Q-LEARNING (YET AGAIN!)

Scott O’Hara
Metrowest Developers Machine Learning Group
12/12/2018

REFERENCES

The material for this talk is 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

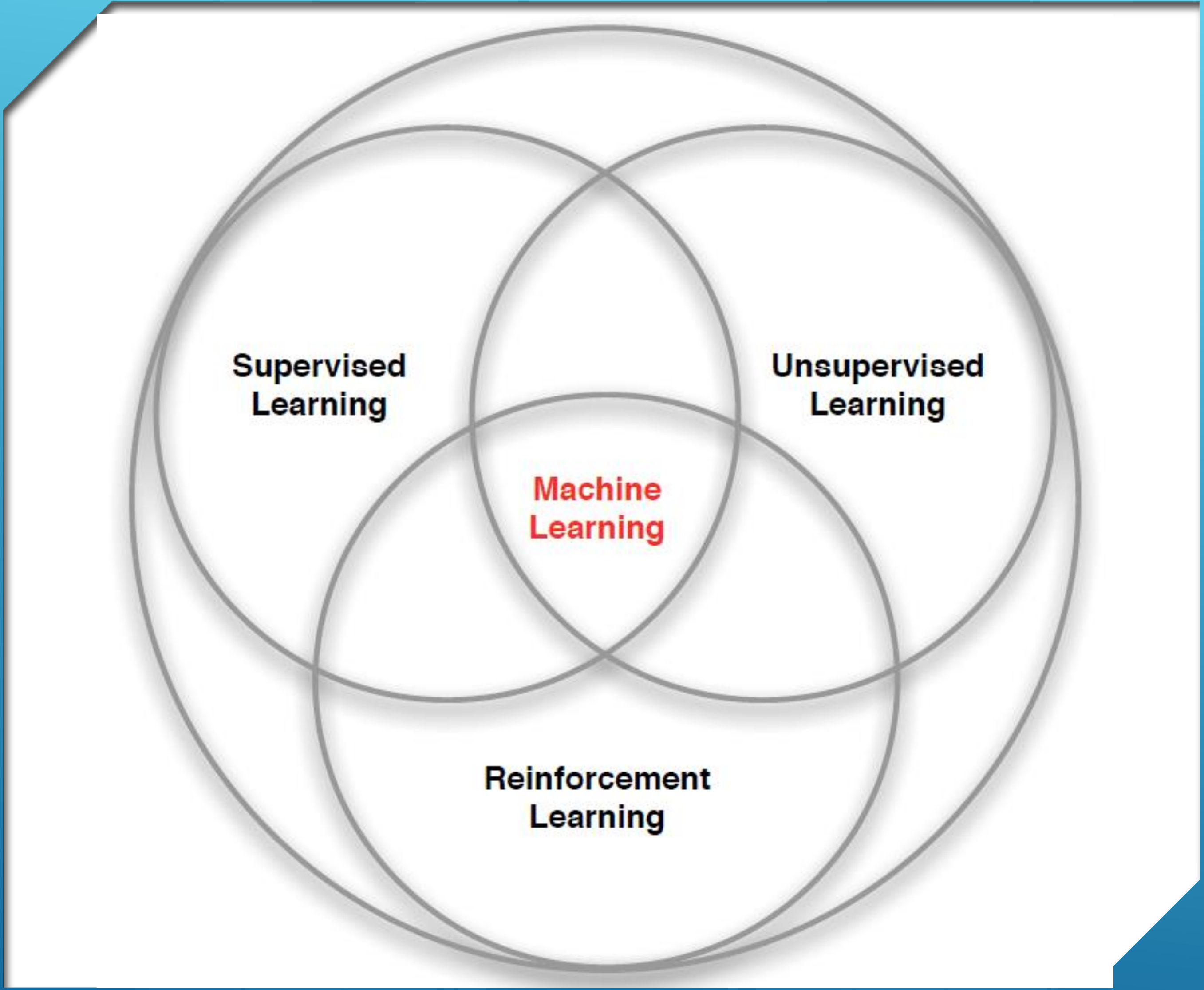
CS181 course at Harvard University:

- *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/

Q-LEARNING: TALK OUTLINE

- Reinforcement Learning
- What is Q-Learning
- Q-Learning Demos
- Approximate Q-Learning

REINFORCEMENT LEARNING



BRANCHES OF MACHINE LEARNING

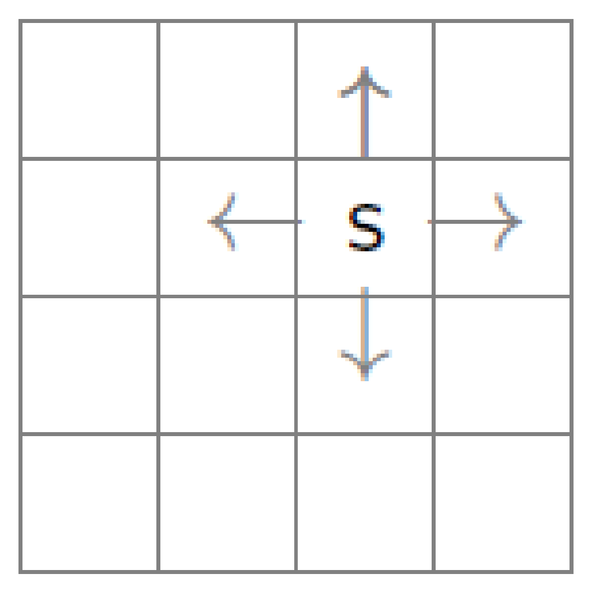
Credit: adapted from lecture slides David Silver, DeepMind, "Introduction to Reinforcement Learning

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).

THE LEARNING FRAMEWORK

- Learning is performed **online**, learn as we interact with the world



- In contrast with supervised learning, there are no training or test sets. The reward is accumulated over interactions with the environment.
- Data is not fixed, more information is acquired as you go.
- The training distribution can be influenced by action decisions.

EXAMPLES OF REINFORCEMENT LEARNING

- 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

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

EXAMPLES OF REINFORCEMENT LEARNING

RL Course by David Silver – Lecture1: Introduction to Reinforcement Learning

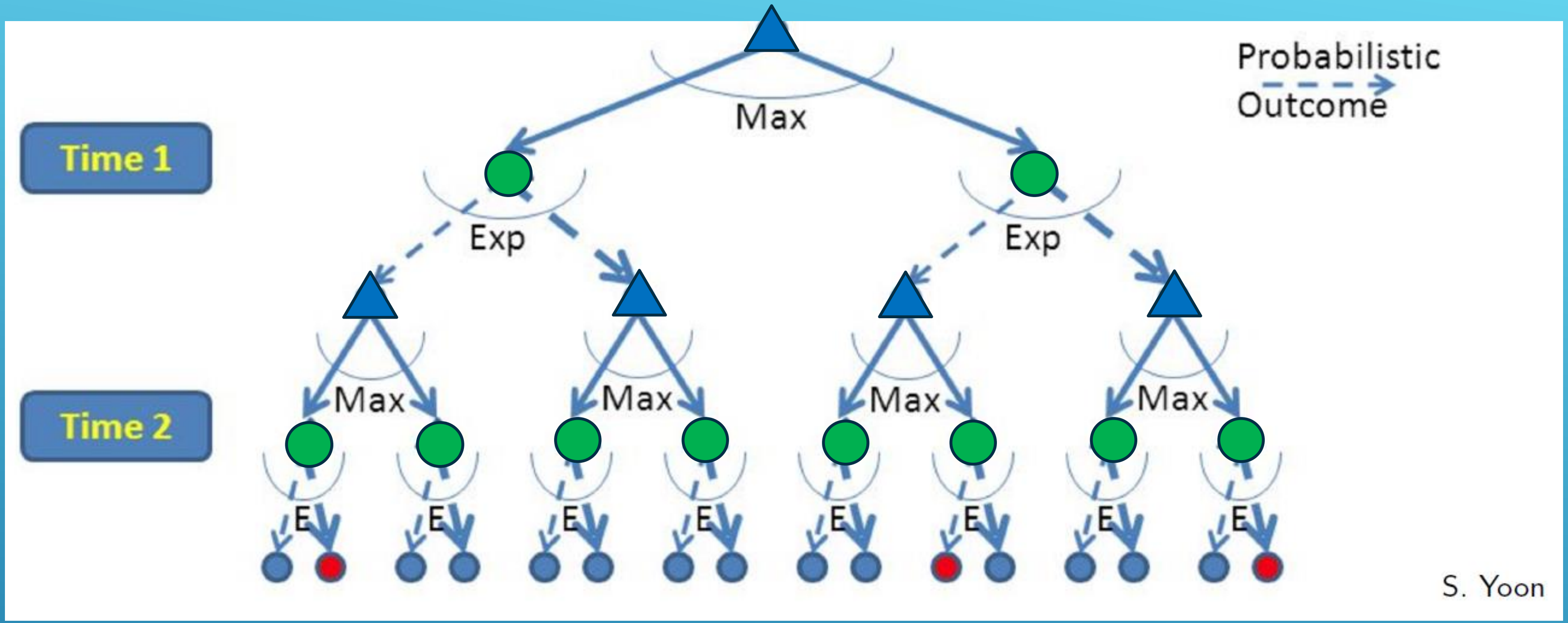
https://www.youtube.com/watch?v=2pWv7GOvuf0

