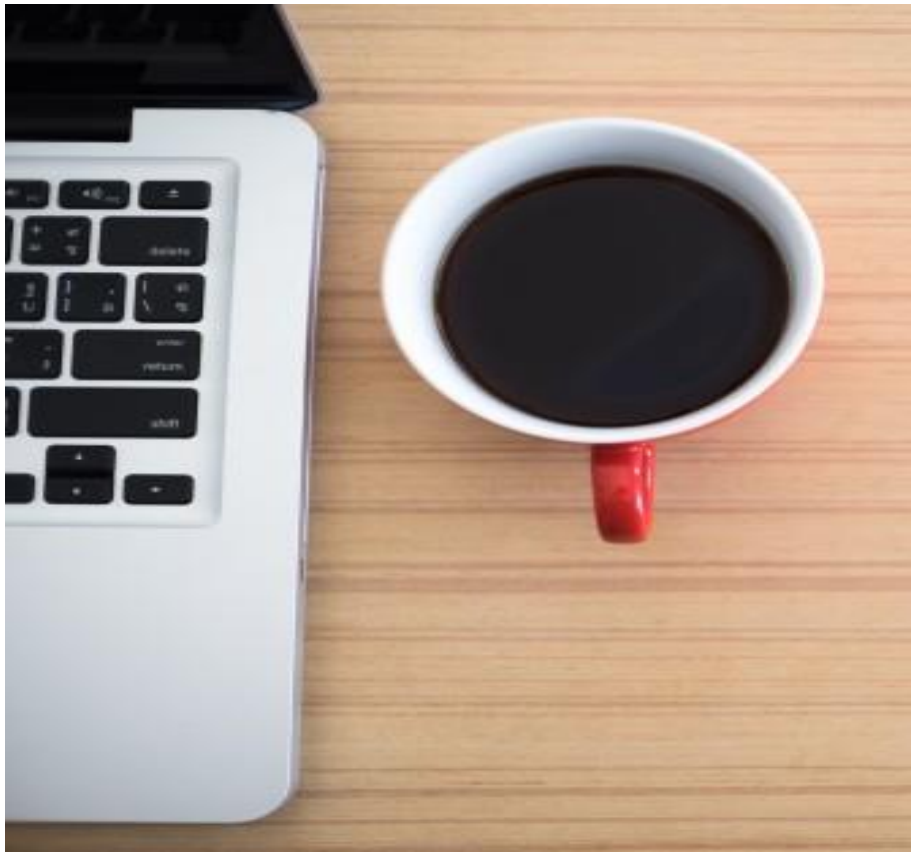
Fourth Industrial Summer School

Advanced Machine Learning

Reinforcement Learning-part2

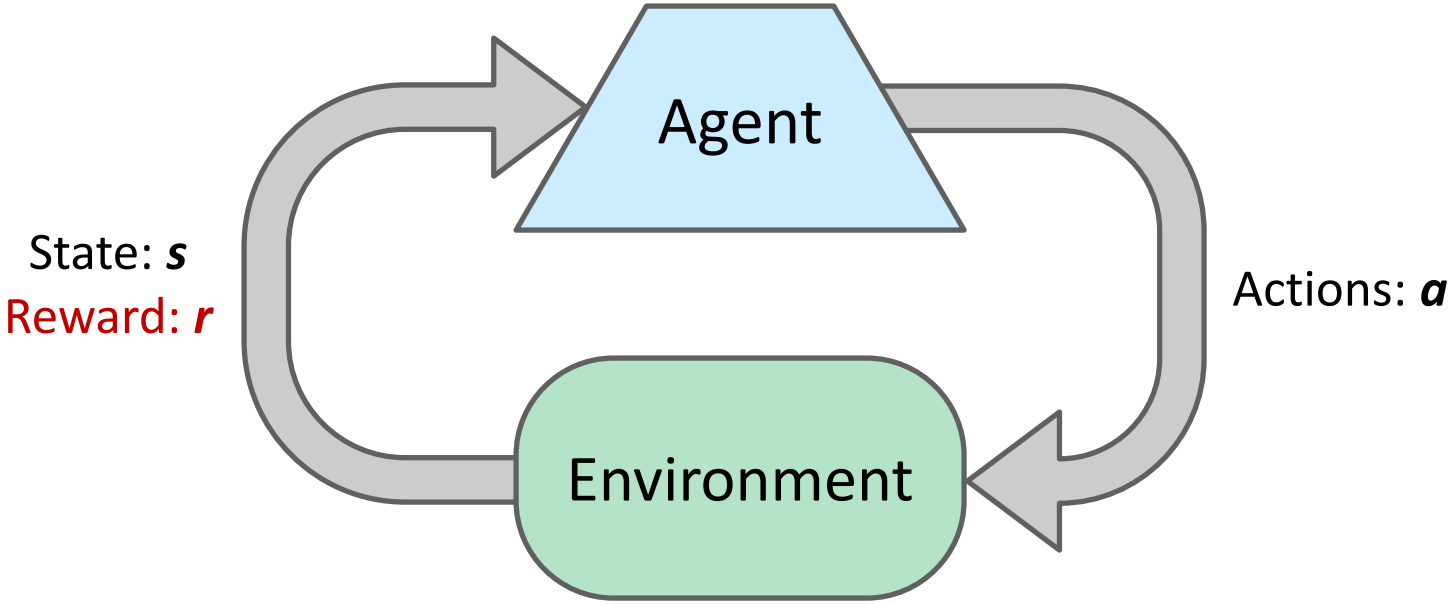
Session Objectives

- ✓Introduction
- ✓Foundations (MDP)
- ✓Reinforcement Learning
- ✓Gym from OpenAI



Reinforcement Learning

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!

Example: Learning to Walk

Initial

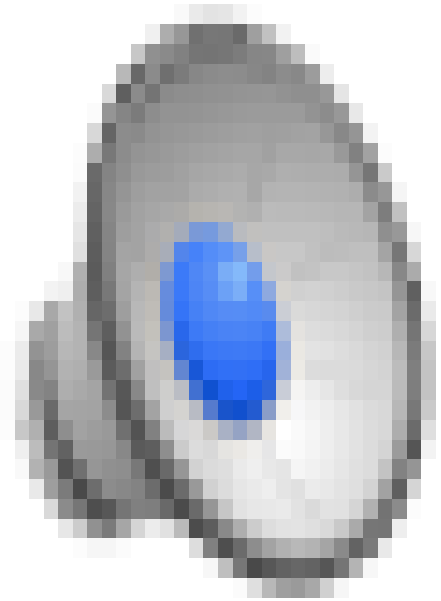
[Kohl and Stone, ICRA 2004]

Example: Learning to Walk (Cont'd)

Finished

[Kohl and Stone, ICRA 2004]

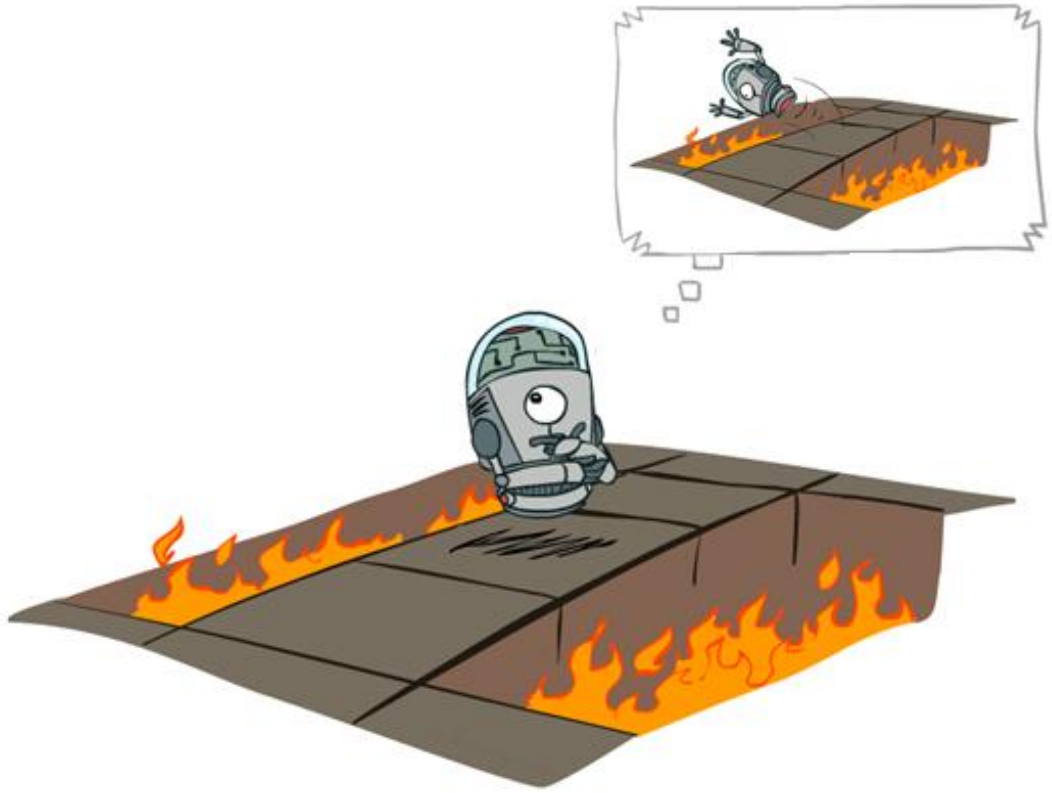
Video of Demo Crawler Bot



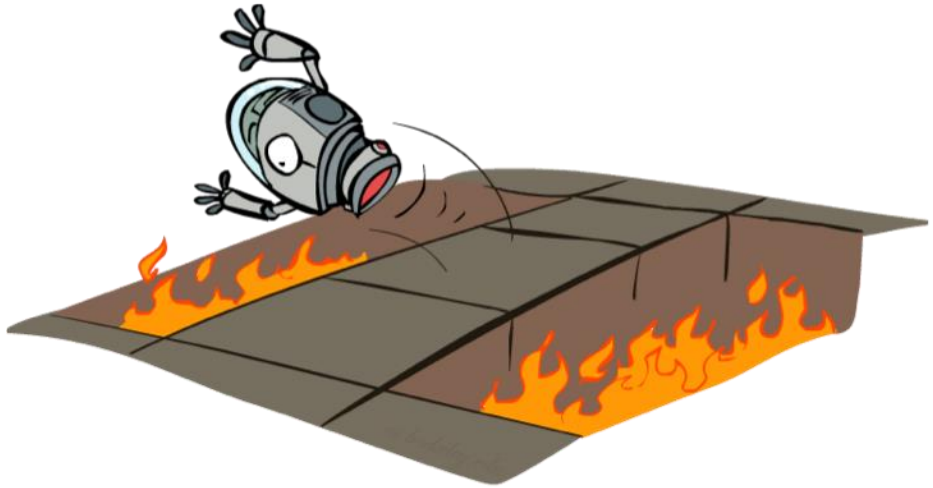
Reinforcement Learning

- Still assume a Markov decision process (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:** We do not know T or R
  - Therefore, we do not know which **states are good** or **what the actions do**
  - **Must actually try actions** and **states out to learn**

Offline (MDPs) vs. Online (RL)

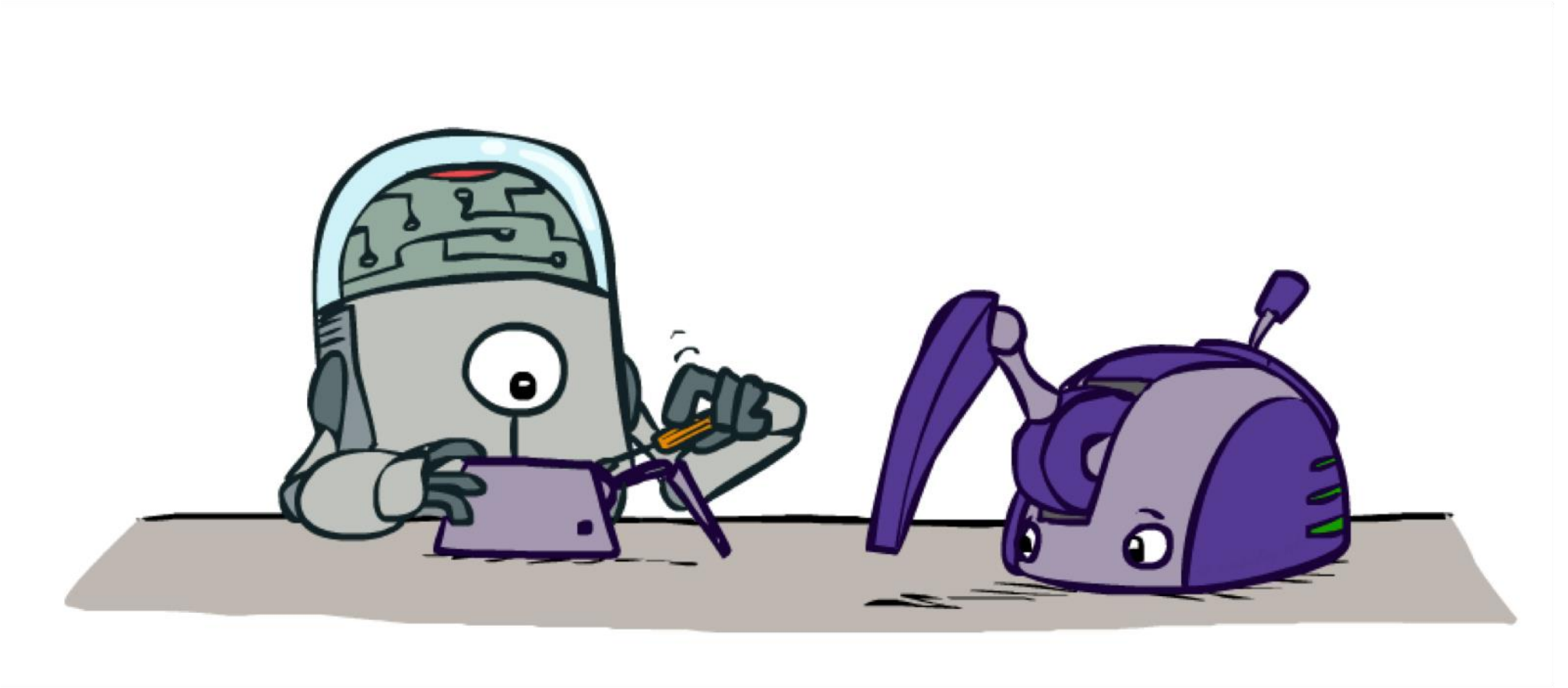


MDP: Offline Solution



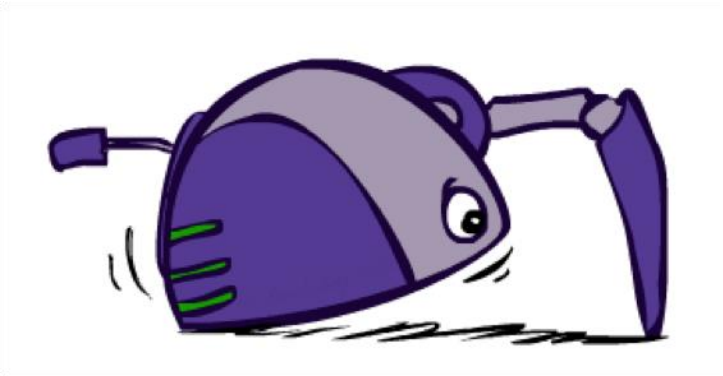
RL: Online Learning

Model-Based Learning



Model-Based Learning

- **Model-Based Idea:**
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

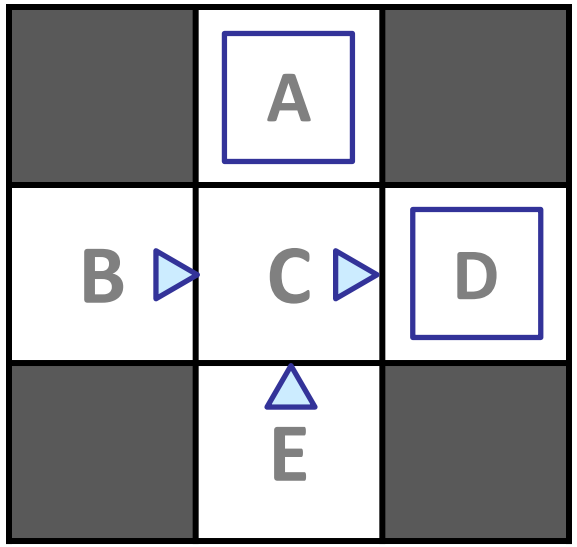


- **Step 1:** Learn empirical MDP model
  - Count outcomes ***s'*** for each ***s***, ***a***
  - Normalize to give an estimate of $\widehat{T}(s, a, s')$
  - Discover each $\widehat{R}(s, a, s')$ when we experience ***(s, a, s')***

- **Step 2:** Solve the **learned MDP**
  - For example, use value iteration, as before

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
...

Question: Are 4 episodes enough to "learn" our MDP?

Example: Expected Age

Goal: Compute expected age of the students in a class

Known P(A): Age distribution

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

$E[A]$: Expected age

$P(a)$: Prob. of age

a: age

$\sum_a P(a) \cdot a$: Weighted average of age

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

Unknown P(A): "Model Based"

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

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

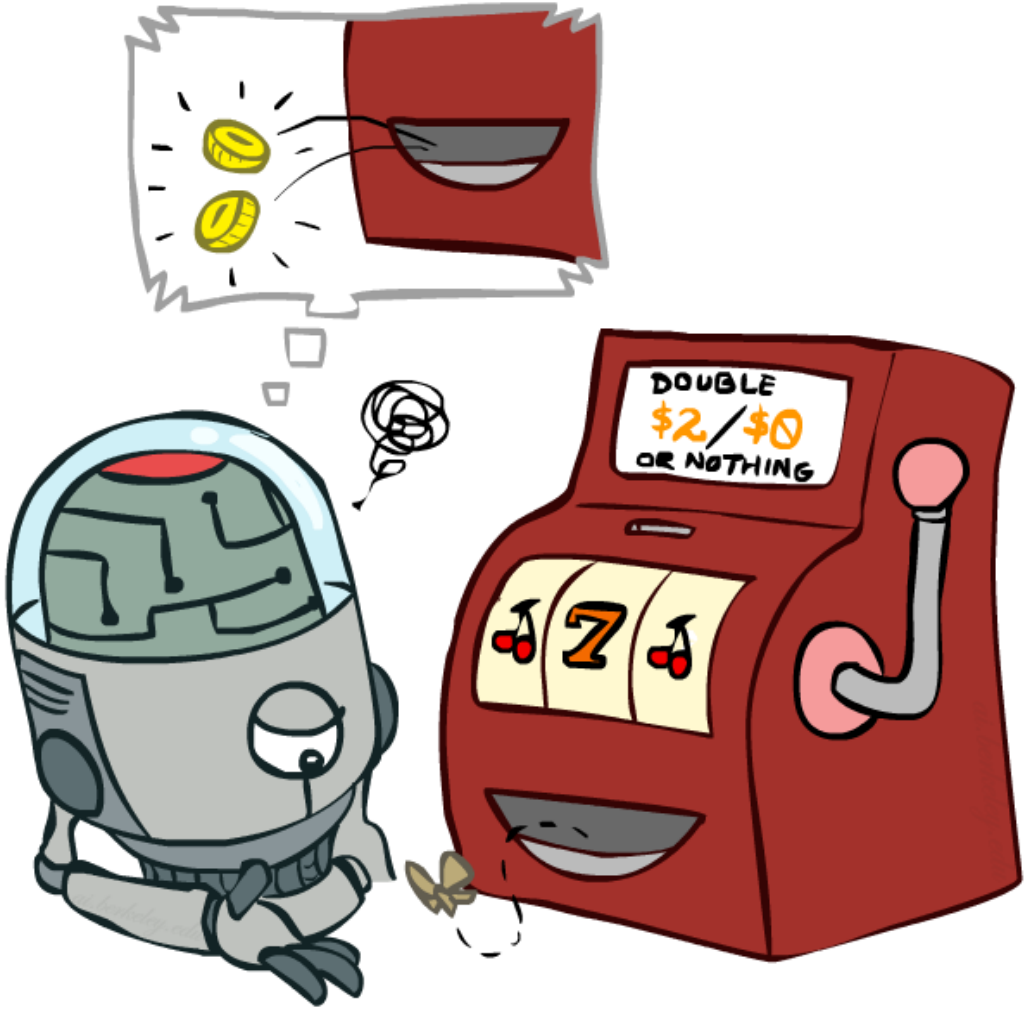
Why does this work? Because eventually you learn the **right model**.

Unknown P(A): "Model Free"

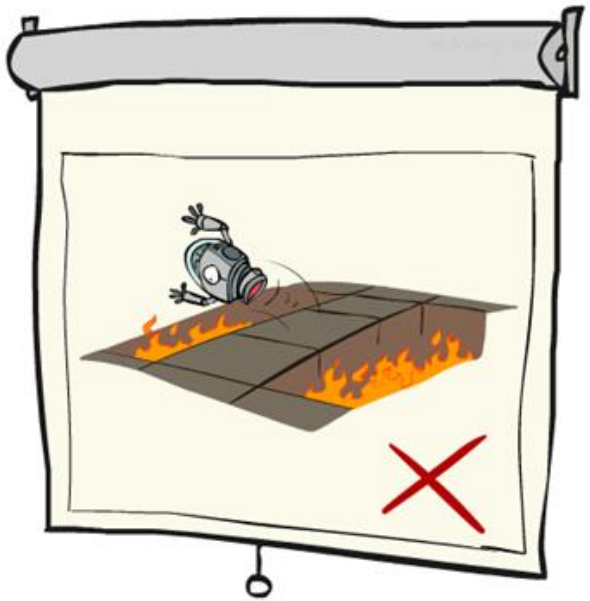
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because **samples appear with the right frequencies**.

Model-Free Learning

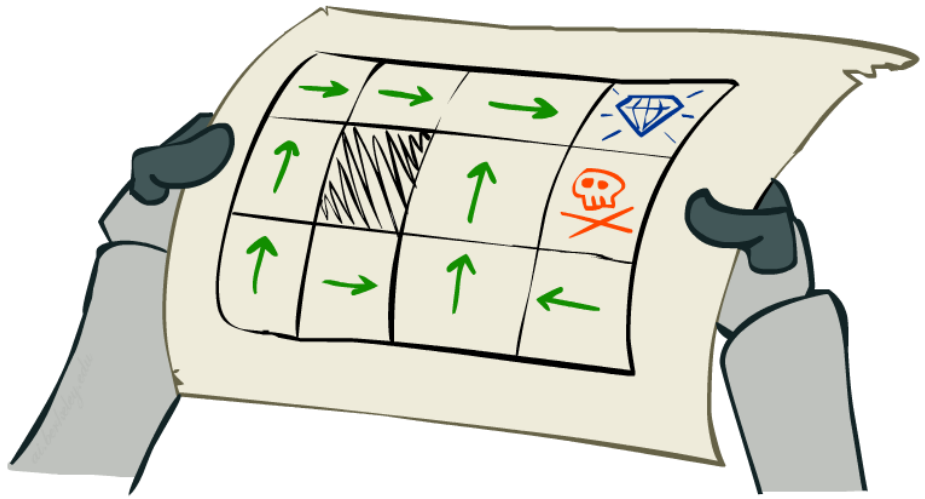


Passive Reinforcement Learning



Passive Reinforcement Learning (Cont'd)

- **Simplified task:** Policy evaluation
  - **Input:** A fixed policy $\pi(s)$
  - You do not know the transitions $T(s,a,s')$
  - You do not know the rewards $R(s,a,s')$
  - **Goal:** Learn the state values



- In this case:
  - Learner is "**along for the ride**"
  - **No choice about what actions to take** (cannot try actions!)
  - Just execute the policy and learn from experience
  - This is **NOT offline planning**! You actually take actions in the world.

