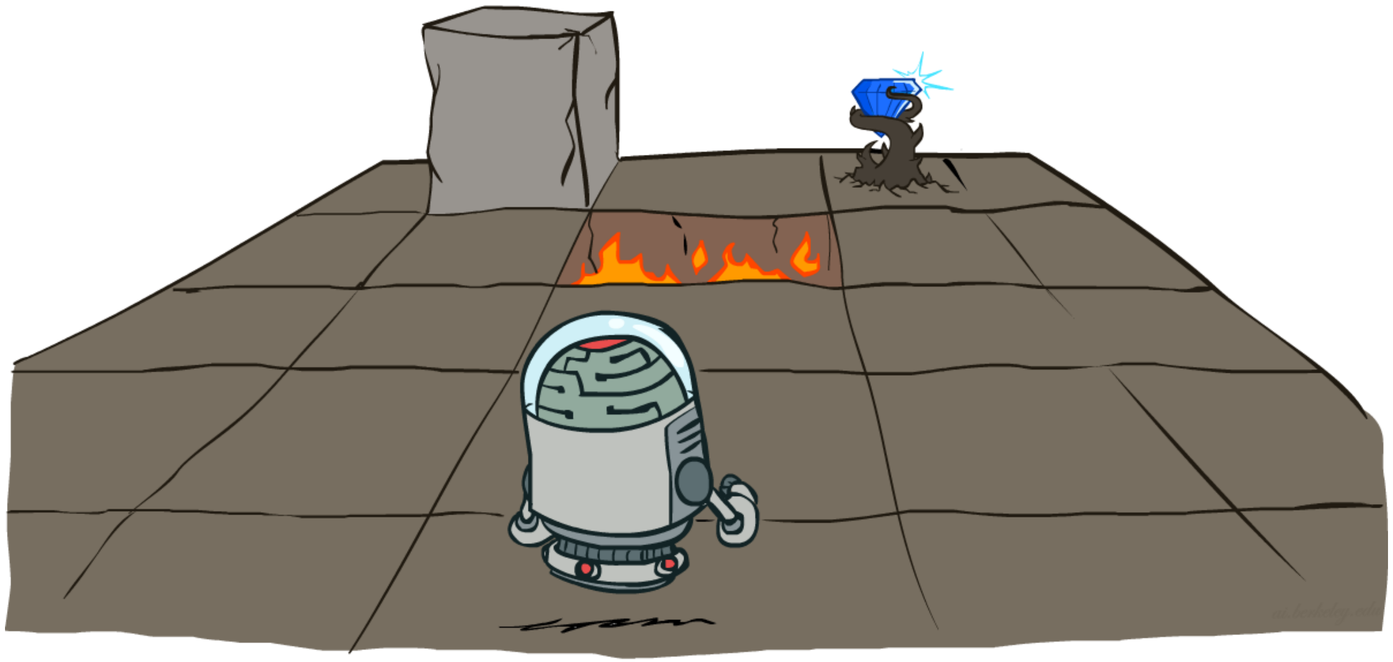
CSCE642 Reinforcement Learning

Chapter 3,4: Markov Decision Processes and dynamic programming

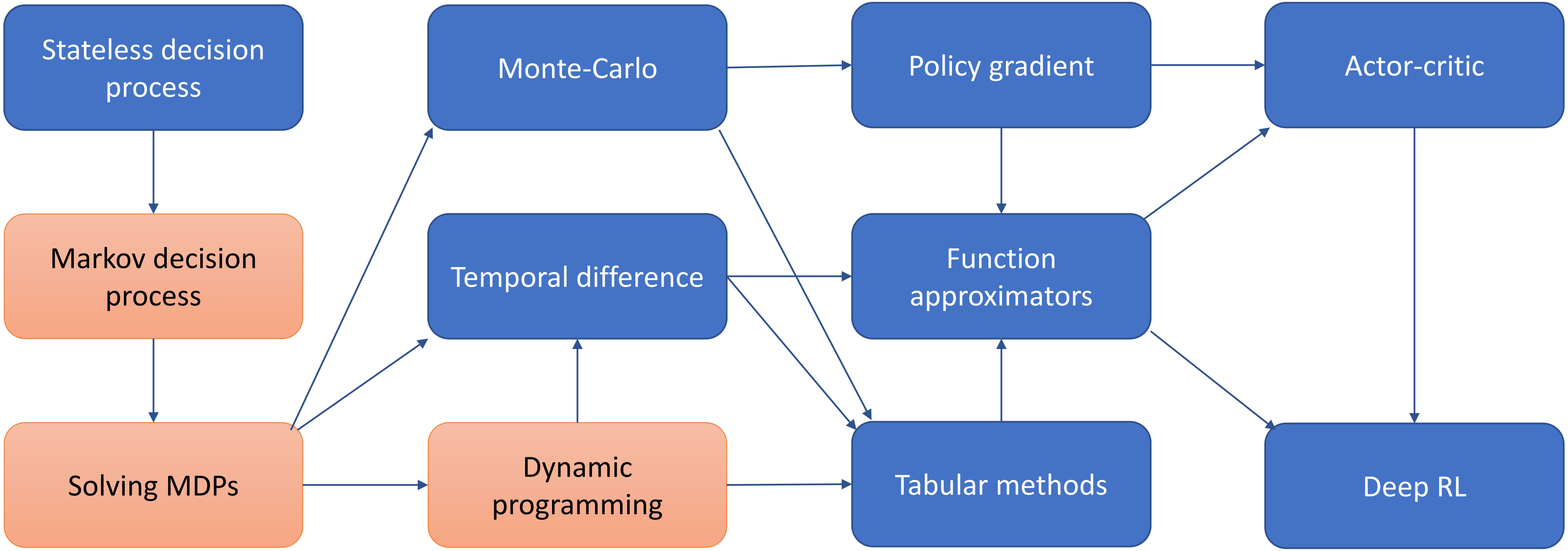


Instructor: Guni Sharon

Today

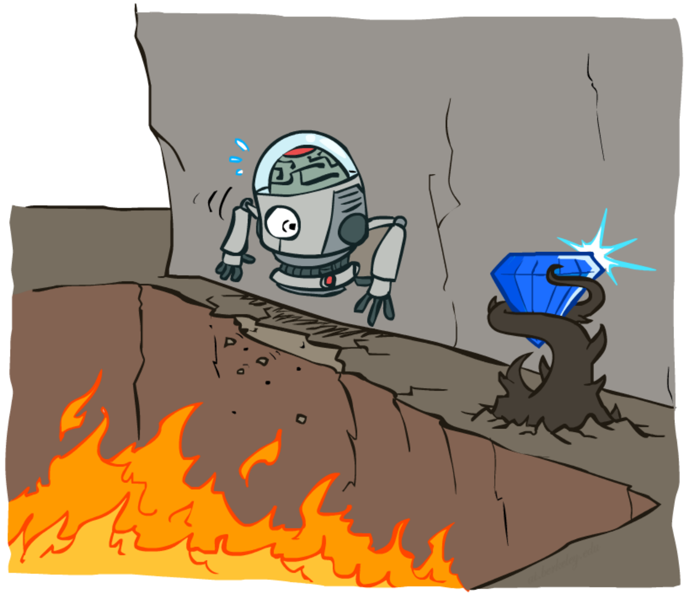
- **Class:**
  - Markov Decision Processes
- **Assignments**:
  - Quiz Multi Armed Bandits on Canvas by Sunday, September 3, EOD.
  - Go over tutorials
    - Linear algebra
    - Basic probability
    - Python
    - Numpy
    - Gym (OpenAI)
- **Project**:
  - Go over the project description and timeline
  - Find a partner (a relevant discussion board is available on Campuswire)

CSCE-689, Reinforcement Learning



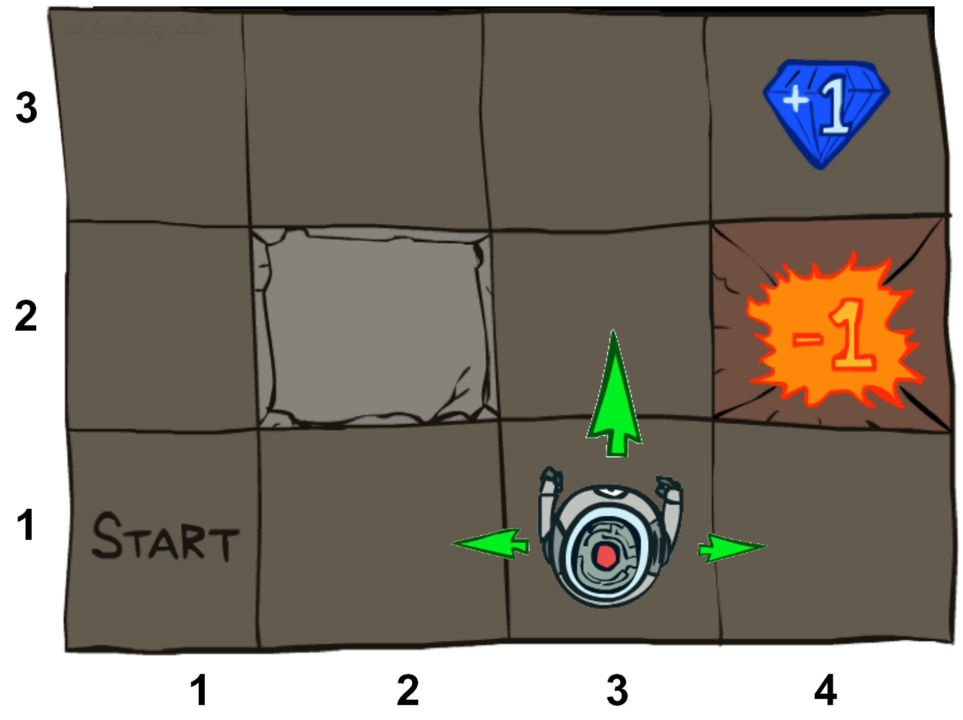
Introducing states

- Multi-armed bandit
  - (Action -> reward) -> (action -> reward) …
- Markov Decision Processes
  - Agent acts in an environment
  - The environment includes a set of possible states
  - At each state, a different set of actions might be applicable
  - Each state, action pair have a different expected outcome (reward and next state)
  - [State -> action -> reward -> state] -> action -> reward …



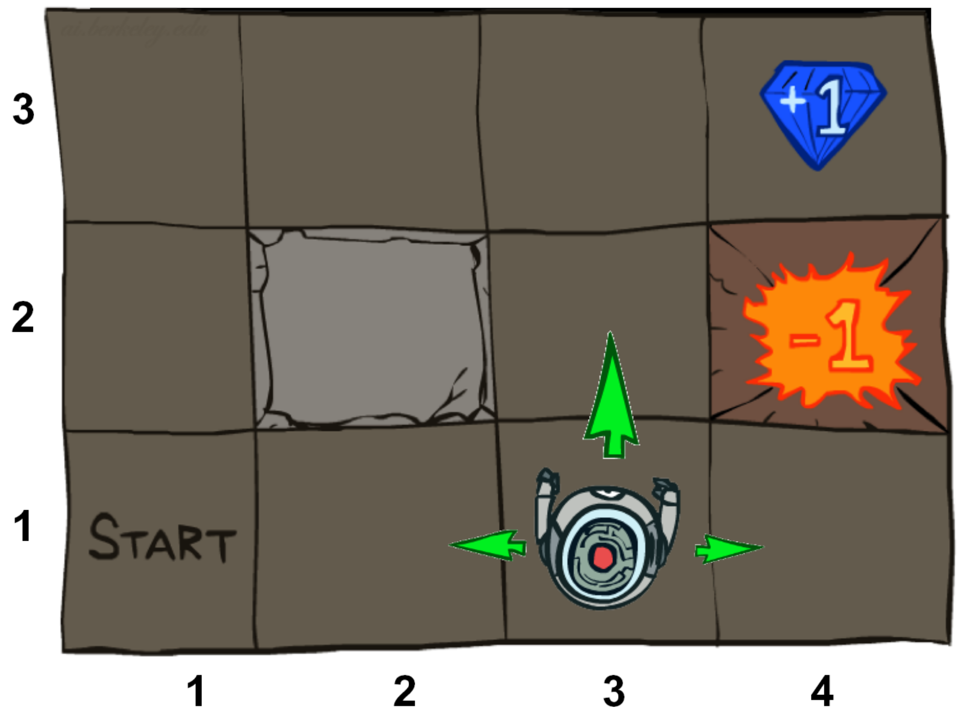
Example: Grid World

- 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 (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have taken, the agent stays put
- The agent receives rewards each time step
  - -0.1 per step (battery loss)
  - +1 if arriving at (4,3) and -1 for arriving at (4,2)
- Objective: maximize accumulated rewards



Markov Decision Processes

- An MDP is defined by:
  - A set of states $\mathcal{S}$
  - A set of actions $\mathcal{A}$
  - State-transition probabilities $P(s'|s,a)$
    - Probability of arriving to s' after performing a at s
    - Also called the model dynamics
  - A reward function $R(s,a,s')$
    - The utility gained from arriving at s' after performing a at s
    - Sometimes just $R(s,a)$ or even $R(s)$
  - A start state
  - Maybe a terminal state



Today

- We will assume that the MDP model is known
  - The distribution over outcomes (reward and next state) for each state and action
- Not a practical assumption but it will help us get started
- Assuming the same for the bandits problem directly implies the optimal policy
- Why is this not the case for MDPs?