12:25 – 22.00

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

Q-LEARNING

RL IS LIKE A GAME AGAINST NATURE



- Reinforcement learning is like a game-playing algorithm.
- Nodes where you move are called **states**: S ()
- Nodes where nature "moves" are called **Q-states**: <S,A> ()

QUANTITIES TO OPTIMIZE

- The value (utility) of a **state s:**

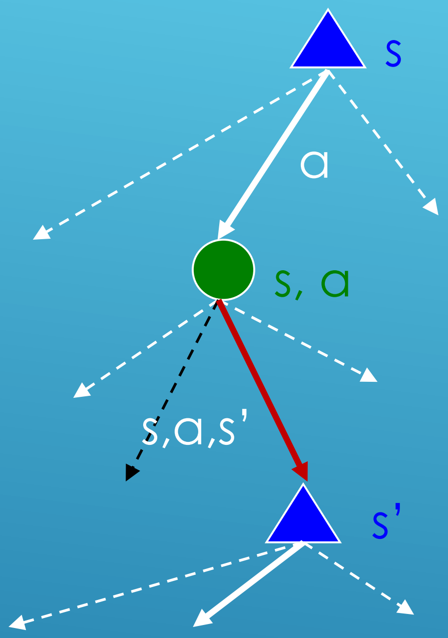
  V(s) = expected utility starting in s and acting optimally

- The value (utility) of a **q-state (s,a)**:

  Q(s,a) = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  π(s) = optimal action from state s



s is a *state*

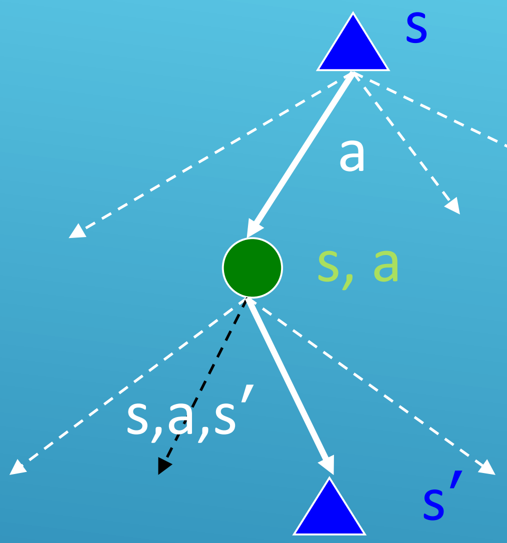
(s, a) is a *q-state*

(s,a,s') is a *transition*

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

THE BELLMAN EQUATIONS

- The Bellman Equations define a relationship, which when satisfied guarantees that $\boldsymbol{V(s)}$ and $\boldsymbol{Q(s,a)}$ are optimal for each state and action.
- This in turn guarantees that the policy π^* is optimal.
- There is one equation $\boldsymbol{V^*(s)}$ for each state *s*.
- There is one equation $\boldsymbol{Q^*(s,a)}$ for each state *s* and action *a*.

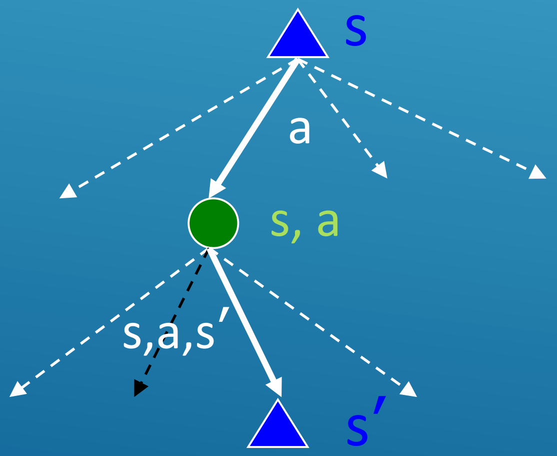


THE BELLMAN EQUATIONS

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

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$



- **States:** $s_1, \dots, s_n$
- **Actions:** $a_1, \dots, a_m$
- **Reward Function:**

  $R(s,a,s') \in R$
- **Transition model:**

  $T(s,a,s') = P(s'|s,a)$
- **Discount factor:** $\gamma \in [0,1]$

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

THE OPTIMAL VALUE UTILITY EQUATION V*

- Focusing on different Bellman Equations Gives Different Algorithms
- The V* equation gives rise to these algorithms previously discussed:
  - Expectimax
  - Value Iteration
  - Policy Iteration

$$V^*(s) = \max_a Q^*(s,a) \quad \textbf{[1]}$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right] \quad \textbf{[2]}$$

$$\Downarrow$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

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

Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

THE OPTIMAL VALUE UTILITY EQUATION Q*

The Q* equation gives rise to the Q-Learning algorithm.

$$V^*(s) = \max_a Q^*(s,a) \qquad \textbf{[1]}$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right] \qquad \textbf{[2]}$$

$$\Downarrow$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

Credit: CS188 – Intro to AI, UC Berkeley, ai.berkely.edu
Credit: CS181 – Machine Learning, D. Parkes, Harvard Univ, Spring 2017

MODEL-BASED VS. MODEL-FREE

- <u>Model-Based RL</u>
  - Explicitly learn:
    - the transition model T(s,a,s')
    - The reward model R(s,a,s')
  - Use Value Iteration, etc. to find optimal policy π^*.

- <u>Model-Free RL</u>
  - Don't learn T(s,a,s') and R(s,a,s').
  - Q-Learning – learn Q(s,a) directly.

MODEL-BASED RL PROS AND CONS

- **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 RL PROS AND CONS

- **Pros:**
  - Solution procedure is much 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").

Q-LEARNING

- What to do about *T(s,a,s')* and *R(s,a,s')*, since we don't have these functions?

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

- Use sampling to learn Q(s,a) values as you go
  - Receive a sample transition: $\boxed{(s,a,r,s')}$
  - Consider your old estimate: $\boxed{Q(s,a)}$
  - Consider your new sample estimate: $\boxed{Q(s,a) = r + \gamma \max_{a'} Q_k(s',a')}$
  - Incorporate the new estimate into a running average based on the learning rate α:

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

Q-LEARNING UPDATE RULE

- On transitioning from state *s* to state *s'* on action *a*, and receiving reward *r*, update:

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

- α is the **learning rate**
  - A large α results in quicker learning but may not converge.
  - α is often decreased as learning goes on.
- γ is the **discount rate** i.e., discounts future rewards

Q-LEARNING ALGORITHM

For each state s and action *a:*

$$\boxed{Q(s,a) \leftarrow 0}$$

Begin in state s:

Repeat:

Given s and the set of actions $A_s$associated with s:

- **CHOOSE ACTION a** $\in A_s$ **← *HOW?***
- Apply action a.
- Receive reward r and new state s'.
- With transition <s,a,r,s'>, update Q(s,a)

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

$$\boxed{\mathrm{S} \leftarrow s'}$$

CHOOSING AN ACTION: EXPLORATION VS EXPLOITATION

- **Exploit:** maximize the expected utility now.
- **Explore:** choose an action that will help you improve your model.
- **How to Exploit?** use the current policy.
  - e.g., for Q-Learning in state s, choose action with largest Q(s,a)
- **How to Explore?**
  - choose an action randomly
  - choose an action you haven't chosen yet
  - choose an action that will take you to an unexplored state.

