Reinforcement Learning for Games

Matthew Guzdial

guzdial@ualberta.ca

Announcements

- **Nov 23 Course Totals**: Median 72.4, average 70.17, out of 80.05 base points and 83 with extra credit (due to quiz 1 and 3).
- Quiz 5 Friday @ 11am (no lecture), 48 hours, etc.
- Assignment 5 due in two Mondays (helper video online)

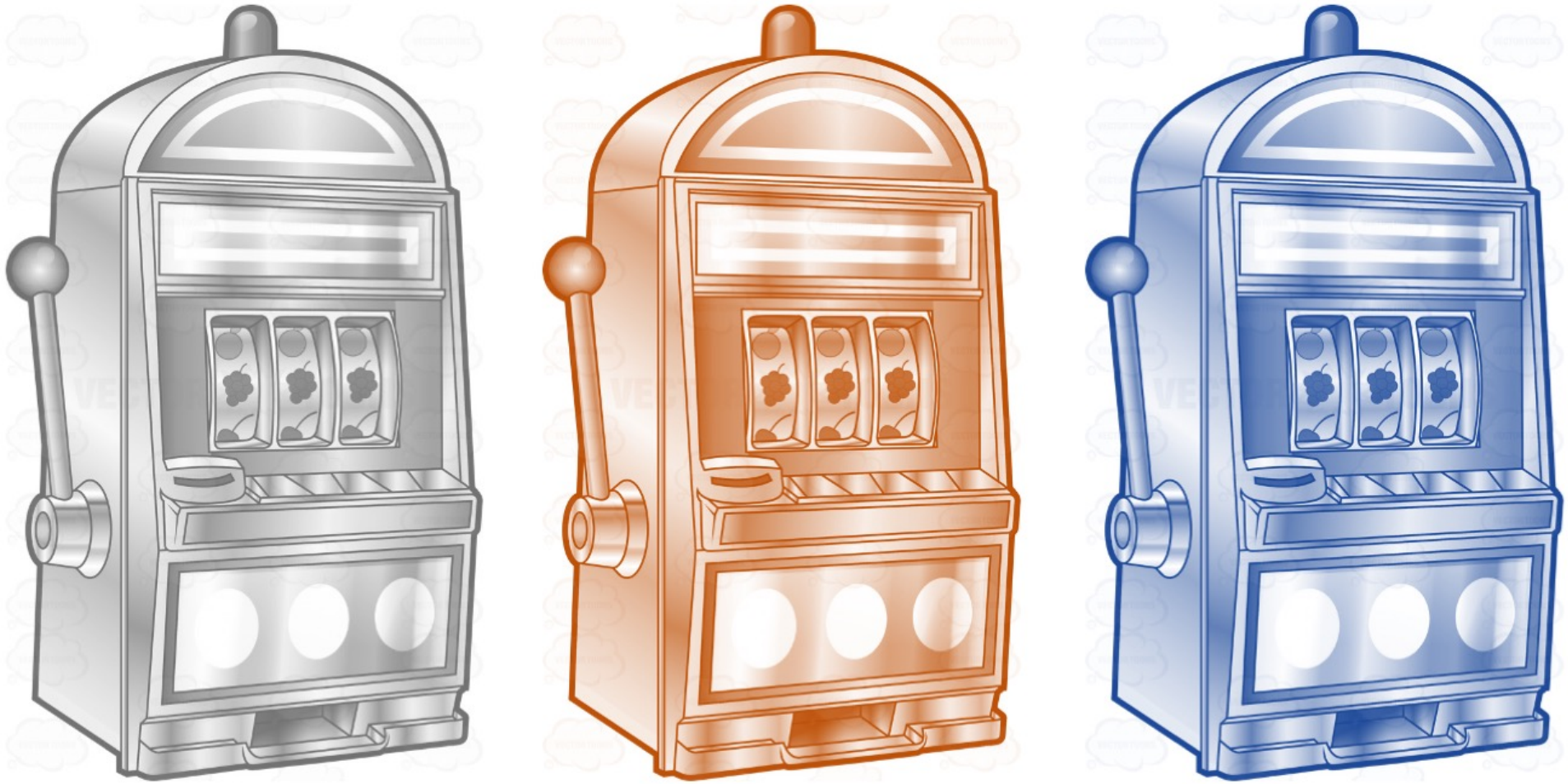
One-armed Bandit Problem

One-armed Bandit Process

Pull #	Response	Believed Probability of Jackpot
1	WIN (1.0)	1.0
2	LOSS (0.0)	0.5
3	LOSS (0.0)	0.33…
4	LOSS (0.0)	0.25
…	…	…
N.	LOSS (0.0)	0.1

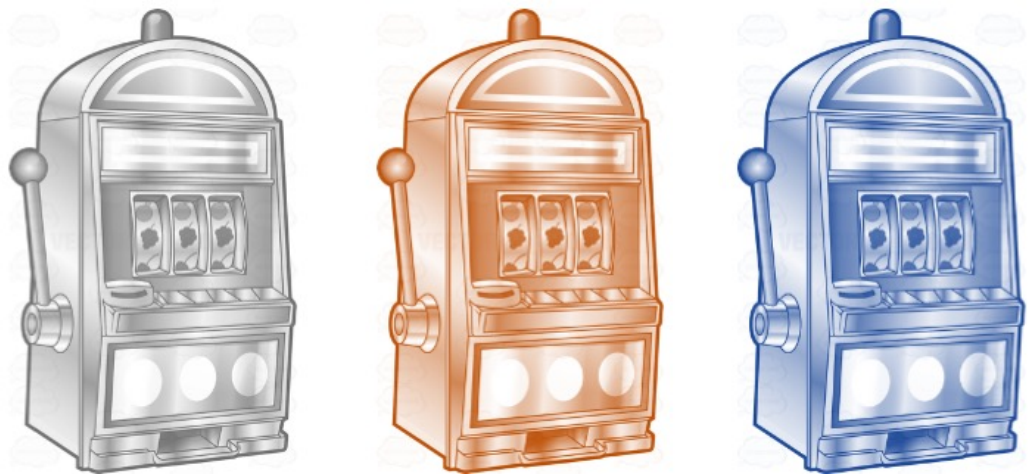
Unknown True Probability

Multi-armed Bandit Problem



Now what?

- We can’t just keep playing one bandit to figure out its potential for reward (money)
- Want to maximize reward across all bandits
- We need to trade off making money with current knowledge and gaining knowledge
  - *Exploitation vs. Exploration*



Just Exploitation

Greedy Strategy where we always pick the machine we think will give us the best reward (reminder: machines give rewards according to a probability)

Machine	Response	Belief (B,O,G)
Gray	LOSS	(-,-,0.0)
Orange	WIN	(0,1,0)
Blue	LOSS	(0,1,0)
Orange	LOSS	(0,0.5,0)
…	…	…

50%

25%

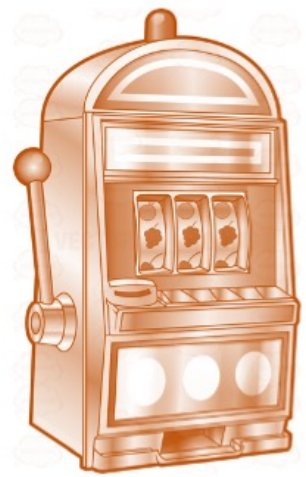
10%

Just Exploration

Always just pick a random machine, no matter what we believe.

Machine	Response	Belief (B,O,G)
Gray	LOSS	(-,-,0.0)
Orange	WIN	(0,1,0)
Blue	LOSS	(0,1,0)
Gray	WIN	(0.5,0.5,0)
…	…	…

50%



25%

10%

Epsilon (ϵ) Greedy
(ϵ-Greedy)

(ϵ=0.1)

- 90% of the time try the best machine according to our current beliefs
- 10% of the time take random action

