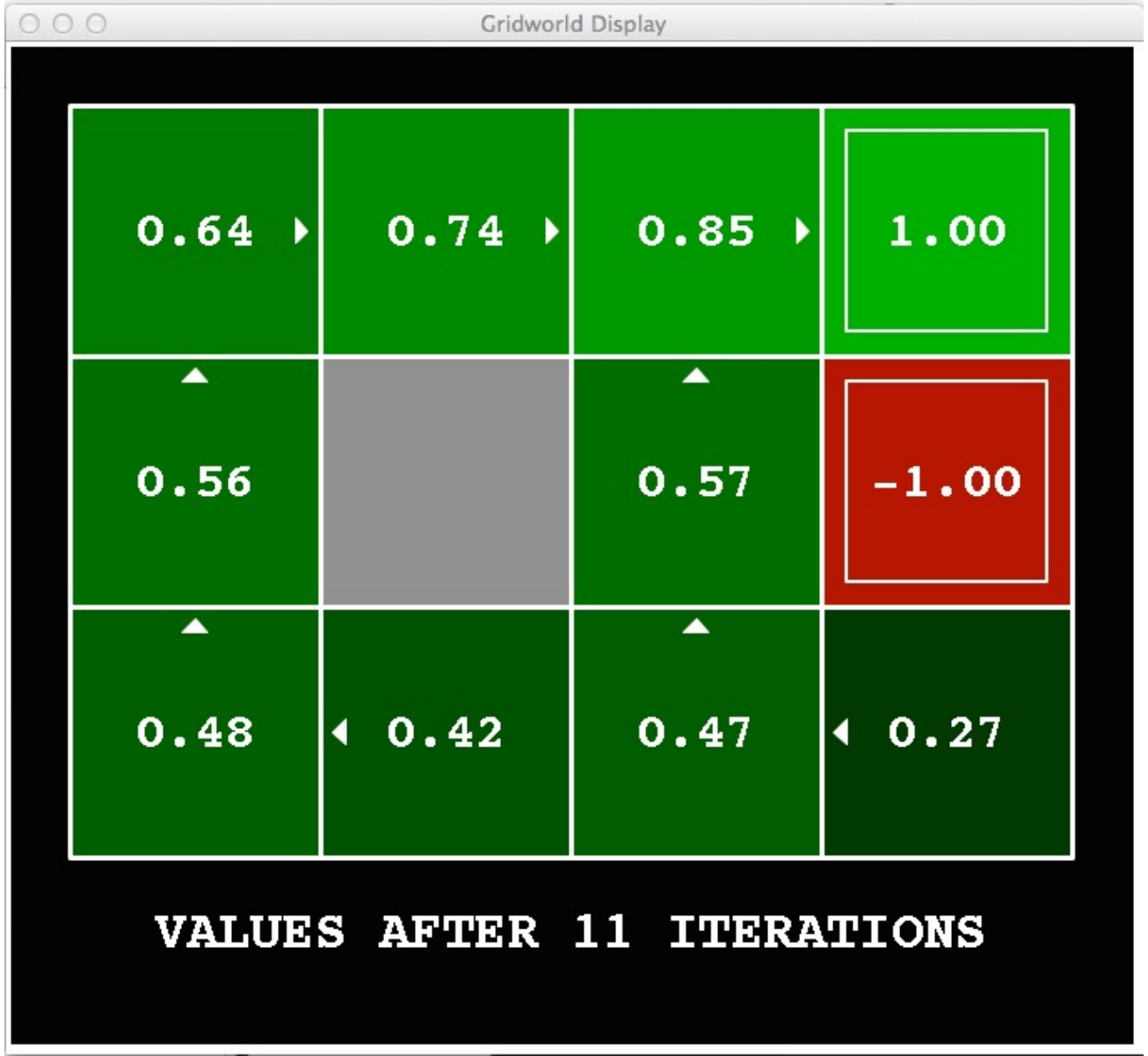
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



