



Bridging Reinforcement Learning and Creativity: Implementing Reinforcement Learning in Processing

Jieliang Luo	Sam Green
Media Arts & Technology	Computer Science
<u>jieliang@ucsb.edu</u>	<u>sam.green@cs.ucsb.edu</u>

University of California, Santa Barbara
Santa Barbara, CA, 93106

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the owner/author(s).

SA '18 Courses, December 04-07, 2018, Tokyo, Japan

ACM 978-1-4503-6026-5/18/12.

10.1145/3277644.3277796

Abstract

Artists are underrepresented in the reinforcement learning (RL) community due to the steep learning curve involved in in-depth understanding of RL algorithms. However, artists can play an important role in the RL community by defining innovative problems, designing creative environments, and creating novel applications. As a popular tool for artists to experiment with programming, Processing has been highly adapted by many artists as their entry point to programming. Given the popularity of Processing in the creative community, we use this tutorial as a steppingstone to bridge RL and creativity by introducing RL core concepts in Processing. The purpose of this workshop is twofold: 1) to attract more artists to the RL community by demonstrating RL demos in their familiar IDE; 2) to demystify RL problems by implementing them in a high-level language without any external libraries. Importantly, this tutorial is not about introducing a specific programming language, but will focus on how to analyze, frame, and solve RL problems.

Course Motivations

- Why we need artists in RL

For decades, media artists have played an essential role in advancing AI, exploring its full extent, and pushing the boundaries. Back to the 1970s, artists like Harold Cohen started to investigate the use of machines for autonomous drawings. Cohen's iconic project AARON, a computer program creating original artistic physical images, has been used as an artistic equivalent of the Turing Test. In the modern AI age, GANs and RNNs have been widely adapted for artistic expressions that were featured at major technical conferences, such as *I Touch You And You Touch Me* at SIGGRAPH Asia 2017, and *Magenta and deeplearn.js: Real-time Control of DeepGenerative Music Models in the Browser* at NIPS 2017. ECCV 2018 even opens a workshop called *Computer Vision for Fashion, Art and Design* and actively calls for artworks.

As a breakthrough technology in AI in 2015, deep reinforcement learning is an efficient class of sequential decision-making algorithms that have achieved remarkable success in a broad range of applications, such as image classification, robotic manipulation, and strategic games. The most well-known example of RL is AlphaGo, a computer program that plays the board game Go and outperforms top human Go players. Recently, Prof. Sergey Levine's lab from UC Berkeley used reinforcement learning to create interactive natural skeleton animations that has great potentials to be adapted by animation artists. Unlike other machine learning techniques, such as GANs or RNNs, in which media artists are actively engaged, RL has resulted in very few artworks. The current RL tools require an in-depth understanding of RL algorithms that can create a barrier for artists to explore RL. On the other hand, many intriguing RL problems, such as robotic architecture, cannot be solved without artists' engagement.

Regarding the collaboration between artists and computer scientists, one common misconception is that artists are only responsible for proposing creative ideas, and computer scientists are responsible for solving technical challenges. However, this solid distinction obscures the communication between the two groups as the languages might be entirely different. A better approach would be enabling artists to have some hands-on experience in solving some basic RL problems, which can inspire them to provide more constructive ideas. In addition, artistic/creative applications of RL can generate a larger social impact and attract more public attention to the RL community.

- Why Processing

As discussed in the previous section, artists need easier access to basic RL problems. Popular deep learning libraries, such as PyTorch and TensorFlow, are designed primarily for researching algorithms and optimizations. On the other hand, Processing, initiated by Ben Fry and Casey Reas in 2003, has become the steppingstone for many artists to practice programming in their artworks. Given the facts that Processing does not require any extra configurations to install and that its code can be compiled across platforms, it is convenient for artists to rapidly examine new technologies. Many

important libraries, such as OpenCV, OSC, and Box2D, all have their Processing versions, and many artworks based on those libraries have been featured at major conferences and art galleries. A small sample of examples includes: 1) *The Ghost in the Dandelion* at IEEE Vis 2017, 2) *Abstract Reality* at SIGGRAPH Asia 2017, and 3) *California Drought Impact v2* at CHI 2017.

Thus far, neither neural network libraries nor RL demos are available for Processing. Such technologies are, however, completely implementable by only using the native Processing language. For example, the tabular Q-learning algorithm can be implemented within 200 lines of code without any external libraries. Jieliang (Rodger) Luo, one of the presenters, has implemented the algorithm in Processing during OpenAI's one-day Hackathon. The work has been featured on the OpenAI's website¹.

¹<https://blog.openai.com/hackathon-follow-up/>

Course Overview

5 minutes: Welcome and Introduction

Welcome, overview of course & motivation for attending. Speaker introductions

10 minutes: What is Reinforcement Learning

A brief introduction to the concepts of reinforcement learning

10 minutes: Reinforcement Learning and Creativity

Discussion the importance to have reinforcement learning accessible to creative community

10 minutes: Why Processing

A brief introduction to Processing and the reasons of choosing it as the entrance of reinforcement learning for non-experts

35 minutes: Q-Learning & Policy Gradient

Explain two fundamental reinforcement learning algorithms

30 minutes: Implementing Tabular Q-Learning in Processing

Discuss how to create a reinforcement learning environment
Show how to implement tabular q-learning in Processing without external libraries

5 minutes: Next Steps, Conclusion, Questions & Answers

Discuss the future trends, wrap up, questions

Goals & Target Audience

The general purpose of this workshop is twofold: 1) to attract more artists to the RL community by demonstrating RL demos in their familiar IDE; 2) to demystify RL problems by implementing them in a high-level language without any external libraries. That said, the tutorial is NOT about introducing a specific programming language, but will focus on how to analyze, frame, and solve RL problems. The highlights covered in the tutorial, such as defining environments and implementing neural networks, are normally neglected because libraries like OpenAI Gym or PyTorch have taken care that. But those details are important not only for artists but also for non-experts to have a better understanding of RL problems. Therefore, the target audience of the tutorial are artists and non-experts who are interested to apply reinforcement learning in creative domain.

Presenters Information

Jieliang Luo is a researcher and media artists working with reinforcement learning, robotics, and visualization. Currently, he is a Ph.D. candidate and lecturer in Media Arts & Technology at UC Santa Barbara, where he teaches Reinforcement Learning, Data Visualization, and Intelligent Machine Vision. His research focuses on bridging reinforcement learning and creativity by exploring potential simulated and physical platforms for artists, designers, and architects to have easy access to reinforcement learning. He also has worked for Autodesk Robotics Lab as a robotic and machine learning researcher since June, 2018.

Sam Green is a Ph.D. candidate in Computer Science, at the University of California, Santa Barbara. His research is focused on numerical and architectural optimizations for reinforcement learning, as well as visualization diagnostics for CNN-based RL policies. Prior to attending UCSB, Sam was a Senior Member of Technical Staff, at Sandia National Laboratories, where he gained five years of experience contributing to and leading cryptographic hardware assessment R&D. Sam holds a master's in Applied Math and a bachelor's in Computer Science from the University of Central Arkansas.

In the Spring of 2018, the two presenters gave a graduate seminar course at UCSB in deep RL. The course covered Value Iteration, Deep Q-Learning, Policy Gradients, Actor-Critic, Deep Deterministic Policy Gradients, Temporal Difference Models, and World Models. Instruction was provided for both theory and coding implementation.

Bridging Reinforcement Learning and Creativity: Implementing Reinforcement Learning in Processing

Jieliang Luo Media Arts & Technology

Sam Green Computer Science

University of California, Santa Barbara

SIGGRAPH Asia 2018

Tokyo, Japan

December, 2018

Course Agenda

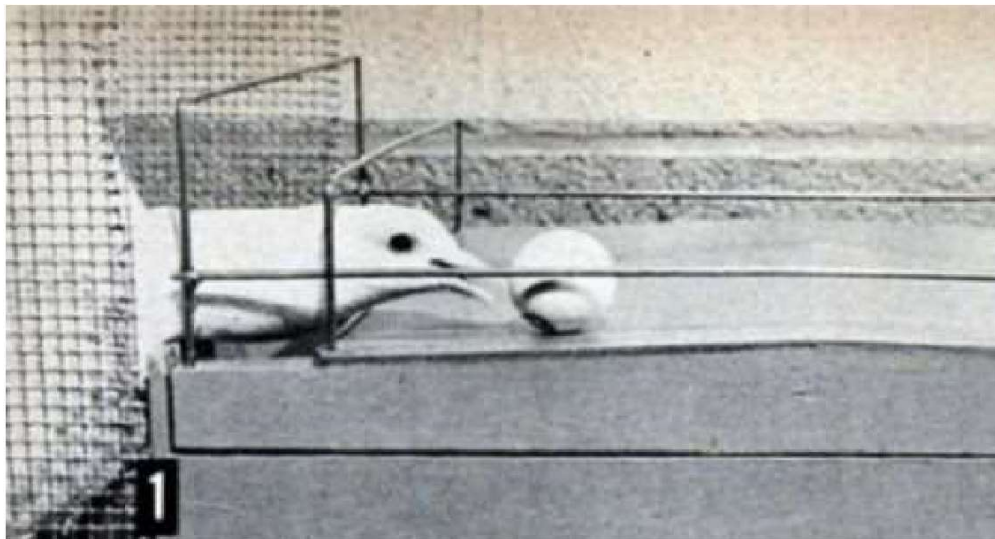
- What is reinforcement learning
- Reinforcement learning and creativity
- Why Processing
- Q-Learning & Policy Gradient
- Implementing Tabular Q-Learning in Processing
- Next Steps

Chapter I:

What is reinforcement learning?

Trial-and-error Learning

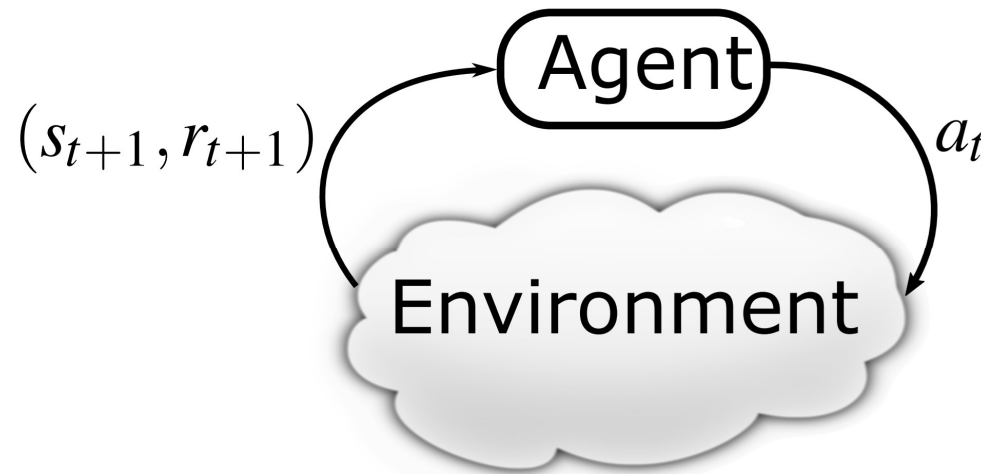
- B.F. Skinner, 20th century psychologist
- Also called *operant conditioning*
- Give rewards or punishments to encourage behavior



Pigeon learning ping pong

Credit: Psychology Pictures

Reinforcement Learning Paradigm



- **Agent** – Makes actions on an environment. Attempts to collect rewards
- **Environment** – Hosts the agent. Partially influenced by agent. Gives agent rewards

AlphaZero

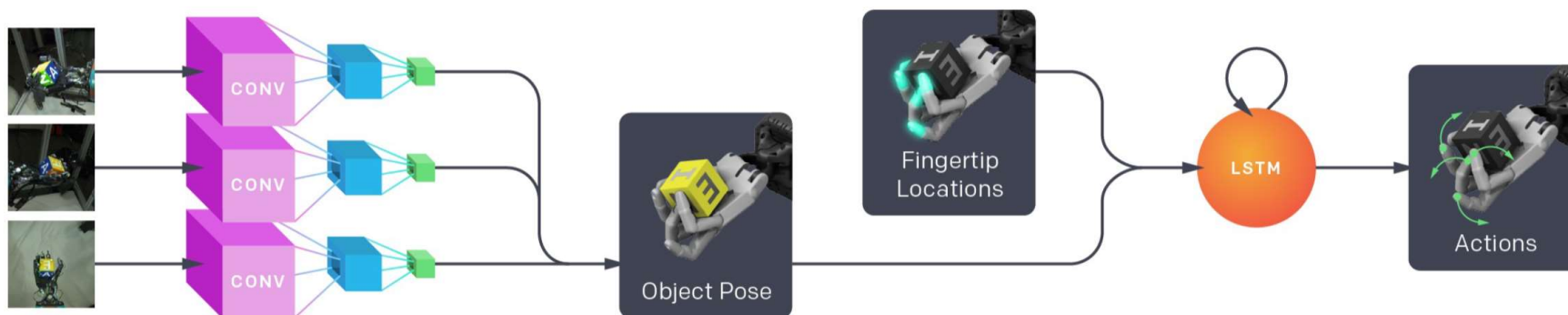
- Trained RL agent to play
 - Chess
 - Go
 - Shogi
- Used a single algorithm to achieve super human levels of performance
- State observation for Go includes
 - Last 8 board configurations for player 1 and 2
 - Current player's color



Credit: Wikipedia

Dactyl

- Used robotic hand and RL to achieve human-level dexterity
- Reward was cube position relative to desired
- Used three cameras and finger tip locations for state observation



Dactyl



Chapter II:

Reinforcement Learning + Creativity

Reinforcement Learning + Creativity



A TDK SA90 Type II Compact Cassette, developed by Philips

Credit: "A TDK SA90 Type II Compact Cassette" from Wikipedia (public domain)