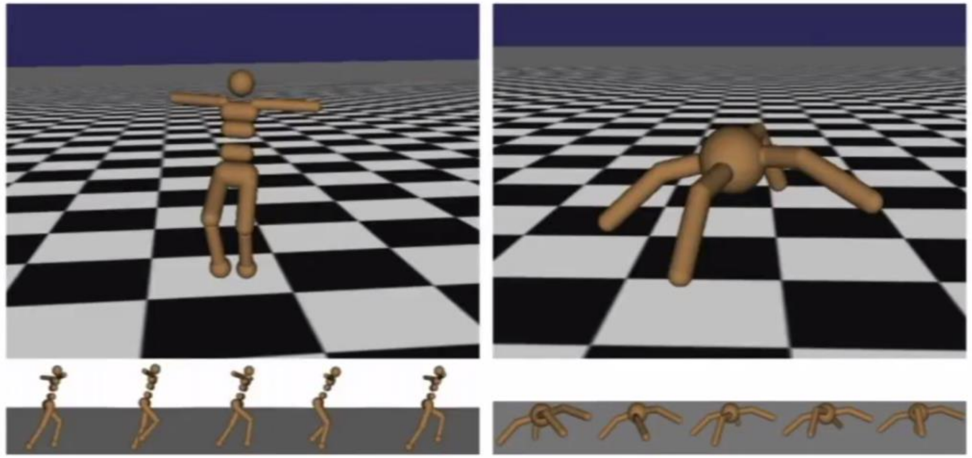
MDP formalization - Autonomous driving

[“Learning to Drive in a Day”, Kendall et. al., 2018]

- State:
  - image from front camera
- Actions:
  - Steering from [-1,1], acceleration from [0,1], brake from [0,1]
- Reward:
  - Drive off road = -10
  - Hit a person = -1000
  - Drive at speed limit = +2
  - Align with lane = +4
  - Run a stop sign = -5
  - Erratic driving= -3
  - …
- State-transition probabilities:
  - defined by stochasticity in action outcomes and other actuators

MDP formalization - Robot locomotion ["Policy gradient reinforcement learning for fast quadrupedal locomotion", Kohl & Stone, 2004]

- State:
  - angle and position of joints, obstacles/pits
- Actions:
  - torques applied to joints
- Reward:
  - forward speed
- State-transition probabilities:
  - defined by stochasticity in action outcomes and obstacles



Feedback Control For Cassie With Deep Reinforcement Learning

Zhaoming Xie[1], Glen Berseth[1], Patrick Clary[2], Jonathan Hurst[2], Michiel van de Panne[1]

[1]University of British Columbia, [2]Oregon State University

MDP formalization – Video games [“Playing Atari with deep reinforcement learning”, Mnih et al., 2013]

- State:
  - raw pixels
- Actions:
  - game controls
- Reward:
  - change in score
- State-transition probabilities:
  - defined by stochasticity in game evolution

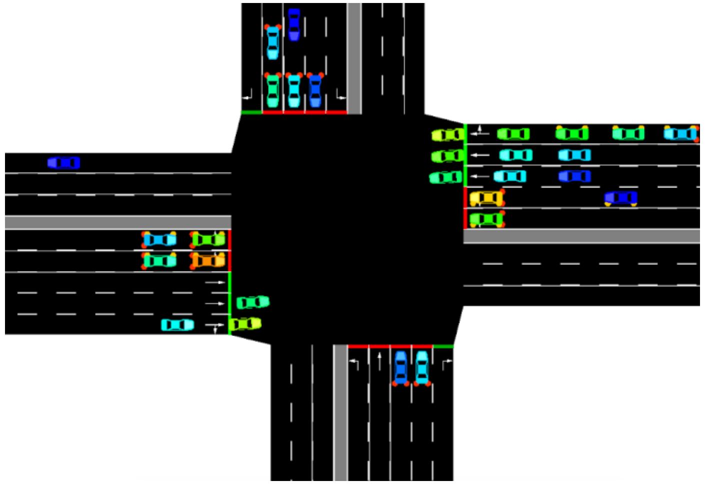
Is this decision process Markovian?
No. History dictates the ball’s velocity vector which matters for optimal action selection.
Solution (Mnih et al.): a state includes the 4 last frames

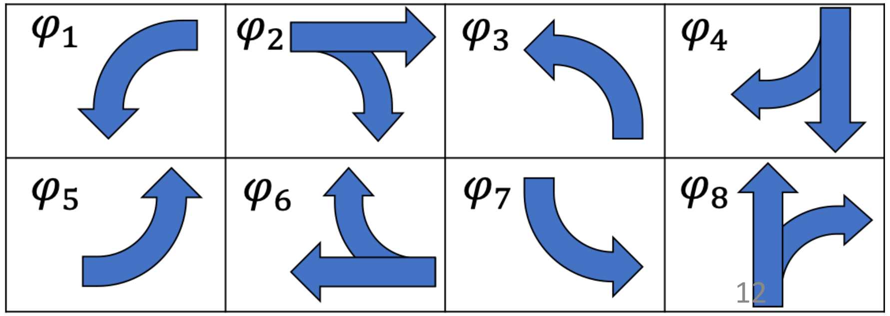
Before training

peaceful swimming

MDP formalization - Traffic signal control ["Learning an Interpretable Traffic Signal Control Policy", Ault et al., 2020]

- State:
  - current signal assignment (green, yellow, and red assignment for each phase)
  - For each lane: number of approaching vehicles, accumulated waiting time, number of stopped vehicles, and average speed of approaching vehicles
- Actions:
  - signal assignment: $\{\phi^g, \phi^y, \phi^r\}$
- Reward:
  - Reduction in traffic delay
- State-transition probabilities:
  - defined by stochasticity in approaching demand





Action Values

E/W	E/EL	W/WL	EL/WL	NPL/SPL	NPL/S	N/NL	N/SPL	N/S	S/SL	NL/SL	Avg Reward
-2.07	-8.3	-6.97	-10.4	1.87	-3.71	-9.89	-2.75	-5.06	-10.08	-5.6	-0.1418

Regulatable Values

Phase	(Queue	Approach	Wait	Speed	QLength	Avg Wait	Queue	Approach	Wait	Speed	QLength	Avg Wait)	{FULL	PART	NO	PERM}	
E/W	0.23	0.19	0.02	0.14	0.6	0.23	1.13	0.0	0.89	0.0	0.78	1.01	0.0	1.07	0.0	0.0	5.62
E/EL	0.28	0.06	0.0	0.63	0.28	0.0	0.19	0.0	0.06	0.0	0.11	0.07	0.0	1.93	0.0	0.0	3.25
W/WL	0.0	0.03	0.0	0.82	0.0	0.0	0.92	0.26	0.51	0.01	0.4	0.54	0.0	1.33	0.0	0.0	4.65
EL/WL	0.0	0.0	0.0	0.37	0.0	0.0	0.59	0.01	0.19	0.0	0.44	0.2	0.0	0.0	2.29	0.0	4.08
NPL/SPL	0.13	0.01	0.0	0.09	0.04	0.99	2.16	0.03	0.59	0.0	0.62	0.84	1.22	0.0	0.0	0.0	6.74
NPL/S	0.06	0.0	0.09	0.16	0.04	0.63	1.69	0.03	0.76	0.0	0.74	0.63	1.25	0.0	0.0	0.0	6.02
N/NL	0.0	0.01	0.0	0.02	0.0	0.01	0.0	0.0	0.0	0.01	0.0	0.0	0.03	0.0	0.0	0.0	0.0
N/SPL	0.0	0.05	0.0	0.29	0.0	0.0	0.61	0.13	0.17	0.03	0.12	0.07	0.55	0.0	0.0	0.0	0.81
N/S	0.0	0.1	0.0	0.24	0.0	0.0	0.44	0.24	0.0	0.08	0.08	0.0	0.78	0.0	0.0	0.0	0.92
S/SL	0.0	0.0	0.0	0.0	0.0	0.0	1.11	0.17	0.27	0.85	0.45	0.06	0.41	0.0	0.0	0.0	1.21
NL/SL	0.0	0.0	0.0	0.0	0.0	0.0	0.33	0.0	0.38	0.01	0.33	0.59	0.52	0.0	0.0	0.0	0.85

Initial performance

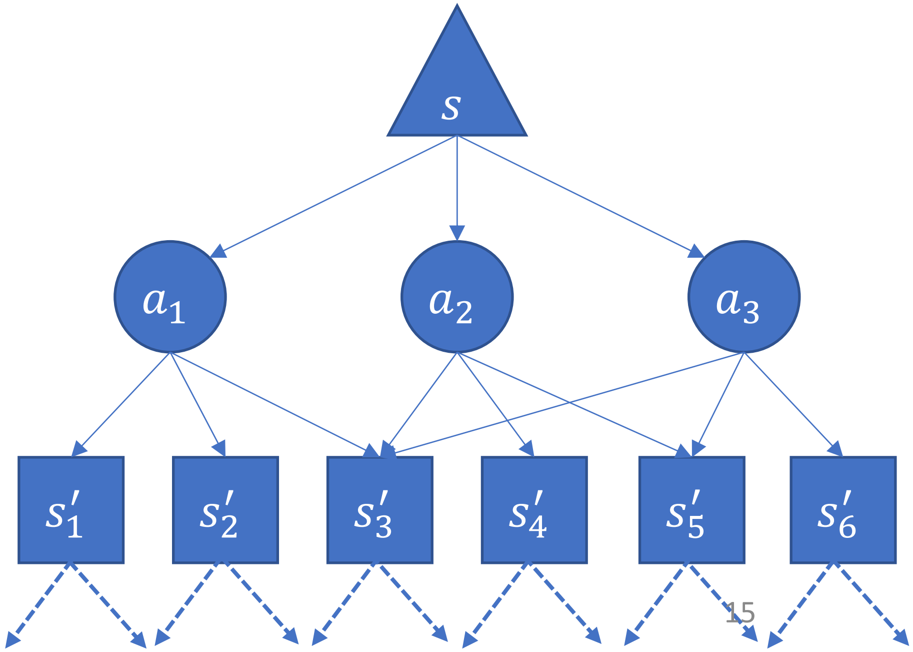
After 20 simulated days

Phase	(Queue	Approach	Wait	Speed	QLength	Avg Wait	Queue	Approach	Wait	Speed	QLength	Avg Wait)	{FULL	PART	NO	PERM}	
E/W	0.93	0.0	0.57	0.25	0.18	0.52	0.5	0.0	0.21	0.32	0.22	0.08	1.22	0.0	0.0	0.0	4.59
E/EL	0.6	0.0	0.59	0.03	0.46	0.2	0.64	0.0	0.43	0.01	0.24	0.32	1.98	0.0	0.0	0.0	6.95
W/WL	0.14	0.0	0.15	0.07	0.03	0.0	0.35	0.06	0.93	0.02	0.25	0.09	1.9	0.0	0.0	0.0	3.97
EL/WL	0.18	0.92	0.02	0.11	0.1	0.18	0.97	0.38	0.5	0.0	0.62	0.54	1.23	0.0	0.0	0.0	5.55
NPL/SPL	0.0	0.01	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.35	0.0	0.0	0.0	0.0	0.0	2.11	0.97
NPL/S	0.0	0.04	0.0	0.04	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.94	2.09
N/NL	0.0	0.82	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.99	1.63
N/SPL	0.0	0.01	0.0	0.65	0.0	0.0	0.0	0.0	0.0	0.38	0.0	0.0	0.0	1.23	0.0	0.0	1.27
N/S	0.0	0.08	0.0	0.72	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
S/SL	0.0	0.1	0.0	0.7	0.0	0.0	0.0	0.99	0.0	0.51	0.0	0.0	0.0	0.0	1.38	0.0	3.18
NL/SL	0.0	0.03	0.0	0.0	0.0	0.0	0.0	0.99	0.0	0.06	0.0	0.0	0.0	0.0	0.0	1.85	2.03

MDP - Objective

A set of states $s \in \mathcal{S}$
A set of actions $a \in \mathcal{A}$
State-transition probabilities $P(s'|s,a)$
A reward function $R(s,a,s')$

- Compute a policy: what action to take at each state
  - $\pi : S \rightarrow A$
- Compute the **optimal** policy: maximum expected reward, π^*
- $\pi^*(s) =?$
- $= \underset{a}{\operatorname{argmax}}\left[\sum_{s'} P(s'|s,a) R(s,a,s')\right]$
  - Must also optimize over the future (next steps)
- $= \underset{a}{\operatorname{argmax}}\left[\sum_{s'} P(s'|s,a) \left(R(s,a,s') + \underbrace{\mathbb{E}_{\pi^*}[G|s']}_{v^*(s')}\right)\right]$



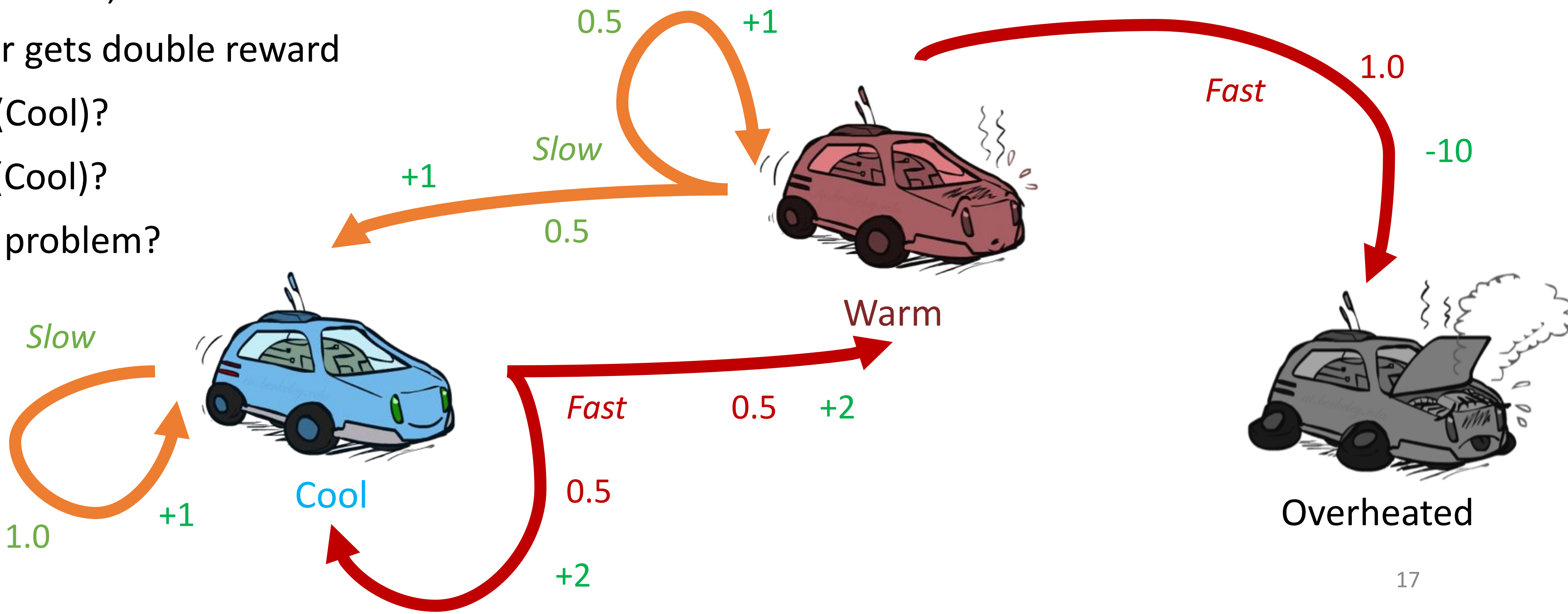
Notation

- π^* - a policy that yields the maximal expected sum of rewards
- G- observed sum of rewards, i.e., $\sum r_t$
- $v^*(s)$ - the expected sum of rewards from being at s then following π^*
  - $= \mathbb{E}_{\pi^*}\ [G|s]$

Race car example

$$\max_a \left[\sum_{s'} P(s'|s,a) \left(R(s,a,s') + v^*(s')\right) \right]$$

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward
- What is $V^*(\text{Cool})$?
- What is $\pi^*(\text{Cool})$?
- What's the problem?