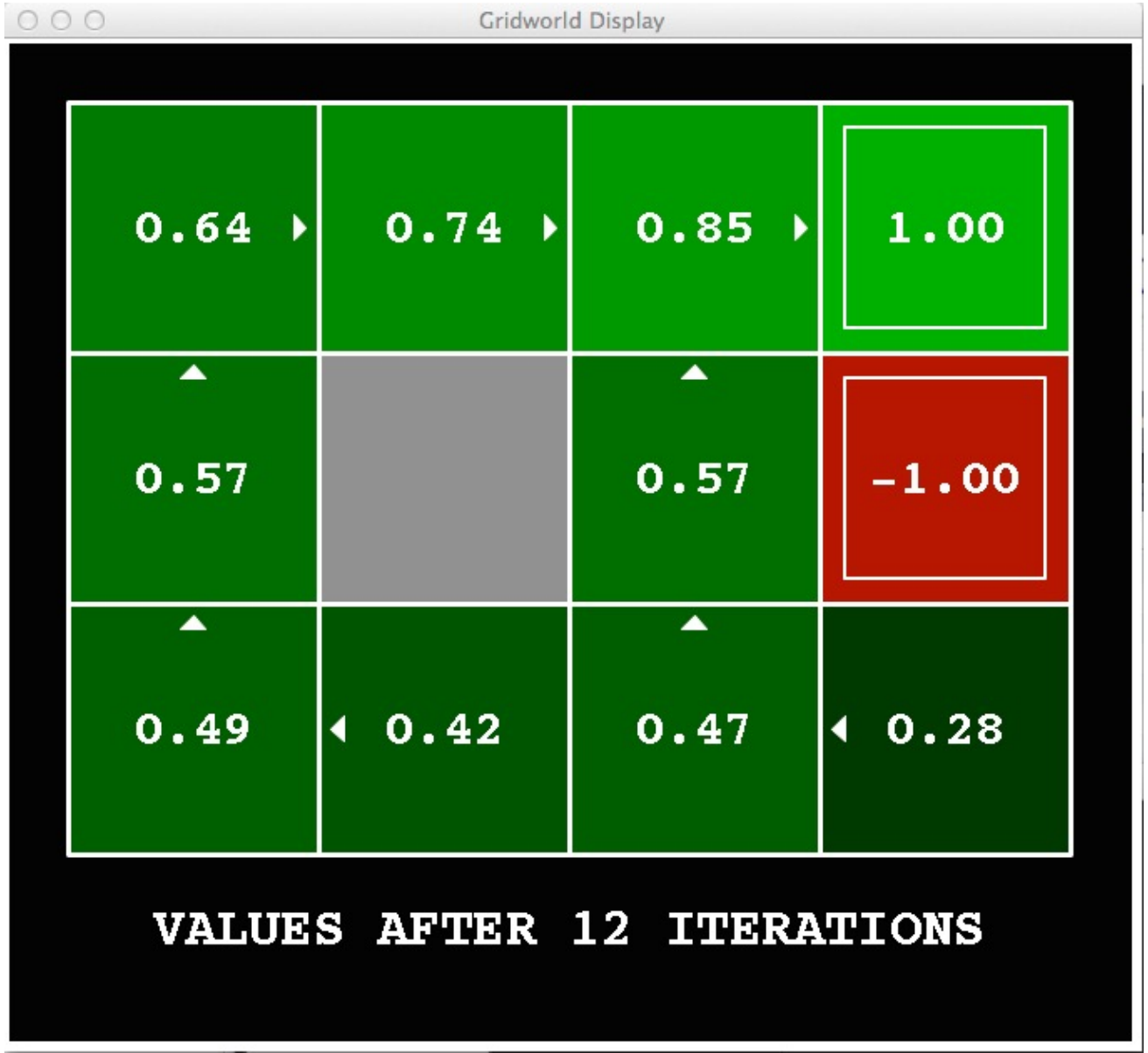
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



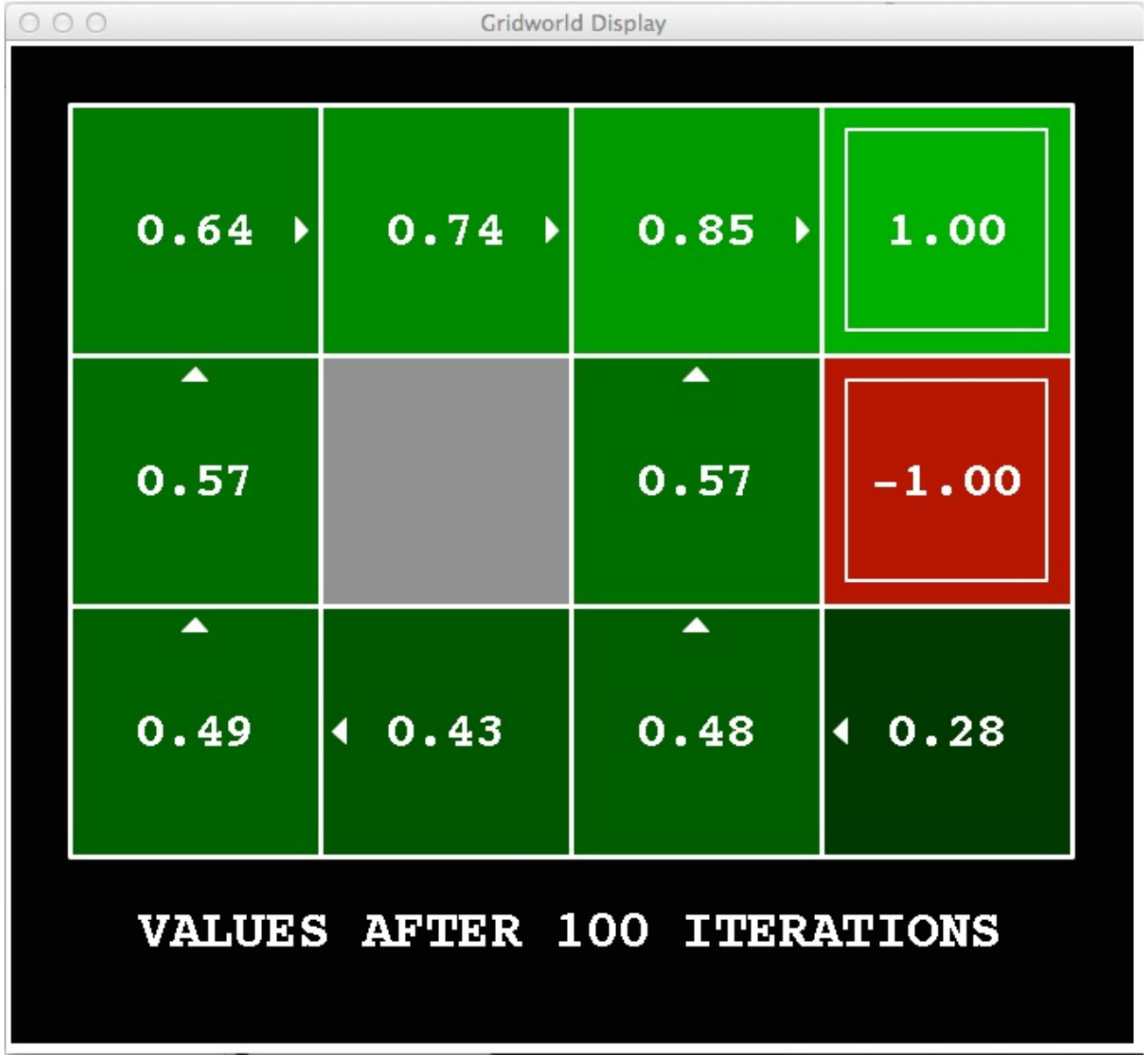
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

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 Iteration

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

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

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

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

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

MCTS?