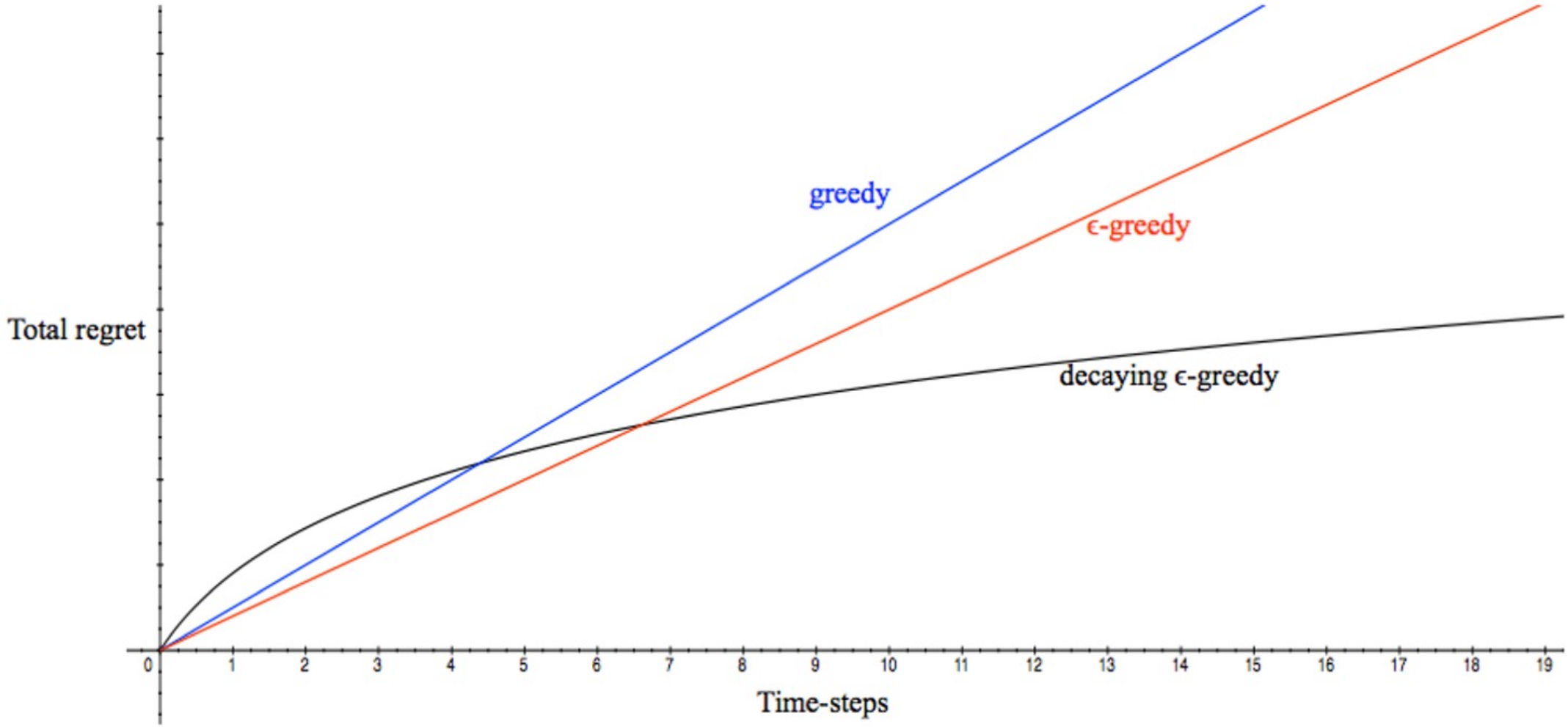
Epsilon (ϵ) Greedy
(ϵ-Greedy)

(ϵ=0.1)

- 90% of the time try the best machine according to our current beliefs
- 10% of the time take random action

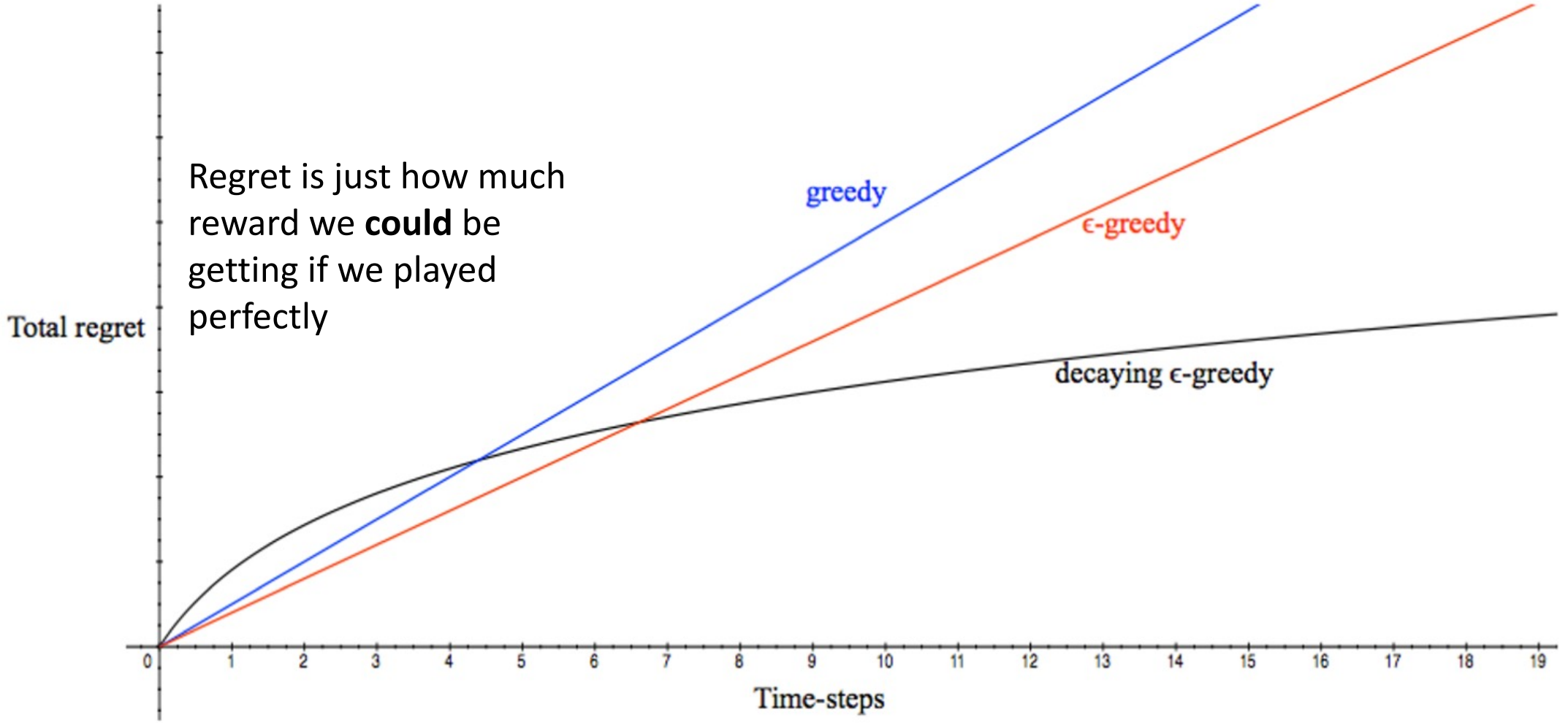
Now we can “escape” from early greedy mistakes!

Different Strategies and Regret



Different Strategies and Regret

Regret is just how much reward we **could** be getting if we played perfectly



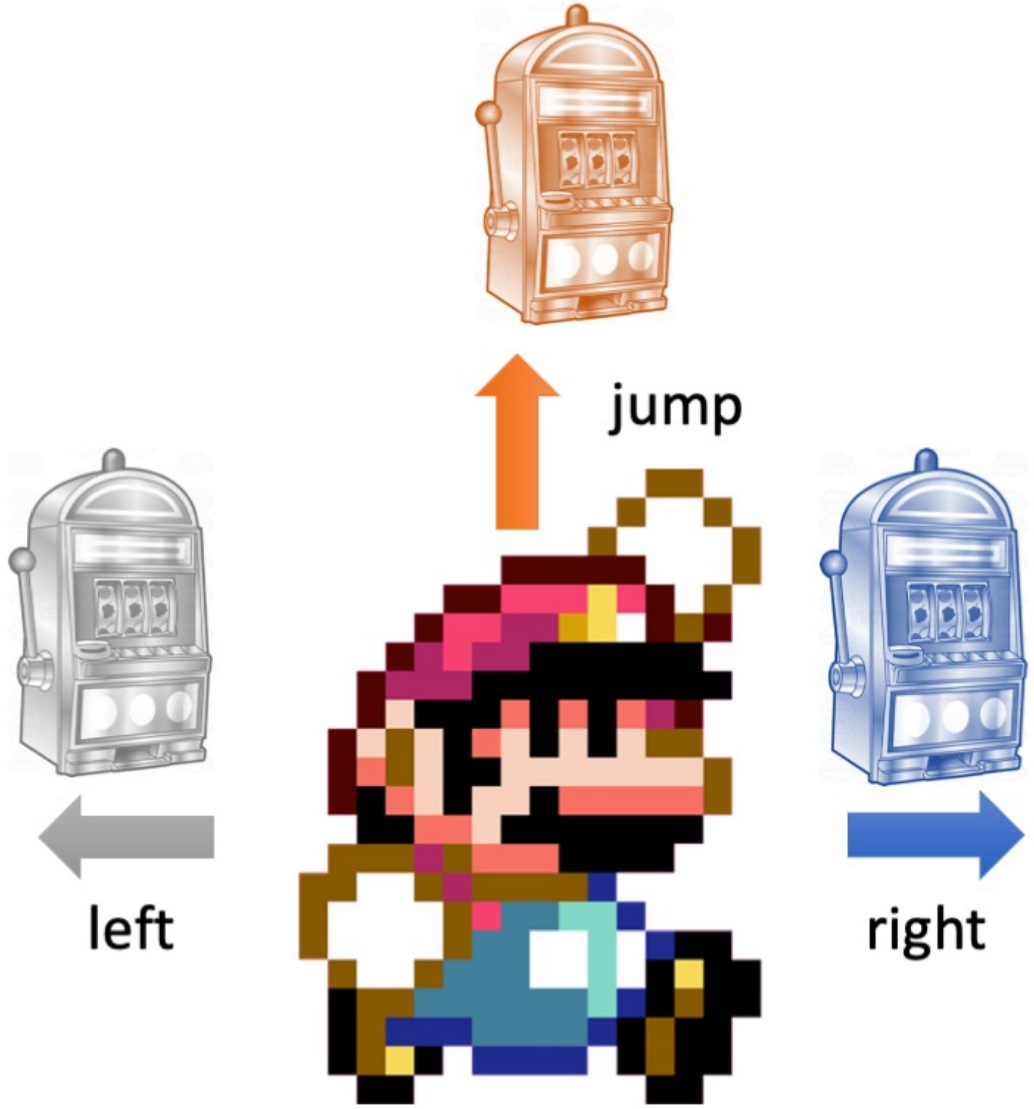
Decaying ϵ-Greedy

- Drop ϵ lower and lower according to some schedule
- Now: logarithmic asymptotic regret