Discount factor

- As the agent traverse the world it receives a sequence of rewards
- Which sequence has higher utility?
  - $\tau_1 = +1, +1, +1, +1, +1, +1\ldots$
  - $\tau_2 = +2, +2, +2, +2, +2, +2\ldots$

$$\sum_{t=0}^{\infty} 1 = \sum_{t=0}^{\infty} 2 = \infty$$

- Let's decay future rewards exponentially by a factor, $0 \leq \gamma < 1$

$$\sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$$

Geometric series

- Now τ_2 yields higher utility than τ_1

Discounting

- It’s reasonable to maximize the sum of rewards
- It’s also reasonable to prefer rewards now to rewards later
- Discount factor: values of rewards decay exponentially

1

Worth Now

γ

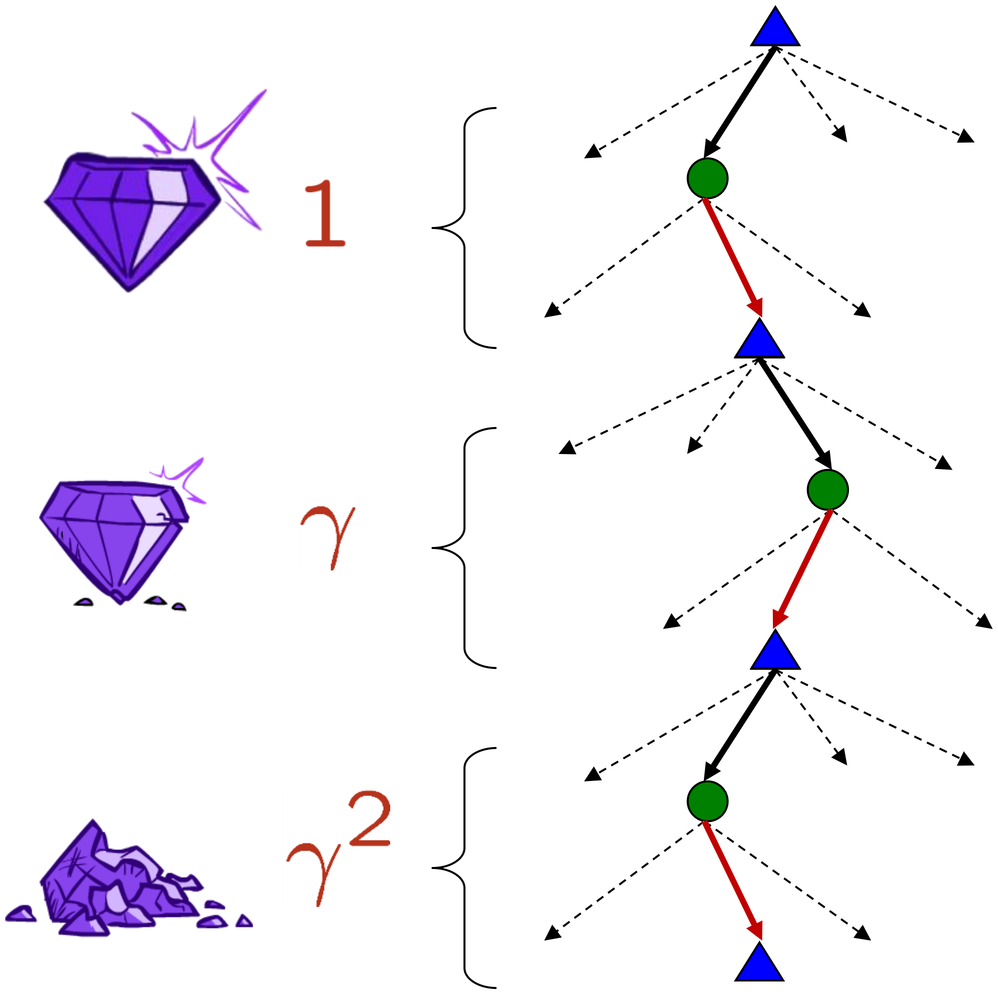
Worth Next Step

γ^2

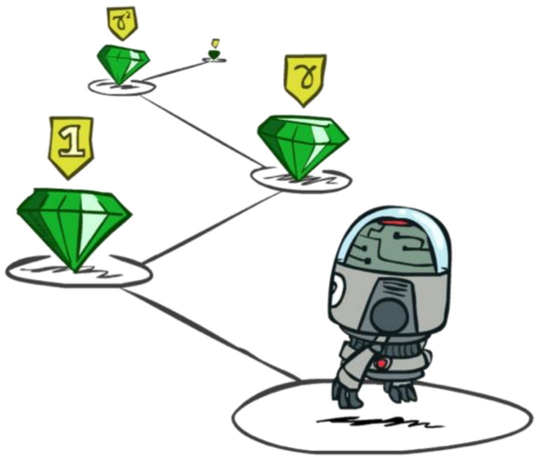
Worth In Two Steps

Discounting

- How to discount?
  - Each time we descend a level, we multiply by the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge
- Example: discount of 0.5
  - G(r=[1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - G([1,2,3]) < G([3,2,1])



Quiz: Discounting



- Given grid world:

10				1
a	b	c	d	e

  - Actions: East, West, and Exit (‘Exit’ only available in terminal states: a, e)
  - Rewards are given only after an exit action
  - Transitions: deterministic
- Quiz 1: For $\gamma = 1$, what is the optimal policy?

10				1

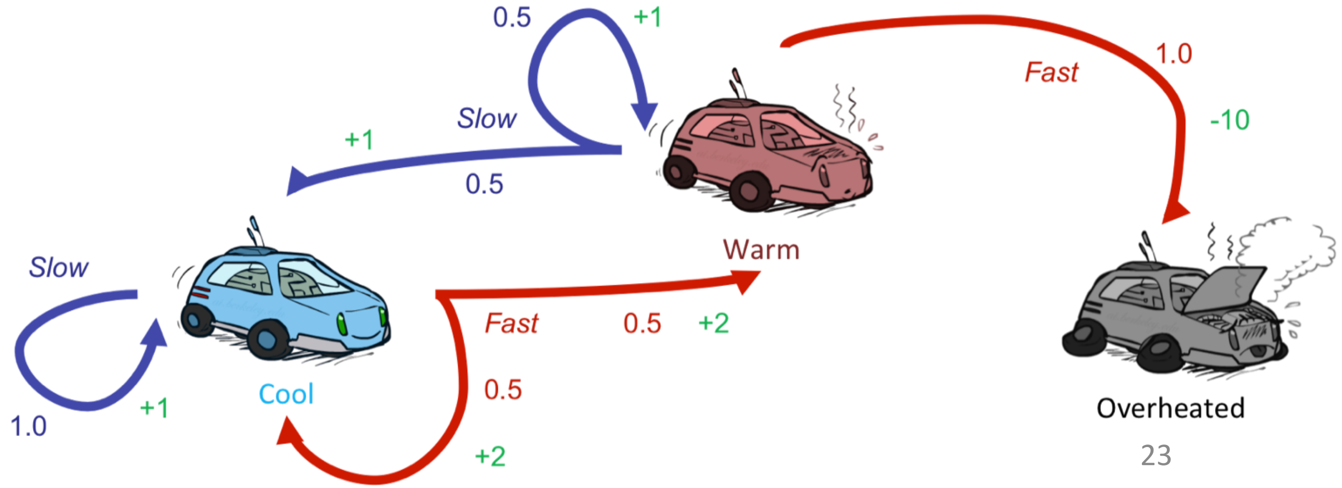
- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

10				1

- Quiz 3: For which γ are West and East equally good when in state d?

Race car example

- Consider a discount factor, $\gamma = 0.9$
- What is $v^*(Cool)$
- $= \max_a \left[r(s,a) + \sum_{s'} p(s'|s,a)\ \gamma\ v^*(s') \right]$
- $= \max[1 + 0.9 \cdot 1 v^*(Cool), 2 + 0.9 \cdot 0.5 v^*(Cool) + 0.9 \cdot 0.5 v^*(Warm)]$
    - Computing…
    - …Stack overflow
- Work in iterations



Value iteration

Value iteration

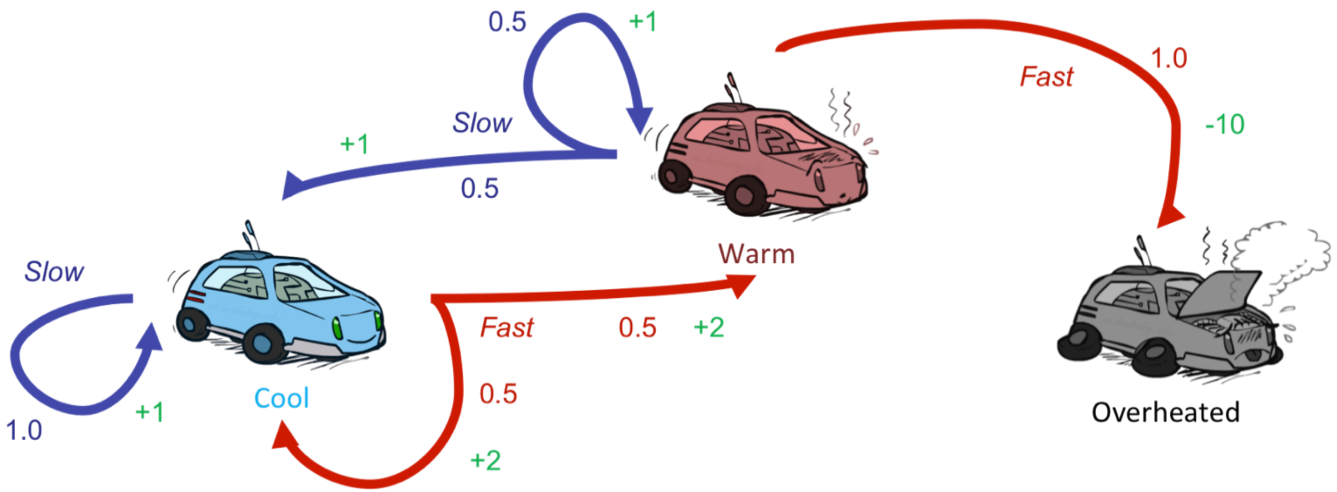
Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$

Value Iteration

V_0	0	0	0
V_1	2	1	0
V_2	3.35	2.35	0



$\gamma = 0.9$

Bellman operator

$$V_{k+1}(s) \leftarrow \max_a \left[r(s,a) + \sum_{s'} p(s'|s,a)\ \gamma V_k(s') \right]$$

$$= \max_a \left[\sum_{s'} p(s'|s,a)\ (r(s,a,s') + \gamma V_k(s')) \right]$$

Value iteration – convergence

- **Lemma**: the Bellman operator, $B(.)$, is a *contraction mapping*
  - For any state values assignment, V_1, V_2:
    - $\max_{s\in\mathcal{S}} |B(V_1(s)) - B(V_2(s))| \le \gamma \max_{s\in\mathcal{S}} |V_1(s) - V_2(\mathrm{s})|$
- **Proof**:

  1. $\max_{s} |B(V_1(s)) - B(V_2(s))| = \max_{\mathrm{s}}[|\max_{\mathrm{a}}(\mathrm{r(s,a)} + \gamma \sum_{s'} P(s'|s,a)V_1(s')) - \max_{\mathrm{a}}(\mathrm{r(s,a)} + \gamma \sum_{s'} P(s'|s,a)V_2(s'))|]$
     $\le \max_{\mathrm{s}}[|\max_{\mathrm{a}}(\mathrm{r(s,a)} + \gamma \sum_{s'} P(s'|s,a)V_1(s') - [\mathrm{r(s,a)} + \gamma \sum_{s'} P(s'|s,a)V_2(s')])|]$

  - The last stage follows the fact that: $\max_{x} f(x) - \max_{x} g(x) \le \max_{x}[f(x) - g(x)]$

  2. $= \gamma \max_{\mathrm{s}} [|\max_{\mathrm{a}}(\sum_{s'} P(s'|s,a)V_1(s') - \sum_{s'} P(s'|s,a)V_2(s'))|] = \gamma \max_{\mathrm{s}} \left[\max_{\mathrm{a}}\left(\sum_{s'} P(s'|s,a)|V_1(s') - V_2(s')|\right)\right]$
     $\le \gamma \max_{\mathrm{s}}[|V_1(s) - V_2(s)|]$

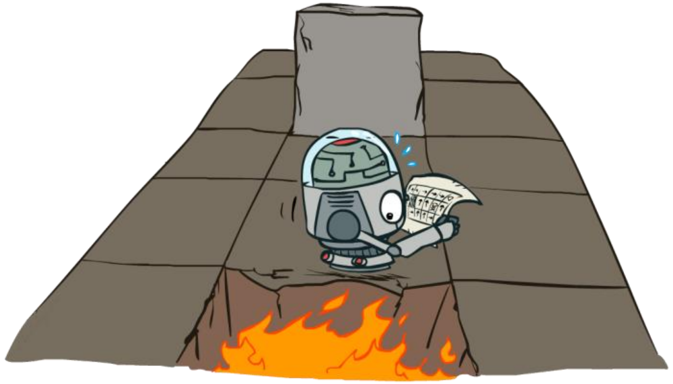
  - The last stage follows the fact that for all s', $|V_1(s') - V_2(s')| \le \max_s[|V_1(s) - V_2(s)|]$ and for any a and s, $\sum_{s'} P(s'|s,a) = 1$ and $P(s'|s,a)$ is never negative, i.e., $\sum_{s'} P(s'|s,a)|V_1(s') - V_2(s')|$ is a linear combination of values that are not larger than $\max_s[|V_1(s) - V_2(s)|]$

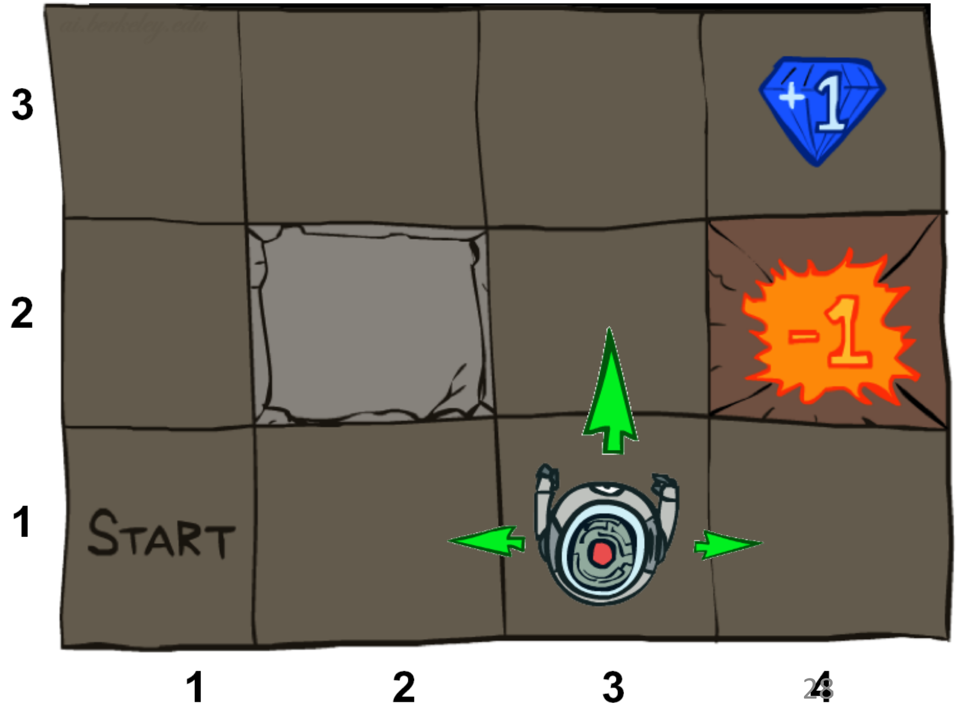
Value iteration – convergence

- **Theorem**: Value iteration converges to the true utility value
  - $V_{k\rightarrow\infty} \rightarrow V^*$
- **Proof**:
  - The Bellman operator has a fixed point at V^* i.e., $B(V^*) = V^*$
  - A *contraction mapping* has at most one fixed point. Moreover, the Banach fixed-point theorem states that every contraction mapping on a nonempty complete metric space (M) has a unique fixed point, and that for any x in M the iterated function sequence x, f (x), f (f (x)), f (f (f (x))), ... converges to the fixed point

