

# Reinforcement Learning Packages

Trevor Gordon  
trevor.gordon@columbia.edu  
Columbia University  
October 28, 2022

# Overview

## OpenAI Gym

- Overview
- Gym Environments
- Working Example

## Other

- Stable Baselines 3


Example With OpenAI Gym and Stable Baselines



# OpenAI Gym Overview

What is OpenAI Gym?

"Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball." - OpenAI

- Separate the implementation of the environment from the implementation of the learning algorithm.
  - Make it easy to compare various RL algorithms on the same environment.
  - Create an accessible environment for those who wish to learn Deep RL. See also OpenAI Spinning Up in Deep RL.
- 

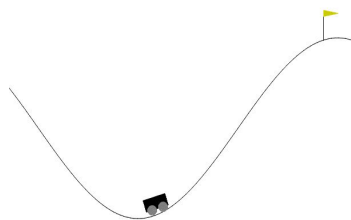
# Why OpenAI Gym

- The need for better benchmarks. Computer Vision has data sets such as ImageNet, but what is the equivalent for Reinforcement Learning?
- Lack of standardization of the environments used in publications. Differences in the reward function or set of available actions can significantly change difficulty of a task. This can make it difficult to reproduce published research and compare results from different papers.

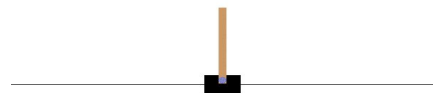


# Types Of Environments

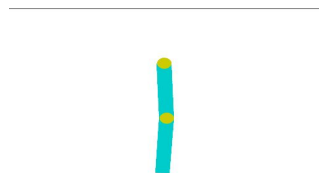
# Classic Control



Mountain Car



Cartpole



Acrobat



Pendulum

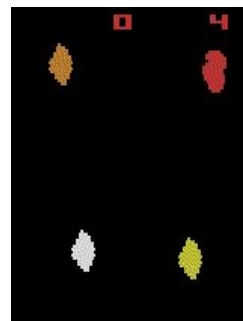
# Atari



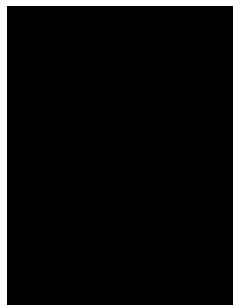
Assault



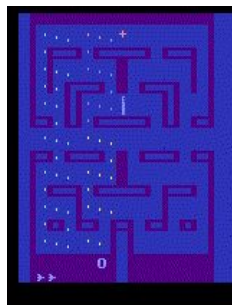
Bank Heist



Asteroids



Amidar

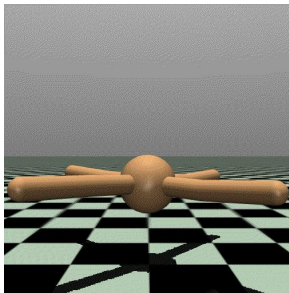


Alien

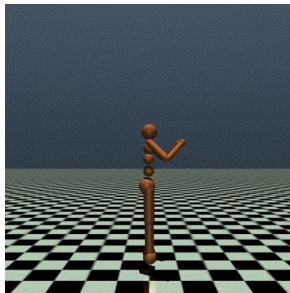


Air Raid

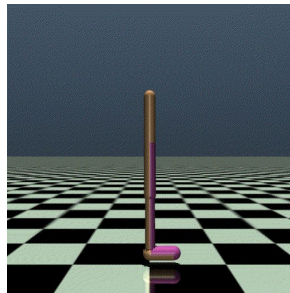
# Mujoco



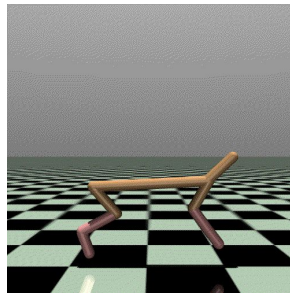
Ant



Humanoid



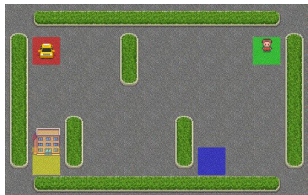
Walker2D



Cheetah



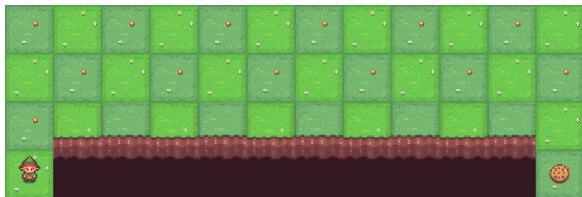
# Toy Text



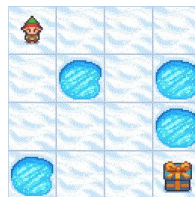
Taxi



Blackjack



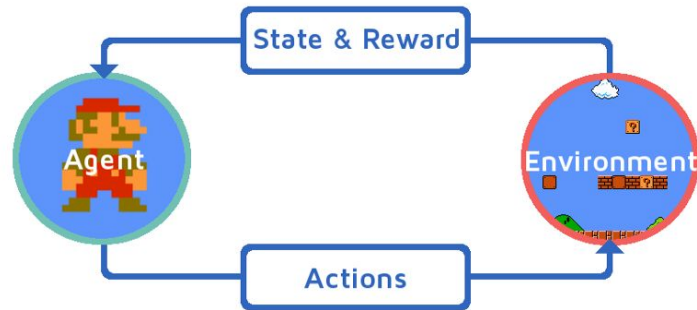
Cliff Walking



Frozen Lake

# Agent-Environment Loop

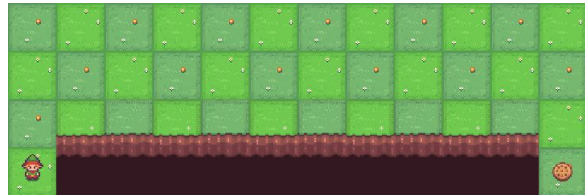
- **Observation:** The new state given by the environment
- **Reward:** Environment defined reward.
- **Done:** A boolean to indicate if the episode has finished
- **Info:** Additional info only for debugging



# Observation & Action Space

- The environment class has an `action_space` and `observation_space` that specify exact what needs to be given to and what will be returned from the environment
- Some types of Spaces for both
  - a. Discrete: A discrete space where  $a(s) \in \{0, 1, \dots, n\}$
  - b. Box: An  $n$  dimensional continuous space with optional bounds
  - c. Dict: A dictionary of space instances (ex: `Dict({"position": Discrete(2), "velocity": Discrete(3)})`)

# Cliff Walking Environment



## Description

The board is a 4x12 matrix, with (using NumPy matrix indexing):

- [3, 0] as the start at bottom-left
- [3, 11] as the goal at bottom-right
- [3, 1..10] as the cliff at bottom-center

If the agent steps on the cliff, it returns to the start. An episode terminates when the agent reaches the goal.

## Reward

Each time step incurs -1 reward, and stepping into the cliff incurs -100 reward.

## Action Space

There are 4 discrete deterministic actions:

- 0: move up
- 1: move right
- 2: move down
- 3: move left

