MDP
and methods do solve it

With cute videos

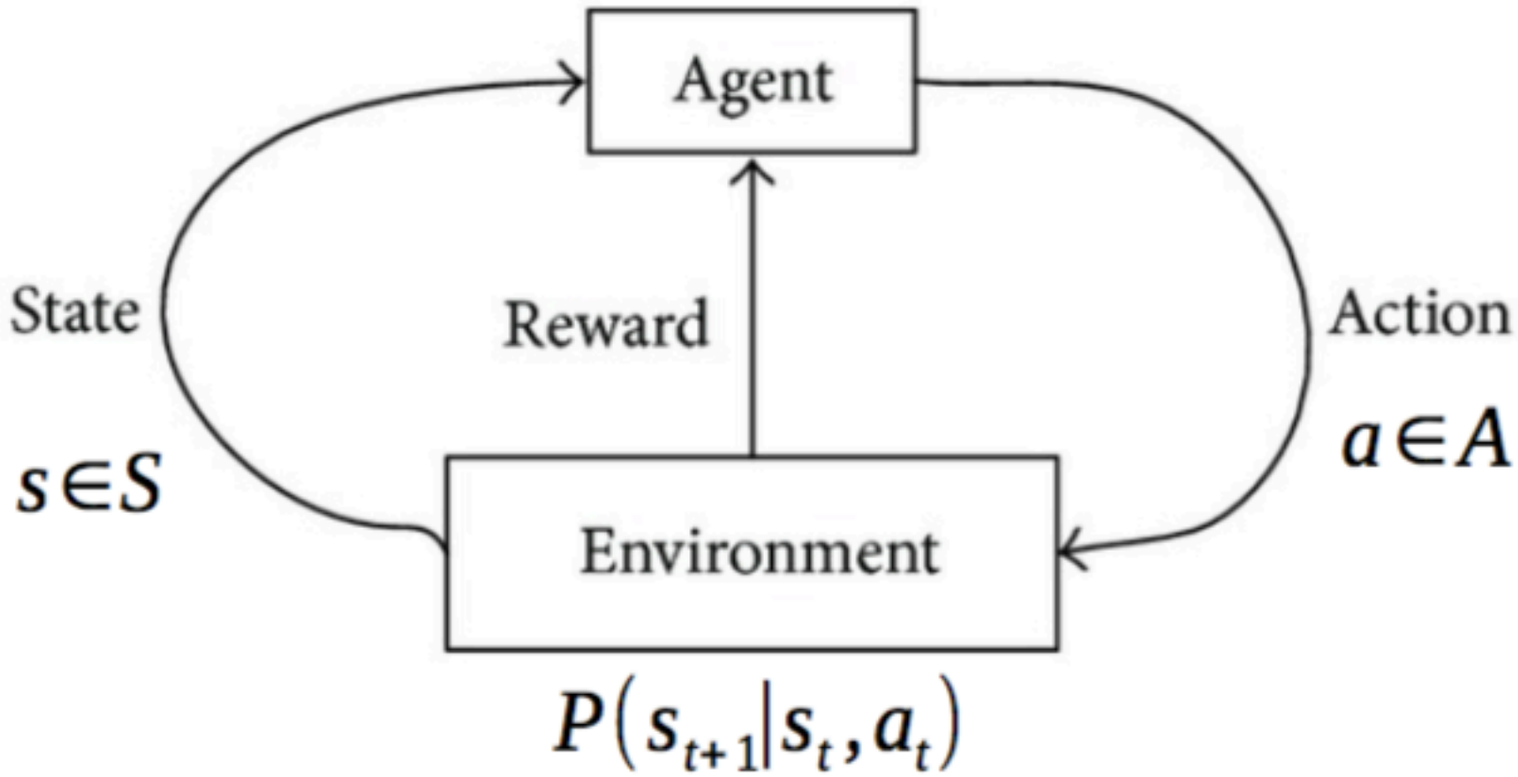
This presentation, as well as some practical tasks,
mostly borrowed from SHAD course

https://github.com/yandexdataschool/Practical_RL/

MDP

Markov Decision Process

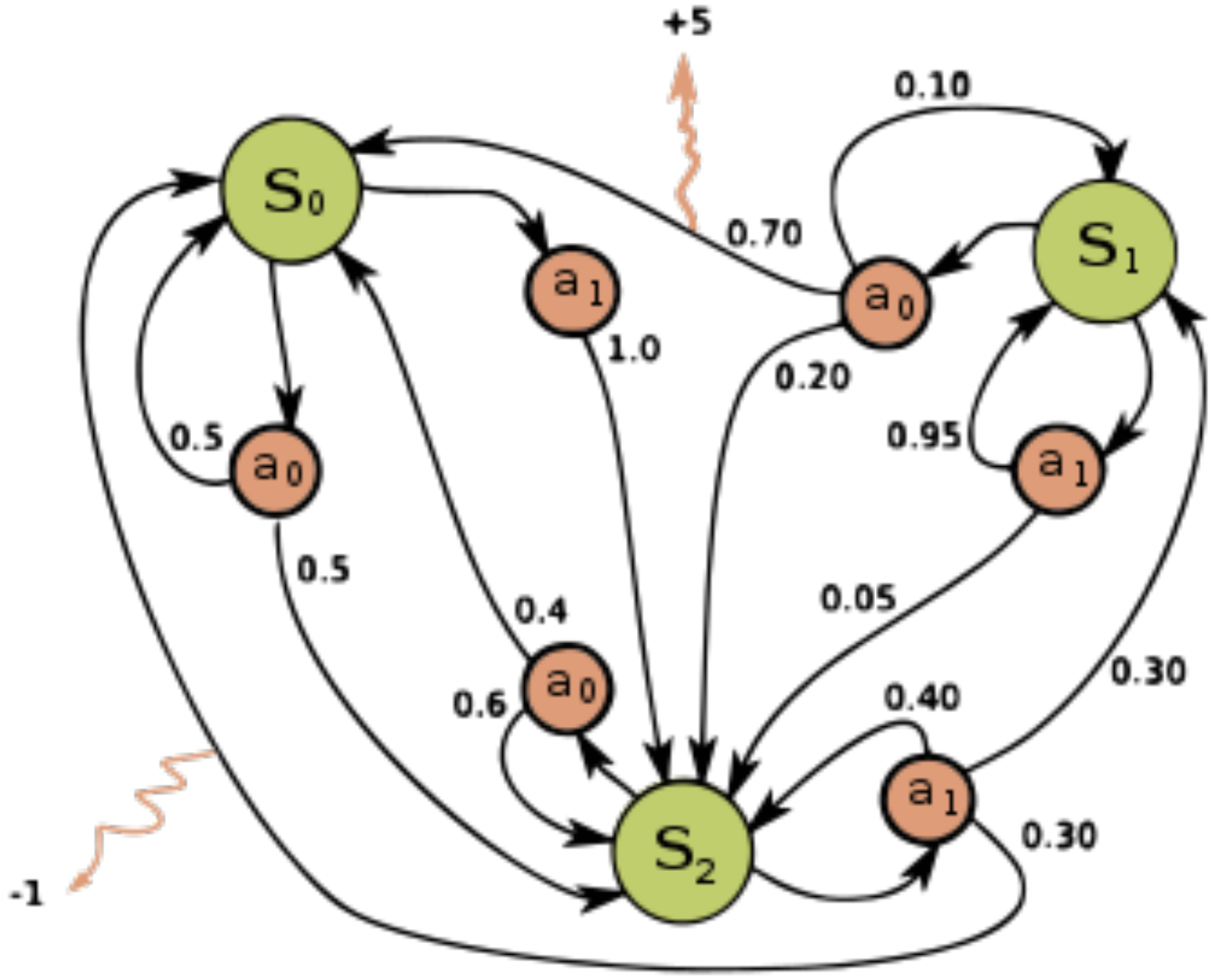
Markov Decision Process



Markov assumption

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}) = P(s_{t+1}|s_t, a_t)$$

Markov Decision Process



$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}) = P(s_{t+1}|s_t, a_t)$$

Total reward

Total reward for session:

$$R = \sum_t r_t$$

Agent's policy:

$$\pi(a|s) = P(\text{take action } a | \text{in state } s)$$

Problem: find policy with highest reward:

$$\pi(a|s) : E_\pi[R] \rightarrow max$$

Objective

The easy way:

$E_{\pi} R$ is an expected sum of rewards that agent with policy π earns per session

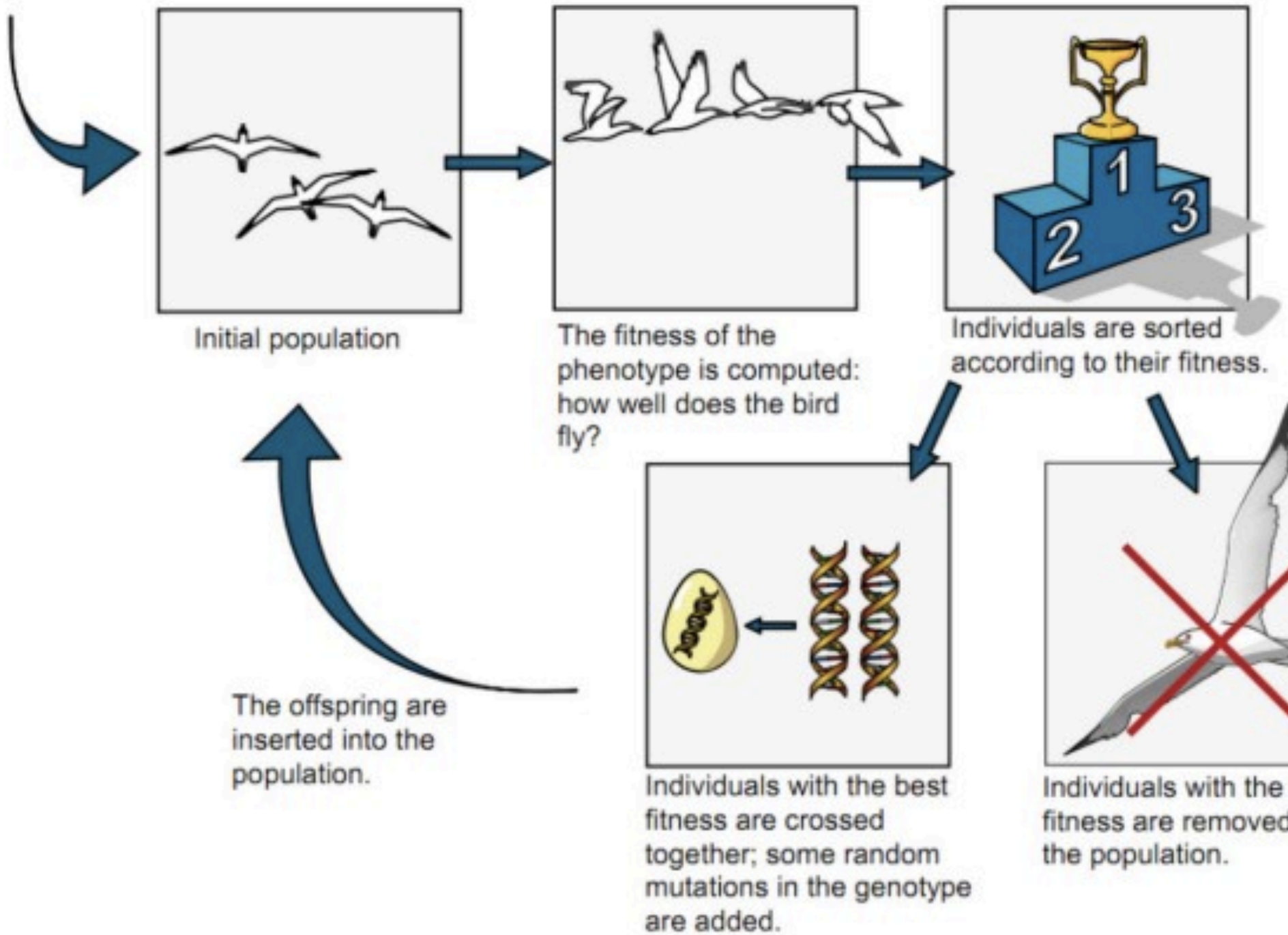
The hard way:

$$\underset{s_0 \sim p(s_0),\, a_0 \sim \pi(a|s_0),\, s_1, r_0 \sim P(s',r|s,a)}{E} \quad \underset{}{E} \quad \underset{}{E} \quad \dots \quad \underset{s_T, r_T \sim P(s',r|s_{T-1},a_{t-1})}{E} \left[r_0 + r_1 + r_2 + \dots + r_T \right]$$

How to solve?

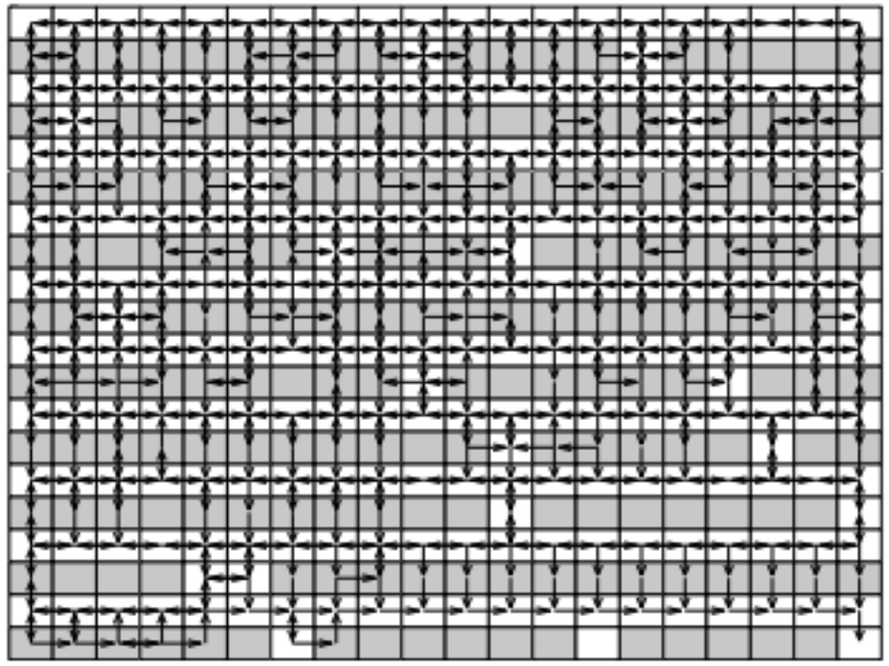
Ideas?

1. Genetic algorithms

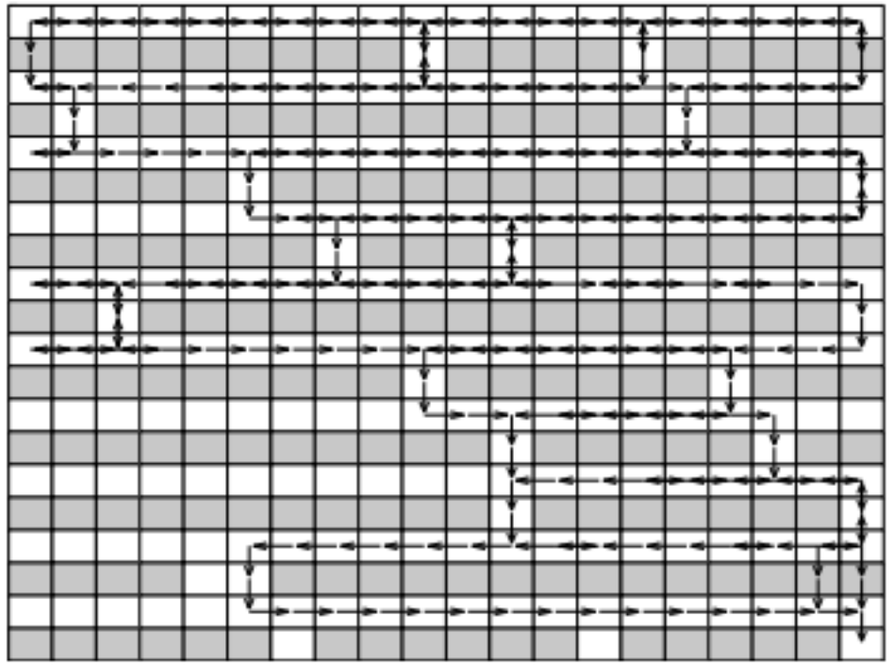


2. Cross-entropy method

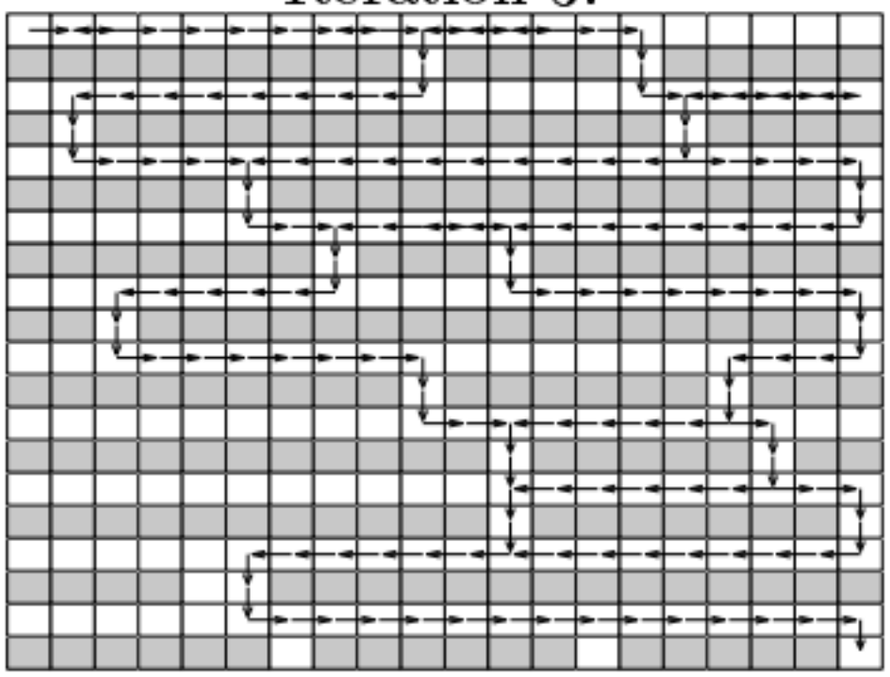
Iteration 1:



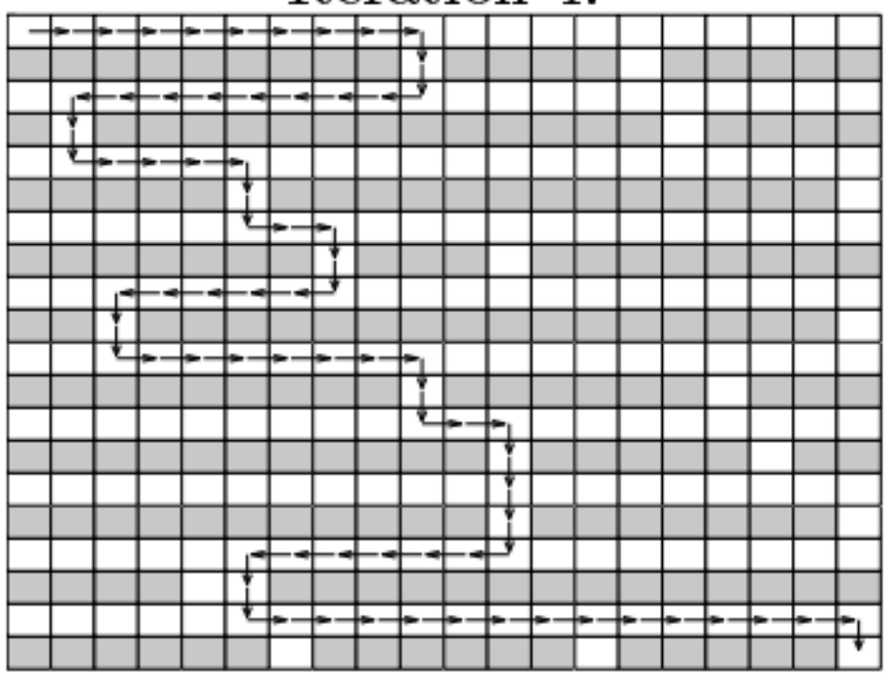
Iteration 2:



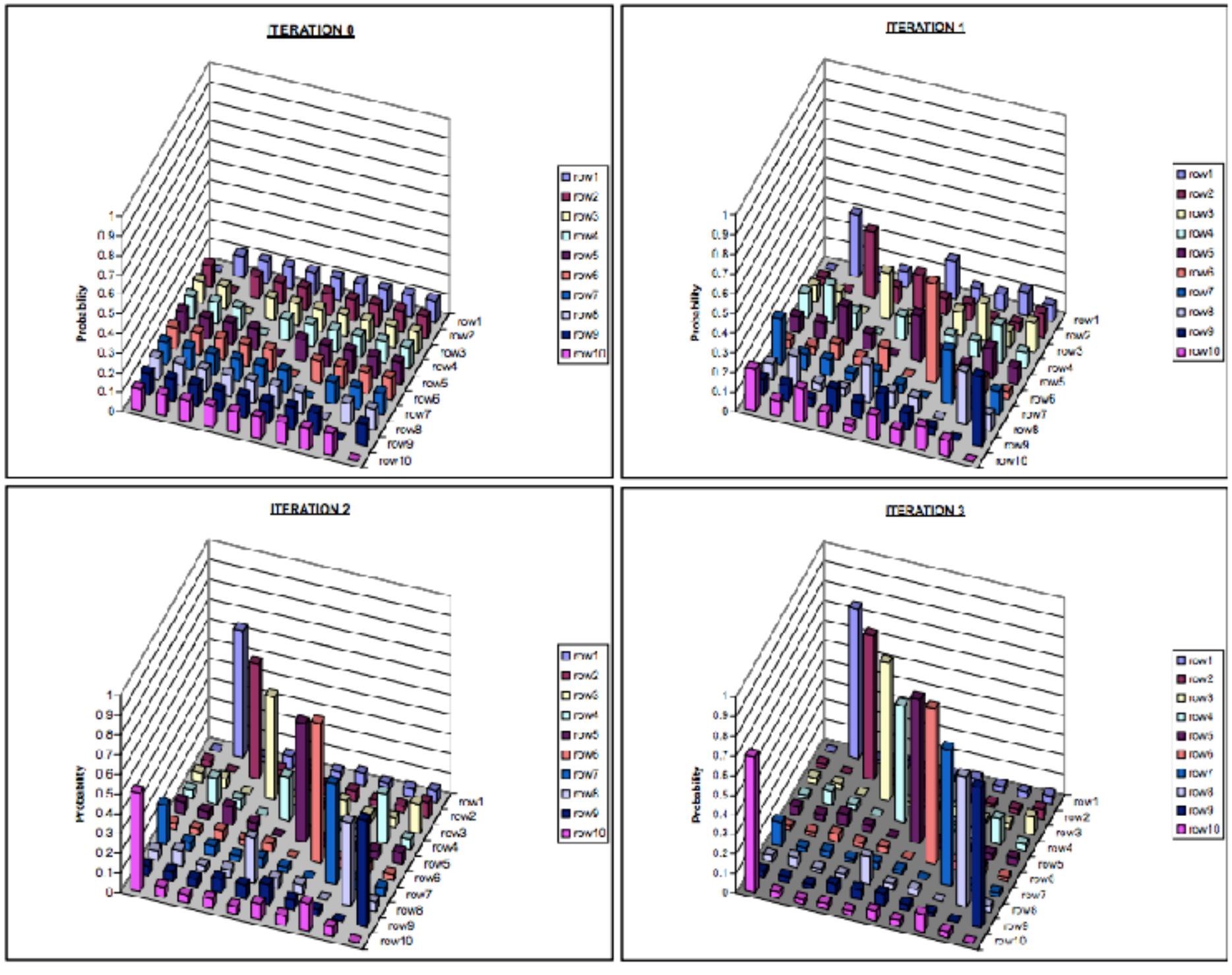
Iteration 3:



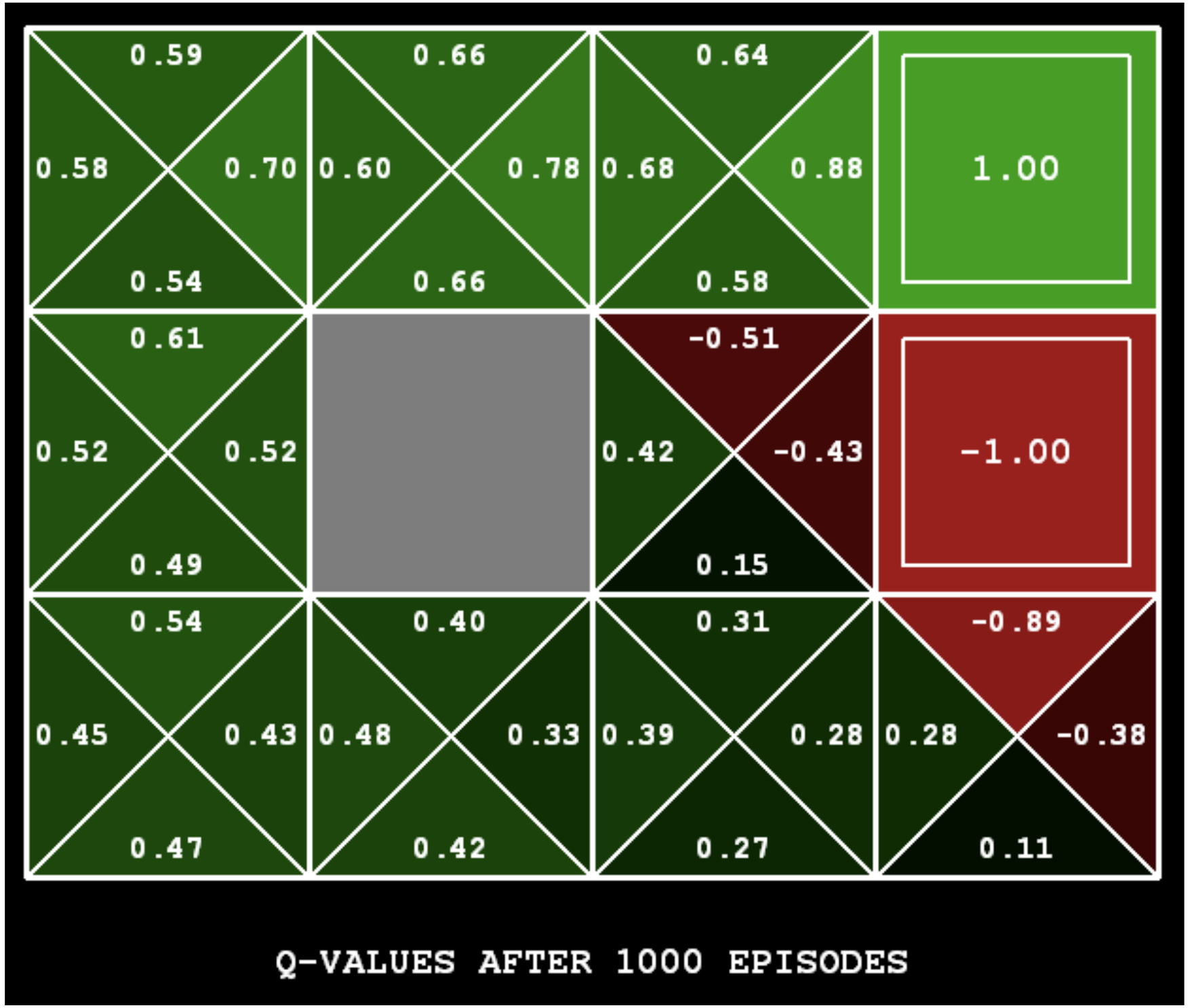
Iteration 4:



2. Cross-entropy method

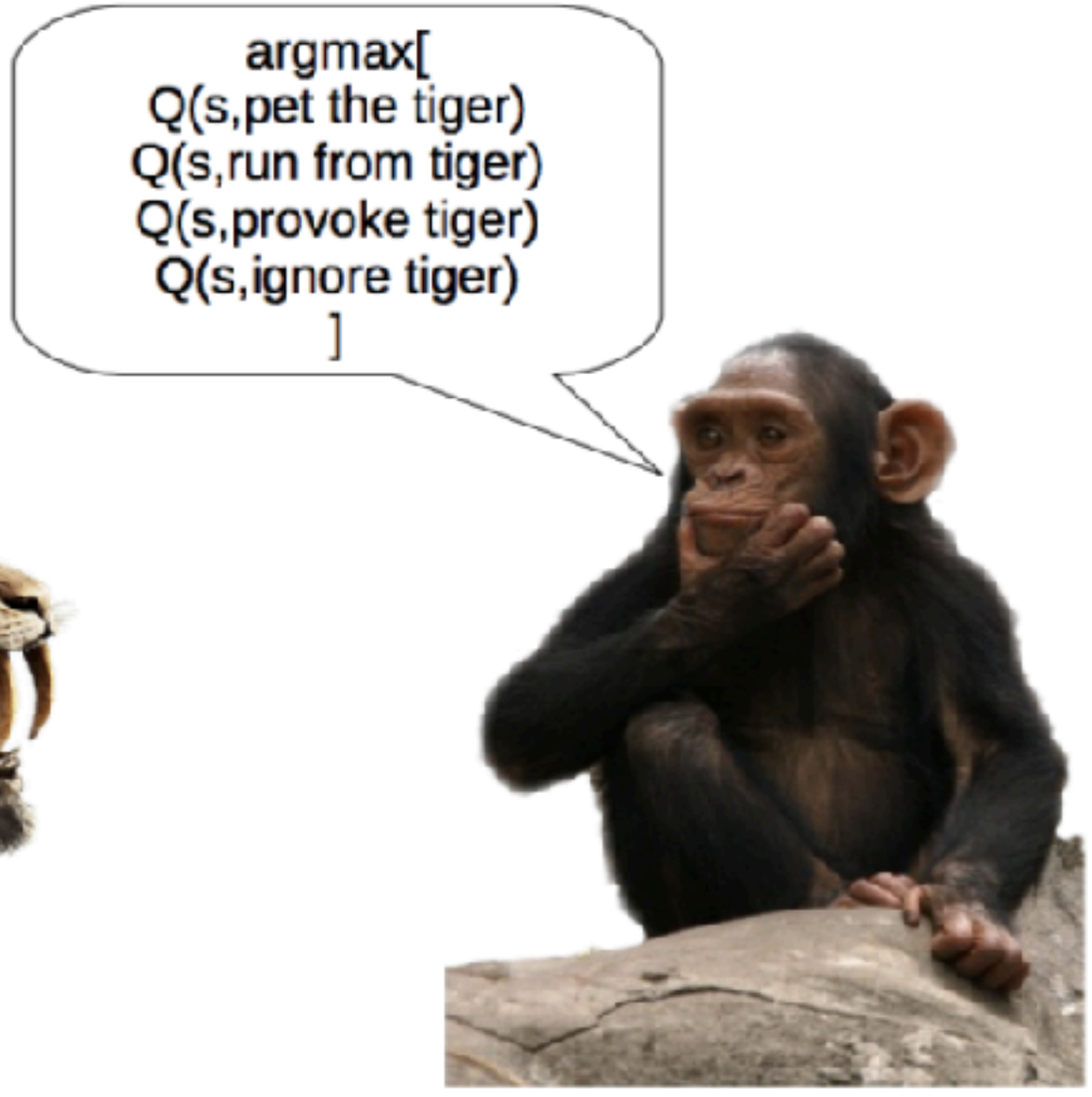


3. Q-learning


0.59
0.58
0.70
0.54
0.66
0.60
0.78
0.66
0.64
0.68
0.88
0.58
1.00
0.61
0.52
0.52
0.49
-0.51
0.42
-0.43
0.15
-1.00
0.54
0.45
0.43
0.47
0.40
0.48
0.33
0.42
0.31
0.39
0.28
0.27
-0.89
0.28
-0.38
0.11
Q-VALUES AFTER 1000 EPISODES

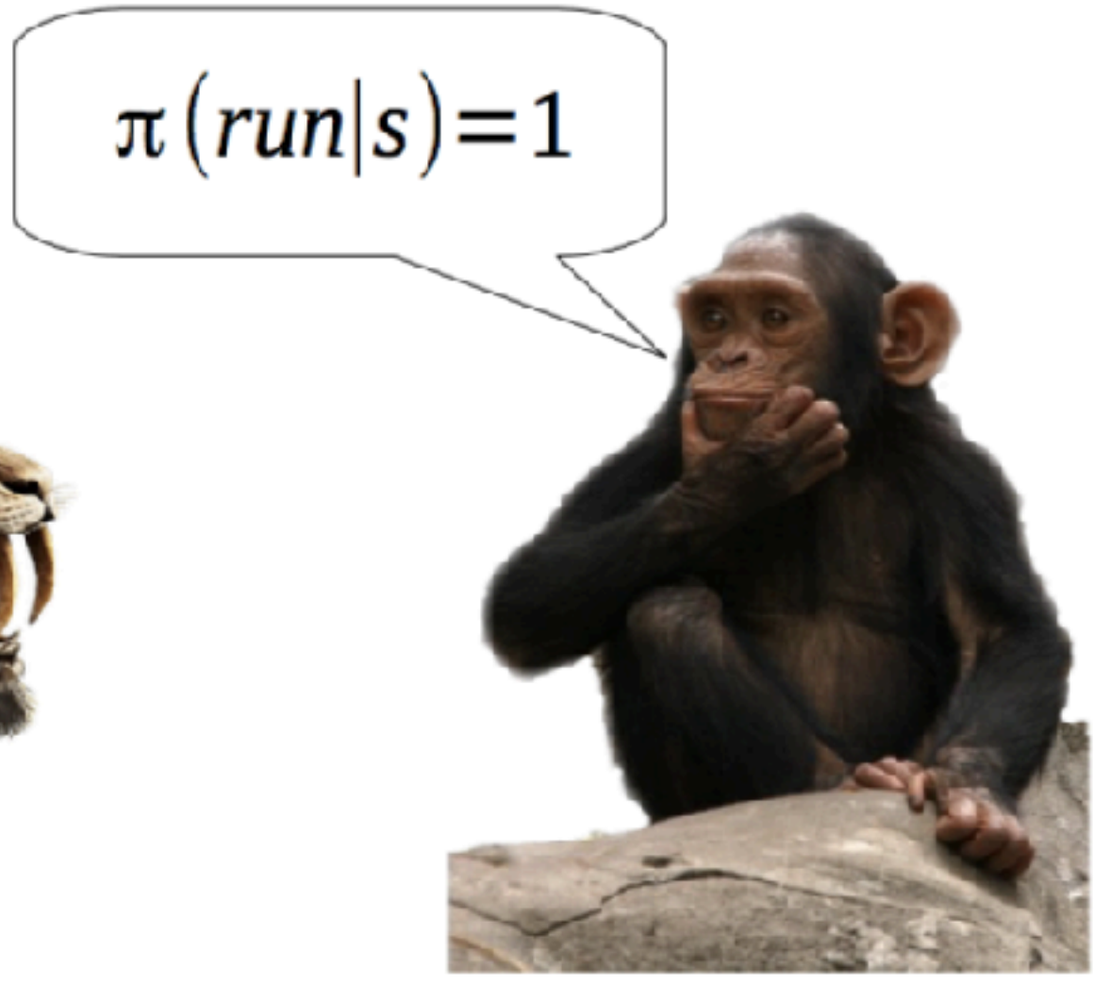
3. Policy-based

NOT how humans survived



3. Policy-based

how humans survived



Genetic algorithms

Genetic algorithms

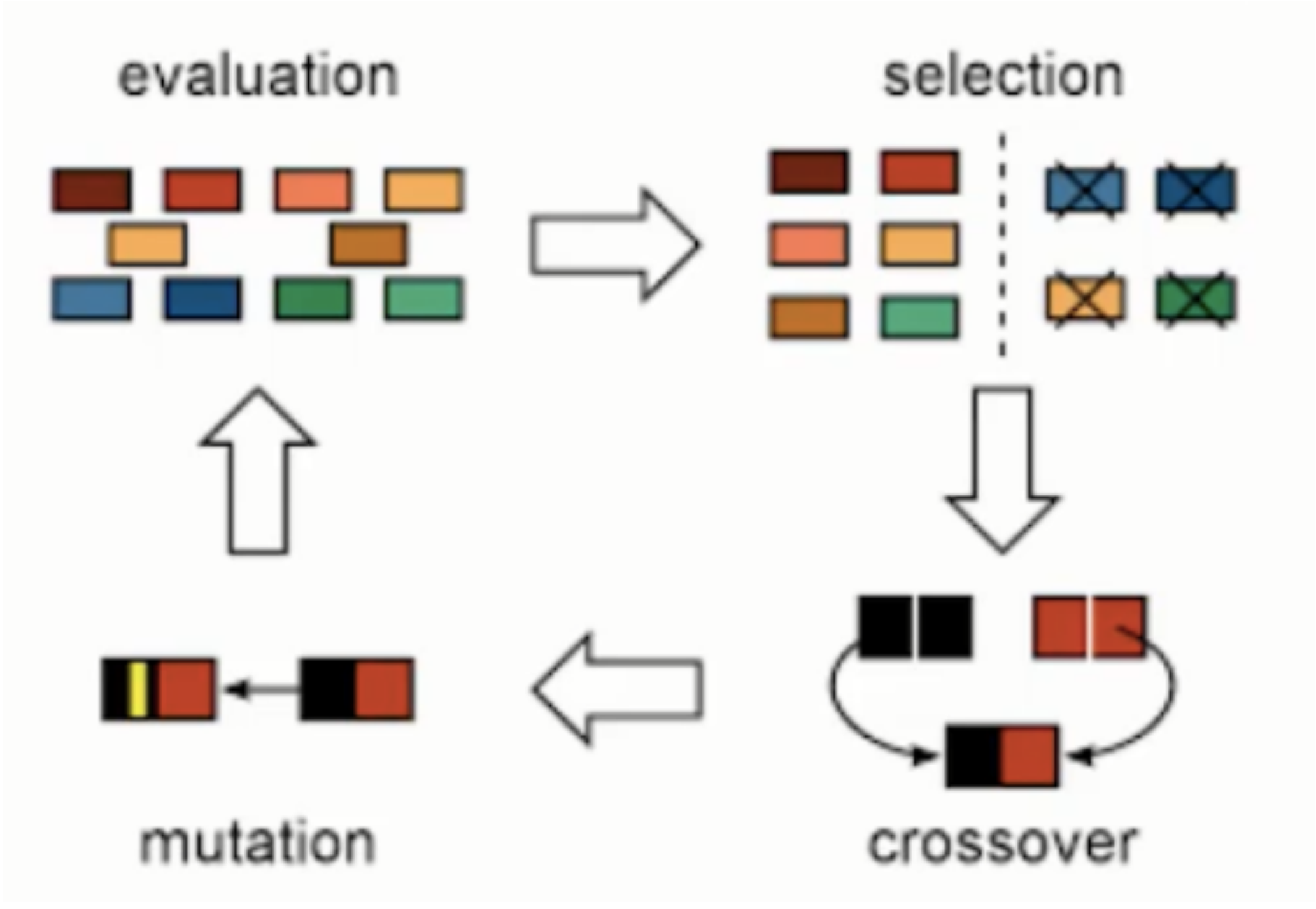
Context: FrozenLake

- A grid world with a goal tile and ice holes

SFFF (S: starting point, safe)
FHFH (F: frozen surface, safe)
FFFH (H: hole, fall to your doom)
HFFG (G: goal, where the frisbee is located)

