CS 343H: Honors AI

Lecture 12:

Reinforcement Learning, part 1

2/25/2014

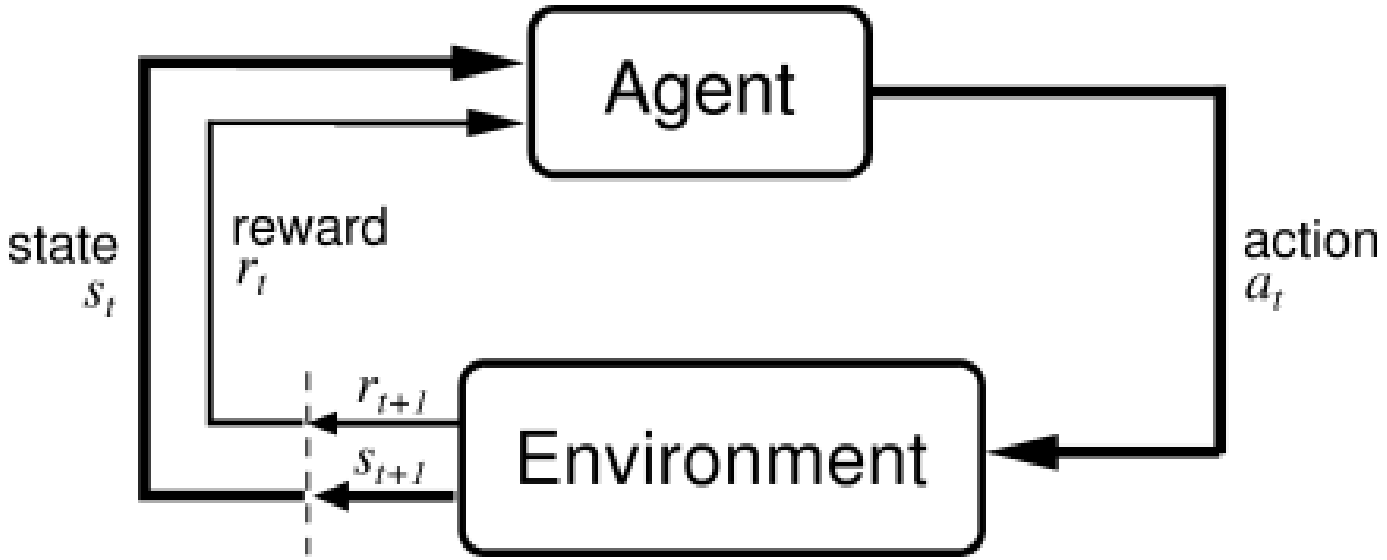
Kristen Grauman

UT Austin

Slides courtesy of Dan Klein, UC Berkeley

Reinforcement Learning

- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes



Recall: Robot gait learning

AUSTIN VILLA
ROBOT SOCCER TEAM
THE UNIVERSITY OF TEXAS AT AUSTIN

- Home
- Nao League
- Sim. League
- 4 Legged League
- @Home
- Research
- Papers
- Competitions
- Members
- Press
- Downloads
- Related Sites
- Restricted

Learning to Walk

To learn to walk faster, the Aibos evaluated different gaits by walking back and forth across the field between pairs of beacons, timing how long each lap took. **The learning was all done on the physical robots with no human intervention** (other than to change the batteries). To speed up the process, we had three Aibos working simultaneously, dividing up the search space accordingly.

Experimental Setup
(11.0 MB MPEG)

Initial Gait
(2.8 MB MPEG)

