Artificial Intelligence

Algorithms and Applications with Python

Lectorial 05

Dr. Dominik Jung
dominik.jung42@gmail.com

Agenda

5.1 Basics and Repetition

5.2 Implementing an Intelligent Agent

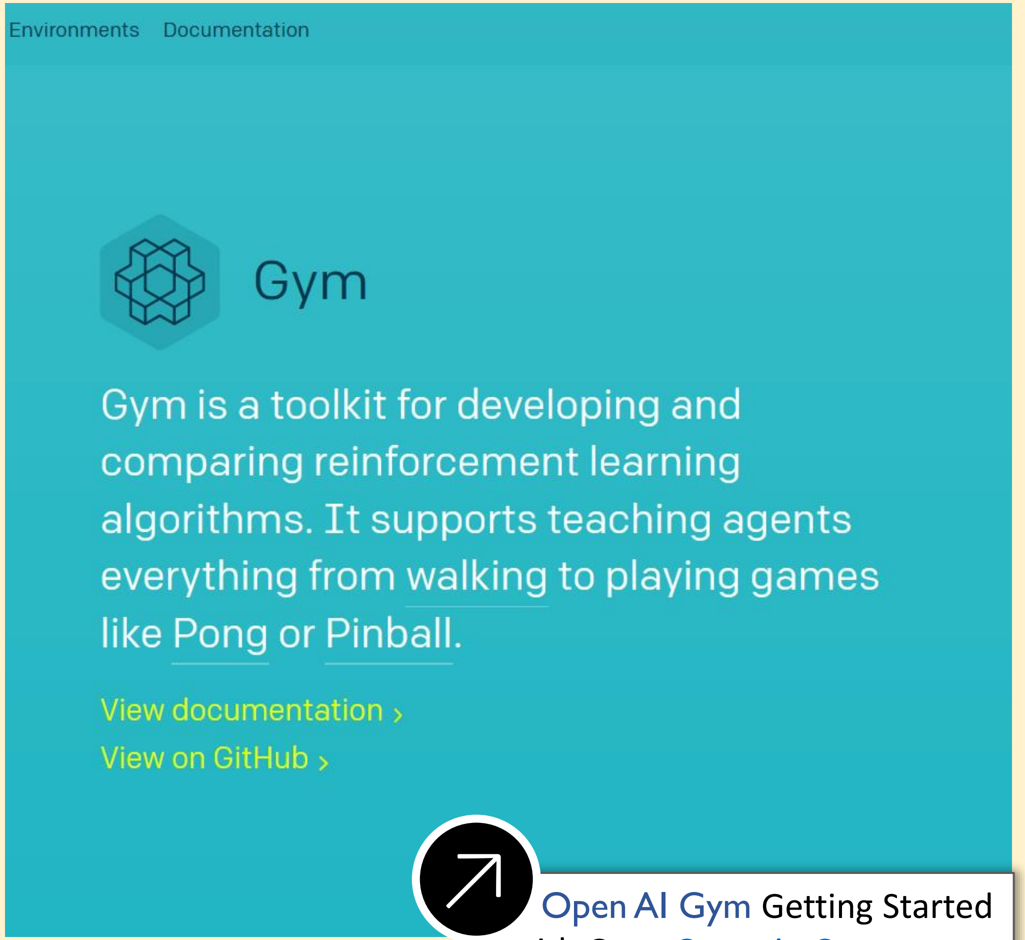
Note

- This is a lectorial: I will explain/repeat the most important concepts and then you try to solve the programming task by your own

- You are explicitly encouraged to solve this task in groups. And I will help you and give suggestions. However, there is no perfect solution, you will get a possible solution.

- If the task is too hard for you at the moment – relaxe ☺. Just look at the task again at a later point in the course.