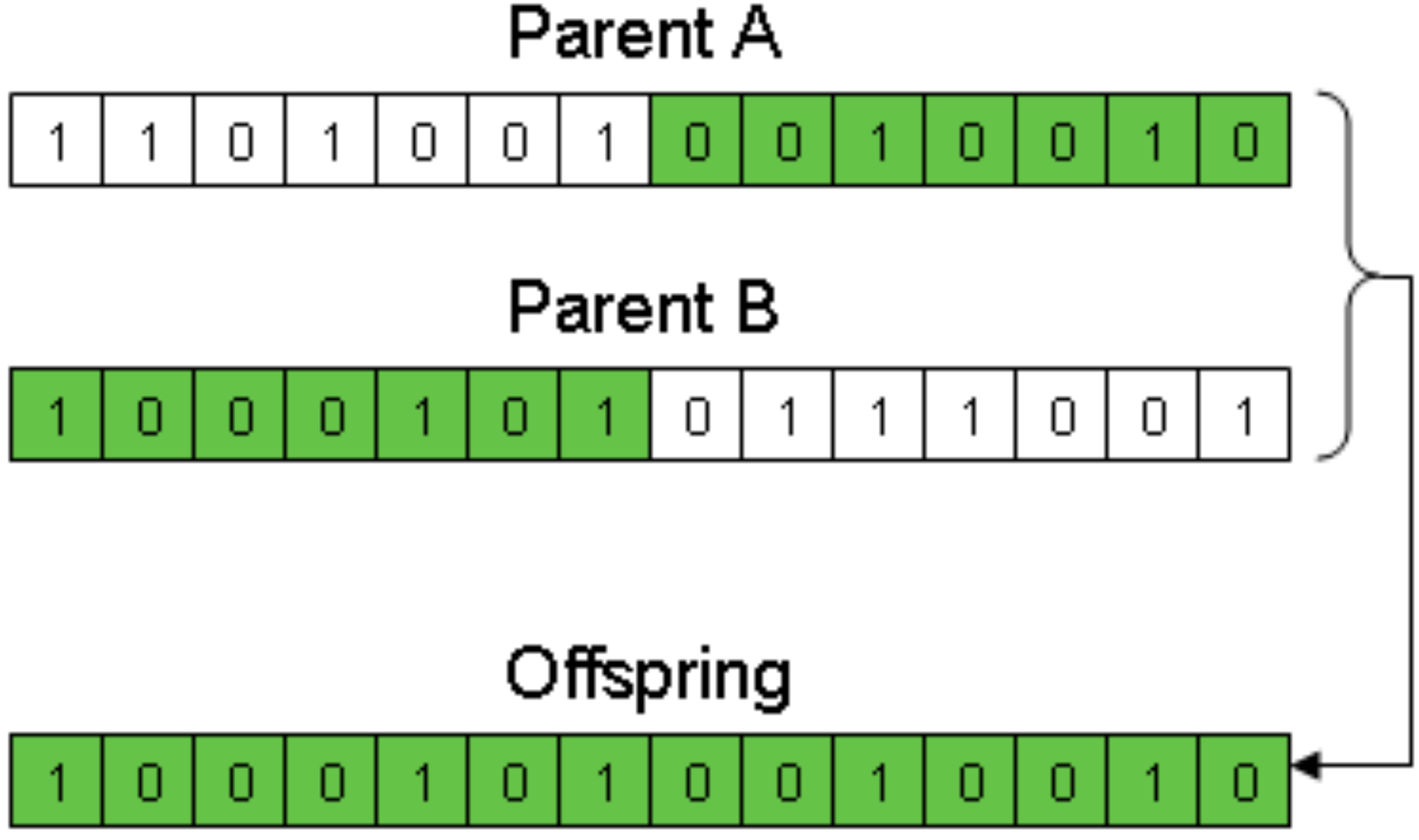
Quiz: what states, actions and rewards are used?

Genetic algorithms



Genetic algorithms

Cross over



Genetic algorithms

- Pros
  - It sorta sometimes works
  - You get to feel like a god!
- Cons
  - Convergence not guaranteed
  - Requires a lot of samples
  - A lot of parameter tuning
  - Hard to scale on large state spaces

Genetic algorithms

Examples:

Genetic Cars

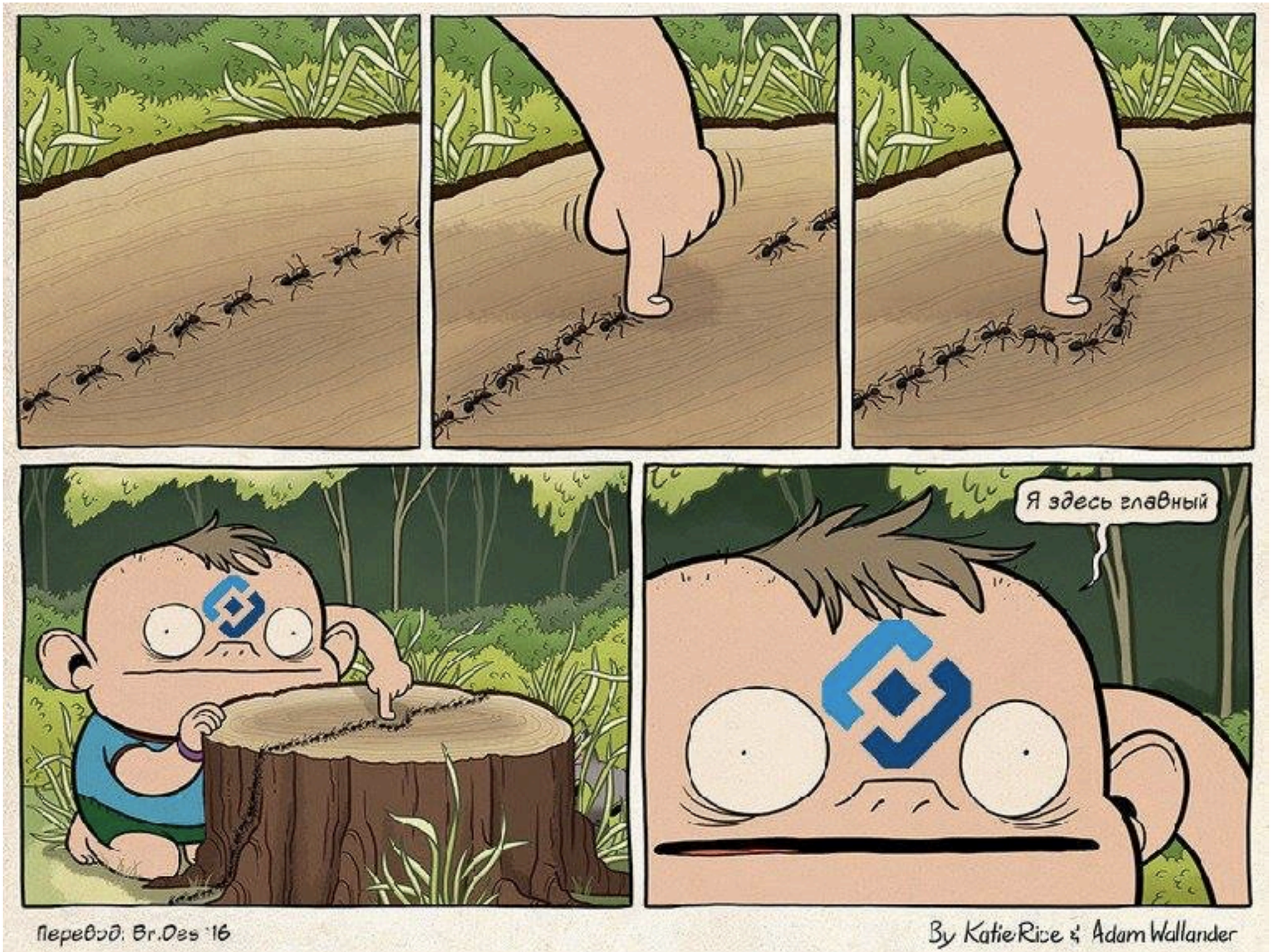
Musical Novelty Search

Stack some layers:

OpenAI paper

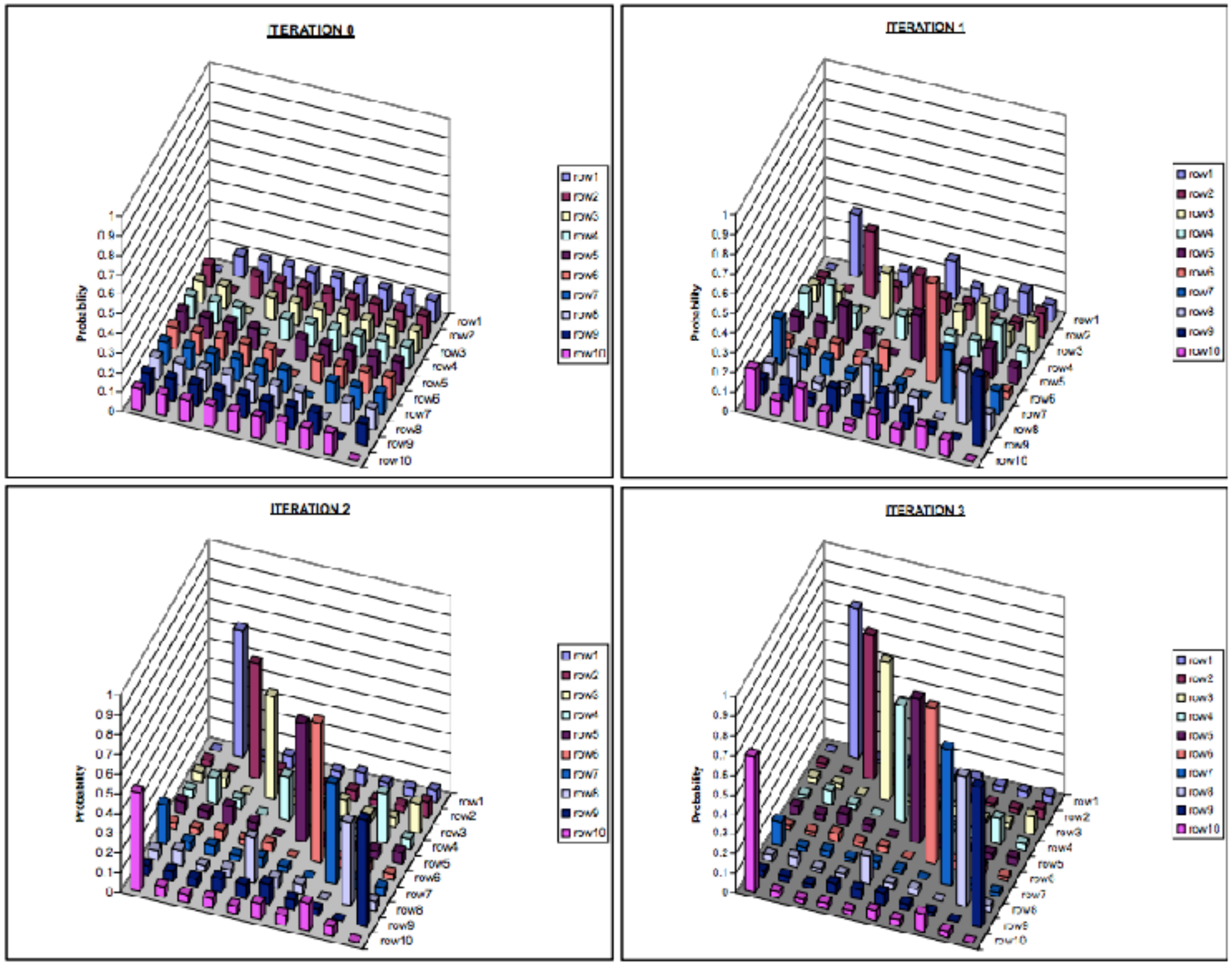
Genetic algorithms

Let’s fill like a God!



Cross-entropy method

Cross-entropy method



Cross-entropy method

General idea:

Play a few sessions

Update your policy

Repeat

Cross-entropy method

Intialize policy

Repeat:

- Sample N[100] sessions

- Pick M[25] best sessions, called **elite** sessions

- Change policy so that it prioritizes
  actions from elite sessions

Cross-entropy method

- Policy is a matrix

$$\pi(a|s) = A_{s,a}$$

- Sample N games with that policy
- Get M best sessions (elites)

$$Elite = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_k, a_k)]$$

Cross-entropy method

- Policy is a matrix

$$\pi(a|s) = A_{s,a}$$

- Sample N games with that policy
- Take M best sessions (elites)
- Aggregate by states

$$\pi(a|s) = \frac{\sum_{s_t, a_t \in Elite} [s_t = s][a_t = a]}{\sum_{s_t, a_t \in Elite} [s_t = s]}$$

Cross-entropy method

- Policy is a matrix

$$\pi(a|s) = A_{s,a}$$

- Sample N games with that policy
- Take M best sessions (elite)
- Aggregate by states

$$\pi(a|s) = \frac{took\ a\ at\ s}{was\ at\ s}$$

In M best games

Cross-entropy method

Practice time!

Cross-entropy method

If your environment has infinite/large state space

Approx cross-entropy method

If your environment has infinite/large state space

Approx cross-entropy method

- Policy is approximated
  - Neural network predicts $\pi_w(a|s)$ given s
  - Linear model / Random Forest / ...

Can't set $\pi(a|s)$ explicitly

All state-action pairs from M best sessions

$$Elite = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_k, a_k)]$$

Approx cross-entropy method

Neural network predicts $\pi_w(a|s)$ given s

All state-action pairs from M best sessions

$$Elite=[(s_0,a_0),(s_1,a_1),(s_2,a_2),\ldots,(s_k,a_k)]$$

Maximize likelihood of actions in “best” games

$$\pi = \underset{\pi}{argmax} \sum_{s_i,a_i \in Elite} \log \pi(a_i|s_i)$$

Approx cross-entropy method

- Initialize NN weights $W_0 \leftarrow random$

- Loop:
  - Sample N sessions
  - $Elite = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_k, a_k)]$
  - $W_{i+1} = W_i + \alpha \nabla [\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i | s_i)]$

Approx cross-entropy method

- Initialize NN weights **nn = MLPClassifier(...)**

- Loop:
  - Sample N sessions
  - $Elite = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_k, a_k)]$
  - ***nn.fit(elite_states,elite_actions)***

Approx cross-entropy method

- Continuous state space
- Model $\pi_w(a|s) = N(\mu(s), \sigma^2)$
  - Mu(s) is neural network output
  - Sigma is a parameter or yet another network output
- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate
  - $$W_{i+1} = W_i + \alpha \nabla [\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i|s_i)]$$

Approx cross-entropy method

- Initialize NN weights **nn = MLPRegressor(...)**

- Loop:
  - Sample N sessions
  - $Elite = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_k, a_k)]$
  - ***nn.fit(elite_states,elite_actions)***

Approx cross-entropy method

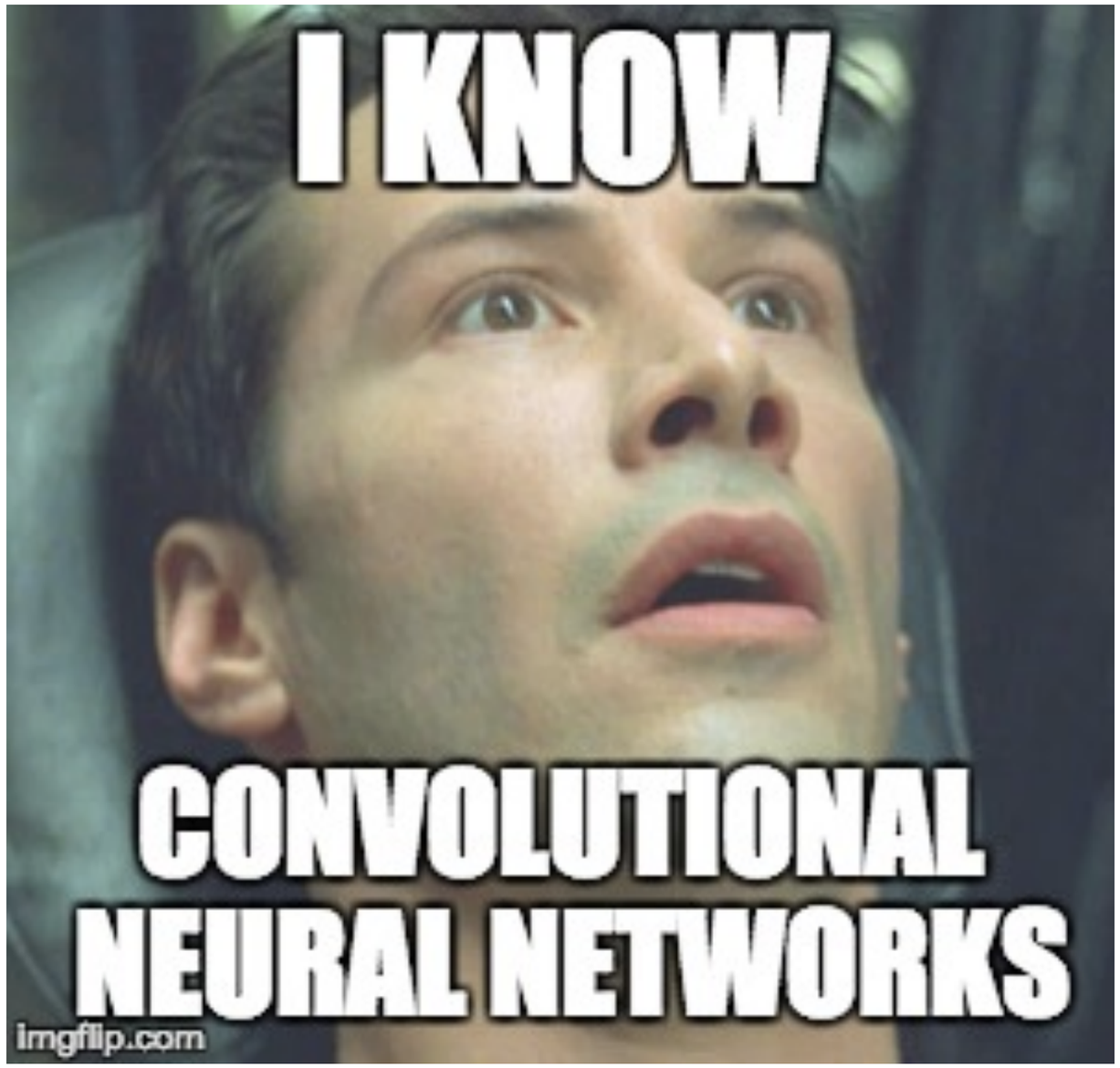
Tricks

- Remember sessions from 3-5 past iterations
  - Threshold and use all of them when training
  - May converge slower if env is easy to solve.
- Regularize with entropy
  - to prevent premature convergence.

- Parallelize sampling
- Use RNNs if partially-observable (later)

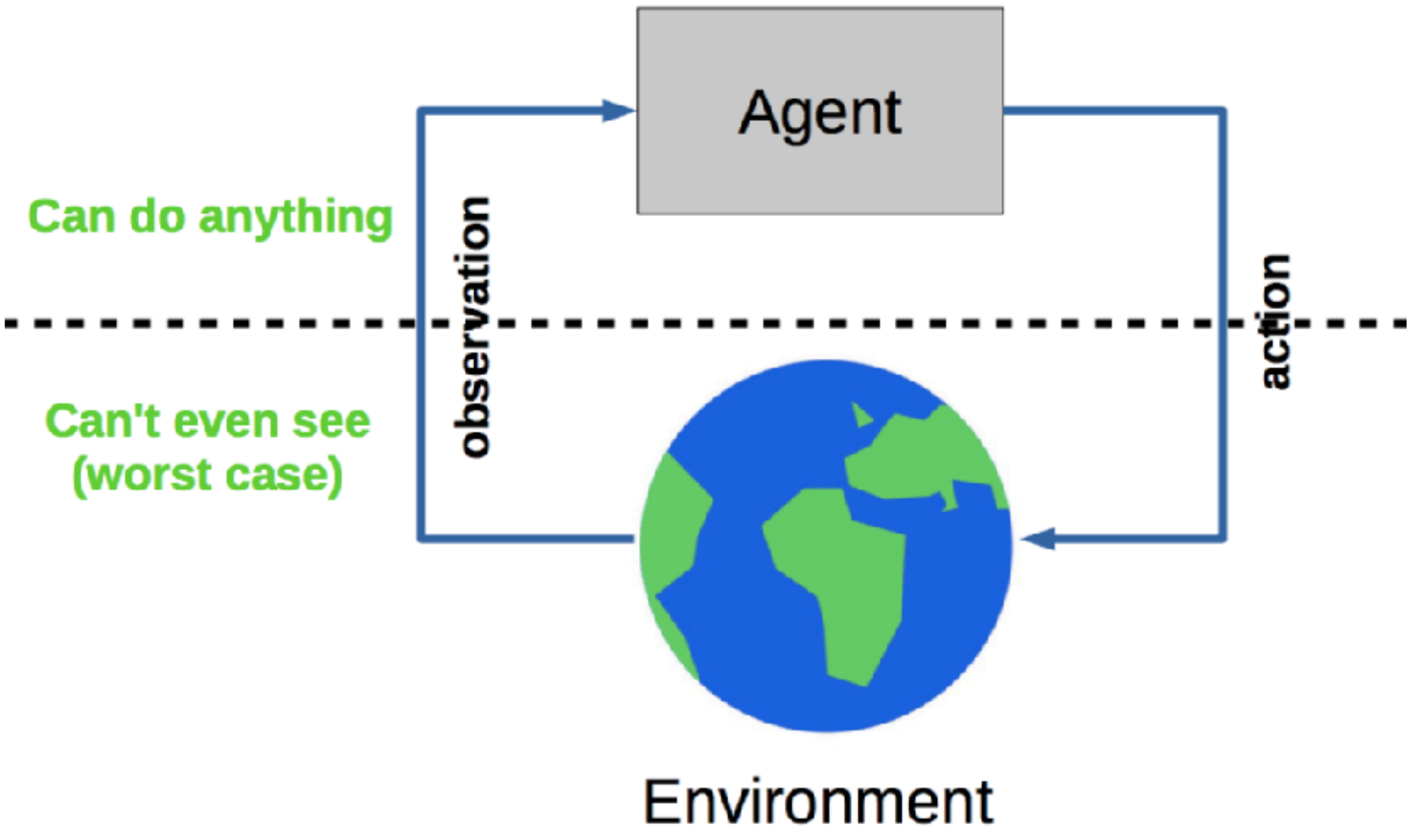
Approx cross-entropy method

Practice time!