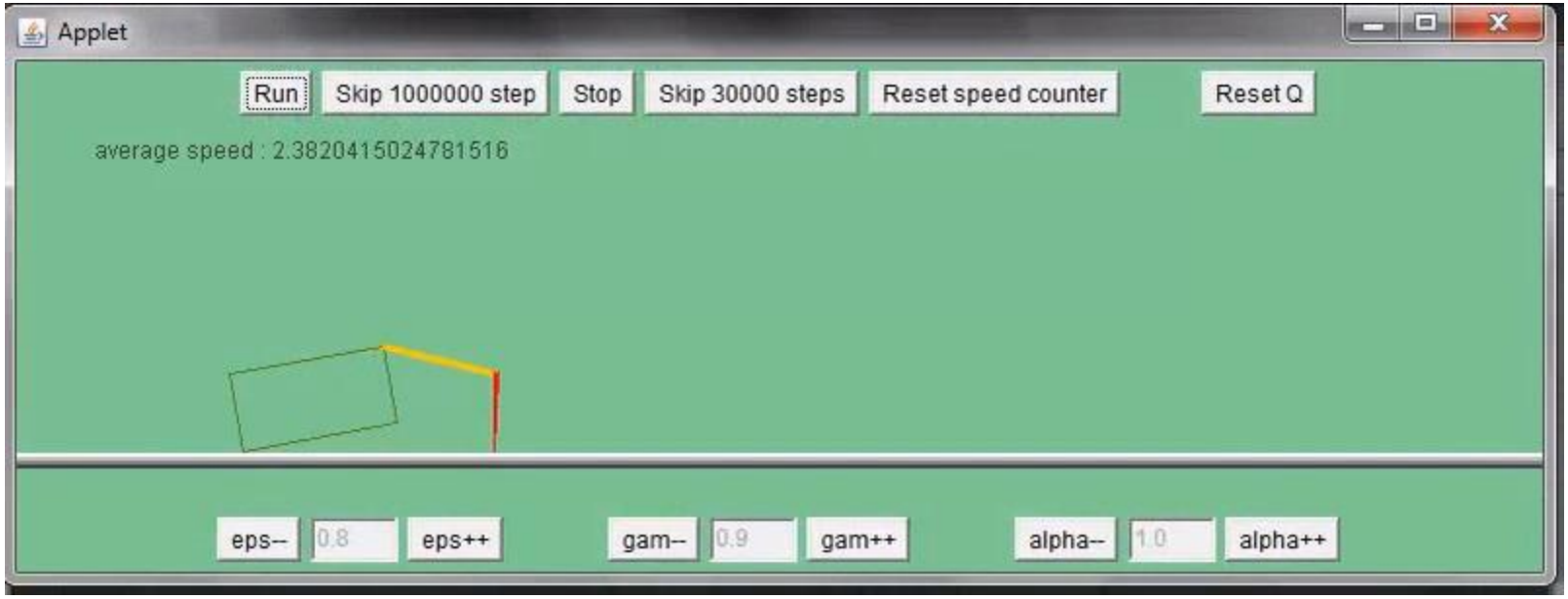
Initially, the Aibo's gait is clumsy and fairly slow (less than 150 mm/s). We deliberately started with a poor gait so that the learning process would not be systematically biased towards our best hand-tuned gait, which might have been locally optimal.

Midway through the training process, the Aibo is moving much faster than it was initially. However, it still exhibits some irregularities that slow it down.

Example: Toddler robot

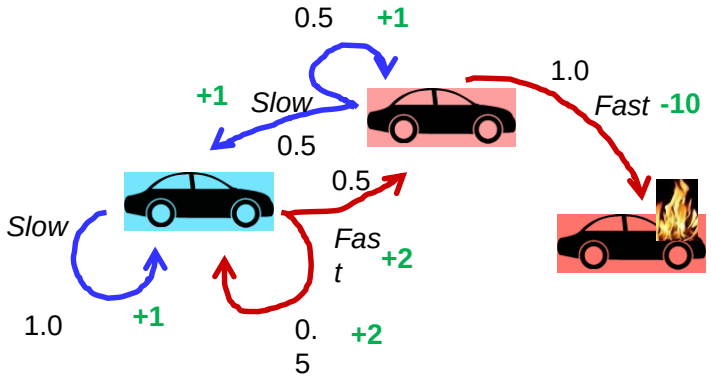
- Russ Tedrake et al., MIT

Example: Crawler



Reinforcement Learning

- Still assume an MDP:
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi(s)$



- New twist: don't know T or R
  - I.e. don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Model-Based Learning

- Idea:
  - Learn the model empirically through experience
  - Solve for values as if the learned model were correct

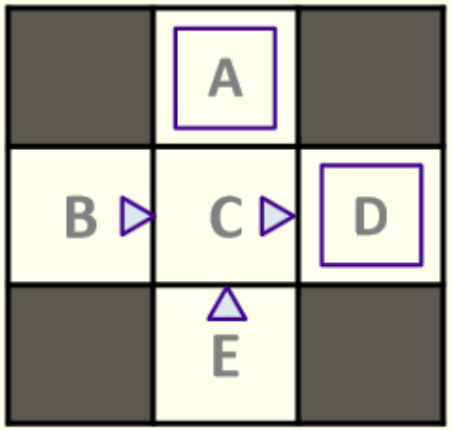
- Step 1: Simple empirical model learning
  - Count outcomes s' for each s,a
  - Normalize to give estimate of **T(s,a,s')**
  - Discover **R(s,a,s')** when we experience (s,a,s')

