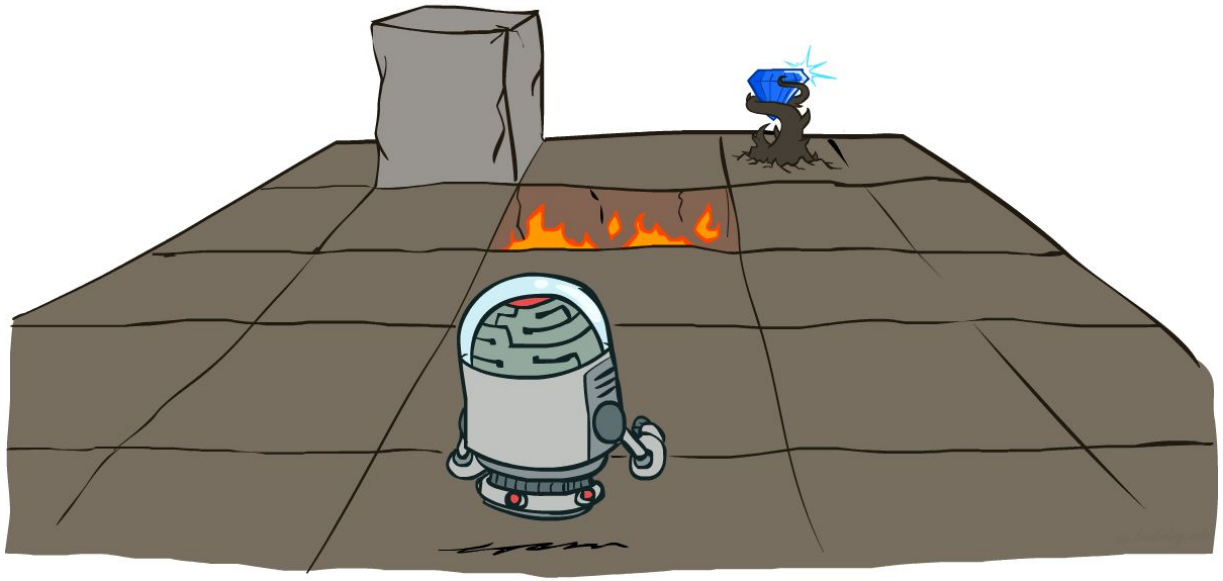
Reinforcement Learning

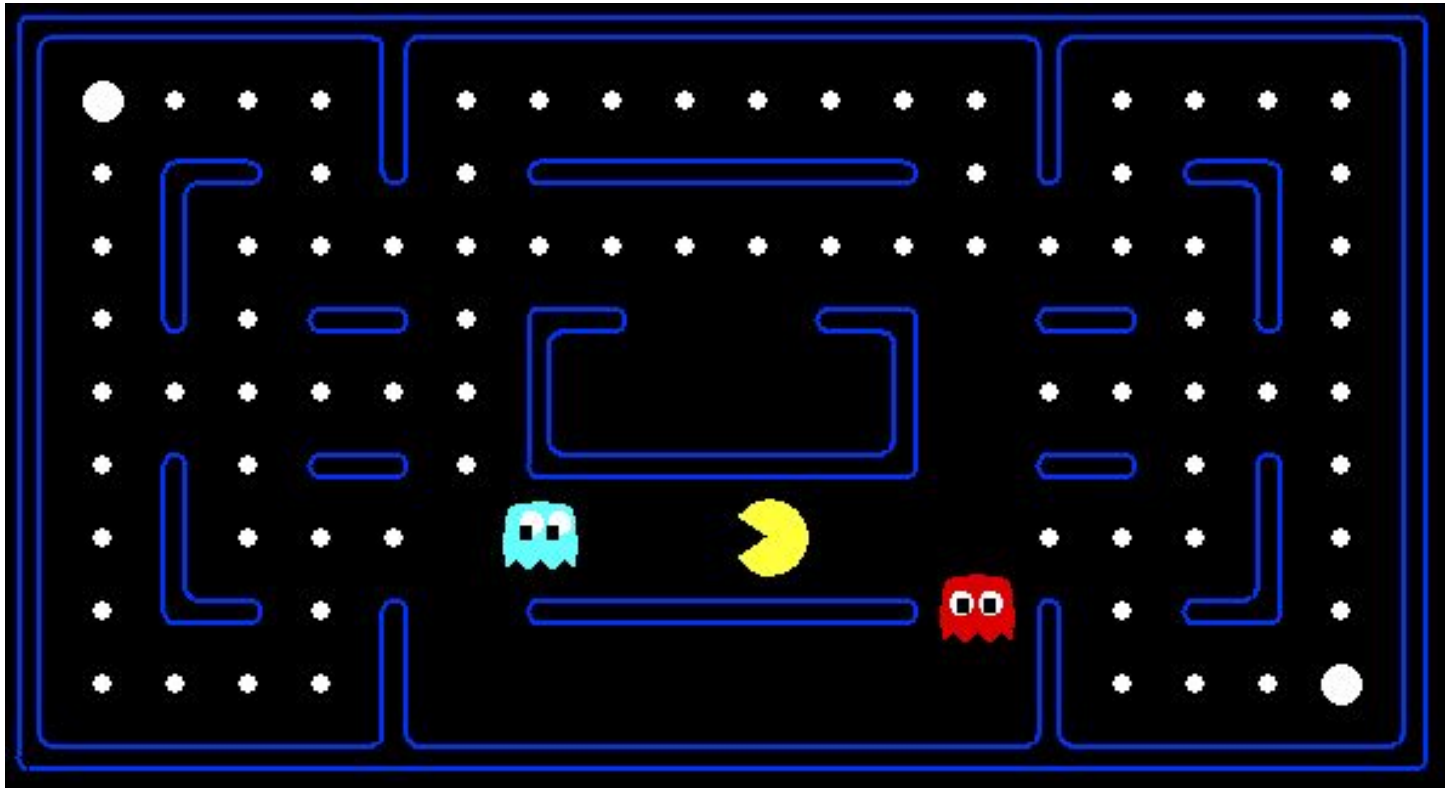
Markov Decision Processes



Instructor: Ngoc-Hoang LUONG, PhD.

[These slides were adapted from the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. http://ai.berkeley.edu.]

Behavior from Computation

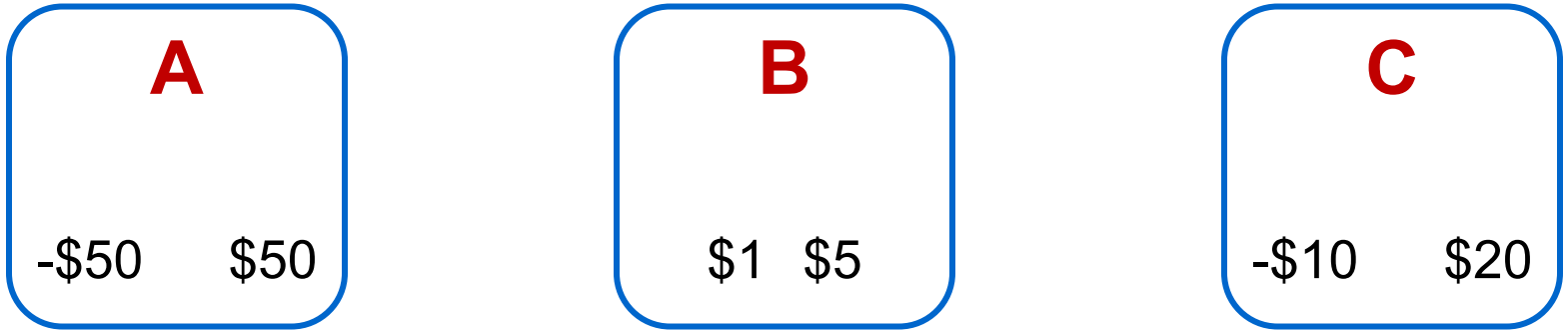


Video of Demo Mystery Pacman

Adversarial Games

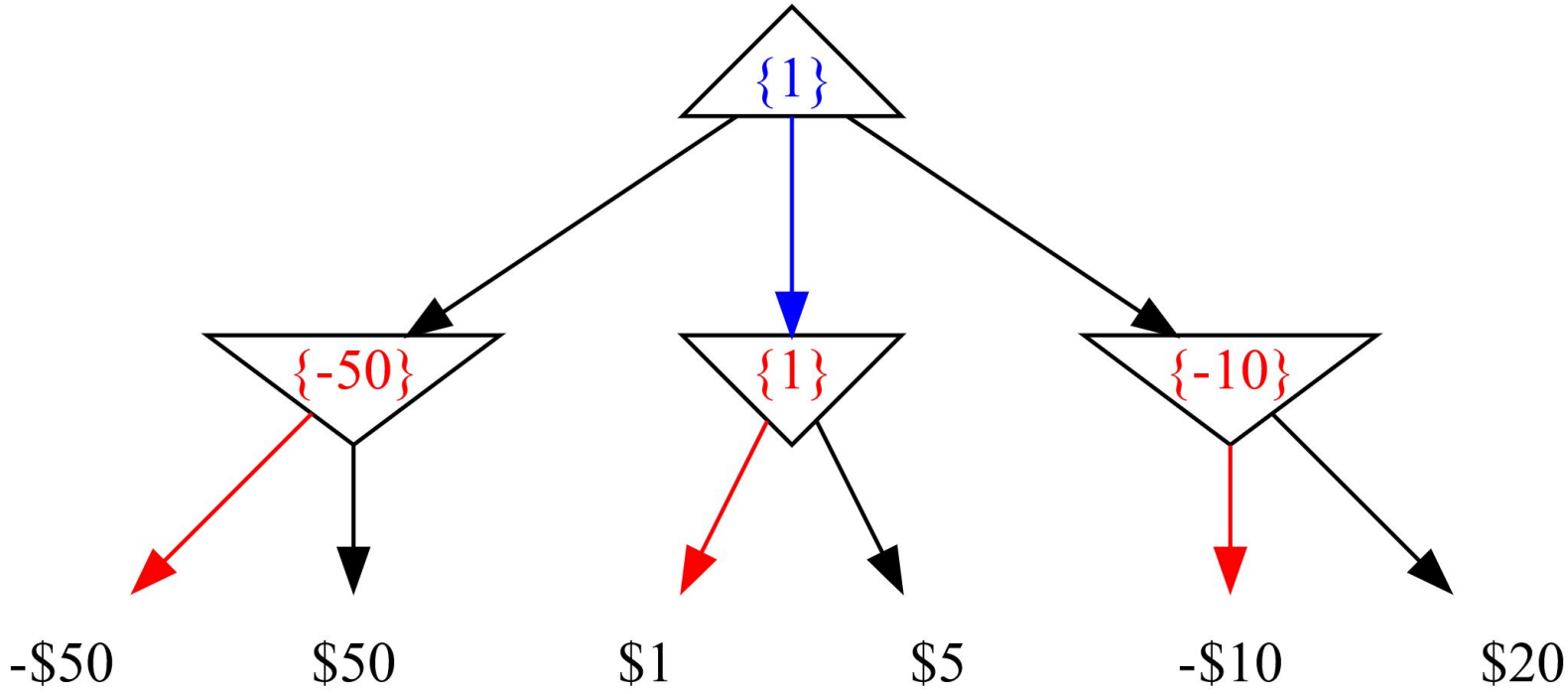
A simple game

- You choose one of the three bins.
- I choose a number from that bin.
- Your goal is to maximize the chosen number.

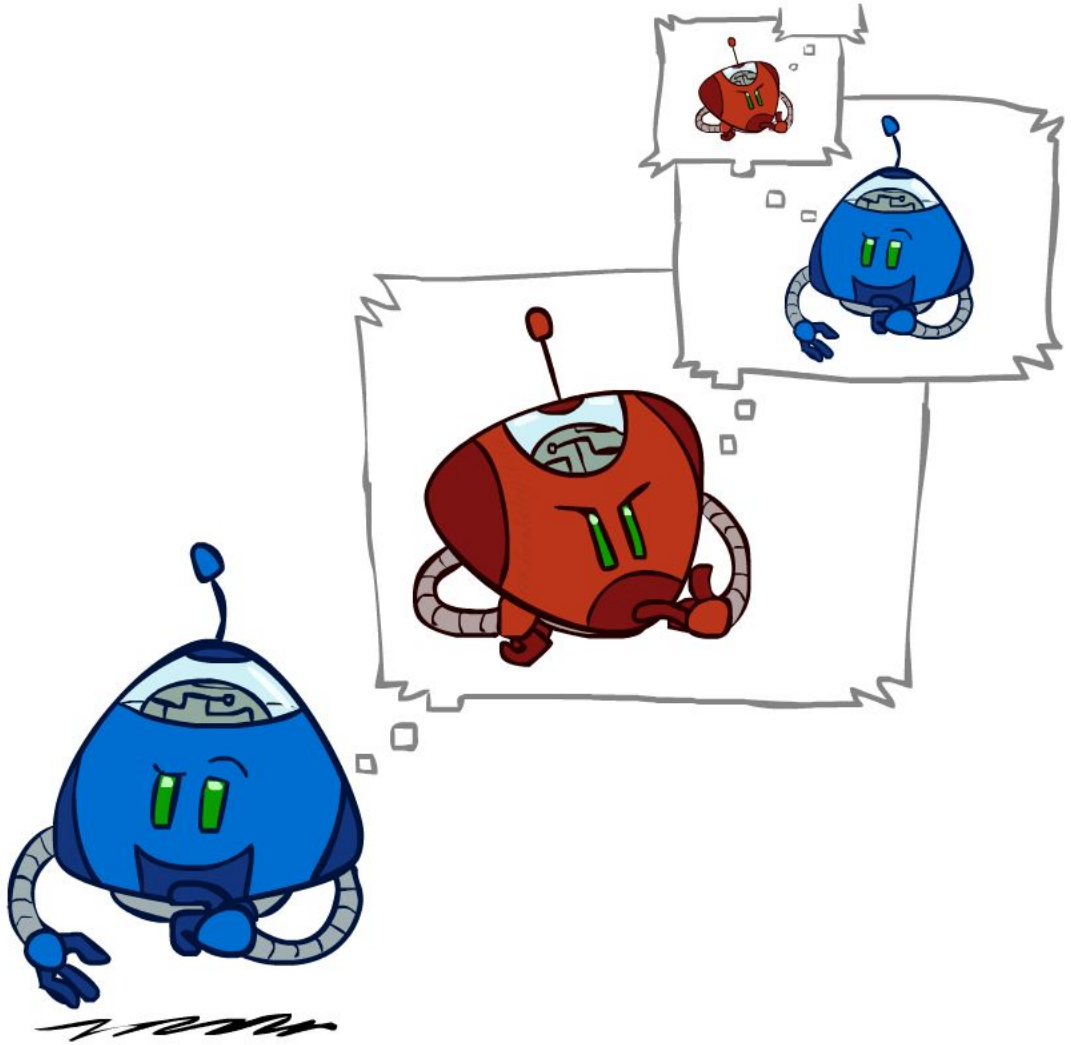


Game tree

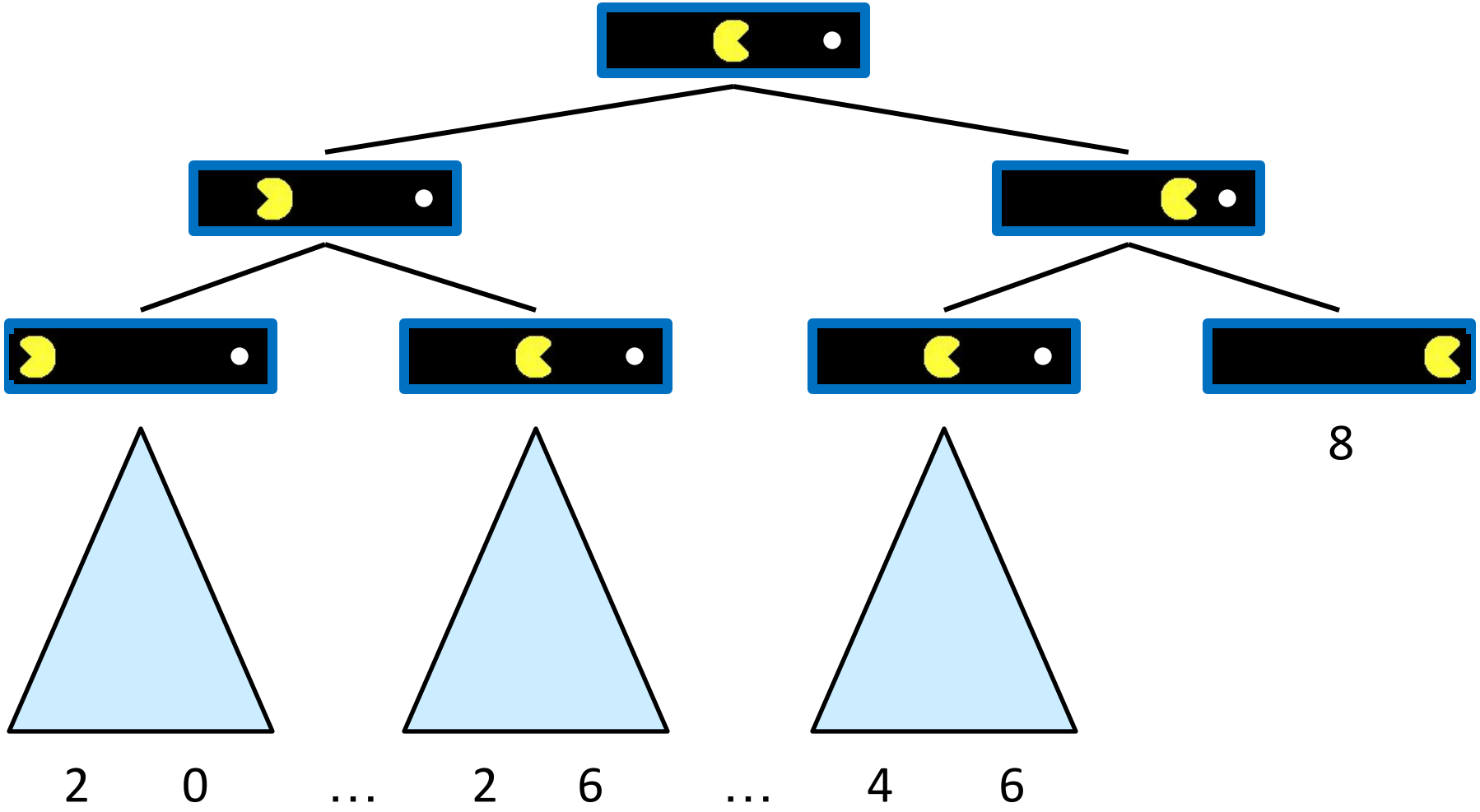
- Each node is a decision point for a player.
- Each root-to-leaf path is a possible outcome of the game.
- Your goal is to maximize the chosen number.



Adversarial Search

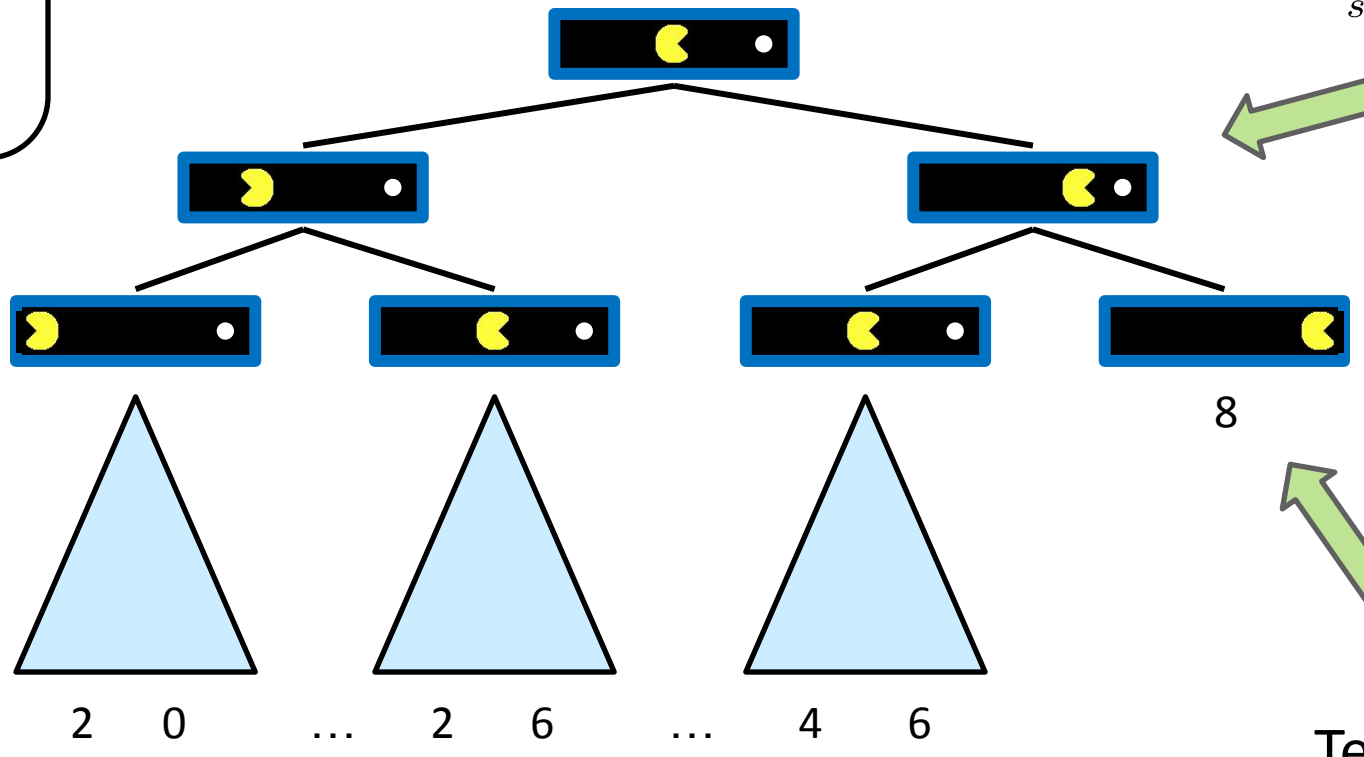


Single-Agent Trees



Value of a State

Value of a state:
The best achievable outcome (utility) from that state



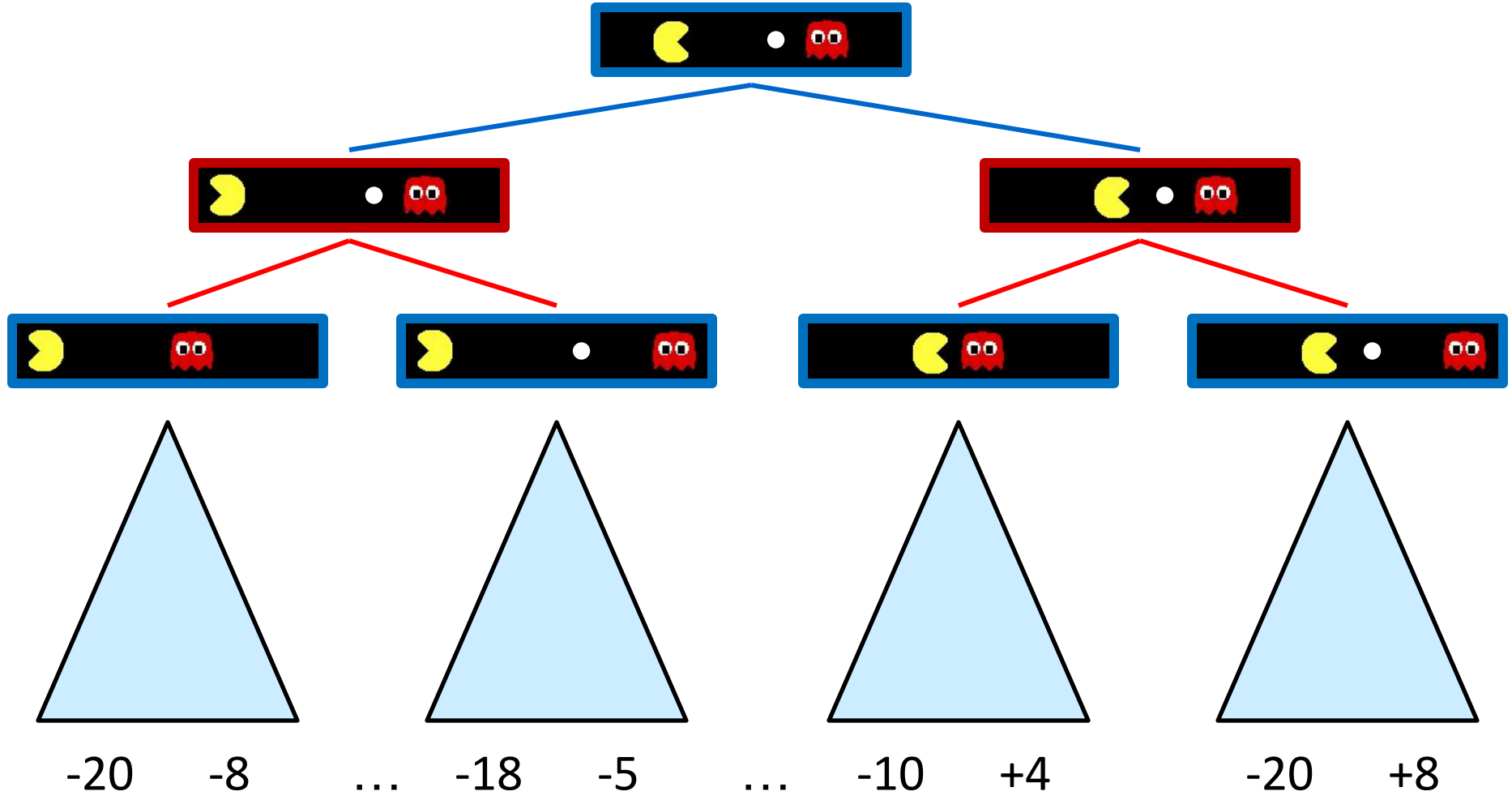
Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

Terminal States:

$$V(s) = \text{known}$$

Adversarial Game Trees



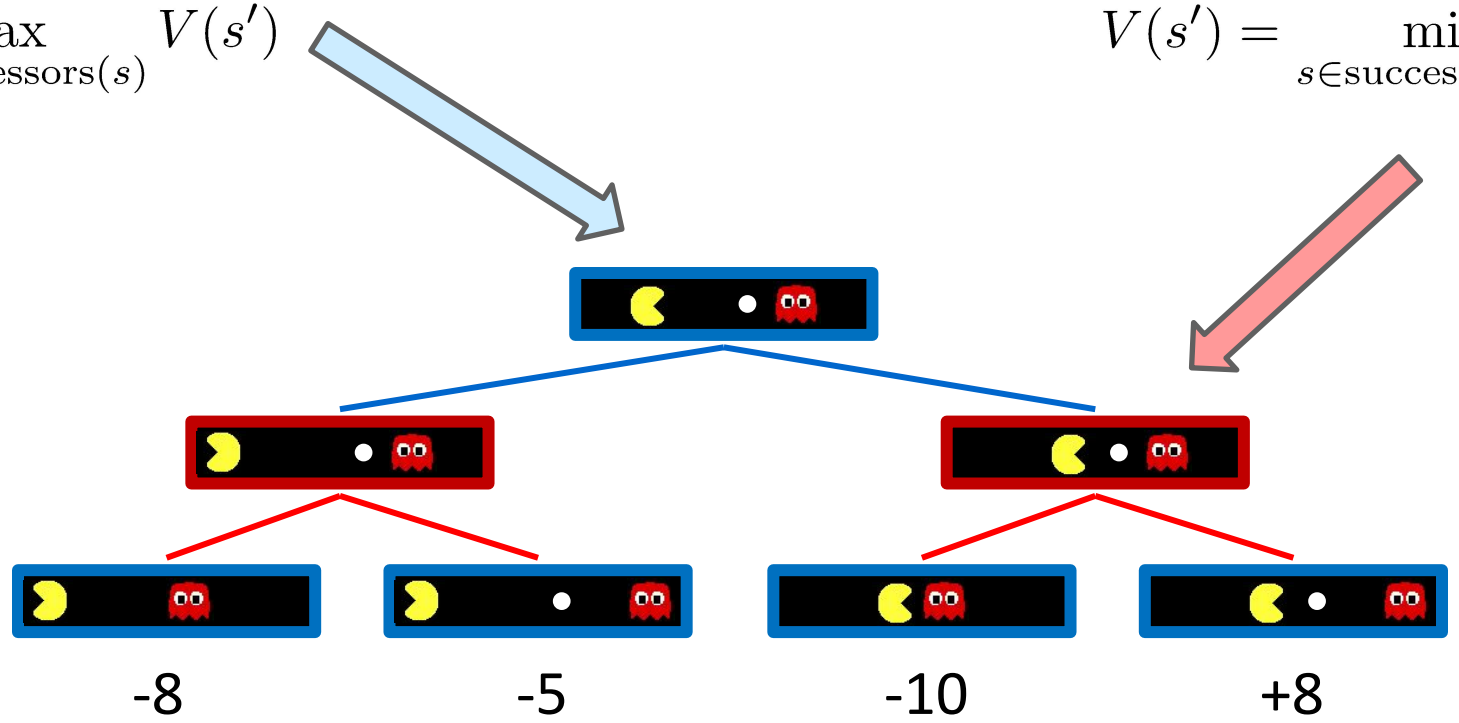
Minimax Values

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

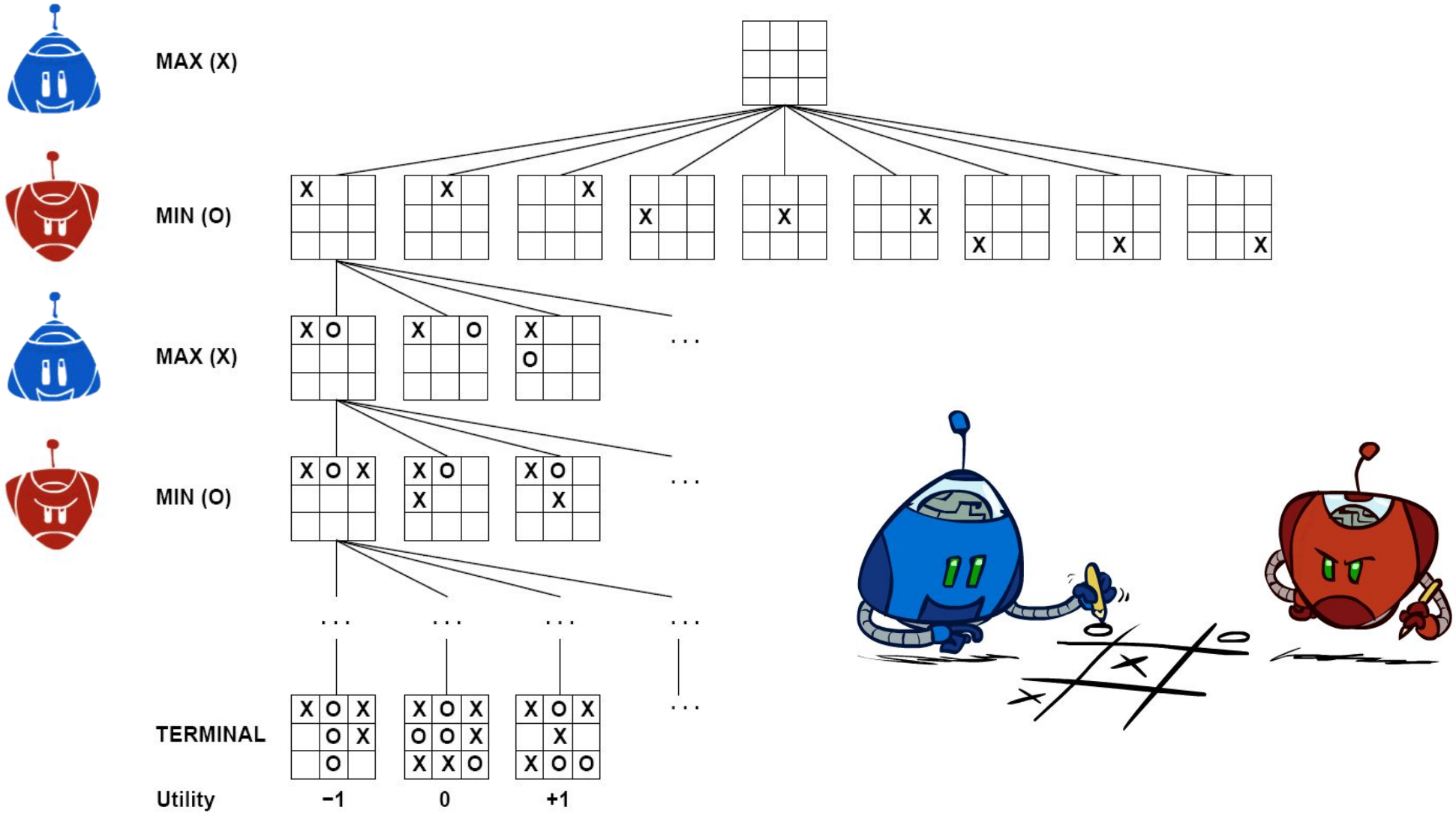
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

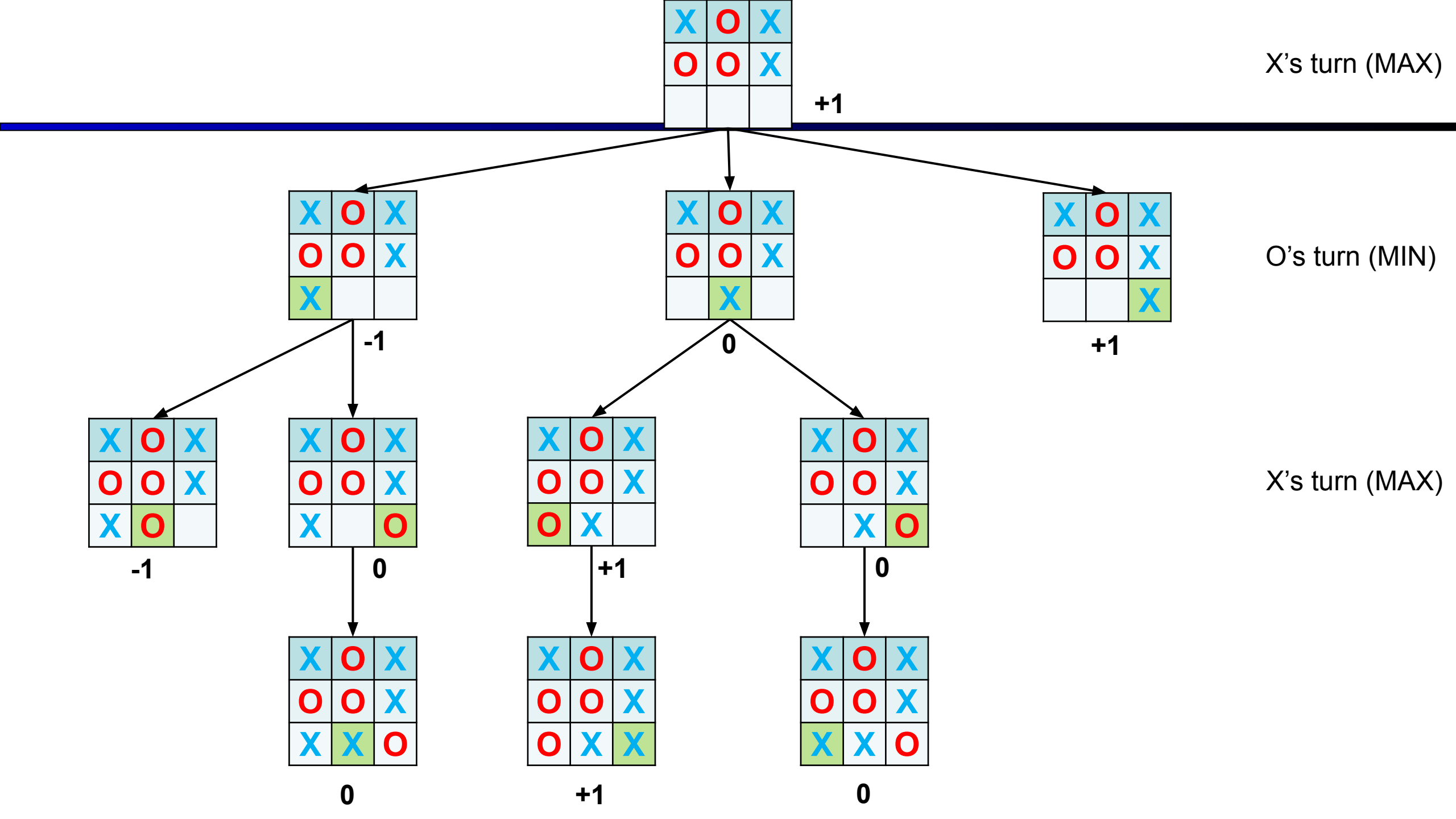


Terminal States:

$$V(s) = \text{known}$$

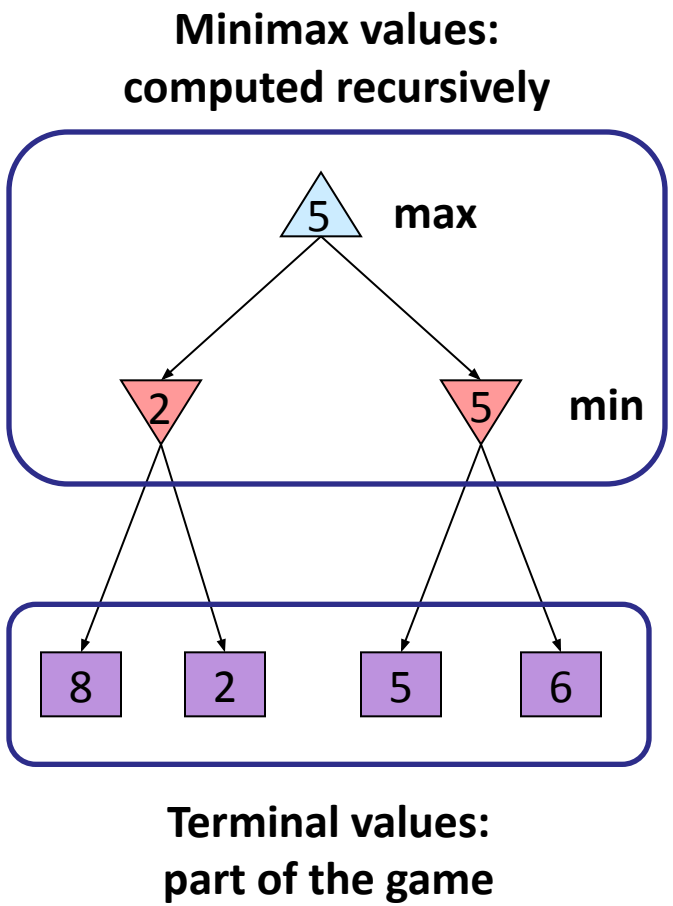
Tic-Tac-Toe Game Tree




X's turn (MAX)
+1
O's turn (MIN)
-1
0
+1
X's turn (MAX)
-1
0
+1
0
0
+1
0

Adversarial Search (Minimax)