Direct Evaluation

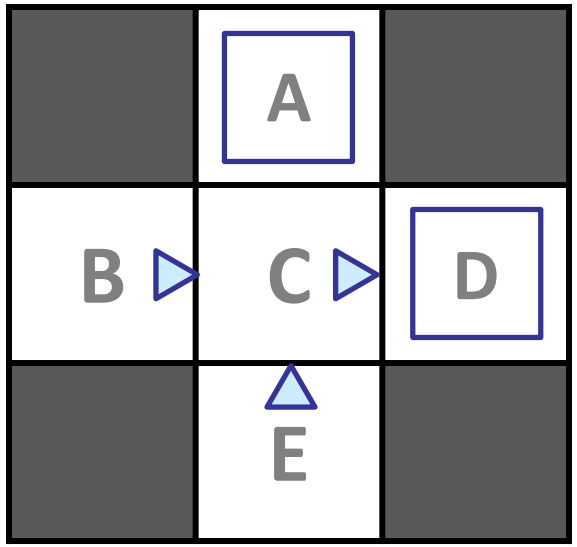
- **Goal:** Compute values for each state under π

- **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

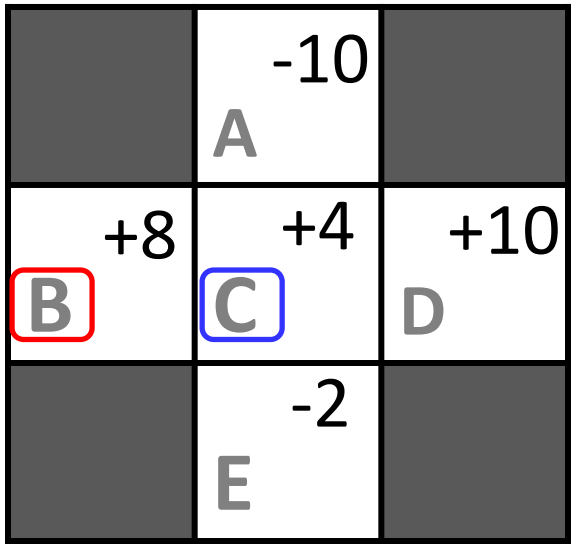
Episode 3

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



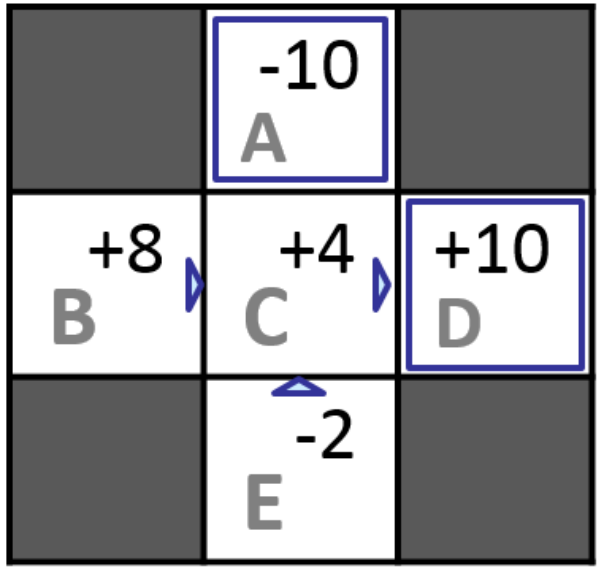
$V^{\pi}(B) = ((+8) + (+8)) / 2 = 8$

$V^{\pi}(C) = ((+9) + (+9) + (+9) + (-11)) / 4 = 4$

Problems with Direct Evaluation

- **Question:** What is good about direct evaluation?
  - It is easy to understand
  - It does not require any knowledge of ***T*** and ***R***
  - It eventually computes the correct average values, using just sample transitions
- **Question:** What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Output Values



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

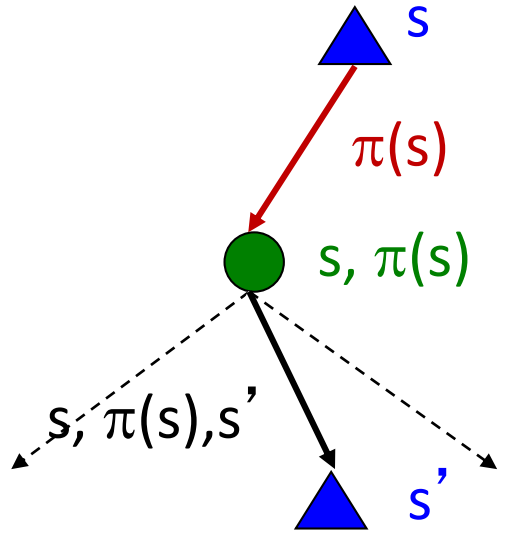
Answer: *The value of state **E** is not taking advantage of the value in state **C** which contradicts Bellman equations!*

Why Not Use Policy Evaluation?

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

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

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} [\quad\quad + \gamma V_k^{\pi}(s')]$$



  - This approach fully exploited the connections between the 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***?
  - In other words, how do we take a weighted average without knowing the weights?

Sample-Based Policy Evaluation?

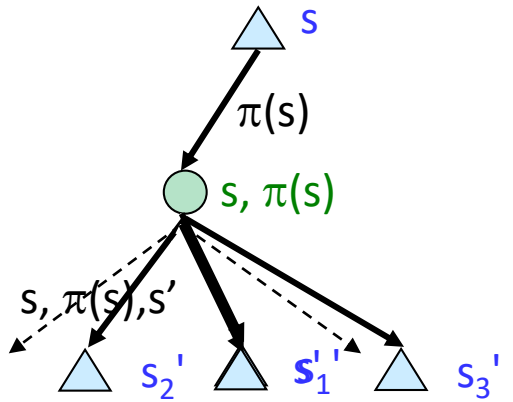
- We want to improve our estimate of ***V*** by computing these averages:

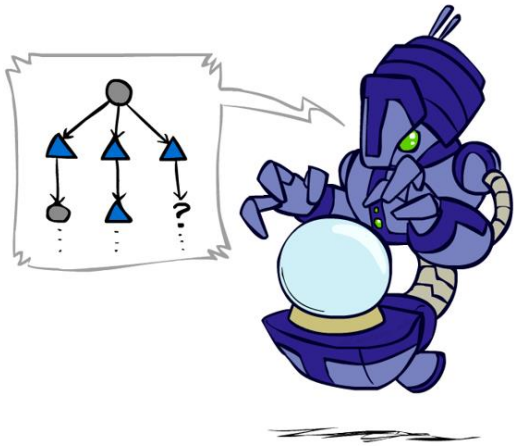
$$V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}_{k}(s')]$$

- **Idea:** Take samples of outcomes ***s'*** (by doing the action ***a***!) and average

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V^{\pi}_{k}(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V^{\pi}_{k}(s'_2)$$
$$\cdots$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V^{\pi}_{k}(s'_n)$$

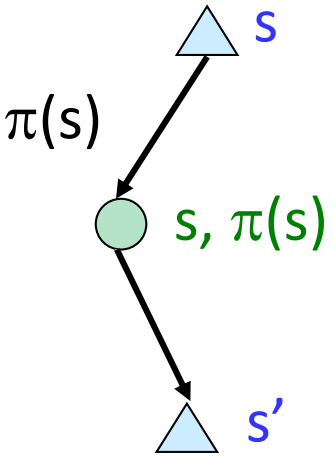
$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n} \sum_{i} sample_i$$





TD Learning (Cont'd)

- **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*** of values
  - **Policy still fixed (π)**, still doing evaluation!
  - Move values toward value of whatever successor occurs: ***Running average***

Inaccurate at the beginning of the update process

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

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

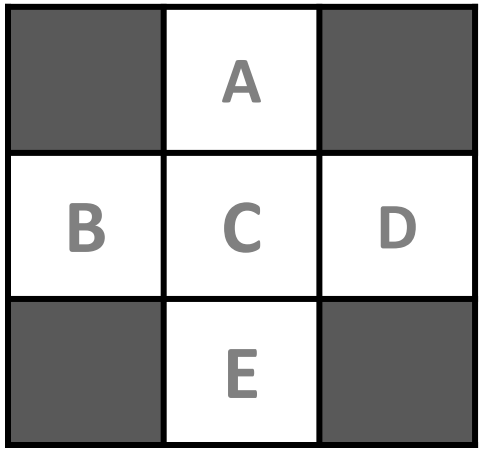
Reordering

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

No need to store experience (Episodes).
We store only values V(s).

Example: TD Learning

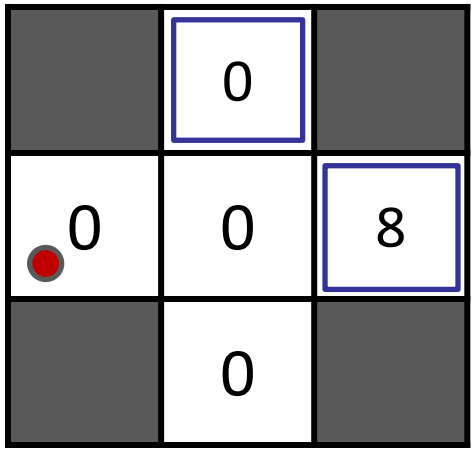
States



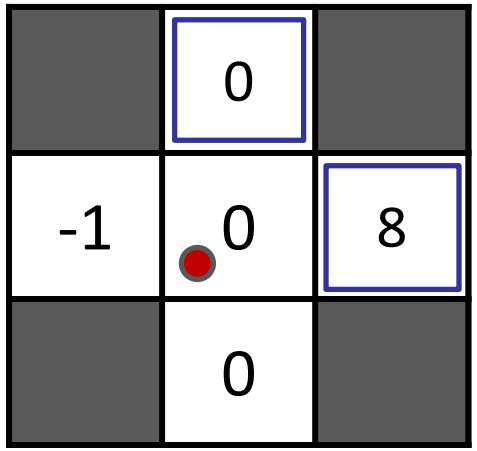
Assume: $\gamma = 1$, $\alpha = 1/2$

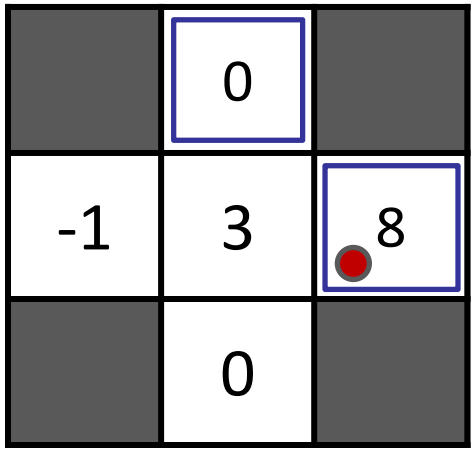
Observed Transitions

B, east, C, -2



C, east, D, -2





V^π(B) ← (1 − 0.5) · 0 + 0.5 · [-2 + 1 · 0]= -1

$$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

V^π(C) ← (1 − 0.5) · 0 + 0.5 · [-2 + 1 · 8]= 3

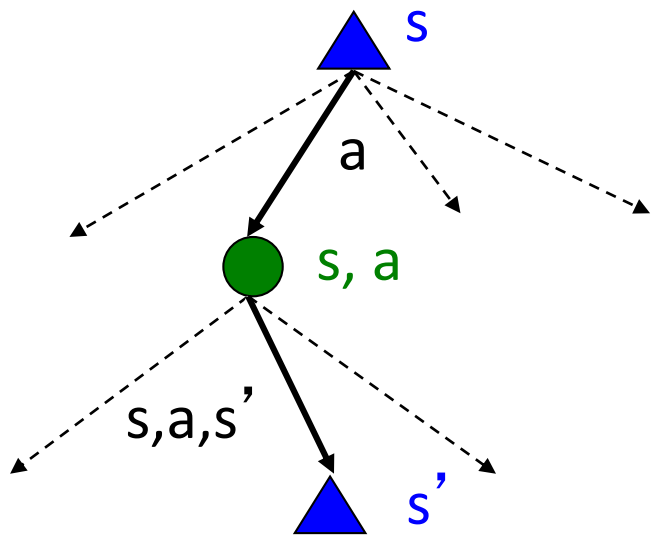
Problems with TD Value Learning

- TD value leaning is a ***model-free*** way to do policy evaluation, mimicking Bellman updates with ***running sample averages***

- However, if we want to ***turn values into a (new) policy***, we ***will not be able to do so***:

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

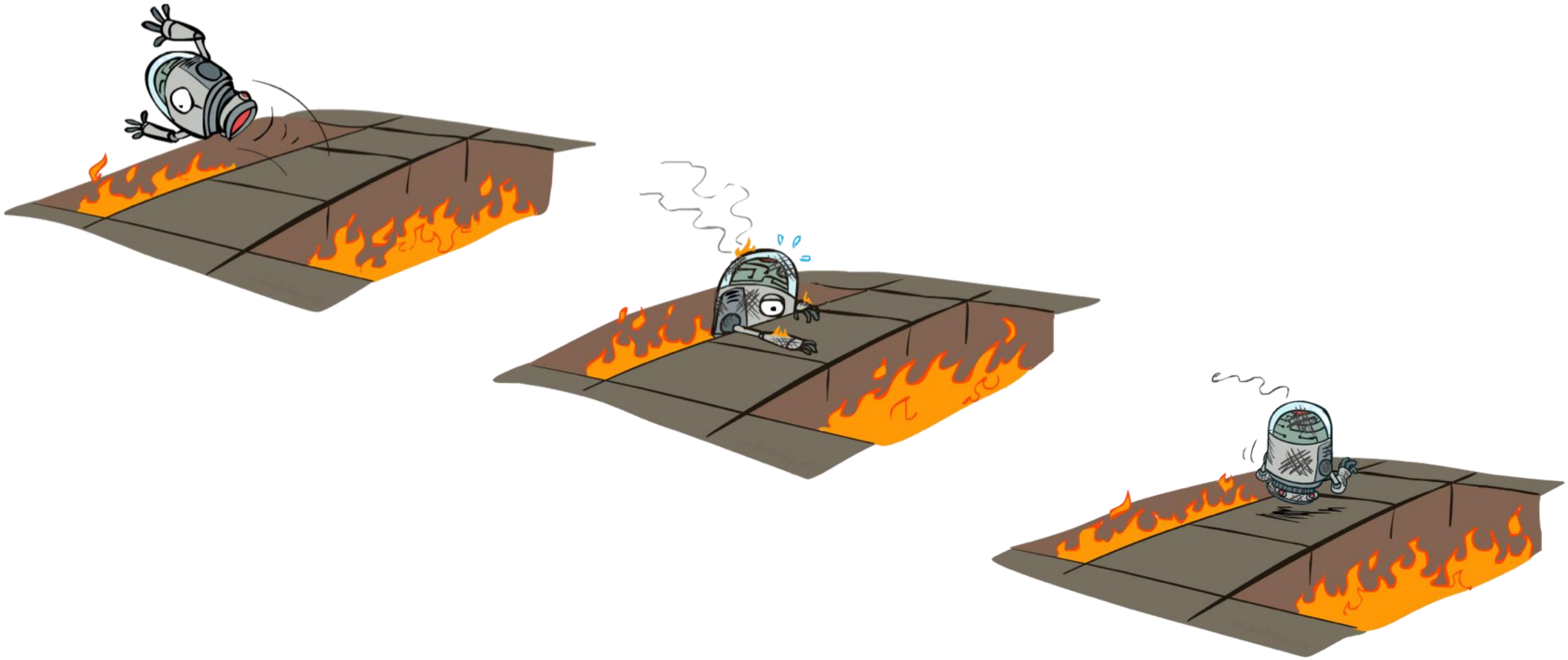
$$Q(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right]$$



- **Idea:** Learn Q-values, not values

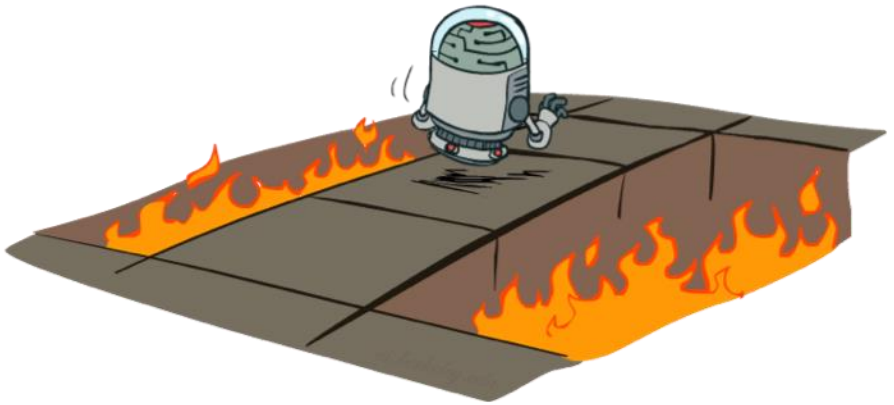
- Makes ***action selection model-free*** too!

Active Reinforcement Learning



Active Reinforcement Learning (Cont'd)

- Full reinforcement learning: Optimal policies (like value iteration)
  - You do not know the transitions ***T(s, a, s')***
  - You do not know the rewards ***R(s, a, s')***
  - You ***choose*** the ***actions now***
  - **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…

Detour: Q-Value Iteration

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

The ***max*** operator **cannot be approximated by a running average!**

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

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

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

$V_k(s')$: One quantity

An average quantity can be approximated by a running average!

Q-Learning Algorithm

- **Q-Learning Algorithm:** ***Sample-based*** Q-value iteration

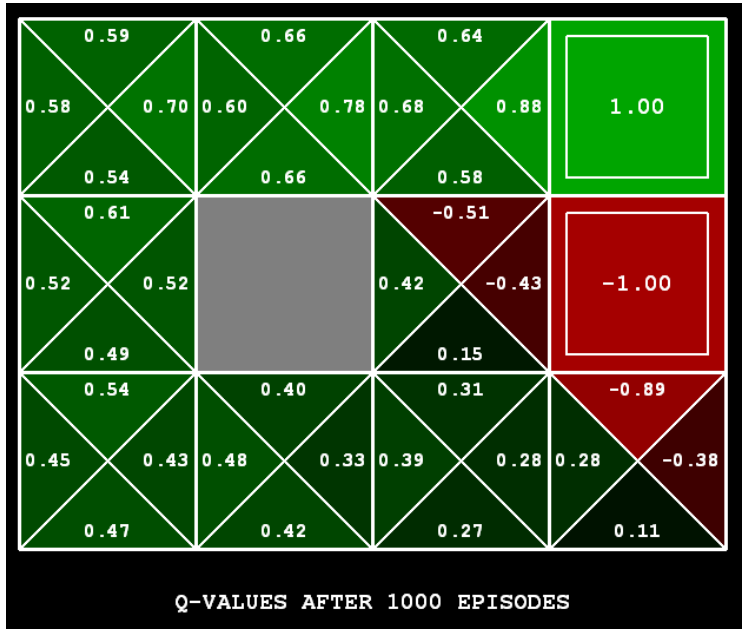
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- 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:

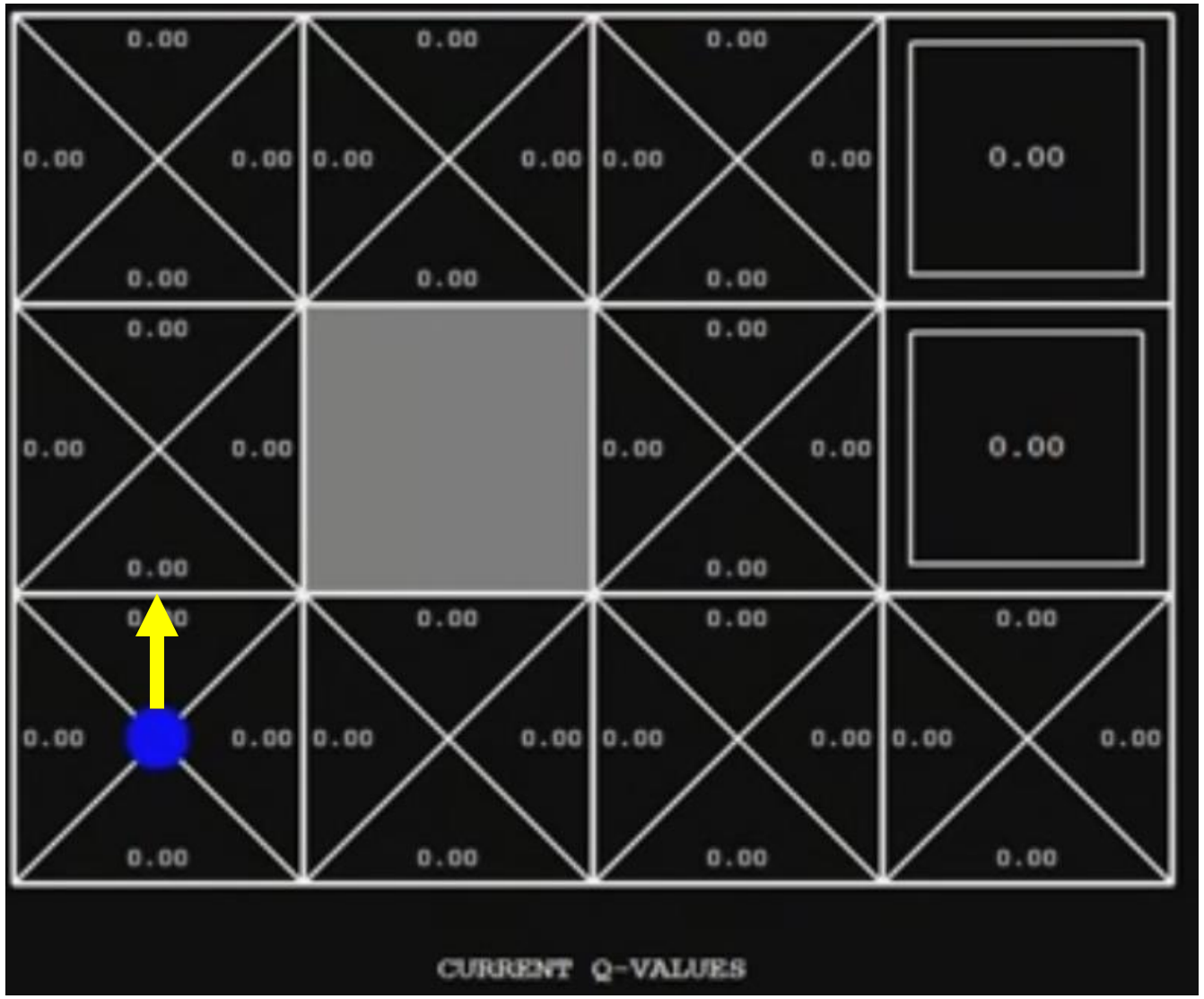
$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

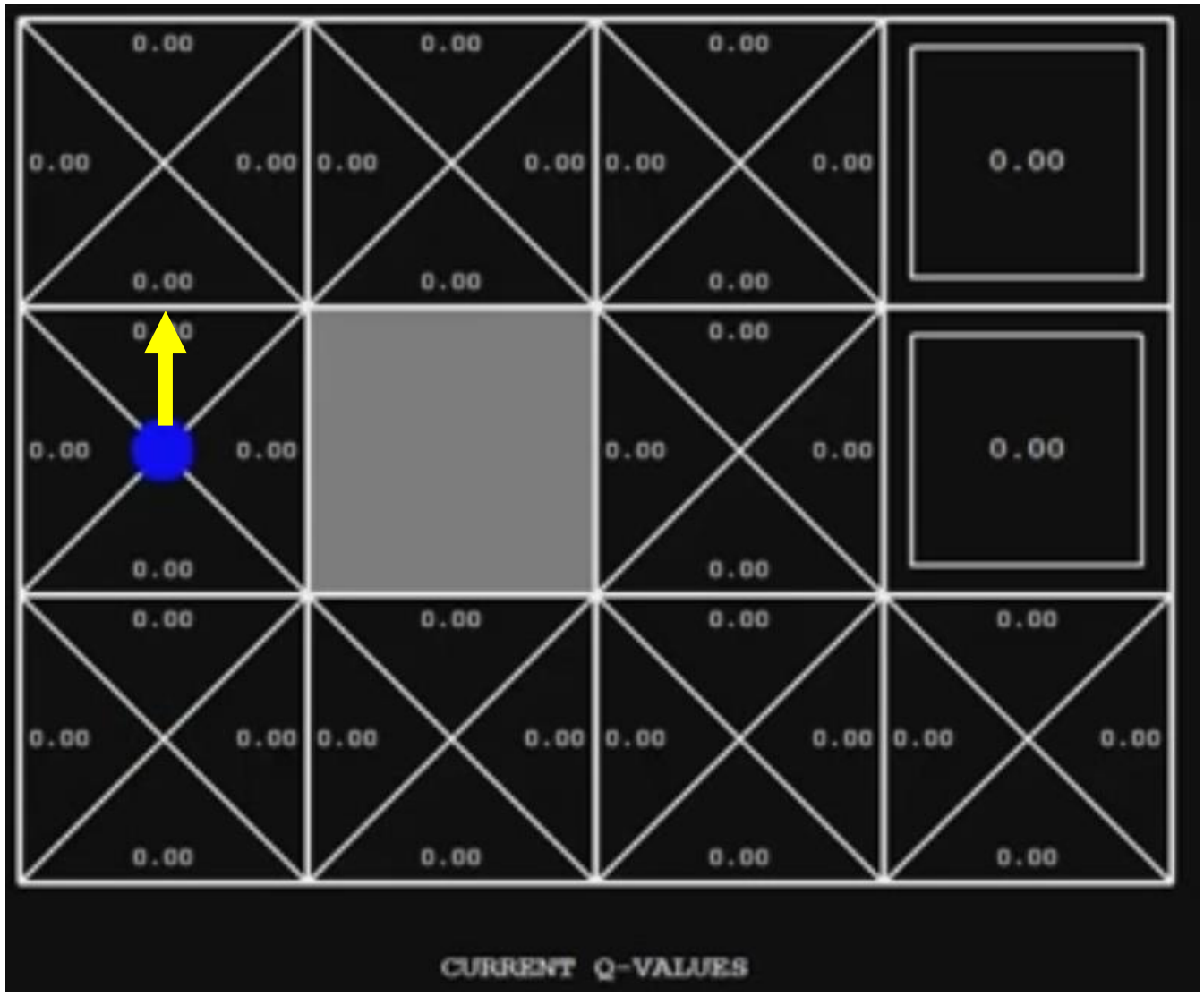
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$