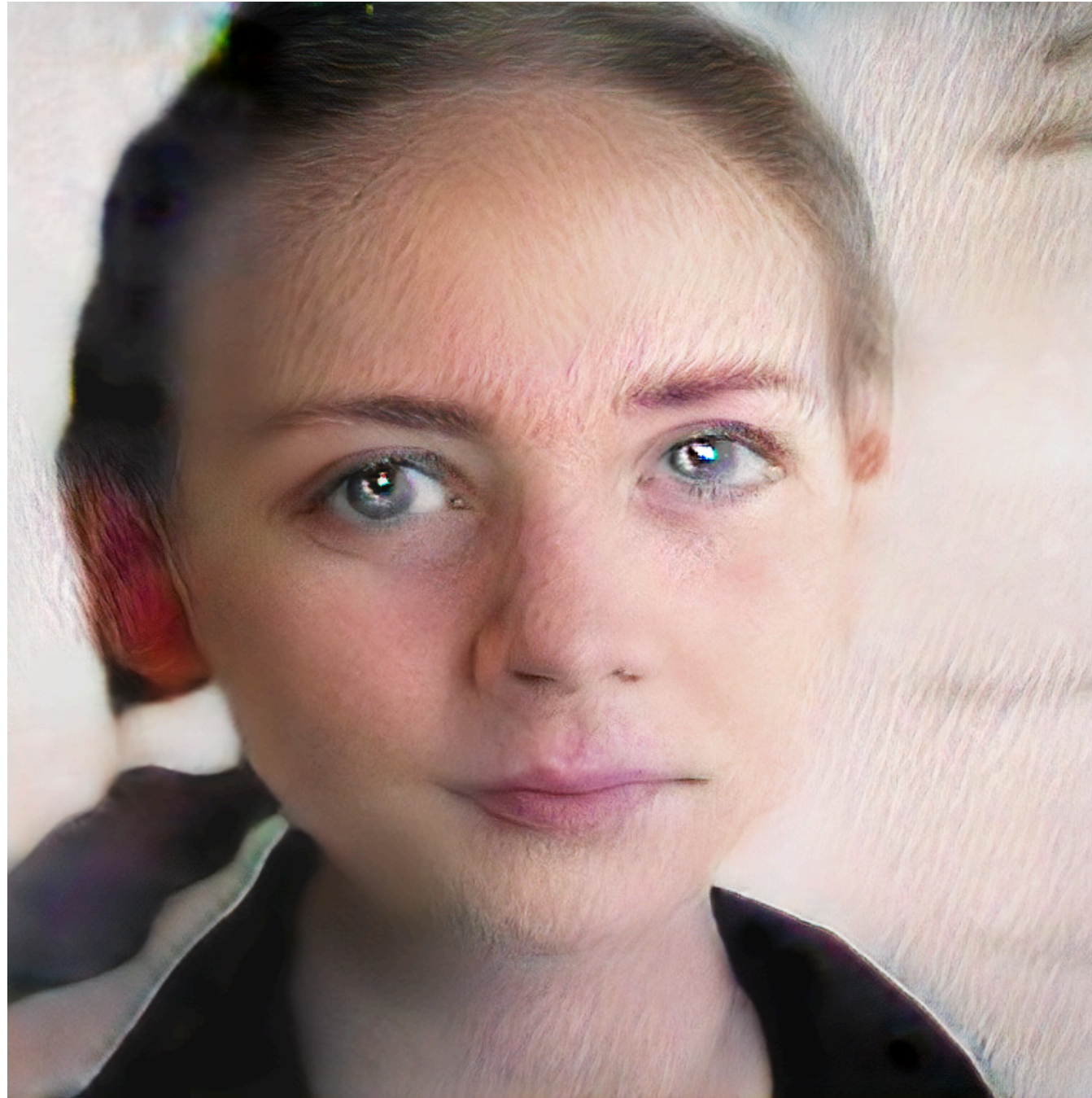
Reinforcement Learning + Creativity



An untitled AARON drawing

Credit: "An untitled AARON drawing" from Computer History Museum

Reinforcement Learning + Creativity

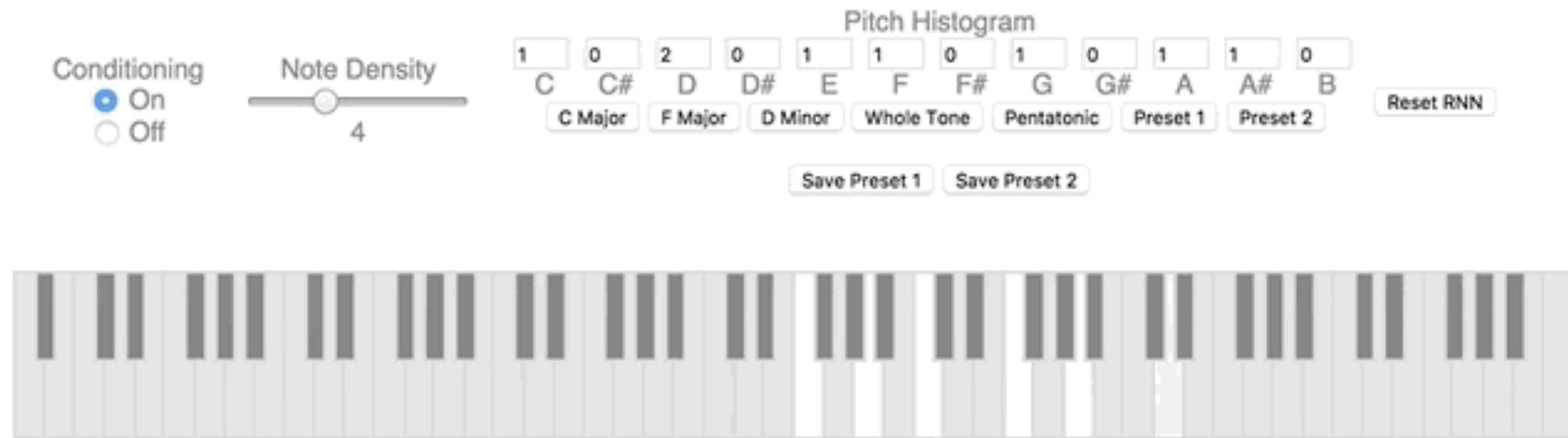


Portraits of Imaginary People

Credit: Mike Tyka's Portraits of Imaginary People from the NIPS 2017 creativity art gallery

Reinforcement Learning + Creativity

Performance RNN

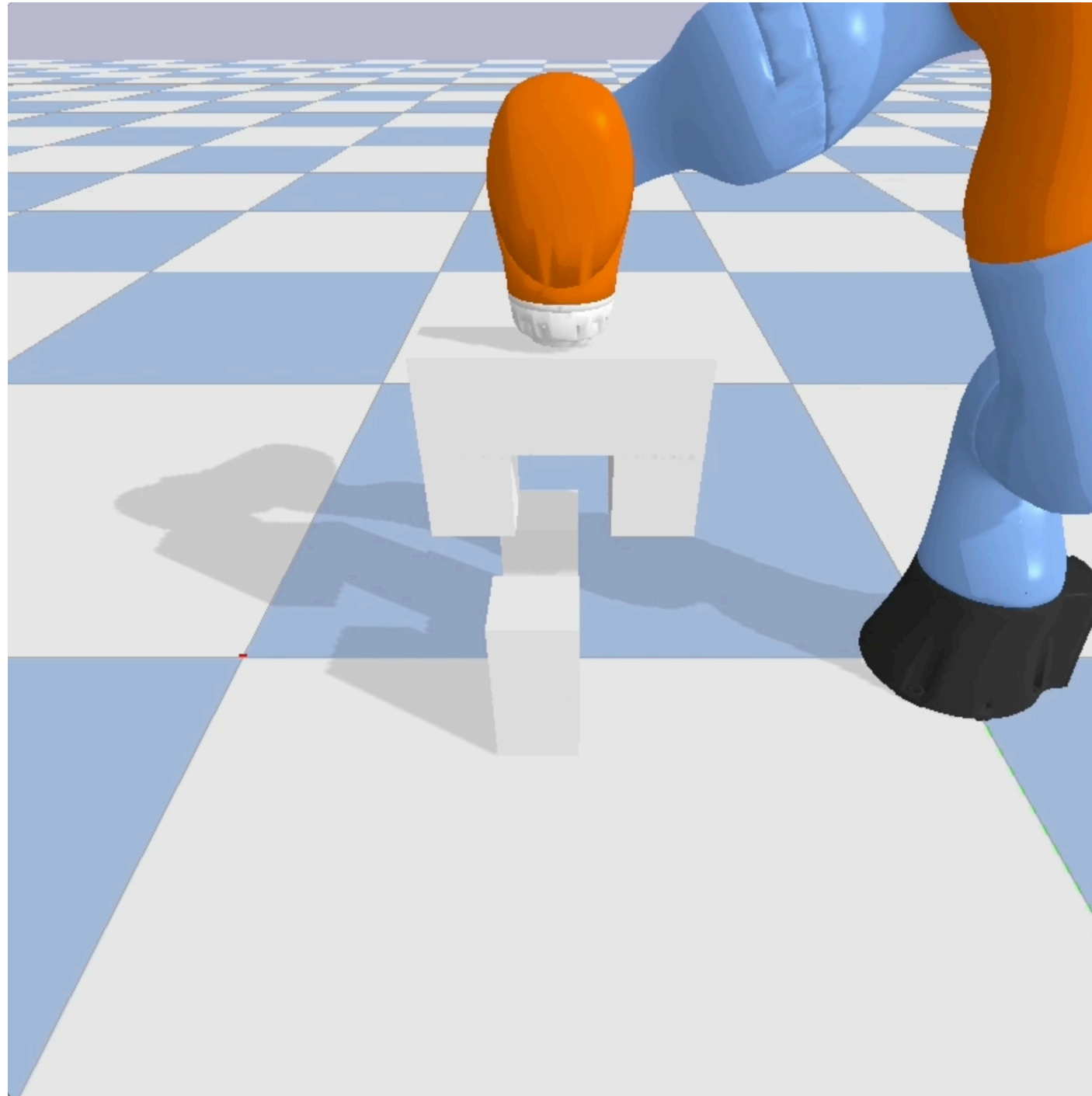


[Performance RNN](#) was trained in TensorFlow on MIDI from piano performances. It was then ported to run in the browser using only Javascript in the [deeplearn.js](#) environment.

Performance RNN

Credit: Performance RNN from magenta.tensorflow.org

Reinforcement Learning + Creativity



Robotic Joint-Assembly Tasks

Credit:Autodesk Robotics Lab

Reinforcement Learning + Creativity

Why we don't see many creative applications in RL?

Reinforcement Learning + Creativity

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
 Initialize a random process $\mathcal{N}$ for action exploration
 Receive initial observation state s_1
 for t = 1, T **do**
 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
 Execute action a_t and observe reward r_t and observe new state s_{t+1}
 Store transition (s_t, a_t, r_t, s_{t+1}) in R
 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\begin{aligned}\theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}\end{aligned}$$

end for
end for

Hurdle I: In-depth understanding of RL algorithms

Credit: Deep Deterministic Policy Gradient algorithm by DeepMind

Reinforcement Learning + Creativity

```
for episode in range(MAX_EPISODES):
    state = env.reset()

    for step in range(MAX_STEPS):
        if episode > warmup_episode:
            # select action
            action = actor.forward(Variable(torch.Tensor(state)))
            action += random_process.sample()
        else:
            action = np.random.uniform(-1., 1., action_dim)

        # execute action
        next_state, reward, done, info = env.step(action)

        # store transition in the memory
        memory.push(state, action, next_state, reward)

        if episode > warmup_episode:
            # sample a random mini batch
            #transitions = memory.sample(BATCH_SIZE)
            states, actions, next_states, rewards = memory.sample(BATCH_SIZE)

            # get a batch of q target values
            next_action_values = actor_target.forward(Variable(torch.Tensor(next_states)))
            next_q_values = critic_target.forward(Variable(torch.Tensor(next_states), next_action_values))
            target_q_batch = torch.Tensor(rewards) + discount*next_q_values

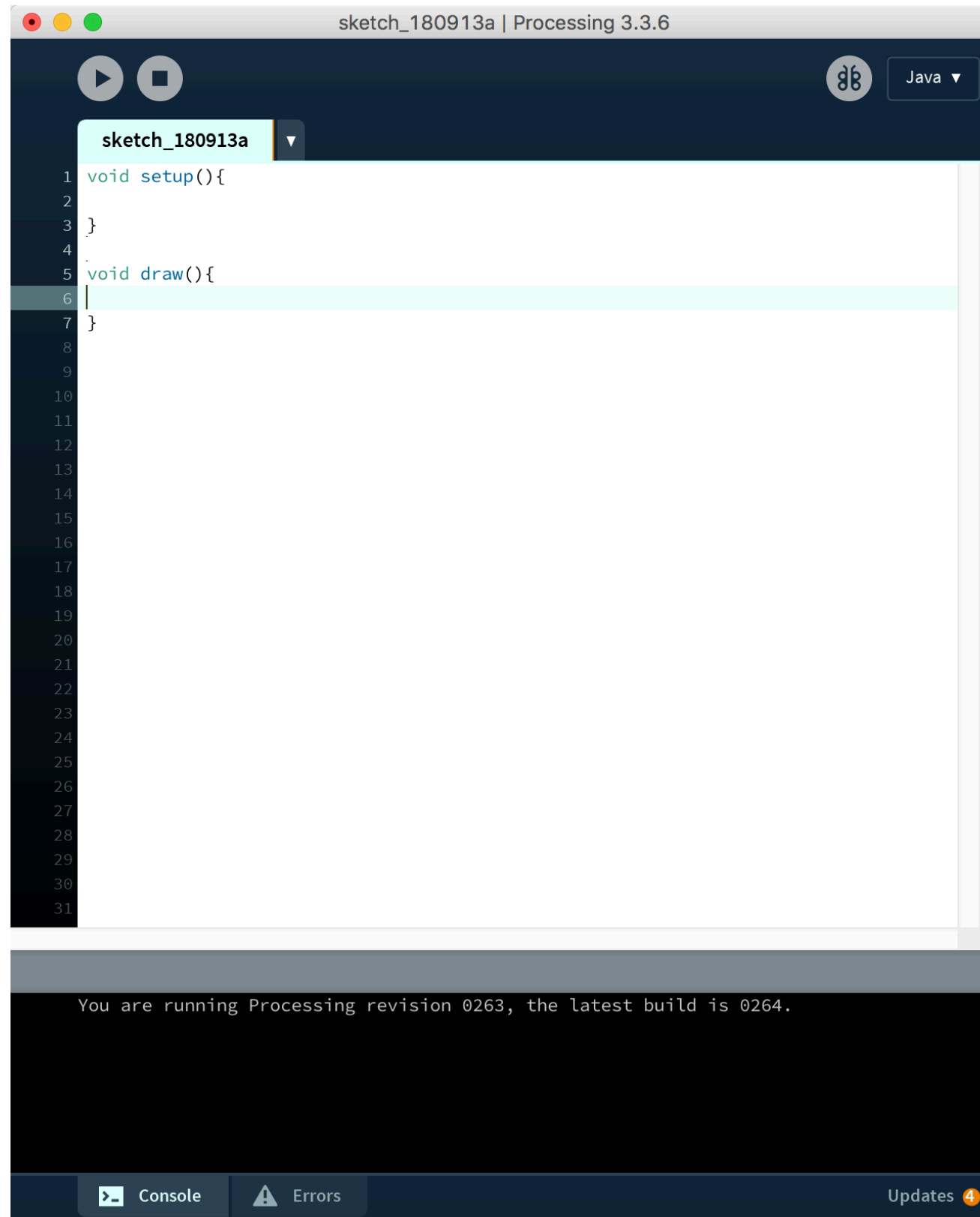
            # update critic
            q_batch = critic.forward(Variable(torch.Tensor(states)), Variable(torch.Tensor(actions)))
```

