Reinforcement Learning

AI Systems Implementation

George Langroudi

Overview

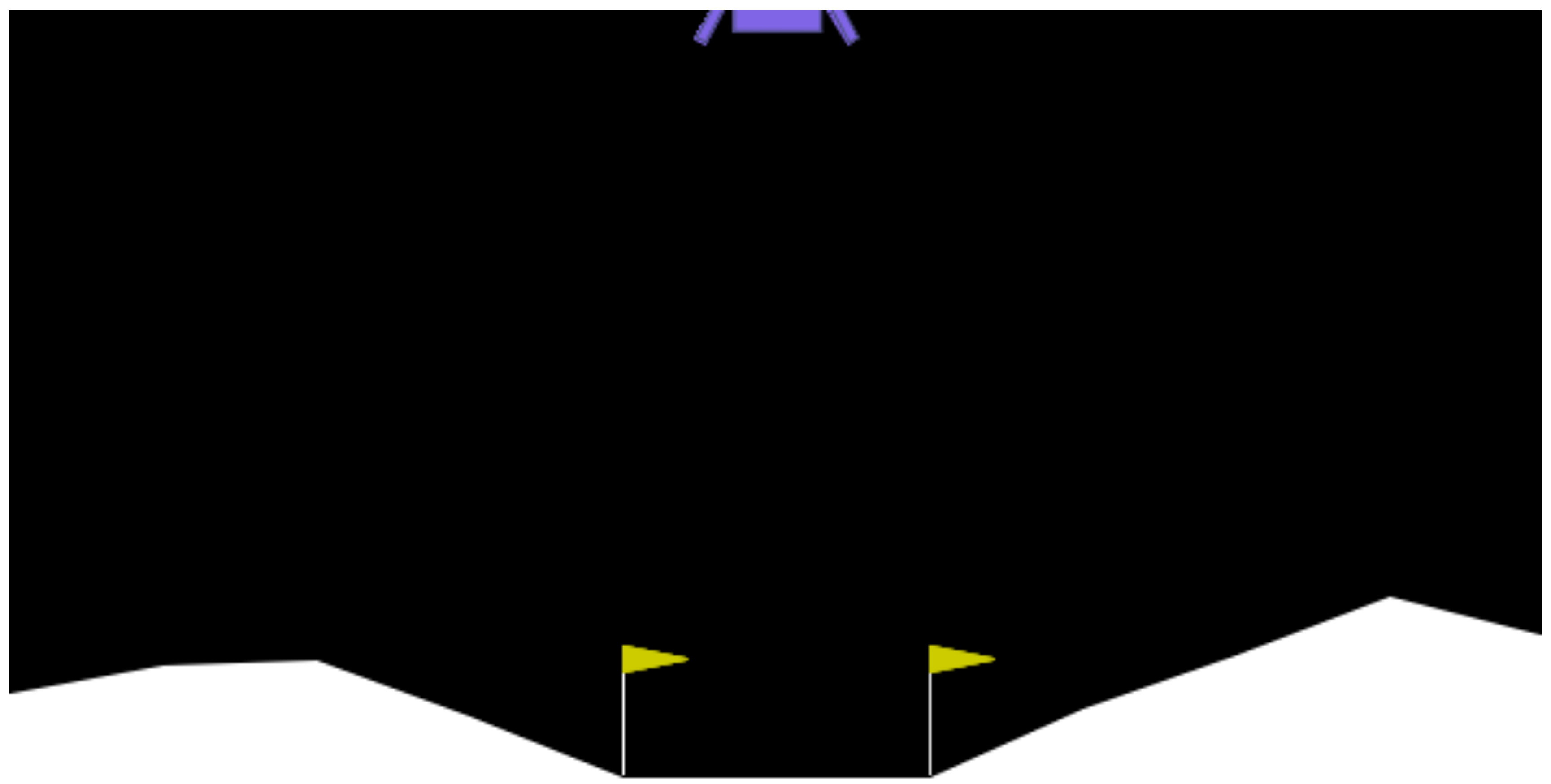
- What is the In-Class Test?
- What is Reinforcement Learning
- The OpenAI Gym / Farama Foundation Gymnasium
- Basic Policies
- Neural Network based Policy
- Q-Learning

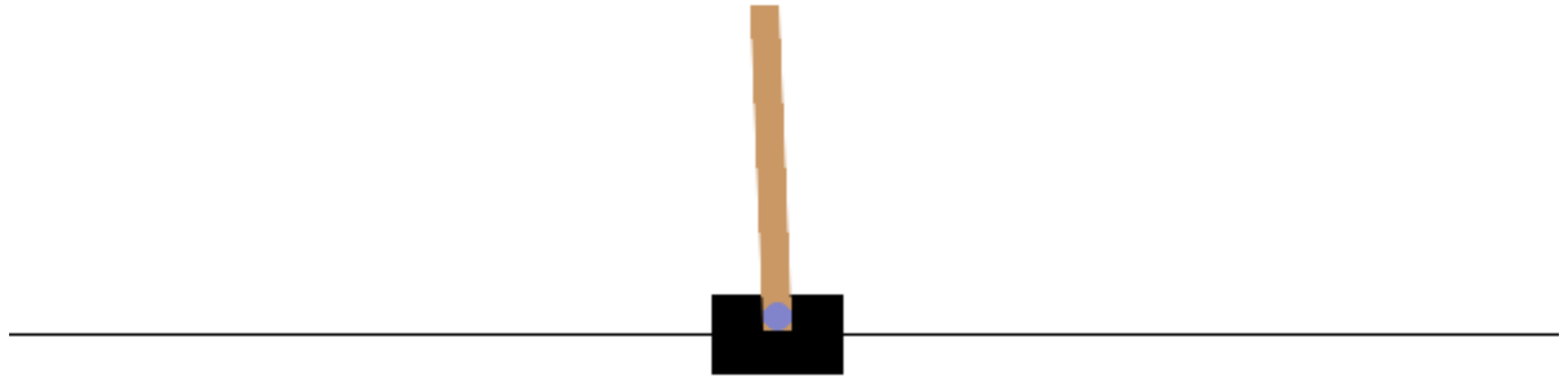
In-Class Test

- Examination Conditions
- You will be provided a Jupyter Notebook
- Each section will have a task, and a cell to enter your answer into
- You may not look at your previous classwork, but you can look at the documentation for relevant libraries
- 90 minutes
- COMP5850 - Different class time to usual, double check your timetable!

What is Reinforcement Learning?

- Algorithms capable of learning behaviours
- Each system typically has 5 elements:
  - Environments
  - States
  - Agents
  - Actions
  - Rewards





Farama Gymnasium

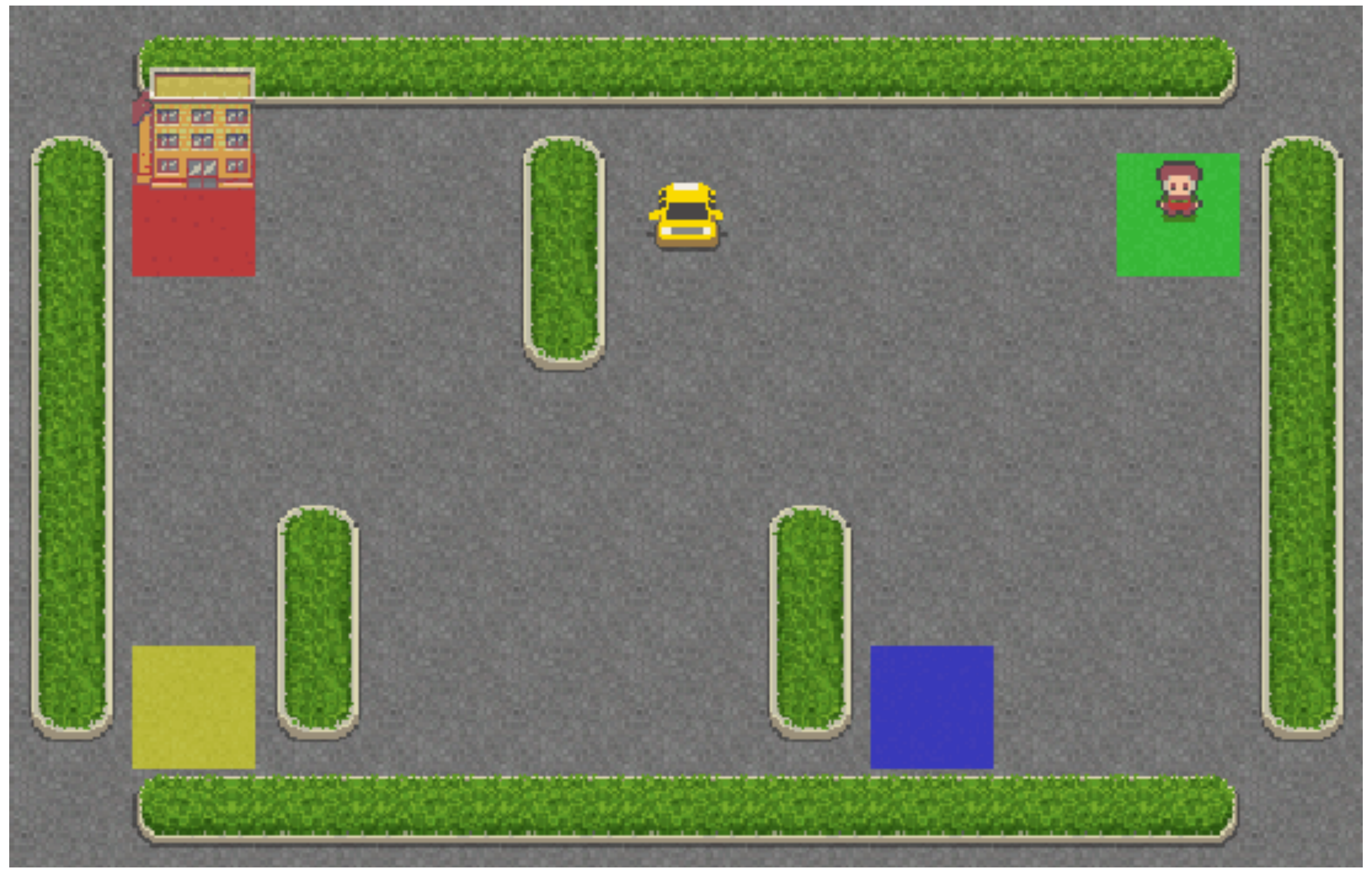
Environments

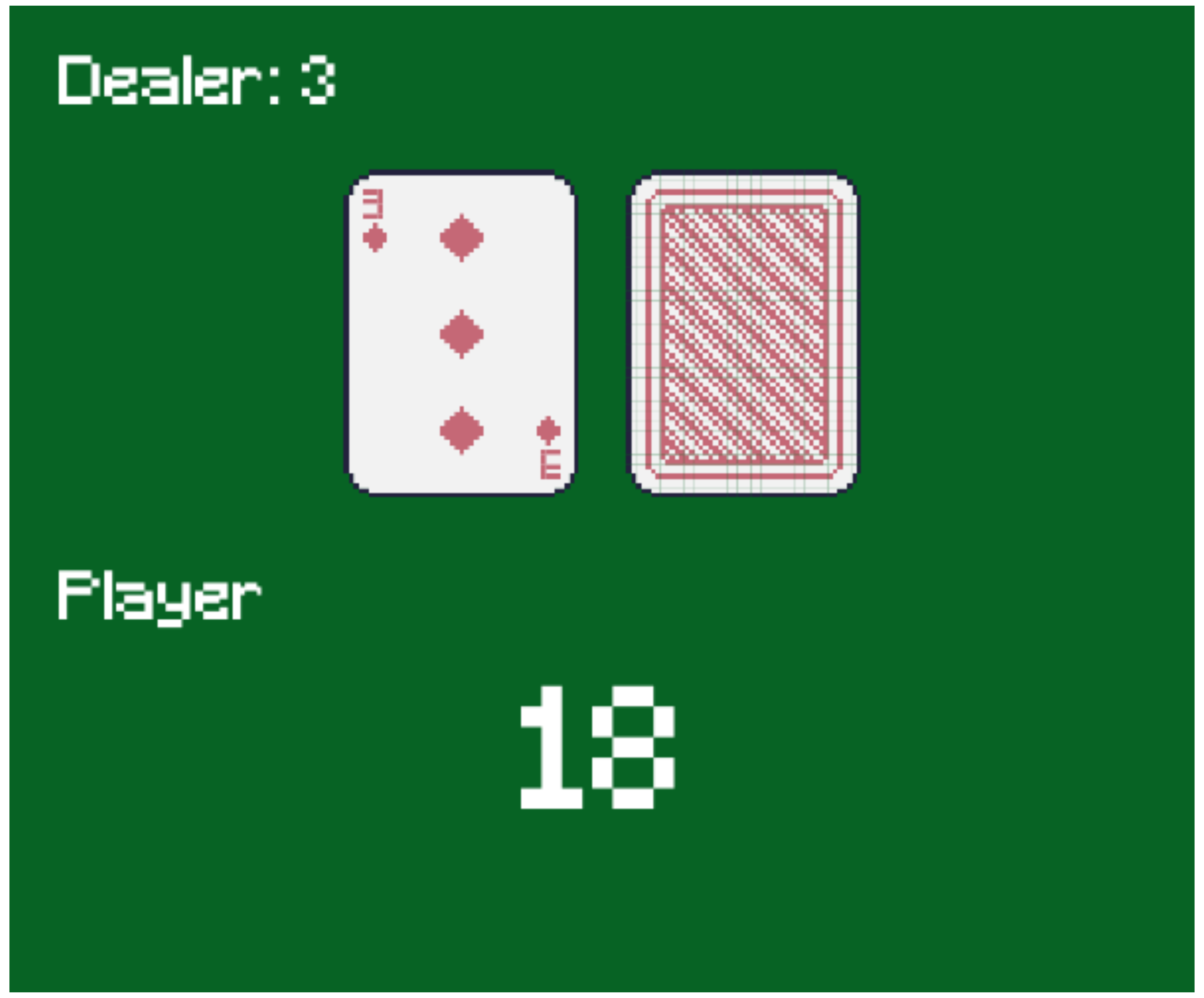
- Formerly OpenAI Gym
- A collection of simulations and environments
- Each environment allows you to simulate step-by-step whilst connecting to an agent
- Huge variety of environments available