Q-learning

Earlier:


Agent
Can do anything
observation
action
Can't even see
(worst case)
Environment

Earlier:

- R(z) – evaluated at the very end
- Metaheuristics (genetic algorithm, etc.)
- Stochastic optimization (crossentropy method)

Earlier:

- Both need a full session to start learning
- Requires a lot of interaction
  - A lot of crashed robots / simulations

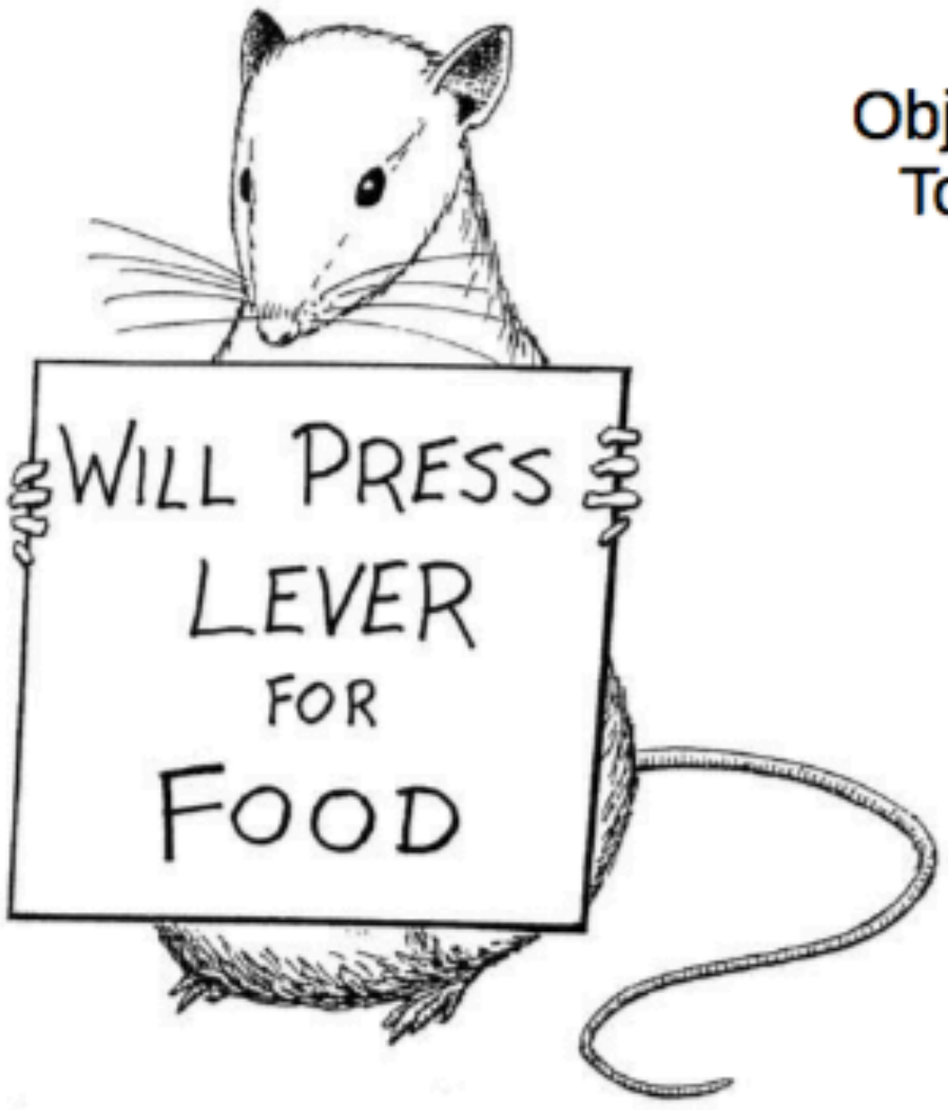
Now:

MDP formalism: reward on each tick

Classic MDP(Markov Decision Process)
Agent interacts with environment

- Environment states: $s \in S$
- Agent actions: $a \in A$
- State transition: $P(s_{t+1}|s_t, a_t)$
- Reward: $r_t = r(s_t, a_t)$

Discounted reward MDP



Objective:
Total action value

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

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) const$$

γ ~ patience
Cake tomorrow is γ as good as now

Reinforcement learning:
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow max$$

Discounted reward MDP

Objective:
Total action value

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

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) const$$

Trivia: which γ corresponds to "only current reward matters"?

Reinforcement learning:

- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow max$$

Discounted reward MDP

Objective:
Total action value

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

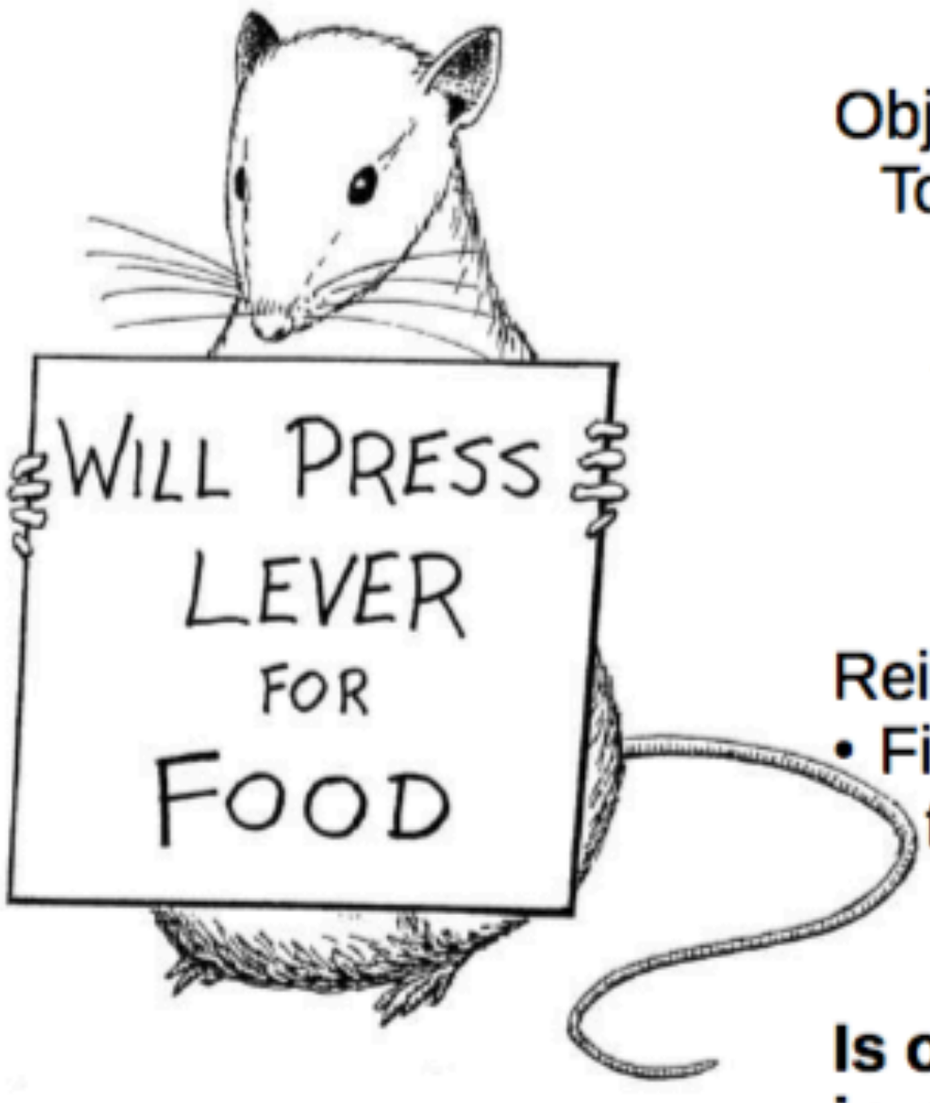
$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) const$$

Trivia: which γ corresponds to "only current reward matters"?

Reinforcement learning:
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow max$$

Discounted reward MDP



Objective:
Total reward

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

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) const$$

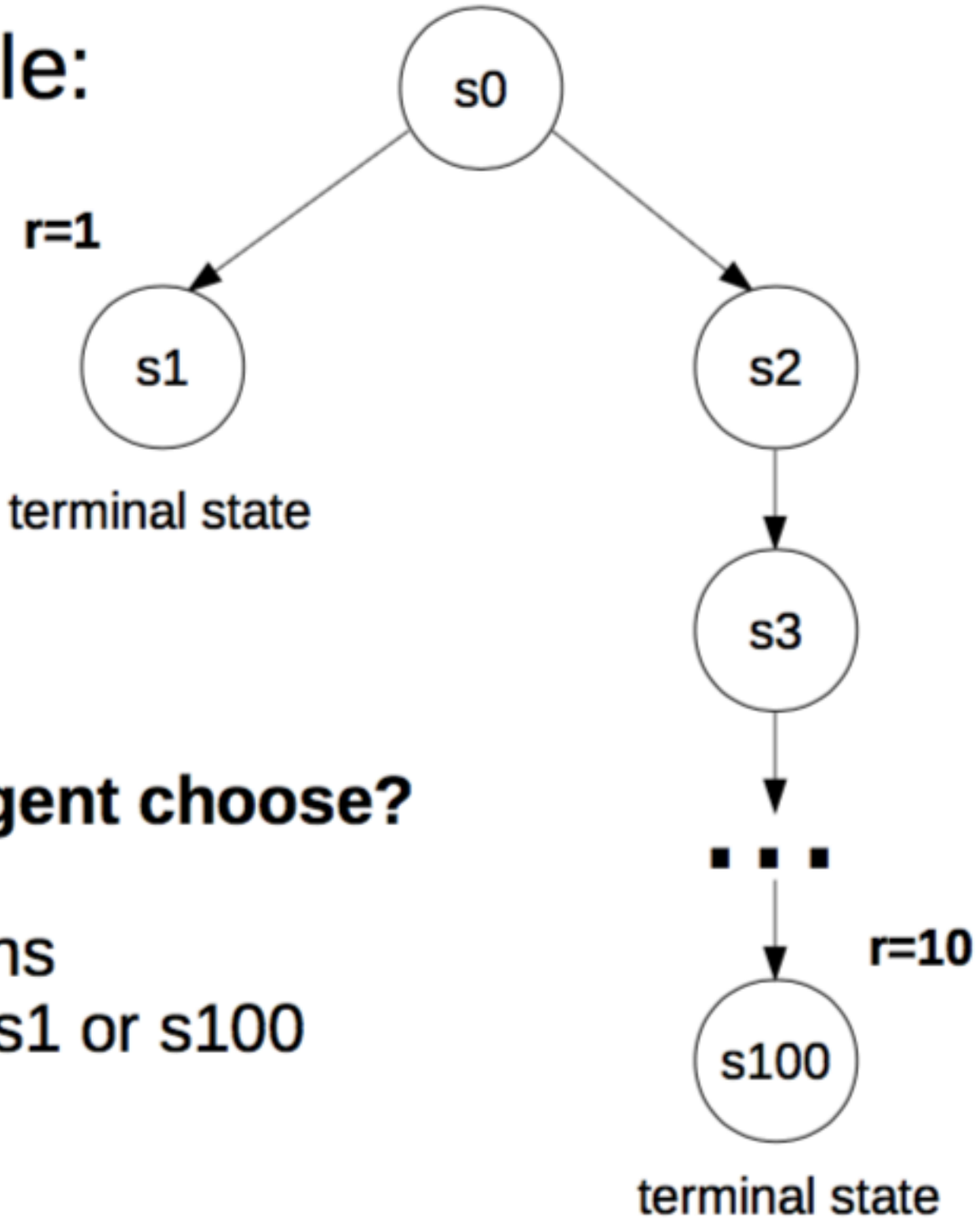
Reinforcement learning:
- Find policy that maximizes the expected reward

$$\pi = P(a|s) : E[R] \rightarrow max$$

Is optimal policy same as it would be in monte-carlo (if we add-up all r_t)?

Discounted reward fails #1

Trivial example:

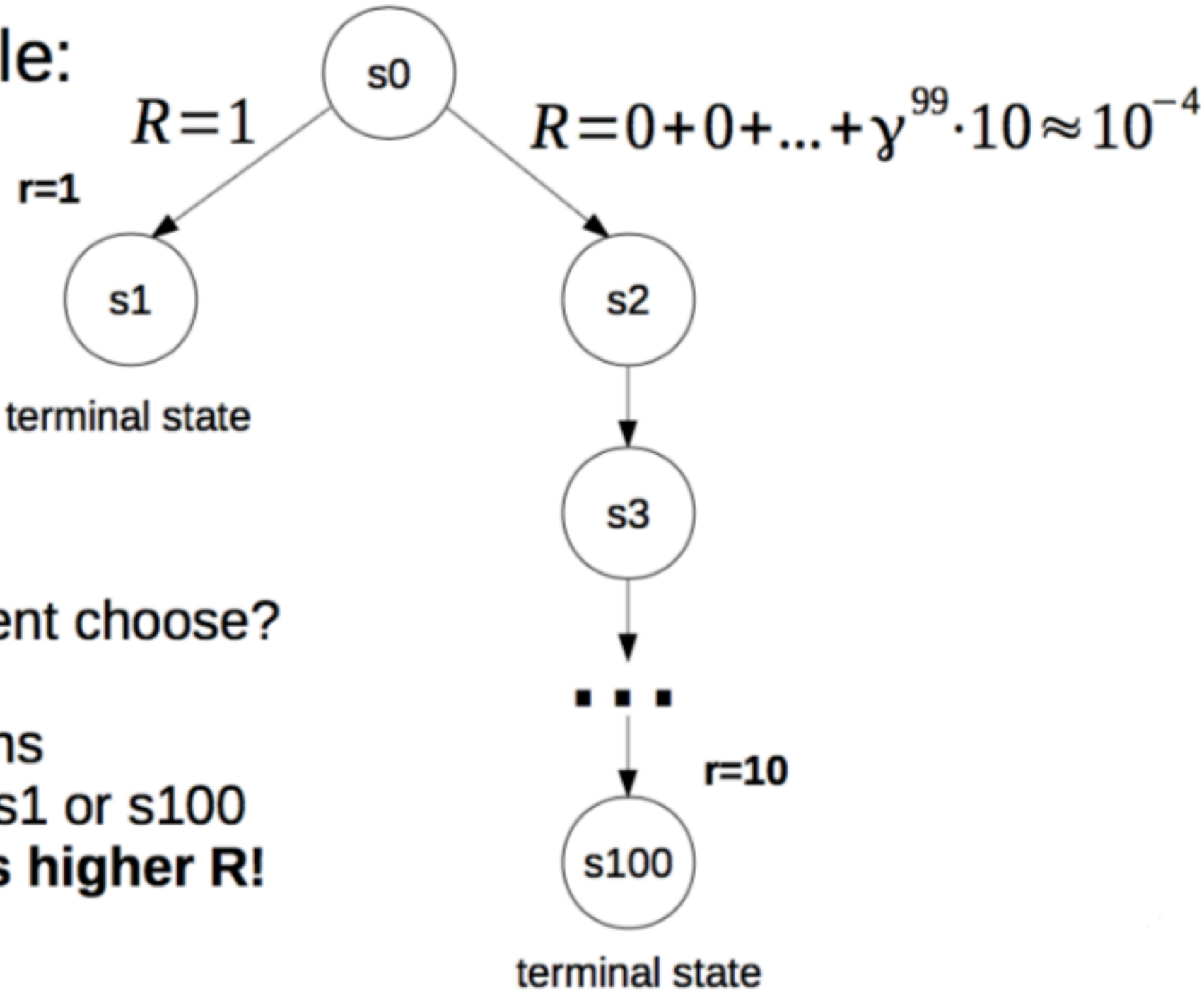


What path will agent choose?

- γ=0.9
- arrows = actions
- game ends at s1 or s100

Discounted reward fails #1

Trivial example:



What path will agent choose?

- γ=0.9
- arrows = actions
- game ends at s1 or s100
- **left action has higher R!**

Discounted reward fails #2

Deephack'17 qualification round, Atari Skiing

- You steer the red guy
- Session lasts ~5k steps

- You get -3~-7 reward each tick
  (faster game = better score)

- At the end of session, you get up to r=-30k
  (based on passing gates, etc.)

