CIS 471/571(Fall 2020): Introduction to Artificial Intelligence

Lecture 9: MDPs (Part 2)

Thanh H. Nguyen

Source: http://ai.berkeley.edu/home.html

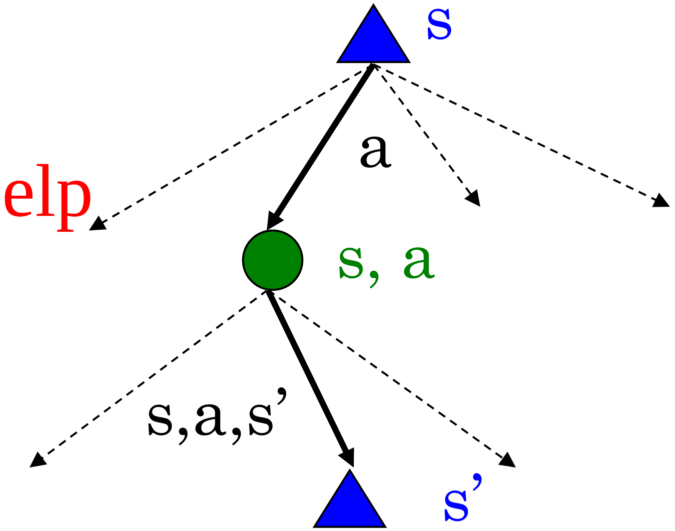
Announcement

- Project 3: Reinforcement Learning
  - Deadline: Nov 10th, 2020

- Homework 3: MDPs and Reinforcement Learning
  - Will be posted tomorrow
  - Deadline: Nov 09th, 2020

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:

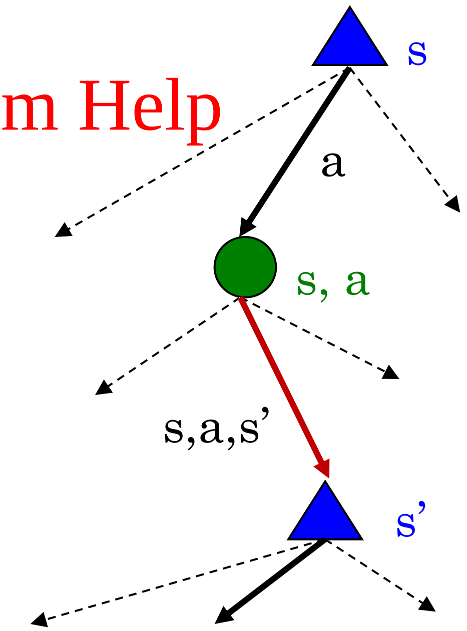
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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



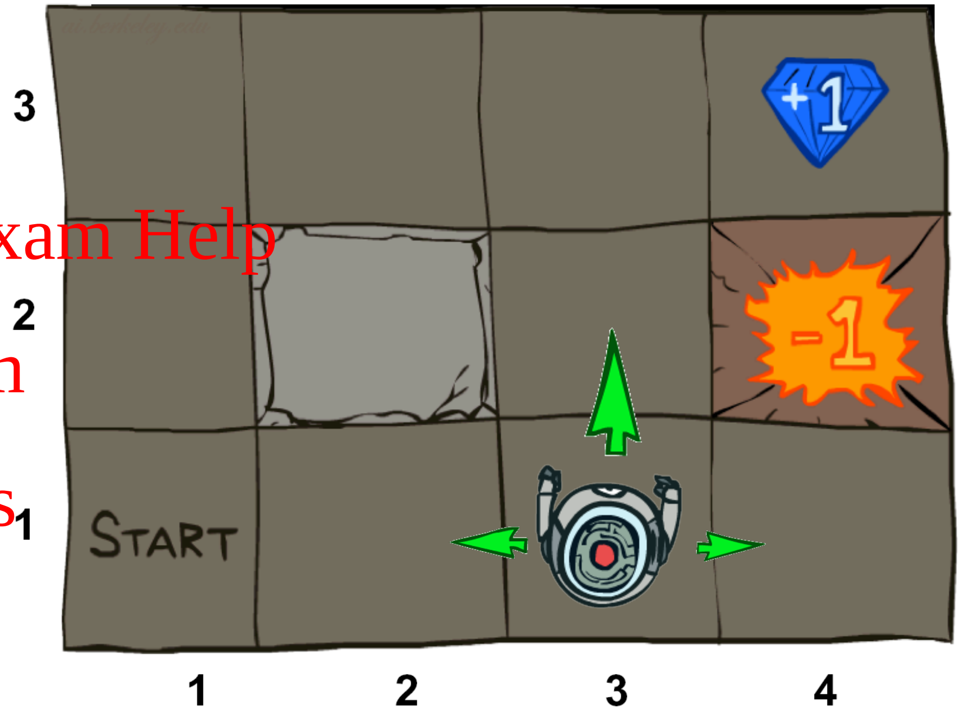
s is a *state*

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

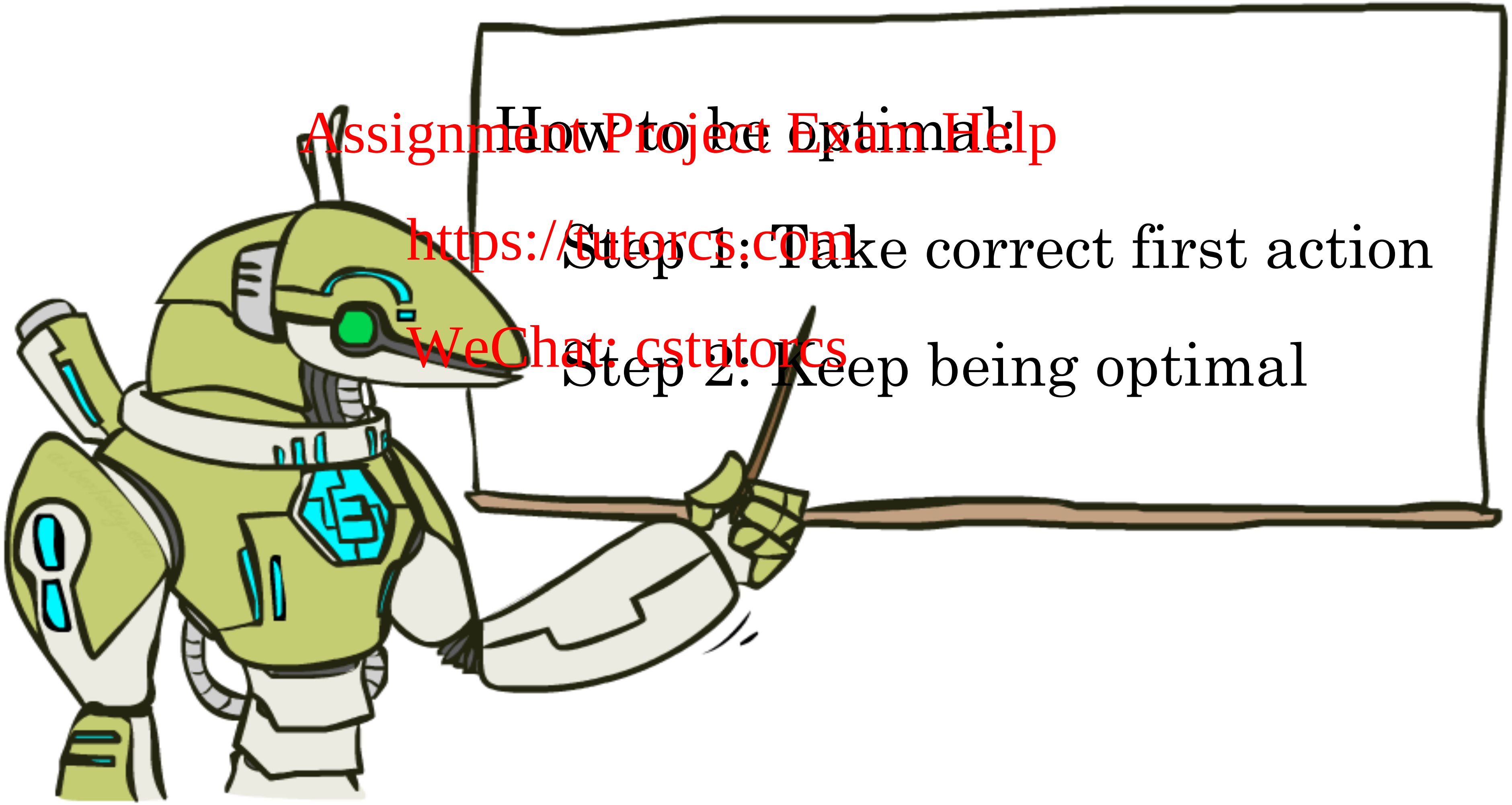
(s,a,s’) is a *transition*

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



The Bellman Equations



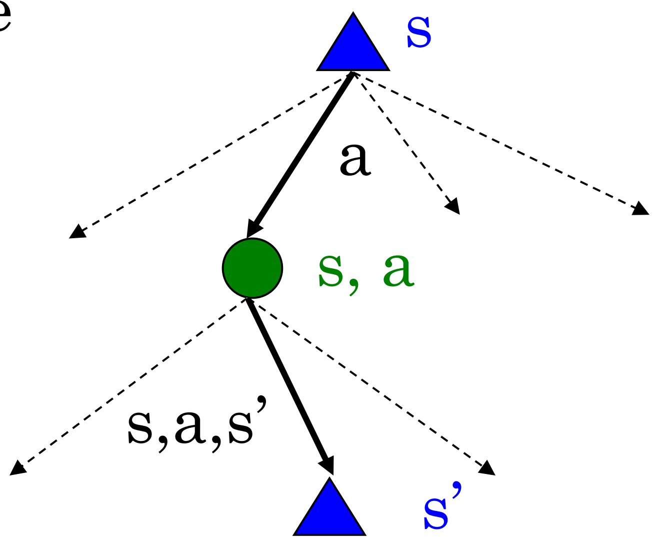
How to be optimal:

Step 1: Take correct first action

Step 2: Keep being optimal

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

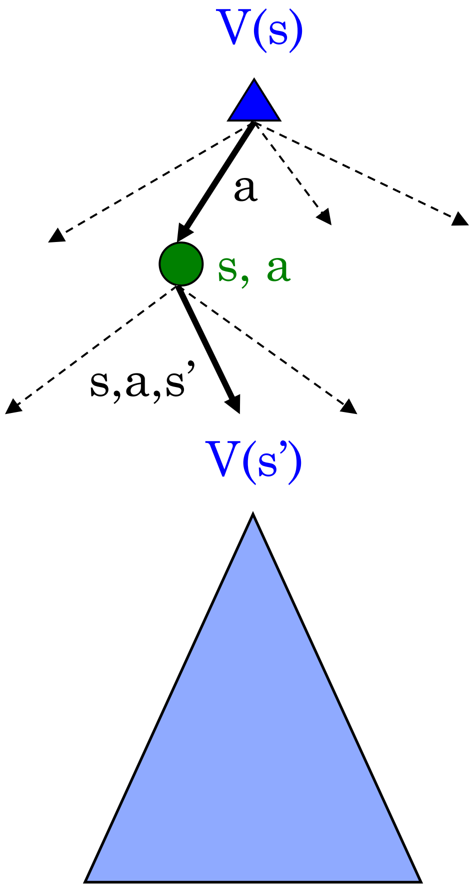
- Bellman equations characterize the optimal values:

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

- Value iteration computes them:

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

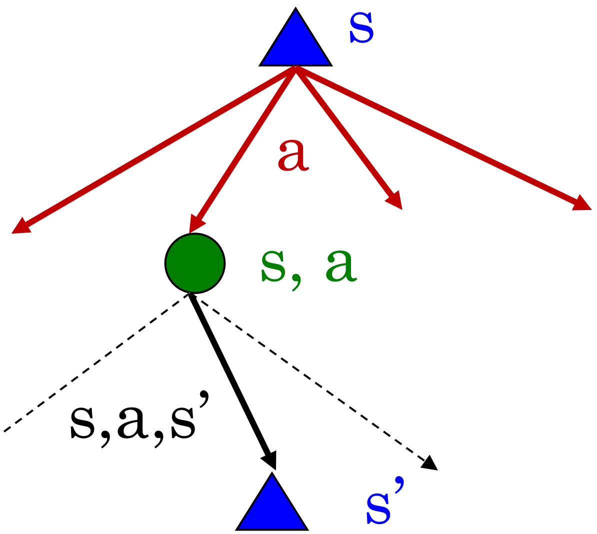
- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do



Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

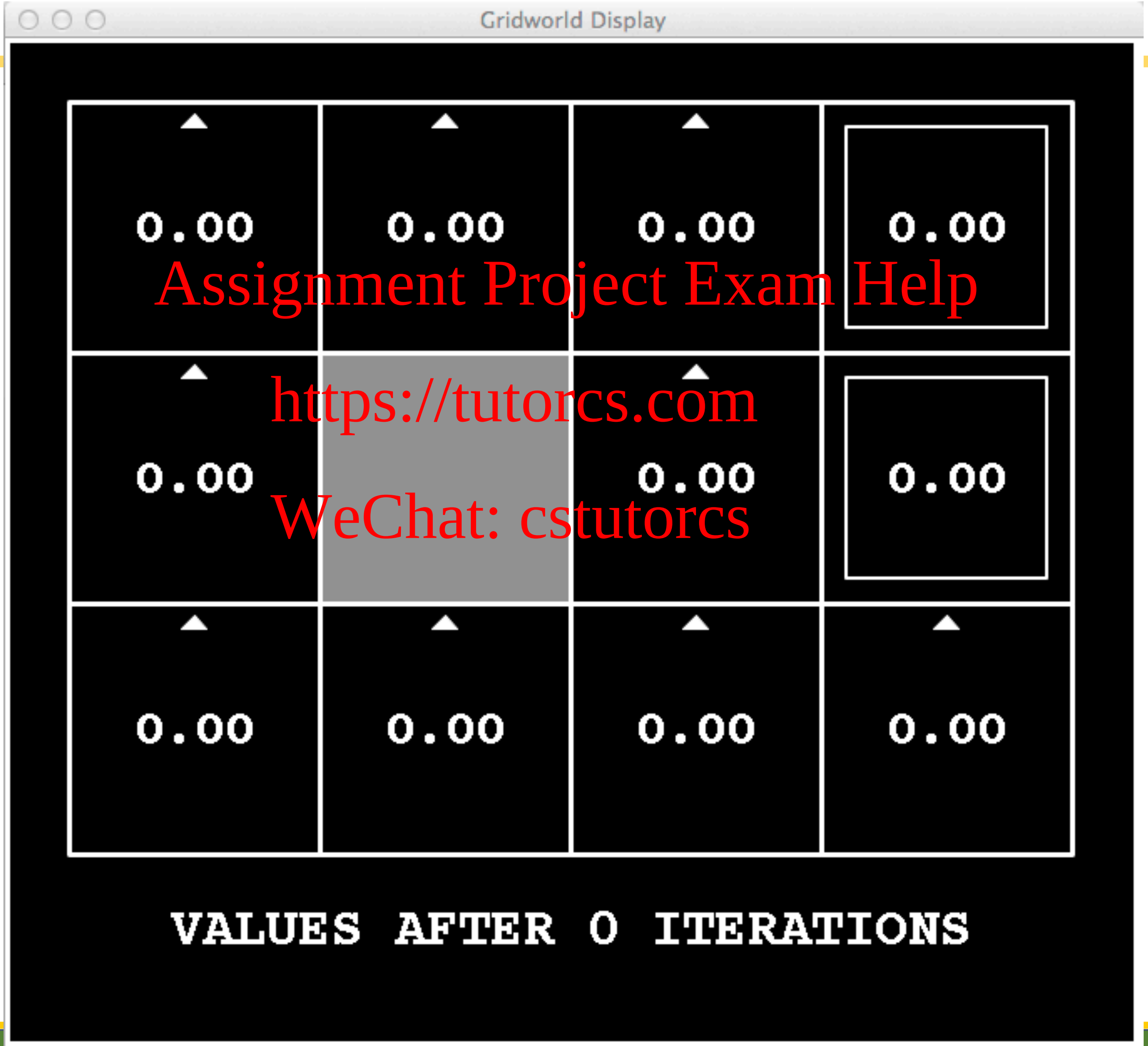


- Problem 1: It's slow – $O(S^2A)$ per iteration

- Problem 2: The "max" at each state rarely changes

- Problem 3: The policy often converges long before the values

k=0

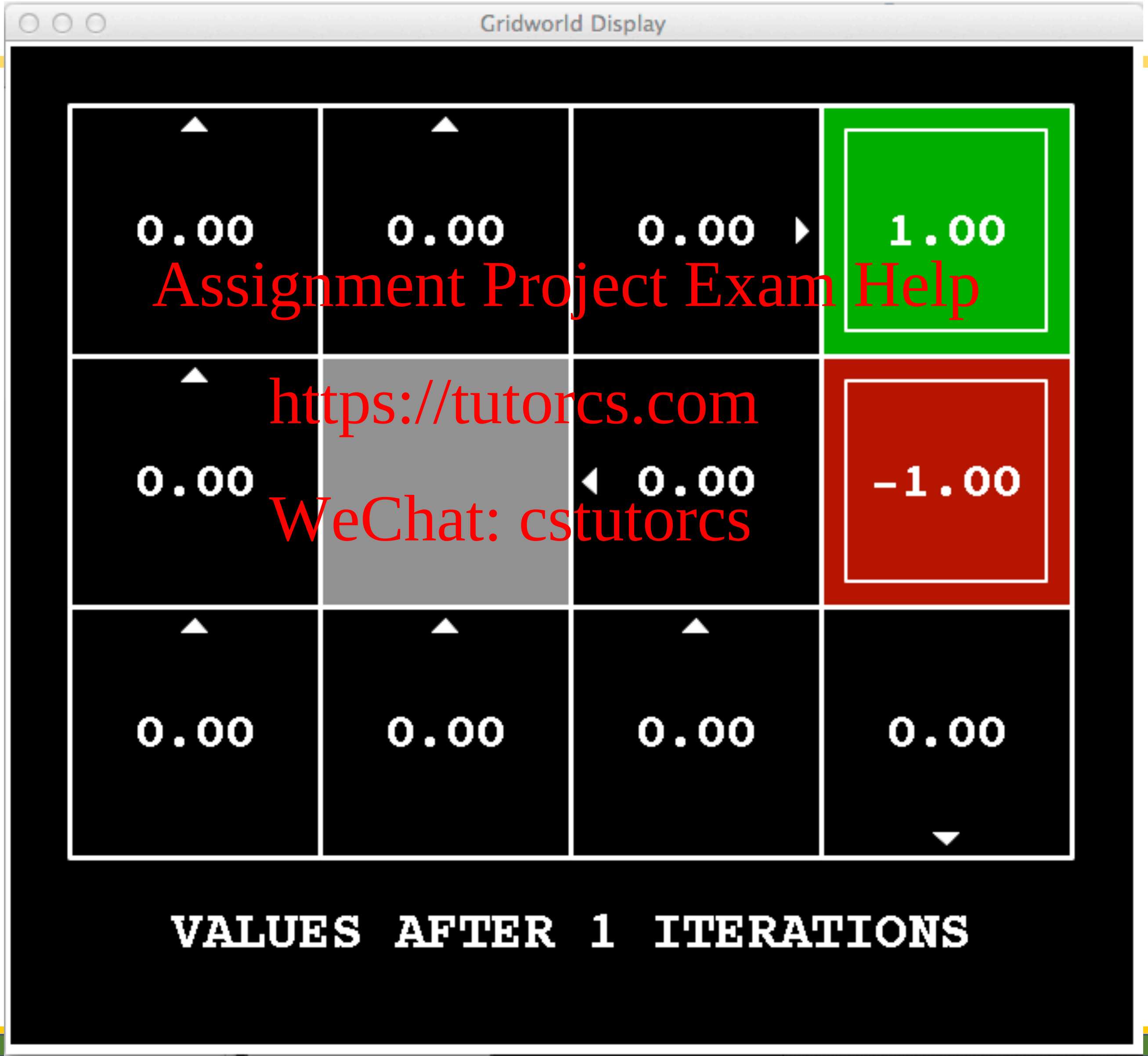


▲ 0.00	▲ 0.00	▲ 0.00	0.00
▲ 0.00		▲ 0.00	0.00
▲ 0.00	▲ 0.00	▲ 0.00	▲ 0.00

VALUES AFTER 0 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=1

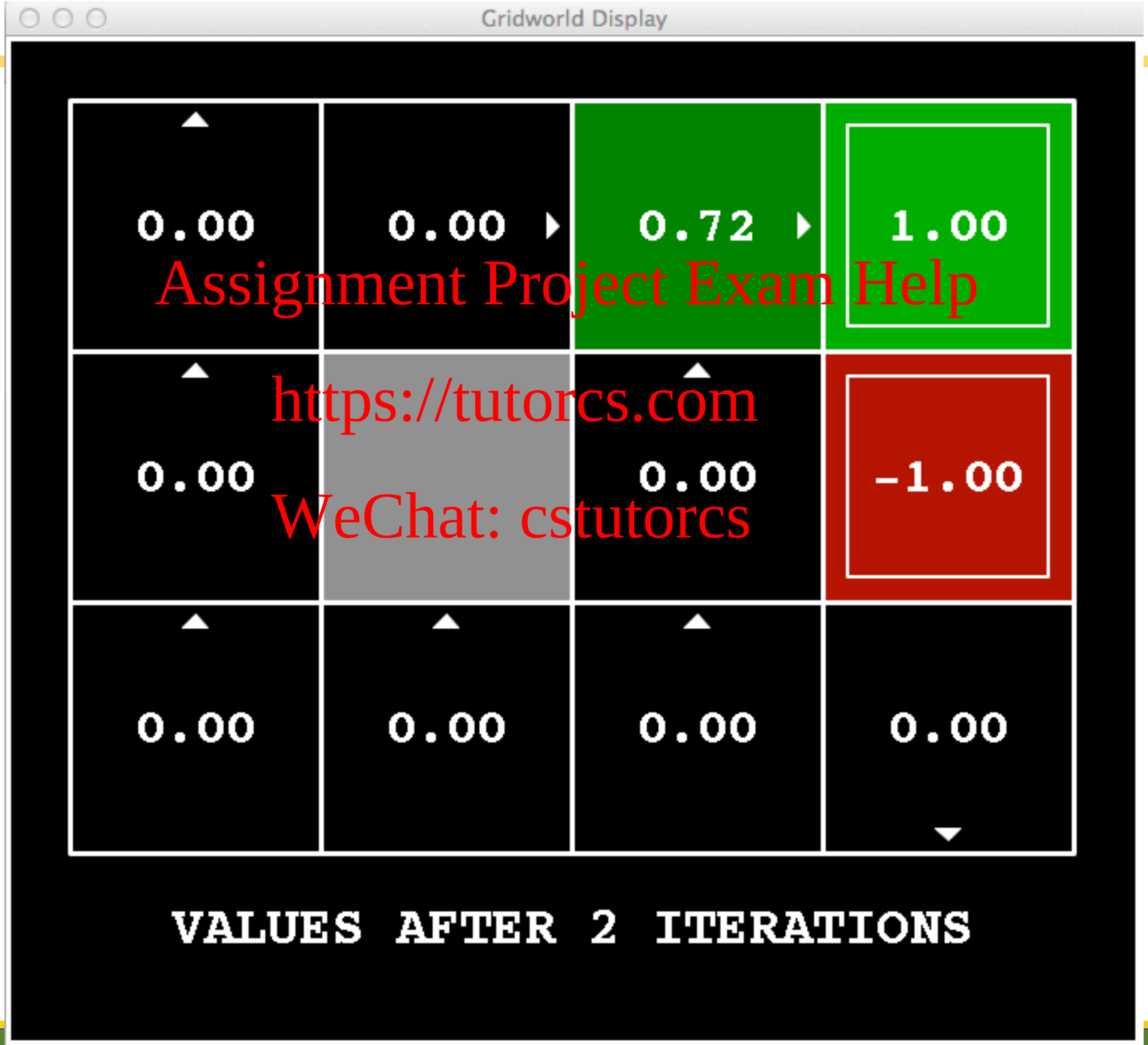


▲ 0.00	▲ 0.00	0.00 ▶	1.00
▲ 0.00		◀ 0.00	-1.00
▲ 0.00	▲ 0.00	▲ 0.00	0.00 ▼

VALUES AFTER 1 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=2

