CS260R Reinforcement Learning

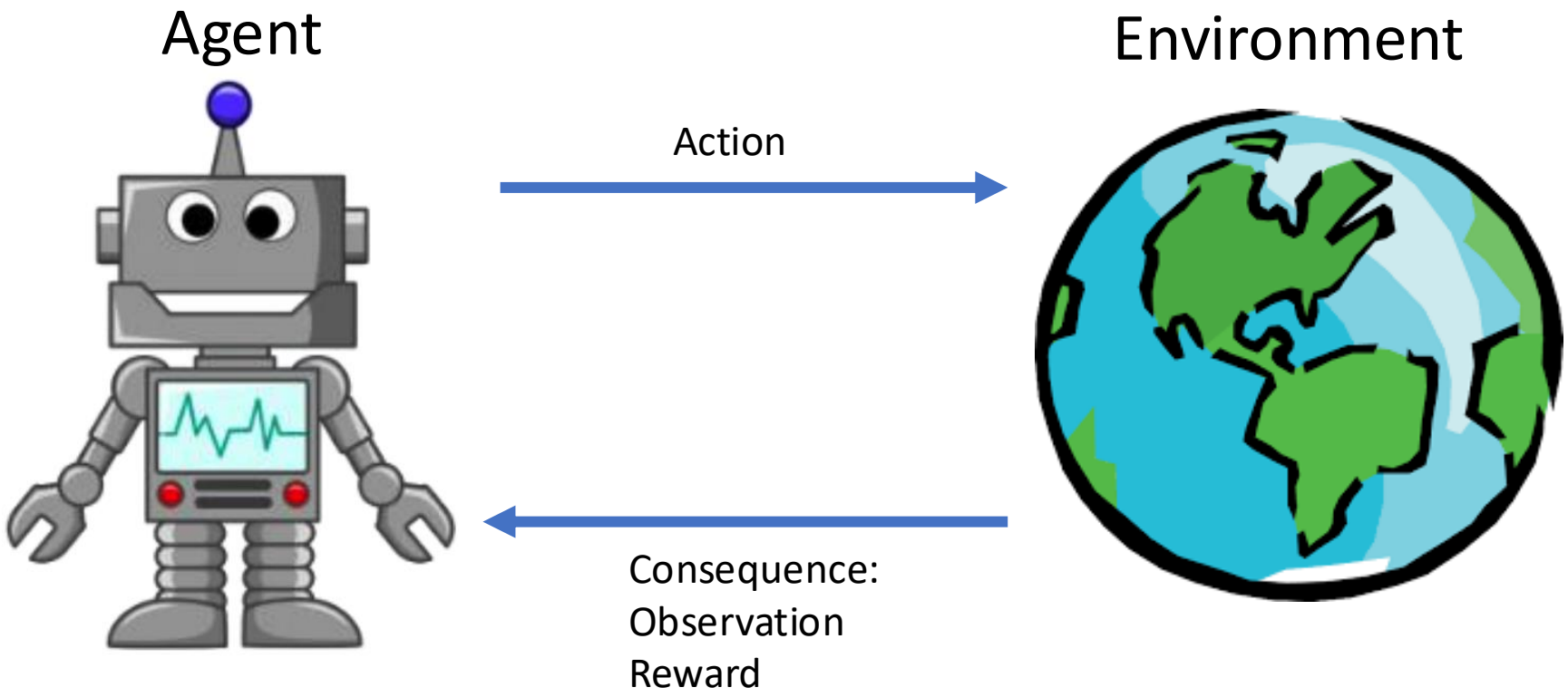
Lecture 2: RL basics and Coding with RL

Bolei Zhou

UCLA Computer Science

Agent and Environment

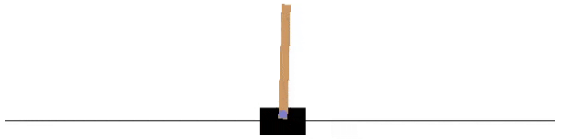
- The agent learns to interact with the environment



State/Observation

- S: representation of the current situation at time step t

Cartpole

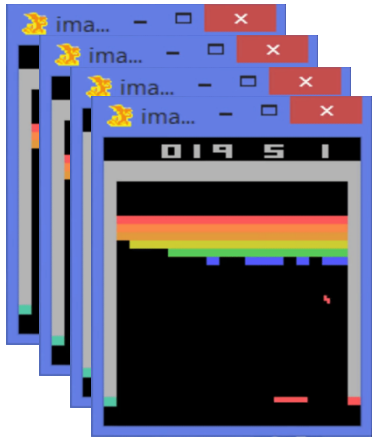


Observation

Type: Box(4)

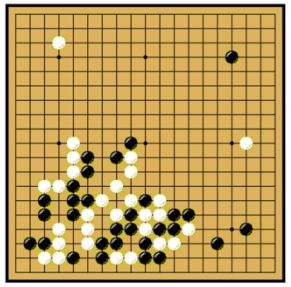
Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Breakout



Stack of four frames

AlphaGo



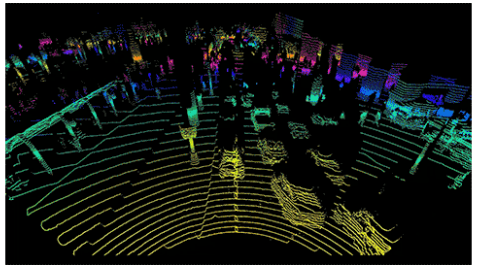
Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

19x19x48 features

Driving

RGB



LiDAR

Rewards

- A reward is a scalar feedback signal
- Rewards indicate how well agent is doing at step t
- Reinforcement Learning is based on the maximization of rewards:

All goals of the agent can be described by the maximization of expected cumulative reward.

Examples of Rewards

- Chess players play to win:

+/- reward for wining or losing a game

- Deer struggles to stand:

+/- reward for running with its mom or being eaten

- Manage stock investment

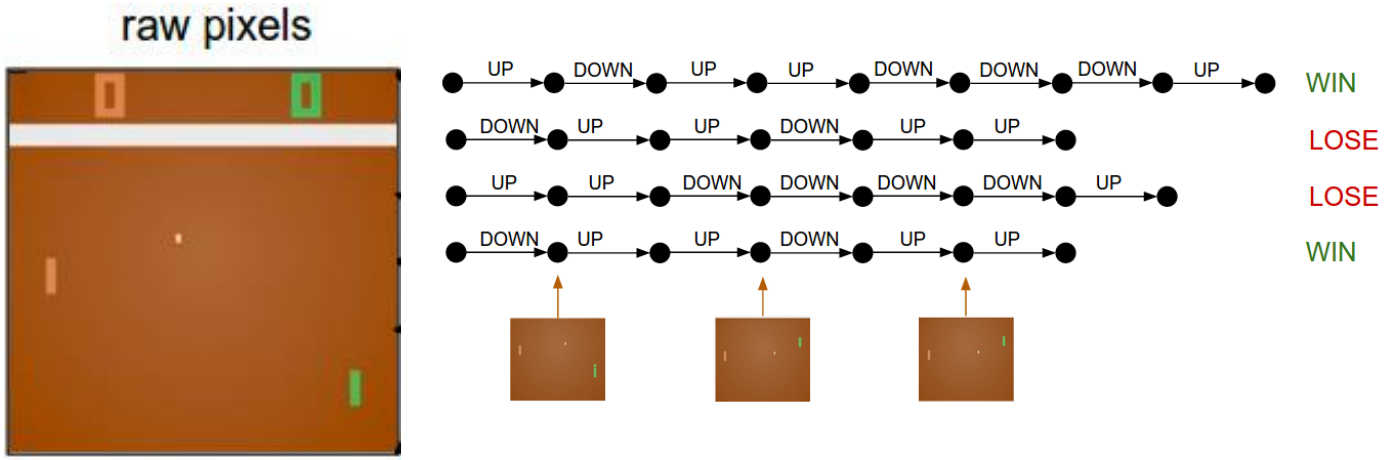
+/- reward for each profit or loss in $

- Play Atari games

+/- reward for increasing or decreasing scores

Sequential Decision Making

- Objective of the agent: select a series of actions to maximize total future rewards
- Actions may have long term consequences
- Reward may be delayed
- Trade-off between immediate reward and long-term reward



Sequential Decision Making

- The rollout trajectory is the sequence of observations, actions, rewards.

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

- What happens next depends on the history
- State is the function used to determine what happens next

$$S_t = f(H_t)$$

Sequential Decision Making

- Environment state and agent state

$$S_t^e = f^e(H_t) \qquad\qquad S_t^a = f^a(H_t)$$

- **Full observability**: agent directly observes the environment state, formally as Markov decision process (MDP)

$$O_t = S_t^e = S_t^a$$

Sequential Decision Making

- Environment state and agent state

$$S_t^e = f^e(H_t) \qquad\qquad S_t^a = f^a(H_t)$$

- **Full observability**: agent directly observes the environment state, formally as Markov decision process (MDP)

$$O_t = S_t^e = S_t^a$$

- **Partial observability**: agent indirectly observes the environment, formally as partially observable Markov decision process (POMDP)
  - Black jack (only see public cards), Atari game with pixel observation,

Agent must construct its own state representation, as the beliefs of the environment state:

$$S_t^a = (P(S_t^e = s_1), ..., P(s_t^e = s_n))$$

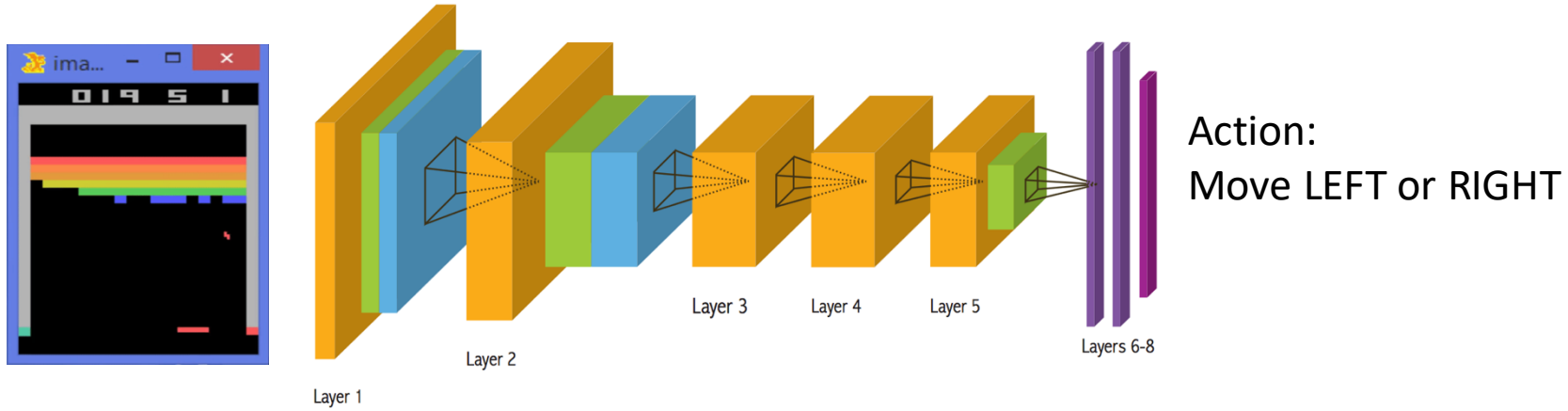
Major Components of an RL Agent

An RL agent may include one or more of these components:

- Policy: agent's behavior function
- Value function: how good is each state or action
- (World) Model: agent's state representation of the environment

Policy

- A policy is the agent's behavior model
- It is a map function from state/observation to action.
- Stochastic policy: Probabilistic sample $\pi(a|s) = P[A_t = a|S_t = s]$
- Deterministic policy: $a^* = \arg\max_a \pi(a|s)$



Policy

- $\pi(a|s)$: Given the current state s, which action to take

90% go straight, 10% stop, 0% turn left, 0% turn right

100% stop

50% go straight, 50% change to right lane

Value function

- Value function: expected discounted sum of future rewards under a particular policy π
- Discount factor weights immediate vs future rewards
- Used to quantify goodness/badness of states and actions

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

- Q-function (could be used to select among actions)

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right].$$

Value function V(s)

- V(s): Evaluating how good a state is

Urban highway

High v(s)

Rocky terrain

Low v(s)

Sandy terrain

Low v(s)

Value function V(s)

- $V^{\pi}(s)$: Evaluating how good a state is under a certain policy π

Urban highway

Could be low v(s) for a beginner driver

Rocky terrain

Sandy terrain

could be high v(s) for an expert driver

Action-value function Q(s, a)

- Evaluating how good an action a is in a state s

Q(s, go-straight) = 10

Q(s, turn-left) = -100

Q(s, turn-right) = 2?

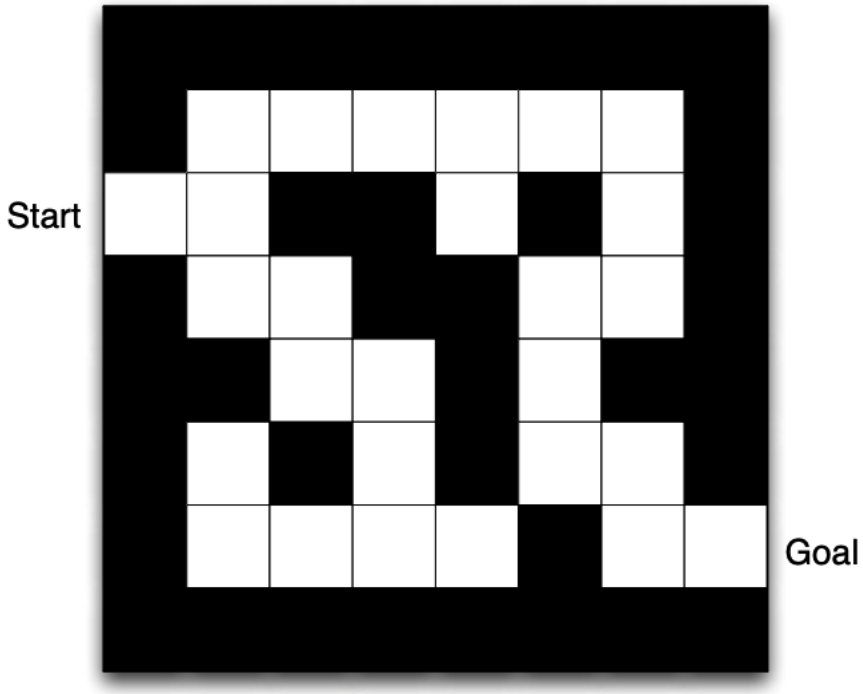
(World) Model

A model predicts what the environment will do next

Predict the next state: $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$

Predict the next reward: $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$

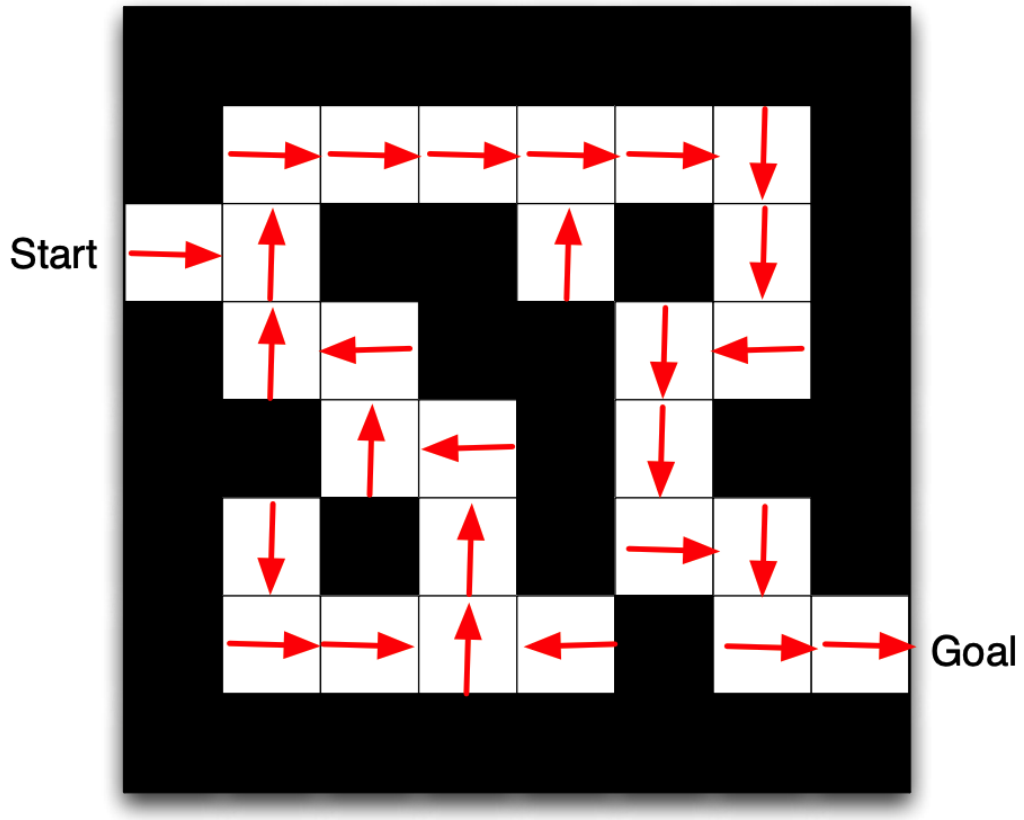
Maze Example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

