CS6375: Machine Learning

Gautam Kunapuli

Reinforcement Learning

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Reinforcement Learning

Supervised learning: Given **labeled** data $(\boldsymbol{x}_i, y_i), i = 1, \ldots, n$, learn a function $f : \boldsymbol{x} \rightarrow y$

- Categorical y : **classification**
- Continuous y : **regression**

Rich feedback from the environment: the learner is told exactly what it should have done

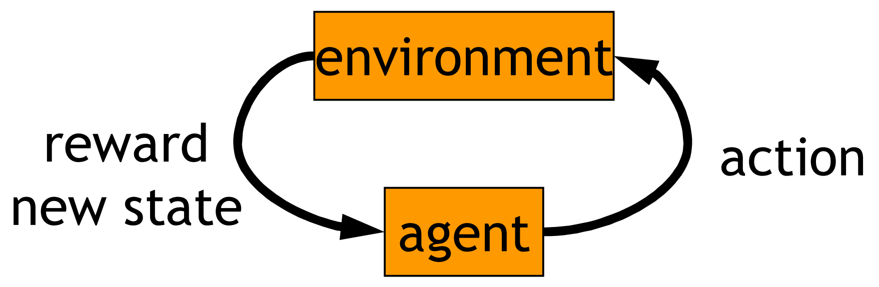
Unsupervised learning: Given **unlabeled** data $\boldsymbol{x}_i$, $i = 1, \ldots, n$, can we infer the underlying structure?

- Clustering
- dimensionality reduction,
- density estimation

No feedback from the environment: the learner receives no labels or any other information

Reinforcement Learning is learning from Interaction: learner (agent) receives feedback about the appropriateness of its actions while interacting with an environment, which provides numeric **reward** signals

Goal: Learn how to take actions in order to maximize reward

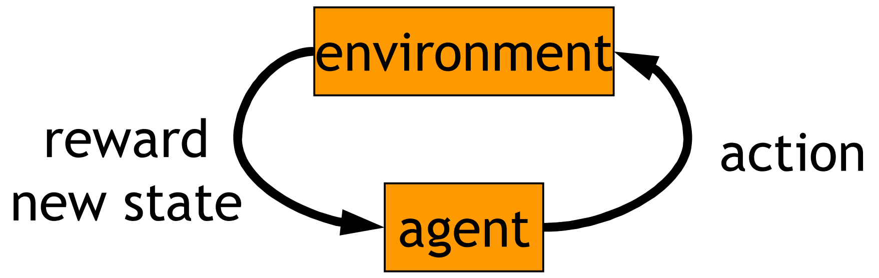


Reinforcement Learning: Key Features

- The learner is **not told what actions to take**, instead it find finds out what to do by **trial-and-error search and acting in the world**
  - e.g.: players trained by playing thousands of simulated games, with no expert input on what are good or bad moves
- The environment is **stochastic**
- The **reward may be delayed**, so the learner may need to sacrifice short-term gains for greater long-term gains
  - e.g.: a player might get reward only at the end of the game, and needs to assign credit to moves along the way
- The learner has to balance the need to **explore** its environment and the need to **exploit** its current knowledge
  - e.g.: one has to try new strategies but also to win games

Reinforcement Learning is learning from Interaction: learner (agent) receives feedback about the appropriateness of its actions while interacting with an environment, which provides numeric **reward** signals

Goal: Learn how to take actions in order to maximize reward

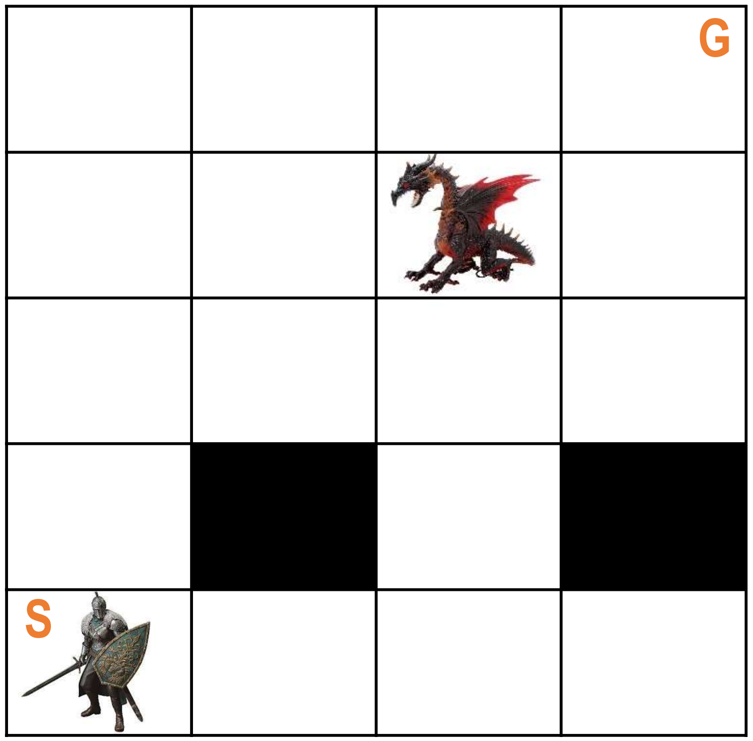


Example: Grid World

Example: Learn to navigate from beginning/start state (S) to goal state (G), while avoiding obstacles

Autonomous "**agent**" interacts with an **environment** through a series of **actions**
• trying to find the way through a maze
• actions include turning and moving through maze
• agent earns rewards from the environment under certain (perhaps unknown) conditions

The agent's goal is to maximize the reward
• we say that the agent learns if, over time, it improves its performance

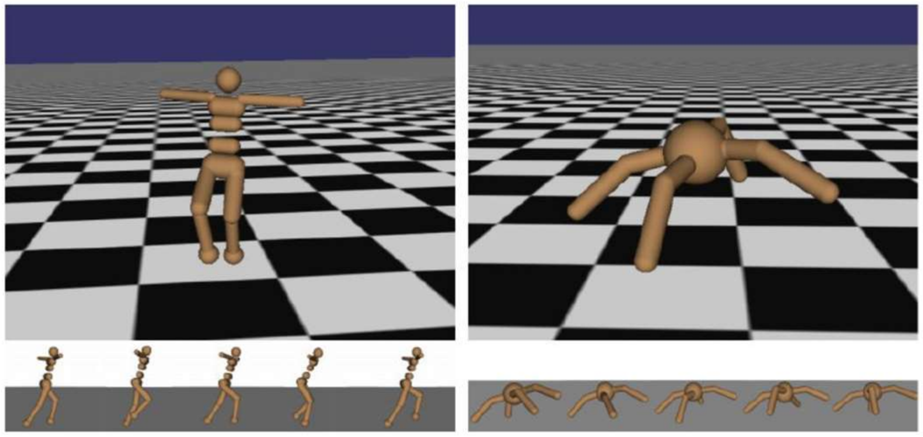


actions are what the agent actually wants to do

effects are what actually happens after the agent executes the chosen action; note that these are *stochastic*

Actions	**Effects**
→ (right)	→ (60%), ↓ (40%)
↑ (up)	↑ (100%)
← (left)	← (100%)
↓ (down)	↓ (70%), ← (30%)

Applications of Reinforcement Learning


Schulman et al (2016)

Robot Locomotion (and other control problems)

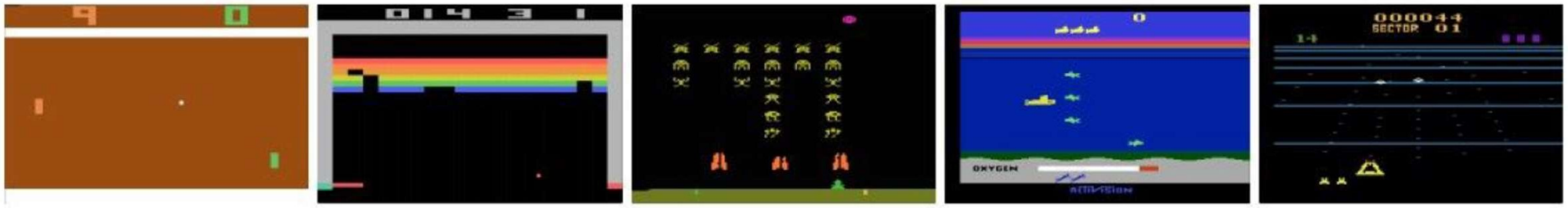
Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

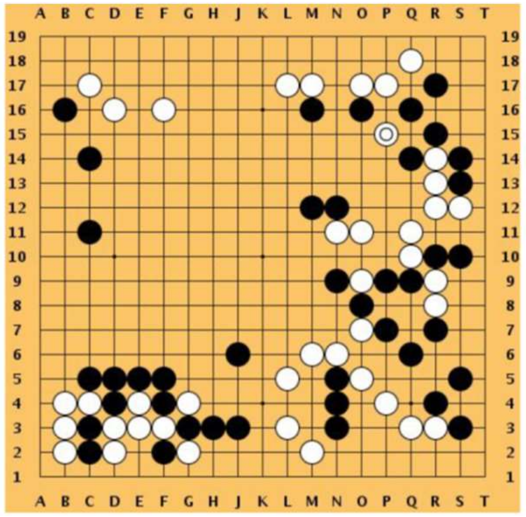
Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step



Applications of Reinforcement Learning



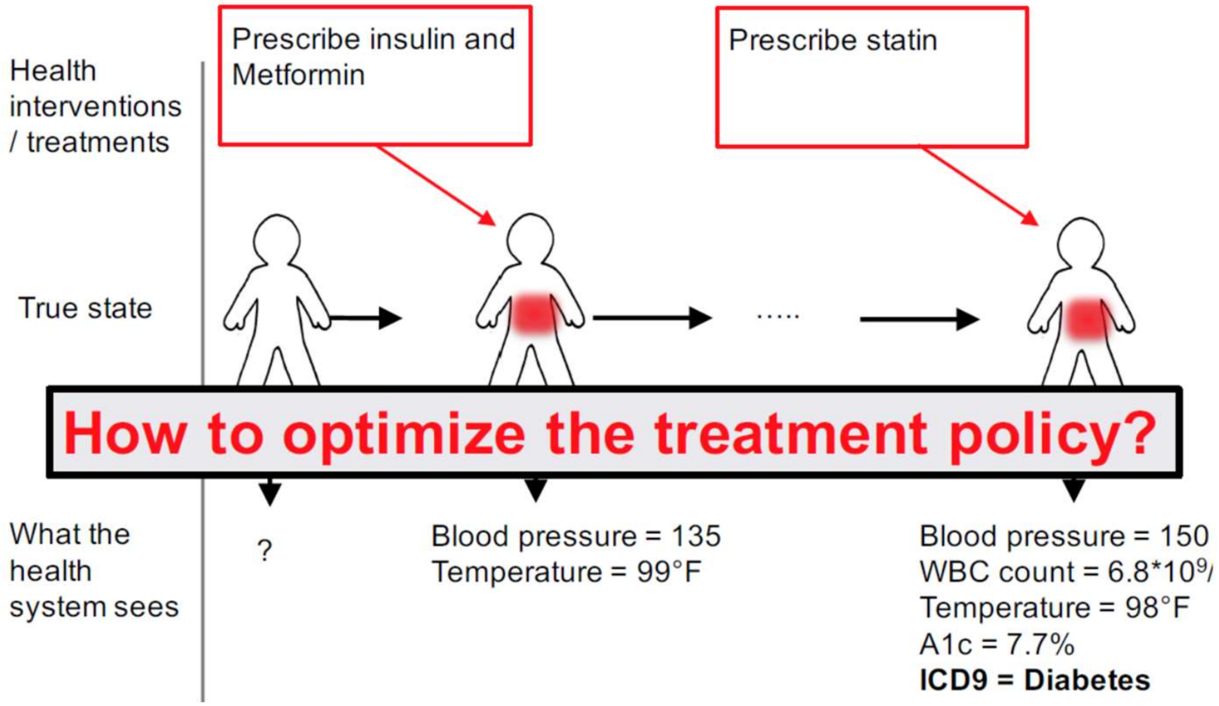
Go!

Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Treatment Planning
Objective: Find the best treatment policy

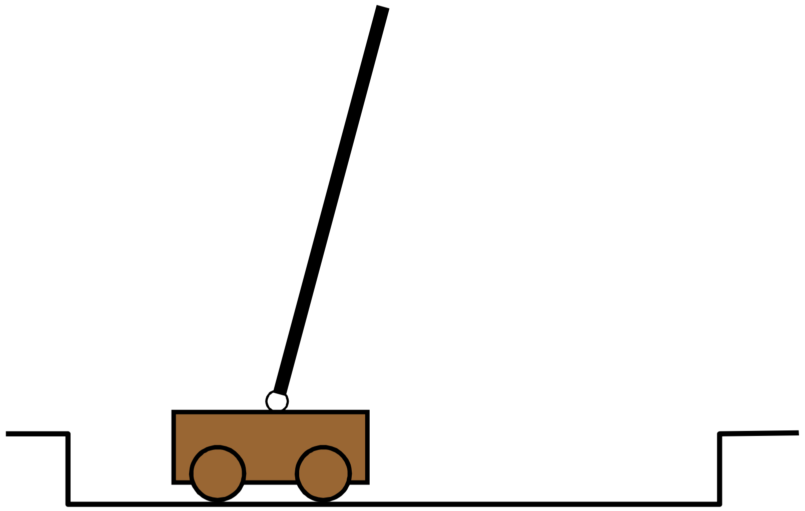
State: Patient health data every 6 months
Action: Clinical interventions and treatment
Reward: negative rewards for deterioration positive rewards for improvement



Reinforcement Learning

Other examples

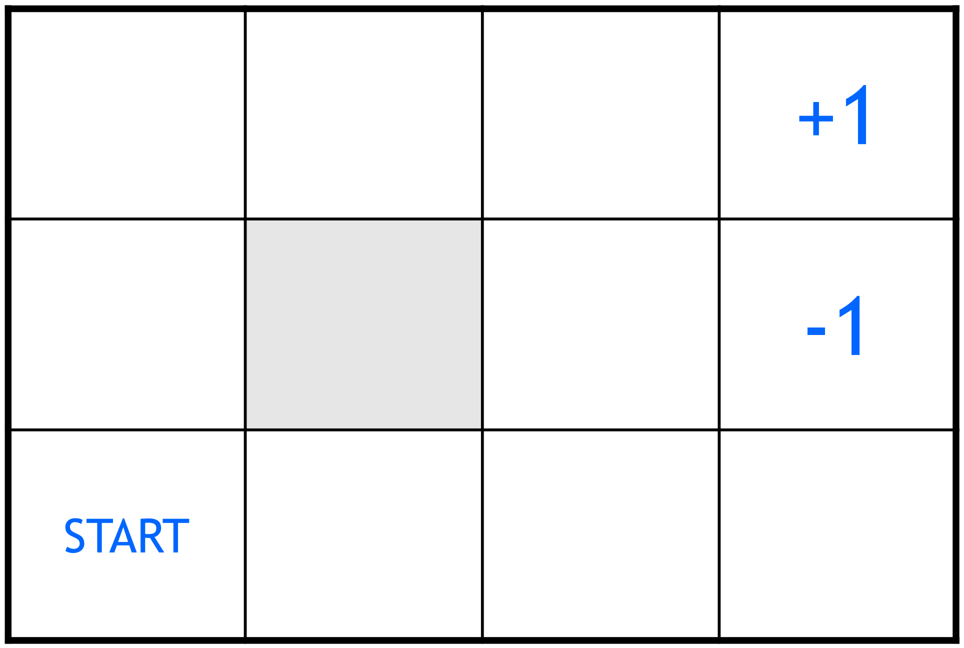
- pole-balancing
- TD-Gammon [Gerry Tesauro]
- helicopter [Andrew Ng]



General challenge: no teacher who would say “good” or “bad”

- is reward “10” good or bad?
- rewards could be delayed
- similar to control theory
    - more general, fewer constraints
- ***explore the environment and learn from experience***
    - not just blind search, try to be smart about it

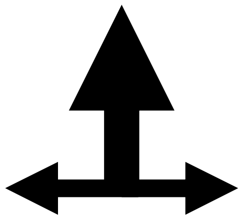
Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

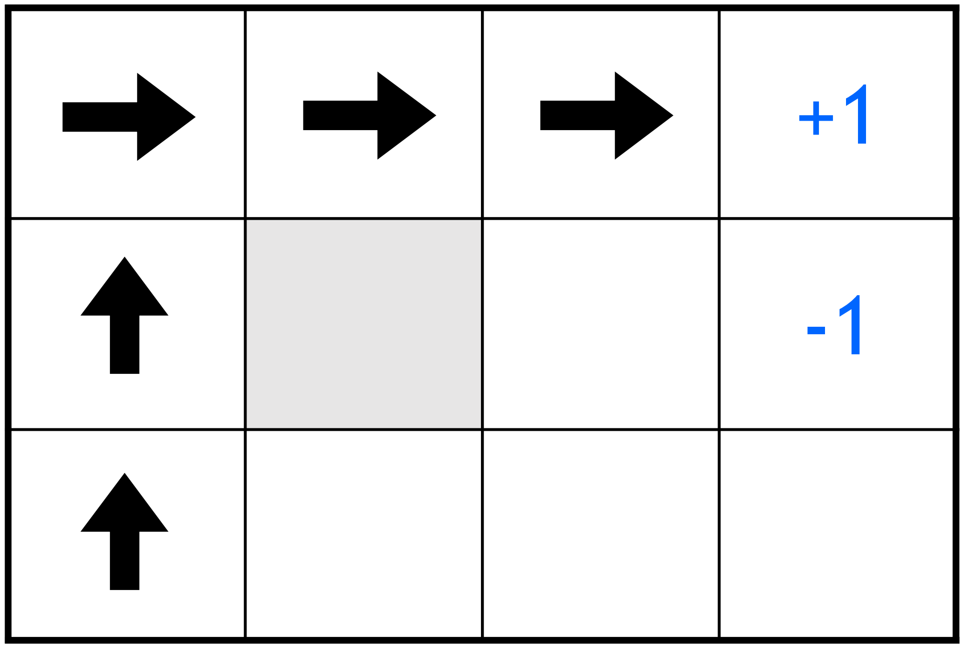
80%	move UP
10%	move LEFT
10%	move RIGHT



reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards
- What is the **solution**? What does the agent **learn**?

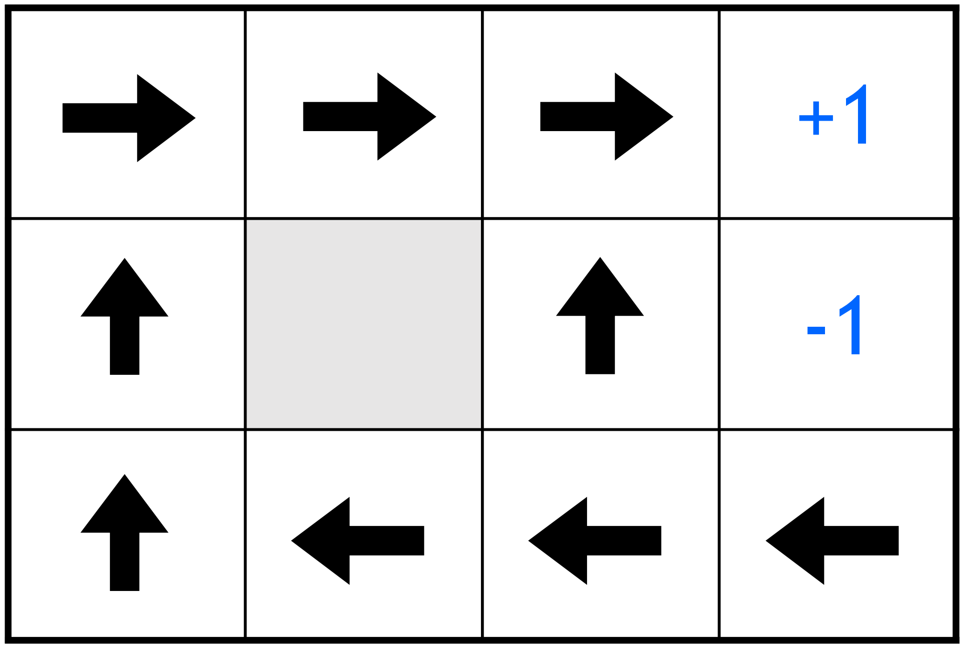
Is This A Solution?



- only if actions are **deterministic**
  - this path is a **plan**
  - not guaranteed to work as actions are **stochastic** (actions have probabilistic effects)

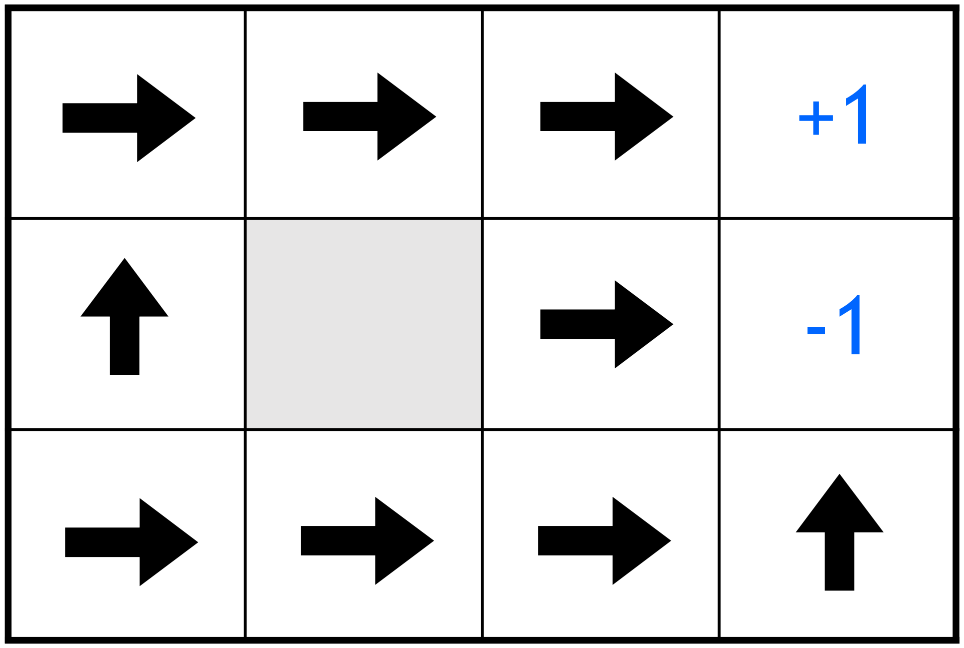
- we need a **policy**
  - mapping from each state to an action
  - agent tries to learn an **optimal policy**; but optimal in terms of what?