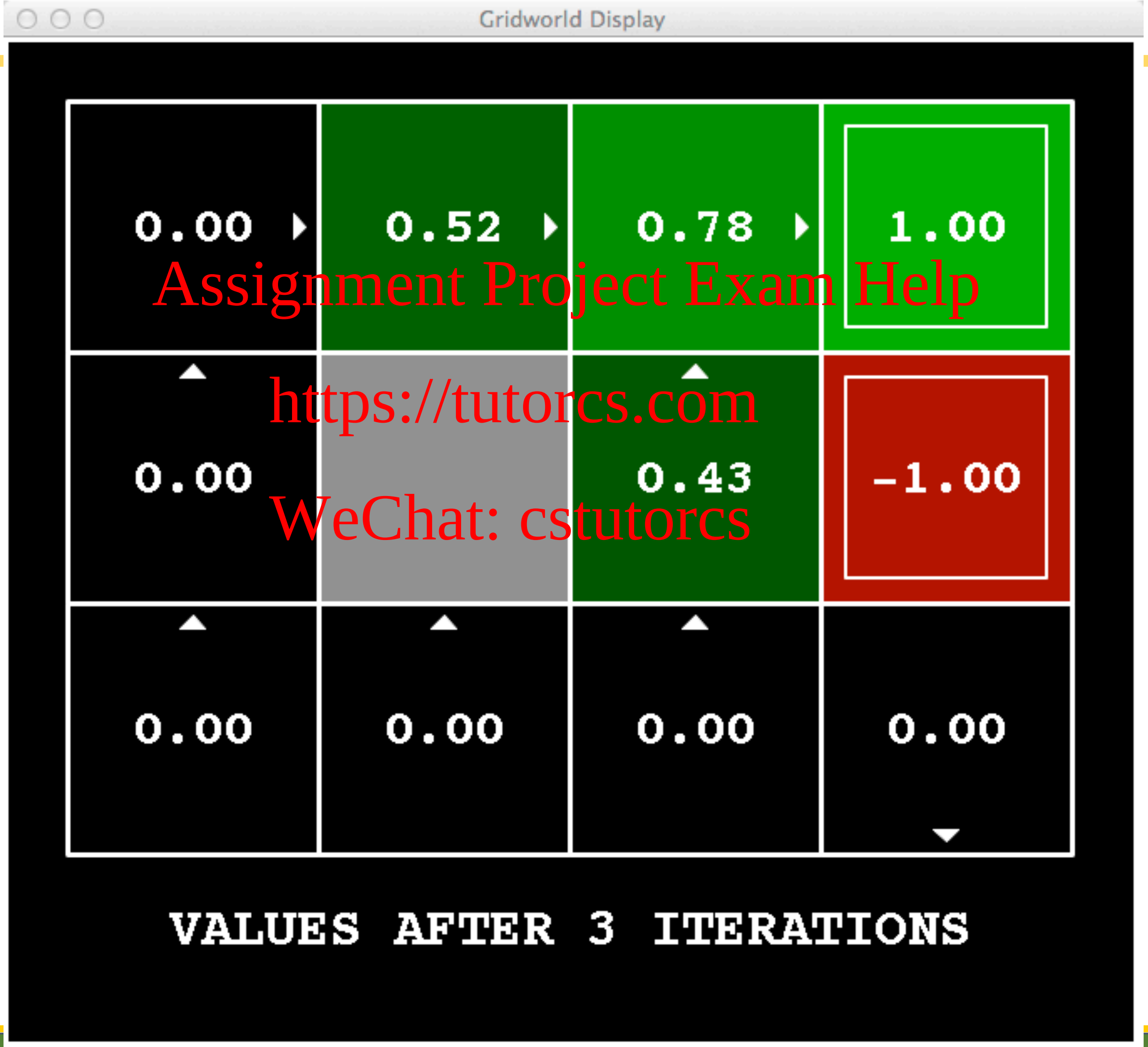


▲ 0.00	0.00 ▶	0.72 ▶	1.00
▲ 0.00		▲ 0.00	-1.00
▲ 0.00	▲ 0.00	▲ 0.00	0.00 ▼

VALUES AFTER 2 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

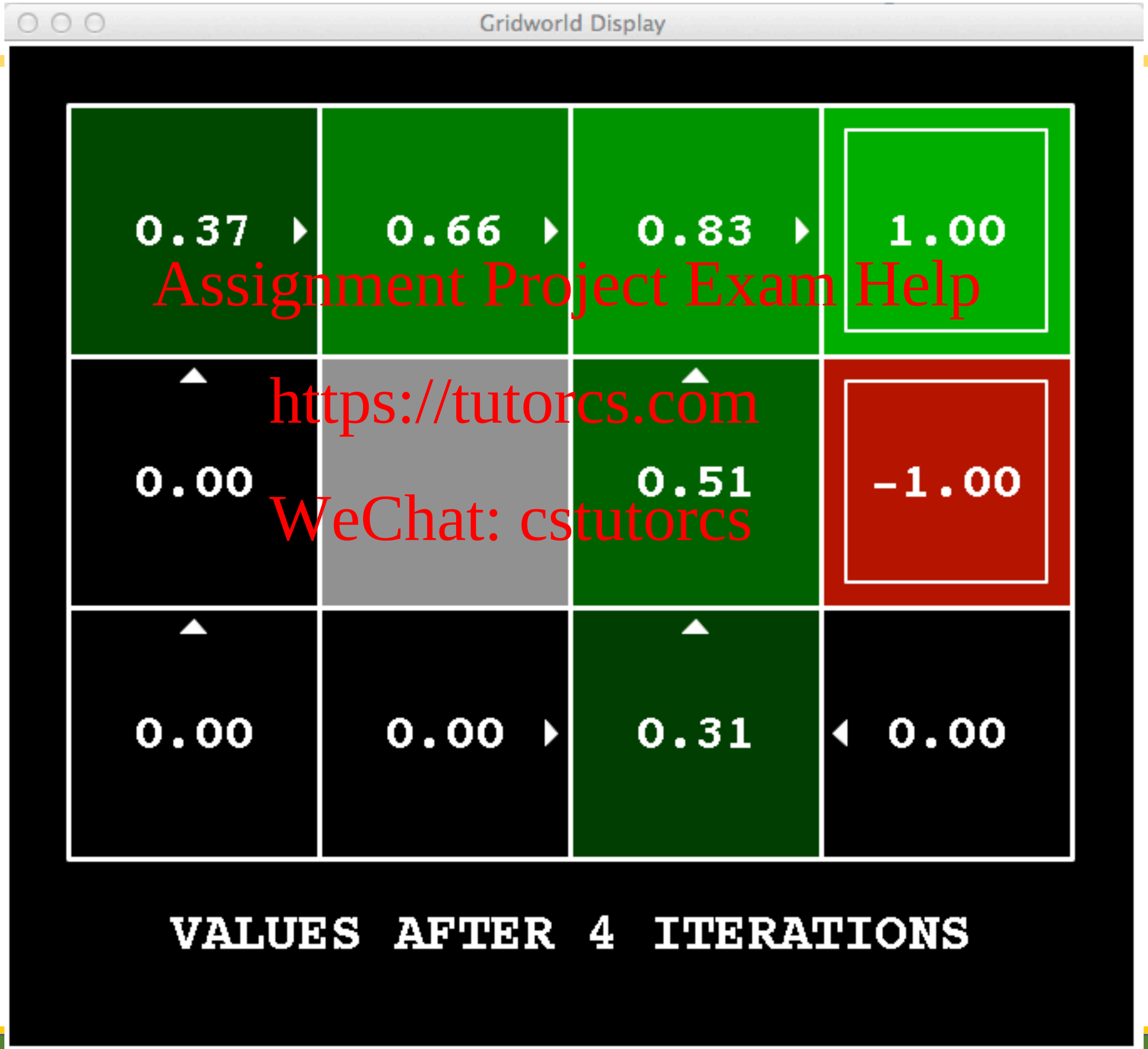


0.00 ▶	0.52 ▶	0.78 ▶	1.00
▲ 0.00		▲ 0.43	-1.00
▲ 0.00	▲ 0.00	▲ 0.00	0.00 ▼

VALUES AFTER 3 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=4

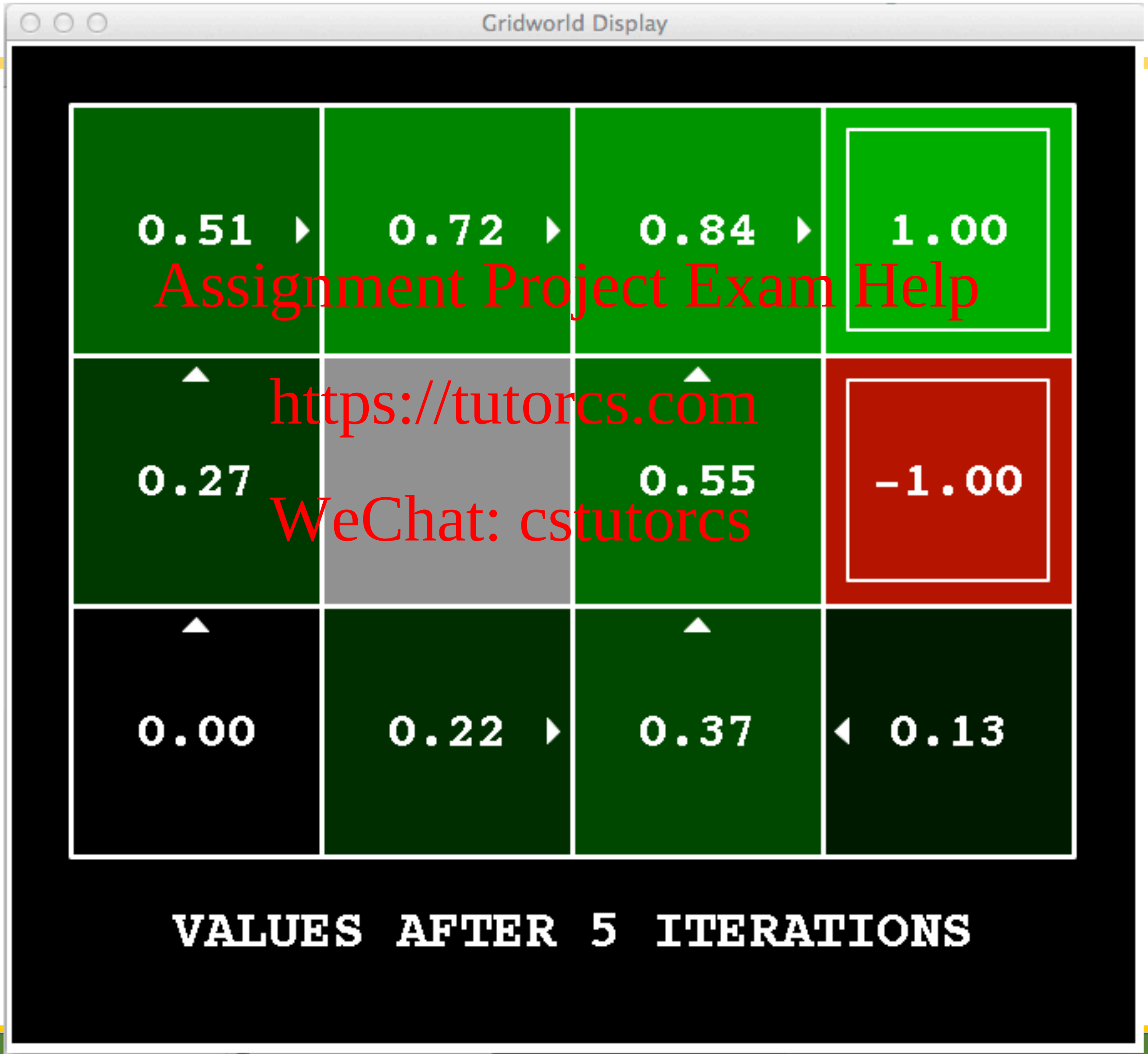


0.37 ▶	0.66 ▶	0.83 ▶	1.00
▲ 0.00		▲ 0.51	-1.00
▲ 0.00	0.00 ▶	▲ 0.31	◀ 0.00

VALUES AFTER 4 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=5

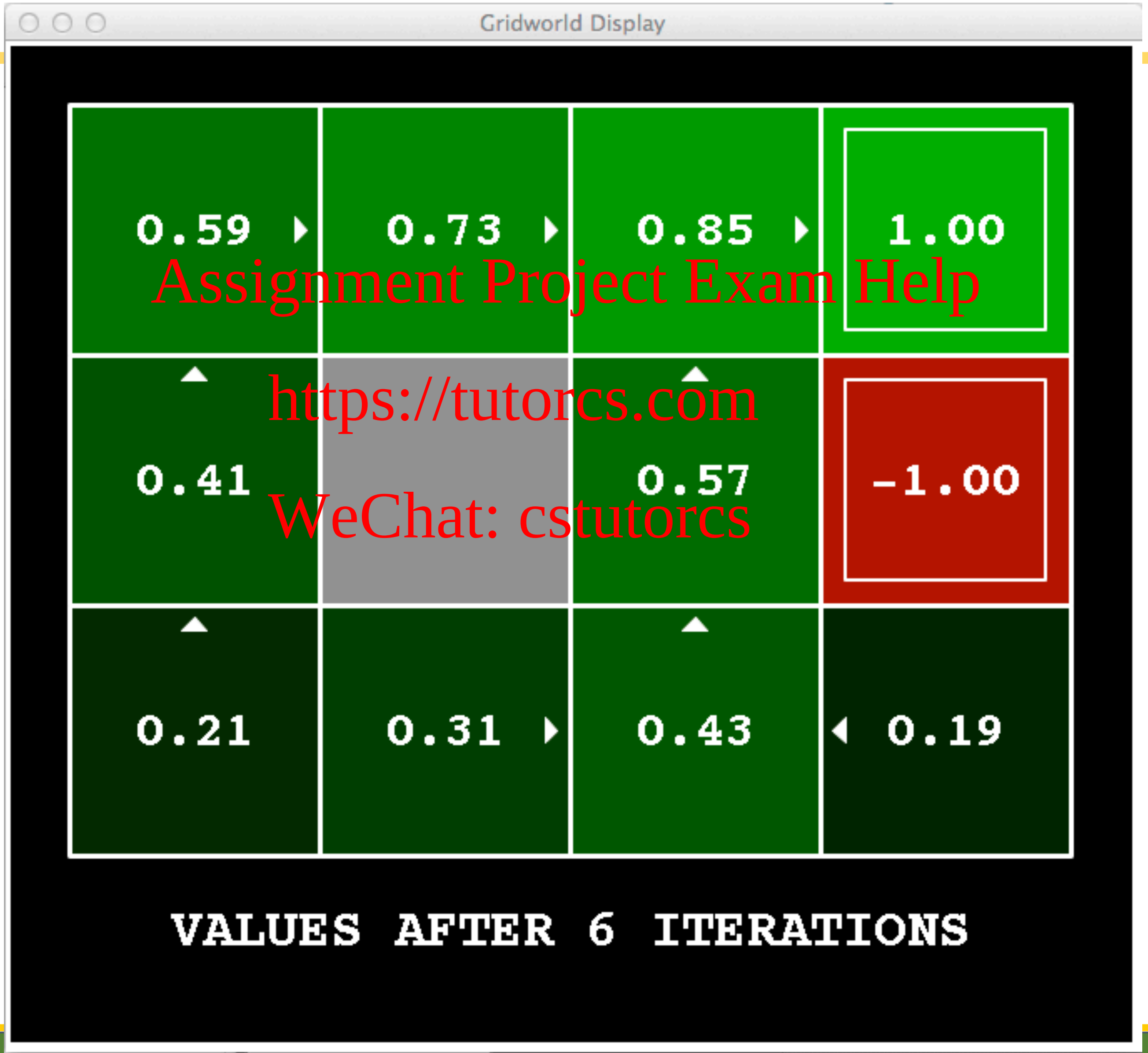


0.51 ▶	0.72 ▶	0.84 ▶	1.00
▲ 0.27		▲ 0.55	-1.00
