INTRODUCING REINFORCEMENT LEARNING

Scott O'Hara

Metrowest Developers Machine Learning Group

REFERENCES

The material for these talks are primarily drawn from the slides, notes and lectures of these courses:

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:

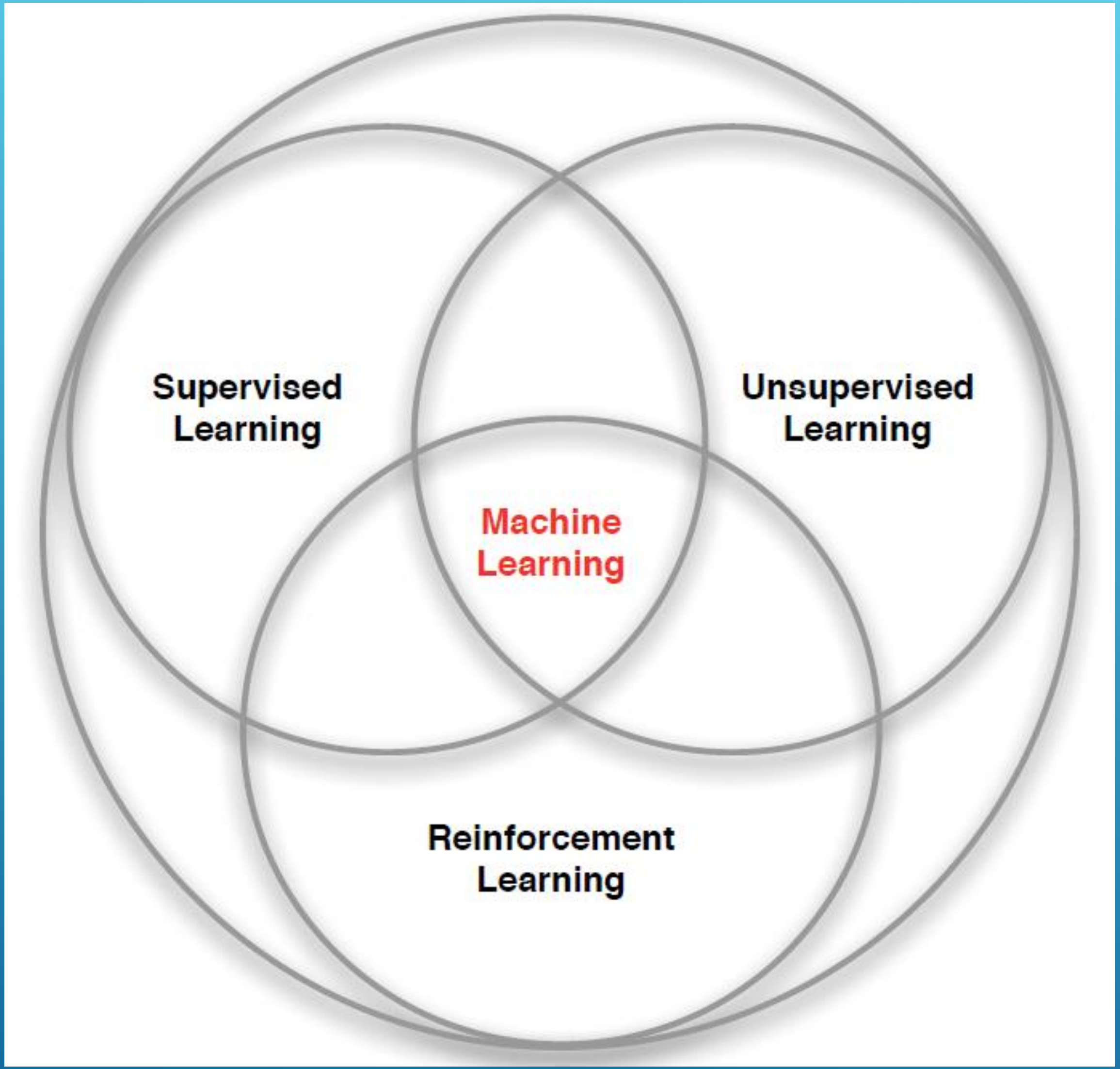
- Introduction to Reinforcement Learning http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

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.