- Step 2: Solve the learned MDP
  - Iterative policy evaluation, for example

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')]$$

Example: Model-based learning

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

Example: Expected Age

Goal: Compute expected age of CS 343 students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Works because eventually you learn the right model

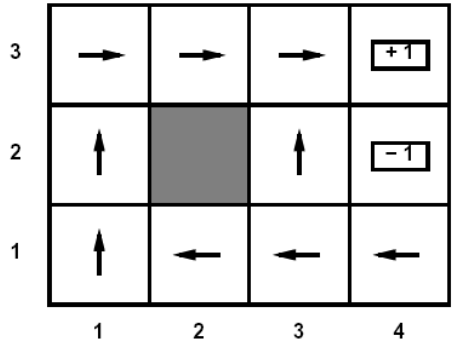
Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Works because samples appear with the right frequencies.

Passive reinforcement learning

- Simplified task
  - You are given a policy $\pi(s)$
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: learn the state values
  - … what policy evaluation did



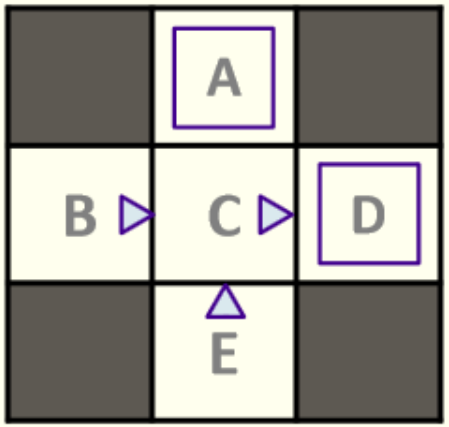
- In this case:
  - Learner "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
    - We'll get to the active case soon
  - This is NOT offline planning! You actually take actions in the world and see what happens…

Direct evaluation

- Goal: compute values for each state under π
- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be.
  - Average those samples
- This is called direct evaluation

Example: direct evaluation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

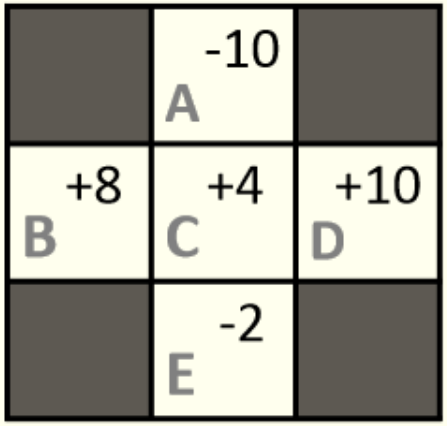
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

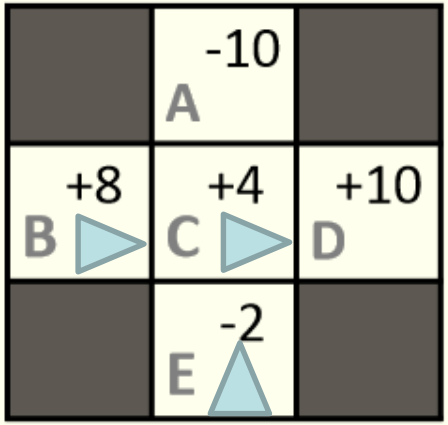
E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



Problems with direct evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions.

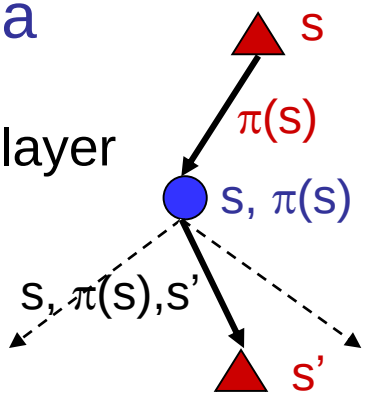


- What's bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

If B and E both go to C under this policy, how can their values be different?

Why not use policy evaluation?

- Simplified Bellman updates to calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer using current V



$$V_0^{\pi}(s) = 0$$

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')]$$

  - This approach exploited connections between states
  - Unfortunately we need T and R to do it!