Optimal Policy

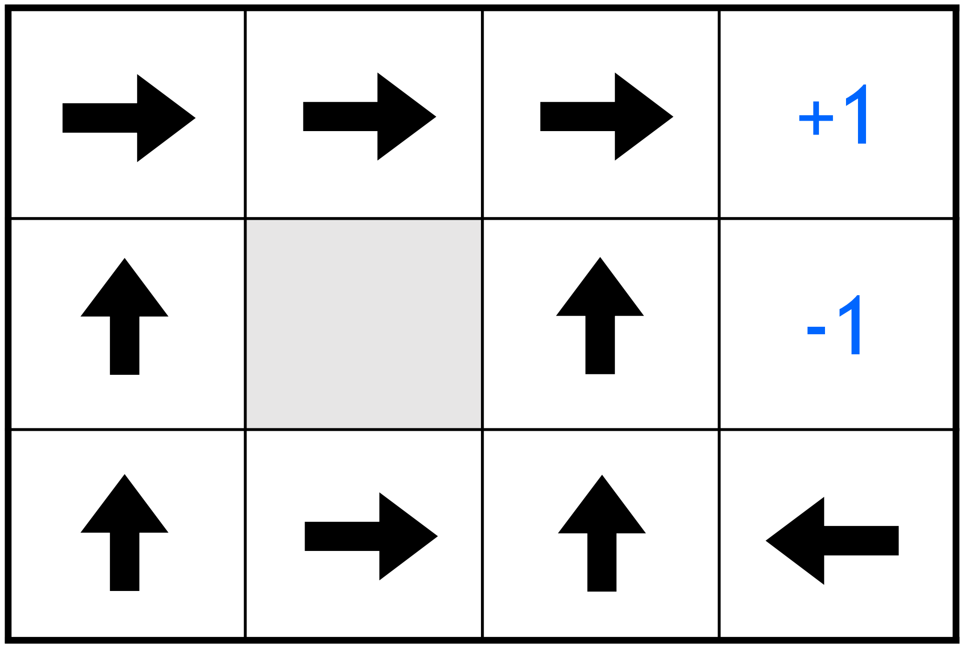


The **optimal policy** will **change with** the kind of **rewards** the agent receives at each episode!

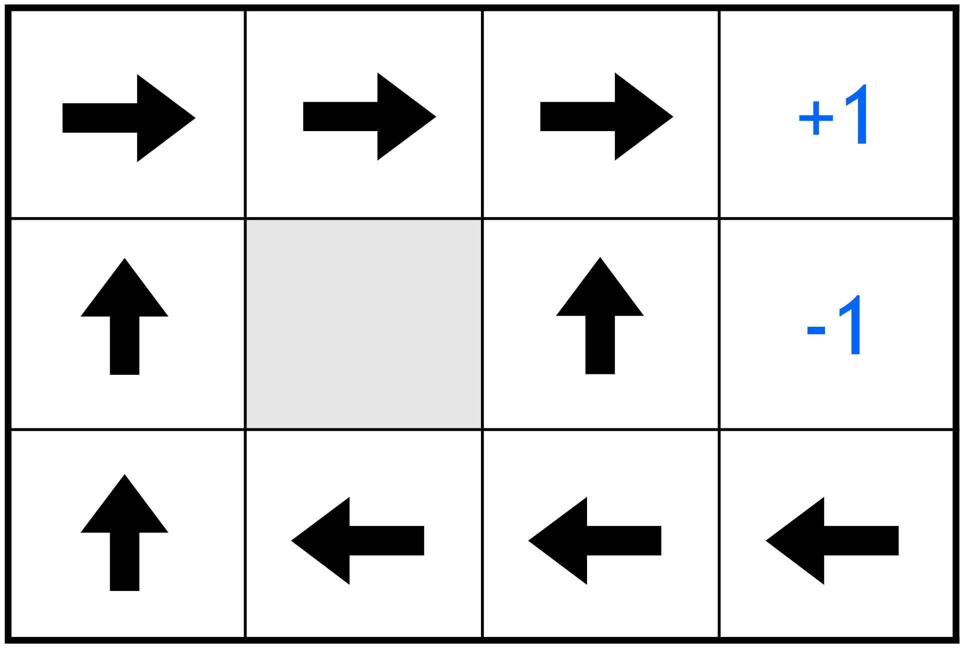
Reward for Each Step: -2.0



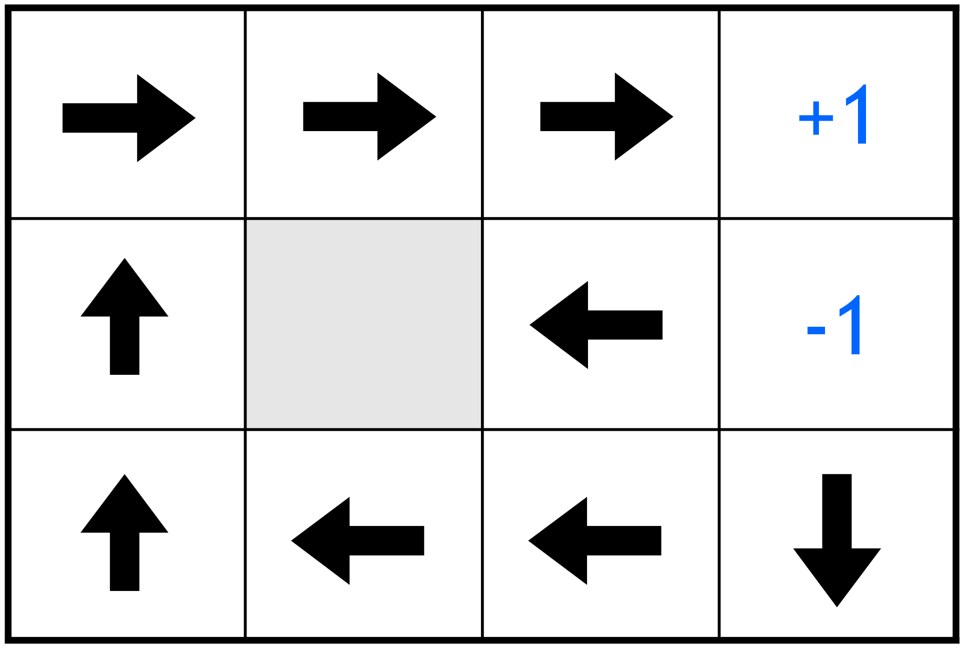
Reward for Each Step: -0.1



Reward for Each Step: -0.04

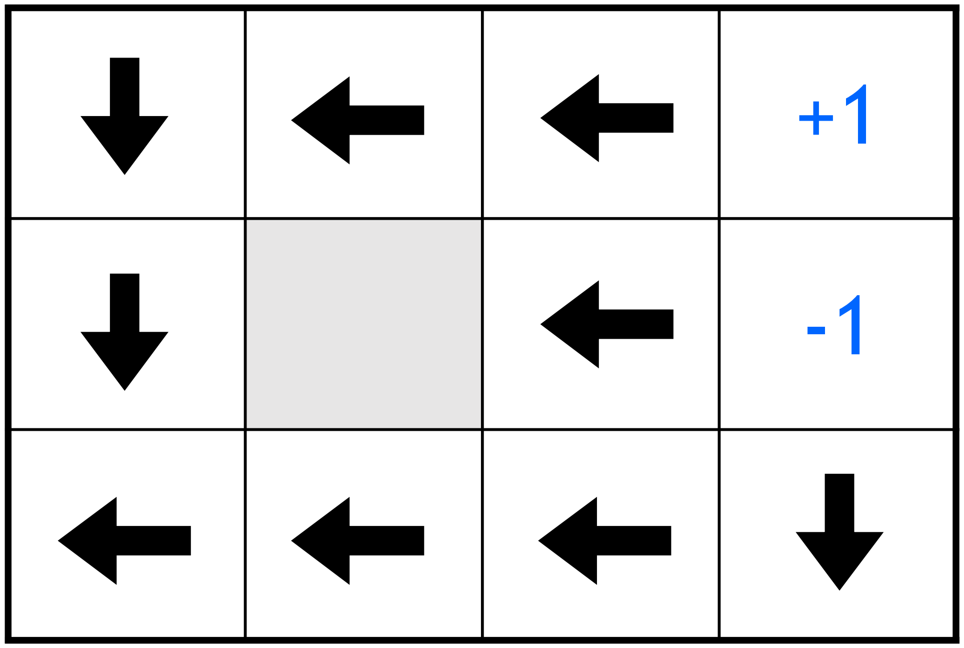


Reward for Each Step: -0.01

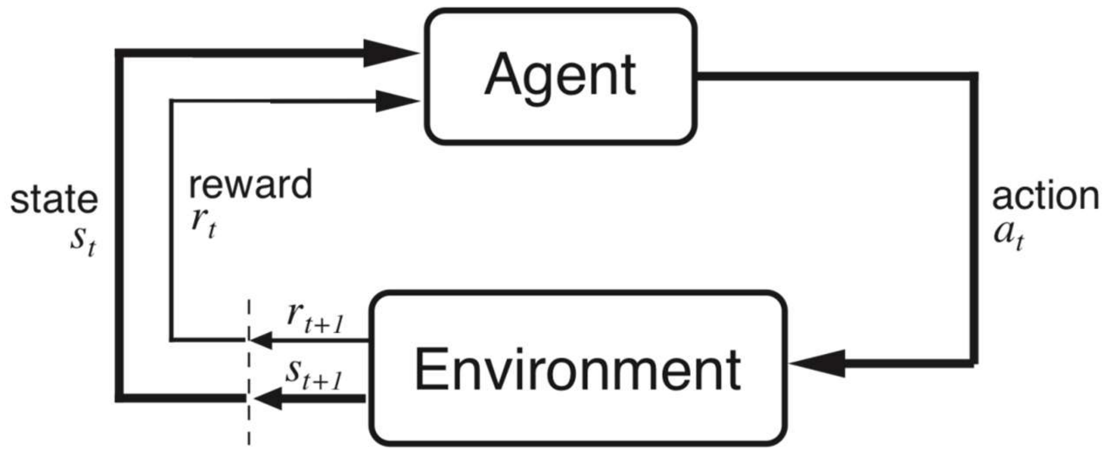


Adapted from slides by Peter Bodik

Reward for Each Step: +0.01



Formalizing RL: The Agent-Environment Interface



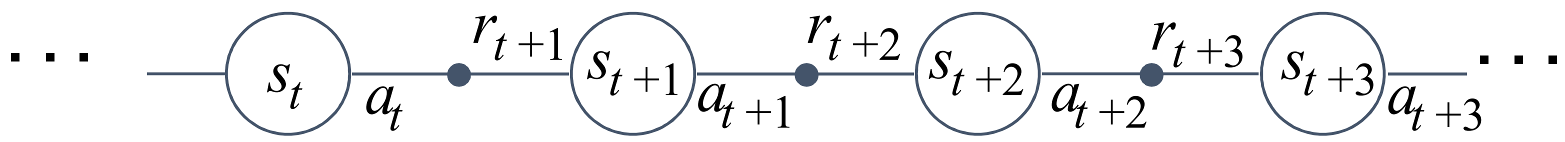
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}

