In [6]:

```
from cartpole import CartPoleEnv
import numpy as np
import random

env = CartPoleEnv()
env.reset()



def discretize(val,bounds,n_states):
    discrete_val = 0
    if val <= bounds[0]:
        discrete_val = 0
    elif val >= bounds[1]:
        discrete_val = n_states-1
    else:
        discrete_val = int(round((n_states-1)*((val-bounds[0])/(bounds[1]-bounds[0]))))
    return discrete_val



def discretize_state(vals, s_bounds, n_s):
    discrete_vals = []
    for i in range(len(n_s)):
        discrete_vals.append(discretize(vals[i], s_bounds[i], n_s[i]))
    return np.array(discrete_vals, dtype=np.int)



def epsilon_greedy_action(env, Q, state, epsilon=0.3):
    n = random.uniform(0, 1)
    if n<= epsilon:
        return np.random.randint(env.action_space.n)
    else:
        return np.argmax(Q[tuple(state)])
```

In [11]:

```
def SARSA_Q(env, episodes=1000, gamma=0.91, alpha=0.1):
    n_s = np.array([7, 7, 7, 7])
    n_a = env.action_space.n
    Q = np.zeros(np.append(n_s, n_a))

    # tablica zwierająca granice przedziałów
    s_bounds = np.array(list(zip(env.observation_space.low, env.observation_space.high)))
    s_bounds[1] = (-1.0, 1.0)
    s_bounds[3] = (-1.0, 1.0)
    # konieczna konwersja typu
    s_bounds = np.dtype('float64').type(s_bounds)

    for i in range(episodes):
        finished = False
        obs = env.reset()

        S = discretize_state(obs, s_bounds, n_s)
        A = epsilon_greedy_action(env, Q, S)
        #print("======================================= ", i)
        while not finished:
            obs,R,finished,info = env.step(A)
            next_S = discretize_state(obs,s_bounds,n_s)
            next_A = epsilon_greedy_action(env, Q, next_S)


            indices = tuple(np.append(S,A))
            next_indices = tuple(np.append(next_S,next_A))
            #print(indices)
            Q[indices] += alpha * (R + gamma * Q[next_indices] - Q[indices])

            #print("S:{0} A:{1}".format(S,A))
            #print("Sn:{0} An:{1}".format(next_S,next_A))
            #print("Ns: ",n_s)
```

```
            #print(" ")
            S = next_S
            A = next_A


    return Q
```

In [12]:

```
Q = SARSA_Q(env,2000)
print(np.max(Q))  #zaokrąglamy do dwóch miejsc po przecinku
```

10.534362432870406

In [ ]:

In [ ]: