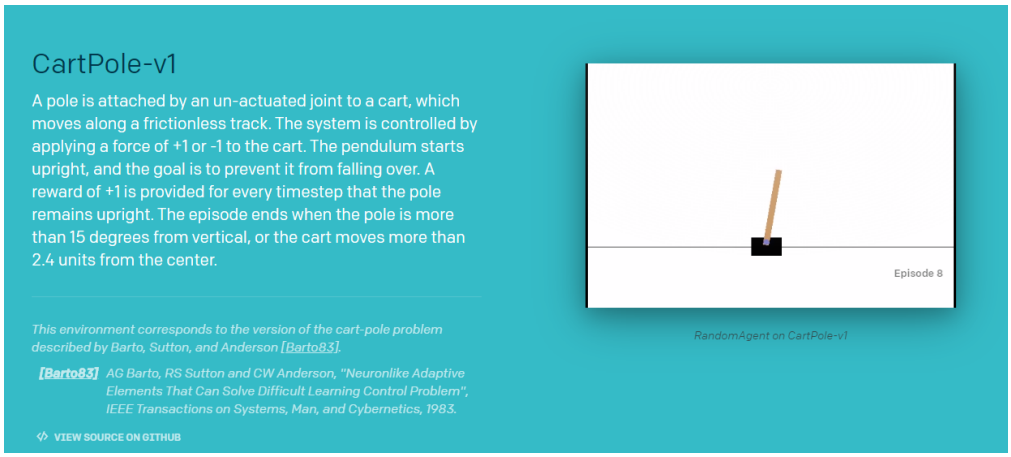
Project B (OpenAI Gym agent) First Report

-----Agent with predefined rules

2017201980
李平山
10/18

Introductions:

Using 'CartPole-v1' environment in gym, game introduction:



[Retrieved from https://gym.openai.com/envs/CartPole-v1/]

Test Environment:

- windows 10, python 3,

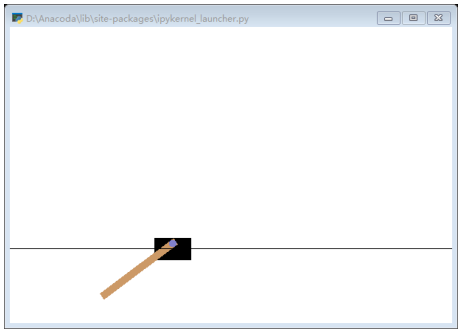
Experimental Procedure：

1）A random Demo on 'CartPole-v1':

action = env.action_space.sample()

```
In [8]: import gym

env = gym.make('CartPole-v1')
env.reset()
for _ in range(500):
    env.render()
    env.step(env.action_space.sample()) # take a random action
env.close()
```



[find attached demo video '_random.mp4']

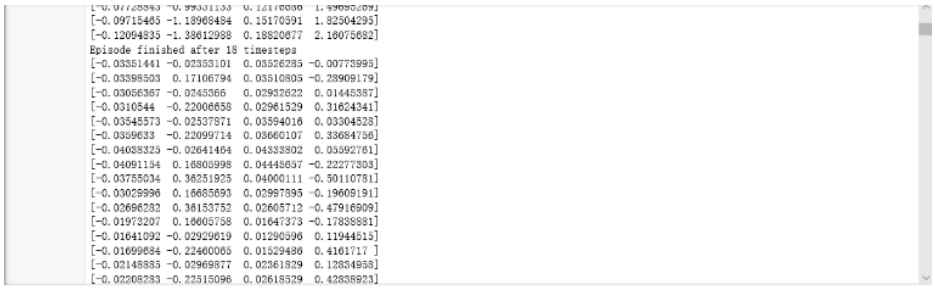
2) See observations on random agent:

2.1 some important elements on the environment:

- Observation(object): an environment-specific object representing your observation of the environment.
- Reward(float): amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
- Done(boolean): 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.\
- Info(dict): diagnostic information useful for debugging. It can sometimes be useful for learning

2.2 results on random agents

```
In [8]: 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()
```