TYPES OF MACHINE LEARNING

Credit: adapted from lecture slides David Silver, DeepMind, "Introduction to Reinforcement Learning

TYPES OF MACHINE LEARNING

There are (at least) 3 broad categories of machine learning problems:

Supervised Learning

$\boldsymbol{Data = \{(x_1, y_1), \ldots, (x_n, y_n)\}}$

e.g., linear regression, decision trees, SVMs

Unsupervised Learning

$\boldsymbol{Data = \{x_1, , \ldots, x_n\}}$

e.g., K-means, HAC, Gaussian mixture models

Reinforcement Learning

$\boldsymbol{Data = \{s_1, a_1, r_1, s_2, a_2, r_2 \ldots\}}$

an agent learns to act in an uncertain environment by training on data that are sequences of **state, action, reward.**

REINFORCEMENT LEARNING: THE BASIC IDEA

- Select an action.
- If action leads to reward, reinforce that action.
- If action leads to punishment, avoid that action.
- Basically, a computational form of Behaviorism (Pavlov, B. F. Skinner).

CHARACTERISTICS OF REINFORCEMENT LEARNING

- Learning happens as the agent interact with the world.
- Unlike supervised learning, there are no training or test sets. Training is guided by rewards and punishments.
- The amount of "training 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/Ih8EfvOzBOY

3 mins 45 secs

Credit: Two Minute Papers - Károly Zsolnai-Fehér

REINFORCEMENT LEARNING EXAMPLE: TERRAIN TRAVERSAL

Terrain Traversal with Reinforcement Learning
https://youtu.be/_yjHPu1aYCY

2 mins 38 secs

Credit: Two Minute Papers #26- Károly Zsolnai-Fehér

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)

Credit: David Silver, DeepMind, "Introduction to Reinforcement Learning

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.

<https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=855934986> [accessed 11 September 2018]

THE DECISION FRAMEWORK FOR MDPS

Modeling Uncertainty

Use probability to model uncertainty about the domain.

Modeling an Agent's Objectives

Use utility to model an agent's objectives.

Decision Policy

The goal is to design a decision policy, describing how the agent should act in all possible states in order to maximize its expected utility.

UNCERTAINTY IS MODELED WITH PROBABILITY

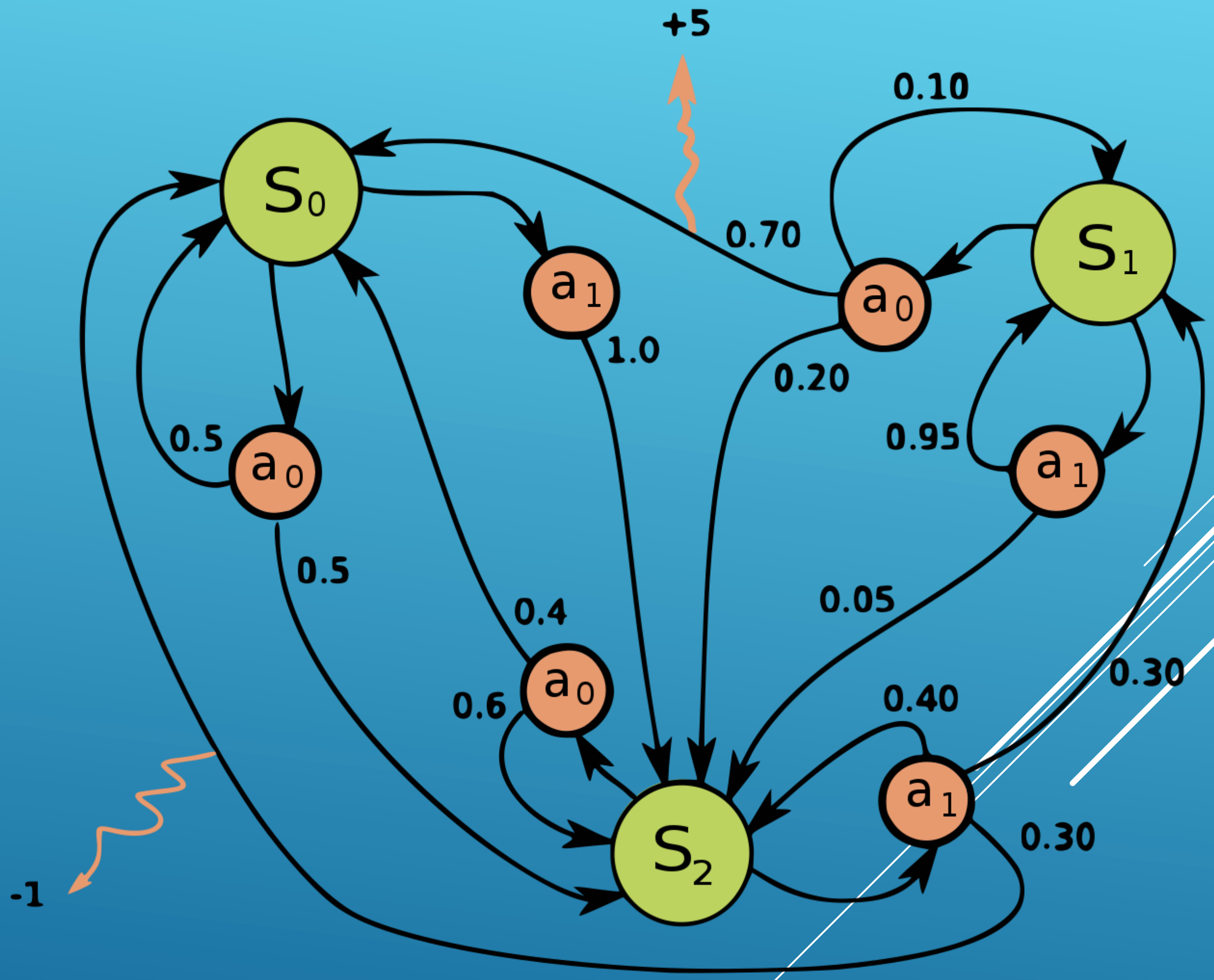
- The agent may not know the current state of the world
- The effects of the agent's action might be unpredictable.
- Things happen that are outside the agent's control.
- The agent may be uncertain about the correct model of the world.