- Downside: we need to have enough domain knowledge to come up with a good schedule beforehand

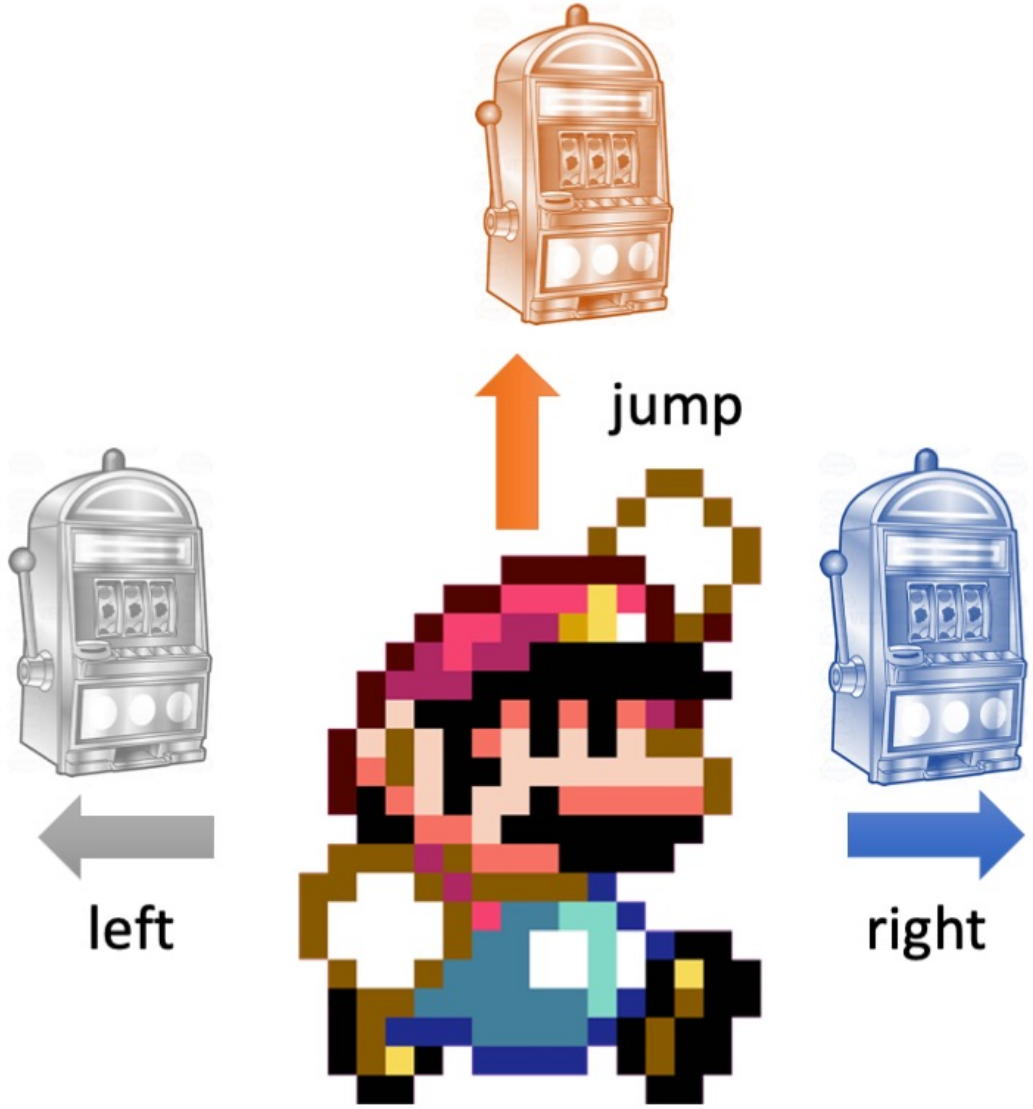
What does this have to do with games?

- We can map this to any automated game playing task.
- Each bandit is now an action we could take at a particular time/place.



What does this have to do with games?

- We can map this to any automated game playing task.
- Each bandit is now an action we could take at a particular time/place.
- Reward: Game Score



But these values will probably differ based on context…

Value of going left versus right will be different if an enemy is to the left or the right.

State

- Multi-armed bandit problem is a "single state" or "stateless" **Markov Decision Process**

Andrey Markov

State

- Multi-armed bandit problem is a "single state" or "stateless" **Markov Decision Process**

- But in games (and most cases we care about) there is more than one state

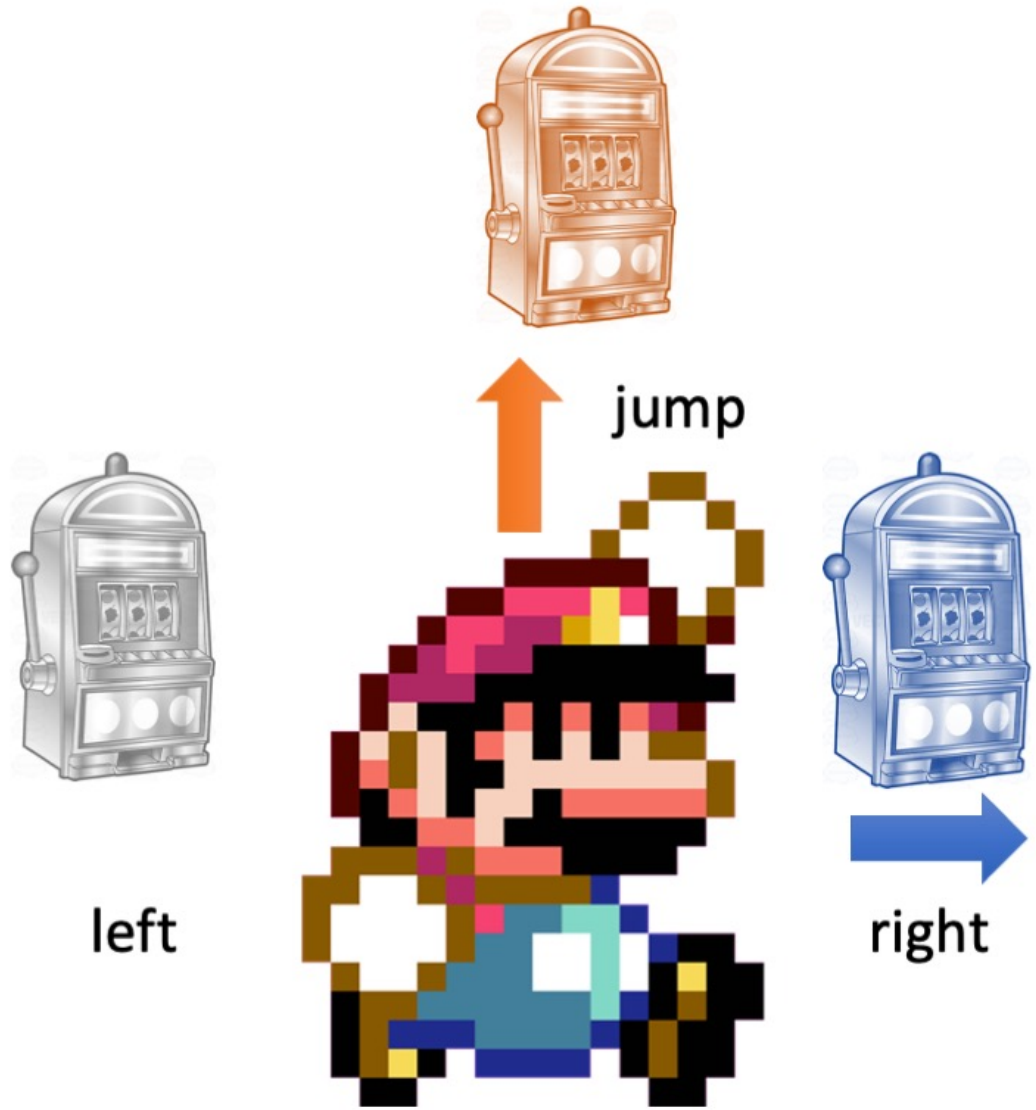
Andrey Markov

Markov Decision Process

- *S*: finite set of states (Markovian state: state has all info we need to make the optimal decision)
- *A*: finite set of actions
- *T(s,a,s1)*: Transition probability that action a takes us from state s to state s1
- *R(s,s1)*: Reward for transitioning from state s to state s1
- γ*:* Discount factor ([0-1]). Demonstrates the difference in importance between future rewards and present rewards

High-level Idea

If the multi-armed bandit problem was a single state MDP, we can think of learning a strategy to play a game as solving this problem for *every state of the game*



State Representation Examples

What pixels are present on the screen at this time?

THIS IS A MASSIVE STATE SPACE.

32x32x255x255x255 =

16,979,328,000 states

State Representation Examples

Grid Representations

- Segment all locations to tiles
- All enemies represented individually? Or just as “enemy”
- How much of the level to include in the state? Just screen? Smaller? Larger?

State = 9*15*14 = 1890 states

State Representation Examples

List of facts

-enemy to right

-powerup above

-fire mario

-on ground

-etc…

State space: variable

State Representation Tradeoffs

- More complex state representations ensure that an agent has all info needed
- Less complex state representations speed up training (more states “look” the same), but make it too simple and no longer Markovian.

