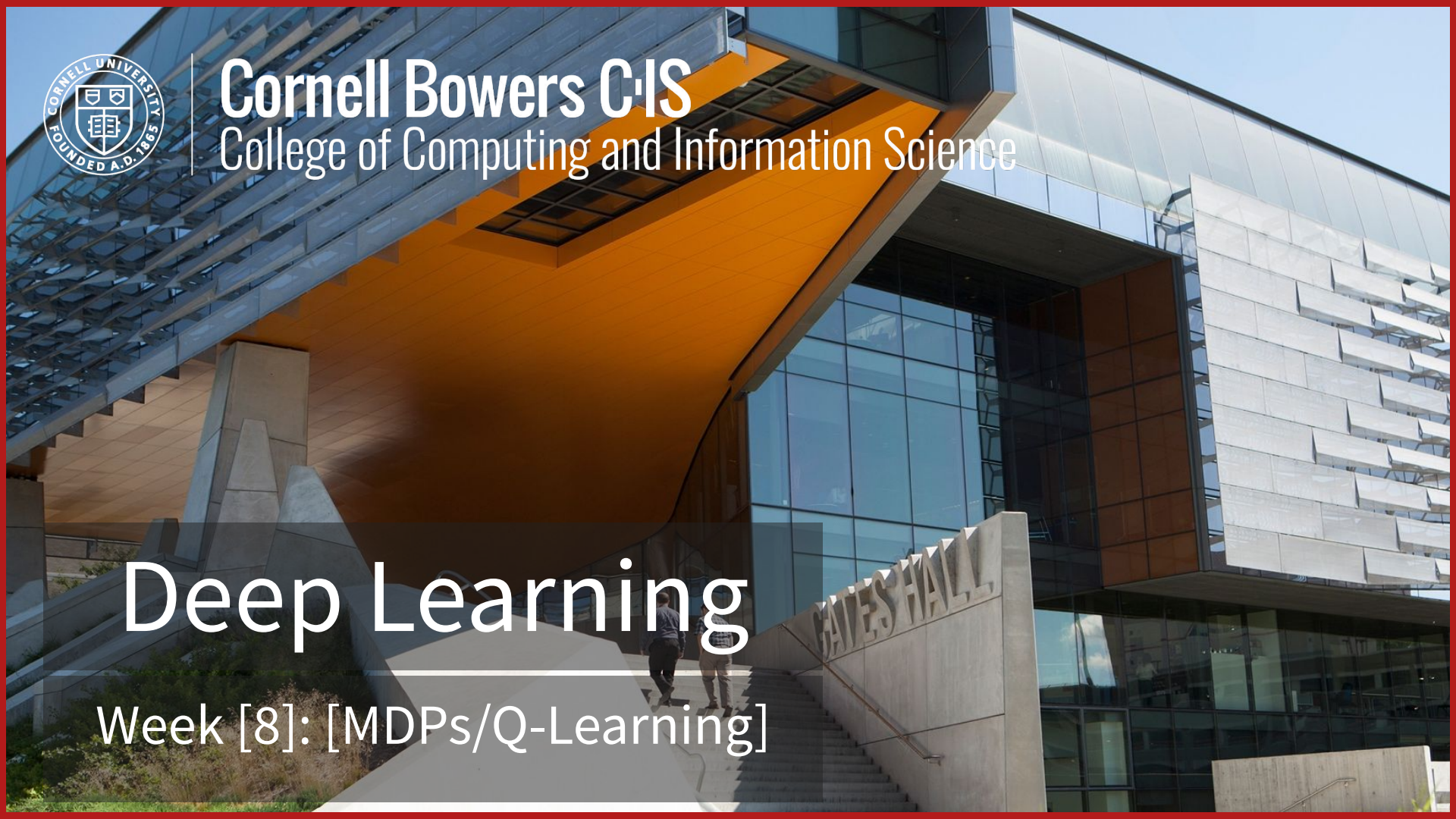


Deep Learning

Week [8]: [MDPs/Q-Learning]

Overview

- Reinforcement Learning
- Markov Decision Processes
- Policy
- Value
- Action-Value

Recap: Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Cat

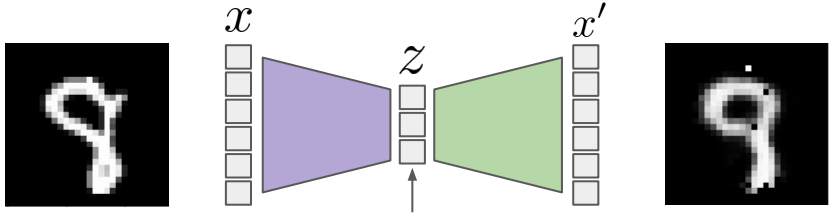
Classification

Recap: Unsupervised Learning

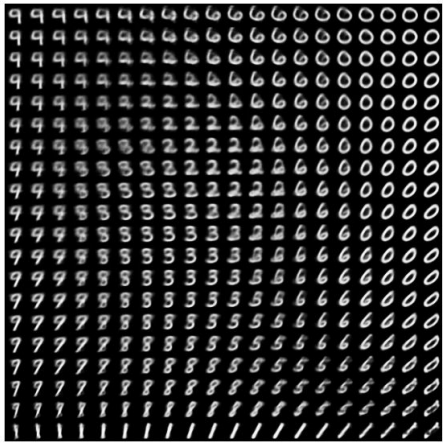
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

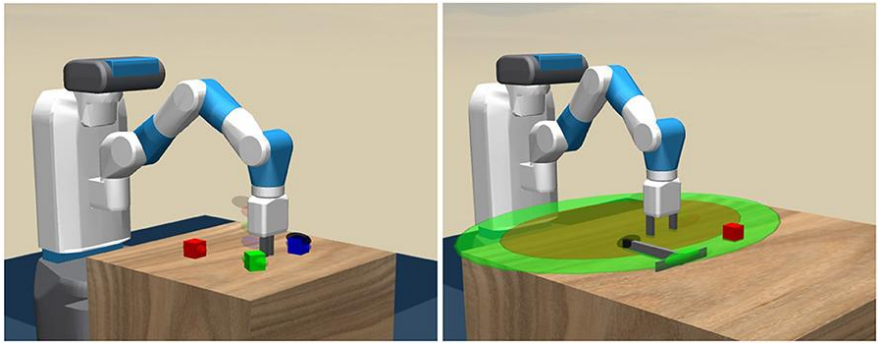


latent variable



Limitations of Supervised Learning

Can a regression/classification algorithm learn to perform these tasks successfully?

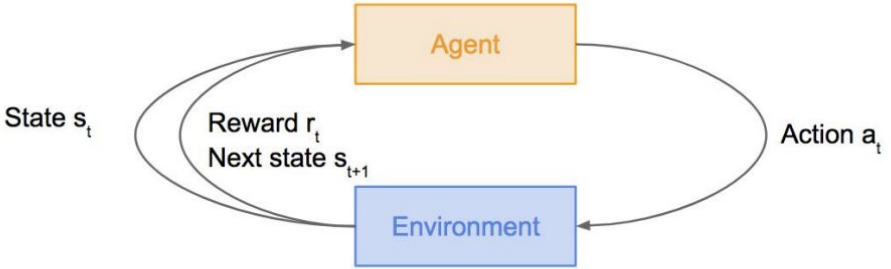


Robot learning to pick up blocks

Waymo self-driving car

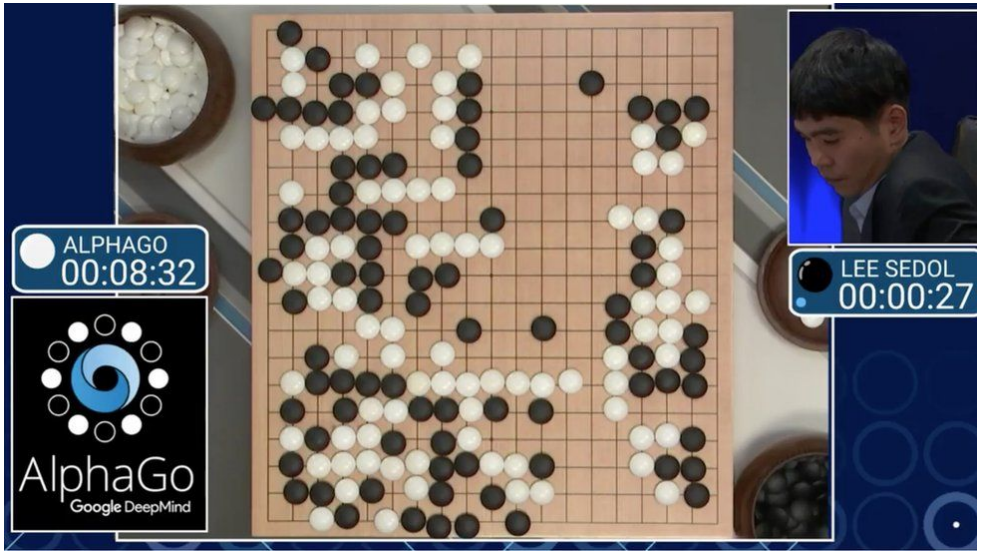
Reinforcement Learning

Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

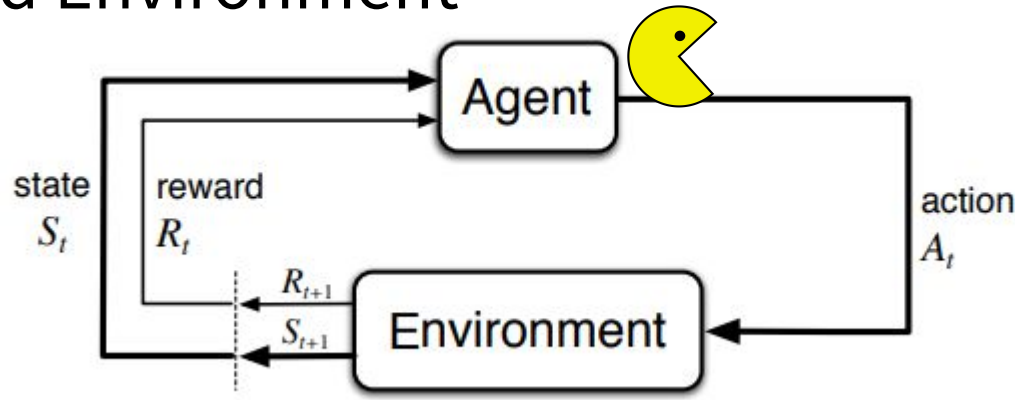


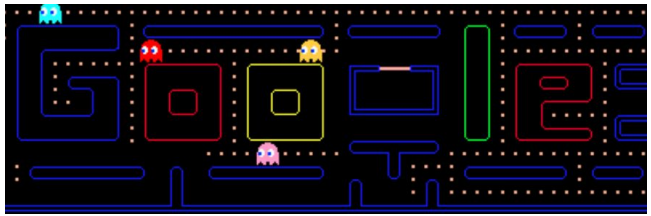
Goal: Learn how to take actions in order to maximize reward

Using Reinforcement Learning to play games



Agent and Environment

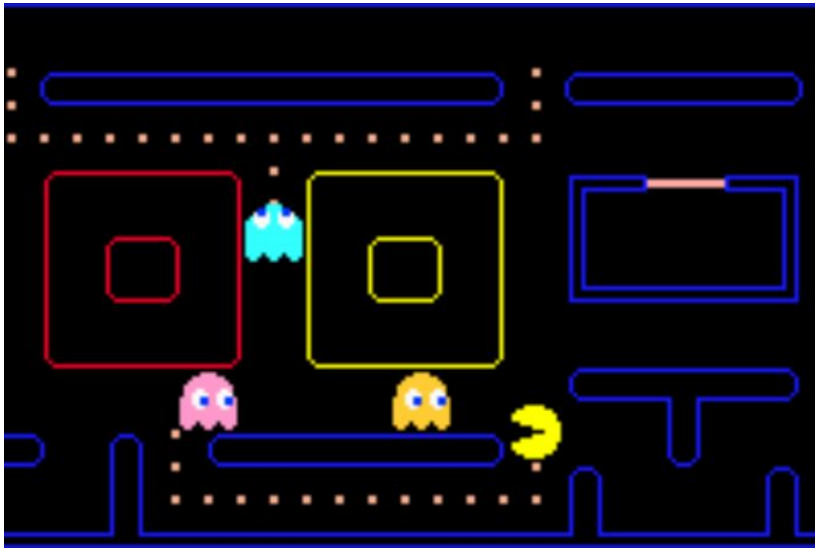




Agent:	Environment:
- Perceives environment. - Makes decisions. - Aims to maximize reward.	- The external context in which an agent operates and interacts with - Provides feedback to agent

Discussion: Rewards

What are ways to measure rewards in these games?



Supervised Learning

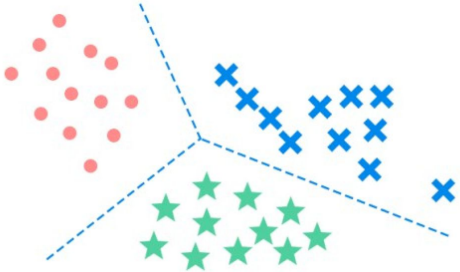
- Learns from a labeled dataset
- Classification
  Regression

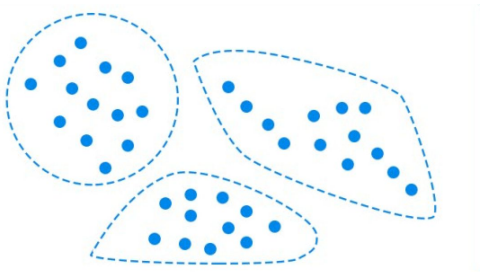
Unsupervised Learning

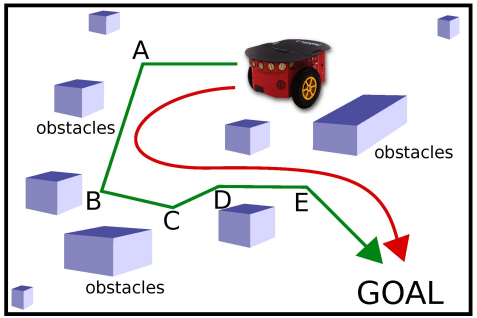
- Find patterns in unlabeled data
- Clustering
  Generative Models

Reinforcement Learning

- Agent interacts with an environment and learns to maximize a reward
- Game Playing
  Robot Navigation