GOALS ("HAPPINESS") MODELED WITH UTILITY

1. Utility is a real number.
2. The higher the utility, the "happier" your robot is.
3. Utility is based on the assumptions of **Utility Theory**, which if obeyed, make you rational.
4. For example, if you prefer reward A to reward B, and you prefer reward B to reward C, then you should prefer reward A to reward C.

MARKOV DECISION PROCESSES

- States: $s_0, \ldots, s_n$
- Actions: $a_0, \ldots, a_m$
- Reward Function:
- Transition model:

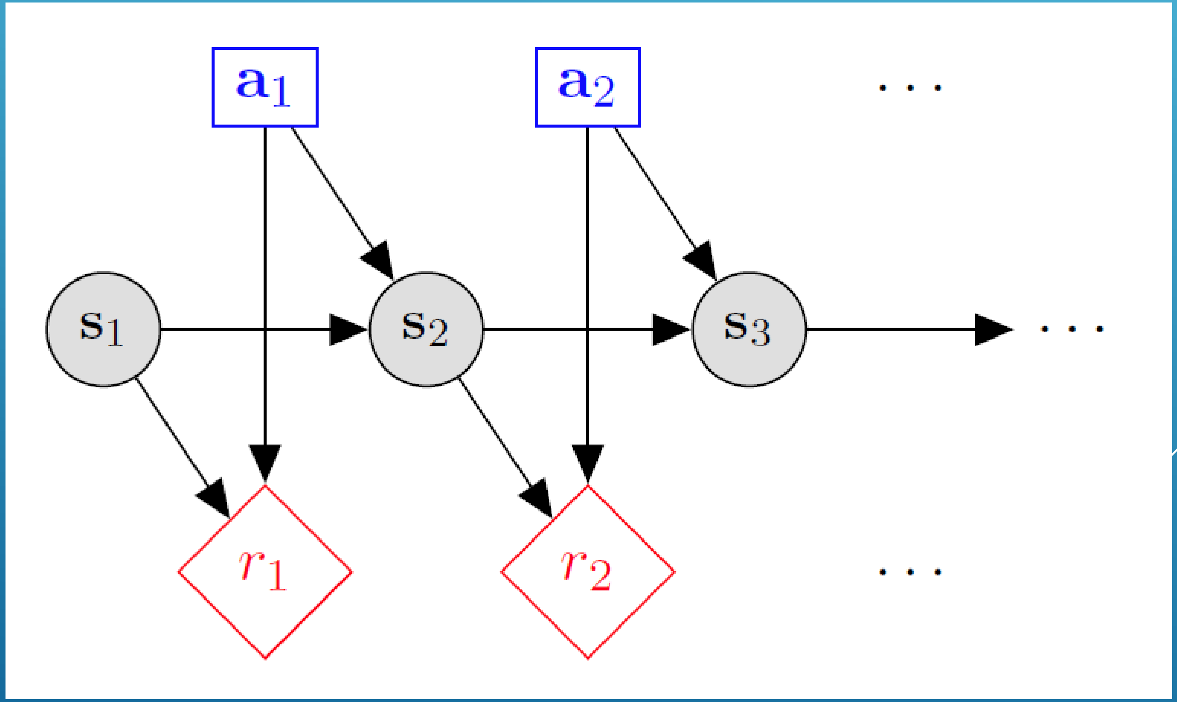


MDP GOAL: FIND AN OPTIMAL POLICY π

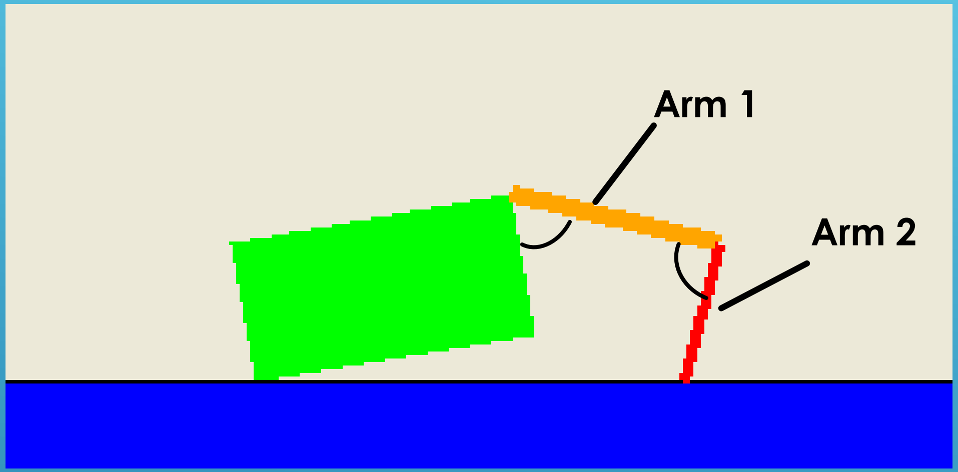
GOAL: find a **policy π** that tells you what action to take in each state. We want to find 'rewarding' policies.

n state policy:

$$\pi(s_1) = a_1$$
$$\pi(s_2) = a_2$$
$$\pi(s_3) = a_3$$
$$\dots$$
$$\pi(s_n) = a_n$$



APPLICATION: CRAWLER ROBOT



- **States:** <location, arm 1 angle, arm 2 angle>
- **Actions:** raise Arm 1, lower Arm 1, raise Arm 2, lower Arm 2
- **Reward Function:** +1 if robot moves right, -1 if robot moves left
- **Transition model:** model of box movement caused by arm movements.

SCOPE OF MDP APPLICABILITY

The Markov Decision Process is a general probabilistic framework and can be applied in many different scenarios.

Planning

- Full access to the MDP, compute an optimal policy.
- “How do I act in a known world?"

Policy Evaluation

- Full access to the MDP, compute the `value' of a fixed policy.
- “How will this plan perform under uncertainty?“

Reinforcement Learning

- Limited MDP access: reward and transition models unknown.
- “Can I learn to act in an uncertain world?"

<https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=855934986> [accessed 11 September 2018]

LOOKING AHEAD

- **Bellman equations**
  Mathematical underpinning for MDP and RL algorithms

- **MDP algorithms**
  Planning and policy evaluation algorithms based on perfect knowledge of the states, actions, rewards and transitions of the problem space.

- **Reinforcement learning algorithms:**
  Policy optimization 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 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 various MDP algorithms and 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.

MODEL-FREE REINFORCEMENT LEARNING

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

- Q-Learning
- SARSA
- Monti-Carlo
- Approximate Q-Learning
- Dyna-Q
- Deep Q-Learning
- …