▲ 0.00	0.22 ▶	▲ 0.37	◀ 0.13

VALUES AFTER 5 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=6

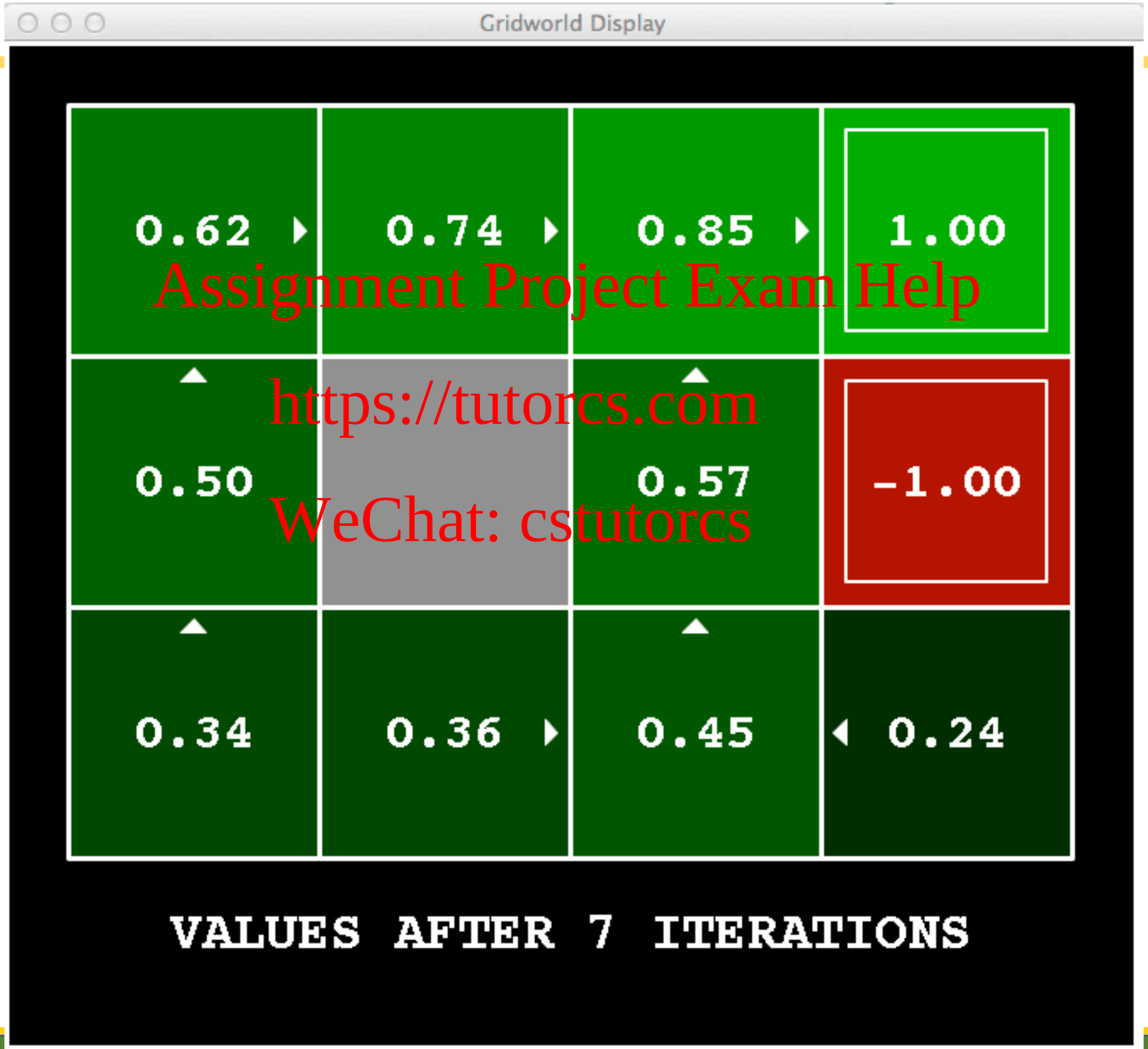


0.59 ▸	0.73 ▸	0.85 ▸	1.00
▴ 0.41		▴ 0.57	-1.00
▴ 0.21	0.31 ▸	▴ 0.43	◂ 0.19

VALUES AFTER 6 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=7

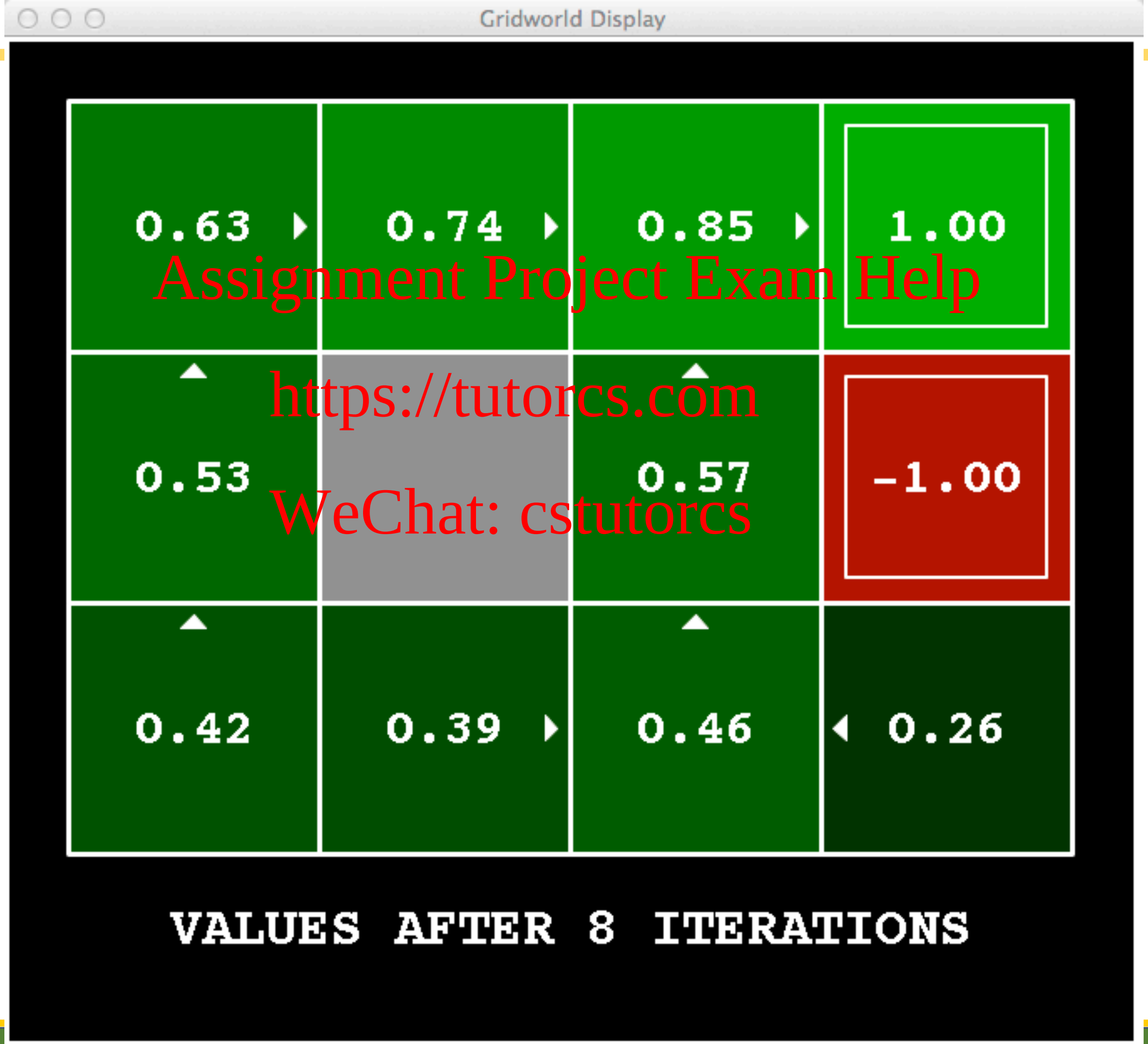


0.62 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.50		▴ 0.57	-1.00
▴ 0.34	0.36 ▸	▴ 0.45	◂ 0.24

VALUES AFTER 7 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=8

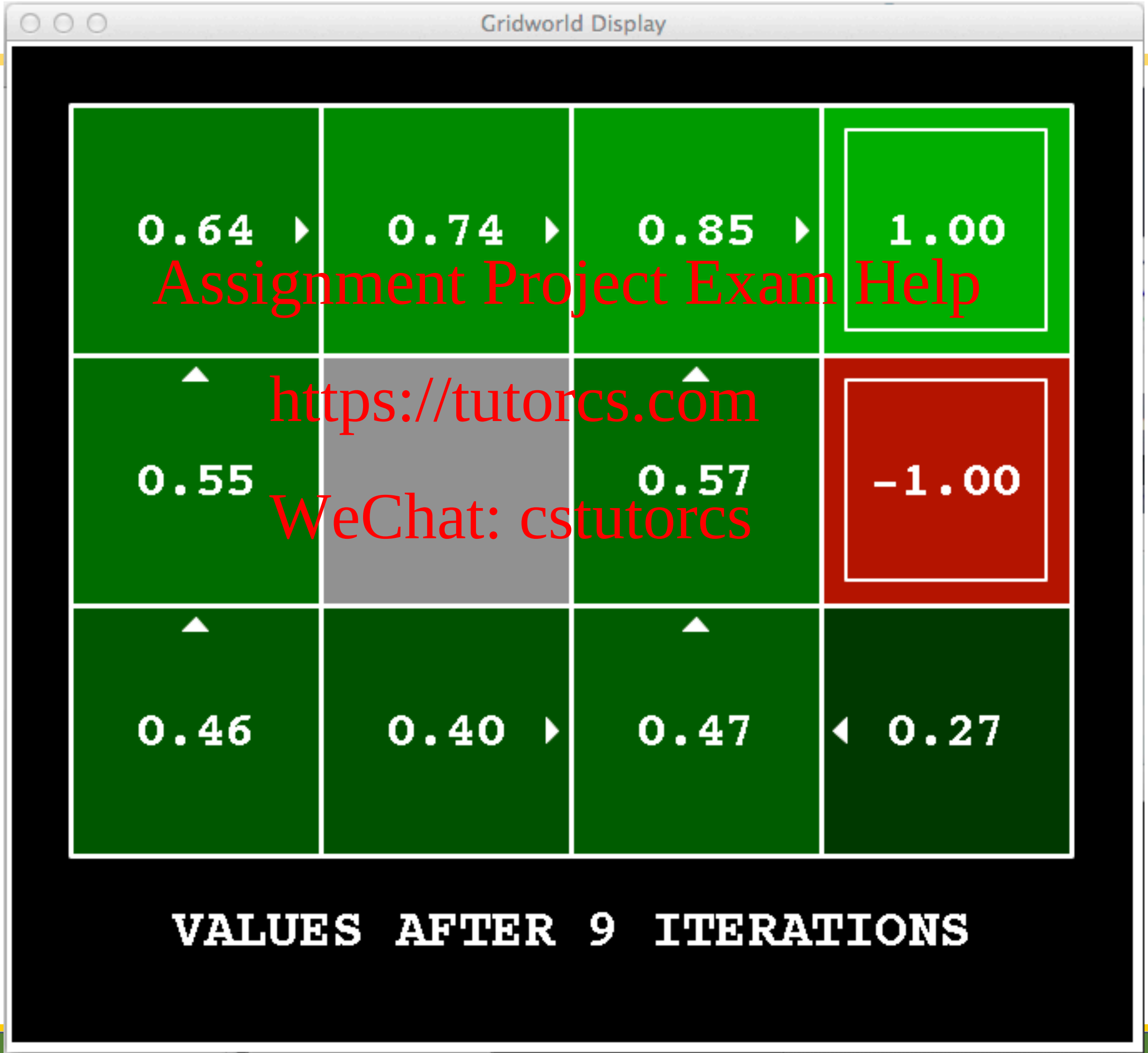


0.63 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.53		▲ 0.57	-1.00
▲ 0.42	0.39 ▶	▲ 0.46	◀ 0.26