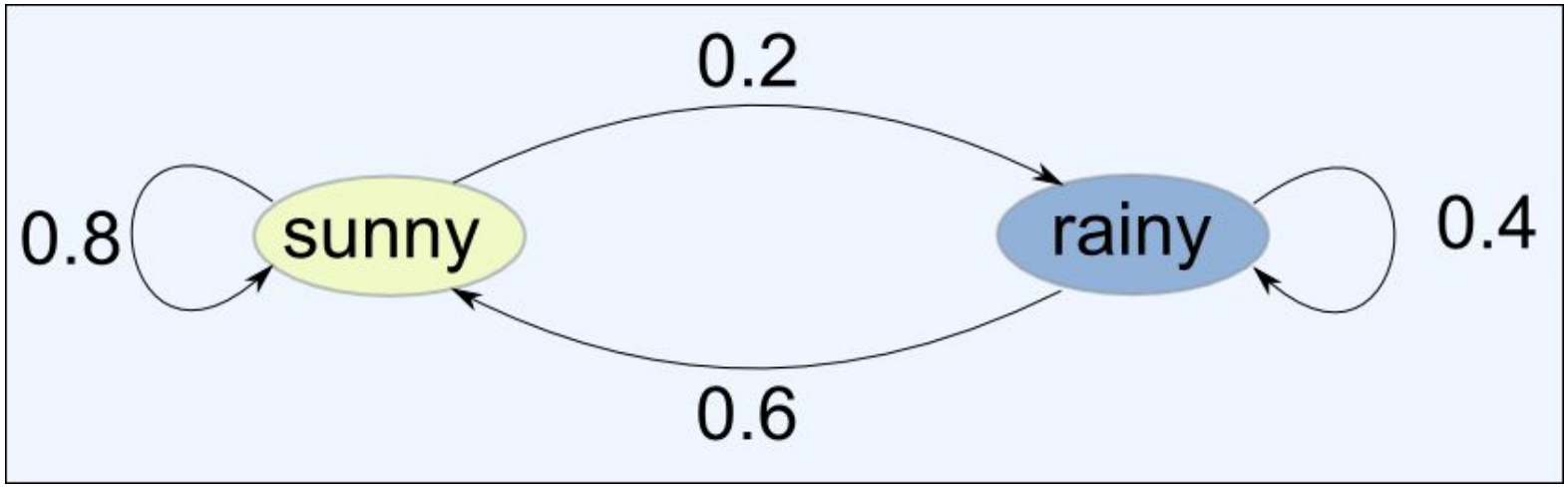






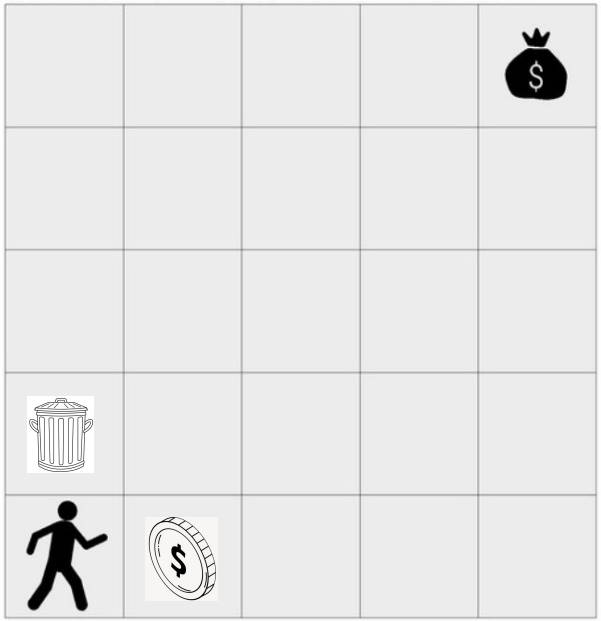
Markov Process

Probabilistic events in which state at time $t+1$ solely depends on state at time t



Markov Decision Process (MDP)

Framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker



$\mathcal{S}$:

$\mathcal{A}$:

$\mathcal{P}$:

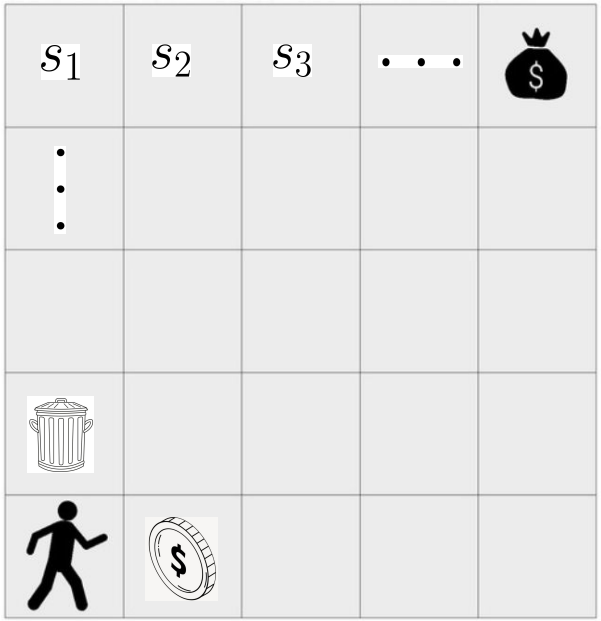
$\mathcal{R}$:

γ:

State Space

$$S = \{s_1, s_2, s_3, \ldots\}$$

– $\mathcal{S}$: Set of states representing the environment.



$\mathcal{S}$:

$\mathcal{A}$:

$\mathcal{P}$:

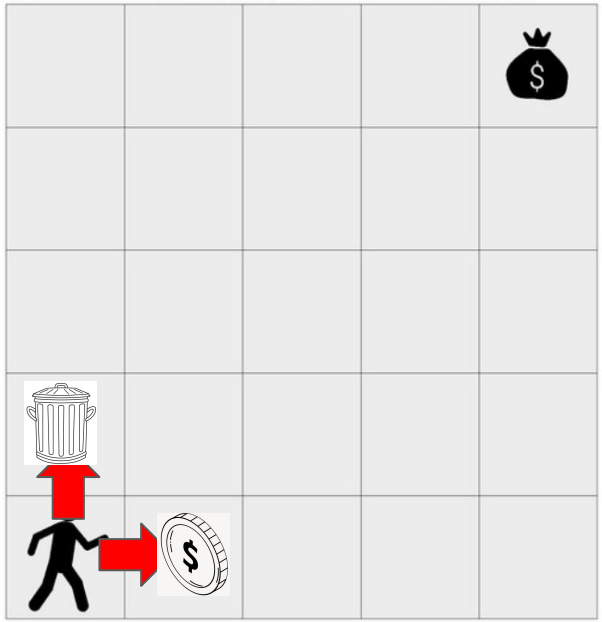
$\mathcal{R}$:

γ:

Action Space

$$\mathcal{A} = \{\text{Up}, \text{Down}, \text{Left}, \text{Right}\}$$

– $\mathcal{A}$: Set of actions the agent can take.