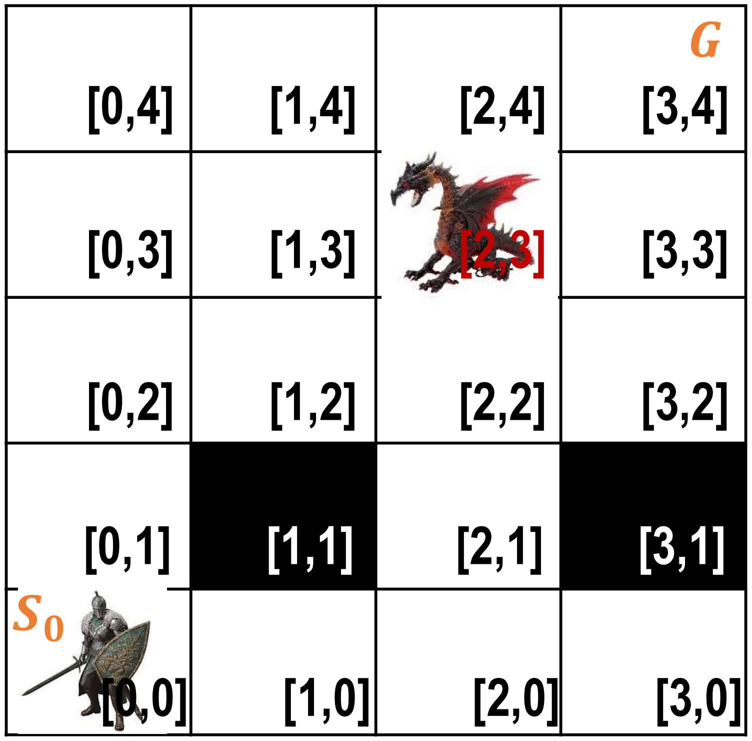


Formalizing RL: Markov Decision Processes

- set of **states** S, set of **actions** A, **initial state** S_0
  - for grid world, can be cell coordinates
- **transition model** $P(s, a, s')$
  - $P(\, [1,1], \uparrow, [1,2]\,) = 0.8$
- **reward function** $r(s)$
  - $r(\,[3,4]\,) = +1$
- **goal**: maximize cumulative reward in the long run
- **policy**: mapping from S to A
  - $\pi(s)$ or $\pi(s, a)$ (deterministic vs. stochastic)
- **discount factor**

Reinforcement Learning

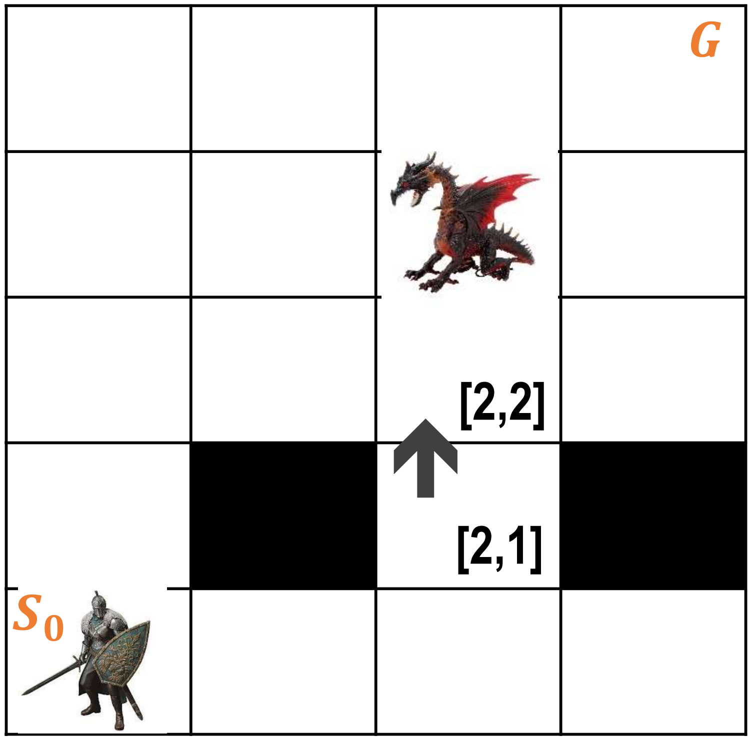
- transitions and rewards usually not available
- how to change the policy based on experience
- how to explore the environment



Actions	Transition Probabilities
→ (right)	→ (60%), ↓ (40%)
↑ (up)	↑ (100%)
← (left)	← (100%)
↓ (down)	↓ (70%), ← (30%)

Formalizing RL: The Markov Property

- "**the state**" at step *t*, means whatever information is available to the agent at step *t* about its environment – **snapshot of the world**
- the state can include immediate "**sensations**", highly processed observations, and structures built up over time from sequences of observations
- ideally, a state should summarize past sensations so as to retain all "**essential**" **information**, i.e., it should have the **Markov Property**:
    - *conditional probability distribution of* ***future states*** *depends only upon the* ***present state****, not on the sequence of events that preceded it*



$$\Pr\left\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, K, r_1, s_0, a_0\right\} =$$

$$\Pr\left\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\right\}$$

for all s', r, and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, K, r_1, s_0, a_0$.

Formalizing RL: Rewards and Policy

- **episodic tasks**: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze
- **non-episodic tasks**: no episodes; infinite game. e.g. a self-driving car
- **additive rewards**
  - $R = r(s_0) + r(s_1) + r(s_2) +$
  - infinite value for continuing tasks
- **discounted rewards**
  - rewards are **discounted** by a discount factor $\gamma \in [0, 1)$
  - $R = r(s_0) + \gamma * r(s_1) + \gamma^2 * r(s_2) + \ldots$

Learning Problem: Find a policy that maximizes the **total expected reward**, $E_\pi[\![\sum_{t=1}^{\infty} \gamma^t r_t]\!]$

A policy $\pi(s)$ or $\pi(s, a)$ is the prescription by which the agent selects an action to perform

- **Deterministic**: the agent observes the state of the system and chooses an action
- **Stochastic**: the agent observes the state of the system and then selects an action, at random, from some probability distribution over possible actions

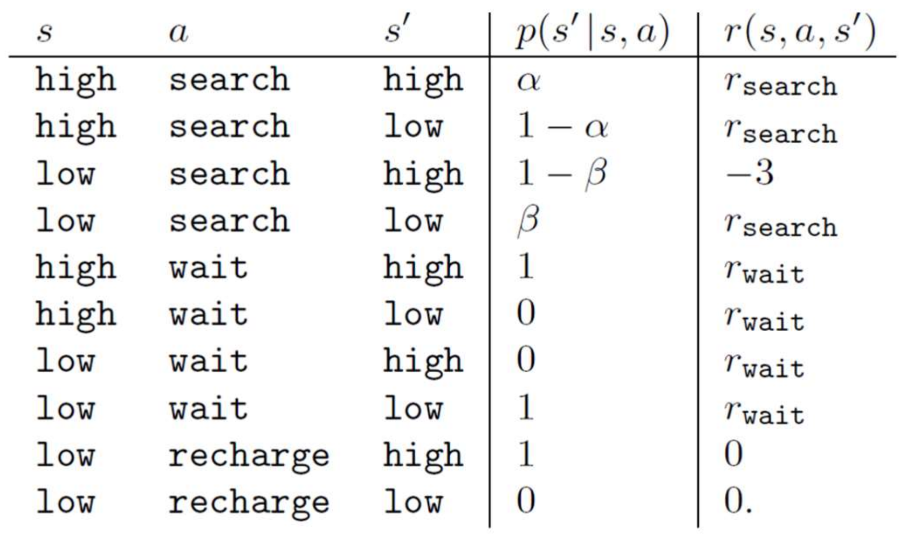
Example: The Recycling Robot

A **mobile robot** collects empty soda cans in an office. This agent has to decide whether to:

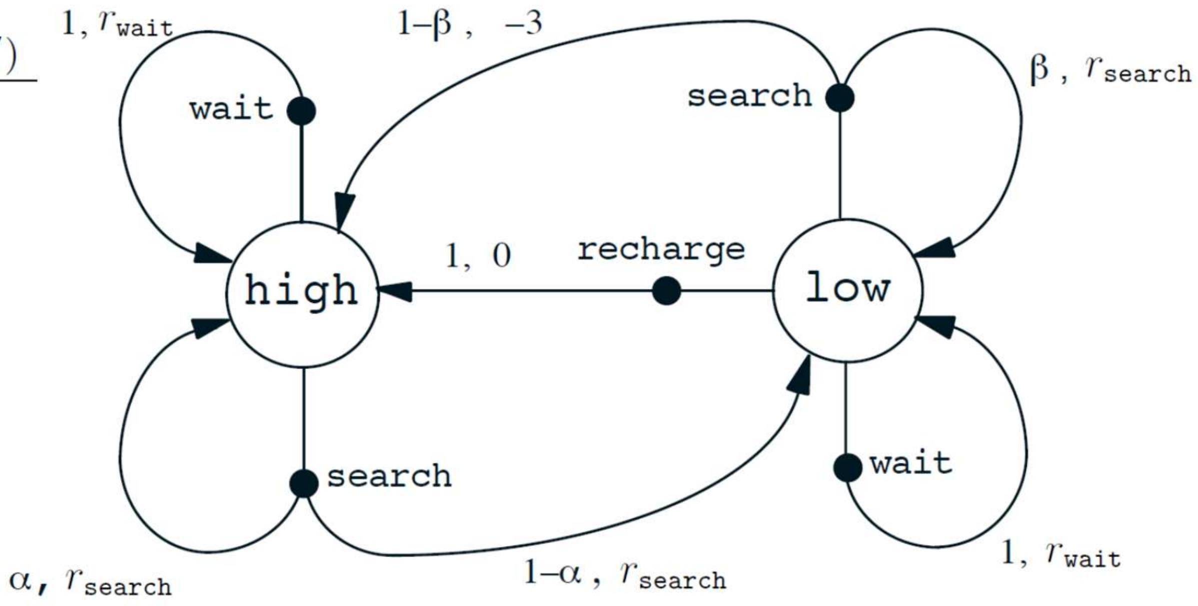
- actively **search** for a can for a certain period of time;
- **remain** stationary and wait for someone to bring it a can, or
- head back to its home base to **recharge** its battery.

Agent has

- three actions, and the state is primarily determined by the state of the battery;
- rewards might be zero most of the time,
    - but then become positive when the robot secures an empty can,
    - or large and negative if the battery runs all the way down.