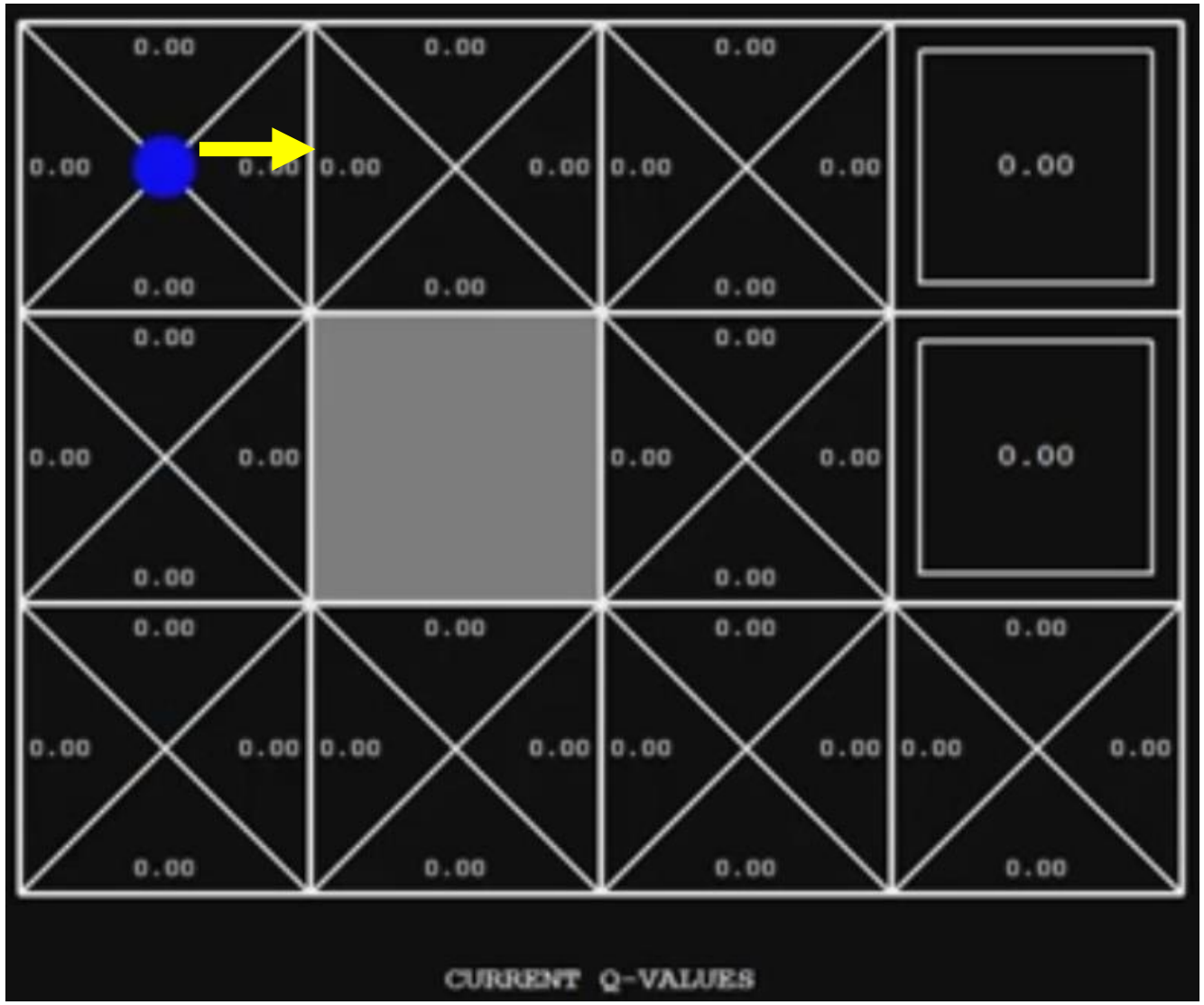
Q-Learning Algorithm (Cont'd)



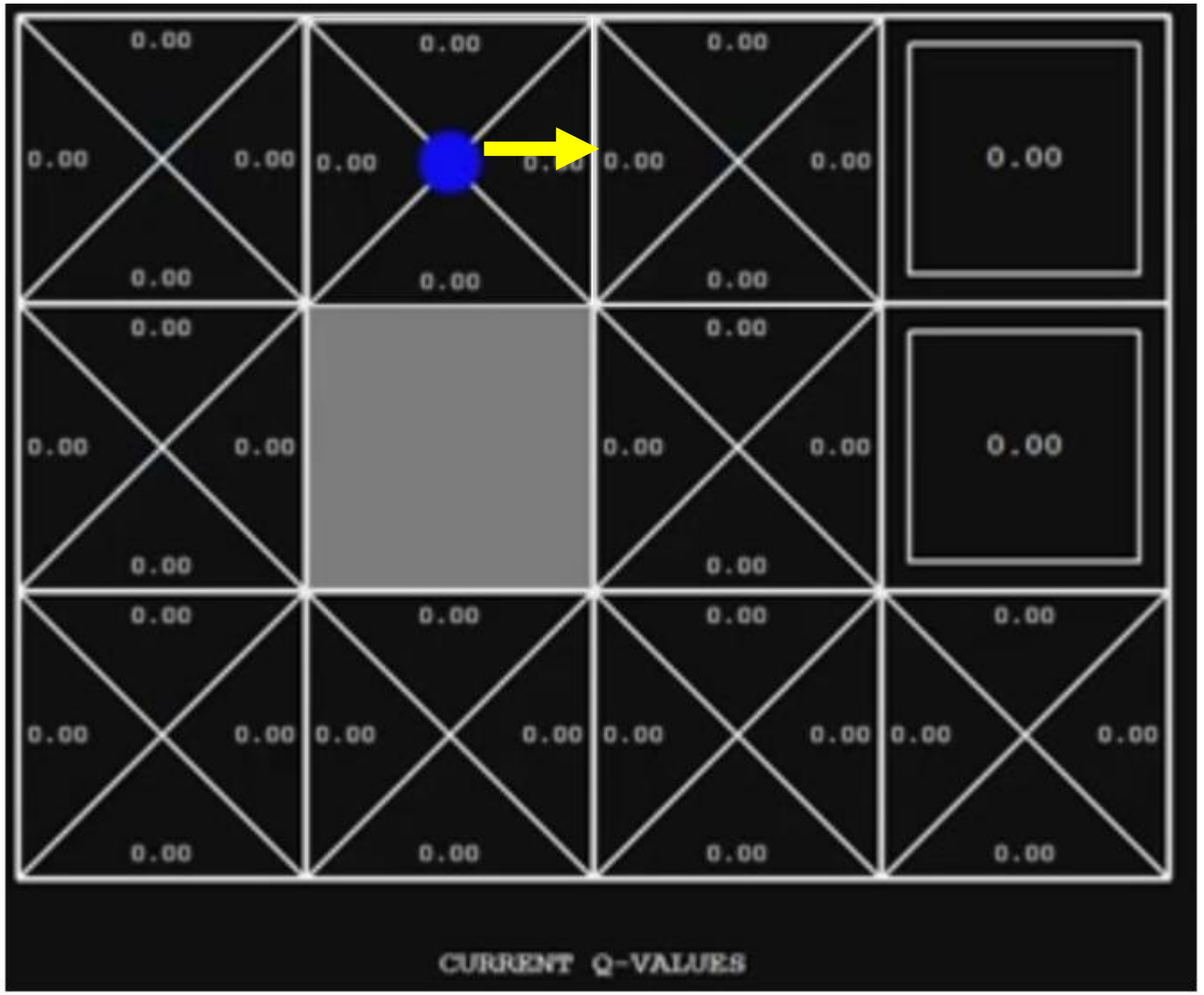
Q-Learning Algorithm (Cont'd)



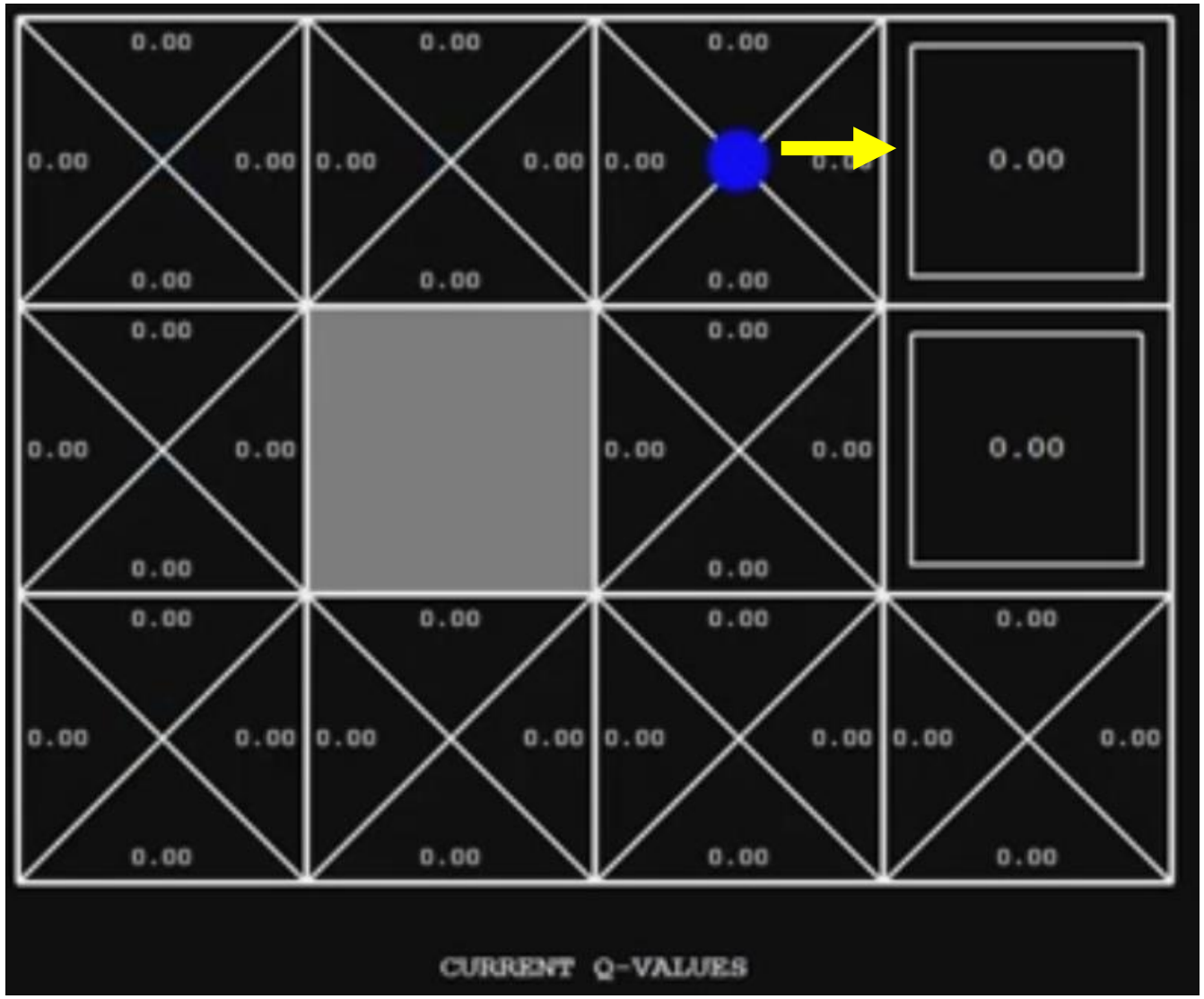
Q-Learning Algorithm (Cont'd)



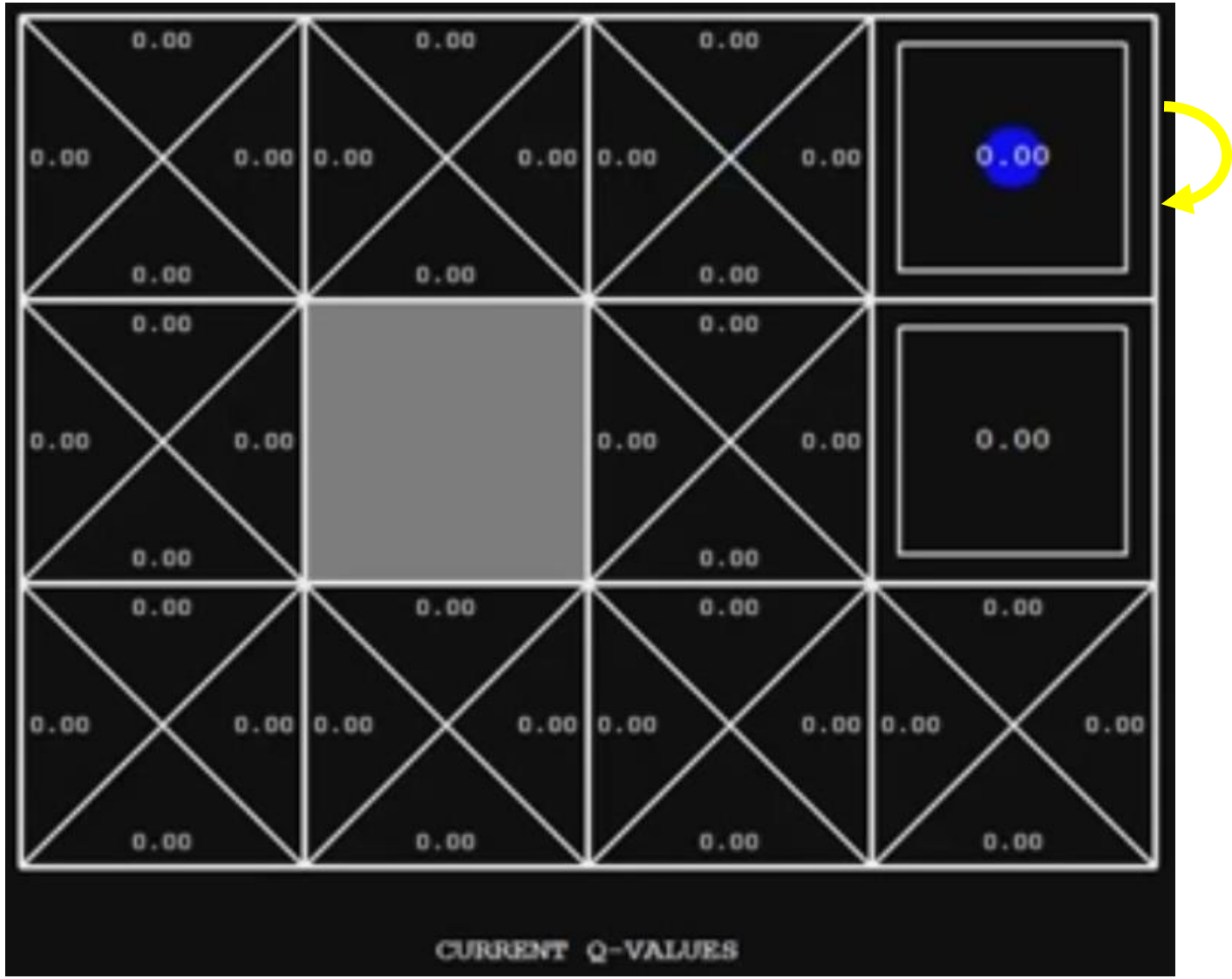
Q-Learning Algorithm (Cont'd)



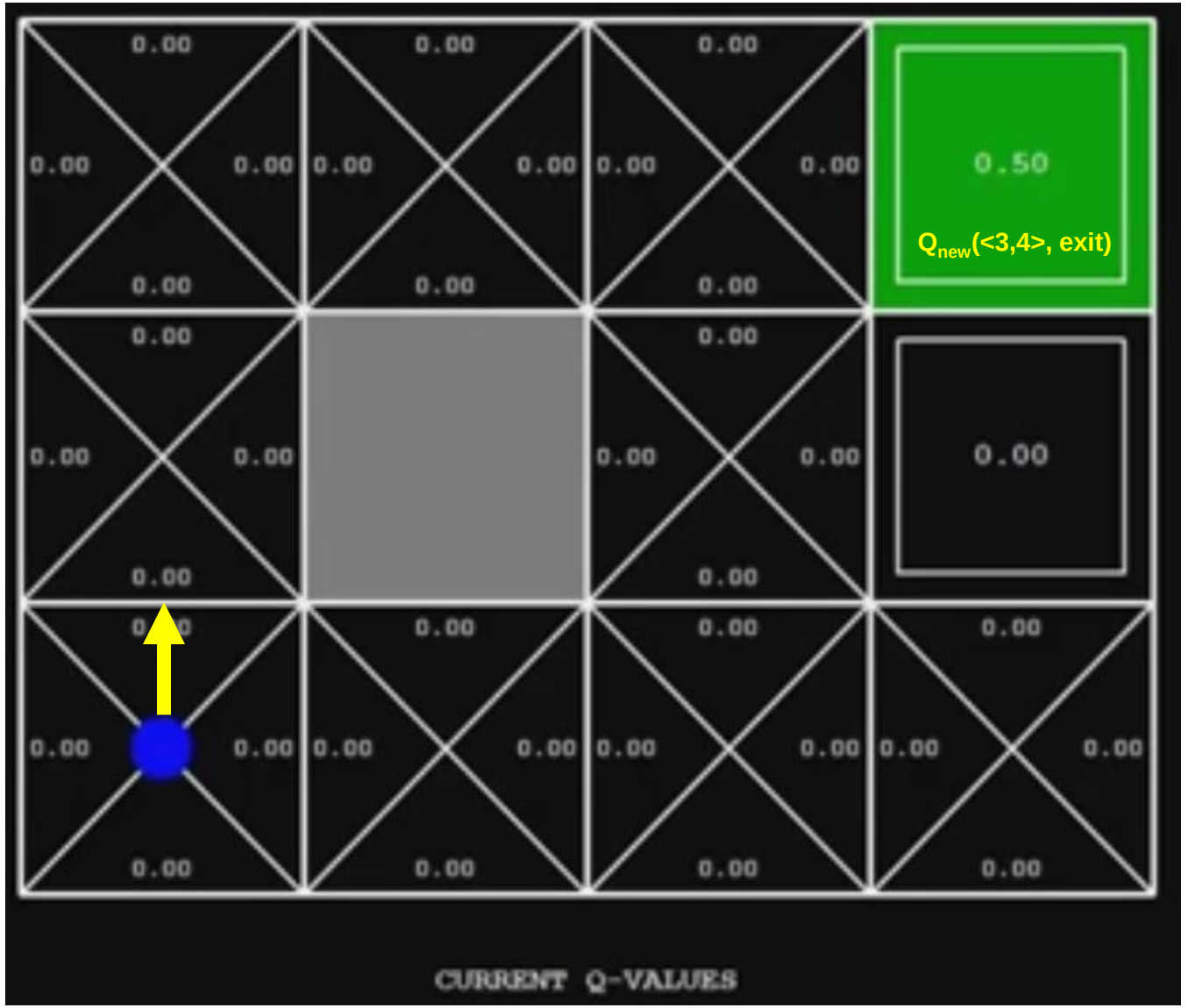
Q-Learning Algorithm (Cont'd)



Q-Learning Algorithm (Cont'd)



Q-Learning Algorithm (Cont'd)

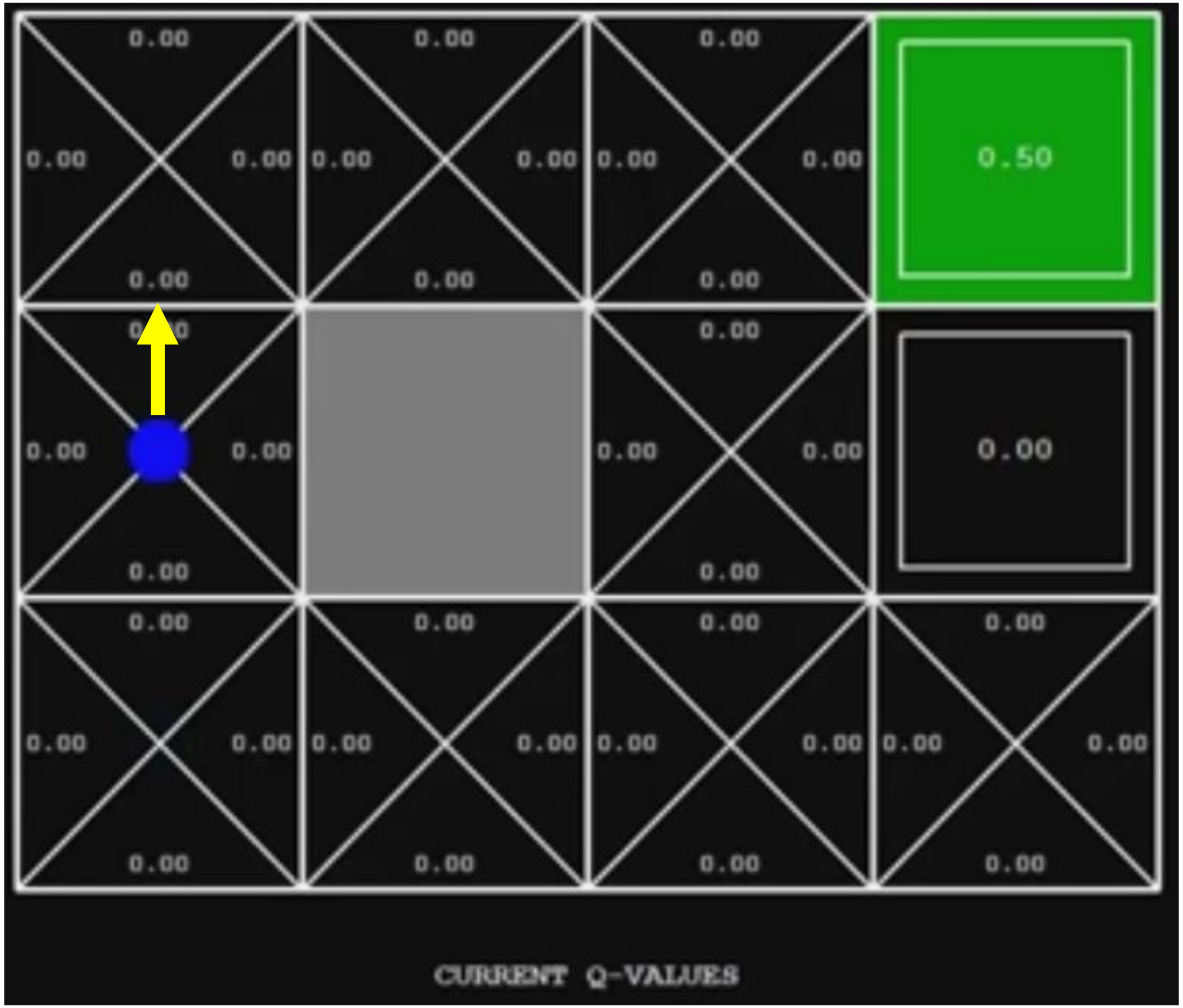


$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

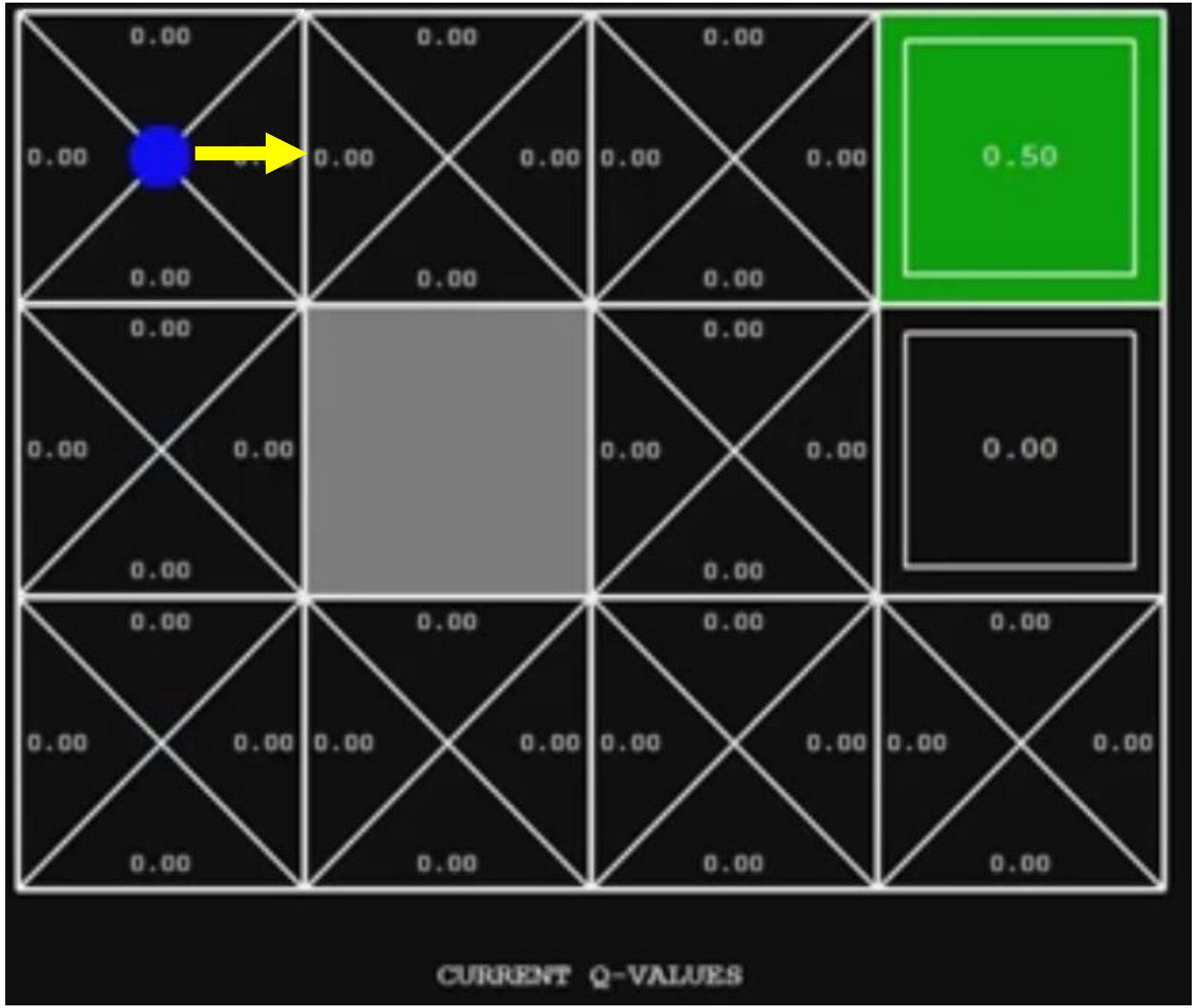
$= (1 - 0.5) \cdot Q_{old}(<3,4>, exit) + 0.5 * 1$