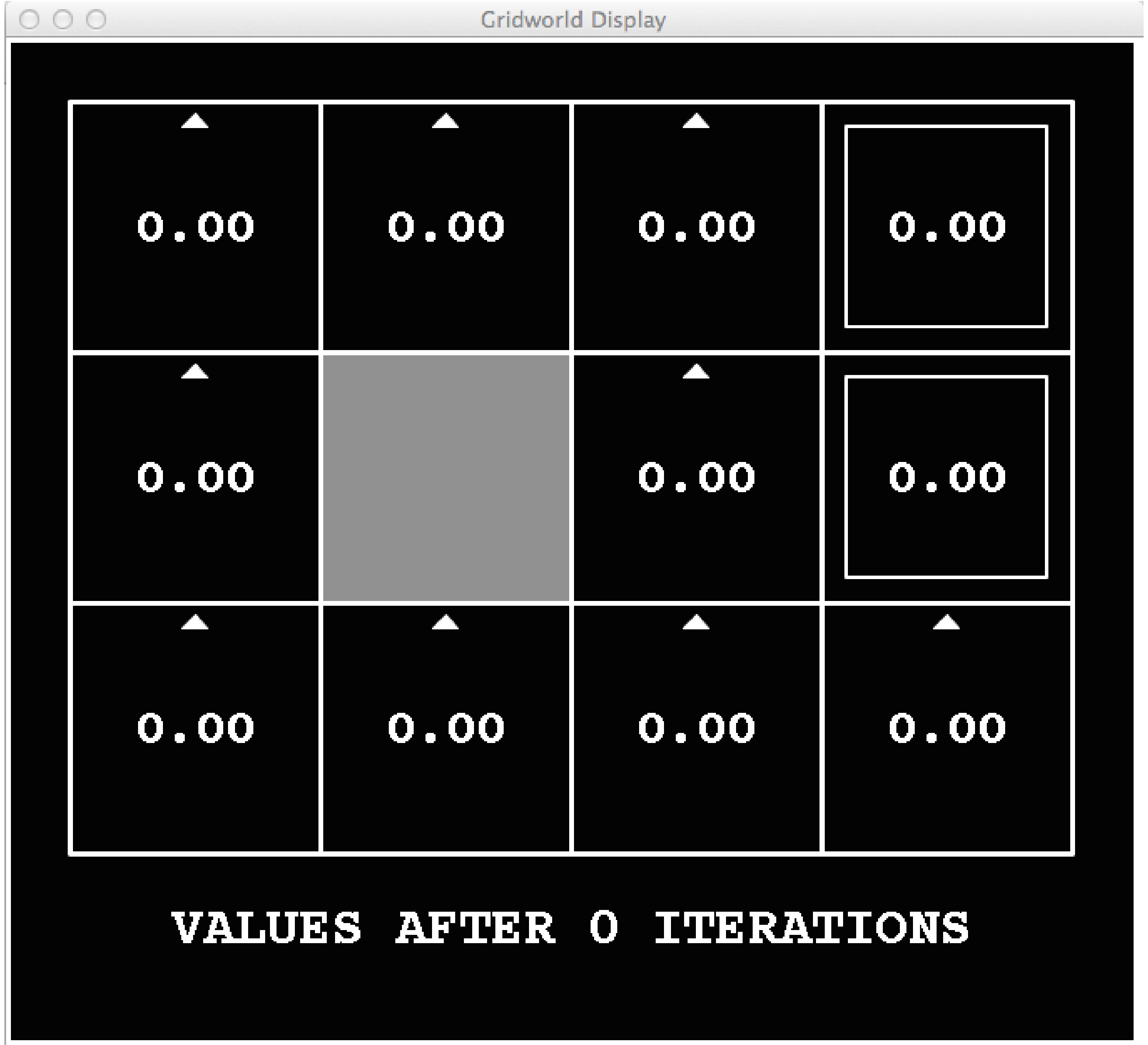
Example: Grid World

- 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 taken, the agent stays put
- The agent receives rewards each time step
  - Small negative reward each step (battery drain)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of (discounted) rewards



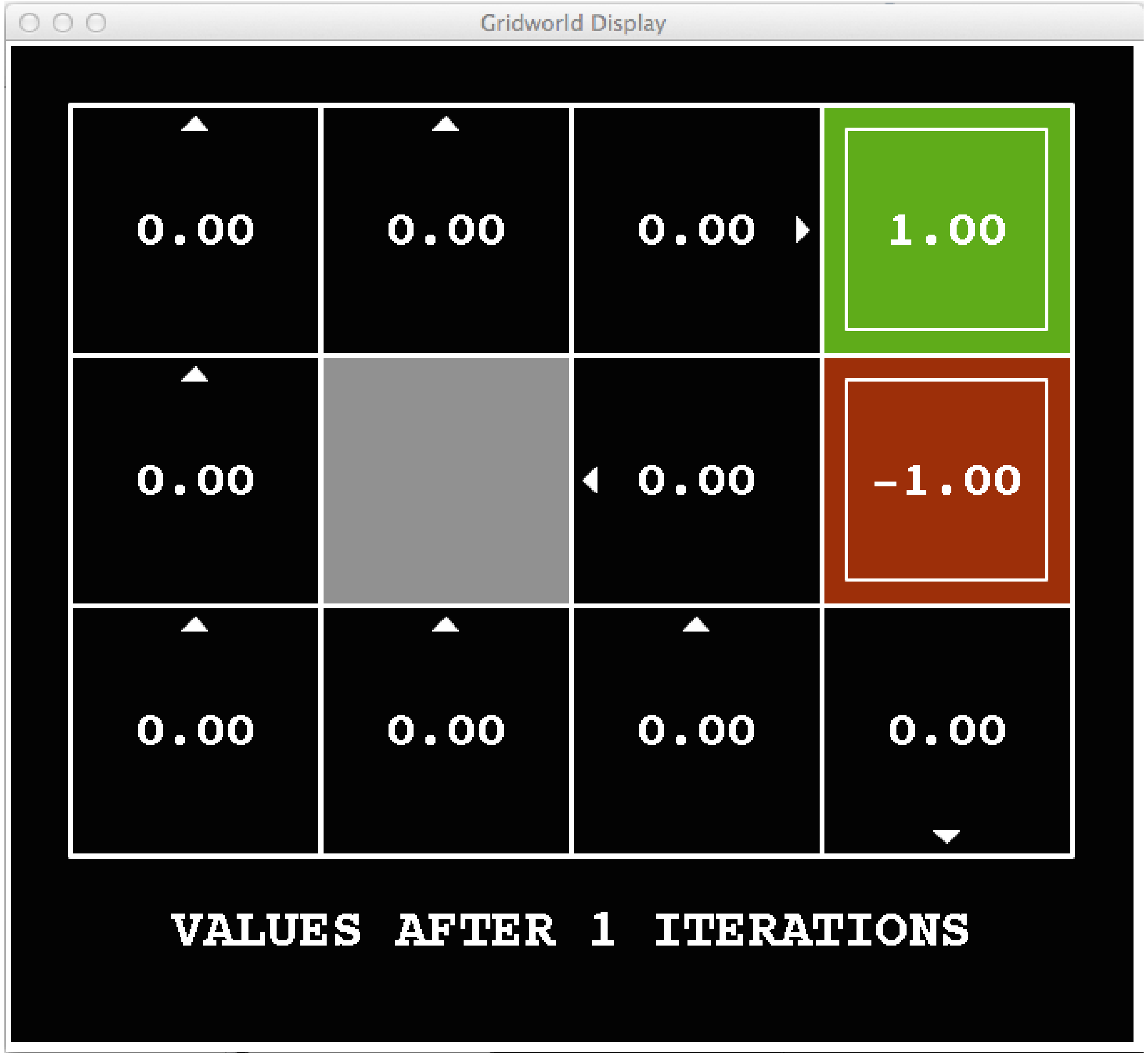


k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

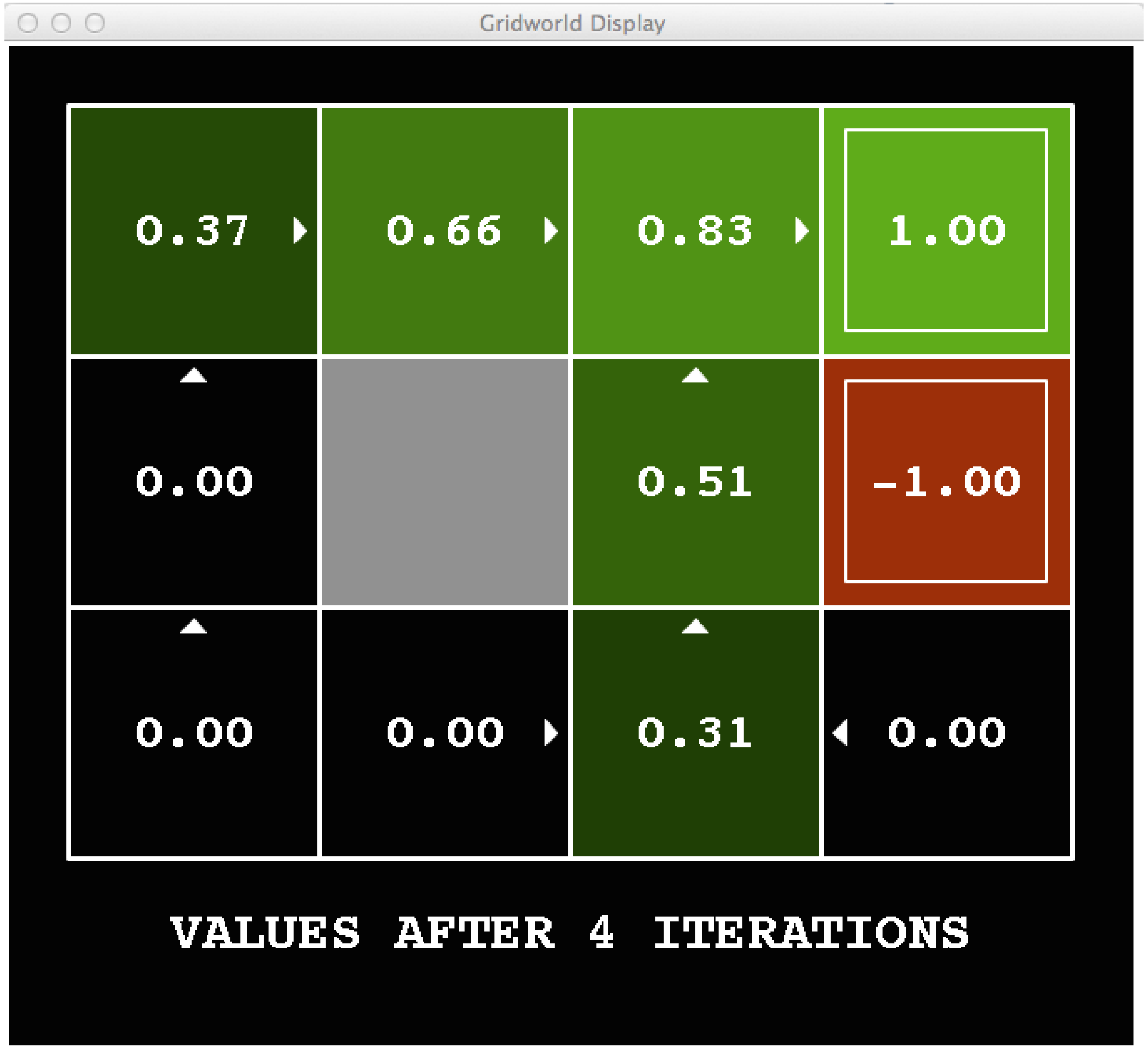
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