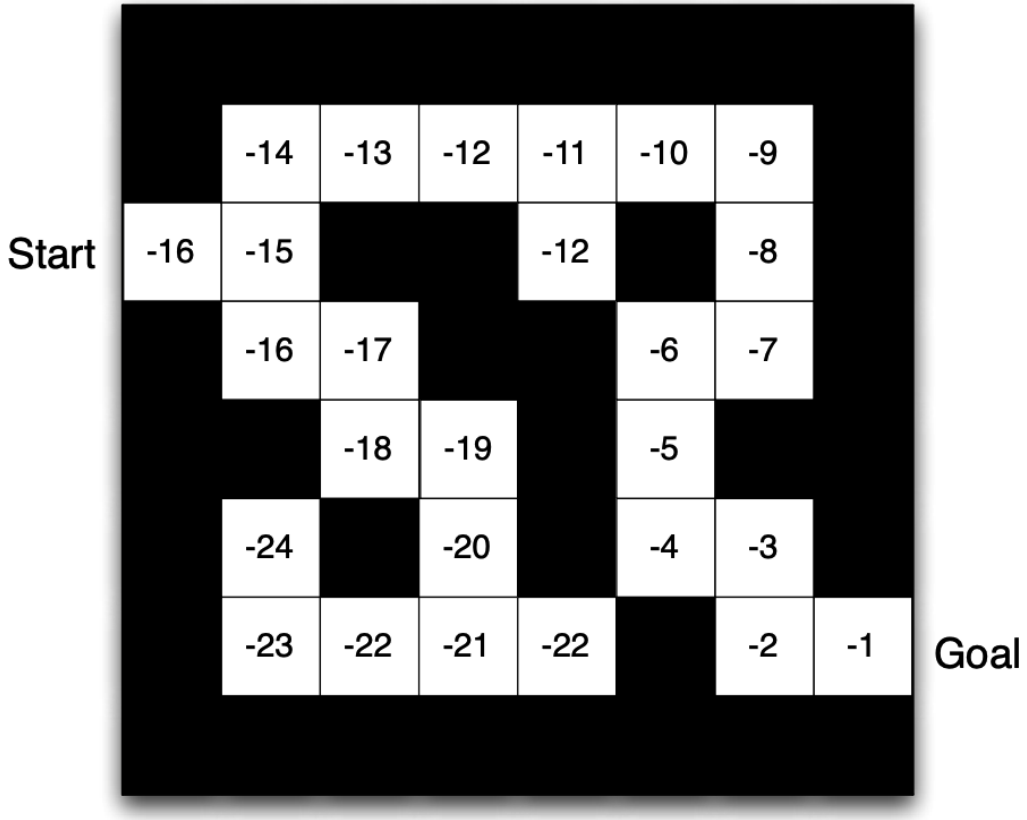
From David Silver Slide

Maze Example: Policy-based



- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value function-based



- Numbers represent value $v_\pi(s)$ of each state s

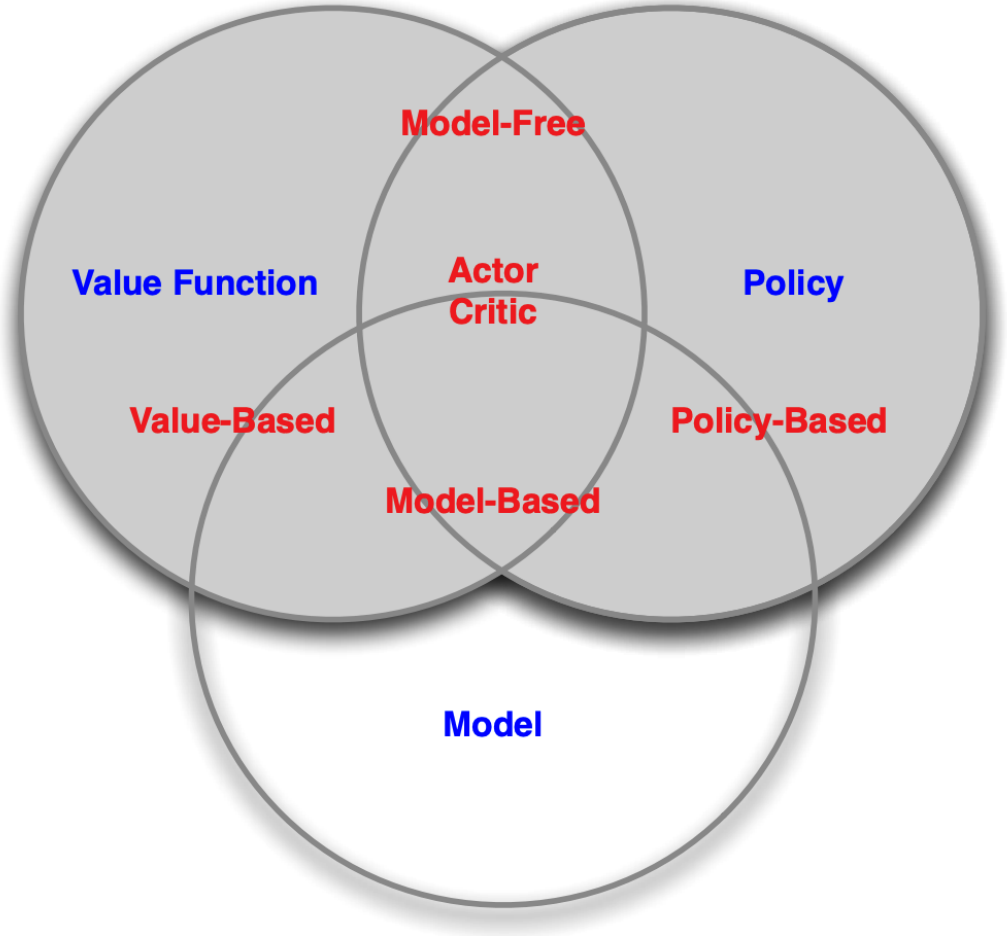
Types of RL Agents based on What the Agent Learns

- Value-based
  - Explicit: Value function
  - Implicit: Policy (can derive a policy from value function)
- Policy-based
  - Explicit: policy
  - No value function
- Actor-Critic:
  - Explicit: policy and value function

Types of RL Agents on if there is model

- Model-based
  - Explicit: model
  - May or may not have policy and/or value function
- Model-free
  - Explicit: value function and/or policy function
  - No model.

Types of RL Agents



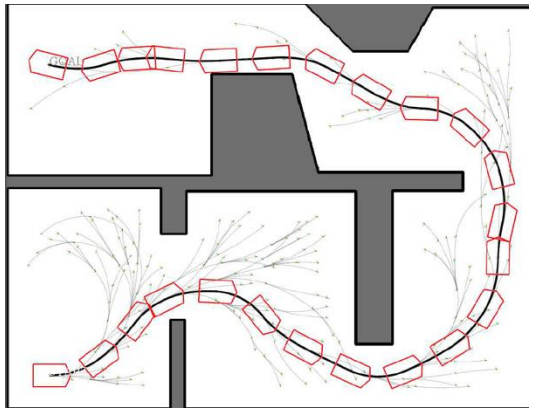
Credit: David Silver's slide

Comparison between Planning and Reinforcement Learning

- Planning
  - Given model about how the environment works.
  - Compute how to act to maximize expected reward without external interaction.
- Reinforcement learning
  - Agent doesn’t know how world works
  - Interacts with world to implicitly learn how world works
  - Agent improves policy (also involves planning)

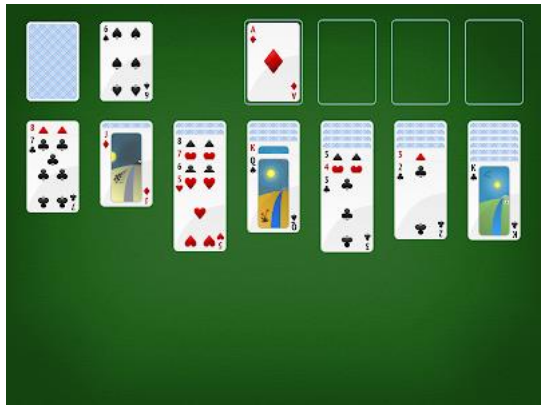
Planning

Path planning



Map is known
All the rules of the vehicle are known
Planning algorithms: dynamic programming, A* search, tree search, …

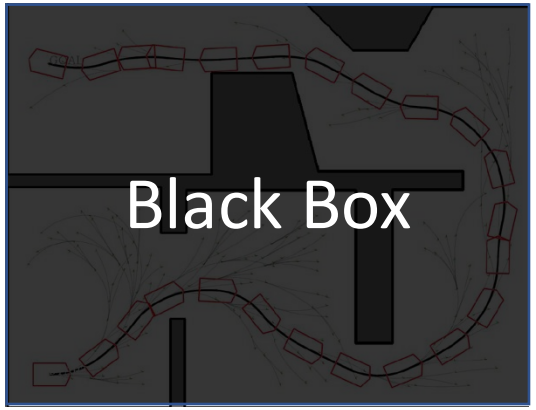
Patience



Rules of the game are known.
Planning algorithms: dynamic programming, tree search

Reinforcement Learning

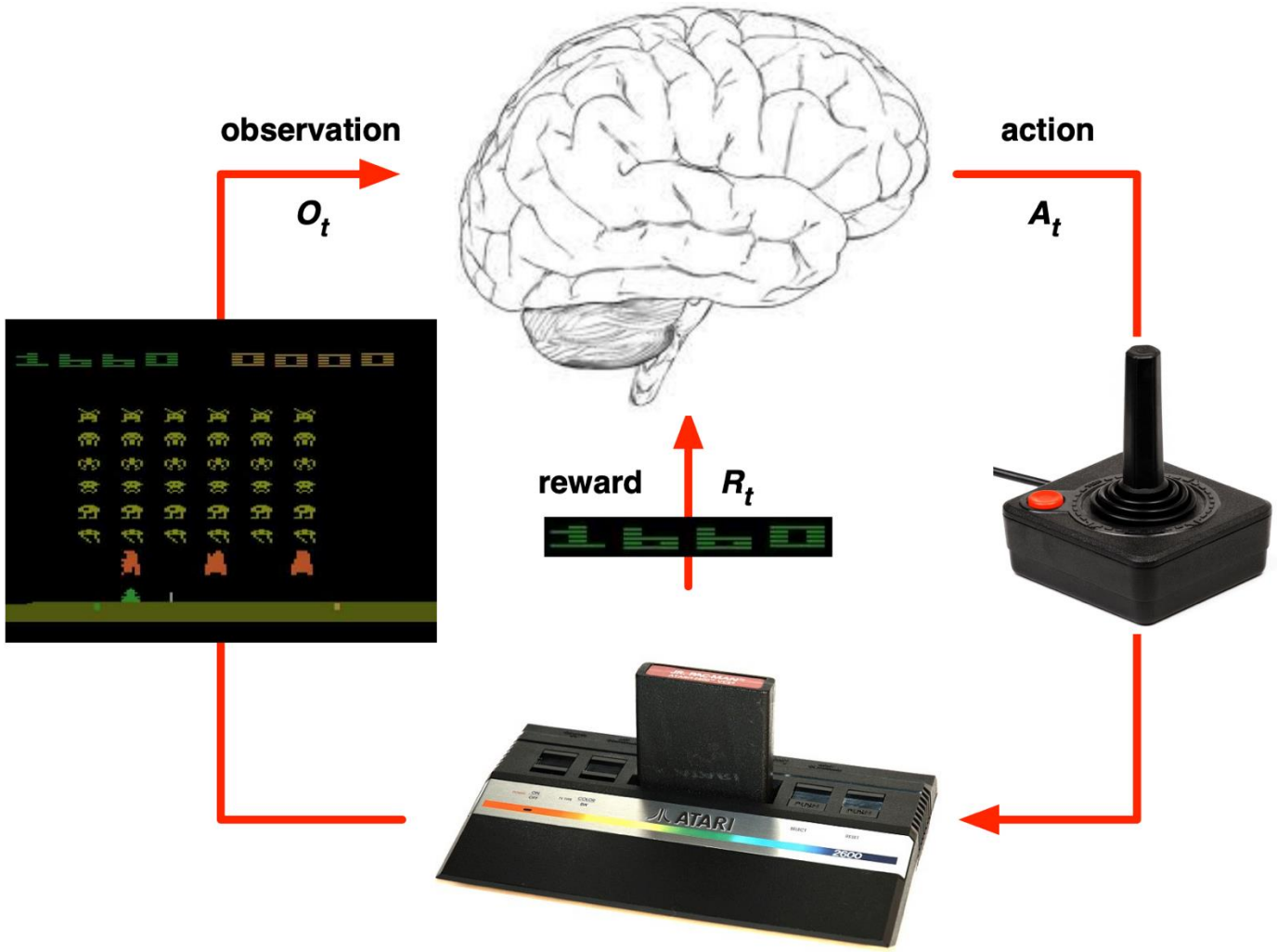
Path planning



Patience

- No rules or knowledge about the environment.
- Learn directly by taking actions and seeing what happens.
- Try to find a good policy over time that yields high reward.
- Planning is needed in inference or forward pass.