$Q_{new}(<3,4>, exit) = 0.5 \cdot 0 + 0.5 = 0.5$

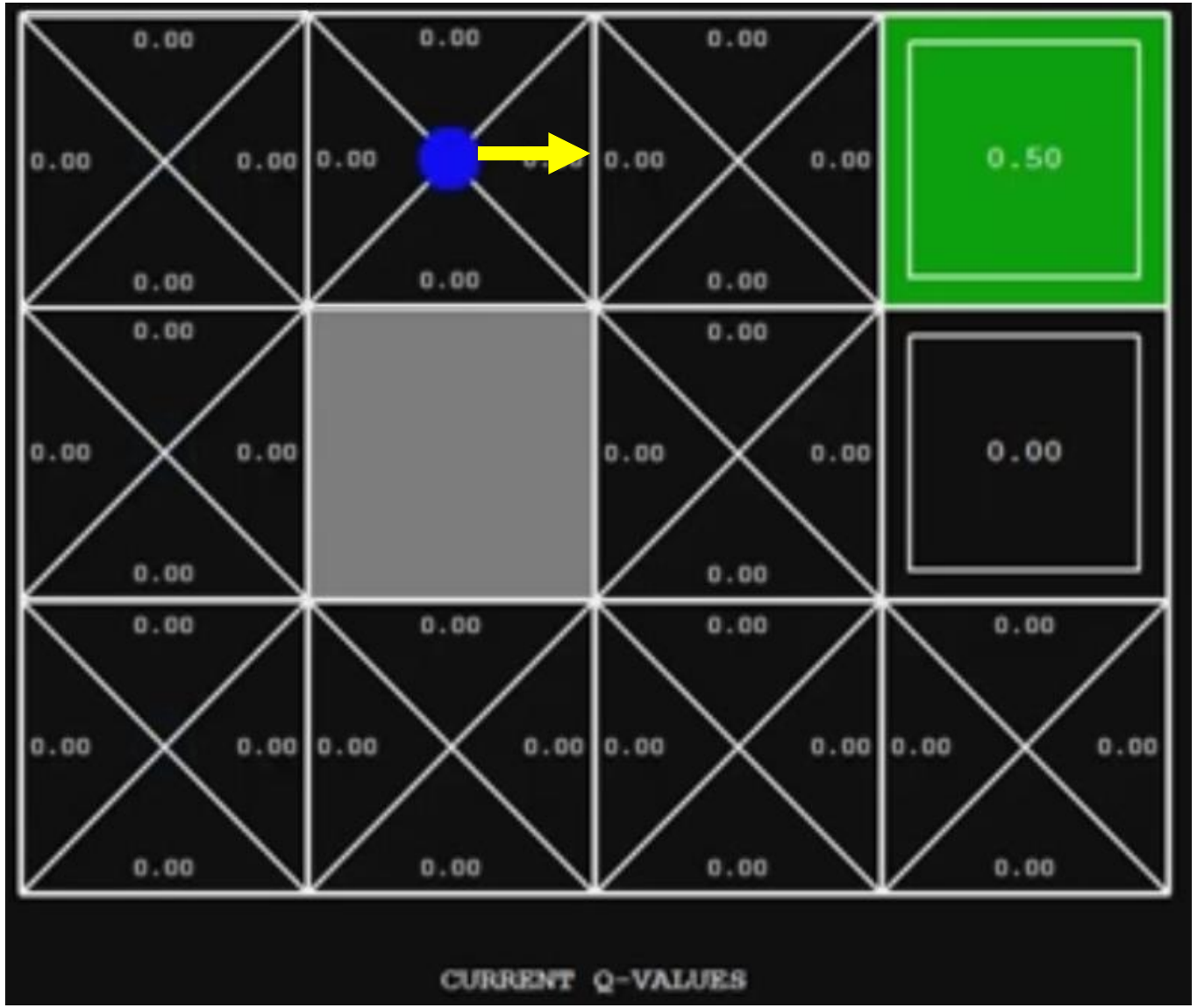
Q-Learning Algorithm (Cont'd)



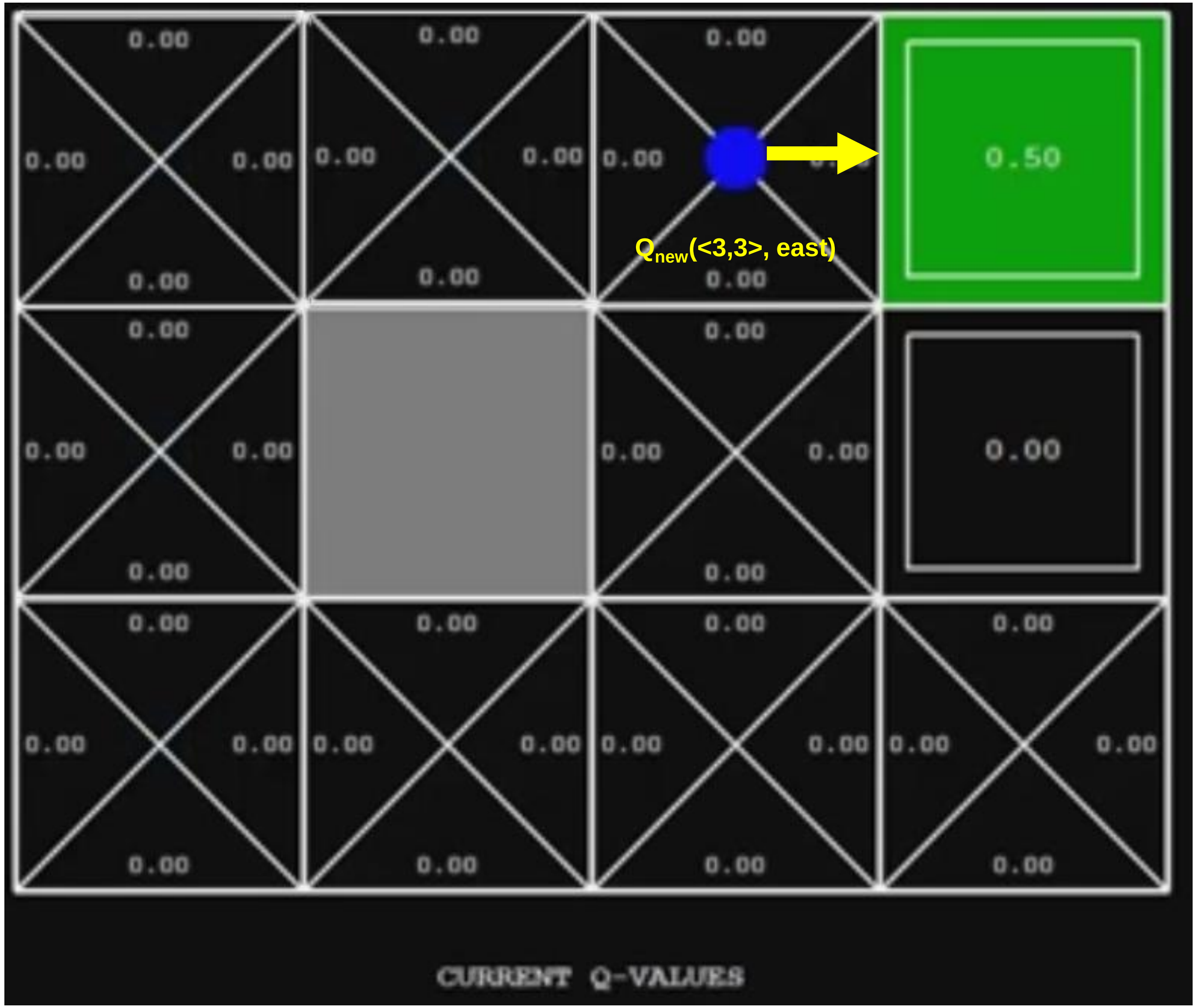
Q-Learning Algorithm (Cont'd)



Q-Learning Algorithm (Cont'd)



Q-Learning Algorithm (Cont'd)

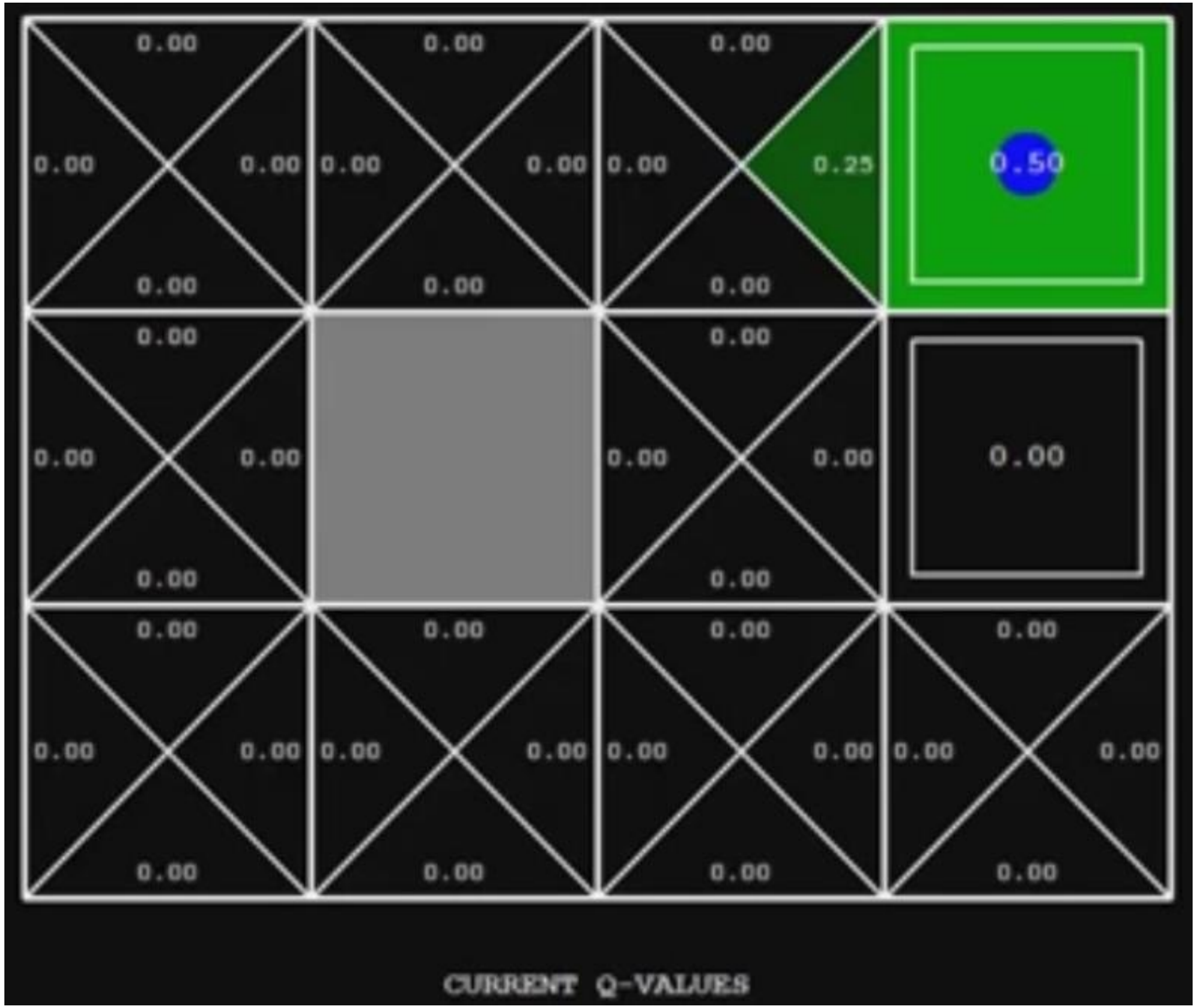


$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$

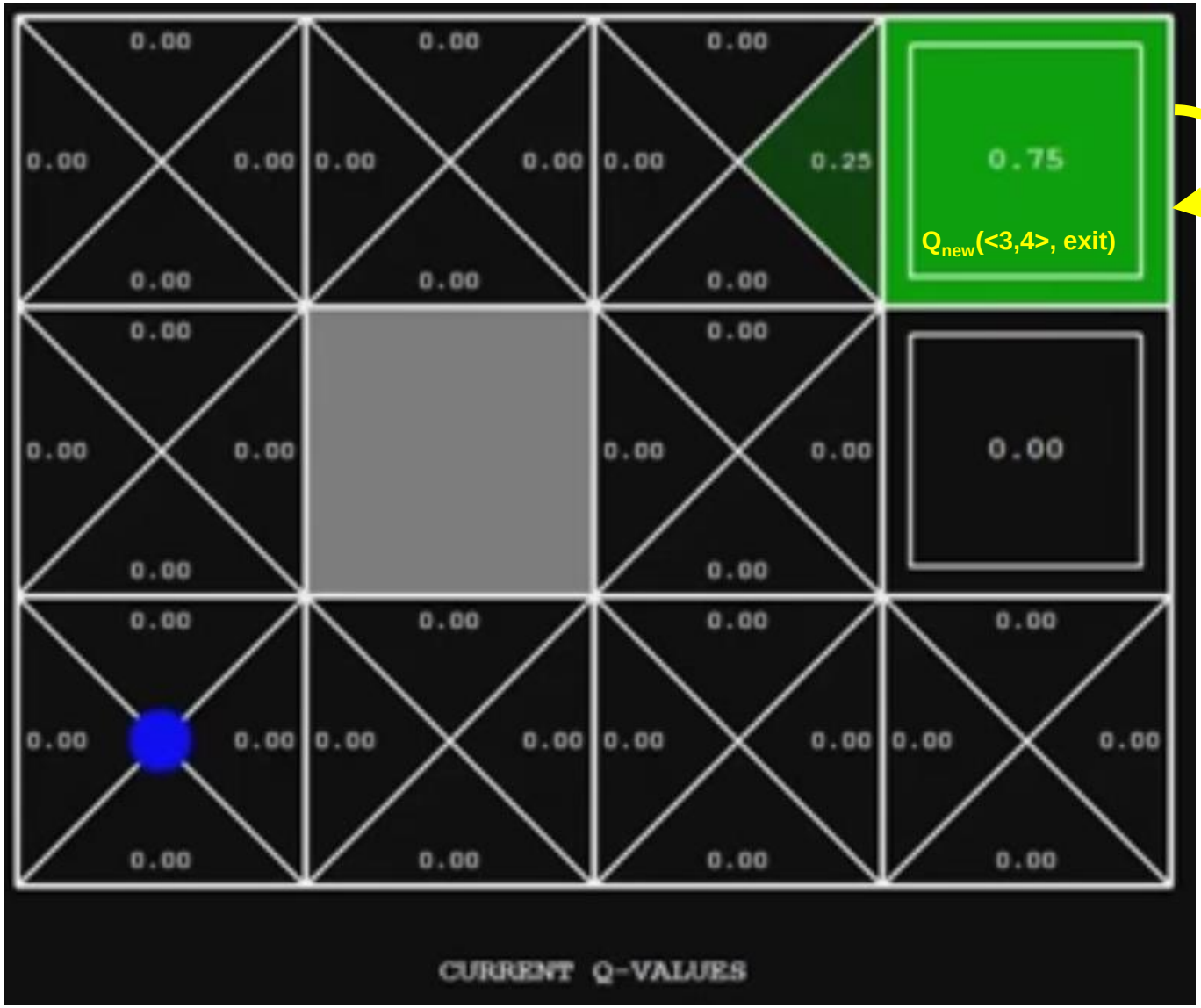
$= (1 - 0.5) \cdot Q_{old}(\langle 3,3\rangle, \text{east}) + 0.5 * 0.5$

$Q_{new}(\langle 3,3\rangle, \text{east}) = 0.5 \cdot 0 + 0.25 = 0.25$

Q-Learning Algorithm (Cont'd)



Q-Learning Algorithm (Cont'd)

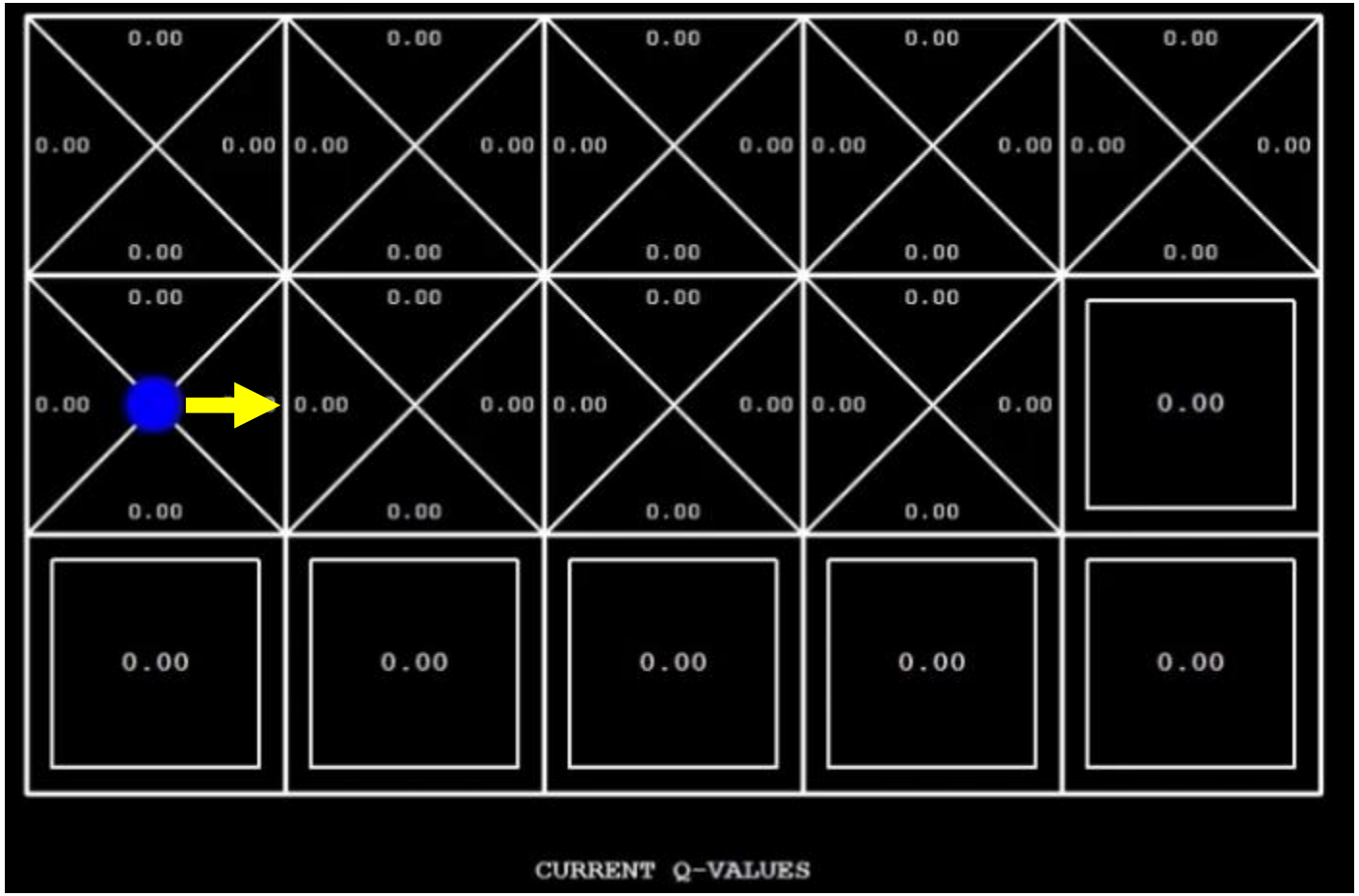


$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$

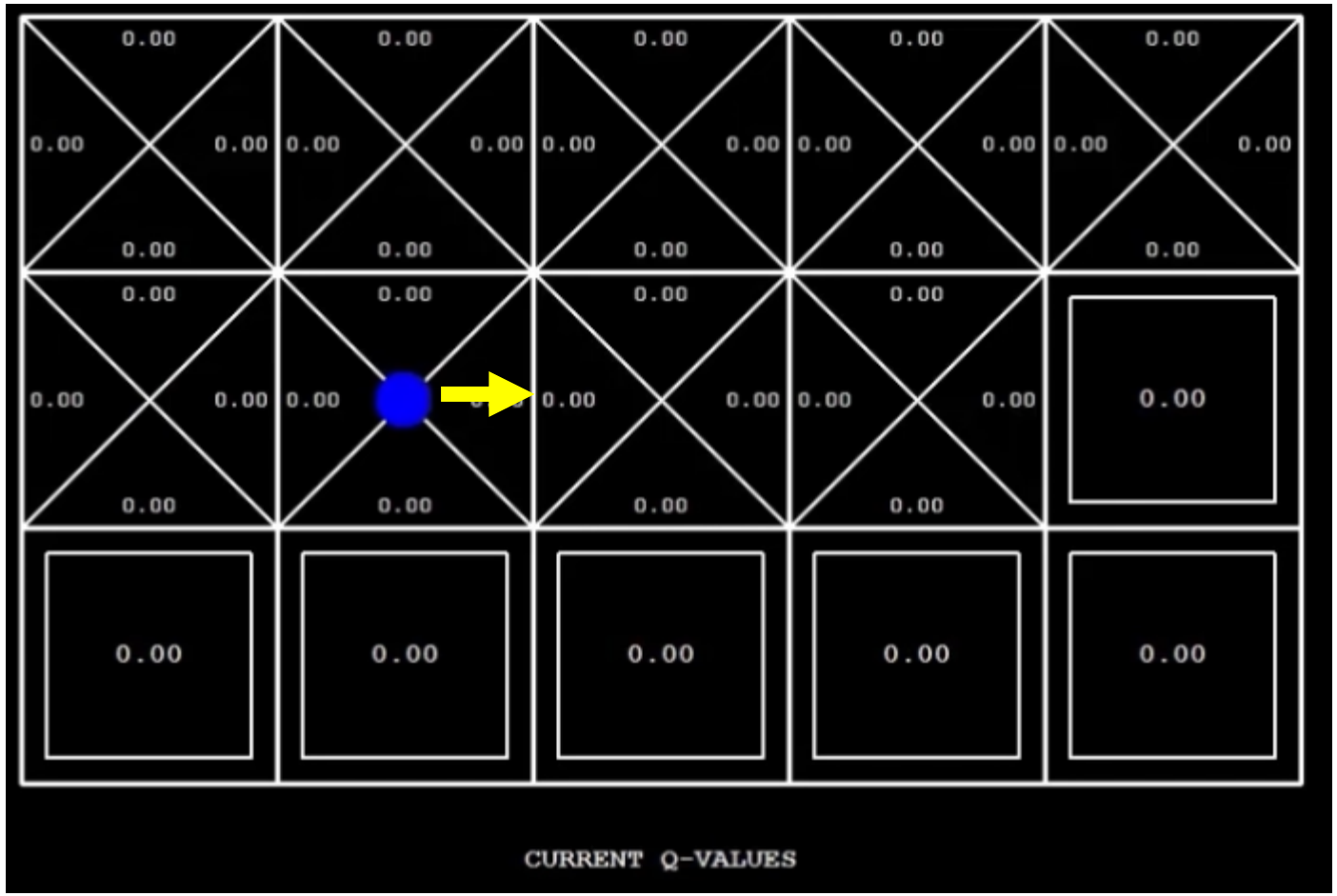
$= (1 - 0.5) \cdot Q_{old}(\langle 3,4\rangle, exit) + 0.5 * 1$