- Goal: find the middle ground. Simplest state representation that still allows for optimal performance

PQ1 https://tinyurl.com/guz-pq32 https://forms.gle/c3TfYiHKu9F5ephi7

What state representations would you use for each of the following? Why?

- Tic-Tac-Toe
- A simple platformer
- A real time strategy game (Civilization/Starcraft)

My answers

- Tic-Tac-Toe: The Gameboard
- Simple platformer: grid of screen width and height, with values for collideable elements, breakable elements, empty spaces, and enemy types (flying, etc)
- RTS: World map + list of facts about currently running commands (building troops, moving troops, upgrading building, etc)

How do we know what to do in each state?

- A policy: Mapping of states to the action to take in that state
- $\pi(s) \rightarrow a$
- The purpose of a **Reinforcement Learning algorithm** is to approximate an optimal policy.

Example

S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red

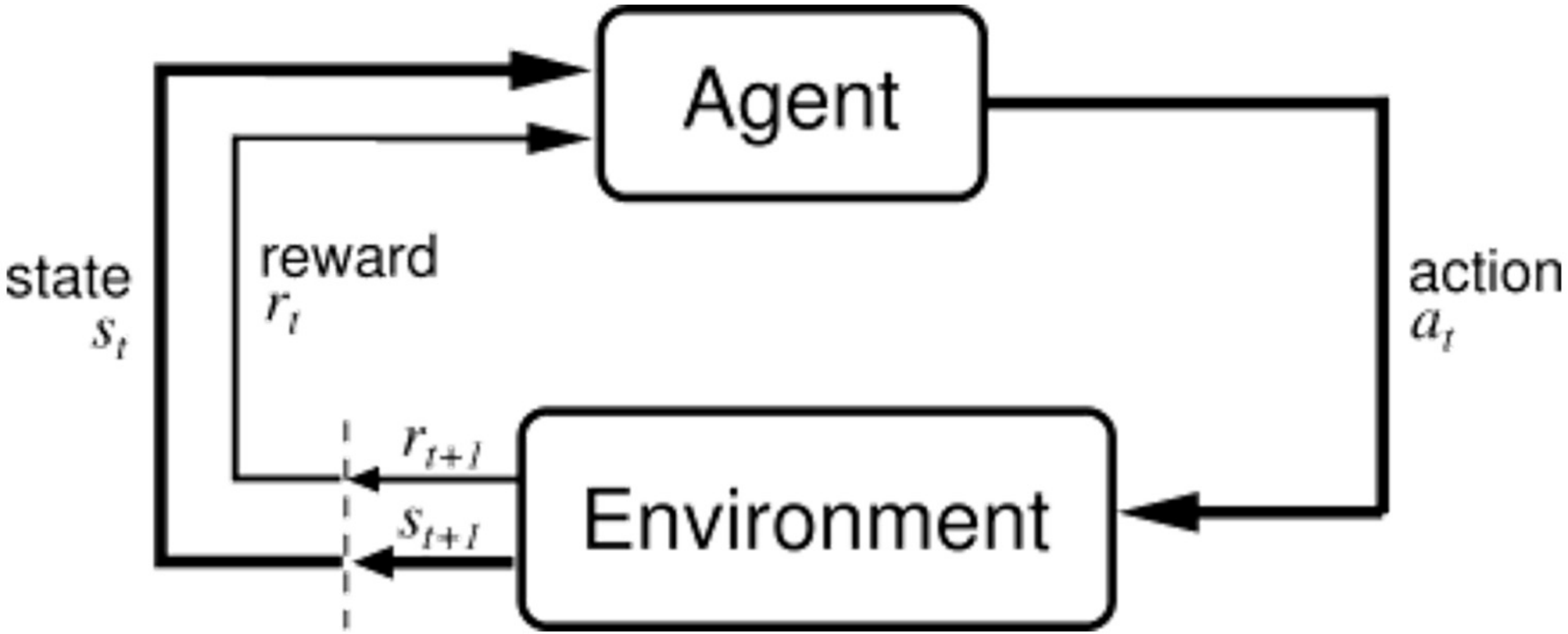
π_1

Up	Left		Down
Up	Left	Left	Down
Up	Left	Down	Down
Up or Right	Right	Right	Right

π_2

Down	Down		Up
Right	Right	Right	Up
Right	Right	Up	Up
Up or Right	Right	Up	Up

So how do we learn the best policy?
Lots of RL approaches to try to do this



Q-learning algorithm (Most general RL approach)

Assign Q[S,A] arbitrarily

observe current state s

repeat

select and carry out an action a
observe reward r and state s'
$Q[s,a] \leftarrow Q[s,a] + \alpha*(r+\gamma*\max_{a'}Q[s',a']-Q[s,a])$

until termination

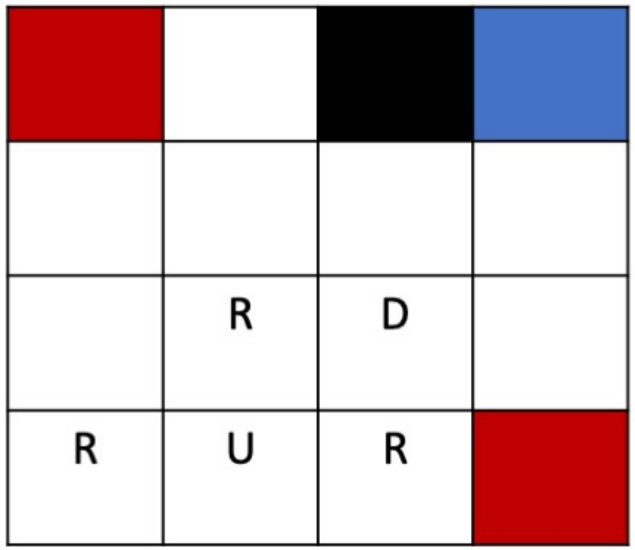
Example

S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
$a = 0.1$

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0		
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	

Rollout 1



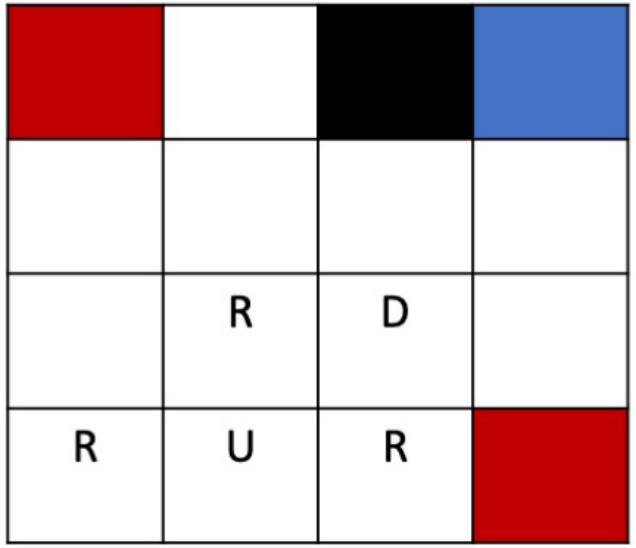
S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
a = 0.1

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0		
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	

Update 1

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

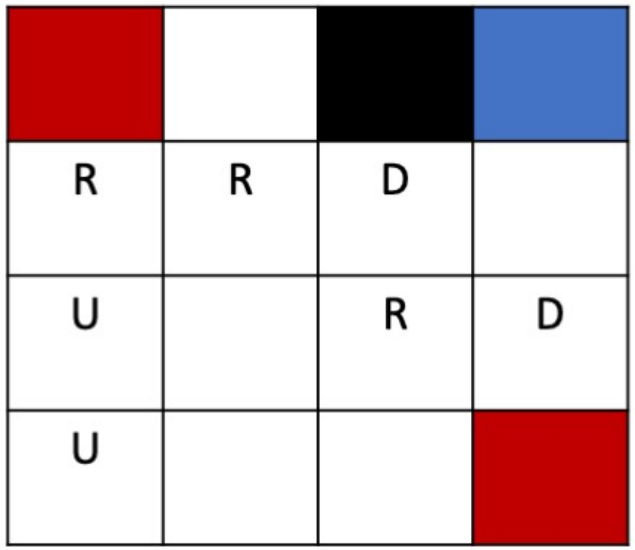


S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
$a = 0.1$

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0	**NOTE: the values here are wrong, but the basic process is what's important.**	
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: -0.01 Down: 0 Left: 0	Up: 0 Right: 0 Down: -0.1 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: 0 Right: -0.0001 Down: 0 Left: 0	Up: -0.001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -1 Down: 0 Left: 0	

Rollout 2



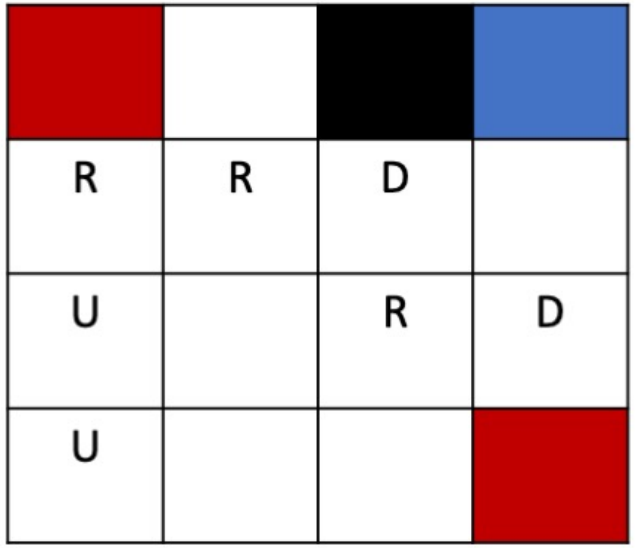
S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
a = 0.1