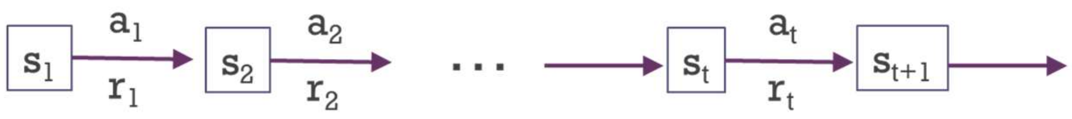
s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	r_{wait}
low	wait	high	0	r_{wait}
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	$0.$



Adapted from Sutton and Barto

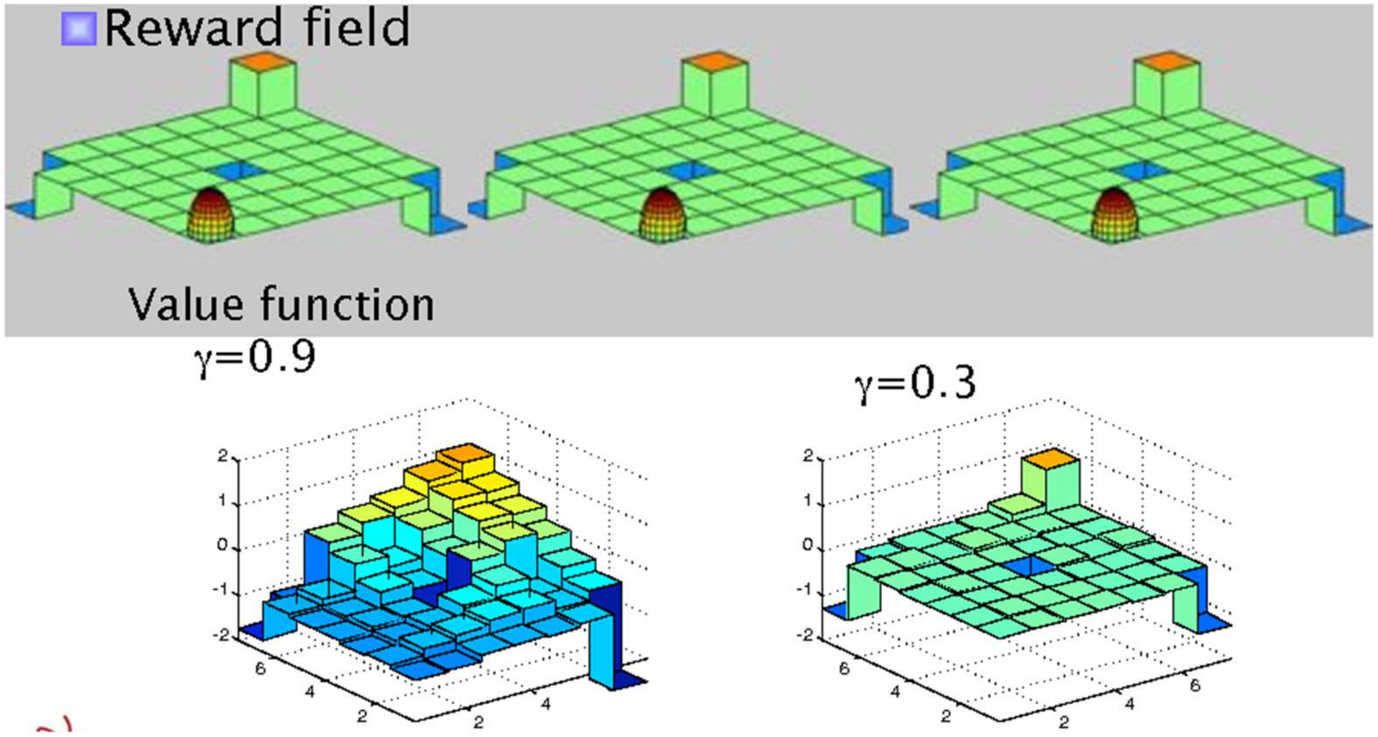
The (State) Value Function

A **value function** assigns a real number to **each state** called its **value**. The **value of a state** (s) is the expected reward *starting from that state* (s) *and then following the policy* π.



$$v^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, a_{t+1}) \,|s_0 = s\right]$$

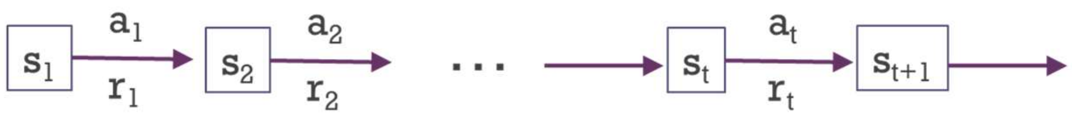
The value function helps us evaluate the quality of a policy π. Informally, the value of a state indicates how much better it is to be in that state than other states, when following π.



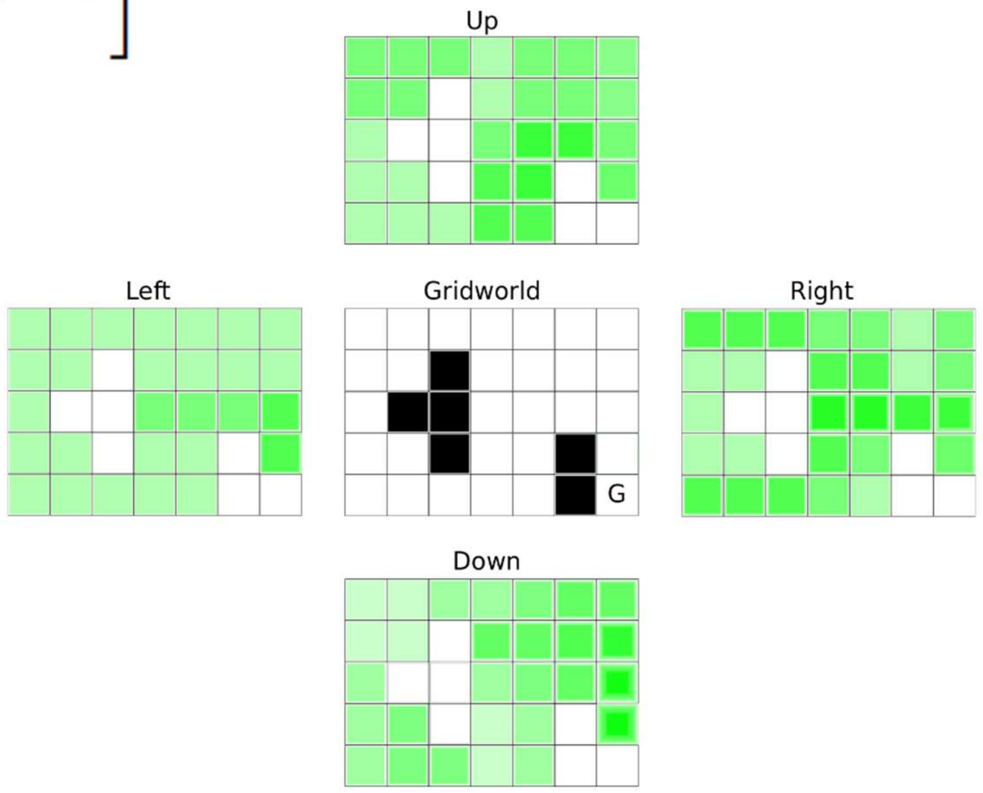
Adapted from slides by Kenji Doya

The (Action) Value Function

The **action value function** assigns a real number to **each state-action pair** called its **q-value**. The **q-value of a state** is the expected reward *starting from that state* (s)*, executing that action* (a) *and then following the policy* π.

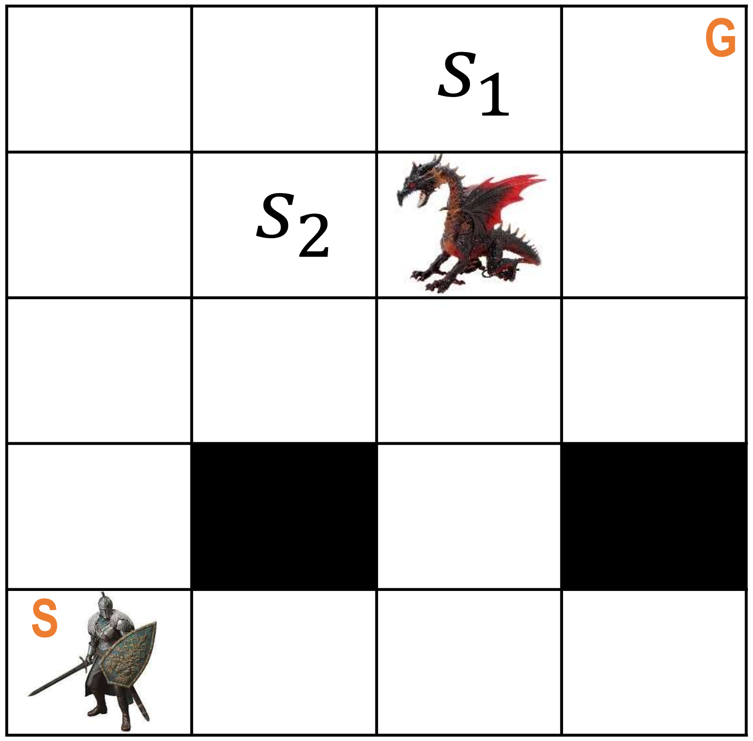


$$q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, a_{t+1}) \,|s_0 = s,\, a_0 = a\right]$$

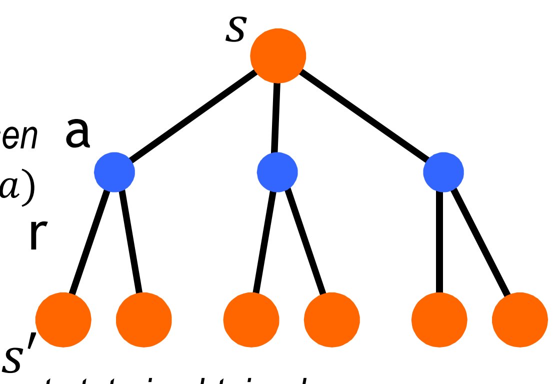


Value Functions

- Which states should have a higher value? s_1 or s_2?
- Which action should have a higher value in s_1? ➔or 🡇?
- Which action should have a higher value in s_2? ➔or 🡅?