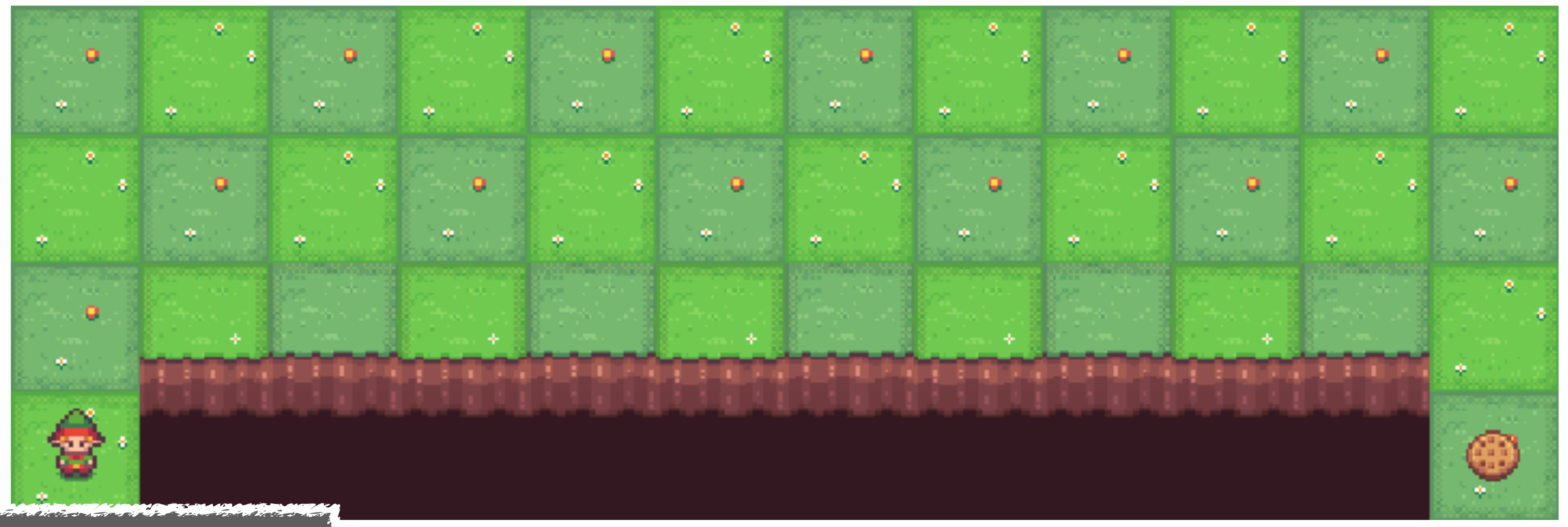
Farama Gymnasium

Toy Text

- Very basic environments
- Very limited states and action spaces
- Designed for basic testing, and designing new models





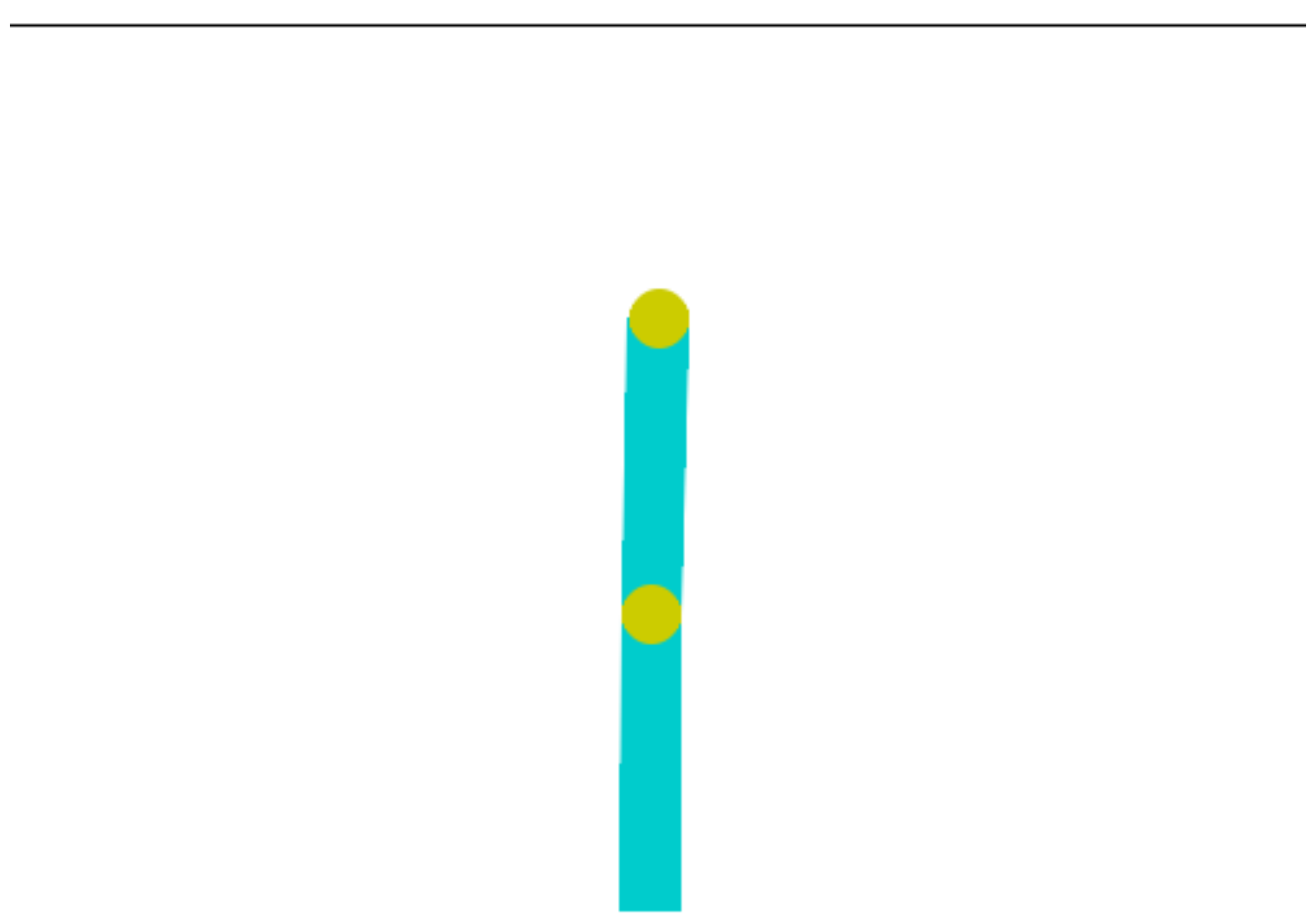


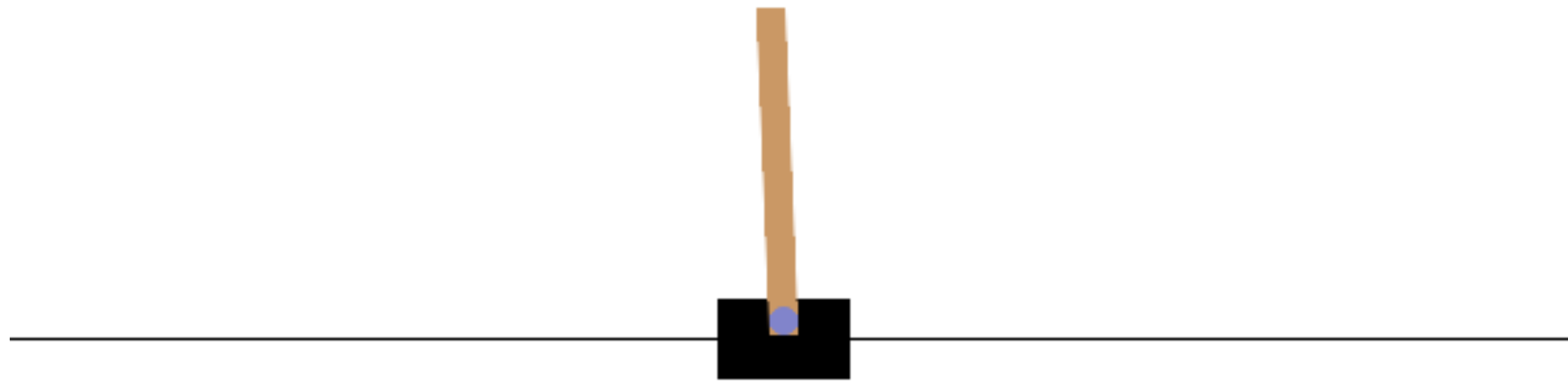
All images courtesy of Farama Foundation