$Q_{new}(\langle 3,4\rangle, exit) = 0.5 \cdot 0.5 + 0.5 = 0.75$

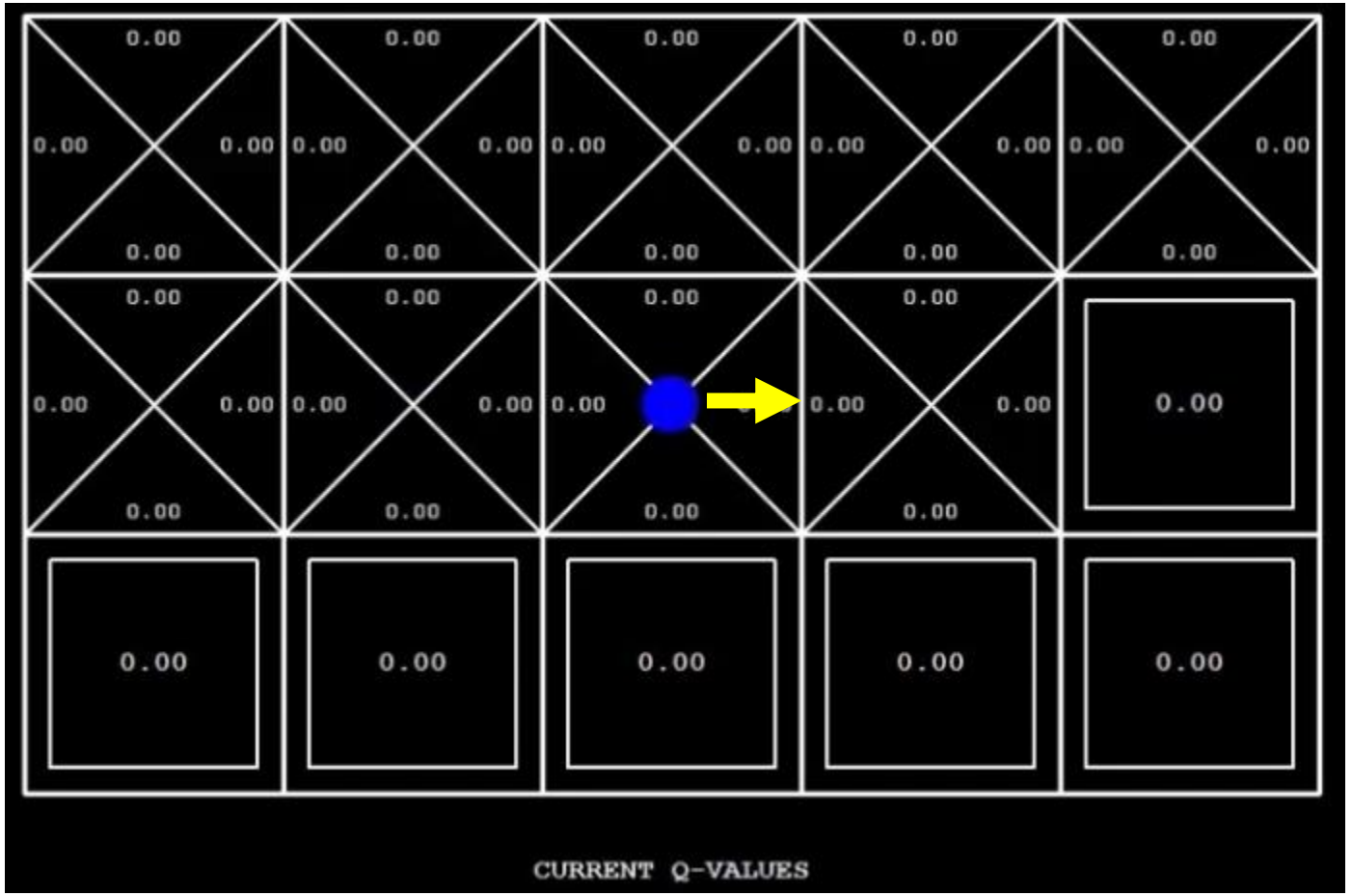
Q-Learning Algorithm: Another Example



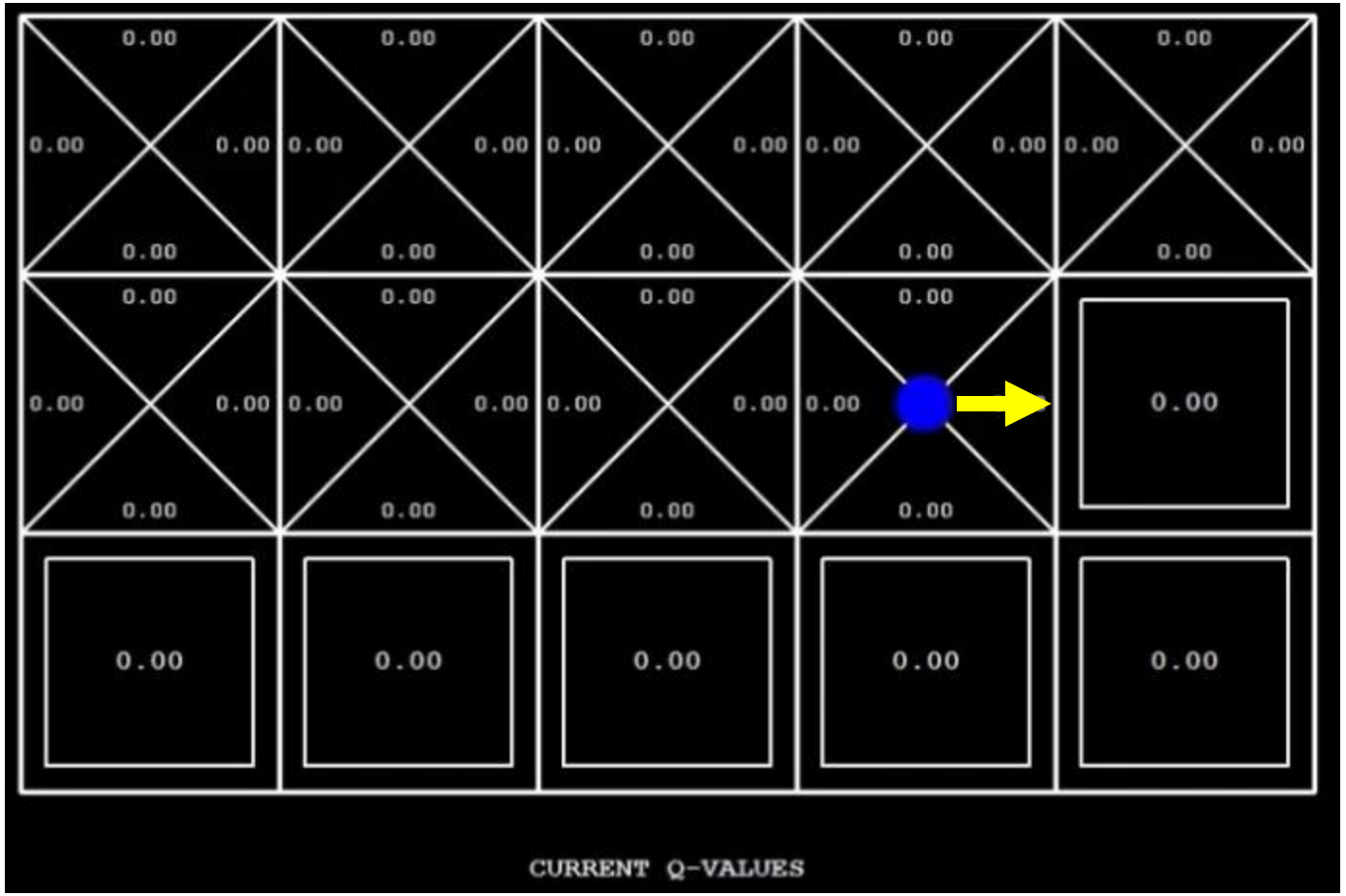
Q-Learning Algorithm: Another Example (Cont'd)



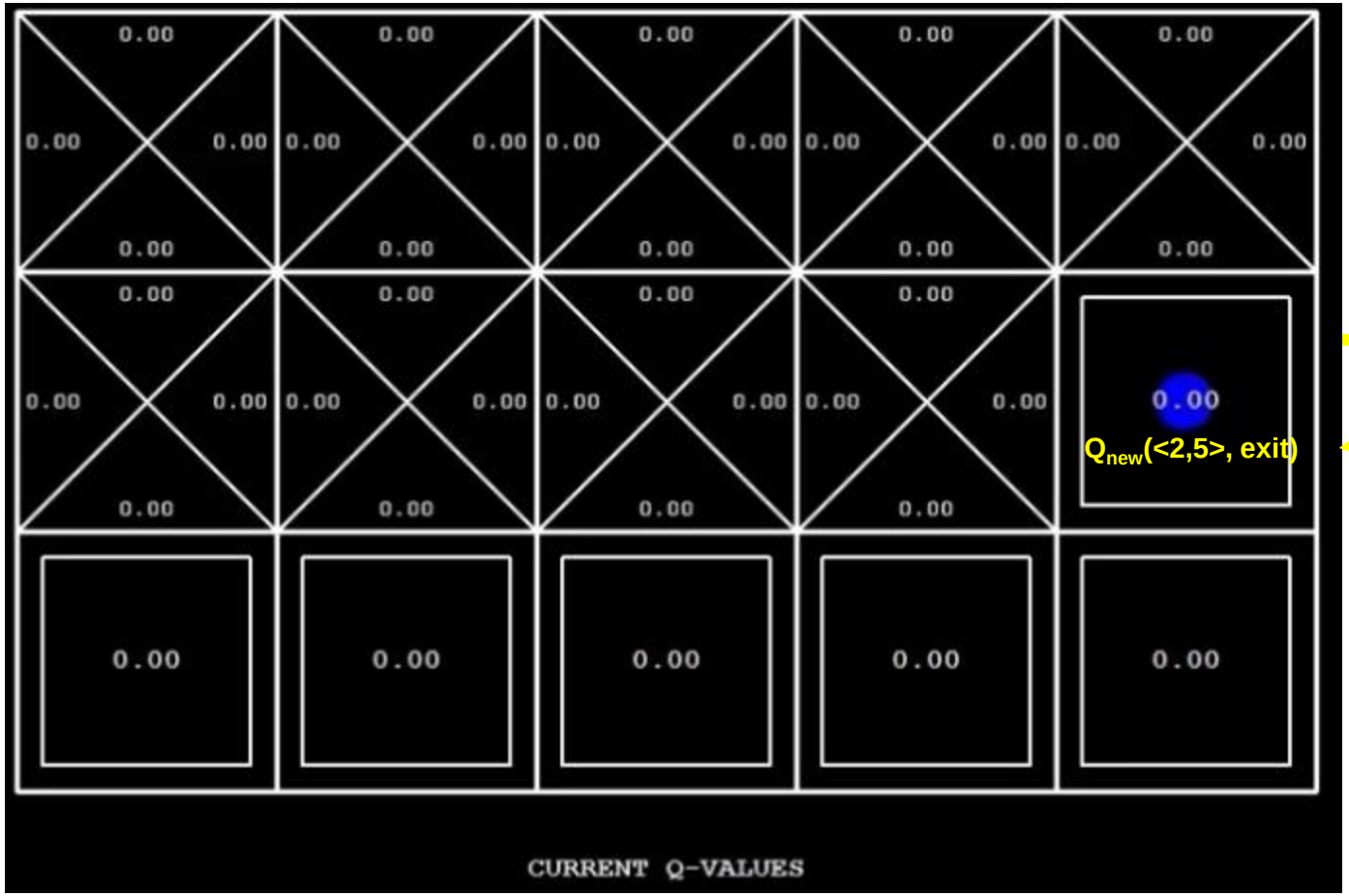
Q-Learning Algorithm: Another Example (Cont'd)



Q-Learning Algorithm: Another Example (Cont'd)



Q-Learning Algorithm: Another Example (Cont'd)



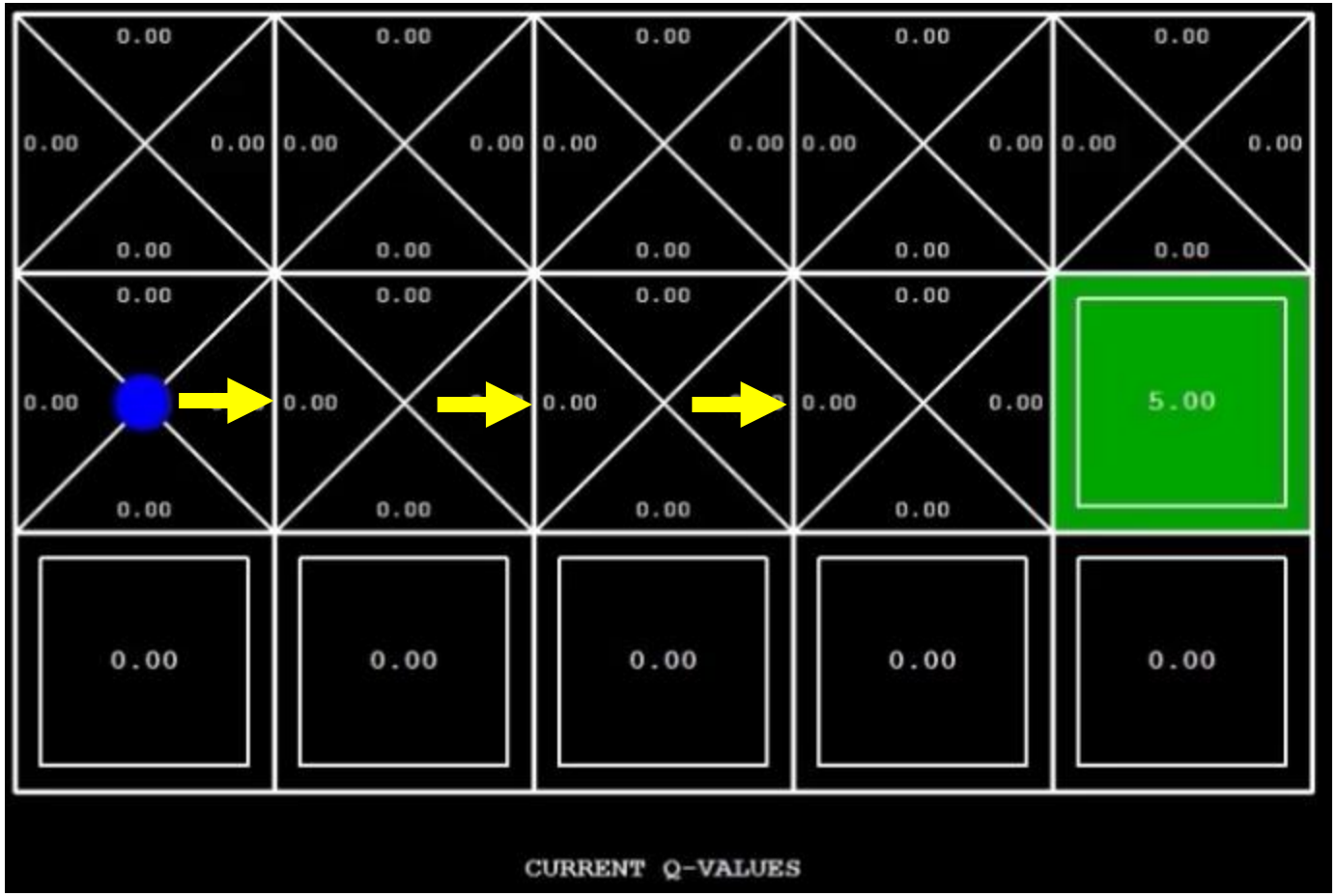
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

Q_{new}(<2,5>, exit)

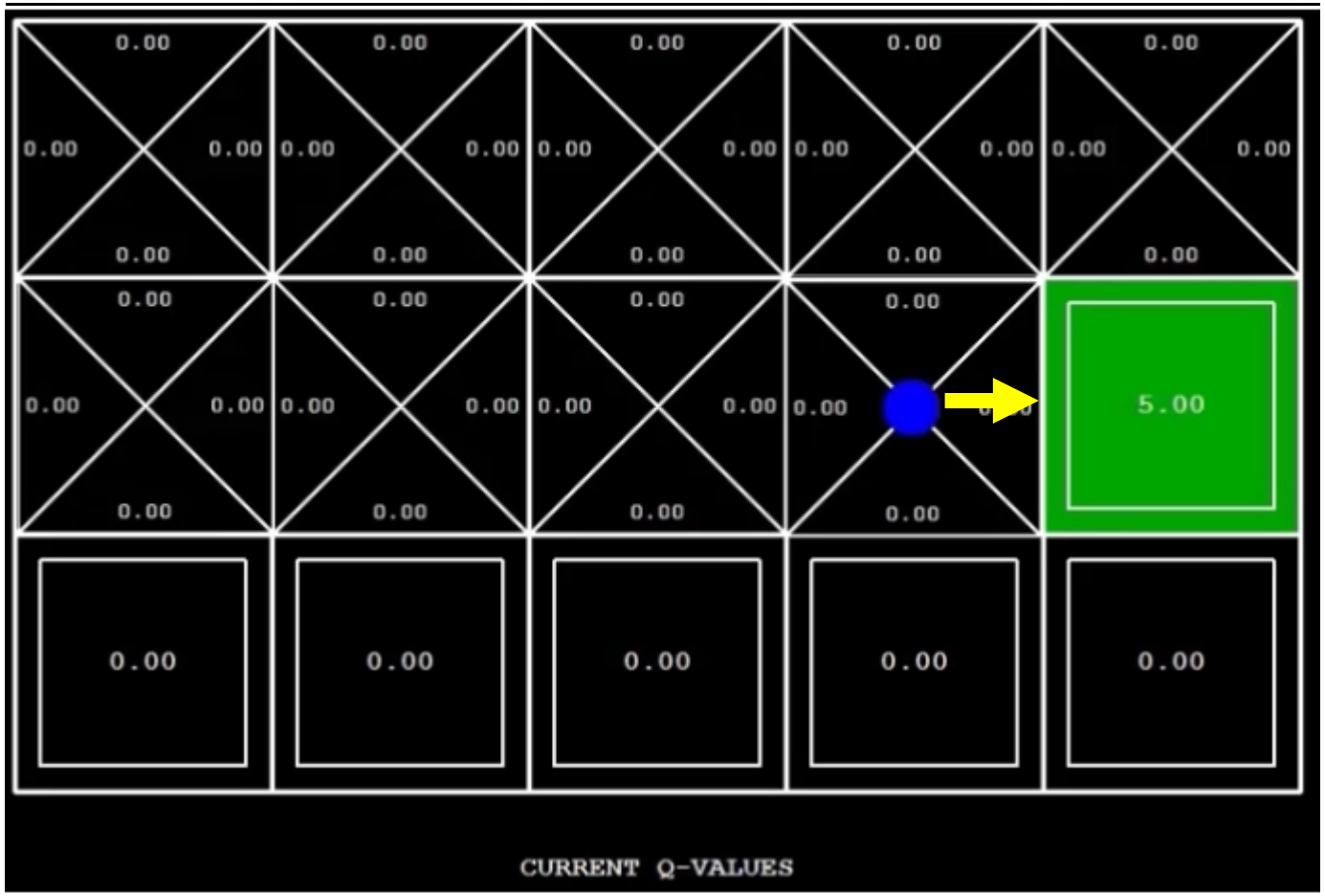
= (1 – 0.5) · Q_{old}(<2, 5>, exit) + 0.5 * 10

Q_{new}(<2, 5>, exit) = 0.5 · 0 + 5 = 5

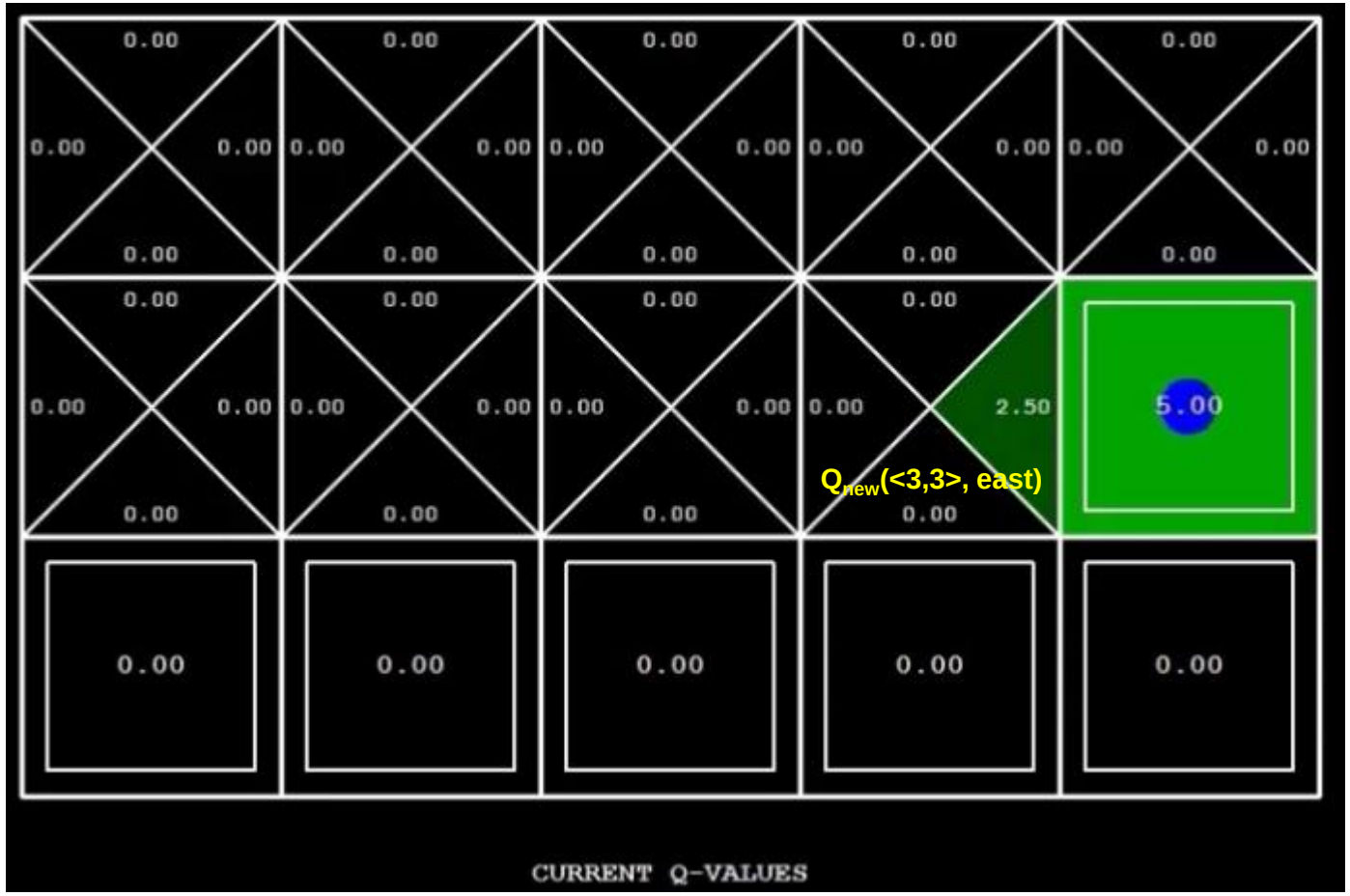
Q-Learning Algorithm: Another Example (Cont'd)



Q-Learning Algorithm: Another Example (Cont'd)



Q-Learning Algorithm: Another Example (Cont'd)

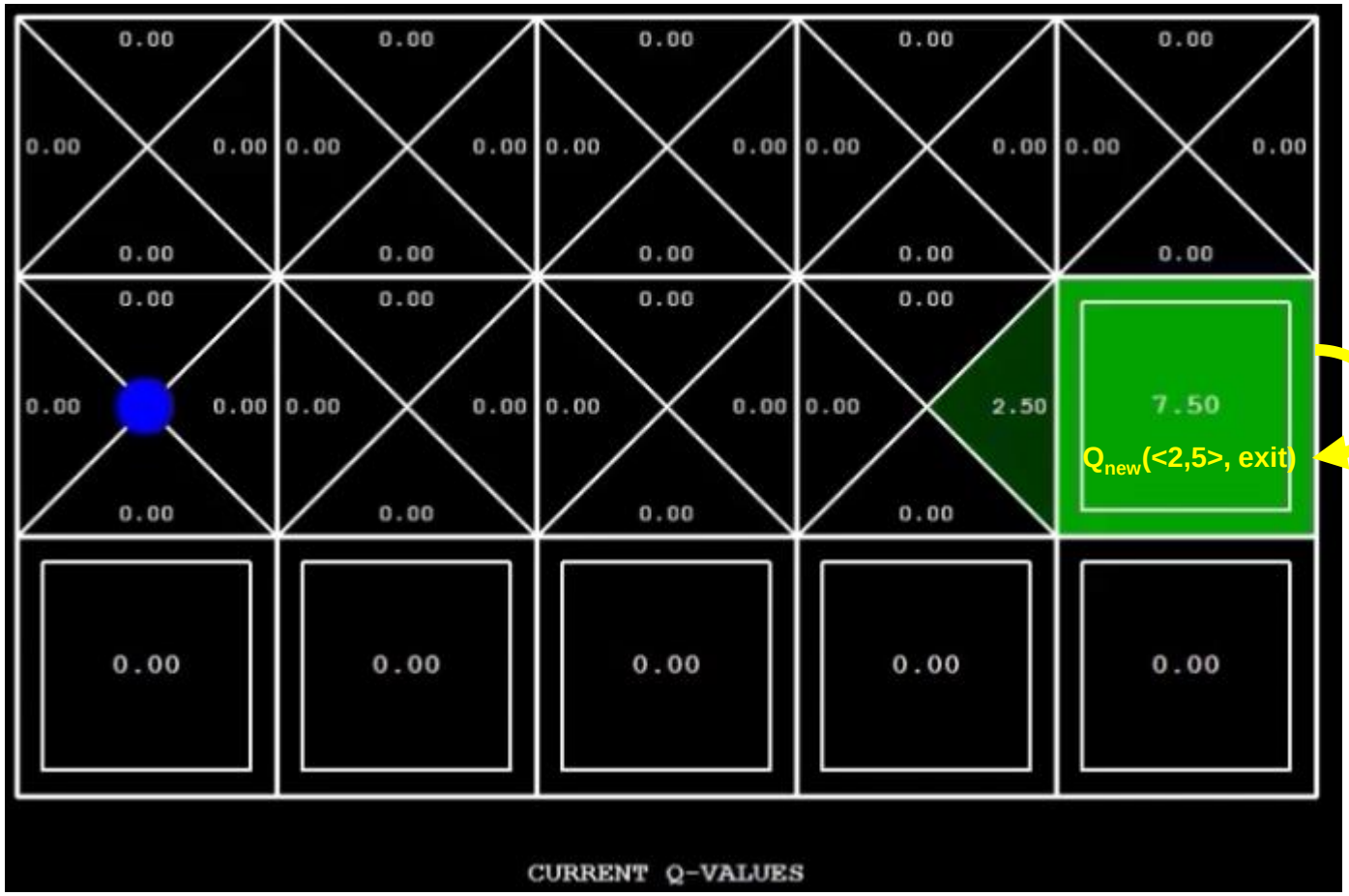


$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

Q_{new}(<3,3>, east) = (1 – 0.5) · Q_{old}(<3,3>, east) + 0.5 * 5

Q_{new}(<3,3>, east) = 0.5 · 0 + 2.5 = 2.5

Q-Learning Algorithm: Another Example (Cont'd)