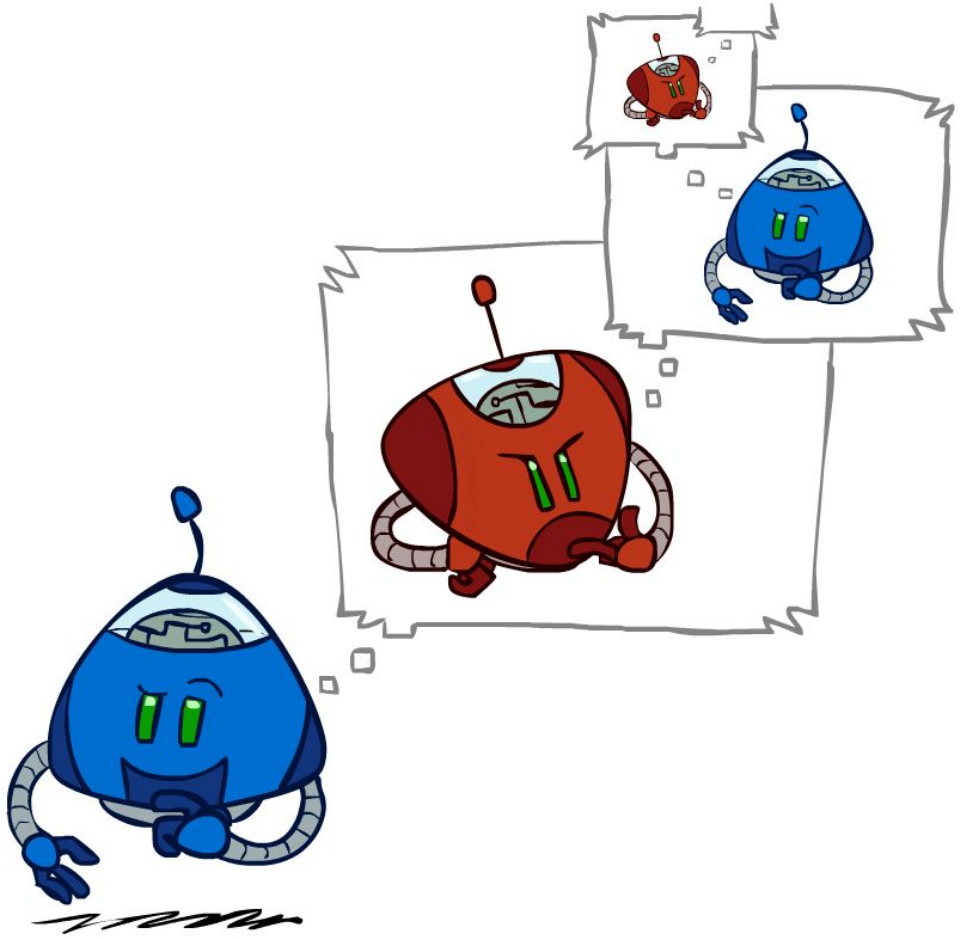
- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary



Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: O(bm)

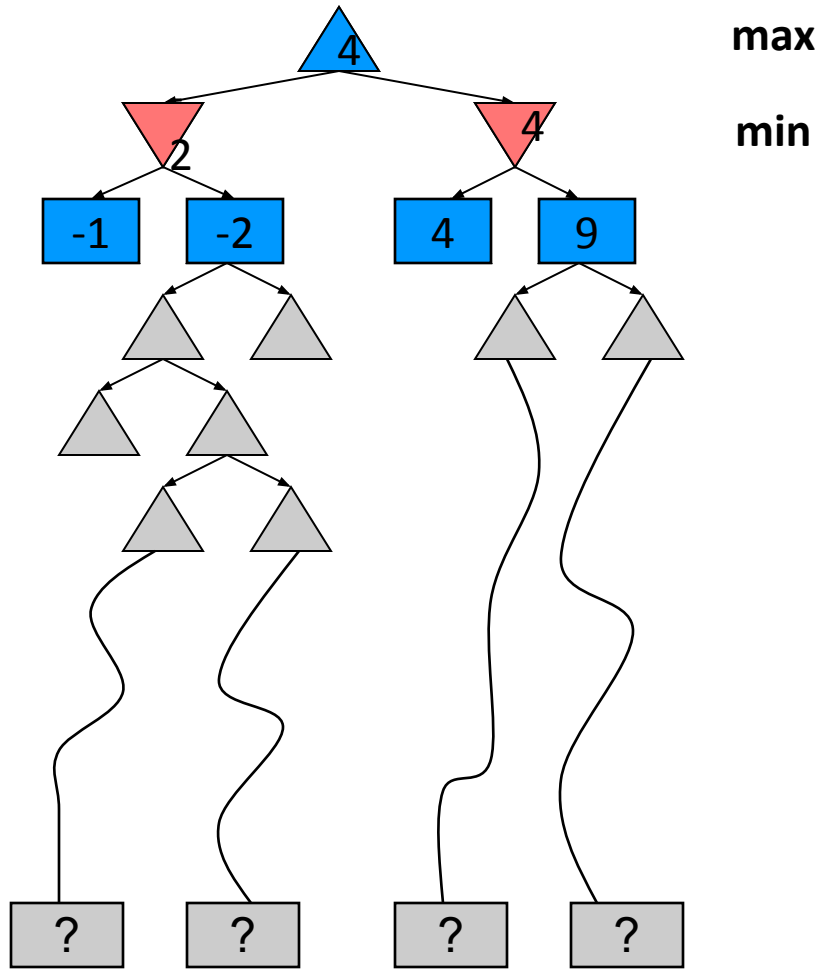
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Resource Limits

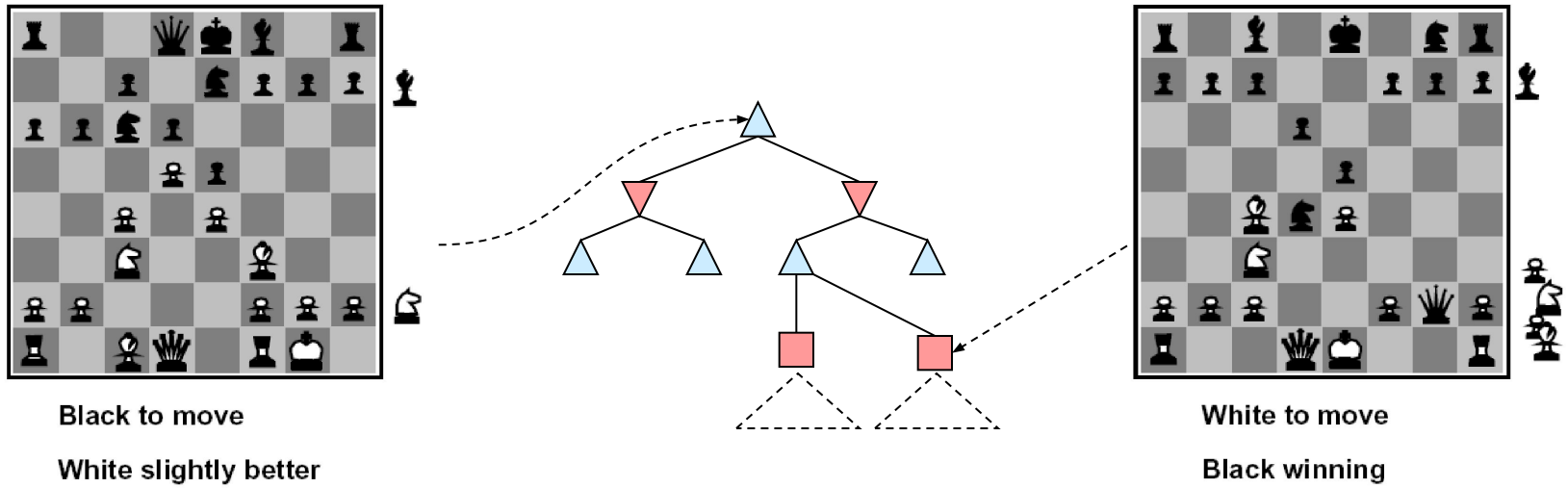
Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search



- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

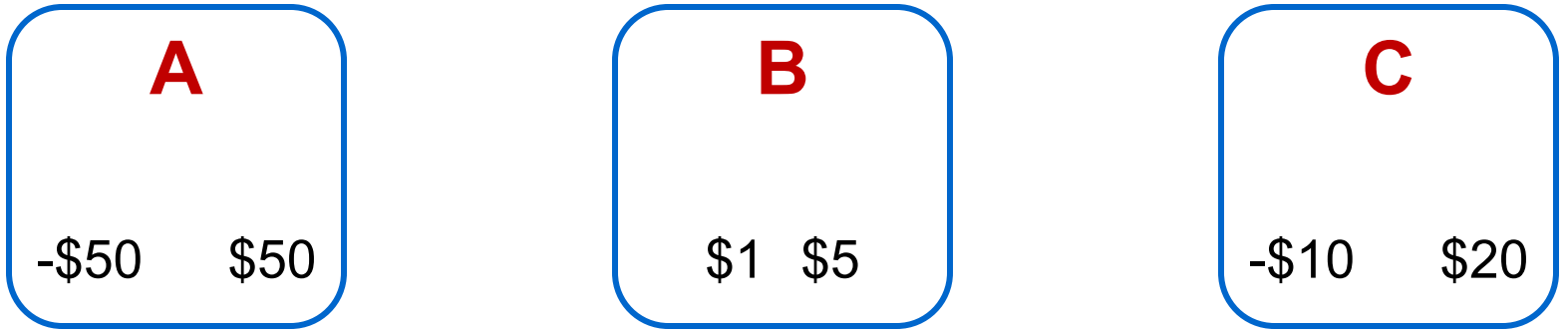
- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Video of Demo Limited Depth (2)

Video of Demo Limited Depth (10)

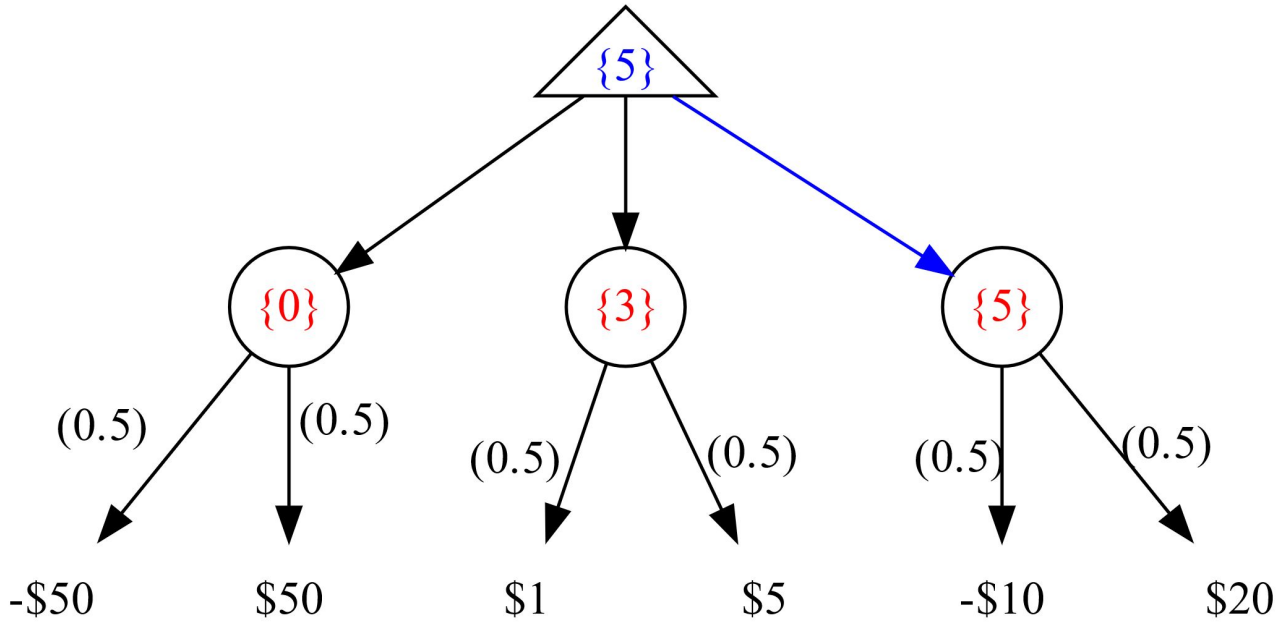
Minimax Properties

- You choose one of the three bins.
- I toss a fair coin. If head, the bigger number is returned. If tail, the smaller number is returned.
- Your goal is to maximize the chosen number.

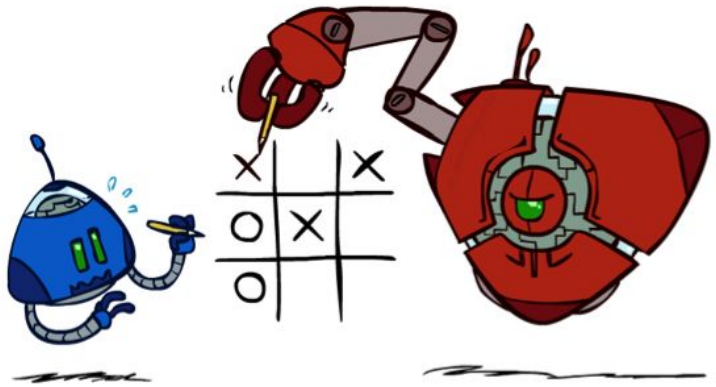


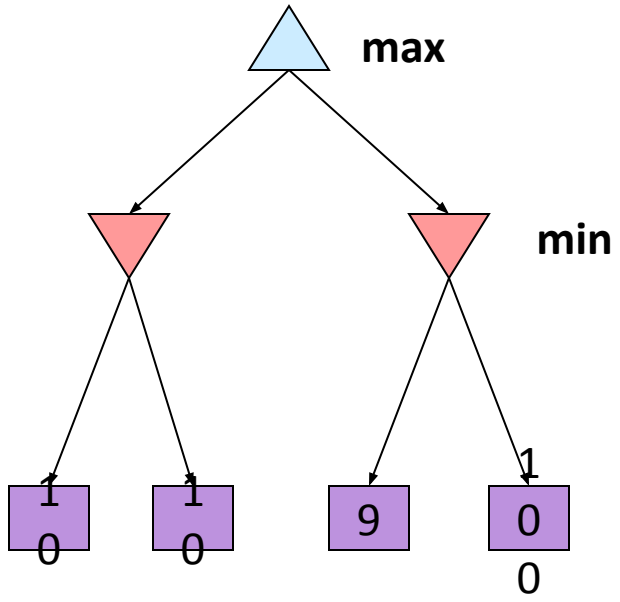
Minimax Properties

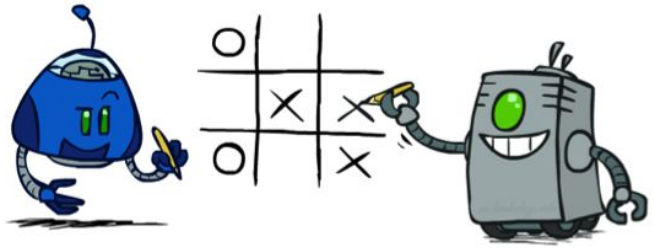
- You choose one of the three bins.
- I toss a fair coin. If head, the bigger number is returned. If tail, the smaller number is returned.
- Your goal is to maximize the chosen number.



Minimax Properties







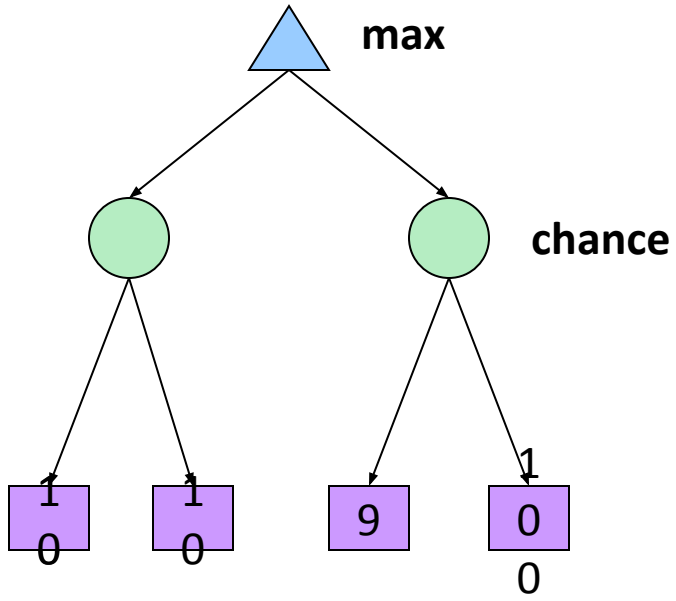
Optimal against a perfect player. Otherwise?

Video of Demo Min vs. Exp (Min)

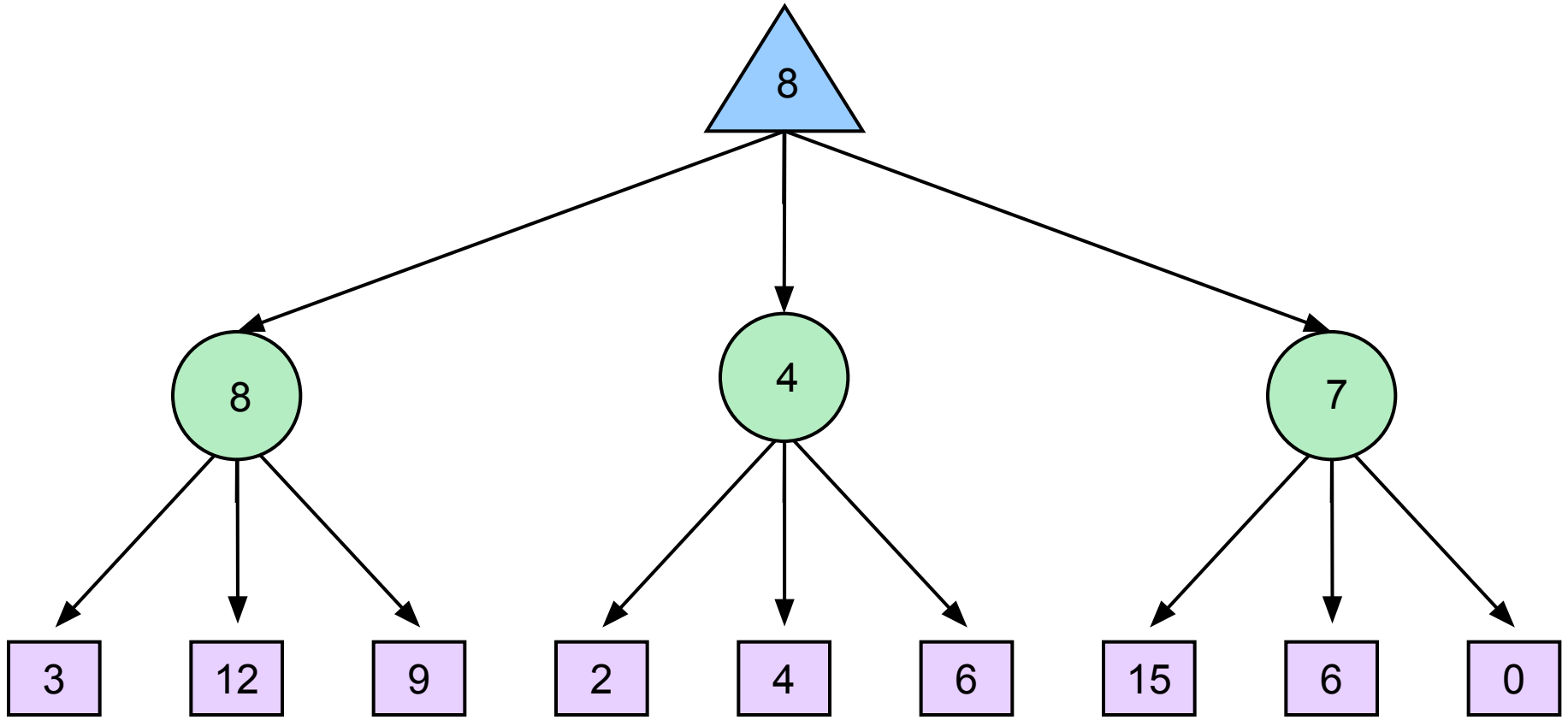
Video of Demo Min vs. Exp (Exp)

Expectimax Search

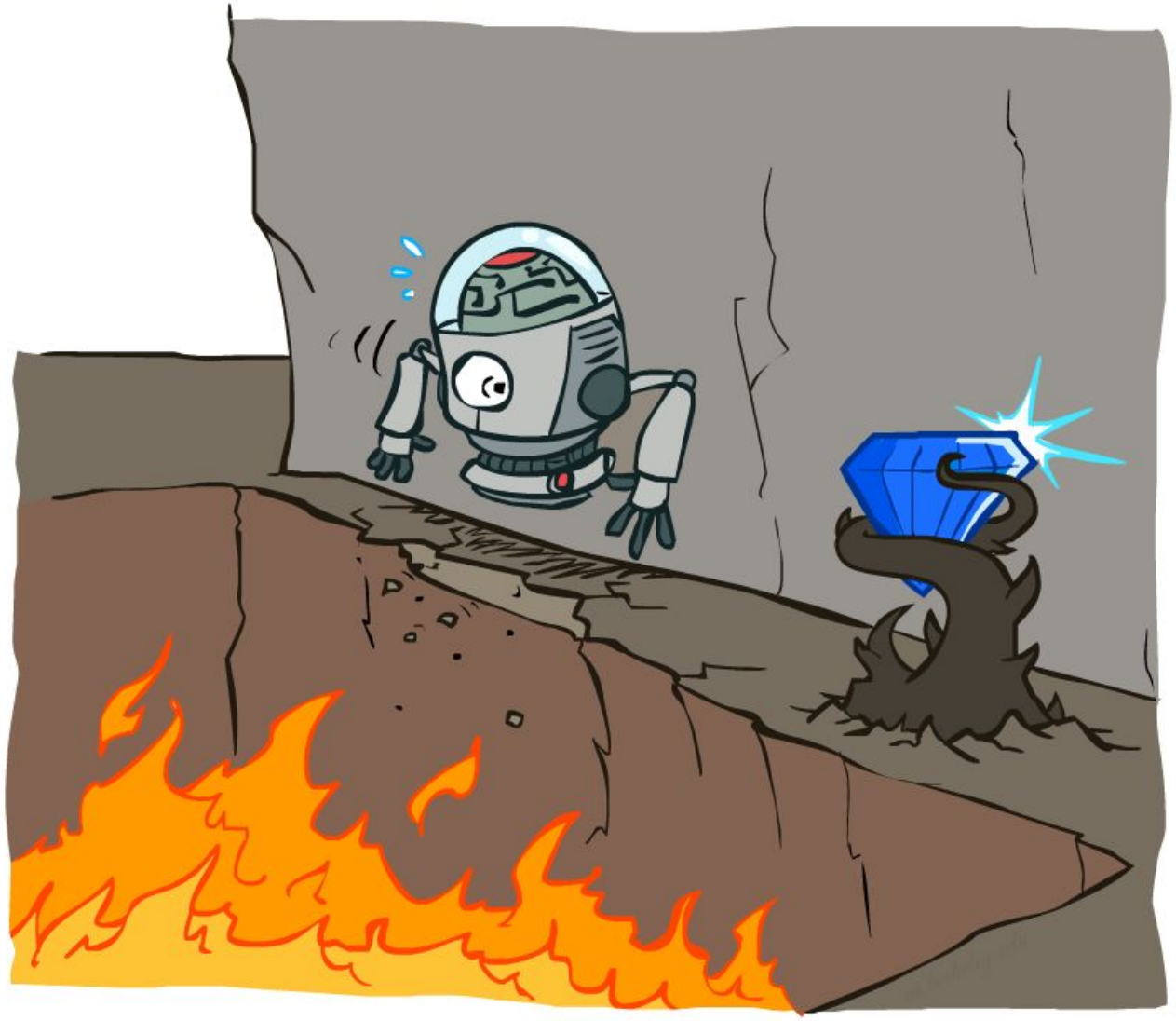
- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes
- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children
- Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes



Expectimax Example

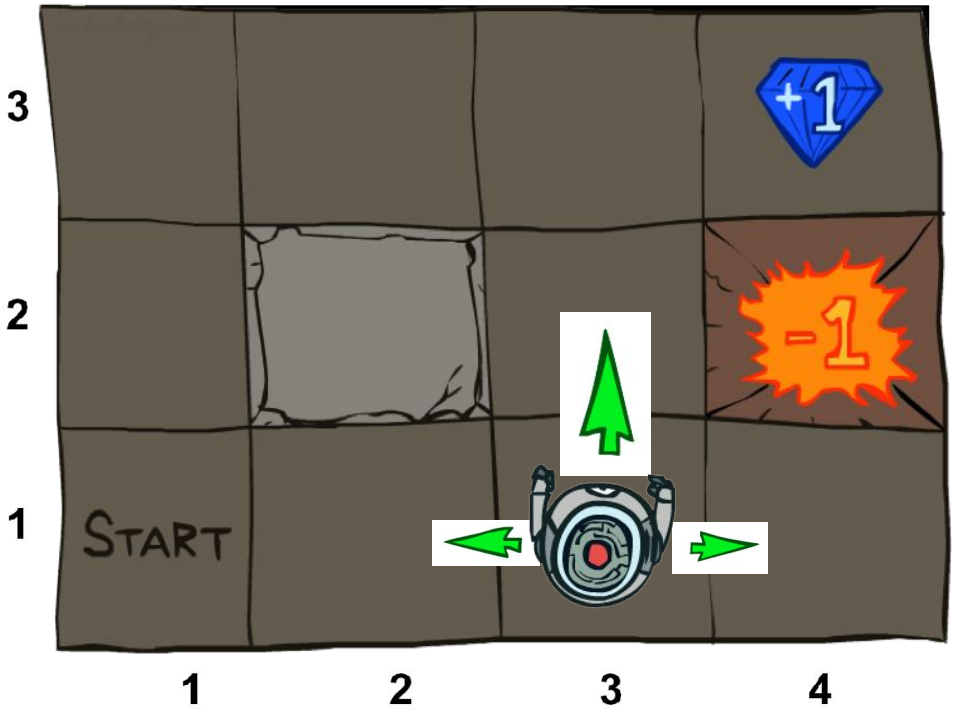


Non-Deterministic Search



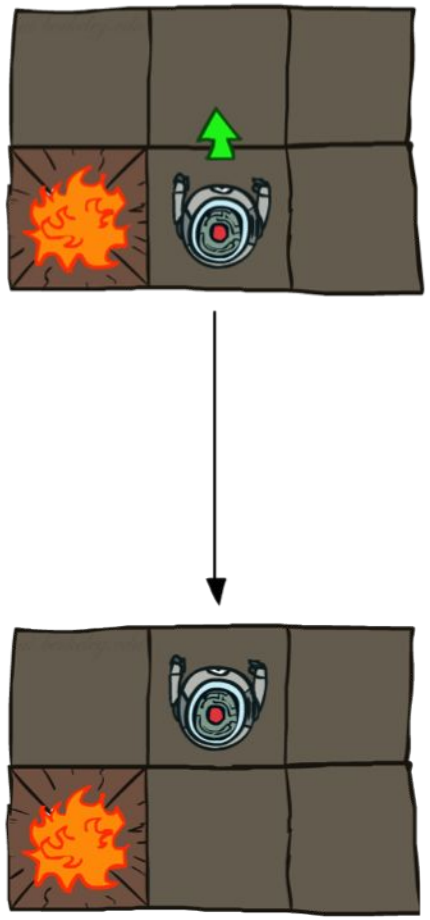
Example: Grid World

- 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 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 rewards

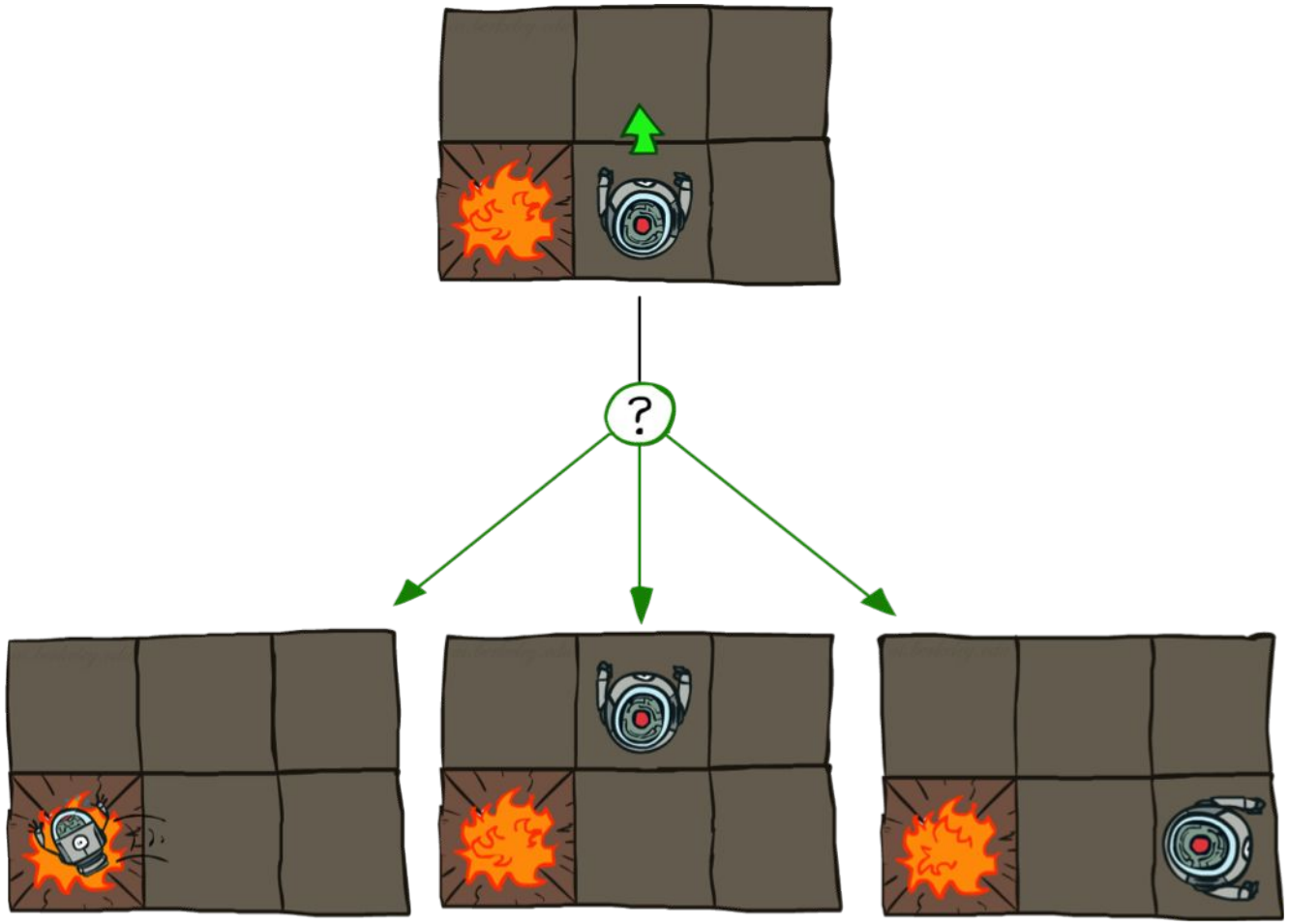


Grid World Actions

Deterministic Grid World

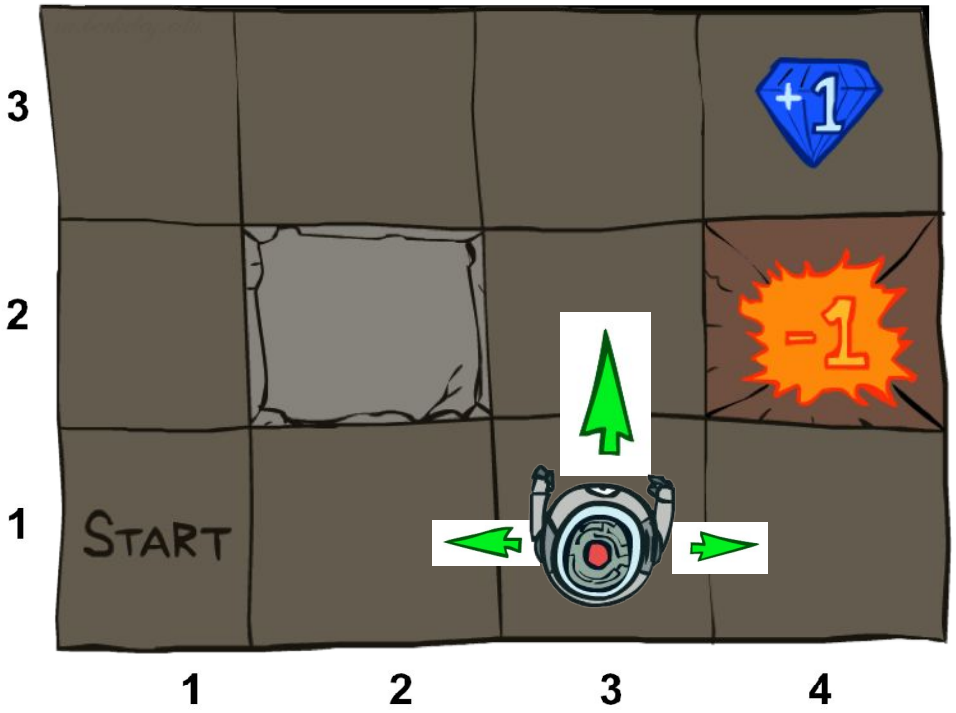


Stochastic Grid World



Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s' | s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe a terminal state



- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll have a new tool soon

What is Markov about MDPs?

