

# INTRODUCING REINFORCEMENT LEARNING

Scott O'Hara

Metrowest Developers Machine Learning Group

01/29/2020

# REFERENCES

The material for this talk is primarily drawn from the **Matiisen** article and the slides, notes and lectures of the courses below:

**“Demystifying Deep Reinforcement Learning,” 2015, Tamber Matiisen**

- ▶ I follow approach up through (non-deep) Q-Learning.
- ▶ <https://www.intel.ai/demystifying-deep-reinforcement-learning/>

**University of California, Berkeley CS188:**

- ▶ *CS188 – Introduction to Artificial Intelligence*, Profs. Dan Klein, Pieter Abbeel, et al. <http://ai.berkeley.edu/home.html>

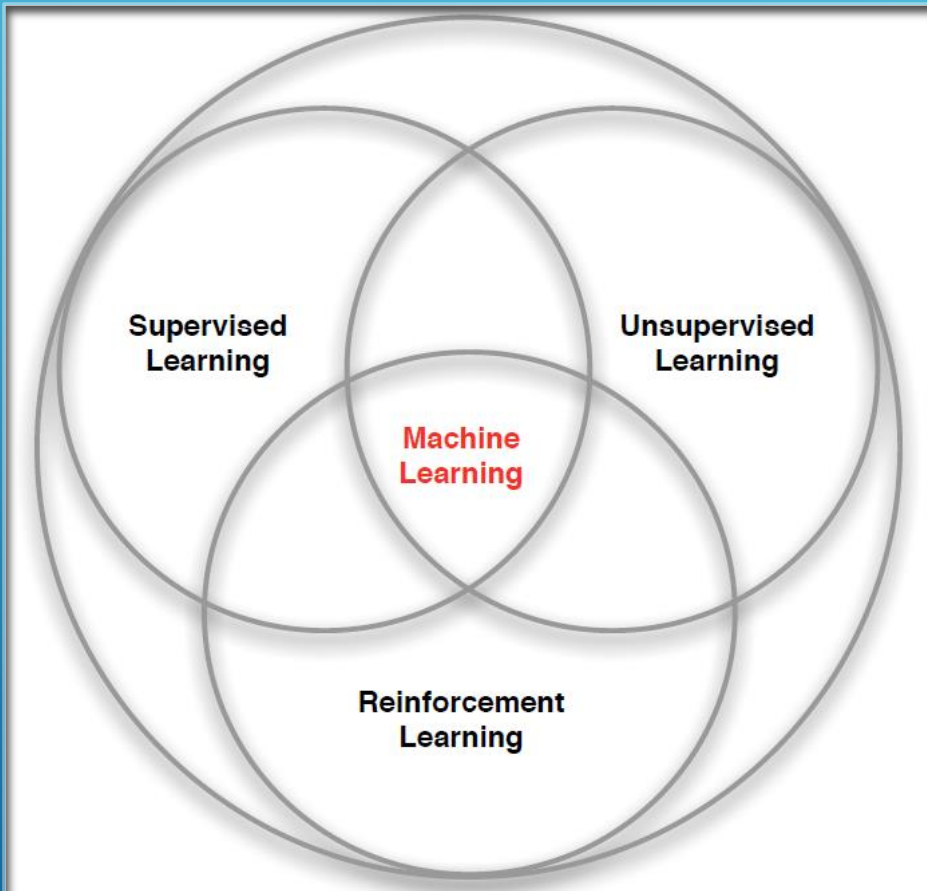
**David Silver, DeepMind:**

- ▶ *Introduction to Reinforcement Learning*  
<http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>

**CS181 course at Harvard University:**

- ▶ *Material used from multiple offerings of the course. Detailed links and references at end of the slide deck.*

# 3 TYPES OF MACHINE LEARNING



**Supervised Learning** – Learn a function from labeled data that maps input attributes to an output label e.g., linear regression, decision trees, SVMs.

**Unsupervised Learning** – Learn patterns in unlabeled data e.g., principle component analysis or clustering algorithms such as K-means, HAC, or Gaussian mixture models.

**Reinforcement Learning** – An agent learns to maximize rewards while acting in an uncertain environment.

# SOME CHARACTERISTICS OF REINFORCEMENT LEARNING

- Learning happens as the agent interact with the world.
- There are no training or test sets. Training is guided by rewards and punishments obtained by acting in an environment.
- The amount of data an agent receives is not fixed. More information is acquired as you go.
- Actions are not always rewarded or punished immediately. “Delayed gratification” is possible.
- Agent actions can affect the subsequent data it receives. E.g., closing a door that can’t be opened again.