## Observations

There are  $3 \times 12 + 1$  possible states. In fact, the agent cannot be at the cliff, nor at the goal (as this results in the end of the episode). It remains all the positions of the first 3 rows plus the bottom-left cell. The observation is simply the current position encoded as [flattened index](#).

# Mujoco Humanoid Environment

## Description

The 3D bipedal robot is designed to simulate a human. It has a torso (abdomen) with a pair of legs and arms. The legs each consist of two links, and so the arms (representing the knees and elbows respectively). The goal of the environment is to walk forward as fast as possible without falling over.

## Rewards

The reward consists of three parts:

- *healthy\_reward*: Every timestep that the humanoid is alive
- *forward\_reward*: A reward of walking forward
- *ctrl\_cost*: A negative reward for penalising the humanoid if it has too large of a control force.
- *contact\_cost*: A negative reward for penalising the humanoid if the external contact force is too large.

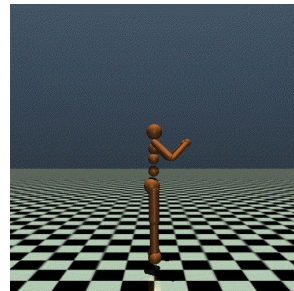
## Action Space

The action space is a Box(-1, 1, (17,)), float32). An action represents the torques applied at the hinge joints.

Actions: hip\_1 (front\_left\_leg), angle\_1 (front\_left\_leg), hip\_2 (front\_right\_leg), right\_hip\_x (right\_thigh), right\_hip\_z (right\_thigh), ...

## Observation Space

Observations consist of positional values of different body parts of the Humanoid, followed by the velocities of those individual parts (their derivatives) with all the positions ordered before all the velocities.



# Simple Code Example - Random Actions

```
import gym
env = gym.make("CartPole-v1")
state = env.reset()
for _ in range(1000):
    env.render()
    action = env.action_space.sample()
    state, reward, done, info = env.step(action)
    if done:
        state = env.reset()
env.close()
```



# Stable-Baselines-3

# Stable-Baselines-3 Overview

What is Stable-Baselines-3?

“Stable Baselines3 (SB3) is a set of reliable implementations of reinforcement learning algorithms in PyTorch.” - Stable-Baselines-3

- Where OpenAI gym focuses on the **environment**, stable-baselines-3 focuses on the **learning**
- Allow you to implement deep reinforcement learning solutions in just a few lines





# Stable-Baselines-3 Benefits

- Access to most popular deep reinforcement learning algorithms. Ex: DQN, PPO, A2C, DDPG
- Allow you to implement deep reinforcement learning solutions in just a few lines
- Vectorized training in parallel
- Unified structure for all algorithms
- Under active development
- Possibility to further customize out of the box solutions



# What is Stable-Baselines-3 Doing in the background?

- Look at the size of the action and observation space for the given environment
- Create deep neural networks with input/output spaces to match the environment
- Instantiate parallel copies of agents



# Simple Code Example - Learning

```
import gym
from stable_baselines3 import PPO
env = gym.make("CartPole-v1")
model = PPO("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=10_000)
obs = env.reset()
for i in range(1000):
    action, _states = model.predict(obs, deterministic=True)
    obs, reward, done, info = env.step(action)
    env.render()
    if done:
        obs = env.reset()
env.close()
```



# Useful Links

[Link to stable-baselines-tutorial](#)

[Link to openai-gym documentation](#)

[Link to stable-baselines-3 documentation](#)