$\mathcal{S}$: State

$\mathcal{A}$: Action

$\mathcal{P}$:

$\mathcal{R}$:

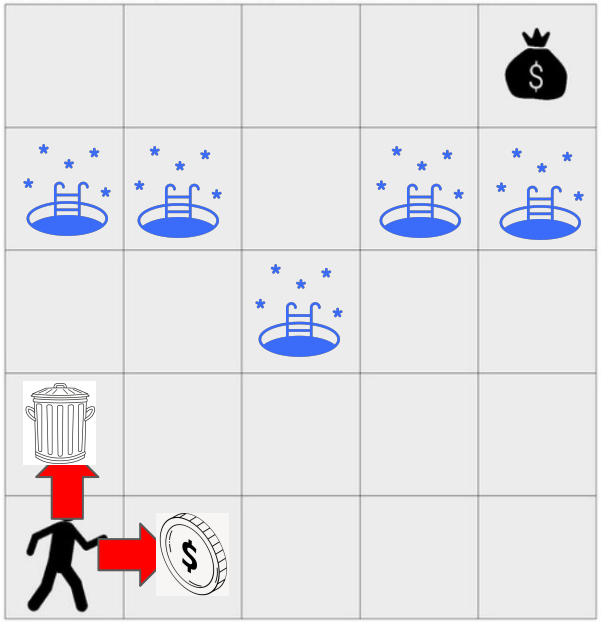
γ:

Transition Function

– $\mathcal{P}$: Transition probability function, $\mathcal{P}(s'|s,a)$.

The agent's actions do not always go as planned:

- E.g. ice sends agent 80% of the time to their intended direction, 20% to a random direction



$\mathcal{S}$: State

$\mathcal{A}$: Action

$\mathcal{P}$: Transition

$\mathcal{R}$:

γ:

Transition Function

– $\mathcal{P}$: Transition probability function, $\mathcal{P}(s'|s,a)$.

- For Markov decision processes, “Markov” means:

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

$$=$$

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

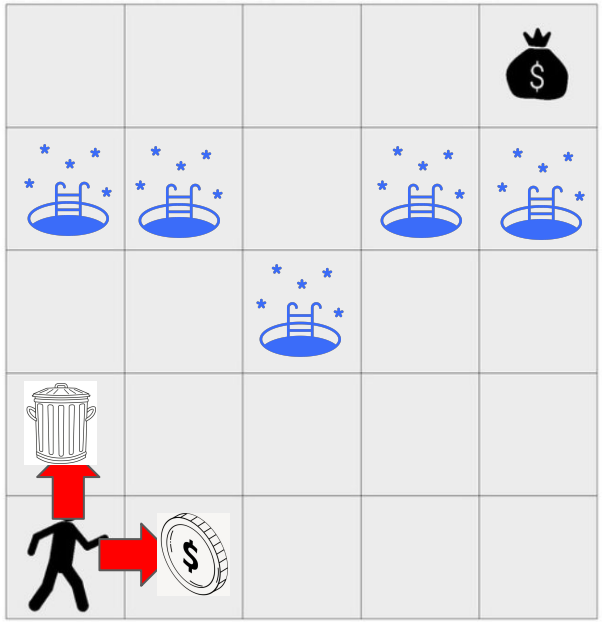
Reward Function

action

starting

arrival

– $\mathcal{R}$: Reward function, $\mathcal{R}(s, a, s')$



$\mathcal{S}$: State

$\mathcal{A}$: Action

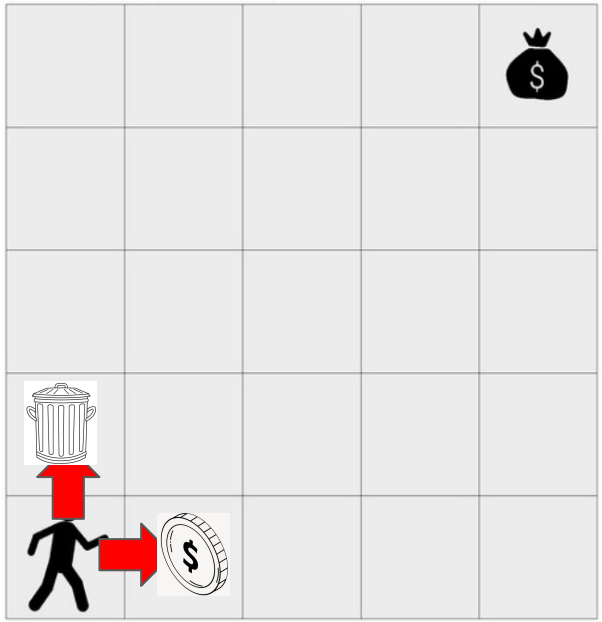
$\mathcal{P}$: Transition

$\mathcal{R}$: Reward

γ:

Discount Factor

– $\gamma \in [0, 1]$: Discount factor that balances immediate and future rewards.



$\mathcal{S}$: State

$\mathcal{A}$: Action

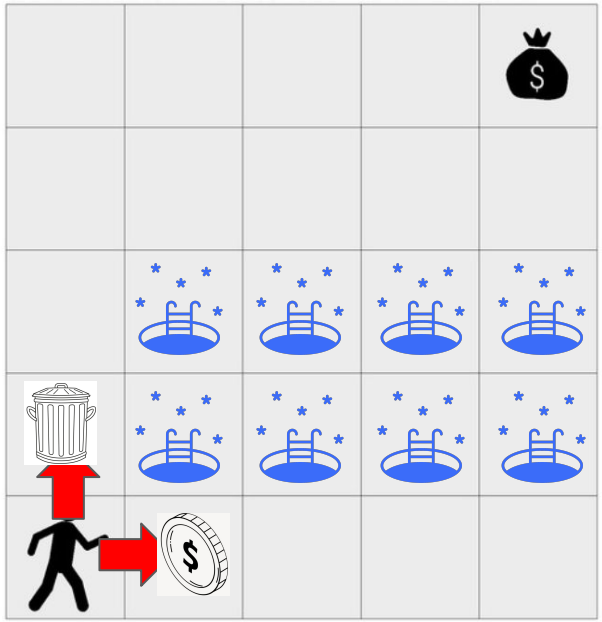
$\mathcal{P}$: Transition

$\mathcal{R}$: Reward

γ: Discount Factor

Discount Factor

– $\gamma \in [0, 1]$: Discount factor that balances immediate and future rewards.



$\mathcal{S}$: State

$\mathcal{A}$: Action

$\mathcal{P}$: Transition

$\mathcal{R}$: Reward

γ: Discount Factor

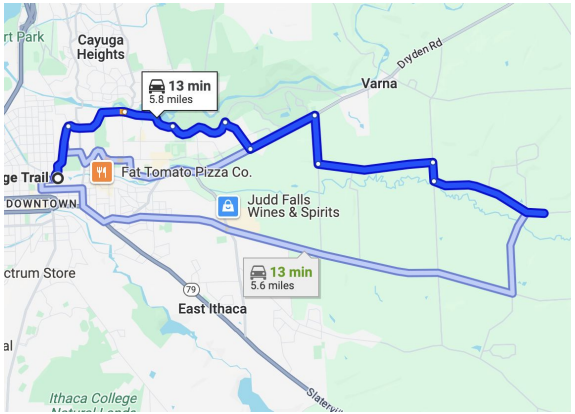
Markov Decision Process (MDP)

- MDPs provide a framework for modeling sequential decision-making problems.
- An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:
  - $\mathcal{S}$: Set of states representing the environment.
  - $\mathcal{A}$: Set of actions the agent can take.
  - $\mathcal{P}$: Transition probability function, $\mathcal{P}(s'|s, a)$.
  - $\mathcal{R}$: Reward function, $\mathcal{R}(s, a, s')$
  - γ: Discount factor, $\gamma \in [0, 1]$.

Discussion: Identify the **state**, **action, transition** and **reward** in each.