# REINFORCEMENT LEARNING

## EXAMPLE: PLAYING ATARI BREAKOUT

Google DeepMind's Deep Q-Learning &  
Superhuman Atari Gameplays

<https://youtu.be/lh8EfvOzBOY>

3 mins 45 secs

# REINFORCEMENT LEARNING

## EXAMPLE: TERRAIN TRAVERSAL

Terrain Traversal with Reinforcement Learning

[https://youtu.be/\\_yjHPu1aYCY](https://youtu.be/_yjHPu1aYCY)

2 mins 38 secs

# REINFORCEMENT LEARNING EXAMPLES:

- Fly stunt maneuvers in a helicopter
- Defeat the world champion at Backgammon
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play Atari games better than humans

<https://youtu.be/2pWv7GOvuf0?t=940>

(15:40 – 22.00 [6 mins 20 secs])

# MARKOV DECISION PROCESSES

- The **Markov Decision Process** (MDP) provides a mathematical framework for reinforcement learning.
- An MDP is used to model optimal decision-making in situations where outcomes are uncertain.



# THE MDP DECISION FRAMEWORK

## **Markov decision processes model uncertainty**

Use probability to model uncertainty about the domain.



## **Markov decision processes model an agent's objectives**

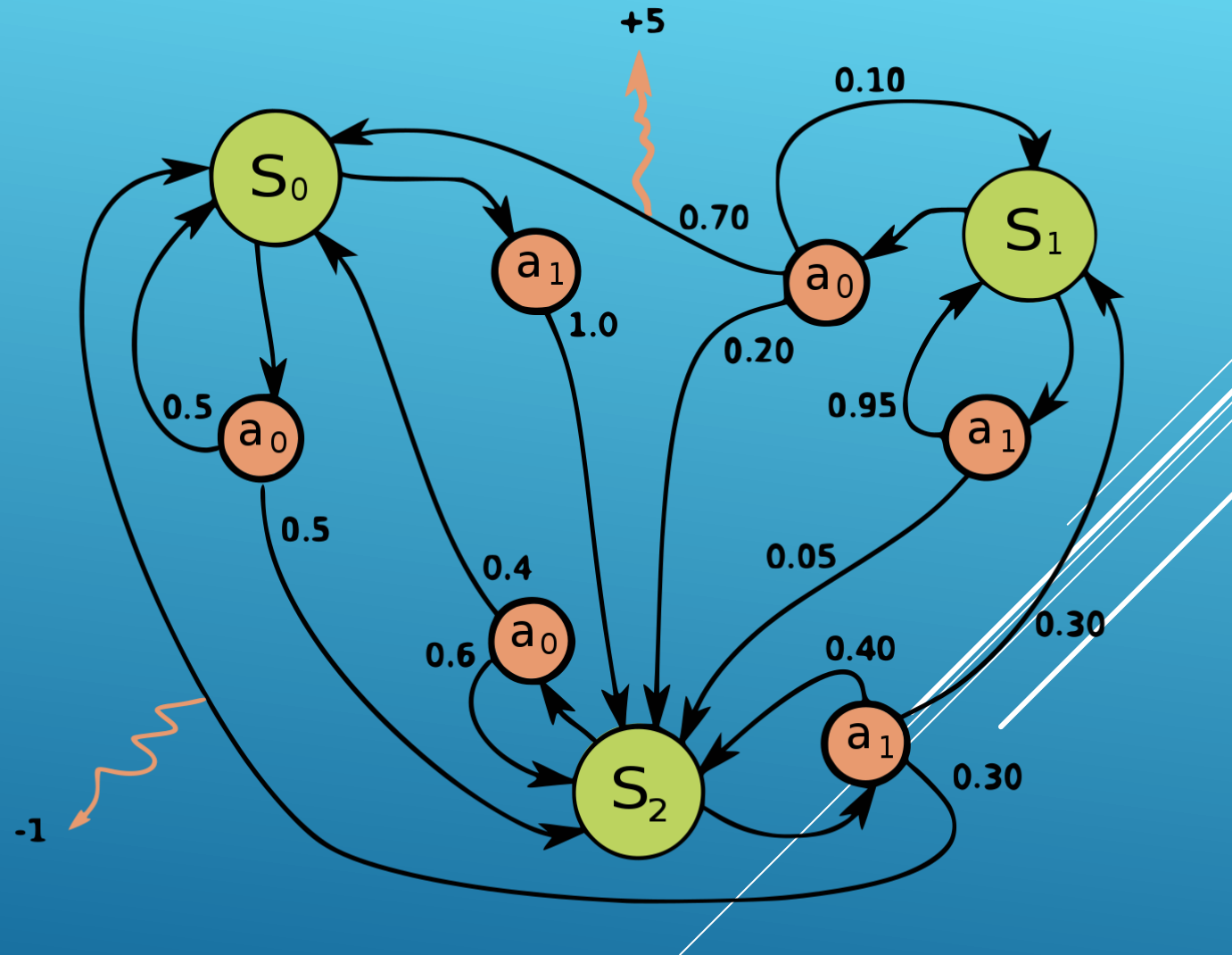
Use utility to model an agent's objectives. The higher the utility, the “happier” your agent is.