Hurdle II: Current RL tools are designed for scientific researches

Chapter III:

Why Processing

Why Processing



A screenshot of Processing IDE

Why Processing



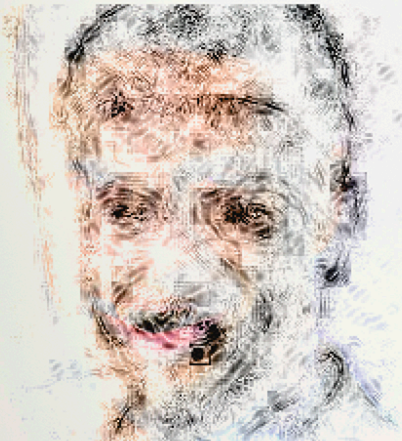
Abstract Reality, Jieliang Luo, SIGGRAPH Asia 2017

Why Processing



Machinery Interference
Jieliang Luo, Times Arts Museum, Beijing, 2018

Why Processing



Ferret, Windsor Tie, and Abaya



Swimming Trunks, Brassiere, and Wig



Band Aid, Nematode, and Lipsticks



Maillot, Thunder Snake, and Sunglass



Windsor Tie, Tick, and Proboscis Monkey



Hair Slide, Brassiere, and Sunglass



Brassiere, Pedestal, and Indian Cobra



Indian Cobra, Butternut Squash, and Snail



Brassiere, Ice Lolly, and Yorkshire Terrier

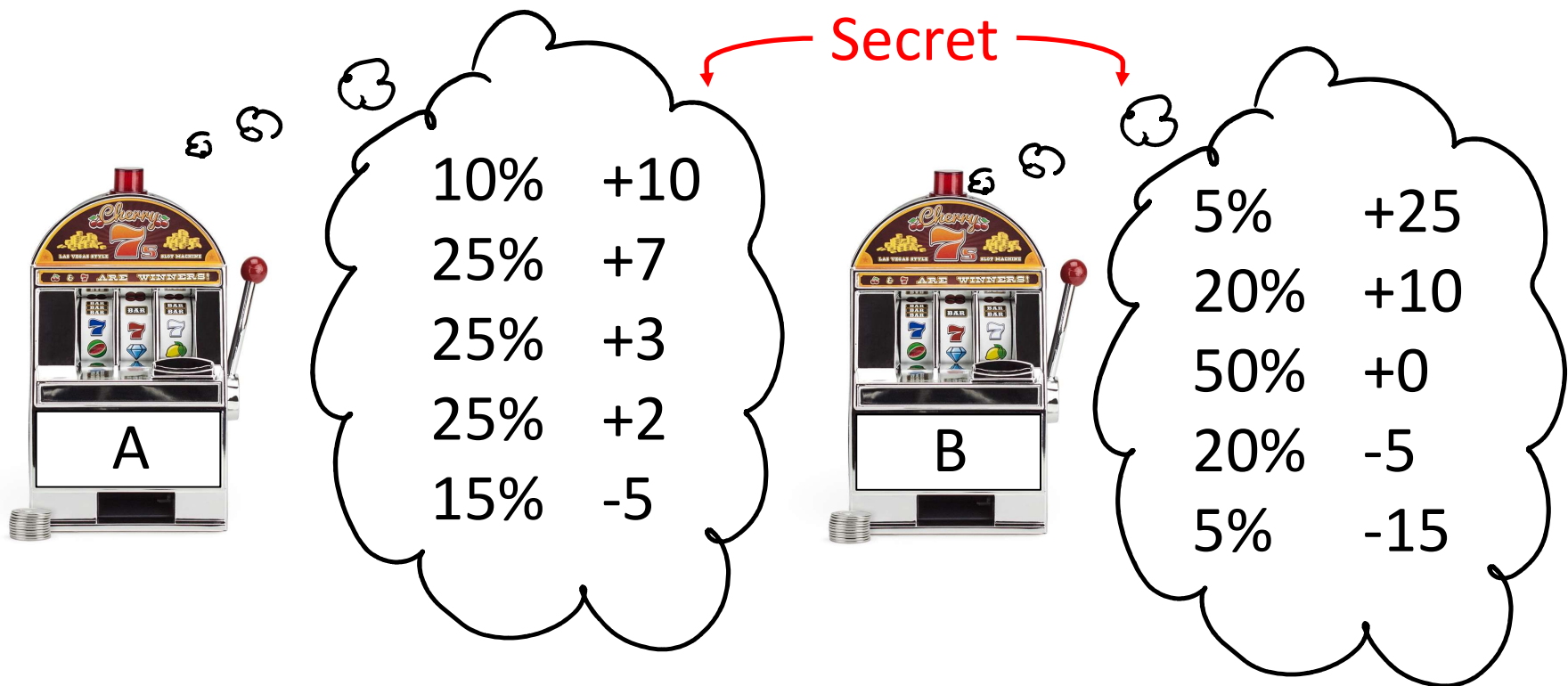


Tick, Platypus, and Eggnog

Chapter IV:

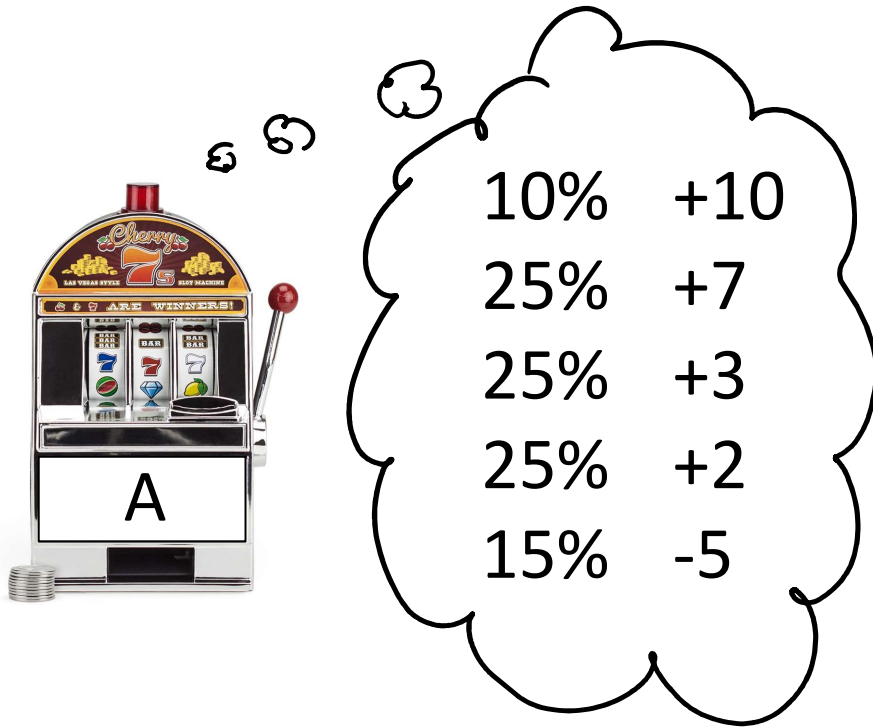
Q-Learning & Policy Gradient

Q: How to assign value to actions?



Idea: Pull lever many times and keep track of each reward and its frequency

Secret

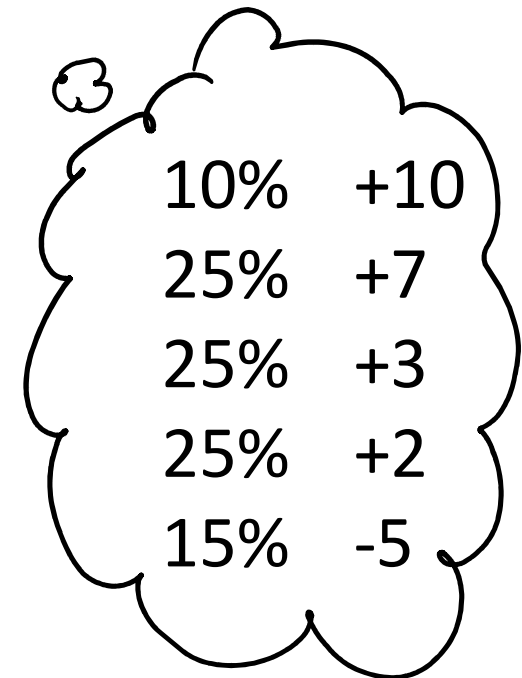


Value	Frequency
+10	
+7	
+3	
+2	
-5	

Idea: Pull lever many times and keep track of the average reward

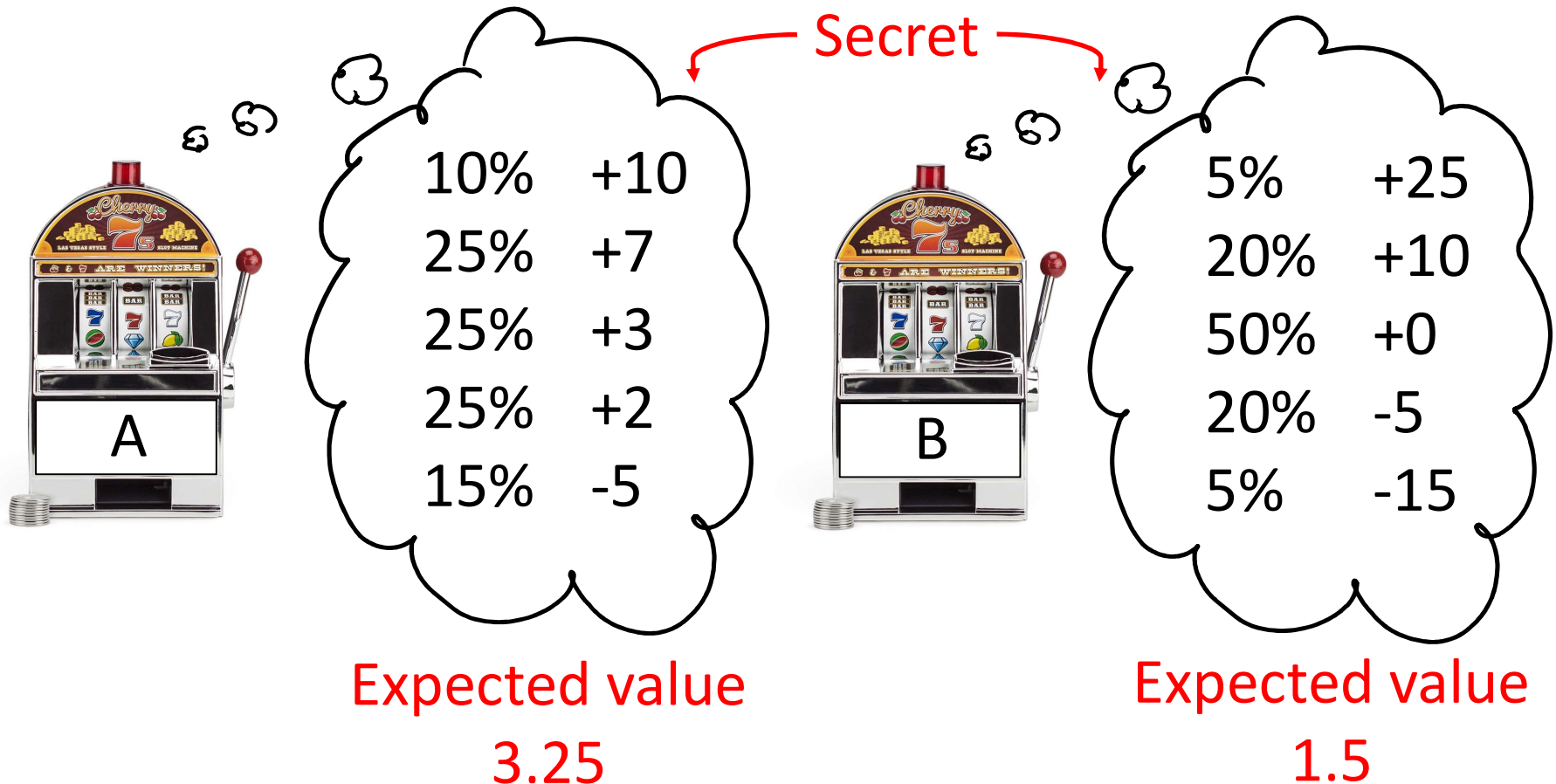
<u>Try</u>	<u>Reward</u>	<u>Average</u>	
1	7	$7/1$	$= 7$
2	-5	$(7-5)/2$	$= 1$
3	7	$(7+7-5+7)/3$	$= 3$
4	7	$(7+7-5+7)/4$	$= 4$
.	.	.	
.	.	.	
.	.	.	
N		Total rewards/N ≈ 3.25	

Secret



Expected value
3.25

Use average value for decision making



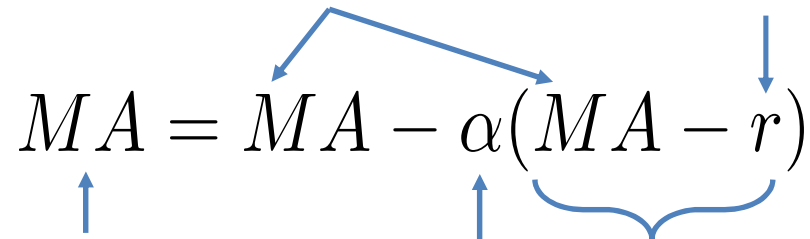
∴ Play A

How to calculate efficiently?