Atari Example: Reinforcement Learning



- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

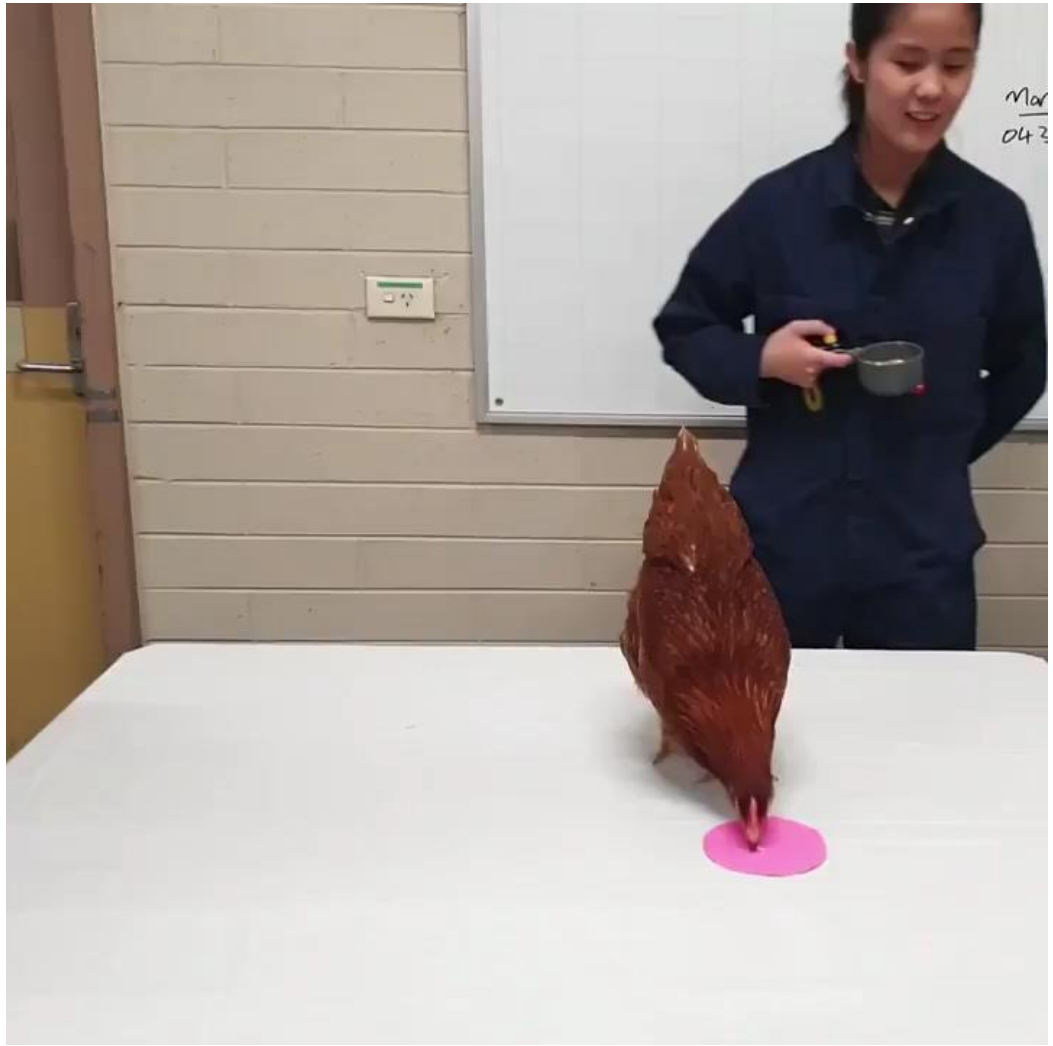
Exploration and Exploitation

- Agent only experiences what happens for the actions it tries!
- How should an RL agent balance its actions?
  - Exploration: trying new things that might enable the agent to make better decisions in the future
  - Exploitation: choosing actions that are expected to yield good reward given past experience
- Often there may be an exploration-exploitation tradeoff
  - May have to sacrifice reward in order to explore & learn about potentially better policy

Exploration and Exploitation

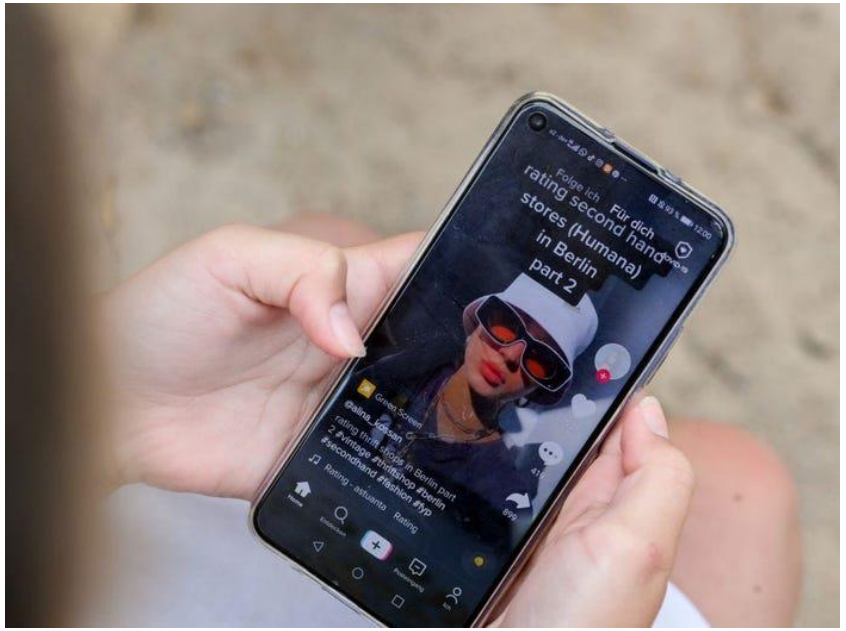
- Restaurant Selection
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Online Banner Advertisements
  - Exploitation: Show the most successful advert
  - Exploration: Show a different advert
- Oil Drilling
  - Exploitation: Drill at the best known location
  - Exploration: Drill at a new location
- Game Playing
  - Exploitation: Play the move you believe is
  - Exploration: play an experimental move

RL example: train a hen for visual recognition

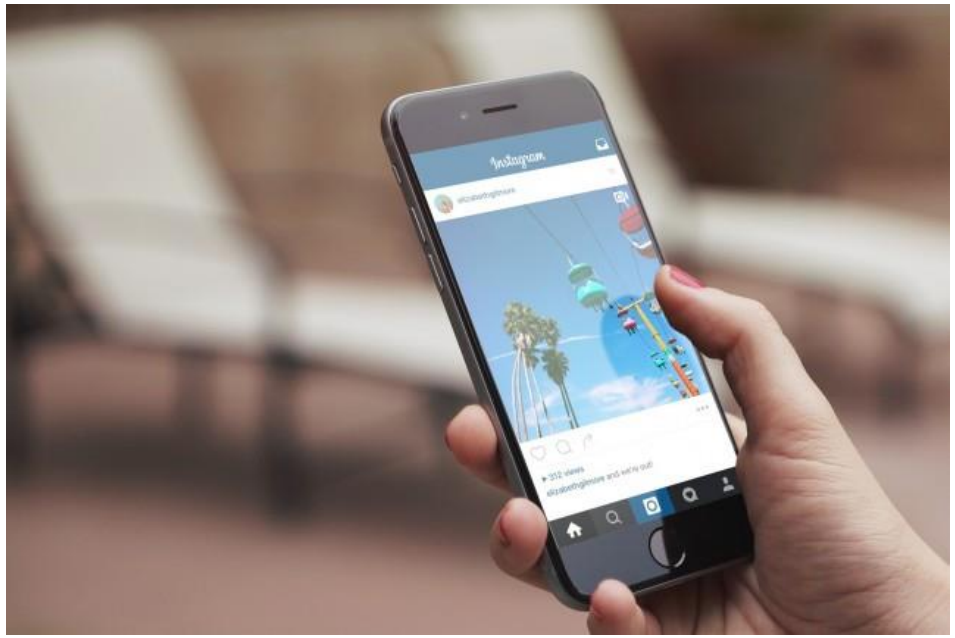


Probably you are like a hen being trained?

Social networking apps harness your reward behaviors



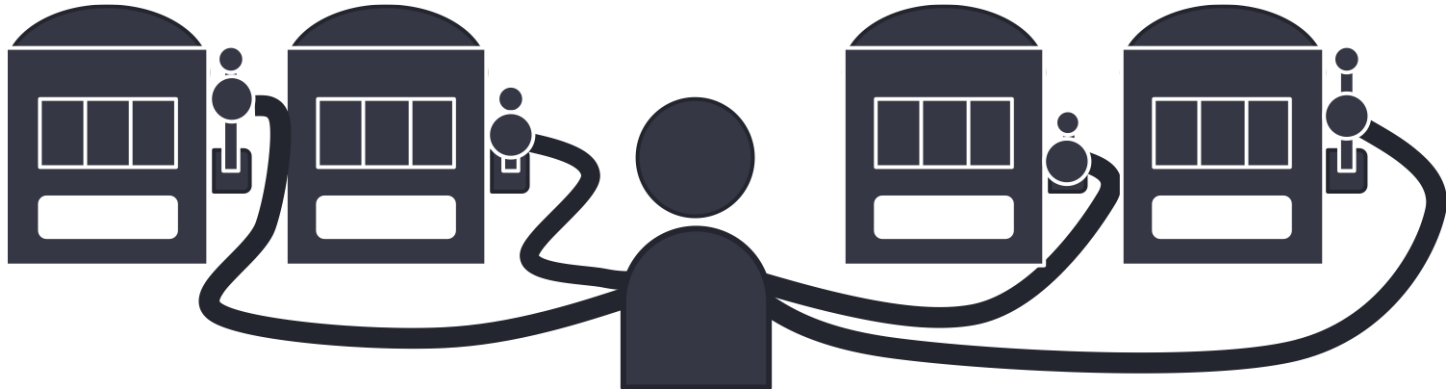
Tiktok



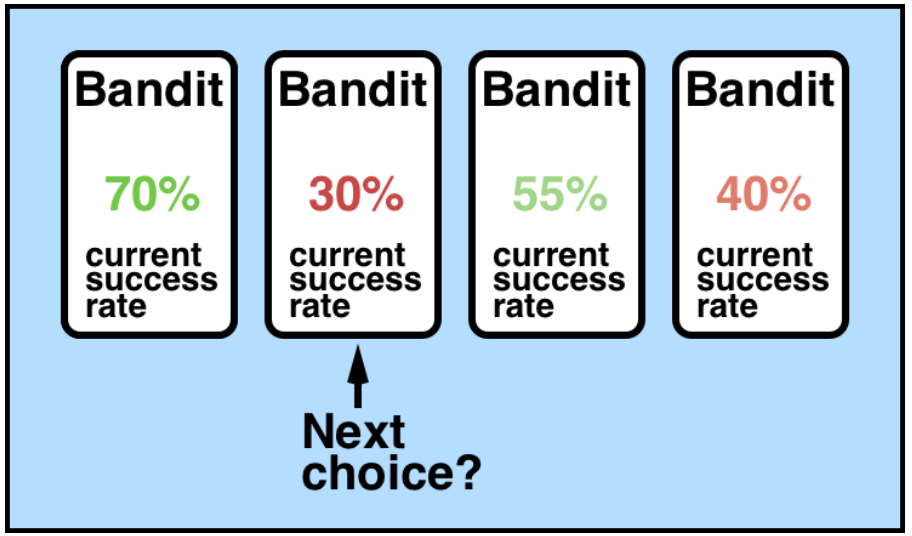
Instagram short video

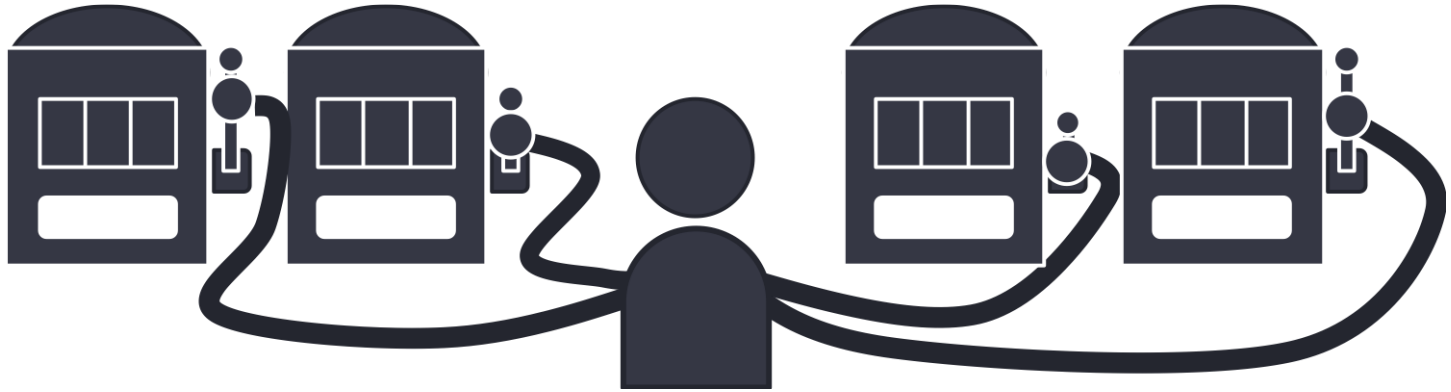
Multi-armed Bandit Example

Imagine you get into a casino in Las Vegas and you have **100 game chips**, there are **4 slot machines** to play with, how to maximize your return?
What's your thought?



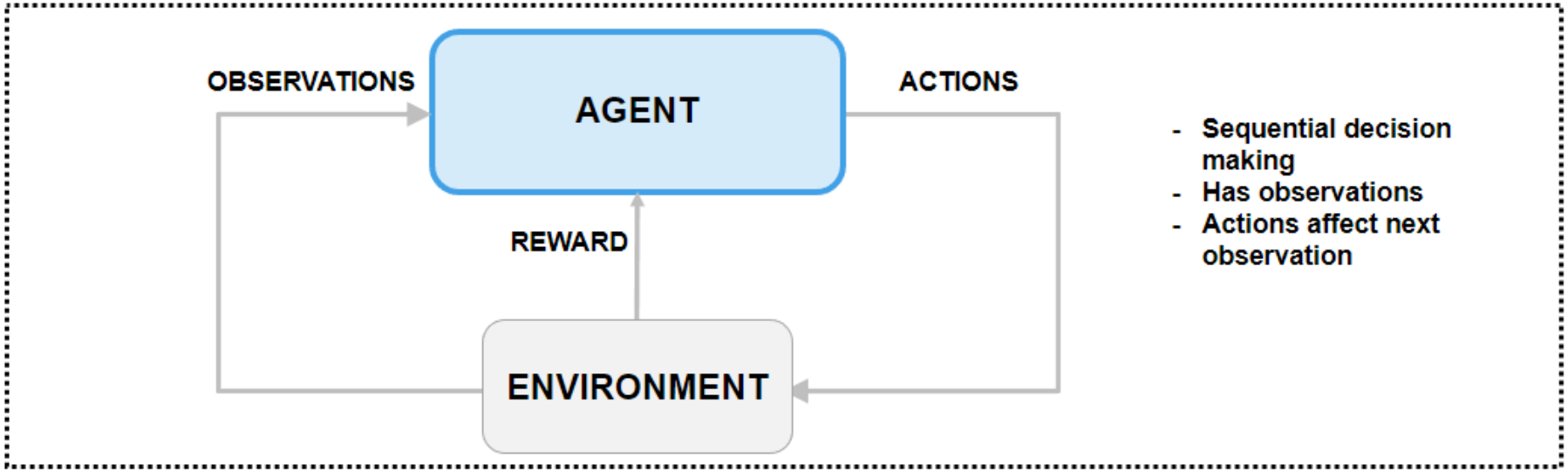
Multi-armed Bandit Example





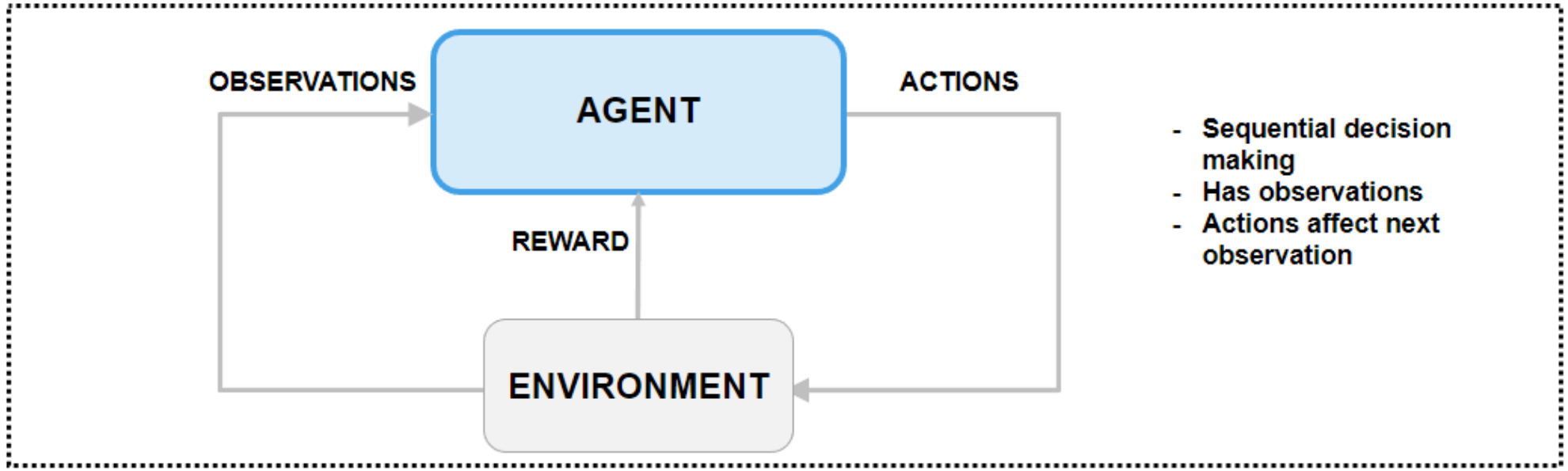
Multi-armed Bandit is One-step RL

Reinforcement Learning



Multi-Armed Bandits

Reinforcement Learning



- Sequential decision making
- Has observations
- Actions affect next observation