- "Markov" generally means that given the present state, the future and the past are independent

- For Markov decision processes, "Markov" means action outcomes depend only on the current state

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

$$=$$

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

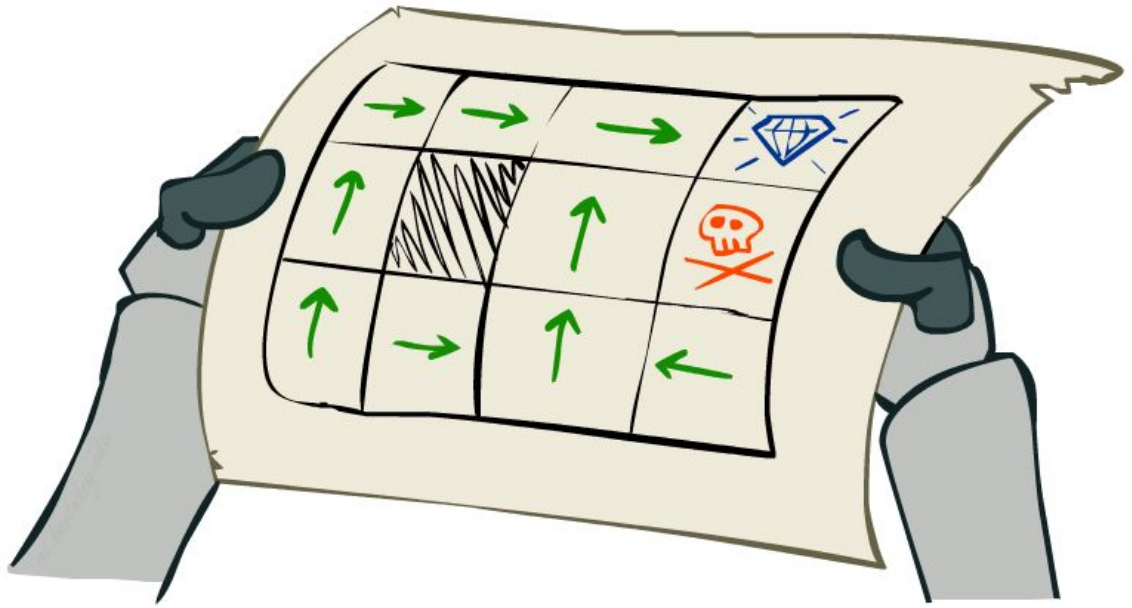
Andrey Markov
(1856-1922)

Policies

- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

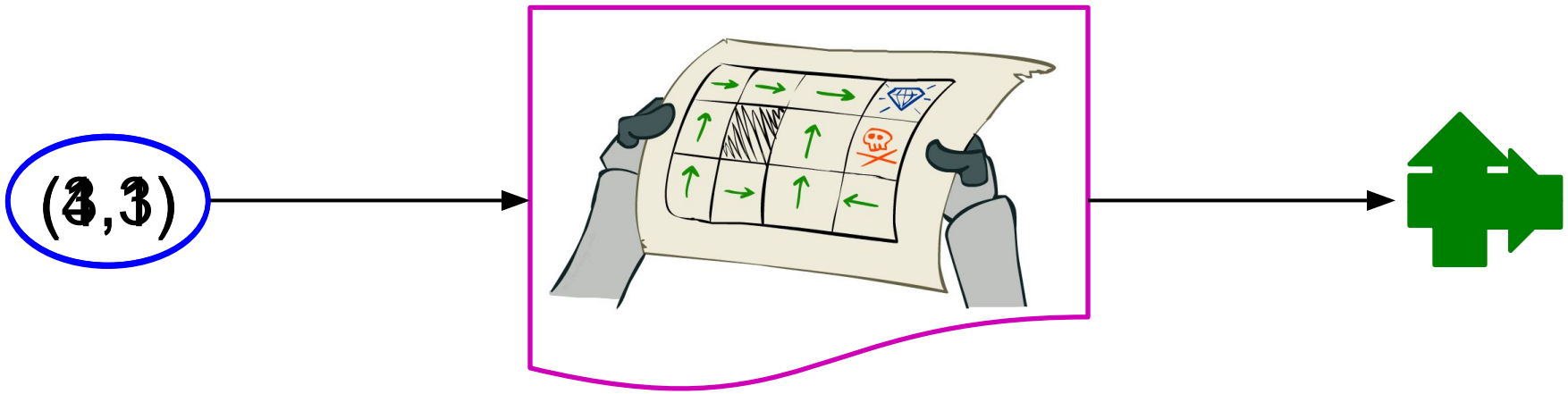
- A policy $\pi: S \rightarrow A$ gives an action for each state.
  - E.g, $\pi((1,1))$ = "north", $\pi((4,1))$ = "west", $\pi((3,3))$="east", $\pi((3,2))$="north",…

- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - An optimal policy is one that maximizes expected utility if followed

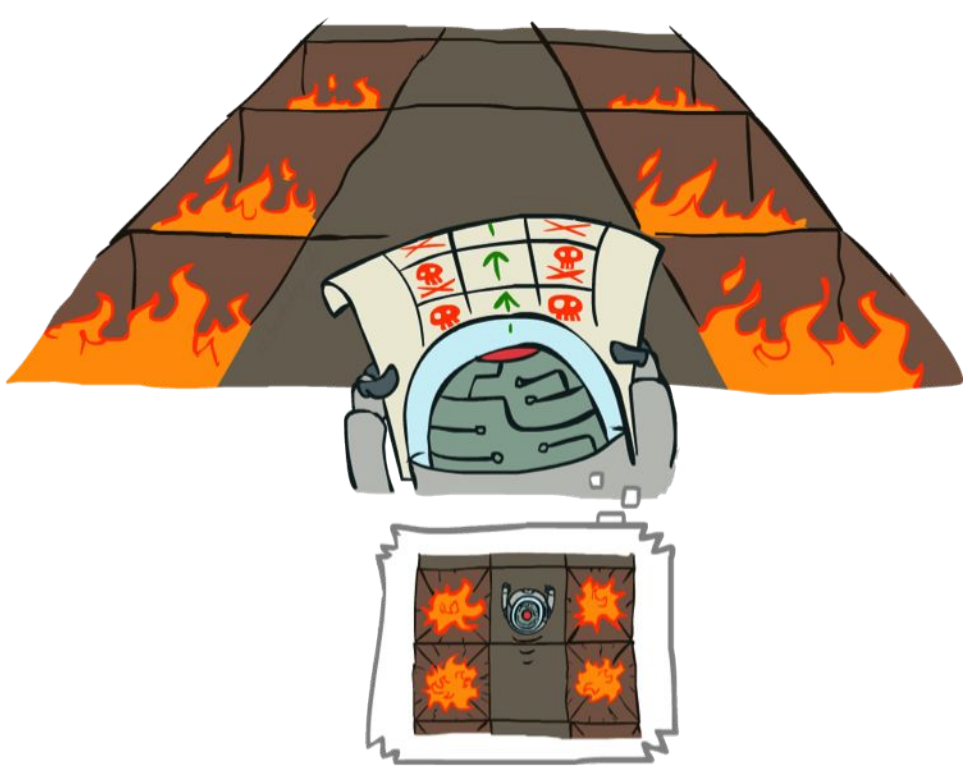


Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

Policy

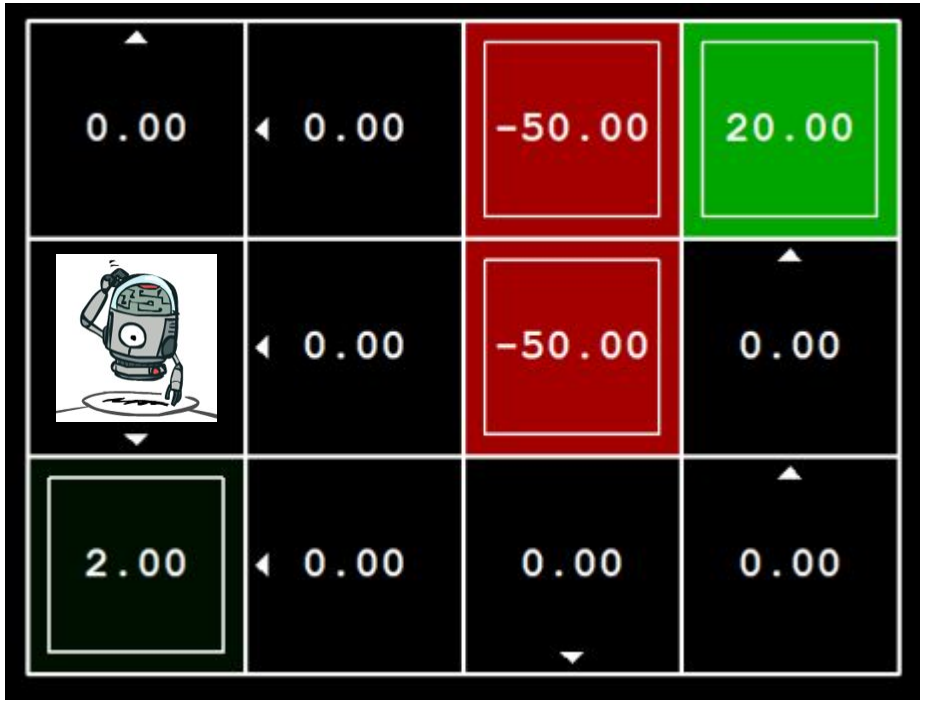


Example: Bridge Crossing



	-100.00	-100.00	-100.00	-100.00	-100.00	
1.00		0.00	0.00	0.00	0.00	10.00
	-100.00	-100.00	-100.00	-100.00	-100.00	

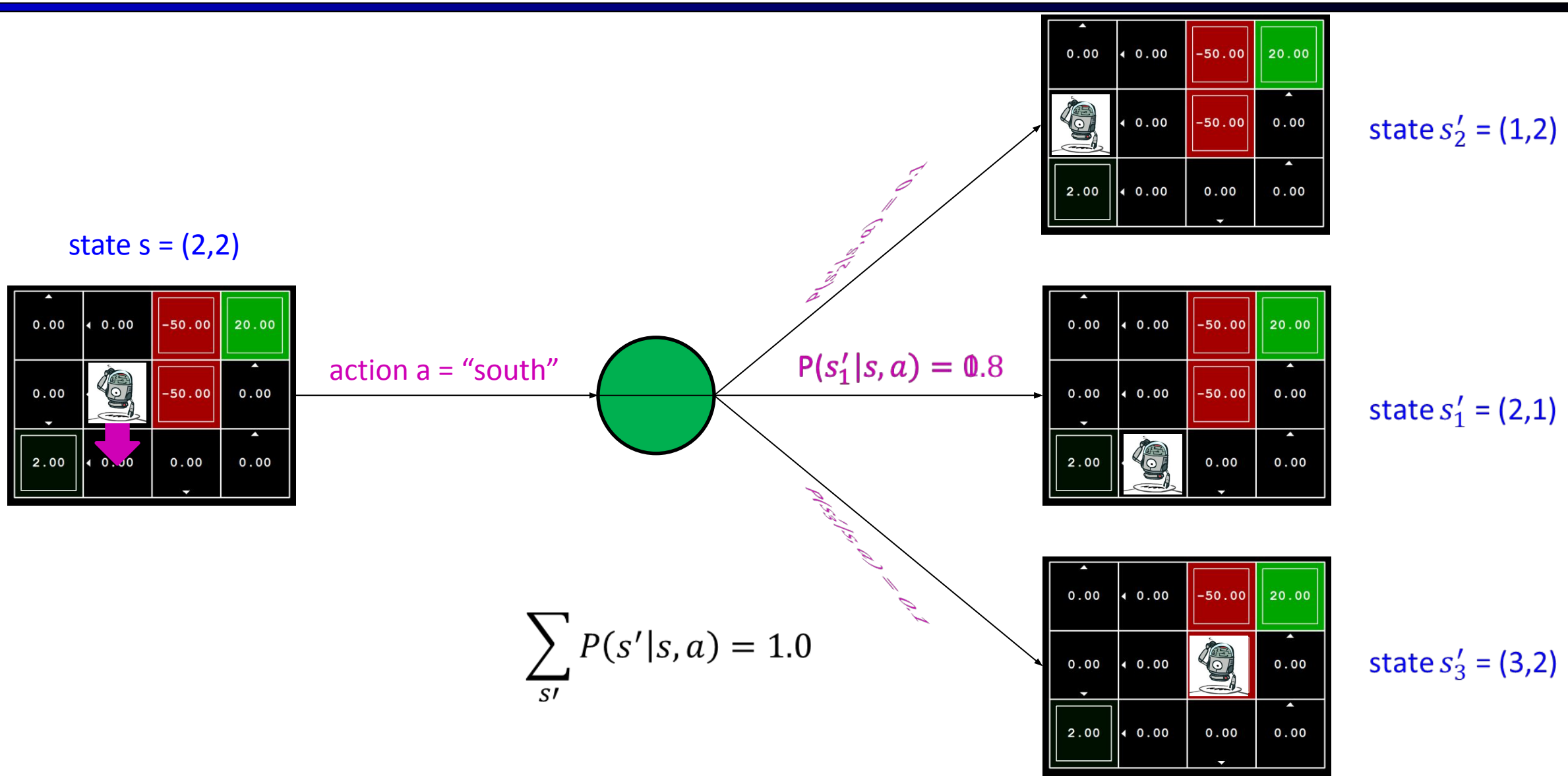
Example: Volcano



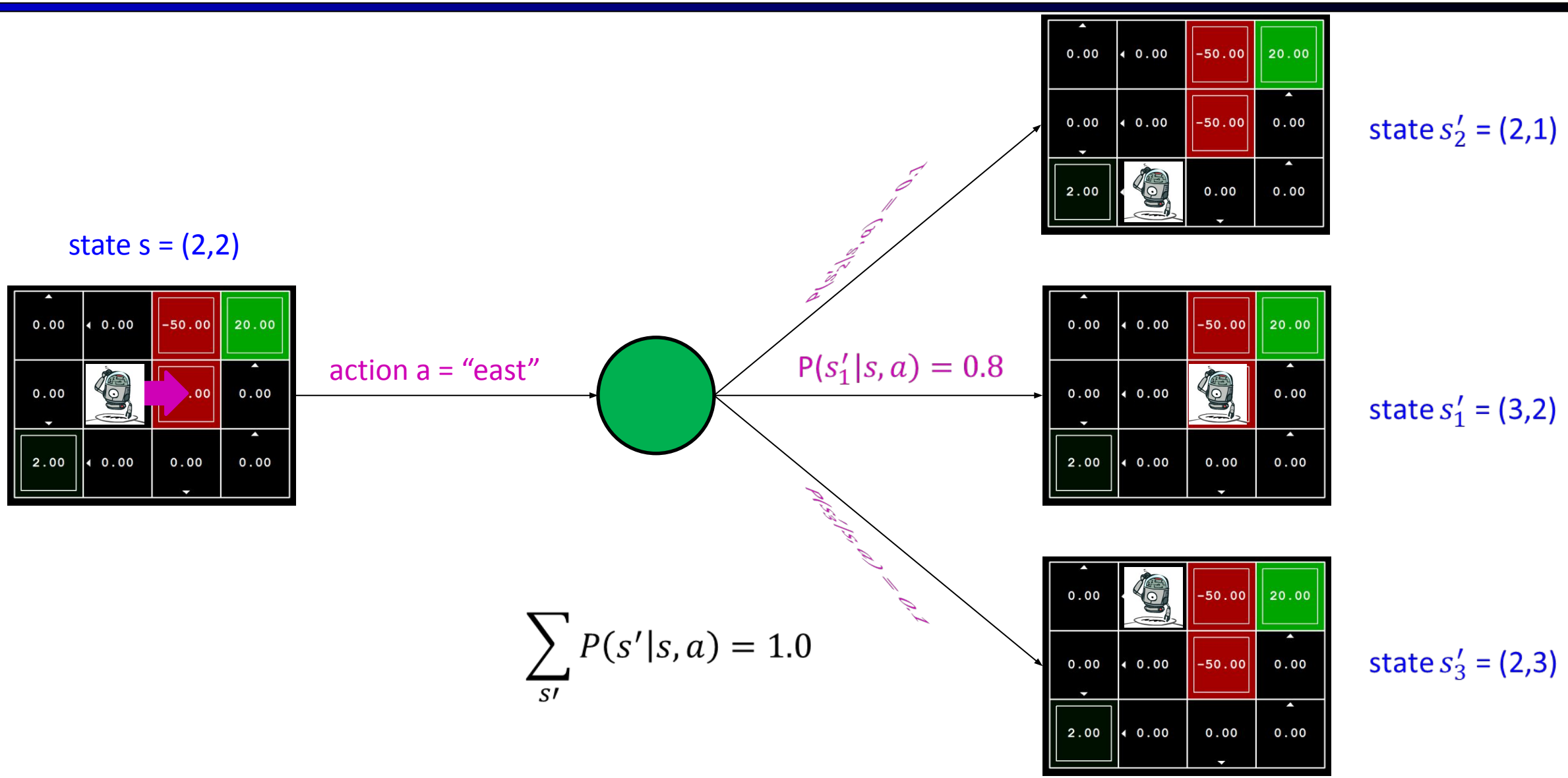
123RF.com

shutterstock.com

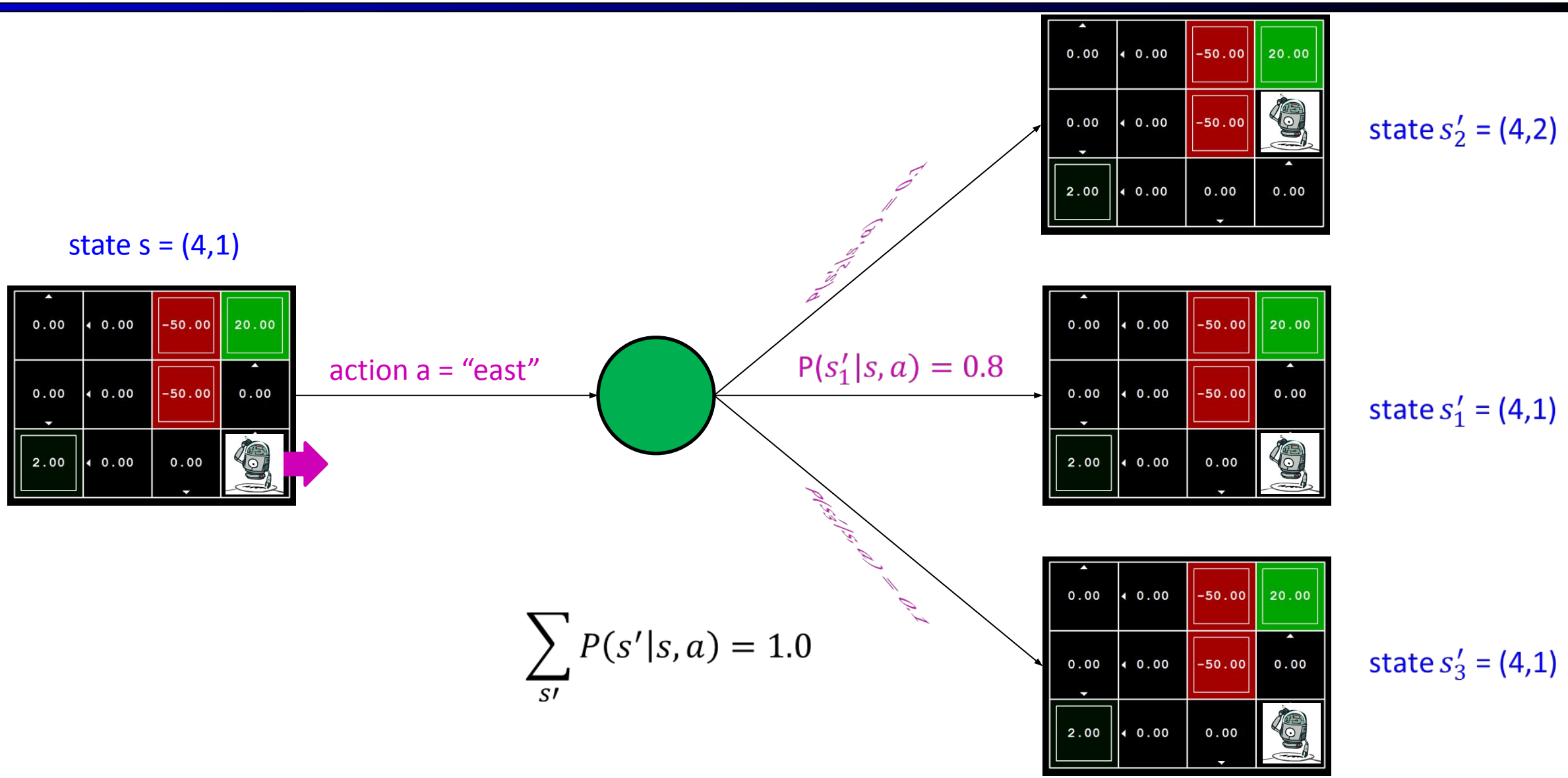
MDP Search Tree



MDP Search Tree

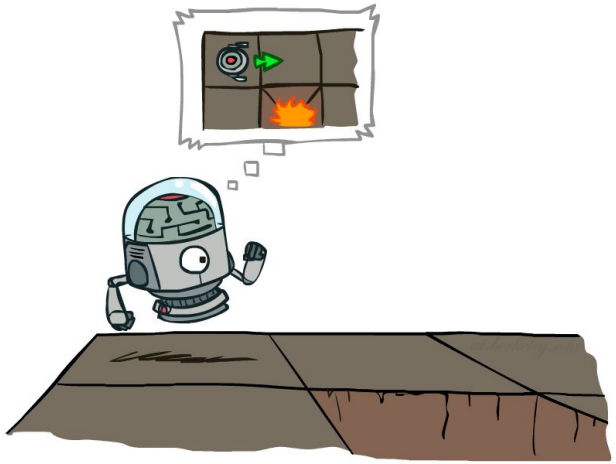


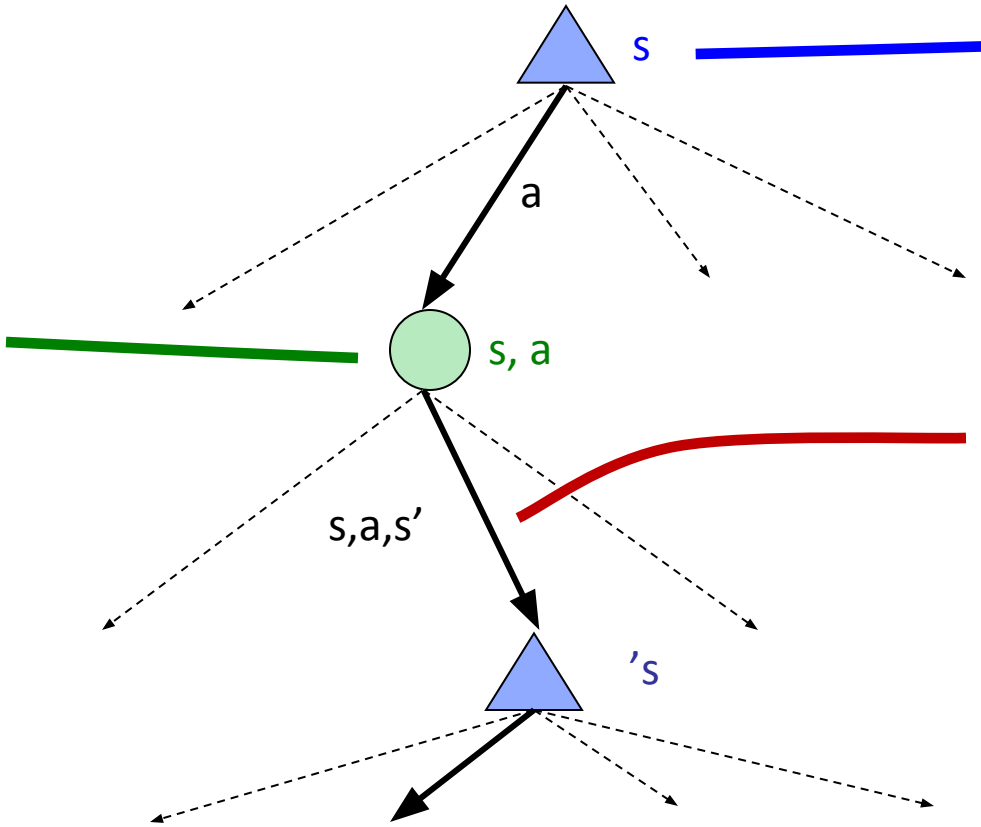
MDP Search Tree



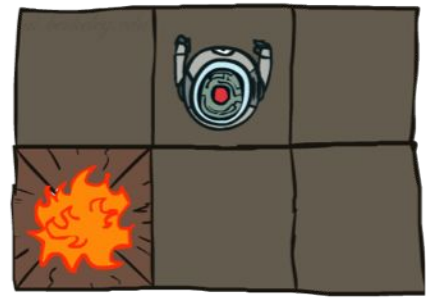
MDP Search Trees

- Each MDP state projects an expectimax-like search tree





s is a *state*

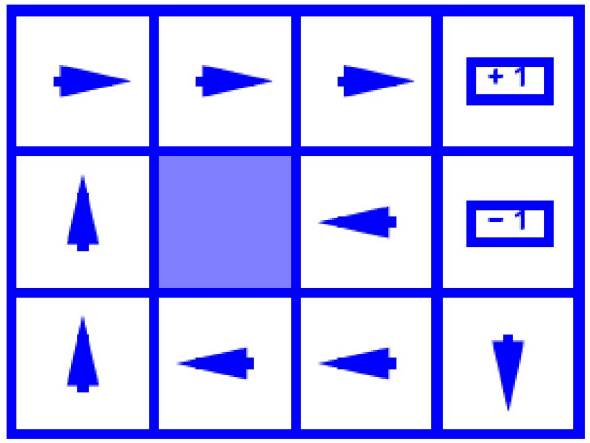


(s,a,s') called a *transition*

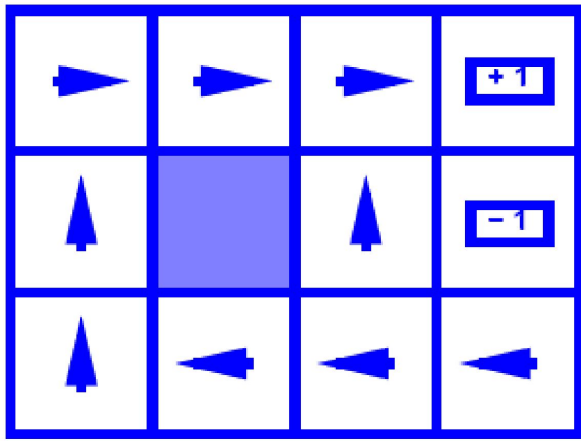
T(s,a,s') = P(s'|s,a)

R(s,a,s')

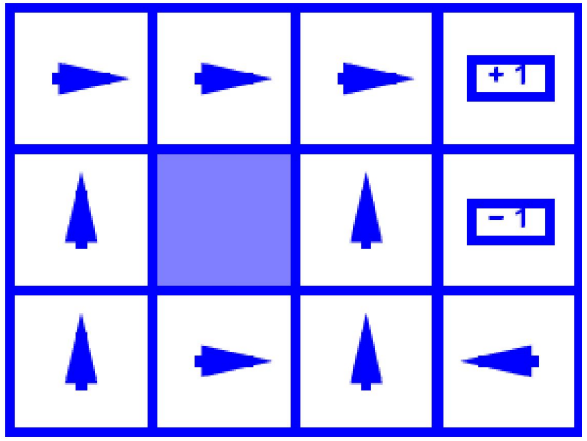
Optimal Policies



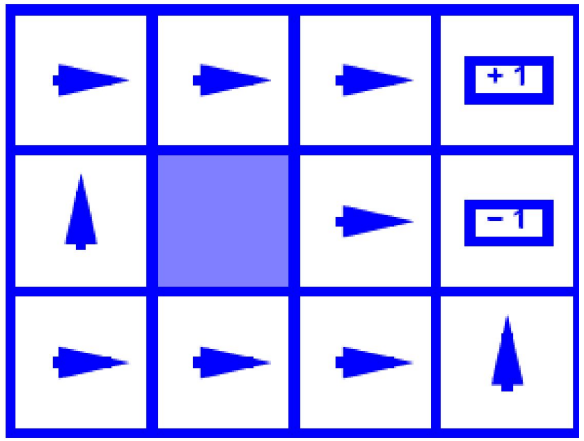
R(s) = -0.01



R(s) = -0.03



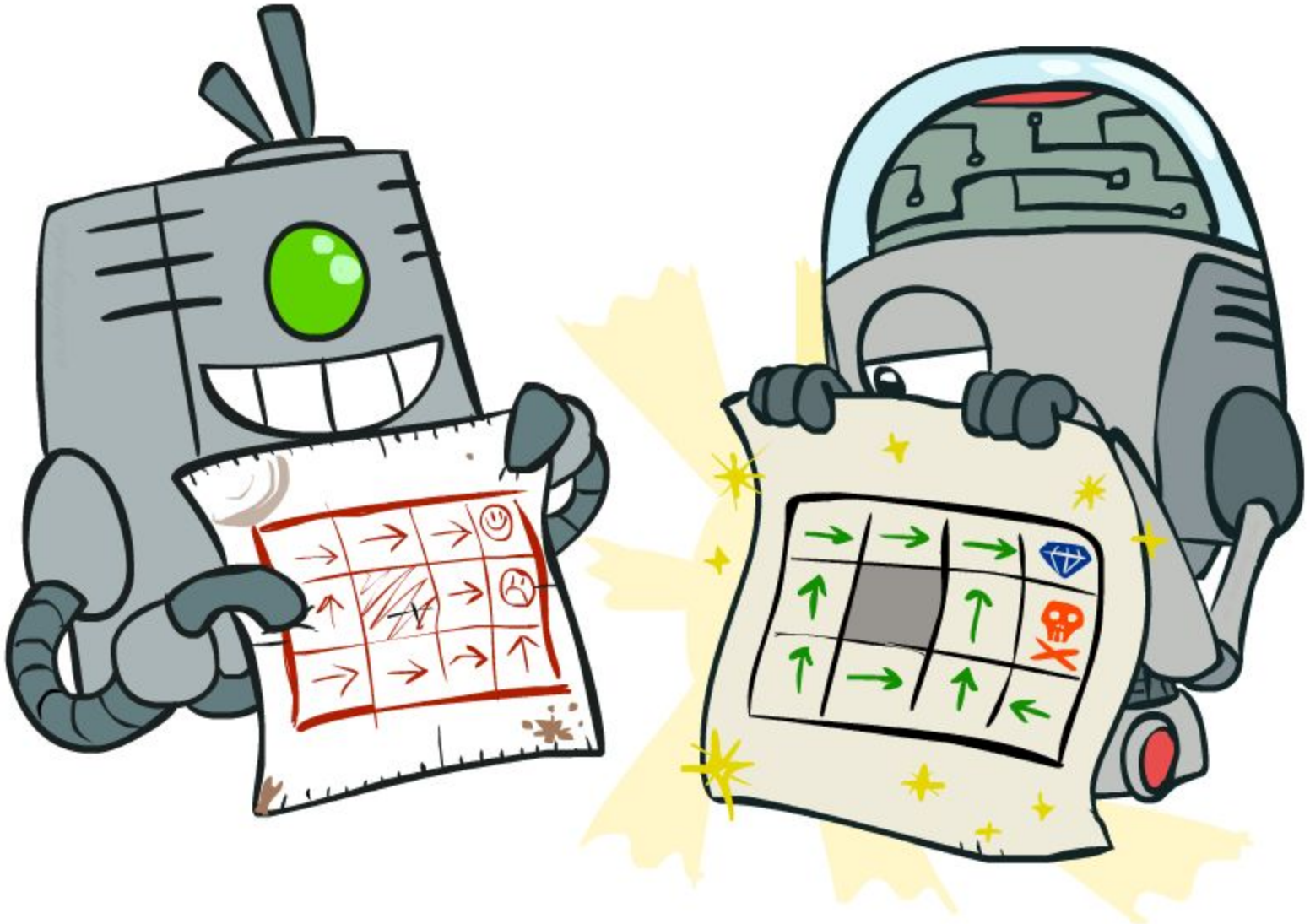
R(s) = -0.4



R(s) = -2.0

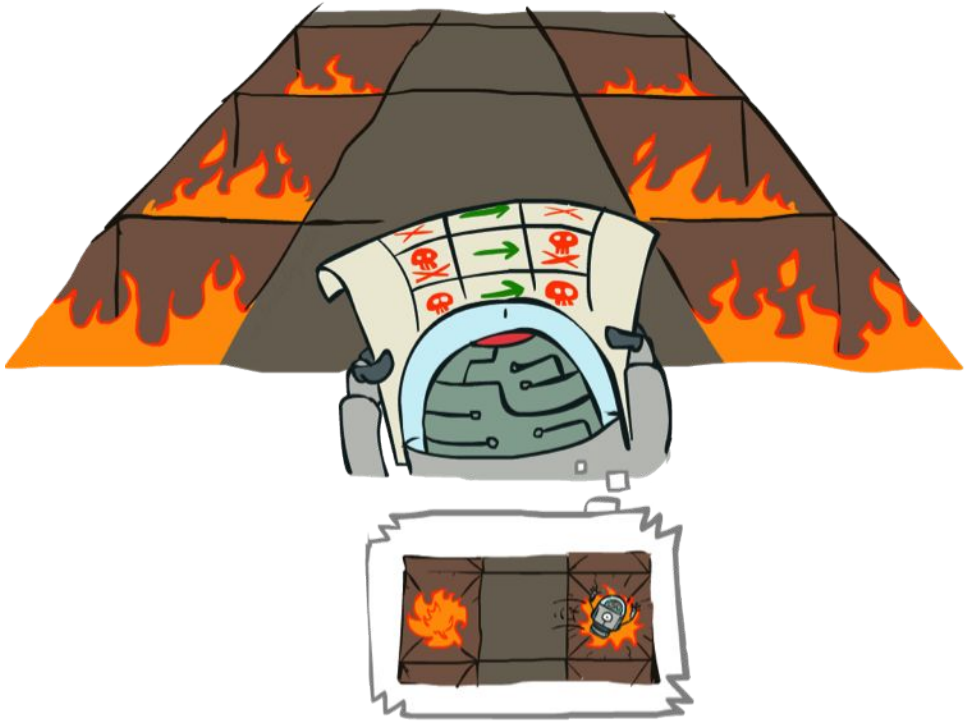
R(s) = the "living reward"

Policy Evaluation

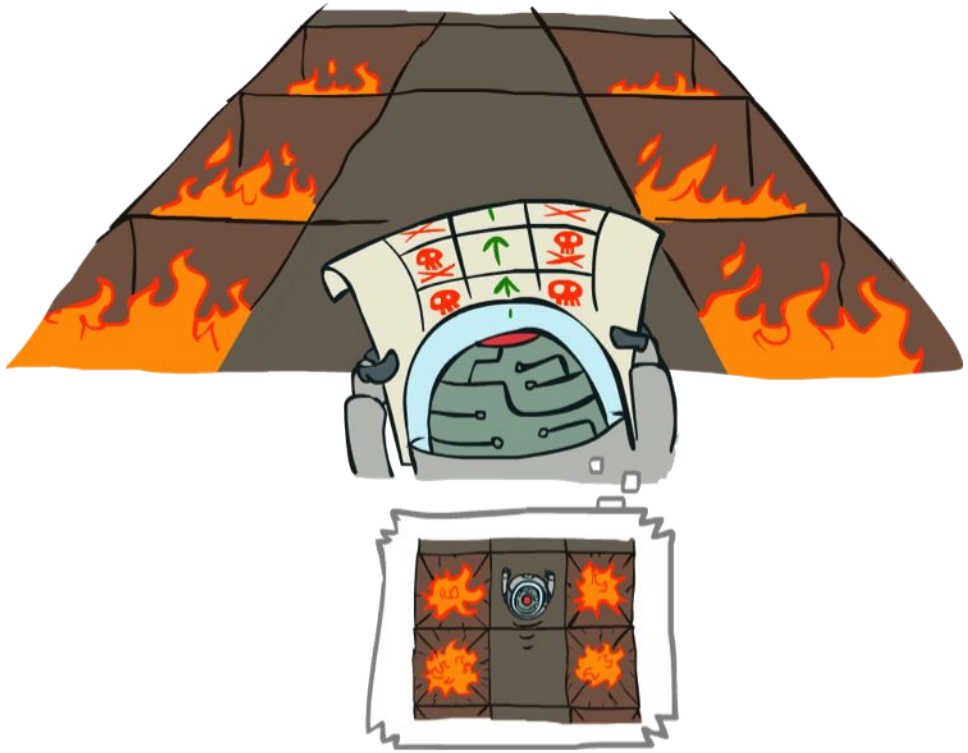


Example: Policy Evaluation

Always Go Right



Always Go Forward



Example: Policy Evaluation

π_1

s $\pi_1(s)$

State	Action
(2,1)	“east”
(2,2)	“east”
(2,3)	“east”
(2,4)	“exit”
(1,3)	“exit”
…	…

s $\pi_2(s)$

State	Action
(2,1)	“north”
(2,2)	“north”
(2,3)	“north”
(2,4)	“exit”
(1,3)	“exit”
…	…

π_2

Example: Policy Evaluation

π_1

s $\pi_1(s)$ $V^{\pi_1}(s)$

State	Action	Value
(2,1)	“east”	-9.06
(2,2)	“east”	-8.25
(2,3)	“east”	0.76
(2,4)	“exit”	100
(1,3)	“exit”	-10
…	…	…

$\leq$

s $\pi_2(s)$ $V^{\pi_2}(s)$

State	Action	Value
(2,1)	“north”	32.62
(2,2)	“north”	48.23
(2,3)	“north”	69.60
(2,4)	“exit”	100
(1,3)	“exit”	-10
…	…	…

π_2

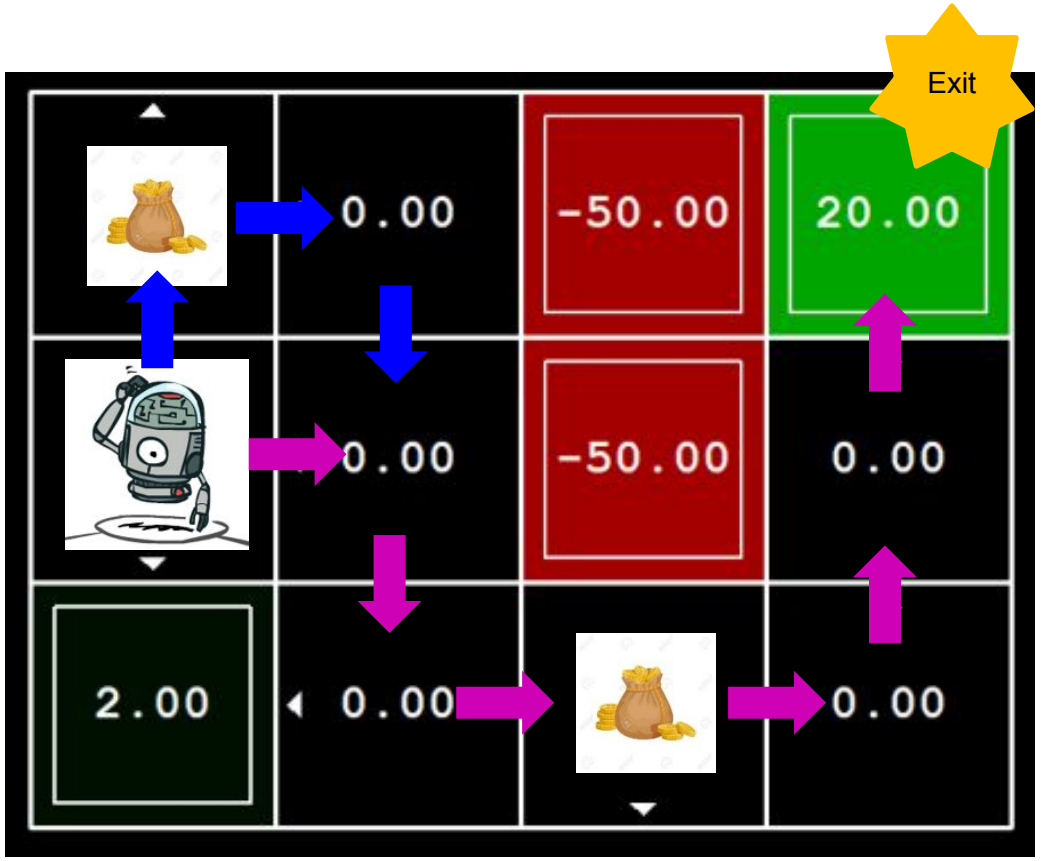
Policy Evaluation

- $V^{\pi}(s)$: the expected utility obtained by starting from state s and following policy π.

V^{π}: the state-value function for policy π.

$\pi_2 \geq \pi_1$ if and only if $V^{\pi_2}(s) \geq V^{\pi_1}(s)$, for all $s \in S$.