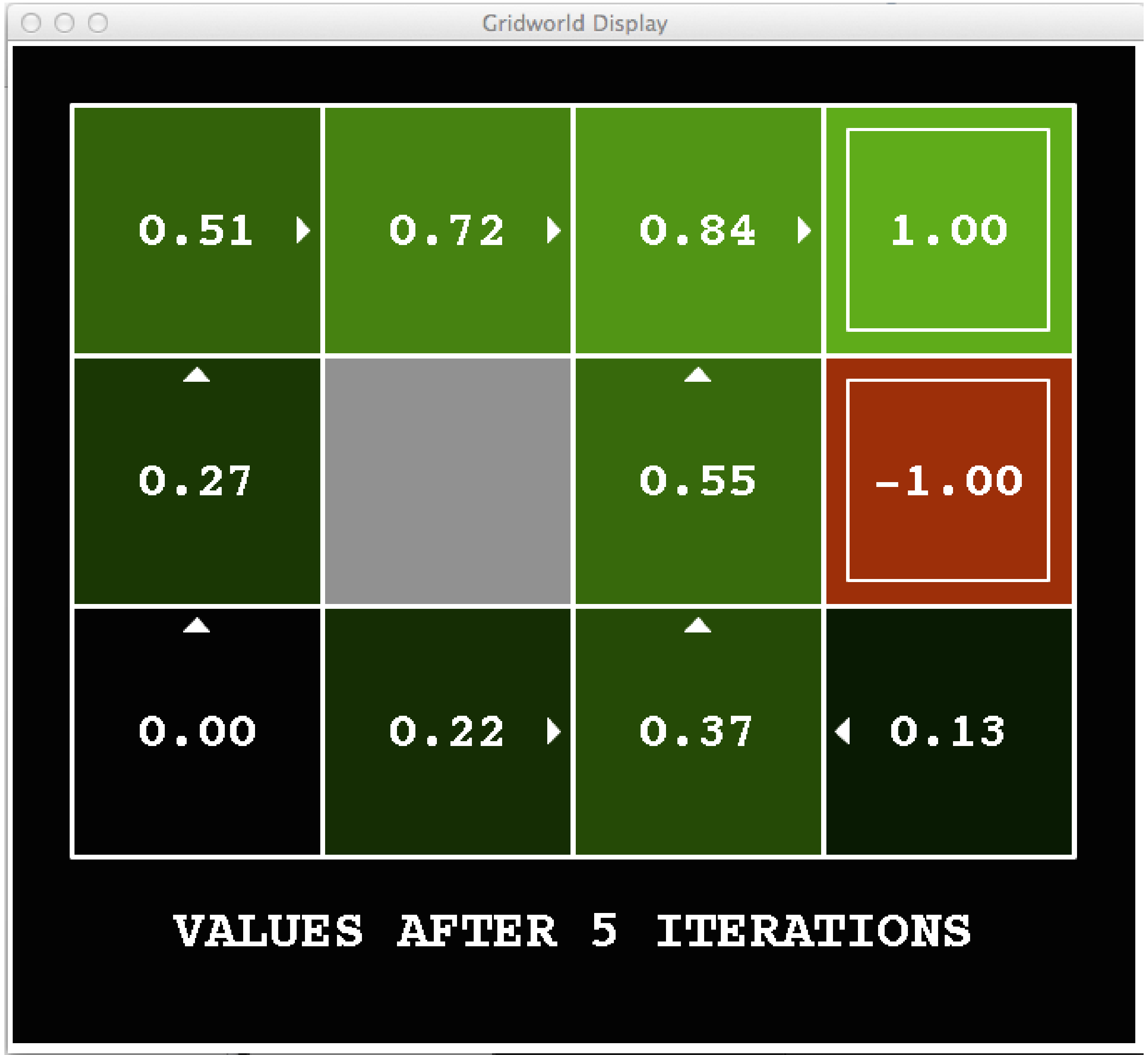
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



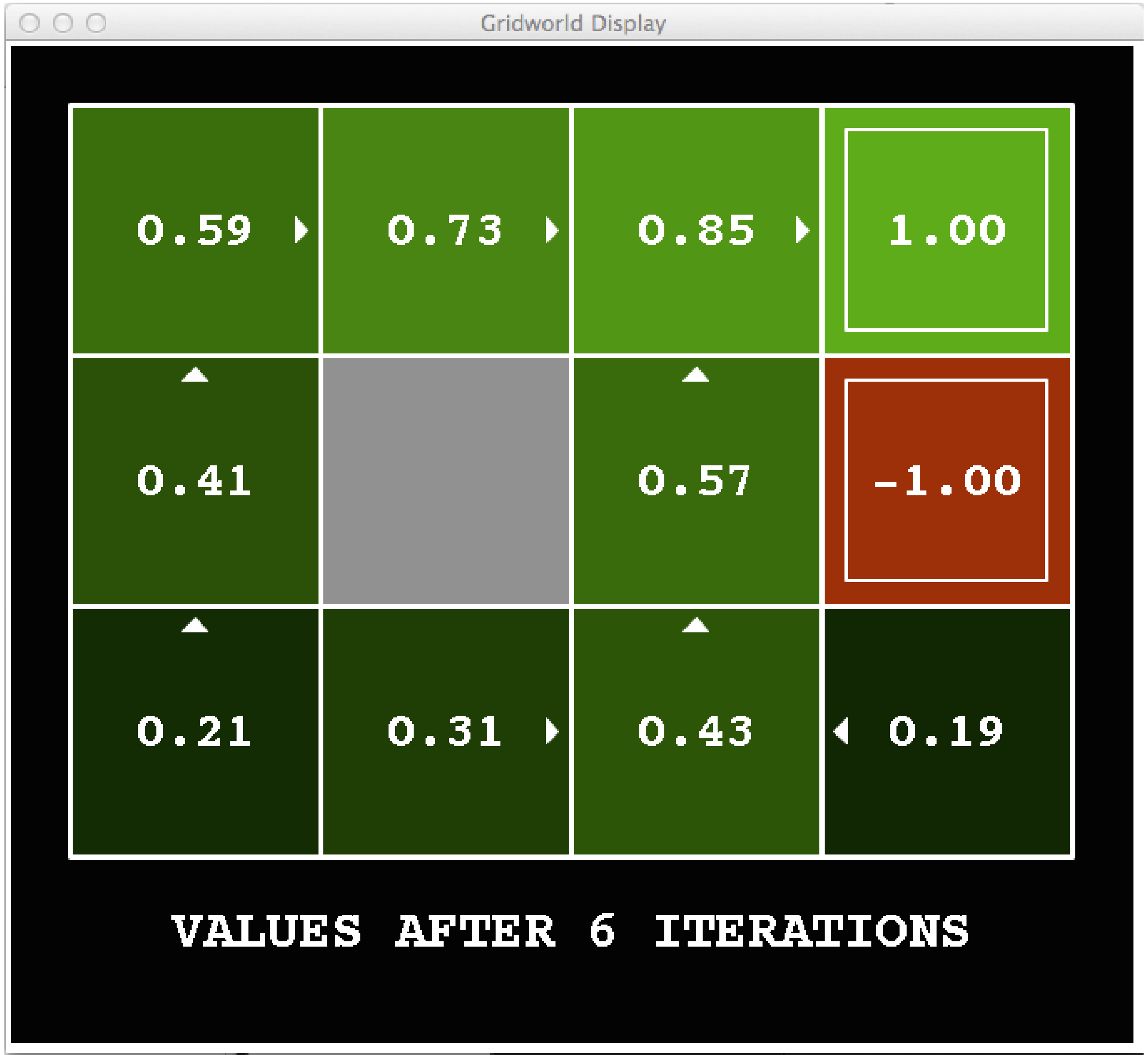
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



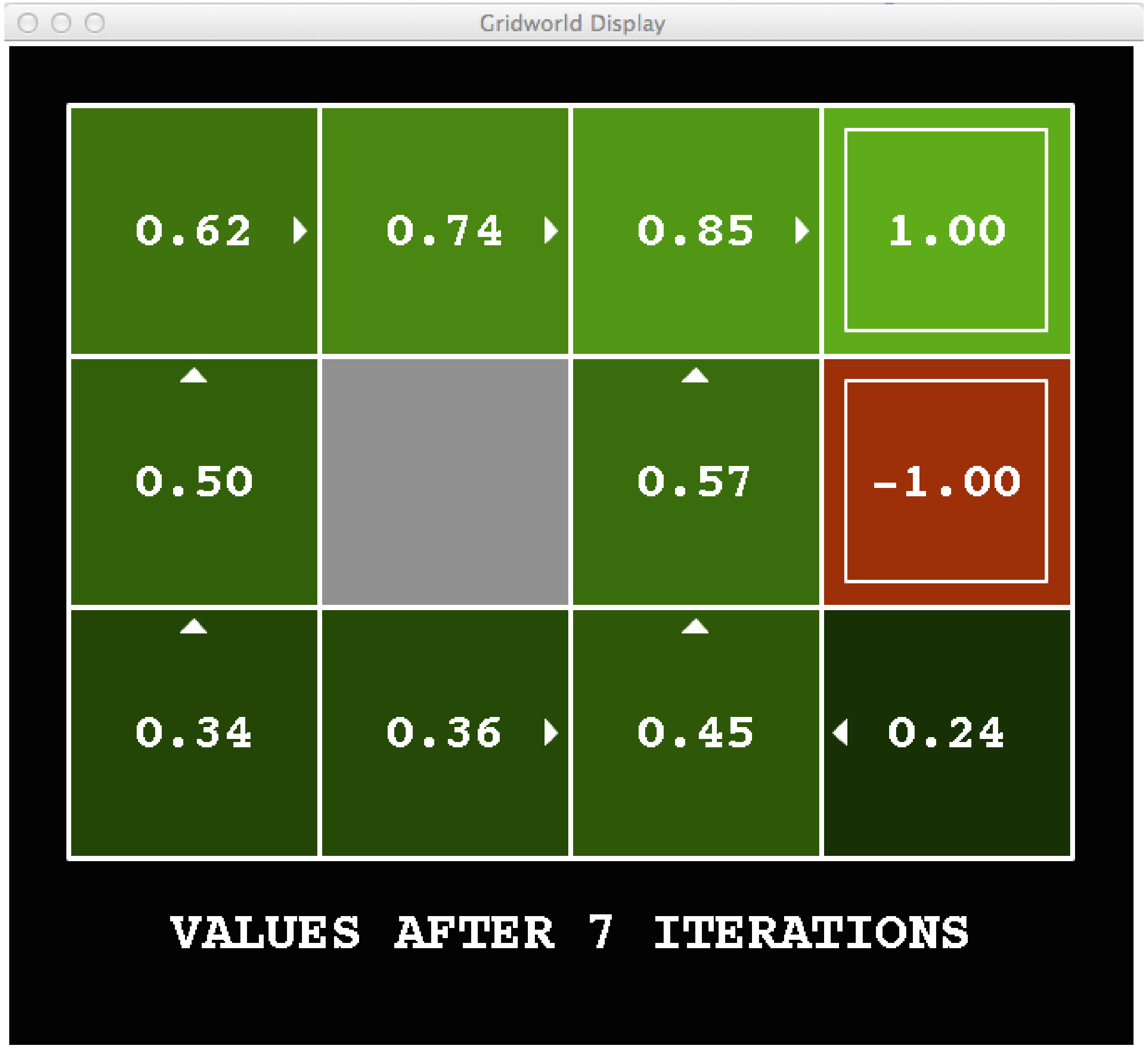
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



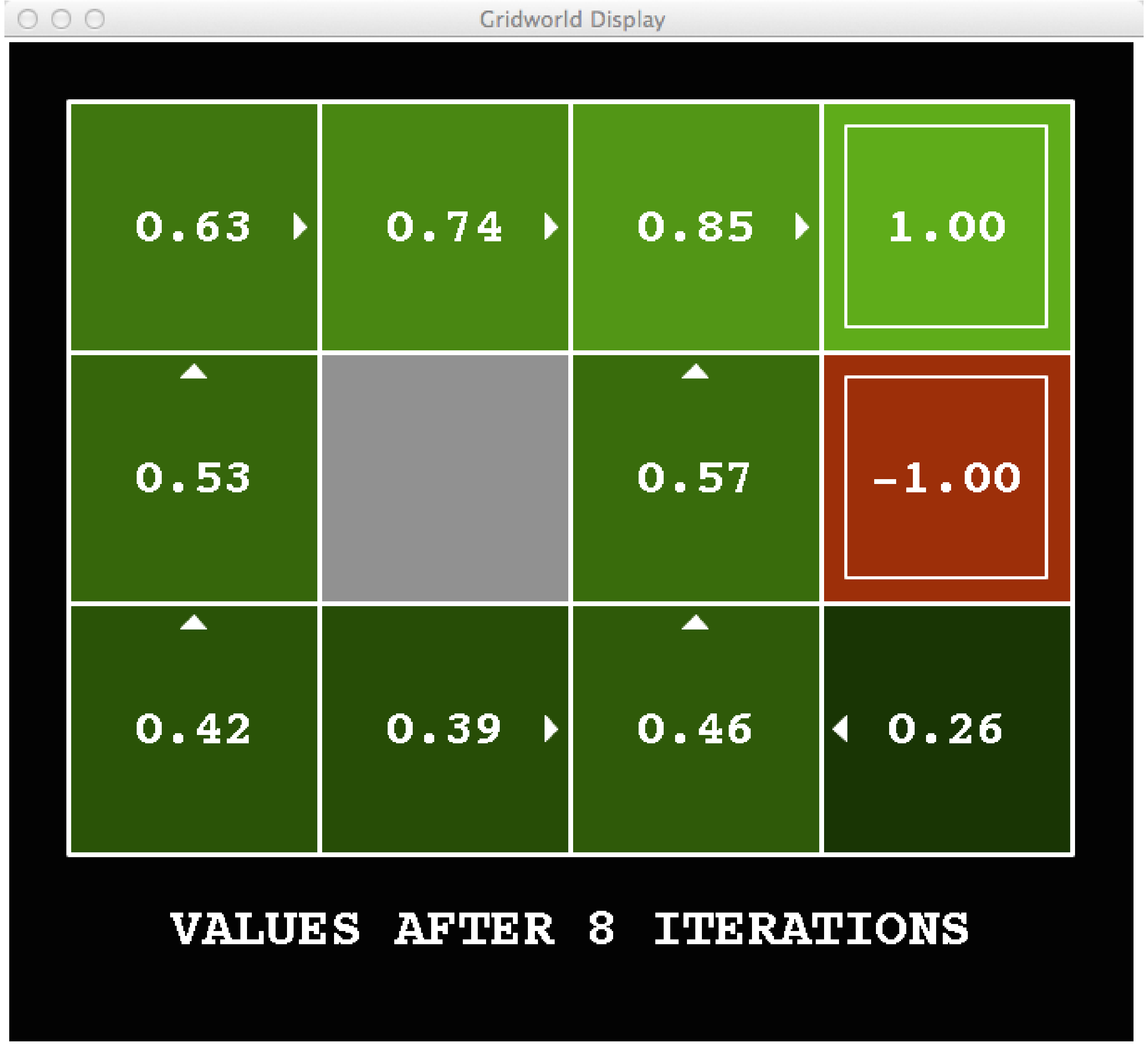
Noise = 0.2
Discount = 0.9
Living reward = 0