<u>Try</u>	<u>Reward</u>	<u>Average</u>		Must recalculate numerator each time!
1	7	$7/1$	$= 7$	
2	-5	$(7-5)/2$	$= 1$	
3	7	$(7+7+7)/3$	$= 3$	
4	7	$(7+7-5+7+7)/4$	$= 4$	What if N is 1,000,000?
.	.	.		
.	.	.		
.	.	.		
N		Total rewards/N ≈ 3.25		

Use moving average (MA)

Estimate from last step Reward from this step

$$MA = MA - \alpha(MA - r)$$


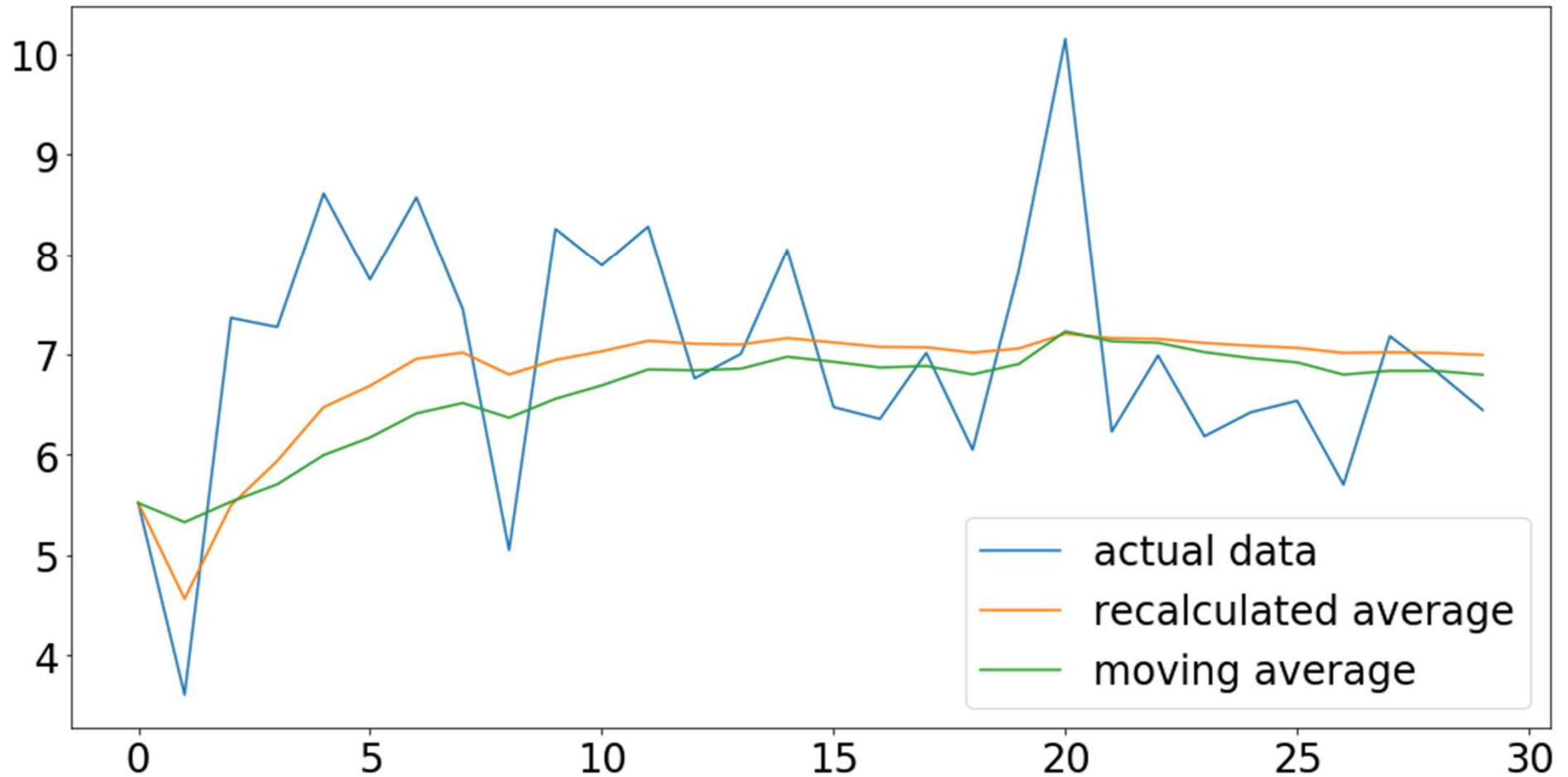
Updated estimate Step size Estimate error

<u>Try</u>	<u>Reward</u>	<u>MA</u> ($\alpha = .1$)
1	7	7 (MA initialized to first reward)
2	-5	5.8
3	7	5.92
4	7	6.03

.
.
.

Over time, MA converges
to expected value.

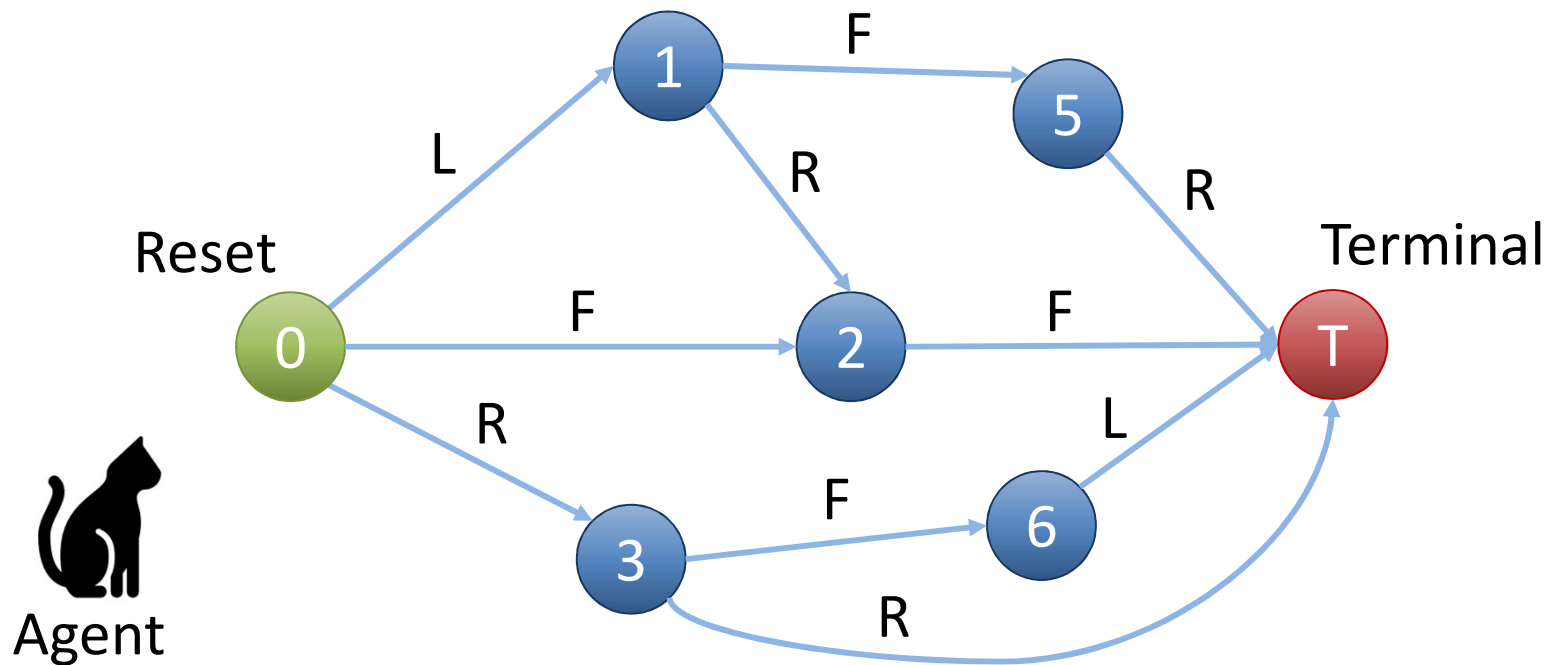
Recalculated vs. Moving Average



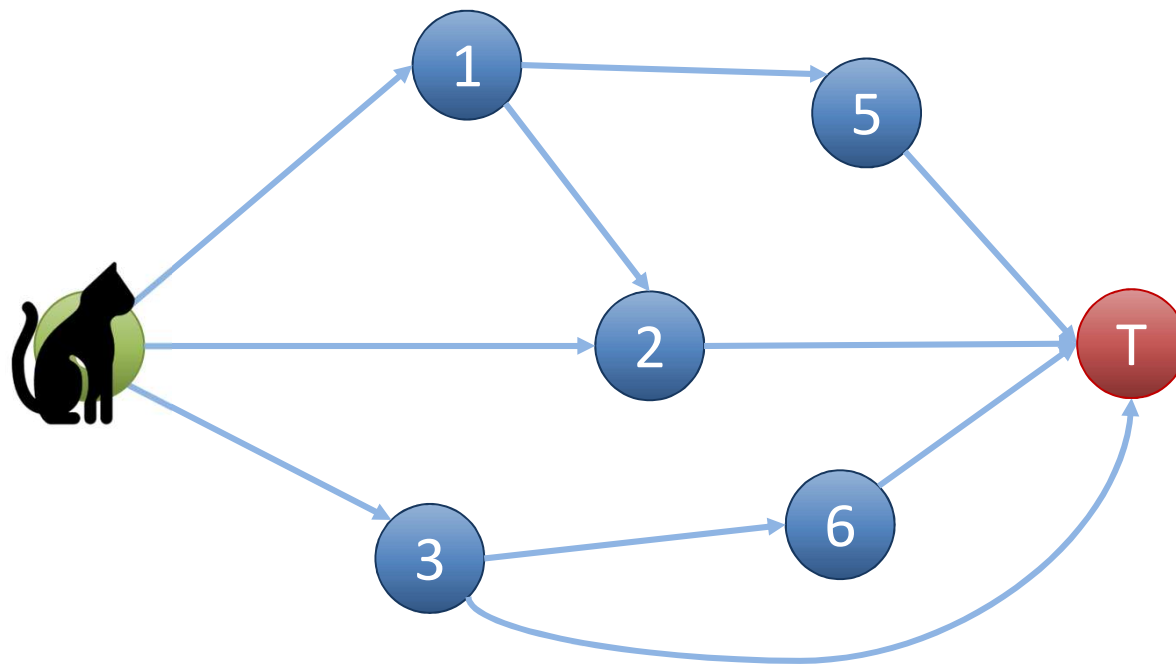
The “noise” of the moving average is worth the computational savings.