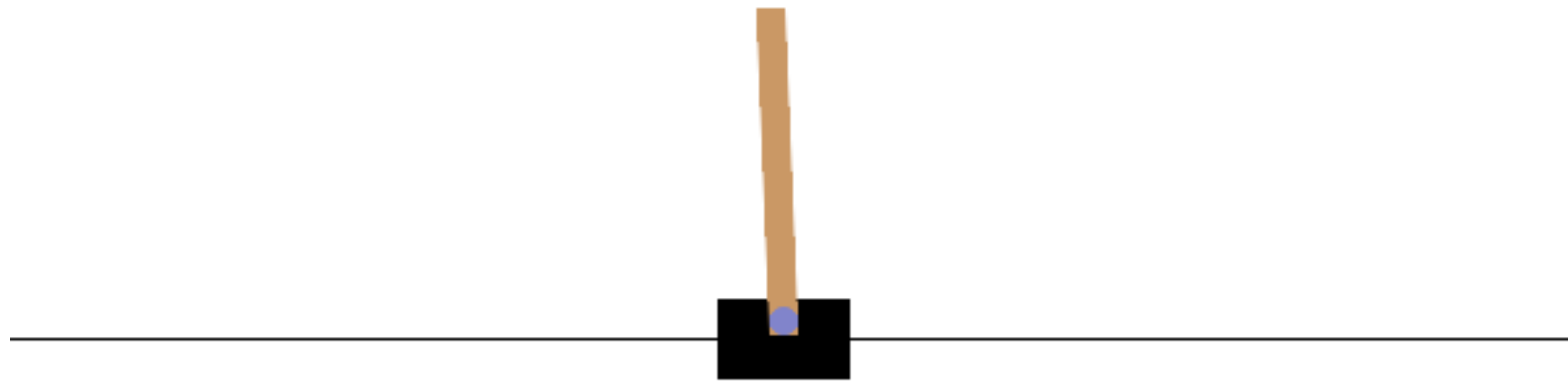
Farama Gymnasium

Classic Control

- A set of “classic” environments
- Typically simple environments, with basic “states”
- Great for early experimentation