Utility of a sequence of actions

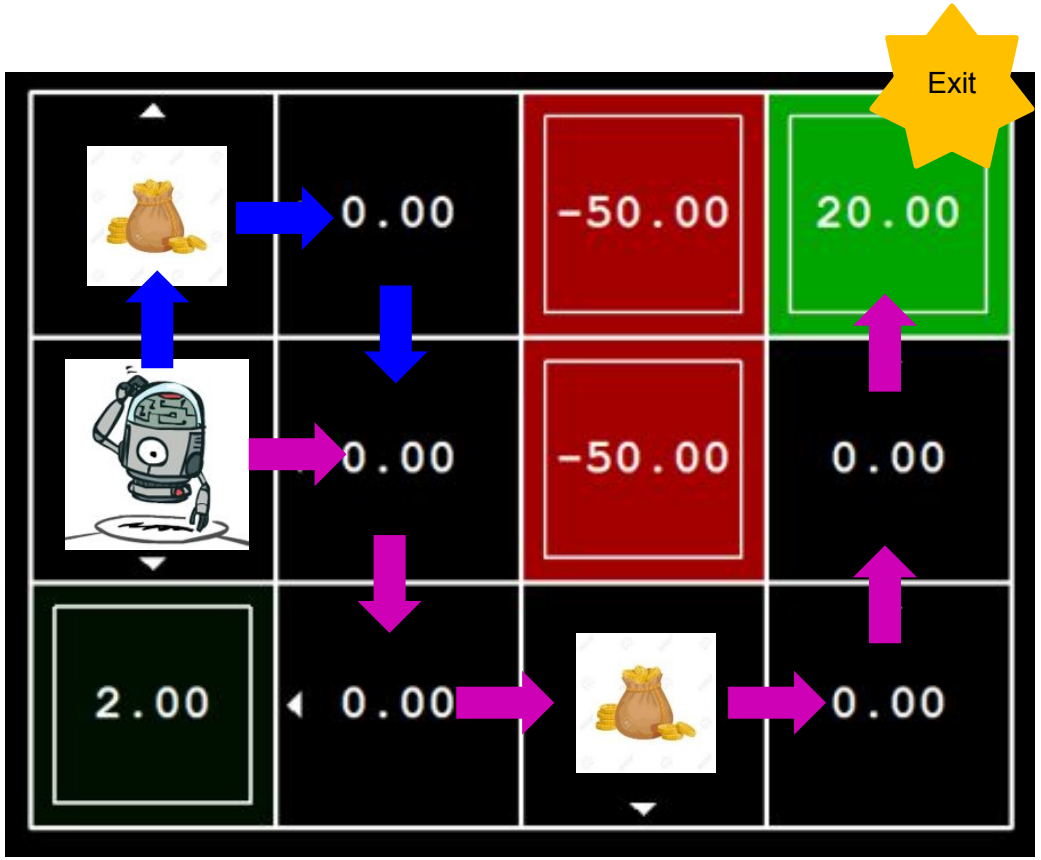


Living Reward = 0, Noise = 0

$$U = 0 + 0 + (0 + 0.5) + 0 + 0 + 0 + 20 = 20.5$$

$$U = (0 + 0.5) + 0 + 0 + 0 + (0 + 0.5) + 0 + 0 + 0 + 20 = 21$$

Utility of a sequence of actions



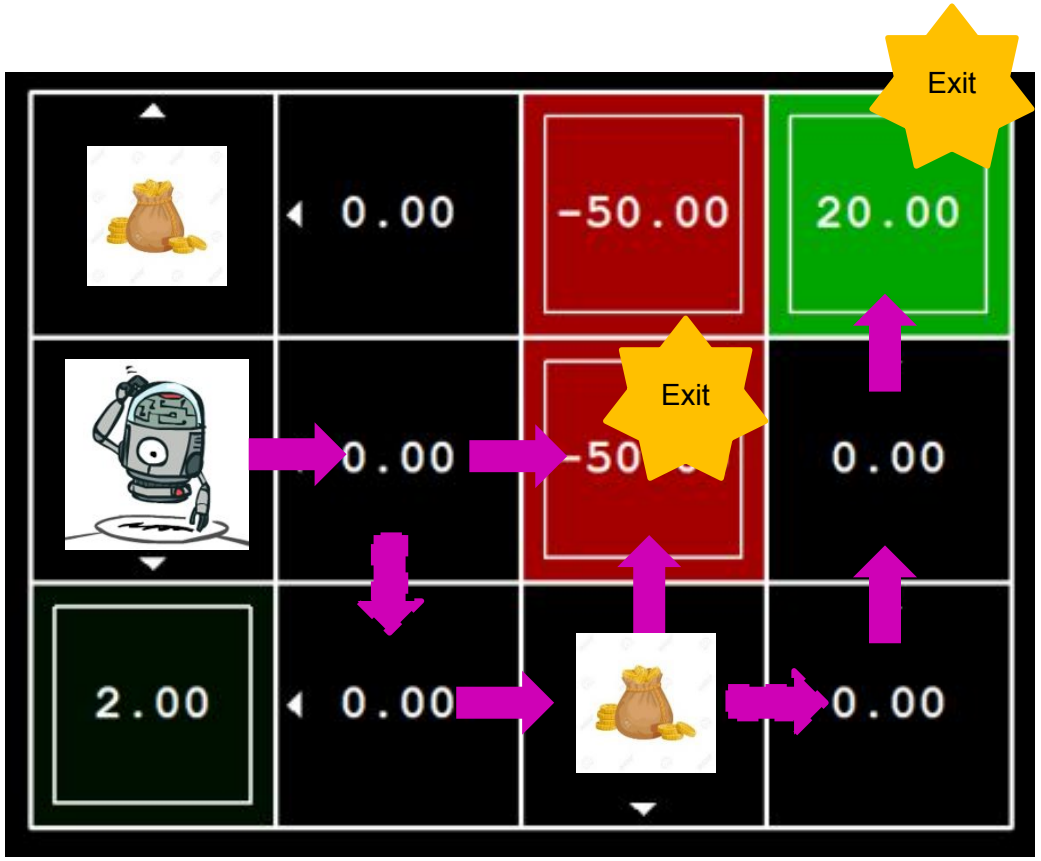
Living Reward = -0.3, Noise = 0.0

$$U = -0.3 - 0.3 + (-0.3 + 0.5) - 0.3 - 0.3 - 0.3 + 20 = 18.7$$

$$U = (-0.3 + 0.5) - 0.3 - 0.3 - 0.3 + (-0.3 + 0.5)$$
$$-0.3 - 0.3 - 0.3 + 20 = 18.6$$

Utility ~ Sum of Rewards

Utility of a sequence of actions



Living Reward = -0.3, Noise = 0.2

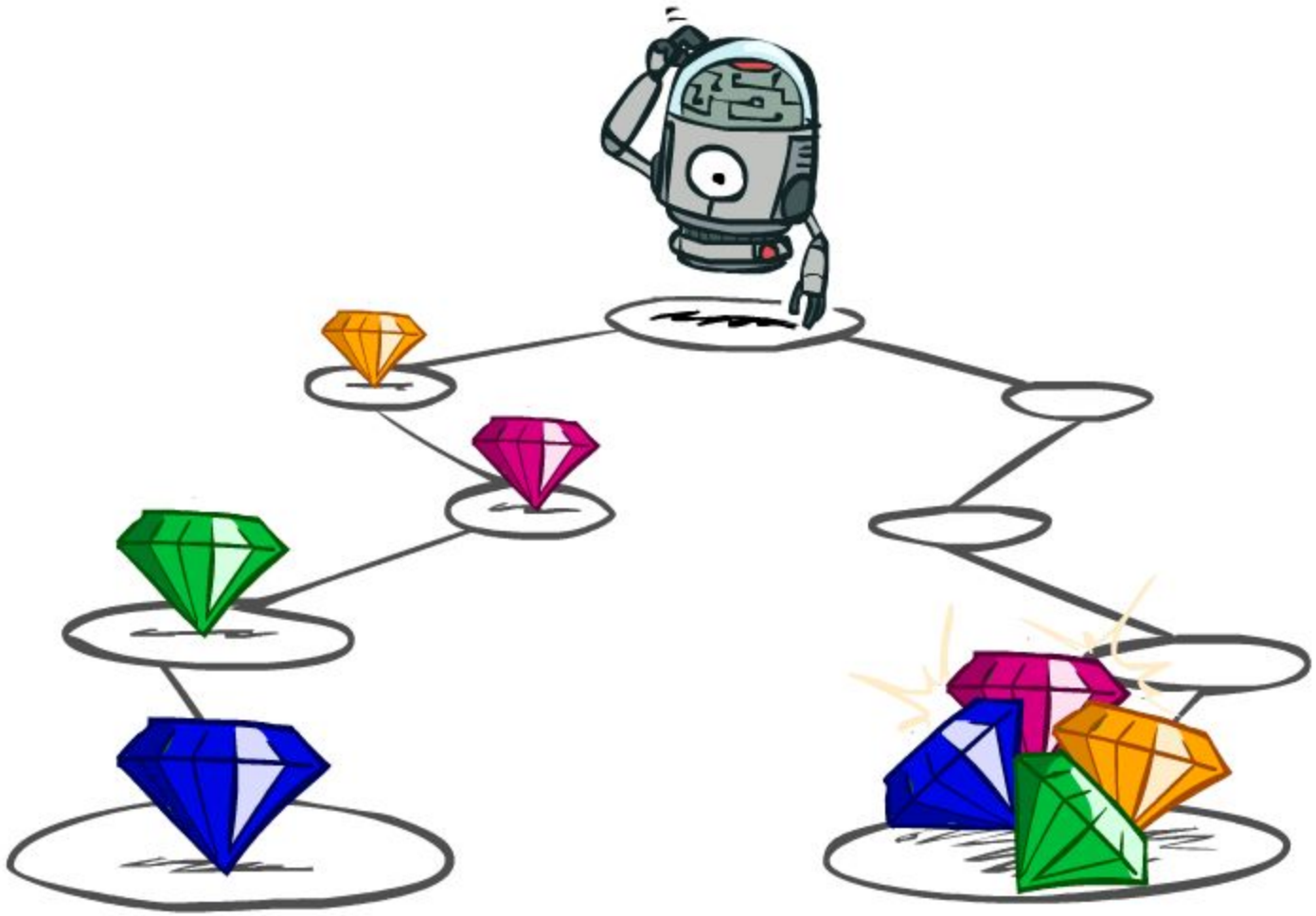
$$U = -0.3 - 0.3 + (-0.3 + 0.5) - 0.3 - 0.3 - 0.3 + 20 = 18.7$$

$$U = -0.3 - 0.3 + (-0.3 + 0.5) - 0.3 - 50 = -50.7$$

$$U = -0.3 - 0.3 - 50 = -50.6$$

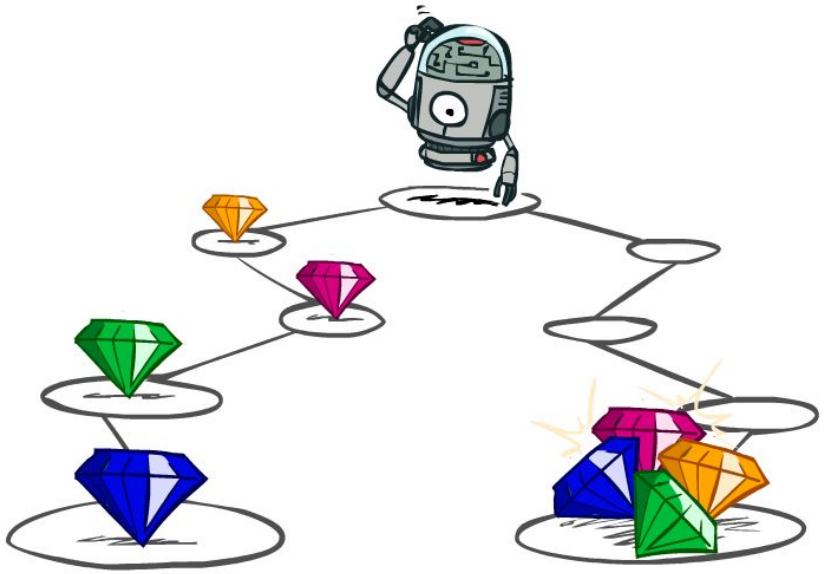
Expected Utility ~ Sum of Rewards

Utilities of Sequences



Utilities of Sequences

- What preferences should an agent have over reward sequences?
- More or less? [1, 2, 2] or [2, 3, 4]
- Now or later? [0, 0, 1] or [1, 0, 0]



Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- One solution: values of rewards decay exponentially

1

Worth Now

γ

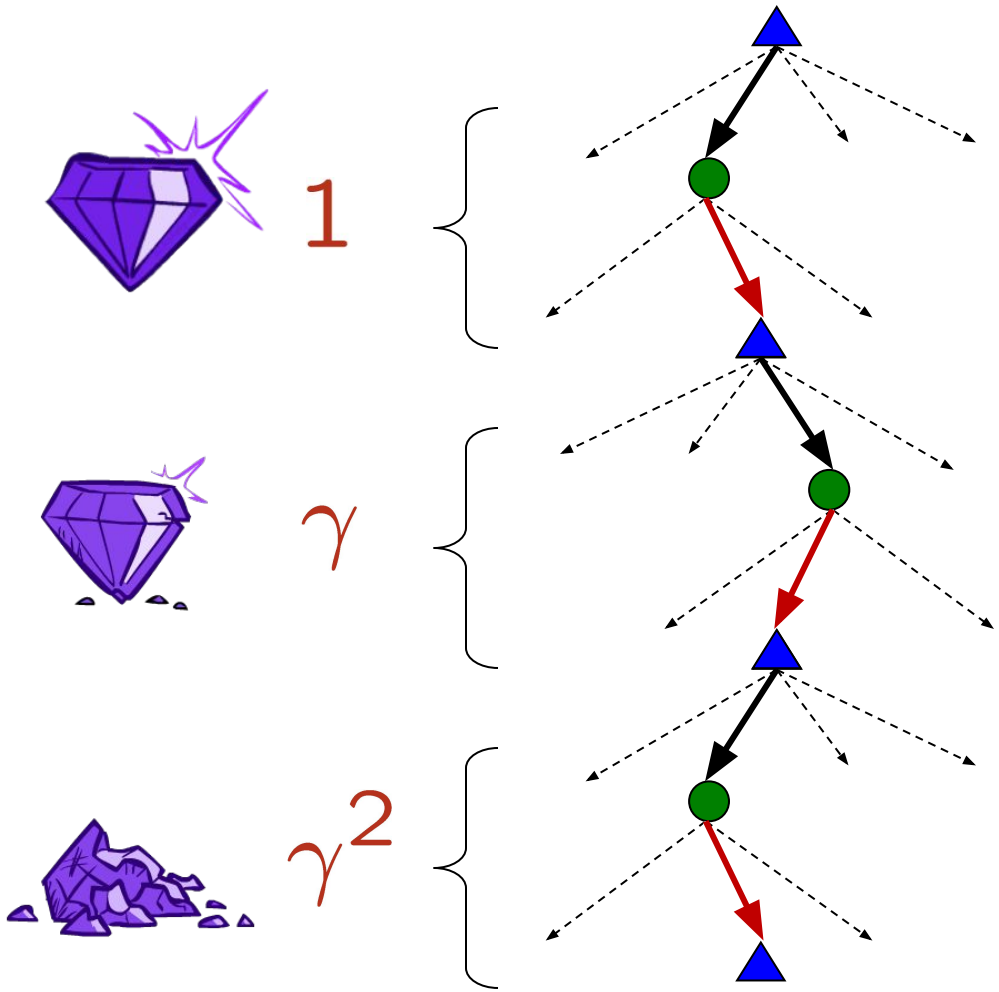
Worth Next Step

γ^2

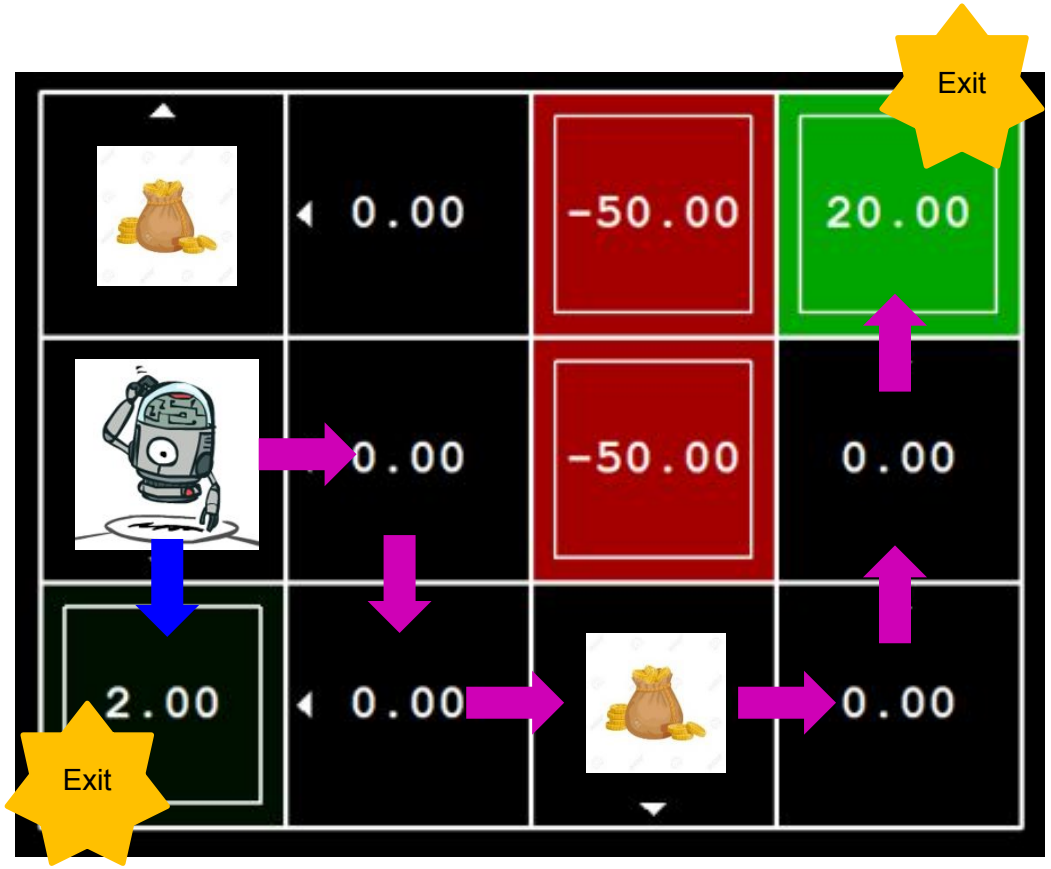
Worth In Two Steps

Discounting

- How to discount?
  - Each time we descend a level, we multiply in 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
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



Utility of a sequence of actions



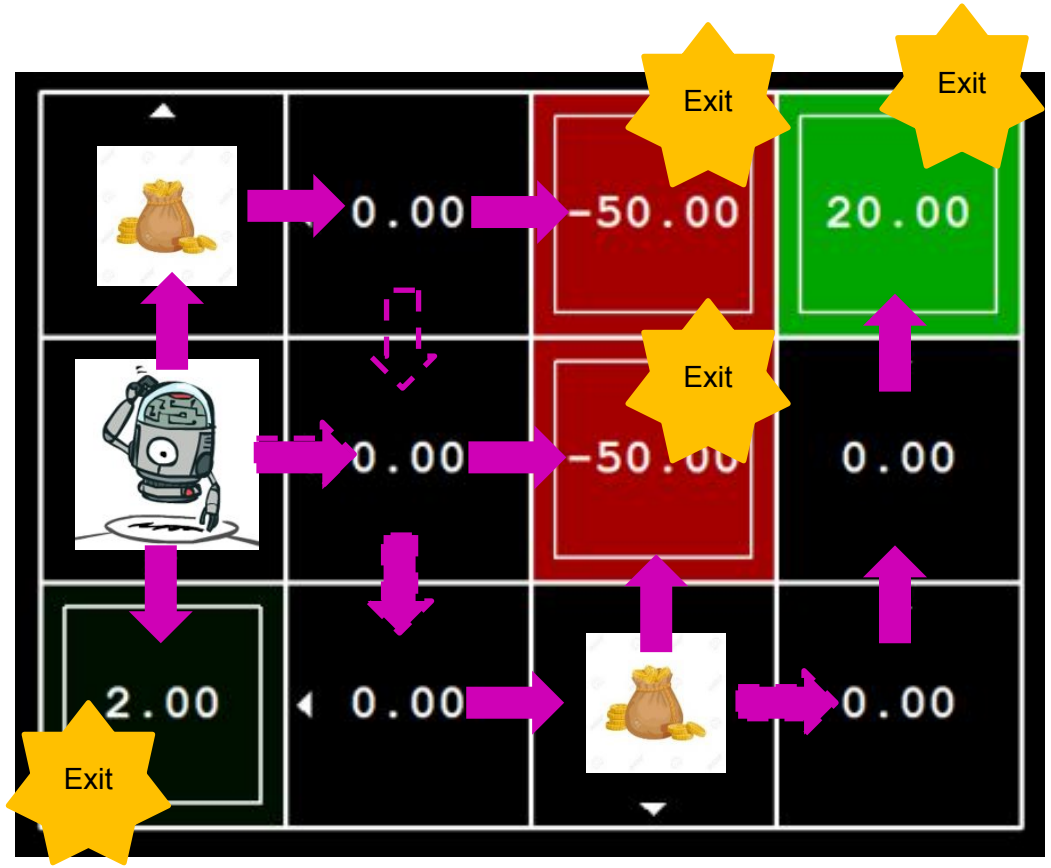
Living Reward = -0.3, Noise = 0.0

$$U = (-0.3) + (0.9) \times (-0.3) + (0.9)^2 \times (-0.3 + 0.5) + (0.9)^3 \times (-0.3) + (0.9)^4 \times (-0.3) + (0.9)^5 \times (-0.3) + (0.9)^6 \times 20 = 9.628$$

$$U = (-0.3) + (0.9) \times 2 = 1.5$$

Utility of Sum of Discounted Rewards

Utility of a sequence of actions



Living Reward = -0.3, Noise = 0.2

$$U = (-0.3) + (0.9) \times (-0.3) + (0.9)^2 \times (-0.3 + 0.5) + (0.9)^3 \times (-0.3) + (0.9)^4 \times (-0.3) + (0.9)^5 \times (-0.3) + (0.9)^6 \times 20 = 9.628$$

$$U = (-0.3) + (0.9) \times (-0.3) + (0.9)^2 \times (-0.3 + 0.5) + (0.9)^3 \times (-0.3) + (0.9)^4 \times (-50) = -33.432$$

$$U = (-0.3) + (0.9) \times (-0.3) + (0.9)^2 \times (-50) = -41.07$$

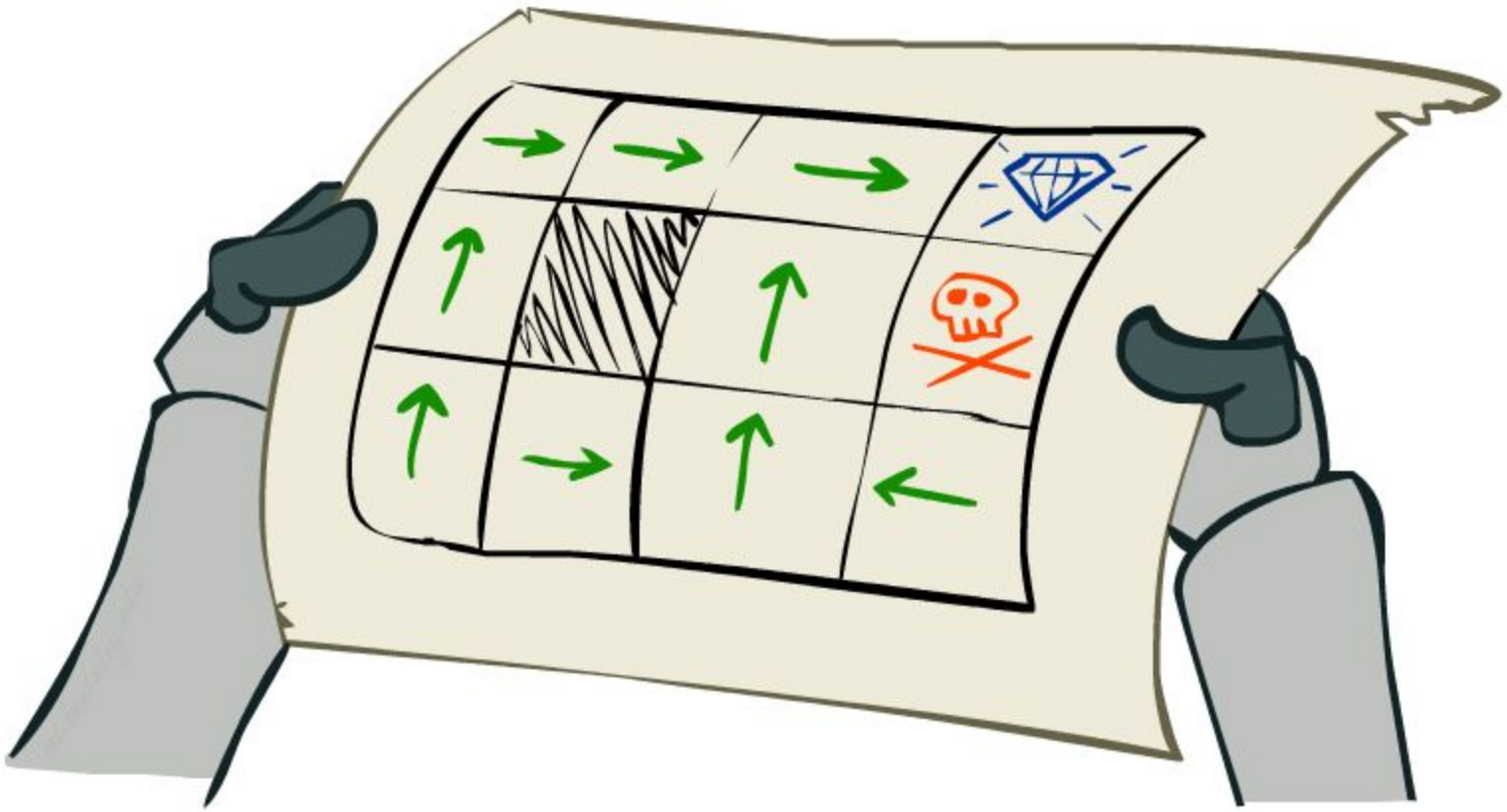
$$U = (-0.3) + (0.9) \times (2) = 1.5$$

$$U = (-0.3 + 0.5) + (0.9) \times (-0.3) + (0.9)^2 \times (-0.3) + (0.9)^3 \times (-50) = -36.763$$

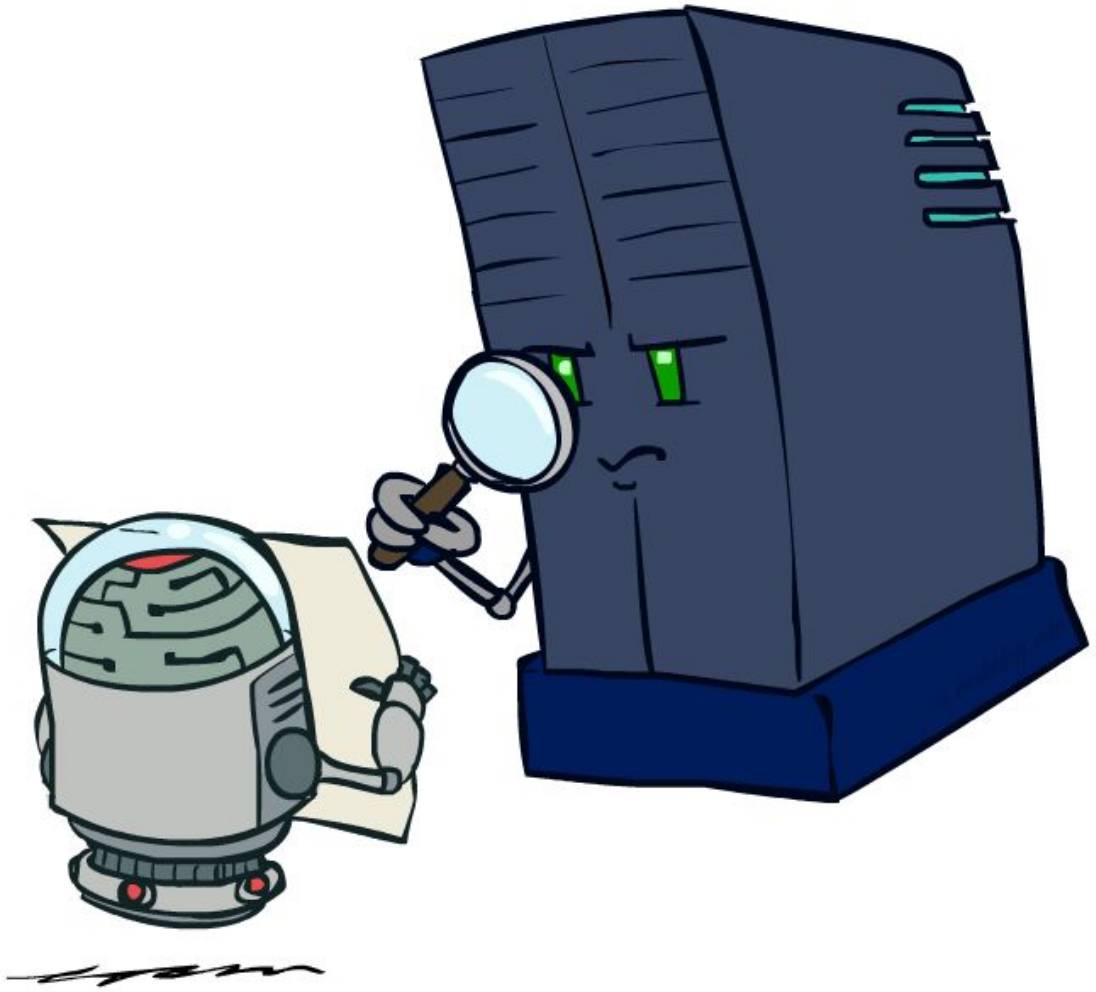
Expected Utility = Average sum of Discounted Rewards

Utility = sum of Discounted Rewards

Solving MDPs



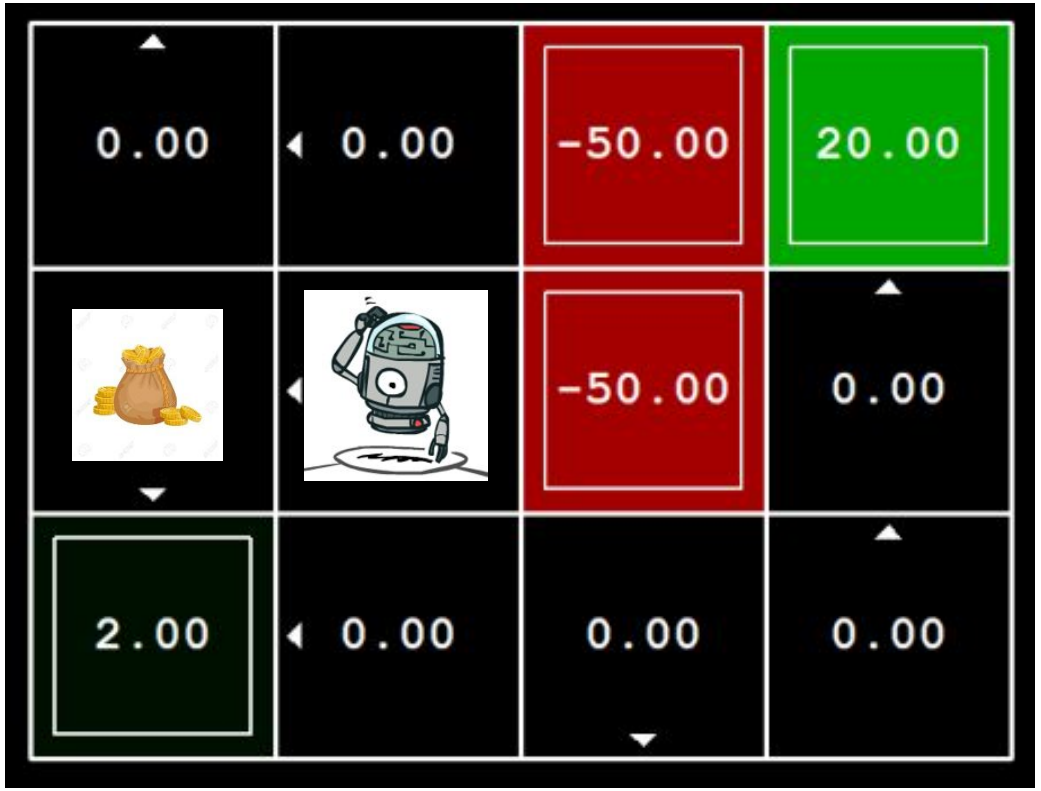
Policy Evaluation



Policy Evaluation

Regarding the current state,
how good is it according to a policy π?

state s = (2,2)



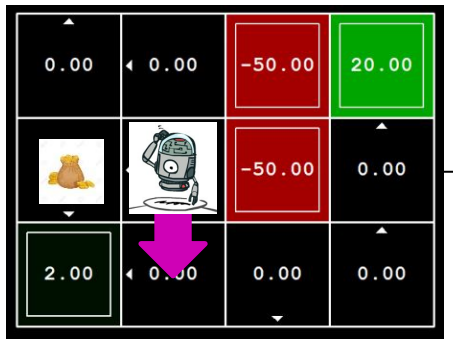
Policy Evaluation

$$\sum_{s'} P(s'|s,a) = 1.0$$

$V^\pi(s)$
expected utility
of state s=*(2,2)*
by following π

$$V^\pi(s) = Q^\pi(s,\pi(s)) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

state s = (2,2)



action a = "south"

$Q^\pi(s,a)$

$P(s_1'|s,a) = 0.8$

$R(s,a,s_1') = -0.3$

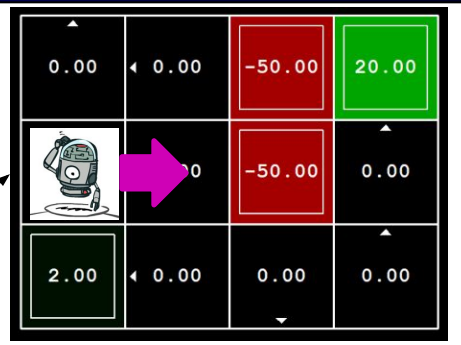
$\pi(s)$

State	Action
(2,2)	
(1,2)	"east"
(2,1)	"east"
(3,2)	"exit"
…	…

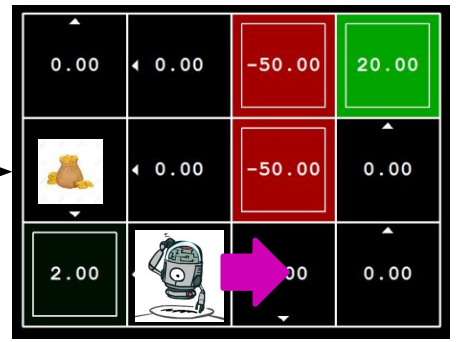
Expected Utility of taking action
a="south"* in state *s = (2,2)
and then following π

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$$

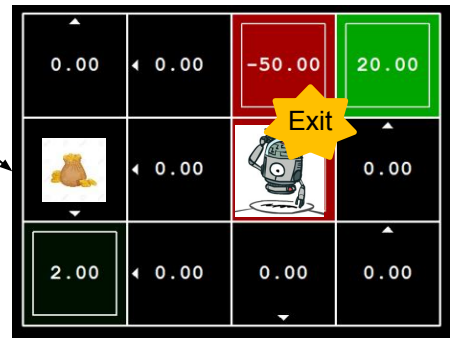
?



state s_2' = (1,2)
expected utility
of state *(1,2)*
by following π
$V^\pi(s_2')$



state s_1' = (2,1)
expected utility
of state *(2,1)*
by following π
$V^\pi(s_1')$

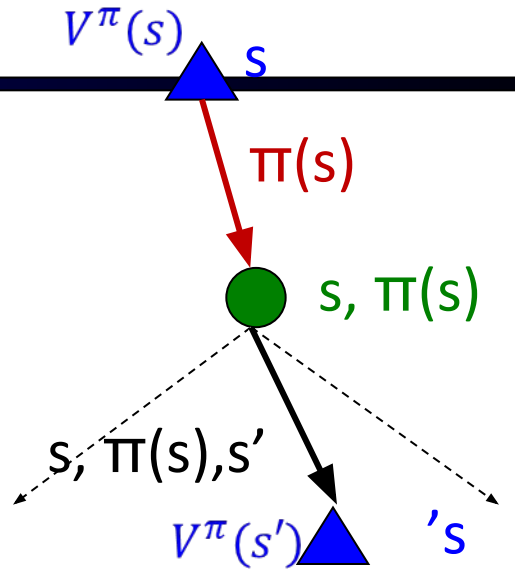


state s_3' = (3,2)
expected utility
of state *(3,2)*
by following π
$V^\pi(s_3')$

Policy Evaluation

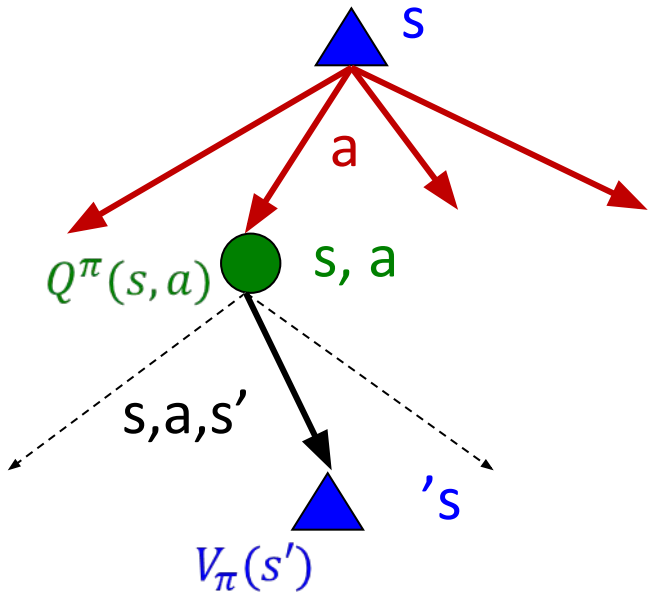
- $V^{\pi}(s)$: the expected utility obtained by starting from state s and following policy π.

V^{π}: the state-value function for policy π.



$Q^{\pi}(s,a)$: the expected utility obtained by starting from state s, taking action a, and then following policy π.

Q^{π}: the action-value function for policy π.