Tetris

Chess



Navigation

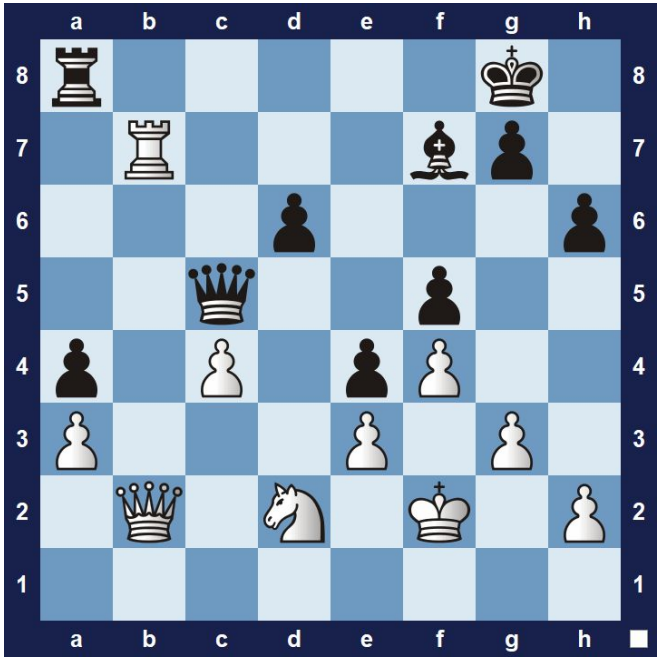
Markov Decision Process (MDP)

An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

Framework for modeling agent in environment

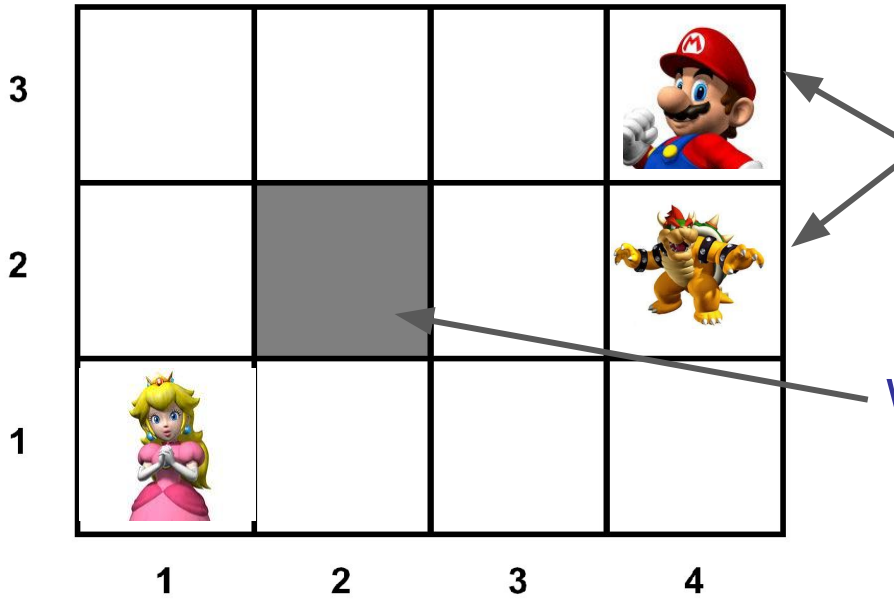
We still need to find agent **policy** to maximize reward!!!

E.g. which piece to move to max chance of winning



Simplified Setup

Princess Peach gets reward +1 for Mario, but -1 for Bowser



Small (negative) “reward” each step

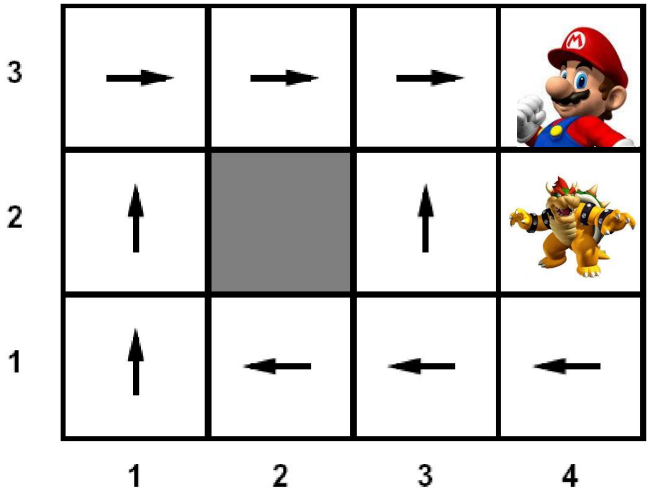
terminal states

Walls block path

Policy

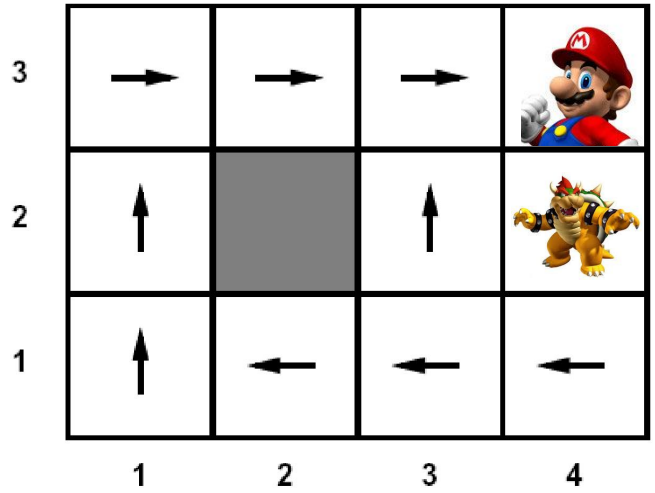
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected cumulative reward if followed

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



Policy

- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes **expected cumulative reward** if followed

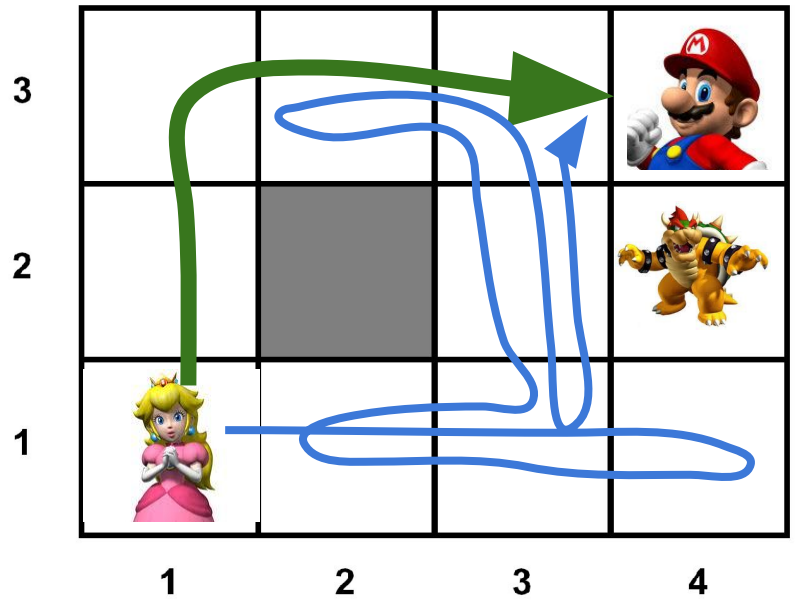


$$\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s'_t)$$

Discount factor

Reward given state, action at t

Recall: Discounting



- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher value than later rewards

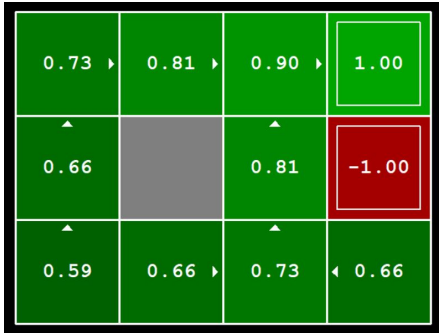
Running Recap

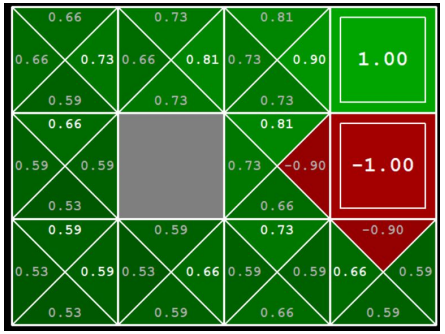
- Policy = Choice of action for each state
- Value = ???
- Q (Action-Value) = ???