VALUES AFTER 8 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=9

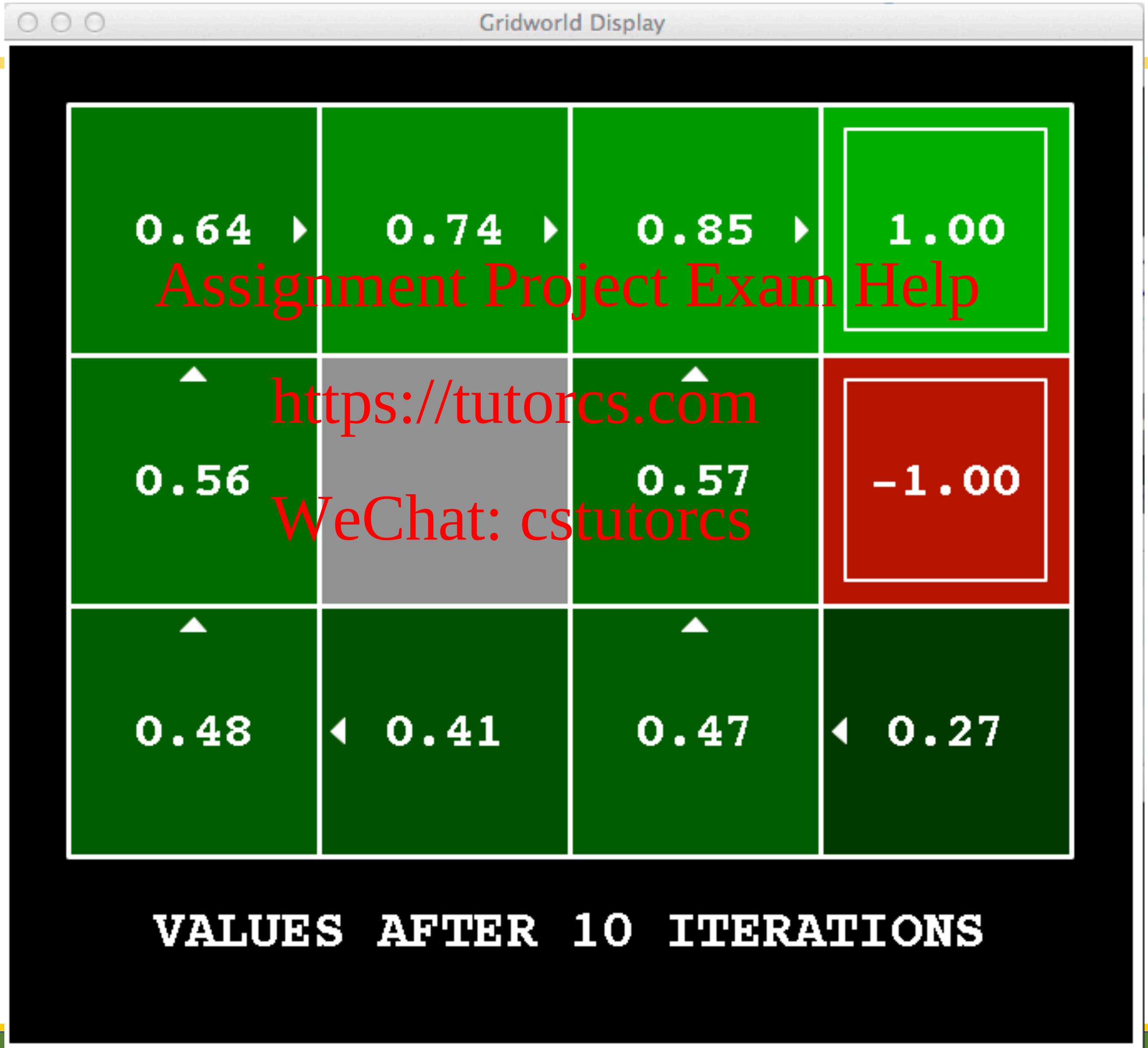


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.55		▲ 0.57	-1.00
▲ 0.46	0.40 ▶	▲ 0.47	◀ 0.27

VALUES AFTER 9 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=10

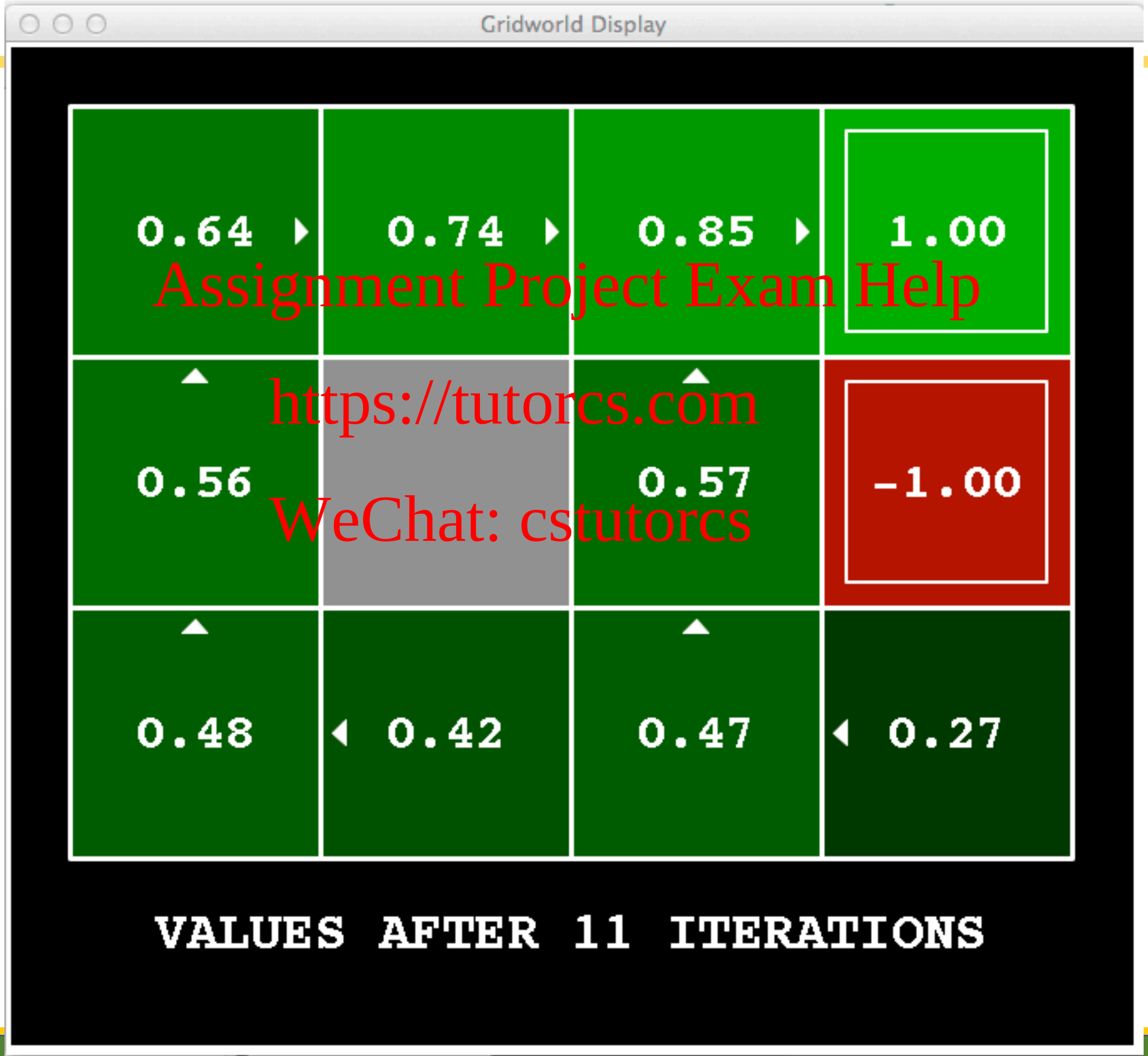


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.56		▲ 0.57	-1.00
▲ 0.48	◀ 0.41	▲ 0.47	◀ 0.27

VALUES AFTER 10 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=11

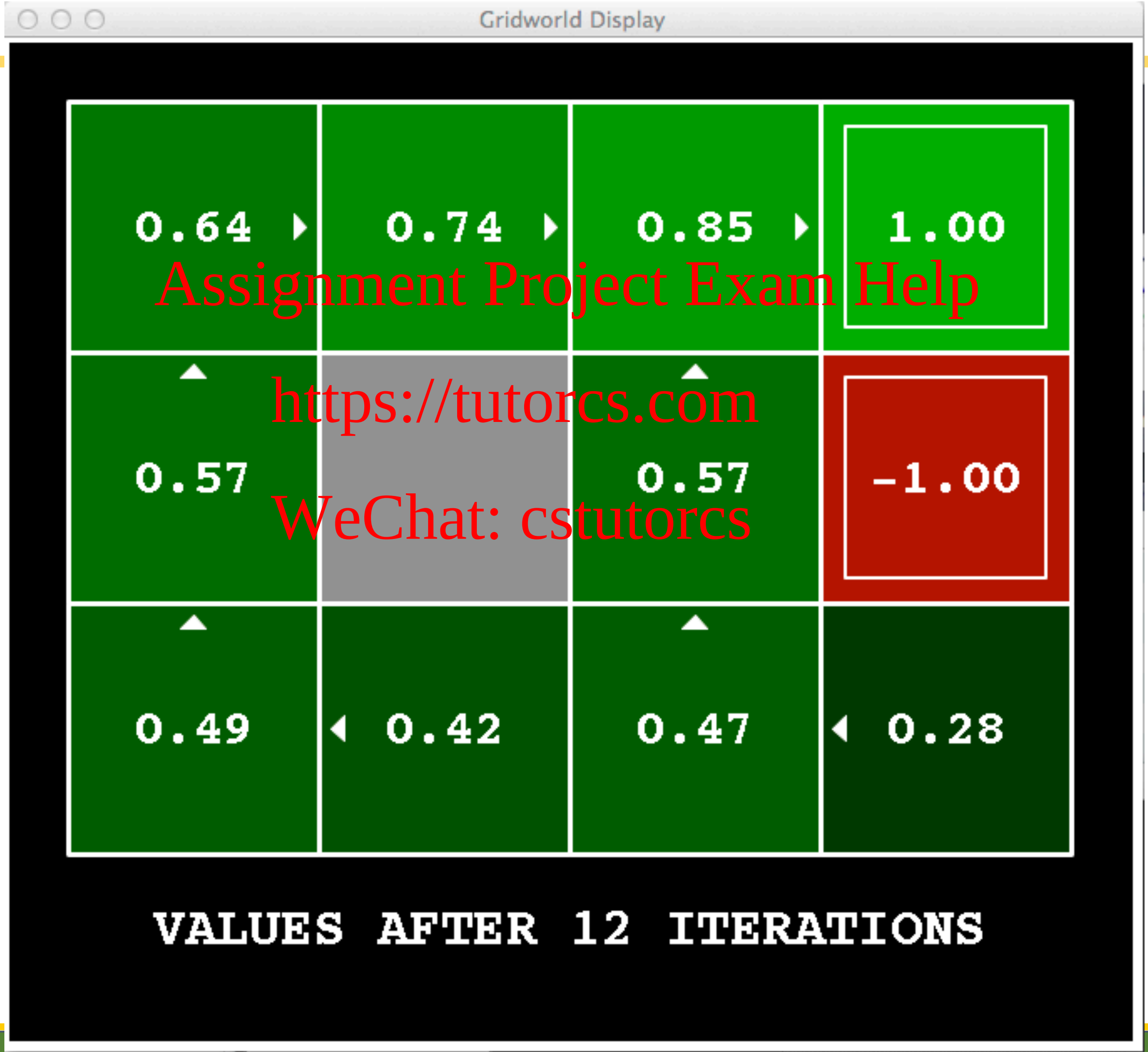


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.56		▲ 0.57	-1.00
▲ 0.48	◀ 0.42	▲ 0.47	◀ 0.27