EXPLORATION STRATEGY: ϵ-GREEDY

- Explore with probability ϵ. Exploit with probability $1 - \epsilon$.
- Weaknesses:
  - Does not exploit when learning has converged.
- Uses:
  - appropriate if the world is changing.

EXPLORATION STRATEGY: BOLTZMANN

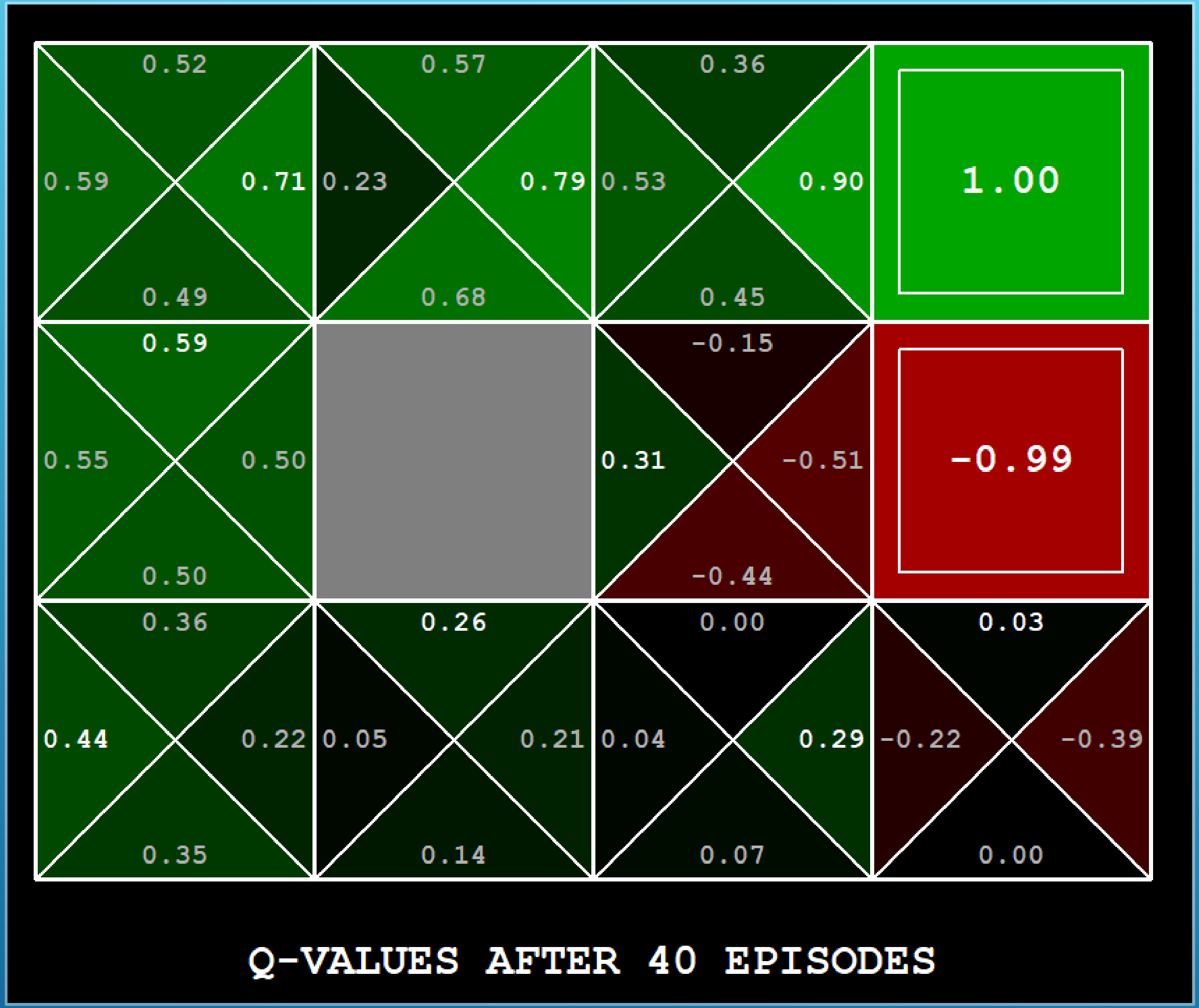
- In state s, choose action a with probability p:

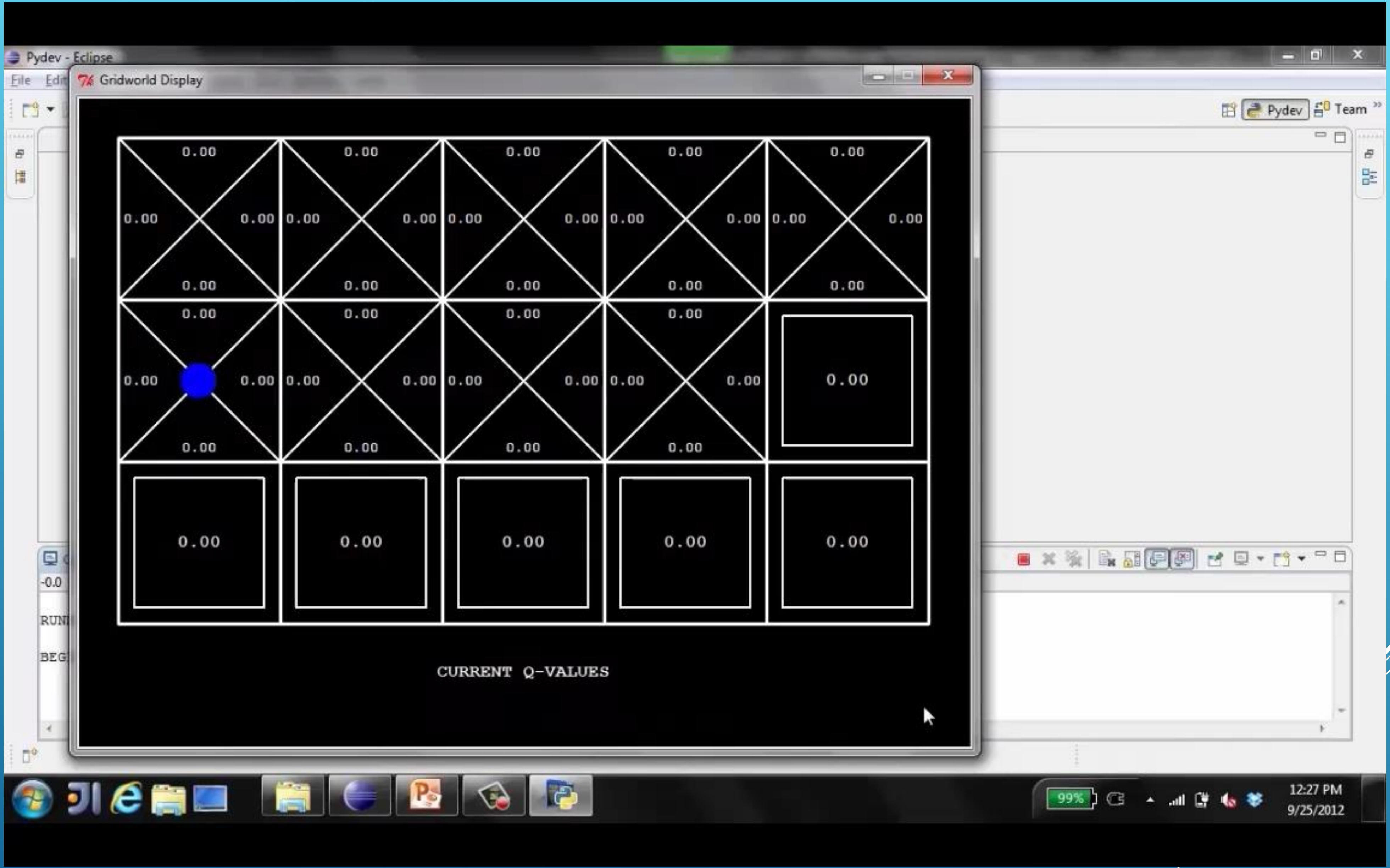
$$p = \frac{e^{\frac{Q(s,a)}{t}}}{\sum_{a'} e^{\frac{Q(s,a')}{t}}}$$

- Simulated annealing: *t* is a "temperature" which "cools" over time.
- High temperature means more exploration
  - $e^{\frac{Q(s,a)}{t}} \to 1$, implies actions taken at about equal probability.
- As t cools, exploration is reduced.
  - $e^{\frac{Q(s,a)}{t}} \to \infty$, largest $Q(s,a)$ becomes most probable.