Openai – Gym

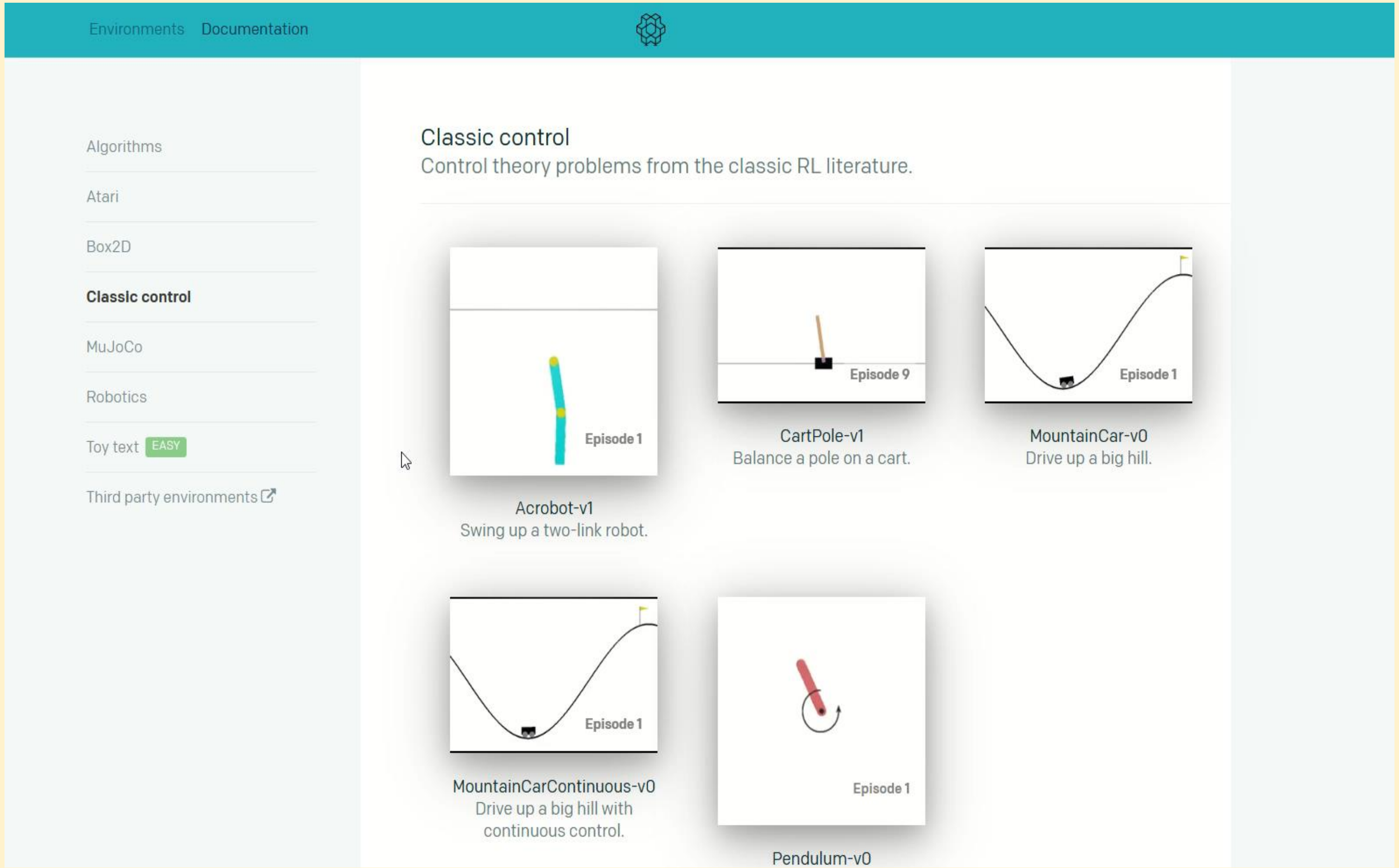


Open AI Gym Getting Started with Gym: [Openai - Gym]

- Toolkit for developing and comparing learning agents
- Collection of popular AI test problems (environments) to evaluate intelligent agents
- Makes no assumptions about the structure of the agent
- Compatible with any numerical computation library, such as TensorFlow or Theano