VALUES AFTER 11 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=12

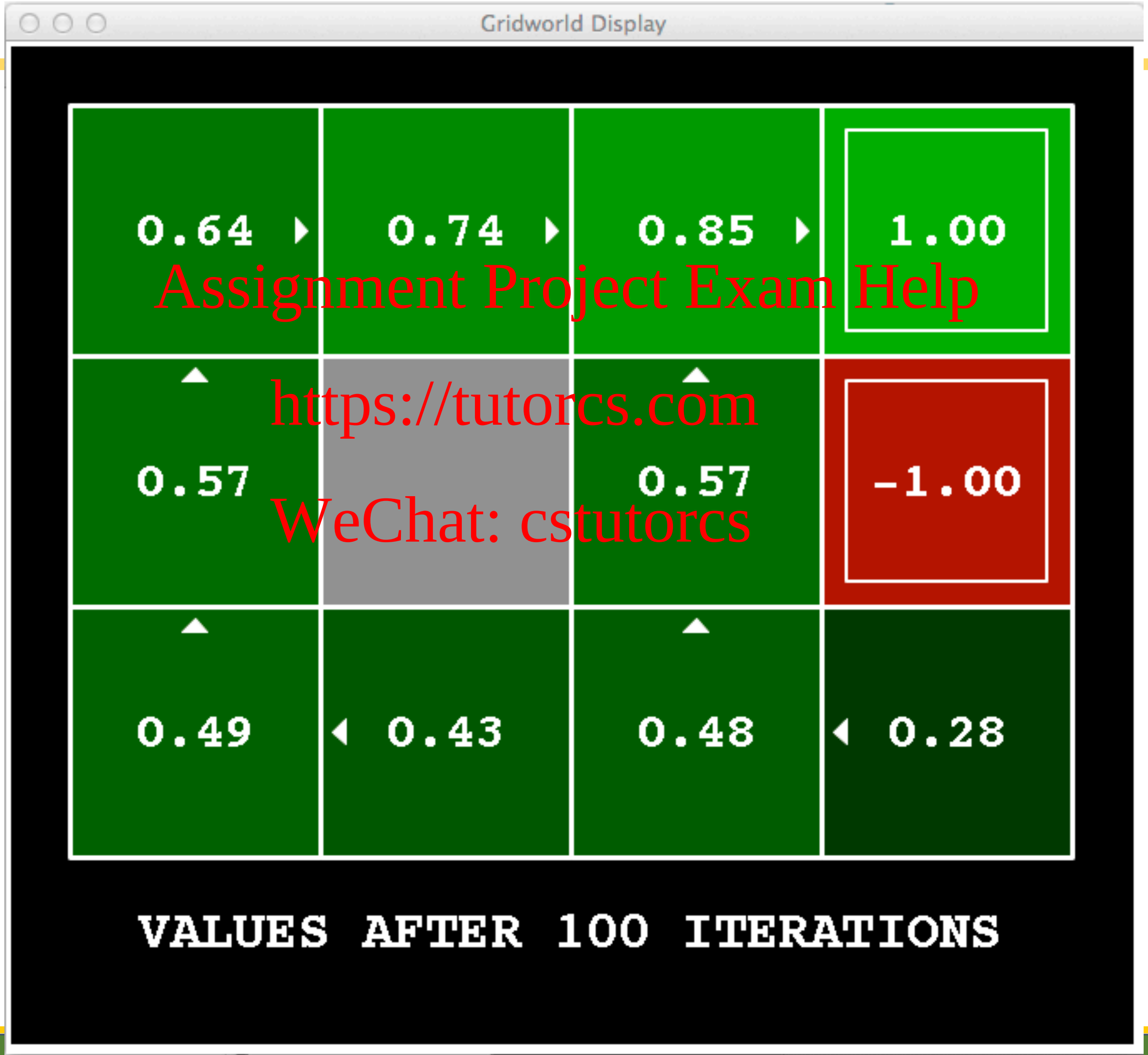


0.64 ▸	0.74 ▸	0.85 ▸	1.00
▴ 0.57		▴ 0.57	-1.00
▴ 0.49	◂ 0.42	▴ 0.47	◂ 0.28

VALUES AFTER 12 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



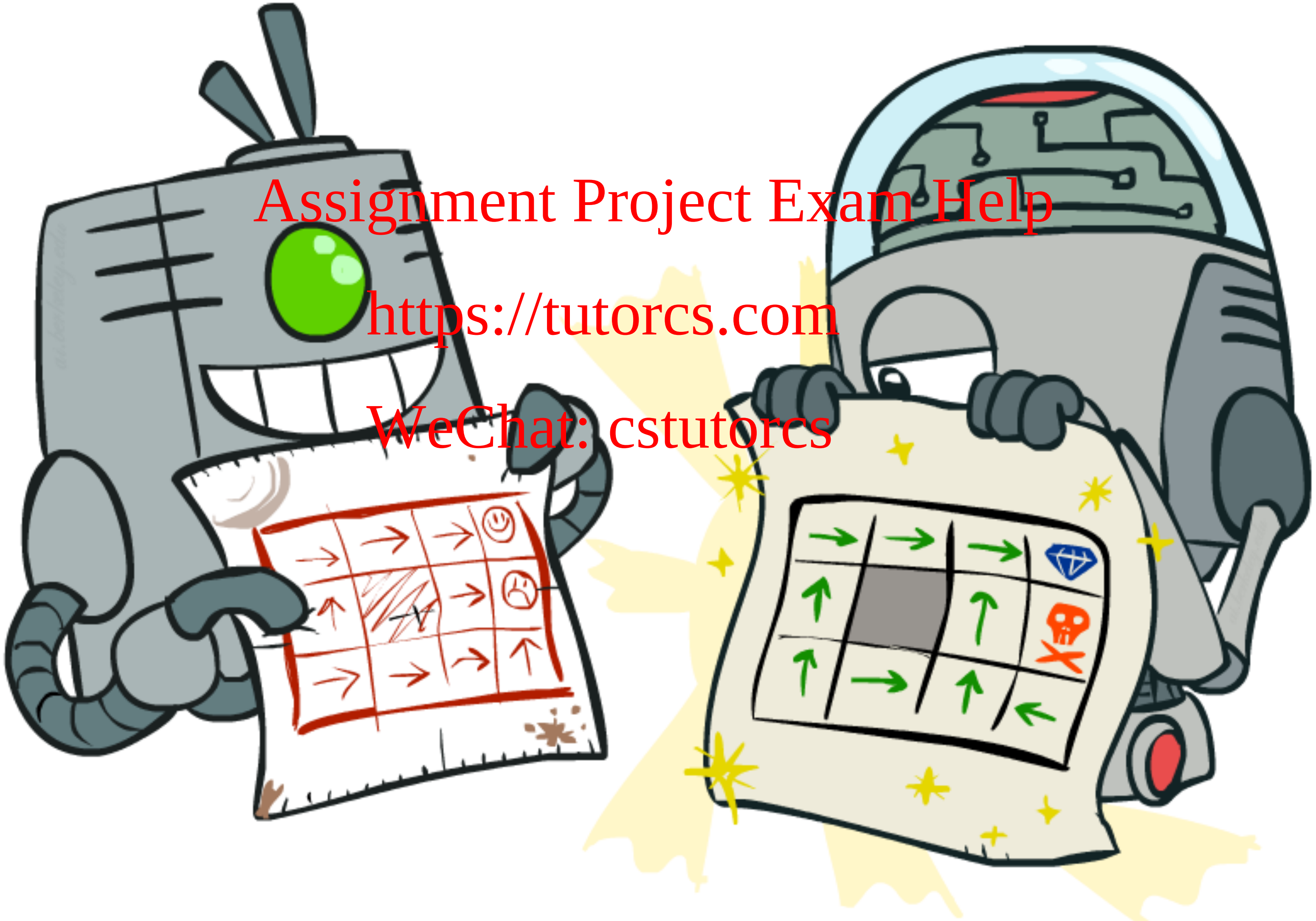
Gridworld Display

0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.57		▲ 0.57	-1.00
▲ 0.49	◀ 0.43	▲ 0.48	◀ 0.28

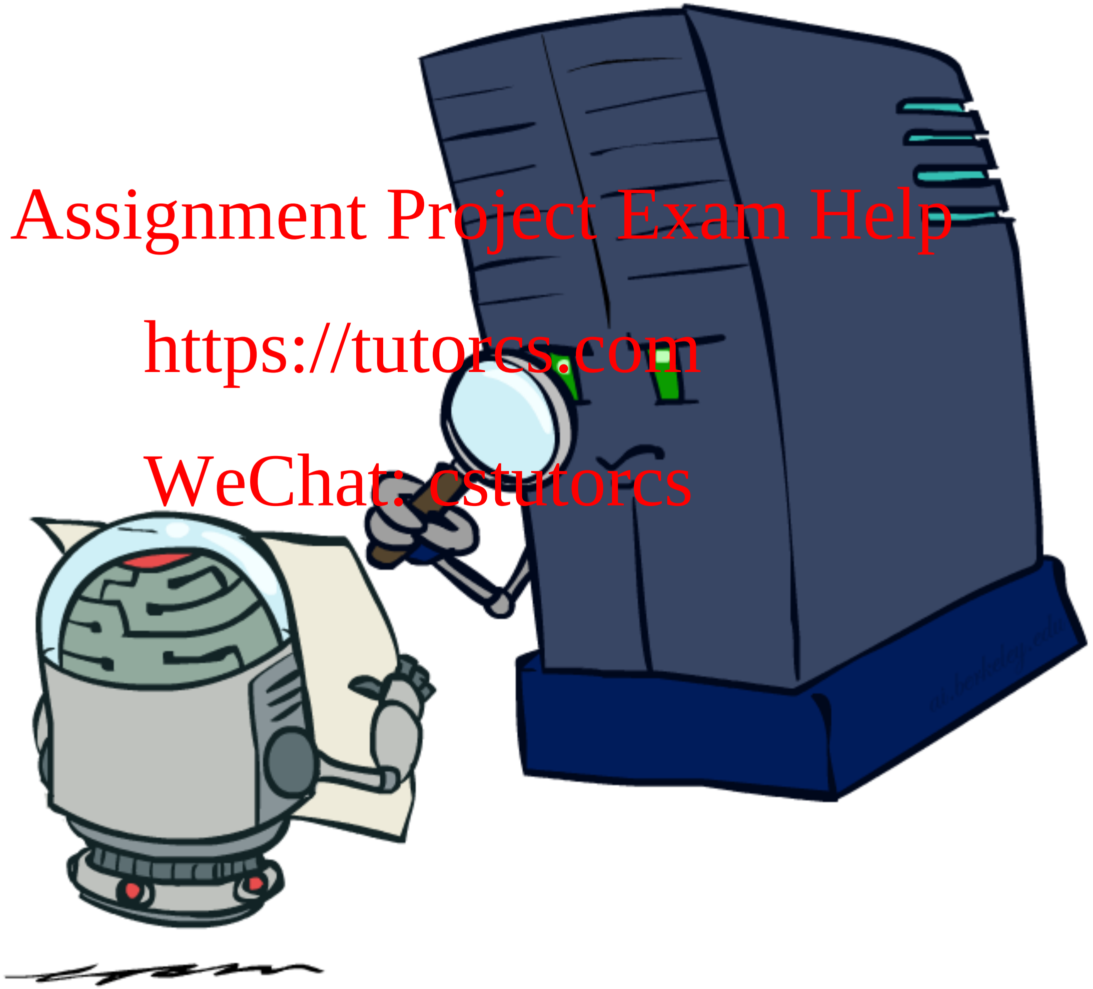
VALUES AFTER 100 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

Policy Methods

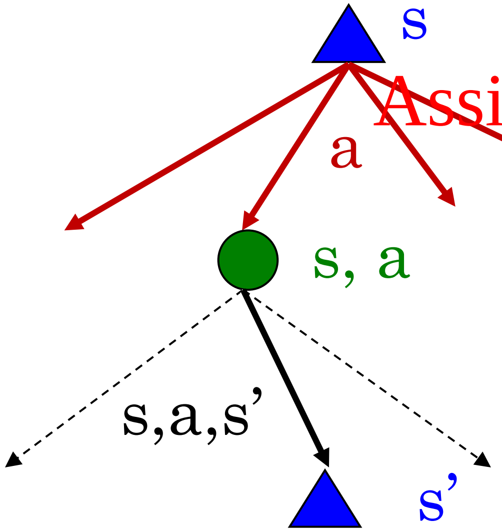


Policy Evaluation

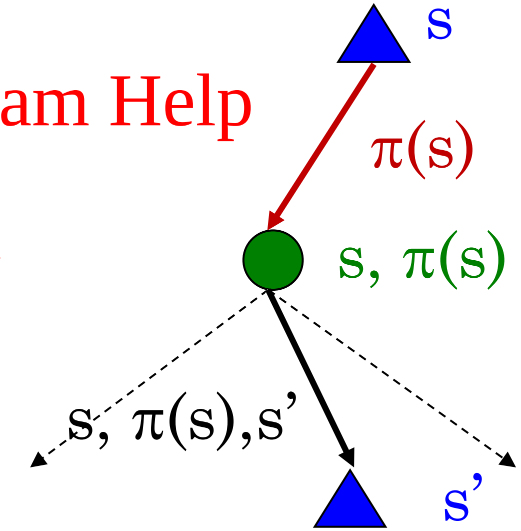


Fixed Policies

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

Utilities for a Fixed Policy

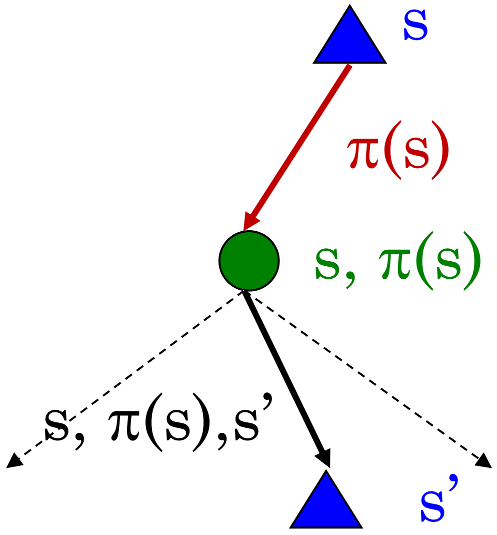
- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- 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