Q-LEARNING DEMOS

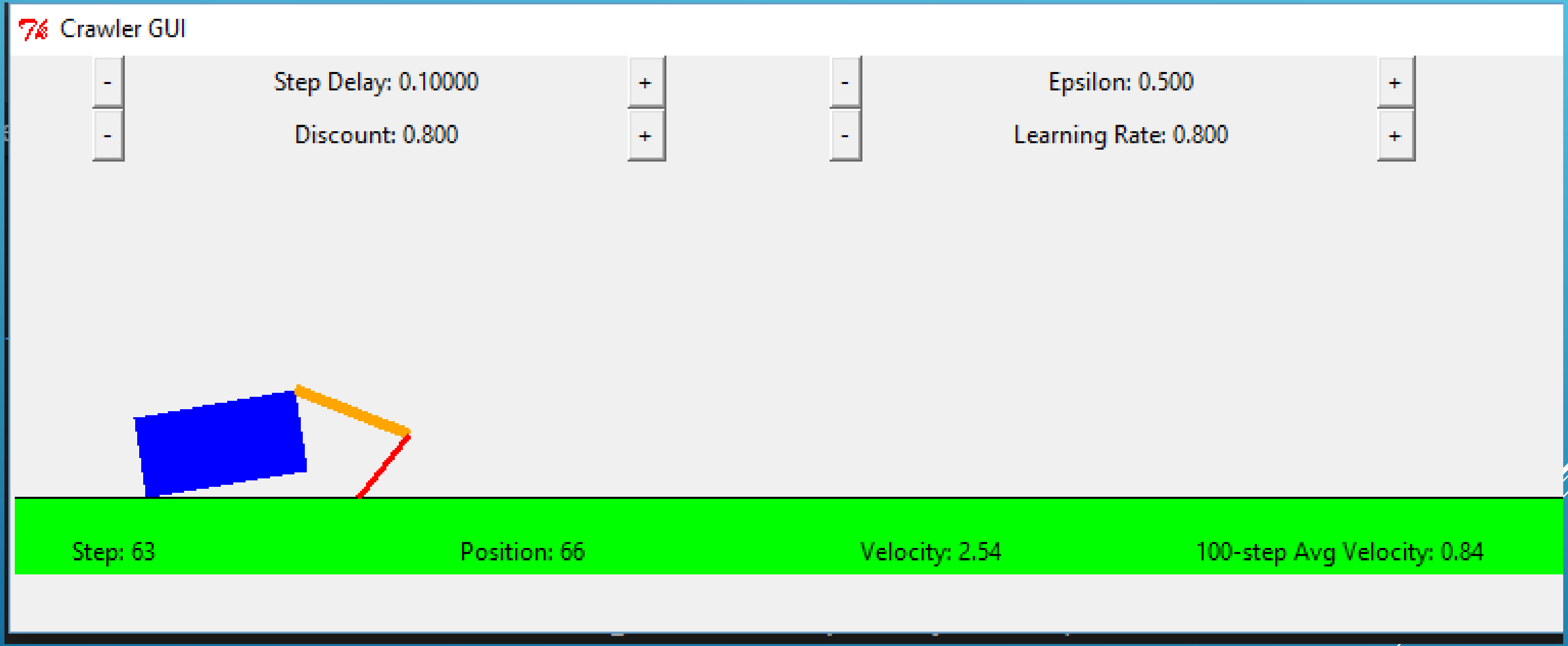
Q-LEARNING EXAMPLE: GRIDWORLD





VIDEO OF Q-LEARNING DEMO -- GRIDWORLD

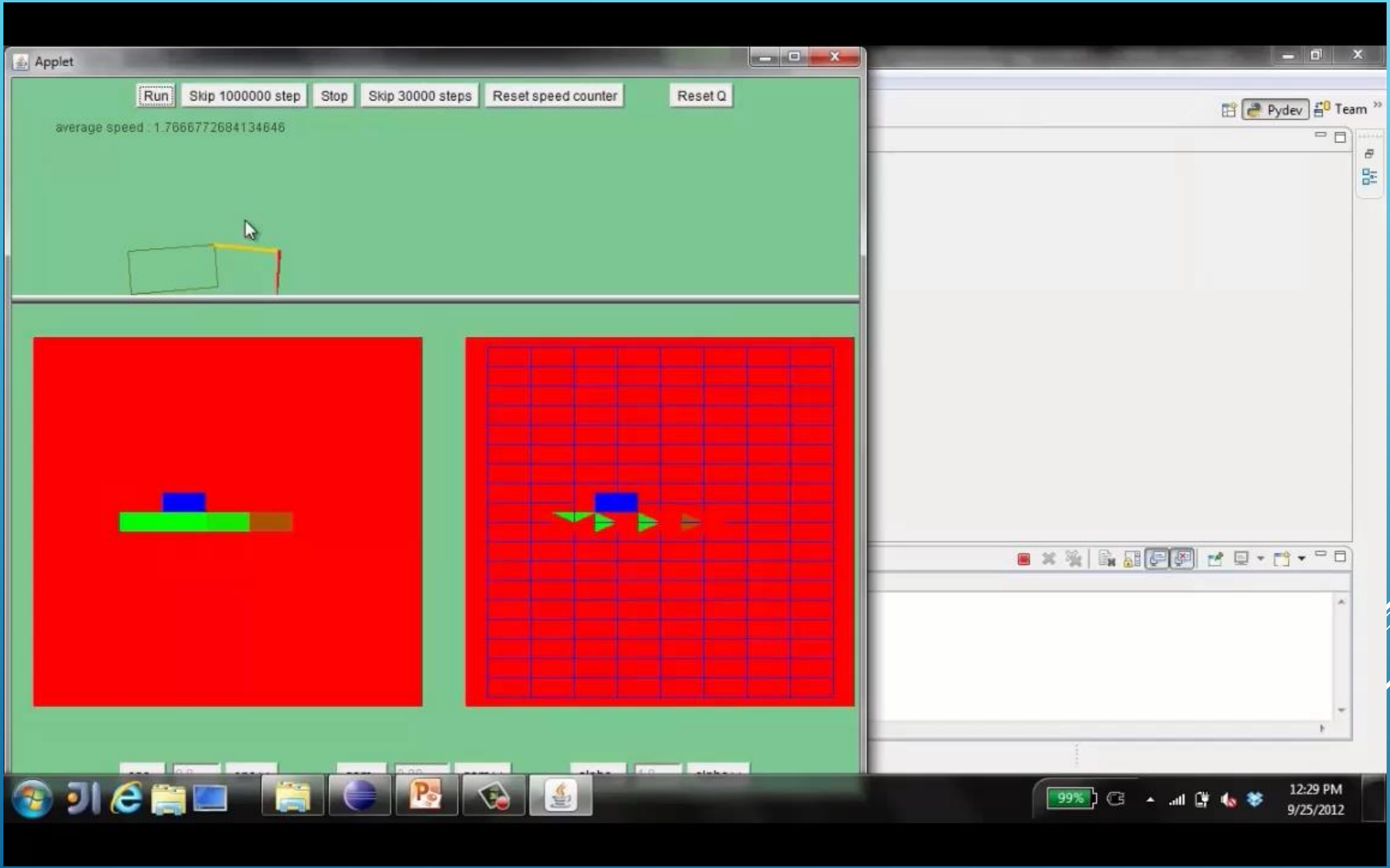
Q-LEARNING EXAMPLE: CRAWLER ROBOT



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

VIDEO 1 OF Q-LEARNING DEMO -- CRAWLER

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



VIDEO 2 OF Q-LEARNING DEMO -- CRAWLER

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

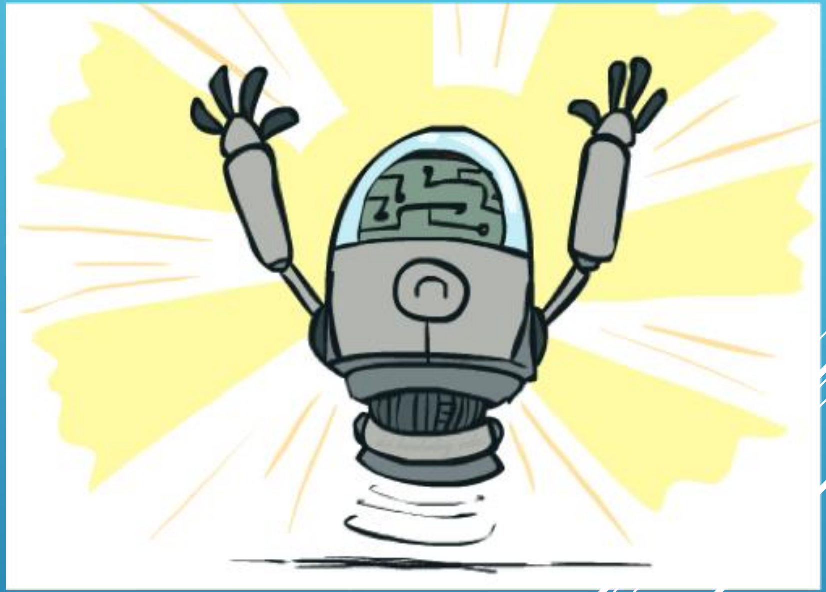
Q-LEARNING EXAMPLE: DISCOUNT EFFECT

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

Q-LEARNING PROPERTIES