Markov Reward Process

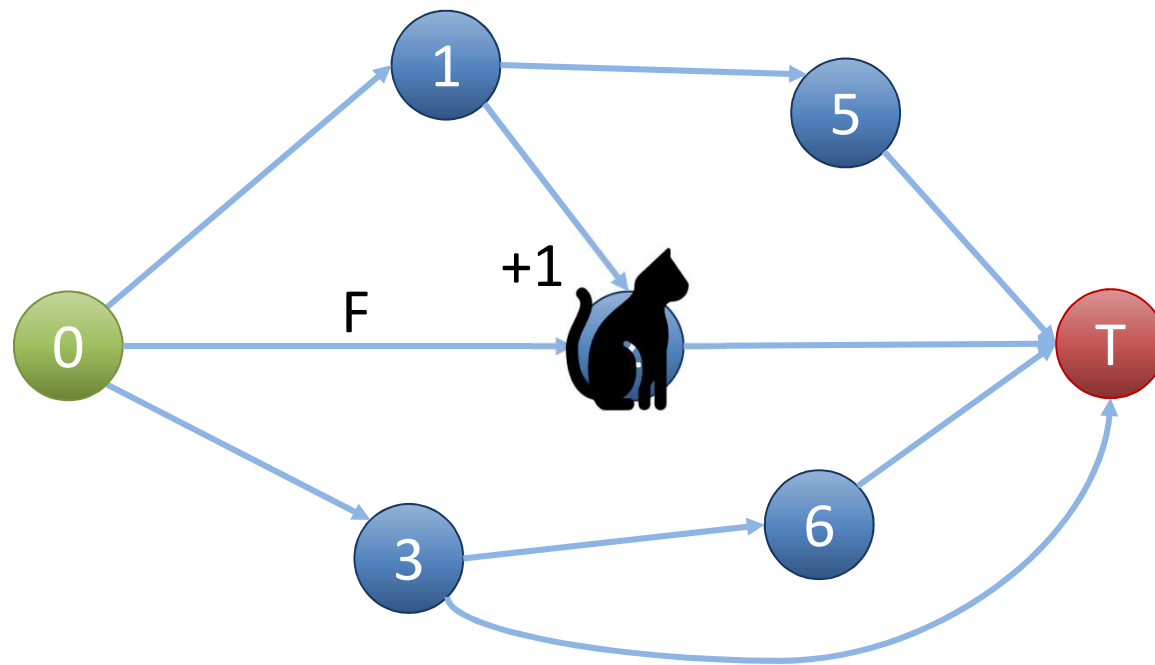
- A mathematical model where agent *tries* to extract rewards from environment



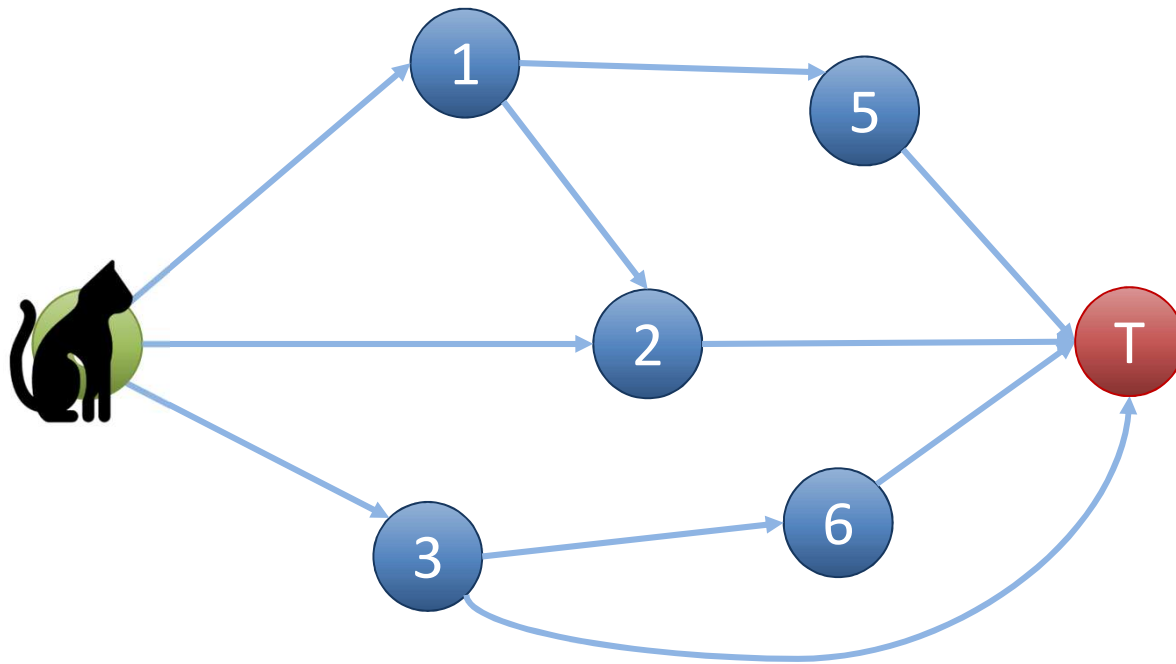
Rewards given at state transition



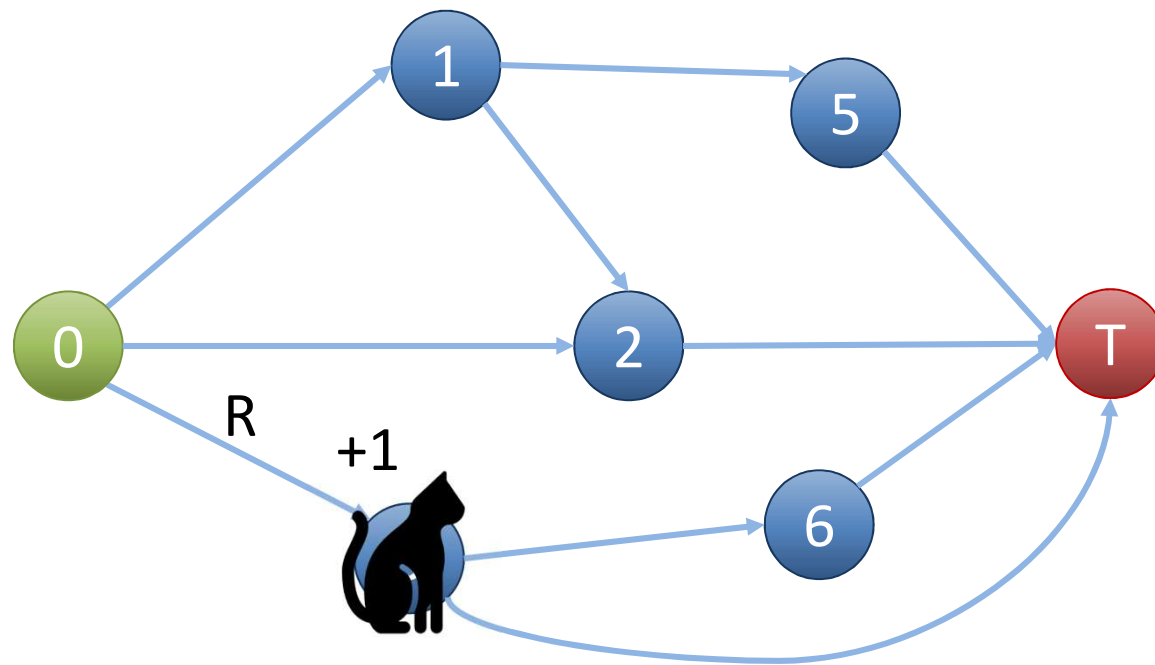
Rewards given at state transition



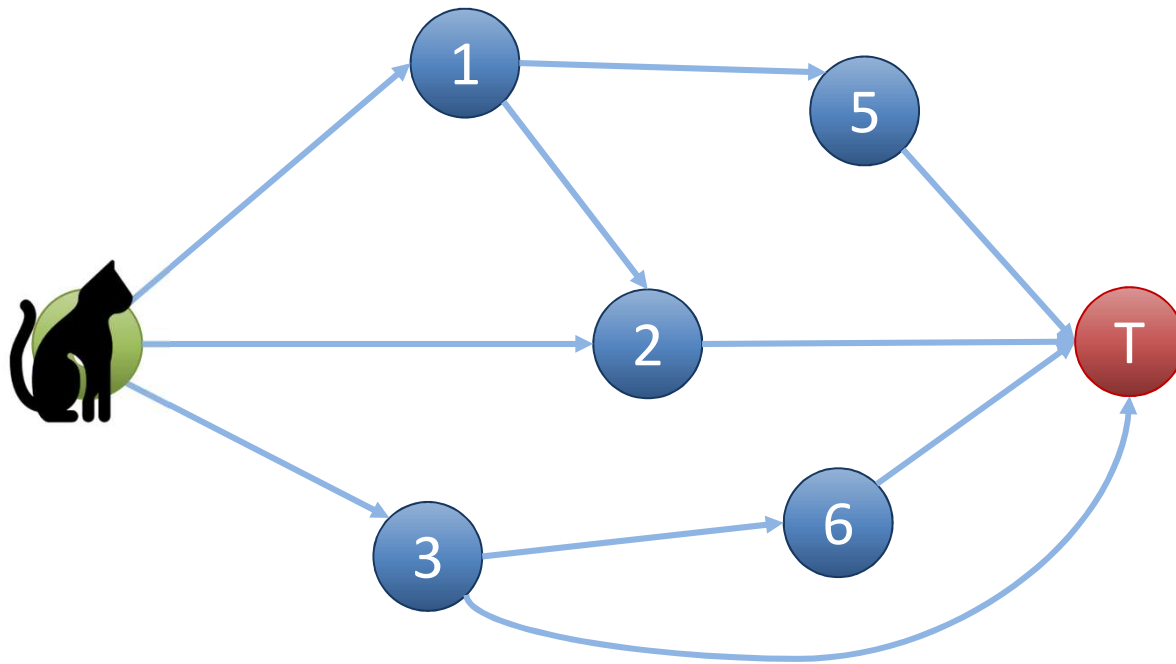
Rewards may be stochastic



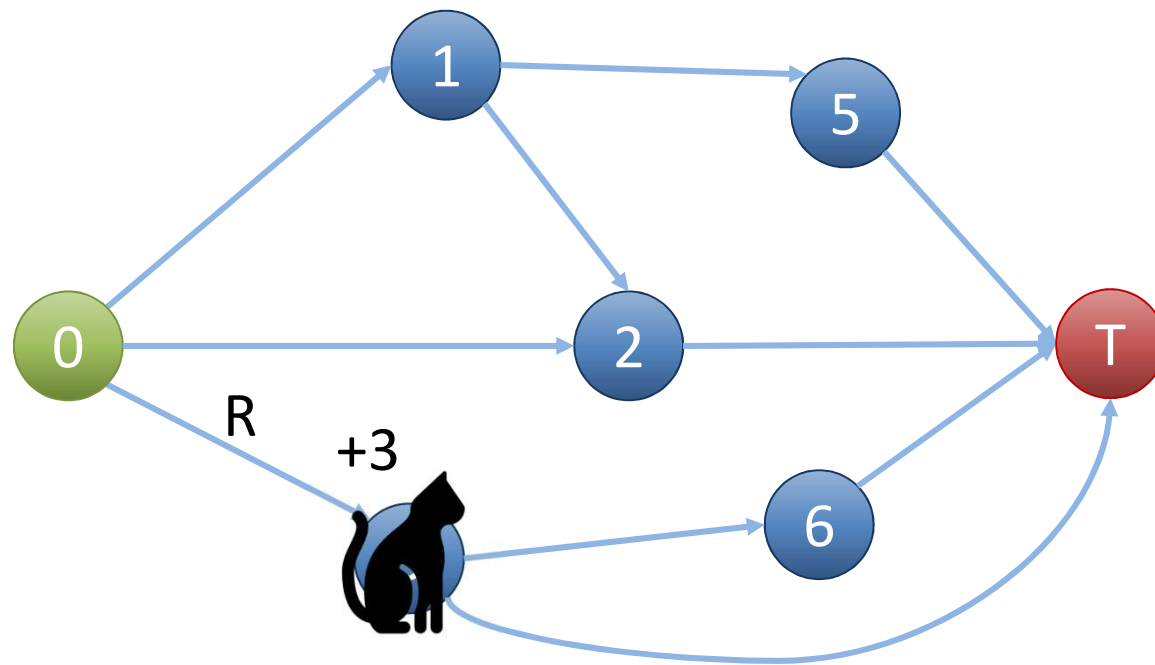
Rewards may be stochastic



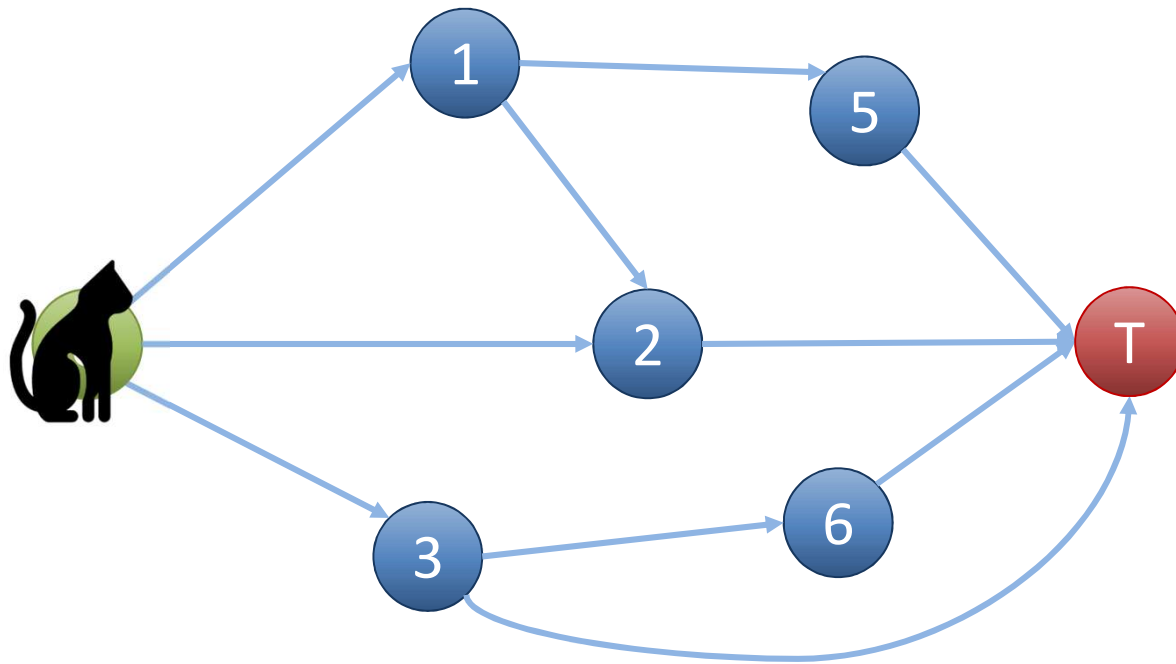
Rewards may be stochastic



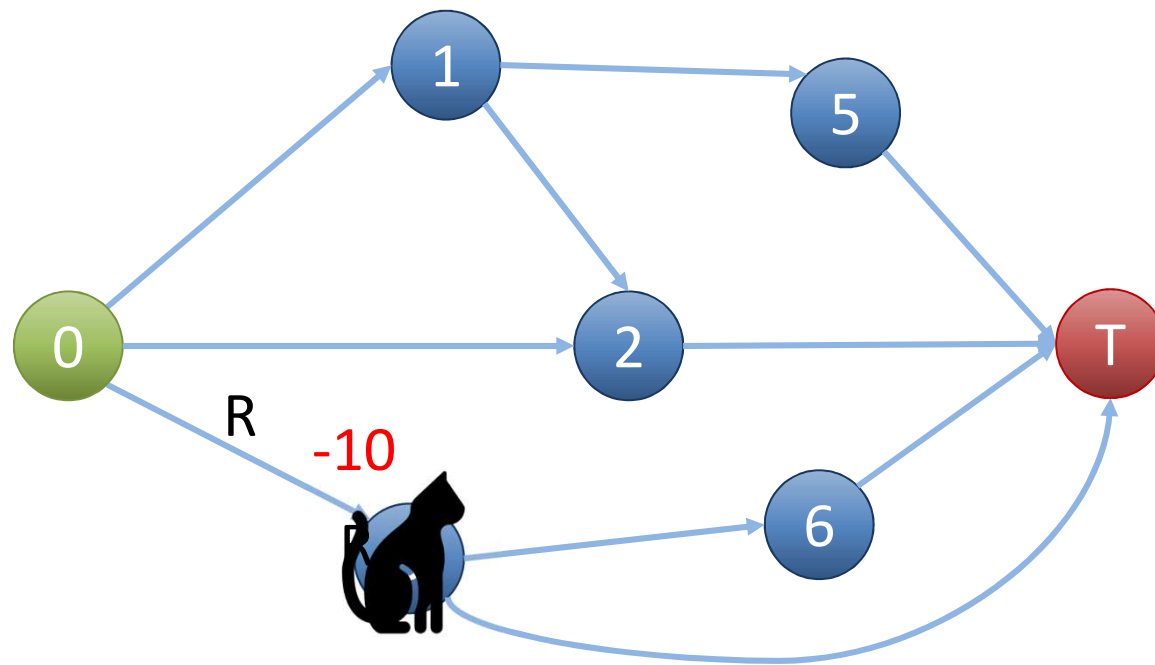
Rewards may be stochastic



Rewards may be stochastic



Rewards may be stochastic



Agent needs to maximize the *return*

- The *return* (R) is the expected sum of rewards:

Timesteps in episode

$$R = \sum_{t=0}^{T-1} \gamma^t r_t$$

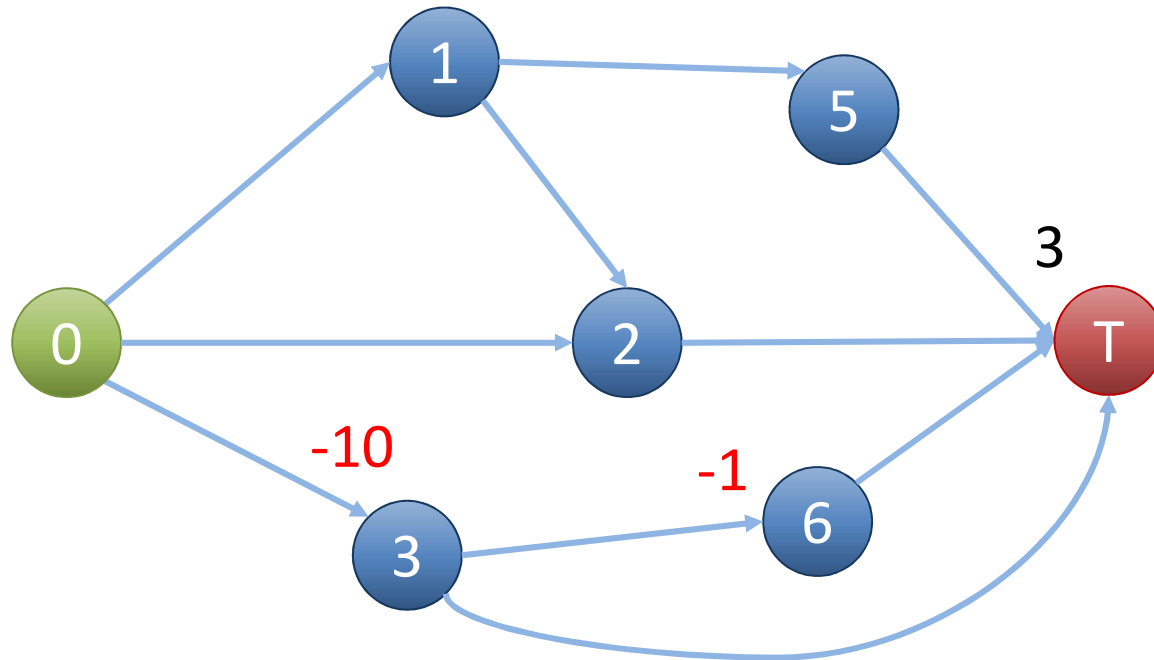
← Reward at timestep

Discount factor

- Example of impact of discount factor

γ	r_0	r_1	r_2	R
1	-7	3	6	2
.5	-7	3	6	-4

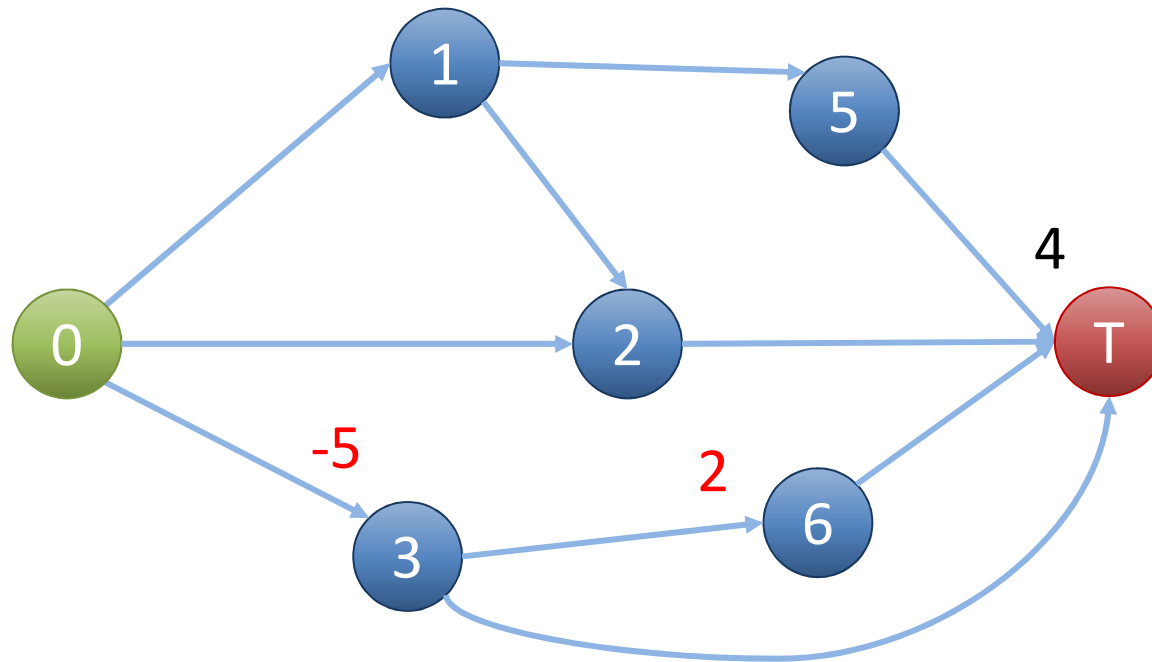
Track the moving average of the return
to evaluate actions ($\gamma = .99, \alpha = .1$)



$$R = .99^0(-10) + .99^1(-1) + .99^2(3) = -8.05$$

$$Q(0, R) = -8.05$$

Track the moving average of the return
to evaluate actions ($\gamma = .99, \alpha = .1$)



$$R = .9$$

$$Q(0, R) = Q(0, R) - \alpha(Q(0, R) - R)$$

$$Q(0, R) = -8.05 - .1(-8.05 - .9) = -7.2$$

How to use the information?

- Explore the environment many times
- Calculate Q-values for storing the value of decisions
- After training, use the Q-values to make actions

Q-value table for states 0, 1, and 2

State	Action	Q-value
0	L	-4.5
0	F	3.0
0	R	4.7
1	F	4
1	R	-8.5
2	F	3.2

Derive a policy by taking the action with largest Q-value:

$$a_t = \operatorname{argmax}_a Q(s, a)$$

Monte Carlo vs Temporal Difference

- Previous approach was *Monte Carlo*: perform entire rollouts to calculate returns
- *Temporal difference* methods are online: update Q-values as soon as possible

The diagram illustrates the Temporal Difference (TD) update equation. The equation is $Q(s, a) = Q(s, a) - \alpha(Q(s, a) - r + \arg\max_{a'} Q(s', a'))$. Annotations include: 'Old estimate' with a blue arrow pointing to the first $Q(s, a)$ term; 'Target value' with a blue bracket over the $r + \arg\max_{a'} Q(s', a')$ term; and 'TD error' with a blue bracket under the entire term being subtracted, $(Q(s, a) - r + \arg\max_{a'} Q(s', a'))$.

$$Q(s, a) = Q(s, a) - \alpha(Q(s, a) - r + \arg\max_{a'} Q(s', a'))$$

Old estimate

Target value

TD error

Clarifying temporal difference error

$$\begin{array}{ccc} \text{Old estimate} & & \text{New estimate} \\ \underbrace{Q(s, a)} & - r + & \underbrace{\operatorname{argmax}_{a'} Q(s', a')} \\ \uparrow & & \uparrow \\ \text{New information} & & \text{Old information} \end{array}$$

Q-values approximate the expected value when the current action “a” is taken in state “s” and then the best known actions are taken in all subsequent steps.

Exploration vs Exploitation

- So far we have exhaustively explored the environment
- How to reuse information we already have?
- One approach is ϵ -greedy: choose random action (explore) with probability ϵ , choose greedy action (exploit) with probability $1 - \epsilon$
- ϵ commonly set to .05

ϵ -greedy example in state 0

State	Action	Q-value
0	L	-4.5
0	F	3.0
0	R	4.7

- Use random number generator to get a value:
`rand = random()`
- If $\text{rand} < \epsilon$ then pick L, F, or R randomly
- Otherwise pick action R (it has highest Q-value)

Algorithm 1 Tabular Q-Learning

```
1: Set all  $Q$ -values to 0
2: while  $Q$ -table has not converged do:
3:     Reset the agent and environment
4:     while Episode not terminated do
5:         Choose  $a$  from  $Q(s, \star)$  using  $\epsilon$ -greedy
6:         Take action  $a$ , observe  $r$  and  $s'$ 
7:          $Q(s, a) = Q(s, a) - \alpha(Q(s, a) - r + \operatorname{argmax}_{a'} Q(s', a'))$ 
8:          $s \leftarrow s'$ 
9:     end while
10: end while
11: Return trained  $Q$ -table
```

Intro to Gradient Methods

- Q-Learning tracks the average *return* for state-action pairs
- *Gradient methods* assign a numerical preference to state-action pairs:

$$H(s_t, a_t)$$

- Preference converted to probability with softmax:

$$\Pr\{A_t|s_t\} = \pi(a_t|s_t) = \frac{e^{H(s_t, a_t)}}{\sum_{b=1}^k e^{H(s_t, b)}}$$

Intro to Gradient Methods

- Update preference using following rule

$$H(s_t, A_t) = H(s_t, A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi(A_t|s_t))$$

where A_t is the action taken,

and R_t is the reward at time t,

and $\bar{R}_t$ is the average reward at time t

Approximate methods

- If number of states is extremely high and/or number of actions is high (e.g. images)
- Then tables become impractical:
 $\text{\#entries} = \text{\#states} \times \text{\#actions}$
- Use regression to approximate the table
- Neural network used for regression
- Basis for all modern reinforcement learning
- Uses similar update rules for the network

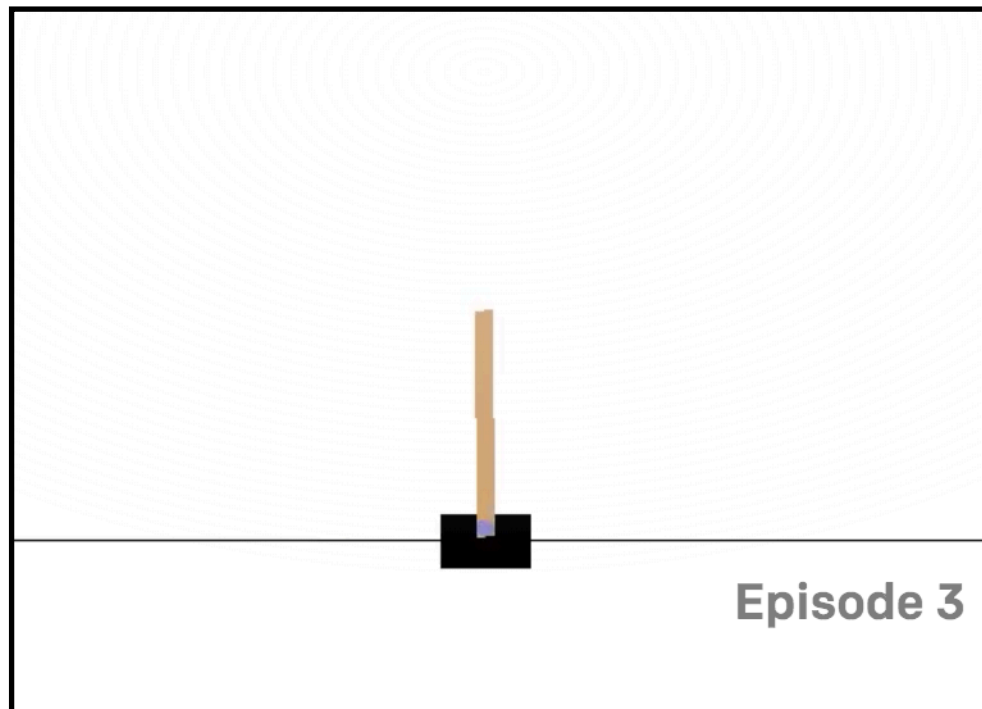
Chapter V:

Implementing Tabular Q-Learning in Processing

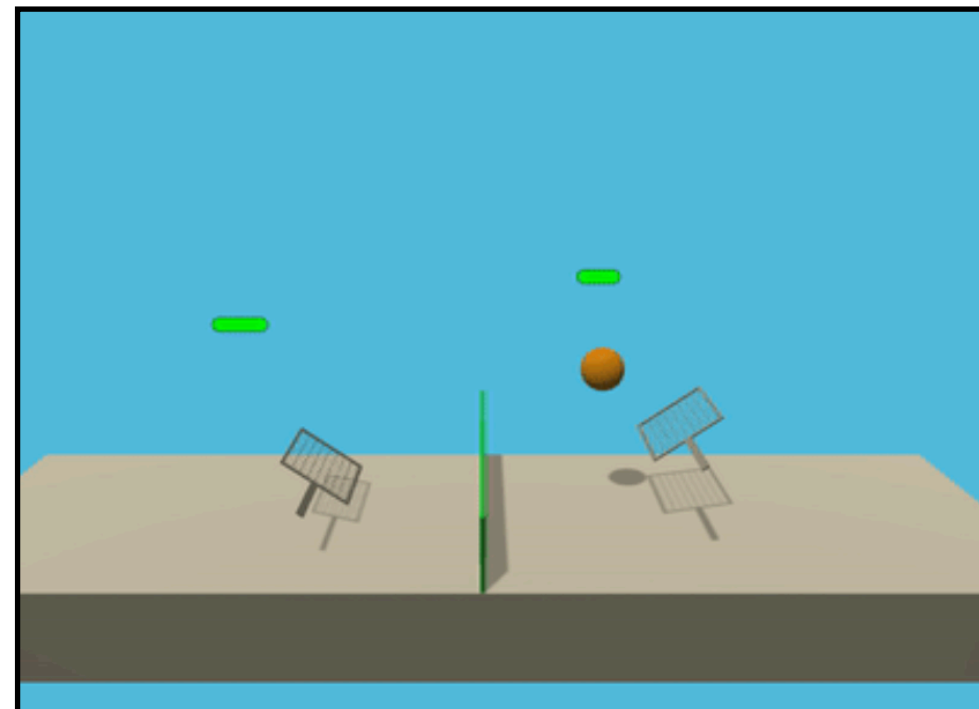
Implementing Tabular Q-Learning in Processing

Part I: Create a RL environment

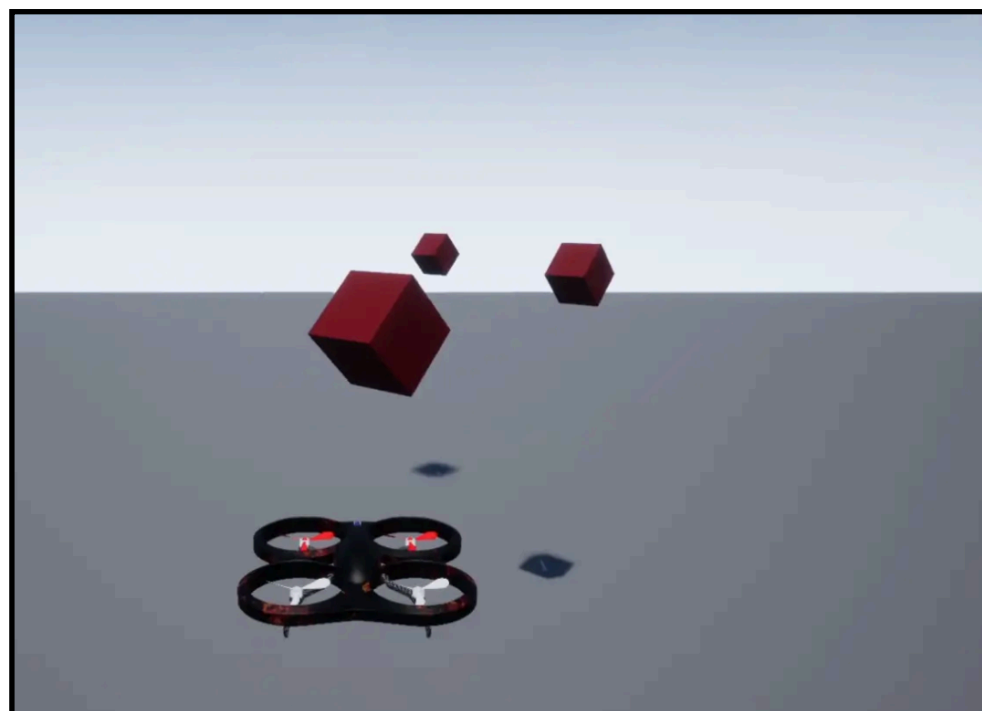
Reinforcement learning environments



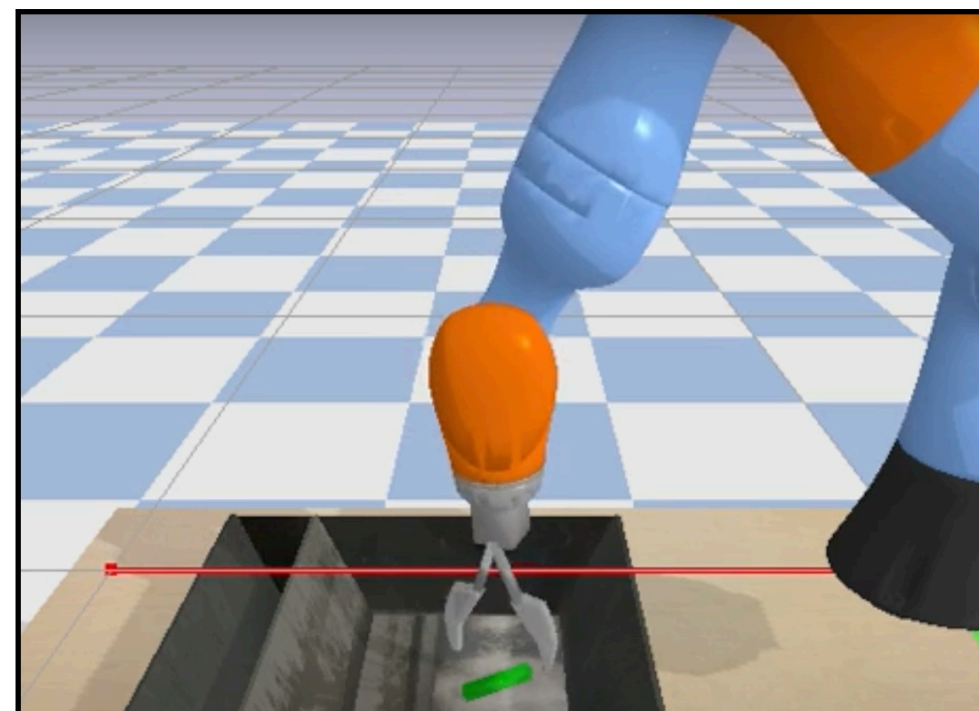
CartPole, OpenAI Gym



Tennis, Unity ML

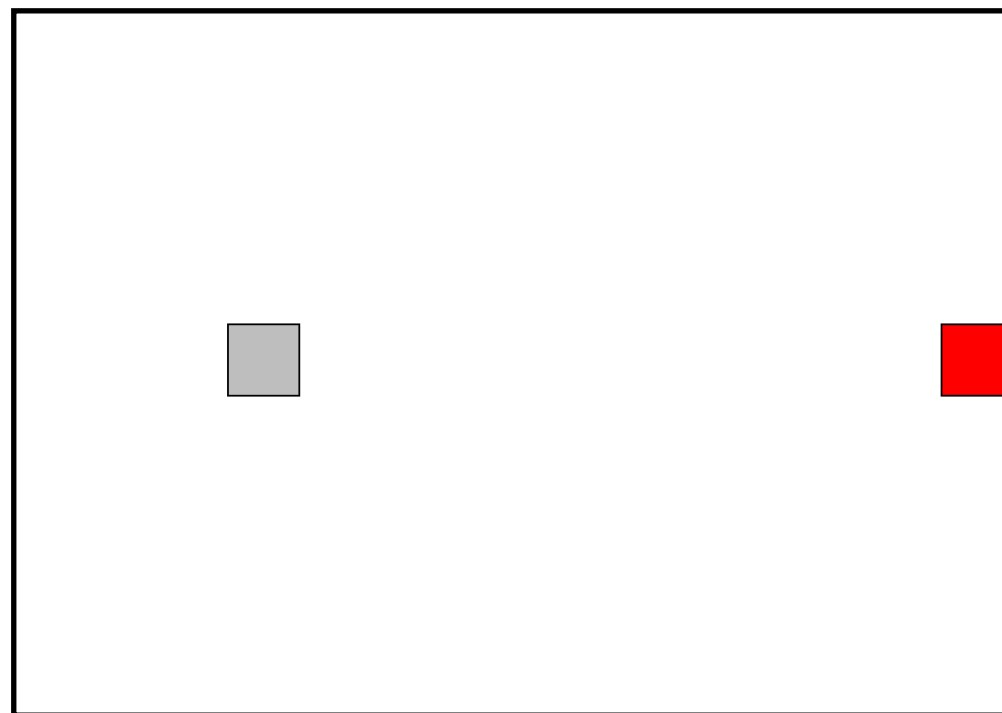


Collect Box, Luo & Green



KUKA Grasp, pyBullet

Reinforcement learning environments



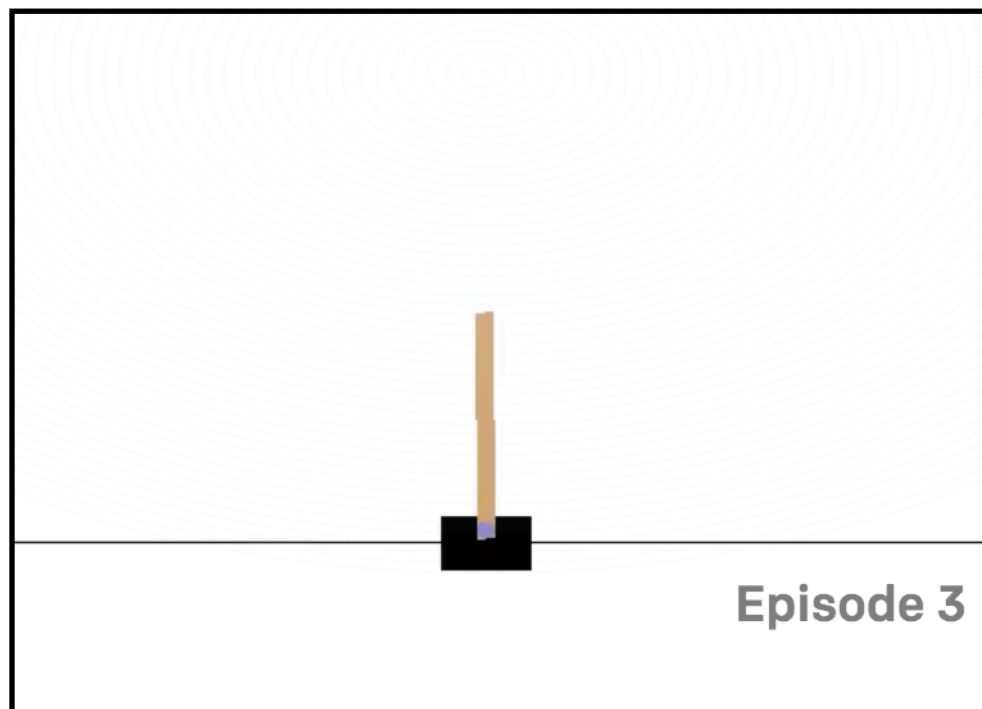
Grid in Processing, Luo & Green

Reinforcement learning environments

- Observations
- Actions
- Starting/Reset State
- Terminal State
- Step Function
- Reward
- Render Function
- Sample Function

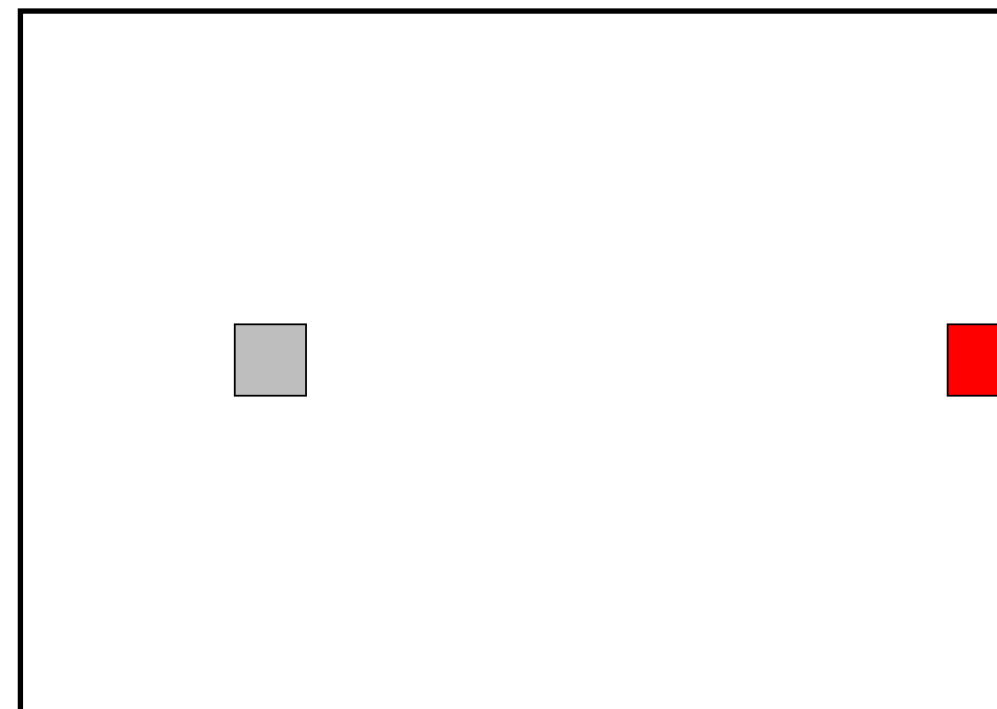
Reinforcement learning environments

- Observations



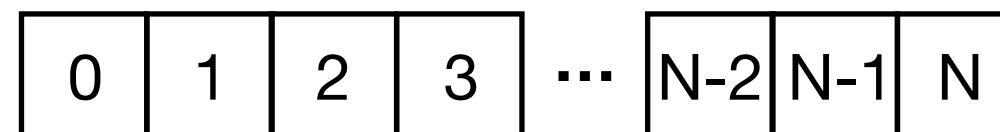
Type: Box

Observation	Min	Max
Cart Position	-4.8	4.8
Cart Velocity	-Inf	Inf
Pole Angle	-24°	24°
Pole Velocity At Tip	-Inf	Inf