k=7



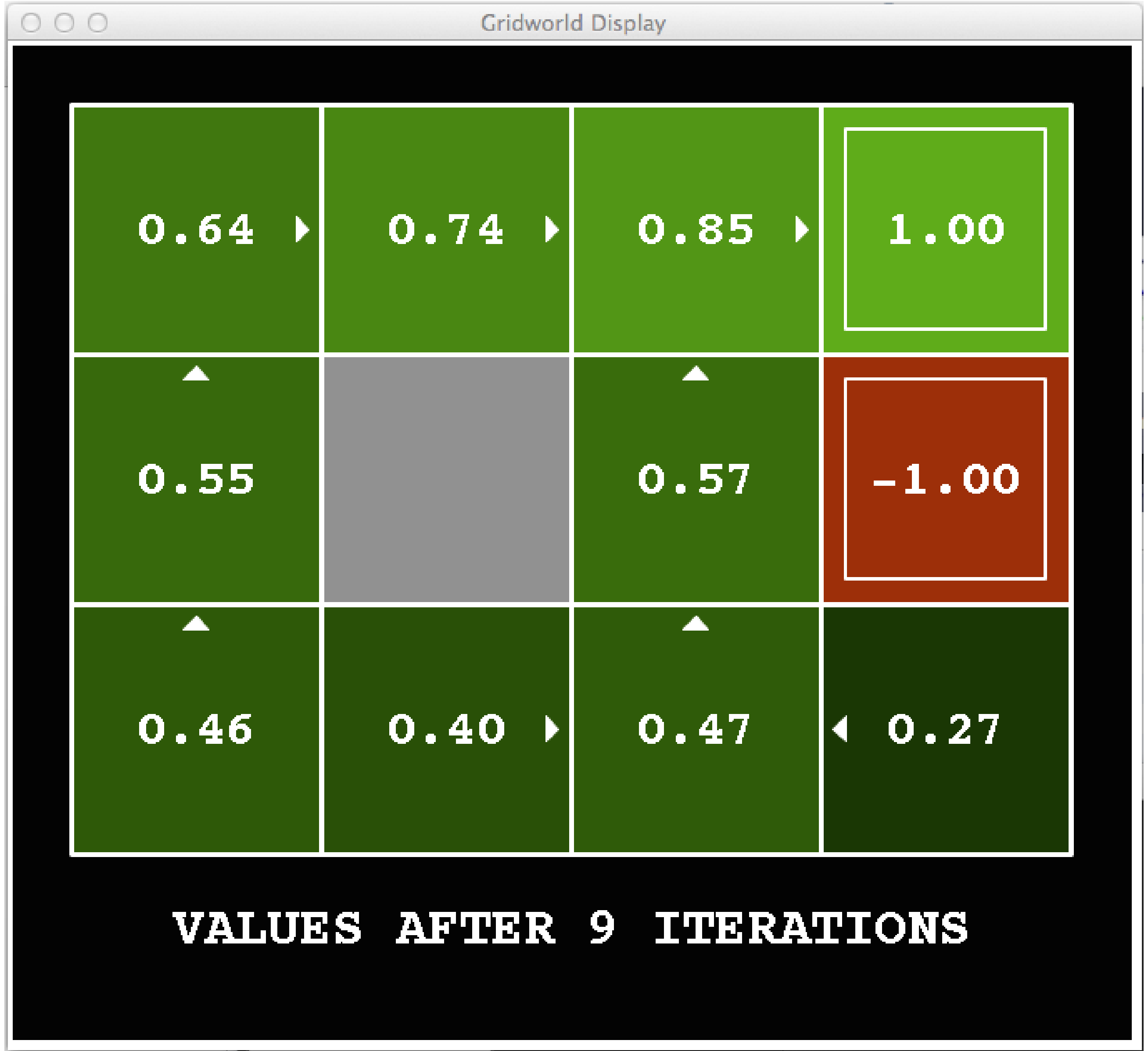
Noise = 0.2
Discount = 0.9
Living reward = 0

k=8



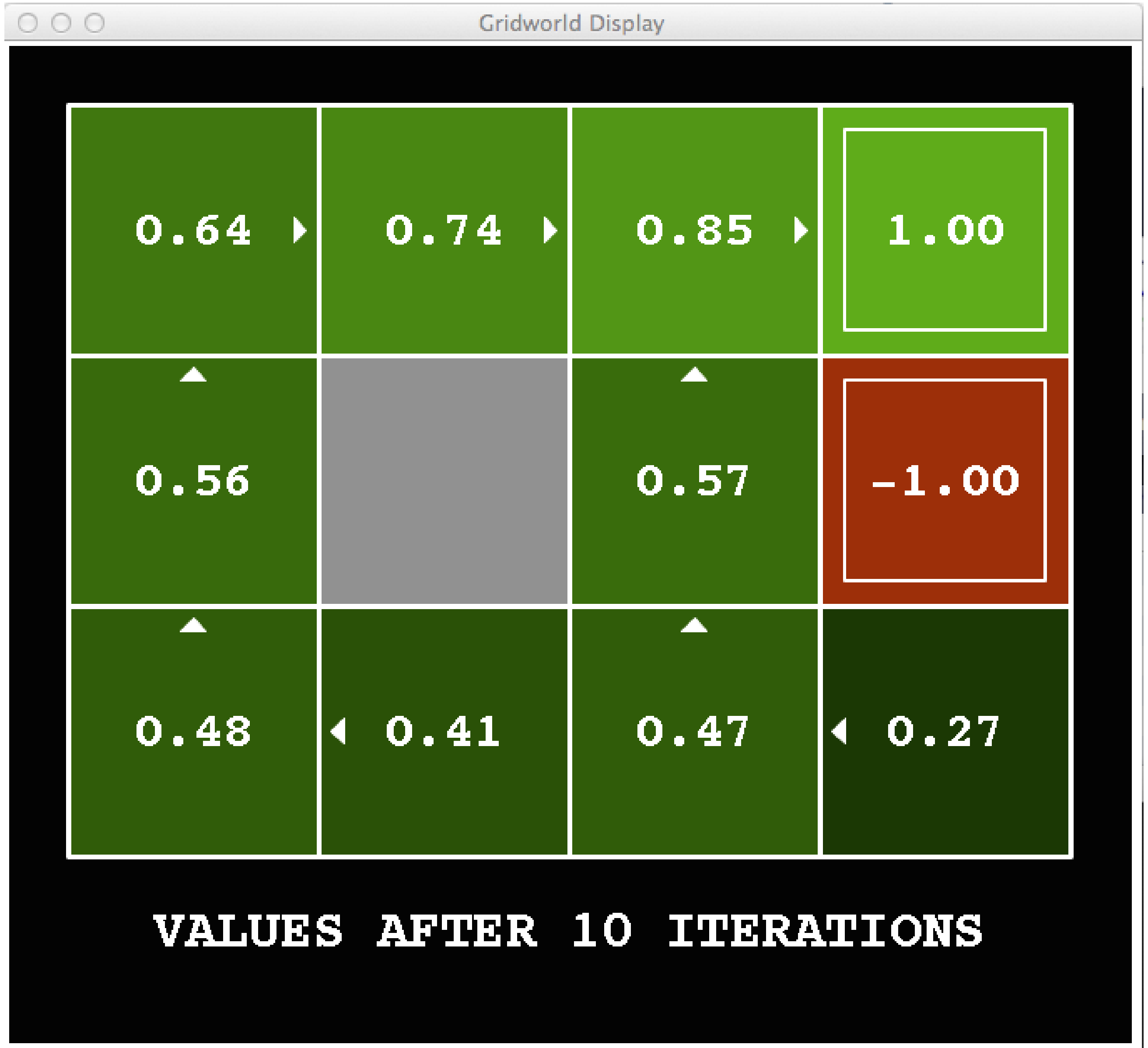
Noise = 0.2
Discount = 0.9
Living reward = 0

k=9



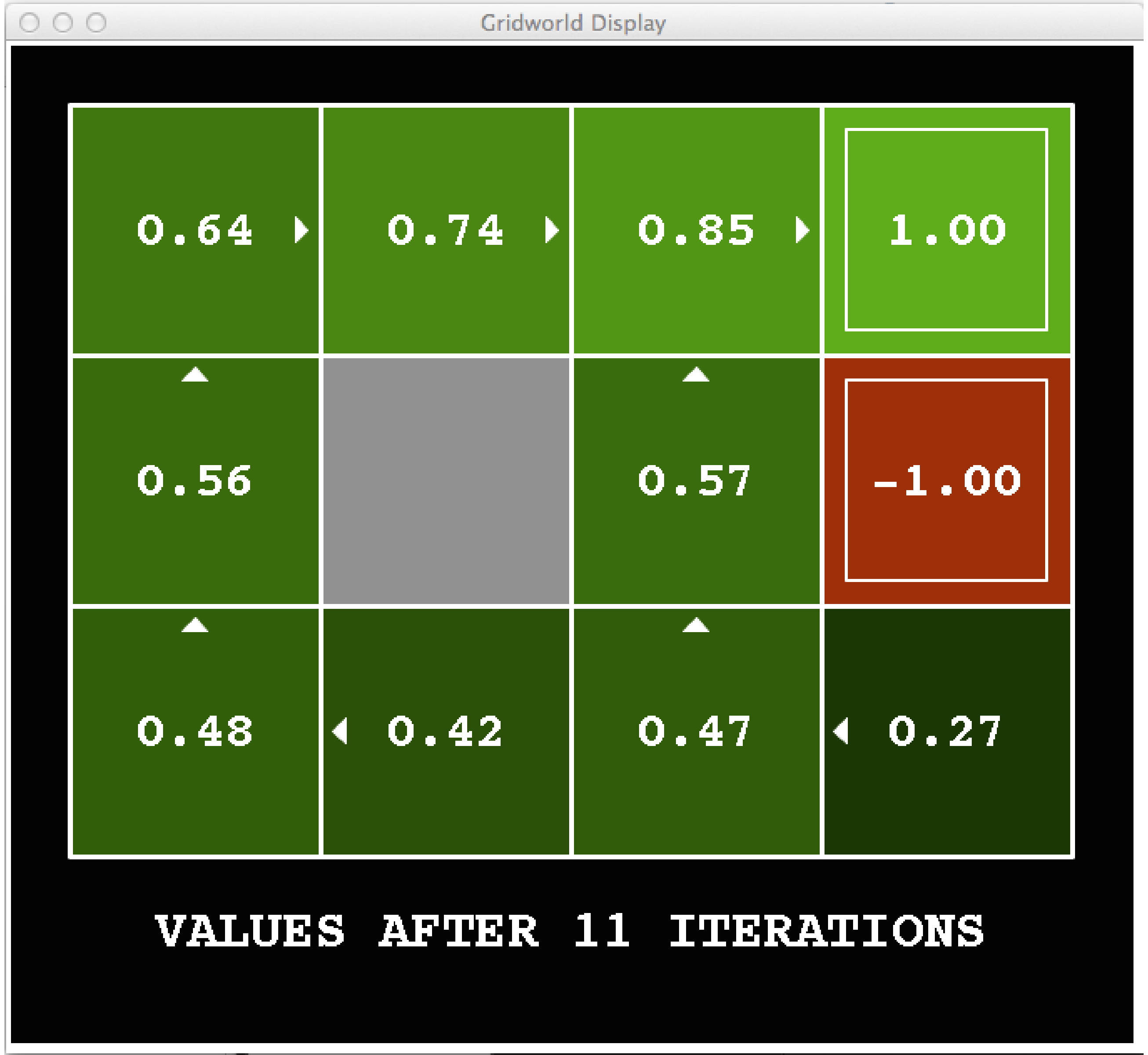
Noise = 0.2
Discount = 0.9
Living reward = 0

k=10



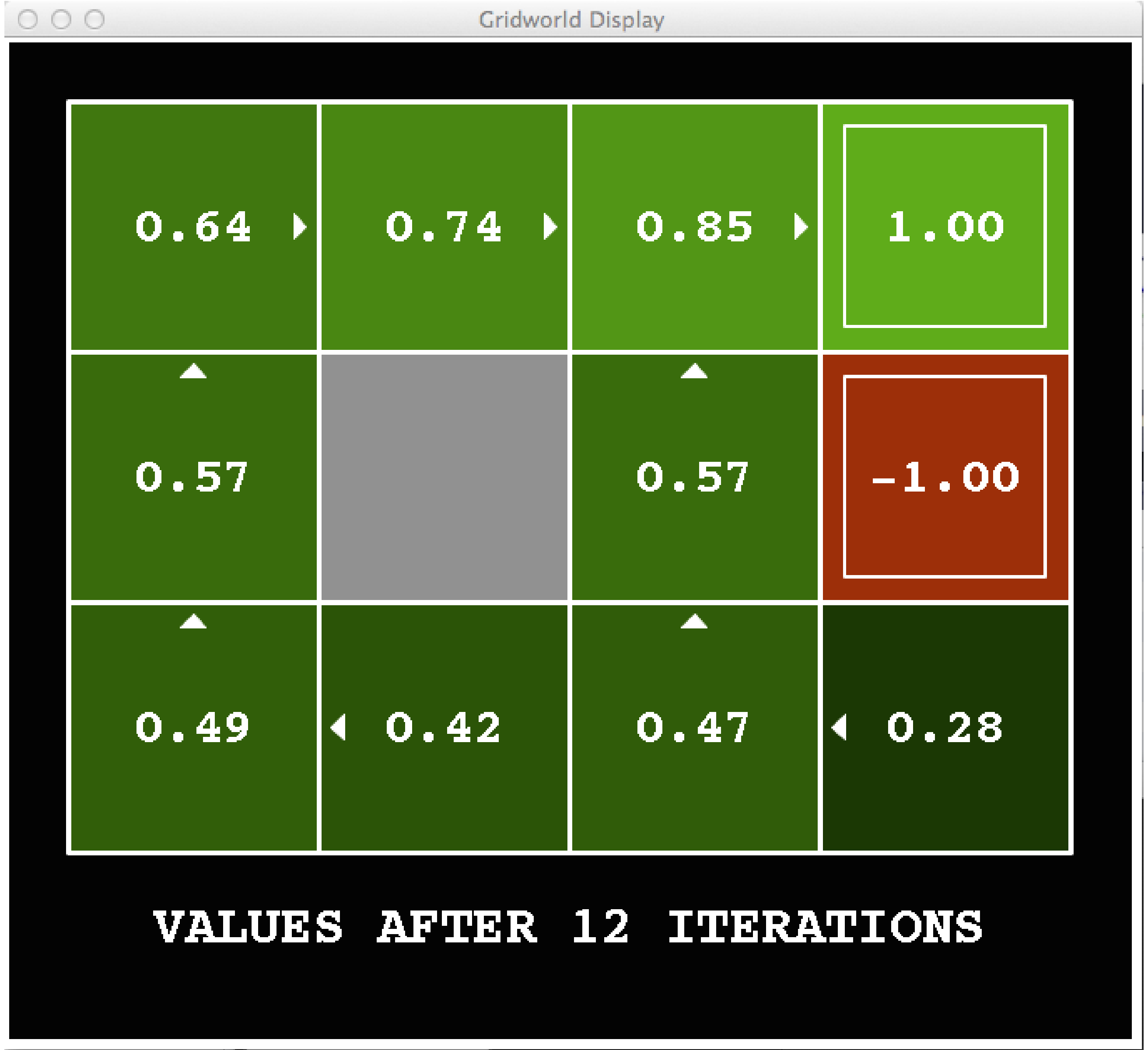
Noise = 0.2
Discount = 0.9
Living reward = 0

