# CS 6501: Deep Learning for Computer Graphics

# Deep Reinforcement Learning

Connelly Barnes

Slides based on those by David Silver

# Outline

- **Introduction to Reinforcement Learning**
- Value-based deep reinforcement learning
  - Q-learning

# Introduction to Reinforcement Learning

- "How software agents ought to take **actions** in an **environment** so as to maximize some notion of cumulative **reward**." – Wikipedia
- Studied in various disciplines:
  - Game theory
  - Control theory
  - Operations research
  - Swarm intelligence
  - Statistics
  - Economics
  - …

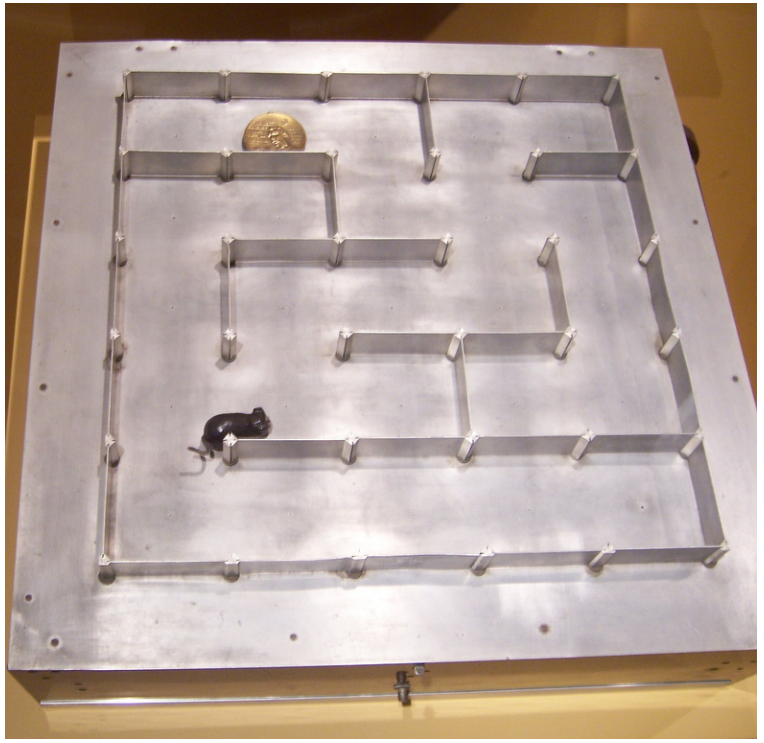
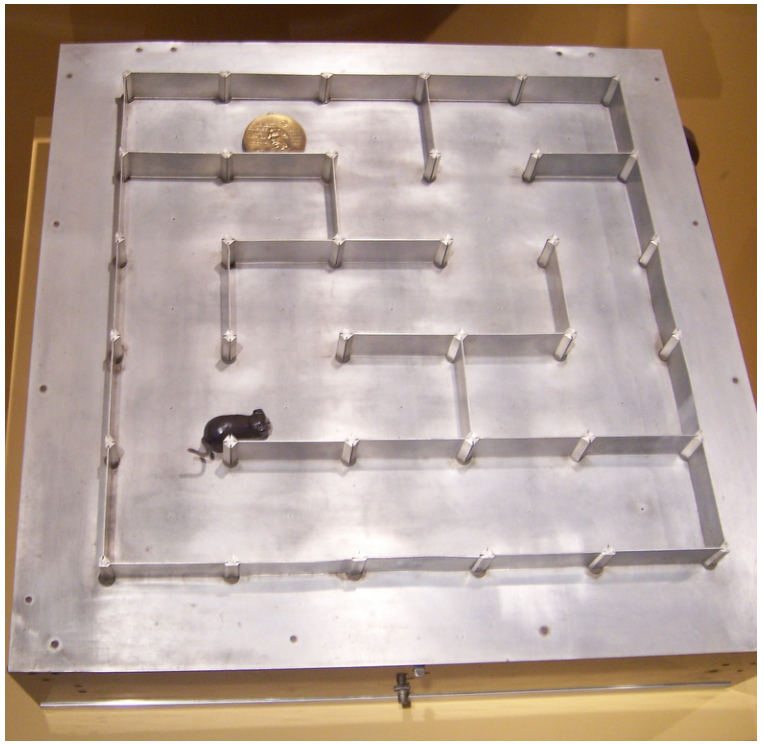