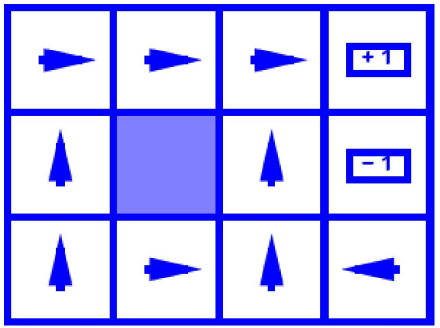
Cornell Bowers CIS

Optimal Values

- Why? Optimal values define optimal policies!
- Define the value of a state s:
  $V^*(s)$ = expected reward starting in s and acting optimally
- Define the Q-value of (s,a):
  $Q^*(s,a)$ = expected reward starting in s, taking action a and thereafter acting optimally
- Define the optimal policy:
  $\pi^*(s)$ = optimal action from state s







Bellman Equations

- Definition of "optimal values" leads to a simple one-step lookahead relationship

  Optimal rewards = maximize over first action and then follow optimal policy

- Formally:

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

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

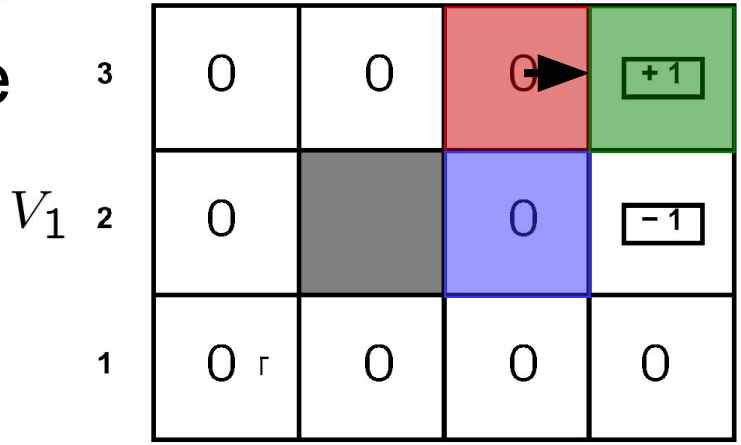
$$V^*(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\Big[R(s,a,s') + \gamma\, V^*(s')\Big]$$

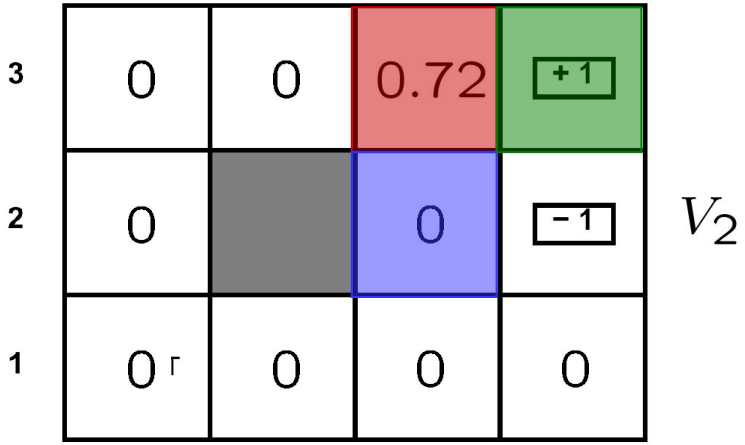
Value Iteration

- Idea:
  - Start with $V_0^*(s) = 0$
  - Given V_i^*, calculate the values for all states at next step:
  - This is called a value update or Bellman update
  - Repeat until convergence

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Example





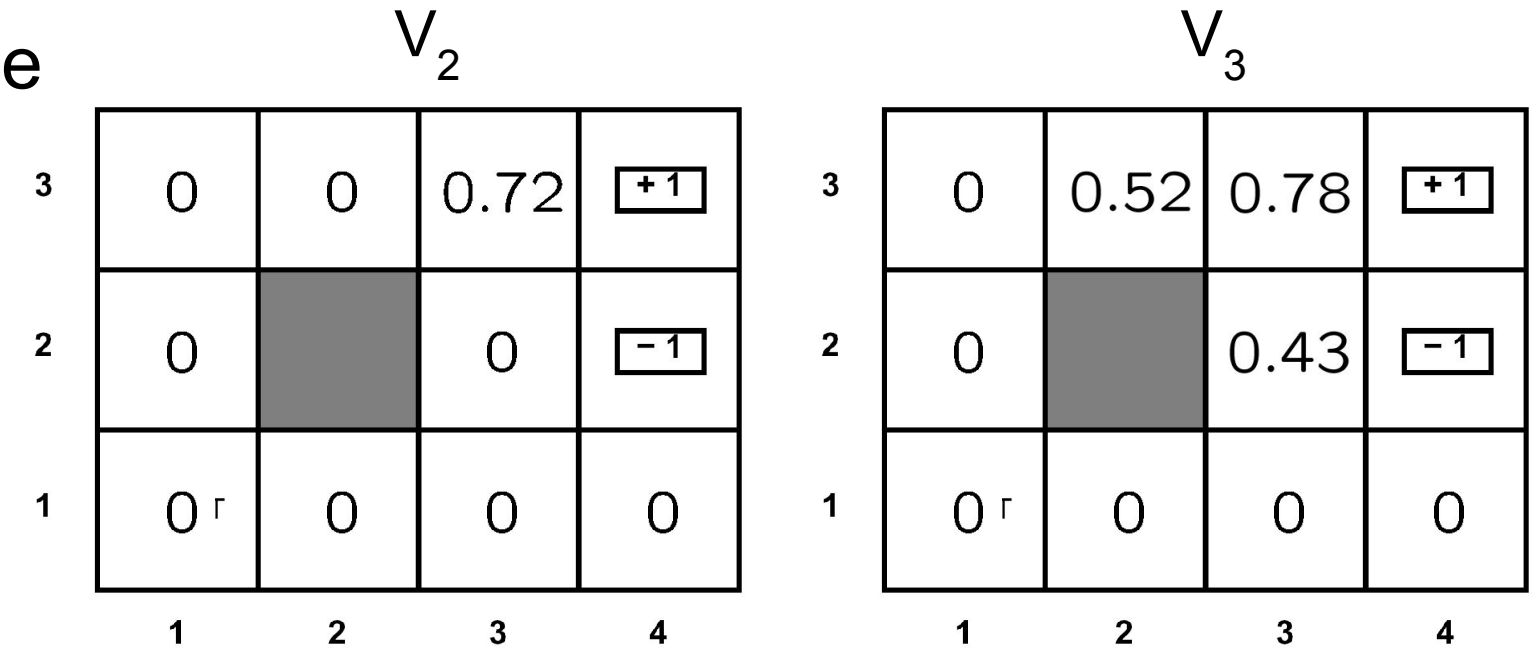
$$V_{i+1}(s) = \max_a \sum_{s'} \mathcal{P}(s'|s,a)\left[R(s,a,s') + \gamma\, V_i(s')\right]$$