- Recap: MCTS for games:

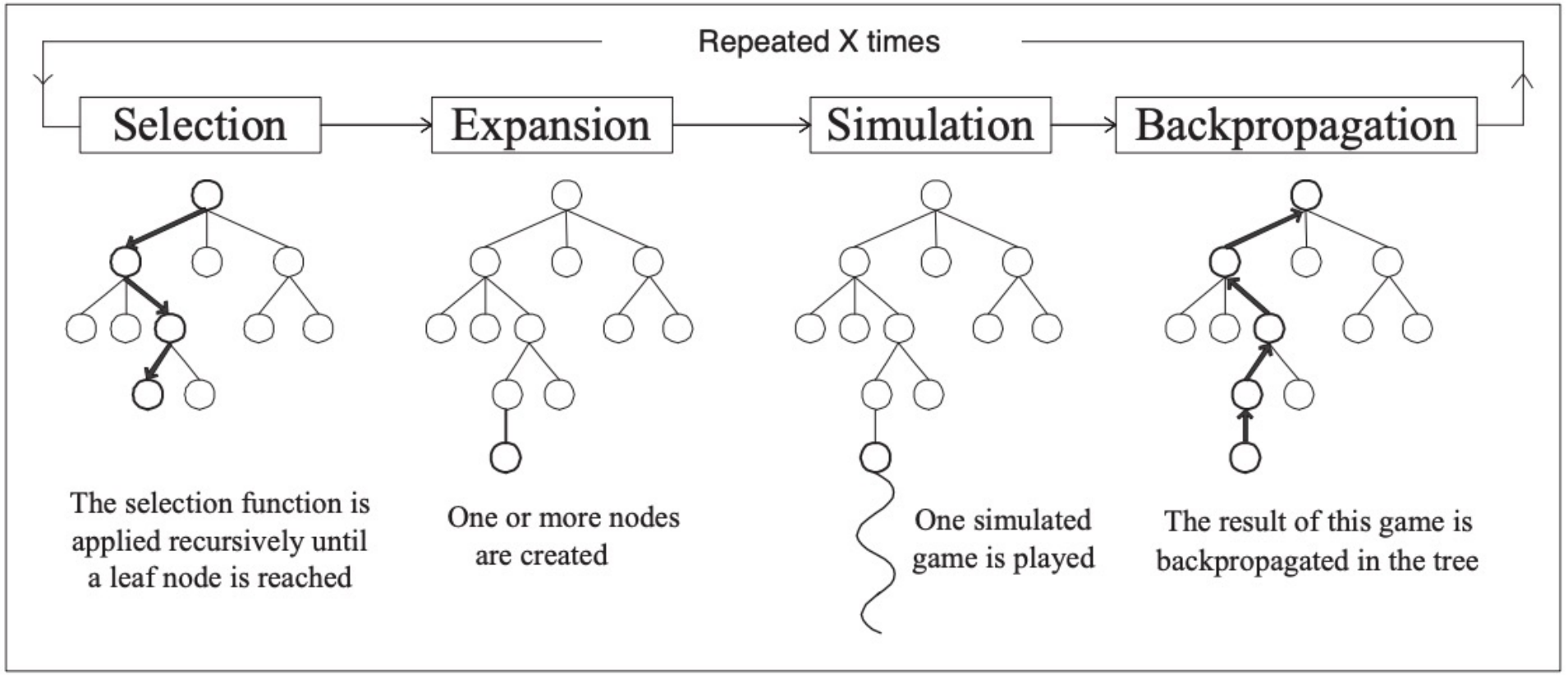


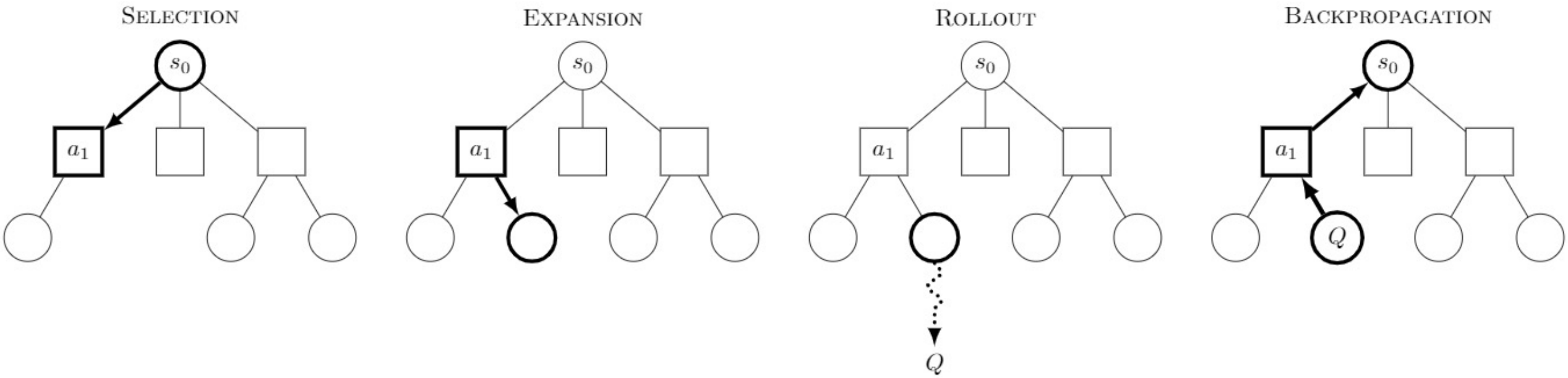
Image credit: Chaslot et al.