Photo from Flickr

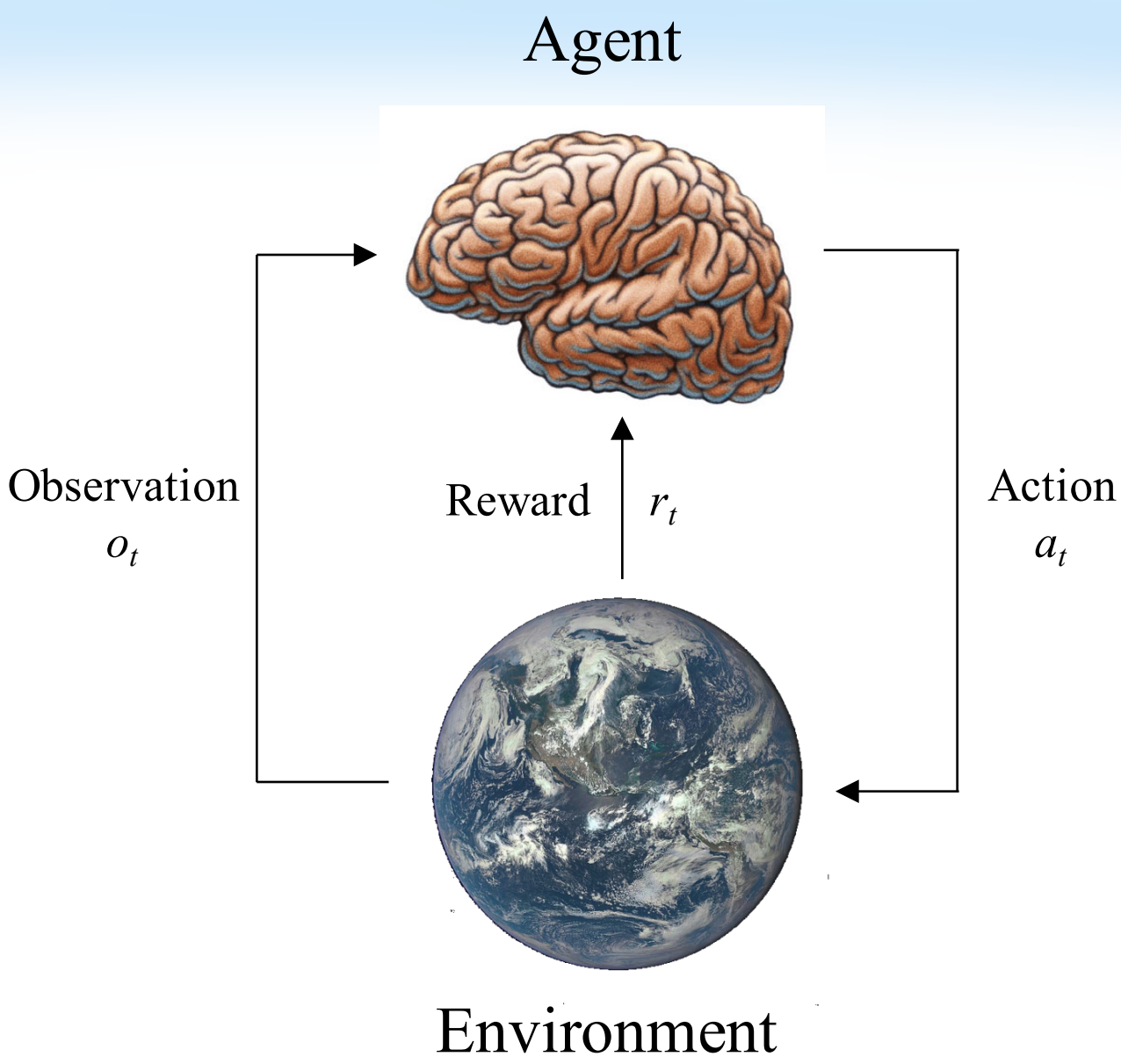
# Introduction to Reinforcement Learning

- Goal: determine the best action for every state.
- Optimize an objective function, such as average reward per time unit, or discounted reward.
- Few states: dynamic programming.
- Many states: reinforcement learning, deep reinforcement learning.