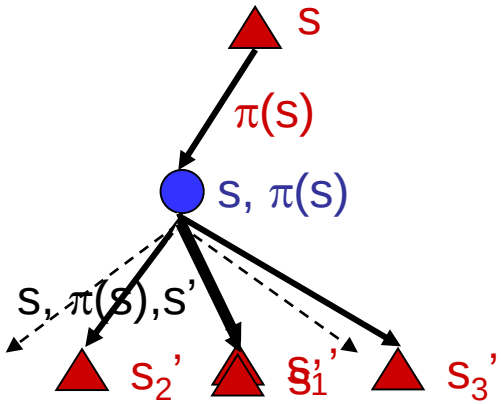
- Key question: how can we do this update to V without knowing T and R?
  - i.e., how do we take a weighted average without knowing the weights?

Sample-Based Policy Evaluation?

We want to improve our estimate of V by computing these averages:

$$V^{\pi}_{i+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}_{i}(s')]$$

- Idea: take samples of outcomes s' (by doing the action!) and average.



$$sample_1 = R(s, \pi(s), s'_1) + \gamma V^{\pi}_{i}(s'_1)$$

$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{i}(s'_2)$$

$$\ldots$$

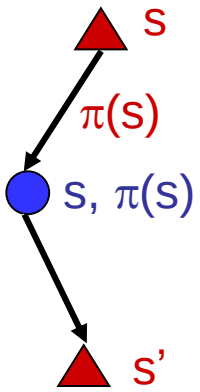
$$sample_k = R(s, \pi(s), s'_k) + \gamma V^{\pi}_{i}(s'_k)$$

$$V^{\pi}_{i+1}(s) \leftarrow \frac{1}{k} \sum_{i} sample_i$$

Almost! But we can't rewind time to get sample after sample from state s.

Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s,a,s',r)
  - Likely outcomes s' will contribute updates more often



- Temporal difference learning
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: "running average"

Sample of V(s): $$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

Update to V(s): $$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$$

Same update: $$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$$

Exponential Moving Average

- Exponential moving average
  - The running interpolation update:

$$\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

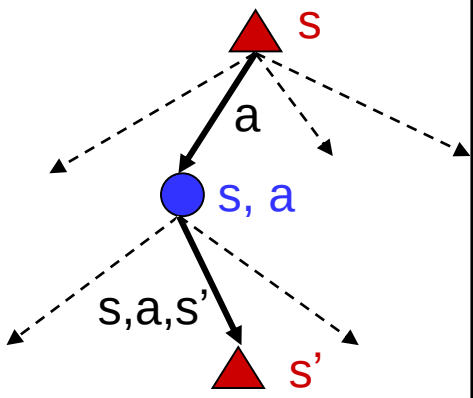
  - Forgets about the past (distant past values were wrong anyway)

Problems with TD Value Learning

- TD value leaning is model-free way to do policy evaluation, mimicking Bellman updates with running averages
- However, if we want to turn values into a (new) policy, we're sunk:

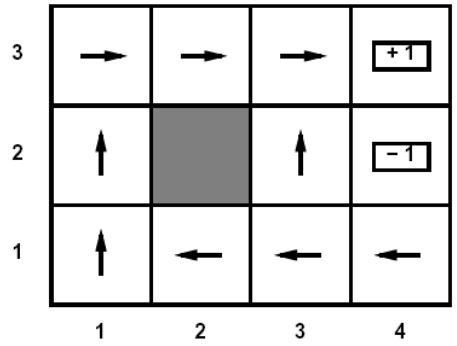
$$\pi(s) = \arg\max_a Q^*(s,a)$$

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



- Idea: learn Q-values directly
- Makes action selection model-free too!

Active reinforcement learning



- Full reinforcement learning
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You can choose any actions you like
  - Goal: learn the optimal policy / values

- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Q-Value Iteration

- Value iteration: find successive depth-limited values
  - Start with $V_0(s) = 0$, which we know is right
  - Given Vi, calculate the depth i+1 values for all states:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Qi, calculate the depth i+1 q-values for all q-states:

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

Q-Learning

- Q-Learning: sample-based Q-value iteration

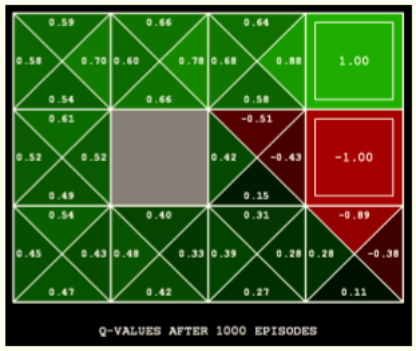
$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a') \right]$$