- Q-learning with gamma=0.99 fails
  it doesn't learn to pass gates

What's the problem?

http://rl.deephack.me/

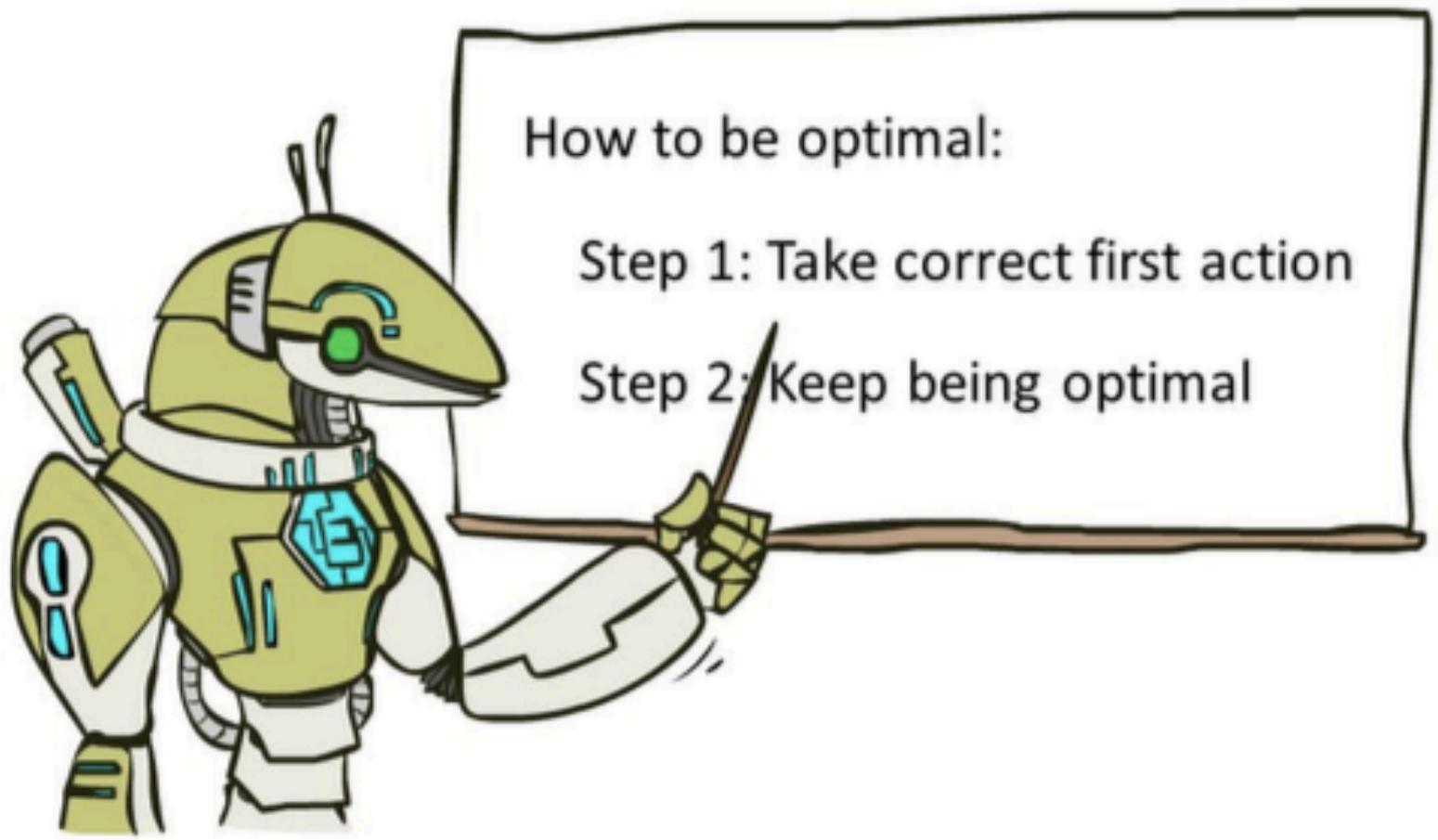
Discounted reward fails #3

CoastRunner7 experiment (openAI)

- You control the boat
- Rewards for getting to checkpoints
- Rewards for collecting bonuses
- What could possibly go wrong?
- “Optimal” policy video:

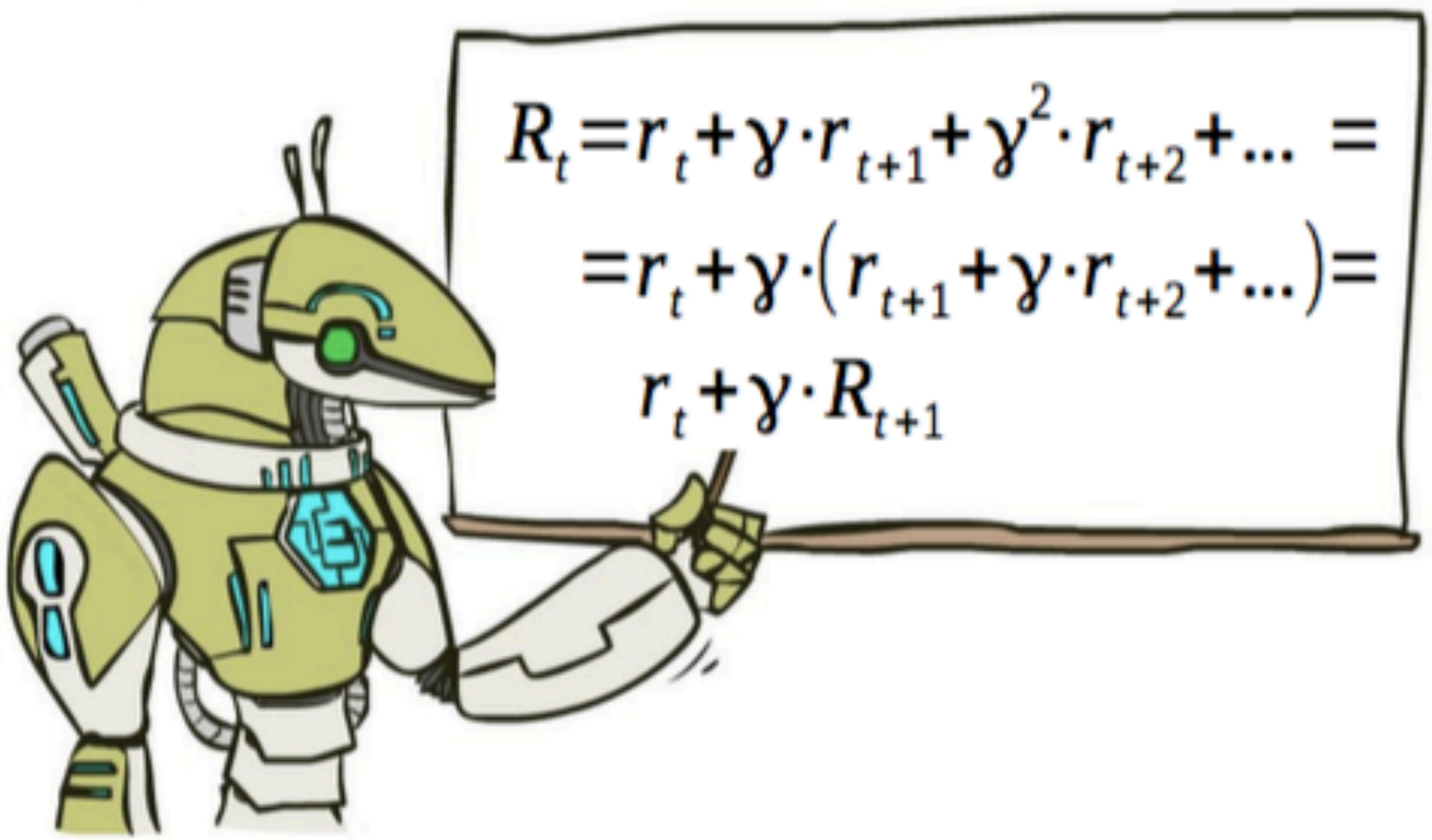
https://openai.com/blog/faulty-reward-functions/

Optimal policy



Recurrent optimal strategy definition

Optimal policy



We rewrite R with sheer power of math!

Value iteration (Temporal Difference)

Idea:

- For each state, obtain V(state)

$$V(s) = max_a [r(s,a) + \gamma \cdot V(s'(s,a))]$$

Definition V(s) – expected total reward R that can be obtained starting from state s under optimal policy

Value iteration (Temporal Difference)

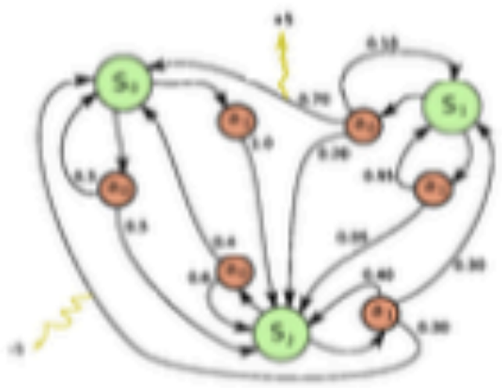
Idea:

- For each state, obtain V(state)

$$V(s) = max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V(s')]$$

Stochastic
action outcome

Trivia: if we know the exact V(s) for all states, how do we determine the best actions?



Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$

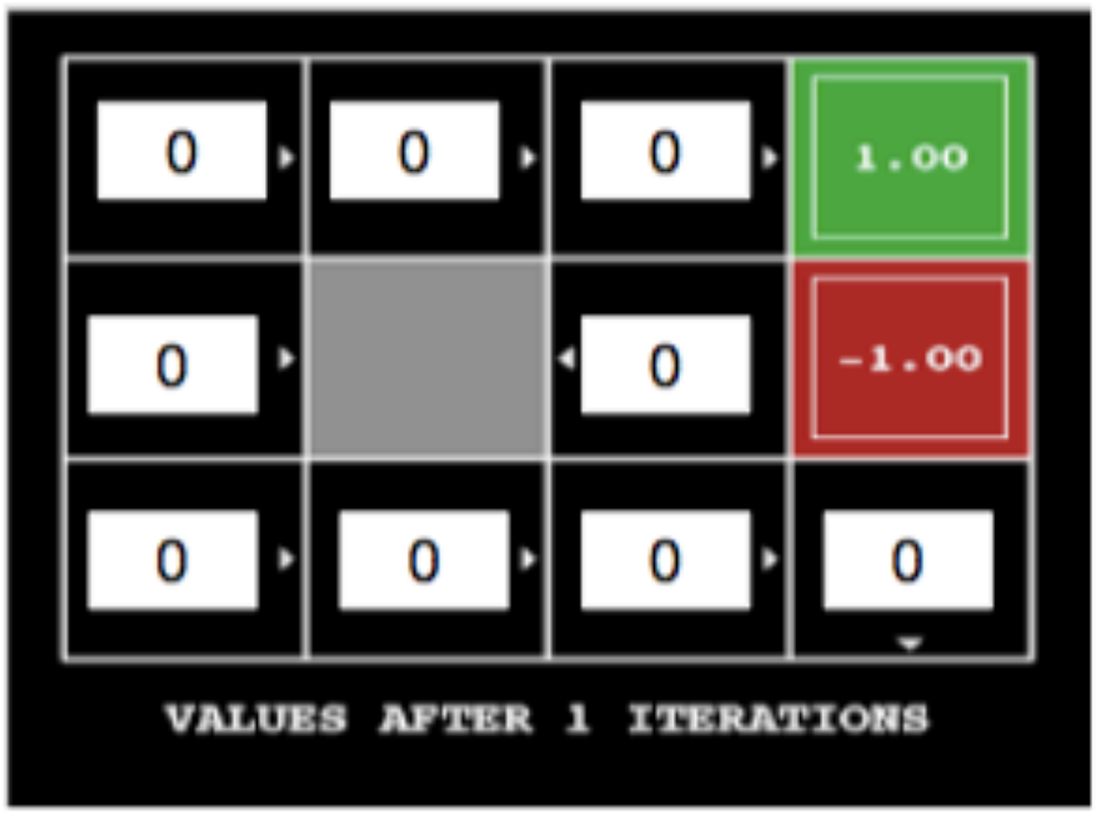
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



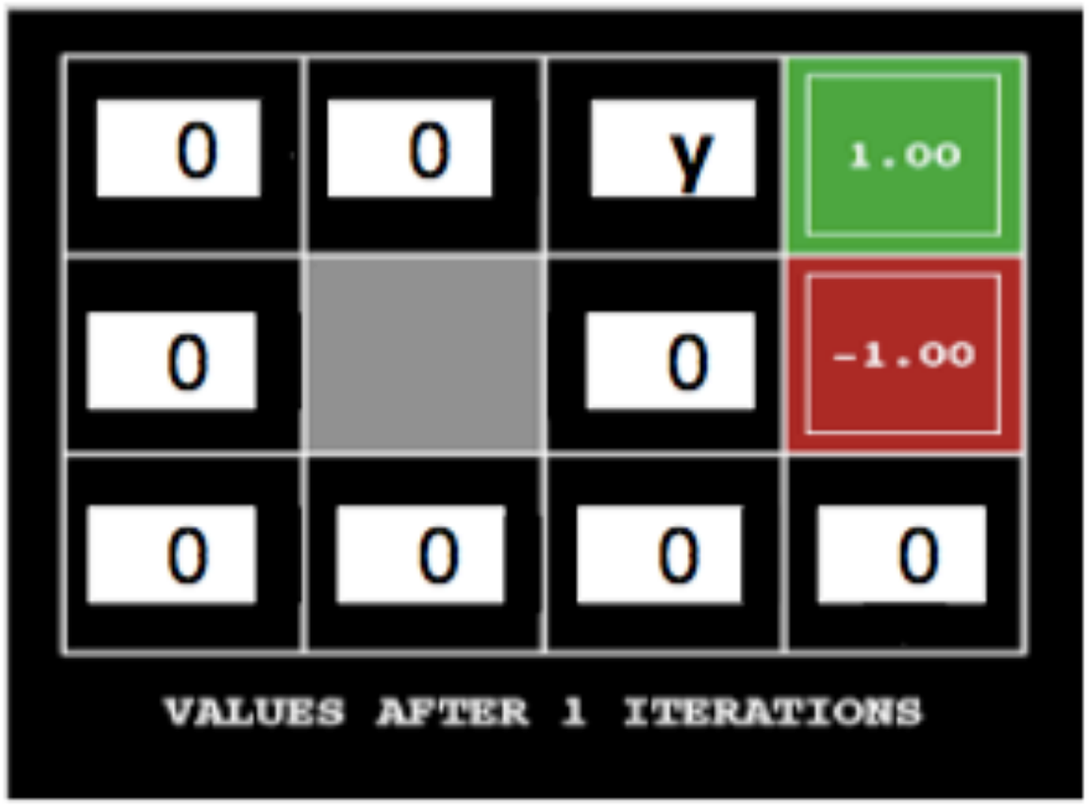
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



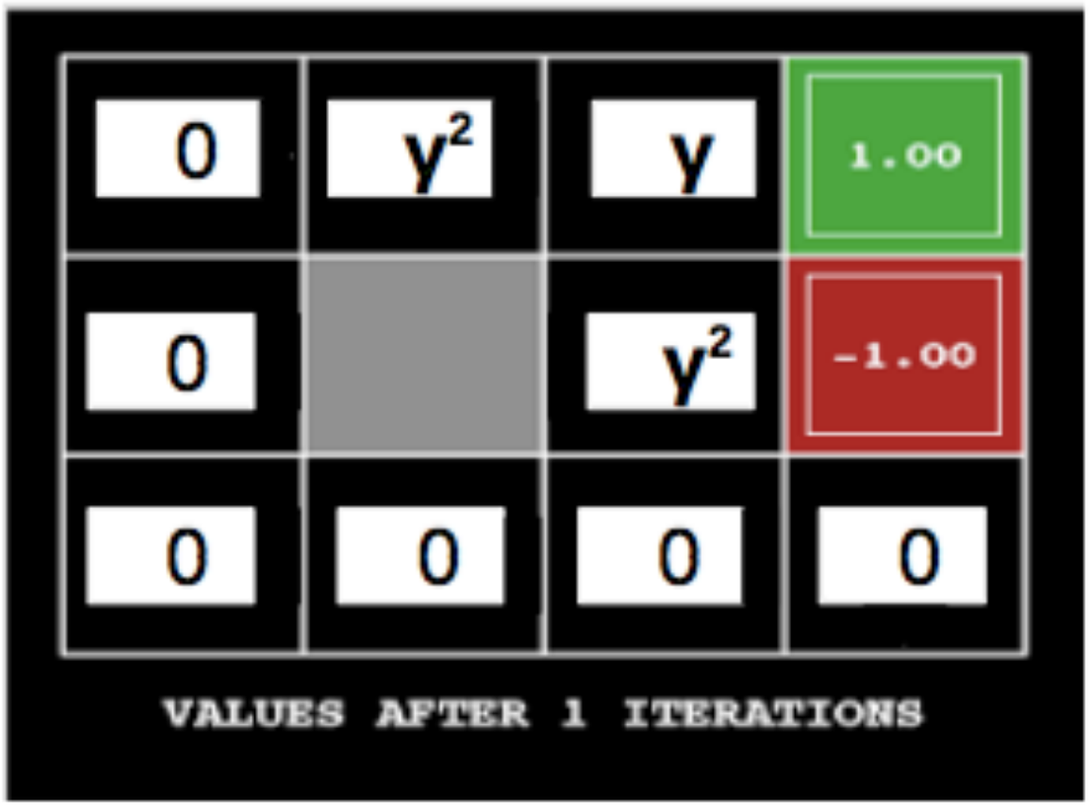
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



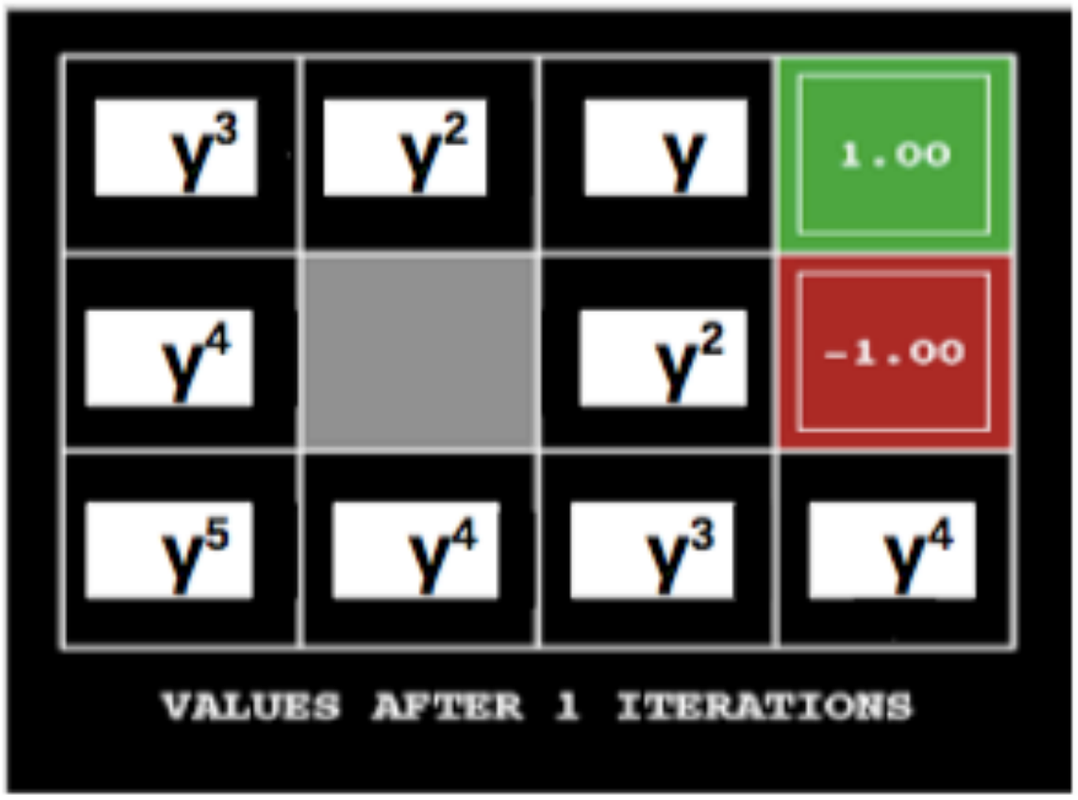
Value iteration (TD)

Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



Value iteration (TD)

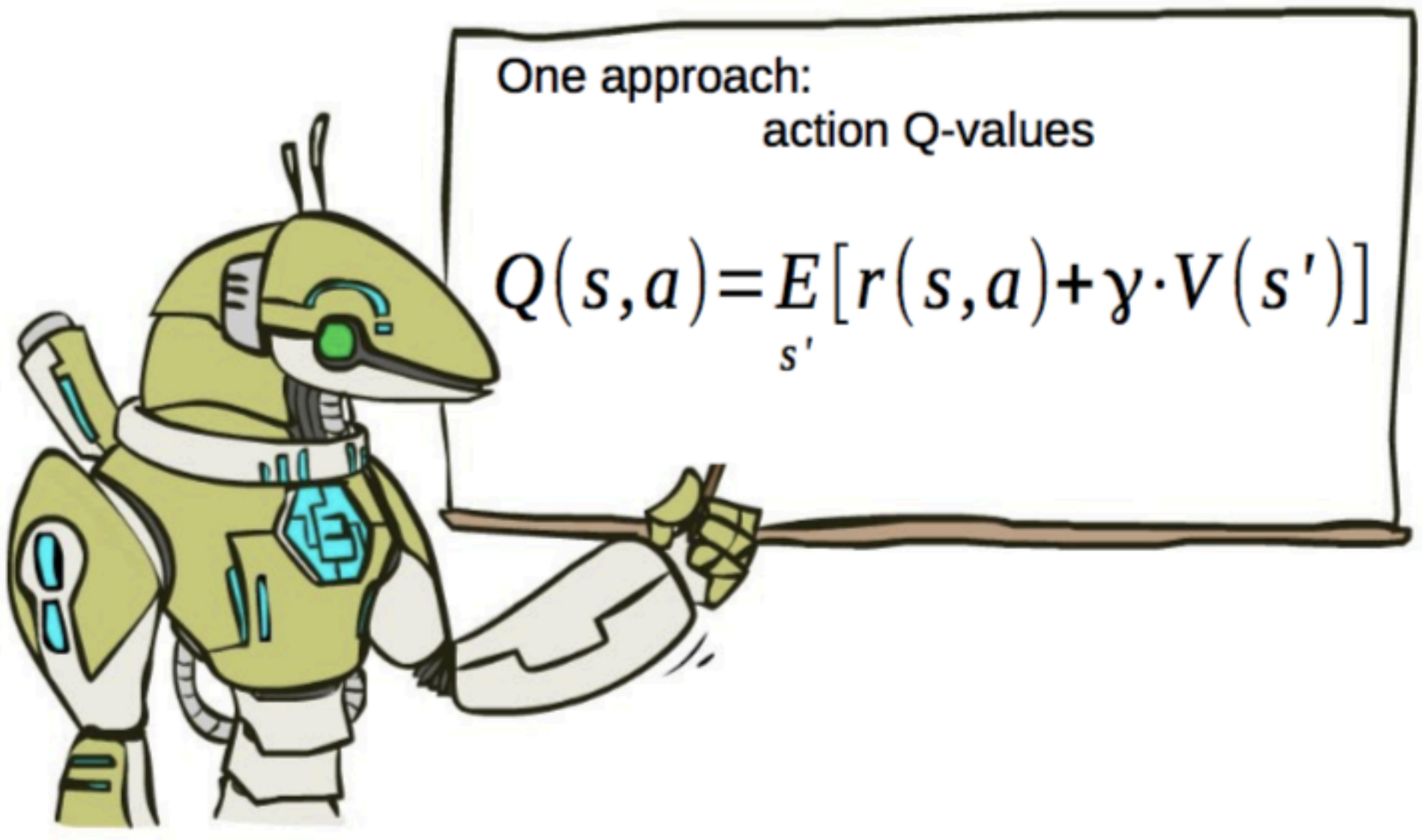
Idea:

- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a[r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$

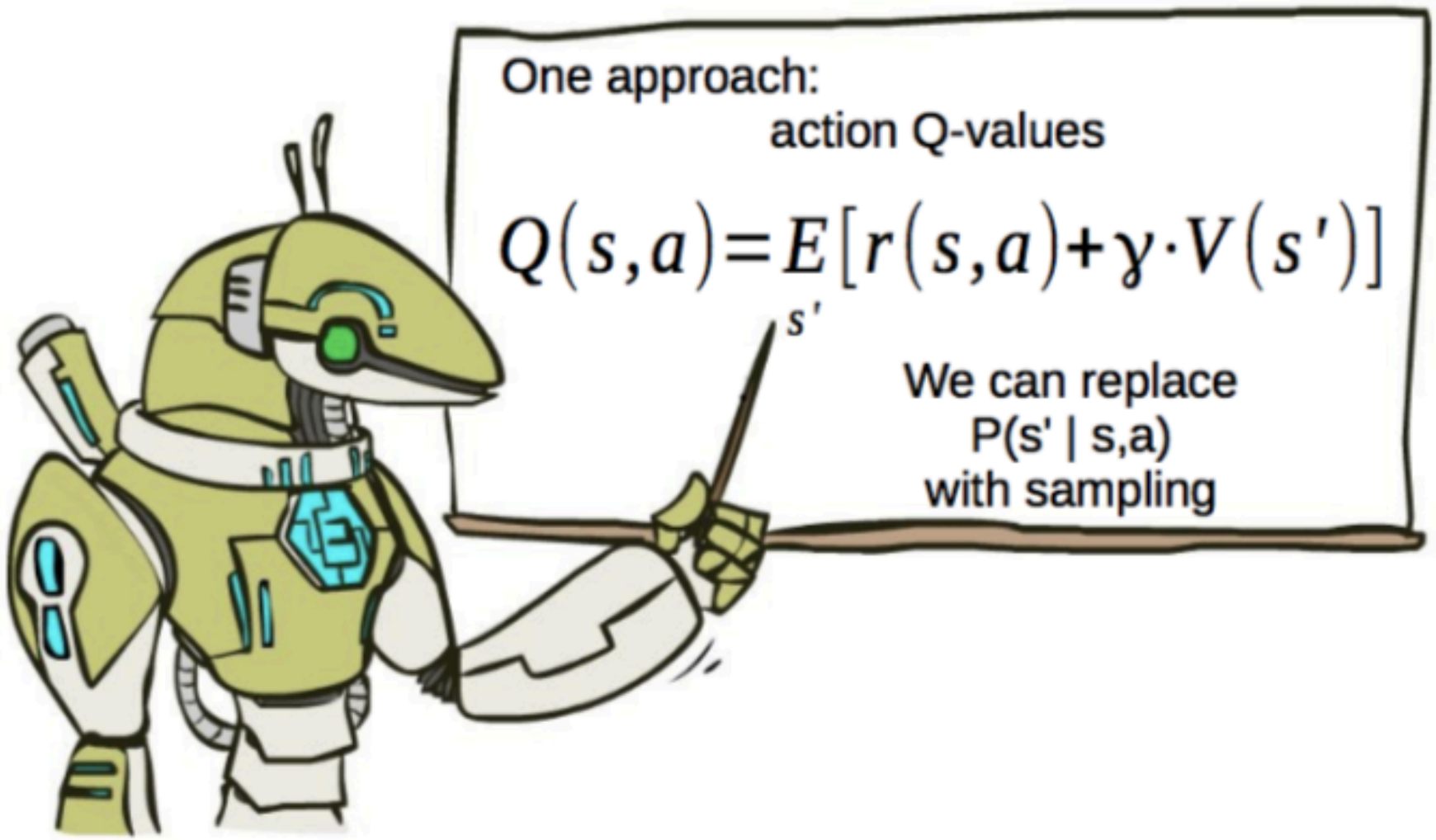
From V to Q



Action value Q(s,a) is the expected total reward **R** agent gets from state **s** by taking action **a** and following policy **π** from next state.

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

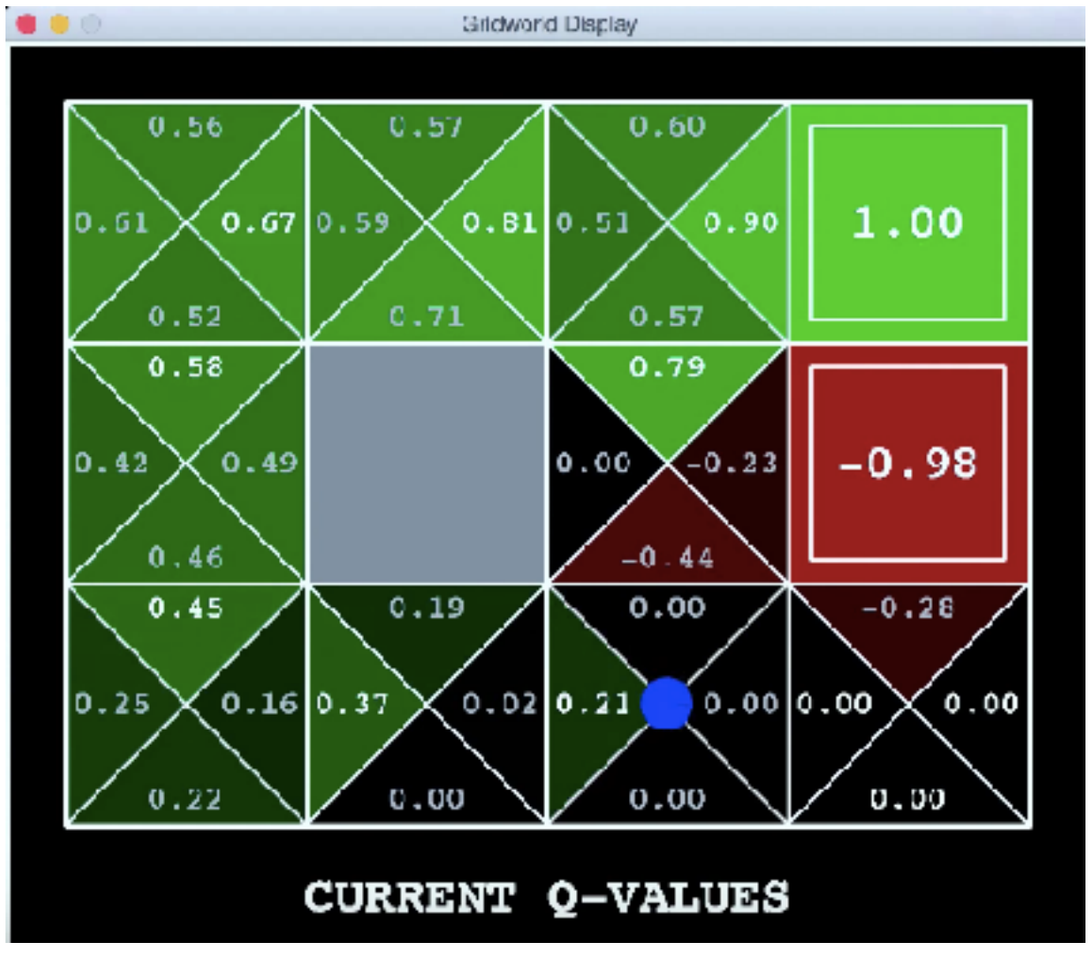
From V to Q

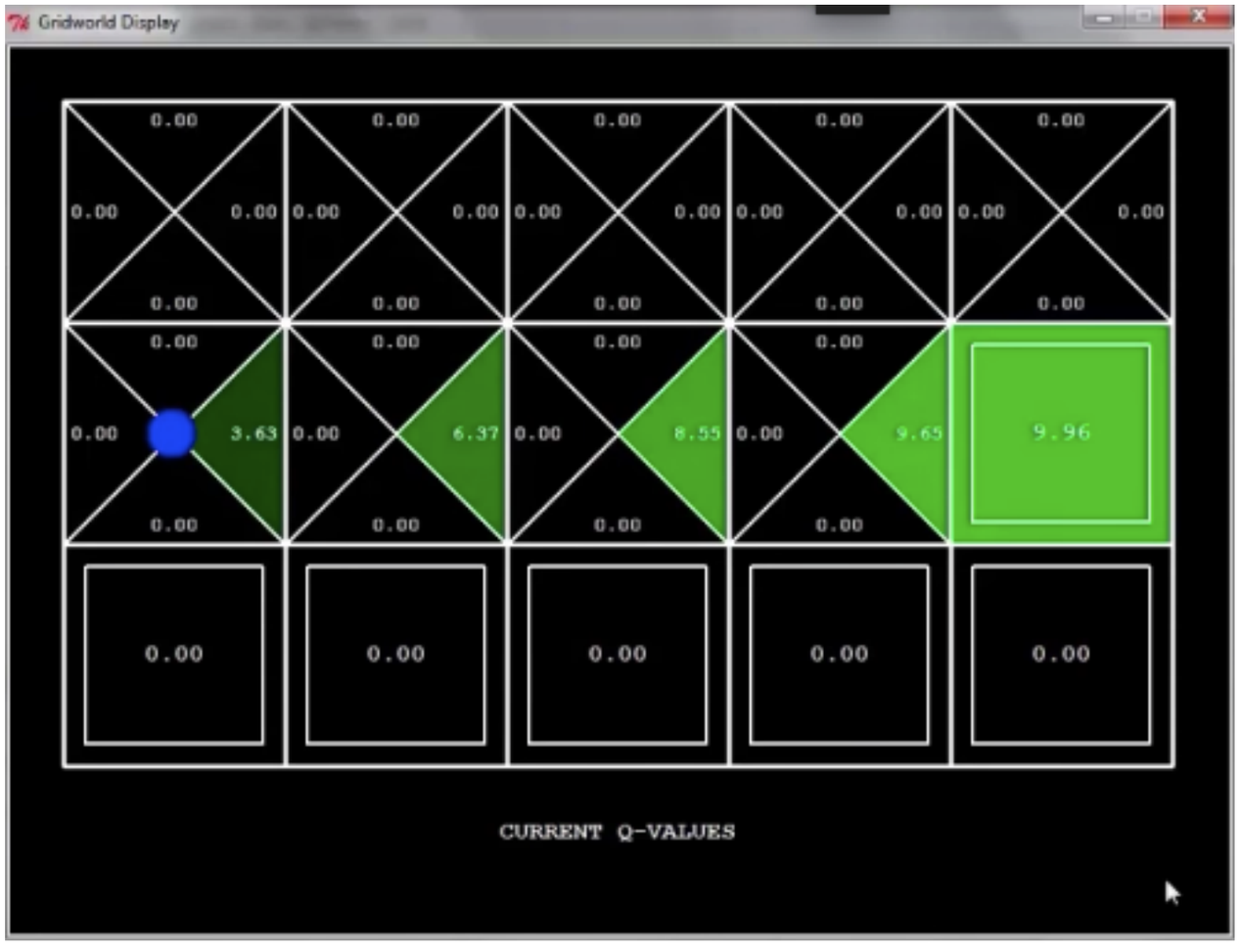


$$Q(s_t,a_t)\leftarrow\alpha\cdot(r_t+\gamma\cdot max_{a'}Q(s_{t+1},a'))+(1-\alpha)Q(s_t,a_t)$$

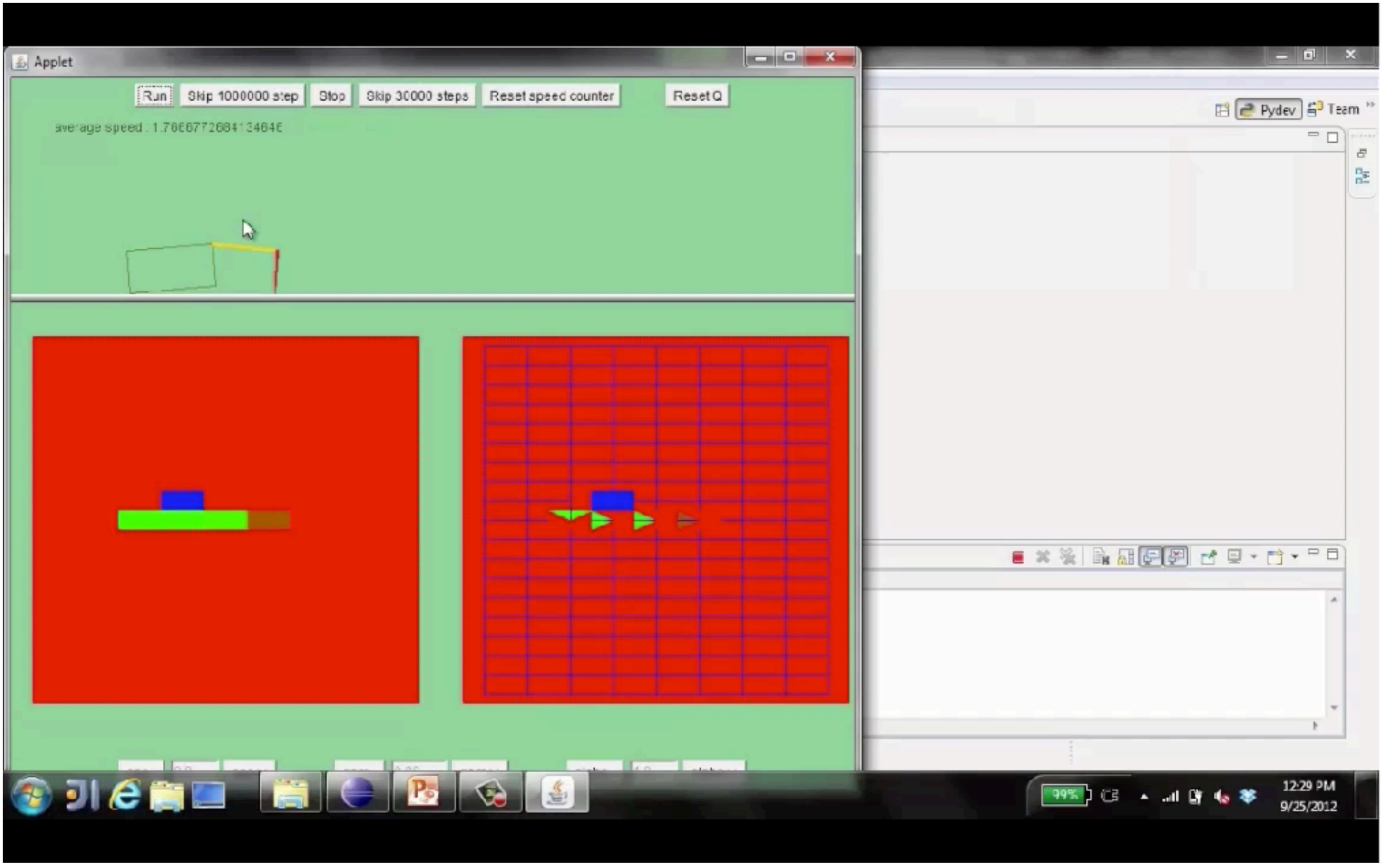
$$\pi(s):argmax_a Q(s,a)$$

From V to Q

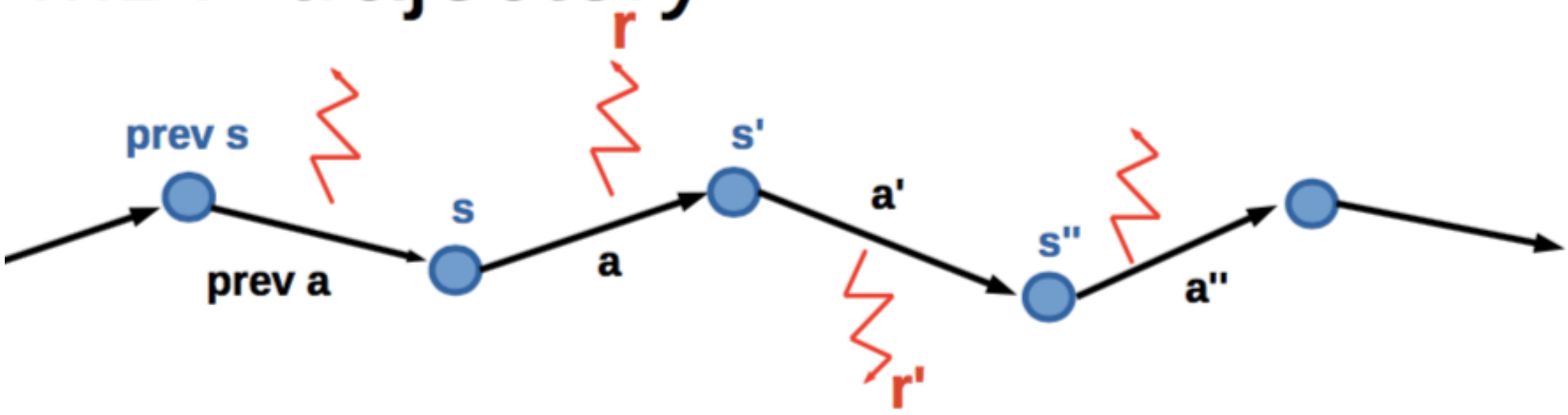

Gridworld Display
0.56
0.61
0.67
0.52
0.57
0.59
0.81
0.71
0.60
0.51
0.90
0.57
1.00
0.58
0.42
0.49
0.46
0.79
0.00
-0.23
-0.44
-0.98
0.45
0.25
0.16
0.22
0.19
0.37
0.02
0.00
0.21
0.00
0.00
-0.28
0.00
0.00
CURRENT Q-VALUES