k=11



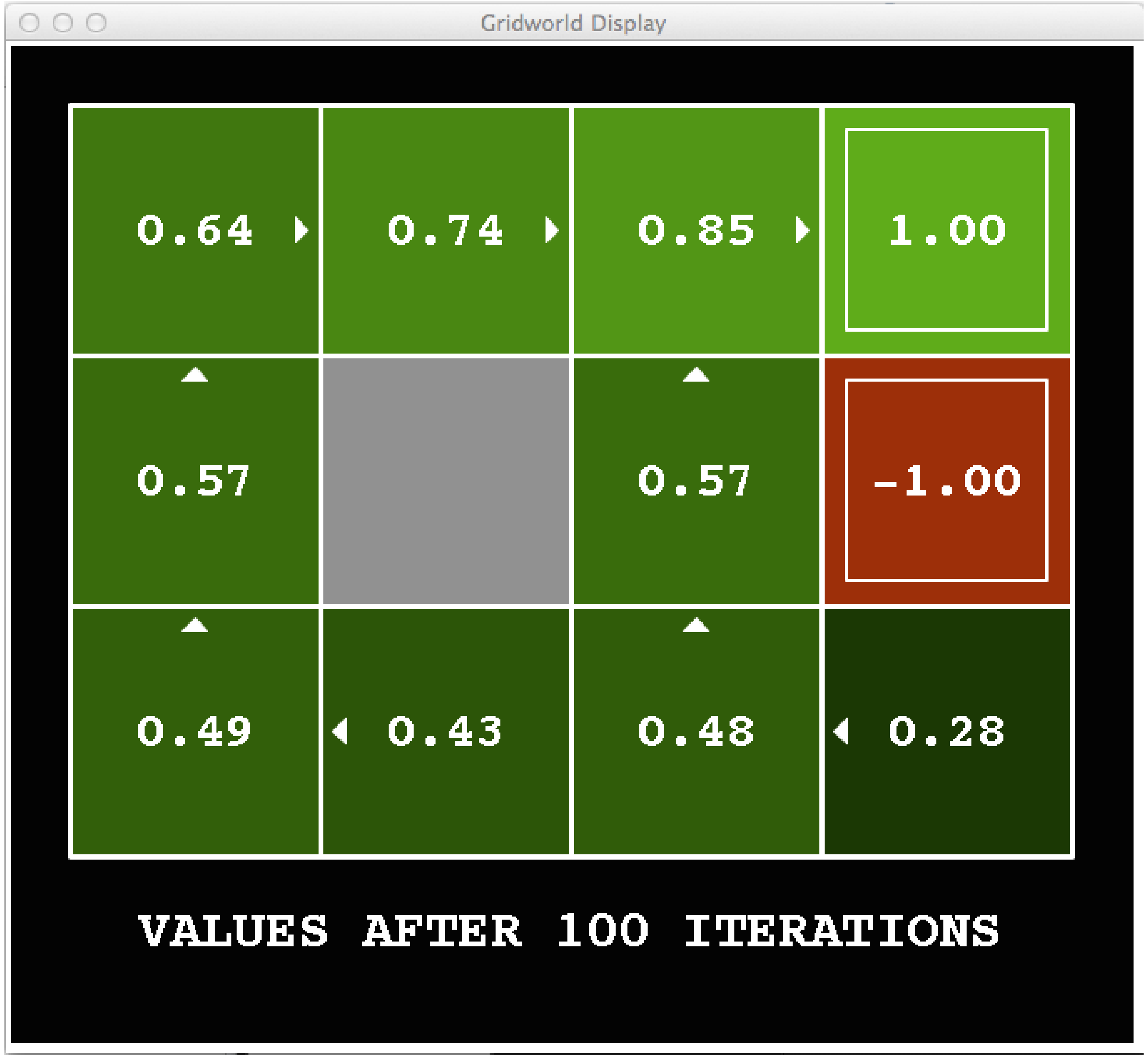
Noise = 0.2
Discount = 0.9
Living reward = 0

k=12



Noise = 0.2
Discount = 0.9
Living reward = 0

k=100



Noise = 0.2
Discount = 0.9
Living reward = 0

Drawbacks of Value Iteration

- Value iteration repeats the Bellman updates:
- $V_{k+1}(s) \leftarrow \max_a \left[R(s,a) + \sum_{s'} P(s'|s,a)\ \gamma V_k(s') \right]$

$$= \max_a \left[\sum_{s'} P(s'|s,a)\ (R(s,a,s') + \gamma V_k(s')) \right]$$

- Issue 1: It's slow – O(S^2A) per iteration
  - Do we really need to update every state at every iteration?
- Issue 2: A policy cannot be easily extracted
  - Policy extraction requires another O(S^2A)
- Issue 3: The policy often converges long before the values
  - Can we identify when the policy converged?
- Issue 4: requires knowing the model, $P(s'|s,a)$, and the reward function, $R(s,a)$
- Issue 5: requires discrete (finite) set of actions
- Issue 6: infeasible in large state spaces

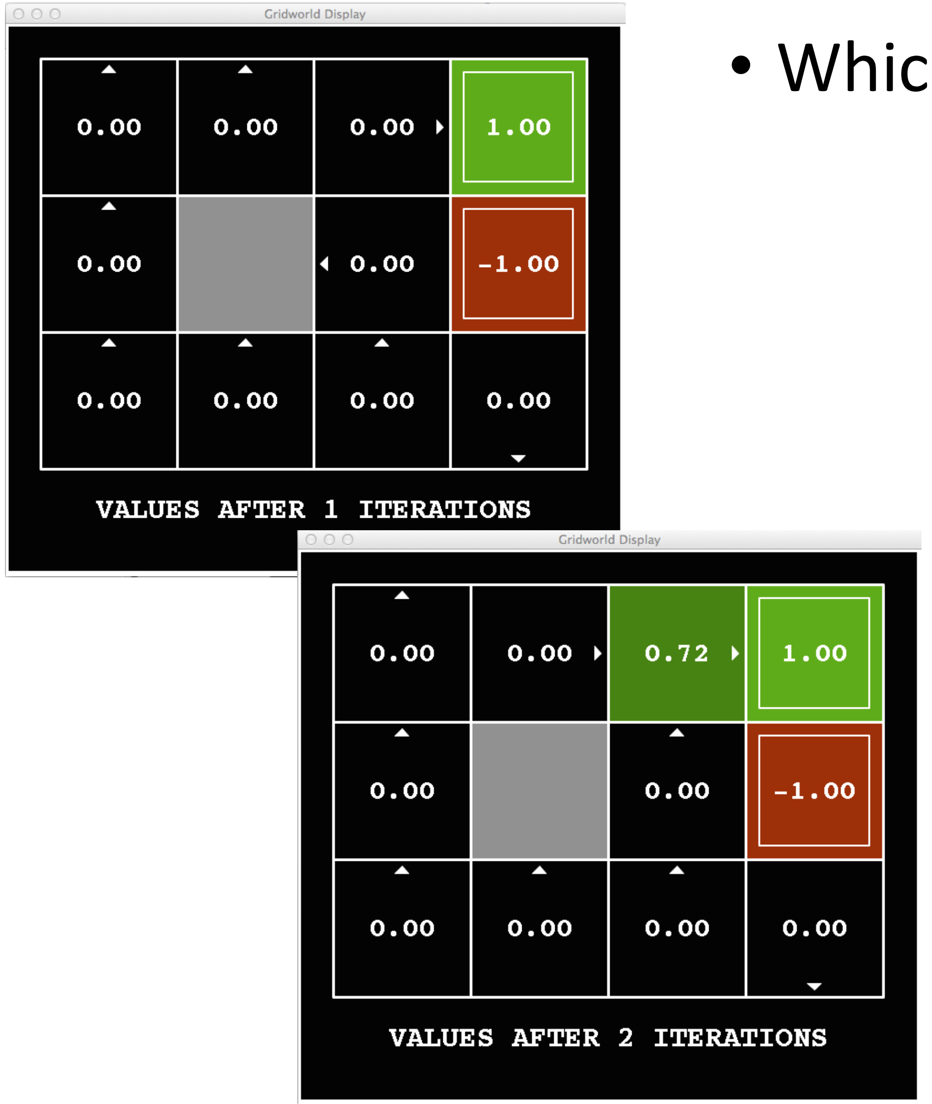
Solutions (briefly, more later…)

- Issue 1: It's slow – $O(S^2A)$ per iteration
  - Asynchronous value iteration
- Issue 2: A policy cannot be easily extracted
  - Learn q (action) values
- Issue 3: The policy often converges long before the values
  - Policy-based methods
- Issue 4: requires knowing the model and the reward function
  - Reinforcement learning
- Issue 5: requires discrete (finite) set of actions
  - Policy gradient methods
- Issue 6: infeasible for large (or continues) state spaces
  - Function approximators

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

- Asynchronous value iteration
- In value iteration, we update every state in each iteration
- Actually, *any* sequences of Bellman updates will converge if every state is visited infinitely often regardless of the visitation order
- Idea: prioritize states whose value we expect to change significantly

Asynchronous Value Iteration



- Which states should be prioritized for an update?

A single update per iteration

Algorithm 3 Prioritized Value Iteration

repeat
  $s \leftarrow \arg\max_{\xi \in S} H(\xi)$
  $V(s) \leftarrow \max_{a \in A} \{R(s,a) + \gamma \sum_{s' \in S} Pr(s'|s,a) V(s')\}$
  for all $s' \in SDS(s)$ **do**
    // recompute $H(s')$
  end for
until convergence

$$SDS(s) = \{s' \colon \exists a, p(s|s',a) > 0\}$$

For the home assignment set:

$$H(s) = \left| V(s) - \max_{a} \left\{ R(s,a) + \gamma \sum_{s'} \Pr(s'|s,a) V(s') \right\} \right|$$

Double the work?

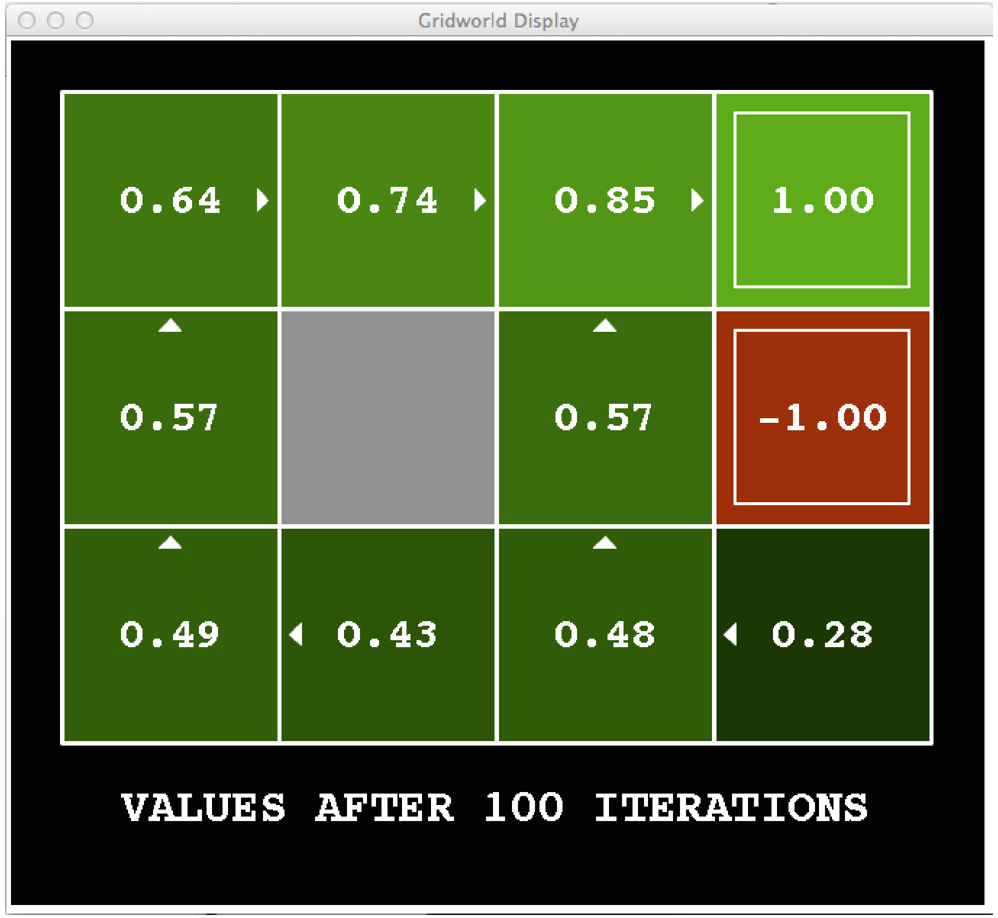
For the home assignment set:

$$H(s\quad) = \left| V(s) - \max_{\mathrm{a}} \left\{ R(s\quad, a) + \gamma \sum_{s'} \Pr(s'|s\quad, a) V(s') \right\} \right|$$

- Computing priority is similar to updating the state value (computational effort)
- Why double work?
  - If we computed the priority, we can go ahead and update the value for free
- Notice that we don’t need to update the priorities for the entire state space
- For many of the states the priority doesn’t change following an updated value for a single state s
- Only states s’ with $\sum_a p(s|s', a) > 0$ might be updated

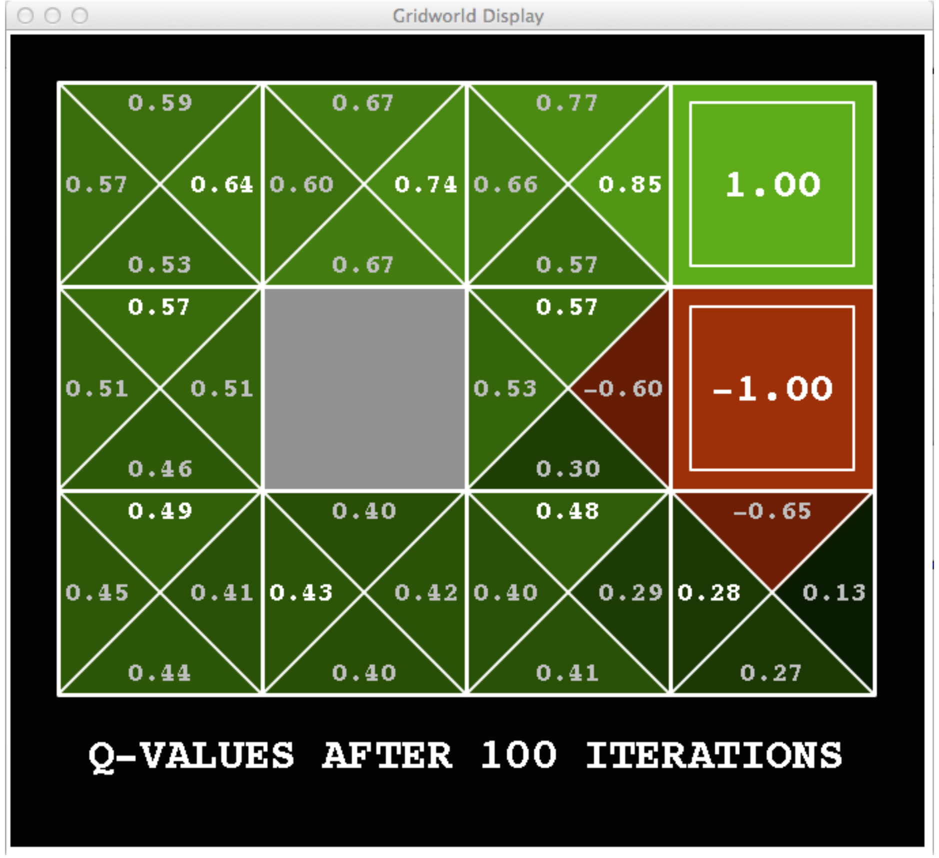
Issue 2: A policy cannot be easily extracted