MODEL-BASED VS MODEL-FREE

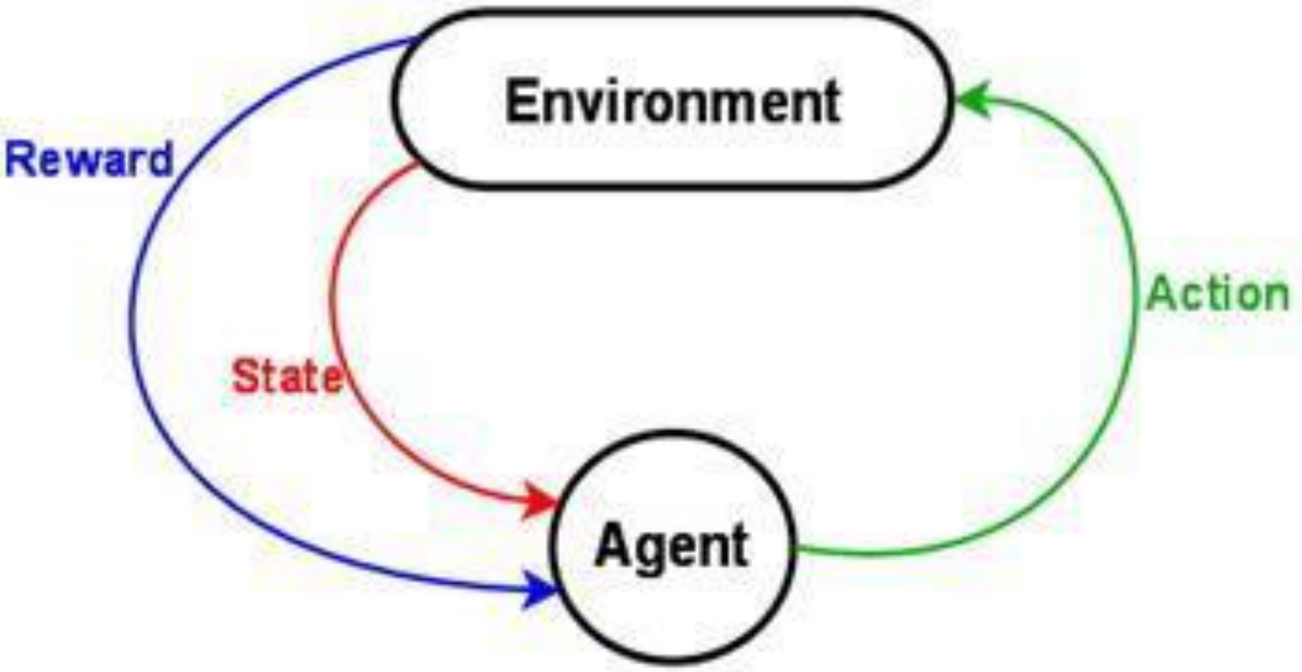
Model-Based

- **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.

THE REINFORCEMENT LEARNING PROBLEM



$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

DISCOUNTED REWARD

Total Future Reward:

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

Total Reward for One Episode:

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

Total Discounted Future Reward (γ is the discount factor between 0 and 1)

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

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

<https://www.intel.ai/demystifying-deep-reinforcement-learning> [accessed 29 January 2020]

DISCOUNTED REWARD

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

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

- If we set the discount factor γ=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 γ=0.9.
- If our environment is deterministic and the same actions always result in same rewards, then we can set discount factor γ=1.