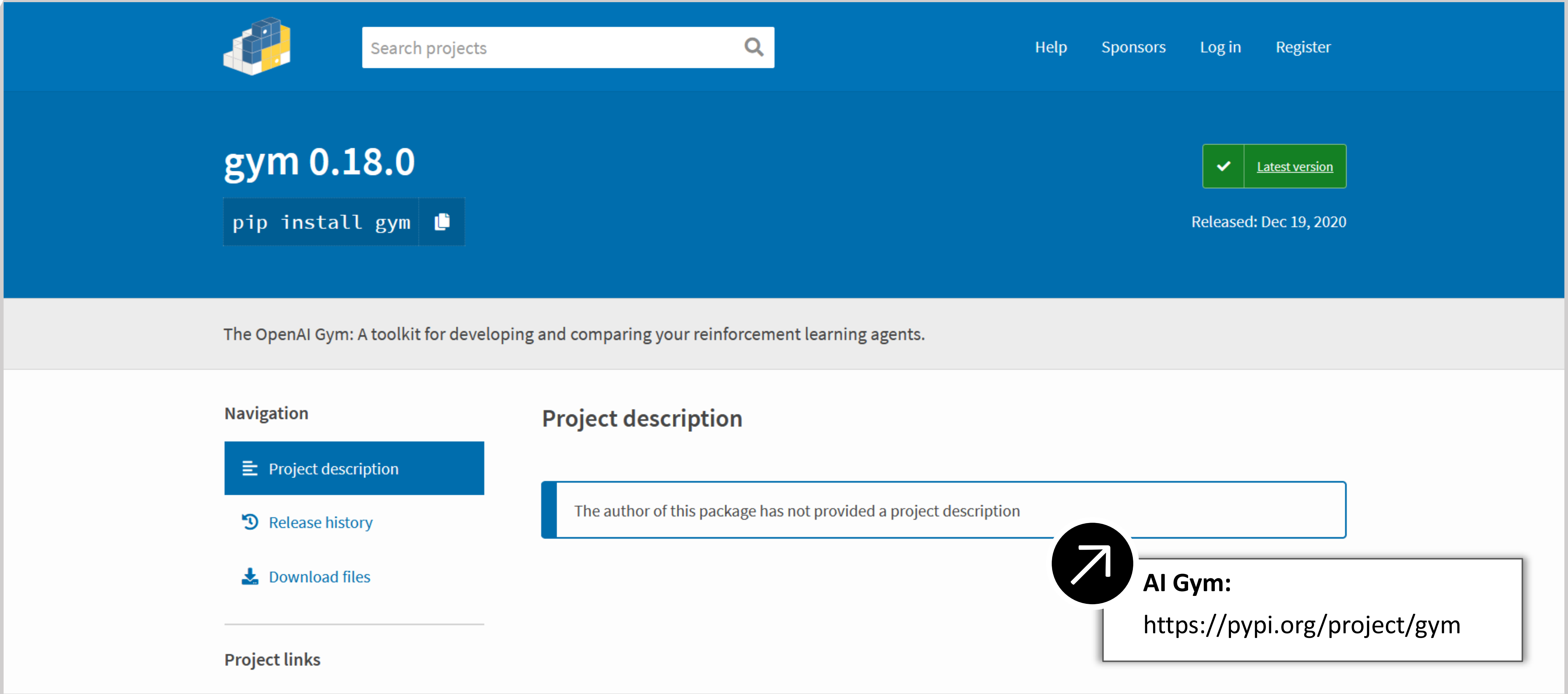
In ce Administration; ↗ www.clipsrules.net (2020)

Openai Gym Examples



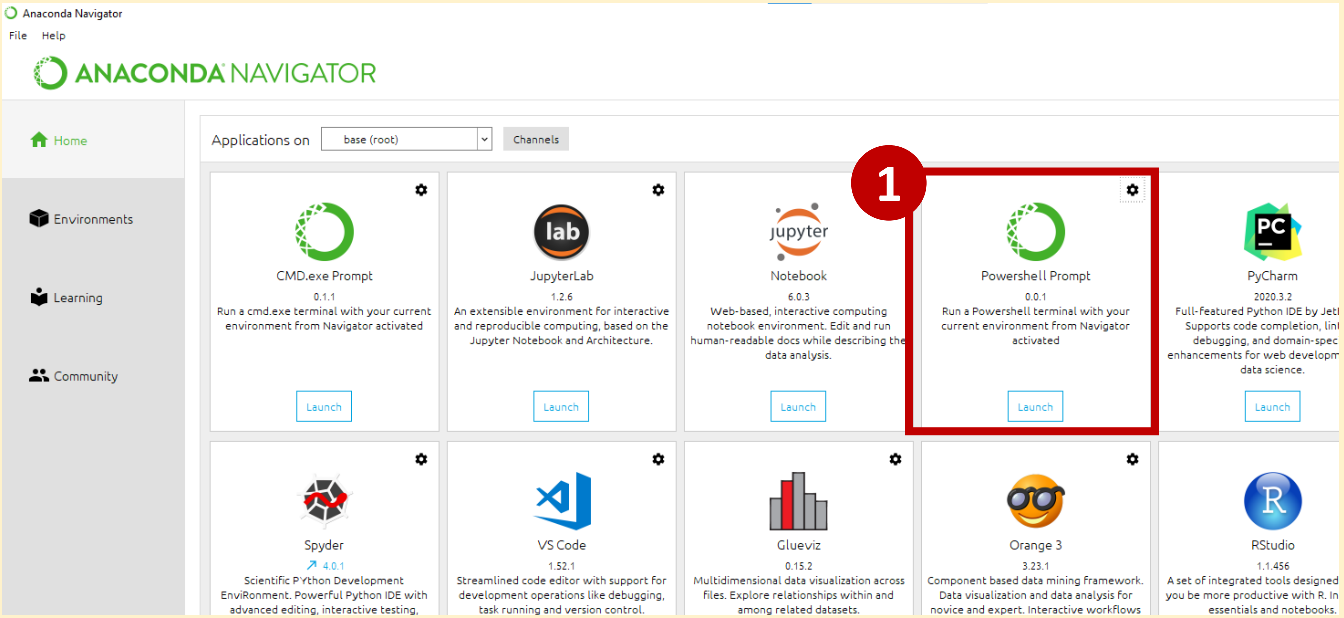
- Many popular AI learning problems from different areas available
- Contains complex games to test the learning performance of your agents like space invaders