Policy Evaluation

$$V^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

$$V^{\pi}([2,4]) = 1.0(100 + 0.9(0.0)) = 100$$

$$V^{\pi}([1,3]) = 1.0(-10 + 0.9(0.0)) = -10$$

$$V^{\pi}([3,2]) = 1.0(-10 + 0.9(0.0)) = -10$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots.$$

$$V^{\pi}([2,1]) = 0.8\left(-0.3 + 0.9V^{\pi}([3,1])\right) + 0.1\left(-0.3 + 0.9V^{\pi}([2,2])\right) + 0.1\left(-0.3 + 0.9V^{\pi}([2,1])\right)$$

$$V^{\pi}([2,2]) = 0.8\left(-0.3 + 0.9V^{\pi}([3,2])\right) + 0.1\left(-0.3 + 0.9V^{\pi}([2,3])\right) + 0.1\left(-0.3 + 0.9V^{\pi}([2,1])\right)$$

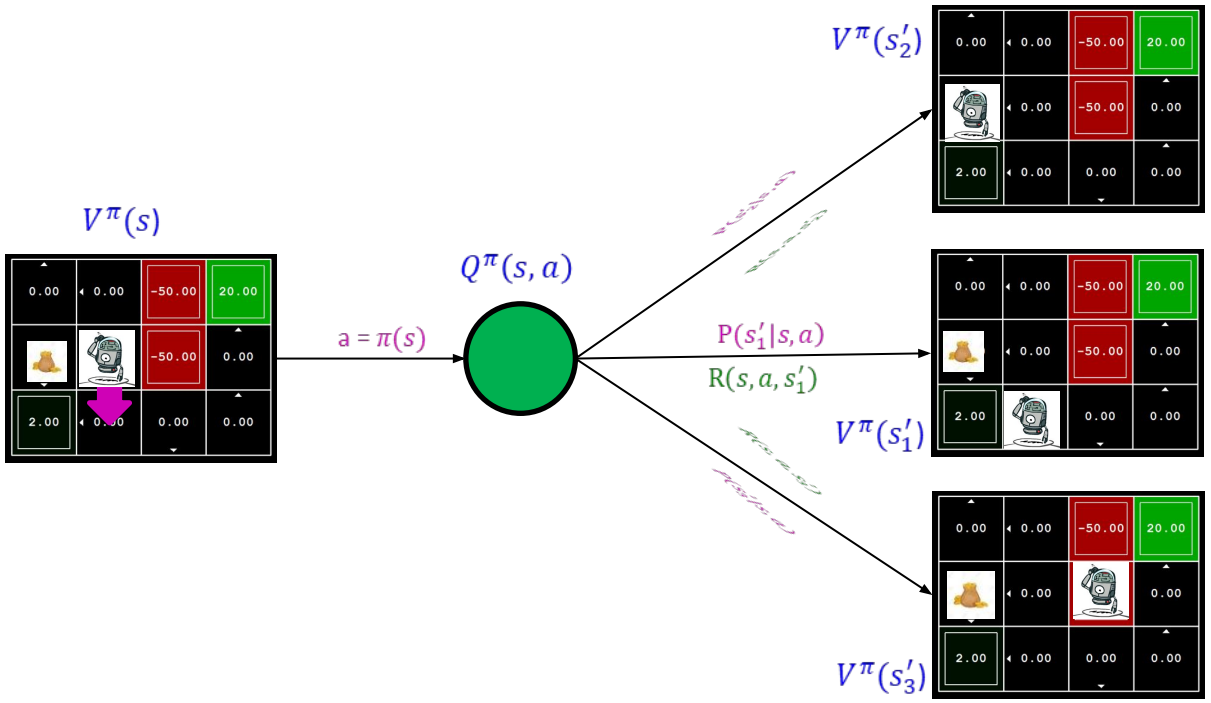
$$V^{\pi}([2,3]) = 0.8\left(-0.3 + 0.9V^{\pi}([3,3])\right) + 0.1\left(-0.3 + 0.9V^{\pi}([2,4])\right) + 0.1\left(-0.3 + 0.9V^{\pi}([2,2])\right)$$

[2,1]	“east”
[2,2]	“east”
[2,3]	“east”
[2,4]	“exit”
[1,3]	“exit”
…	…

Solving a system of linear equations

(Iterative) Policy Evaluation

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s)) = \sum_{s'} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$



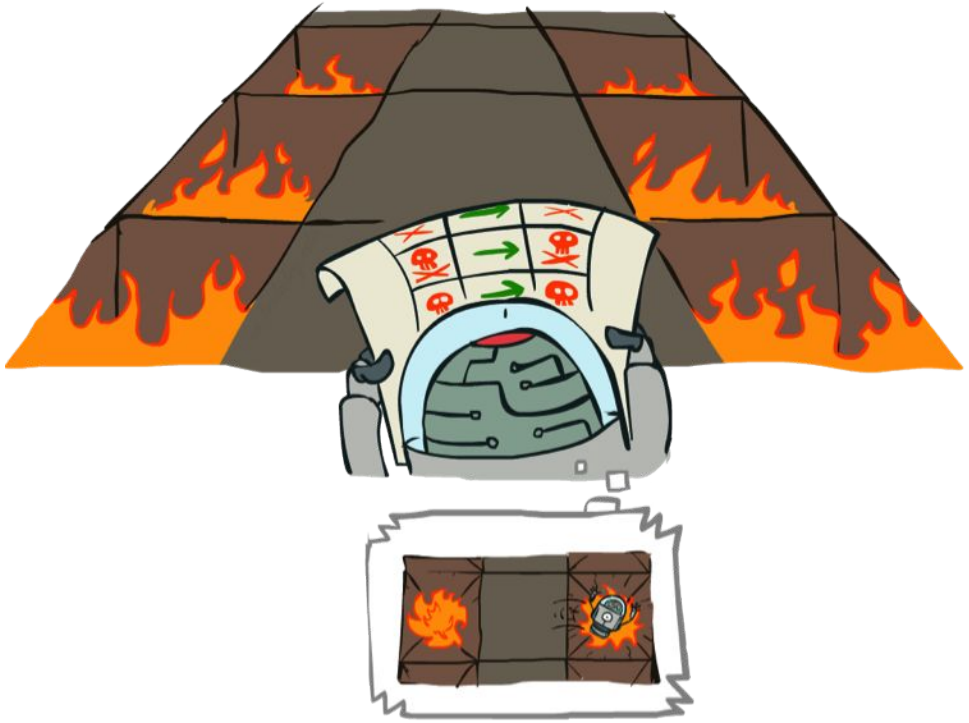
Iterative Approach

1. Initialize $V_0^{\pi}(s) \leftarrow 0$ for all states s.
2. Repeat until convergence: $t = 1, 2, 3, \ldots$
   // While $\max_s(|V_t^{\pi}(s) - V_{t-1}^{\pi}(s)|) > \varepsilon \approx 0.00001$
   - For each state s:

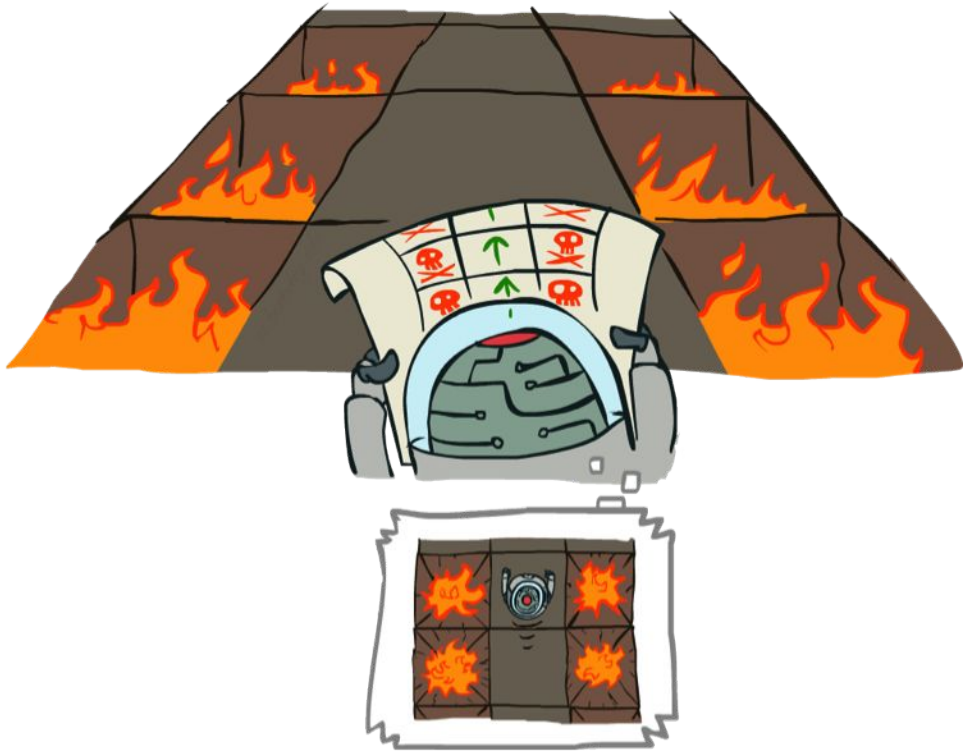
$$V_t^{\pi}(s) \leftarrow \underbrace{\sum_{s'} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma V_{t-1}^{\pi}(s')]}_{Q_{t-1}^{\pi}(s, \pi(s))}$$

Example: Policy Evaluation

Always Go Right



Always Go Forward



Example: Policy Evaluation

Noise = 0.2
Discount = 0.9
Living reward = -0.3

π_1

$\pi_1(s)$

(2,1)	“east”
(2,2)	“east”
(2,3)	“east”
(2,4)	“exit”
(1,3)	“exit”
…	…

$\pi_2(s)$

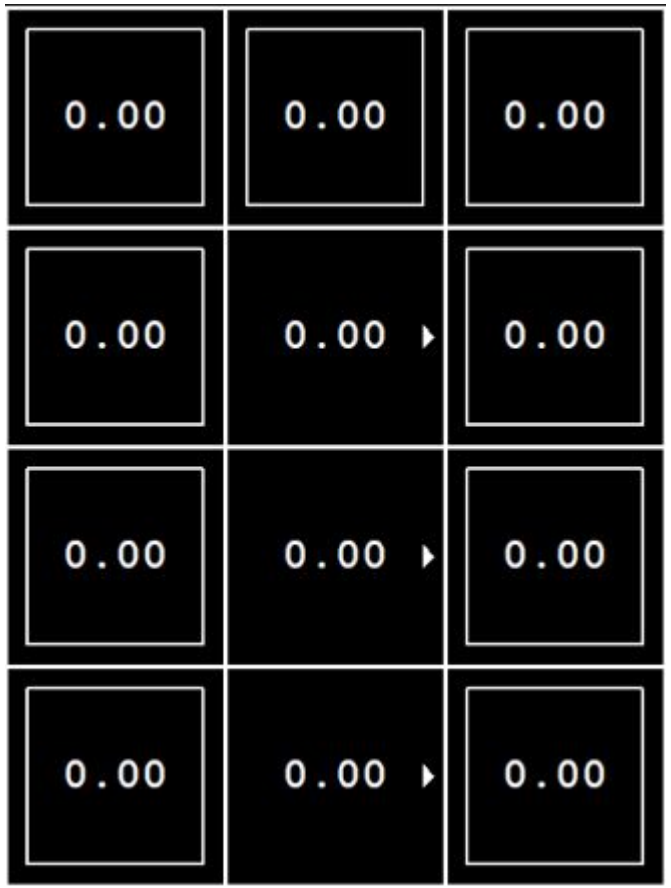
(2,1)	“north”
(2,2)	“north”
(2,3)	“north”
(2,4)	“exit”
(1,3)	“exit”
…	…

π_2

t=0

Noise = 0.2
Discount = 0.9
Living reward = -0.3

$$V_0^{\pi}(s) \leftarrow 0$$



t=0 □ t=1

Noise = 0.2
Discount = 0.9
Living reward = -0.3

$$V_t^\pi(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^\pi(s')]$$

$$Q_{t-1}^\pi(s,\pi(s))$$

$V_{t-1}^\pi(s)$

0.00	0.00	0.00
0.00	0.00 ▸	0.00
0.00	0.00 ▸	0.00
0.00	0.00 ▸	0.00

$$V_1^\pi(2,4) = 100$$

$$V_1^\pi(1,1) = -10$$

$$V_1^\pi(2,2) = (0.8) \times \left(-0.3 + 0.9V_0^\pi(3,2)\right) + (0.1) \times \left(-0.3 + 0.9V_0^\pi(2,3)\right) + (0.1) \times \left(-0.3 + 0.9V_0^\pi(2,1)\right) = -0.3$$

$V_t^\pi(s)$

-10.00	100.00	-10.00
-10.00	-0.30 ▸	-10.00
-10.00	-0.30 ▸	-10.00
-10.00	-0.30 ▸	-10.00

t=1 □ t=2

Noise = 0.2
Discount = 0.9
Living reward = -0.3

$$V_t^\pi(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^\pi(s')]$$

$$Q_{t-1}^\pi(s,\pi(s))$$

$V_{t-1}^\pi(s)$

-10.00	100.00	-10.00
-10.00	-0.30 ▸	-10.00
-10.00	-0.30 ▸	-10.00
-10.00	-0.30 ▸	-10.00

$$V_2^\pi(2,2) = (0.8) \times \left(-0.3 + 0.9V_1^\pi(3,2)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_1^\pi(2,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_1^\pi(2,1)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(-0.3))$$
$$+(0.1) \times (-0.3 + 0.9(-0.3)) \approx -7.55$$

$$V_2^\pi(2,3) = (0.8) \times \left(-0.3 + 0.9V_1^\pi(3,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_1^\pi(2,4)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_1^\pi(2,2)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(100))$$
$$+(0.1) \times (-0.3 + 0.9(-0.3)) \approx 1.47$$

$V_t^\pi(s)$

-10.00	100.00	-10.00
-10.00	1.47 ▸	-10.00
-10.00	-7.55 ▸	-10.00
-10.00	-7.55 ▸	-10.00

t=2 □ t=3

Noise = 0.2
Discount = 0.9
Living reward = -0.3

$$V_t^\pi(s) \leftarrow \underbrace{\sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^\pi(s')]}_{Q_{t-1}^\pi(s,\pi(s))}$$

$V_{t-1}^\pi(s)$

-10.00	100.00	-10.00
-10.00	1.47 ▸	-10.00
-10.00	-7.55 ▸	-10.00
-10.00	-7.55 ▸	-10.00

$$V_3^\pi(2,2) = (0.8) \times \left(-0.3 + 0.9V_2^\pi(3,2)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_2^\pi(2,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_2^\pi(2,1)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(1.47))$$
$$+(0.1) \times (-0.3 + 0.9(-7.55)) \approx -8.05$$

$$V_3^\pi(2,1) = (0.8) \times \left(-0.3 + 0.9V_2^\pi(3,1)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_2^\pi(2,2)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_2^\pi(2,1)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(-7.55))$$
$$+(0.1) \times (-0.3 + 0.9(-7.55)) \approx -8.86$$

$V_t^\pi(s)$

-10.00	100.00	-10.00
-10.00	0.82 ▸	-10.00
-10.00	-8.05 ▸	-10.00
-10.00	-8.86 ▸	-10.00

t=3 → t=4

Noise = 0.2
Discount = 0.9
Living reward = -0.3

$$V_t^{\pi}(s) \leftarrow \underbrace{\sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^{\pi}(s')]}_{Q_{t-1}^{\pi}(s,\pi(s))}$$

$V_{t-1}^{\pi}(s)$

-10.00	100.00	-10.00
-10.00	0.82 ▸	-10.00
-10.00	-8.05 ▸	-10.00
-10.00	-8.86 ▸	-10.00

$$V_4^{\pi}(2,2) = (0.8) \times \left(-0.3 + 0.9V_3^{\pi}(3,2)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_3^{\pi}(2,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_3^{\pi}(2,1)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(0.82))$$
$$+(0.1) \times (-0.3 + 0.9(-8.86)) \approx -8.22$$

$$V_4^{\pi}(2,1) = (0.8) \times \left(-0.3 + 0.9V_3^{\pi}(3,1)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_3^{\pi}(2,2)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_3^{\pi}(2,1)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(-8.05))$$
$$+(0.1) \times (-0.3 + 0.9(-8.86)) \approx -9.02$$

$V_t^{\pi}(s)$

-10.00	100.00	-10.00
-10.00	0.78 ▸	-10.00
-10.00	-8.22 ▸	-10.00
-10.00	-9.02 ▸	-10.00

t=4 □ t=5

Noise = 0.2
Discount = 0.9
Living reward = -0.3

$$V_t^{\pi}(s) \leftarrow \underbrace{\sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^{\pi}(s')]}_{Q_{t-1}^{\pi}(s,\pi(s))}$$

$V_{t-1}^{\pi}(s)$

-10.00	100.00	-10.00
-10.00	0.78 ▸	-10.00
-10.00	-8.22 ▸	-10.00
-10.00	-9.02 ▸	-10.00

$$V_5^{\pi}(2,2) = (0.8) \times \left(-0.3 + 0.9V_4^{\pi}(3,2)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_4^{\pi}(2,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_4^{\pi}(2,1)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(0.78))$$
$$+(0.1) \times (-0.3 + 0.9(-9.02)) \approx -8.24$$

$$V_5^{\pi}(2,3) = (0.8) \times \left(-0.3 + 0.9V_4^{\pi}(3,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_4^{\pi}(2,4)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_4^{\pi}(2,2)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(100))$$
$$+(0.1) \times (-0.3 + 0.9(-8.22)) \approx 0.76$$

$V_t^{\pi}(s)$

-10.00	100.00	-10.00
-10.00	0.76 ▸	-10.00
-10.00	-8.24 ▸	-10.00
-10.00	-9.05 ▸	-10.00

t=5 □ t=6

Noise = 0.2
Discount = 0.9
Living reward = -0.3

$$V_t^\pi(s) \leftarrow \underbrace{\sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^\pi(s')]}_{Q_{t-1}^\pi(s,\pi(s))}$$

$V_{t-1}^\pi(s)$

-10.00	100.00	-10.00
-10.00	0.76 ▸	-10.00
-10.00	-8.24 ▸	-10.00
-10.00	-9.05 ▸	-10.00

$$V_6^\pi(2,2) = (0.8) \times \left(-0.3 + 0.9V_5^\pi(3,2)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_5^\pi(2,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_5^\pi(2,1)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(0.76))$$
$$+(0.1) \times (-0.3 + 0.9(-9.05)) \approx -8.25$$

$$V_6^\pi(2,3) = (0.8) \times \left(-0.3 + 0.9V_5^\pi(3,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_5^\pi(2,4)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_5^\pi(2,2)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(100))$$
$$+(0.1) \times (-0.3 + 0.9(-8.24)) \approx 0.76$$

$V_t^\pi(s)$

-10.00	100.00	-10.00
-10.00	0.76 ▸	-10.00
-10.00	-8.25 ▸	-10.00
-10.00	-9.06 ▸	-10.00

t=6 → t=7

Noise = 0.2
Discount = 0.9
Living reward = -0.3

$$V_t^\pi(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^\pi(s')]$$

$$Q_{t-1}^\pi(s,\pi(s))$$

$V_{t-1}^\pi(s)$

-10.00	100.00	-10.00
-10.00	0.76 ▸	-10.00
-10.00	-8.25 ▸	-10.00
-10.00	-9.06 ▸	-10.00

$$V_7^\pi(2,2) = (0.8) \times \left(-0.3 + 0.9V_6^\pi(3,2)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_6^\pi(2,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_6^\pi(2,1)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(0.76))$$
$$+(0.1) \times (-0.3 + 0.9(0.05)) \approx 8.25$$

$$\max_s(|V_t^\pi(s) - V_{t-1}^\pi(s)|) < \varepsilon$$

$$V_7^\pi(2,3) = (0.8) \times \left(-0.3 + 0.9V_6^\pi(3,3)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_6^\pi(2,4)\right)$$
$$+(0.1) \times \left(-0.3 + 0.9V_6^\pi(2,2)\right)$$
$$= (0.8) \times (-0.3 + 0.9(-10))$$
$$+(0.1) \times (-0.3 + 0.9(100))$$
$$+(0.1) \times (-0.3 + 0.9(-8.25)) \approx 0.76$$

$V_t^\pi(s)$

-10.00	100.00	-10.00
-10.00	0.76 ▸	-10.00
-10.00	-8.25 ▸	-10.00
-10.00	-9.06 ▸	-10.00

Example: Policy Evaluation

Noise = 0.2
Discount = 0.9
Living reward = -0.3

Always Go Right

$V^{\pi_1}(s)$

$\geq$

Always Go Forward

$V^{\pi_2}(s)$

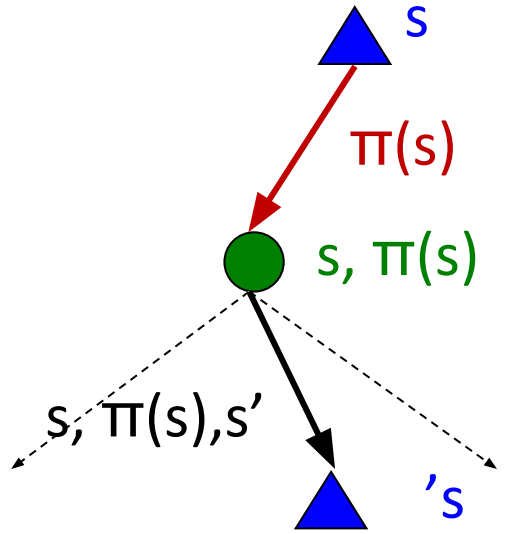
Summary: Policy Evaluation

- How do we calculate $V^{\pi}(s)$ for a fixed policy π?

- Idea 1: Turn recursive Bellman equations into updates

$$V_0^{\pi}(s) \leftarrow 0$$

$$V_t^{\pi}(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^{\pi}(s')]$$

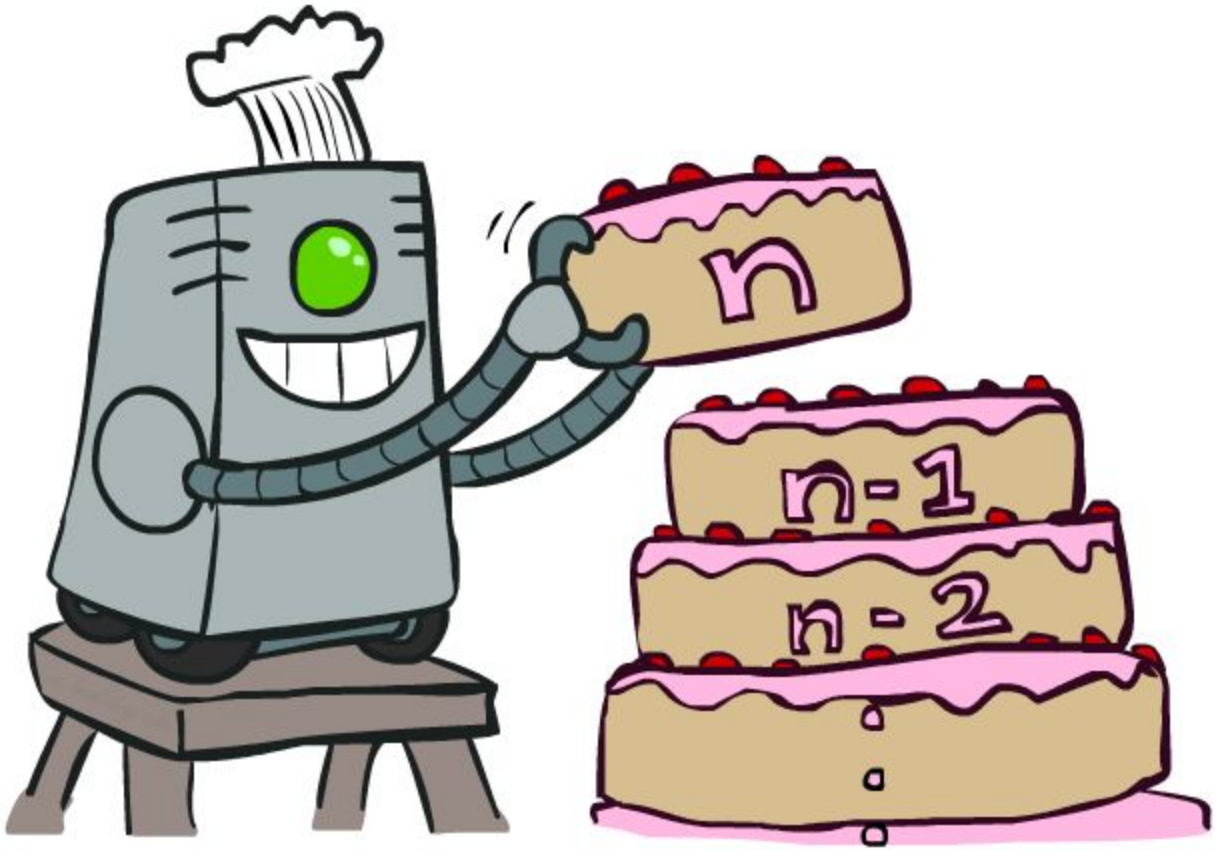


- Efficiency: $O(S^2)$ per iteration

- Idea 2: Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

$$V^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

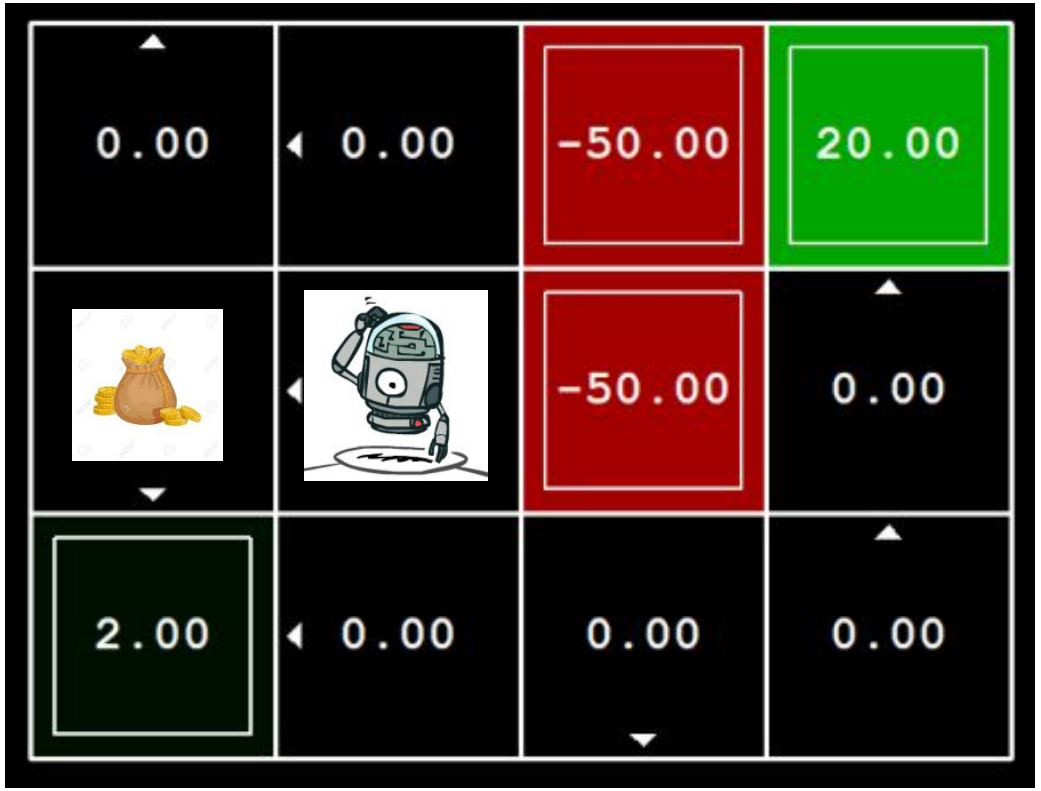
Value Iteration



Policy Evaluation

Regarding the current state,
how good is it according to a policy π?

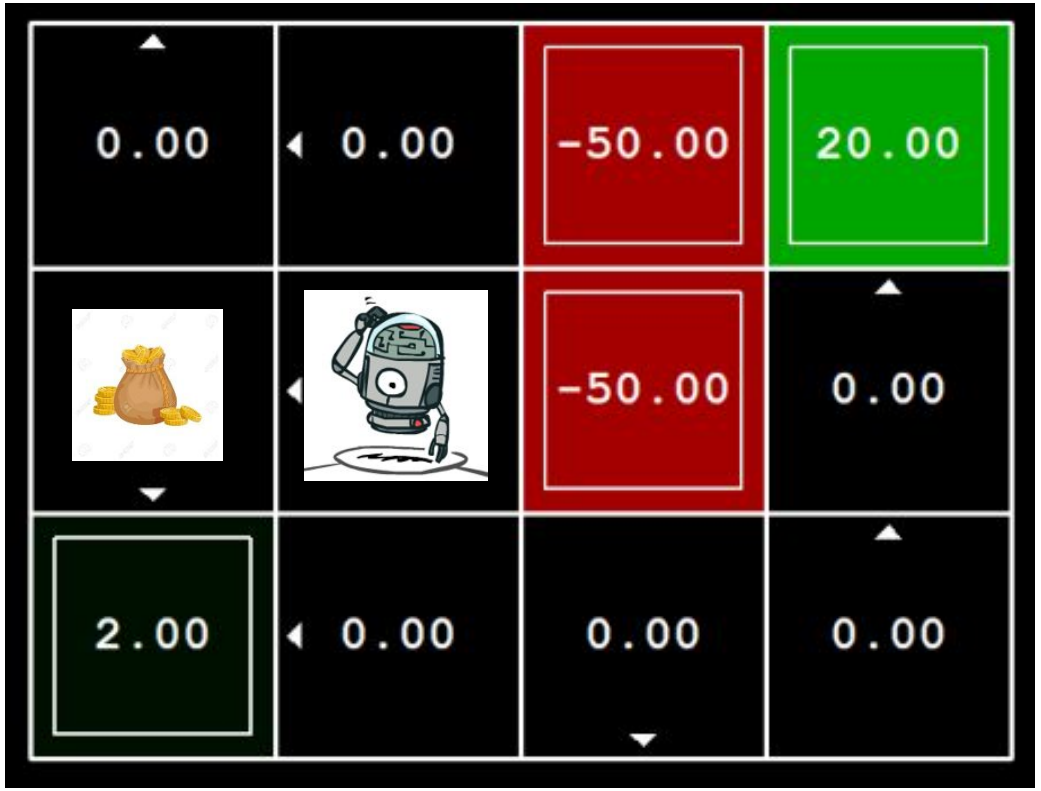
state s = (2,2)



Value Iteration

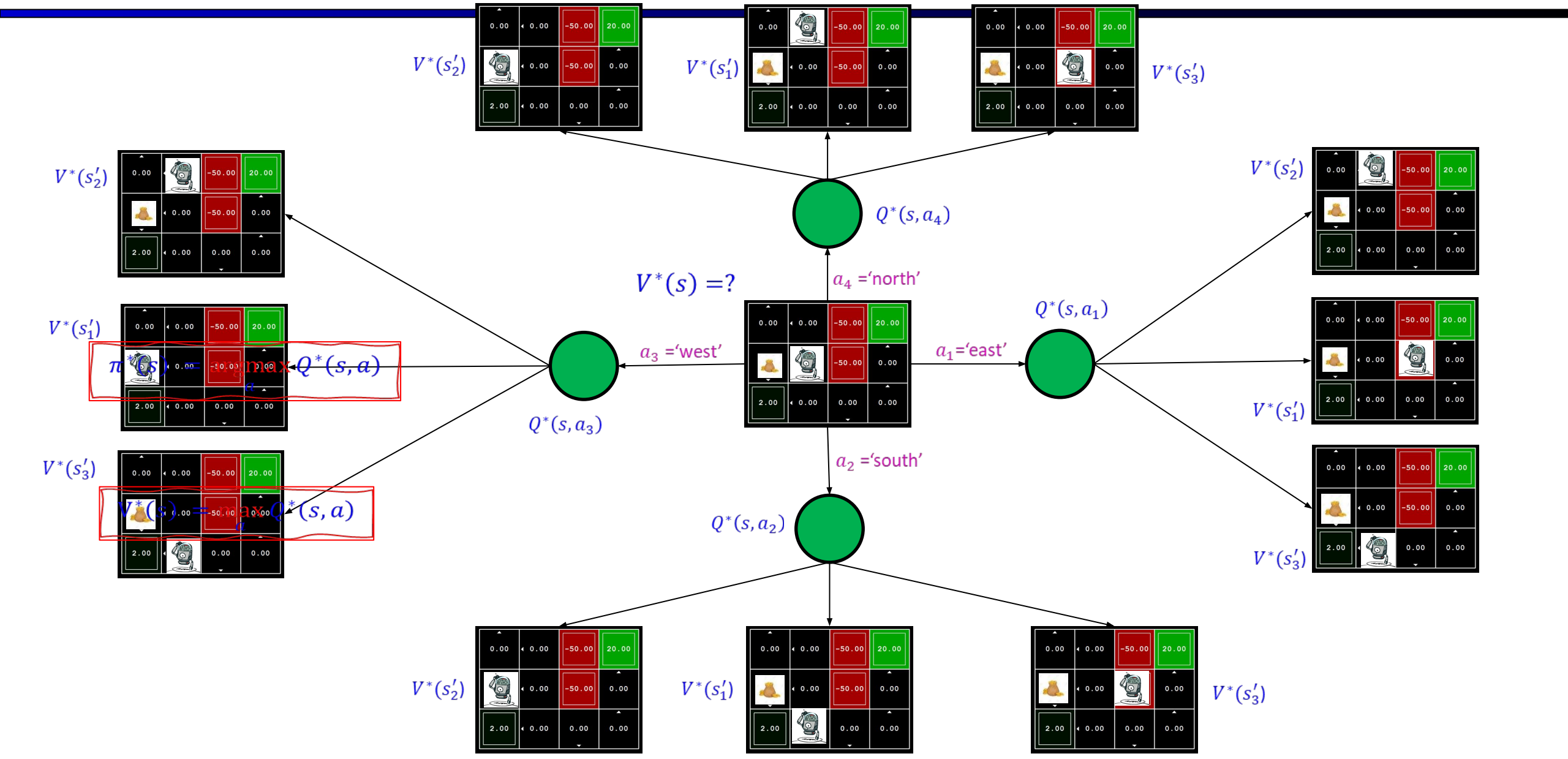
Regarding the current state,
how good is it according to an optimal policy?

state s = (2,2)



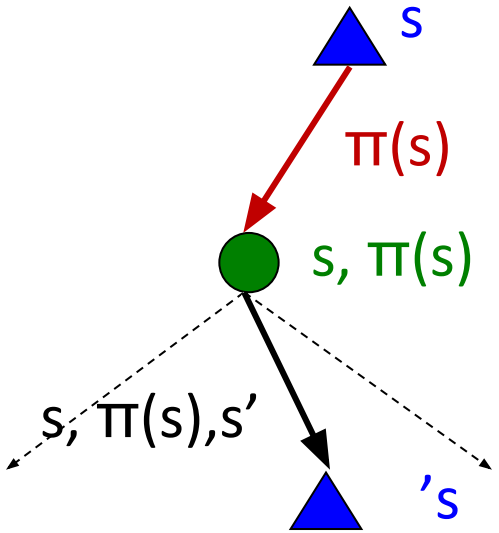
$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

Value Iteration

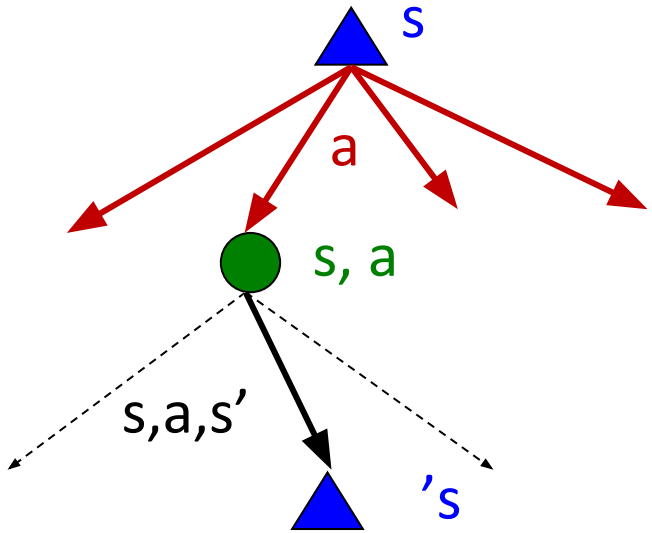


Optimal Policy

Do what π says to do



Do the optimal action



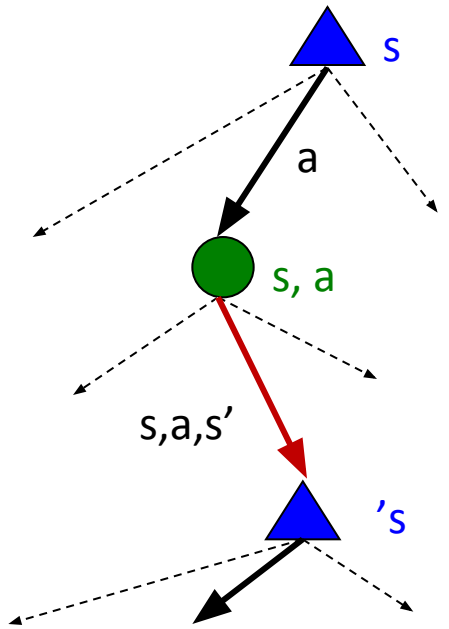
Expectimax trees max over all actions to compute the optimal values

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 $= \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$



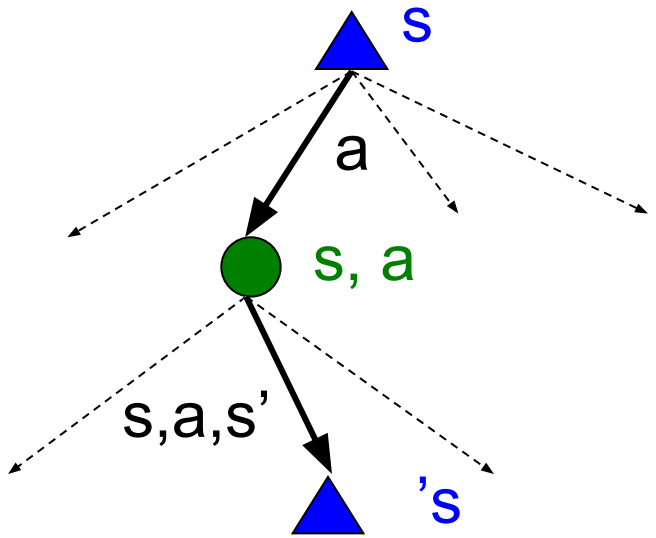
s is a *state*

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

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

Optimal Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!



- Recursive definition of optimal values:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

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

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$$

Bellman Equations

- Recursive definition of value:

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

- Bellman Equation:

*Necessary condition for optimality in optimization problems formulated as **Dynamic Programming***

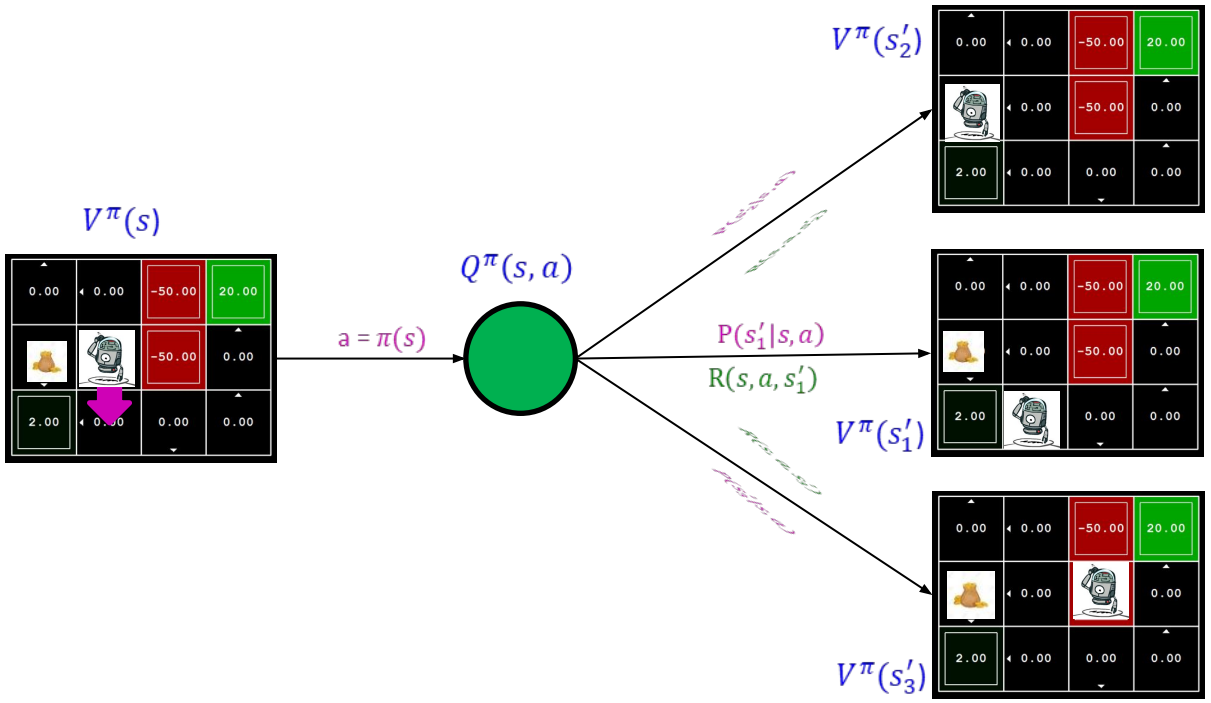
Richard E. Bellman
(1920–1984)

- Dynamic Programing:

Process to simplify an optimization problem by breaking it down into an optimal substructure.