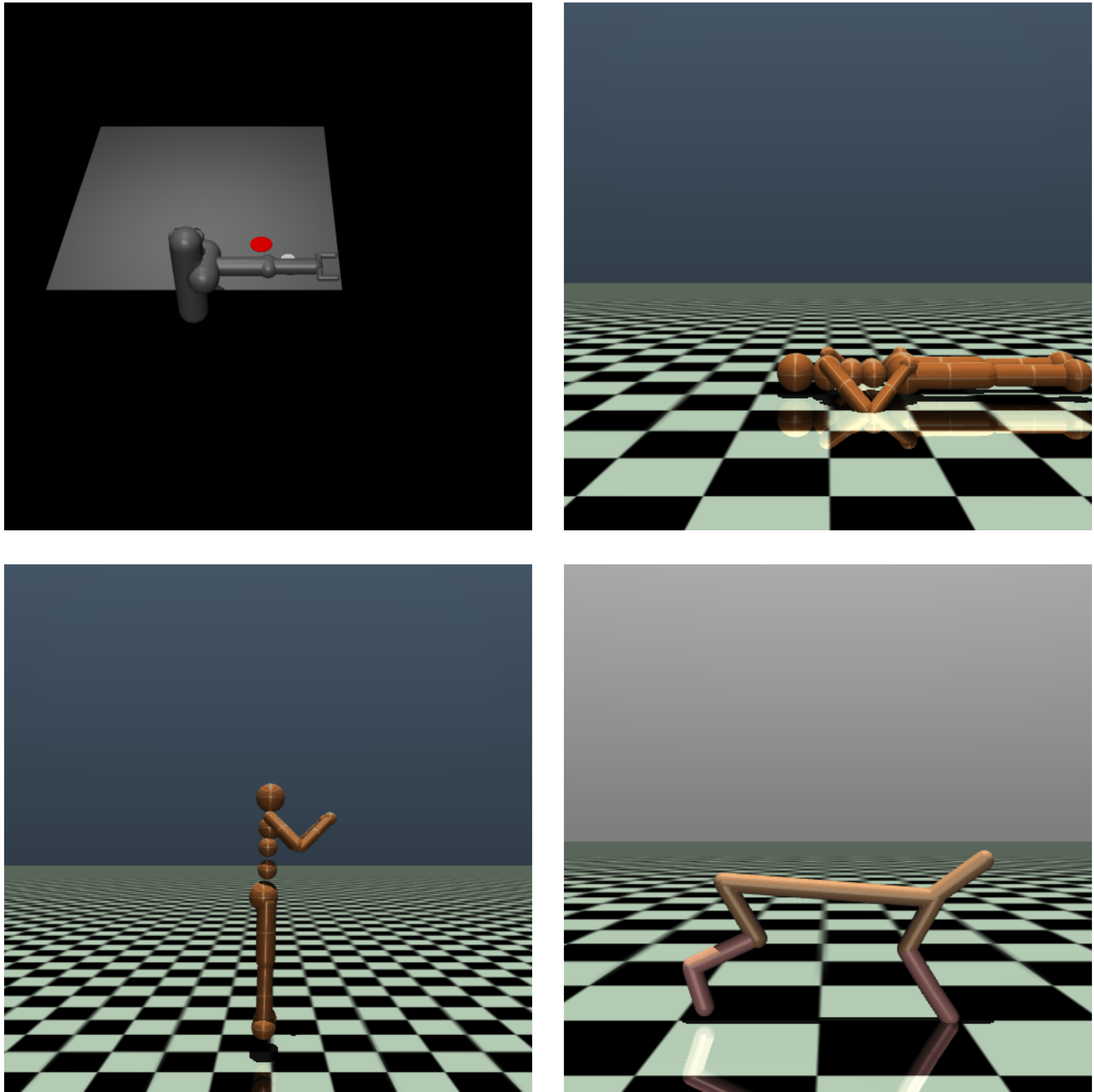




All images courtesy of Farama Foundation

Farama Gymnasium
MuJoCo

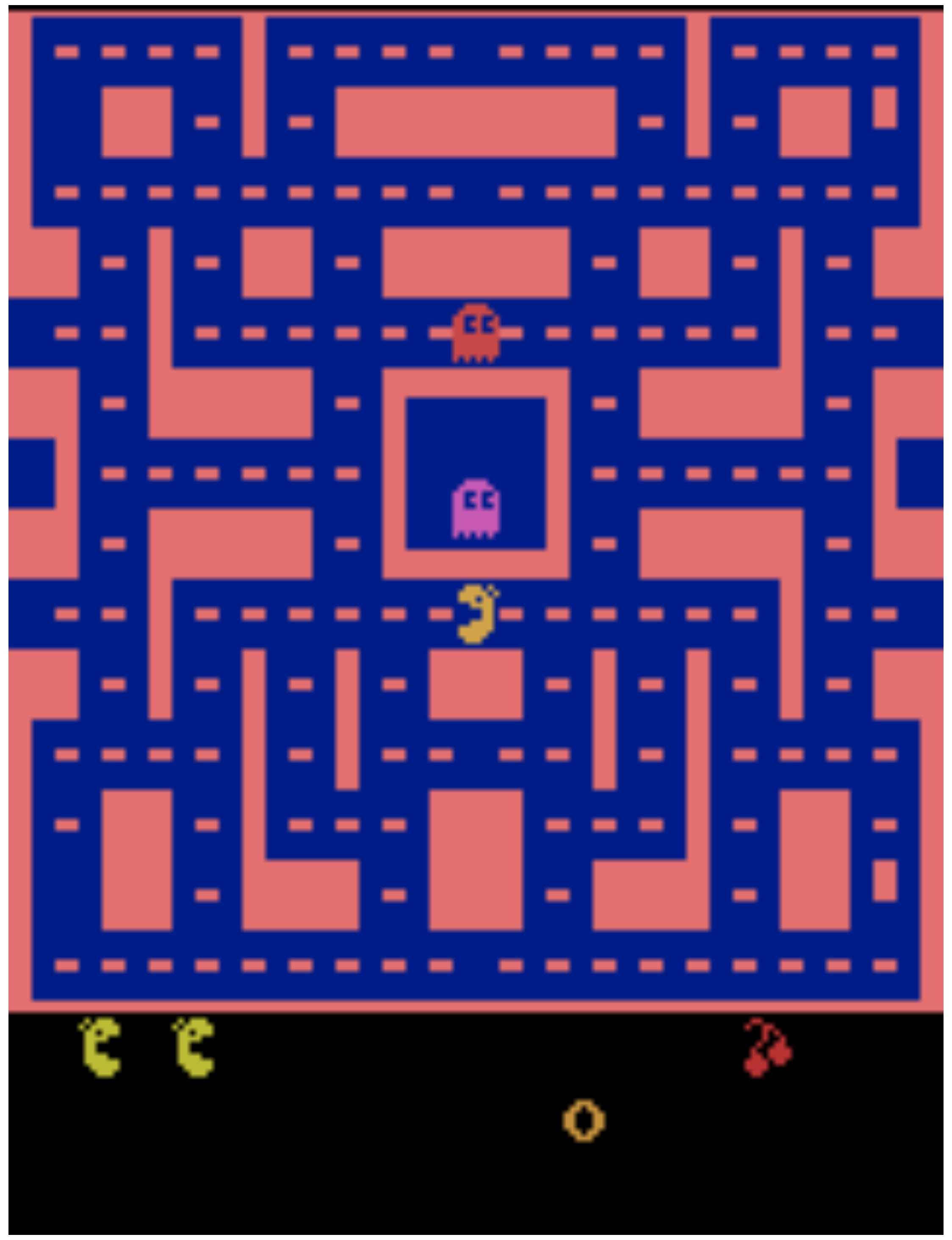
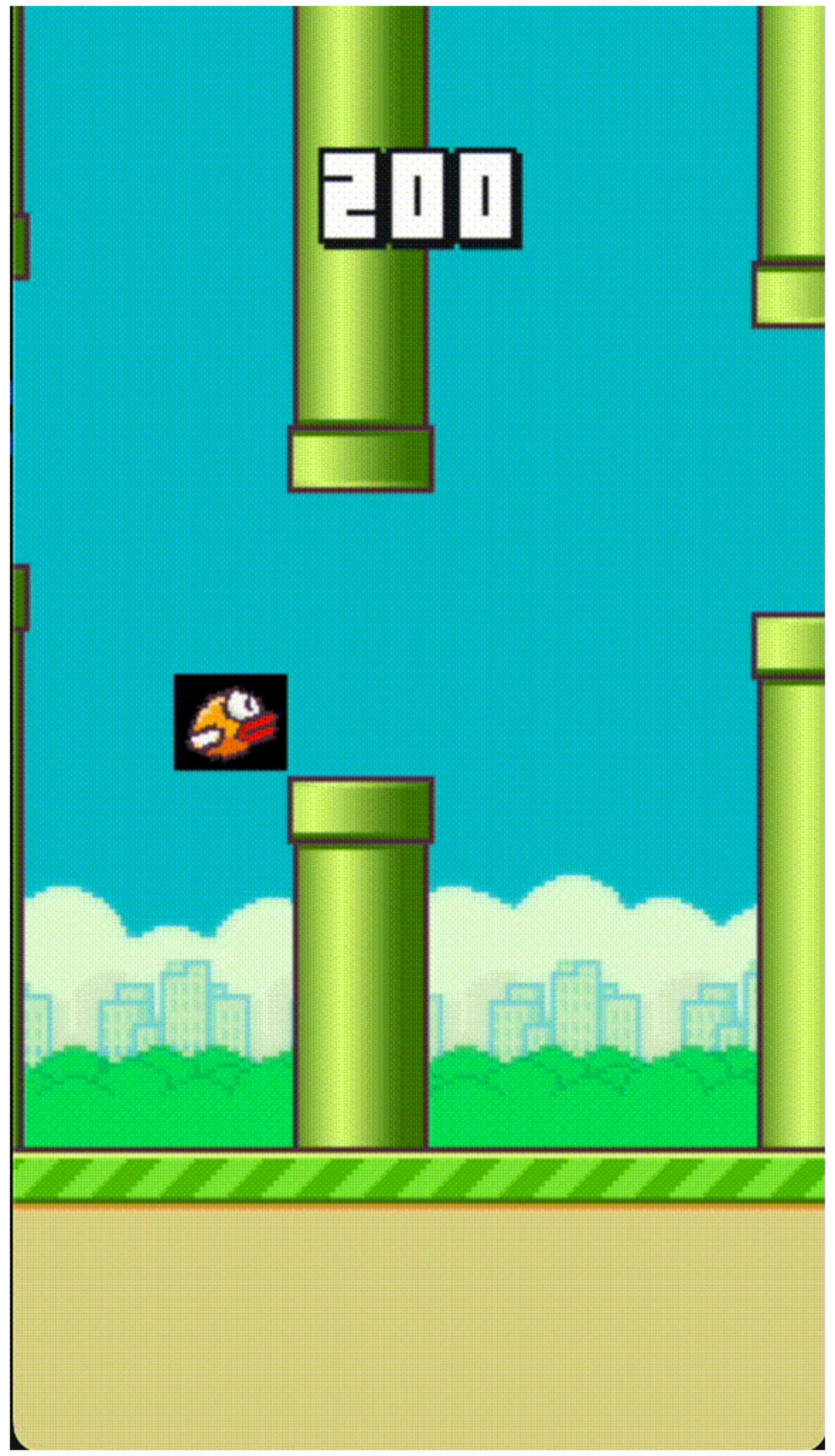
- Advanced simulations utilising MuJoCo physics
- Much more complex scenarios, with more open states and environments
- More advanced experiments