AI gym



AI Gym:
https://pypi.org/project/gym

Install AI gym



- Experta is not available in the default anaconda repository, hence we have to install it from pypi

- Start your anaconda shell

2

- And type in the following pip install command:

```
pip install gym
```

3

- Install these packages if you want to play the games by yourself (see task later)

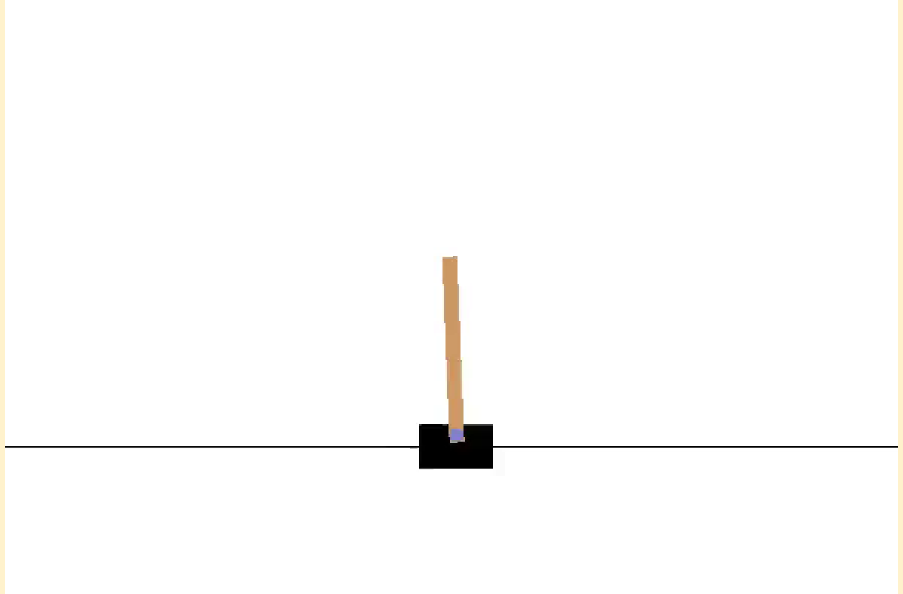
```
pip install --no-index -f https://github.com/Kojoley/atari-py/releases atari_py
```