MCTS?

- Recap: MCTS (UCT) for games:
  - Selection: starting from the root, traverse down the tree based on UCB1 heuristics
  - Expansion: once reaching a leaf node, create a new child node for it
  - Simulation (rollout): from the new node, simulate the game until termination and return the outcome
  - Backpropagation (backup): traverse back to the root, updating utilities and visitation counts along the way
  - Repeat the above 4 steps until convergence or exhausting the search time

MCTS?

- MCTS (UCT) for MDPs:



UCB1 or not?

Leaf node?

Till termination?

Accumulate winning counts?

Image credit: Robert Moss

MCTS?

- MCTS (UCT) for MDPs:
  - Selection: starting from the root, traverse down the tree based on UCB1 heuristics
    - Under a V-node, use UCB1 to select action branches
    - Under a Q-node, sample the next state according to the transition probabilities
  - Expansion: once reaching a leaf node, create a new child node for it
    - Under a V-node, if not all actions that been tried, expand a new Q node for a new action
    - Under a Q-node, if the sampled next state is not a children yet, add a new V node for it.
  - Rollout: from the new node, simulate the world until termination and return the total discounted reward
  - Backup: traverse back to the root, updating utilities and visitation counts along the way
  - Repeat the above 4 steps until convergence or exhausting the search time

MCTS?

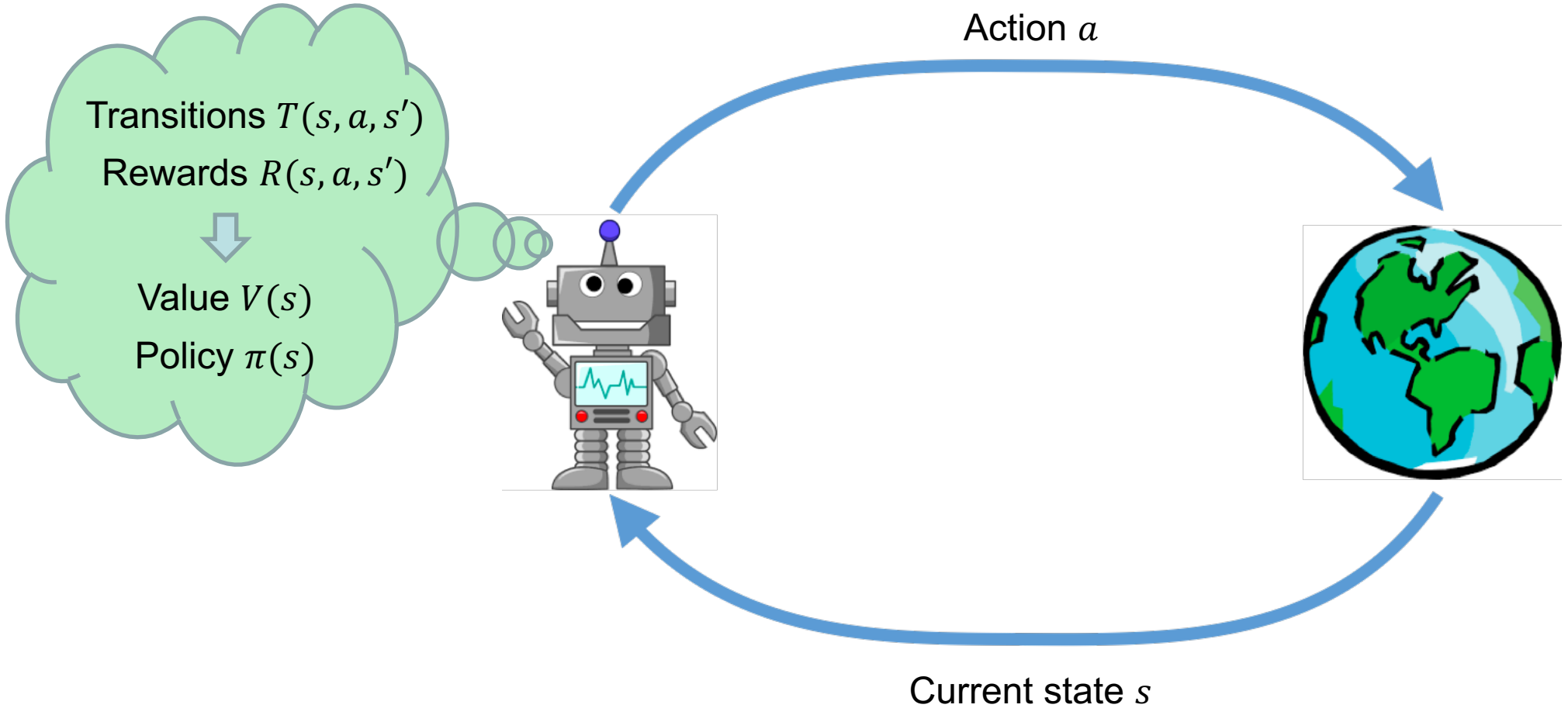
- MCTS (UCT) for MDPs:
  - Selection: starting from the root, traverse down the tree based on UCB1 heuristics
  - Expansion: once reaching a leaf node, create a new child node for it
  - Rollout: from the new node, simulate the world until termination and return the total discounted reward
    - Simulate the robot using a rollout policy, and the world dynamics using the transition function
    - Simulate until reaching terminal state or exhausting the horizon of the problem
      - number of simulation steps from the root to the end of rollouts $\leq$ the remaining horizon T
  - Backup: traverse back to the root, updating utilities and visitation counts along the way
    - Utilities = total discounted rewards
    - Accumulate reward and discount at each step
  - Repeat the above 4 steps until convergence or exhausting the search time