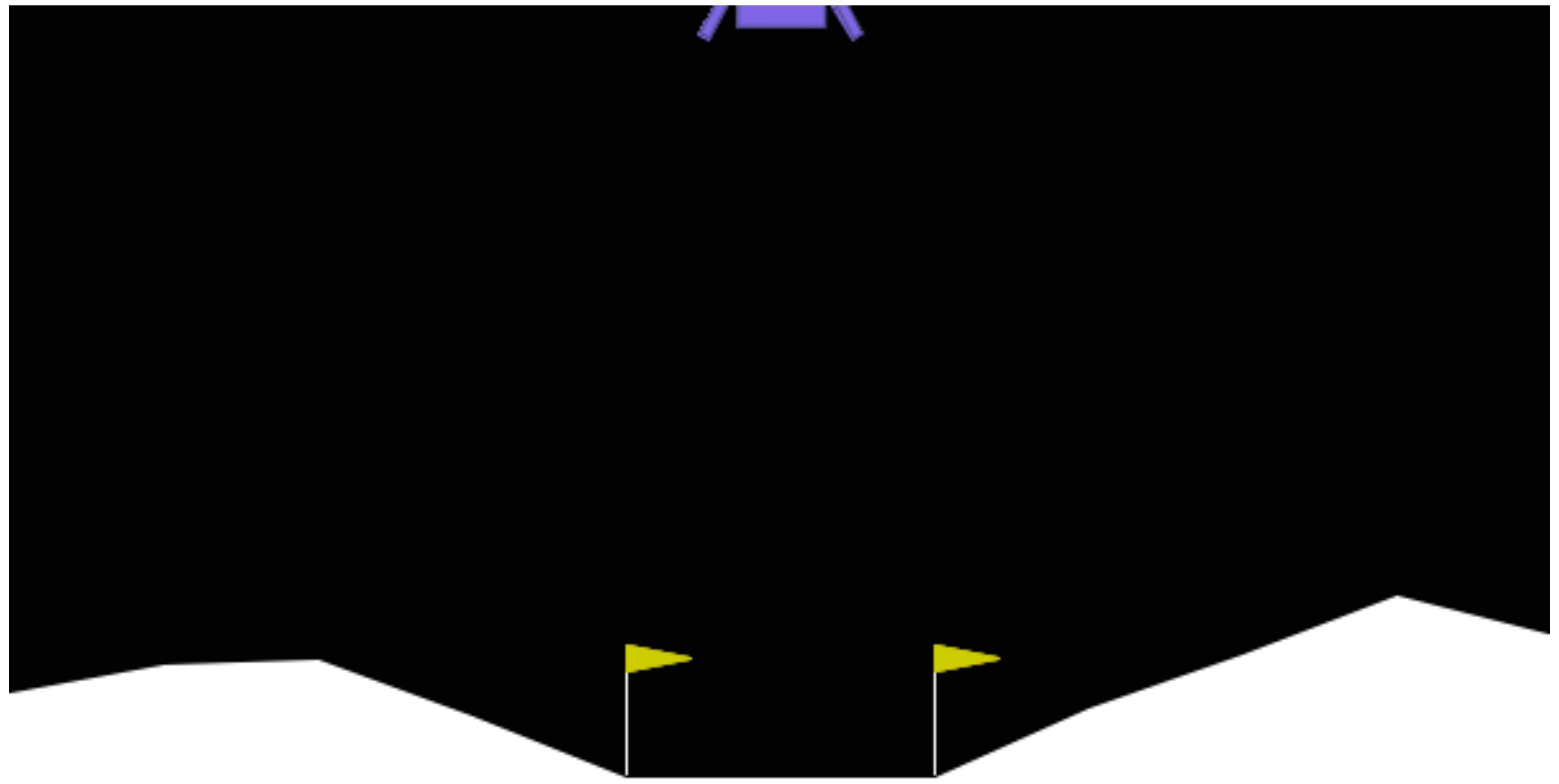


All images courtesy of Farama Foundation

Farama Gymnasium
Others

- Box2D
  - Simply physics environments
- Atari
  - Literal Atari 2600 games
- Third Party
  - Pretty much anything!



All images courtesy of Farama Foundation

Some more definitions

- States (S_t)
  - Represent the current state of the scenario. Could be an array of values
- Actions (A_t)
  - An array representing some forces to be applied in the Simulation.
- Rewards (R_t)
  - A value representing how the system performed
- Agents
  - The system that reads the states, provides the actions by following it's *policy*, and receives the rewards
- Every system can be represented as (S_t, A_t, R_t), i.e. $S_0, A_0, R_0, S_1, A_1, R_1, S_2 \ldots$

Policies

- Imagine we want the elf to reach the present, but avoid the frozen lakes
- Observation space: 16
    - One for each possible square, telling us it’s current content
- Action space: 4
    - One for each direction we can move
- Reward
    - 1 for reaching the present, 0 for lakes

Bacis Neural Network

```
import tensorflow as tf
from tensorflow import keras

n_inputs = env.observation_space.shape[0] # 4, 1 for each State value

model = keras.models.Sequential([
    keras.layers.Dense(5, activation="relu", input_shape=[n_inputs]),
    keras.layers.Dense(1, activation="sigmoid")
])
```

- We could define a very simple network!
  - 4 input neurons, a 5 neuron hidden layer, and a 1 neuron output
  - But wait… how do we train this?

Credit assignment

- If a model is responsible for a single action on a single step, how do we know what steps were successful?
  - Surely it won't be just the last step that caused the system to fail?
- Instead of evaluating with the reward, lets add a reduction (γ) and evaluate with the sum of all the last reward steps

Credit assignment

- Lets say the last few steps look like this:

$$10 \rightarrow 0 \rightarrow -50$$

- So without a penalty, our reward would be $10 + 0 + -50 = -40$
- Lets say also say $\gamma = 0.8$, then our reward would be

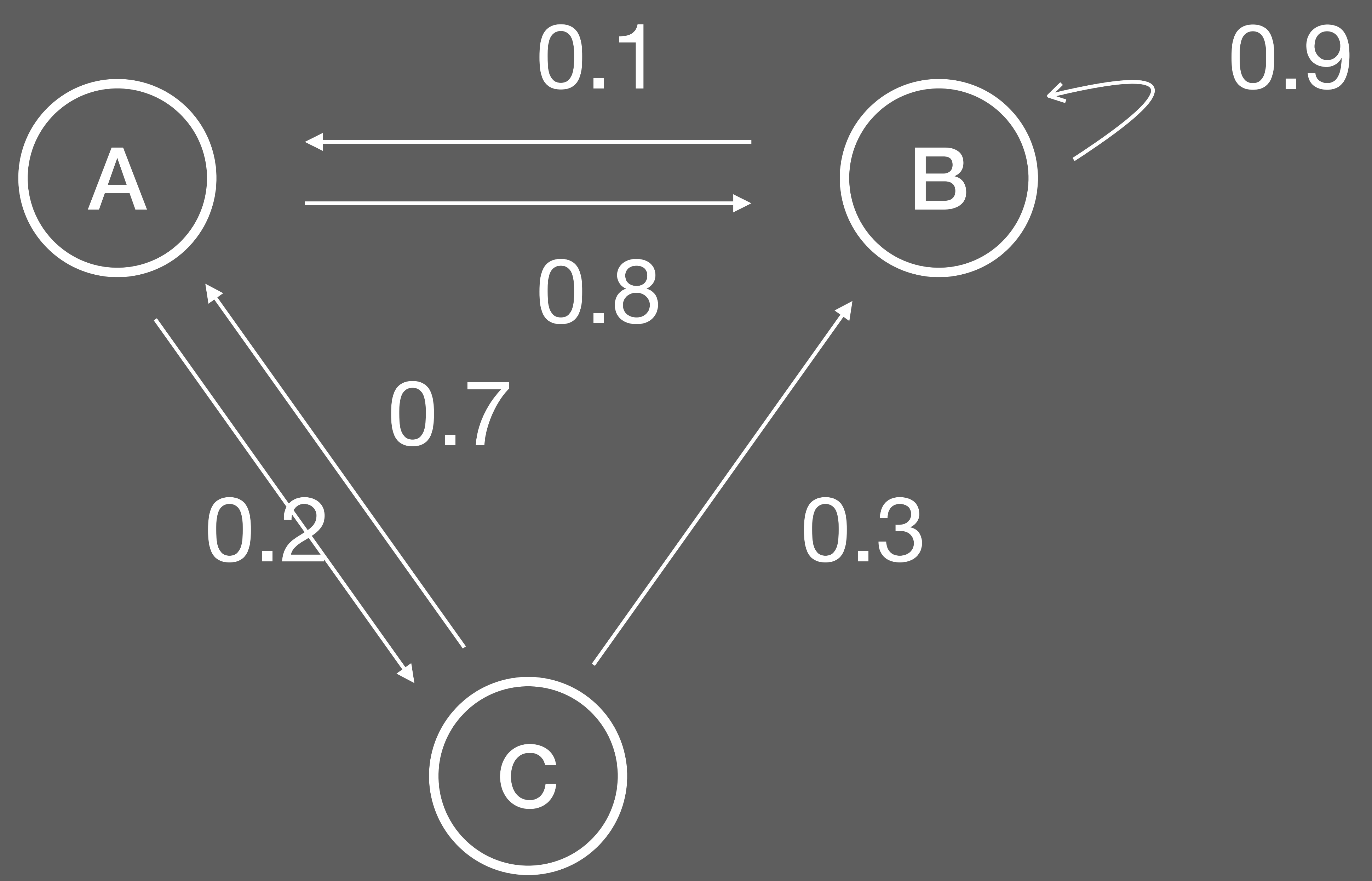
$$10 + \gamma \times 0 + \gamma^2 \times (-50) = -22$$