## **Markov decision processes find an optimal decision policy**

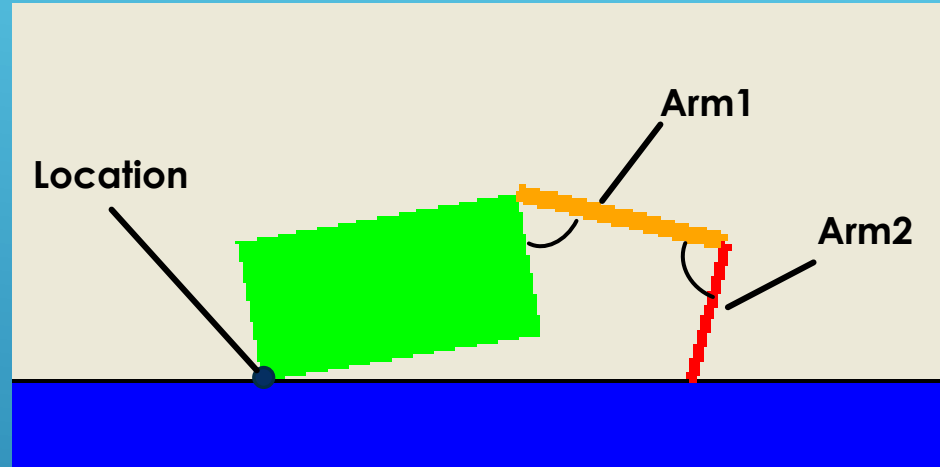
The goal is to discover an optimal decision policy  $\pi$  specifying how the agent should act in all possible states in order to maximize its expected utility.

# MARKOV DECISION PROCESSES

- **States:**  $s_0, \dots, s_n$
- **Actions:**  $a_0, \dots, a_m$
- **Reward Function:** 
- **Transition model:** 
- **Discount factor:**  $\gamma \in [0, 1]$