Summary II: MCTS vs. Value/Policy Iteration

- Value / Policy Iteration
  - Offline planning: precompute actions for all states before execution
  - Suitable for discrete and low-dimensional state-spaces (typically $d \leq 3$)
  - Otherwise need to operate on high-dimensional tables
    - Can take a long-time to finish
- MCTS
  - Online planning: replan optimal action for each current state
  - Suitable for larger state-spaces (even continuous ones with some trick)
  - Approximate solution, but works well in practice
  - Anytime algorithm!
    - Can adapt to different search time limits

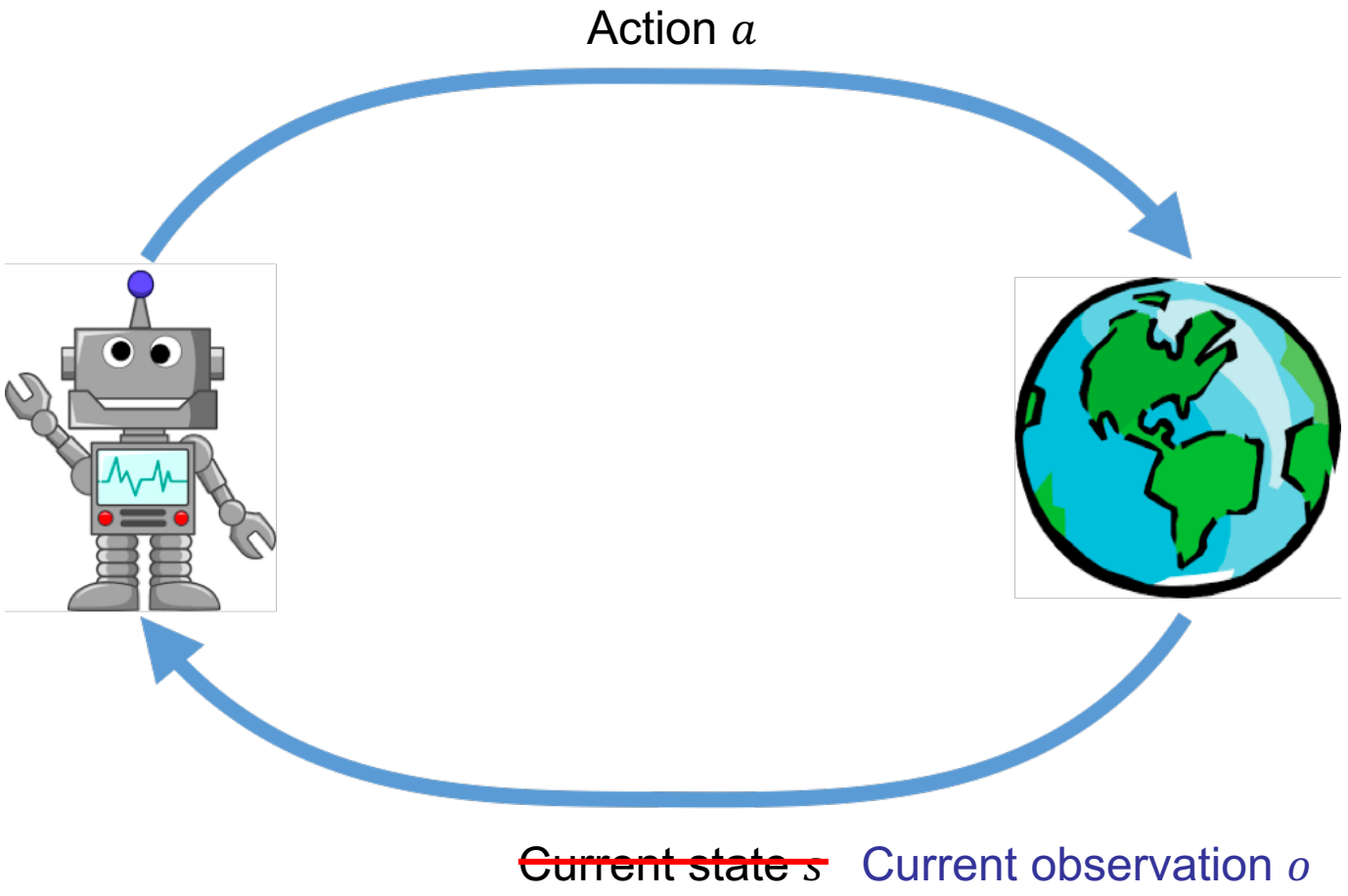
Fully Observable MDPs

- MDP: I can directly observe the state, fully, precisely, without noise



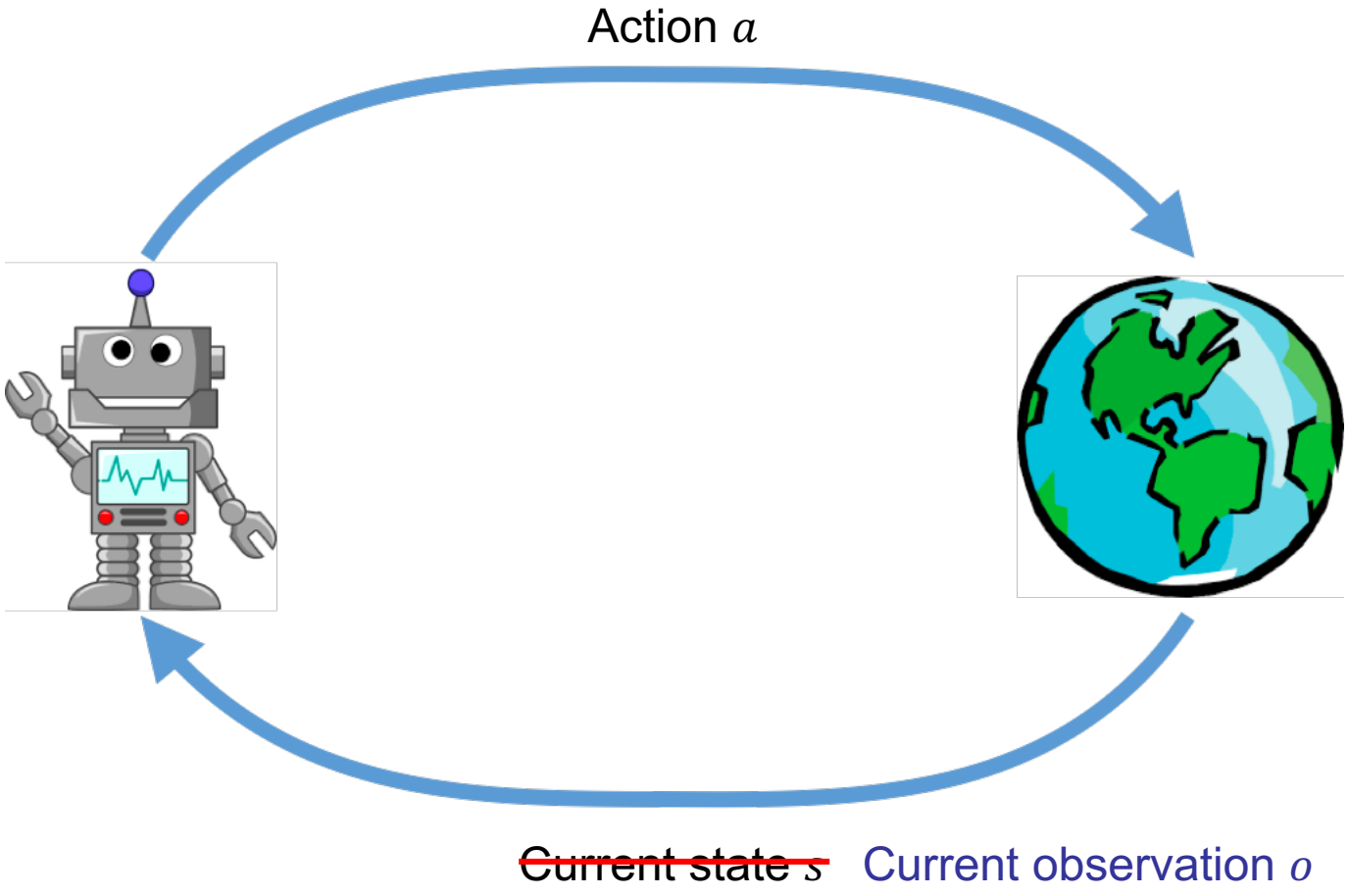
Partially Observable MDPs (POMDPs)