Gridworld Display
0.00
3.63
6.37
8.55
9.65
9.96
CURRENT Q-VALUES

From V to Q

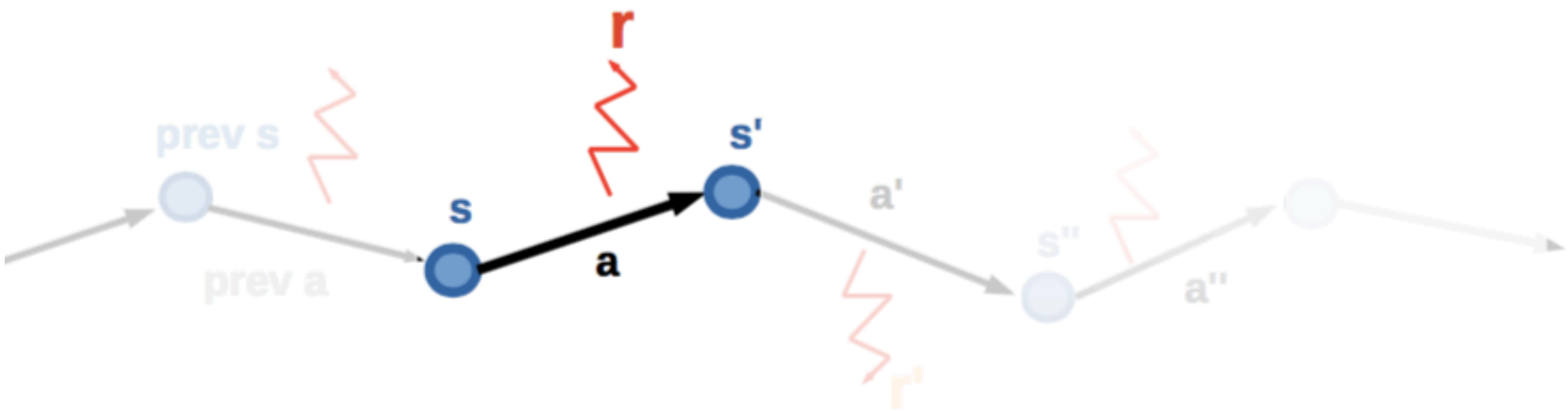

Applet
Run
Skip 1000000 step
Stop
Skip 30000 steps
Reset speed counter
Reset Q
average speed : 1.7666772684134864E
Pydev
Team
99%
12:29 PM
9/25/2012

MDP trajectory



- sample sequence of
  - states (s)
  - actions (a)
  - rewards (r)

Q-learning

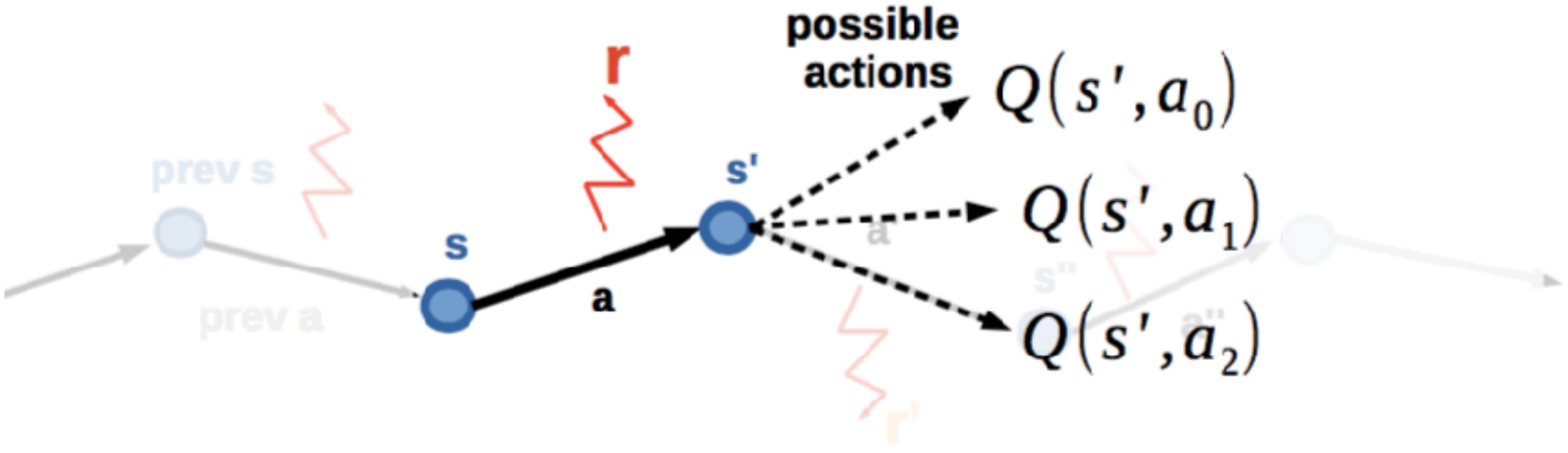


$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$

Loop:

- Sample <**s**,**a**,**r**,**s'**> from env

Q-learning



$$\forall s \in S, \forall a \in A, Q(s,a) \leftarrow 0$$

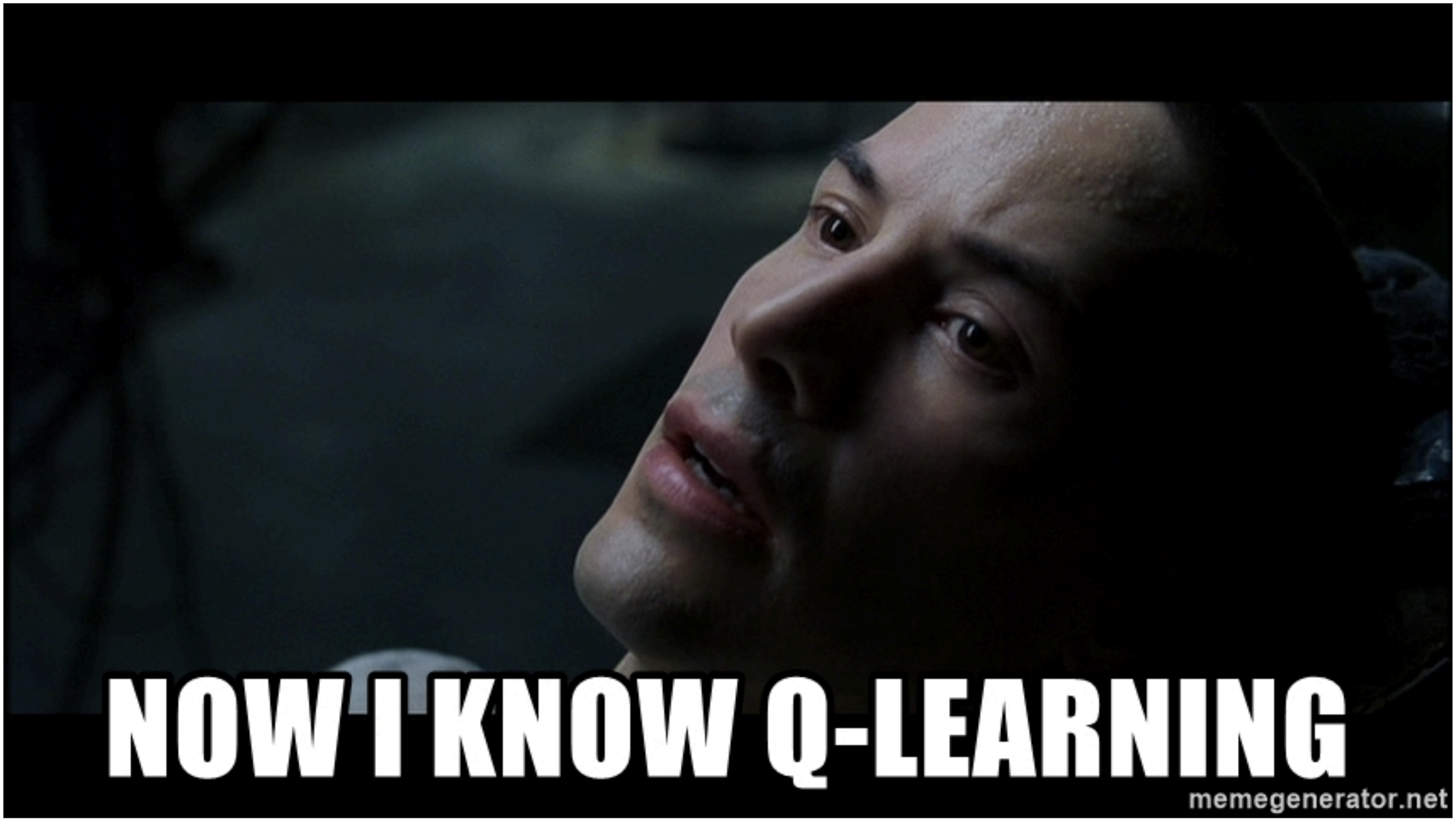
Loop:

- Sample <**s**,**a**,**r**,**s'**> from env

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

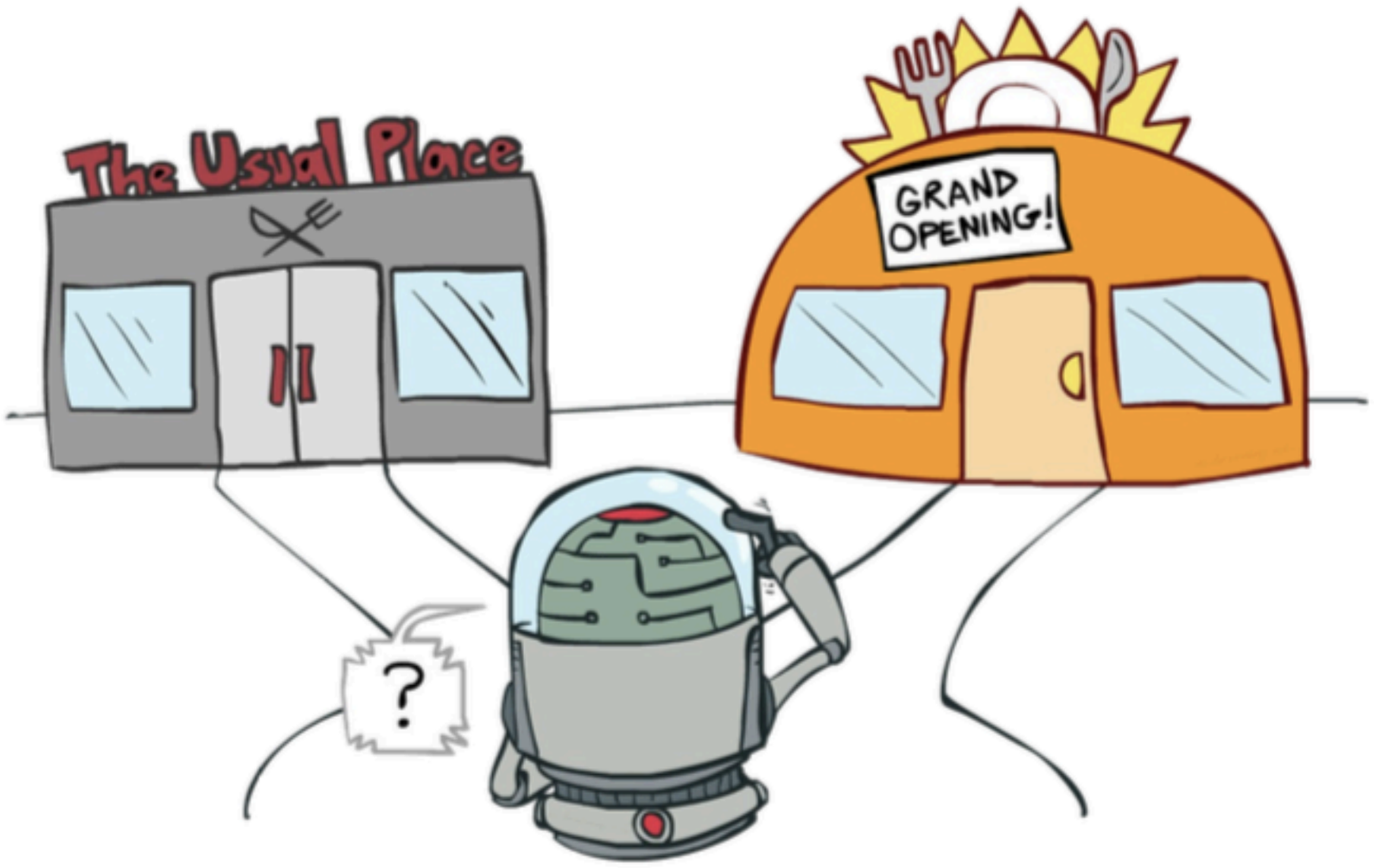
- Update $Q(s,a) \leftarrow \alpha \cdot \hat{Q}(s,a) + (1-\alpha) Q(s,a)$

Practice time!



Exploration Vs Exploitation

Balance between using what you learned and trying to find something even better



Exploration Vs Exploitation

Strategies:

- ε-greedy
  - With probability ε take a uniformly random action; otherwise take optimal action.
- Softmax

  Pick action proportional to softmax of shifted normalized Q-values.

$$P(a) = softmax\left(\frac{Q(a)}{\tau}\right)$$

Nuts and bolts: discount

- Effective horizon $1+\gamma+\gamma^2+\ldots=\frac{1}{(1-\gamma)}$

Heuristic: your agent stops giving a damn in *this many* turns.

Typical values:
- γ=0.9, 10 turns
- γ=0.95, 20 turns
- γ=0.99, 100 turns
- γ=1, infinitely long

Higher γ = less stable algorithm.
γ=1 only works for episodic MDP (finite amount of turns).

Problem:

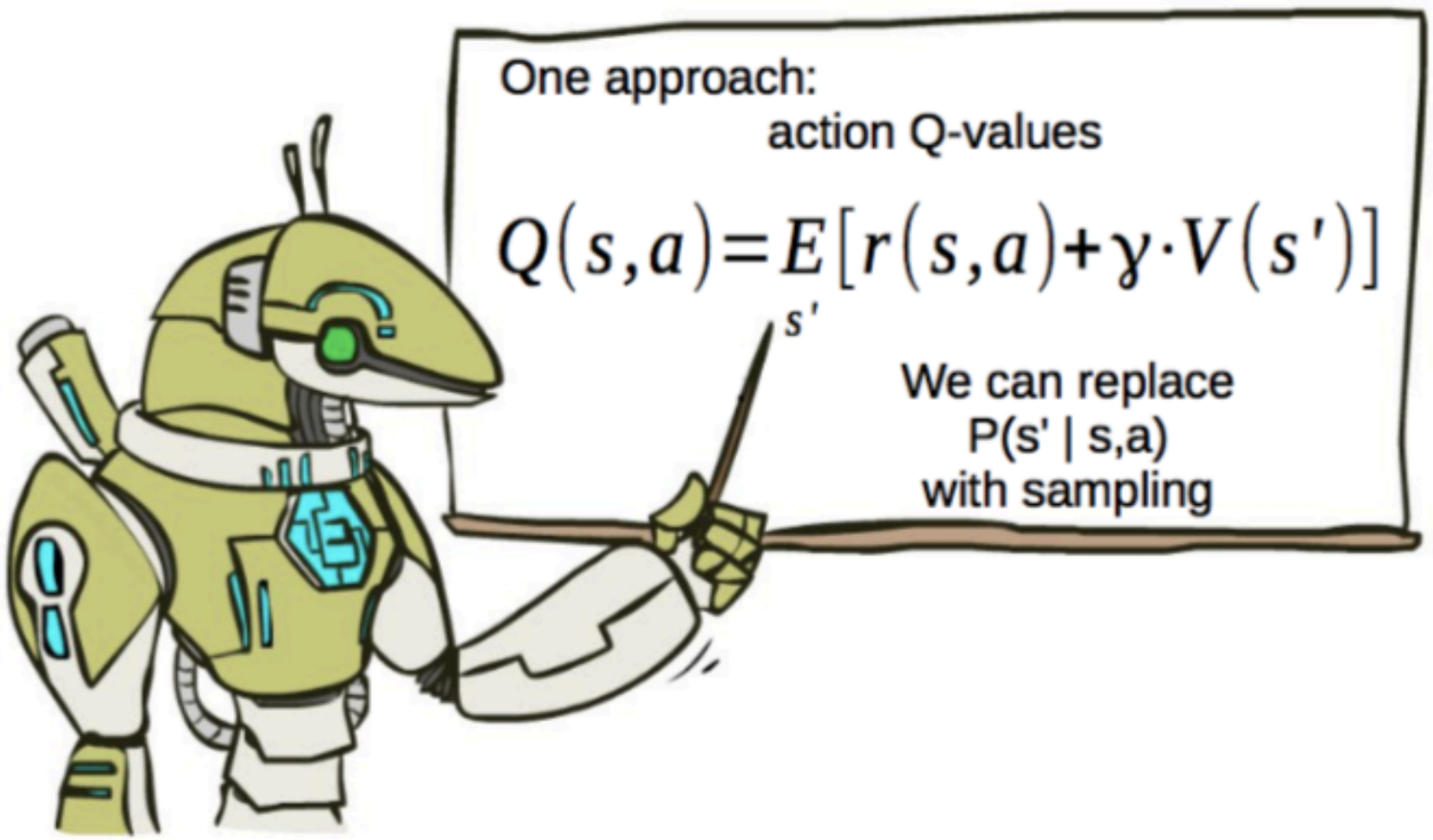
State space is usually large,

sometimes continuous.

And so is action space;

However, states do have a structure, similar states have similar action outcomes.

Recap: Q-learning



$$Q(s_t,a_t) \leftarrow \alpha \cdot (r_t + \gamma \cdot max_{a'} Q(s_{t+1},a')) + (1-\alpha) Q(s_t,a_t)$$

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

From tables to approximations

- Before:
  - For all states, for all actions, remember Q(s,a)

- Now:
  - Approximate Q(s,a) with some function
  - e.g. linear model over state features

$$argmin_{w,b}\left(Q\left(s_t,a_t\right)-\left[r_t+\gamma\cdot max_{a'}Q\left(s_{t+1},a'\right)\right]\right)^2$$