# APPLICATION: CRAWLER ROBOT

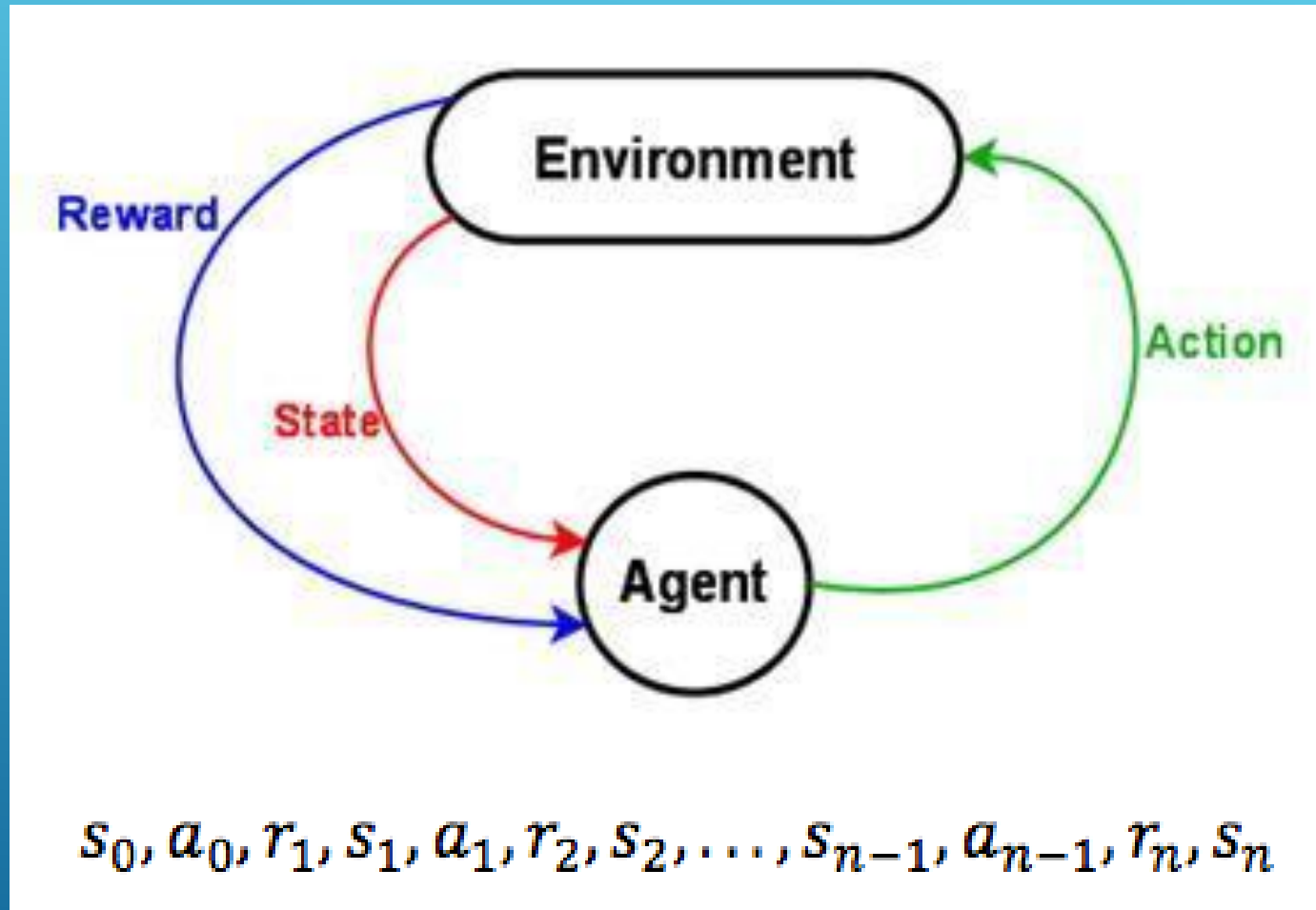


- **States:** <Location, Arm1 angle, Arm2 angle>
- **Actions:** increase Arm1 angle, decrease Arm1 angle, increase Arm2 angle, decrease Arm2 angle.
- **Reward Function:** +1 if robot moves right, -1 if robot moves left.
- **Transition model:** model of box movement caused by arm movements.

# ALGORITHMS BASED ON THE MARKOV DECISION PROCESS

- **There are many algorithms based on the Markov Decision Process**  
These algorithms can be divided into several subclasses. One way to classify these algorithms is based on how much is known about the environment.
- **MDP algorithms**  
MDP-based algorithms assuming perfect knowledge of the states, actions, rewards and transitions of the problem space.
- **Reinforcement learning algorithms:**  
MDP-based algorithms with knowledge of the states and actions but no knowledge of the rewards and transitions of the problem space.

# THE REINFORCEMENT LEARNING PROBLEM



# DISCOUNTED REWARD

Total Future Reward:

$$R = r_1 + r_2 + r_3 + \dots + r_n$$

Total Reward for One Episode:

$$R_t = r_t + r_{t+1} + r_{t+2} + \dots + r_n$$

Total Discounted Future Reward ( $\gamma$  is the discount factor between 0 and 1)

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{n-t} r_n$$

$$R_t = r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \dots)) = r_t + \gamma R_{t+1}$$

# DISCOUNTED REWARD

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots + \gamma^{n-t} r_n$$

$$R_t = r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \dots)) = r_t + \gamma R_{t+1}$$

- If we set the discount factor  $\gamma=0$ , then our strategy will be short-sighted and relies only on immediate rewards.
- If we want to balance between immediate and future rewards, we should set discount factor to something like  $\gamma=0.9$ .
- If the reinforcement learning problem is *episodic* i.e., each learning trial finishes after a finite time then we can set the discount factor  $\gamma=1$  since the total reward will be finite. For *continuous* RL problems, the discount factor must be less than 1 otherwise, the total reward would be infinite.
- A typical strategy for an agent balances (1) *exploiting* the environment by choosing an action that maximizes the (discounted) future reward; and (2) *exploring* the environment by choosing some other action.

# Q-LEARNING

- In Q-learning we define a function  $Q(s, a)$  representing **the maximum discounted future reward when we perform action  $\underline{a}$  in state  $\underline{s}$  and continue optimally from that point on.**

$$Q(s_t, a_t) = \max R_{t+1}$$

- The way to think about  $Q(s, a)$  is that it is “the best possible score [one is likely to get] at the end of an [episode] after performing action  $\underline{a}$  **in state  $\underline{s}$** ”.
- It is called **Q-function**, because it represents the “quality” of a certain action in a given state.



EDITINGA

# Q(S,A) GIVES THE OPTIMAL POLICY

- Suppose you are in state  $s$  and pondering whether you should take action  $a$  or  $b$ . You want to select the action that results in the highest score at the end of game.
- Once you have the magical Q-function, the answer becomes simple – pick the action with the highest Q-value!

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

- Here  $\pi$  represents the policy, the rule how we choose an action in each state.

# THE BELLMAN EQUATION

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

- The Bellman equations are at the heart of all MDP-based algorithms. It can be shown that when the Bellman equations are satisfied, an optimal policy  $\pi$  has been found.
- The main idea in Q-learning is that **we can iteratively approximate the Q-function using the Bellman equation.**
- With the Bellman Equation, we express the Q-value of state  $s$  and action  $a$  in terms of the Q-value of the next state  $s'$ .
- The maximum future reward for state  $s$  and action  $a$  is the immediate reward plus maximum future reward for the next state  $s'$ .

# THE Q-LEARNING ALGORITHM

```
initialize  $Q[num\_states, num\_actions]$  arbitrarily  
observe initial state  $s$   
repeat  
    select and carry out an action  $a$   
    observe reward  $r$  and new state  $s'$   
     $Q[s, a] = Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$   
     $s = s'$   
until terminated
```

- $\alpha$  in the algorithm is a learning rate that controls how much of the difference between previous Q-value and newly proposed Q-value is taken into account.