Value Functions



***agent**: each action is chosen with (policy) probability $\pi(s,a)$*

***environment**: next state is obtained with (transition) probability $P(s' \mid s,a)$*

Unroll the discounted reward:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \cdots$$
$$= r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots)$$
$$= r_t + \gamma R_{t+1} \text{ (recurrence)}$$

Recall the definition of the value function:

$$V_\pi(s) = E_\pi[\![R_t \mid s_t = s]\!] = E_\pi[\![R_t + \gamma\, V_\pi(s_{t+1}) \mid s_t = s]\!]$$

(another recurrence relation)

Unrolling the expectation using transition probabilities:

$$V_\pi(s) = R(s) + \gamma \sum_{a \in A(s)} \pi(s,a) \sum_{s'} P(s' | s,a)\, V_\pi(s')$$

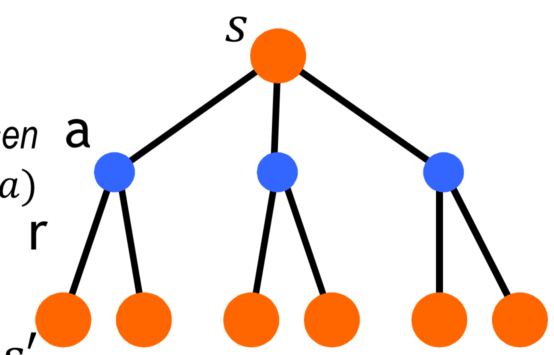
This is one of the **Bellman equations**.

immediate reward

*value of a state is the expected sum of discounted rewards when **starting from that state***

*expected sum of discounted rewards **after the first step** from s **taking into account all possible next states** s' **from all possible next actions** $a \in A(s)$*

Value Functions



***agent**: each action is chosen with (policy) probability $\pi(s,a)$*

***environment**: next state is obtained with (transition) probability $P(s' \mid s,a)$*

Recall the definition of the Q-value function:

$$Q_\pi(s,a) = E_\pi[\![R_t \mid s_t = s, a_t = a]\!]$$
$$= E_\pi[\![R_t + \gamma\, V_\pi(s_{t+1}) \mid s_t = s, a_t = a]\!]$$

Unrolling the expectation using transition probabilities:

$$\underbrace{Q_\pi(s,a)}_{\text{value of a state-action pair}} = \underbrace{R(s)}_{\text{immediate reward}} + \gamma \sum_{s'} P(s'|s,a)\, V_\pi(s')$$

immediate reward

*value of a state-action pair is the expected sum of discounted rewards when **starting from that state and executing that action***

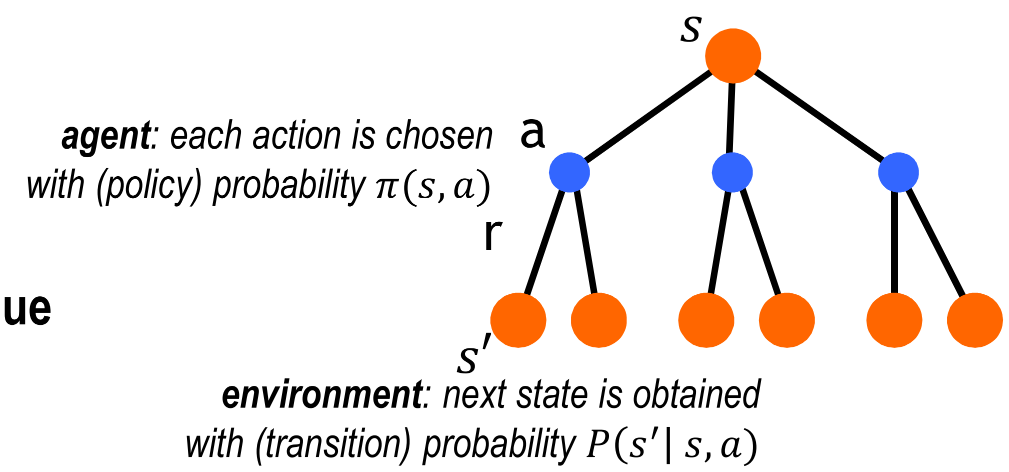
*expected sum of discounted rewards **after the first step** from s **taking into account all possible next states** s' **from executing action** a*

This is another of the **Bellman equations**.

Compare with the state-value function, which considers all actions $a \in A(s)$:

$$V_\pi(s) = R(s) + \gamma \sum_{a \in A(s)} \pi(s,a) \sum_{s'} P(s'|s,a)\, V_\pi(s')$$

Optimal Value Functions



s
a
r
s'
agent: *each action is chosen with (policy) probability* $\pi(s,a)$
environment: *next state is obtained with (transition) probability* $P(s' \mid s,a)$

V^π defines a **partial ordering on policies**, that is, **value functions** are useful for finding the **optimal policy**.

Learning problem: find a policy $\pi*: S \rightarrow A$ such that

$$V^{\pi^*}(s) \geq V^\pi(s)$$

for all $s \in S$ and all policies π.

- any policy satisfying this condition is called an **optimal policy** (*may not be unique*)
- there **always** exists an optimal policy
- optimal policies share the same optimal value function

$$V^*(s) = \max_\pi V^\pi(s)$$

Bellman Optimality Equations

$$V^*(s) = \max_{a \in A(s)} R(s) + \gamma \sum_{s'} P(s'|s,a)\, V^*(s')$$

$$Q^*(s,a) = \max_{a \in A(s)} R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a\prime} Q^*(s',a')$$

- system of *n* (= number of states) non-linear equations describing a recurrence relation between current and next states
- for a finite-state MDP, we obtain a system of linear equations
- solve for $V^*(s)$
- easy to extract the optimal policy

The optimal policy π^* that satisfies these equations is **the optimal policy for all states** s. That is, it does not matter if we start in a state s or a different state s', that is, we can use the same policy π^* no matter the initial state of our MDP.

Solving the MDP: Policy Iteration

Policy iteration (using iterative policy evaluation)

```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

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

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

- *initialize to a random policy*
- *find the value function corresponding to this policy using iterative policy evaluation*
- *can be done by **solving a system of equations***

$$V^{\pi}(s) = R(s) + P_{s\pi(s)}V(s)$$

*or by **iterative policy evaluation***

$$V_{k+1}^{\pi}(s) = \sum_a \pi(s,a) \cdot \sum_{s'} P_{sa}^{s'}\,[R(s') + \gamma V_k(s')]$$

- *improve the policy based on the new values*
- *repeat until policy has converged*

Policy iteration: iteratively perform **policy evaluation + policy improvement**, which are repeated iteratively until policy converges

Solving the MDP: Value Iteration

Drawback of policy iteration: each iteration involves policy evaluation, which may itself be a computationally expensive requiring multiple sweeps through the states
Special case: policy evaluation is stopped after just one sweep (one update of each state)
This algorithm is called **value iteration**.
- effectively combines one sweep of **policy evaluation** and one sweep of **policy improvement**

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
    $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
```