Q-learning as MSE minimization

Given <**s**,**a**,**r**,**s'**> minimize

$$L=[Q(s_t,a_t)-Q^{true}(s_t,a_t)]^2$$

$$L\approx[Q(s_t,a_t)-(r_t+\gamma\cdot max_{a'}Q(s_{t+1},a'))]^2$$

Q-learning as MSE minimization

Given <**s**,**a**,**r**,**s'**> minimize

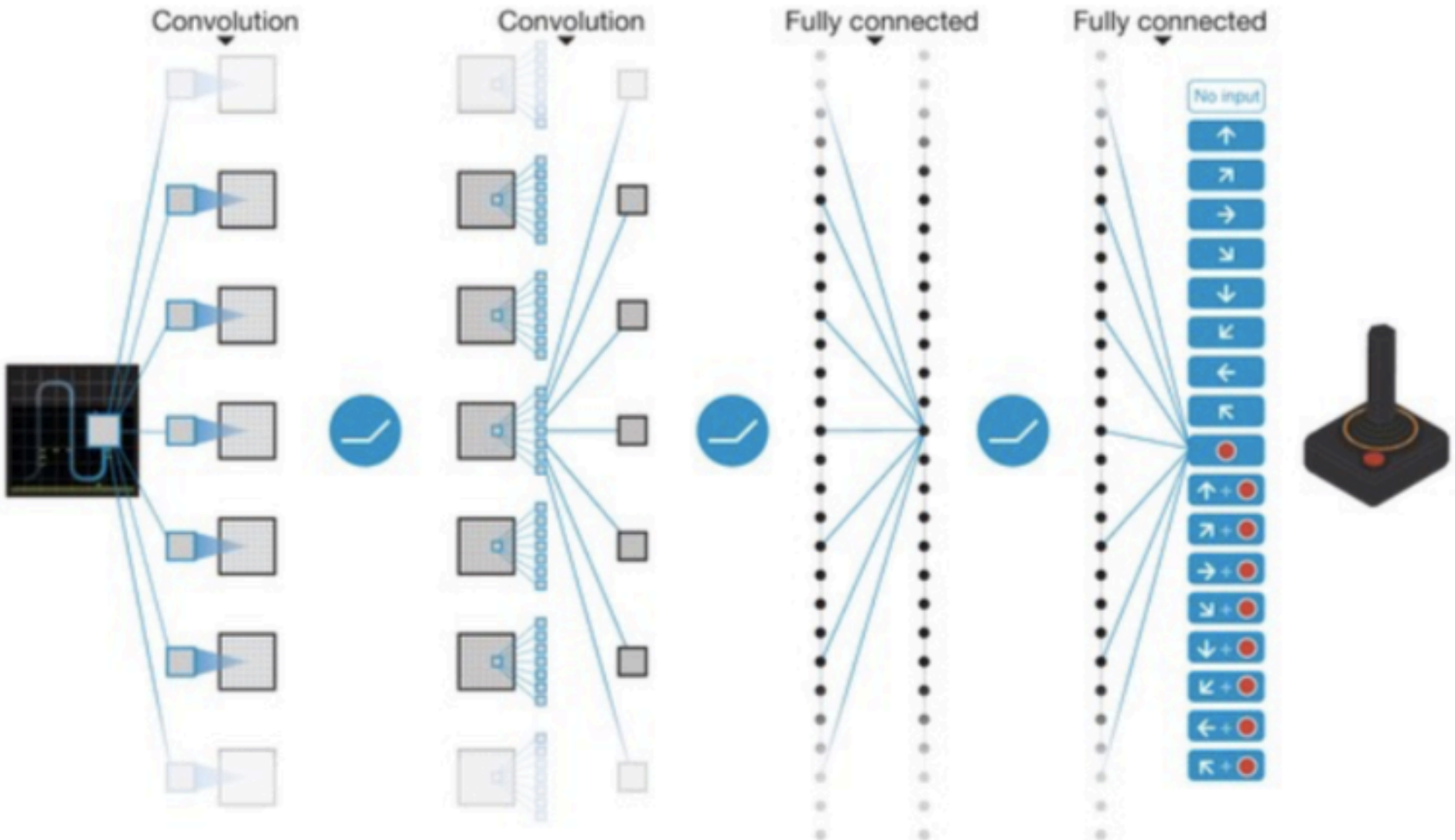
$$L=[Q(s_t,a_t)-Q^{true}(s_t,a_t)]^2$$

const

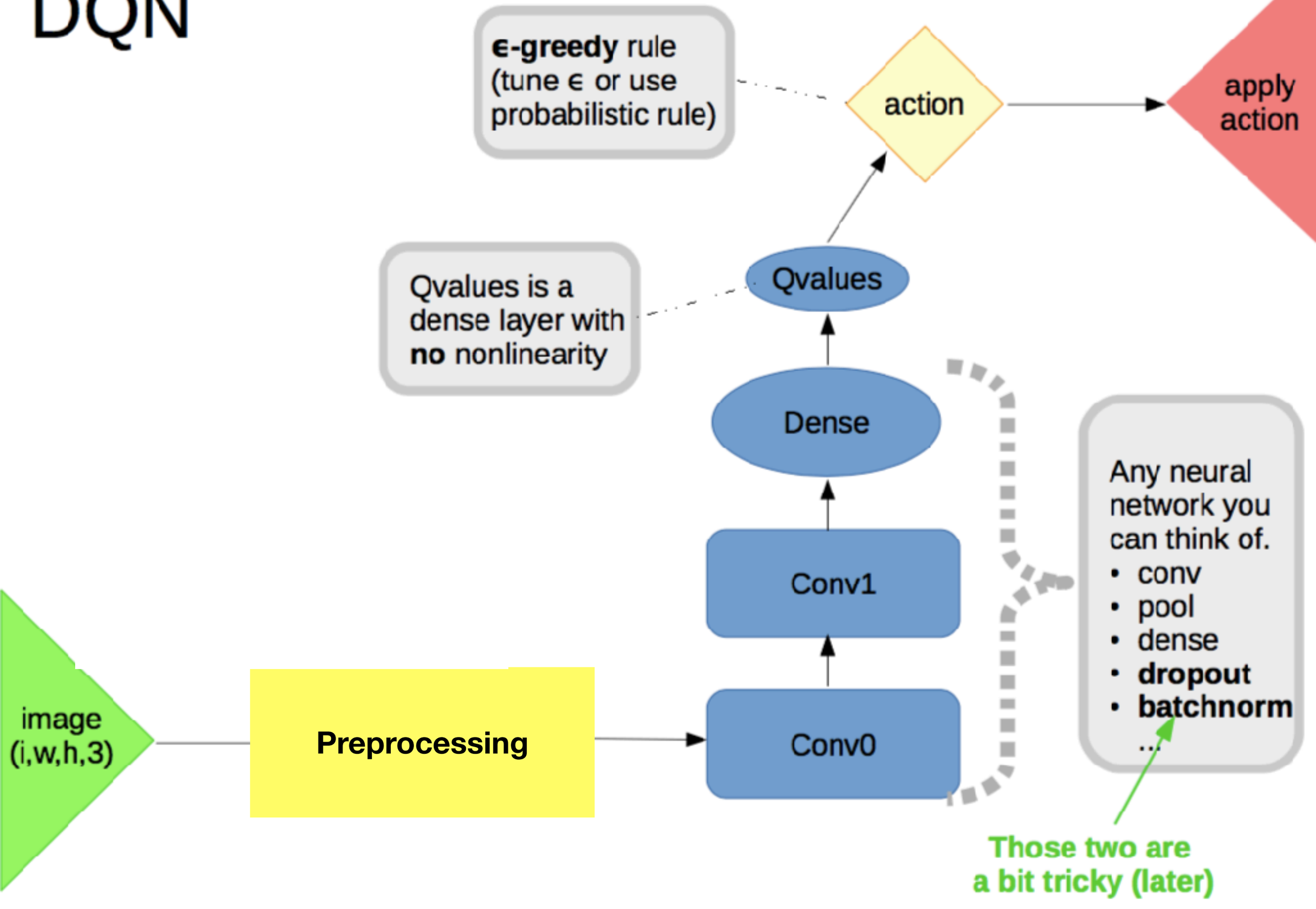
$$L\approx[Q(s_t,a_t)-(r_t+\gamma\cdot max_{a'}Q(s_{t+1},a'))]^2$$

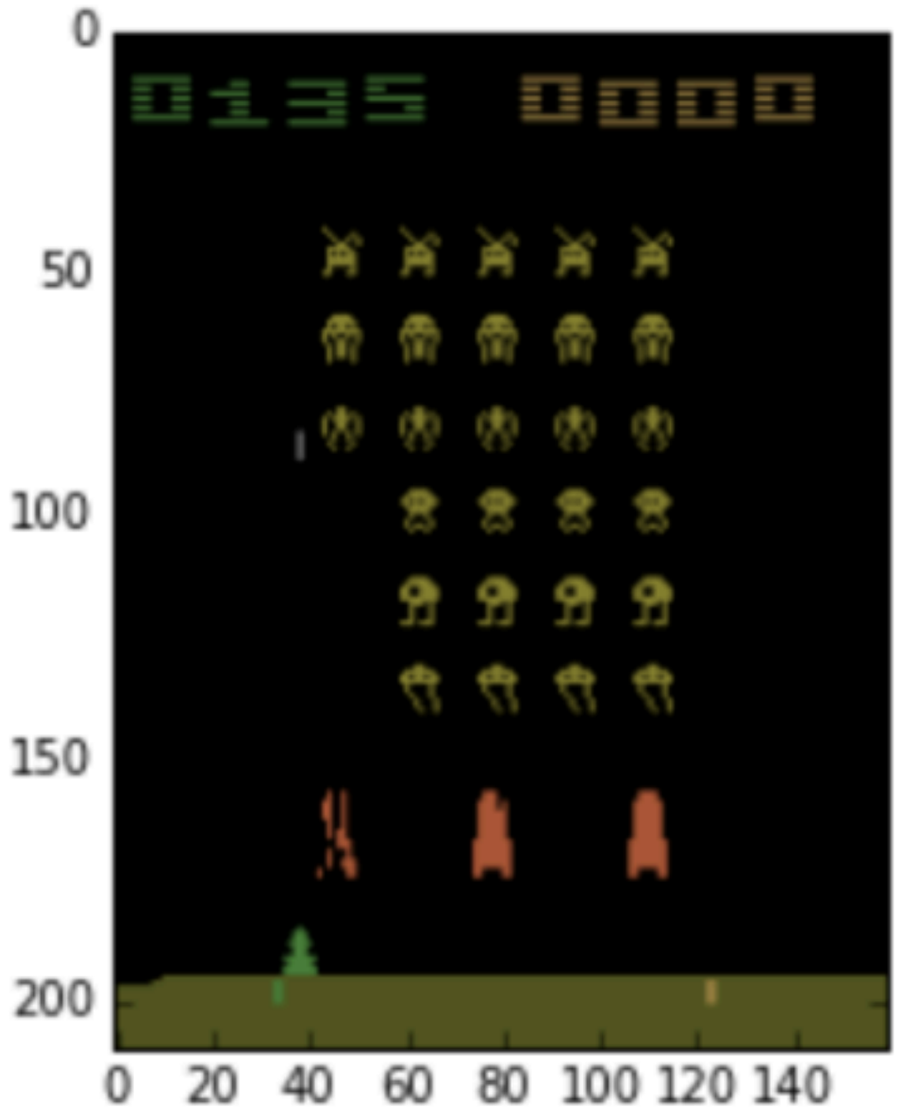
const

Smells like a neural network



DQN


ϵ-greedy rule
(tune ϵ or use
probabilistic rule)
action
apply
action
Qvalues is a
dense layer with
no nonlinearity
Qvalues
Dense
Conv1
Conv0
Any neural
network you
can think of.
• conv
• pool
• dense
• dropout
• batchnorm
...
image
(i,w,h,3)
Preprocessing
Those two are
a bit tricky (later)

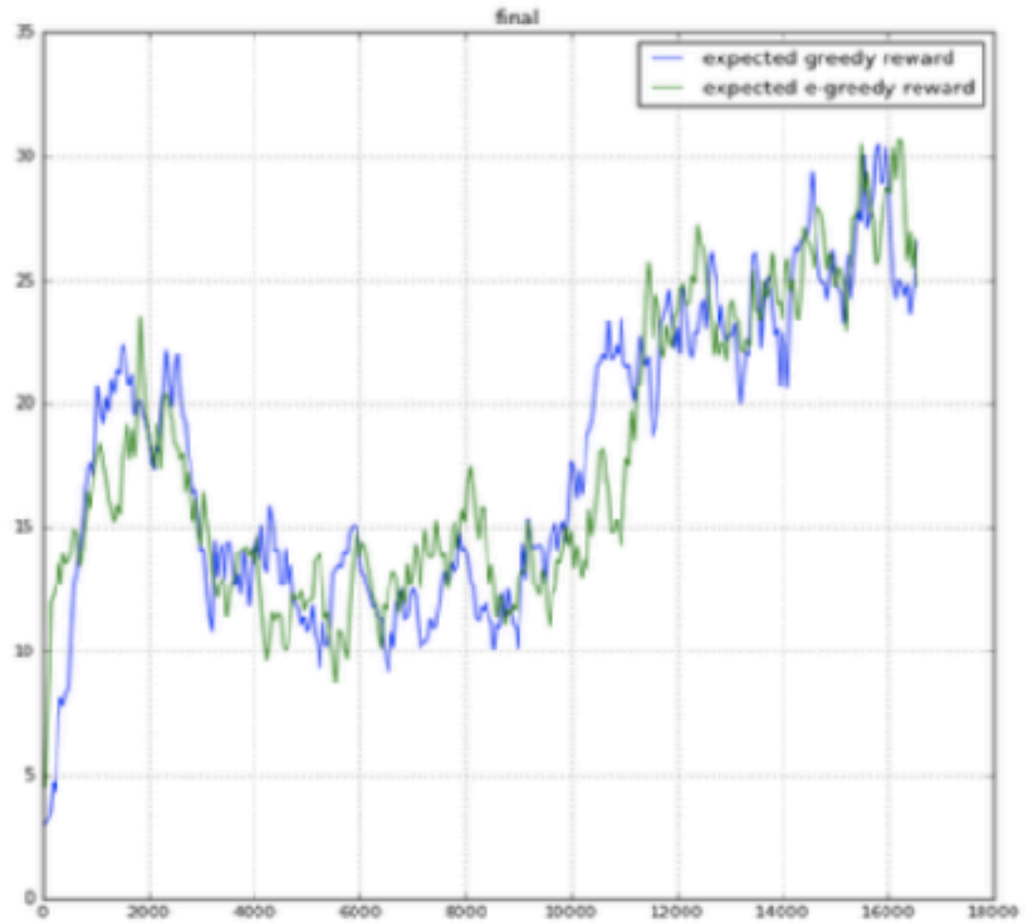


How bad it is if agent spends

next 1000 ticks under the left rock?
(while training)

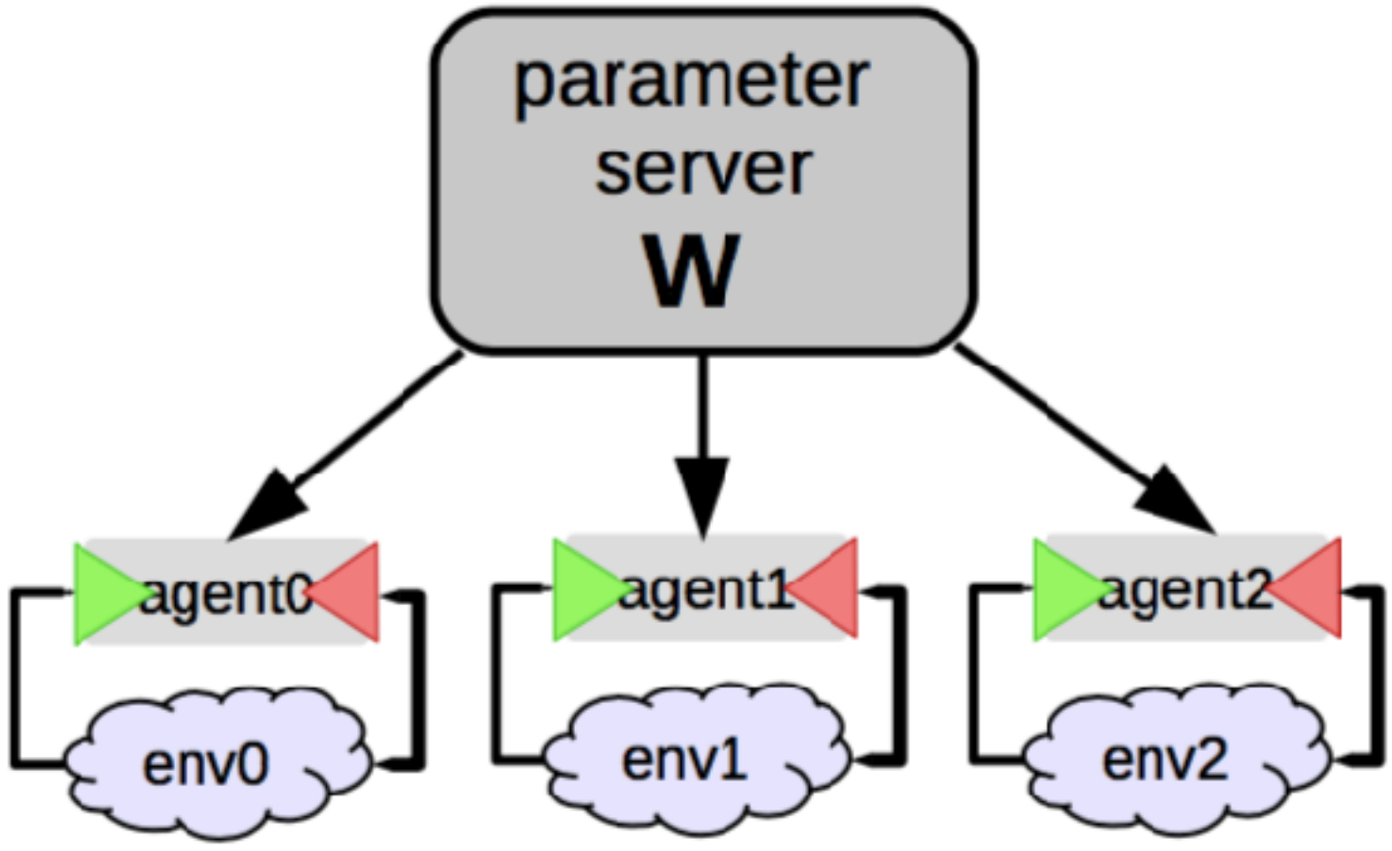
Problem

- Training samples are **not "i.i.d"**,
- Model forgets parts of environment it hasn't visited for some time
- Drops on learning curve
- **Any ideas?**



Multiple agent trick

Idea: Throw in several agents with shared **W**.



- Chances are, they will be exploring different parts of the environment,
- More stable training,
- Requires a lot of interaction

Trivia: your agent is a real robot car. Any problems?

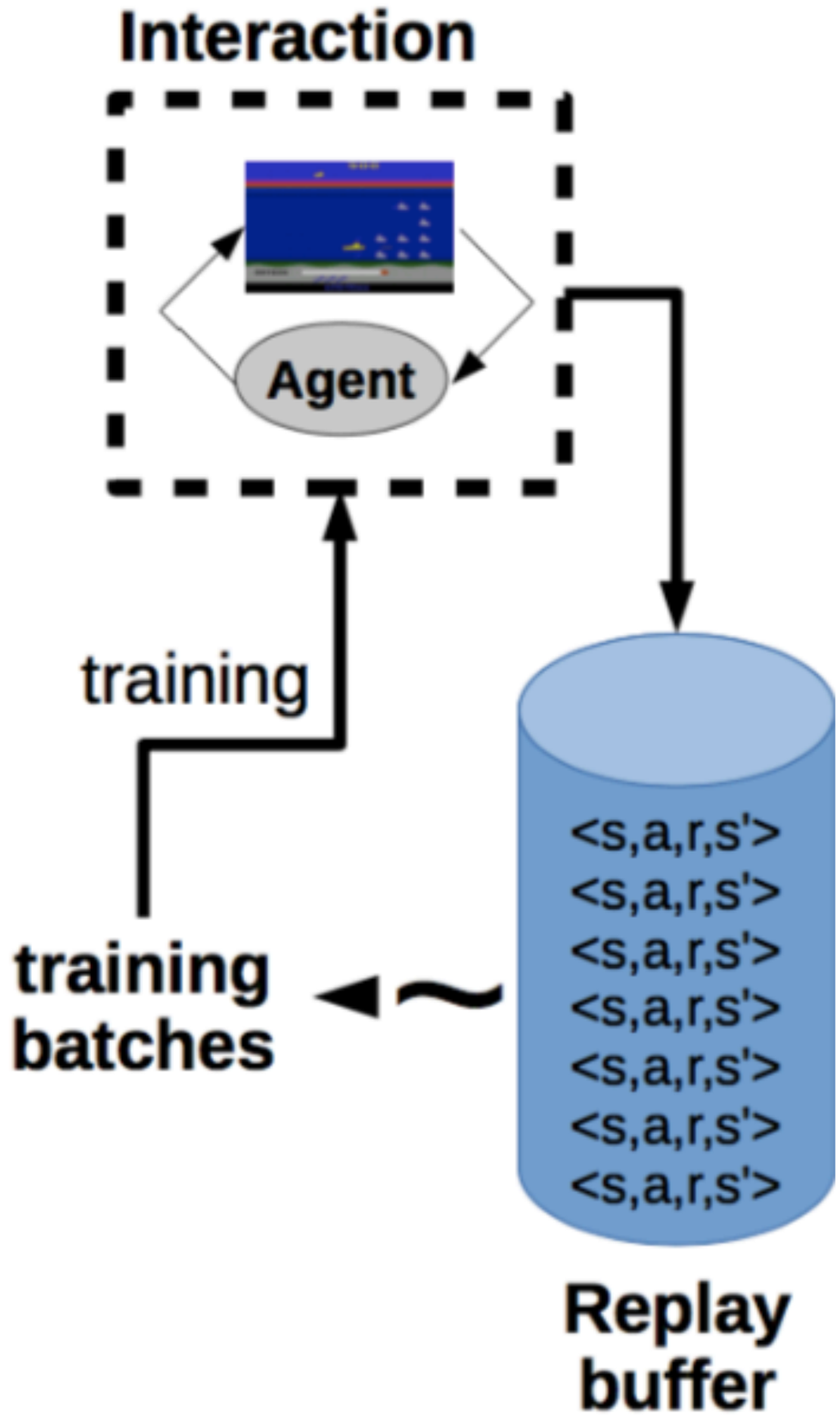
Experience replay

Idea: store several past interactions <*s,a,r,s'*>
Train on random subsamples

- Atari DQN: >10^5 interactions

- Closer to i.i.d
  pool contains several sessions

- Older interactions were obtained under weaker policy



Practice time!