# REWRITING THE $Q(S,A)$ UPDATE

$$Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q_k[s', a'] - Q[s, a])$$

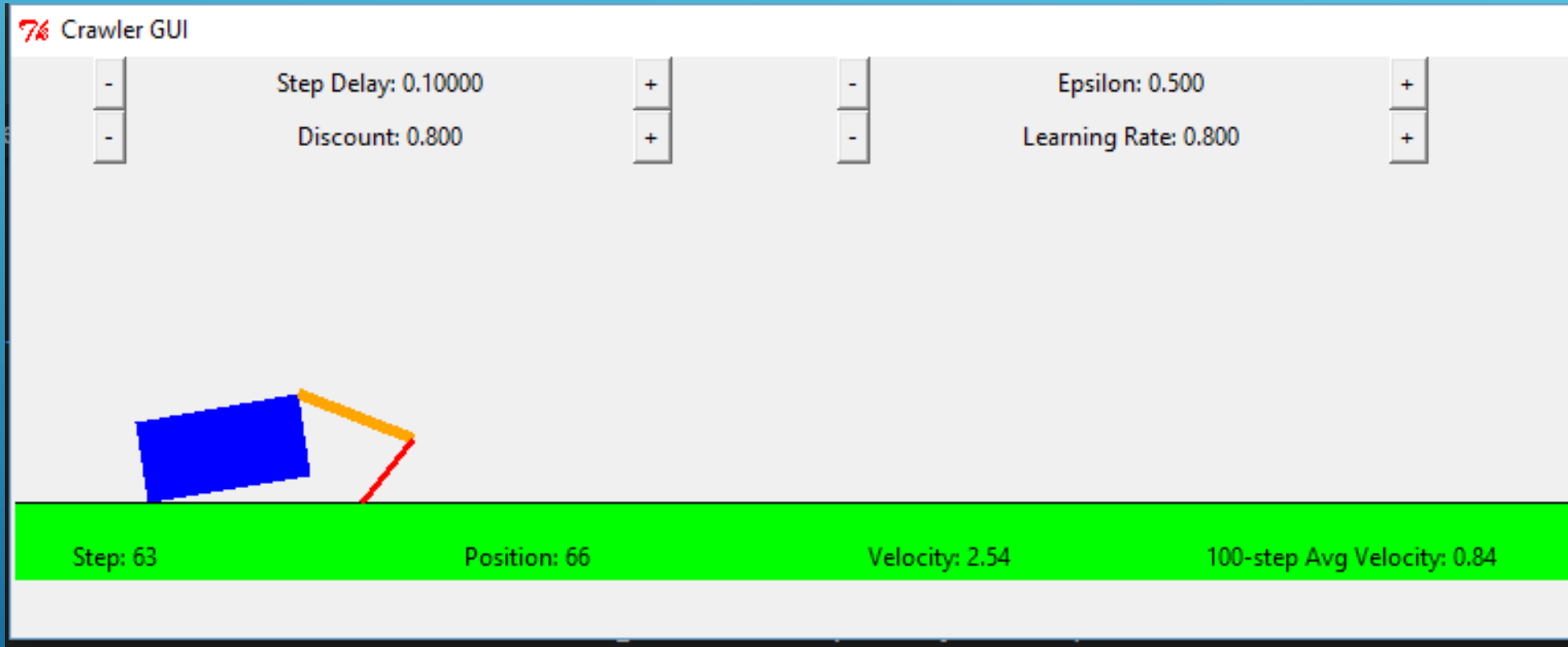
Can be rewritten as:

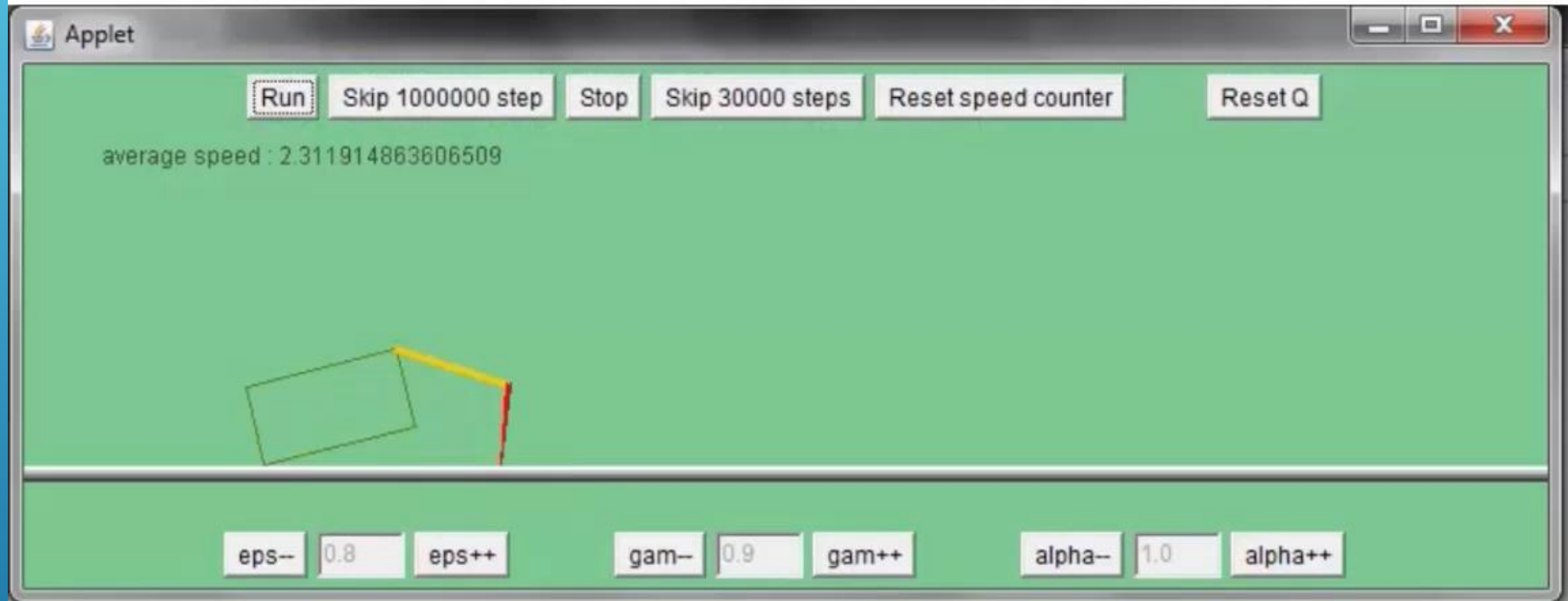
$$Q[s, a] \leftarrow (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q_k[s', a'])$$