$$V_2(\langle 3,3\rangle) = \sum_{s'} T(\langle 3,3\rangle, \text{right}, s')\left[R(\langle 3,3\rangle) + 0.9\, V_1(s')\right]$$

max happens for a=right, other actions not shown

$$= 0.9\,[0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$

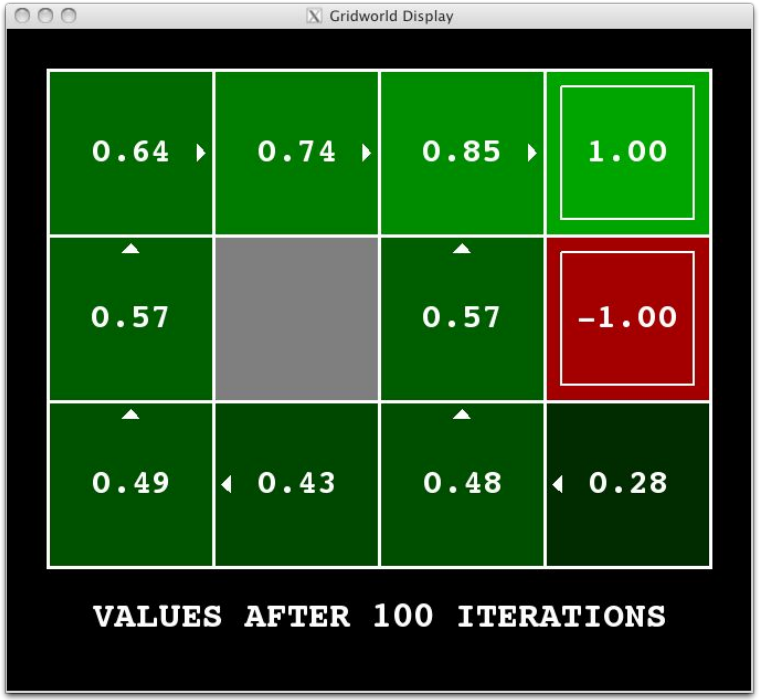
Example



- Information propagates outward from terminal states and eventually all states have correct value estimates

Gridworld


Gridworld Display
0.00
0.00
0.72
1.00
0.00
0.00
-1.00
0.00
0.00
0.00
0.00
VALUES AFTER 2 ITERATIONS


Gridworld Display
0.64
0.74
0.85
1.00
0.57
0.57
-1.00
0.49
0.43
0.48
0.28
VALUES AFTER 100 ITERATIONS

Computing Policy

- Which action should we chose from state s:
  - Given optimal values V?

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

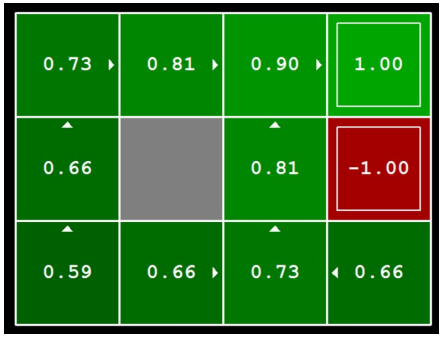
  - Given optimal q-values Q?

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

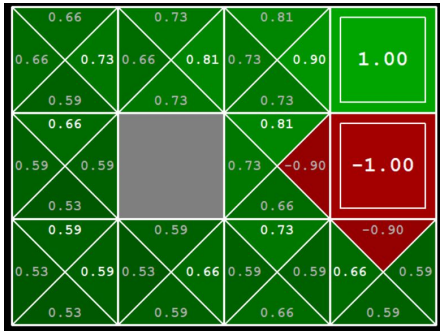
  - Lesson: actions are easier to select from Q's!

Cornell Bowers CIS

Recall: Optimal Values

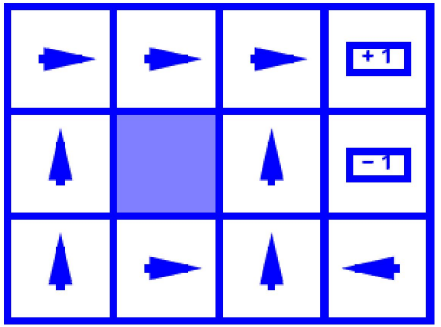


- Why? Optimal values define optimal policies!
- Define the value of a state s:
  $V^*(s)$ = expected reward starting in s and acting optimally



How about a fixed policy?

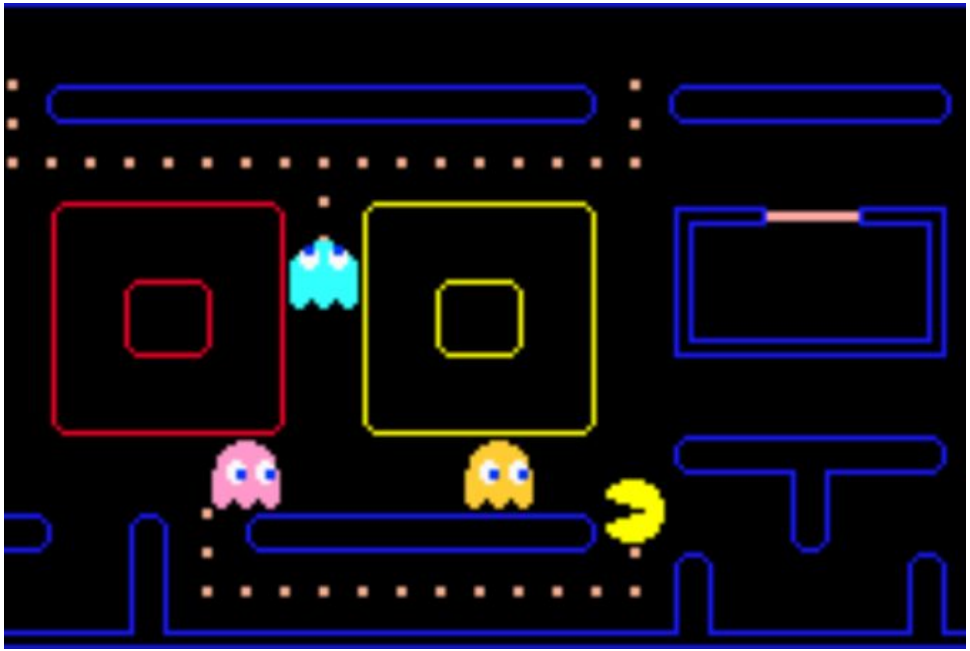
$V^{\pi}(s)$ = expected reward starting in s and following π