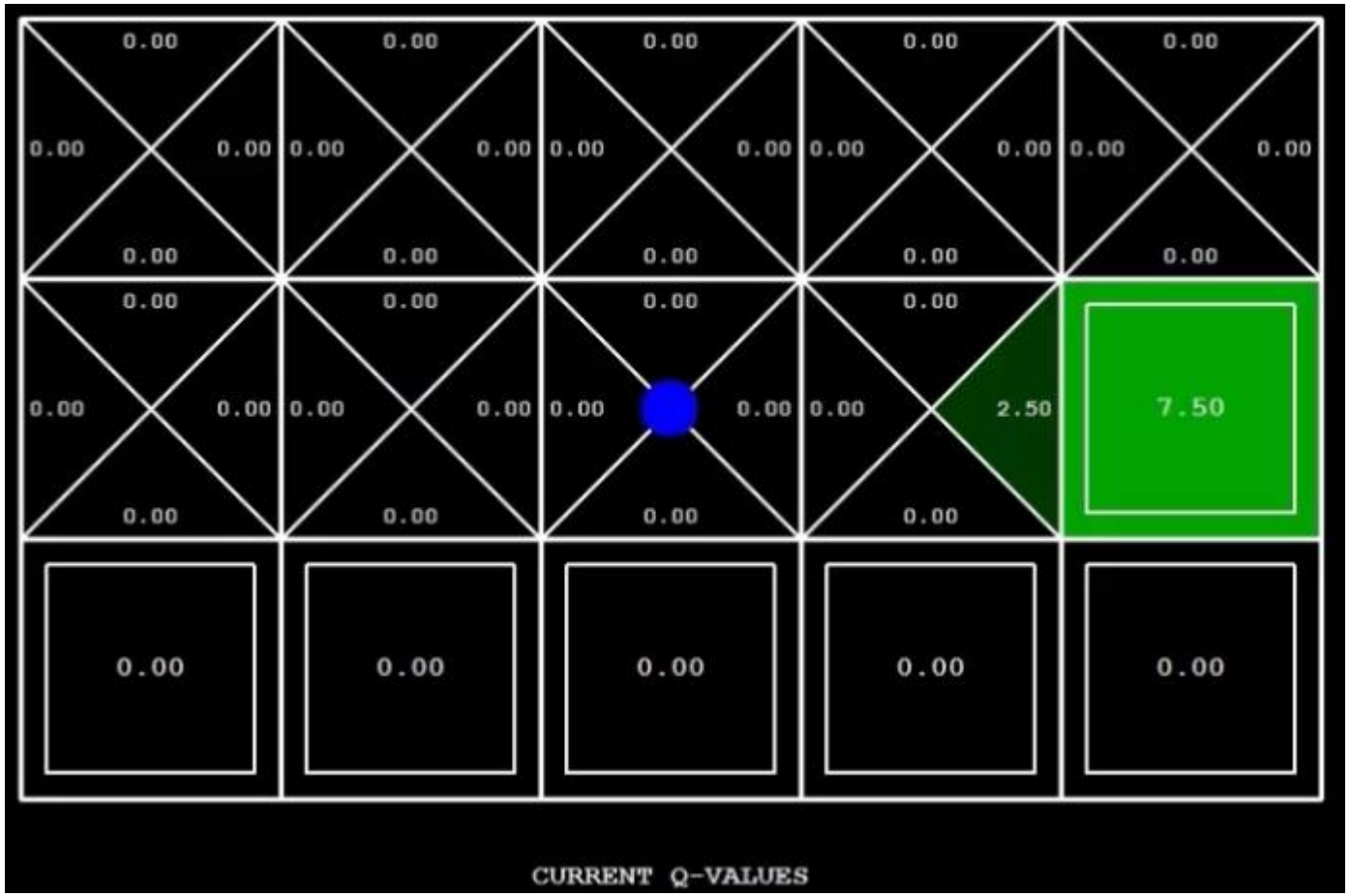


$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

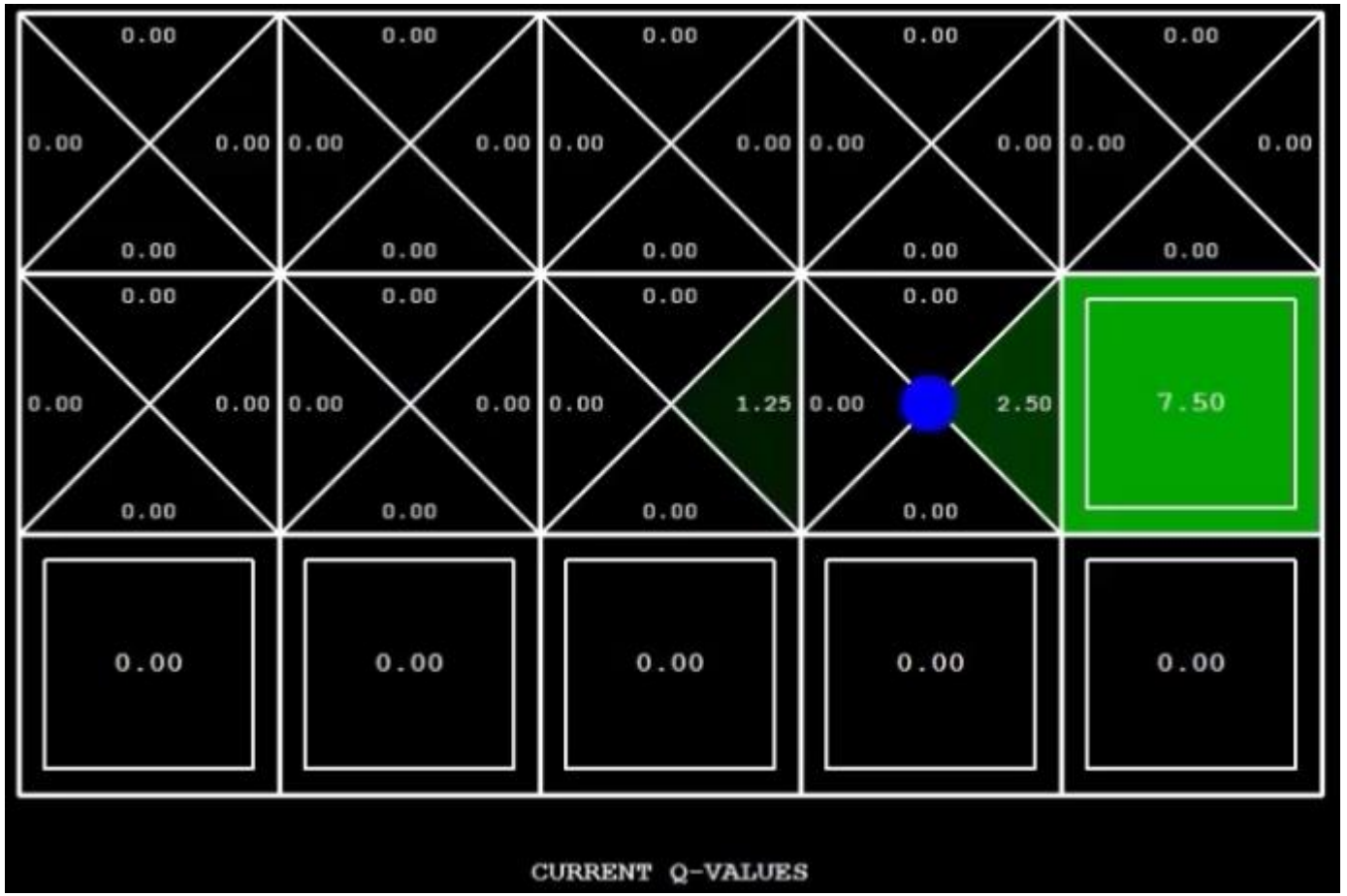
$Q_{new}(<2,5>, exit) = (1 - 0.5) \cdot Q_{old}(<2,5>, exit) + 0.5 * 10$

$Q_{new}(<2,5>, exit) = 0.5 \cdot 5 + 5 = 7.5$

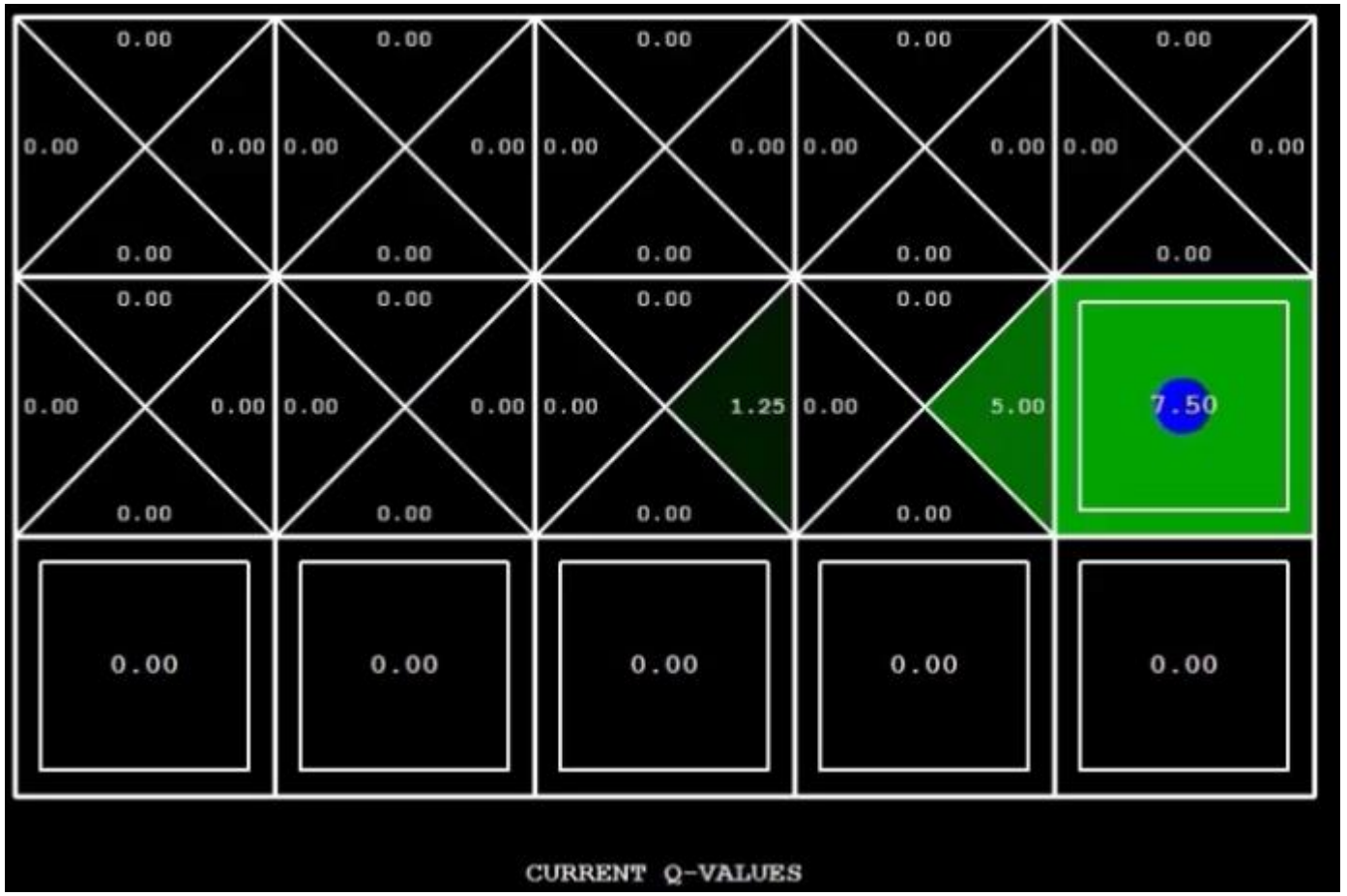
Q-Learning Algorithm: Another Example (Cont'd)



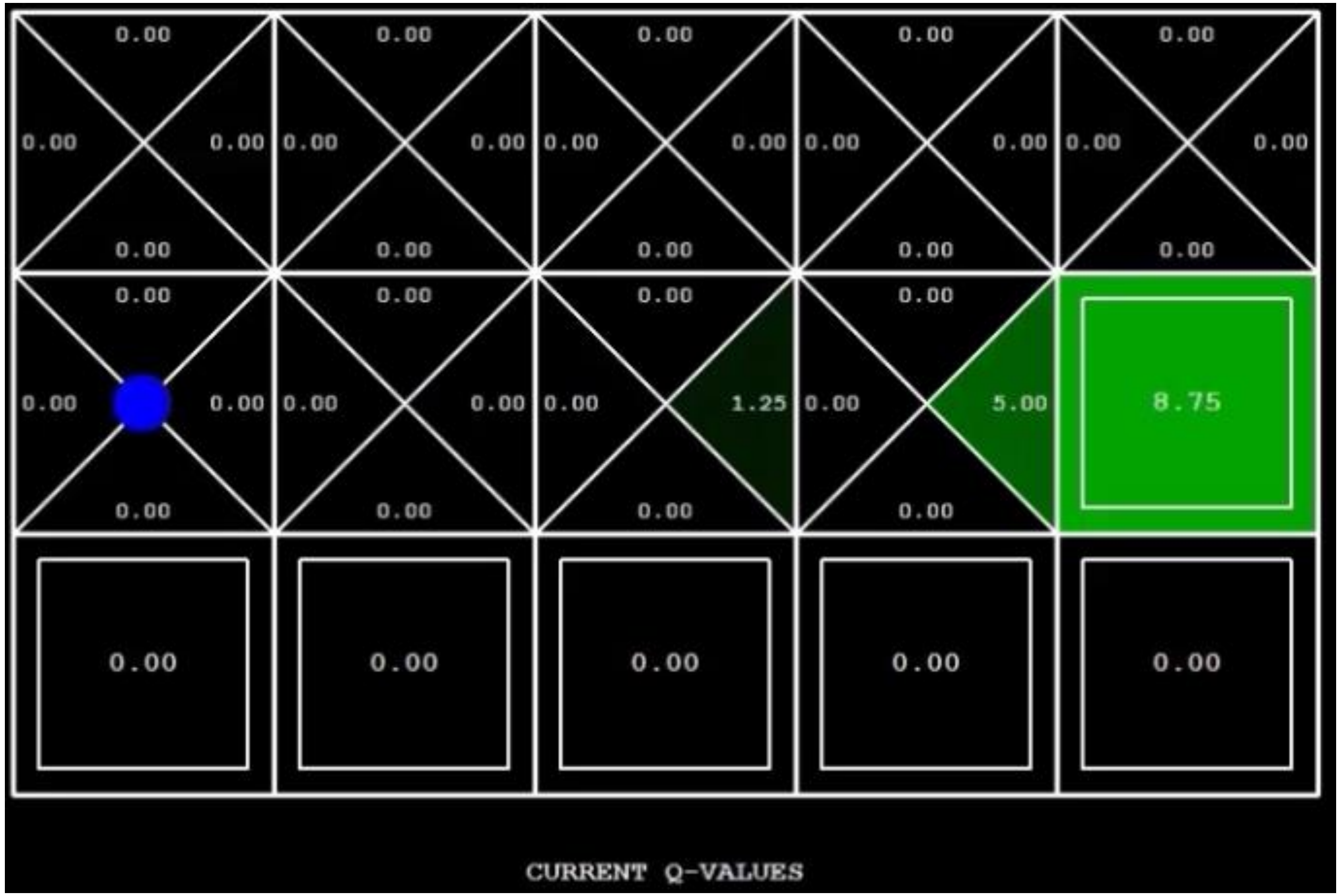
Q-Learning Algorithm: Another Example (Cont'd)



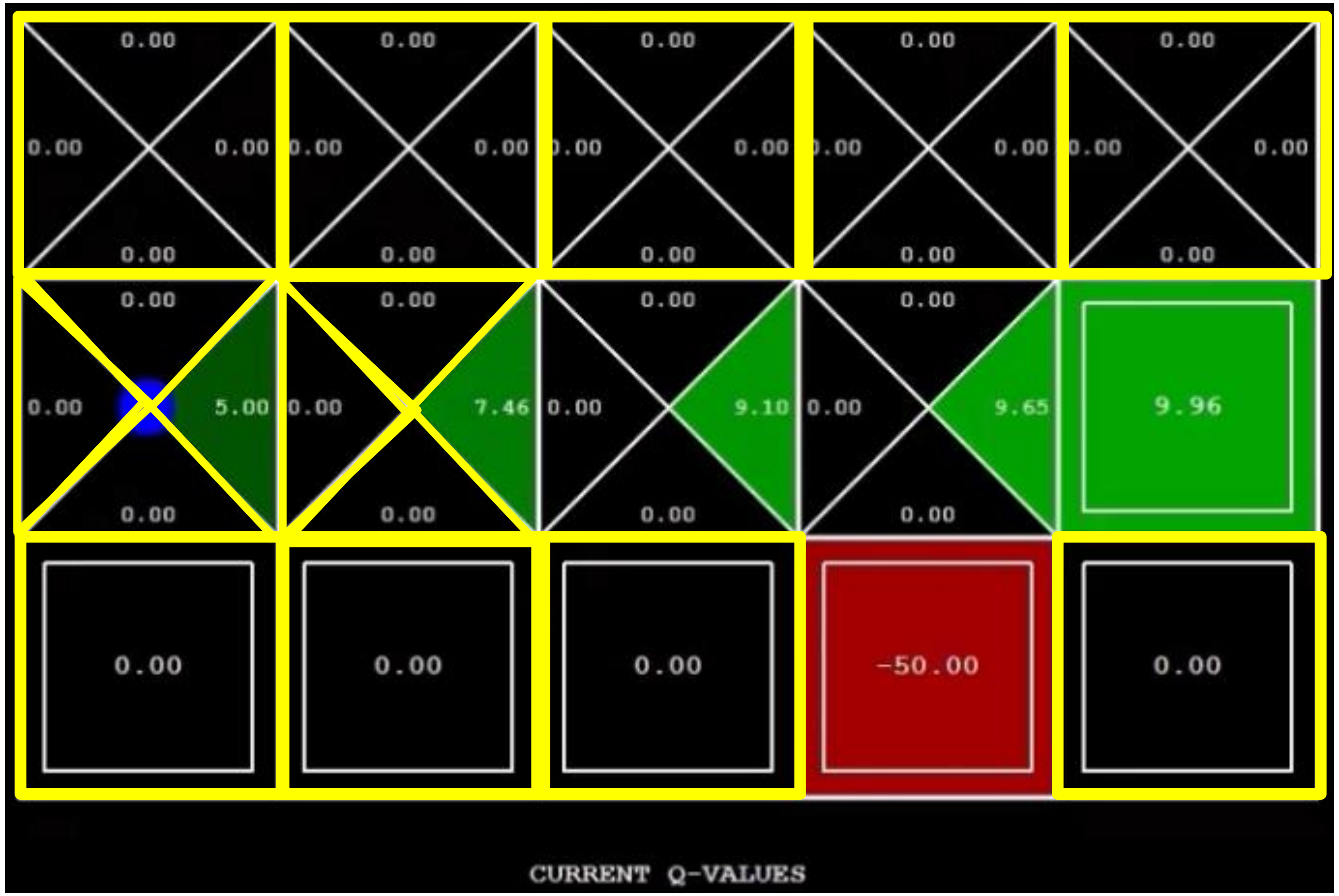
Q-Learning Algorithm: Another Example (Cont'd)



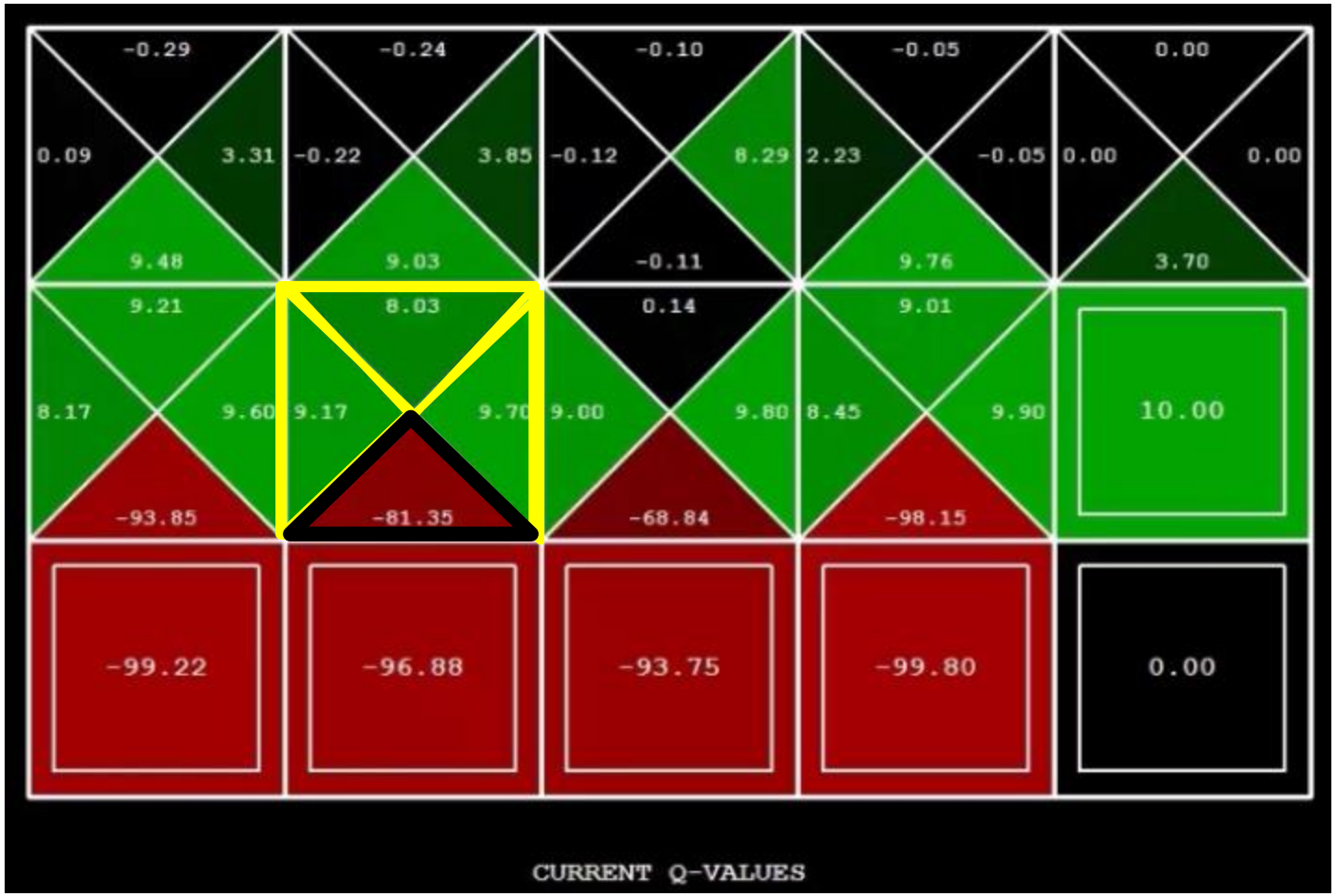
Q-Learning Algorithm: Another Example (Cont'd)



Q-Learning Algorithm: Another Example (Cont'd)

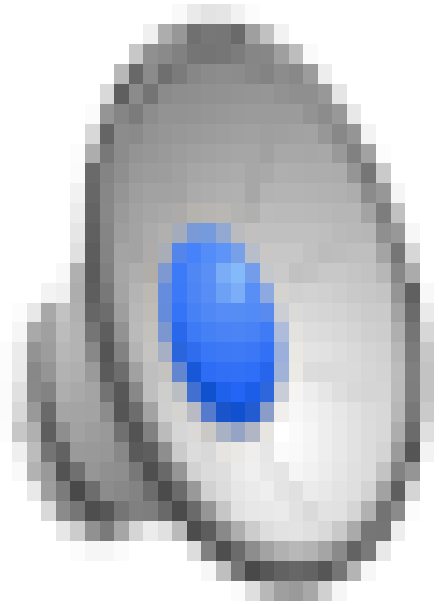


Q-Learning Algorithm: Another Example (Cont'd)

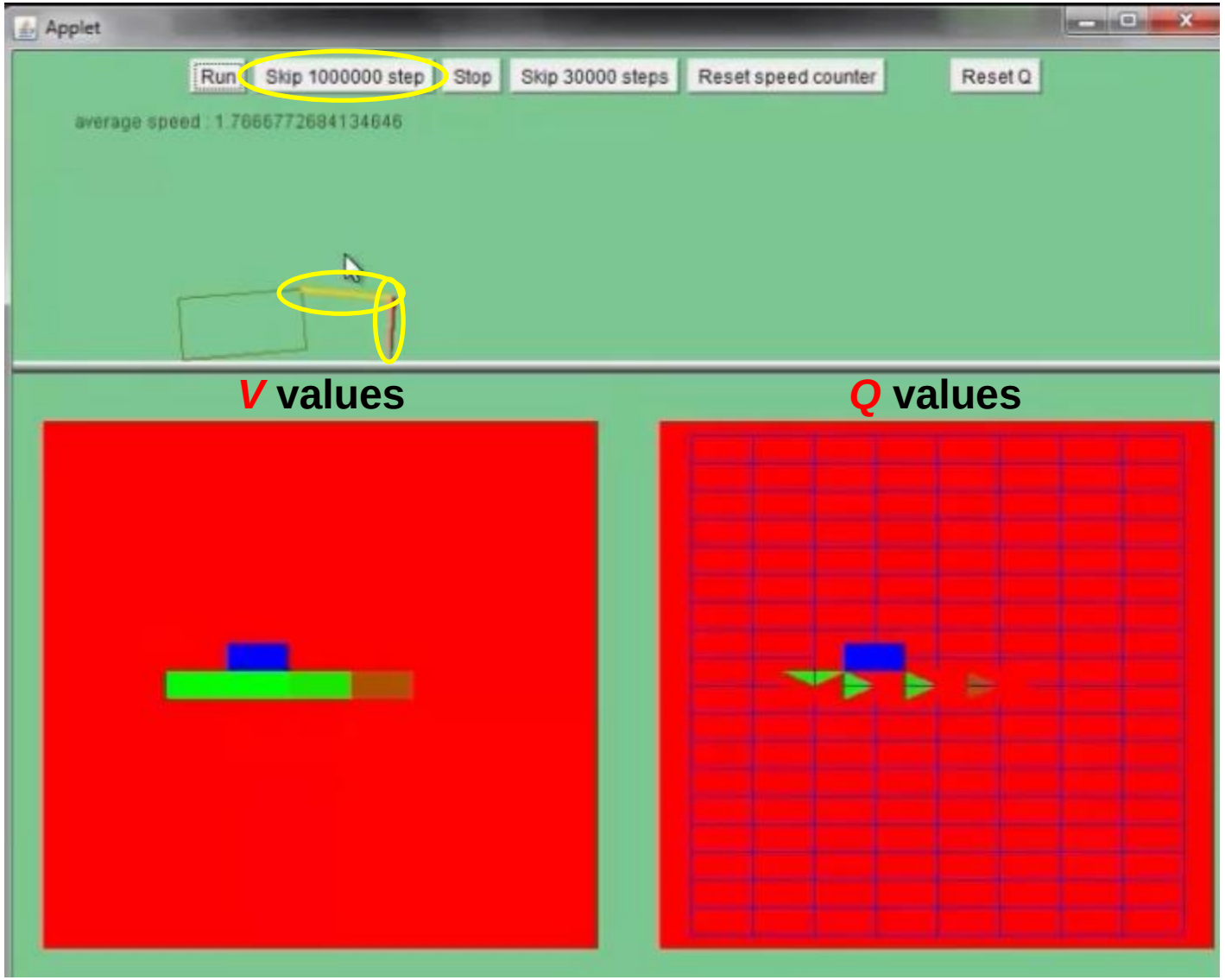


Note: The "goodness" of a state depends on its actions!