# THE Q-LEARNING ALGORITHM CONVERGES ON THE TRUE Q-VALUE

- The  $\max_{a'} Q[s', a']$  that we use to update  $Q[s, a]$  is only an approximation and in early stages of learning it may be completely wrong.
- However the approximation get more and more accurate with every iteration and it has been shown, that if we perform this update enough times, then the Q-function will converge and represent the true Q-value.

# Q-LEARNING EXAMPLE: CRAWLER ROBOT

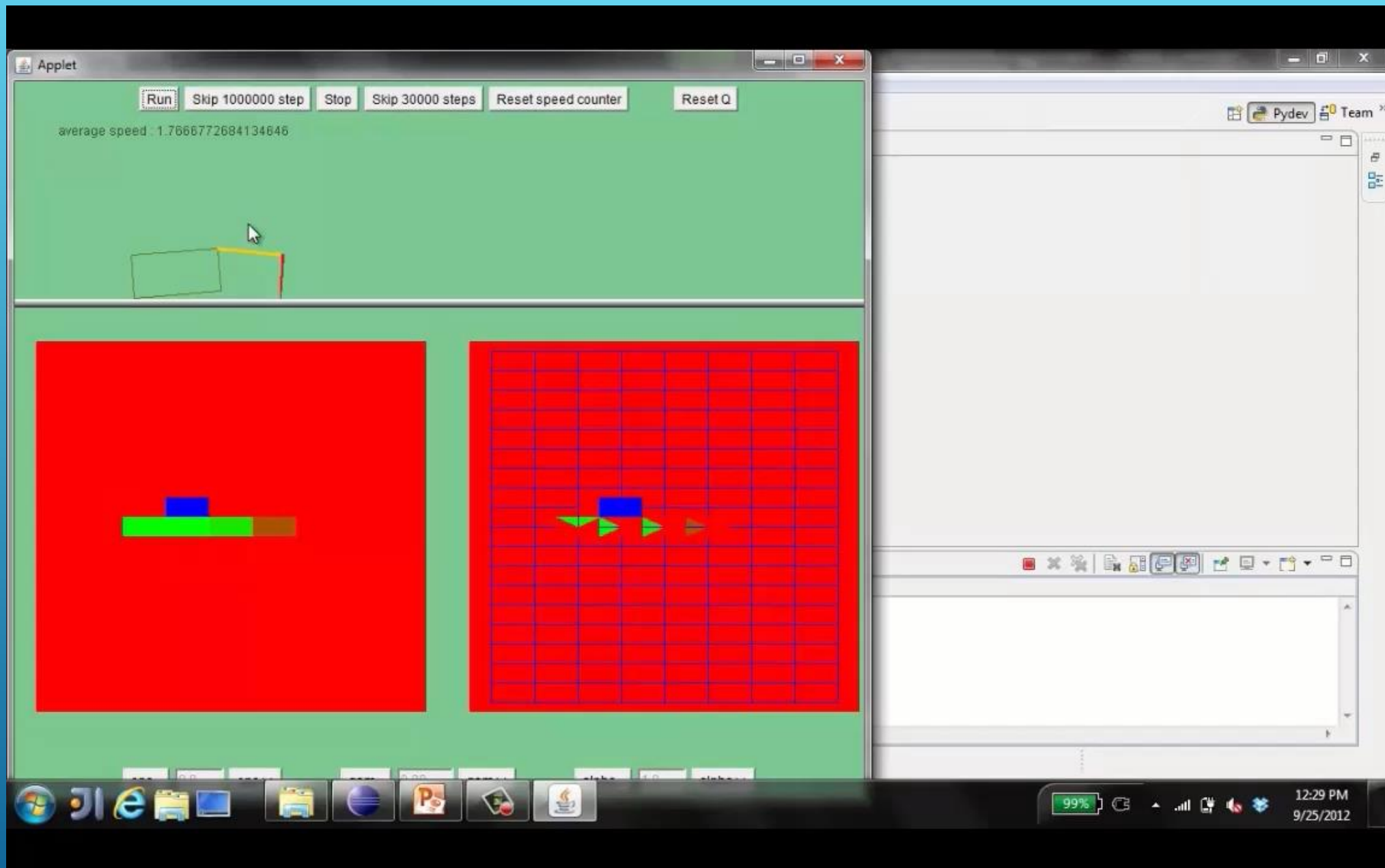




# VIDEO 1 OF Q-LEARNING DEMO -- CRAWLER

Credit: CS188 – Intro to AI, UC Berkeley, [ai.berkeley.edu](http://ai.berkeley.edu)





# VIDEO 2 OF Q-LEARNING DEMO -- CRAWLER

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkeley.edu

# Q-LEARNING EXAMPLE: DISCOUNT EFFECT

Update rule: 
$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) \left[ r + \gamma \max_{a'} Q(s', a') \right]$$