- A good strategy for an agent would be to **always choose an action that maximizes the (discounted) future reward**.

<https://www.intel.ai/demystifying-deep-reinforcement-learning> [accessed 29 January 2020]

Q-LEARNING

- In Q-learning we define a function *Q(s, a)* representing **the maximum discounted future reward when we perform action a in state 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 at the end of the game after performing action a **in state s**".

- It is called **Q-function**, because it represents the "quality" of a certain action in a given state.

<https://www.intel.ai/demystifying-deep-reinforcement-learning> [accessed 29 January 2020]

Q(S,A) GIVES THE OPTIMAL POLICY

- Suppose you are in state 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 really simple – pick the action with the highest Q-value!

$$\pi(s) = argmax_a\, Q(s,a)$$

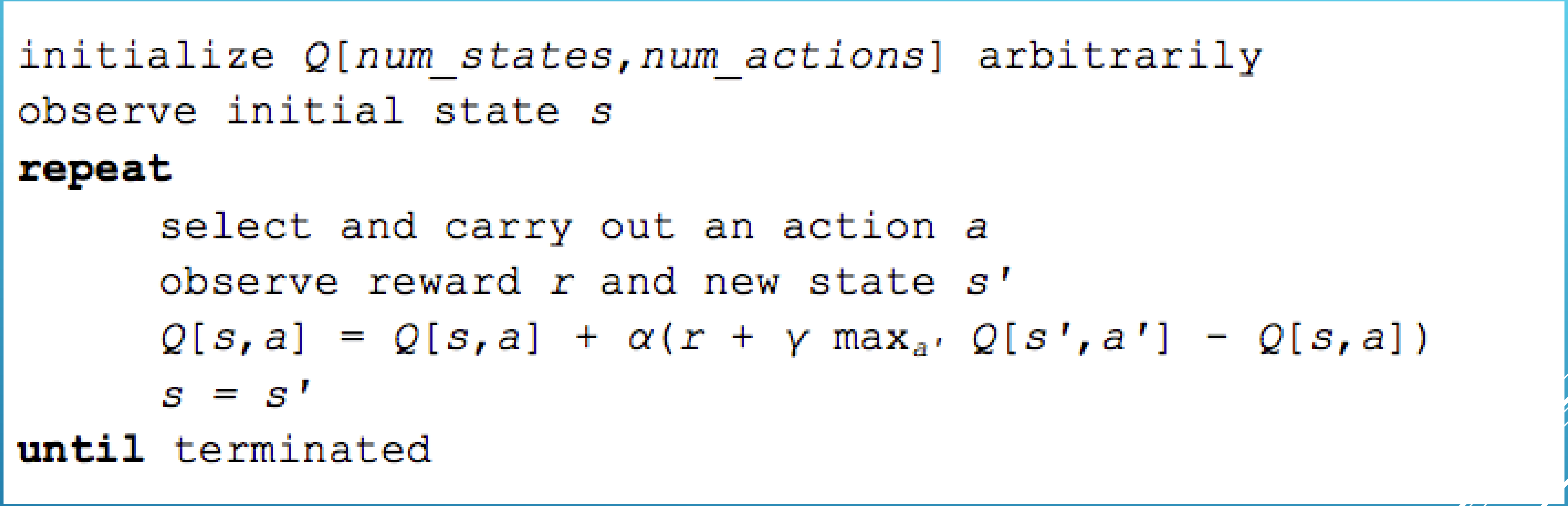
- Here π represents the policy, the rule how we choose an action in each state.