- Given state values, what is the appropriate policy?
  - $\pi(s) \leftarrow \underset{a}{\operatorname{argmax}}\left[R(s,a) + \sum_{s'} P(s'|s,a)\ \gamma V_k(s')\right]$
  - Requires another full value sweep: $O(S^2 A)$
- Learn q (action) values instead
- $Q^*(s,a)$ - the expected sum of rewards from being at s, taking action a and then following π^*



Q-learning

- $Q^*(s, a)$ - the expected sum of rewards from being at s, taking action a and then following π^*
- $\pi^*(s) \leftarrow \underset{a}{\operatorname{argmax}}[Q^*(s, a)]$
- Can we learn Q values with dynamic programming?
    - Yes, similar to value iteration



Q-learning with value iteration

- $V^*(s) \coloneqq \max_a [\sum_{s'} P(s'|s,a) \, (R(s,a,s') + \gamma V^*(s'))]$
- $V^*(s) \coloneqq \max_a [Q^*(s,a)]$
- $Q^*(s,a) \coloneqq \sum_{s'} P(s'|s,a) \, (R(s,a,s') + \gamma V^*(s'))$
- $Q^*(s,a) = \sum_{s'} P(s'|s,a) \, (R(s,a,s') + \gamma \max_a [Q^*(s',a)])$
- Solve iteratively
  - $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a) \, (R(s,a,s') + \gamma \max_a [Q_k(s',a)])$
  - Can also use Asynchronous learning

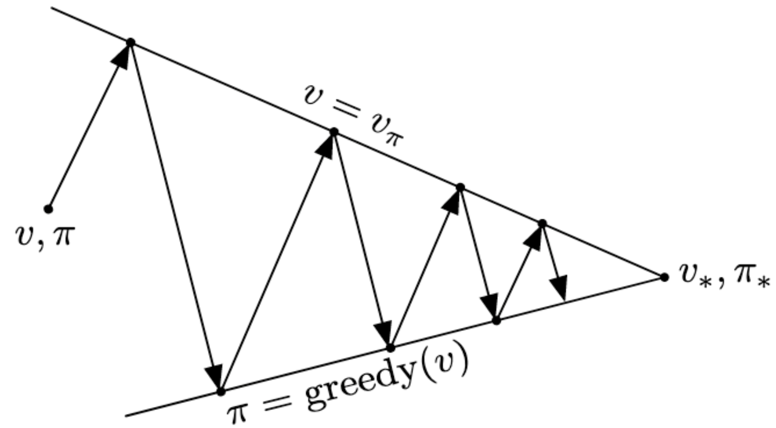
Issue 3: The policy often converges long before the values

- Value iteration converges to the true utility value: $V_{k\to\infty} \to V^*$
- V^* implies the optimal policy: π^*
- Can we converge directly on π^*?
  - Improve the policy in iteration until reaching the optimal one

Policy Iteration

1. Compute V_π: calculate state value for some fixed policy (not necessarily the optimal values, $V_\pi \neq V^*$)
2. Update π: update policy using one-step look-ahead with the resulting (non optimal) values
3. Repeat until policy converges

   (optimal values and policy)



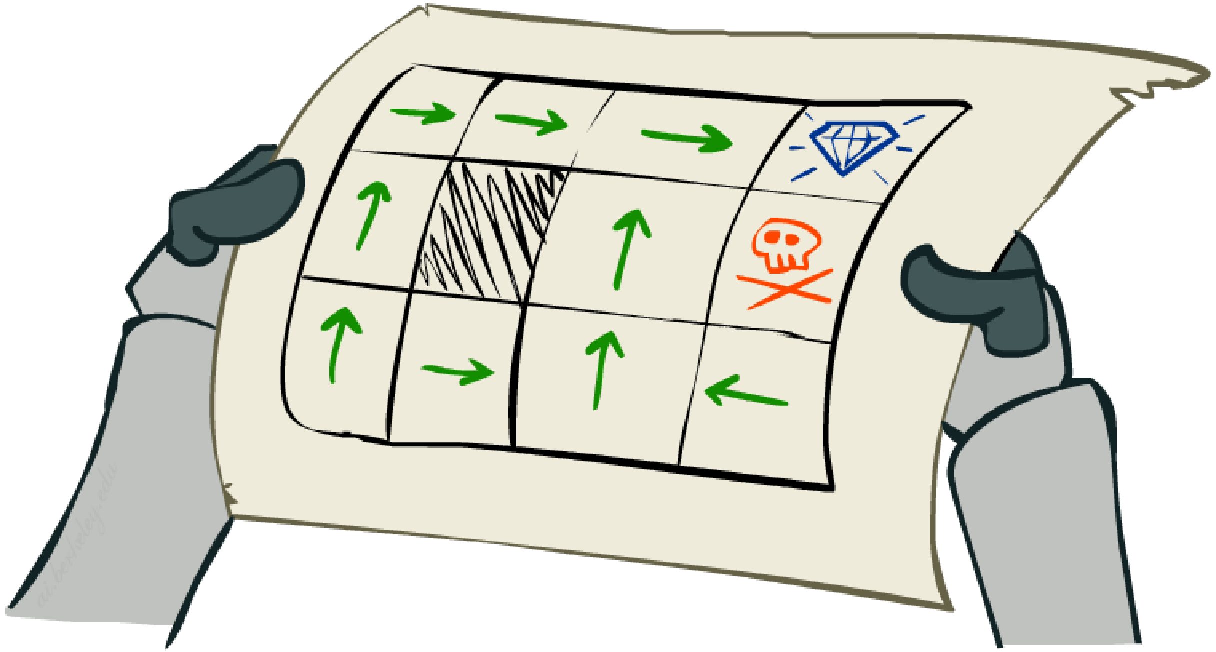
- Guaranteed converges to π^*
  - $\forall s, V_{k>0}(s) \leq V_{k+1}(s)$ i.e., π_i improves monotonically with i
  - A fixed point, $\forall s, V_k(s) = V_{k+1}(s)$, implies π^*

Policy Evaluation

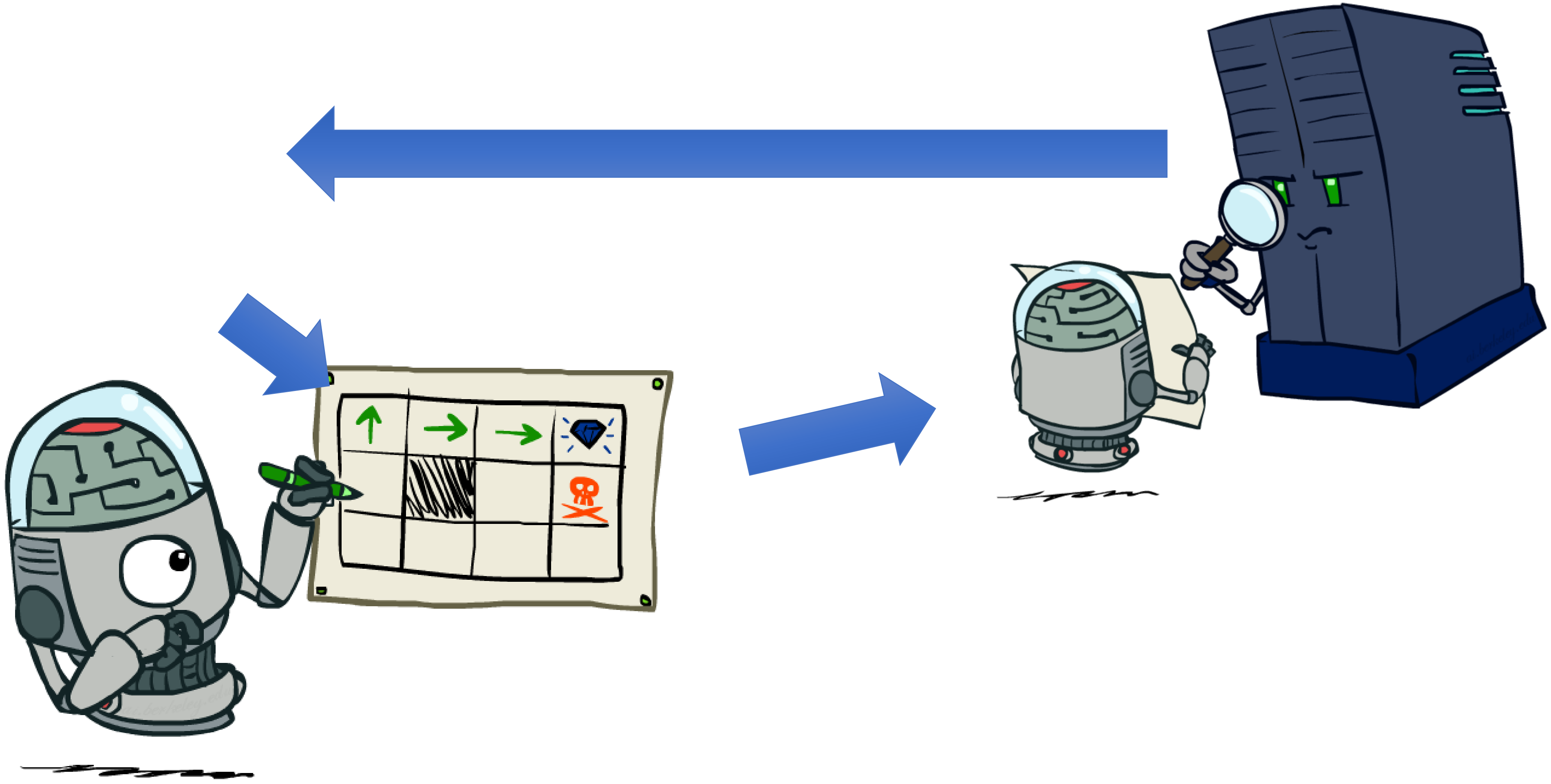
- Why is calculating V_π easier that calculating V^*?
  - Turns non-linear Bellman equations into linear equations
- $v^*(s) = \max_a [\sum_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma v^*(s'))]$
- $v_\pi(s) = \sum_{s'} P(s'|s,\pi(s))\,(R(s,\pi(s),s') + \gamma v^*(s'))$
- Solve a set of linear equations in $O(S^2)$
  - Solve with Numpy (numpy.linalg.solve)
  - Required for your home assignment
  - See: https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html#numpy.linalg.solve

Policy value as a Linear program

- $v_{11} = 0.8 \cdot (-0.1 + 0.95 \cdot v_{12}) + 0.1 \cdot (-0.1 + 0.95 \cdot v_{21}) + 0.1 \cdot (-0.1 + 0.95 \cdot v_{11})$
- $v_{12} = 0.8 \cdot (-0.1 + 0.95 \cdot v_{13}) + 0.2 \cdot (-0.1 + 0.95 \cdot v_{12})$
- …
- $v_{42} = -1$
- $v_{43} = 1$



Policy iteration

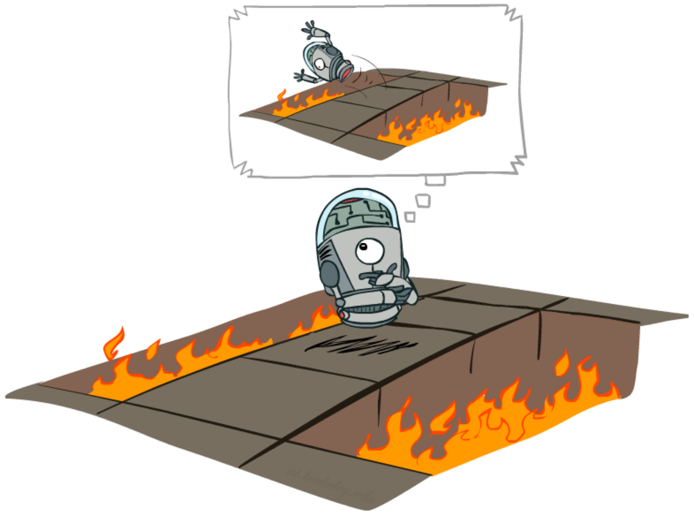


Comparison

- Both value iteration and policy iteration compute the same thing (optimal state 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 defines it
- In policy iteration:
  - We do several passes that update utilities for fixed policies (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)

Issue 4: requires knowing the model and the reward function

- We will explore online learning (reinforcement learning) approaches
- How can we learn the model and reward function from interactions?
- Do we even need to learn them? Can we learn V^*, Q^* without a model?
- Can we do without V^*, Q^* ? Can we run policy iteration without a model?



Offline optimization

Online Learning

Issue 5: requires discrete (finite) set of actions

- We will explore policy gradient approaches that are suitable for continuous actions, e.g., throttle and steering for a vehicle
- Can such approaches be relevant for discrete action spaces?
  - Yes! We can always define a continues action space as a distribution over the discreate actions (e.g., using the softmax function)
- Can we combine value-based approaches and policy gradient approaches and get the best of both?
  - Yes! Actor-critic methods

Issue 6: infeasible in large (or continues) state spaces

- Most real-life problems contain very large state spaces (practically infinite)
- It is infeasible to learn and store a value for every state
- Moreover, doing so is not useful as the chance of encountering a state more than once is very small
- We will learn to generalize our learning to apply to unseen states
- We will use value function approximators that can generalize the acquired knowledge and provide a value to any state (even if it was not previously seen)

Notation

- π^* - a policy that that yields the maximal expected sum of rewards
- $V^*(s)$ - the expected sum of rewards from being at s then following π^*
- $V_\pi(s)$ - the expected sum of rewards from being at s then following π
- $Q^*(s,a)$ - the expected sum of rewards from being at s, taking action a and then following π^*
- $Q_\pi(s,a)$ - the expected sum of rewards from being at s, taking action a and then following π
- G_t - observed sum of rewards following time t, i.e., $\sum_{k=t}^{T} r_k$

What did we learn?

- MDP = States, Actions, Reward function, Transition probabilities
- Meaningful policy evaluation requires a discount factor, finite planning horizon, or terminal states
- Knowing the MDP model allows us to compute the optimal policy offline (through VI or PI)
- PI is usually faster to converge due to simplified bellman updates (eliminating the $\max_a$ operator)
- A known model is not a practical assumption and is not assumed in general RL
    - We will drop this assumption in the next lecture

What next?

- **Class**: Monte-Carlo RL algorithms
- **Assignments**:
  - Value Iteration
  - Asynchronous Value Iteration
  - Policy Iteration
  - Due by Monday, September-23, EOD.
- **Quiz (on Canvas)**:
  - Dynamic programming – by Tuesday, Sep-9, EOD
- **Project**:
  - Find a partner
  - Converge on the project’s topic
  - Define the relevant MDP