INTRODUCTION TO Q-LEARNING

Scott O'Hara
Metrowest Developers Machine Learning Group

5/13/2020

REFERENCES

CS188 - Introduction to Artificial Intelligence course at University of California, Berkeley:

- http://ai.berkeley.edu/home.html
- http://gamescrafters.berkeley.edu/~cs188/sp20/

CS181 - Machine Learning course at Harvard University:

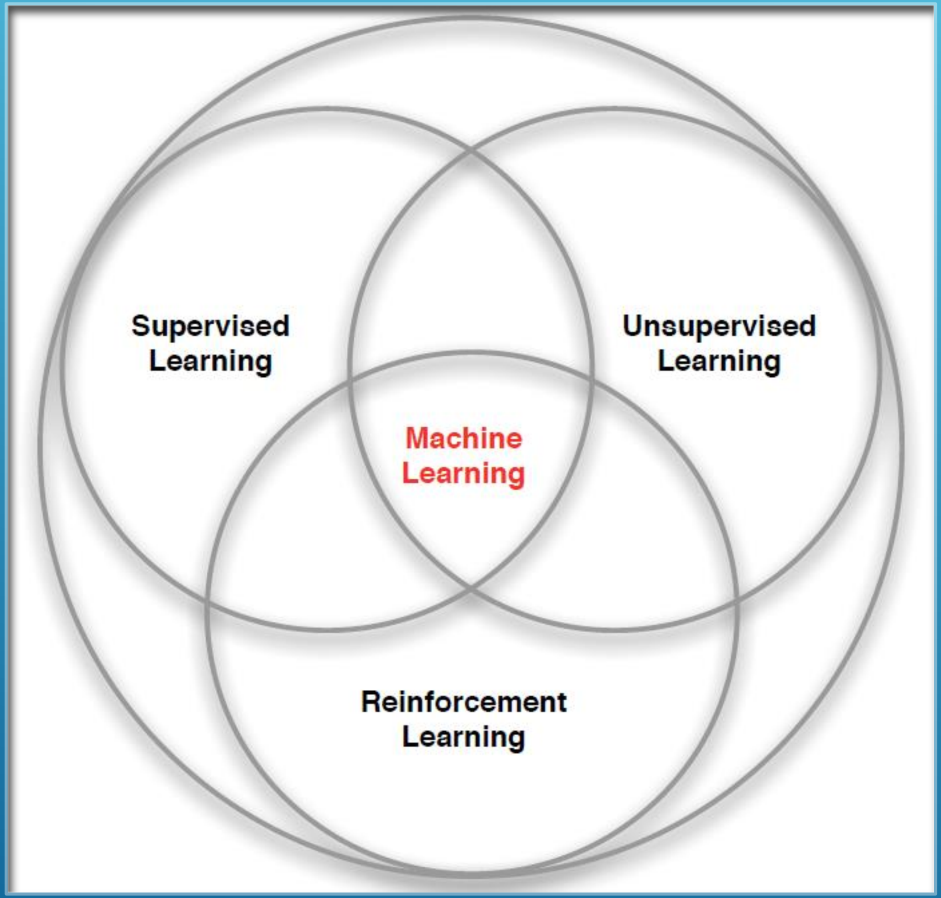
- *Lectures and notes from multiple offerings of CS181: Spring 2009, 2011, 2014 and 2017.*

"Demystifying Deep Reinforcement Learning," 2015, Tambet Matiisen

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

Reinforcement learning: an introduction *R. S. Sutton and A. G. Barto, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.*

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.

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

THE MARKOV DECISION PROCESS

- The ***Markov Decision Process*** (MDP) provides a mathematical framework for reinforcement learning.

- Markov decision processes use **probability** to model **uncertainty** about the domain.

- Markov decision use **utility** to model an agent's **objectives**. The higher the utility, the "happier" your agent is.

- MDP algorithms discover an **optimal decision policy (π)** specifying how the agent should act in all possible states in order to maximize its expected utility.

MARKOV DECISION PROCESSES

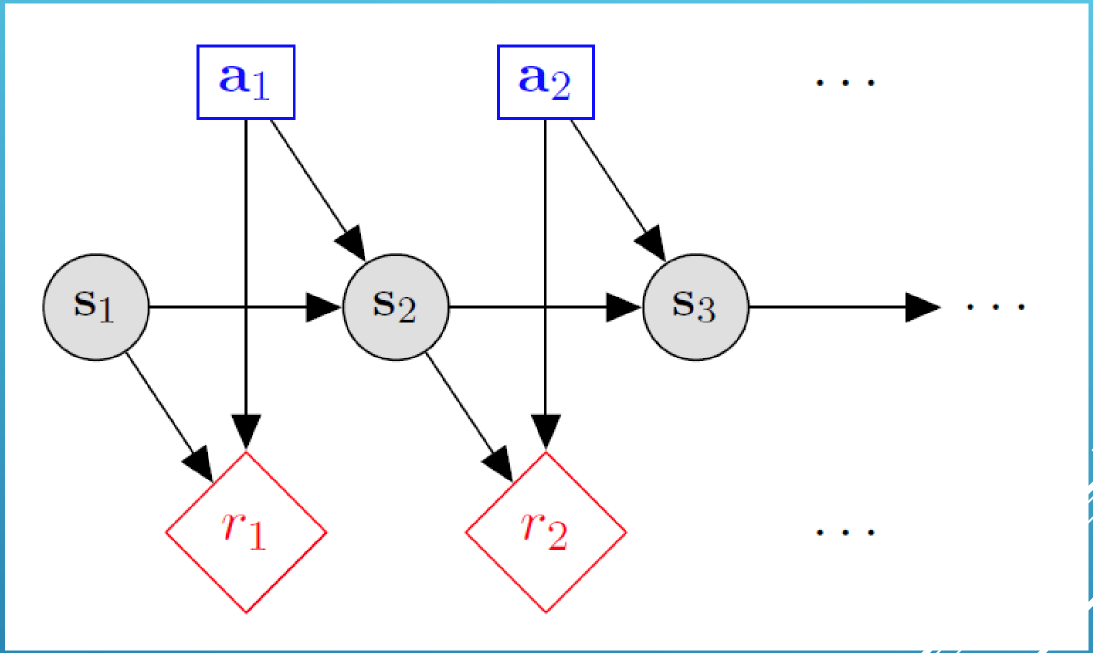
- **States:** $s_1, \ldots, s_n$
- **Actions:** $a_1, \ldots, a_m$
- **Reward model:**