THE BELLMAN EQUATION

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

- The 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
```

- a 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.

<https://www.intel.ai/demystifying-deep-reinforcement-learning> [accessed 29 January 2020]

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:

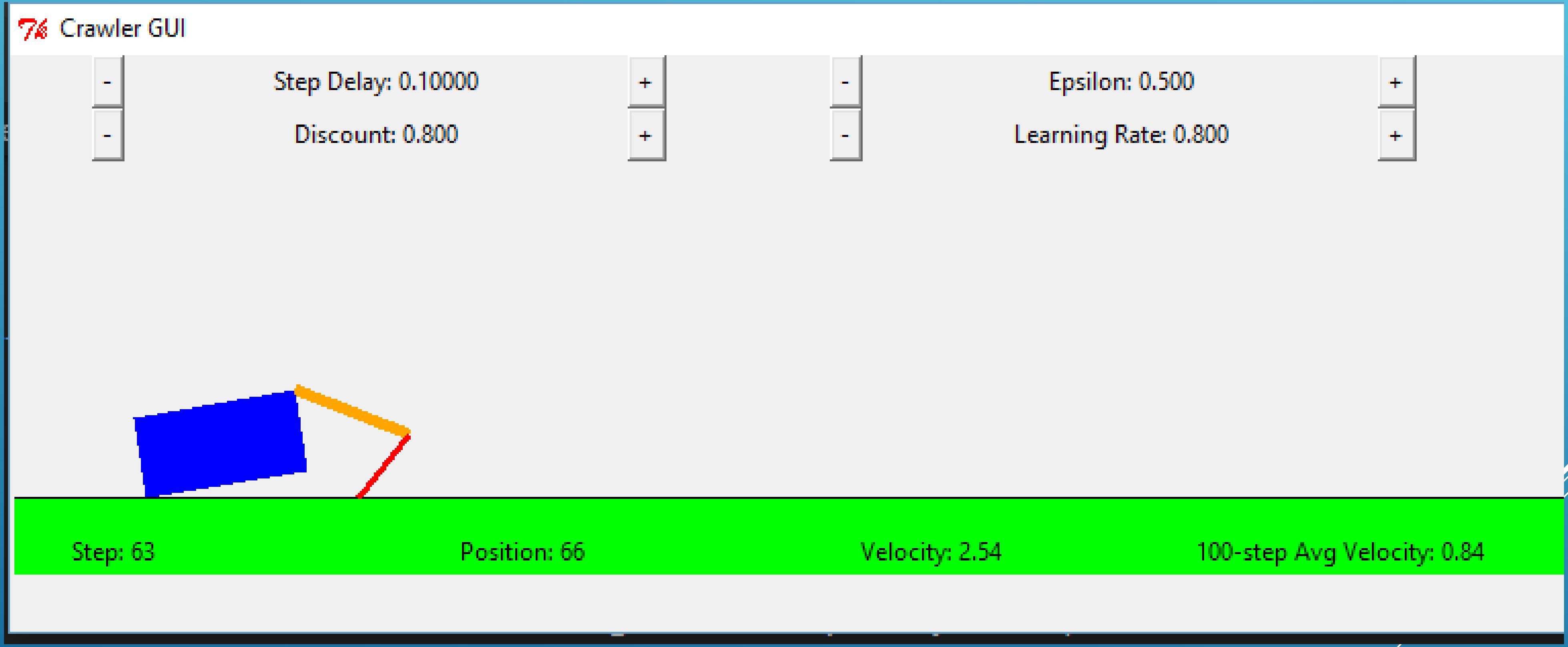
$$Q[s,a] \leftarrow (1-\alpha)Q[s,a] + \alpha(r + \gamma \max_{a'} Q_k[s',a'])$$

<https://www.intel.ai/demystifying-deep-reinforcement-learning> [accessed 29 January 2020]