Policy Evaluation

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s)) = \sum_{s'} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$



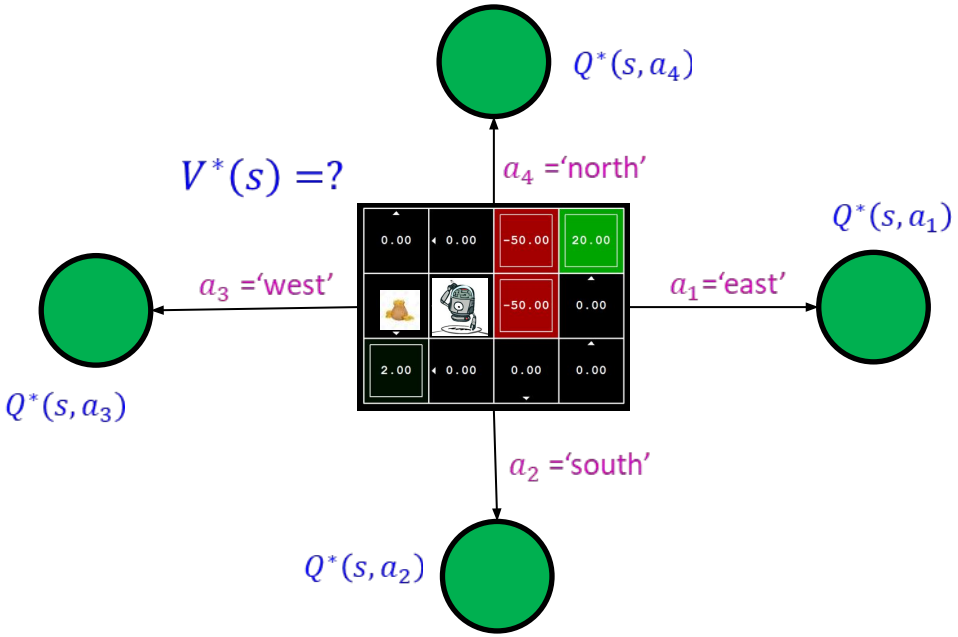
Iterative Approach

1. Initialize $V_0^{\pi}(s) \leftarrow 0$ for all states s.
2. Repeat until convergence: $t = 1, 2, 3, \ldots$
   // While $\max_s(|V_t^{\pi}(s) - V_{t-1}^{\pi}(s)|) > \varepsilon \approx 0.00001$
   - For each state s:

$$V_t^{\pi}(s) \leftarrow \underbrace{\sum_{s'} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma V_{t-1}^{\pi}(s')]}_{Q_{t-1}^{\pi}(s, \pi(s))}$$

Value Iteration

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$



Value Iteration

Evaluate an optimal policy

1. Initialize $V_0^*(s) \leftarrow 0$ for all states s.
2. Repeat until convergence: $t = 1, 2, 3, \ldots$
   // While $\max_s(|V_t^*(s) - V_{t-1}^*(s)|) > \varepsilon \approx 0.00001$
   - For each state s:

$$V_t^*(s) \leftarrow \max_a \underbrace{\sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_{t-1}^*(s')]}_{Q_{t-1}^*(s,a)}$$

Policy Evaluation

Evaluate a specific policy π

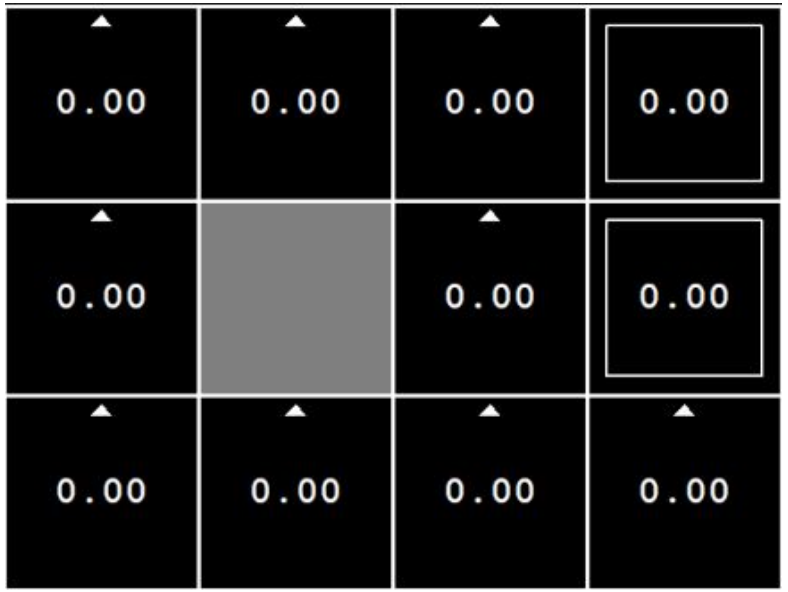
$$V_t^\pi(s) \leftarrow \underbrace{\sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_{t-1}^\pi(s')]}_{Q_{t-1}^\pi(s,\pi(s))}$$

t=0

$$V_0^*(s) \leftarrow 0$$

Noise = 0.2
Discount = 0.9
Living reward = 0

$V_t^*(s)$



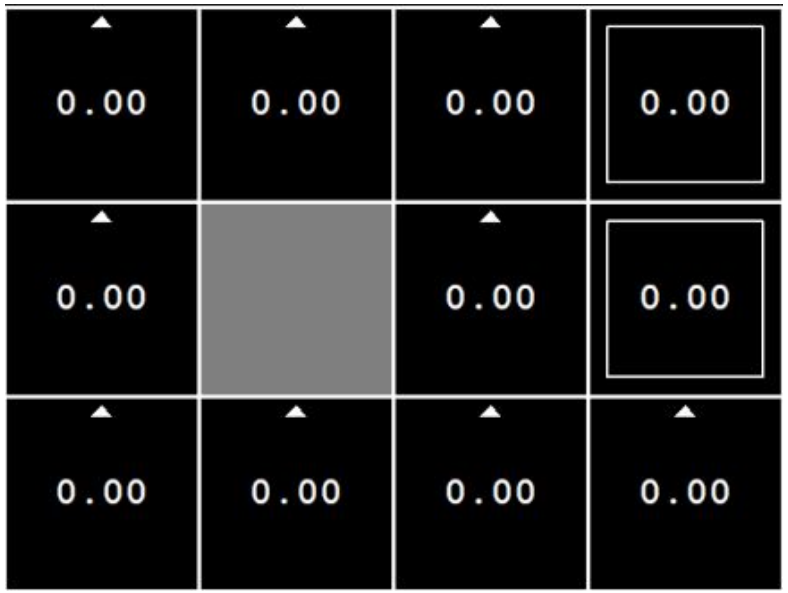
t=1

$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a)\,[\,R(s,a,s') + \gamma V_{t-1}^*(s')\,]$$

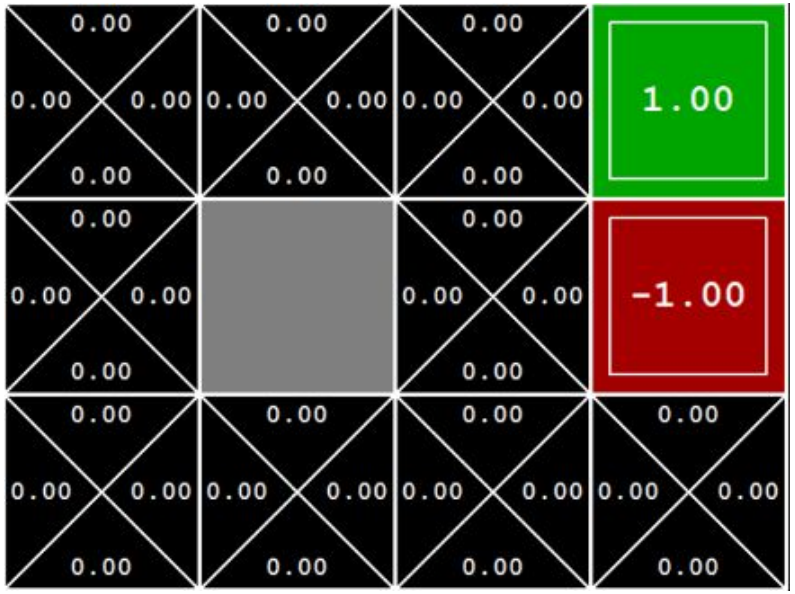
$$Q_{t-1}^*(s,a)$$

Noise = 0.2
Discount = 0.9
Living reward = 0

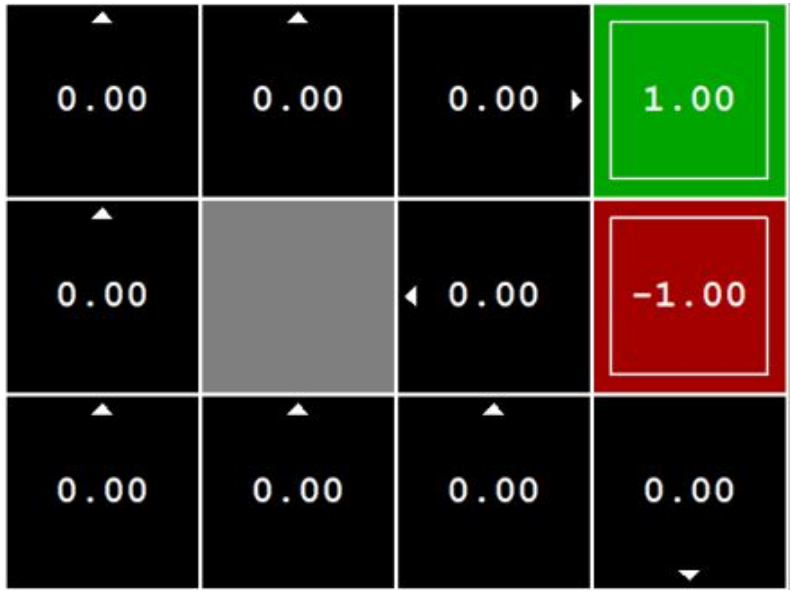
$V_{t-1}^*(s')$



$Q_{t-1}^*(s,a)$



$V_t^*(s')$



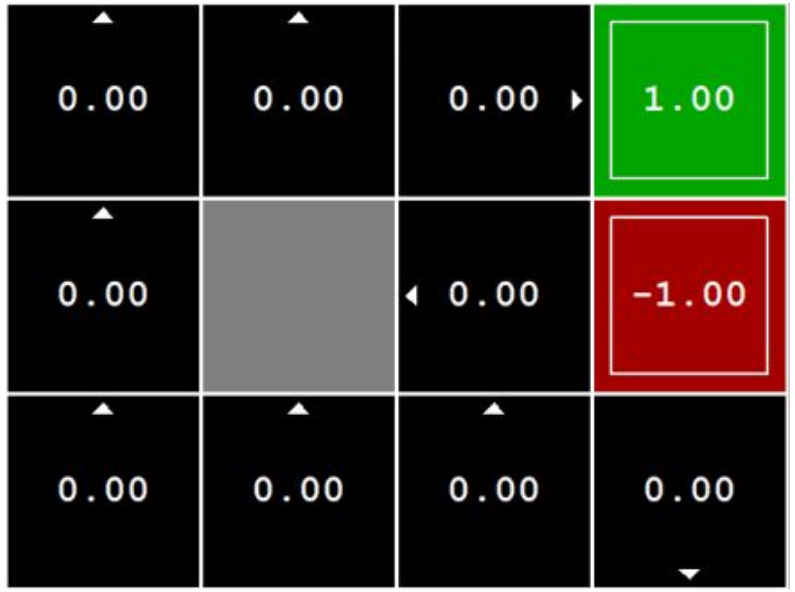
t=2

$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V_{t-1}^*(s') \right]$$

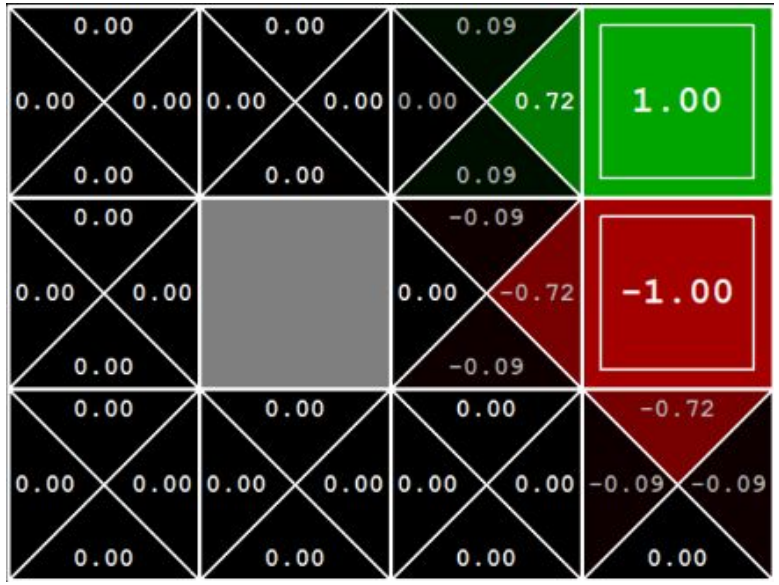
$$Q_{t-1}^*(s,a)$$

Noise = 0.2
Discount = 0.9
Living reward = 0

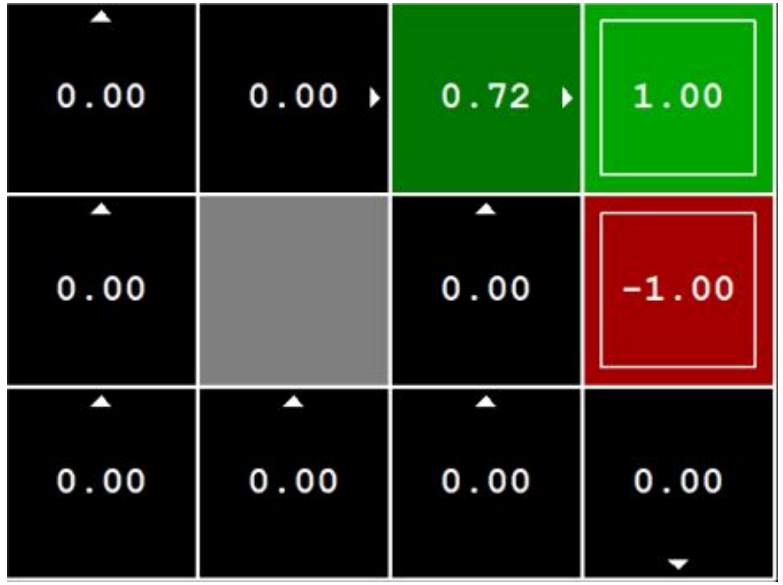
$V_{t-1}^*(s')$



$Q_{t-1}^*(s,a)$



$V_t^*(s')$



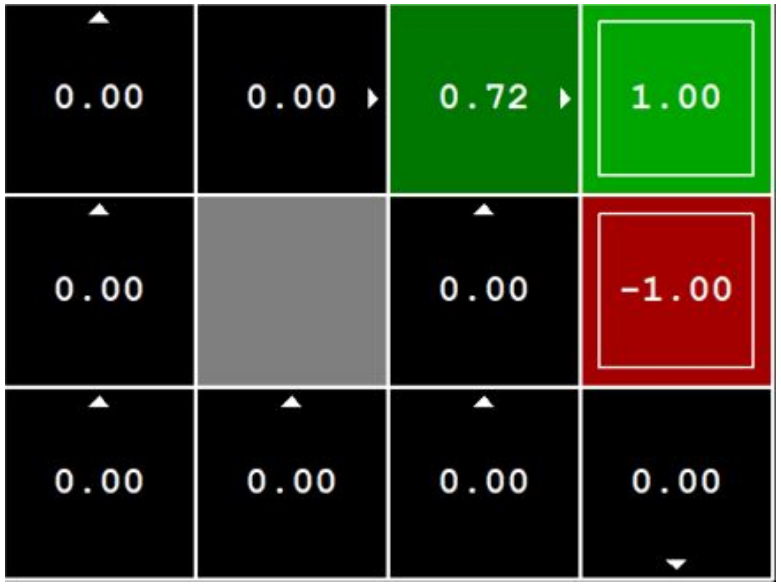
t=3

$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V_{t-1}^*(s') \right]$$

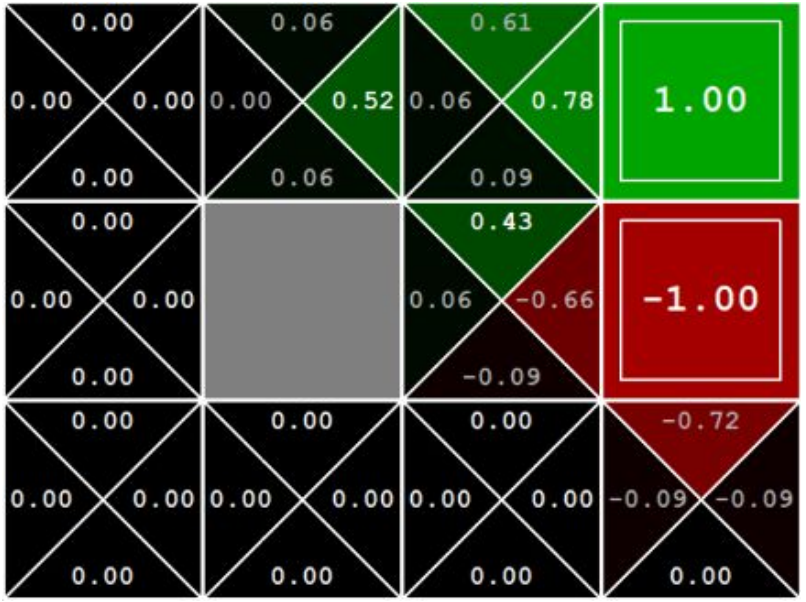
$$Q_{t-1}^*(s,a)$$

Noise = 0.2
Discount = 0.9
Living reward = 0

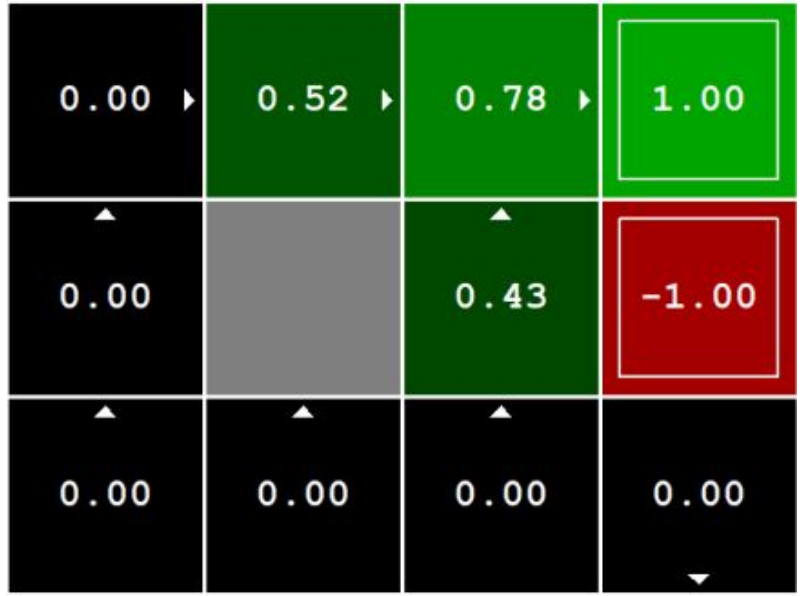
$V_{t-1}^*(s')$



$Q_{t-1}^*(s,a)$



$V_t^*(s')$



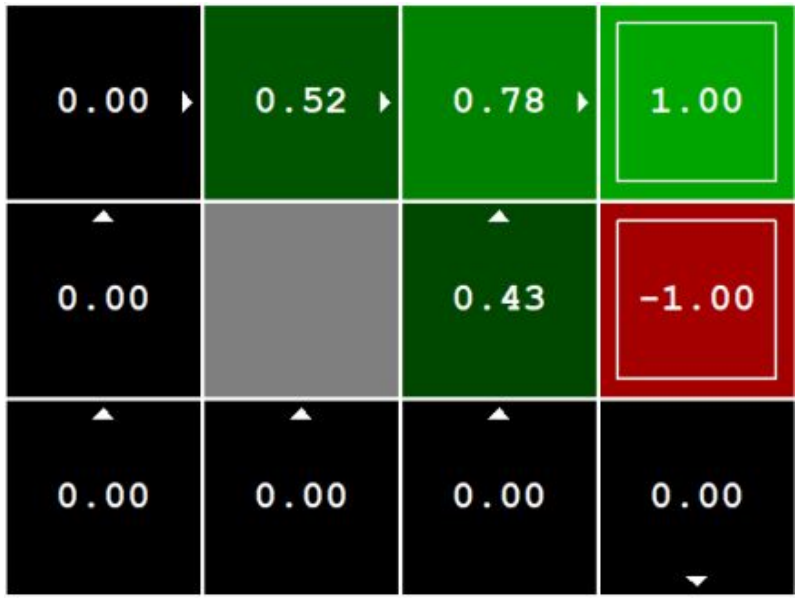
t=4

$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V_{t-1}^*(s') \right]$$

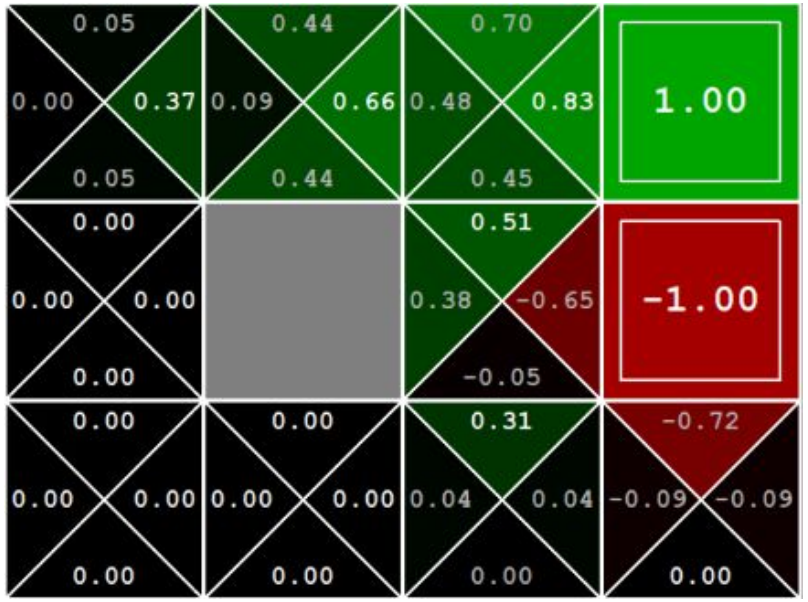
$$Q_{t-1}^*(s,a)$$

Noise = 0.2
Discount = 0.9
Living reward = 0

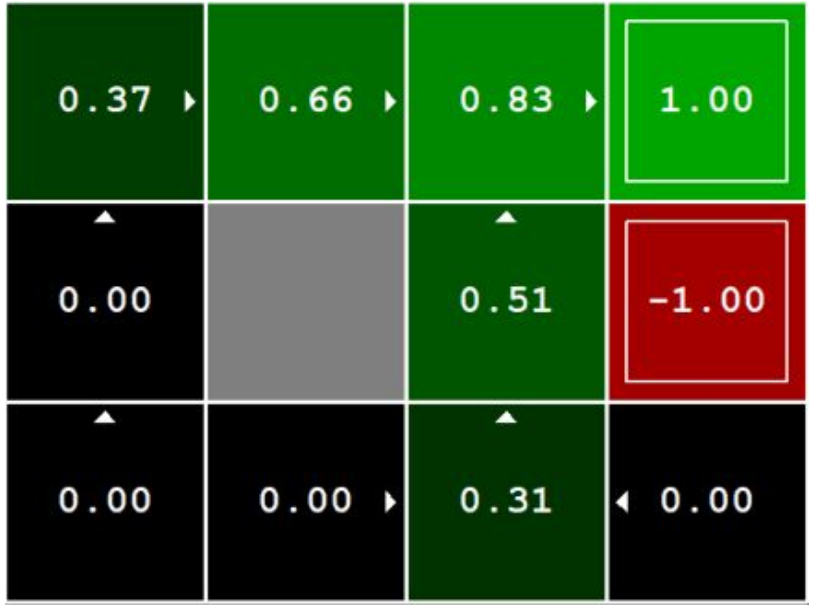
$V_{t-1}^*(s')$



$Q_{t-1}^*(s,a)$



$V_t^*(s')$



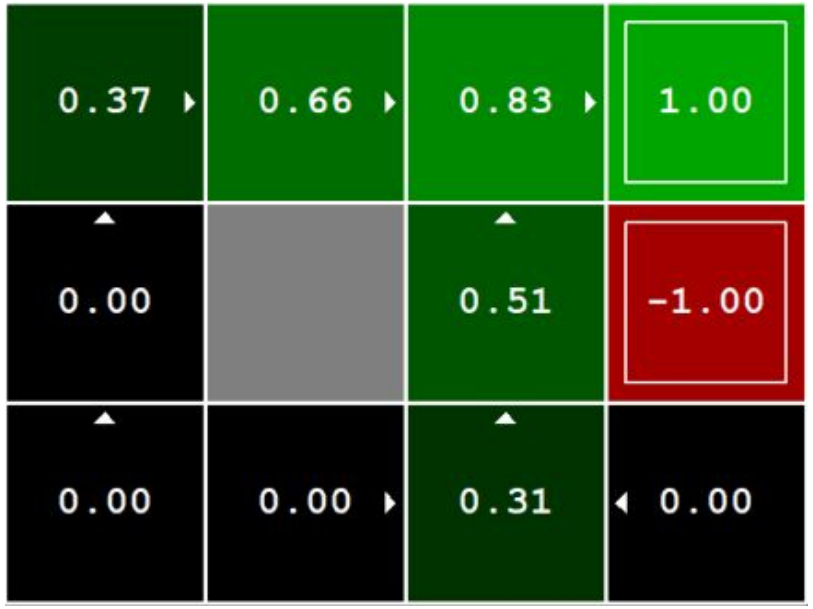
t=5

$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a)\,[\,R(s,a,s') + \gamma V_{t-1}^*(s')\,]$$

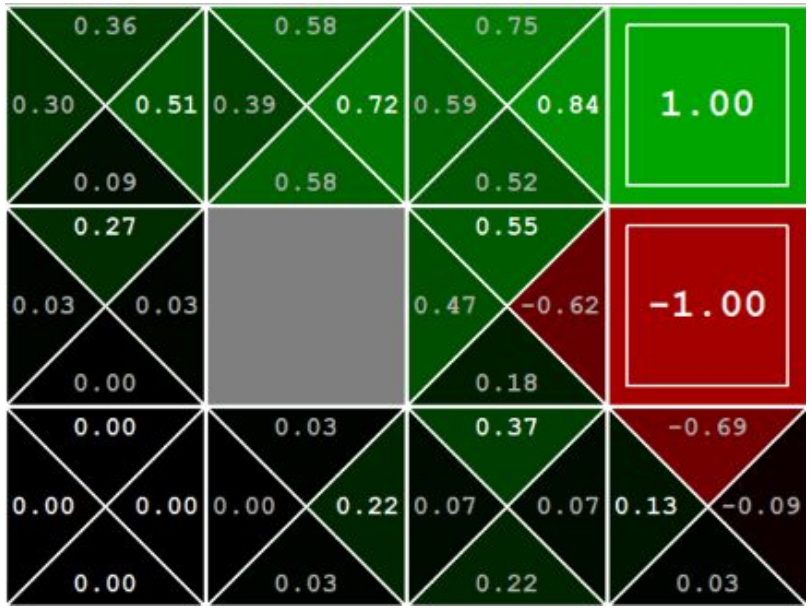
$$Q_{t-1}^*(s,a)$$

Noise = 0.2
Discount = 0.9
Living reward = 0

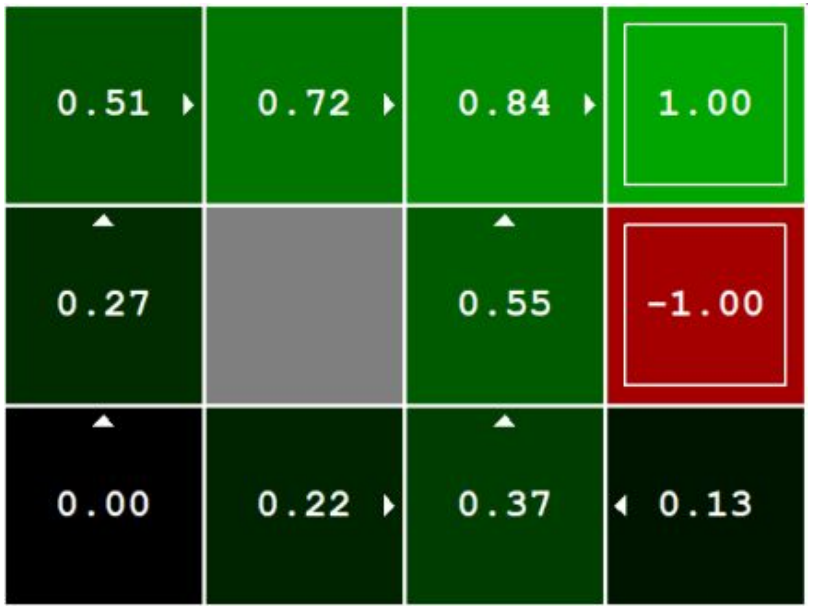
$V_{t-1}^*(s')$



$Q_{t-1}^*(s,a)$



$V_t^*(s')$



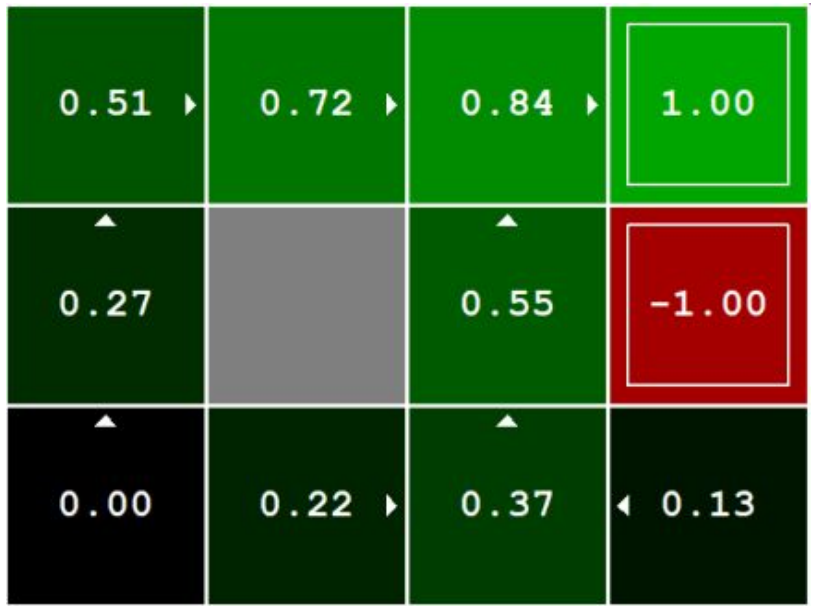
t=6

$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a)\,[\,R(s,a,s') + \gamma V_{t-1}^*(s')\,]$$

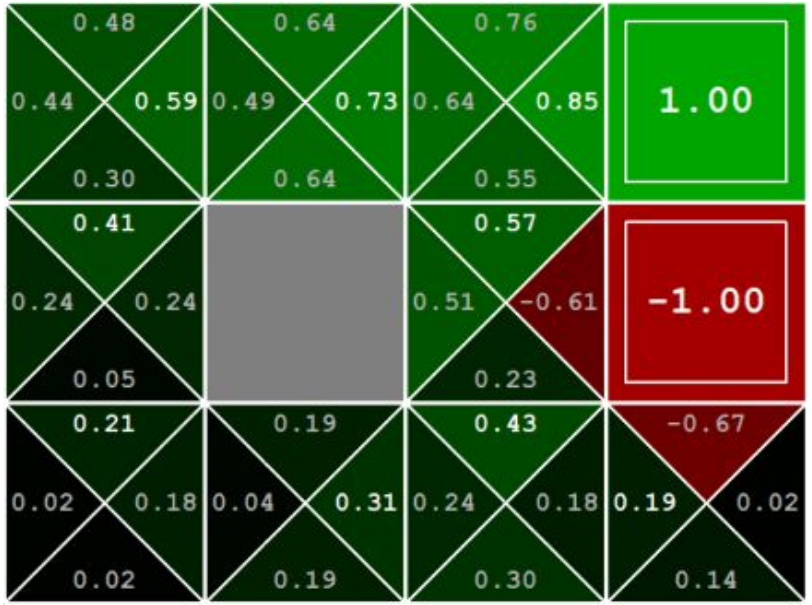
$$Q_{t-1}^*(s,a)$$

Noise = 0.2
Discount = 0.9
Living reward = 0

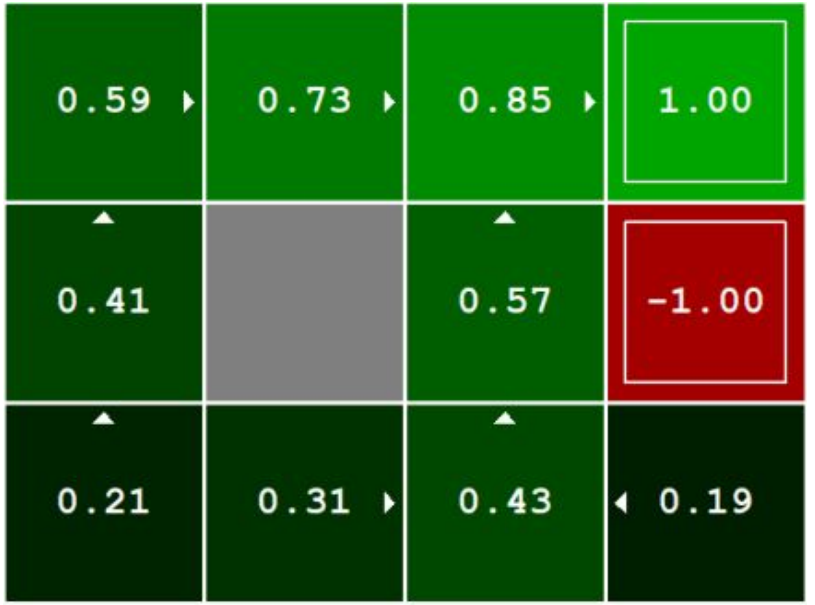
$V_{t-1}^*(s')$



$Q_{t-1}^*(s,a)$



$V_t^*(s')$



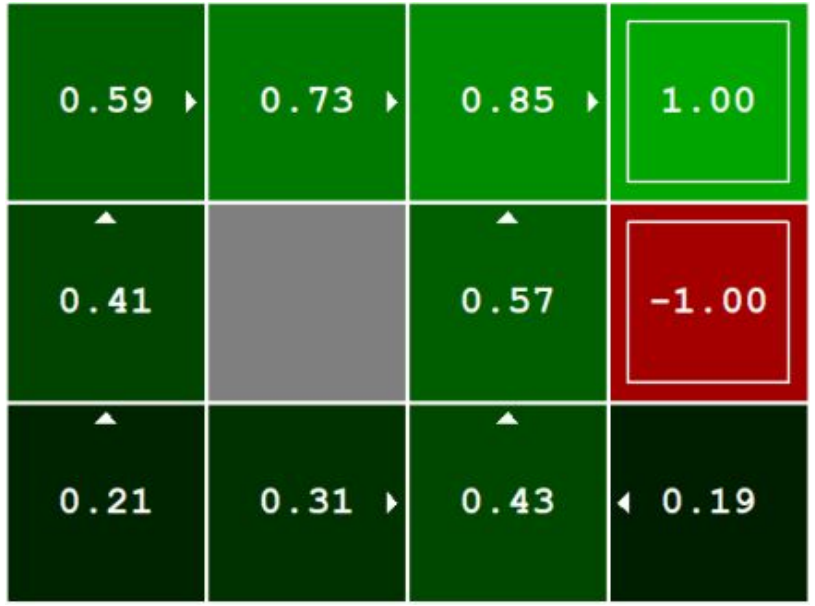
t=7

$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a)\,[\,R(s,a,s') + \gamma V_{t-1}^*(s')\,]$$