$$R(s, a, s') \in R$$

- **Transition model:**

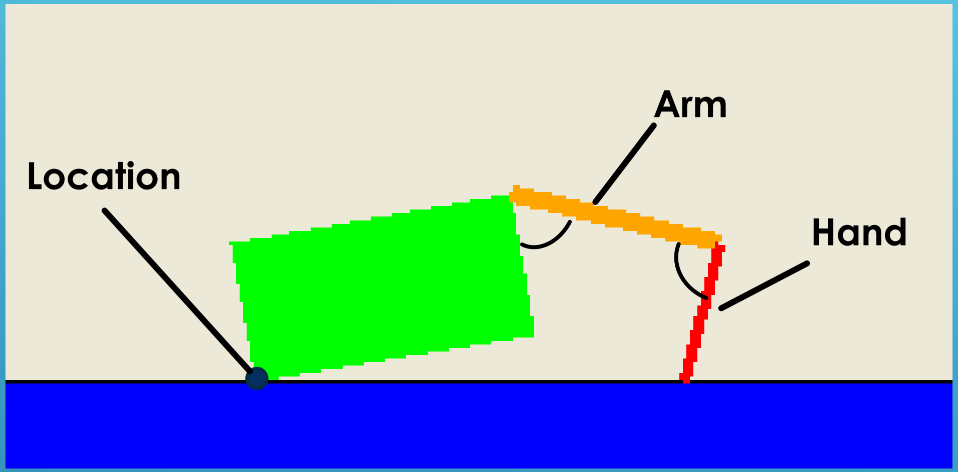
$$T(s, a, s') = P(s'|s, a)$$

- **Discount factor:** $\gamma \in [0, 1]$



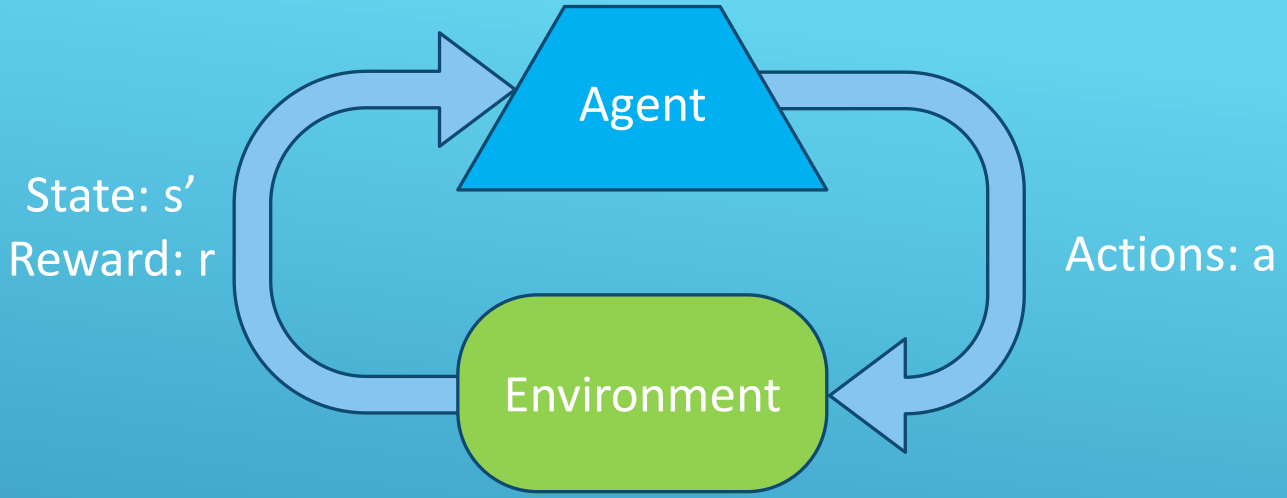
<https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=855934986> [accessed 11 September 2018]

APPLICATION: CRAWLER ROBOT



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

THE REINFORCEMENT LEARNING PROBLEM



- Agent must learn to act to maximize expected rewards.
- Agent knows the current state s, takes an action ***a***, receives a reward ***r*** and observes the next state ***s’***.

$$S_0, A_0, R_0, S_1, A_1, R_2, S_2, A_2, R_2, \dots, S_n, A_n, R_n, S_T$$

- Agent has **no access** to the reward model ***r(s,a,s’)*** or the transition model ***T(s,a,s’)***.

REINFORCEMENT LEARNING ALGORITHMS BASED ON THE MARKOV DECISION PROCESS

- With MDPs, the reward and transition models together are simply called **the model.**
- Reinforcement learning algorithms can be classified by whether the agent has a model the environment.
  - Algorithms with **Distribution Models**
  - **Model-free** Algorithms
  - Algorithms with **Sample-based Models**

ALGORITHMS WITH DISTRIBUTION MODELS

- Probability distributions are known for both the reward and transition models.
- **Value iteration algorithms** build a value function using dynamic programming.
- **Policy iteration algorithms:** build a policy directly by using dynamic programming.
- See my presentation slides from 2/26/2020 entitled **"Model-based Reinforcement Learning"** at:

https://github.com/MetrowestBostonDevelopersMLGroup/MeetingPresentations/tree/master/2020

MODEL-FREE RL ALGORITHMS

- Model-free RL algorithms have knowledge of the states and actions but …
- have no knowledge of the reward and transition models of the problem space and…
- make no attempt to learn to learn these models.
- Instead, attempts to learn the **Q-Values** of every state-action pairs i.e., the expected payoff of a particular action taken in a given state.
- Examples of model-free RL algorithms include: **Q-Learning**, Monte-Carlo, SARSA, Expected SARSA, Deep Q-learning, etc.

ALGORITHMS WITH SAMPLE-BASED MODELS

- Between full-knowledge RL algorithms with a distribution model and model-free RL algorithms there are RL algorithms with sample-based models.

- Sample-based RL algorithms construct approximate transition and reward models by collecting statistics about transitions and rewards from the environment.

- These algorithms exploit their approximate models by using them to periodically improve their value functions or policies off-line.

- Examples of RL algorithms with sample-based models include: **Dyna-Q, real-time dynamic programming (RTDP), Prioritized Sweeping, Expectimax, etc.**

MODEL-BASED RL PROS AND CONS

- **Pros:**
  - Makes greater use of experience.
  - Solves the model optimally given enough experience.

- **Cons:**
  - Requires a computationally more expensive solution procedure.
  - Requires the model to be small enough to solve due to combinatorial issues.

MODEL-FREE RL PROS AND CONS

- **Pros:**
  - The solution procedure is much more efficient.
  - Can handle much larger MDPs.

- **Cons:**
  - Learns more slowly.
  - Does not learn as much as a model-based RL in a single training episode since it doesn’t learn the transition and reward models.

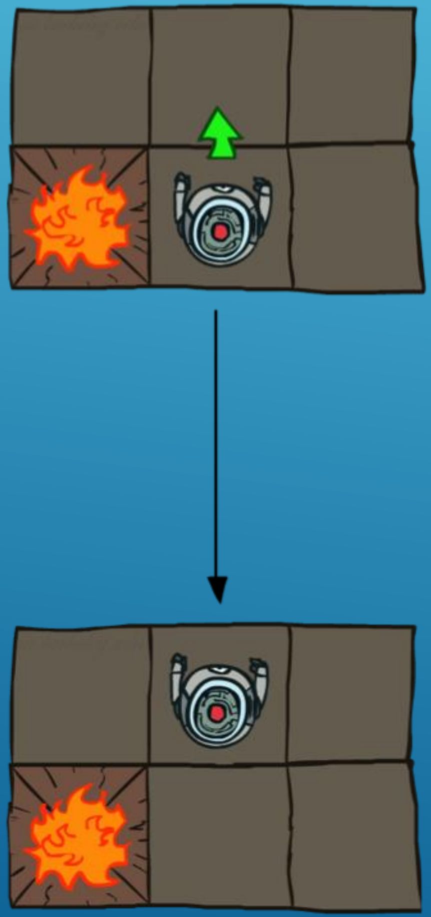
MODEL-FREE → SAMPLE-BASED MODELS→ DISTRIBUTION MODELS

- Generally, these algorithms lie along a continuum from model-free algorithms to model-based.
- Different algorithms represent different trade-offs with respect to:
  - computational complexity
  - model accuracy
  - execution speed
  - memory usage
  - etc.