- POMDP: I can **not** directly observe the state…

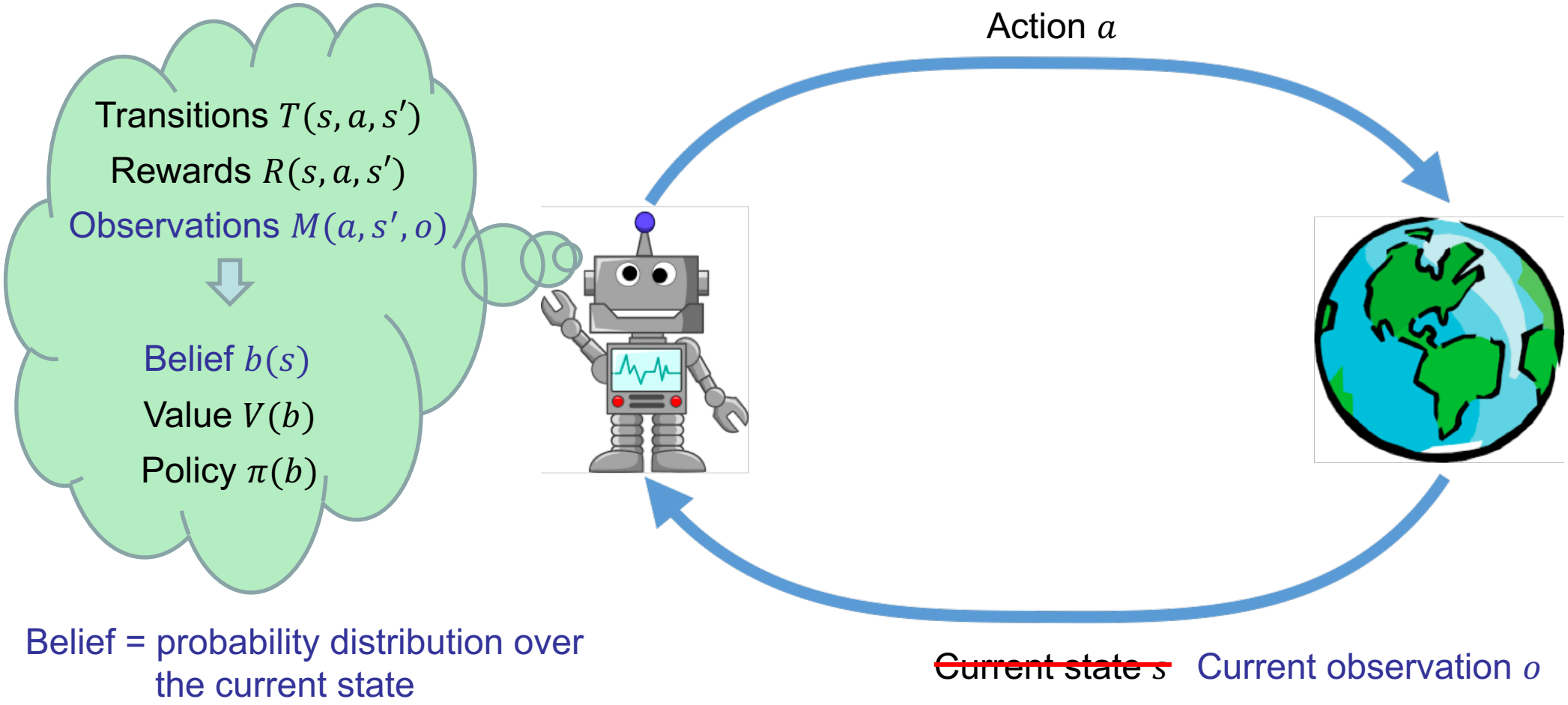


Partially Observable MDPs (POMDPs)

- Observations:
  - Indirectly reflect the state
    - camera images
    - Lidar data
    - …
  - Only has partial information
    - occlusions
    - human factors
    - …
  - Often gives noisy information
    - object detector noise
    - object detector error
    - …

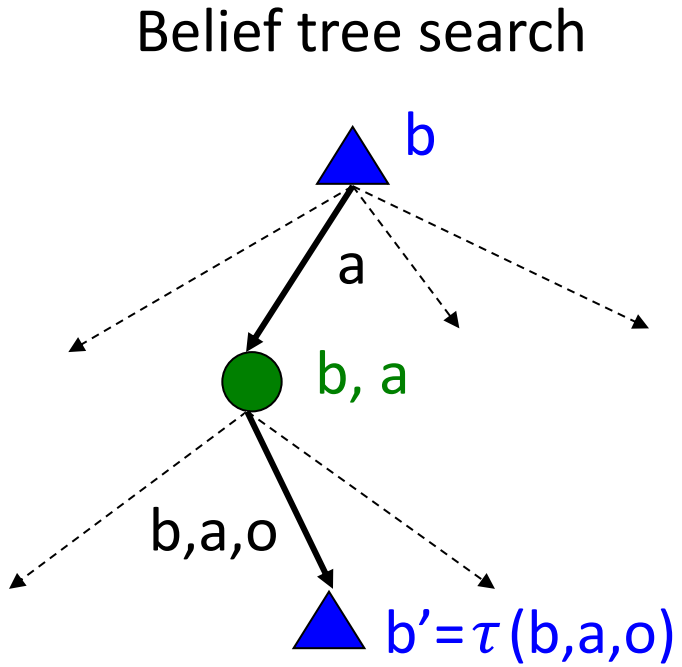


Partially Observable MDPs (POMDPs)



POMDP Formalization

- Partially observable Markov decision processes:
  - States S
  - Actions A
  - Observations O
  - Transition function T(s,a,s') = P(s'|s,a)
  - Observation function M(a,s',z) = P(z|a,s')
  - Reward function R(s,a,s') (and discount γ)
  - Start belief b_0

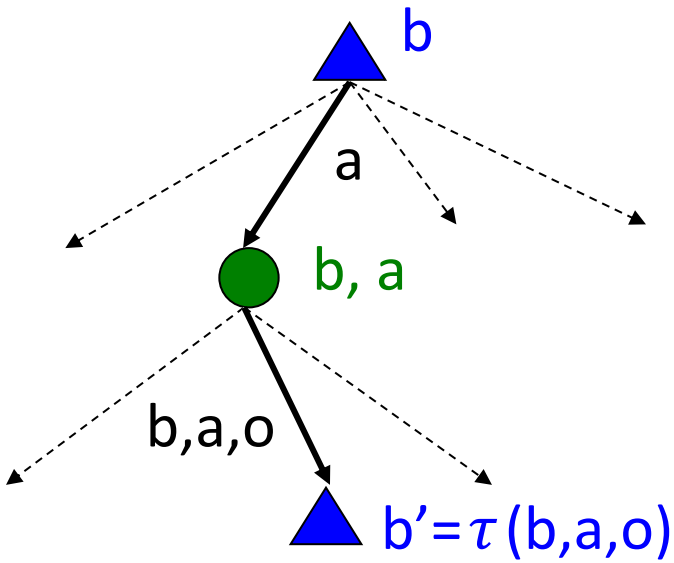


- Quantities:
  - Belief = probability distribution over states
  - Policy = map of beliefs to actions
  - Utility = sum of discounted rewards
  - Values = expected future utility from a belief (max node)
  - Q-Values = expected future utility from a q-state (chance node)

Solving POMDPs

- Much harder to solve than MDPs
  - Search space size:
    - dimension of belief space = size of the state space
  - Need to update belief at each node
    - b'=τ(b,a,z)
    - each node requires significant computation!