# Terminology

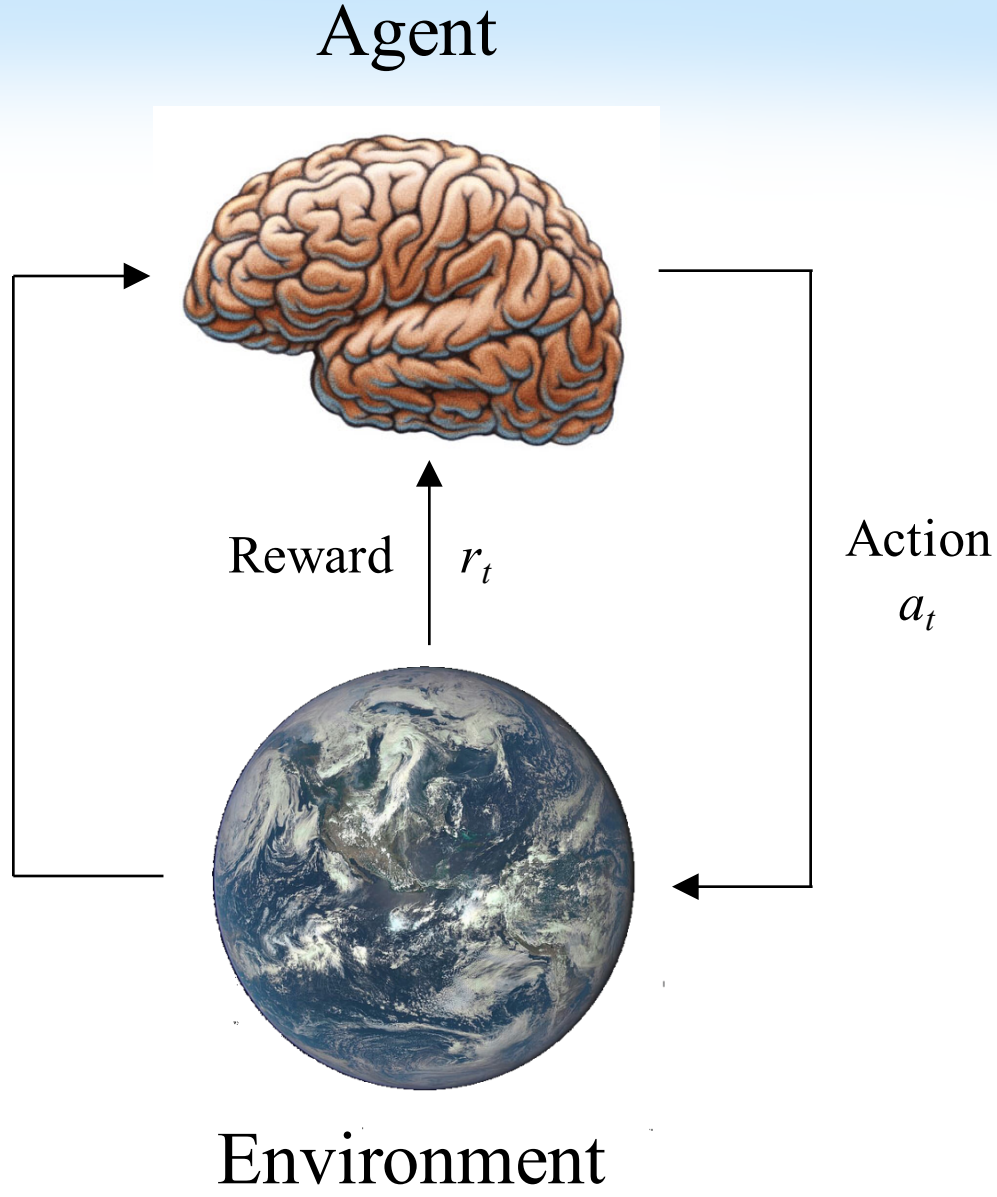
- At each time step *t* the agent:
  - Receives **observation** $o_t$
  - Receives **scalar reward** $r_t$
  - Executes **action** $a_t$
- The environment:
  - Receives action $a_t$
  - Determines observation $o_{t+1}$
  - Determines scalar reward $r_{t+1}$