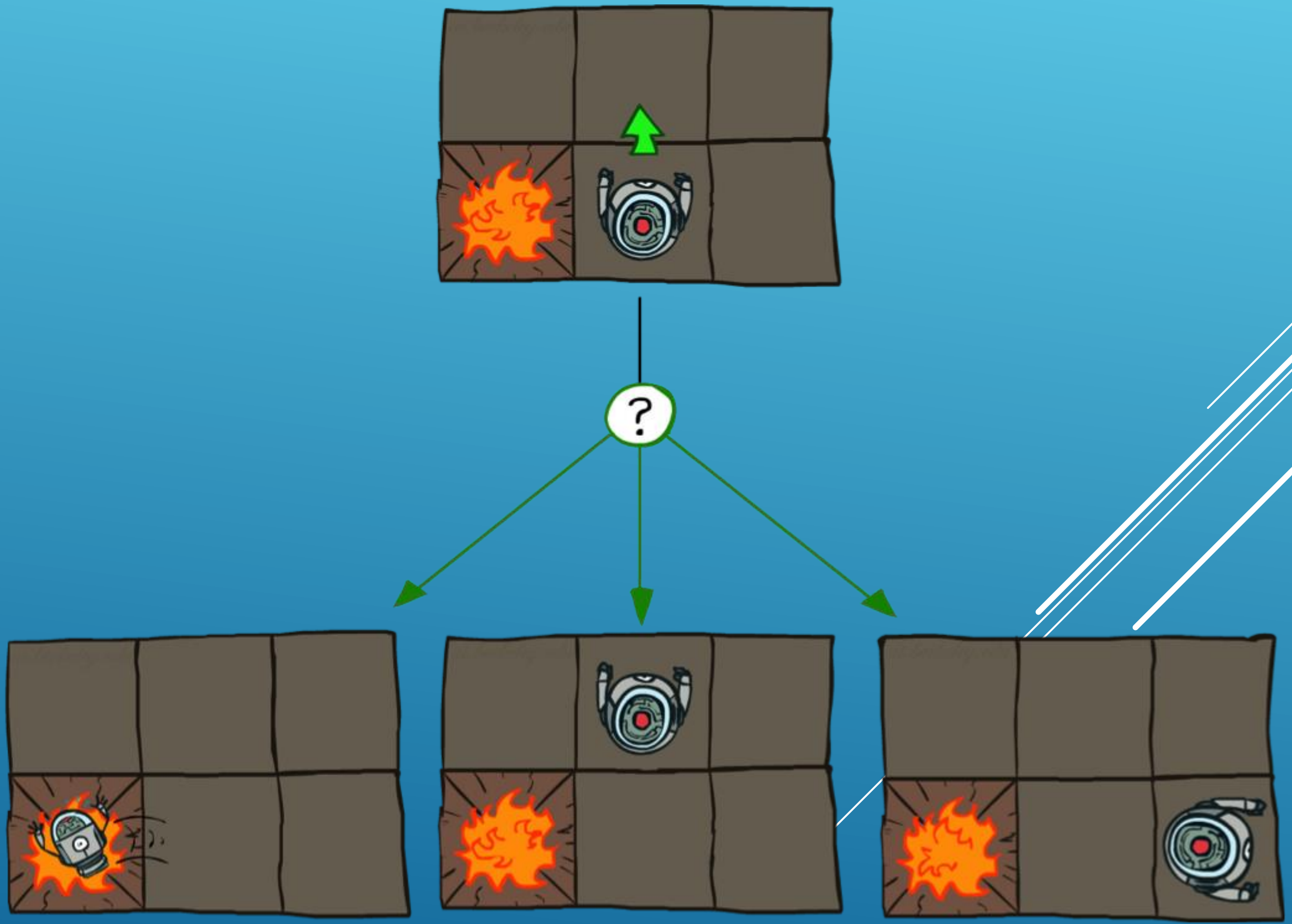
Q-LEARNING

EXAMPLE: GRID WORLD

Deterministic Grid World

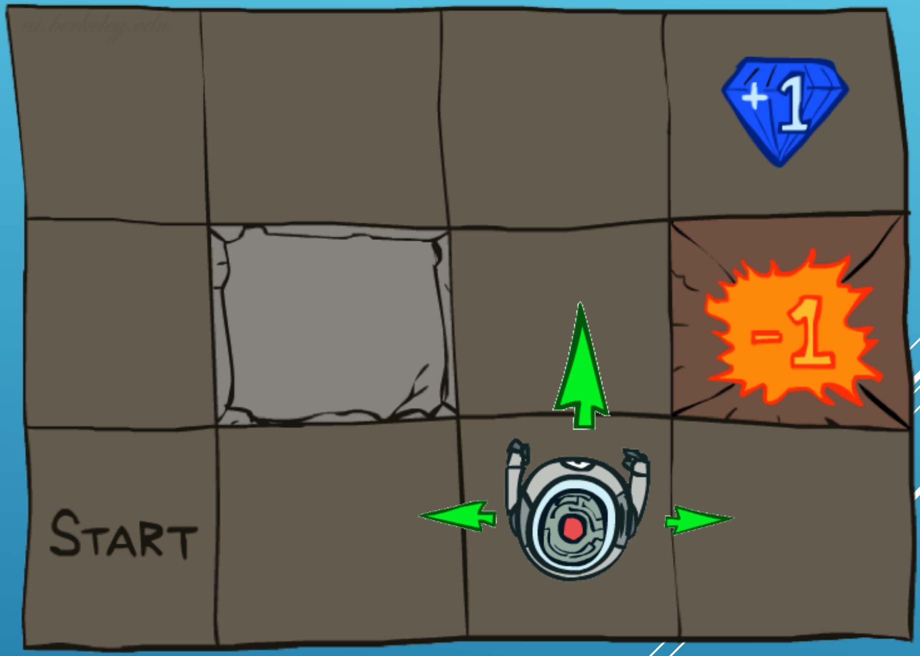


Stochastic Grid World



EXAMPLE: GRID WORLD

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards



Q-LEARNING EXAMPLE: GRIDWORLD

- **States: <x, y>** locations
- **Actions:** move **north, south, east** or **west.**
- **Reward model:**

  +1 if robot moves to <3, 2>

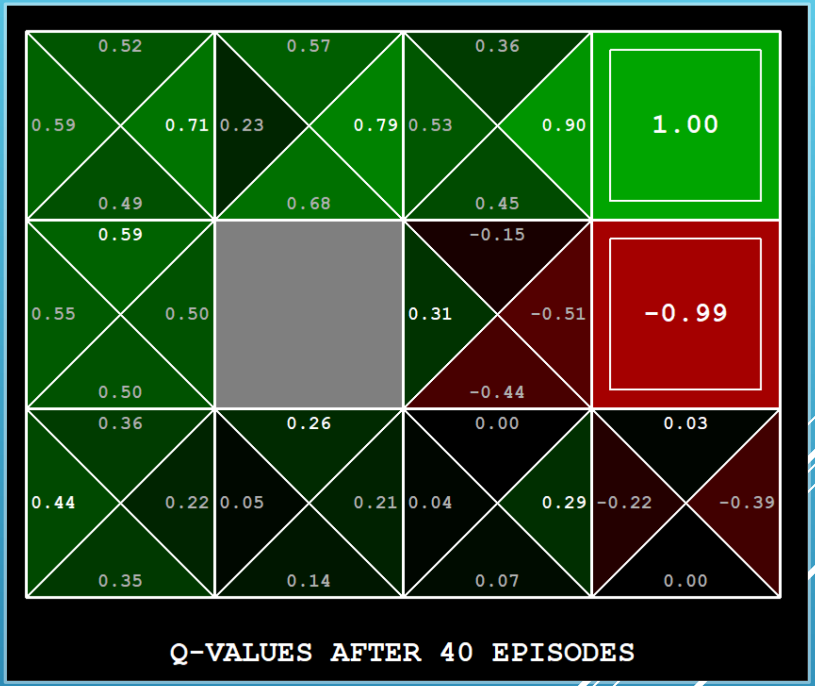
  -1 if robot moves <3, 1>

  otherwise, get "living reward".
- **Transition model:**

  n – probability of unintended (noisy) action.

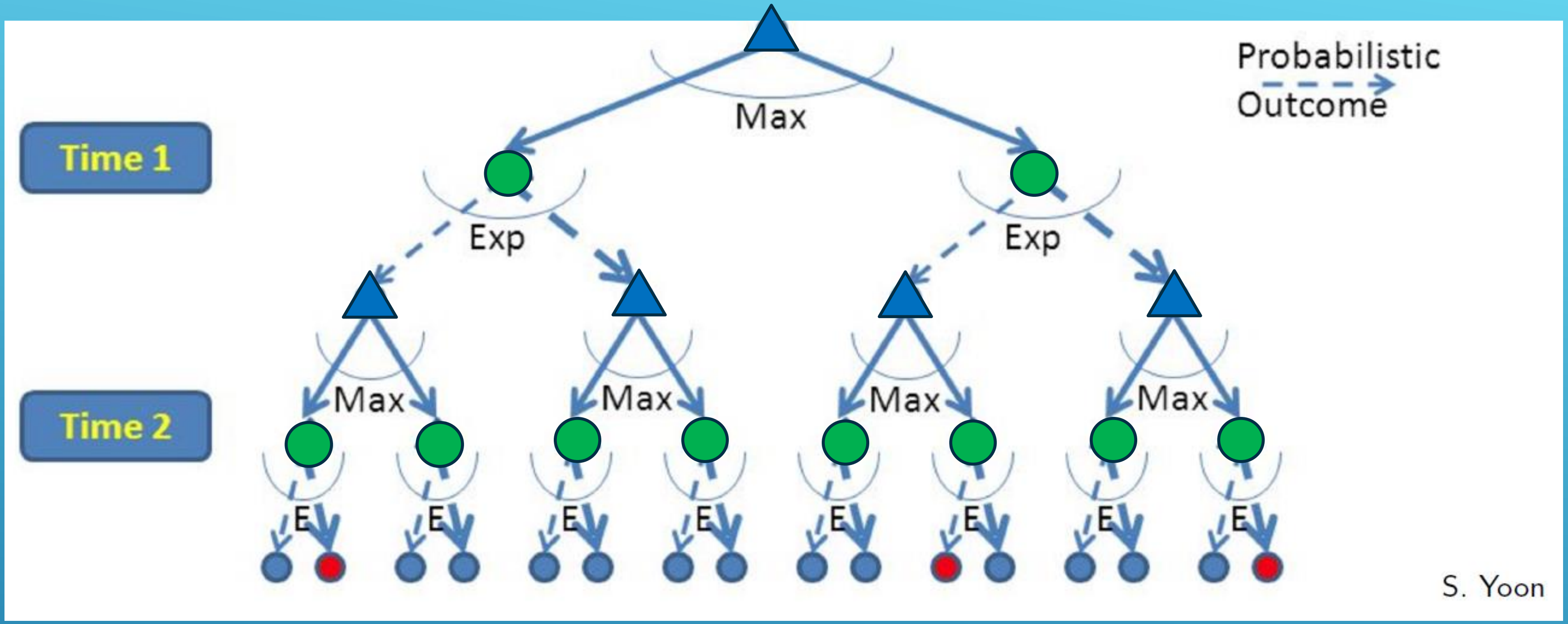
  1-n probability of intended action.

  stay put if you move into a wall.