[a snapshot on the output observations]

Count the timesteps of random agent after 20 episodes:

```
Episode finished after 28 timesteps
Episode finished after 28 timesteps
Episode finished after 10 timesteps
Episode finished after 14 timesteps
Episode finished after 18 timesteps
Episode finished after 12 timesteps
Episode finished after 14 timesteps
Episode finished after 13 timesteps
Episode finished after 12 timesteps
Episode finished after 12 timesteps
Episode finished after 10 timesteps
Episode finished after 28 timesteps
Episode finished after 37 timesteps
Episode finished after 12 timesteps
Episode finished after 23 timesteps
Episode finished after 24 timesteps
Episode finished after 32 timesteps
Episode finished after 41 timesteps
Episode finished after 22 timesteps
Episode finished after 15 timesteps
Average timesteps: 20.25
```

We can see the average result on random agent: 20.25

3) An agent based on greedy rules:

- A very simple naïve idea：actions of the agent cart changes frame-by-frame based and only based on the last action it took, heading to it's opposite direction.

$$ACTION^{n+1} = ACTION^{N} \wedge 1$$

In [11]:
```
import gym
env = gym.make('CartPole-v1')
tmp = 0;

for i_episode in range(20):
    observation = env.reset()
    action = 1;
    for t in range(100):
        env.render()
        #print(observation)
        #action = env.action_space.sample()
        action = action ^ 1
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            tmp += t+1
            break
print("Average timesteps: {} ".format(tmp/20))
env.close()
```

```
Episode finished after 24 timesteps
Episode finished after 30 timesteps
Episode finished after 40 timesteps
Episode finished after 48 timesteps
Episode finished after 60 timesteps
Episode finished after 60 timesteps
Episode finished after 29 timesteps
Episode finished after 31 timesteps
Episode finished after 31 timesteps
Episode finished after 29 timesteps
Episode finished after 25 timesteps
Episode finished after 24 timesteps
Episode finished after 25 timesteps
Episode finished after 24 timesteps
Episode finished after 61 timesteps
Episode finished after 45 timesteps
Episode finished after 22 timesteps
Episode finished after 28 timesteps
Episode finished after 31 timesteps
Episode finished after 31 timesteps
Average timesteps: 34.9
```

Still, count the timesteps of greedy agent after 20 episodes. We can see the average result on random agent: 34.9: Better performance.

4) An agent based on predefined rules:

Define function next move, based on pre_action and last_observation:

```
def next_move(observation,pre):
    return (observation[1] < -0.02 or(observation[1] <= 0 and pre == 0))
```

In [43]:
```
import gym
env = gym.make('CartPole-v1')
tmp = 0;

def next_move(observation,pre):
  return (observation[1] < -0.02 or(observation[1] <= 0 and pre == 0))

for i_episode in range(20):
    observation = env.reset()
    action = 0
    env.step(action)
    tmp = tmp + 1
    for t in range(100):
        env.render()
        #print(observation)
        #action = env.action_space.sample()
        #action = action ^ 1
        tt = action
        observation, reward, done, info = env.step(action)
        action = next_move(observation,tt)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            tmp += t+1
            break
print("Average timesteps: {} ".format((tmp+20)/20))
env.close()
```

```
Episode finished after 34 timesteps
Episode finished after 31 timesteps
Episode finished after 47 timesteps
Episode finished after 32 timesteps
Episode finished after 40 timesteps
Episode finished after 36 timesteps
Episode finished after 24 timesteps
Episode finished after 30 timesteps
Episode finished after 46 timesteps
Episode finished after 34 timesteps
Episode finished after 40 timesteps
Episode finished after 33 timesteps
Episode finished after 31 timesteps
Episode finished after 51 timesteps
Episode finished after 41 timesteps
Episode finished after 59 timesteps
Episode finished after 68 timesteps
Episode finished after 38 timesteps
Episode finished after 33 timesteps
Episode finished after 35 timesteps
Average timesteps: 41.15
```

We can see the average result on random agent: 41.15: about the same result as greedy_agent, better performance than random_agent.