Multi-Armed Bandits

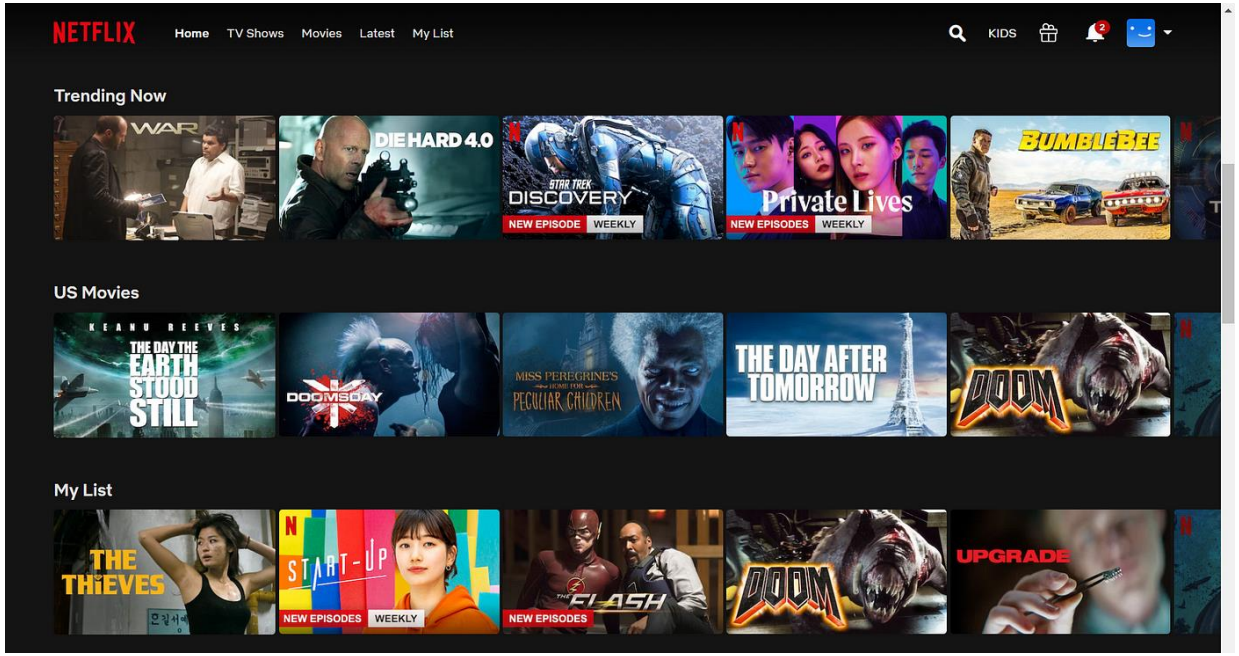
- No sequential decision making
- No observations

Contextual Bandits

- No sequential decision making
- Has observations/context
- Actions do not affect next observation

Contextual Bandits setting is very widely used

- Common applications of contextual bandit algorithms include content recommendations, dynamic pricing, and other adaptive experimentation use cases.



https://www.geteppo.com/blog/netflix-lyft-yahoo-contextual-bandits

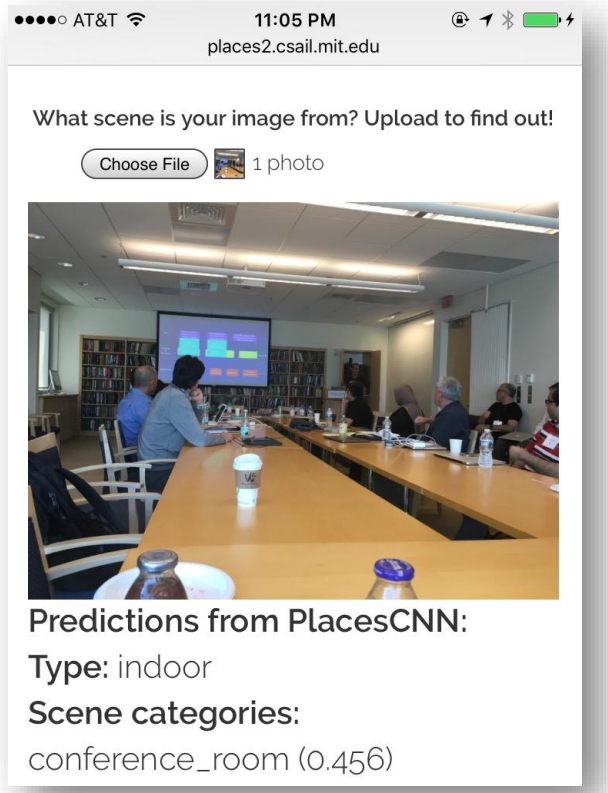
A short break

My Journey into Reinforcement Learning

- I am trained as a computer vision researcher

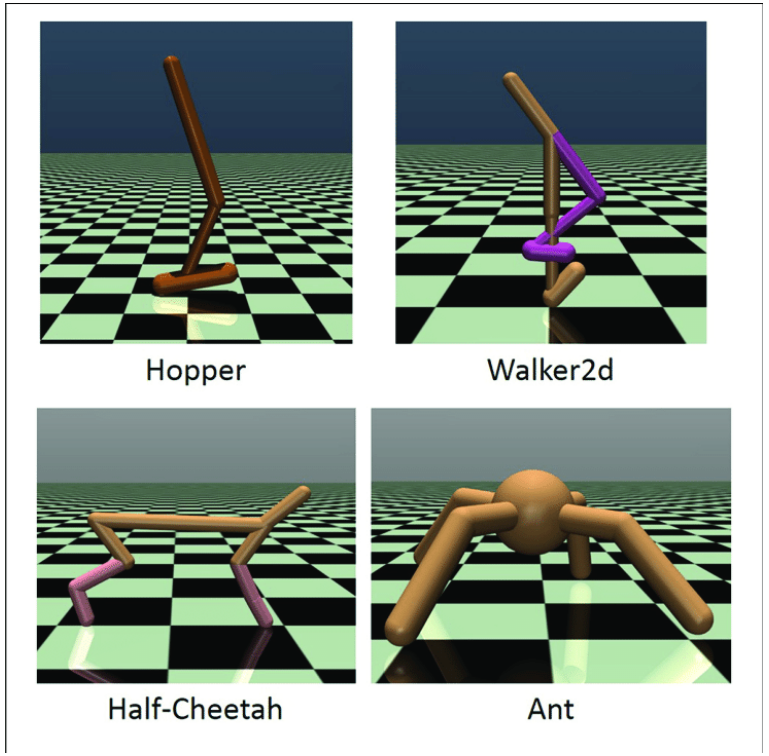
http://places2.csail.mit.edu

Scene recognition demo



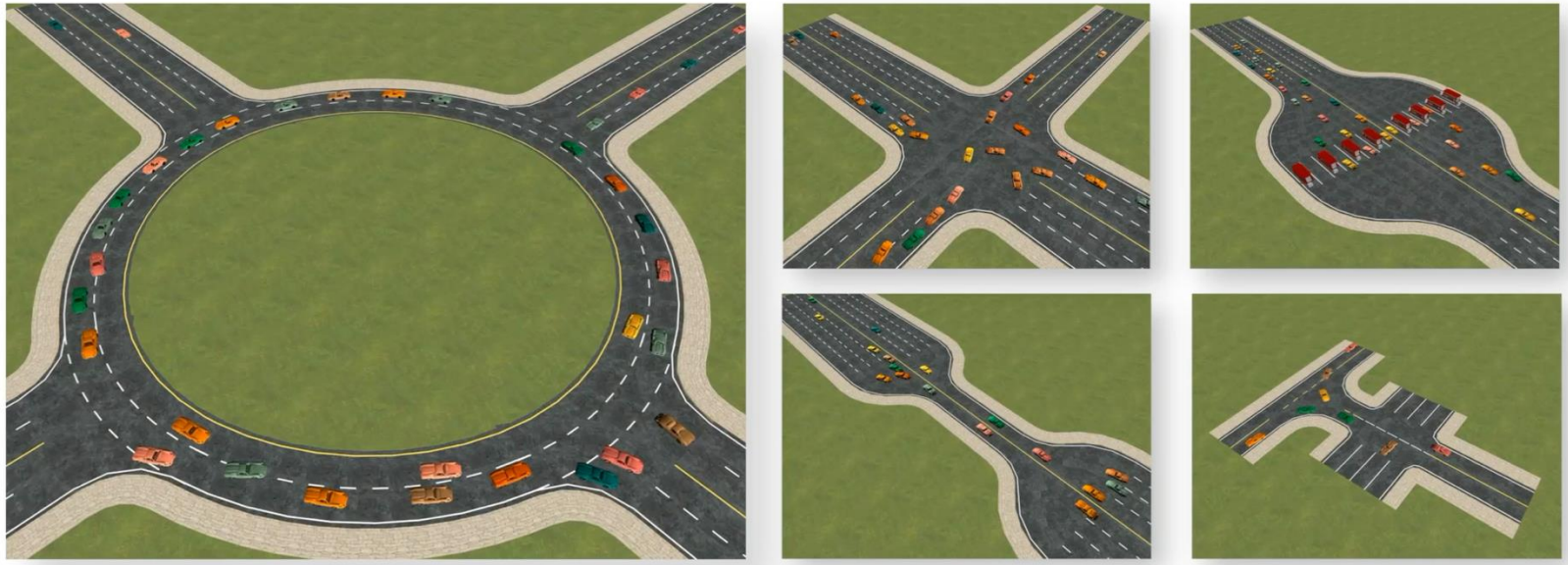
My Journey into Reinforcement Learning

- I believe it is crucial to combine perception and action
- In 2018, after I started looking into reinforcement learning, but a lot of rejections

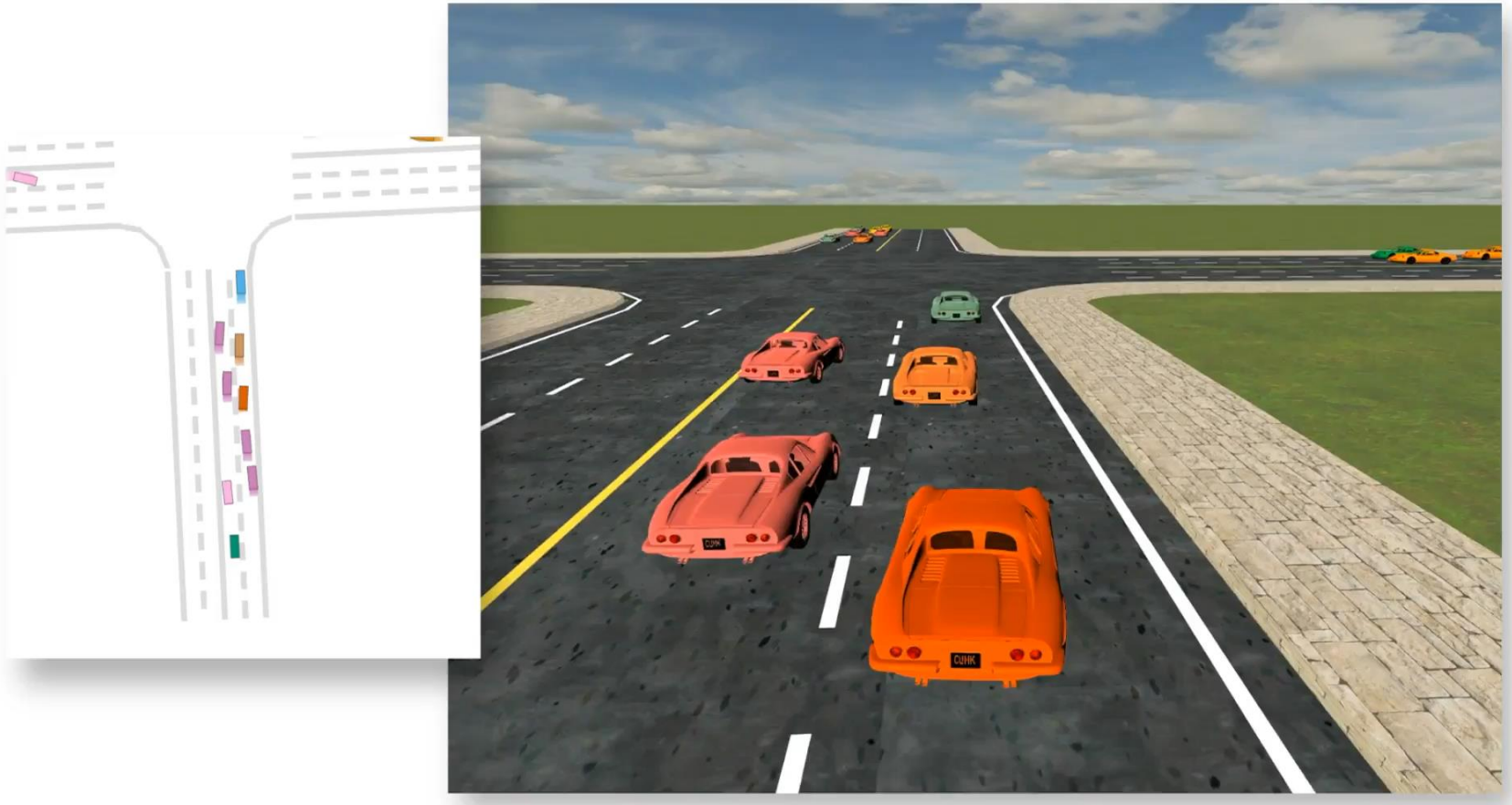


First successful RL project:

- Data-driven simulation: importing real-world data and generating new ones
- Lightweight (500 fps on a PC) with flexibility and extensibility
- Open-source at: https://github.com/metadriverse/metadrive
- Started in 2020, paper rejected 3 times (ICRA/IROS/NeurIPS DB)…almost gave up
- Now it has been downloaded more than 56K times



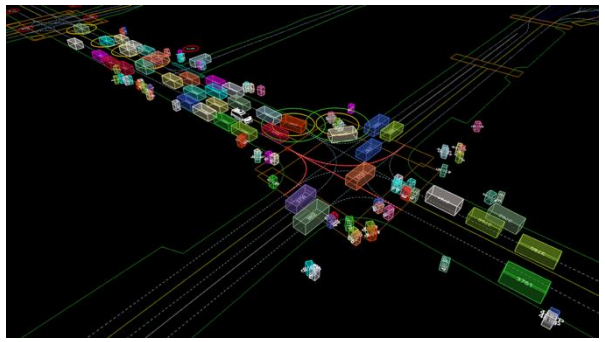
Multi-Agent Reinforcement Learning



https://decisionforce.github.io/CoPO/

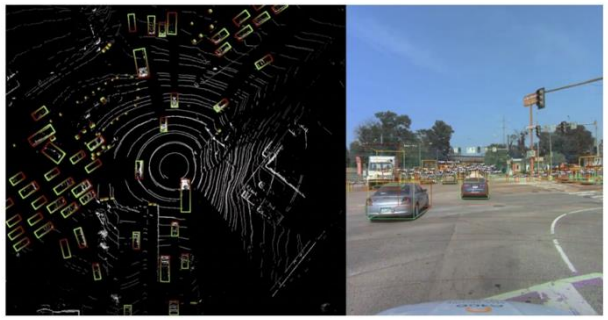
Bringing real-world data into simulator

Waymo Open Data

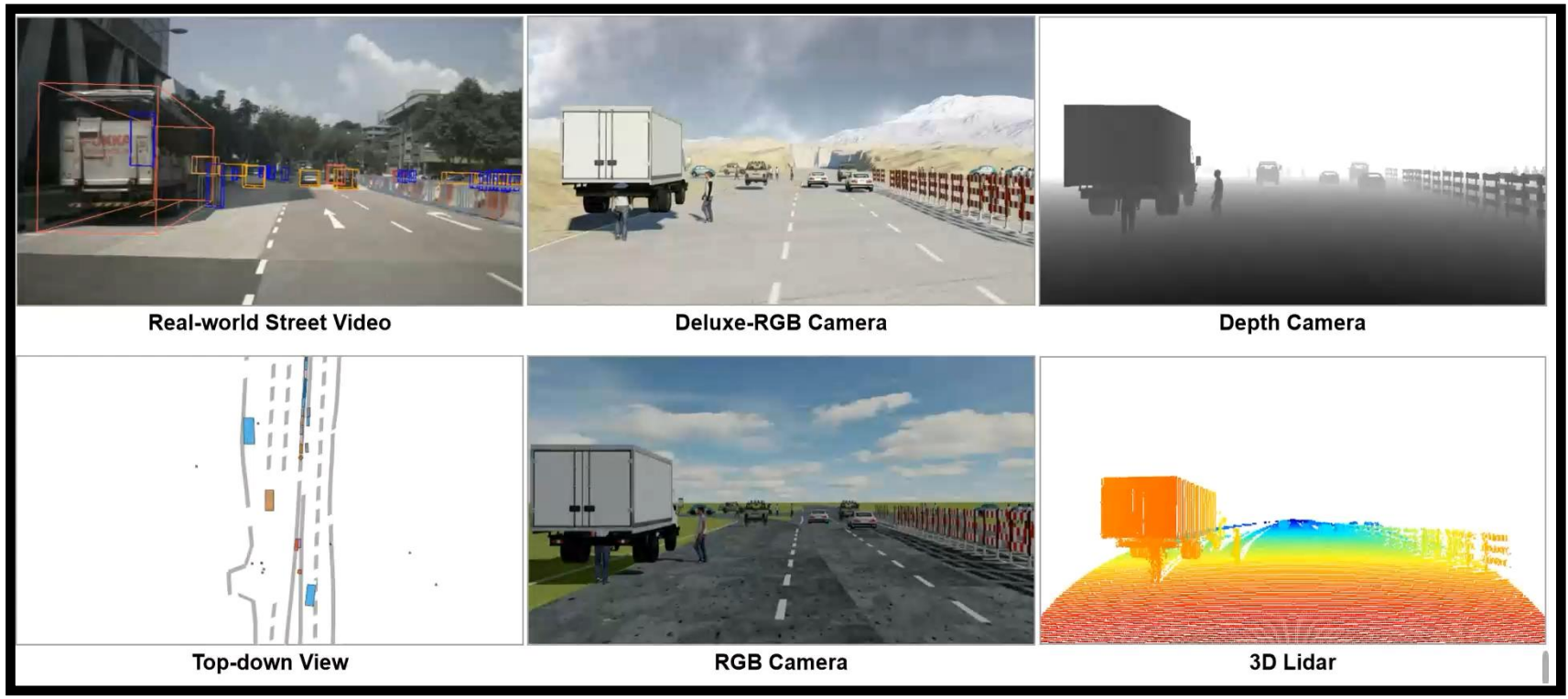


nuScenes & nuPlan

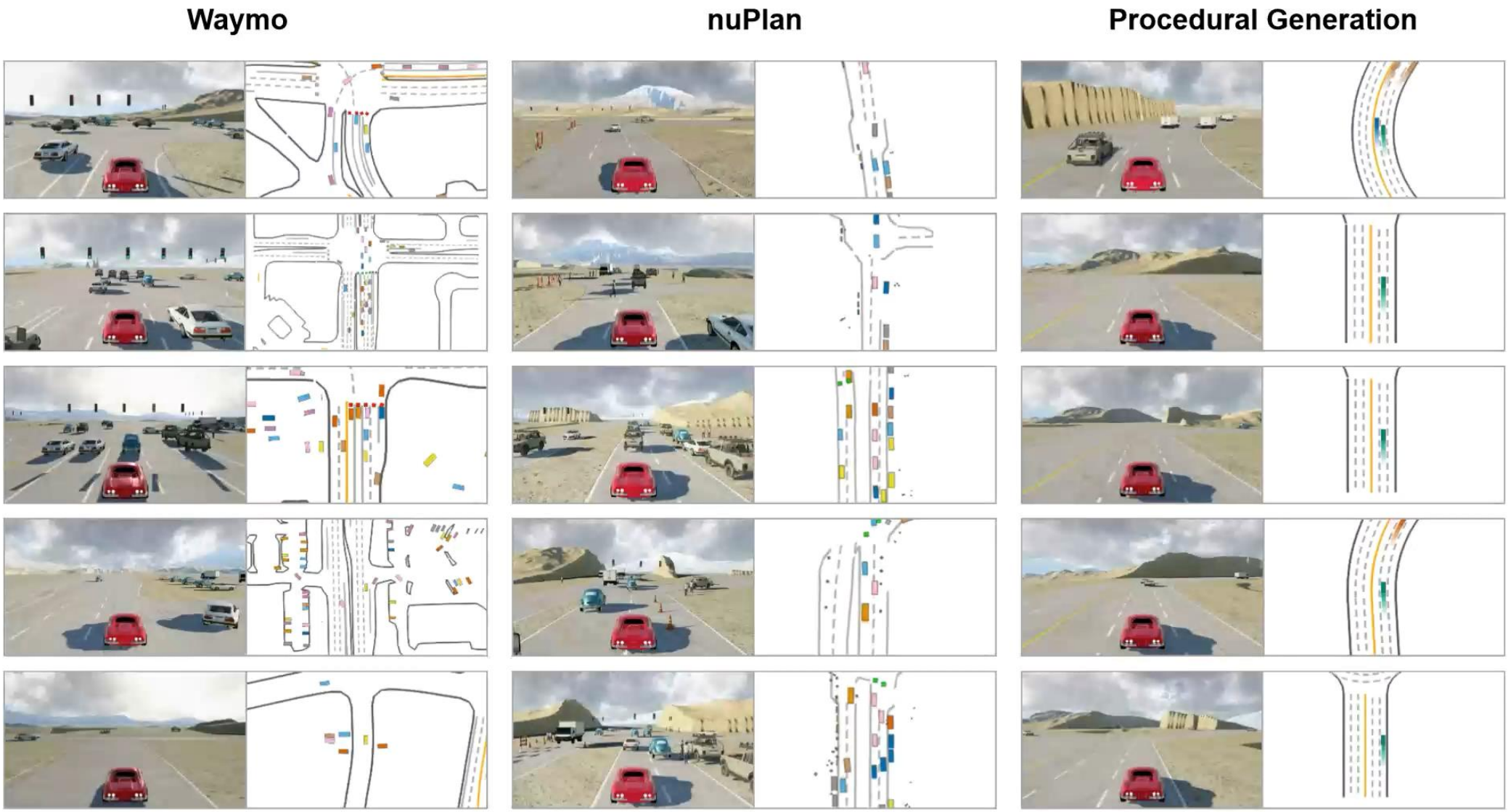
Argoverse 1 &2



Common data and simulation infrastructure (and fully open-source!)



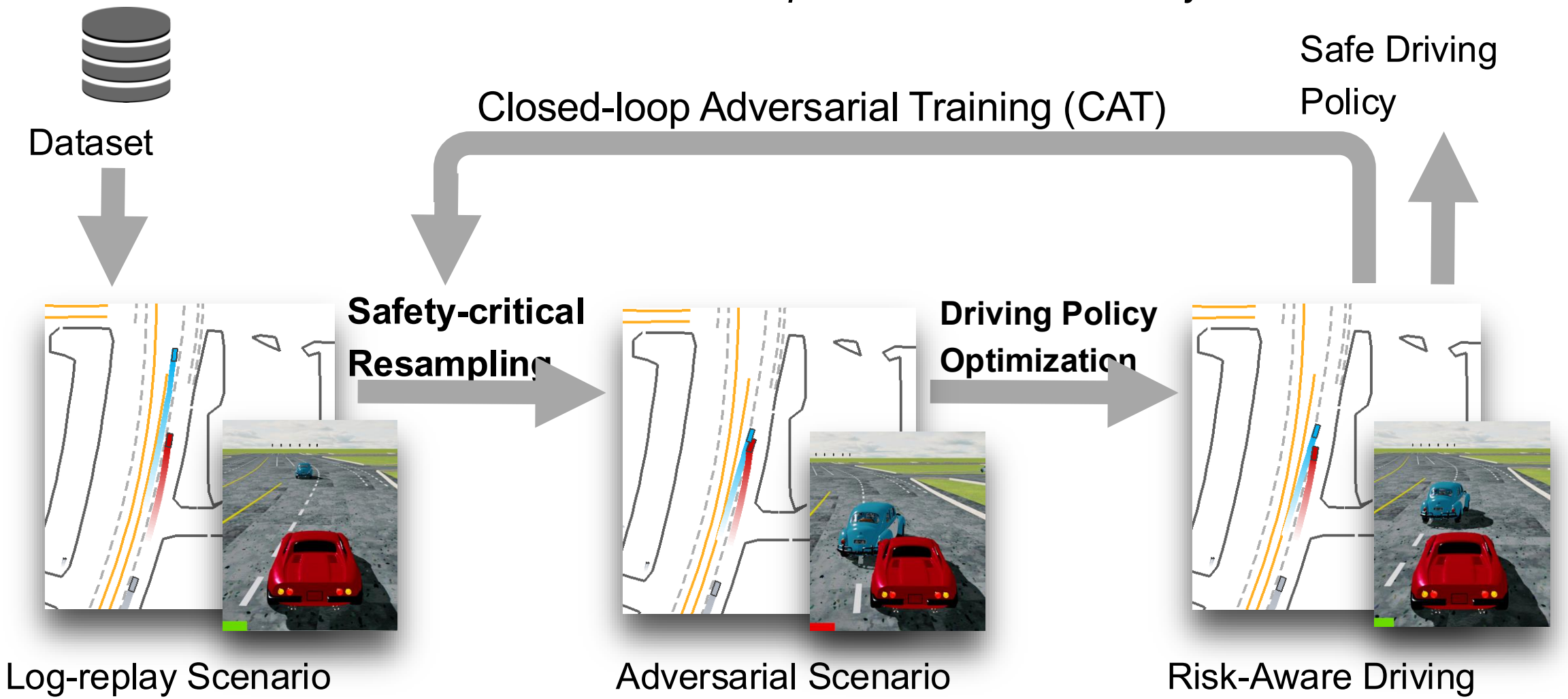
Bringing real-world data into simulator



NeurIPS'23. Data and code are available at https://github.com/metadriverse/scenarionet

Closed-Loop Adversarial Training for Safe Driving

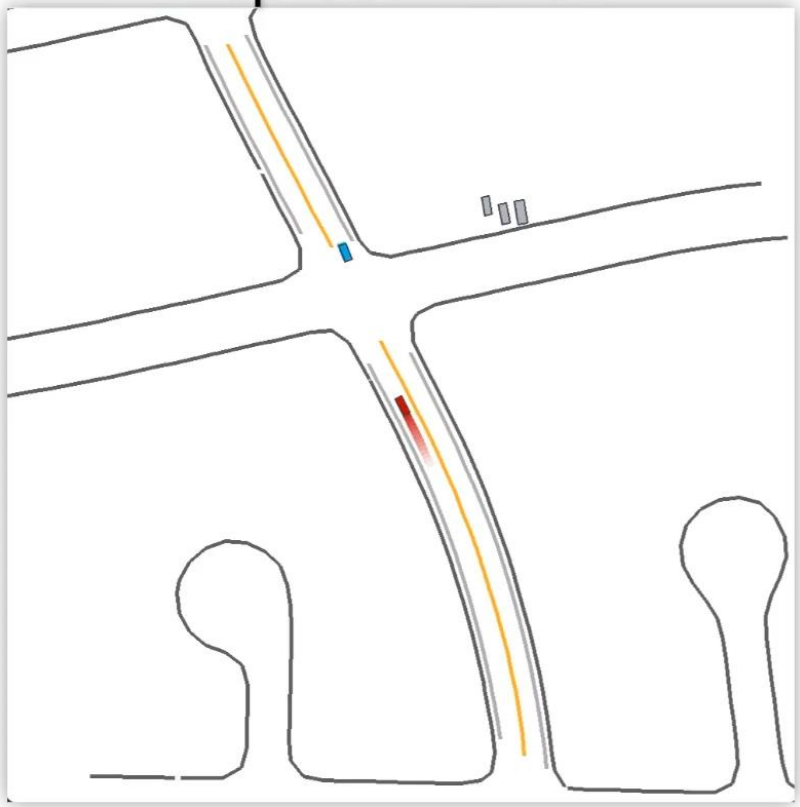
+32.5% improvement in the safety score!



