Źródła:
https://gym.openai.com/docs/
https://www.oreilly.com/learning/introduction-to-reinforcement-learning-and-openai-gym
http://kvfrans.com/simple-algoritms-for-solving-cartpole/
https://github.com/openai/gym#basics
https://gist.github.com/kamil-stasiak/a021cd1fa2cfecd34241eebaae6564d7
https://goo.gl/iW78Ag

Tworzenie wirtualnego środowiska Python 3

Arch: https://wiki.archlinux.org/index.php/Python/Virtual_environment
Ubuntu: http://docs.python-guide.org/en/latest/dev/virtualenvs/#lower-level-virtualenv
PyCharm: https://www.jetbrains.com/help/pycharm-edu/creating-virtual-environment.html

Instalacja Gym

Jeśli korzystamy z PyCharm możemy zainstalować paczki do pythona bez użycia konsoli, klikając w opcjach:
https://www.jetbrains.com/help/pycharm/installing-uninstalling-and-upgrading-packages.html
Jeśli nie, to:
W terminalu, najlepiej po włączeniu wirtualnego środowiska:

```
(venv)$ pip install gym
```

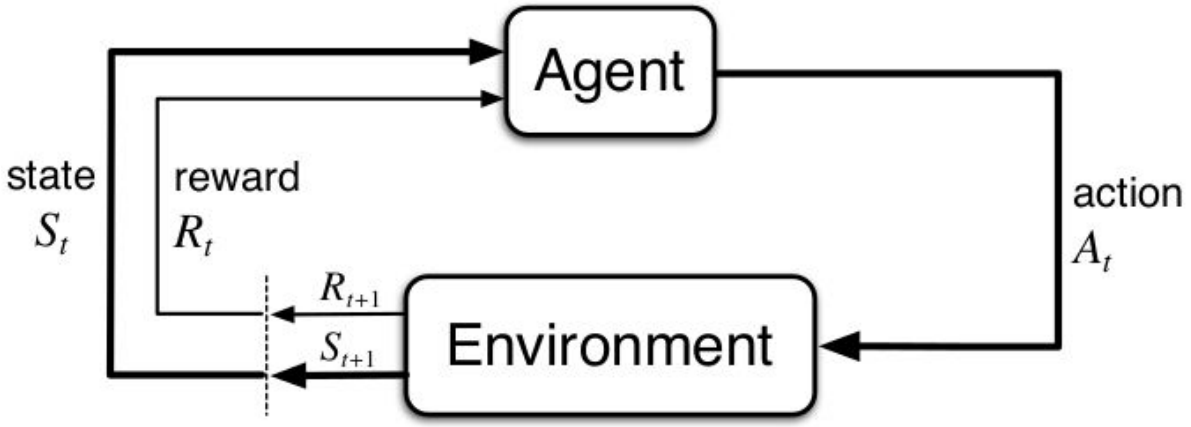
Aby zainstalować dodatkowe środowisko, np. Atari

```
(venv)$ pip install gym[atari]
```

Podstawy OpenAI Gym

Gym jest to zestaw narzędzi do opracowywania i porównywania algorytmów uczenia maszynowego ze wzmocnieniem.

Uczenie ze wzmocnieniem jest to podejście obliczeniowe, w którym agent wchodzi w interakcje z otoczeniem poprzez wykonywanie akcji w celu zmaksymalizowania końcowego wyniku.



Agent (program, który piszemy) wykonuje akcję na środowisku (np. ruch w lewo w grze PacMan). Dostaje informację zwrotną od środowiska: nagroda za wykonany ruch, informacje o aktualnym stanie środowiska (np. gdzie się znajduje gracz itp.). Na podstawie tych wartości oraz wcześniejszych doświadczeń agent podejmuje decyzję o tym jaki ruch wykonać aby wygrać.

OpenAi Gym dostarcza nam wiele środowisk (gier), na których możemy ćwiczyć algorytmy maszynowego uczenia.