Description	Training Steps	<u>Discount</u> ( $\gamma$ )	Learning Rate ( $\alpha$ )	Avg Velocity
Default	~100K	<b>0.8</b>	0.8	~1.73
Low $\gamma$	~100K	<b>0.5</b>	0.8	0.0
High $\gamma$	~100K	<b>0.919</b>	0.8	~3.33
Low $\alpha$	~100K	0.8	0.2	~1.73
High $\alpha$	~100K	0.8	0.9	~1.73
High $\gamma$ , Low $\alpha$	~100K	0.919	0.2	~3.33

# INTRODUCING REINFORCEMENT LEARNING

Scott O'Hara

Metrowest Developers Machine Learning Group

01/29/2020

# DETAILED REFERENCES

“Demystifying Deep Reinforcement Learning,” 2015, Tamber Matisen

- ▶ <https://www.intel.ai/demystifying-deep-reinforcement-learning/>

University of California, Berkeley CS188:

- ▶ *CS188 – Introduction to Artificial Intelligence*, Profs. Dan Klein, Pieter Abbeel, et al. <http://ai.berkeley.edu/home.html>

David Silver, DeepMind:

- ▶ *Introduction to Reinforcement Learning*  
<http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>

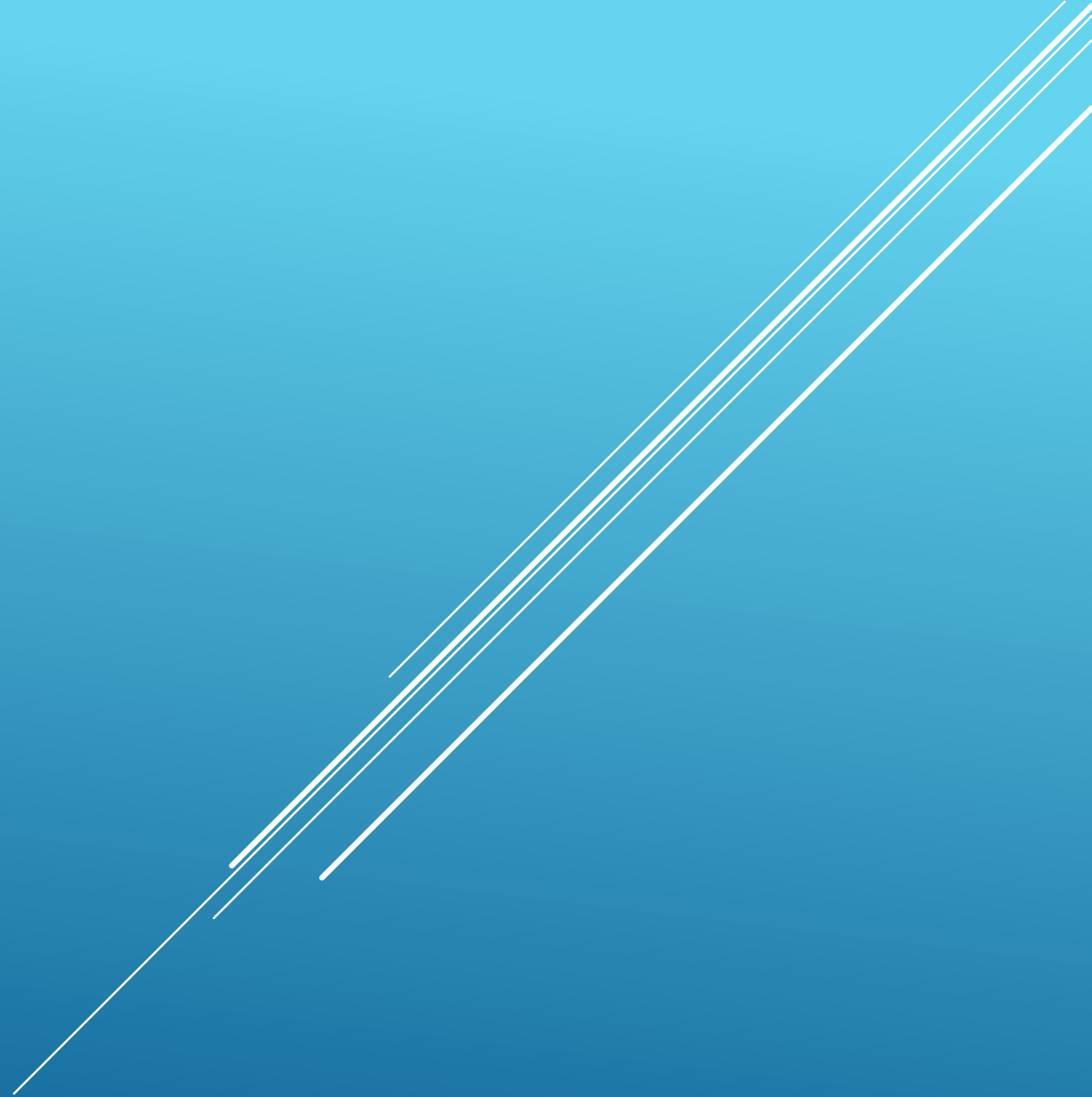
CS181 course at Harvard University:

- ▶ *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Sarah Finney, Spring 2009
- ▶ *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Prof. David C Brooks, Spring 2011
- ▶ *CS181 – Machine Learning*, Prof. Ryan P. Adams, Spring 2014.  
<https://github.com/wihl/cs181-spring2014>
- ▶ *CS181 – Machine Learning*, Prof. David Parkes, Spring 2017.  
<https://harvard-ml-courses.github.io/cs181-web-2017/>

# UC BERKELEY CS188 IS A GREAT RESOURCE

- <http://ai.berkeley.edu/home.html>
- Covers:
  - Search
  - Constraint Satisfaction
  - Games
  - Reinforcement Learning
  - Bayesian Networks
  - Surveys Advanced Topics
  - And more...
- Contains: accessible, high quality YouTube videos, PowerPoint slides and homework.
- Series of projects based on the video game *PacMan*.
- Material is used in many courses around the country.

EXTRA SLIDES



# MDP-BASED ALGORITHMS

- **There are many algorithms based on the Markov Decision Process**  
These algorithms can be divided into several subclasses. One way to classify these algorithms is based on how much is known about the environment.
- **MDP algorithms**  
MDP-based algorithms assuming perfect knowledge of the states, actions, rewards and transitions of the problem space.
- **Reinforcement learning algorithms:**  
MDP-based algorithms with knowledge of the states and actions but no knowledge of the rewards and transitions of the problem space.

# 3 MDP ALGORITHMS

- Assumes complete knowledge of the MDP.
- **Expectimax** – a top-down recursive tree search algorithm similar to the minimax game search algorithm with a fixed search depth (finite horizon) that finds the optimal expected value of the current state.
- **Value Iteration** – a bottom-up dynamic programming algorithm that finds the optimal expected value of every state. There are two variants: 1) *finite horizon value iteration*; and 2) *infinite horizon value iteration*.
- **Policy Iteration** - a bottom-up dynamic programming algorithm that finds the optimal policy.




# REINFORCEMENT LEARNING ALGORITHMS

- Assumes incomplete knowledge of the MDP i.e., no knowledge of the reward or transition models.
- **Model-based reinforcement learning** are based on the various MDP algorithms and try to build the reward and transition models by exploring the environment.
- **Model-free reinforcement learning** ignore the reward and transition models and try to learn the Q and/or V functions directly.

# MODEL-BASED REINFORCEMENT LEARNING

**Model-based reinforcement learning** is based on the typically involve alternating between the following steps:

1. Exploit the environment using the current reward and transition models.
  2. Explore the environment to learn the reward and transition models.
  3. Run an MDP algorithm to create new reward and transition models.
- 
- A series of three parallel white diagonal lines are located in the bottom right corner of the slide, extending from the right edge towards the center.

# MODEL-FREE REINFORCEMENT LEARNING

**Model-free reinforcement learning** does not try to learn the reward and transition models. Instead it attempts to find an optimal policy  $\pi$  using other means. There are many such algorithms including:

**Q-Learning**

**SARSA**

**Monti-Carlo**

**Approximate Q-Learning**

**Dyna-Q**

**Deep Q-Learning (DQN)**

...

# MODEL-BASED VS MODEL-FREE

## Model-Based

- **Pros:**
  - Makes maximal use of experience.
  - Solves model optimally given enough experience.
- **Cons:**
  - Requires computationally expensive solution procedure.
  - Requires the model to be small enough to solve.

## Model-Free

- **Pros:**
  - Solution procedure is relatively more efficient.
  - Can handle much larger models.
- **Cons:**
  - Learns more slowly. Does not learn as much as a model-based RL in a single training episode. ("Leaves information on the table").
  - Unable to make predictions about transitions in the environment.