Demo of Q-Learning Algorithm (Gridworld)



Q-Learning for Robot Crawler

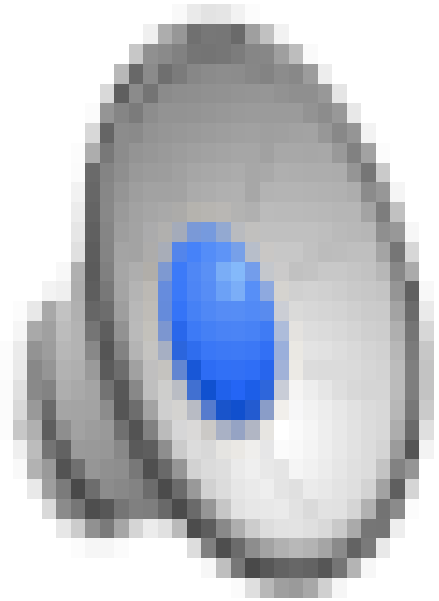


State: 2D state space

[Discretized]

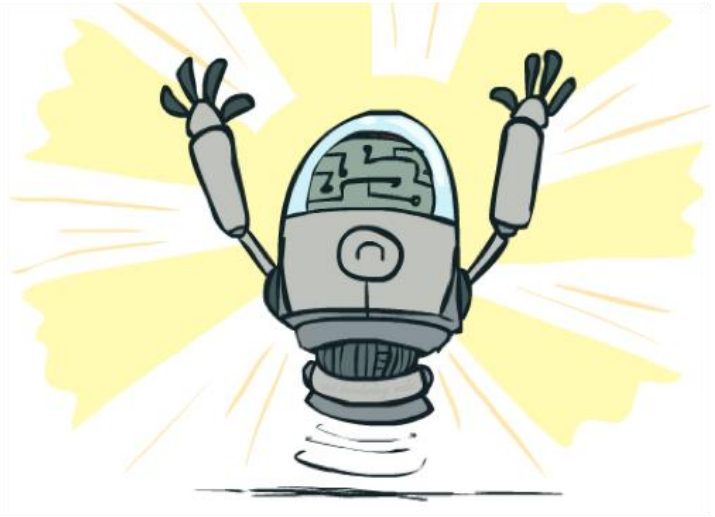
- **Horizontal axis of one joint angle**
- **Vertical axis of the other joint angle**
- **Policy followed 20% of the time**
- **Crawler acts randomly 80% of the time!**

Demo of Q-Learning Algorithm (Crawler)



Q-Learning Properties

- **Amazing result:** Q-learning converges to optimal policy -- even if you are acting suboptimally!



- 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 ***does not matter how you select actions***!

Exploration versus *Exploitation*

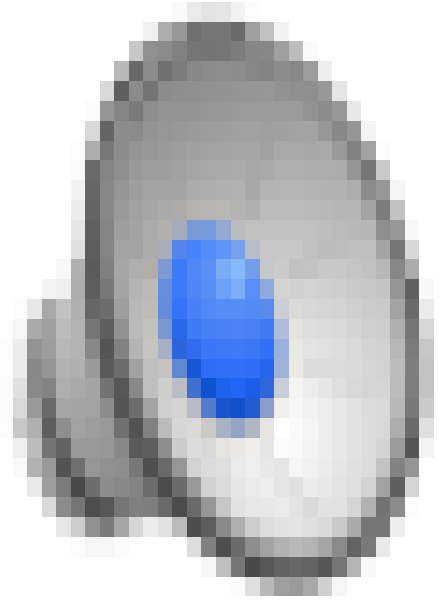
Exploitation

Exploration

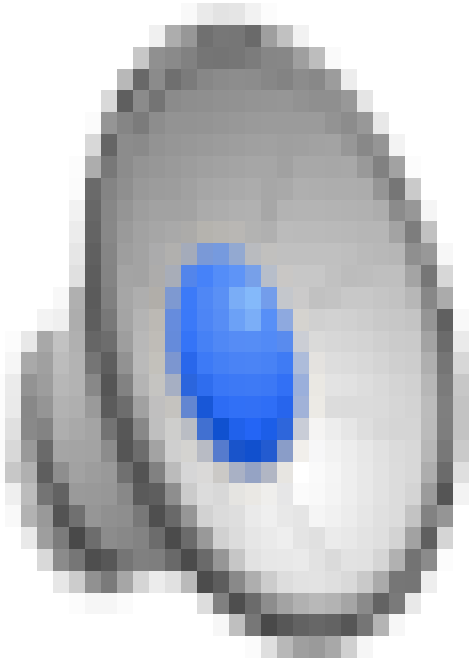
How to Explore?

- Several schemes for forcing exploration
  - Simplest: Random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Q-Learning Demo using Manual Exploration (Bridge Grid)



Q-Learning Demo using Epsilon-Greedy (Crawler)



Exploration Functions

- When to explore?
  - **Random actions:** Explore a fixed amount
  - **Better idea:** Explore areas whose "***badness***" is not (yet) established, eventually stop exploring

- **Exploration function:**
  - Takes a value estimate ***u*** and a visit count ***n***, and returns an optimistic utility. For example:

$$f(u, n) = u + k/n$$

Exploration constant

  - Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$
  - Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

Favors state-actions not seen before

- **Note:** This propagates the "bonus" back to states that lead to unknown states as well!

Exploration Functions (Cont'd)

Note: Remember as time goes by, we have seen many state-action pairs and ***N(s', a')*** goes to infinity.

$$Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a')) \rightarrow \infty$$

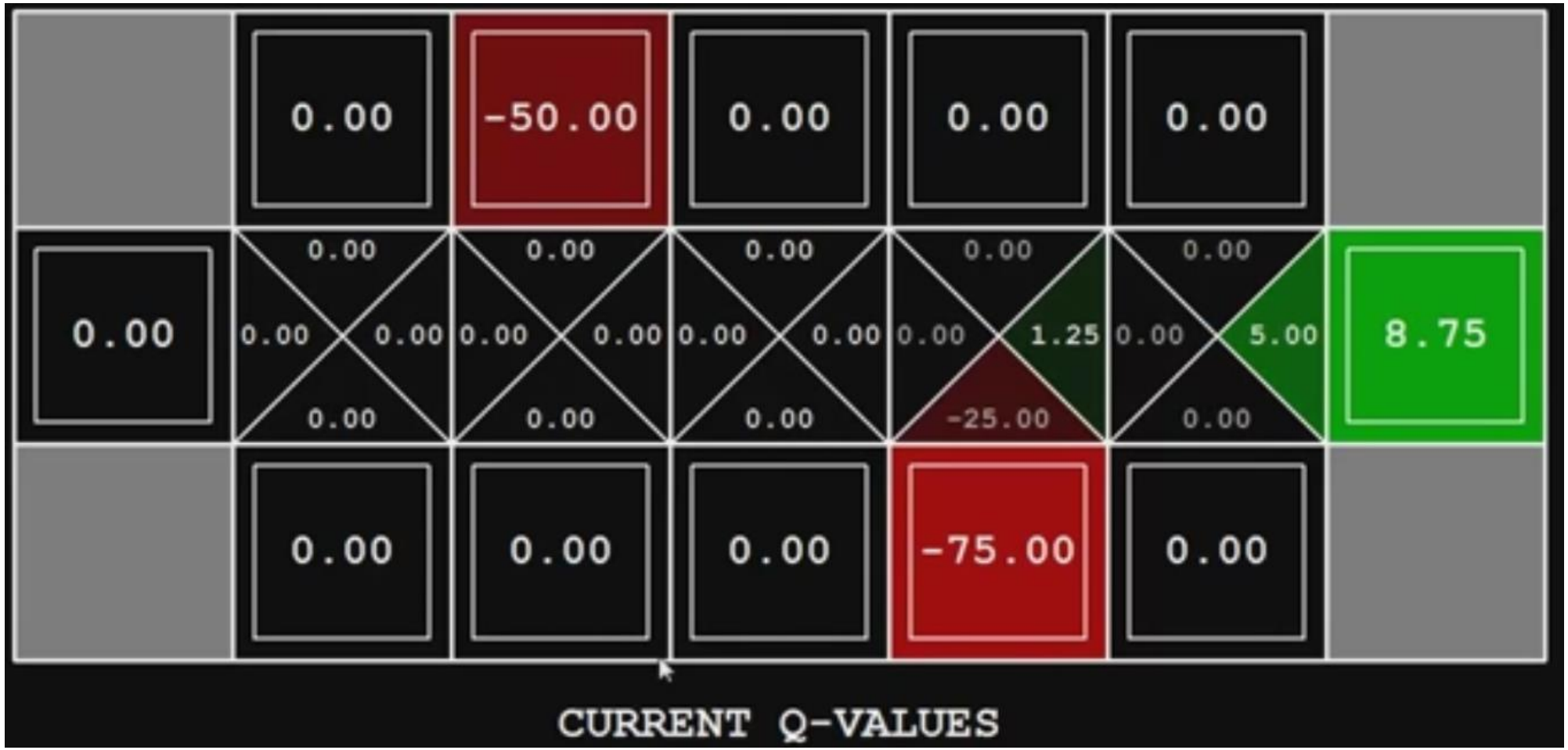
$$f(u,n) = u + k/n$$

$$f(Q(s',a'), N(s',a')) = Q(s',a') + 0 = Q(s',a')$$

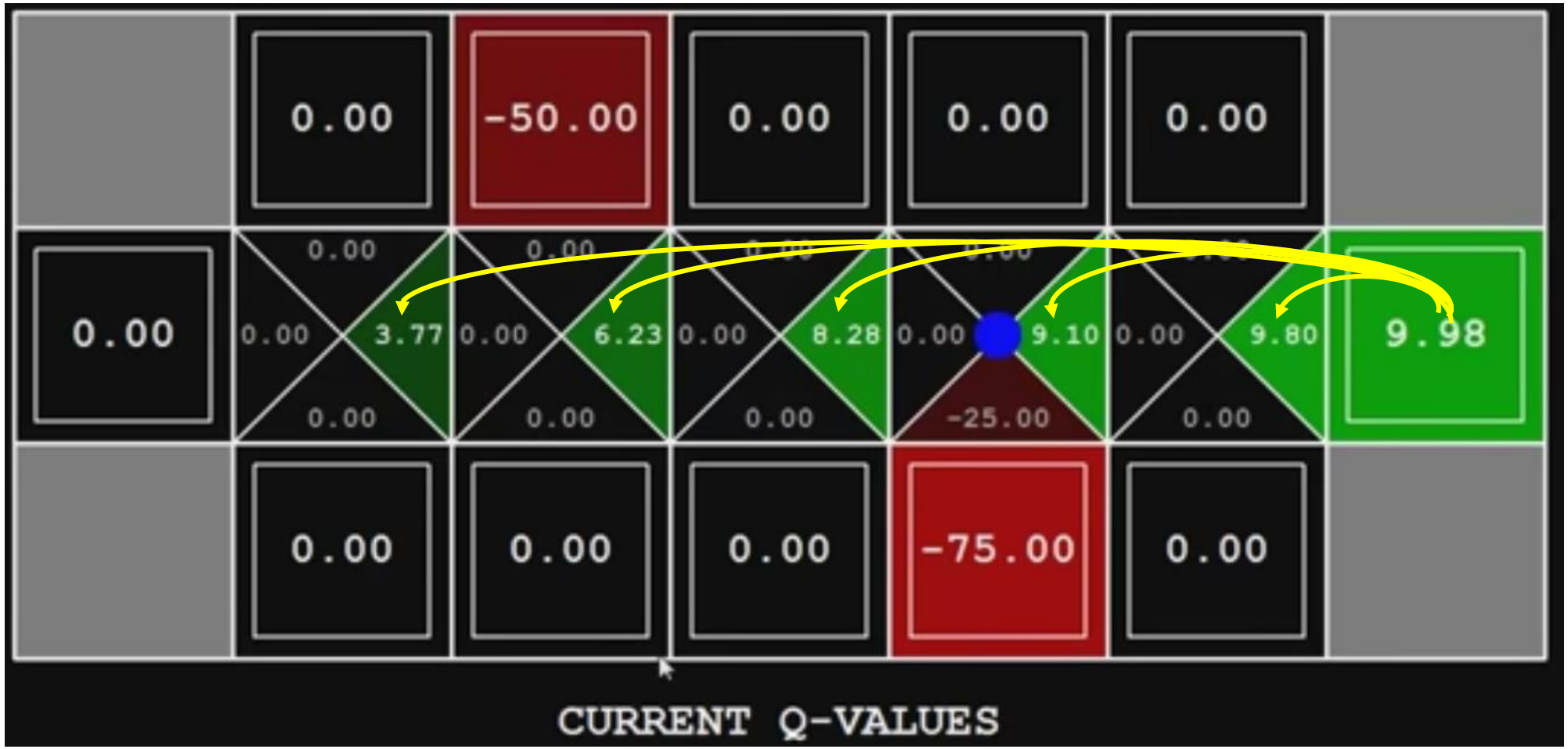
$$Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

Note: At this point, the exploration bonus ***N(s', a')*** will have no effect anymore!

Exploration Functions (Cont'd)



Exploration Functions (Cont'd)

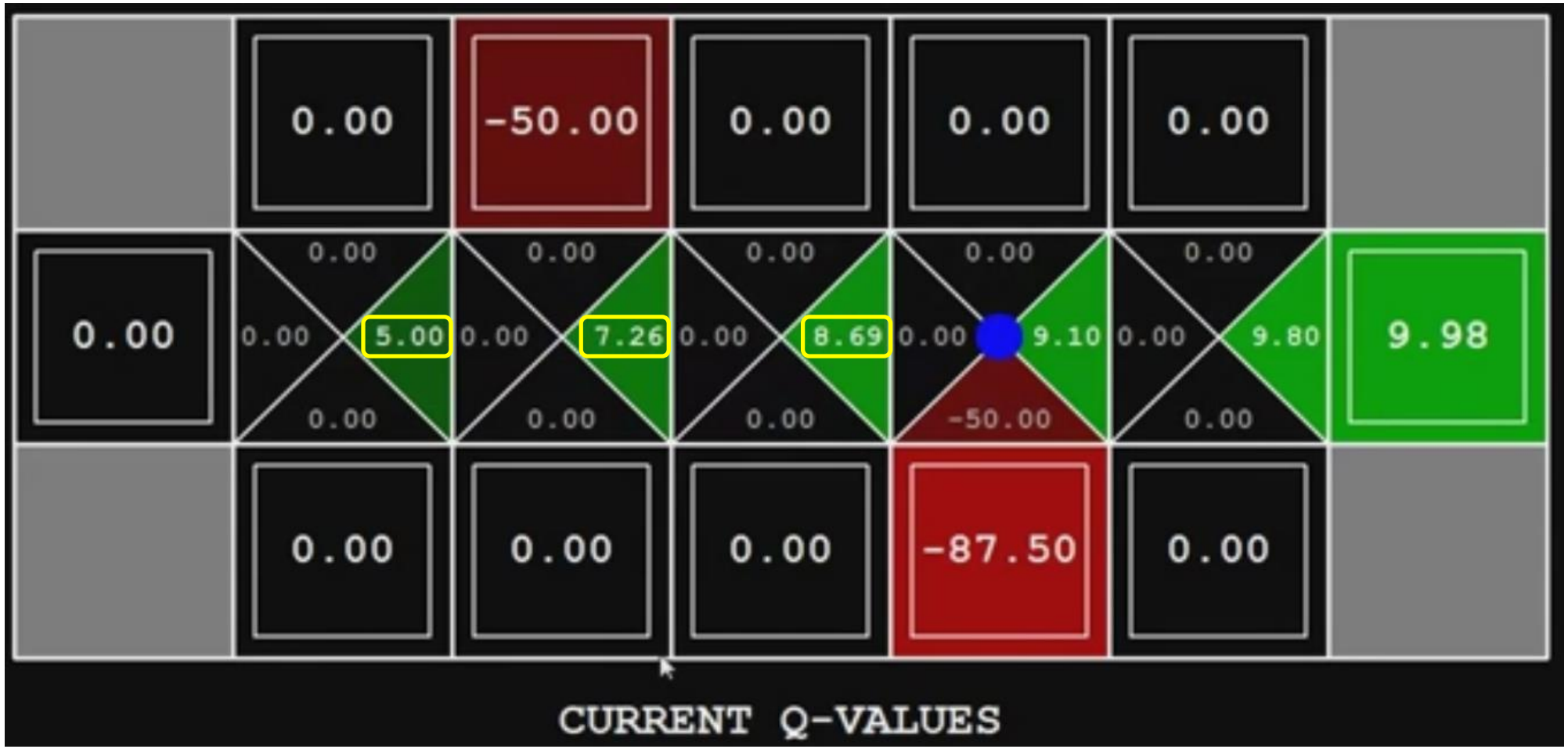


- The values propagate to the left and requires one full run each time to propagate according to the Q-learning implementation.

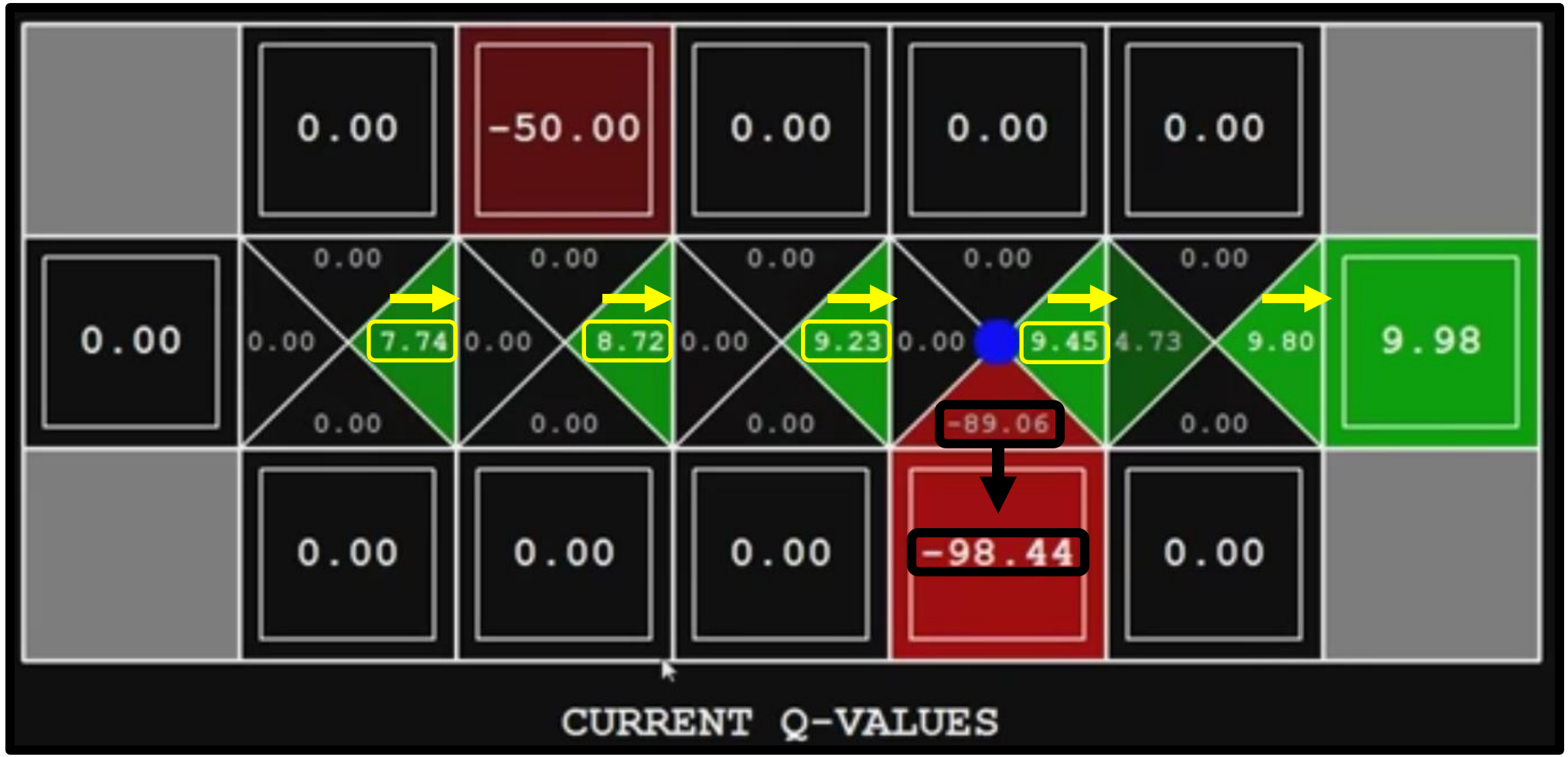
Exploration Functions (Cont'd)

Note: The value of ***8.28*** does not decrease because it is only affected by the maximum value propagated by the next state (i.e., ***9.10***). The negative value of ***-75*** has no effect on its current value!

Exploration Functions (Cont'd)



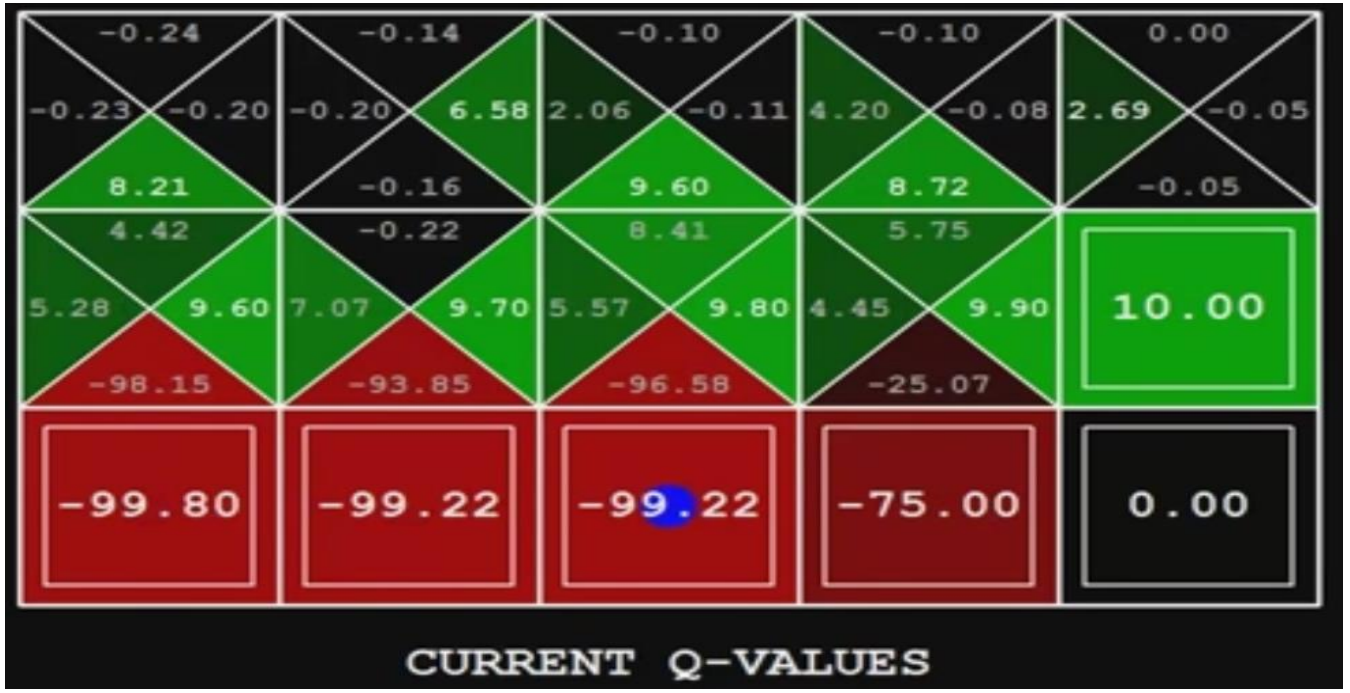
Exploration Functions (Cont'd)



Note: We are still learning that this path is "good".