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0		
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: 0 Right: 0 Down: 0 Left: 0	Up: 0 Right: -0.01 Down: 0 Left: 0	Up: 0 Right: 0 Down: -0.1 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: 0 Right: -0.0001 Down: 0 Left: 0	Up: -0.001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -1 Down: 0 Left: 0	

Update 2

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

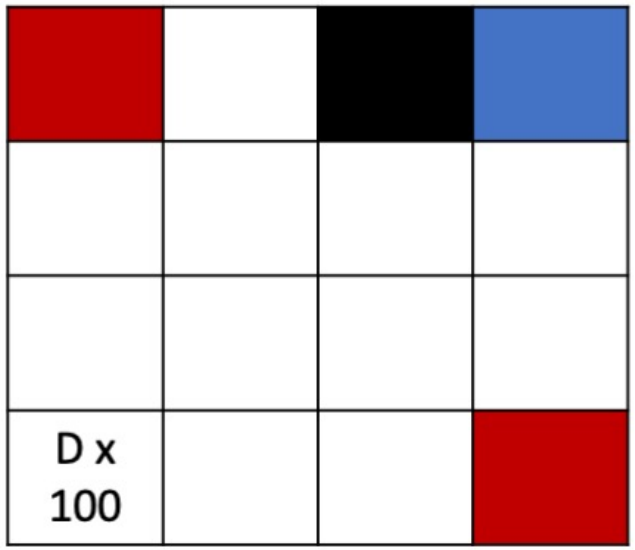


S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
$a = 0.1$

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0		
Up: 0 Right: **-0.0001** Down: 0 Left: 0	Up: 0 Right: **-0.001** Down: 0 Left: 0	Up: 0 Right: 0 Down: **-0.01** Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: **-0.00001** Right: 0 Down: 0 Left: 0	Up: 0 Right: -0.01 Down: 0 Left: 0	Up: 0 Right: **-0.1** Down: -0.1 Left: 0	Up: 0 Right: 0 Down: **-1** Left: 0
Up: **-0.000001** Right: -0.0001 Down: 0 Left: 0	Up: -0.001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -1 Down: 0 Left: 0	

Rollout 3

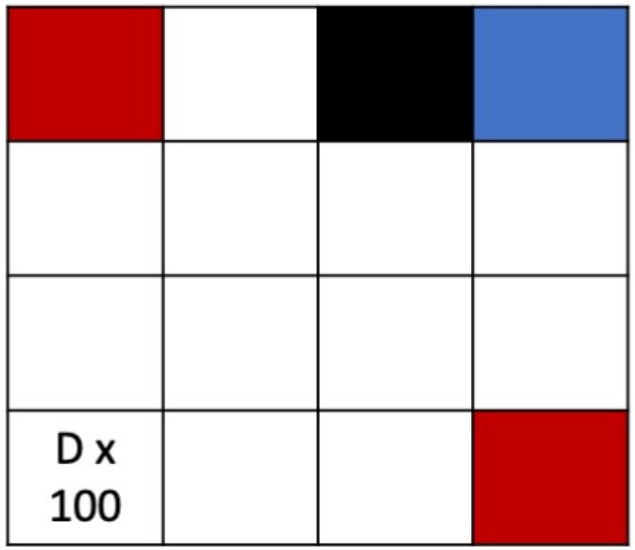


S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
a = 0.1

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0		
Up: 0 Right: -0.0001 Down: 0 Left: 0	Up: 0 Right: -0.001 Down: 0 Left: 0	Up: 0 Right: 0 Down: -0.01 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: -0.00001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -0.01 Down: 0 Left: 0	Up: 0 Right: -0.1 Down: -0.1 Left: 0	Up: 0 Right: 0 Down: -1 Left: 0
Up: -0.000001 Right: -0.0001 Down: 0 Left: 0	Up: -0.001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -1 Down: 0 Left: 0	

Update 3

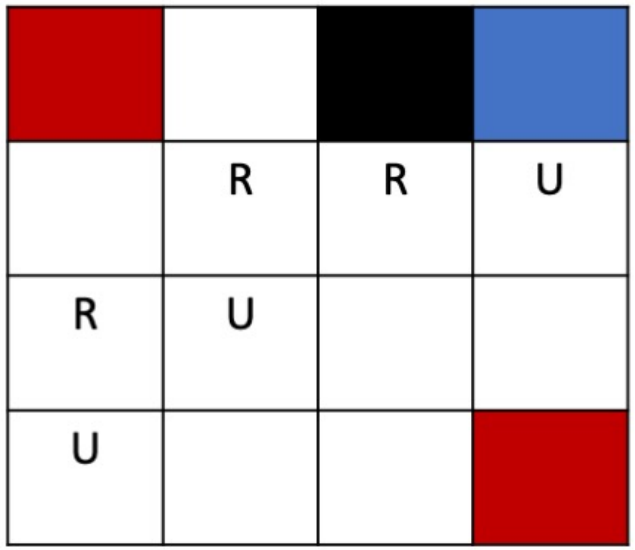


S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
a = 0.1

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0		
Up: 0 Right: -0.0001 Down: 0 Left: 0	Up: 0 Right: -0.001 Down: 0 Left: 0	Up: 0 Right: 0 Down: -0.01 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: -0.00001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -0.01 Down: 0 Left: 0	Up: 0 Right: -0.1 Down: -0.1 Left: 0	Up: 0 Right: 0 Down: -1 Left: 0
Up: -0.000001 Right: -0.0001 Down: **0** Left: 0	Up: -0.001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -1 Down: 0 Left: 0	

Rollout 4



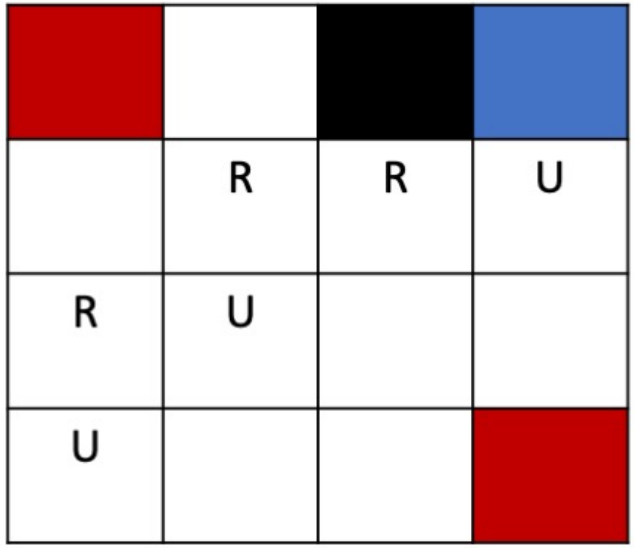
S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
a = 0.1

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0		
Up: 0 Right: -0.0001 Down: 0 Left: 0	Up: 0 Right: -0.001 Down: 0 Left: 0	Up: 0 Right: 0 Down: -0.01 Left: 0	Up: 0 Right: 0 Down: 0 Left: 0
Up: -0.00001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -0.01 Down: 0 Left: 0	Up: 0 Right: -0.1 Down: -0.1 Left: 0	Up: 0 Right: 0 Down: -1 Left: 0
Up: -0.000001 Right: -0.0001 Down: 0 Left: 0	Up: -0.001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -1 Down: 0 Left: 0	

Update 4

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



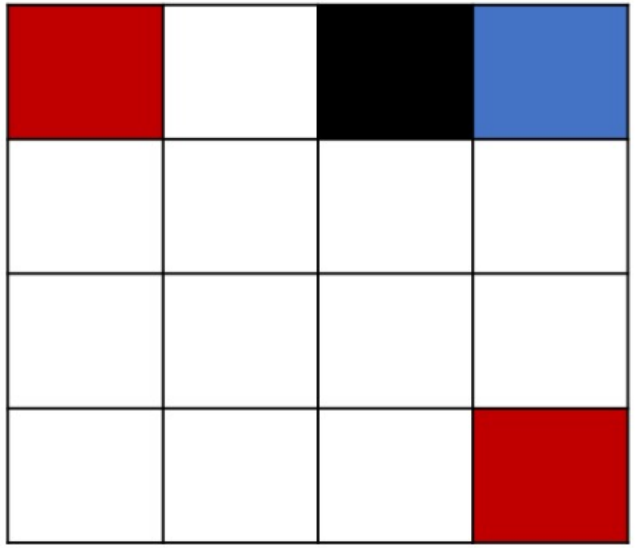
S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
$a = 0.1$

Q table (s, a)

	Up: 0 Right: 0 Down: 0 Left: 0		
Up: 0 Right: -0.0001 Down: 0 Left: 0	Up: 0 Right: **0.009** Down: 0 Left: 0	Up: 0 Right: **0.1** Down: -0.01 Left: 0	Up: **1** Right: 0 Down: 0 Left: 0
Up: -0.00001 Right: **0.0001** Down: 0 Left: 0	Up: **0.001** Right: -0.01 Down: 0 Left: 0	Up: 0 Right: -0.1 Down: -0.1 Left: 0	Up: 0 Right: 0 Down: -1 Left: 0
Up: **0.000009** Right: -0.0001 Down: 0 Left: 0	Up: -0.001 Right: 0 Down: 0 Left: 0	Up: 0 Right: -1 Down: 0 Left: 0	