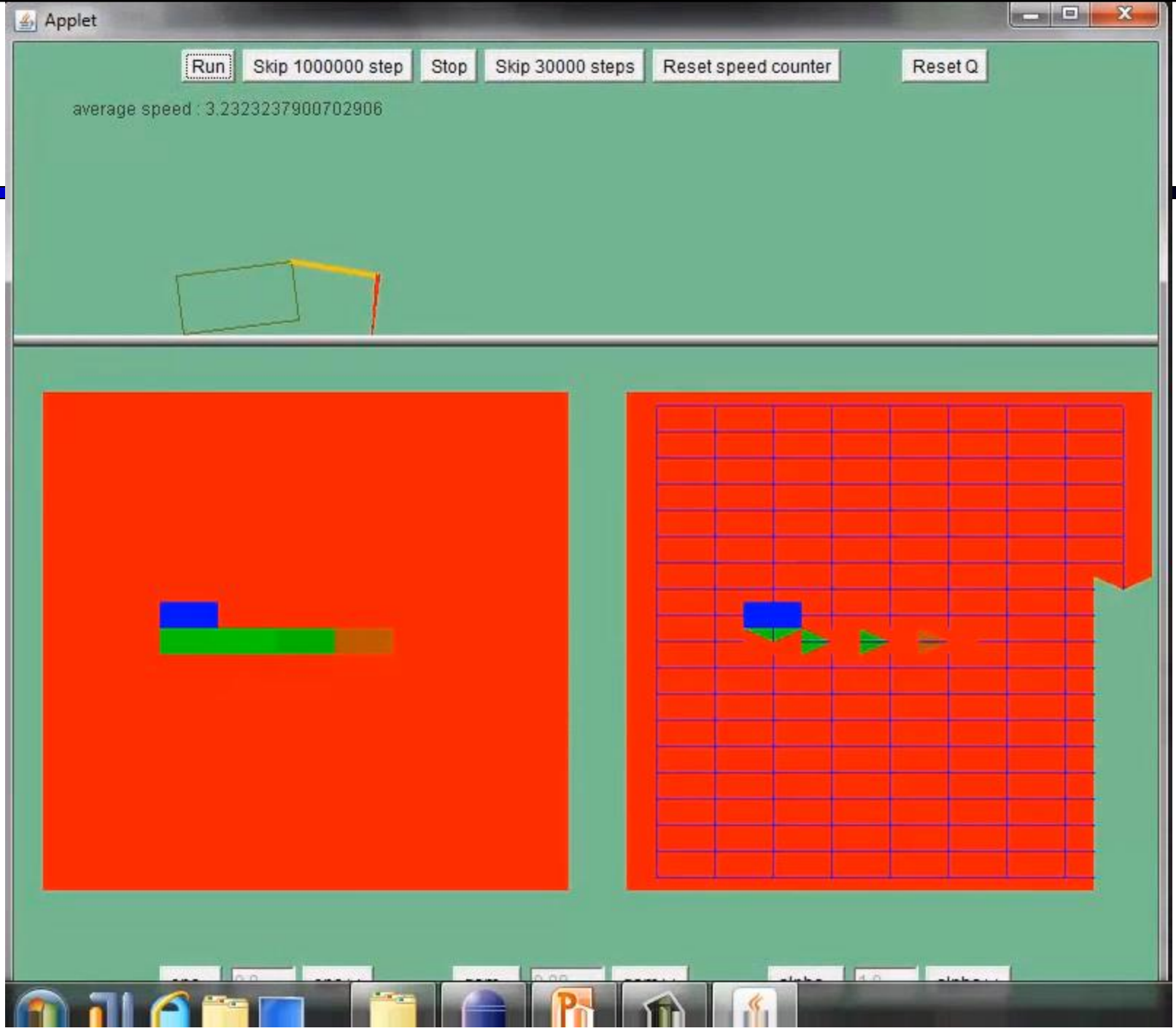
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$




Applet
Run
Skip 1000000 step
Stop
Skip 30000 steps
Reset speed counter
Reset Q
average speed : 3.2323237900702906

Q-Learning Properties

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

The Story So Far: MDPs and RL

Things we know how to do:

- If we know the MDP
  - Compute V*, Q*, π* exactly
  - Evaluate a fixed policy π
- If we don't know the MDP
  - We can estimate the MDP then solve
  - We can estimate V for a fixed policy π
  - We can estimate Q*(s,a) for the optimal policy while executing an exploration policy

Techniques:

- Model-based DPs
  - Value Iteration
  - Policy evaluation
- Model-based RL
- Model-free RL
  - Value learning
  - Q-learning