$$10 + 0.8 \times 0 + 0.8^2 \times (-50) = -22$$

- That way, future rewards are weighted less than current rewards

Markov Models

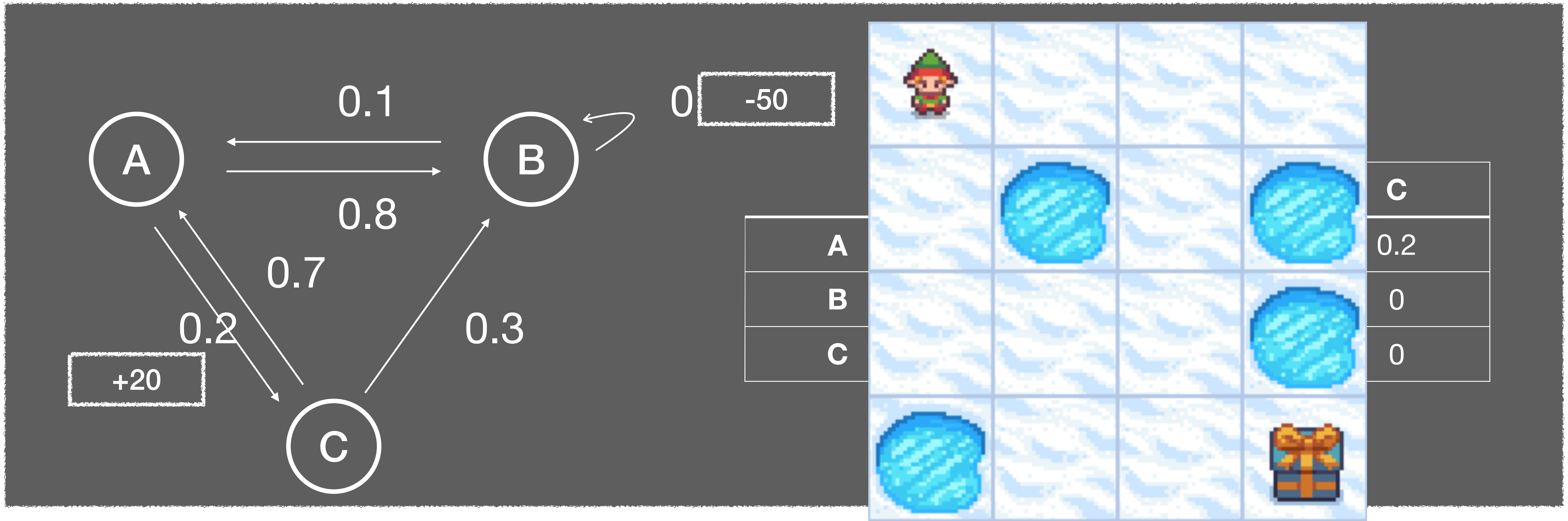
- A Markov Model is a probabilistic model of states
  - The probability of moving to a new state depends on the current state



	A	B	C
A	0	0.8	0.2
B	0.1	0.9	0
C	0.7	0.3	0

Markov Decision Process

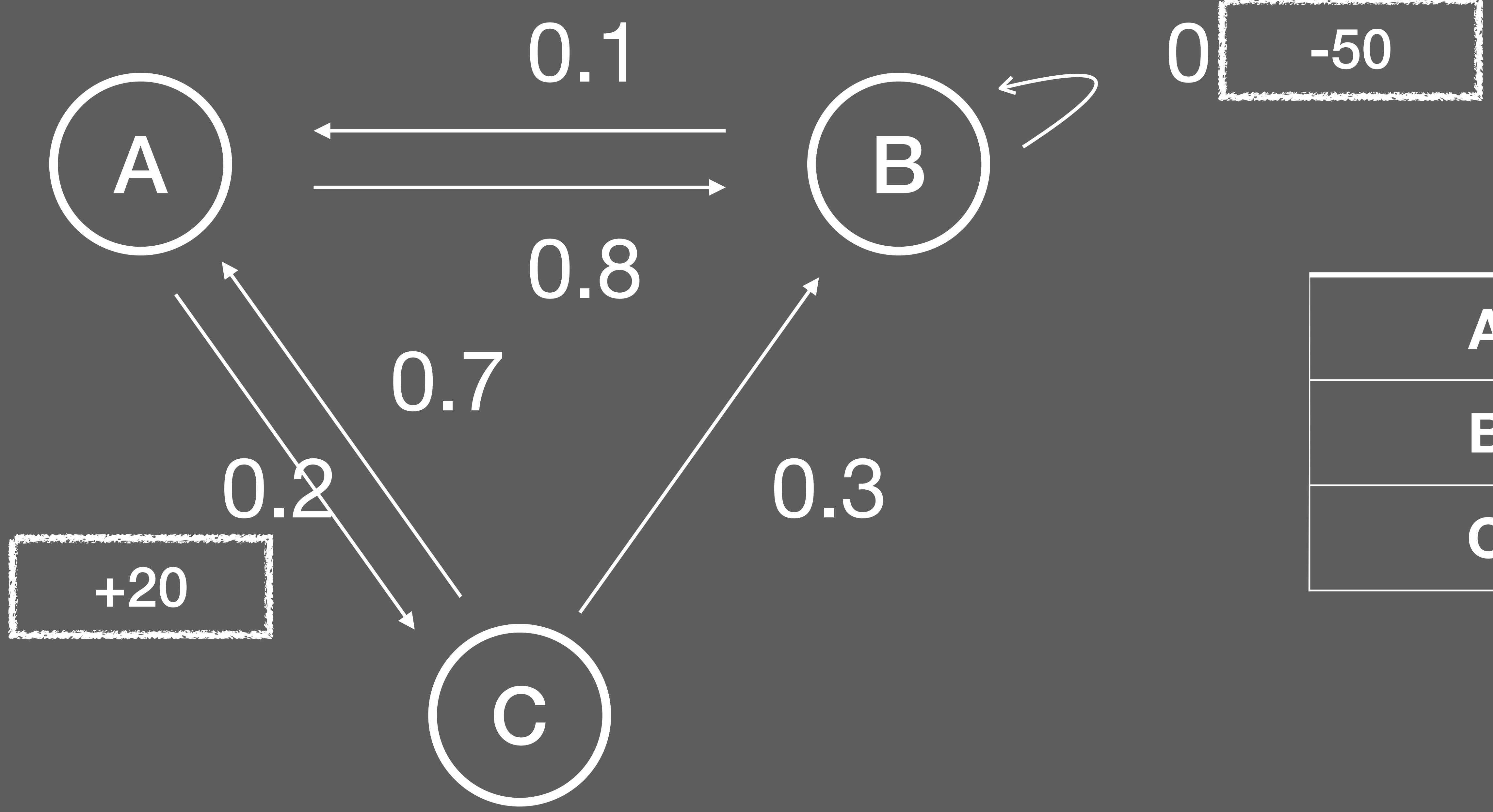
- Now imagine we also model *rewards* as a part of this model
  - Now we can represent the states of our environment!