Update n

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



S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
$a = 0.1$

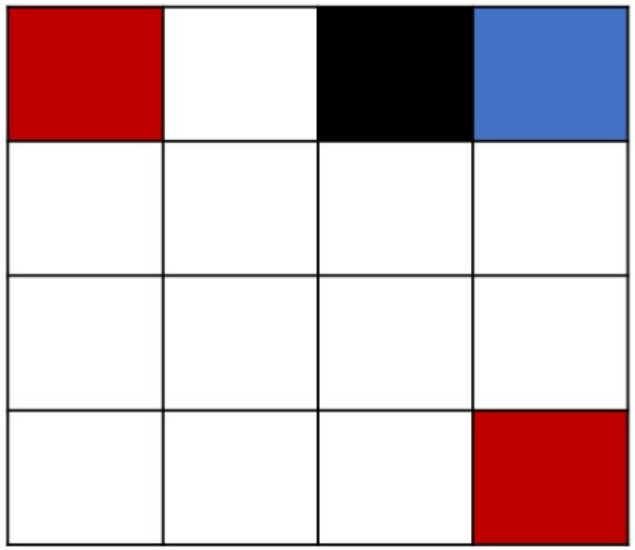
Q table (s, a)

	Up: 0 Right: 0 Down: 0.0001 Left: -1		
Up: -1 Right: 0.0009 Down: 0.0001 Left: 0	Up: 0.0001 Right: 0.009 Down: 0 Left: 0	Up: 0 Right: 0.1 Down: 0.00001 Left: 0	Up: 1 Right: 0.01 Down: 0.001 Left: 0.001
Up: -0.00001 Right: 0.0001 Down: 0.000001 Left: 0	Up: 0.009 Right: -0.01 Down: 0 Left: 0	Up: 0.001 Right: 0.001 Down: 0.00001 Left: 0.000001	Up: 0.01 Right: 0.001 Down: -1 Left: 0.0001
Up: 0.000009 Right: 0.000009 Down: 0 Left: 0	Up: 0.00001 Right: 0.00001 Down: 0 Left: 0	Up: 0.0001 Right: -1 Down: 0 Left: 0	

Get π by finding action a for each state s that maximizes Q(s,a)

Update n

Q table (s, a)



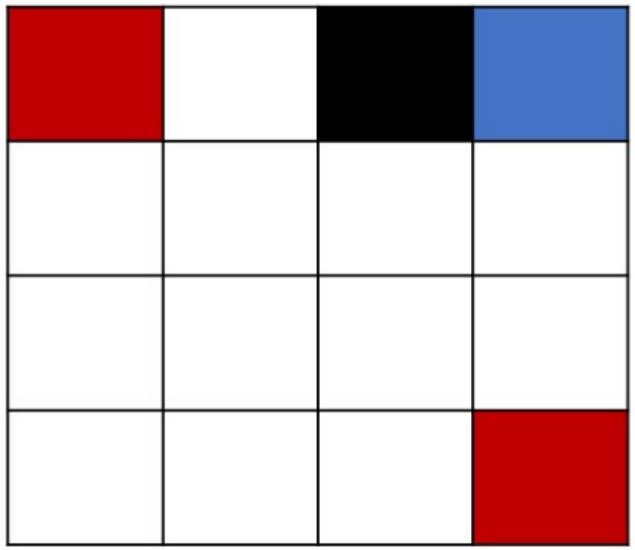
S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
γ = 1
a = 0.1

	Up: 0 Right: 0 Down: 0.0001 Left: -1		
Up: -1 Right: 0.0009 Down: 0.0001 Left: 0	Up: 0.0001 Right: 0.009 Down: 0 Left: 0	Up: 0 Right: 0.1 Down: 0.00001 Left: 0	Up: 1 Right: 0.01 Down: 0.001 Left: 0.001
Up: -0.00001 Right: 0.0001 Down: 0.000001 Left: 0	Up: 0.009 Right: -0.01 Down: 0 Left: 0	Up: 0.001 Right: 0.001 Down: 0.00001 Left: 0.000001	Up: 0.01 Right: 0.001 Down: -1 Left: 0.0001
Up: 0.000009 Right: 0.000009 Down: 0 Left: 0	Up: 0.00001 Right: 0.00001 Down: 0 Left: 0	Up: 0.0001 Right: -1 Down: 0 Left: 0	

Get π by finding action a for each state s that maximizes Q(s,a)

Update n

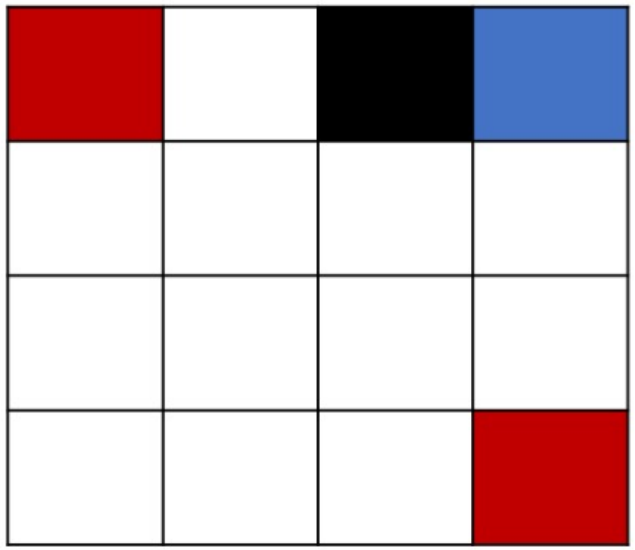
Q table (s, a)



S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
γ = 1
a = 0.1

	Up: 0 Right: 0 **Down: 0.0001** Left: -1		
Up: -1 **Right: 0.0009** Down: 0.0001 Left: 0.00001	Up: 0.0001 **Right: 0.009** Down: 0 Left: 0	Up: 0 **Right: 0.1** Down: 0.00001 Left: 0	**Up: 1** Right: 0.01 Down: 0.001 Left: 0.001
Up: -0.00001 **Right: 0.0001** Down: 0.000001 Left: 0.000001	Up: 0.0001 **Right: 0.0001** Down: 0 Left: 0	Up: 0.001 **Right: 0.001** Down: 0.00001 Left: 0.000001	**Up: 0.01** Right: 0.001 Down: -1 Left: 0.0001
Up: 0.000009 Right: 0.000009 Down: 0.0… Left: 0.0….	**Up: 0.00001** Right: 0.00001 Down: 0 Left: 0	Up: **0.0001** Right: -1 Down: 0.00001 Left: 0.00001	

Update n



S: Each Cell is a State
A: up, down, left, right
T: Actions always succeed
Reward: +1 for blue, -1 for red
$\gamma = 1$
$a = 0.1$

Final π

	Down		
Right	**Right**	**Right**	**Up**
Right	**Right**	**Right**	**Up**
Up	**Up**	**Up**	

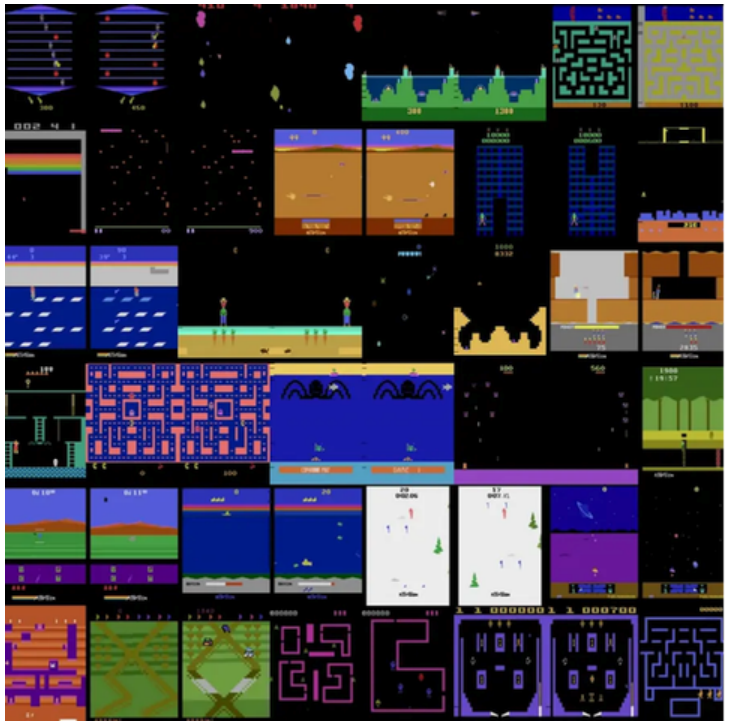
Q Learning

Pros
- We don’t need to have a forward model
- Means we can go into any environment blind

Cons
- Takes up tons of memory depending on |S|*|A|
- Takes a very long time to converge, depending on |S|*|A|

Imagine the Q table for a game like…

Go (AlphaGo, 2015)