CoRL'23. https://metadriverse.github.io/cat/

Generating Safety-Critical Scenarios

Top-down View

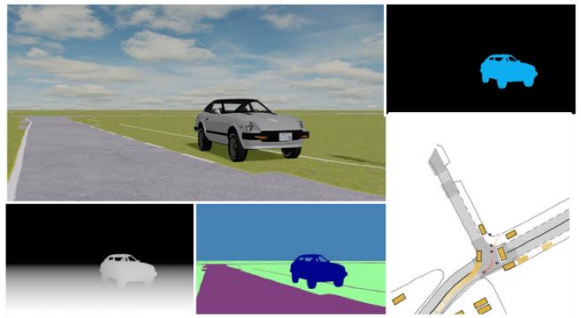


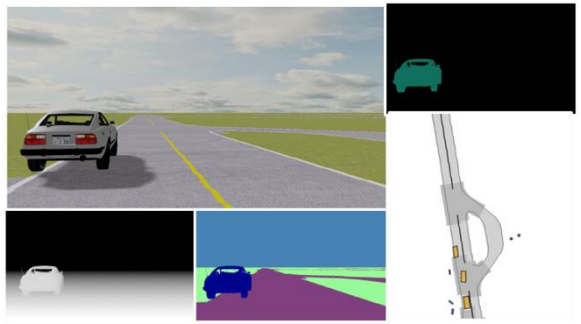
Front View

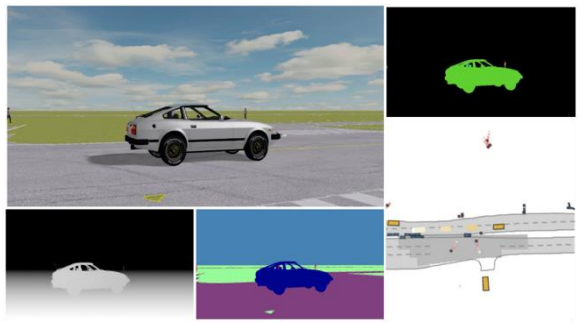
CoRL'23. https://metadriverse.github.io/cat/

GenAI for Rerendering the simulated scenes

Input







Output

London Desert Barcelona Miami

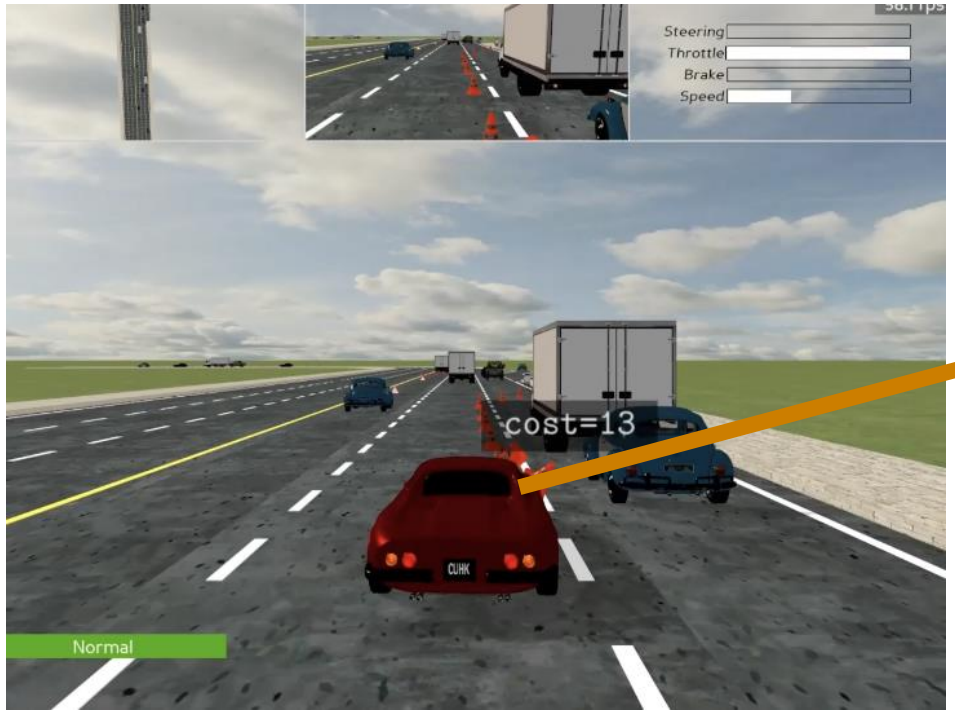
Storm Downtown Atlanta Berlin Switzerland

Big Trees Las Vegas Red sports car Small Town

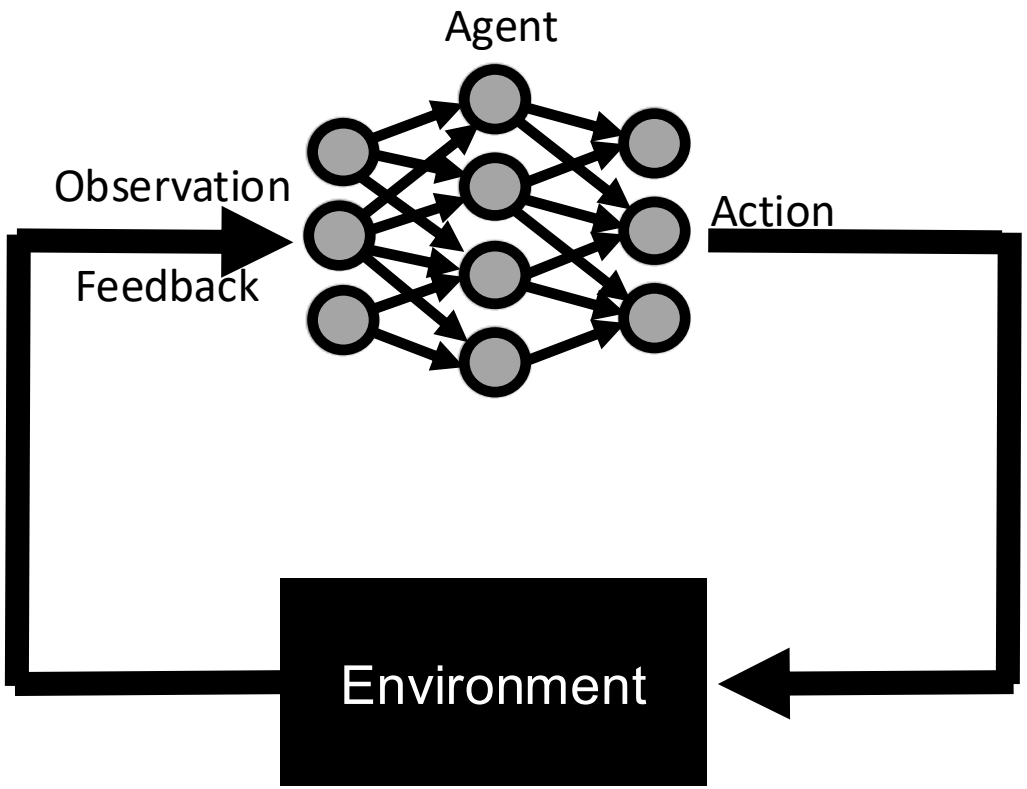
SimGen: Simulator-conditioned Driving Scene Generation. NeurIPS'24. https://metadriverse.github.io/simgen/

How we beginners learn to drive

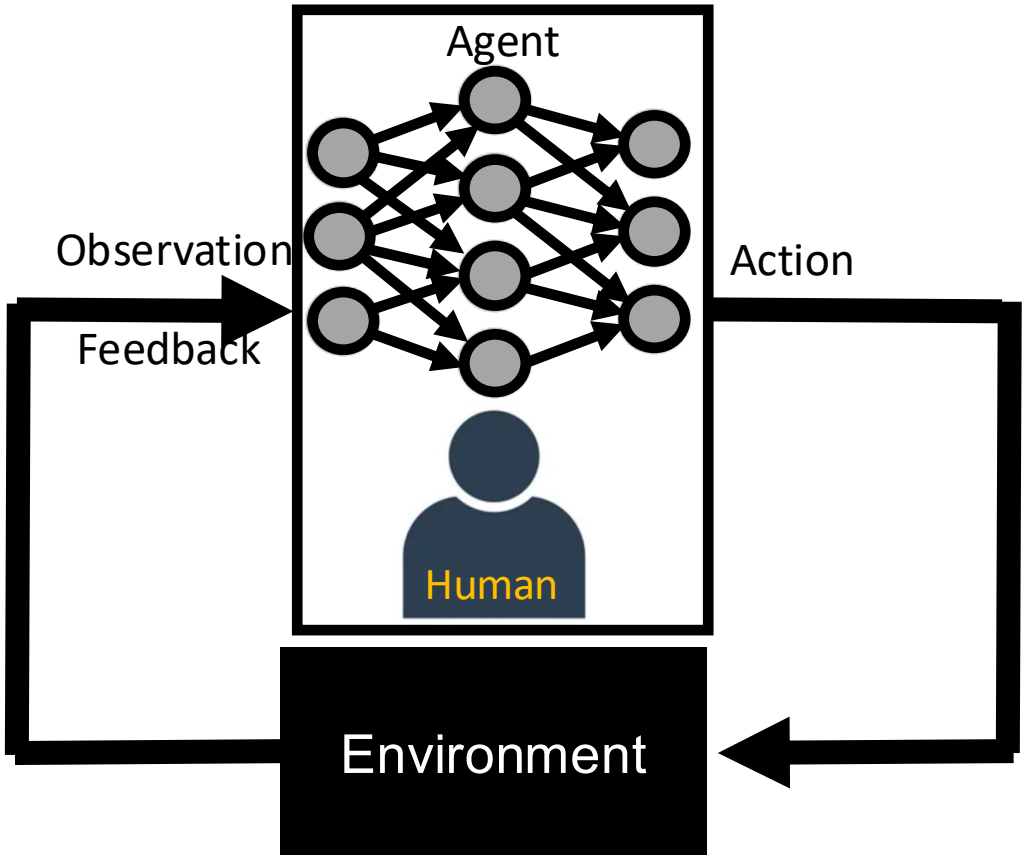
After getting the learner's permit, a beginner can be in the driver's seat and drive under the guidance of a people with License



Human-in-the-loop Learning



Human-in-the-loop Learning



Human will:

- Ensure training safety
- Improve learning efficiency
- Align agent behavior

Learning to Drive from Human Interventions

Cai et al. Predictive Preference Learning from Human Interventions, NeurIPS'25

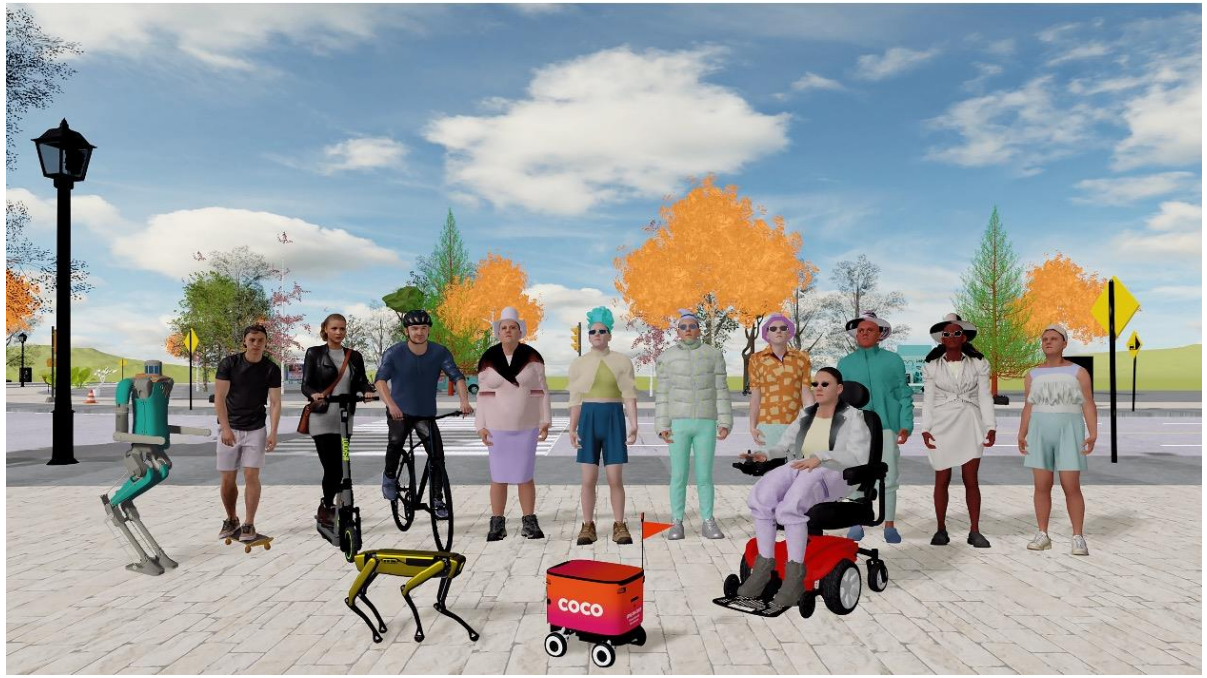
Peng, Mo, Duan, Li, Zhou. Learning from Active Human Involvement through Proxy Value Propagation. NeurIPS'23

Li*, Peng*, Zhou. Efficient Learning of Safe Driving Policy via Human-AI Copilot Optimization. ICLR'22

Simulating Diverse Urban Environments for Micromobility

On-going effort: https://metadriverse.github.io/metaurban/ . Led by Honglin He and Wayne Wu