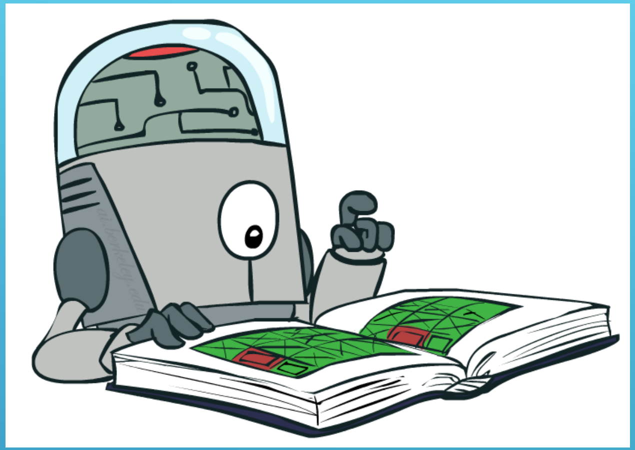
- Amazing result: Q-learning converges to optimal policy -- even if you're acting sub-optimally!
- This is called ***off-policy learning***
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)

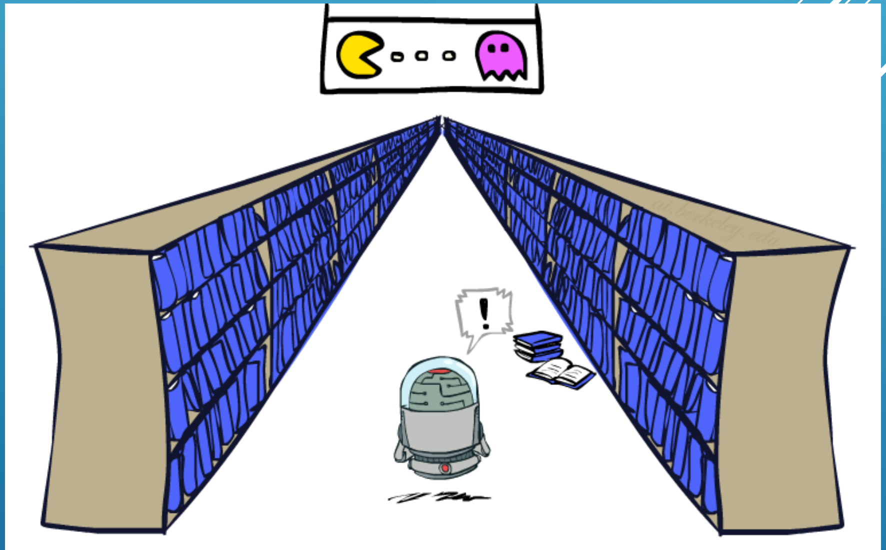


Credit: CS188 – Intro to AI, UC Berkeley, ai.berkely.edu

GENERALIZING ACROSS STATES

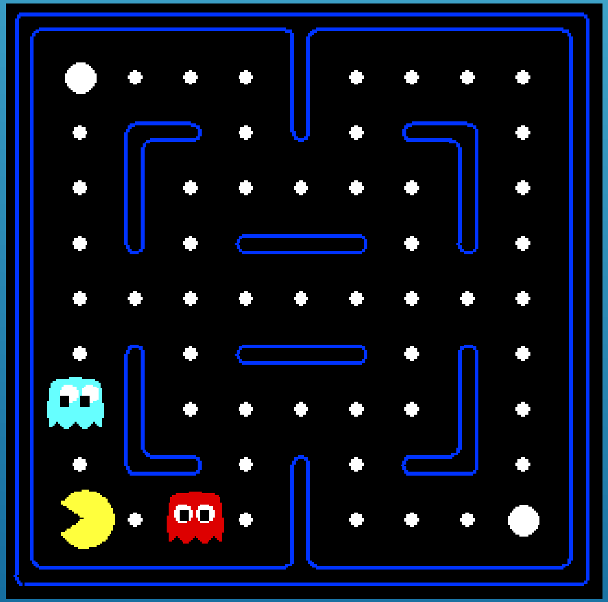
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations



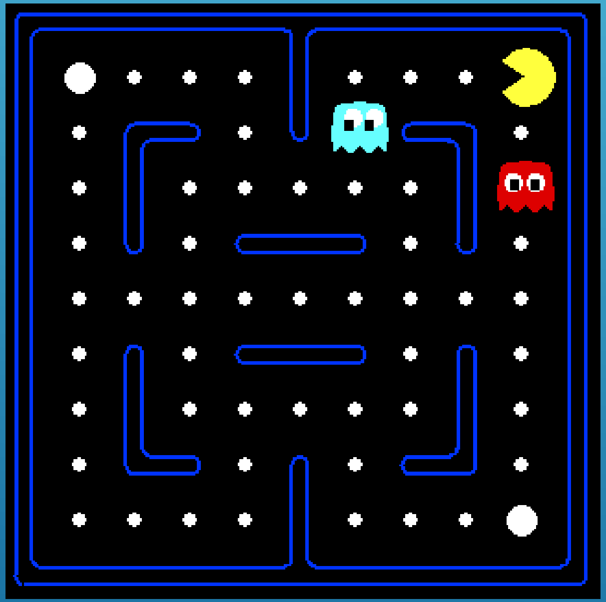


EXAMPLE: PACMAN

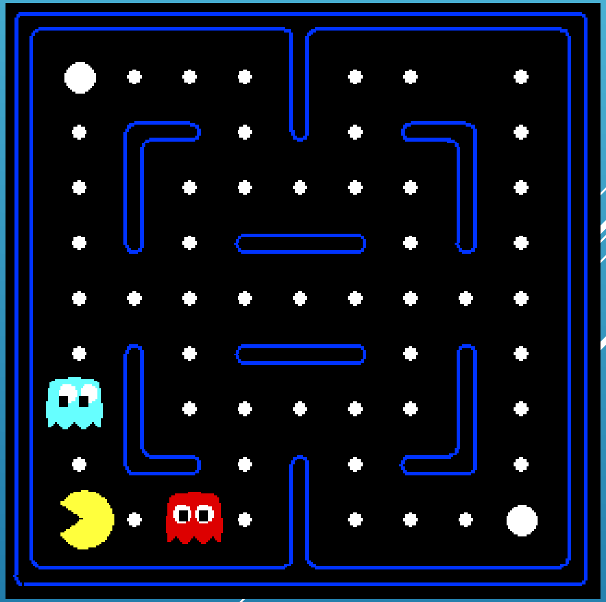
Let's say we discover through experience that this state is bad:



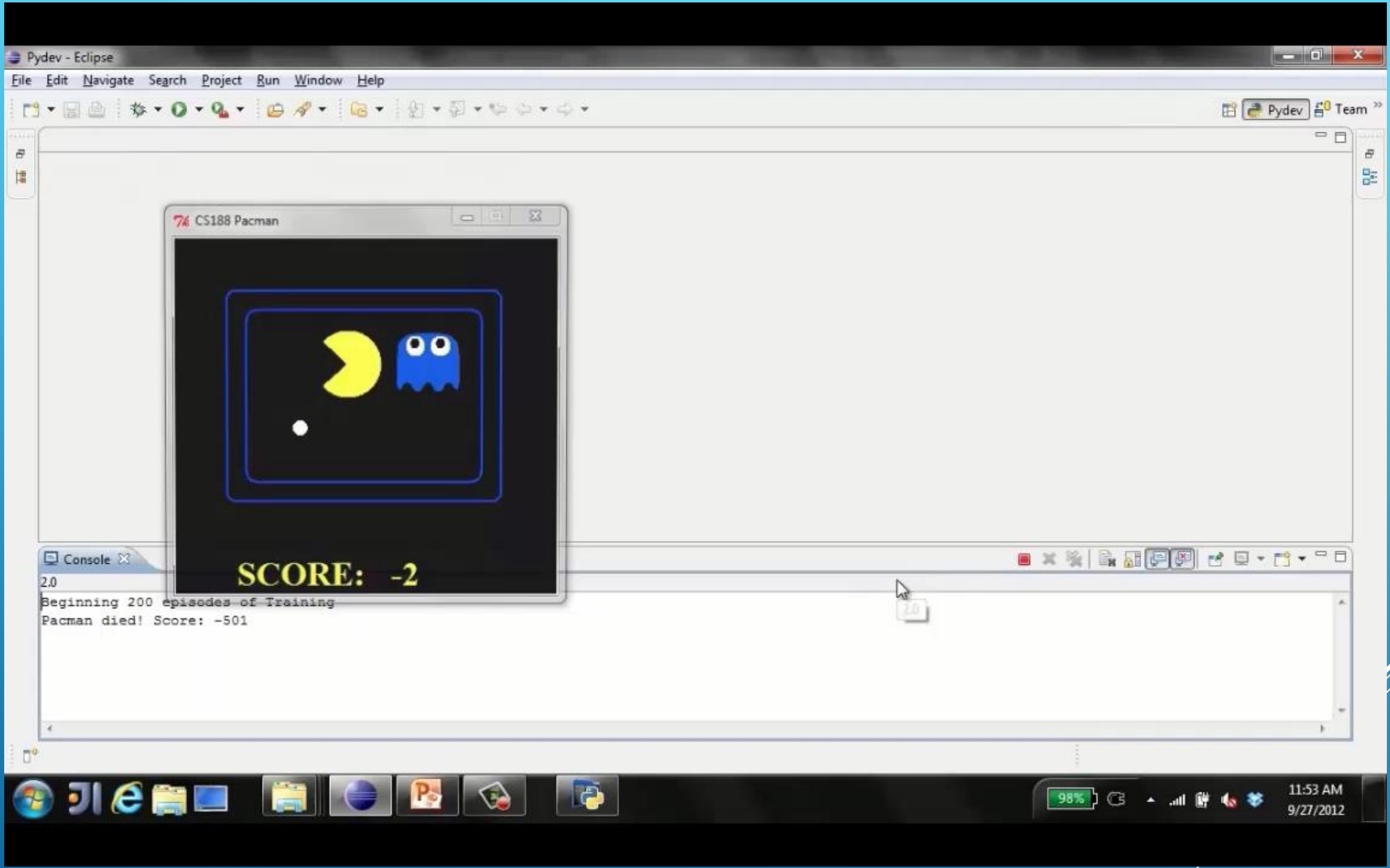
In naïve q-learning, we know nothing about this state:



Or even this one!