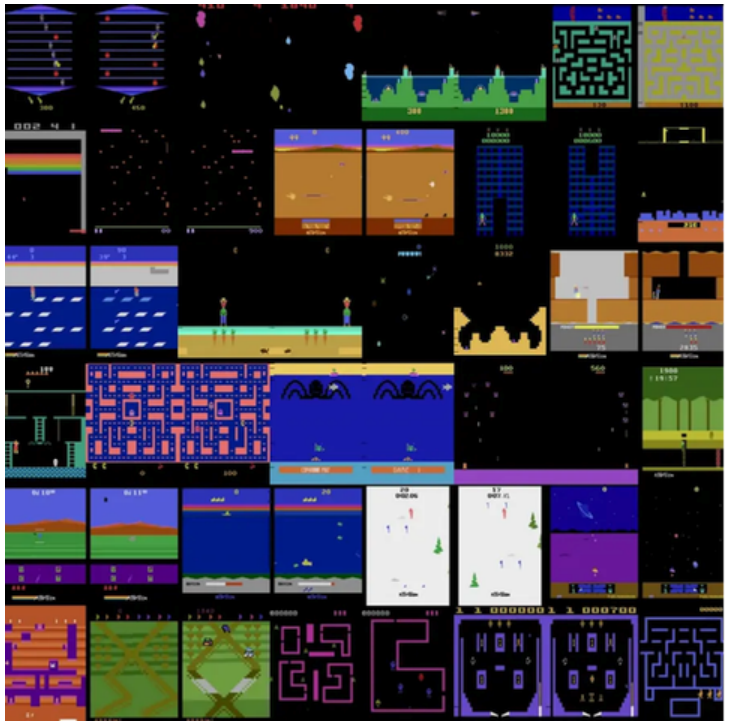
Atari (Agent57, 2020)

Starcraft 2 (Agent57, 2019)

DOTA2 (Open AI Five, 2019)

Imagine the Q table for a game like…

Go (AlphaGo, 2015)

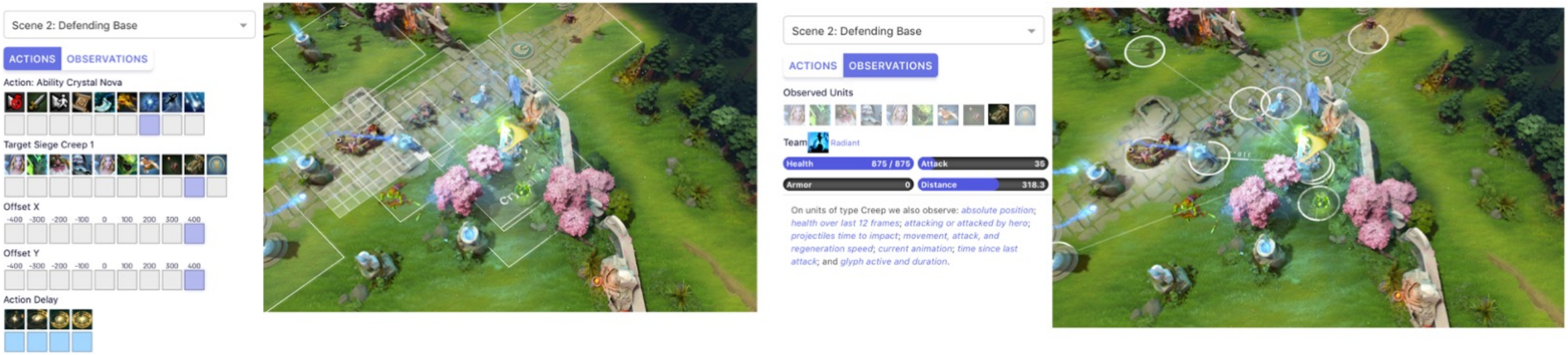


Atari (Agent57, **2020**)

Starcraft 2 (Agent57, **2019**)

DOTA2 (Open AI Five, **2019**)

Not using the actual screen state…



https://openai.com/blog/openai-five/

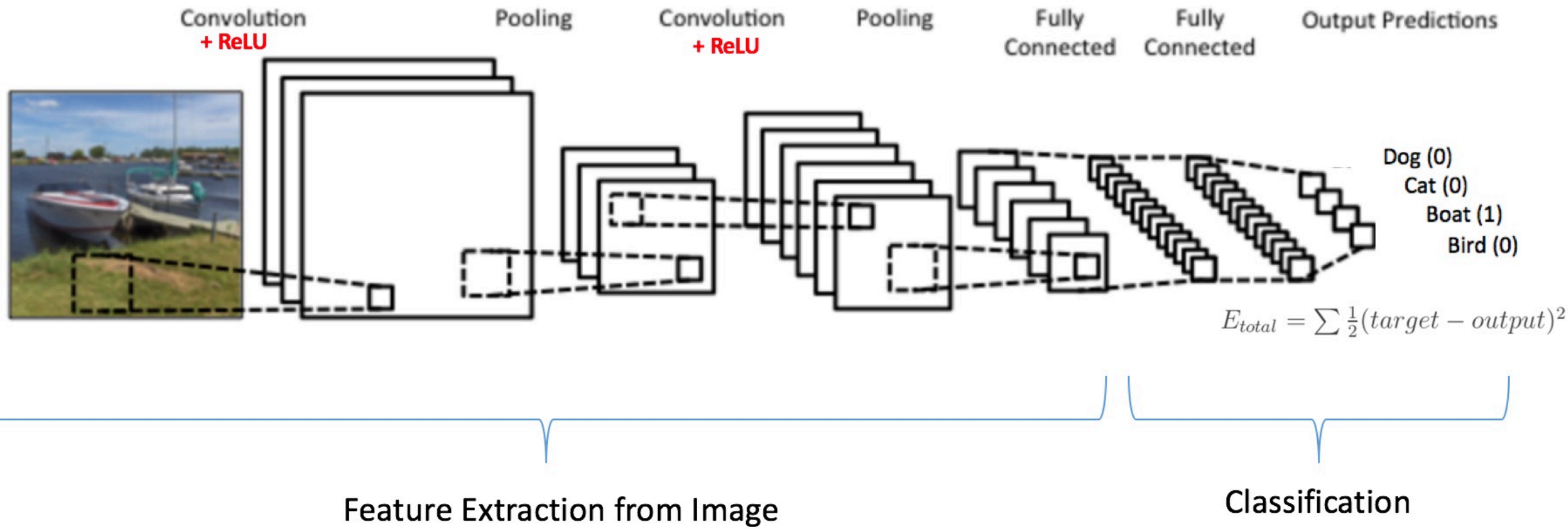
But all of these have 1 major variation over traditional “table-based” (tabular) RL

They used a Deep Neural Network (DNN) as the Q table!

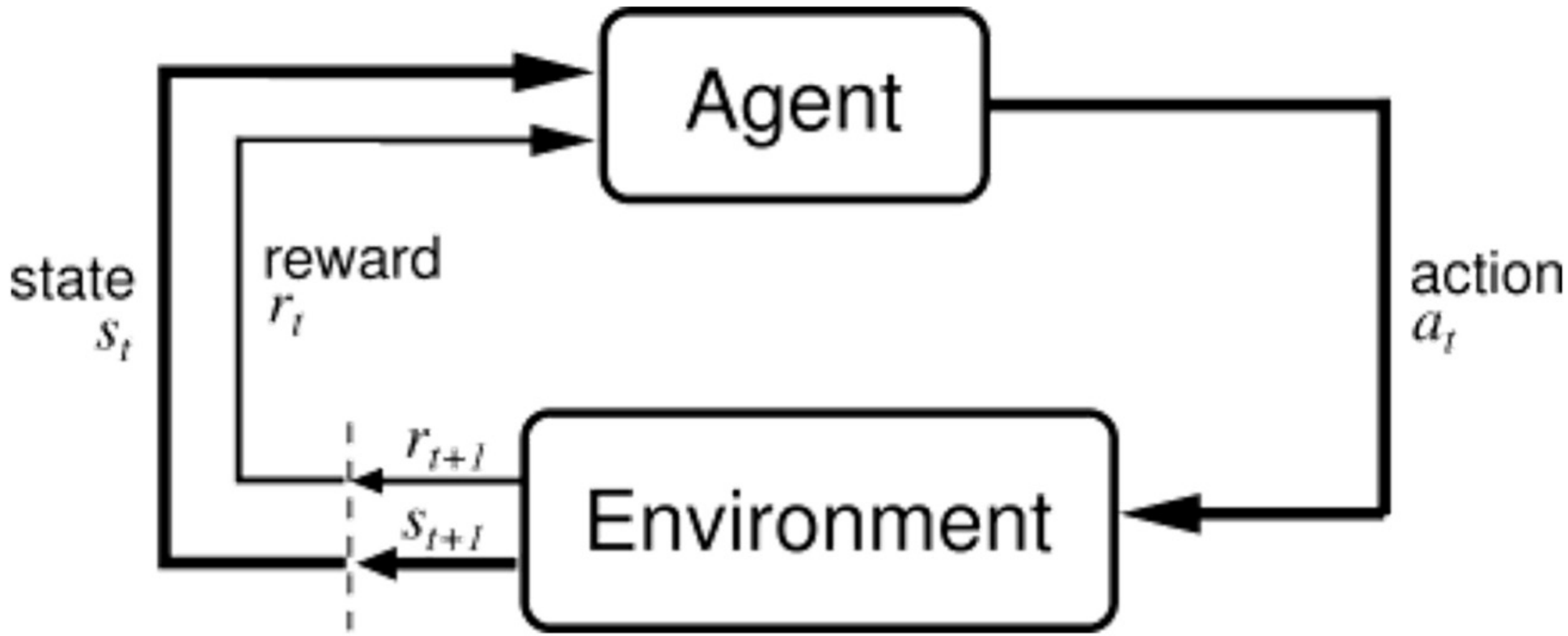
Intuition: Some States Are Similar

- What if we could learn what states were similar, such that we could know to use similar actions in similar states.
- We have infinite training data, since we're using an environment...
- What approach works well with large amounts of training data…