Live Demo

Gym Environments

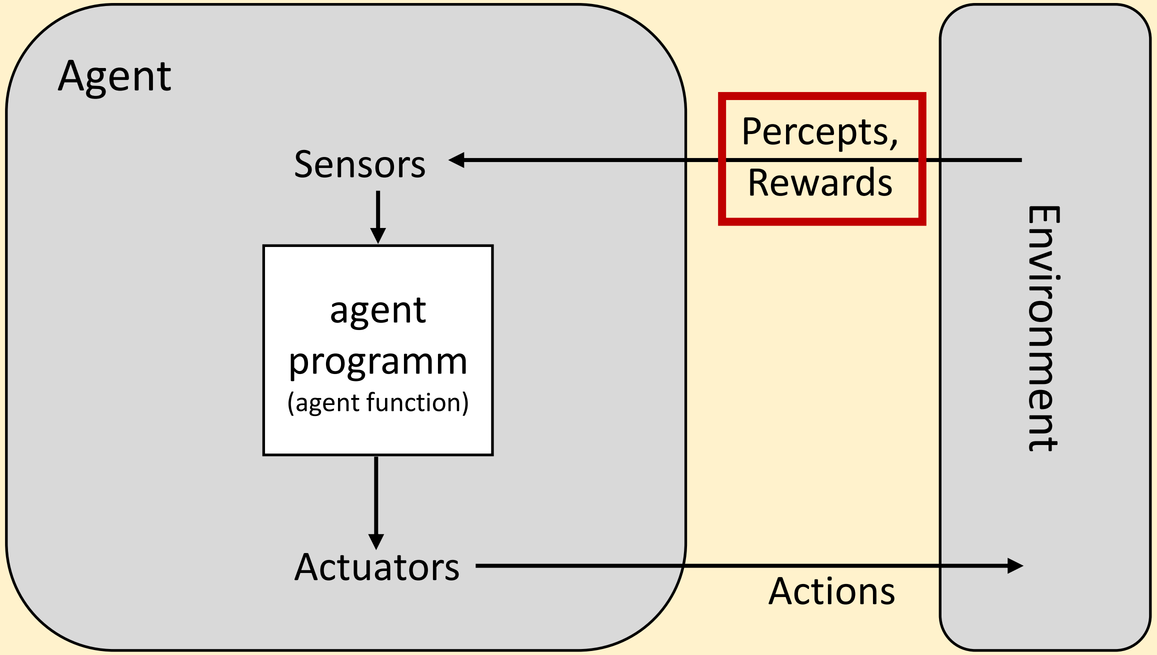
- Run an instance of the CartPole-v0 environment for 1000 timesteps, rendering the environment at each step

```
import gym

env = gym.make('CartPole-v0')
env.reset()

for _ in range(1000):
    env.render()
    env.step(env.action_space.sample())
env.close()
```



Adapted from ↗Openai Gym documentation

Reinforcement Learning Problems



*How an agent can become proficient in an unknown environment, given only its percepts and occasional rewards (*Rusell & Norvig, 2016*).*

Adapted from Rusell, S., & Norvig, P. (2016)

Observations

- The `env.step` functions allows our agent to percept the environment

```
observation, reward, done, info = env.step(action)
```

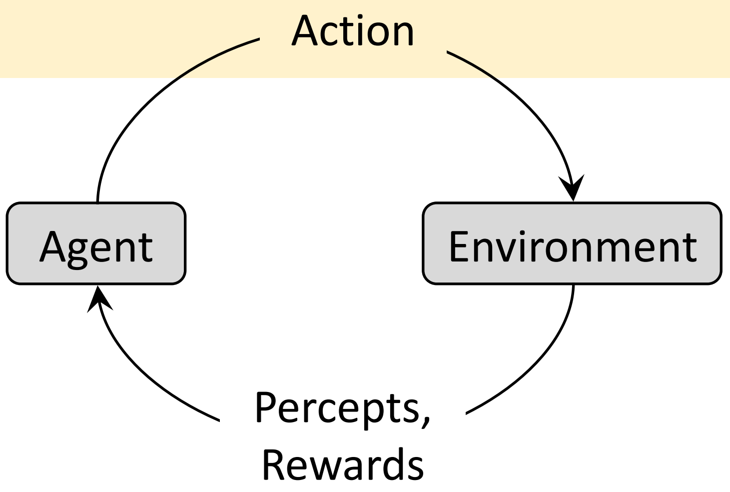
Parameters

`observation`	An environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.
`reward`	Amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
`done`	Whether it's time to reset the environment again. Most (but not all) tasks are divided up into well-defined episodes, and done being True indicates the episode has terminated. (For example, perhaps the pole tipped too far, or you lost your last life.)
`info`	Diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment's last state change). However, official evaluations of your agent are not allowed to use this for learning.