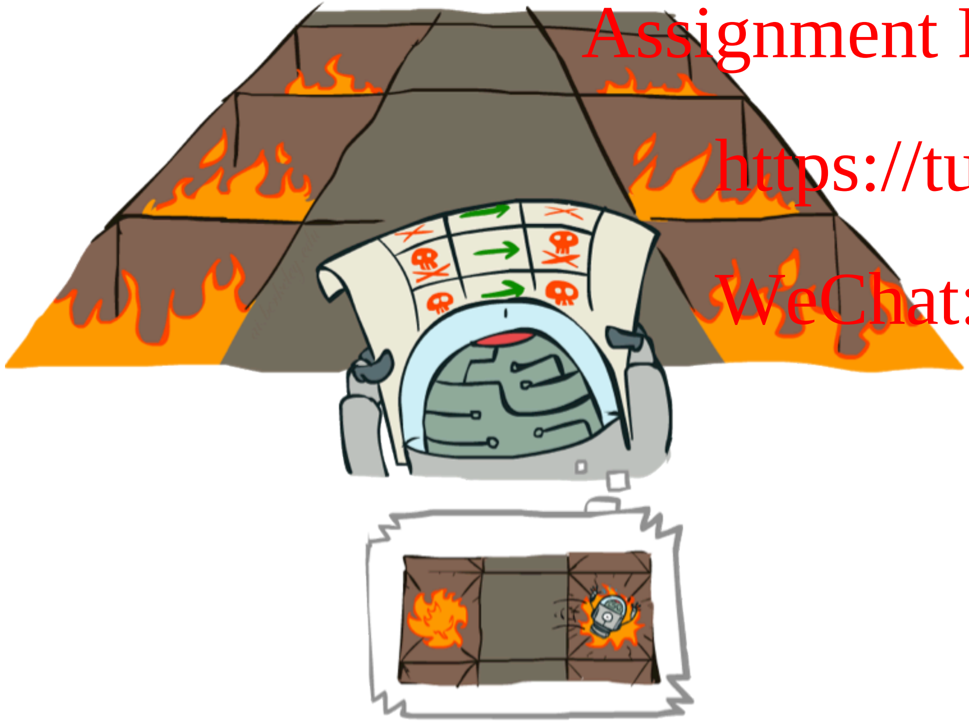
- Recursive relation (one-step look-ahead / Bellman equation):

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

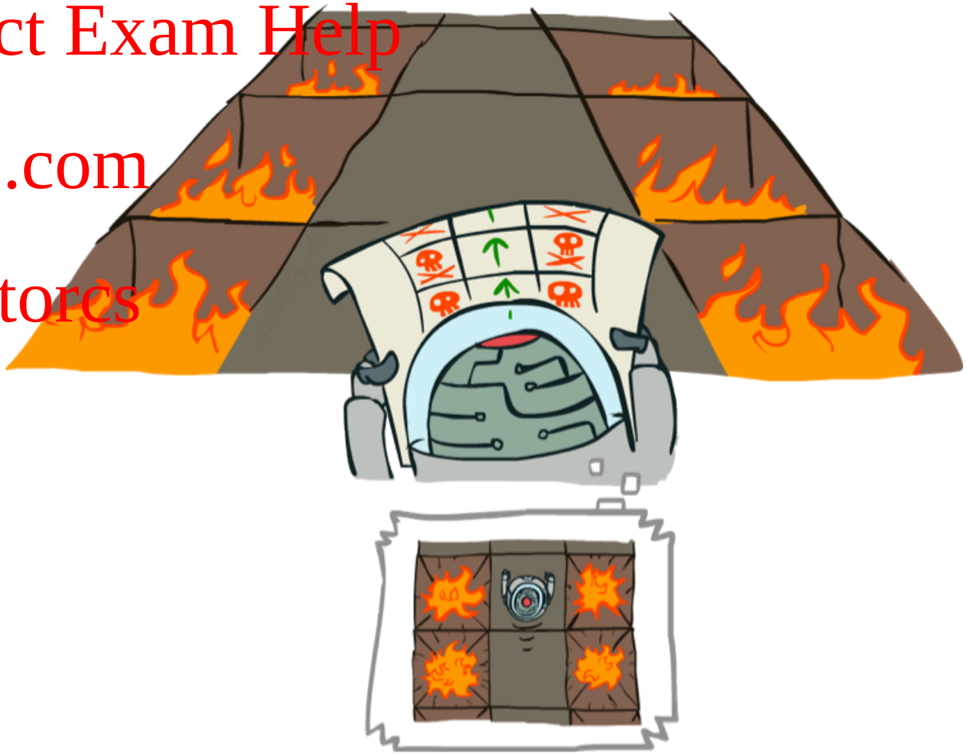


Example: Policy Evaluation

Always Go Right

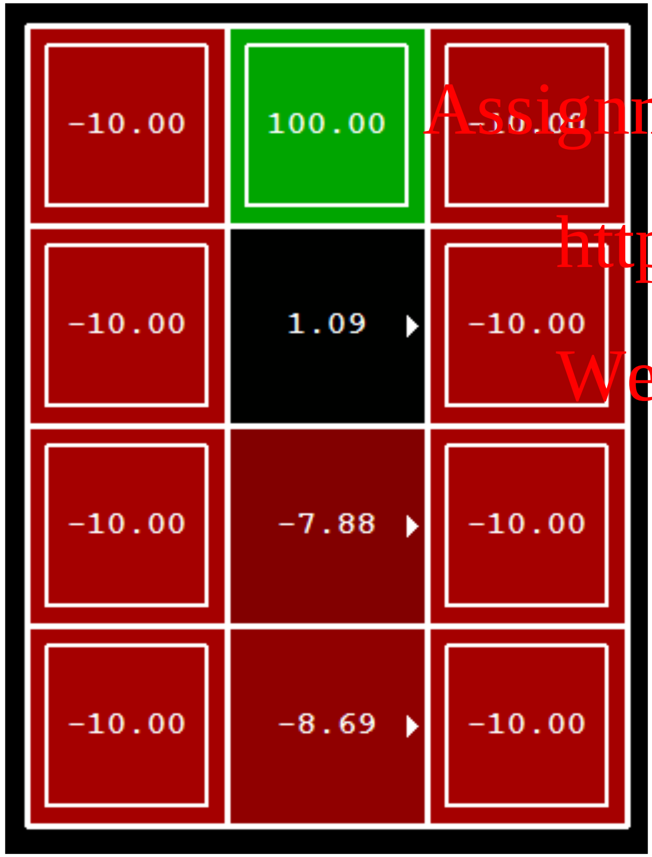


Always Go Forward



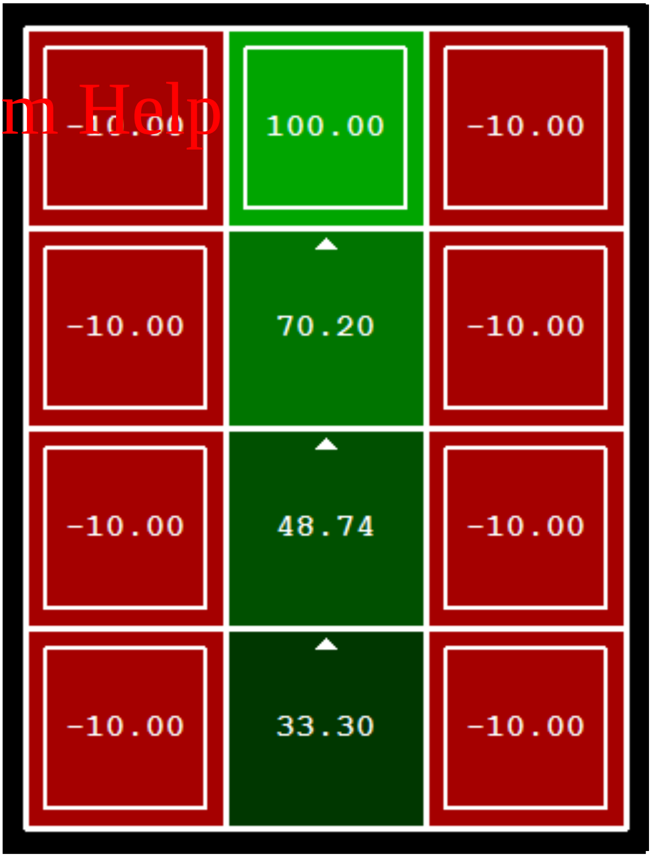
Example: Policy Evaluation

Always Go Right



-10.00	100.00	-10.00
-10.00	1.09 ▸	-10.00
-10.00	-7.88 ▸	-10.00
-10.00	-8.69 ▸	-10.00

Always Go Forward



-10.00	100.00	-10.00
-10.00	▴ 70.20	-10.00
-10.00	▴ 48.74	-10.00
-10.00	▴ 33.30	-10.00

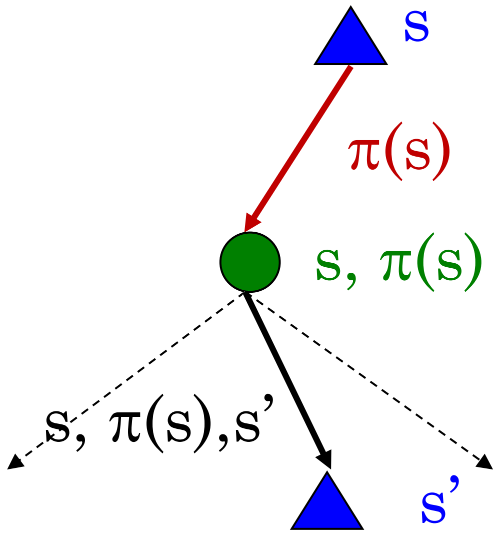
Policy Evaluation

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

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

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

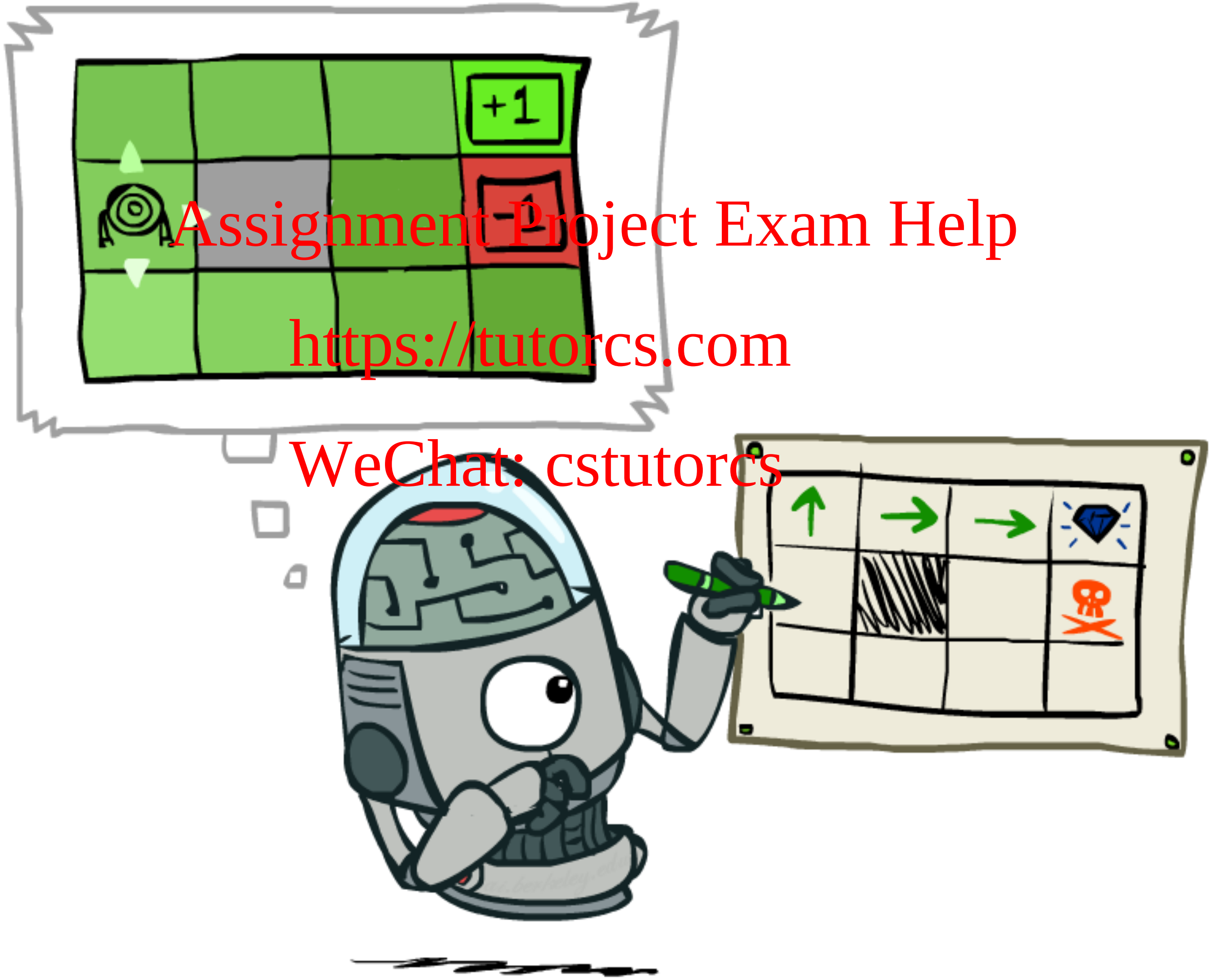
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$