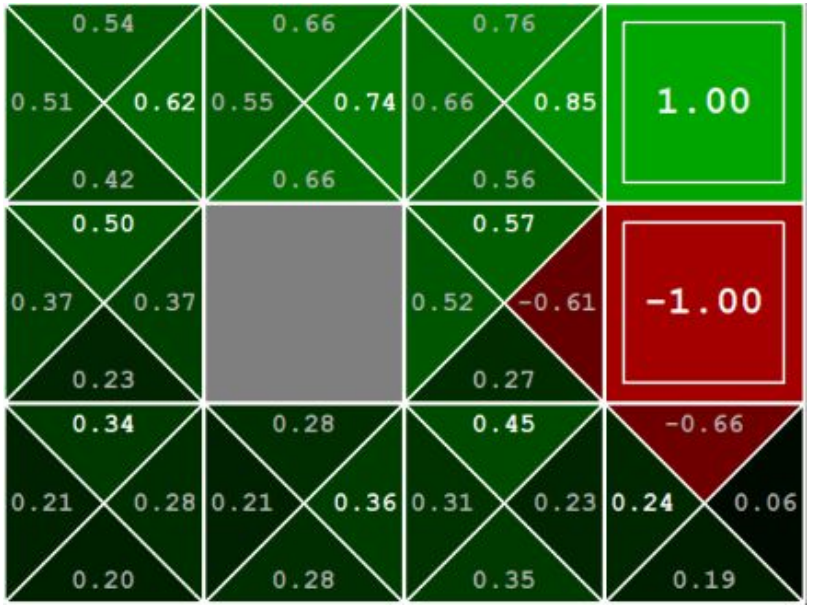
$$Q_{t-1}^*(s,a)$$

Noise = 0.2
Discount = 0.9
Living reward = 0

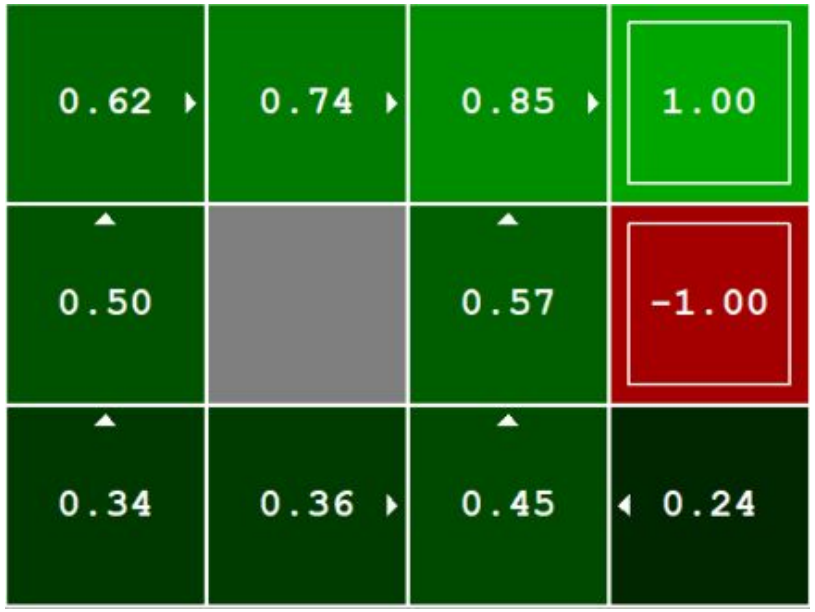
$V_{t-1}^*(s')$



$Q_{t-1}^*(s,a)$



$V_t^*(s')$



t=100

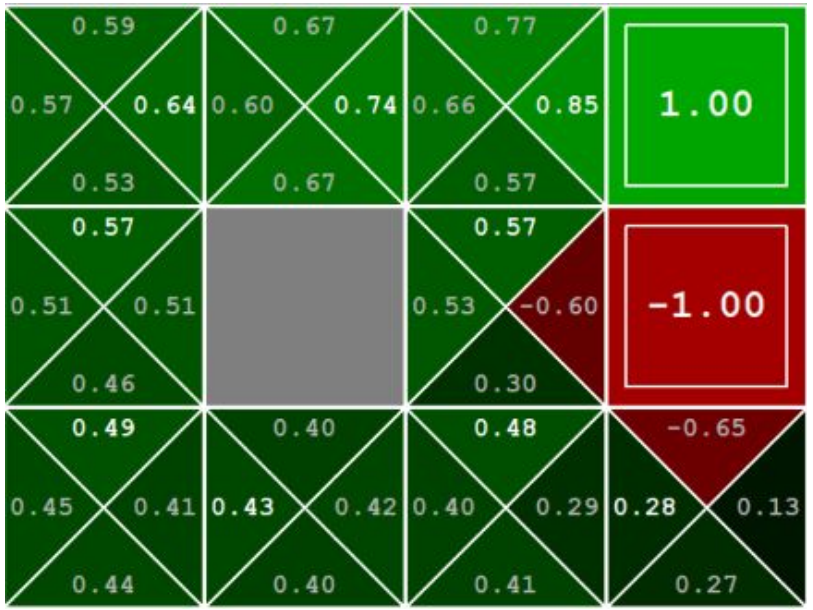
$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a)\,[\,R(s,a,s') + \gamma V_{t-1}^*(s')\,]$$

$$Q_{t-1}^*(s,a)$$

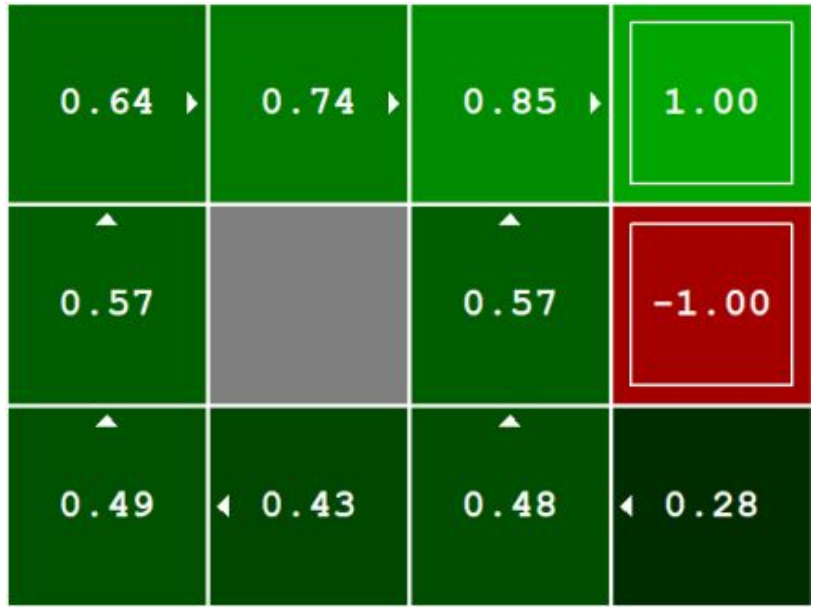
Noise = 0.2
Discount = 0.9
Living reward = 0

$V_{t-1}^*(s')$

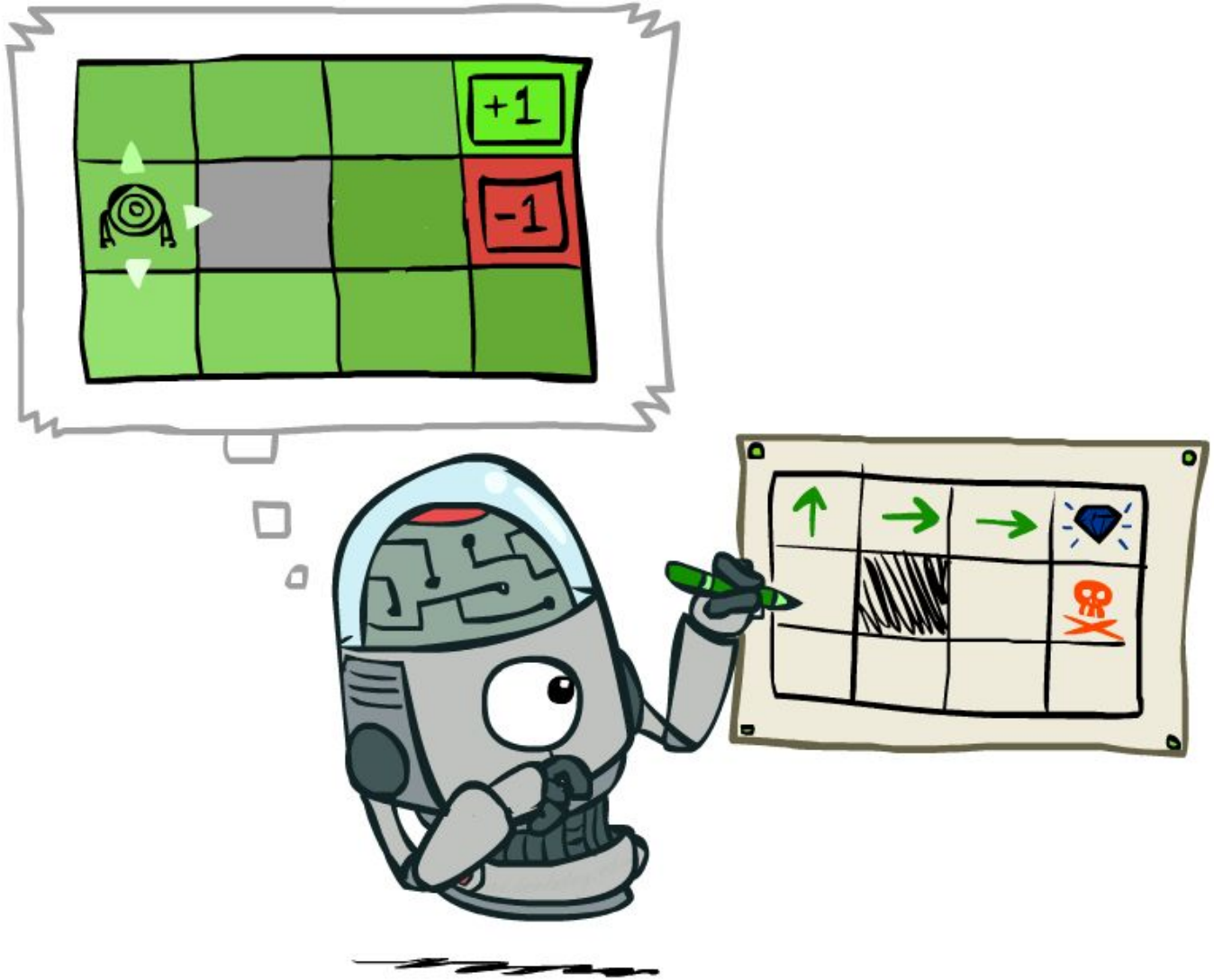
$Q_{t-1}^*(s,a)$



$V_t^*(s')$

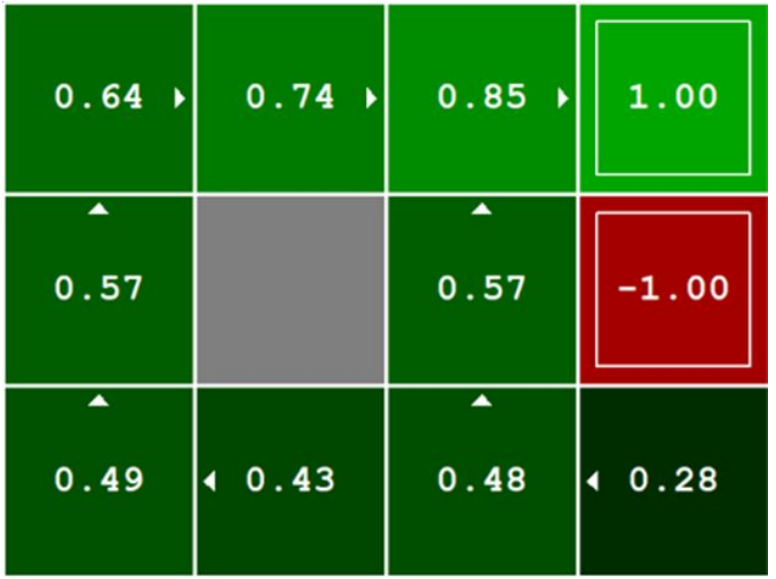


Policy Extraction



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)

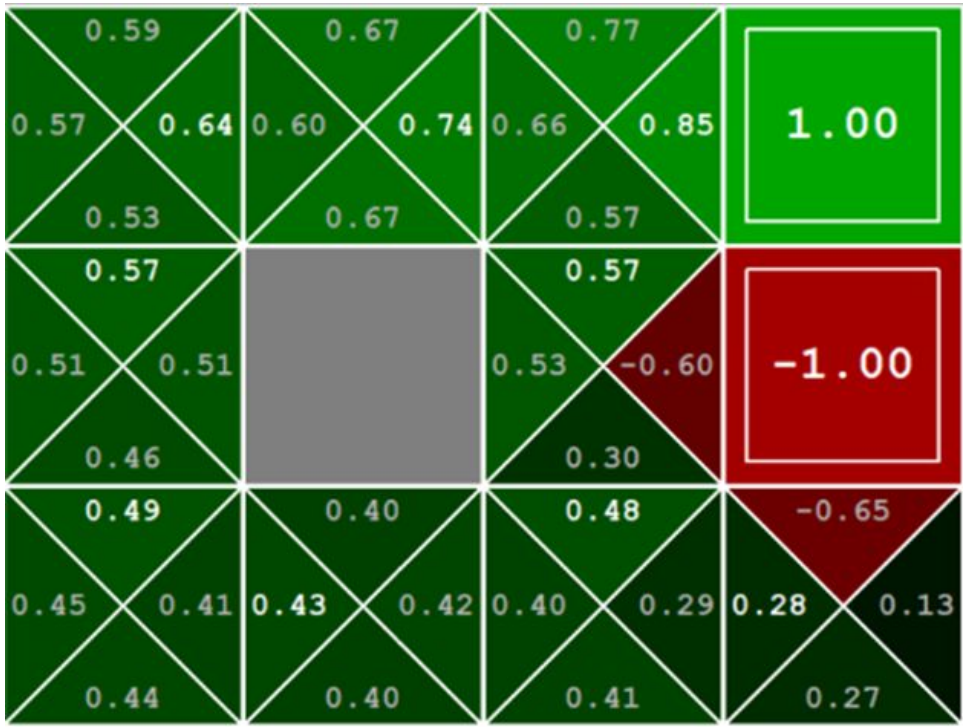


$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

- 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 s
  - Com

$\pi^*(s)$

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

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

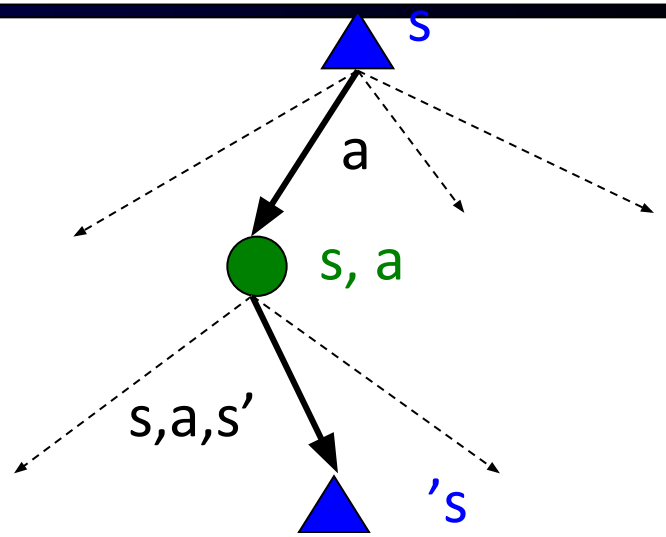
The Bellman Equations

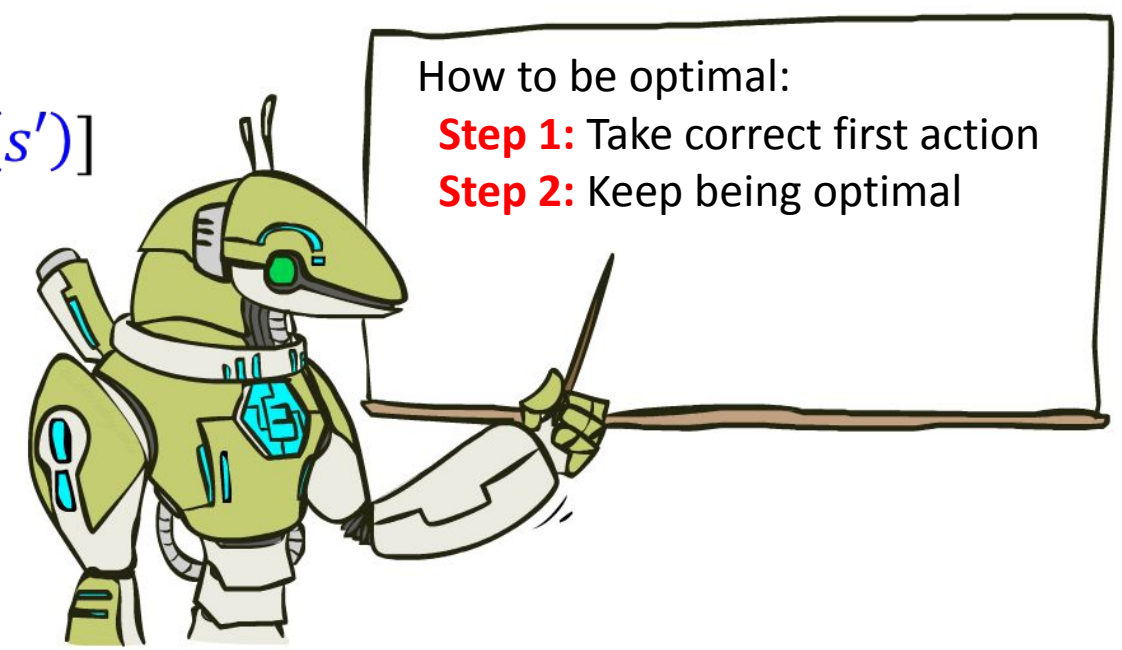
- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

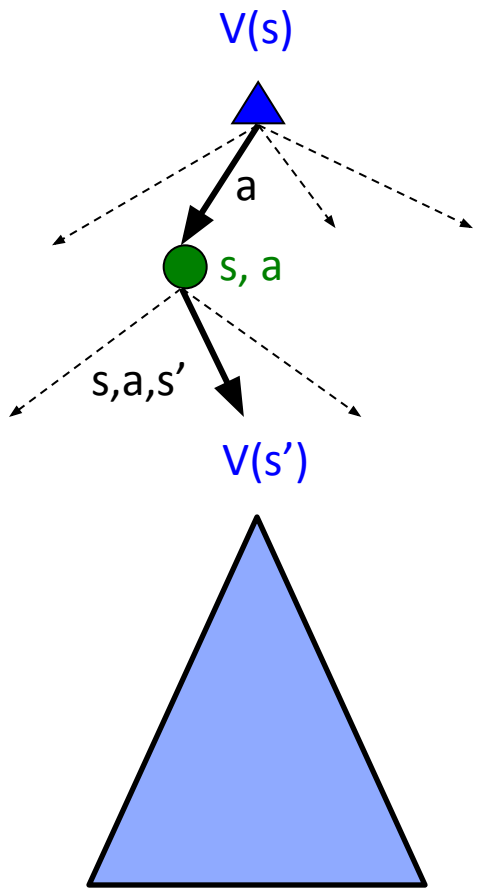




Value Iteration

- Bellman equations characterize the optimal values:

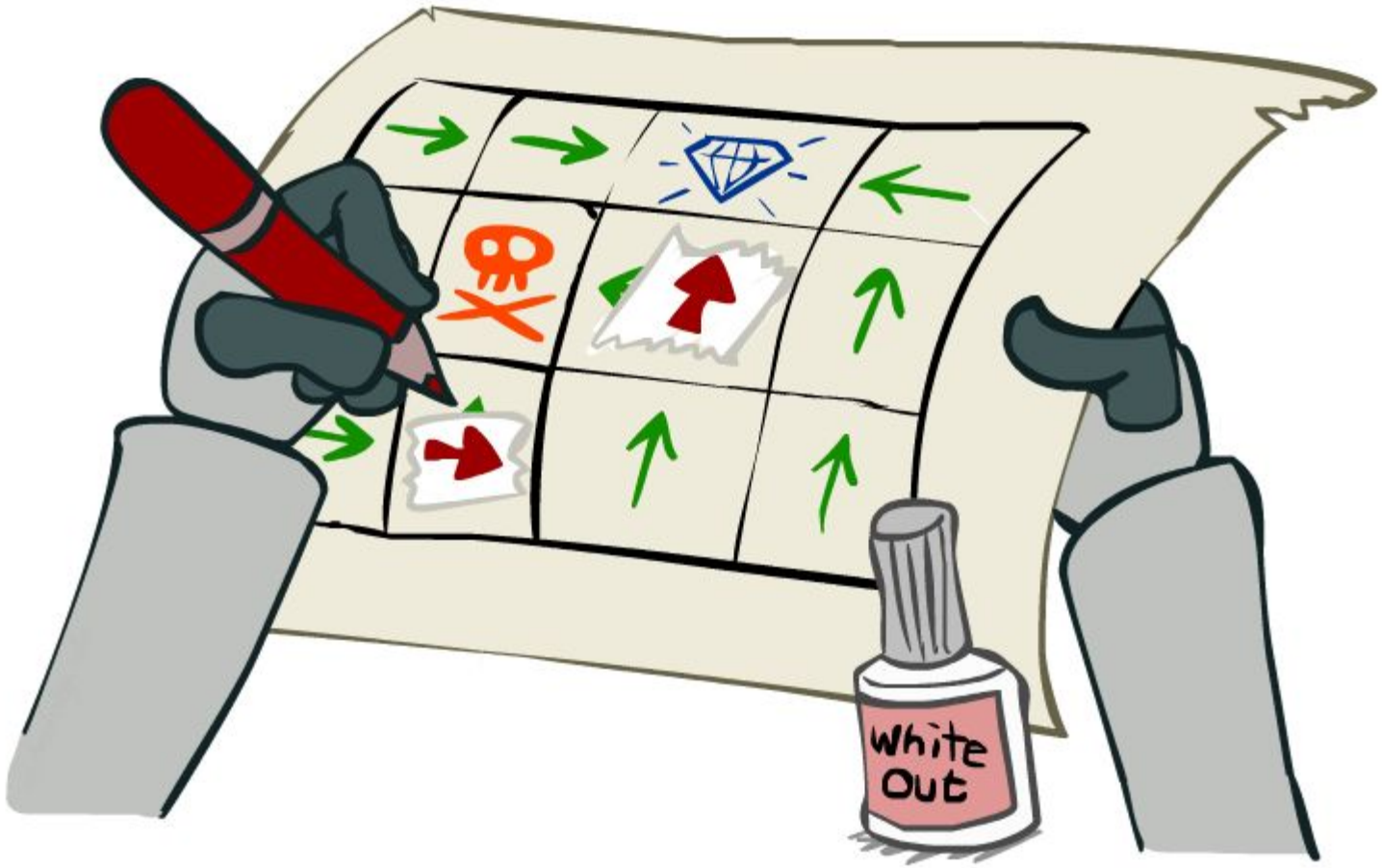
$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$



- Value iteration computes them:

$$V_t^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_{t-1}^*(s')]$$

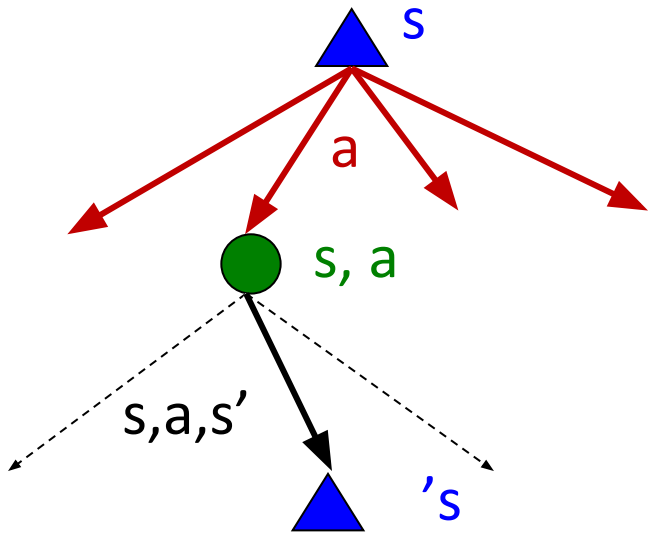
Policy Iteration



Problems with Value Iteration

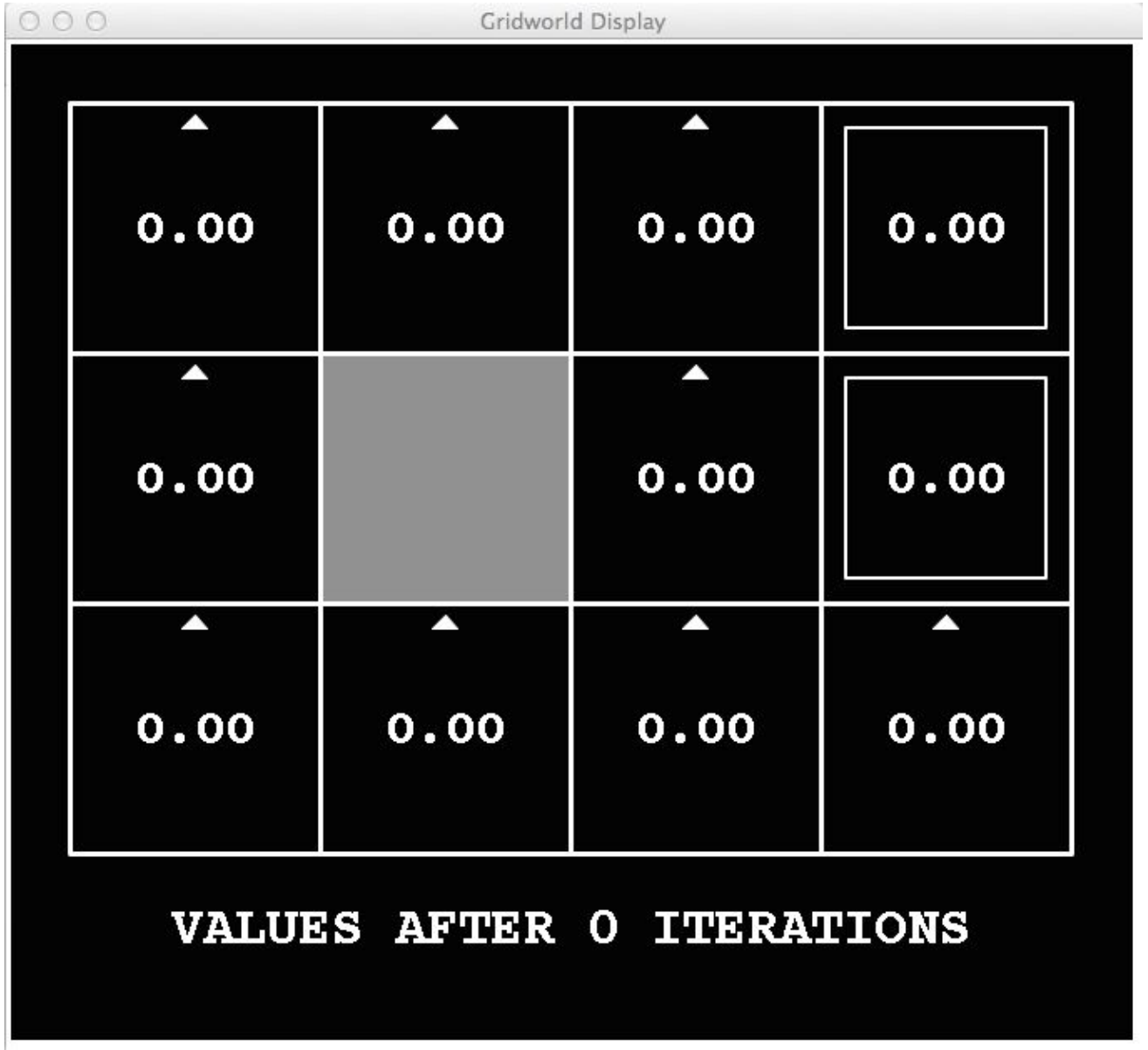
- Value iteration repeats the Bellman updates:

$$V_t^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_{t-1}^*(s')]$$



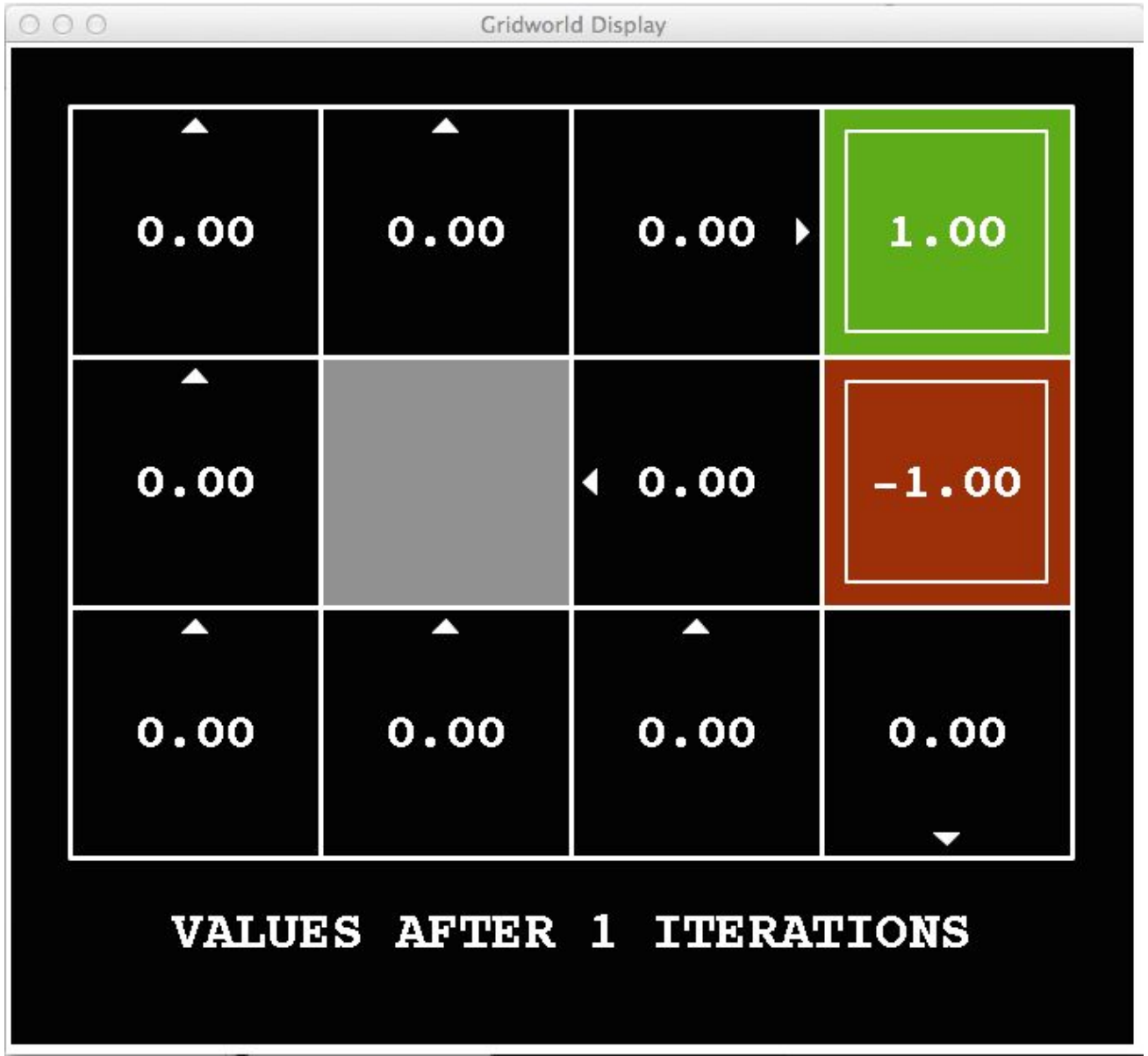
- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

t=0



Noise = 0.2
Discount = 0.9
Living reward = 0

t=1



Noise = 0.2
Discount = 0.9
Living reward = 0

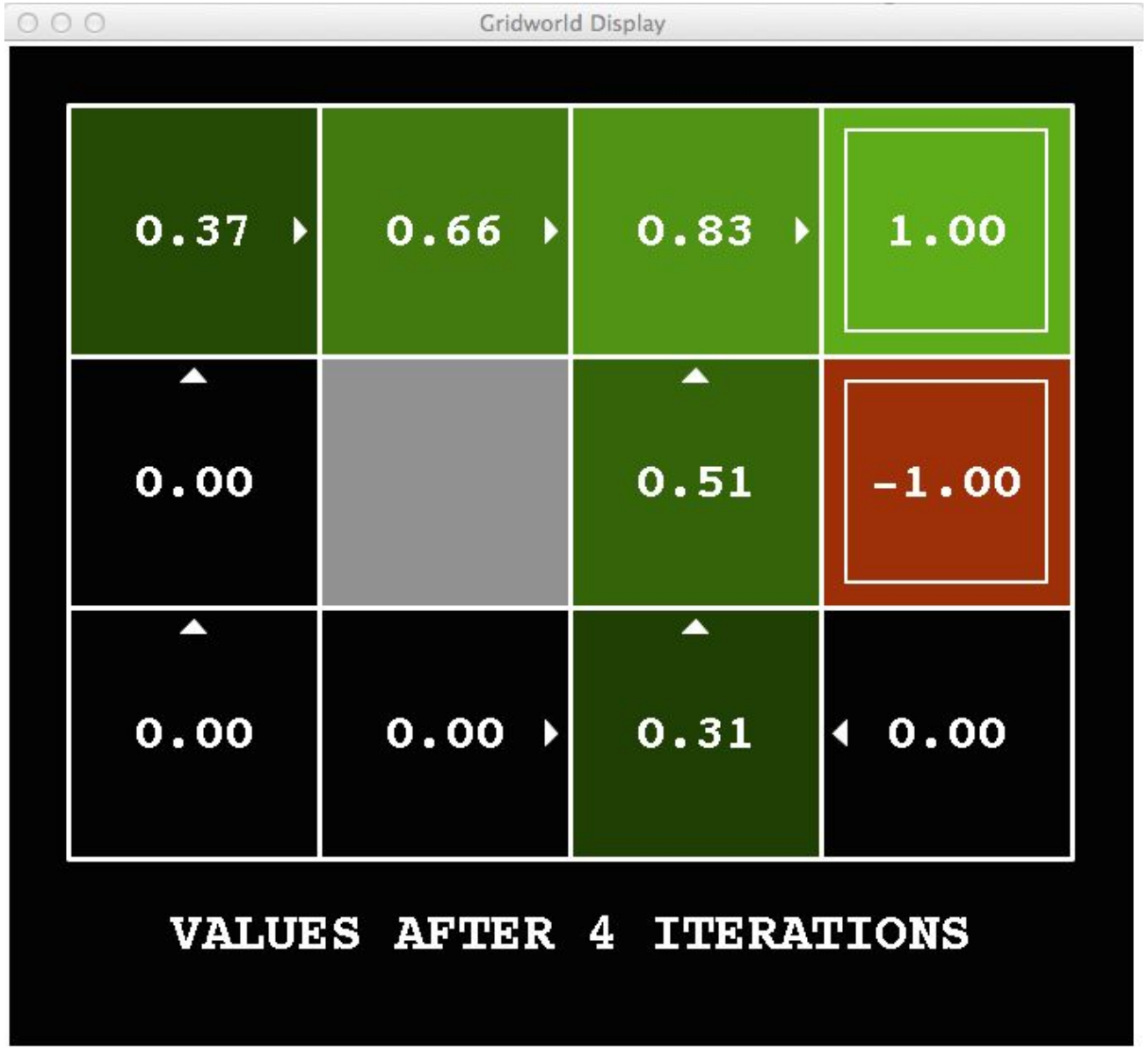
t=2

Noise = 0.2
Discount = 0.9
Living reward = 0

t=3

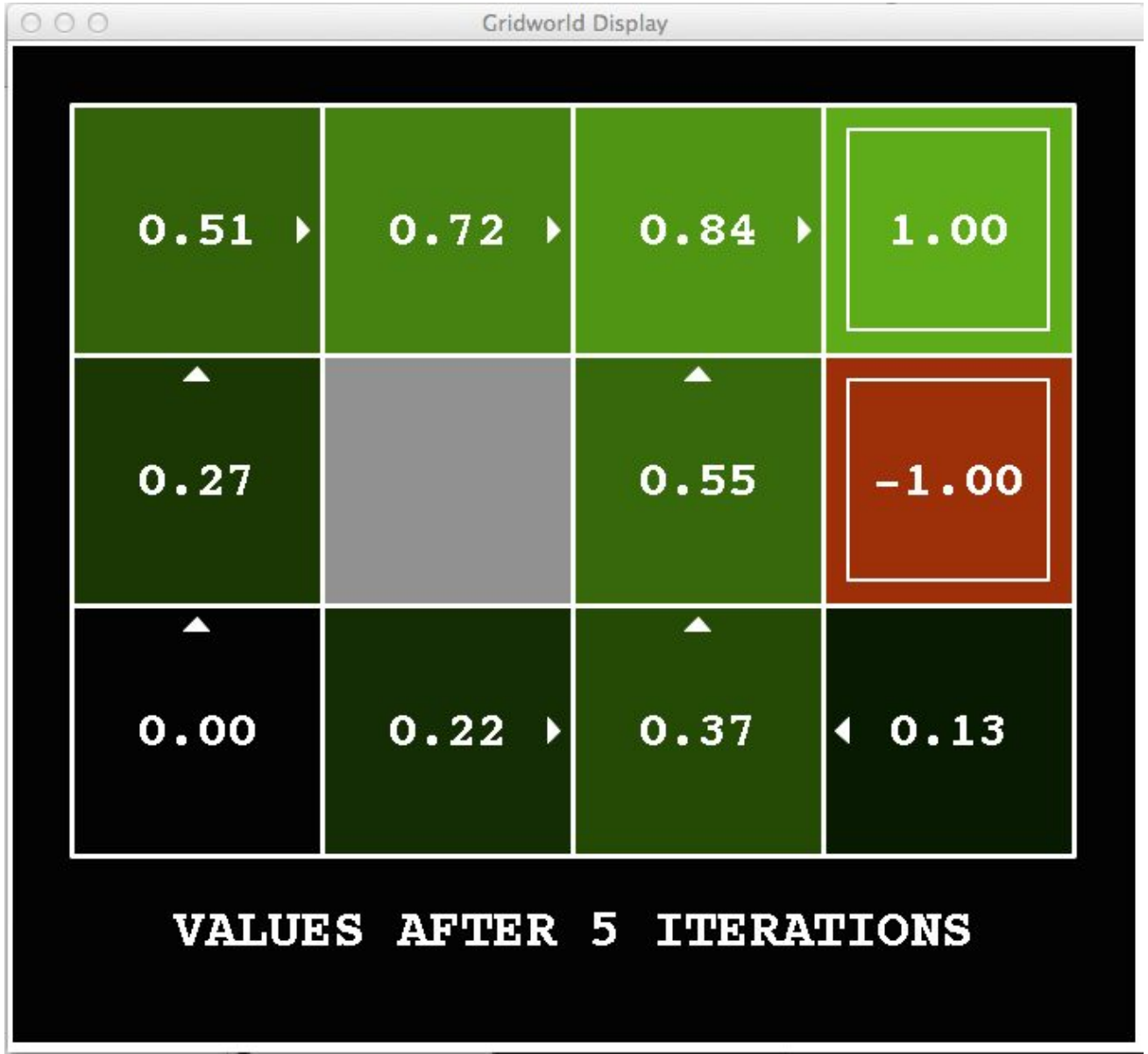
Noise = 0.2
Discount = 0.9
Living reward = 0

t=4



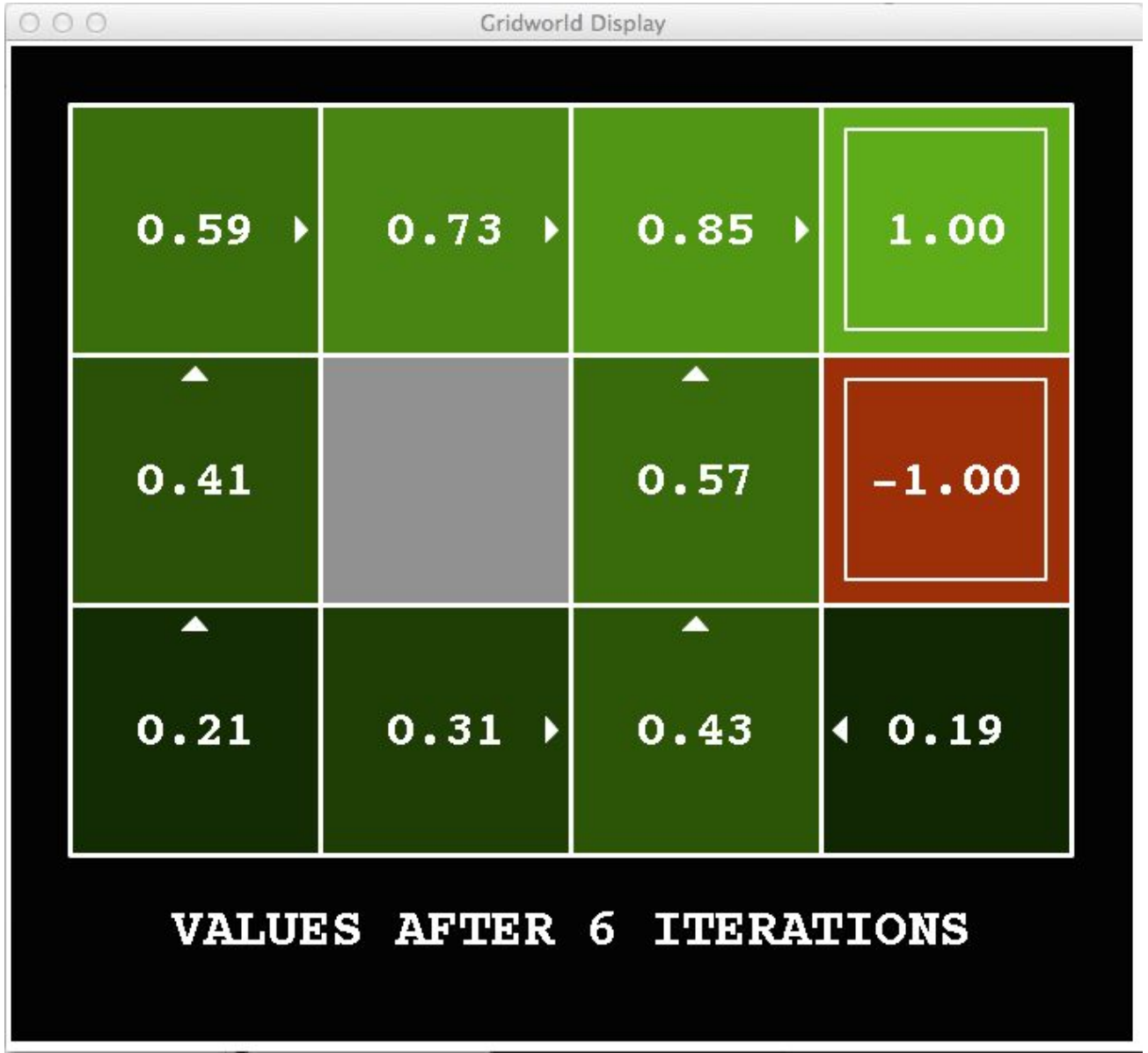
Noise = 0.2
Discount = 0.9
Living reward = 0