Adapted from ↗Openai Gym documentation

Implement Agent-Environment-Interaction

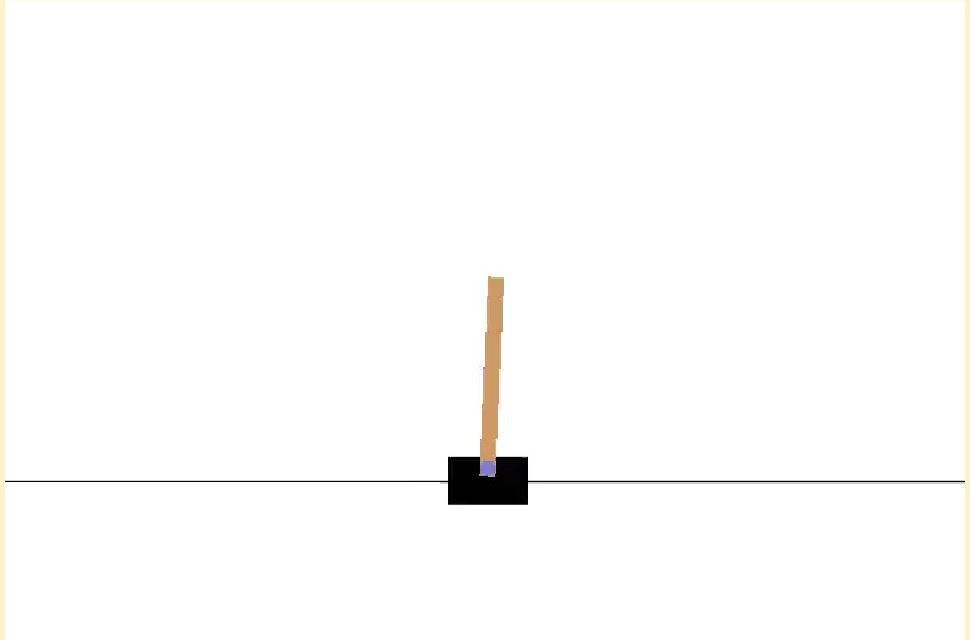
- Let us now implement the agent-environment loop from lecture 2. Each timestep, the agent chooses an `action`, and the `environment` returns an `observation` and a `reward`.




```
import gym
env = gym.make("CartPole-v0")
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```

Adapted from ↗Openai Gym documentation; Rusell, S., & Norvig, P. (2016)

Agent-Environment-Interaction



```
…
[-0.061586   -0.75893141  0.05793238  1.15547541]
[-0.07676463 -0.95475889  0.08104189  1.46574644]
[-0.0958598  -1.15077434  0.11035682  1.78260485]
[-0.11887529 -0.95705275  0.14600892  1.5261692 ]
[-0.13801635 -0.7639636   0.1765323   1.28239155]
[-0.15329562 -0.57147373  0.20218013  1.04977545]

Episode finished after 14 timesteps

[-0.02786724  0.00361763 -0.03938967 -0.01611184]
[-0.02779488 -0.19091794 -0.03971191  0.26388759]
[-0.03161324  0.00474768 -0.03443415 -0.04105167]
…
```

Adapted from ↗Openai Gym documentation; Rusell, S., & Norvig, P. (2016)

Spaces

- Run an instance of the `CartPole-v0` environment for 1000 timesteps, rendering the environment at each step

```
>>> import gym
>>> env = gym.make('CartPole-v0')
>>> print(env.action_space)
Discrete(2)
```

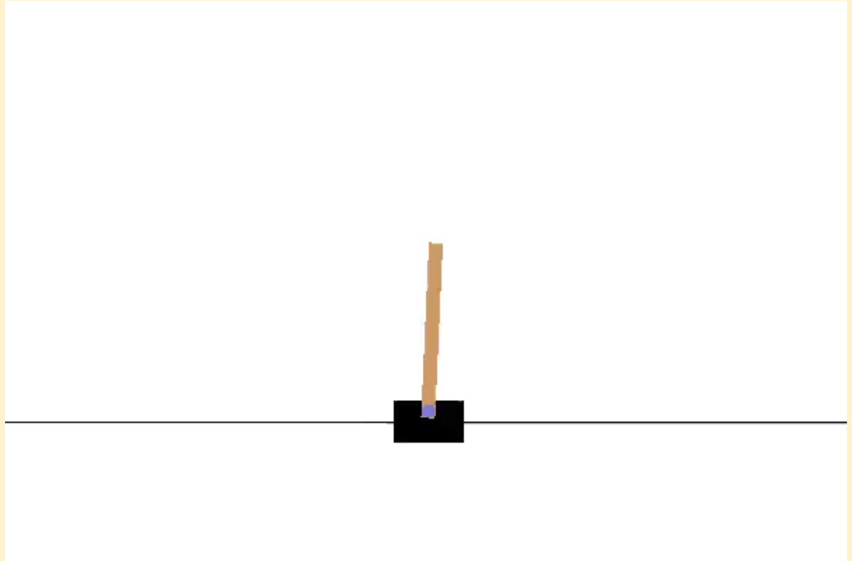
```
>>> print(env.observation_space)
Box(4,)
```

What do these numbers mean?