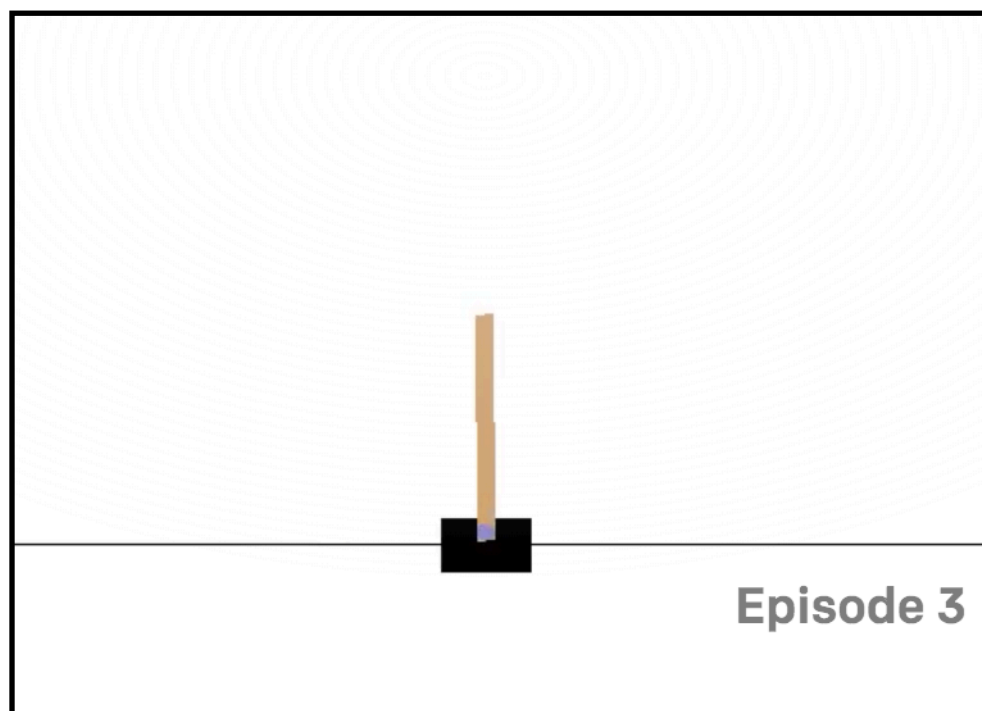
Type: Discrete

Observation	Min	Max
Grey Cube Position	Position 0	Red Cube



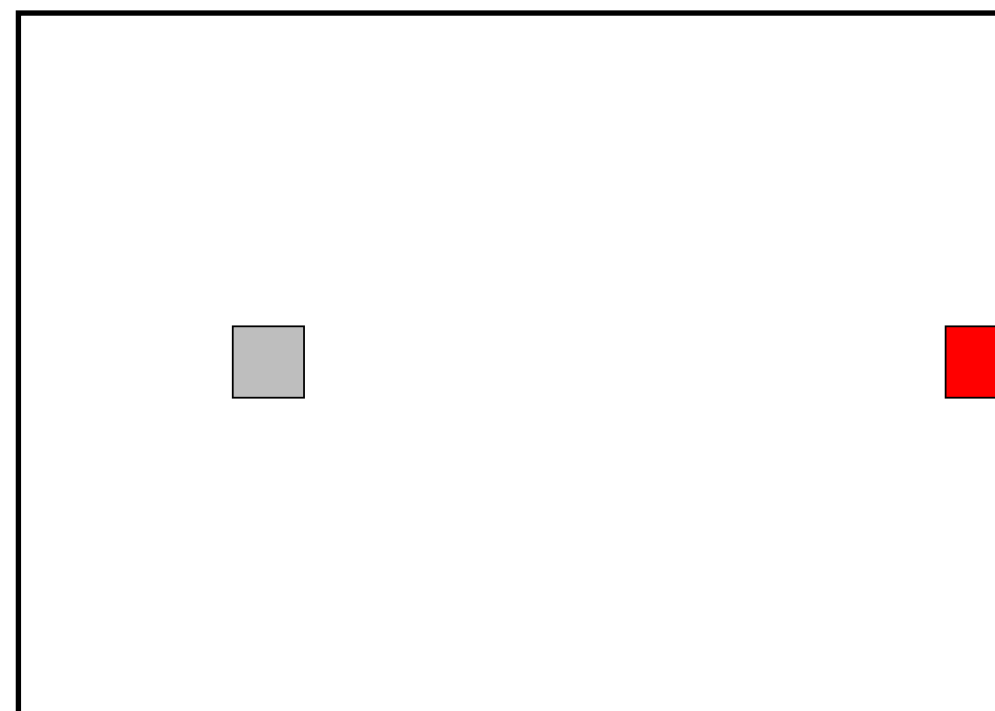
Reinforcement learning environments

- Actions



Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right



Type: Discrete(2)

Num	Action
0	Push cube to the left
1	Push cube to the right

Reinforcement learning environments

- Starting/Reset State

CartPole

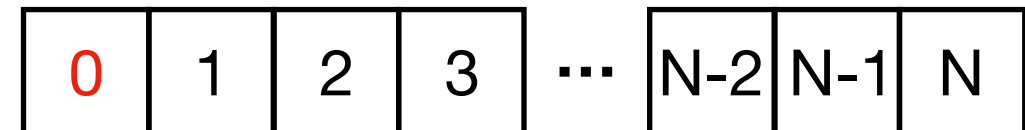
All observations are assigned a uniform random value between ± 0.05

Type: Box

Observation	Min	Max
Cart Position	-4.8	4.8
Cart Velocity	-Inf	Inf
Pole Angle	-24°	24°
Pole Velocity At Tip	-Inf	Inf

Grid in Processing

At state 0



Reinforcement learning environments

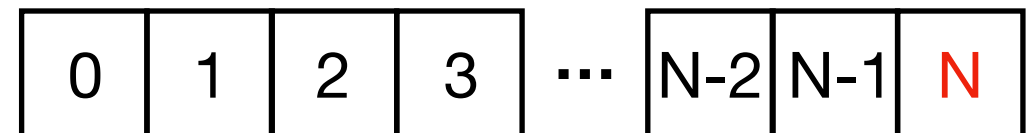
- Terminal State

CartPole

- Pole Angle is more than $\pm 12^\circ$
- Cart Position is more than ± 2.4 (center of the cart reaches the edge of the display)
- Considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials.

Grid in Processing

At state N



Reinforcement learning environments

- Terminal State

CartPole

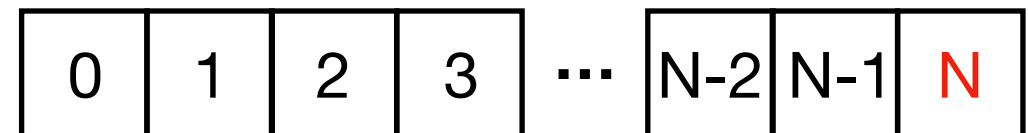
- Pole Angle is more than $\pm 12^\circ$
- Cart Position is more than ± 2.4 (center of the cart reaches the edge of the display)
- Considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials.

- Reward

Reward is 1 for every step taken, including the termination step

Grid in Processing

At state N



Reward is 0 for every step taken, excluding the termination step

Reward is 1 for the termination step

Reinforcement learning environments

- Step Function

observation, reward, done, info = step_function(action)

Observation: any state between 0 and N

Reward: 0 or 1

Done: true or false

Info: none

Codes:

https://github.com/RodgerLuo/RL_Processing/blob/master/Tabular_Q_Learning/Env.pde#L35

Reinforcement learning environments

- Basic Structure in Processing

```
class Grid{  
  
    // initialize parameters  
    Grid(){}  
  
    // reset the state to 0  
    reset(){}  
  
    // agent interacts with the environment  
    step(){}  
  
    // visualize the environment in Processing  
    render(){}  
  
    // randomly select an action  
    sample(){}  
  
}
```

Codes:

https://github.com/RodgerLuo/RL_Processing/blob/master/Tabular_Q_Learning/Env.pde

Implementing Tabular Q-Learning in Processing

Part II: Tabular Q-Learning Algorithm

Tabular Q-Learning Algorithm

Algorithm:

Algorithm 1 Tabular Q-Learning

```
1: Set all  $Q$ -values to 0
2: while  $Q$ -table has not converged do:
3:   Reset the agent and environment
4:   while Episode not terminated do
5:     Choose  $a$  from  $Q(s, \star)$  using  $\epsilon$ -greedy
6:     Take action  $a$ , observe  $r$  and  $s'$ 
7:      $Q(s, a) = Q(s, a) - \alpha(Q(s, a) - r + \operatorname{argmax}_{a'} Q(s', a'))$ 
8:      $s \leftarrow s'$ 
9:   end while
10: end while
11: Return trained  $Q$ -table
```

Implementations:

https://github.com/RodgerLuo/RL_Processing

Live Coding Session

Chapter VI:

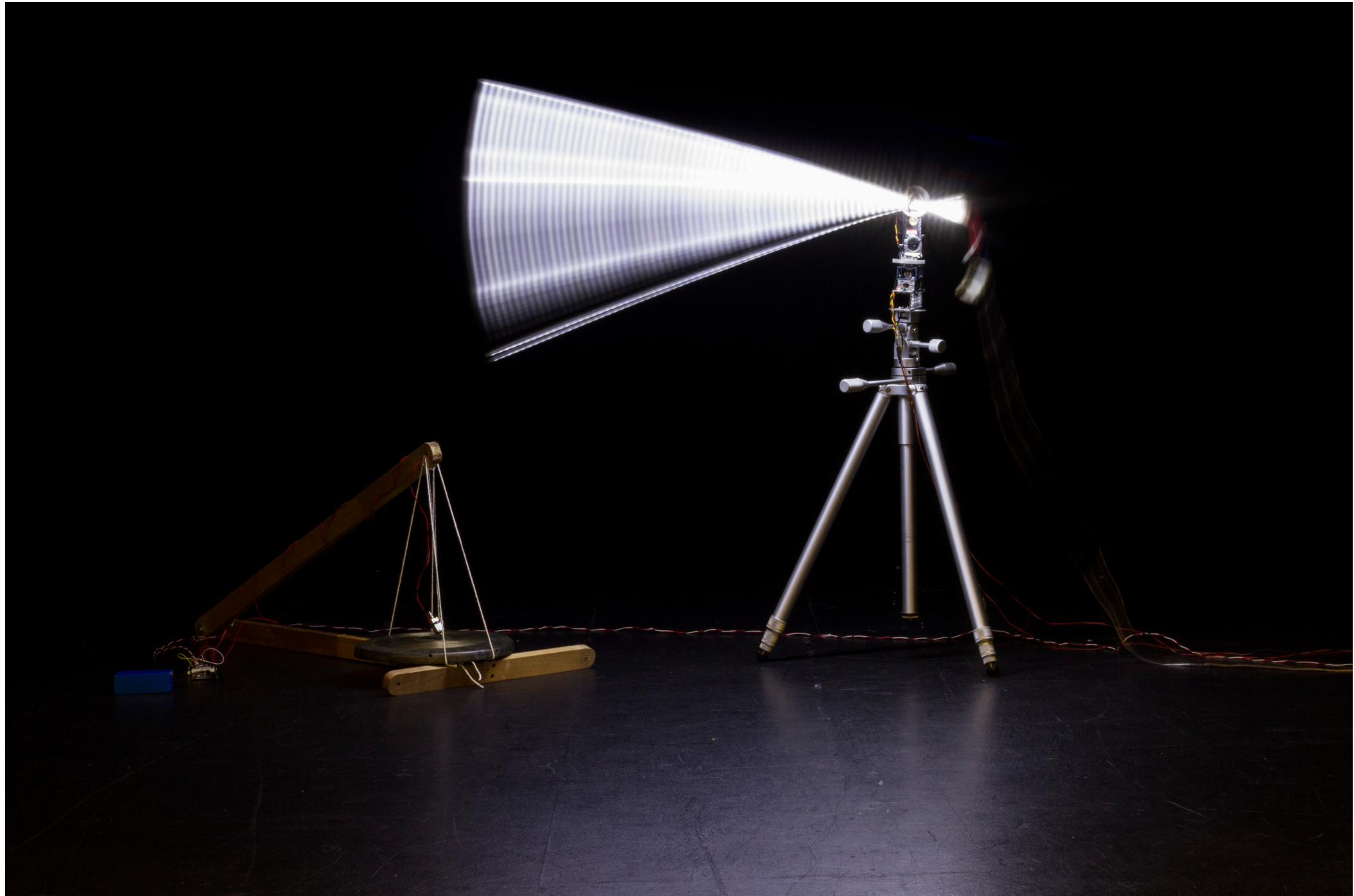
Next Steps

A Reinforcement Learning Library for Processing

- Tabular Q-Learning & Policy Gradient
- Deep Neural Network
- Policy Gradient
- Actor-Critic
- DDPG
- Customizable Environments
- Demos with Physical Objects

A Reinforcement Learning Library for Processing

- Demos with Physical Objects



Obor & Nog

Credit: Andreas Schlegel

Thank you!

Jieliang Luo
jieliang@ucsb.edu

Sam Green
sam.green@cs.ucsb.edu