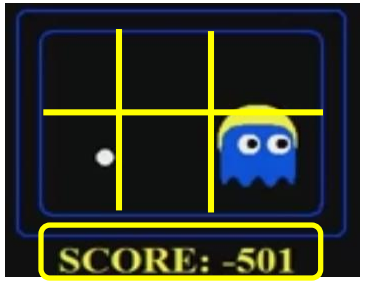
Note: We are still learning that this path is "**bad**".

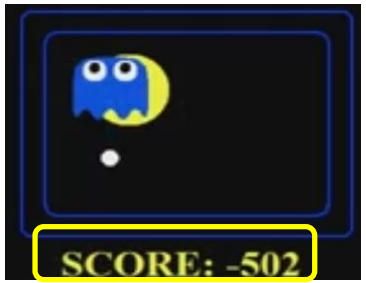
Exploration Functions (Cont'd)

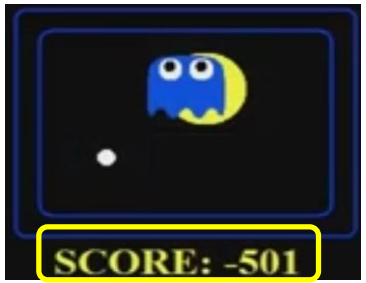


- Another Q-learning example with enough randomness (high ε): Most of q-states are visited!

Exploration Functions (Cont'd)

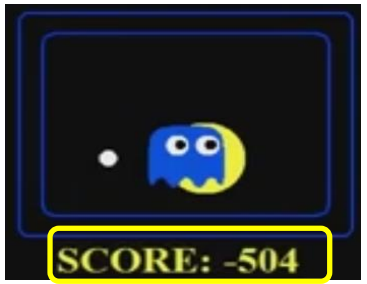






State:

- 6 positions for Pacman
- 6 positions for the ghost
- 36 possible states.



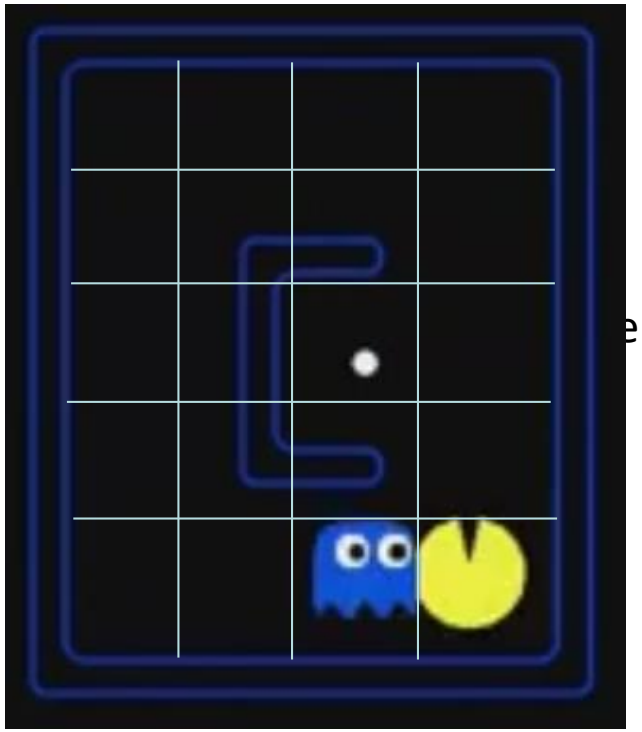
- Pacman using Q-learning with $\varepsilon = 0.02$.

Pacman is learning all the possibilities of he getting eaten by the ghost!

Exploration Functions (Cont'd)

Silent Training:

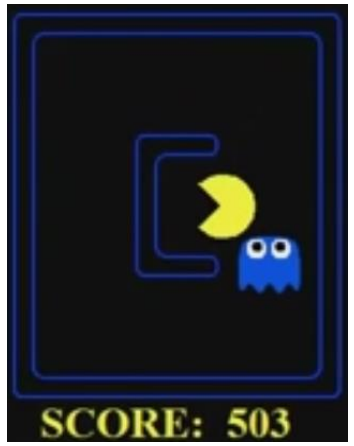
- Use the Q-learning algorithm to learn how to beat the ghost (**2000 episodes**).
- An episode terminates with Pacman winning or dying.
- Run Pacman using the extracted policy.



…… Pacman is winning all the time!

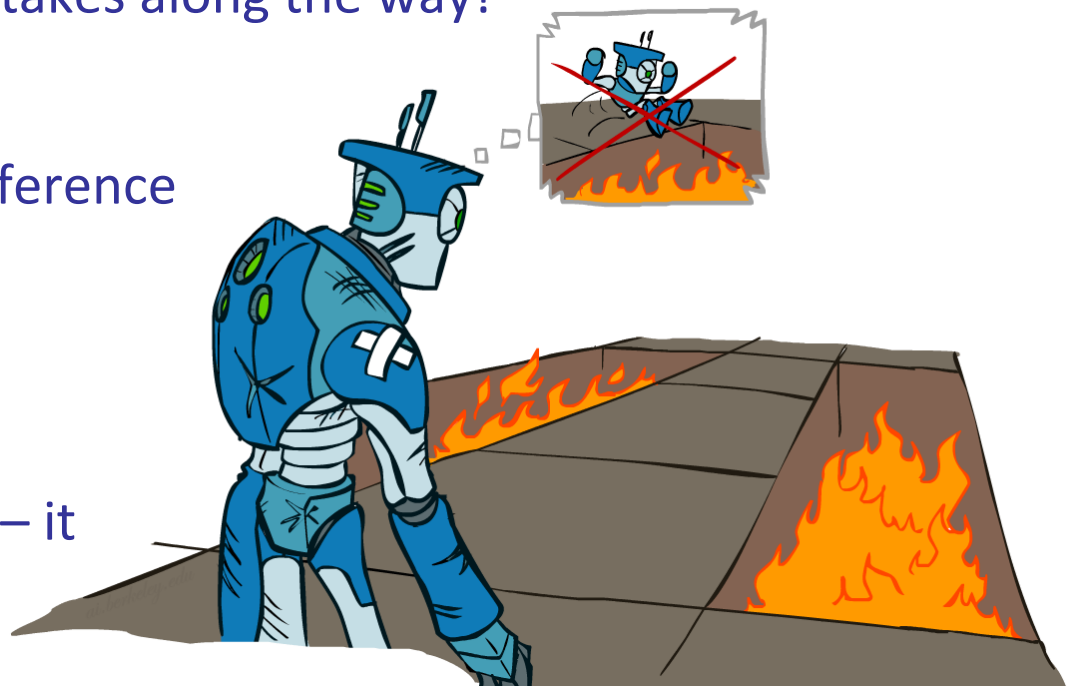
- e same process!

14^2 (Pacman + Ghost) x 4 (2 food pellets) states



Regret

- Even if you learn the optimal policy, you still make mistakes along the way!



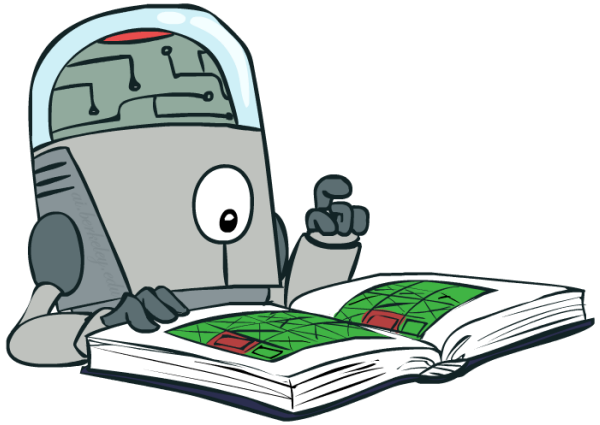
- ***Regret*** is a measure of your total mistake cost: The difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards.

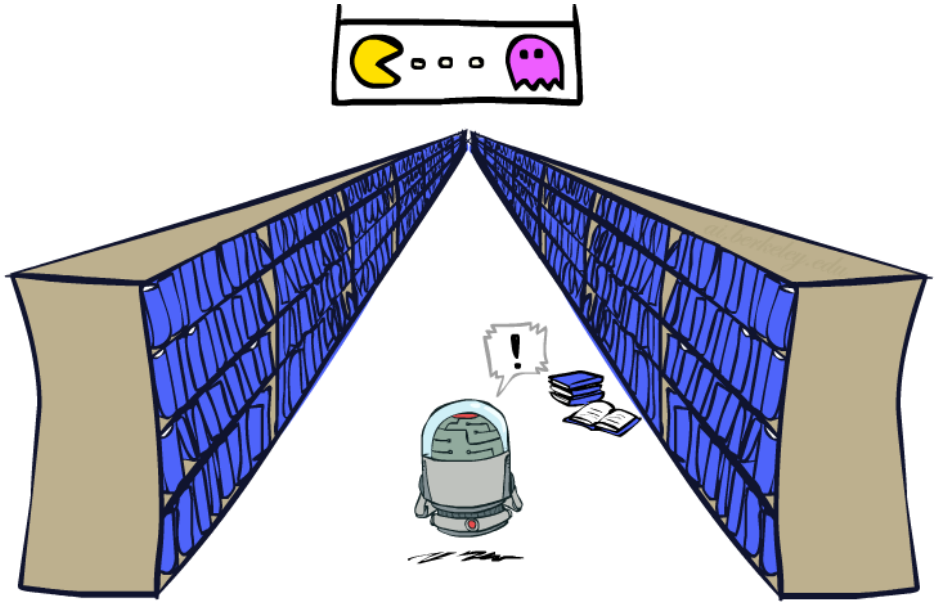
- Minimizing regret goes beyond learning to be optimal – it ***requires optimally learning to be optimal***.

- **Example:** Random exploration and exploration functions both end up optimal, but ***random exploration has higher regret***.

Generalizing Across States

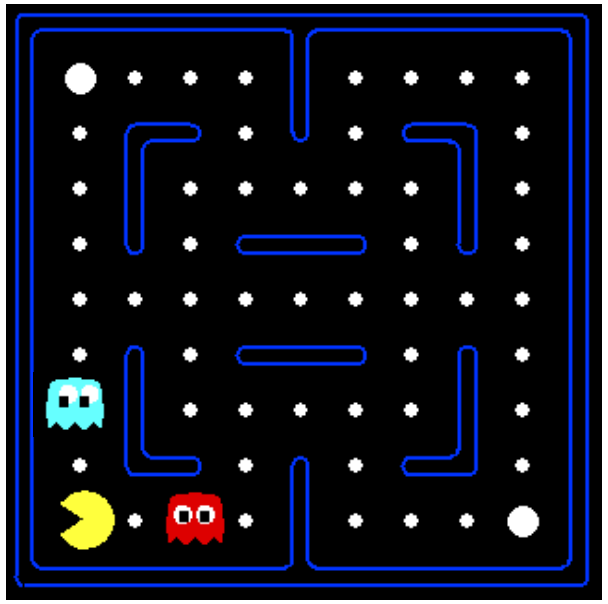
- Basic Q-Learning ***keeps a table*** of ***all q-values***
- In realistic situations, we cannot possibly learn about every single state!
  - **Too many states to visit them all in training.**
  - **Too many states to hold the q-tables in memory**
- Instead, we want to ***generalize***:
  - ***Learn about some small number of training states*** from experience.
  - Generalize ***that experience to new***, ***similar*** situations.
  - This is a ***fundamental idea in machine learning***, and we will see it over and over again.



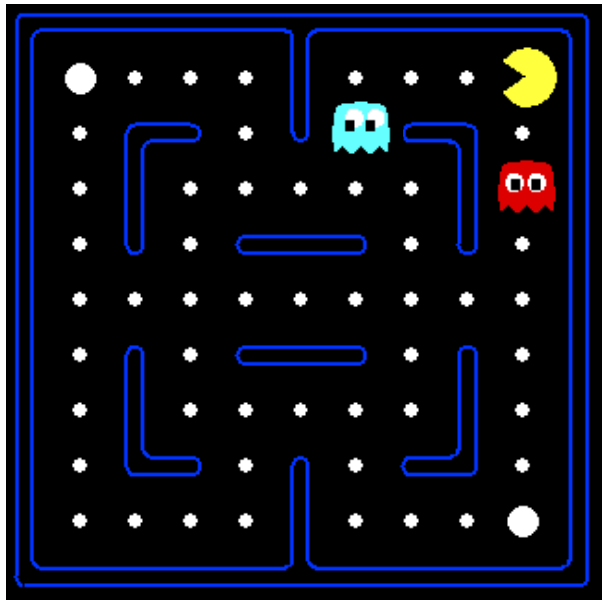


“Similar” States: Revisited [Pacman Example]

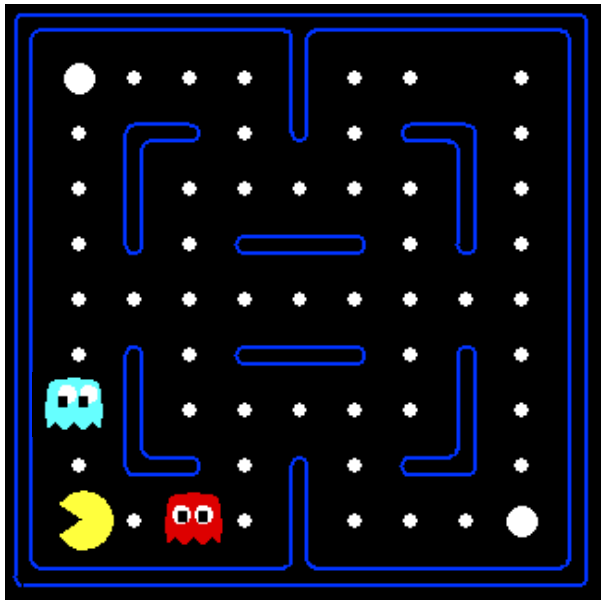
“***Bad***” state discovered through experience.



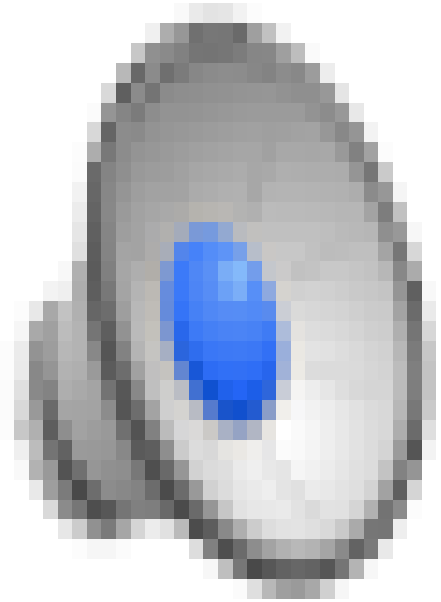
“***Unknown***” (i.e., ***completely new***) state using naïve q-learning.



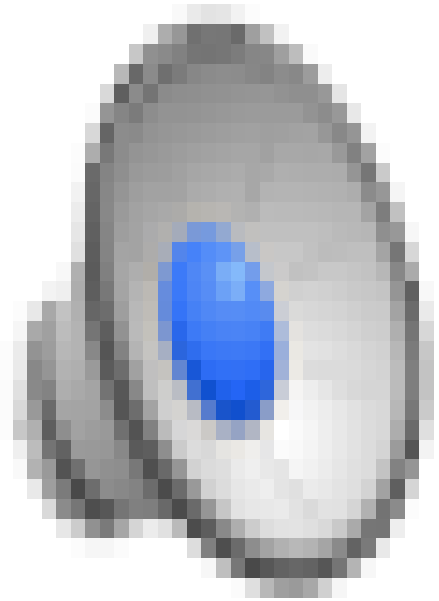
A worst example of unknown state!



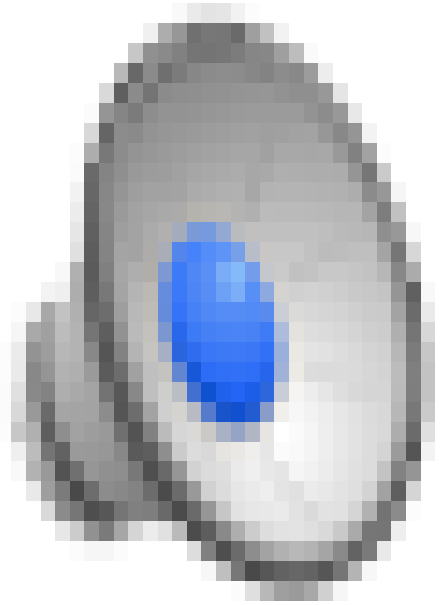
Q-Learning Demo – Pacman (Tiny – Watch All)



Q-Learning Demo – Pacman (Tiny – Silent Train)

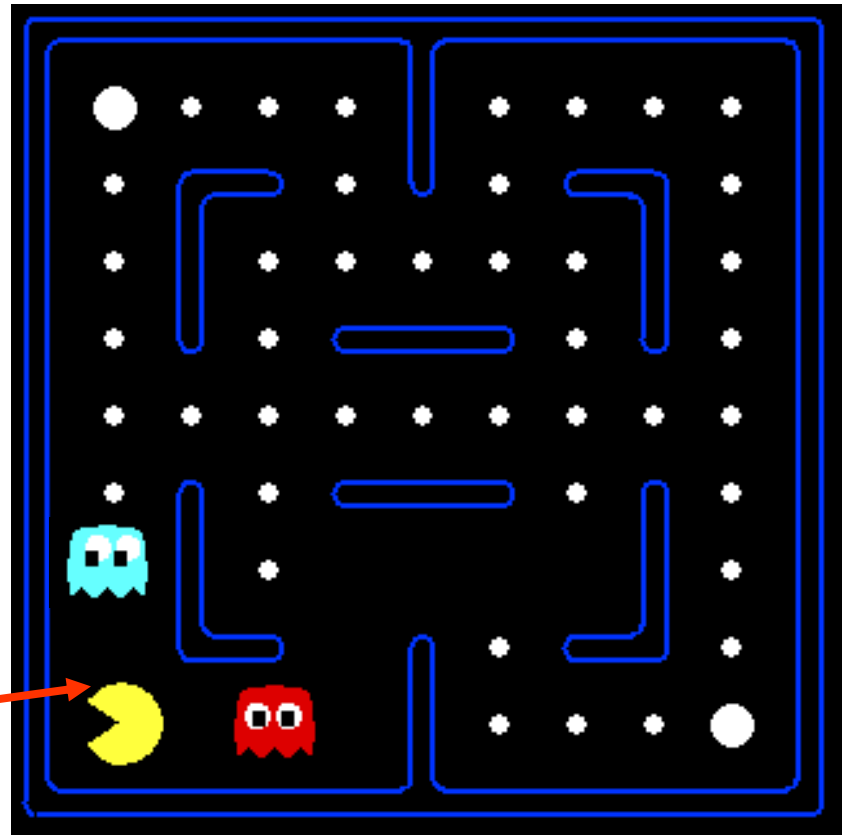


Q-Learning Demo – Pacman (Tricky – Watch All)



Feature-Based Representations

- **Solution:** Describe a state using a vector of features (properties)
  - **Features:** ***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 / (distance to dot)^2
    - Is Pacman in a tunnel? (0/1).
    - …… etc.
    - Too specific feature
    - Is this the exact state I am interested in?



- **Note 1:** Can also describe a ***q-state (s, a)*** with features (e.g., action moves closer to food).
- **Note 2:** However, do not add many "***specific***" features which would require many weights to train!

Linear Value Functions

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

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

"Non-linear" term in the weights!!!

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a) + \boxed{w_1^2} f_{n+1}(s,a)$$

Hard to learn

- **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

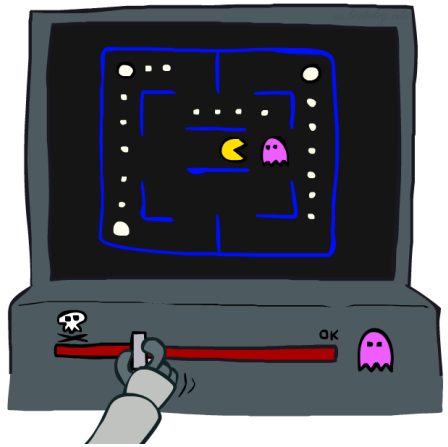
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with **linear Q-functions**:

$$\text{transition} = (s, a, r, s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$$

difference = sample – current estimate



We do not update Q's anymore!

[redacted] Exact Q's

Update the weights of the features instead!

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$

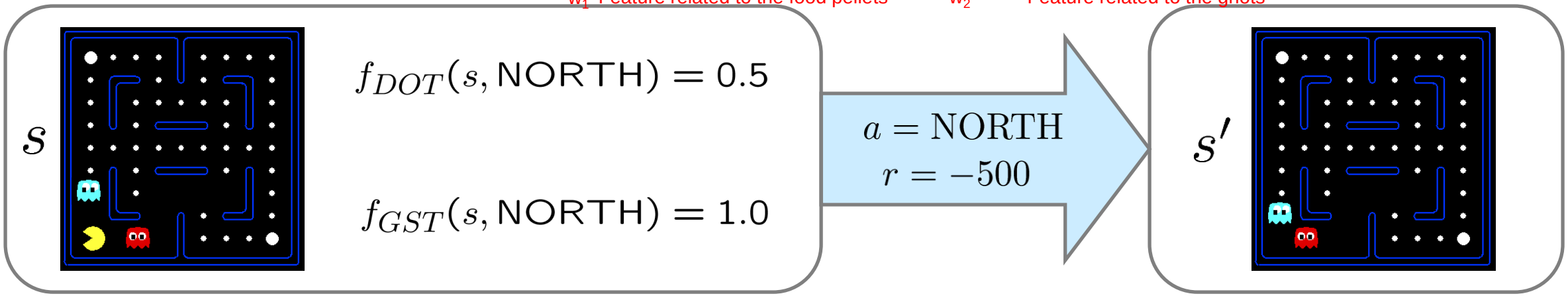
Approximate Q's

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

Example: Q-Pacman

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

w_1 Feature related to the food pellets w_2 Feature related to the ghots



Current estimate $Q(s, \text{NORTH}) = +1$

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

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

Sample – Current estimate

difference $= -501$

$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

Feature Encoding: More afraid from the ghost than eager to get the food pellets!

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

Approximate Q-Learning: Pacman Example

Let us play another episode (which is shown below after few moves)!

So far, Pacman has explored only the states where he can eat food pellets and get 9 points each time!

Question: What about this State?

Note: Pacman moved up ignoring the ghost!

Answer: Now, it knows that this is a “bad” state

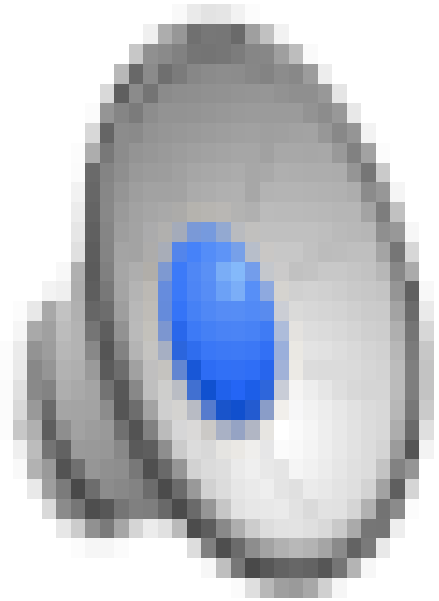
Question:

What is wrong with the ghosts?

First time, Pacman explores a state where he gets eaten by a ghost! (Is this really a “bad” state?)

Note: The feature function is not “encoding” the power pellets! Pacman is simply ignoring them!

Approximate Q-Learning Demo – Pacman



Deep Reinforcement Learning

- Approximate Q-learning using deep neural networks (DQN)
- Input layer → Features (or raw data)
- Output layer → Estimate of Q(s,a)

- Instead of handcrafted features, let the network learn the features by optimizing the weights.

Training DQNs

1. Initialize replay memory capacity.
2. Initialize the network with random weights.
3. *For each episode:*
    1. Initialize the starting state.
    2. *For each time step:*
        1. Select an action.
            - *Via exploration or exploitation*
        2. Execute selected action in an emulator.
        3. Observe reward and next state.
        4. Store experience in replay memory.
        5. Sample random batch from replay memory.
        6. Preprocess states from batch.
        7. Pass batch of preprocessed states to policy network.
        8. Calculate loss between output Q-values and target Q-values.
            - Requires a second pass to the network for the next state
        2. Gradient descent updates weights in the policy network to minimize loss.

https://deeplizard.com/learn/video/xVkPh9E9GfE

OpenAI gym

Introduction

OpenAI gym

- A toolkit for developing and comparing reinforcement learning algorithms.
- The Gym library is a collection of environments (test problems) that we can use test the reinforcement learning algorithms we develop.
- Gym has many environments ranging from simple text based games to Atari games.
- https://gym.openai.com/docs/

Environment

- env = gym.make('FrozenLake-v0')
- env.reset()
- next_state, reward, done, info = env.step(env.action_space.sample())
- env.close()

- https://gym.openai.com/envs/FrozenLake-v0/
- https://gym.openai.com/envs/MountainCar-v0/
- https://gym.openai.com/envs/#classic_control

References

- Lecture slides adapted from: UC Berkeley CS188 Intro to AI, http://ai.berkeley.edu/home.html
- https://gym.openai.com/docs/
- Andrej Karpathy, Deep Reinforcement Learning: Pong from Pixels, http://karpathy.github.io/2016/05/31/rl/