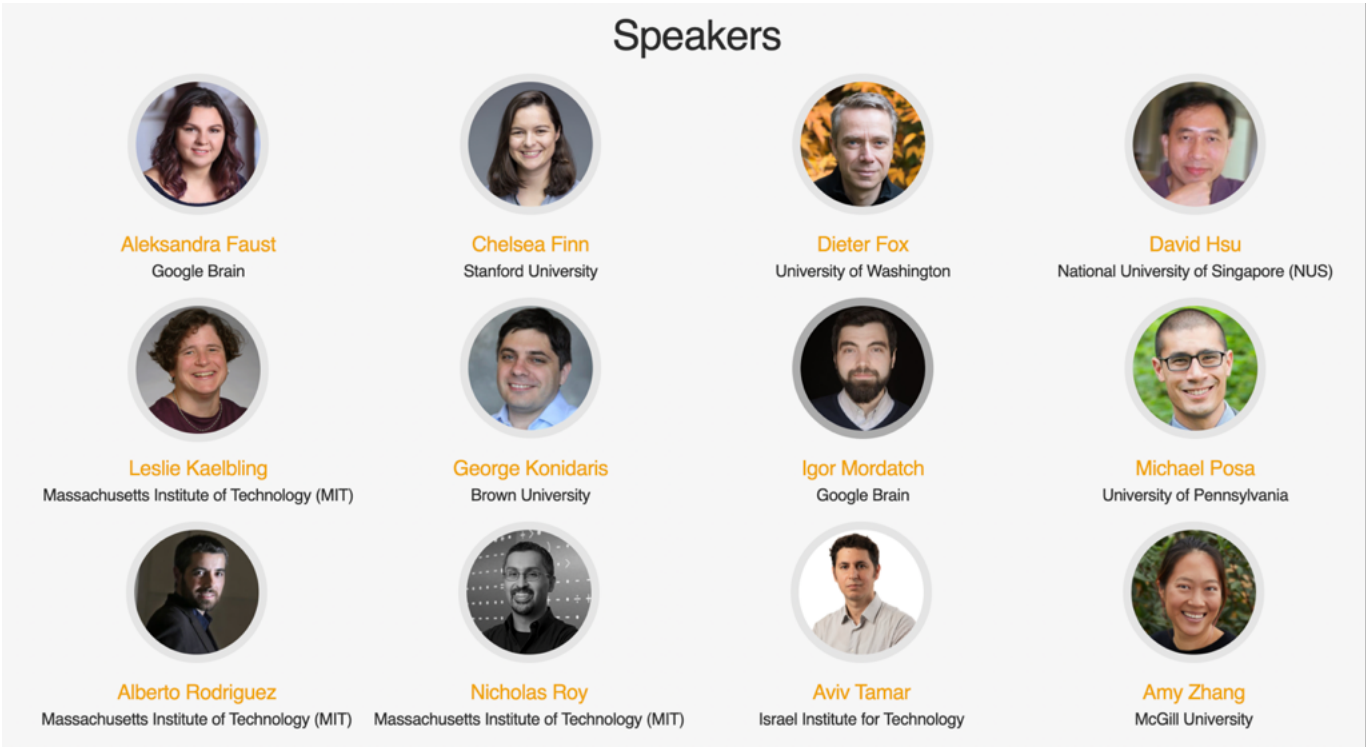
Belief tree search

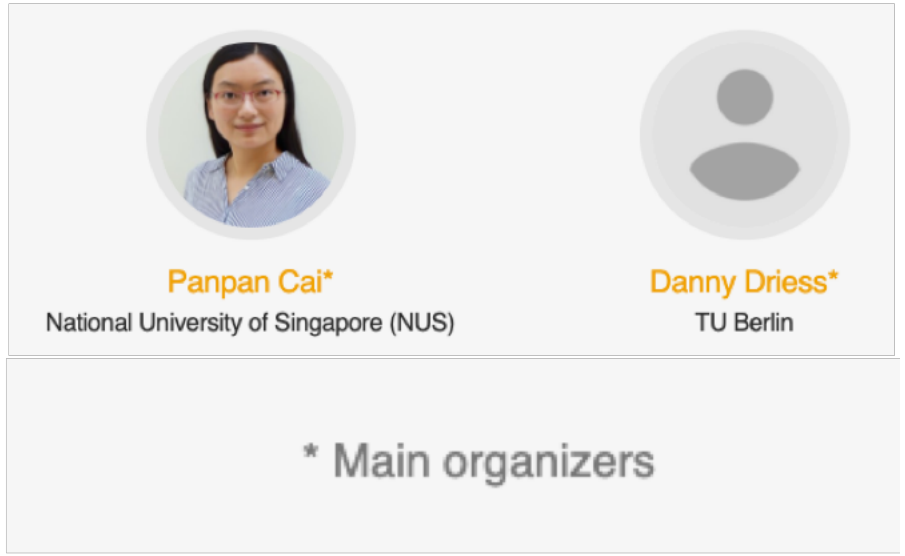


- Solutions
  - Value / policy iteration not dot work anymore!
  - State-of-the-art algorithms are based on:
    - Belief tree search
    - Monte Carlo sampling (from MCTS)
    - Good heuristics (such as rollouts!)
    - Pruning: branch-and-bound (DESPOT)
    - Help from learning (learn heuristics, sampling distributions, macro-actions…)

Integrating Planning and Learning

- One important direction is to use leaning to help planning
- A new and active research area!
- The first workshop on integrating planning and learning held on 2021
  - Workshop website: https://planandlearn.net/





Next Time: Reinforcement Learning!