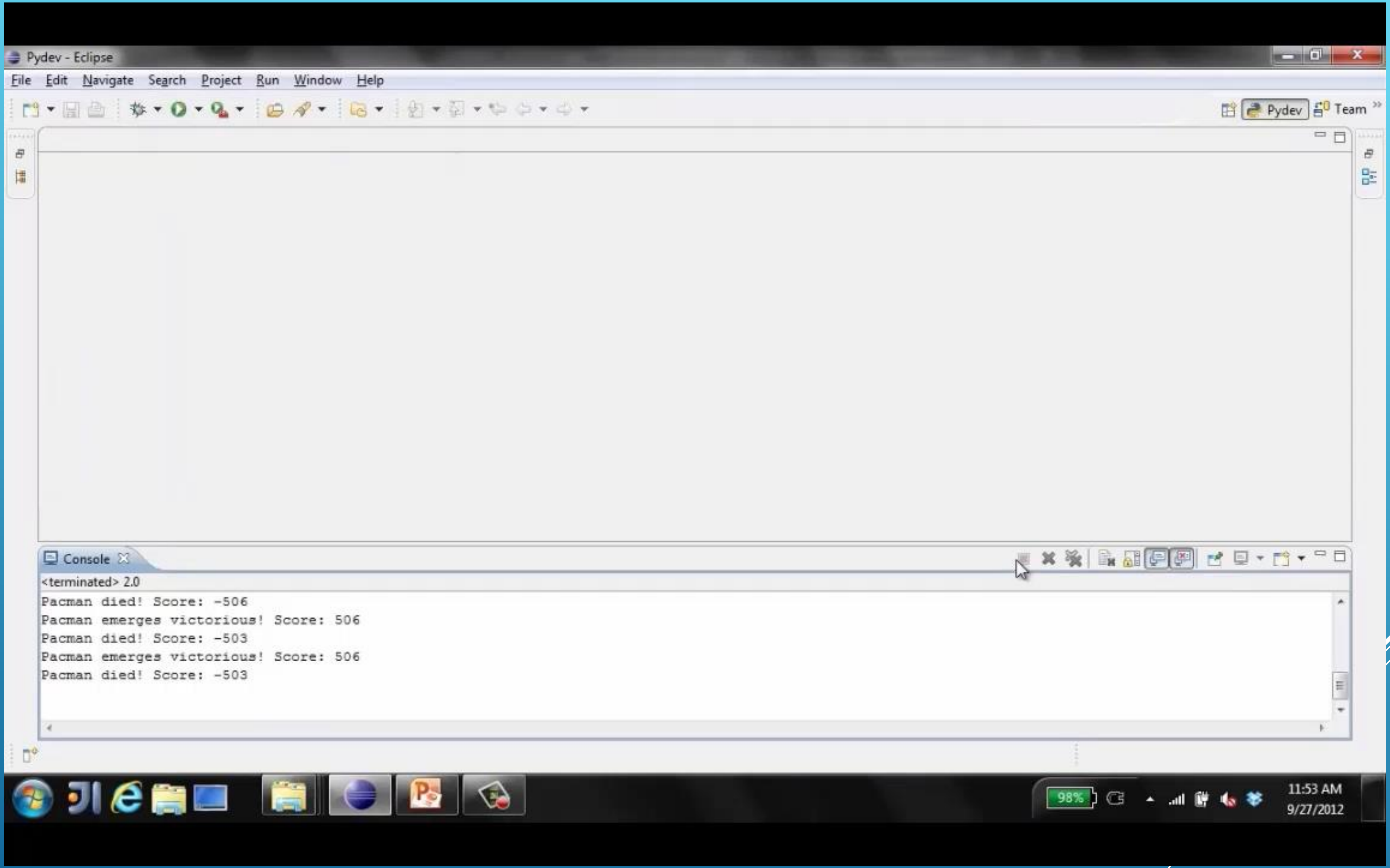


Credit: CS188 – Intro to AI, UC Berkeley, ai.berkely.edu



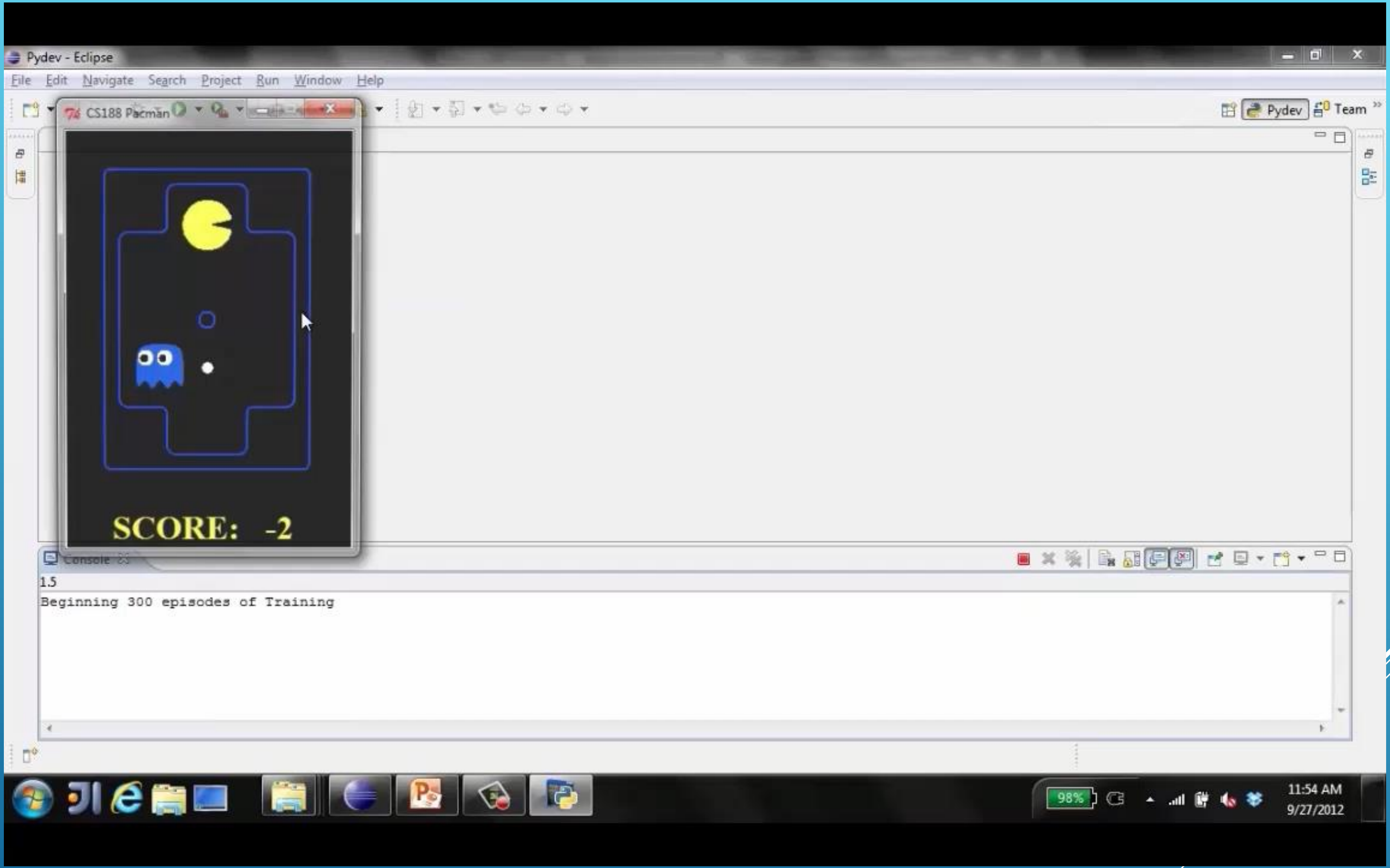
TINY PACMAN DEMO 1

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



TINY PACMAN DEMO 2

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

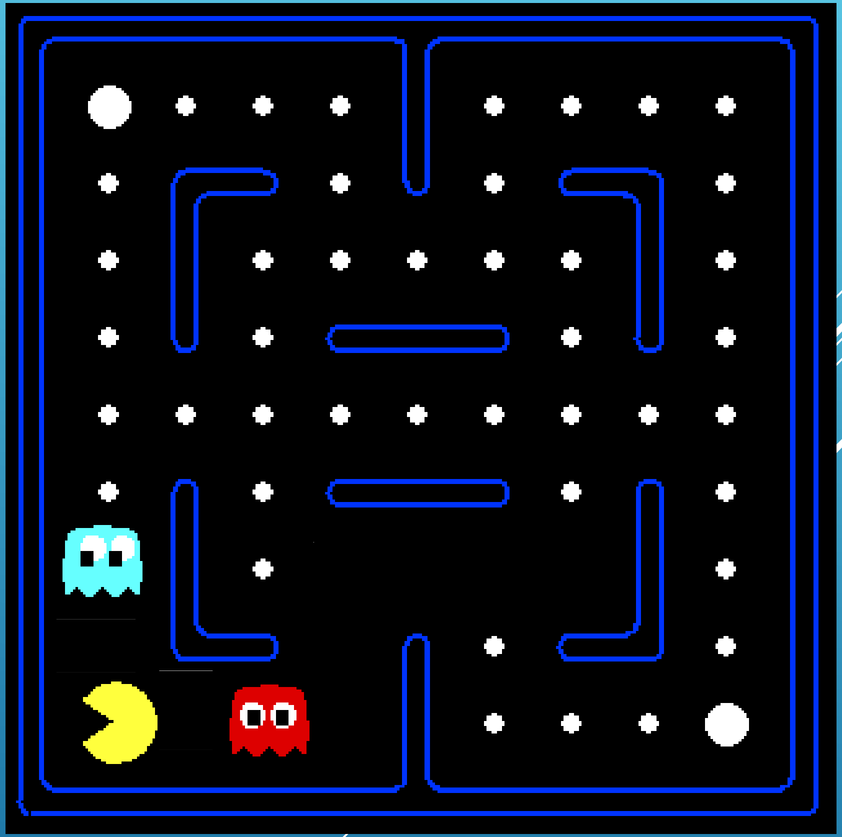


TINY PACMAN DEMO 3

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

FEATURE-BASED REPRESENTATIONS

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



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

LINEAR VALUE FUNCTIONS

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

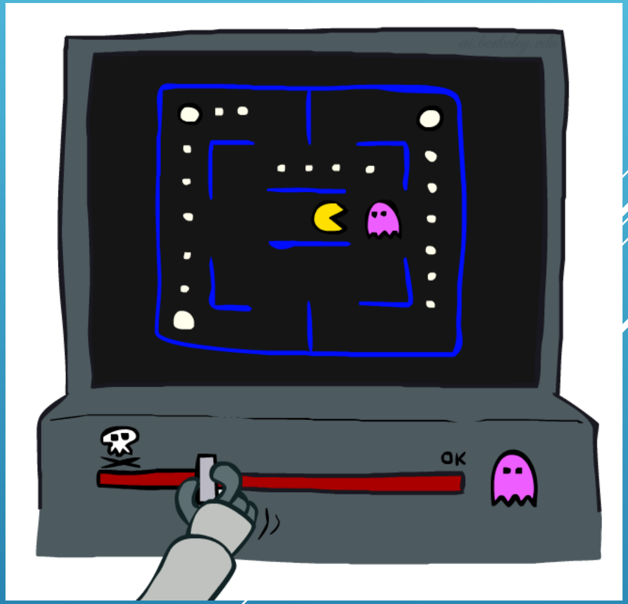
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!