Markov Decision Process

- Is there an optimal way to traverse this scene to maximise the reward?
  - Yes! Bellman's Optimality Equation:

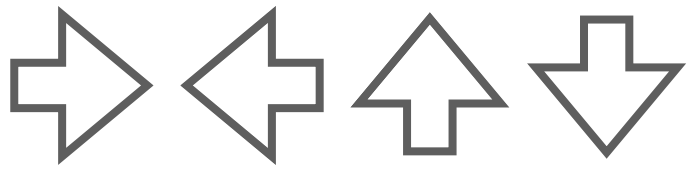
$$V*(s) = max_a \sum_{s} T(s, a, s')[R(s, a, s') + \gamma \cdot V*(s')] \text{ for all } s$$



	A	B	C
A	0	0.8	0.2
B	0.1	0.9	0
C	0.7	0.3	0

Q-Learning

- Models a MDP
- Maintains a Q-Table of shape $Q(s, a)$
  - Ever row represents a current state S_t
  - Each row contains a list for each possible action A_t
  - Each Q-Value shows what the algorithm thinks would be the best decision



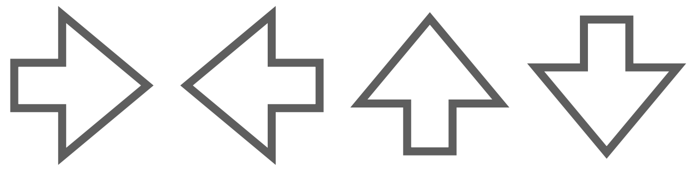
State	Q-Values
1	[0.735, 0.773, 0.773, 0.735]
2	[0.735, 0, 0.814, 0.773]
3	[0.773, 0.857, 0.773, 0.814]
4	[0.814, 0, 0.773, 0.773]
5	[0.773, 0.814, 0, 0.735]
6	[0, 0, 0, 0]
7	[0, 0.902, 0.902, 0]
…	…

Q-Learning

Update Formula

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

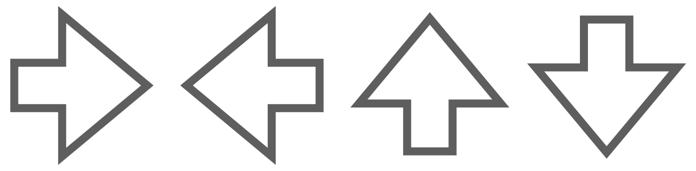
- $Q(s,a)$: The current Q-value for s and a
- r: The immediate reward for taking action a in state s
- α: The learning rate
- s': The next state after taking action a in state s
- a': The next action chosen in s'
- $max_{a'} Q(s',a')$: the maximum possible Q-Value from all possible actions in the next state s



State	Q-Values
1	[0.735, 0.773, 0.773, 0.735]
2	[0.735, 0, 0.814, 0.773]
3	[0.773, 0.857, 0.773, 0.814]
4	[0.814, 0, 0.773, 0.773]
5	[0.773, 0.814, 0, 0.735]
6	[0, 0, 0, 0]
7	[0, 0.902, 0.902, 0]
…	…

Q-Learning

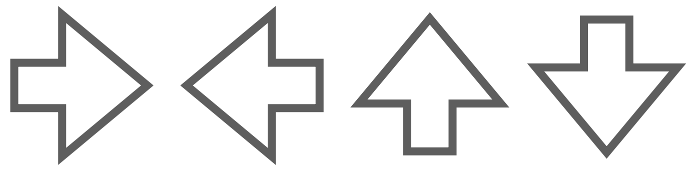
- Train thousands of times, and test as many state/action combinations as possible, then adjust values based on outcomes
- During training, each step is either stochastic or greedy (ϵ-greedy)
    - ϵ probability to be random, $1 - \epsilon$ chance to be Greedy
- Why? Why not just be Greedy?



State	Q-Values
1	[0.735, 0.773, 0.773, 0.735]
2	[0.735, 0, 0.814, 0.773]
3	[0.773, 0.857, 0.773, 0.814]
4	[0.814, 0, 0.773, 0.773]
5	[0.773, 0.814, 0, 0.735]
6	[0, 0, 0, 0]
7	[0, 0.902, 0.902, 0]
…	…

SARSA
State-Action-Reward-State-Action

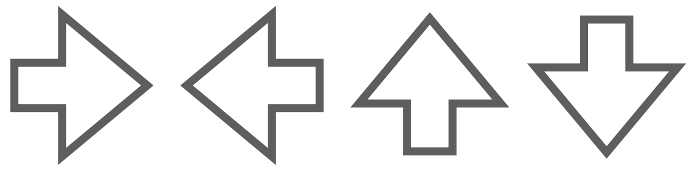
- Similarly to Q-Learning, SARSA maintains a Q-Table
- Initialise a Q-Table with arbitrary values
- Utilise ϵ-greedy search
- $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [r + \gamma \cdot Q(s',a') - Q(s,a)]$
    - How is that different than Q-Learning?
    - $Q(s',a')$, not $max_{a'}Q(s',a')$
    - SARSA values the current action over possible future actions



State	Q-Values
1	[0.735, 0.773, 0.773, 0.735]
2	[0.735, 0, 0.814, 0.773]
3	[0.773, 0.857, 0.773, 0.814]
4	[0.814, 0, 0.773, 0.773]
5	[0.773, 0.814, 0, 0.735]
6	[0, 0, 0, 0]
7	[0, 0.902, 0.902, 0]
…	…

Q-Learning

- So far so good, so why don't we always use Q-Learning or SARSA?
- Well, what if you weren't trying to play a basic game, but wanted to teach something how to walk?
    - How many states are there?
    - How do we model that as a table?
        - How big will the table be?!



State	Q-Values
1	[0.735, 0.773, 0.773, 0.735]
2	[0.735, 0, 0.814, 0.773]
3	[0.773, 0.857, 0.773, 0.814]
4	[0.814, 0, 0.773, 0.773]
5	[0.773, 0.814, 0, 0.735]
6	[0, 0, 0, 0]
7	[0, 0.902, 0.902, 0]
…	…

Deep Q-Learning

- Now, imagine populating that Q-Table for a more complex situation…
  - Suddenly, the table becomes so large it's impossible to model!
- Instead of modelling every possible state, we can fin a function that approximates $Q_\theta(s, a)$
  - Instead, we can use a Deep Neural Network (DNN) to estimate Q-Values
    - This is called a "Deep Q-Network" (DQN
- OK, but how do we use it?
  - The TF-Agents package!