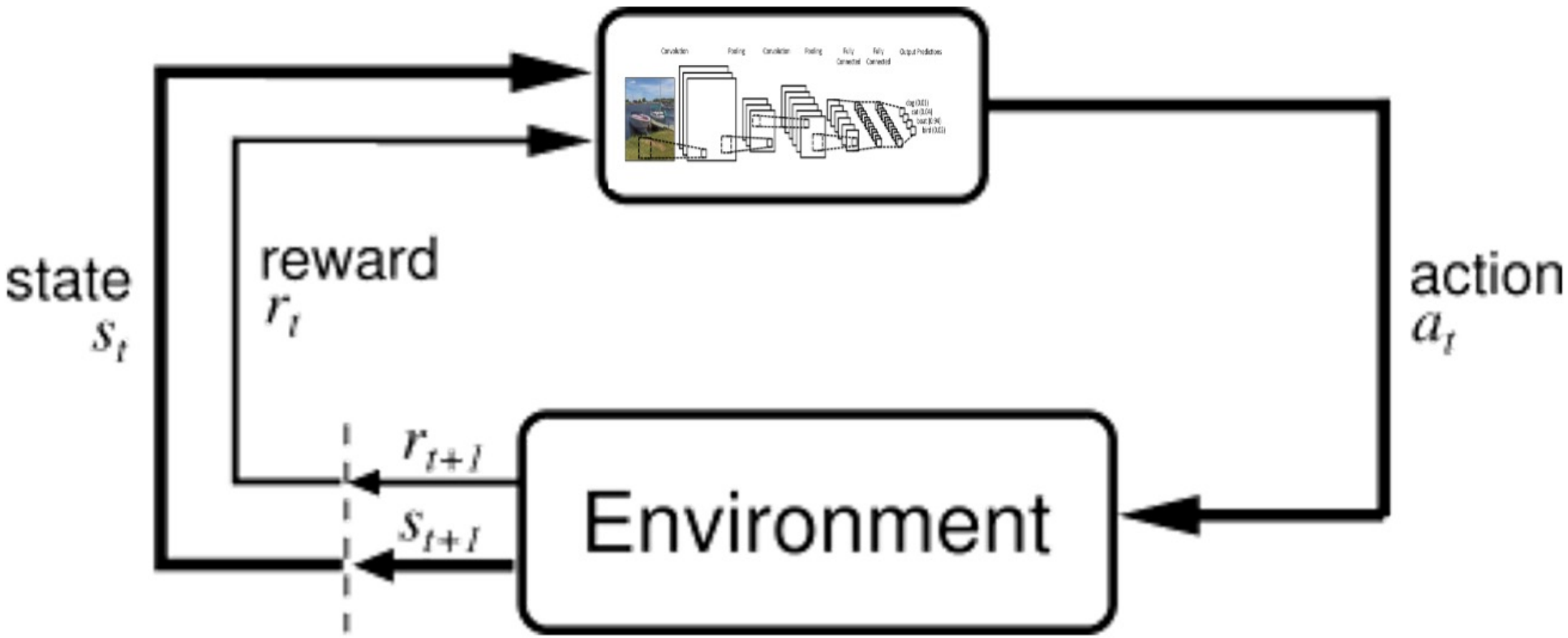
Intuition: Some States Are Similar



So what if we…



So what if we…



Deep Q-Learning

- Uses a deep neural network as the Q table, training it to correctly predict the value of state and action pairs.
- Led to huge improvements in automatic game playing and robot control tasks.
- Still not going to be used in a game anytime soon.

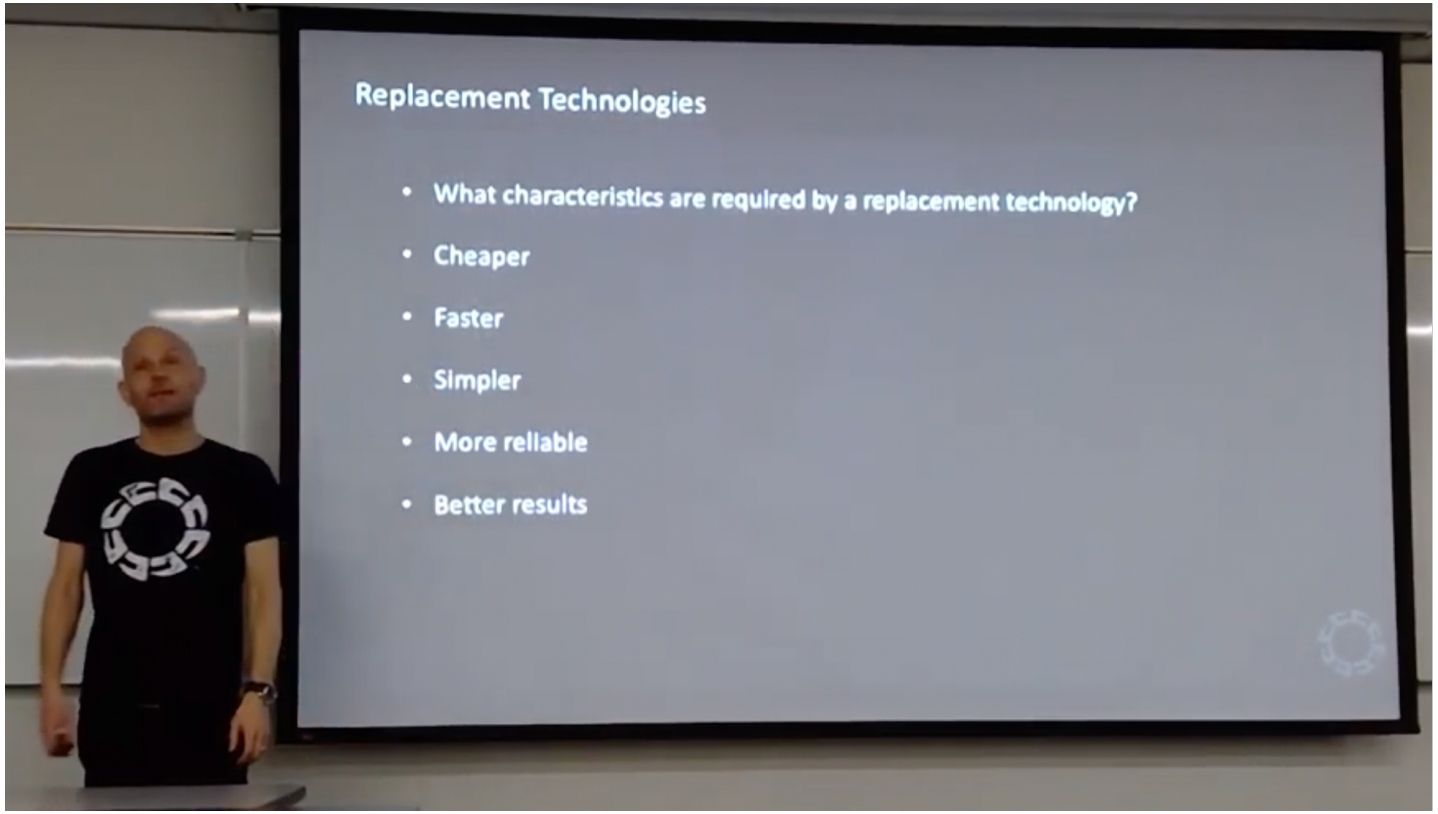
But why not? This sounds great!

- Incredibly Time Consuming: OpenAI Five trained for longer than human civilization has existed
- Resource Intensive: See above + Knowledge of how to apply these approaches
- Quirky: AI doesn’t play like people expect (OpenAI Five spammed useless items constantly)
- Brittle: AI will break if it sees something it never saw before.

But why not? This sounds great!

- Incredibly Time Consuming: OpenAI Five trained for longer than human civilization has existed
- Resource Intensive: See above + Knowledge of how to apply these approaches
- Quirky: AI doesn’t play like people expect (OpenAI Five spammed useless items constantly)
- Brittle: AI will break if it sees something it never saw before.
- Fun?: Players want fun enemies, not optimal ones.

It’s been tried!



Jesse Cluff, EA AI Lead for Gears of War (MARLO 2018)

https://youtu.be/5H_GEIb60_I

Game Development Loop

- Create Prototype with some Prototype AI
- Play game
- Realize game needs tweaking
- Tweak game
- Tweak agents (maybe change a parameter or two)
- Play game
- Realize game needs tweaking
- Etc.

ML Game Development Loop?

- Create Entirety of Game besides AI Agents
- Train AI Agents with ML to play Game (infinite compute)
- Play game with Trained AI Agents
- Realize game needs tweaking
- Tweak game
- Train AI Agents with ML to play Game FROM SCRATCH
- Play game with Trained AI Agents
- Realize game needs tweaking

So where could Deep RL fit in games?

- Playtesting new content for an existing game!
  - Hearthstone New Card Balance Checking https://youtu.be/t5MUuCmm81k?t=831
  - Automated space testing (speculative, not yet in games) https://youtu.be/DKdQFajLfzk

If more time (how?) cover a Q-learning example on the board