```
>>> print(env.observation_space.high)
array([ 2.4        ,         inf,  0.20943951,         inf])

>>> print(env.observation_space.low)
array([-2.4        ,        -inf, -0.20943951,        -inf])
```

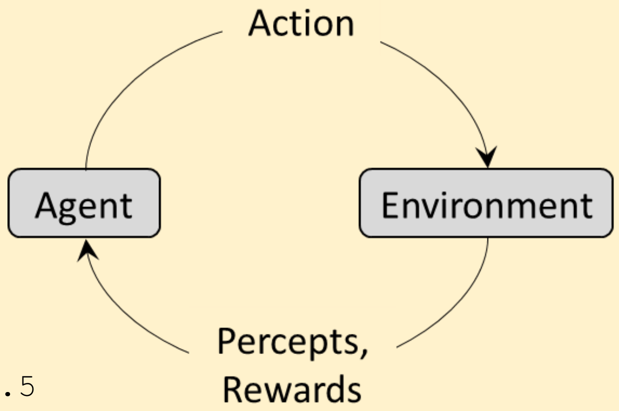
Adapted from ↗[Openai Gym](#) documentation

The `CartPole-v0` game



Num	Observation	Min	Max
0	Car position	-2.4	2.4
1	Car velocity	-Inf	Inf
2	Pole angle	-41.8°	41.8°
3	Pole velocity at tip	-Inf	Inf

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



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

Starting State: All observations are assigned a uniform random value between ±0.5

Episode Termination: Pole angle is more than ±12‟, car position is more than ±2.4 (edge of the display), or episode length is greater than 200.

Solved: Average reward is greater than or equal tp 195 over 100 executive trials

Adapted from ↗Openai Gym documentation

Cross-Entropy Learning

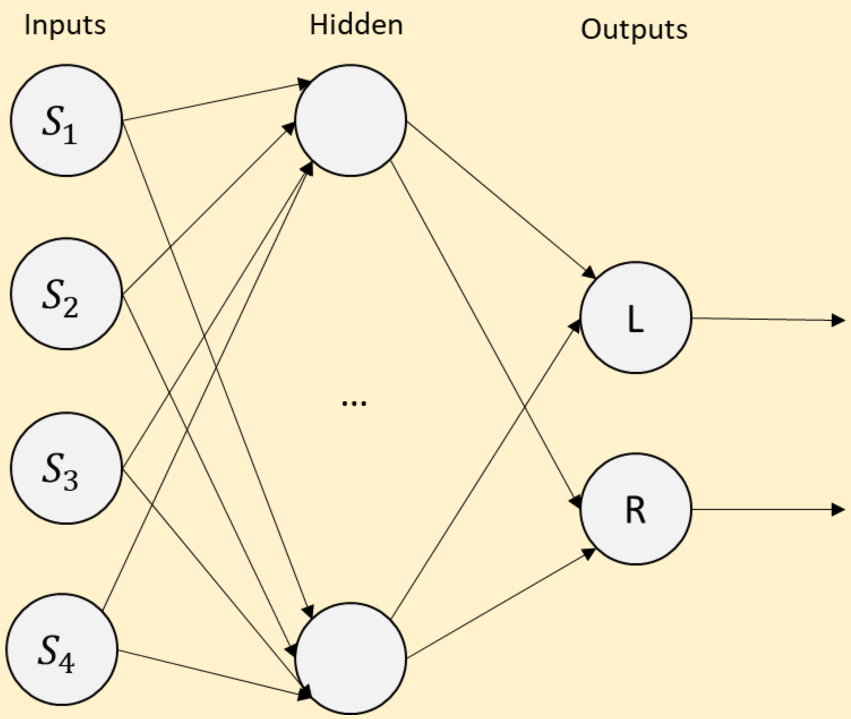
- Use past episodes to find „good“ episodes that can be used for training.

$$e_1\colon\ (L, L, L, \dots) \ \rightarrow\ r_1\colon 50$$
$$e_2\colon\ (R, L, L, \dots) \ \rightarrow\ r_1\colon 80$$
$$e_3\colon\ (L, R, R, \dots) \ \rightarrow\ r_1\colon 0$$

e_1: [-0.061586 -0.75893141 0.05793238 1.15547541] → [LEFT] → 50 → [-0.07676463 -0.95475889 0.08104189 1.46574644]

Cross-Entropy Learning

- Which we want to use with an ANN:



Live Demo

Play the Games on your own

- Play the different games on your own