Populating Simulated Urban Space with Dynamic Agents



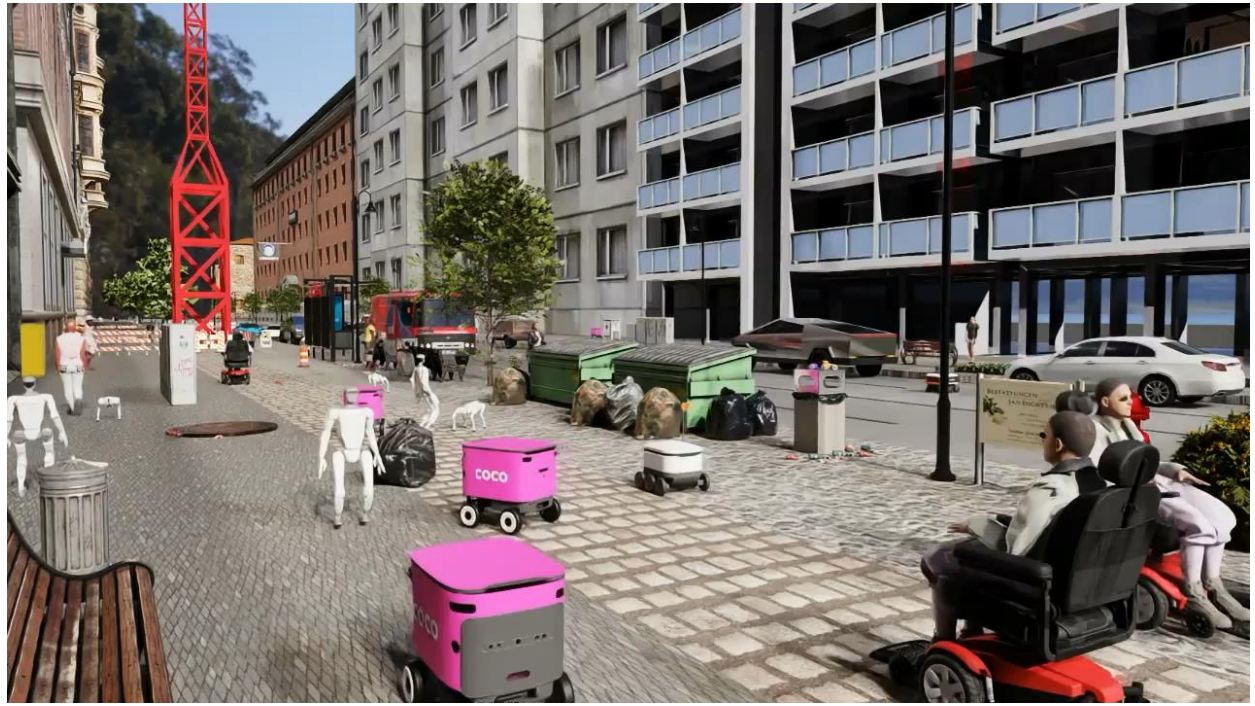
A full spectrum of mobile agents

Parade

On-going effort: https://metadriverse.github.io/metaurban/

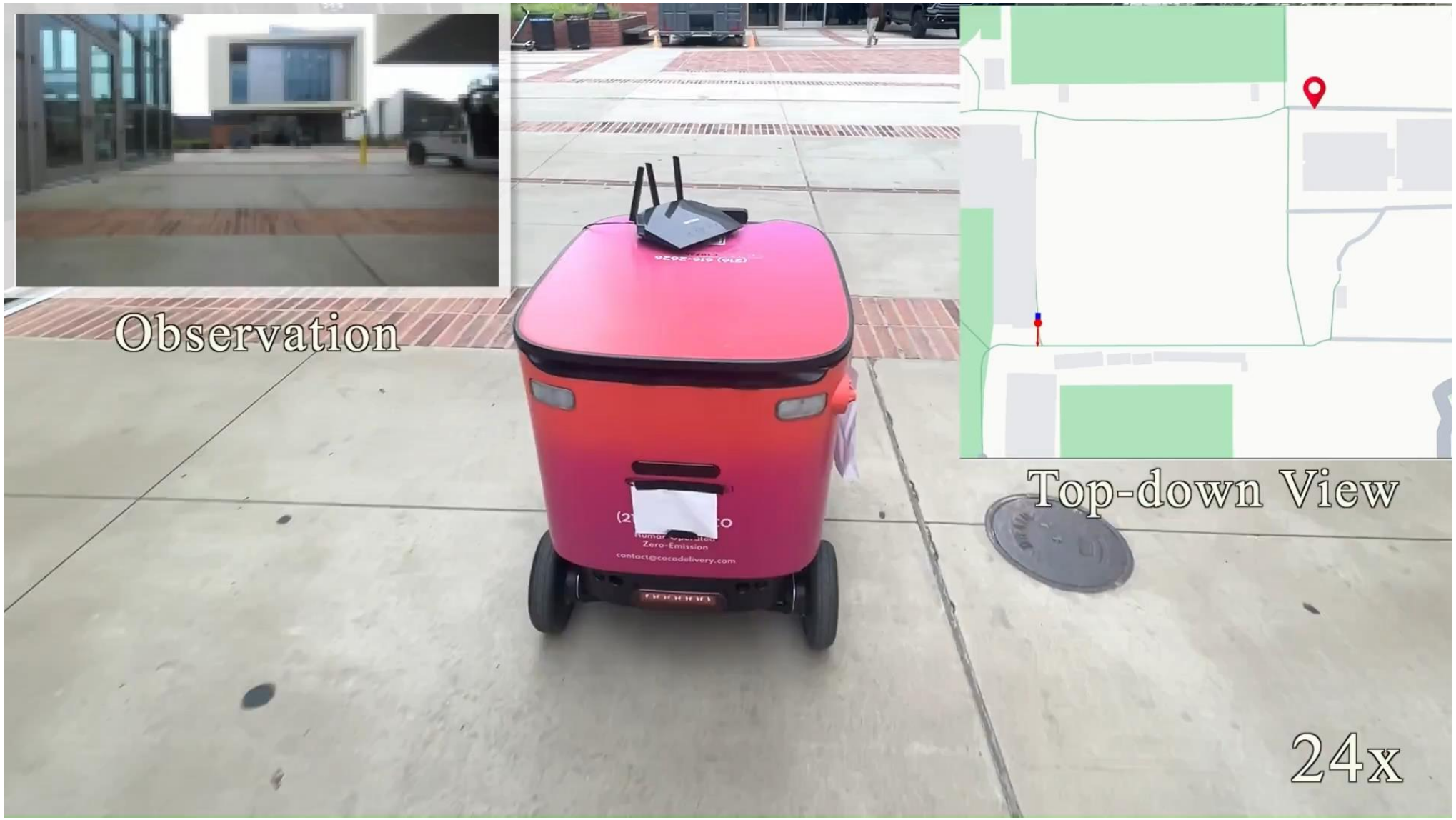
Reinforcement Learning for Social Navigation

More recent work of Urban-Sim based on NVIDIA Omniverse



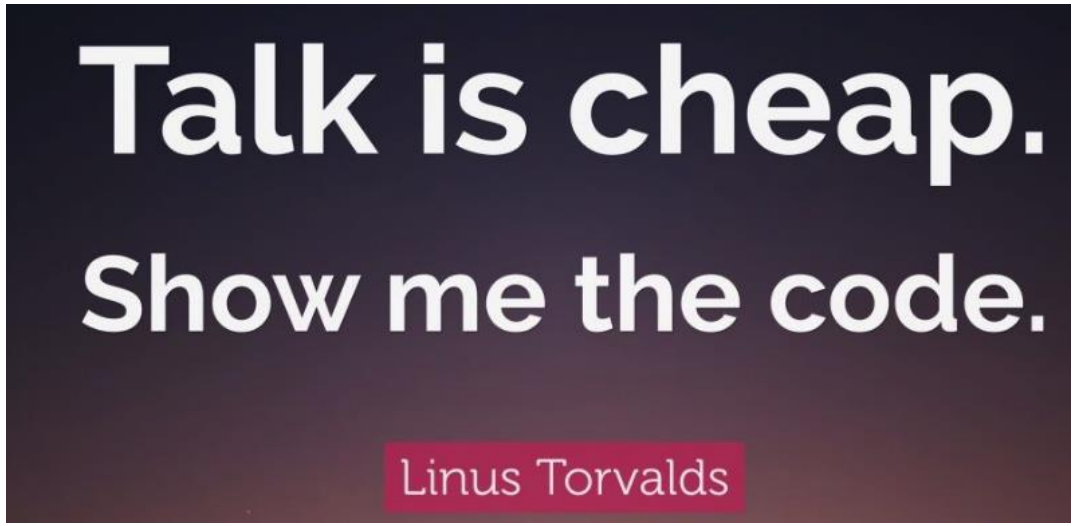
https://github.com/metadriverse/urban-sim

More recent work of urban navigation



https://metadriverse.github.io/s2e/. By Honglin He

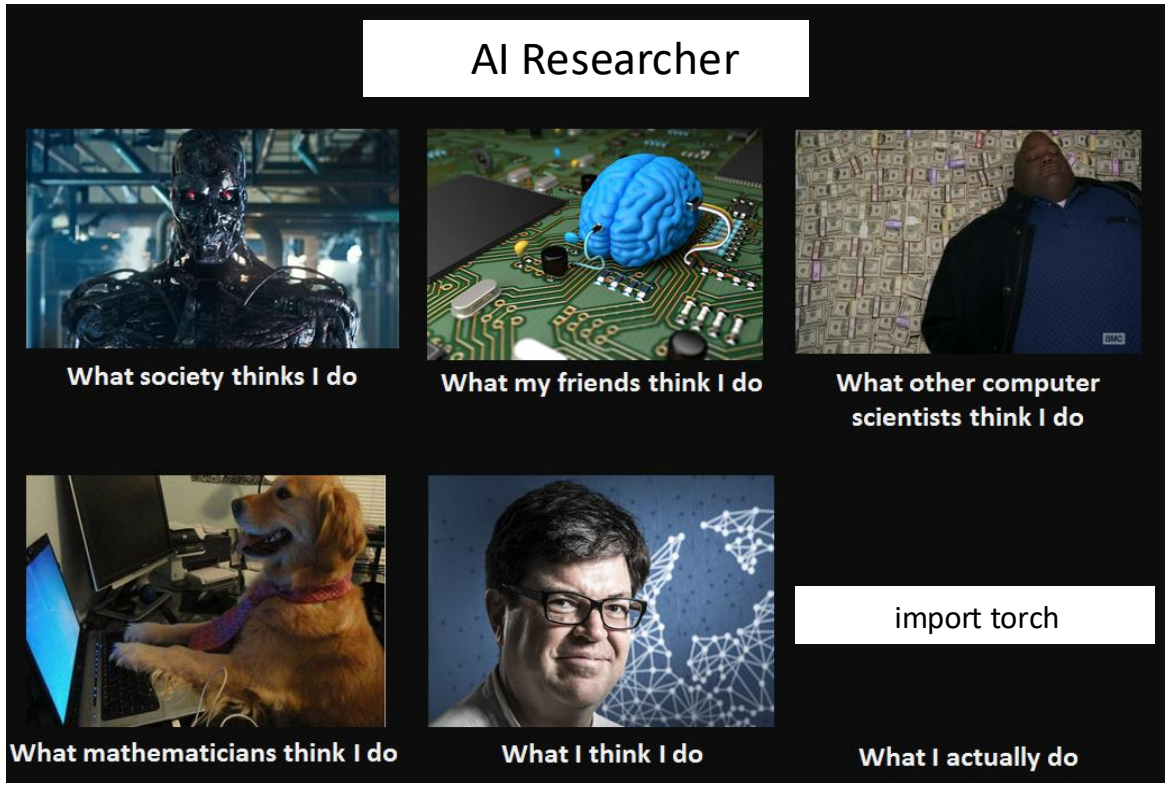
Coding with RL



- Getting hand dirty on reinforcement learning is very important
- Deep learning and AI become more and more empirical
- Trial and error approach to learn reinforcement learning

Coding

- Python coding
- Deep learning libraries: PyTorch or TensorFlow
- https://github.com/ucla-rlcourse/RLexample

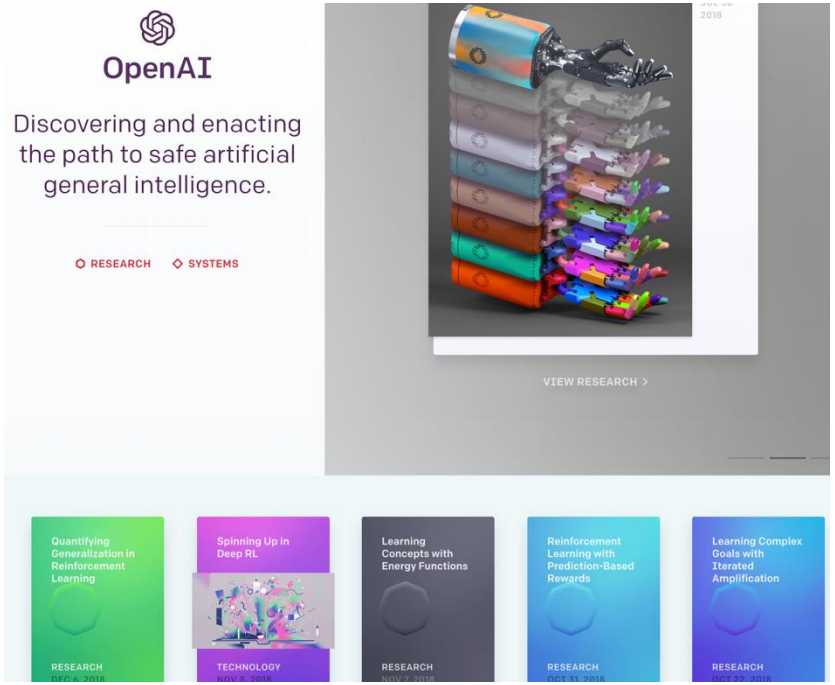


Reinventing Wheels?

No. Start with existing libraries and pay more attentions to the specific algorithms

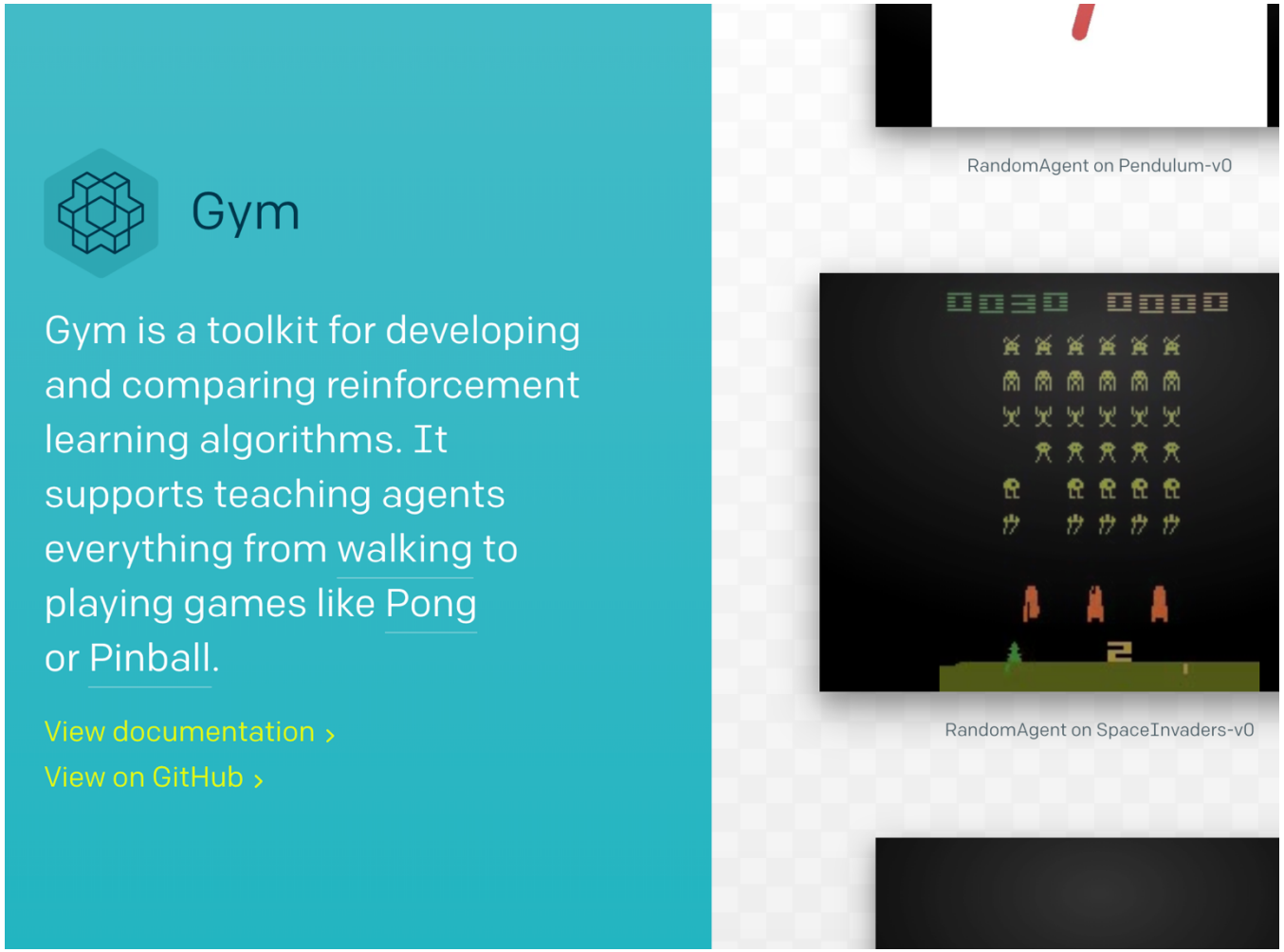
OpenAI: (used to be) specialized in Reinforcement Learning

- https://openai.com/
- OpenAI is (used to be) a non-profit AI research company, discovering and enacting the path to safe artificial general intelligence (AGI).