Calculate the size of the State Space for Pacman in the following environment

Assume there are 100 positions, 4 ghosts, and pacman

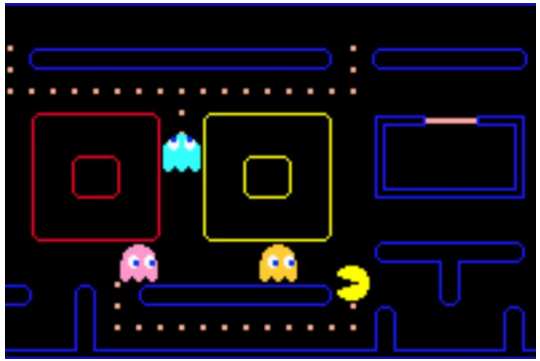
(Can ignore pellets for simplicity)



Calculate the size of the State Space for Pacman in the following environment

4 ghosts and Pacman

$$100 \cdot 100 \cdot 100 \cdot 100 \cdot 100 = 10^{10}$$



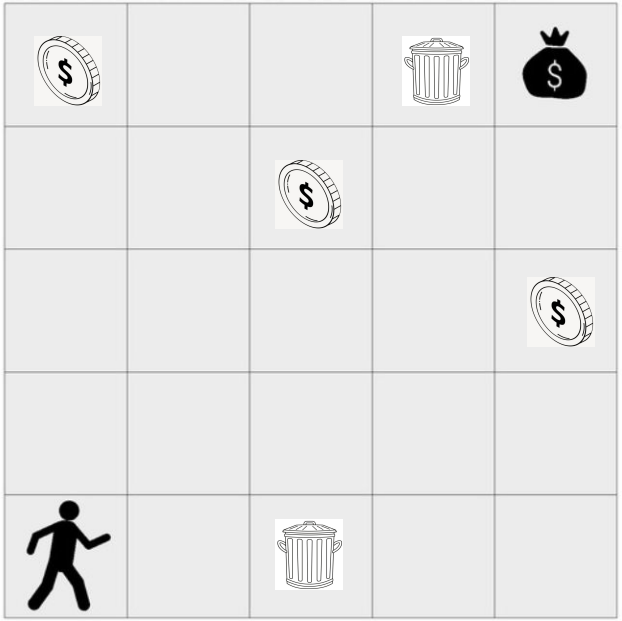
What if we consider pellets? Assuming there are 100 pellets:

$$2^{100} \cdot 10^{10} > 10^{40}$$

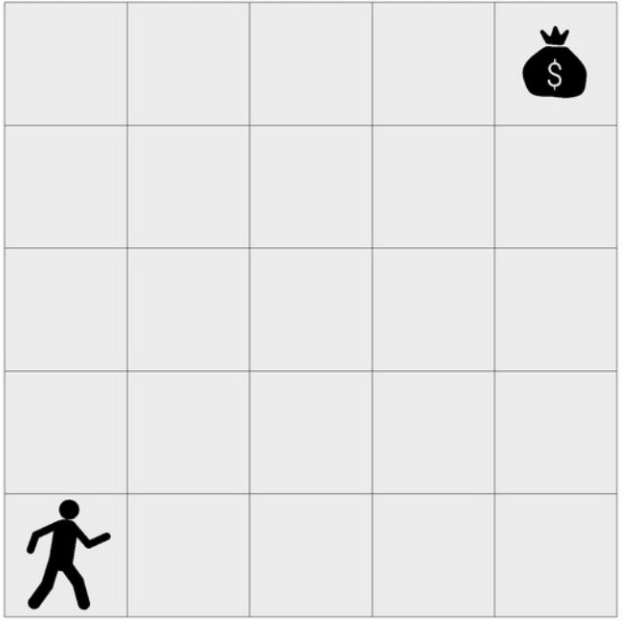
What are the issues?

- As the state/action space increases, the likelihood of reaching a specific state and action decreases
- Unknown environment dynamics
- Sparse Rewards

Dense Rewards



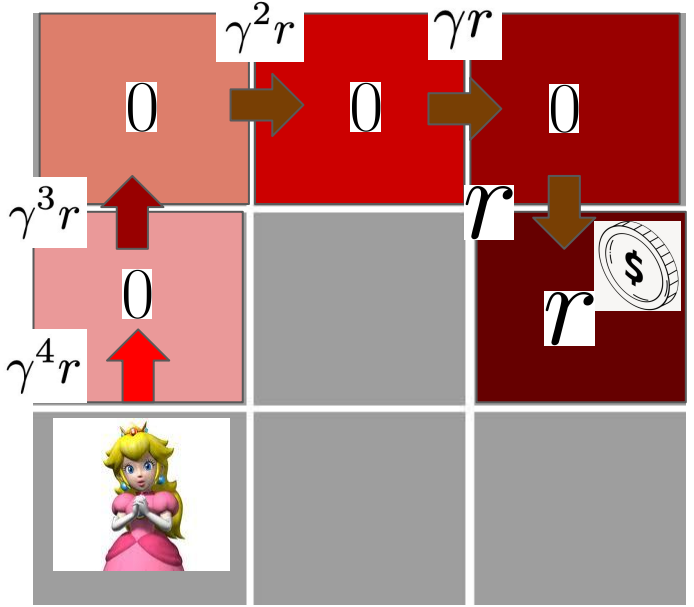
Sparse Rewards



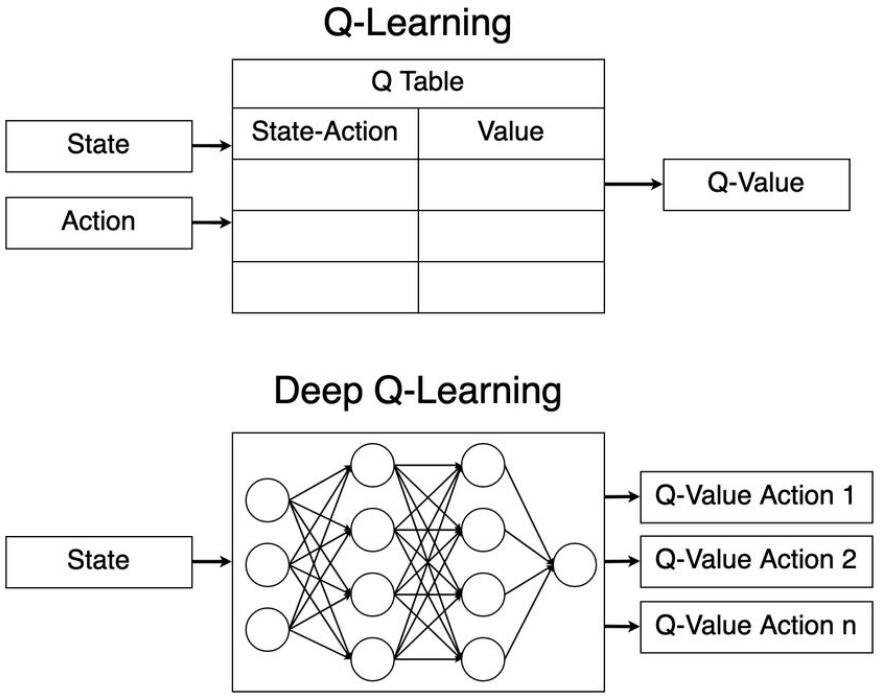
Exploration-Exploitation TradeOff

No Exploration:

- Begin by making random moves
- Find some way to obtain reward
- Stick with that solution



Next Time: How can we use deep learning to improve this?



Recap

- Markov Decision Processes (MDPs)
  - Framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker
- Value function
  - Expected cumulative reward from state s following some policy
- Action-value (Q) function
  - Expected cumulative reward if you take action a in state s and follow some policy