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



Step Delay: 0.10000

Discount: 0.800

Epsilon: 0.500

Learning Rate: 0.800

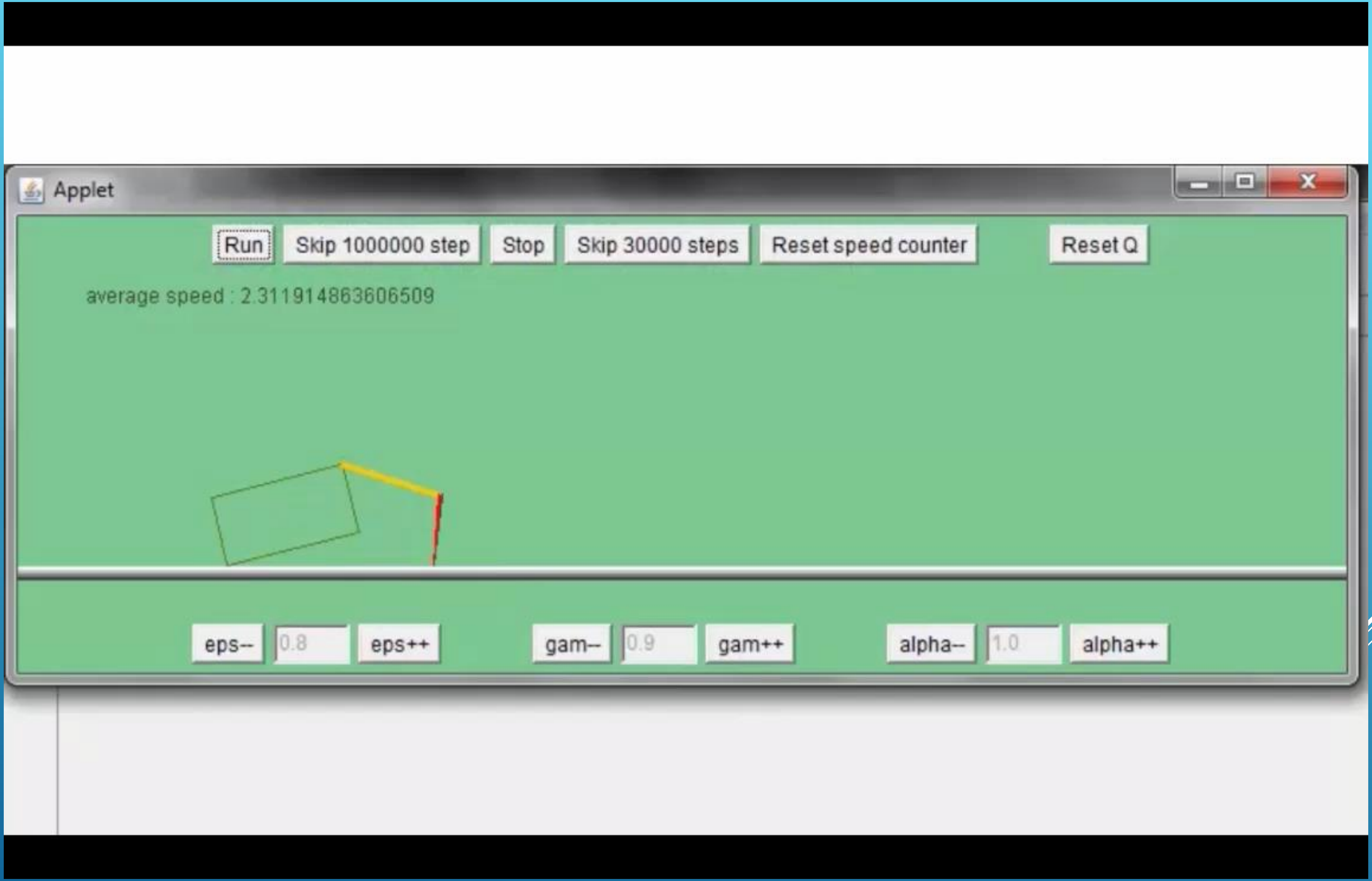
Step: 63

Position: 66

Velocity: 2.54

100-step Avg Velocity: 0.84

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



average speed : 2.3119148636065O9

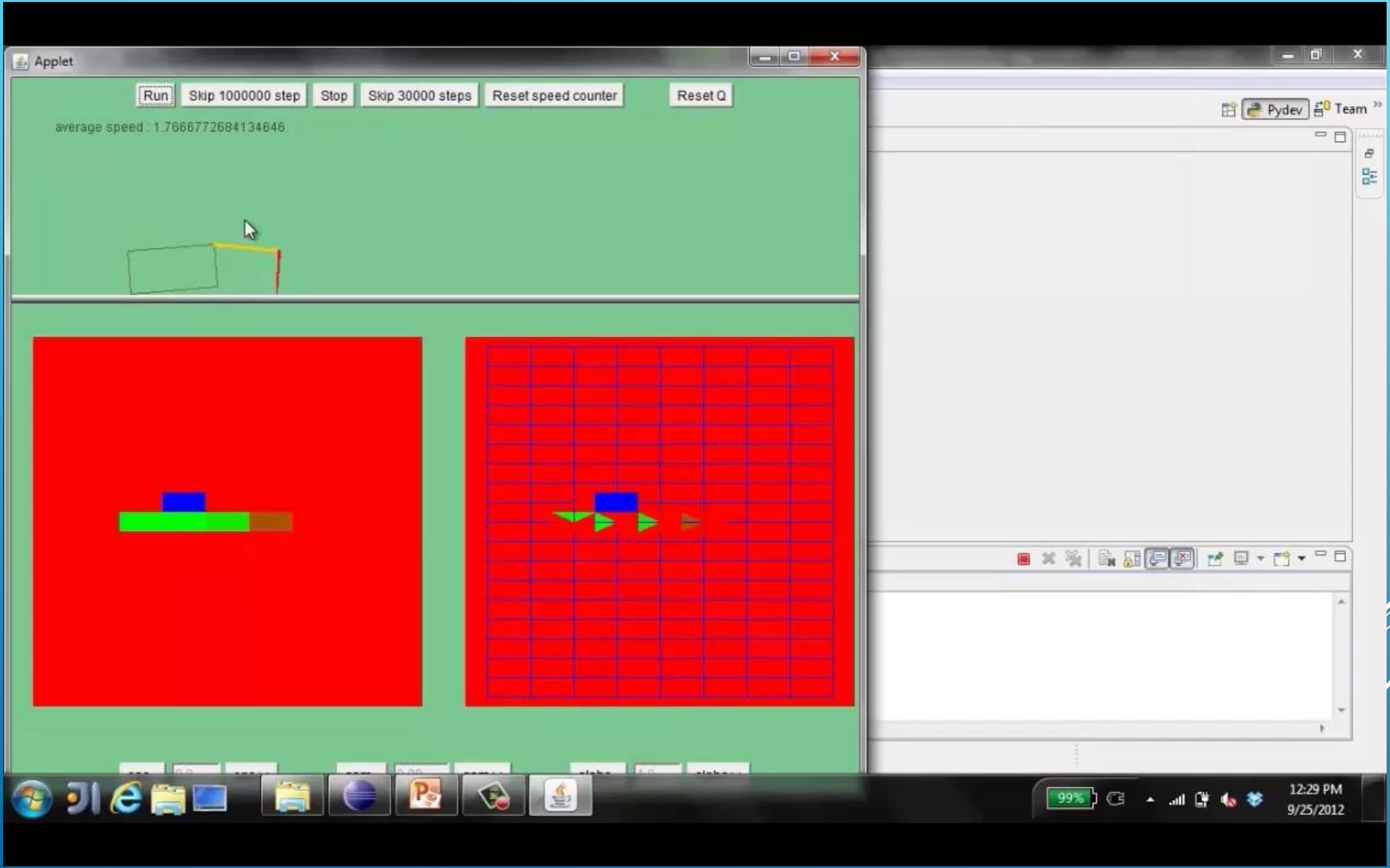
0.8

0.9

1.0

VIDEO 1 OF Q-LEARNING DEMO -- CRAWLER

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



average speed : 1.7666772684134646

VIDEO 2 OF Q-LEARNING DEMO -- CRAWLER

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkely.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	Discount (γ)	Learning Rate (α)	Avg Velocity
Default	~100K	**0.8**	0.8	~1.73
Low γ	~100K	**0.5**	0.8	0.0
High γ	~100K	**0.919**	0.8	~3.33
Low α	~100K	0.8	0.2	~1.73
High α	~100K	0.8	0.9	~1.73
High γ, Low α	~100K	0.919	0.2	~3.33

INTRODUCING REINFORCEMENT LEARNING

Scott O'Hara

Metrowest Developers Machine Learning Group