OpenAI gym library

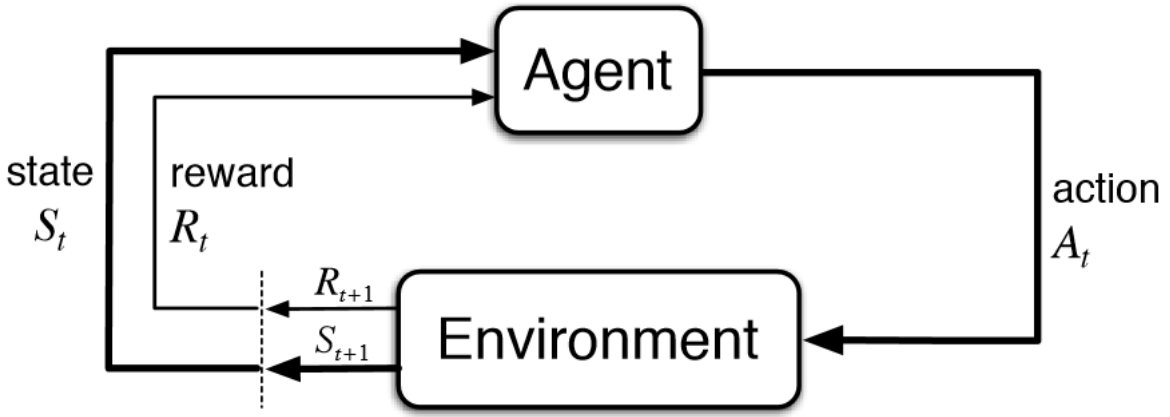
https://gym.openai.com/



https://github.com/openai/retro

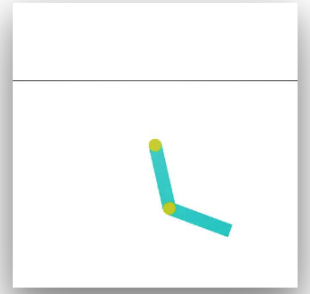
We're releasing the full version of Gym Retro, a platform for reinforcement learning research on games. This brings our publicly-released game count from around 70 Atari games and 30 Sega games to over 1,000 games across a variety of backing emulators. We're also releasing the tool we use to add new games to the platform.

Algorithmic interface of reinforcement learning

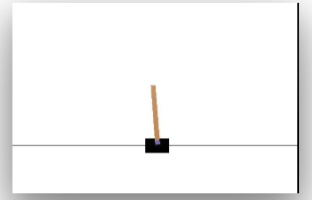



```
import gym
env = gym.make("Taxi-v2")
observation = env.reset()
agent = load_agent()
for step in range(100):
    action = agent(observation)
    observation, reward, done, info = env.step(action)
```

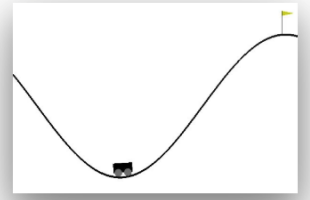
Classic Control Problems



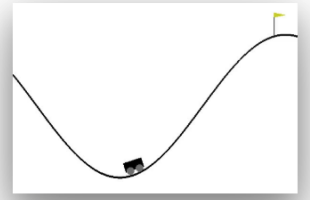
Acrobot-v1
Swing up a two-link robot.



CartPole-v1
Balance a pole on a cart.



MountainCar-v0
Drive up a big hill.

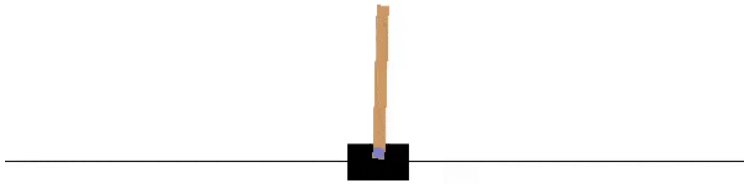


MountainCarContinuous-v0
Drive up a big hill with continuous control.

Pendulum-v0
Swing up a pendulum.

https://gym.openai.com/envs/#classic_control

Example of CartPole-v0



Actions

Type: Discrete(2)

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

Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Reward

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

Episode Termination

1. Pole Angle is more than ±12°
2. Cart Position is more than ±2.4 (center of the cart reaches the edge of the display)
3. Episode length is greater than 200

https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py

Example code

```
import gym
env = gym.make('CartPole-v0’)
env.reset()
env.render() # display the rendered scene
action = env.action_space.sample()
observation, reward, done, info = env.step(action)
```

Example code: Random Agent

python my_random_agent.py CartPole-v0

python my_random_agent.py Pong-ram-v0

python my_random_agent.py Breakout-v0

What is the difference in the format of the observations?

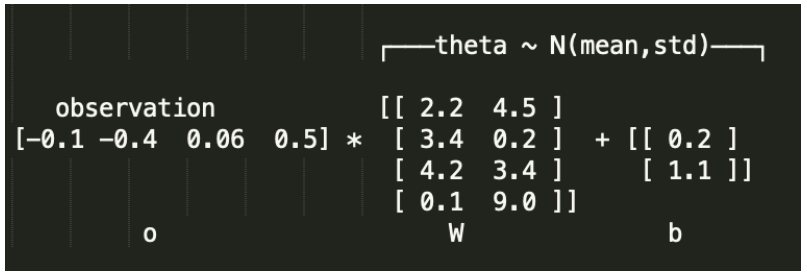
Example code: Naïve learnable RL agent

python my_random_agent.py CartPole-v0

python my_random_agent.py Acrobot-v1

python my_learning_agent.py CartPole-v0

python my_learning_agent.py Acrobot-v1



P_a = oW+b

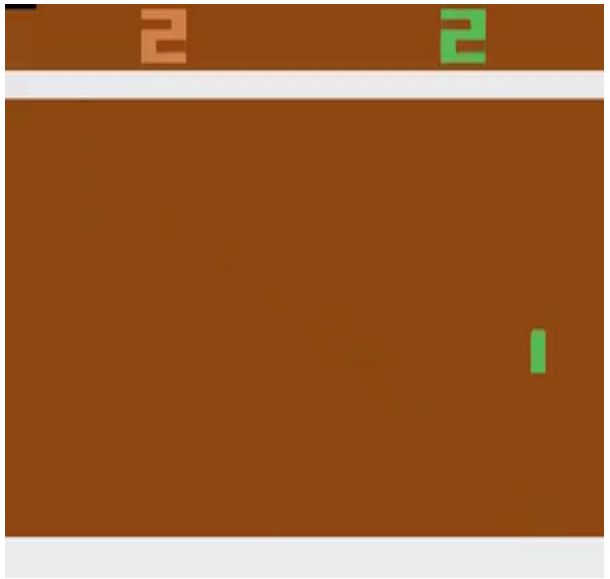
What is the algorithm?

Cross Entropy method (CEM)
https://gist.github.com/kashif/5dfa12d80402c559e060d567ea352c06

Deep Reinforcement Learning Example

- Pong example

```
import gym
env = gym.make('Pong-v0')
env.reset()
env.render() # display the rendered scene
```



```
python my_random_agent.py Pong-v0
```

Deep Reinforcement Learning Example

- Pong example

python pg-pong.py

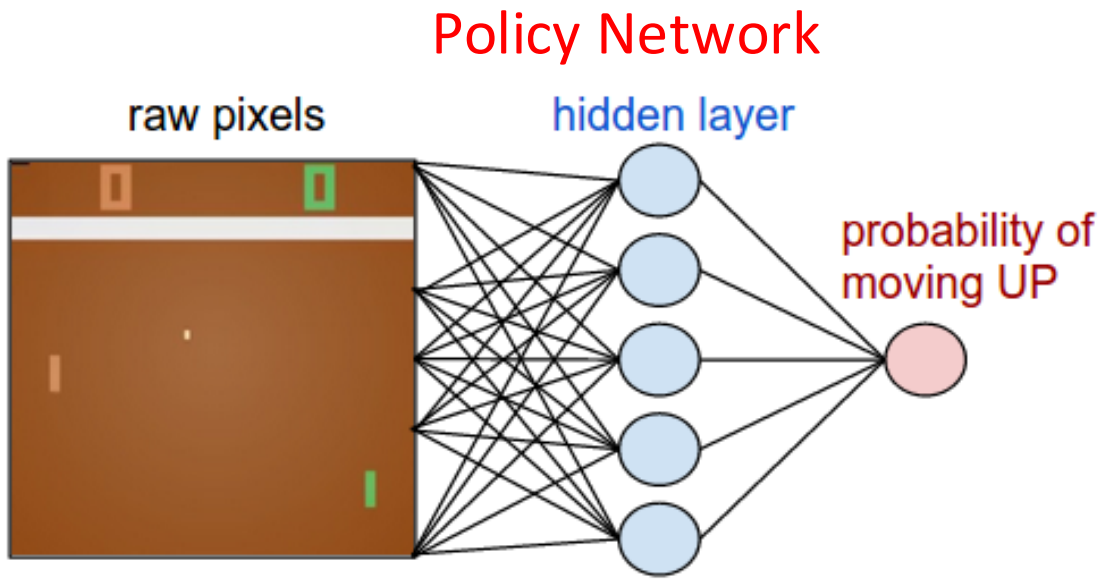
Loading weight: pong_bolei.p (model trained over night)

Deep Reinforcement Learning Example

- Look deeper into the code

```
observation = env.reset()

cur_x = prepro(observation)
x = cur_x - prev_x
prev_x = cur_x
aprob, h = policy_forward(x)
```



Randomized action:

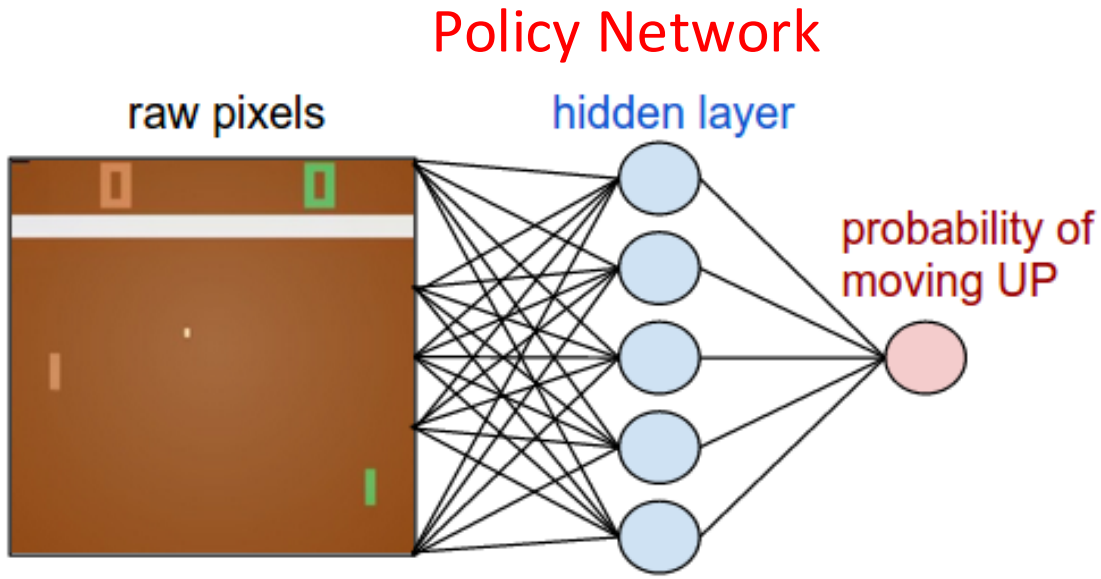
```
action = 2 if np.random.uniform() < aprob else 3 # roll the dice!
```

http://karpathy.github.io/2016/05/31/rl/

Deep Reinforcement Learning Example

- Look deeper into the code

```
h = np.dot(W1, x)
h[h<0] = 0 # ReLU nonlinearity: threshold at zero logp = np.dot(W2, h) # compute log probability of going up
p = 1.0 / (1.0 + np.exp(-logp)) # sigmoid function (gives probability of going up)
```



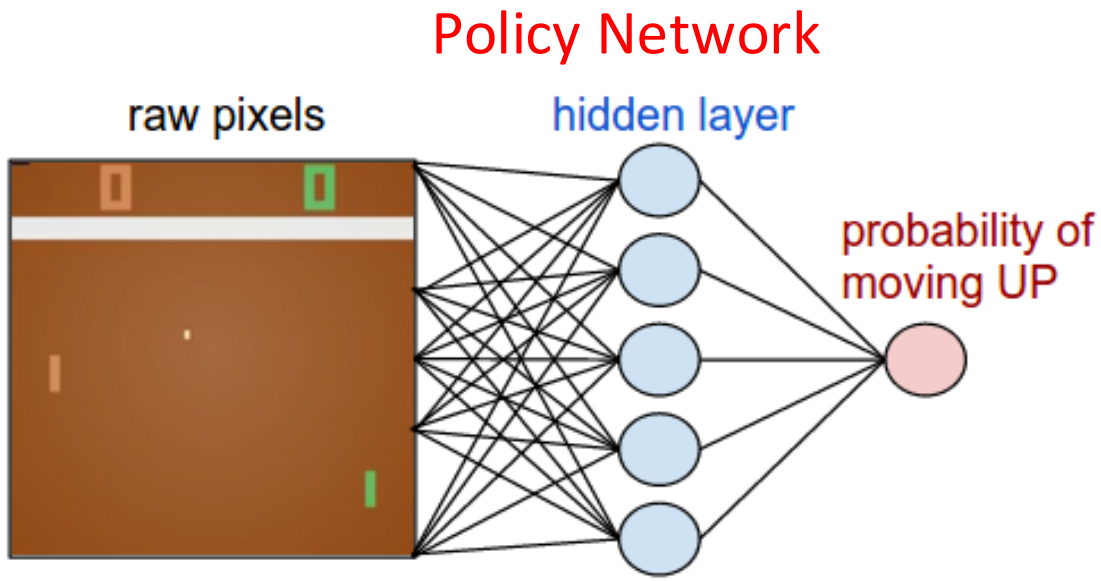
http://karpathy.github.io/2016/05/31/rl/

Deep Reinforcement Learning Example

- Look deeper into the code

How to optimize the W1 and W2?

Policy Gradient! (To be introduced in future lecture)



http://karpathy.github.io/2016/05/31/rl/

What could be the potential problems?

```
import gym
env = gym.make("Taxi-v2")
observation = env.reset()
agent = load_agent()
for step in range(100):
    action = agent(observation)
    observation, reward, done, info = env.step(action)
```

Speed, multiple agents, structure of agent?

MetaDrive Environment

- https://github.com/metadriverse/metadrive

User Interface

Point Cloud

RGB Camera

navigation:
checkpoints
surrounding:
vehicle_1:
position
velocity
checkpoints
...

state:
position
velocity
latitude
longitude
heading
length
...

Sensory Data

Bird-eye View

Depth Camera

Live Demo of MetaDrive and MetaUrban

- MetaDrive: https://github.com/metadriverse/metadrive/tree/upgrade-mac
- MetaUrban: https://github.com/metadriverse/metaurban

Homework and What's Next

- Start working on Assignment 1
- Play with OpenAI gym and the example code

https://github.com/ucla-rlcourse/RLexample

- Try to understand my_learning_agent.py
- Go through this blog in detail to understand pg-pong.py

http://karpathy.github.io/2016/05/31/rl/

- Next week: Markov Decision Process, policy iteration, and value iteration
- Please read **Sutton and Barton: Chapter 1 and Chapter 3**

In case you need it

- Python tutorial: http://cs231n.github.io/python-numpy-tutorial/
- PyTorch tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- Python in Google Colab: https://colab.research.google.com/github/cs231n/cs231n.github.io/blob/master/python-colab.ipynb