t=5



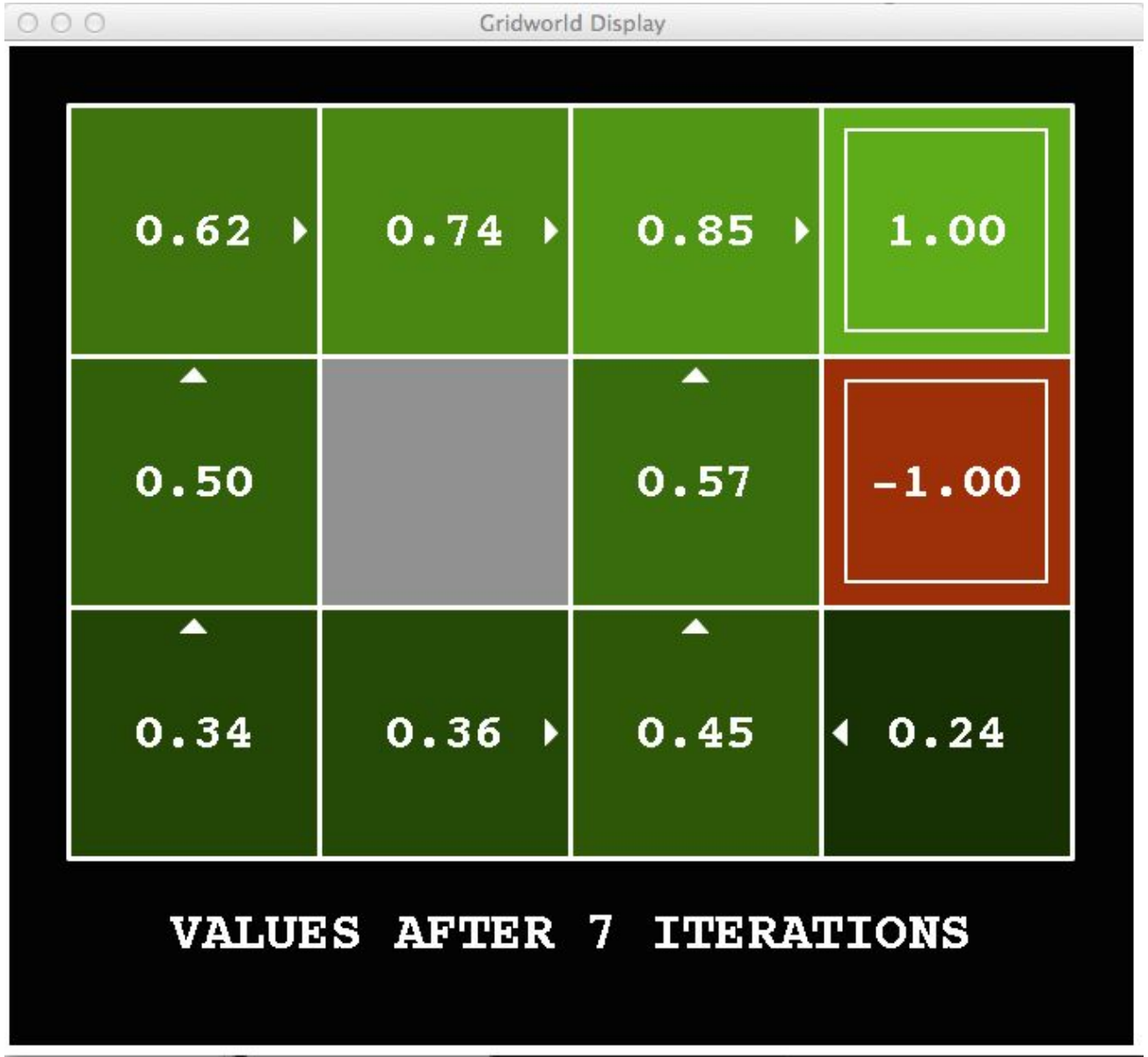
Noise = 0.2
Discount = 0.9
Living reward = 0

t=6



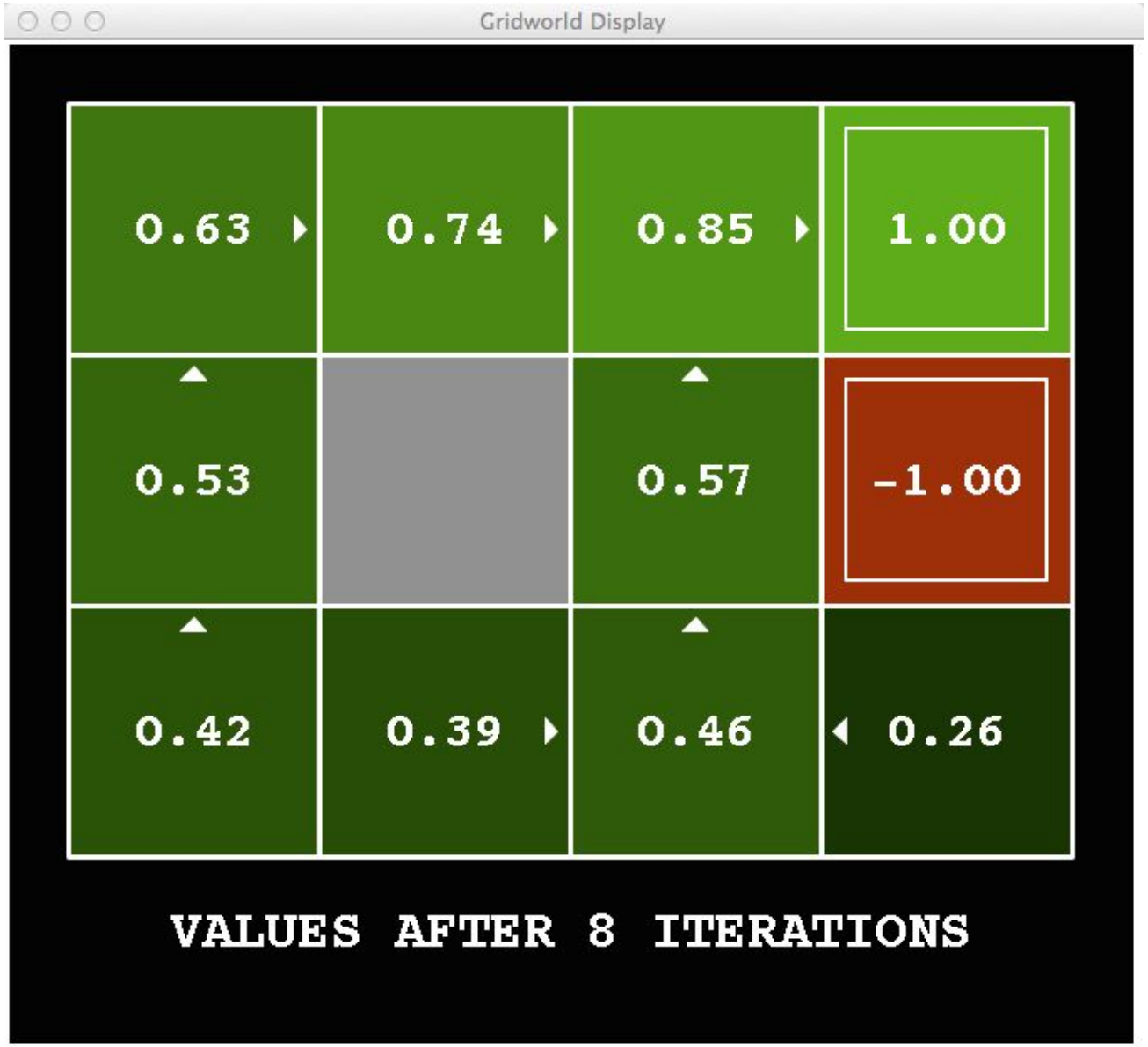
Noise = 0.2
Discount = 0.9
Living reward = 0

t=7



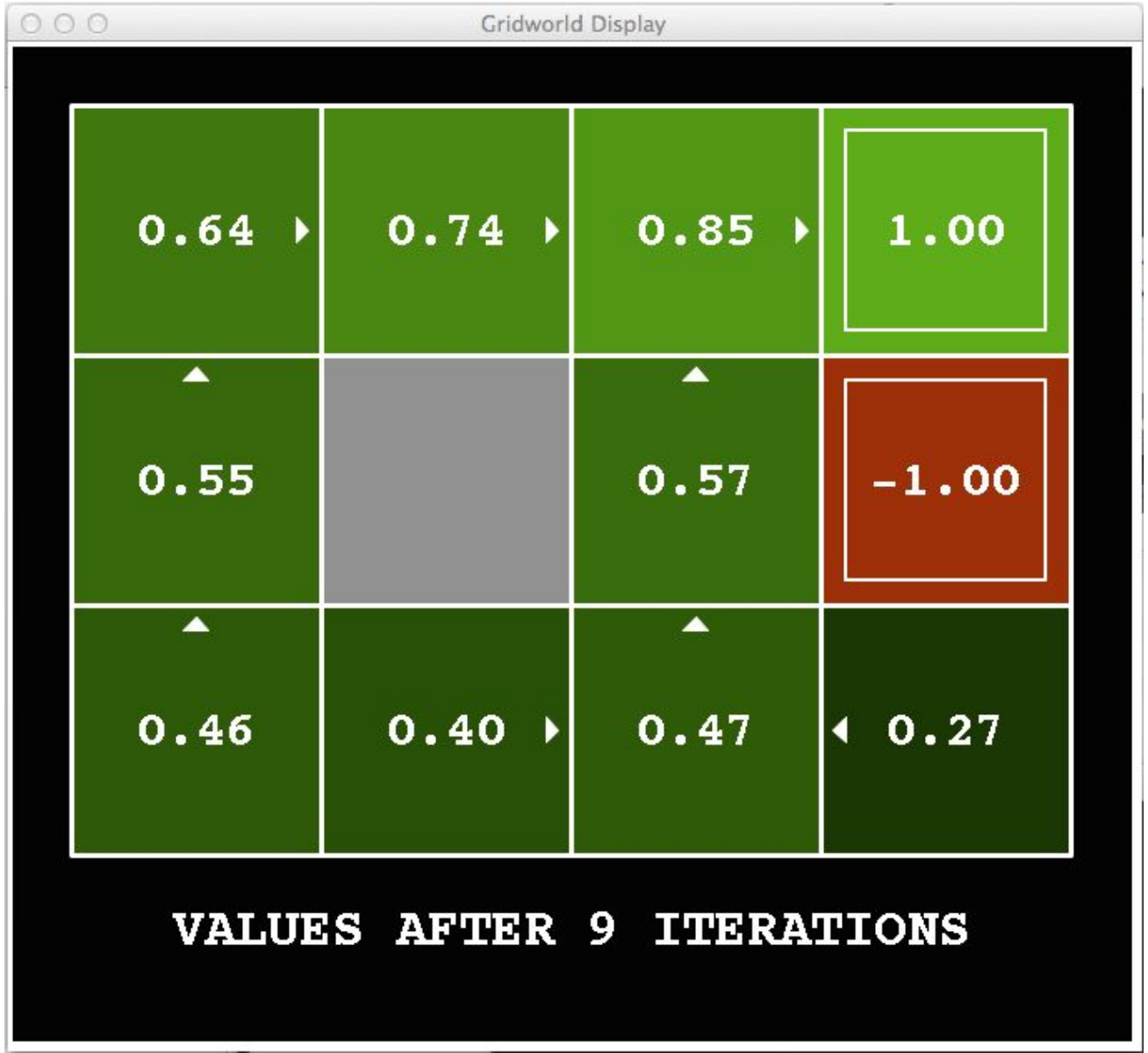
Noise = 0.2
Discount = 0.9
Living reward = 0

t=8



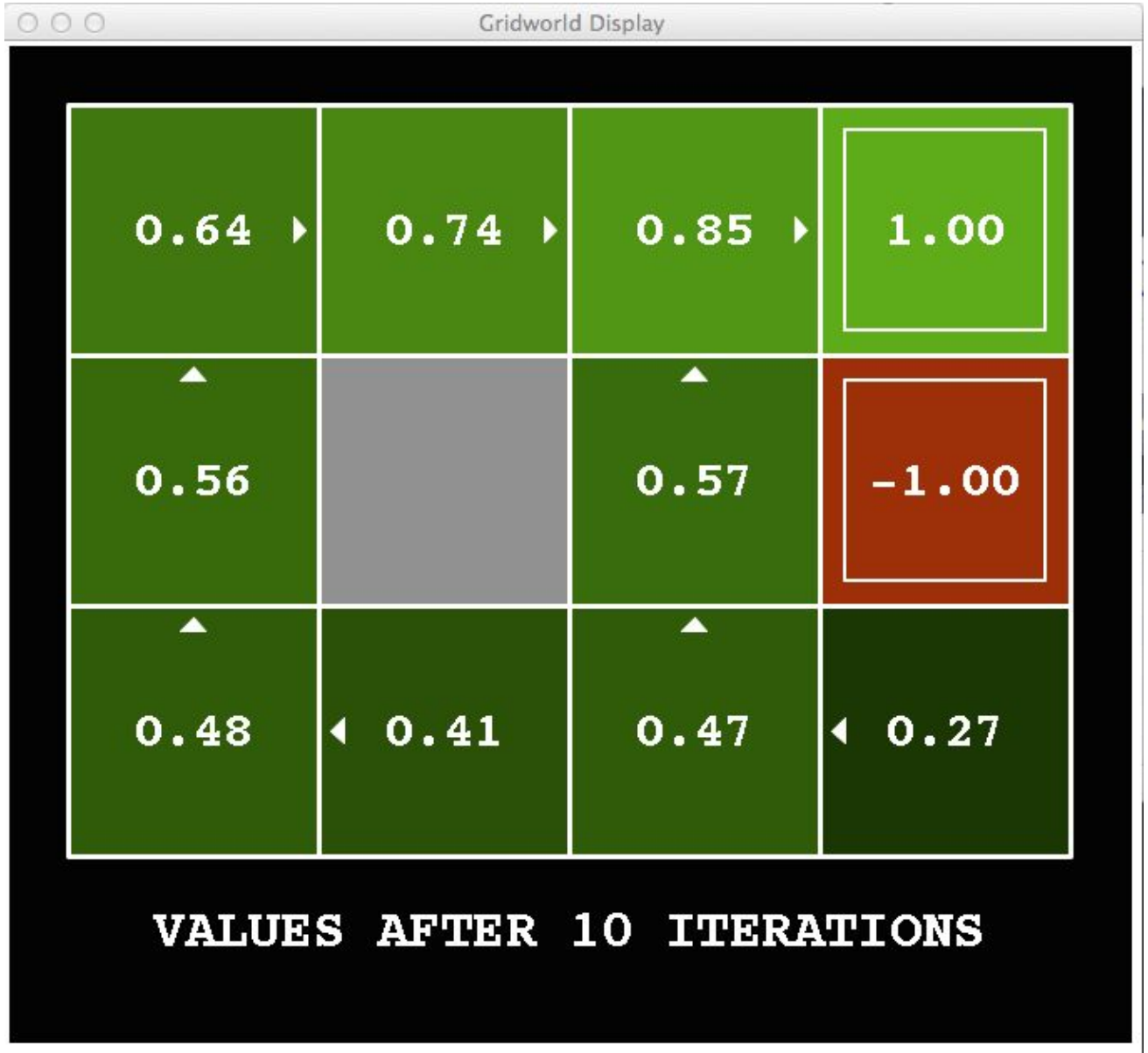
Noise = 0.2
Discount = 0.9
Living reward = 0

t=9



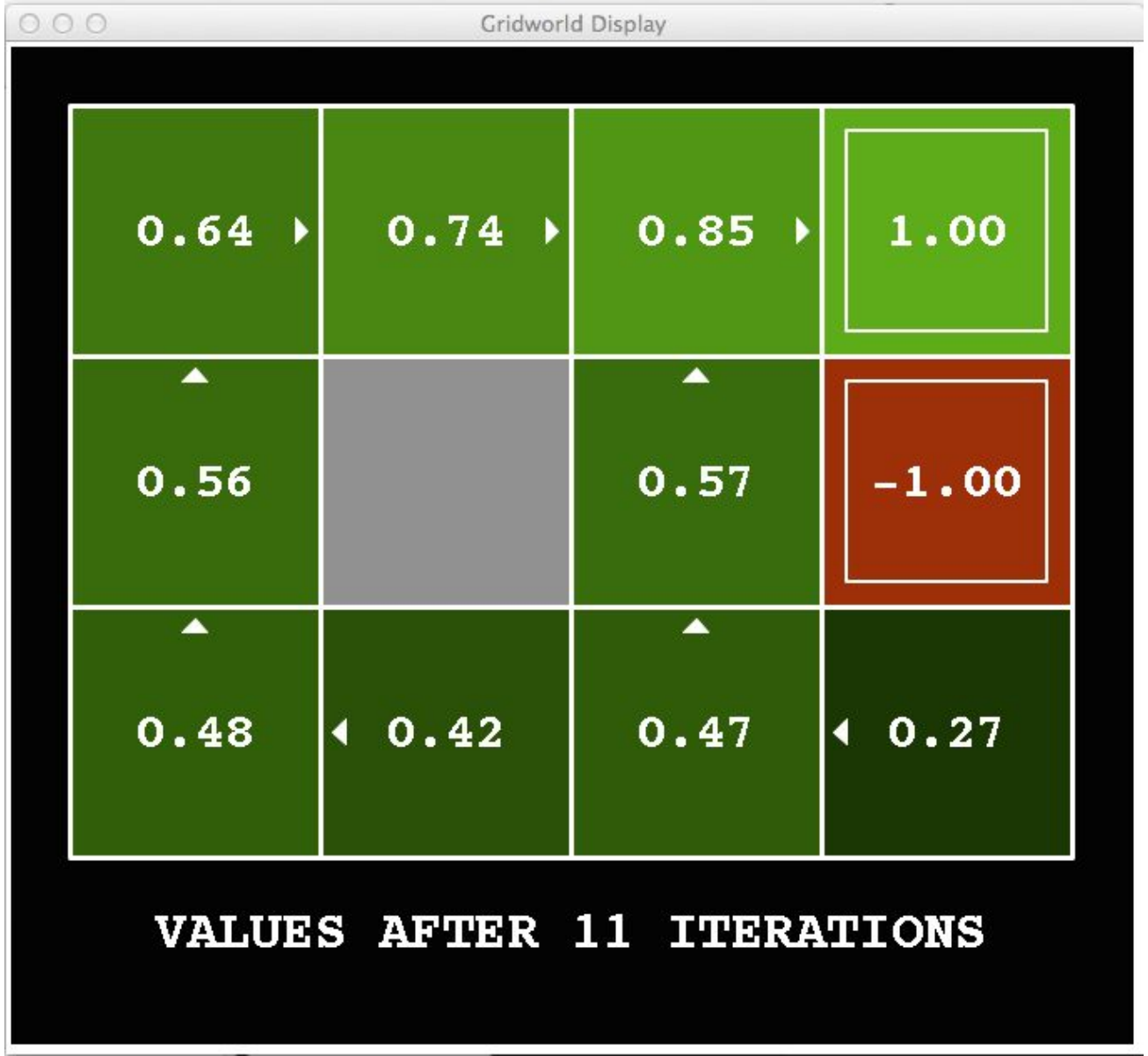
Noise = 0.2
Discount = 0.9
Living reward = 0

t=10



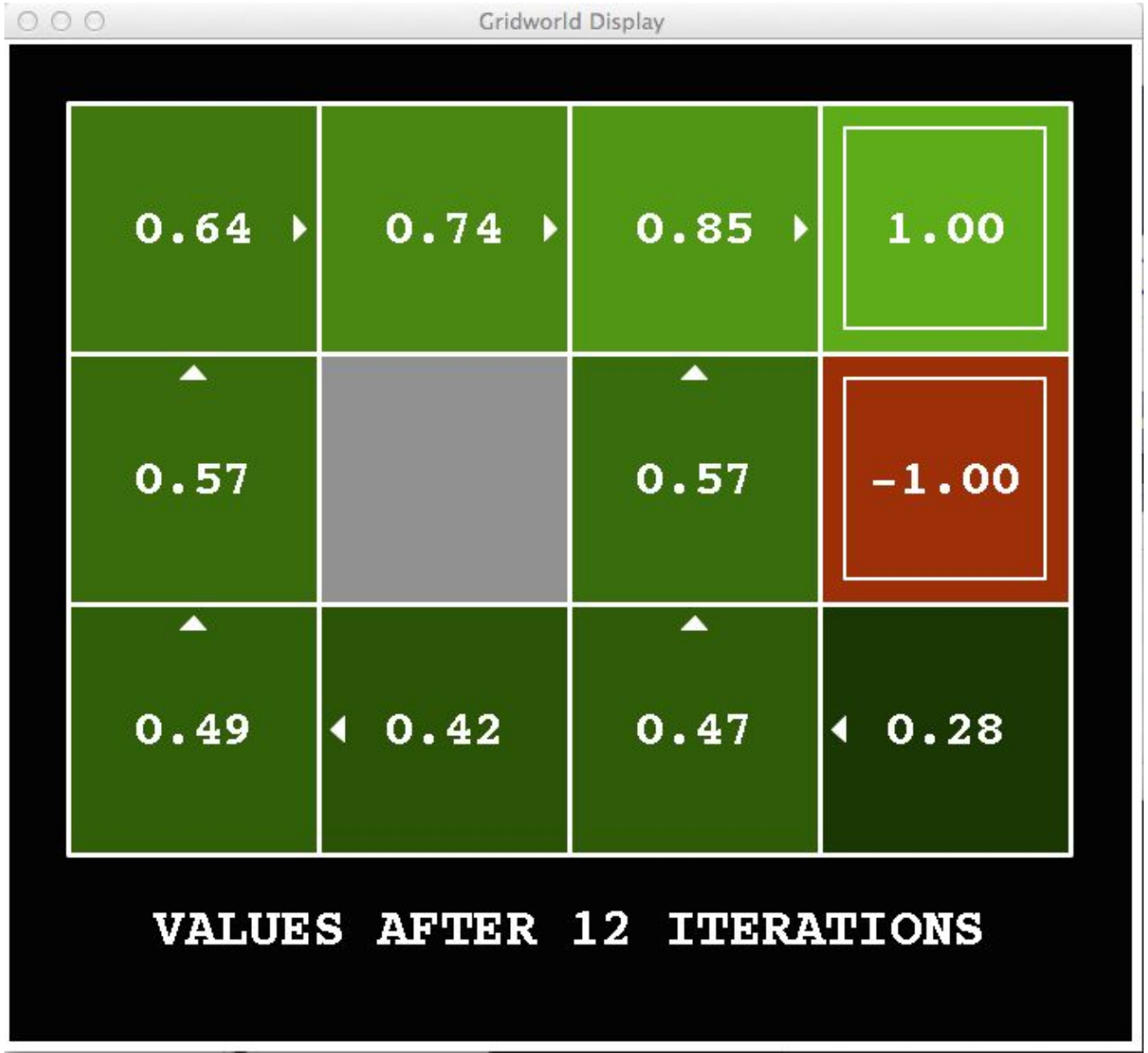
Noise = 0.2
Discount = 0.9
Living reward = 0

t=11



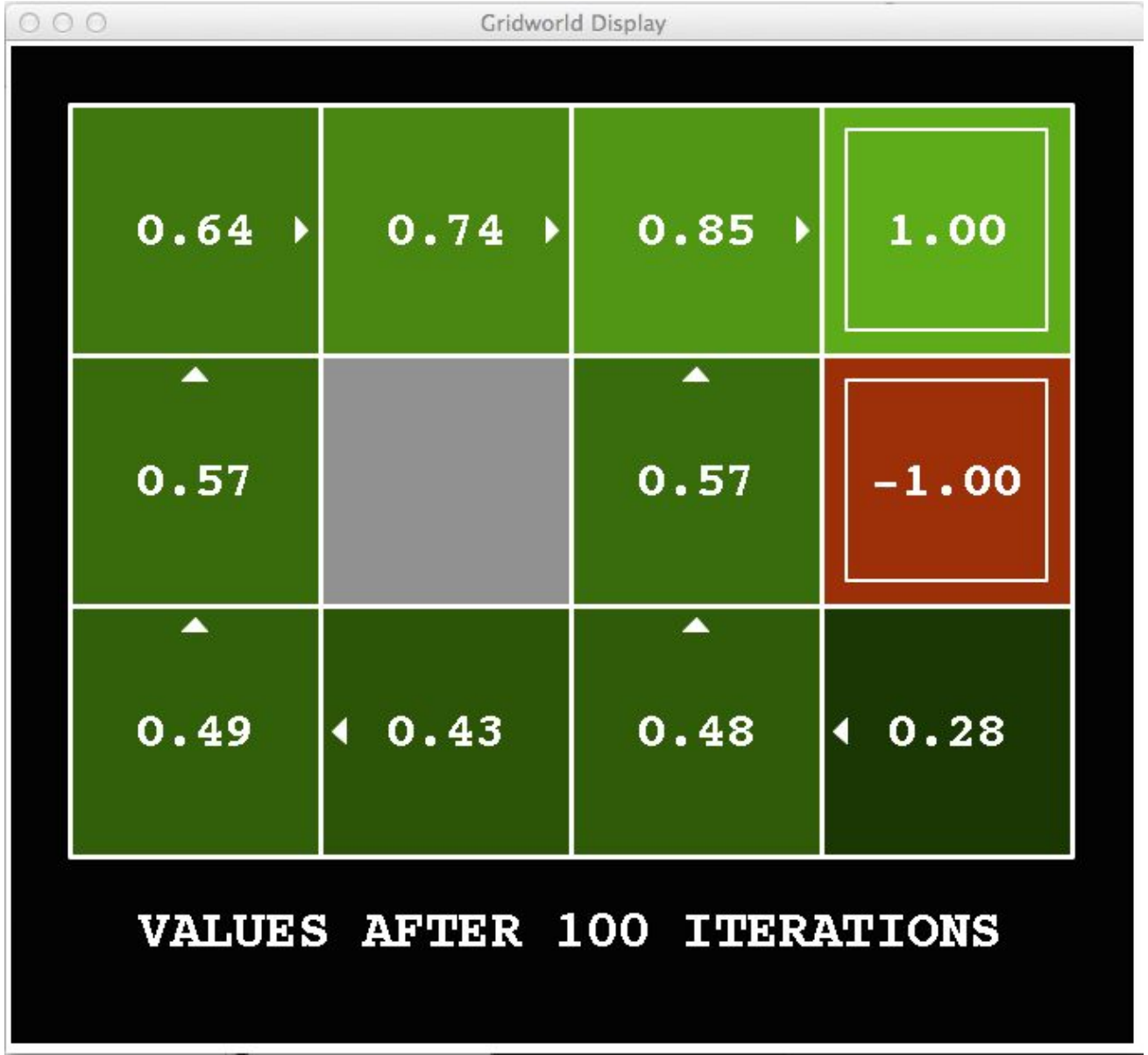
Noise = 0.2
Discount = 0.9
Living reward = 0

t=12



Noise = 0.2
Discount = 0.9
Living reward = 0

t=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

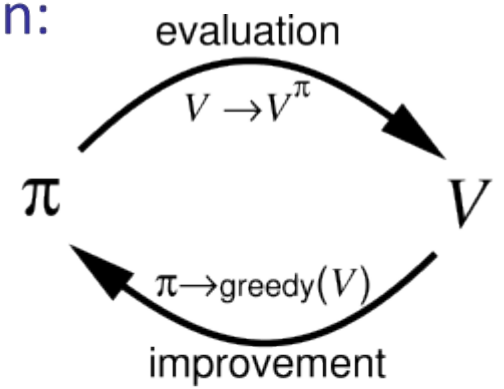
- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Iteration

1. **Initialization**: $i = 0, \pi_i(s) \in A(s)$ arbitrarily for all $s \in S$

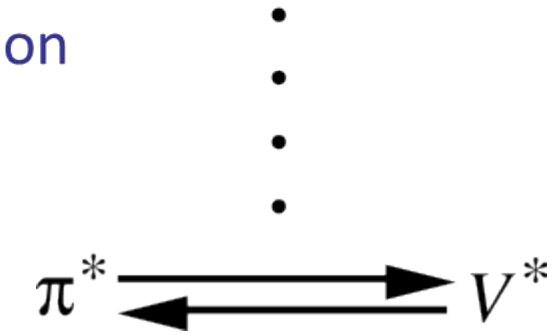
2. **Evaluation**: For fixed current policy π_i, find values with policy evaluation:
   - Iterate until values converge:

$$V_t^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma V_{t-1}^{\pi_i}(s')]$$

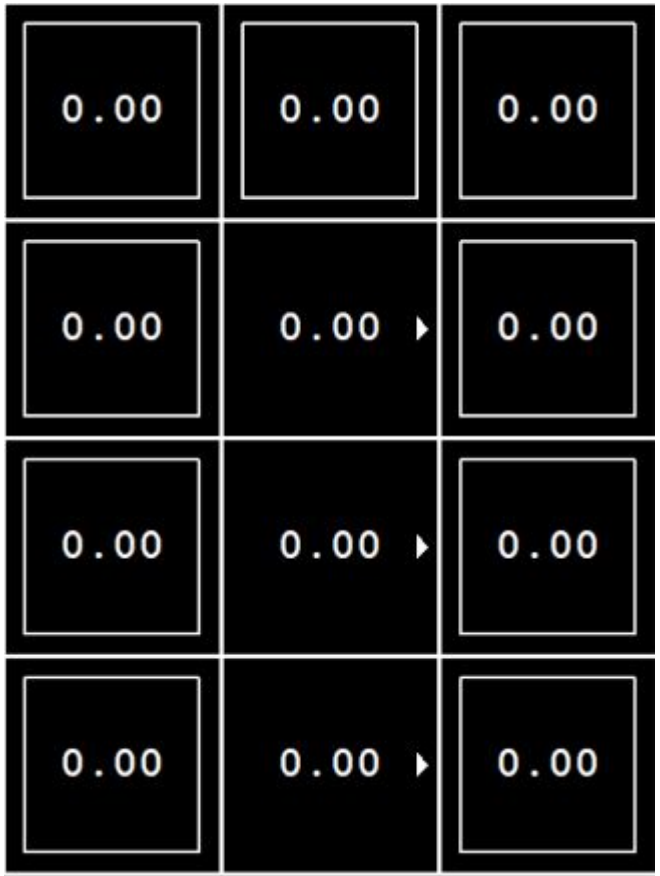


3. **Improvement**: For fixed values, get a better policy using policy extraction
   - One-step look-ahead:

$$\pi_{i+1}(s) = \operatorname*{argmax}_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^{\pi_i}(s')]$$



Policy Evaluation - t=0, t=1



Noise = 0.2
Discount = 0.9
Living reward = 0

$$V_t^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s,\pi_i(s))[R(s,\pi_i(s),s') + \gamma V_{t-1}^{\pi_i}(s')]$$

Policy Evaluation - t=1, t=2

Noise = 0.2
Discount = 0.9
Living reward = 0

$$V_t^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s,\pi_i(s))[R(s,\pi_i(s),s') + \gamma V_{t-1}^{\pi_i}(s')]$$

Policy Evaluation - t=2, t=3

Noise = 0.2
Discount = 0.9
Living reward = 0

$$V_t^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s,\pi_i(s))[R(s,\pi_i(s),s') + \gamma V_{t-1}^{\pi_i}(s')]$$

Policy Evaluation - t=3, t=4

Noise = 0.2
Discount = 0.9
Living reward = 0

$$V_t^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s,\pi_i(s))[R(s,\pi_i(s),s') + \gamma V_{t-1}^{\pi_i}(s')]$$

Policy Evaluation - t=4, t=5

Noise = 0.2
Discount = 0.9
Living reward = 0

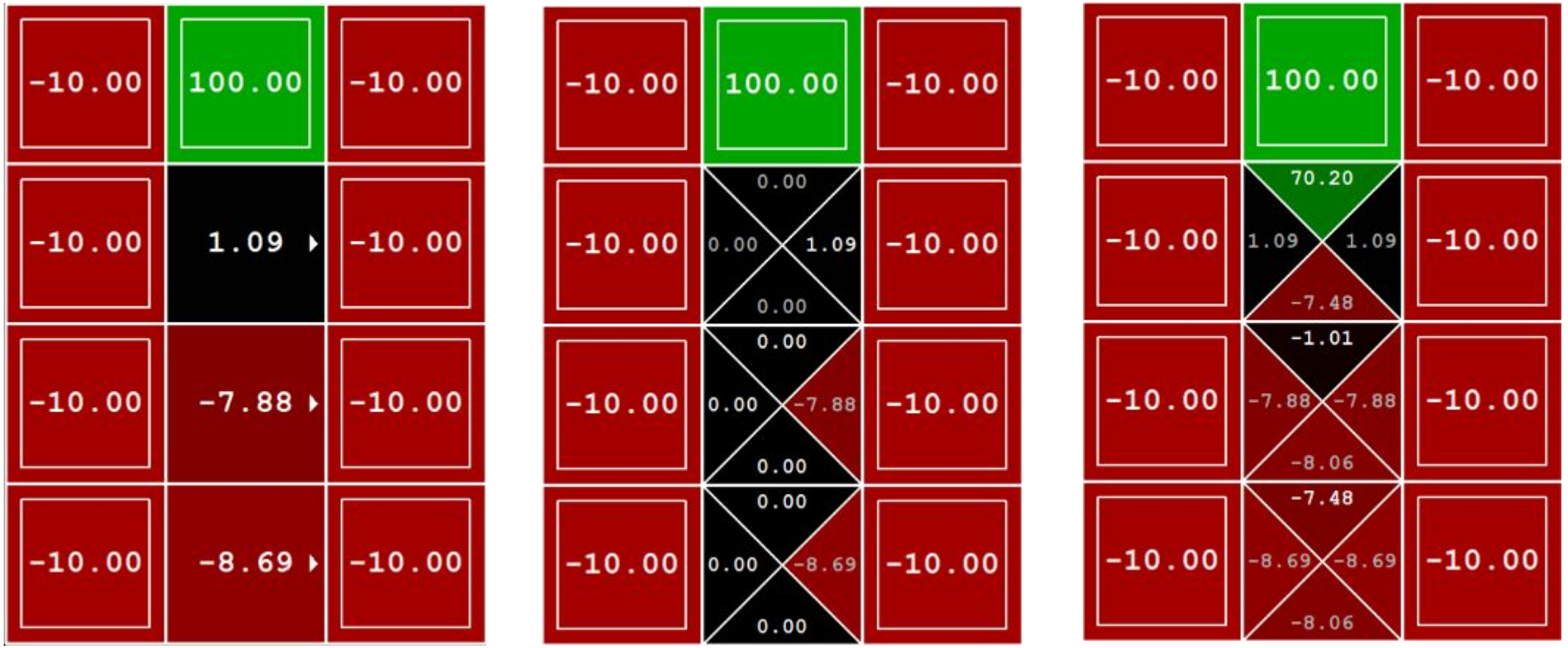
$$V_t^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s,\pi_i(s))[R(s,\pi_i(s),s') + \gamma V_{t-1}^{\pi_i}(s')]$$

Policy Evaluation - t=5, t=6

Noise = 0.2
Discount = 0.9
Living reward = 0

$$V_t^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s,\pi_i(s))[R(s,\pi_i(s),s') + \gamma V_{t-1}^{\pi_i}(s')]$$

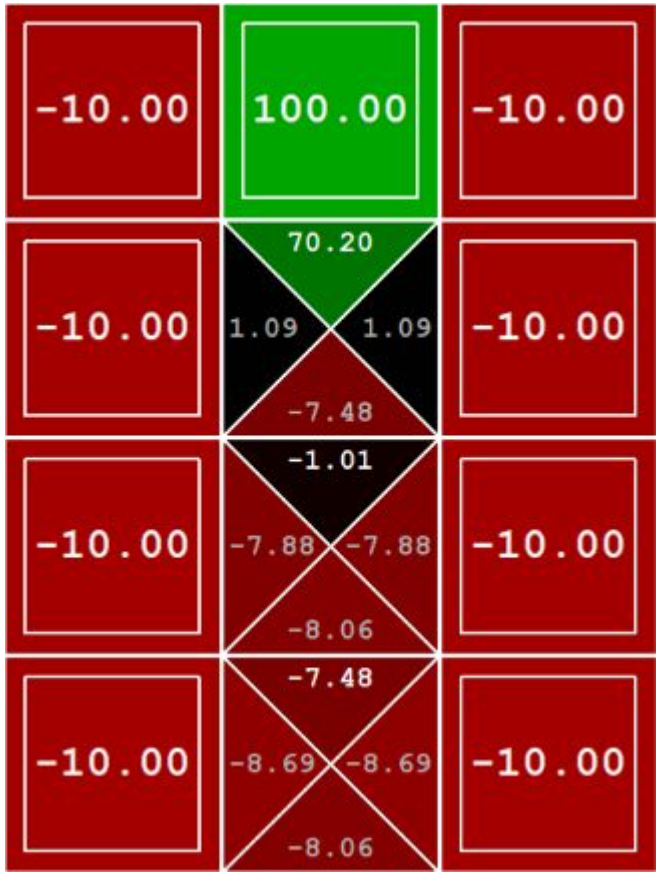
Policy Improvement – t=6



Noise = 0.2
Discount = 0.9
Living reward = 0

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_i}(s')]$$

Policy Improvement – Policy Evaluation



Noise = 0.2
Discount = 0.9
Living reward = 0

$$V_t^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s,\pi_i(s))[R(s,\pi_i(s),s') + \gamma V_{t-1}^{\pi_i}(s')]$$

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