```
import gym
from gym.utils.play import play

env = gym.make("Pong-v4")
play(env, zoom=5)

env = gym.make("Alien-v0")
play(env, zoom=3)

env = gym.make("Assault-ram-v0")
play(env, zoom=3)

…
```

Adapted from ↗[Openai Gym](#) documentation

Live Demo

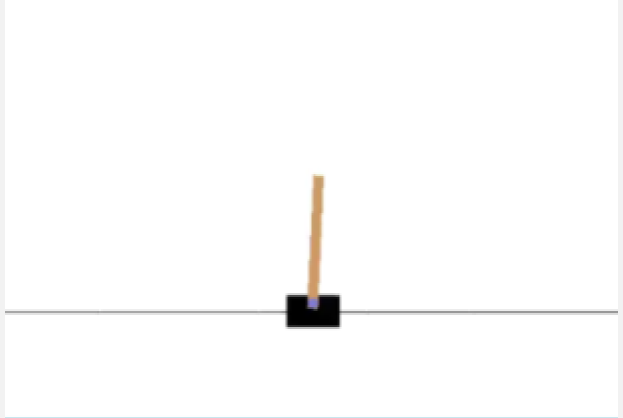
Classroom Case

Case

After your time as a trainee you apply as AI specialist in the *Research and Development Department*. Your future team is responsible for machine learning and automation. Before you can start, you have to solve the following recruitment test from your new team:

Task: Implement an intelligent agent that can solve the `CartPole-v0` game from Openai's `gym` package environments. Explain your design decisions and your code!

- For that purpose get familiar with the agent problem. Install the `gym` package and implement a simple agent class which can be used to play the `CartPole-v0` game. Alternatively you can skip tis step and use the agent template `Lectorial 5 – Intelligent Agent Template.py`, which contains a ready-to-go agent class for this task for some first tests
- Then implement an intelligent agent that is able to solve the problem successfully. In this exercise we will use Cross-entropy learning with an ANN implementation from `tensorflow`. However, you can also use other algorithms to implement an intelligent agent. There is no – perfect – just be prepared for a discussion with your new team from the R&D department ☺ (we will discuss the different solutions in this lectorial)



CartPole-v0
Balance a pole on a cart

Coding Session

Classroom Case

- A ready to go agent template for the ai gym

```
class Agent:
    def __init__(self, env):
        self.env = env

    def get_action(self):
        action = self.env.action_space.sample()
        return action

    def play(self, episodes):
        rewards = [0.0 for i in range(episodes)]

        for i_episode in range(episodes):
            observation = env.reset()
            score = 0.0

            for t in range(100):
                self.env.render()
                action = self.get_action()
                observation, reward, done, info = env.step(action)
                score += reward
                if done:
                    rewards[i_episode] = score
                    print("Scored {} in episode {}".format(score, i_episode+1))
                    break
        return rewards
```

Coding Session

Classroom Case

- Raw structure of our agent:

```
class Agent:
    def __init__(self, env):
        self.env = env
        self.observations = 4
        self.actions = 2
        self.model = self.generate_model(num_input_dim = self.observations, num_output_dim = self.actions)

    def generate_model(self, num_input_dim, num_output_dim):
        pass

    def get_action(self, observation):
        pass

    def get_samples(self, num_episodes):
        pass

    def filter_episodes(self, rewards, episodes, percentile):
        pass

    def train(self, percentile, num_iterations, num_episodes):
        pass

    def play(self, num_episodes):
        pass
```

Classroom Case

Case

Please try to understand how the different phases of AI modelling influence the final result:

- Compute the number of discrete actions (`env.action_space.n`) and observations (`env.observation_space.shape[0]`) for the `get_model` method dynamically from the environment
- Increase the number of nodes and iterations to improve the agent performance

{ Just Keep Coding }