- 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

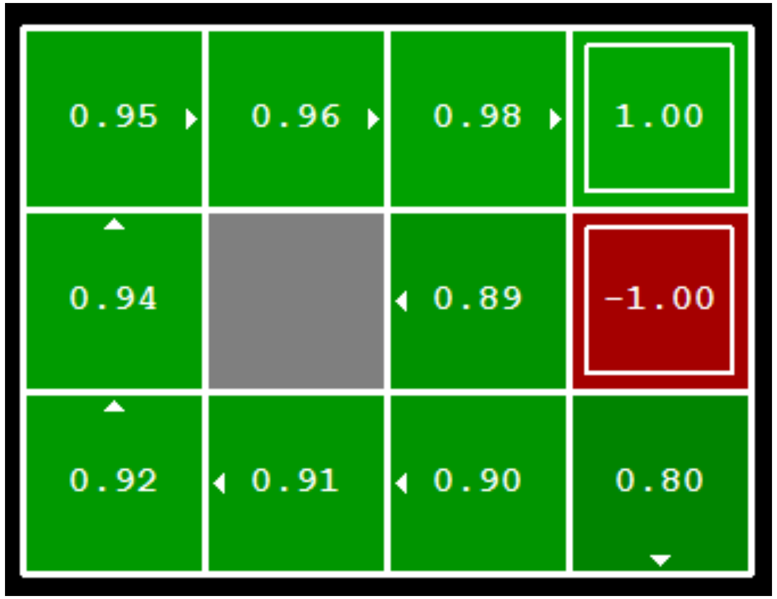


Computing Actions from Values

- 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)



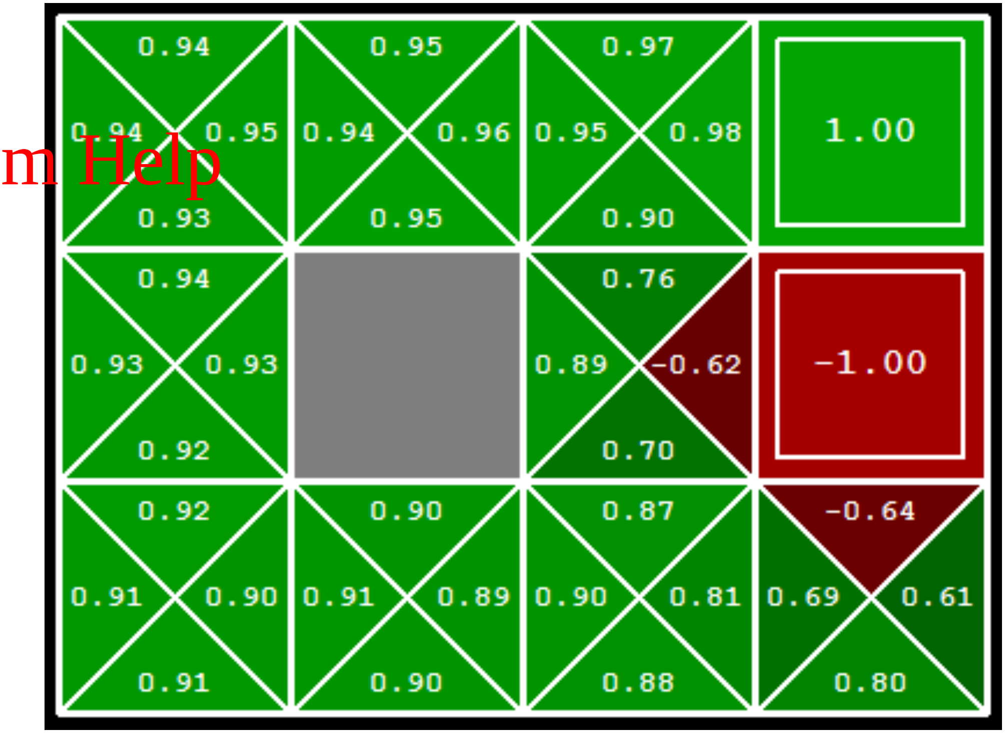
0.95 ▸	0.96 ▸	0.98 ▸	1.00
▴ 0.94		◂ 0.89	-1.00
▴ 0.92	◂ 0.91	◂ 0.90	0.80 ▾

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

- 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:

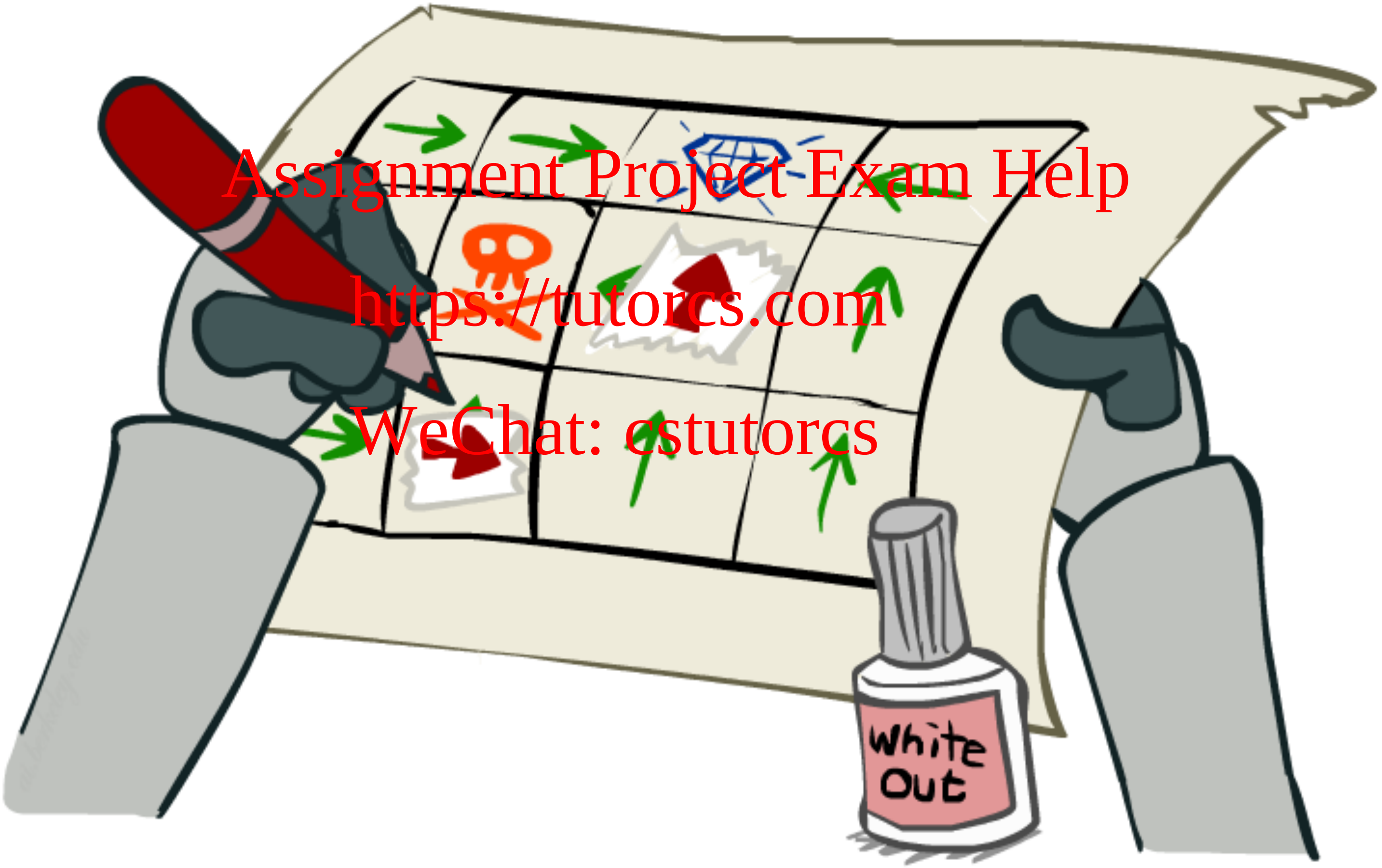


- How should we act?
  - Completely trivial to decide!

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

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

Policy Iteration



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:

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

- 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

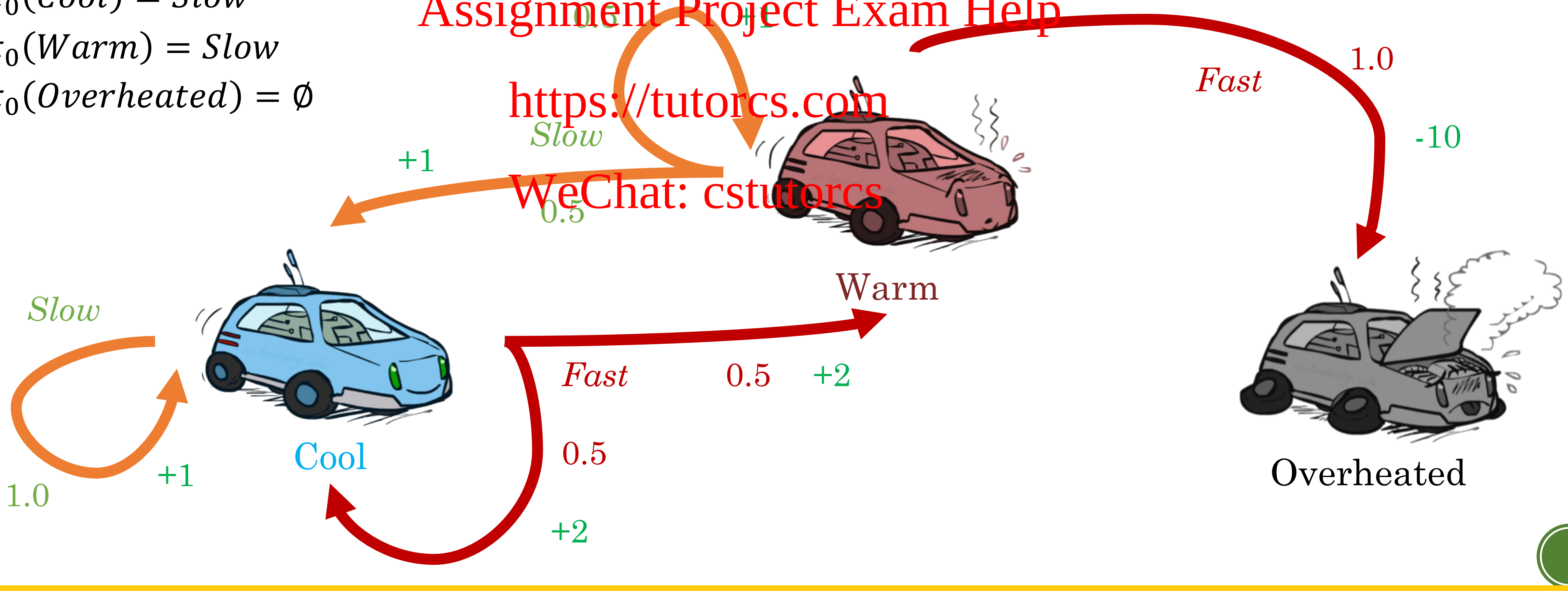
Summary: MDP Algorithms

- 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 look-ahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

Example: Racing

- Discount: $\gamma = 0.1$
- Initial policy
  - $\pi_0(Cool) = Slow$
  - $\pi_0(Warm) = Slow$
  - $\pi_0(Overheated) = \emptyset$


0.5
+1
Slow
+1
0.5
Fast
1.0
-10
Warm
Slow
1.0
+1
Cool
Fast
0.5
+2
0.5
+2
Overheated