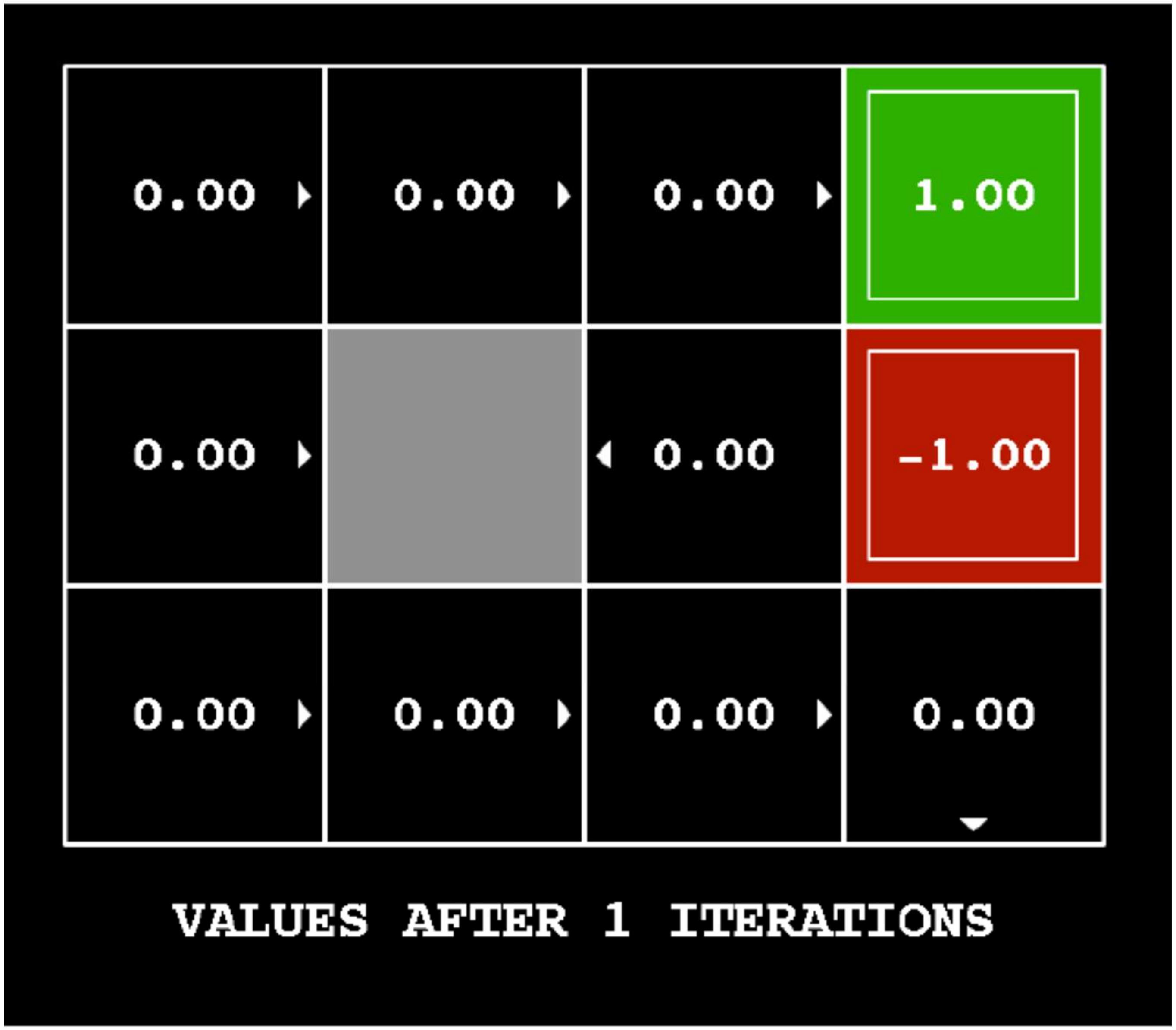
▸ *initialize values to zero*

▸ effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement

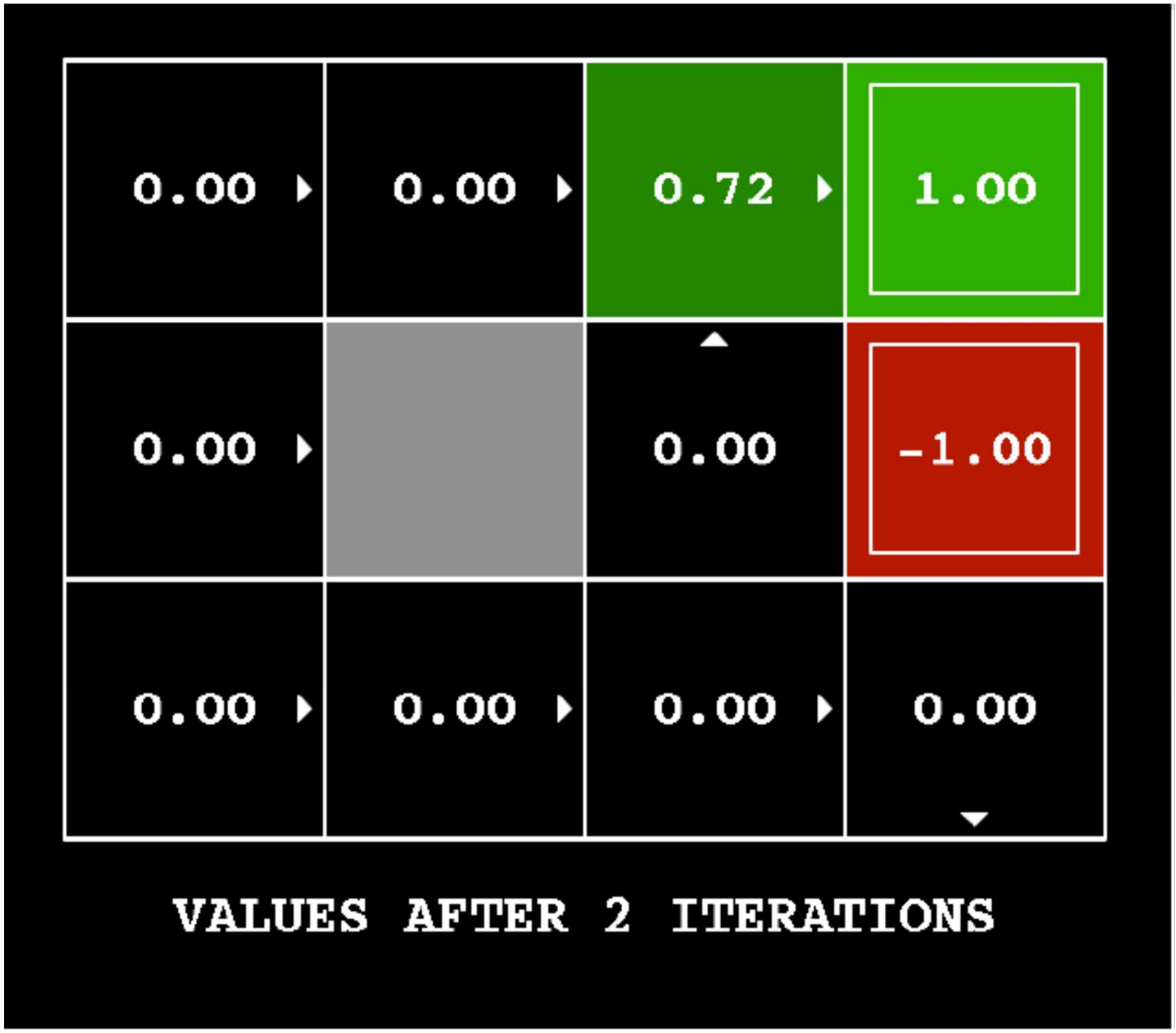
▸ *one-time policy extraction*

Value iteration: directly **find optimal value function** and extract the **optimal policy** from it

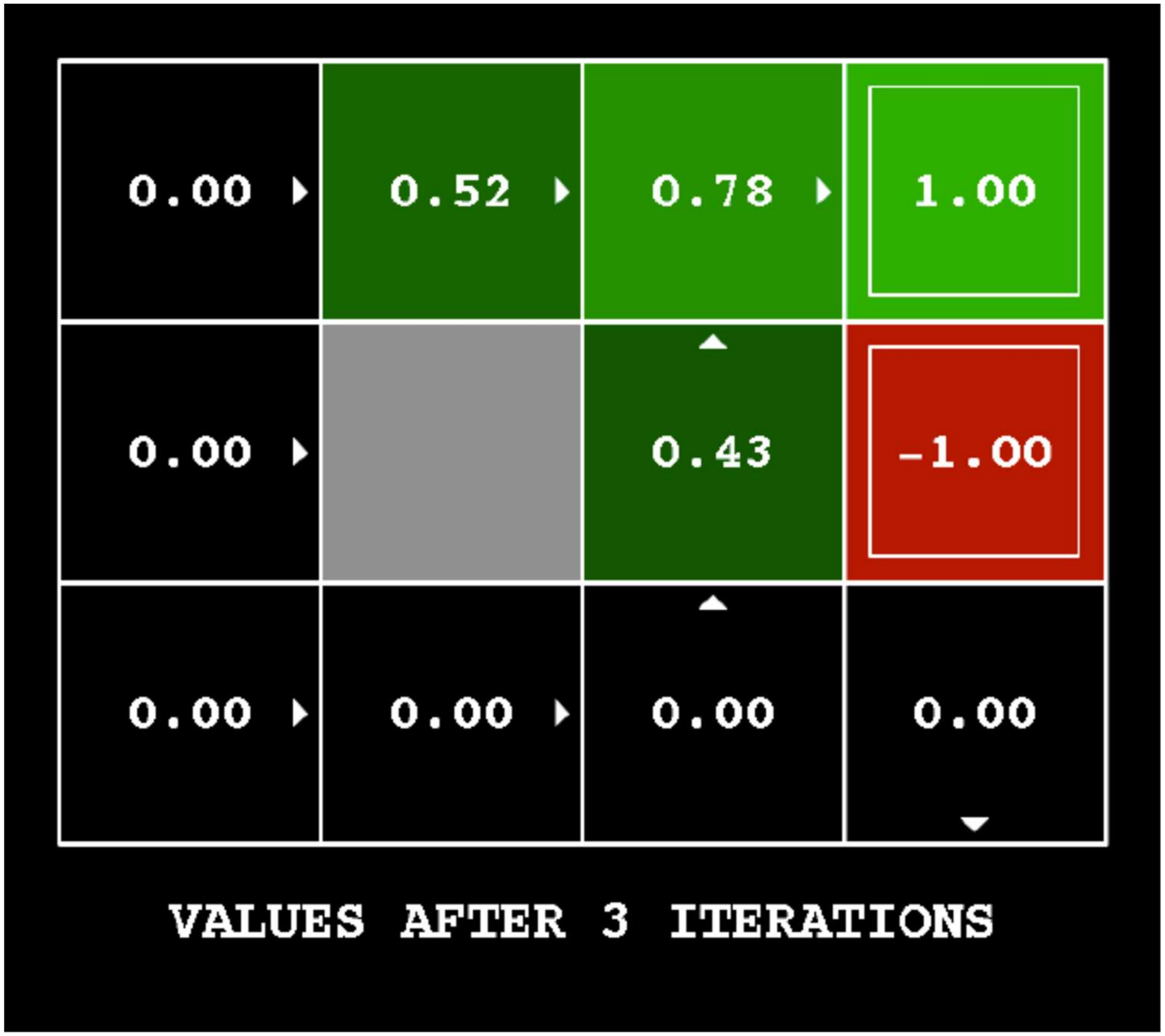
Value Iteration in GridWorld ($\gamma = 0.9$)



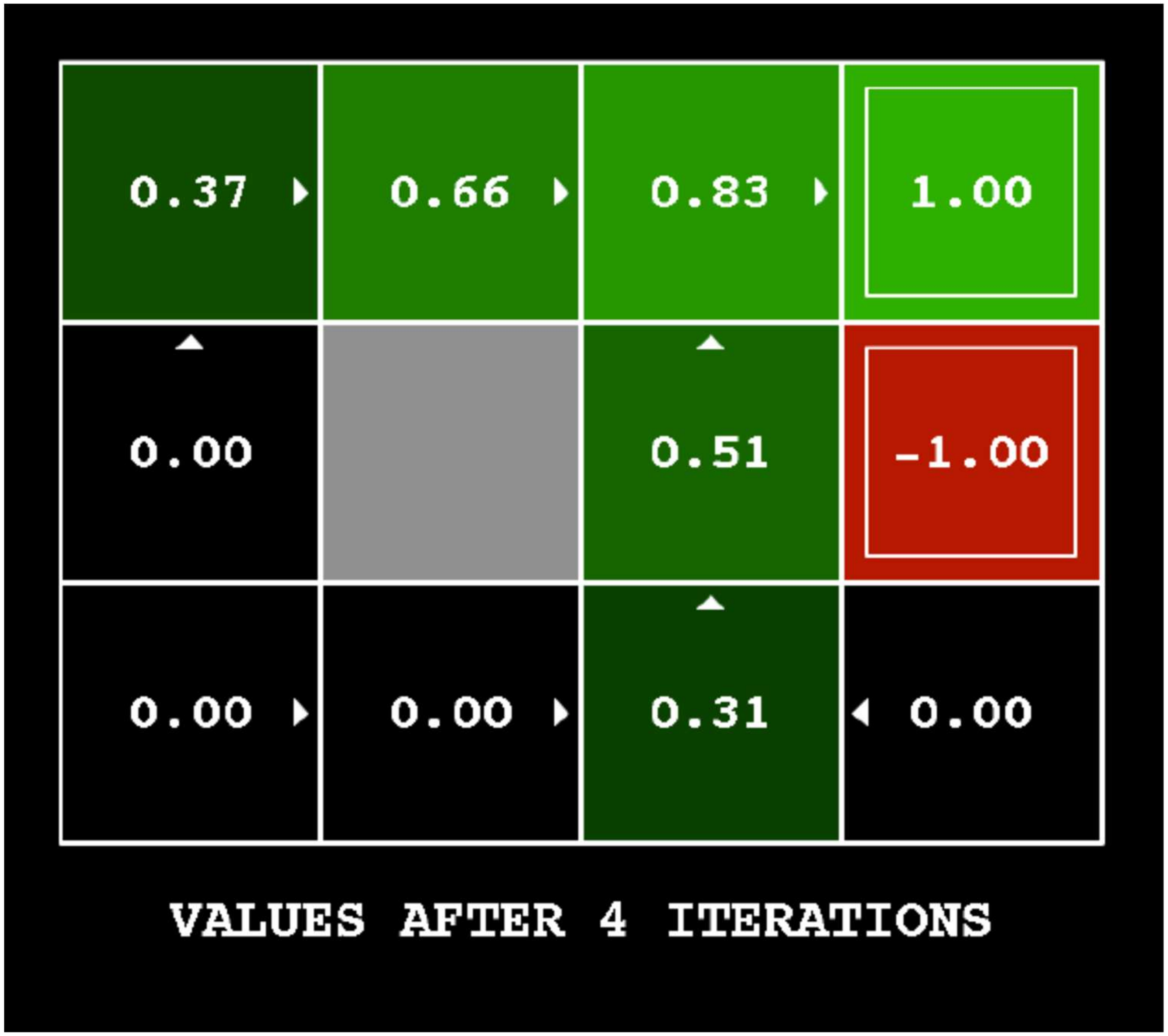
Value Iteration in GridWorld ($\gamma = 0.9$)