DEMO:
# run manually using arrows keys (-m)
python gridworld.py -w 200 -k 40 -s 0.2 -a q -d 1.0 -m

RL IS LIKE A GAME AGAINST NATURE



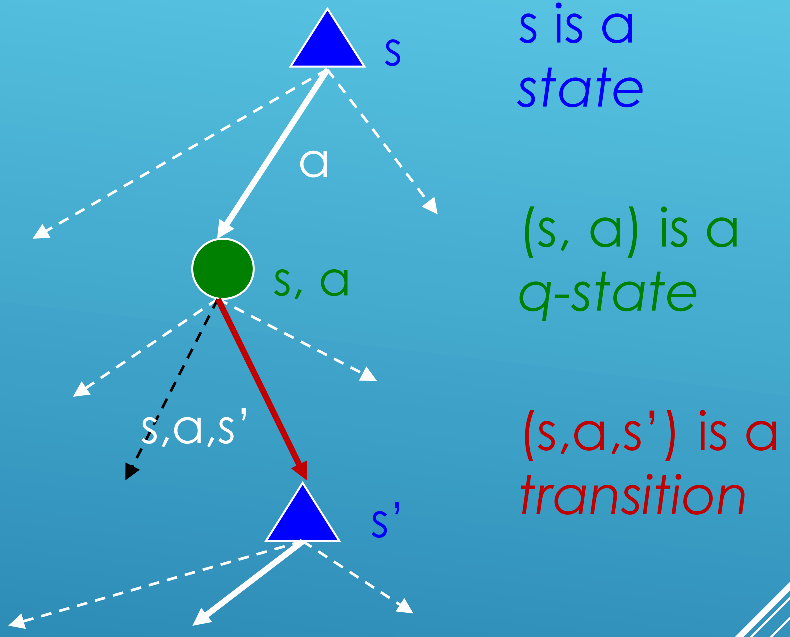
- Reinforcement learning is like a game-playing algorithm.
- Nodes where you move are called **states**: S (▲)
- Nodes where nature "moves" are called **Q-states**: <S,A> (●)

QUANTITIES TO OPTIMIZE

- The value (utility) of a **state s:**
  V(s) = expected utility starting in s and acting optimally thereafter.

- The value (utility) of a **q-state (s,a):**
  Q(s,a) = expected utility when taking action a from state s and acting optimally thereafter.

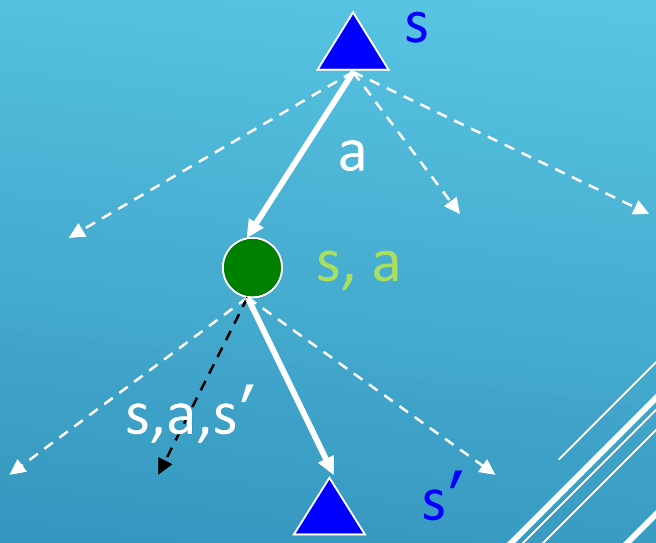
- The policy π:
  π(a|s) = probability of action a from state s



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

THE BELLMAN EQUATIONS

- The Bellman Value Equations define a relationship, which when satisfied gives the expected value for $\boldsymbol{V(s)}$ and $\boldsymbol{Q(s,a)}$ for a given policy $\boldsymbol{\pi}$.
- The Bellman Optimality 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 $\boldsymbol{\pi}^*$ 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.*



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

THE BELLMAN VALUE EQUATIONS

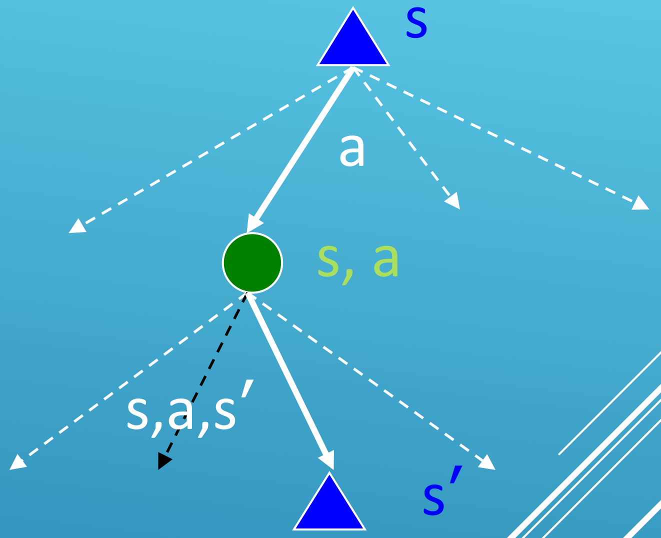
State Value Equation:

$$\boxed{V_\pi(s)} = \sum_{a'} \pi(a'|s') \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_\pi(s)]$$

Action Value Equation:

$$\boxed{Q_\pi(s,a)} = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s',a')\right]$$

Simplifying Assumption: rewards are fixed for (s, a, s')



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

BELLMAN OPTIMALITY EQUATION FOR V^*

State value equation:

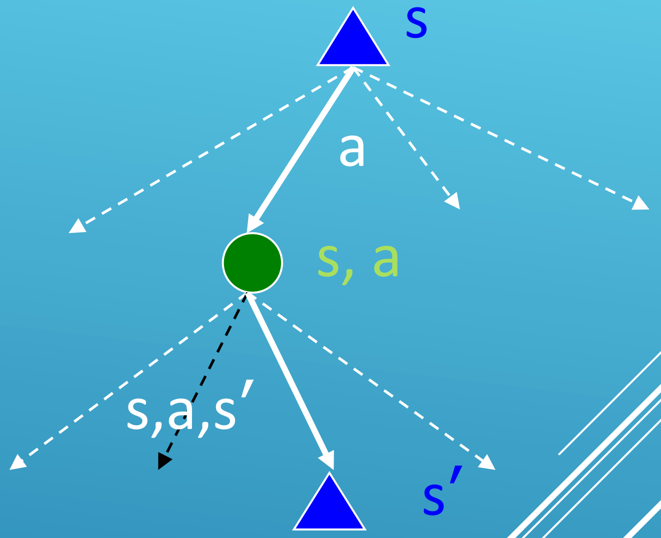
$$\boxed{V_\pi(s)} = \sum_{a'} \pi(a'|s') \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_\pi(s')]$$

Substitute optimal policy:

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

Optimality equation for V^*:

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



Simplifying Assumption: rewards are fixed for (s, a, s')

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

BELLMAN OPTIMALITY EQUATION FOR Q^*

Action value equation:

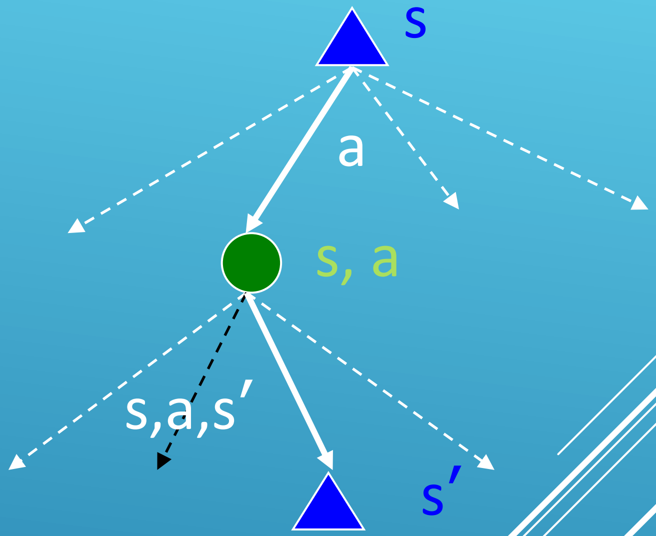
$$\boxed{Q_\pi(s,a)} = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s',a')\right]$$

Substitute optimal policy:

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

Optimality equation for Q^*:

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



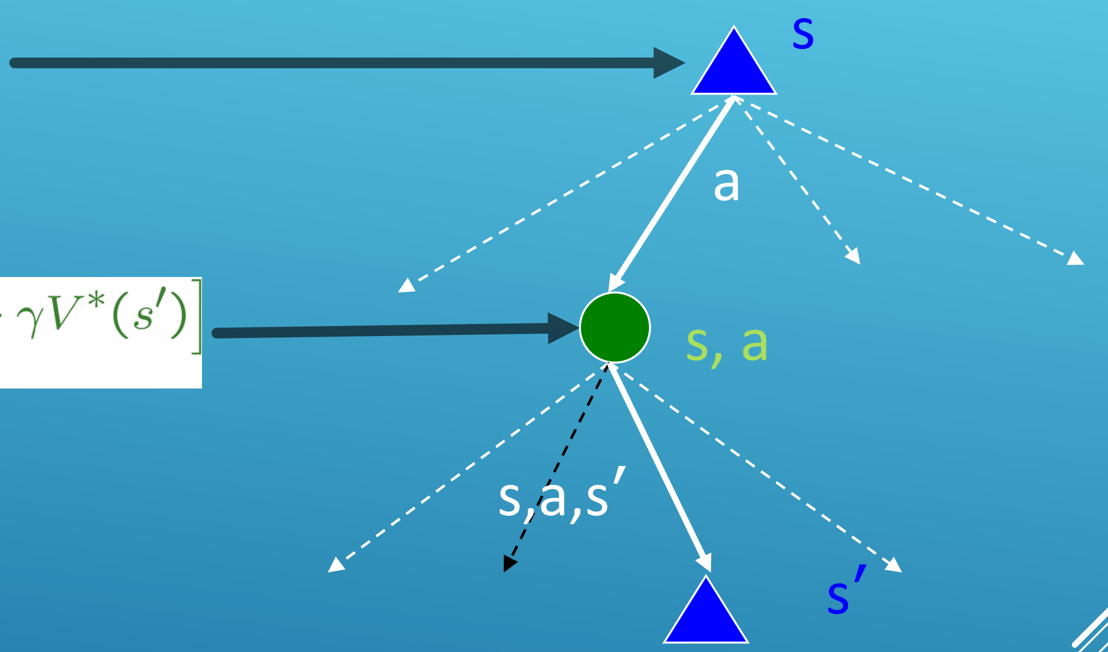
Simplifying Assumption: rewards are fixed for (s, a, s')

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

THE BELLMAN EQUATIONS

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

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



<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:
  - Value Iteration
  - Policy Iteration

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

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

$$\Big\Downarrow$$

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

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.

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

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

$$\Downarrow$$

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

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

Q-LEARNING UPDATE RULE (1)

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

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

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

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

Q-LEARNING UPDATE RULE (2)

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

$$\boldsymbol{Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \Big[\overbrace{r + \gamma \max_{a'} Q_k(s',a')}^{\textbf{target}} - Q_k(s,a) \Big]}$$

- 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 UPDATE RULE: AN ALTERNATE INTERPRETATION

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha\left[\overbrace{r + \gamma \max_{a'} Q_k(s',a')}^{\text{target}} - Q_k(s,a)\right]$$

$$\Downarrow$$

$$Q_{k+1}(s,a) \leftarrow \overbrace{(1-\alpha)Q_k(s,a)}^{\text{old}} + \overbrace{\alpha\left[r + \gamma \max_{a'} Q_k(s',a')\right]}^{\text{new}}$$

Q-LEARNING ALGORITHM

```
For each state s and action a:
        $Q(s,a) \leftarrow arbitrary\ value$
Observe initial state s
Repeat:
        Select and carry out an action a
        Receive reward r and observe new state s'
        With transition <s,a,r,s'>, update Q(s,a):
                $Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left[r + \gamma \max_{a'} Q_k(s',a')\right]$
        $s \leftarrow s'$
Until terminated
```

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.

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

CHOOSING AN ACTION: EXPLORATION VS EXPLOITATION

- How should an agent choose an action? An obvious answer is simply to follow the current policy. However, this is often not the best way to improve your model.
- **Exploit:** use your current model to maximize the expected utility now.
- **Explore:** choose an action that will help you improve your model.

Ε-GREEDY METHOD

- With probability 1 - ϵ:

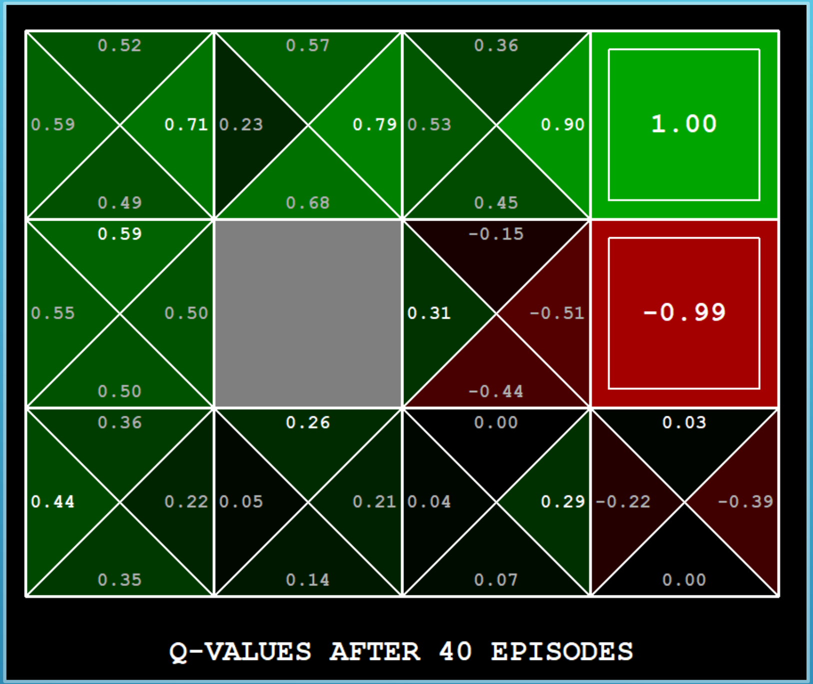
  select the action with the maximum value.

$$A_t = argmax\ Q_t(a)$$

- With probability ϵ:

  randomly select an action from all the actions with equal probability.

Q-LEARNING EXAMPLE: GRIDWORLD

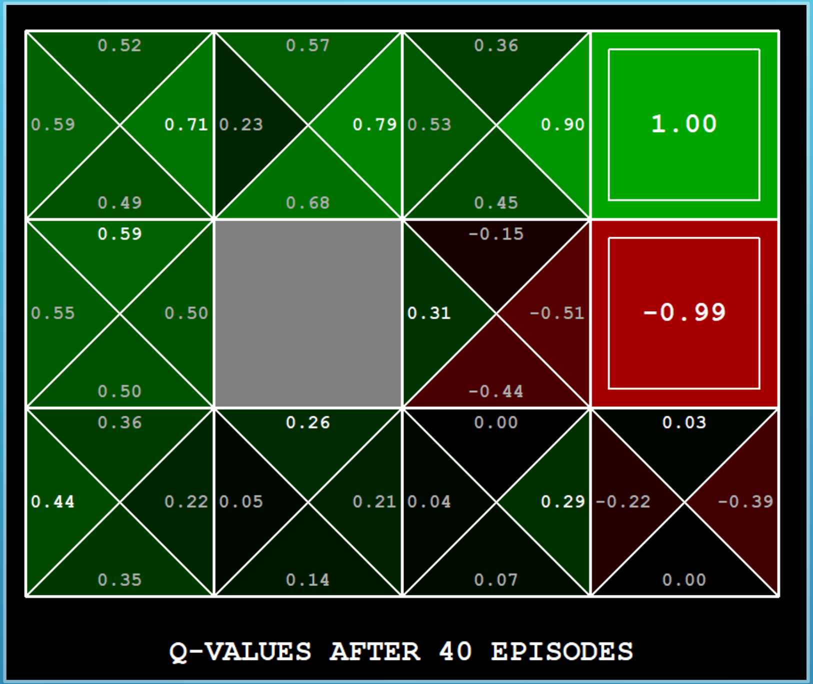


DEMO:

# run manually using arrows keys (-m)

```
python gridworld.py -w 200 -k 40 -s 0.2 -a q -d 1.0 -m
```

Q-LEARNING EXAMPLE: GRIDWORLD

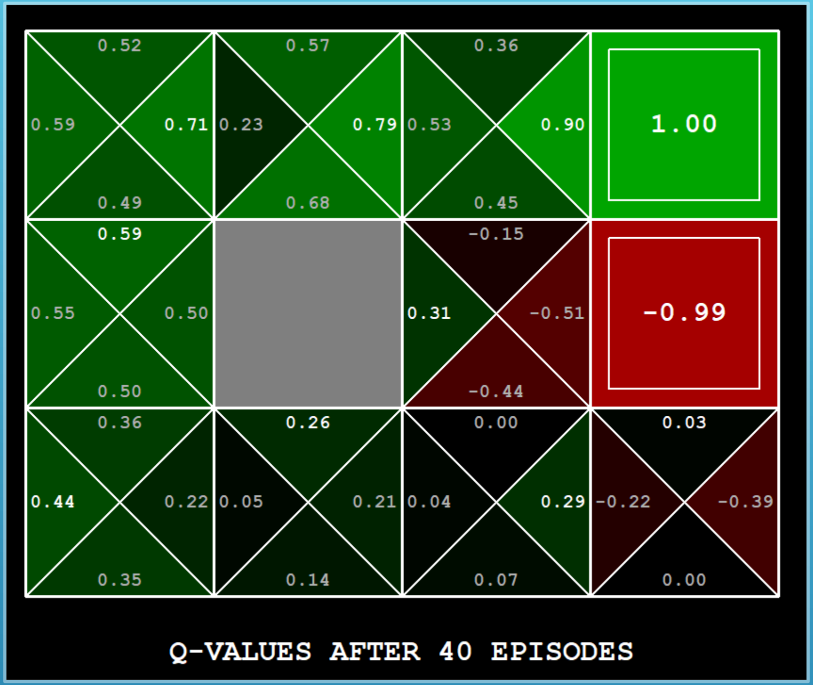


DEMO:

# run fast automatically (-s 2.0) with living reward 0.0

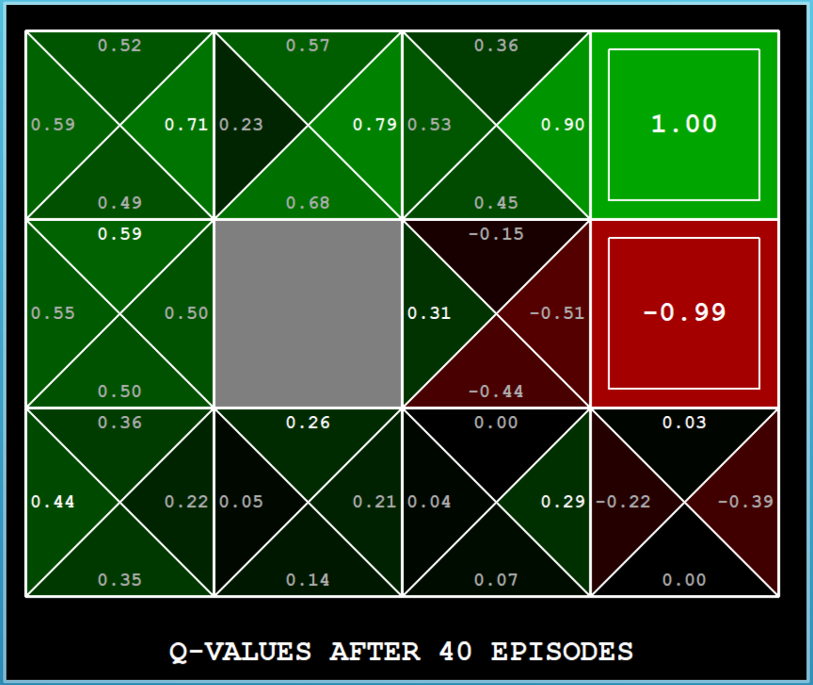
python gridworld.py -w 200 -k 40 -s 2.0 -a q -d 1.0

Q-LEARNING EXAMPLE: GRIDWORLD



DEMO:
# run fast automatically (-s 2.0) with living reward -0.1 (-r -0.1)
python gridworld.py -w 200 -k 100 -a q -d 1.0 -r -0.1

Q-LEARNING EXAMPLE: GRIDWORLD



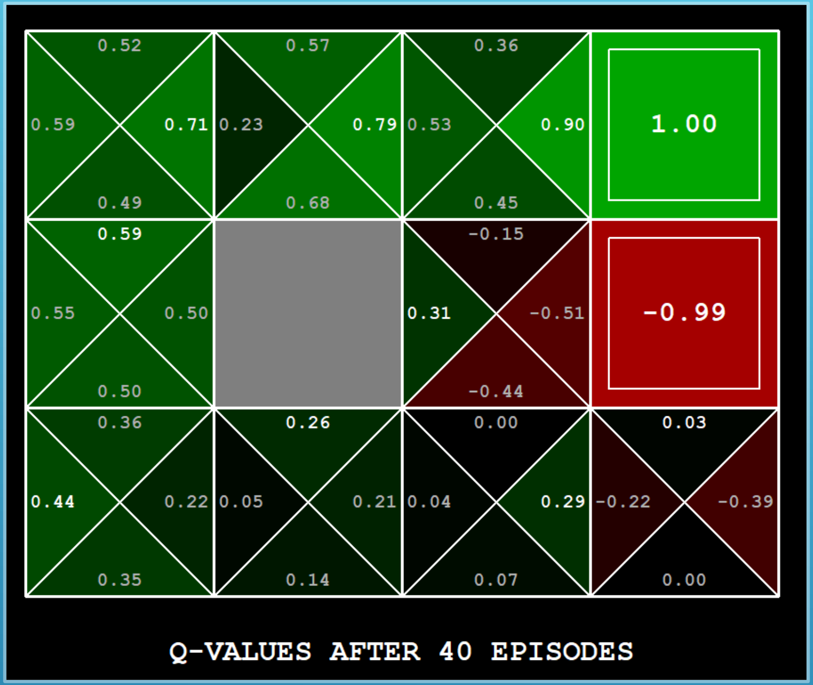
Q-VALUES AFTER 40 EPISODES

DEMO:

# get there carefully with living reward -0.0085

python gridworld.py -w 200 -k 100 -a q -d 1.0 -r -0.0085 -q

Q-LEARNING EXAMPLE: GRIDWORLD

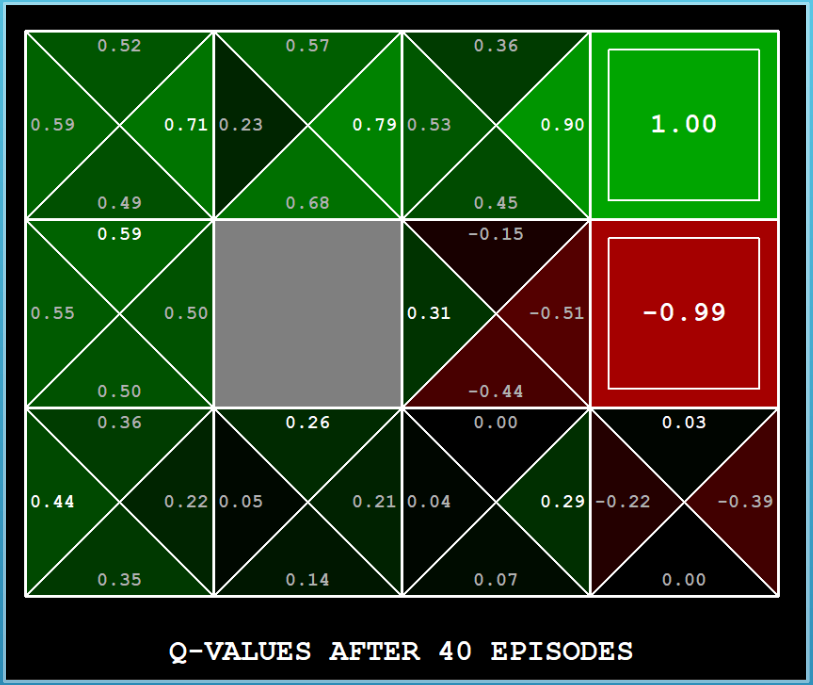


DEMO:

# get there carefully with living reward -0.0085 after 100 episodes

python gridworld.py -w 200 -k 100 -a q -d 1.0 -r -0.0085 -q

Q-LEARNING EXAMPLE: GRIDWORLD

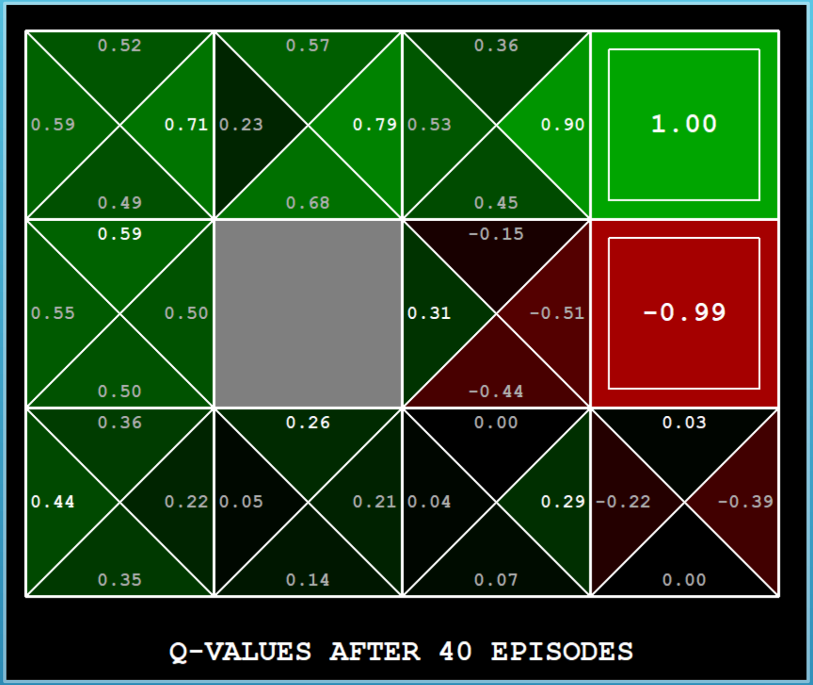


DEMO:

# get there fast with living reward -0.1 after 100 episodes

python gridworld.py -w 200 -k 100 -a q -d 1.0 -r -0.1 -q

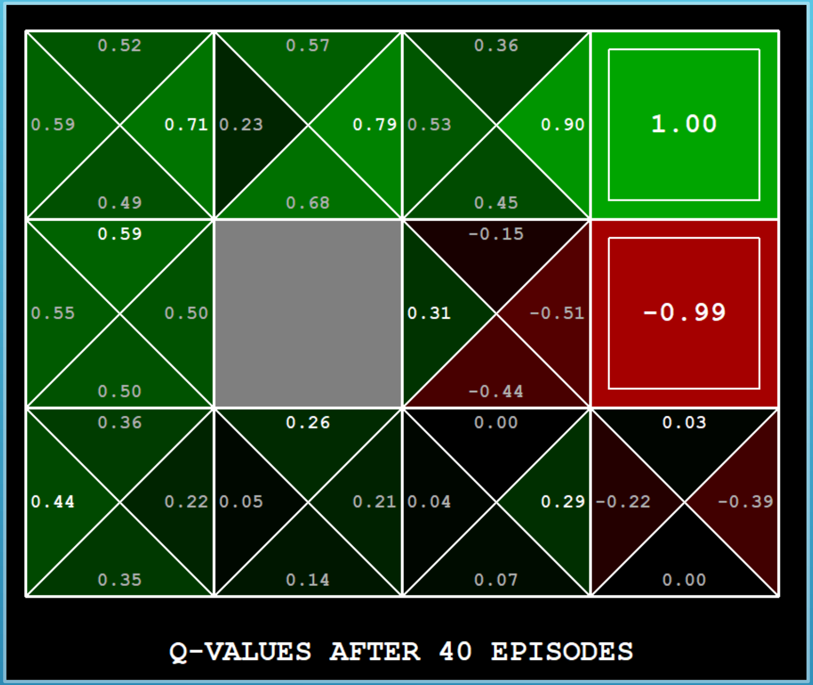
Q-LEARNING EXAMPLE: GRIDWORLD



Q-VALUES AFTER 40 EPISODES

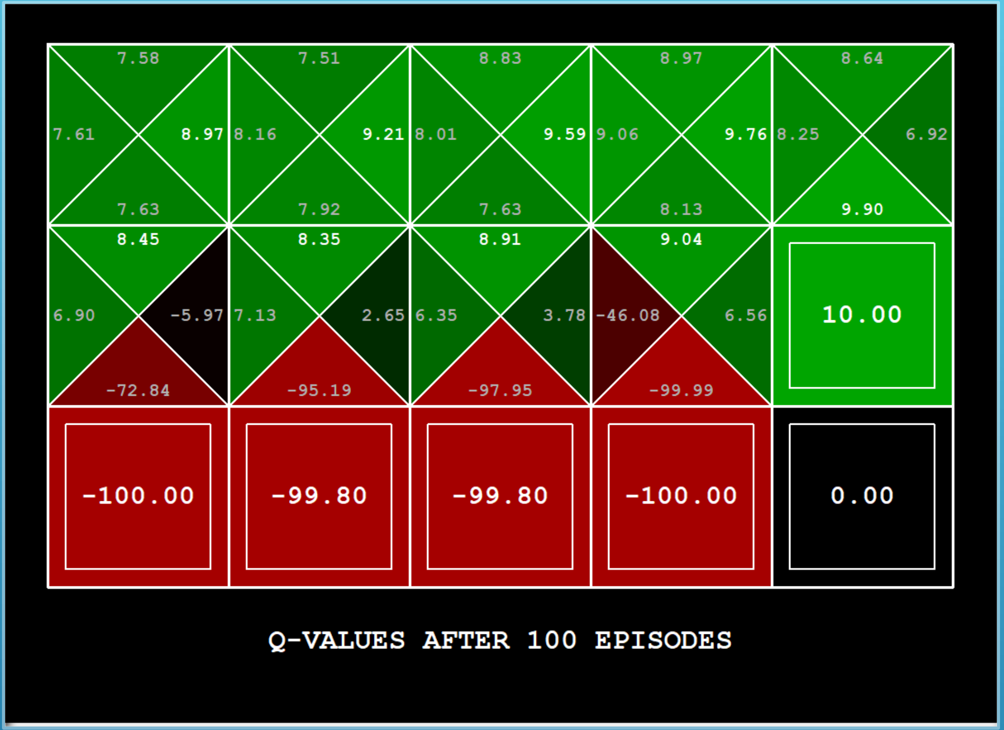
DEMO:
# die fast with living reward -1.0 after 100 episodes
python gridworld.py -w 200 -k 100 -a q -d 1.0 -r -1.0 -q

Q-LEARNING EXAMPLE: GRIDWORLD



DEMO:
# stay alive indefinitely with living reward 1.0 after 100 episodes
python gridworld.py -w 200 -k 100 -a q -d 1.0 -r 1.0 -q

Q-LEARNING EXAMPLE: GRIDWORLD



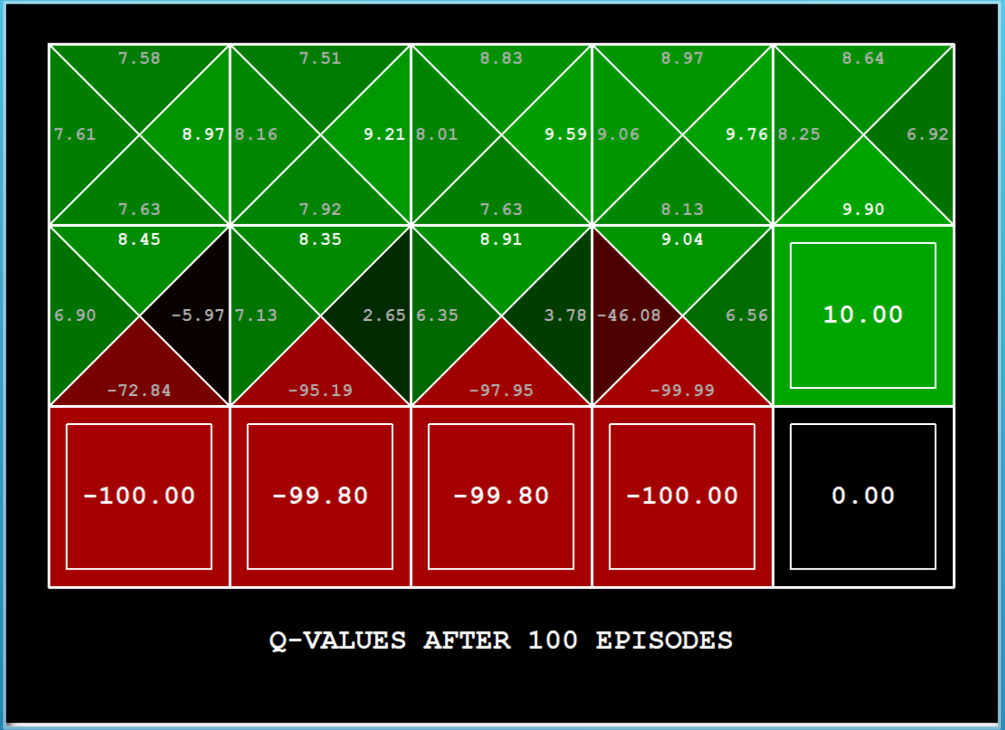
Q-VALUES AFTER 100 EPISODES

DEMO:

# avoid cliff with living reward -0.1 after 100 episodes

python gridworld.py -w 200 -k 100 -a q -d 1.0 -g CliffGrid -l 0.5 -r -0.1 -q

Q-LEARNING EXAMPLE: GRIDWORLD



DEMO:

# ignore the cliff with living reward -25.0 after 100 episodes

python gridworld.py -w 200 -k 100 -a q -d 1.0 -g CliffGrid -l 0.5 -r -25.0 -q

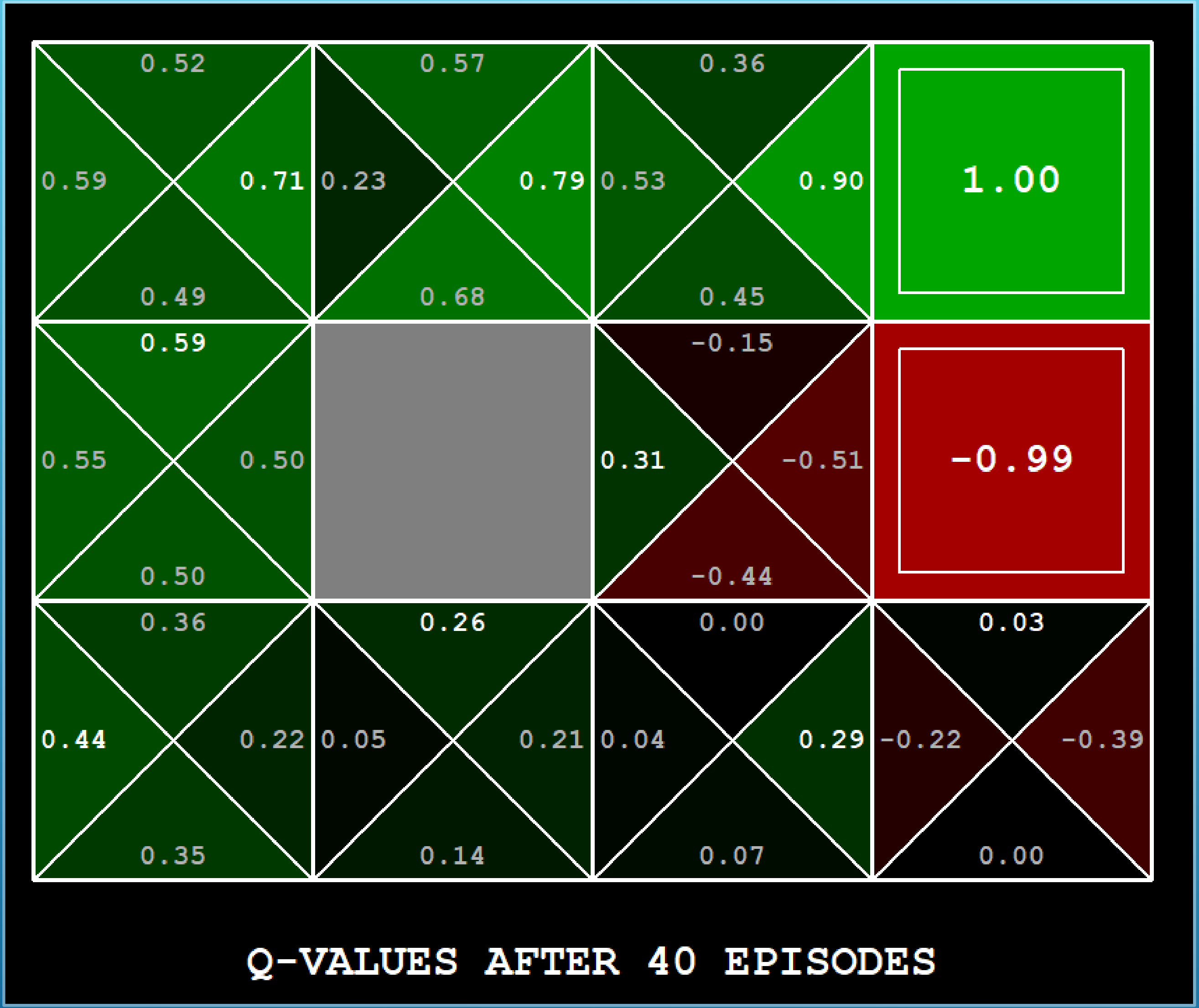
INTRODUCTION TO Q-LEARNING

Scott O'Hara

Metrowest Developers Machine Learning Group

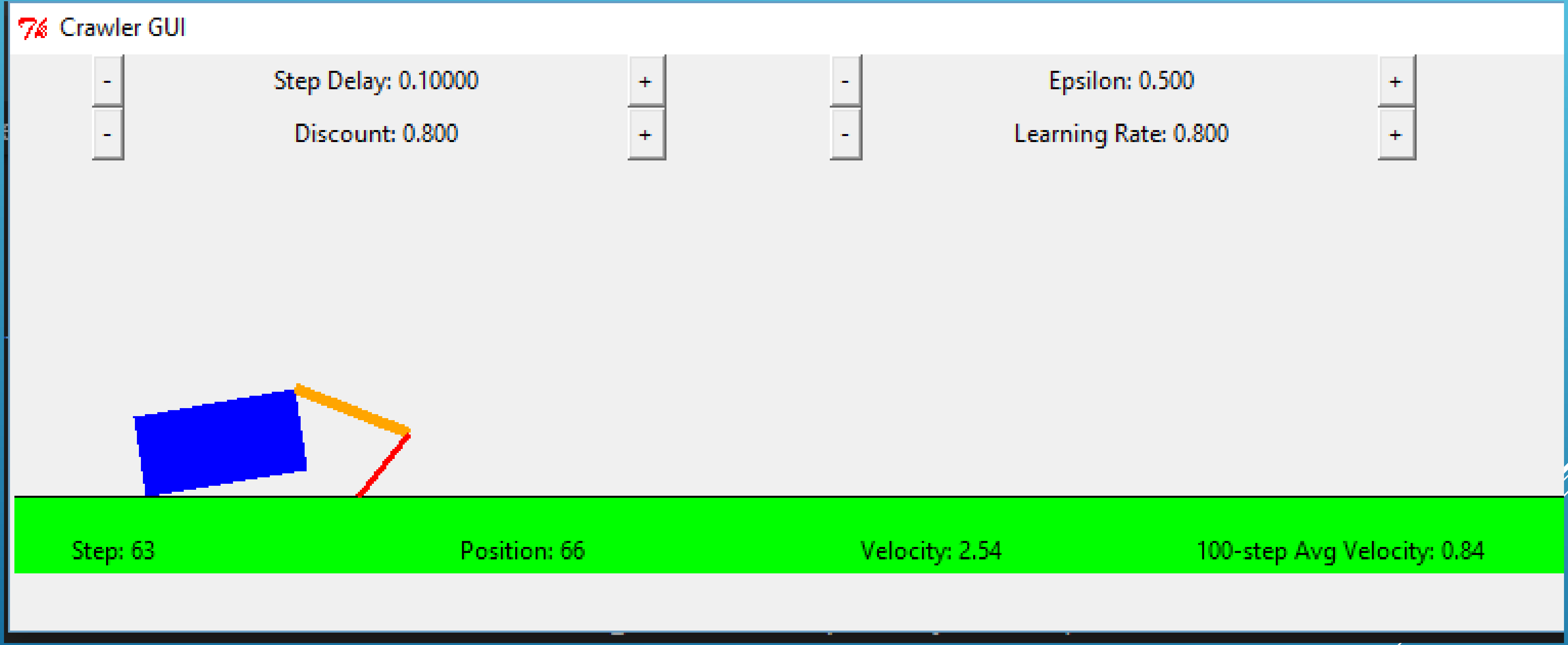
EXTRA SLIDES

Q-LEARNING EXAMPLE: GRIDWORLD



Q-VALUES AFTER 40 EPISODES

Q-LEARNING EXAMPLE: CRAWLER ROBOT



Step Delay: 0.10000

Epsilon: 0.500

Discount: 0.800

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

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

REFERENCES

The material for this talk is primarily drawn from the slides, notes and lectures of these courses with occasional reference to Sutton and Barto's book.

"Demystifying Deep Reinforcement Learning," 2015, Tambet Matiisen

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

CS188 course at University of California, Berkeley:

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

Reinforcement learning: an introduction R. S. Sutton and A. G. Barto, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

UC BERKELEY CS188 IS A GREAT RESOURCE!

- Websites:
  - http://ai.berkeley.edu/home.html
  - http://gamescrafters.berkeley.edu/~cs188/sp20/
  - http://gamescrafters.berkeley.edu/~cs188/{sp | fa}<yr>/
- Covers:
  - Search
  - Constraint Satisfaction
  - Games
  - Reinforcement Learning
  - Bayesian Networks
  - Surveys Advanced Topics
  - And more…
- Features:
  - Contains high quality YouTube videos, PowerPoint slides and homework.
  - Projects are based on the video game PacMan.
  - Material is used in many courses around the country.