Stworzenie środowiska:

env = gym.make('nazwa_środowiska') #np. 'CartPole-v0'

Główne metody:

- env.**step**(action)- wykonanie ruchu/czynności, zwraca informacje:
  - observation (object)- obiekt (specyficzny dla środowiska) reprezentujący obserwację środowiska. Są to dane które opisujące stan środowiska, np. kąt nachylenia, prędkość poruszania się, współrzędne samochodu który sterujemy, współrzędne piłki którą mamy odbić. Są to dane na podstawie których nasz agent będzie podejmował decyzję odnośnie następnego ruchu.
  - reward (float)- nagroda za wykonany ruch. Agent dąży do tego aby otrzymać jak najwięcej punktów. Informują o tym czy był to dobry ruch.
  - done (boolean)- nie można wykonać więcej ruchów, zazwyczaj jest to spowodowane przegraną. Każde środowisko ma określone warunki zakończenia.
  - info (dict)- informacje diagnostyczne do debugowania.
- env.**reset**()- resetuje środowisko do stanu początkowego, zwraca obiekt *observation*.
- env.**render**()- renderowanie aktualnego stanu środowiska.

Przykład: 'CartPole-v0': utrzymanie pręta w pozycji stojącej, to czarne to wózek którym możemy poruszać w prawo lub w lewo, pręt to to brązowe.

```
env = gym.make('CartPole-v0')
observation, reward, done, info = env.step(0)
```

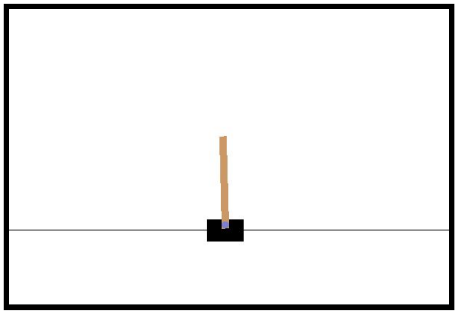
observation jest to wektor składający się z 4 wartości:
np. [0.03247529 -0.00372142 -0.04940884 0.00957897]
type: 'numpy.ndarray'- klasa z biblioteki numpy, wektor n-wymiarowy

Num	**Observation**	**Min**	**Max**
0	Pozycja wagonika	-2.4	2.4
1	Prędkość wagonika	-Inf	Inf
2	Kąt pręta	~ -41.8°	~ 41.8°
3	Prędkość czubka pręta	-Inf	Inf

reward- 1 za każdy krok aż do zakończenia
done- warunki zakończenia:

1. kąt pręta ±20.9°
2. pozycja wagonika jest większa niż ±2.4
3. długość epizodu przekroczyła wartość 200 (ilość wykonanych ruchów)

render() wyświetla:



Przykład 100 losowych kroków dla ‘CartPole-v0’.

```
import gym
import time
env = gym.make('CartPole-v0')
#observation = env.reset()
env.reset() #zawsze na początku resetujemy środowisko do stanu początkowego
for t in range(100):
    env.render() #wyświetlamy sobie środowisko
    time.sleep(1) #wait 1 second

   #bierzemy losową akcję z przestrzeni możliwych akcji w tym środowisku. Dzięki
przestrzeni akcji wiemy jakie są dostępne ruchy, pozwala to na tworzenie agentów, którzy są
uniwersalni.
    action = env.action_space.sample()
```

```
observation, reward, done, info = env.step(action) #wykonujemy akcje/krok
print(observation) #wypisujemy do konsoli wartosc obserwacji
if done: #jesli wystapil warunek konczacy to zakoncz pętlę
    print("Episode finished after {} timesteps".format(t+1))
    break
```

Environment spaces

Każde środowisko posiada swoje przestrzenie (space), które opisują możliwe akcje/ruchy (action_space) oraz obserwacje (observation_space).
Rodzaje przestrzeni (space):

- Discrete(n)- jest to zbiór {0, 1, …, n-1}.
- Box()- obiekt reprezentujący n-wymiarowe “pudełko” (każda z kolejnych liczb mówi o wymiarze macierzy- Box(n1, n2, …, nk) jest to macierz wymiaru n1 x n2 x … x nk)
  Box(4,) -> [a, b, c, d] wektor wymiaru 1x4
  Box(2, 2) -> [[a, b],
  [c, d]] macierz wymiaru 2x2

W poprzednim przypadku wybieraliśmy losową akcję z *env.action_space*.
Np. dla ‘CartPole’

```
print(env.action_space)
```

zwraca obiekt

```
Discrete(2)
```

Jest to zbiór {0, 1}, czyli możliwe ruchy naszego agenta to 0 i 1 (env.step(0) lub env.step(1)).
Z kolei

```
print(env.observation_space)
```

zwraca obiekt

```
Box(4,)
```

Jest to wektor rozmiaru 4: [a, b, c, d].