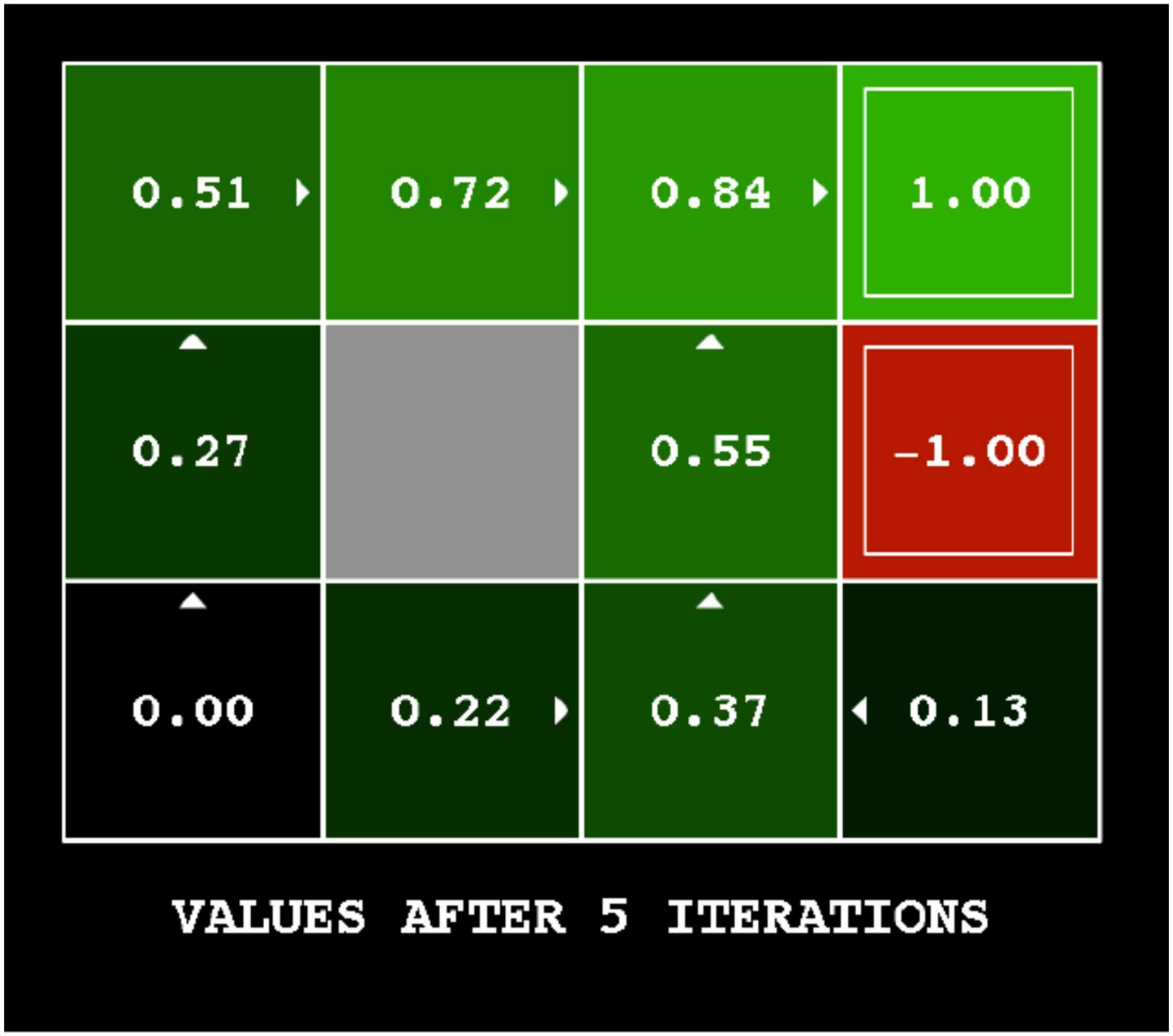
Value Iteration in GridWorld ($\gamma = 0.9$)



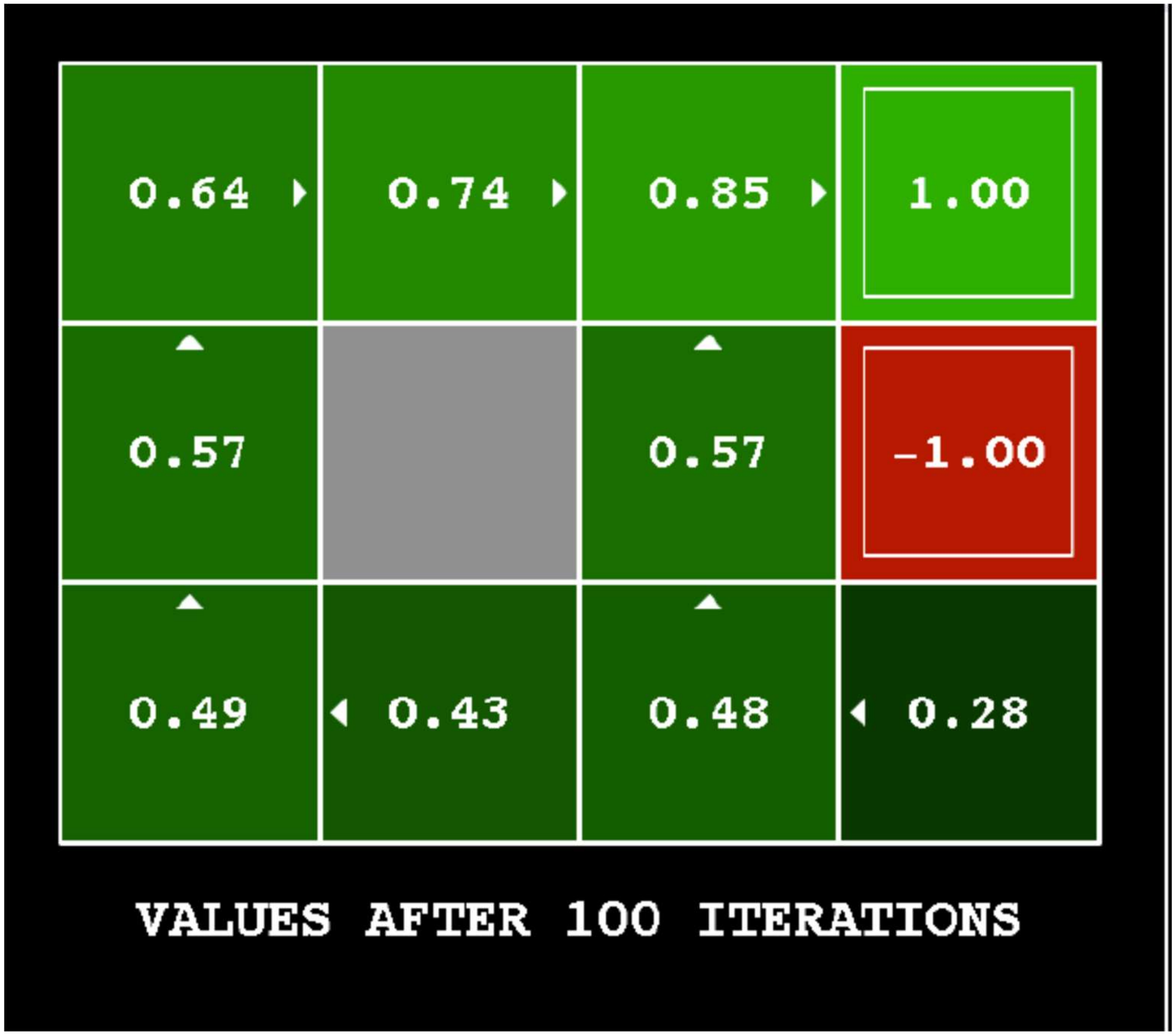
Value Iteration in GridWorld ($\gamma = 0.9$)



Value Iteration in GridWorld ($\gamma = 0.9$)



Value Iteration in GridWorld ($\gamma = 0.9$)



Q-Learning

Full reinforcement learning

- You don't know the **transitions** $P(s, a, s')$
- You don't know the **rewards** $R(s, a, s')$
- You can choose any actions you like
- **Goal:** learn the optimal policy / values
    - Learn the MDP first, then use value/policy iteration (requires learning the MDP: transition and reward functions)
    - Learn only the values (don't learn the MDP or explicitly model it)

- Learner makes choices: **exploration vs. exploitation**
- This is **not** offline planning; you **take actions in the world** and find out what happens!

Value iteration: find successive approximate optimal values

$$V_{i+1}(s) = \max_a \sum_{s'} P(s, a, s')[R(s') + \gamma V_i(s')]$$

Q-values are more useful!

$$Q_{i+1}(s, a) = \sum_{s'} P(s, a, s')\left[R(s) + \gamma \max_{a'} Q_i(s', a')\right]$$

Adapted from slides by Dan Klein

Q-Learning

How should we pick an action to take based on Q?

- Shouldn't always be greedy
    - *we won't explore much of the state space this way*
- Shouldn't always be random
    - *will take a long time to generate a good Q*
- **ϵ-greedy strategy**: with some small probability choose a random action (exploration), otherwise select the greedy action (exploitation)

Initialize:

- Choose an initial state-value function $Q(s,a)$
- Let s be the initial state of the environment

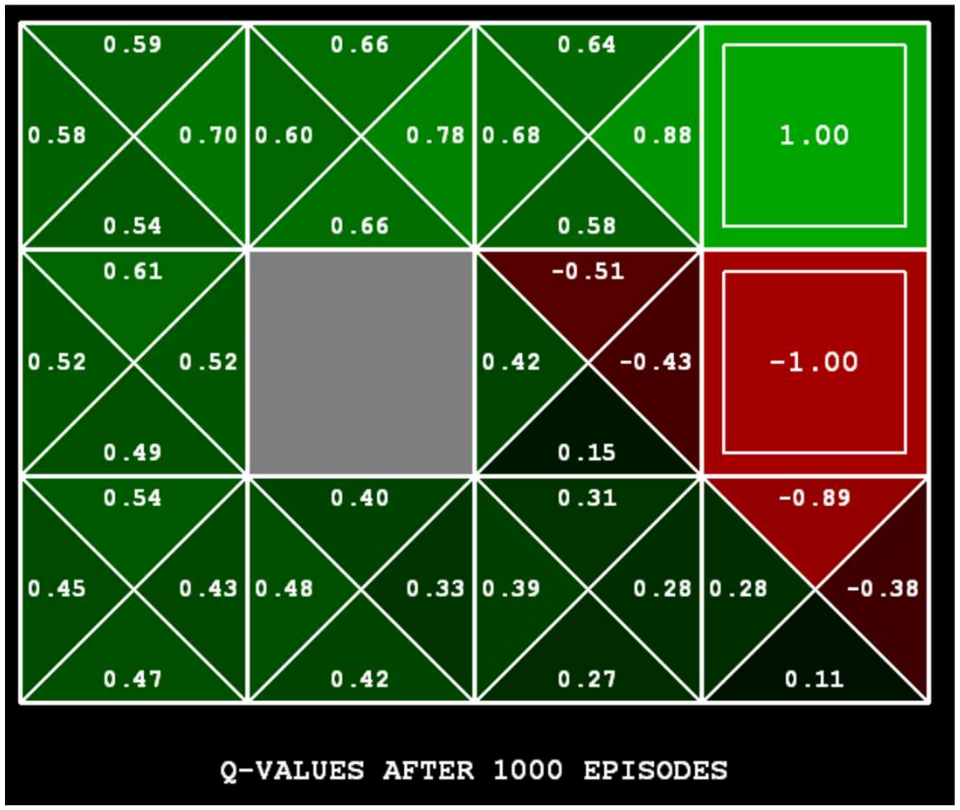
Repeat until convergence:

- Choose an action a_t for the current state s_t based on Q
- Take action a_t and observe the reward r_t and the new state s_{t+1}
- Update Q

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Adapted from slides by Nicholas Ruozzi

Q-Learning



Q-learning produces tables of q-values

Initialize:
- Choose an initial state-value function $Q(s,a)$
- Let s be the initial state of the environment

Repeat until convergence:
- Choose an action a_t for the current state s_t based on Q
- Take action a_t and observe the reward r_t and the new state s_{t+1}
- Update Q

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Adapted from slides by Dan Klein

Learning Rate (α) and Discount Factor (γ)

Explore vs exploit:
- learning rate (α) determines to what extent newly acquired information overrides old information
- $\alpha = 0$ makes the agent learn nothing (exclusively exploiting prior knowledge)
- $\alpha = 1$ makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities).
- in practice, often a constant learning rate is used

Discount factor:
- discount factor (γ) determines the importance of future rewards
- $\gamma = 0$ will make the agent "myopic" (or short-sighted) by only considering current rewards
- $\gamma = 1$ will make the agent strive for a long-term high reward
- if $\gamma > 1$, action values may diverge