APPROXIMATE Q-LEARNING

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with linear Q-functions:
  - transition = (s,a,r,s')
  - difference = $\left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[\text{difference}]$ ← **Exact Q's**
  - $\boldsymbol{w_i} \leftarrow \boldsymbol{w_i} + \alpha[\text{difference}]\, \mathrm{f}_i(s,a)$ ← **Approximate Q's**

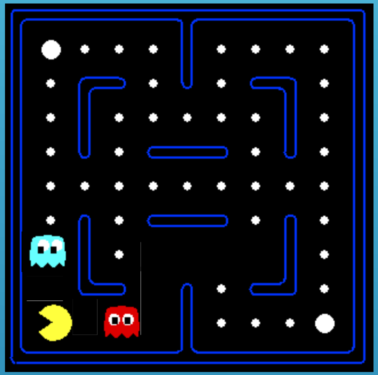


- Intuitive interpretation:
  - Adjust weights of active features.
  - E.g., if something unexpectedly bad happens, blame the features that were on i.e., prefer less all states with that state's features.
- Formal justification: online least squares

EXAMPLE: Q-PACMAN

$$Q(s,a) = 4.0\, f_{DOT}(s,a) - 1.0\, f_{GST}(s,a)$$

s



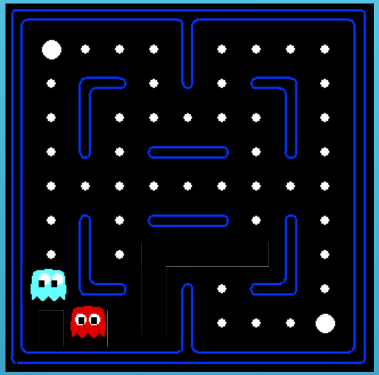
$$f_{DOT}(s, NORTH) = 0.5$$

$$f_{GST}(s, NORTH) = 1.0$$

$a = \text{NORTH}$

$r = -500$

s'



$$Q(s, NORTH) = +1$$

$$r + \gamma \max_{a'} Q(s', a') = -500 + 0$$

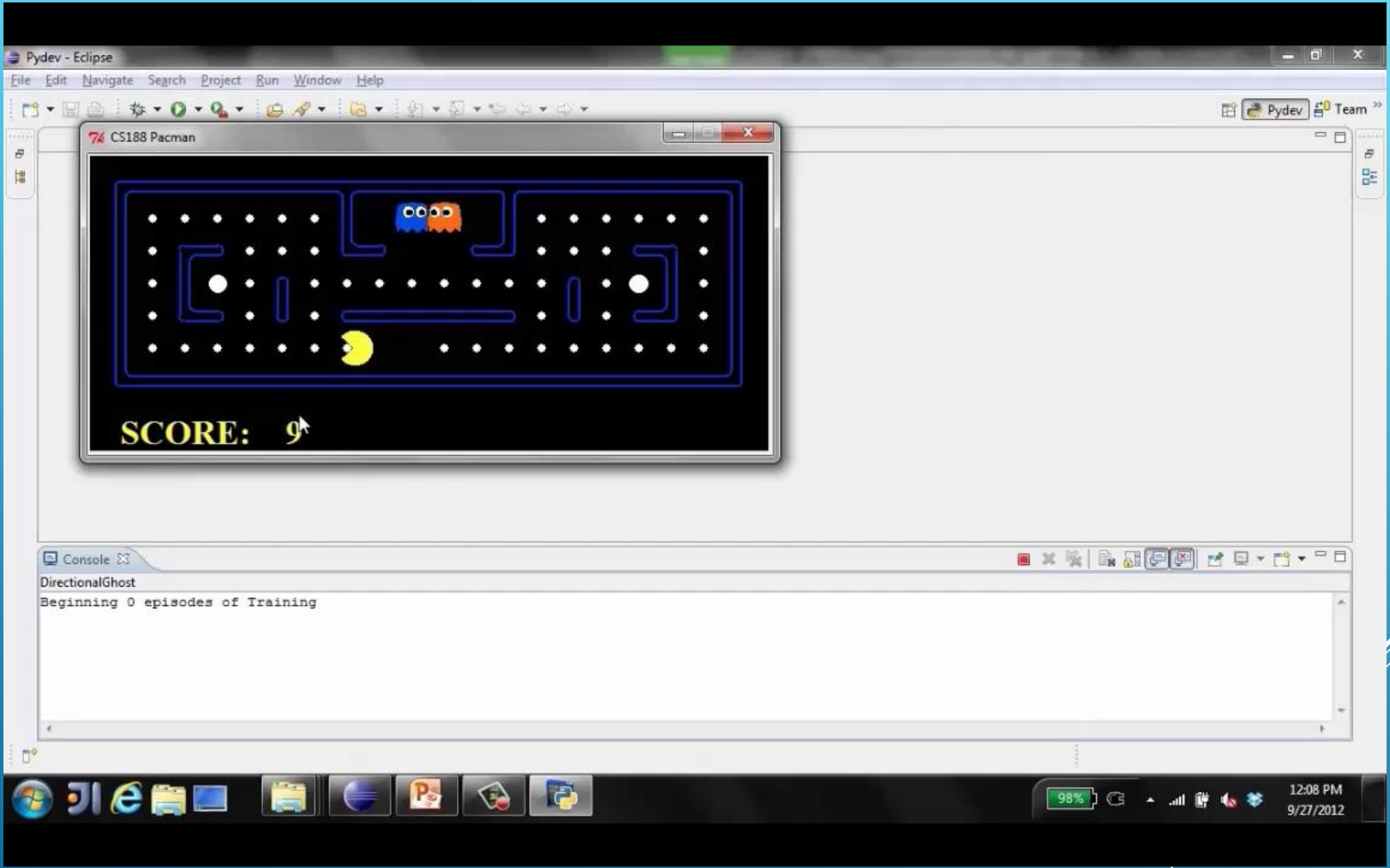
$$Q(s', \cdot) = 0$$

$$\text{difference} = -501$$

$$w_{DOT} \leftarrow 4.0 + \alpha[-501]0.5$$

$$w_{GST} \leftarrow -1.0 + \alpha[-501]1.0$$

$$Q(s,a) = 3.0\, f_{DOT}(s,a) - 3.0\, f_{GST}(s,a)$$



APPROXIMATE Q-LEARNING DEMO -- PACMAN

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

NEXT TIME: MORE RL

Possible Topics:

- **Credit Assignment**
- **Online Learning vs Offline learning**
  - **SARSA vs Q-Learning**
- **Hybrid Approaches:**
  - **Temporal Difference (TD) Value Learning**
  - **Dyna-Q**
- **Partially Observable Markov Decision Processes (POMDPs)**
- **Deep Q-Learning**

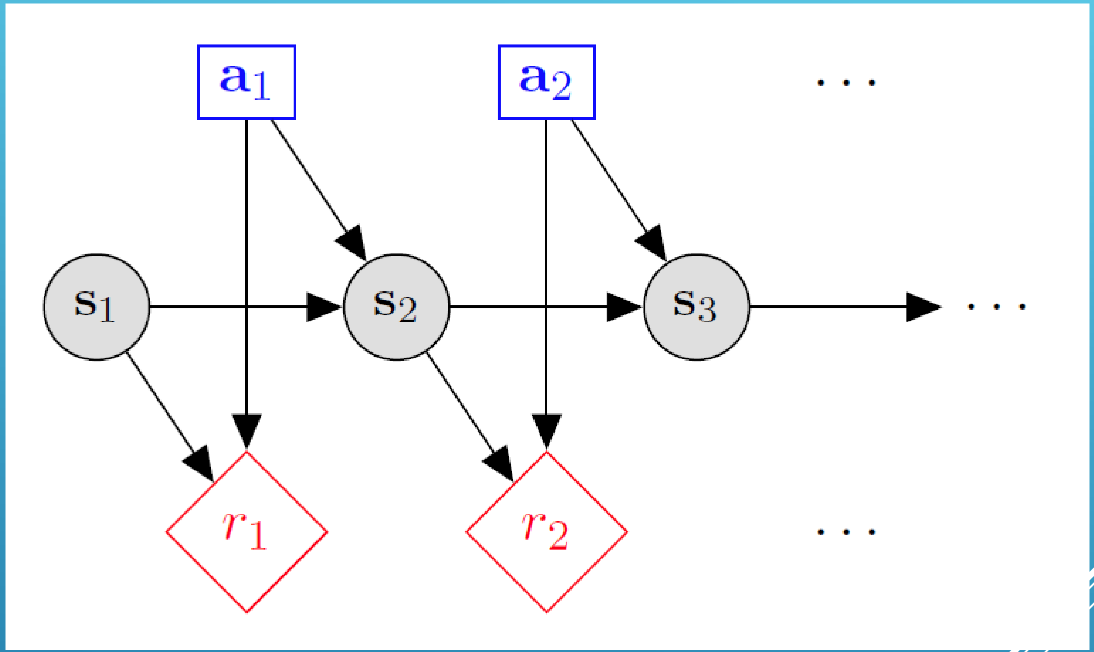
EXTRA SLIDES

MARKOV DECISION PROCESSES

- **States:** $s_1, \ldots, s_n$
- **Actions:** $a_1, \ldots, a_m$
- **Reward Function:**

  $r(s, a, s') \in R$
- **Transition model:**

  $T(s,a,s') = P(s'|s, a)$
- **Discount factor:** $\gamma \in [0, 1]$



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