Slides based on those by David Silver, images [1], [2]

# Modeling Agent Behavior

- Agent can include these components:
  - **Policy**: agent's behavior function
  - **Value function**: how good is each state/action?
  - **Model**: agent's understanding of the environment
- Which one did we work with on the Tic Tac Toe assignment?




Slides based on those by David Silver, images [1], [2]

# Policy

- Determines agent's behavior directly
- Maps from state to action
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathrm{P}[a|s]$

# Value Function

- **Value function** is discounted prediction of future reward
- Q-value function gives expected total reward
  - From state *s* and action *a*
  - Under policy $\pi$
  - With discount factor $\gamma$
  - $Q^{\pi}(s,a) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s,a]$

Slides based on those by David Silver

# Bellman's Principle of Optimality

- "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." – Bellman 1957
- i.e. dynamic programming
- Q-value function:

$$Q^{\pi}(s,a) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s,a]$$

- Bellman equation:

$$Q^{\pi}(s,a) = E_{s',a'}\left[r_{t+1} + \gamma Q^{\pi}(s',a') | s,a\right]$$

Slides based on those by David Silver

# Optimal Value Function

- The maximum achievable value:

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

- Once we have Q* we can act optimally:

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

Slides based on those by David Silver

# Reinforcement Learning Approaches

- **Value-based**
  - Estimate optimal value function $Q^*(s, a)$
- **Policy-based**
  - Estimate the optimal policy $\pi^*$ directly
- **Model-based**
  - Build model of environment, plan future actions using model

Slides based on those by David Silver

# Outline

- Introduction to Reinforcement Learning
- **Value-based deep reinforcement learning**
  - Q-learning

# Q-Learning

- Optimal Q values should follow Bellman equation:

$$Q^{\pi}(s,a) = E_{s',a'}\left[r_{t+1} + \gamma Q^{\pi}(s',a')|s,a\right]$$

- Given an initial Q, iteratively estimate a better one:

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a\,) - Q(s_t,a_t)\right]$$

Reward

Estimate of future optimal value

Old value

Learning rate

Discount factor

Slides based on those by David Silver

# Q-learning Details

- Initial conditions
  - Often high values to encourage exploration
  - No matter what action is selected, update rule will cause it to have lower value than alternatives, so alternatives will be explored also.
- Learning rate
  - Often a constant such as 0.1

# Q-Learning and Neural Networks

- If there is a large number of (state, action) pairs, cannot store $Q(s, a)$ directly.
- Instead, approximate $Q(s, a)$ using a neural network:
  - **Q-function**.

# Q-Learning and Neural Networks

- Supervised learning: can take the value from the right-hand side of the Q-learning rule as the target for a neural network with weights **w.**
- Minimize:

$$E = [r_{t+1} + \gamma \max_{a'} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})]^2$$

- **Diverges** using neural networks due to:
  - Correlations between samples
  - Non-stationary learning target

Slides based on those by David Silver

# Deep Q-Networks (DQN): Experience Replay

- To remove correlations, build a history data-set from the agent's own experiences: $(s_i, a_i, r_{i+1}, s_{i+1})$
- Sample experiences from this data-set and apply the stochastic gradient descent update for the squared loss:

$$E = [r_{t+1} + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w})]^2$$

- To prevent problems with non-stationary learning targets, fix the weights $\mathbf{w}^-$ while performing SGD.
- Discussion question: concretely, how might we use DQN to play Tic Tac Toe? What are the states, actions, rewards? What network architecture?

Slides based on those by David Silver

# Deep Q-Networks (DQN)

- For more details, see the tutorial by David Silver

# Software Packages

- https://github.com/VinF/deer
- https://github.com/rlpy/rlpy