```
"""
CS 229 Machine Learning
Question: Reinforcement Learning - The Inverted Pendulum
"""
from __future__ import division, print_function
from math import sin, cos, pi
import matplotlib.pyplot as plt
import matplotlib.patches as patches

class CartPole:
    def __init__(self, physics):
        self.physics = physics
        self.mass_cart = 1.0
        self.mass_pole = 0.3
        self.mass = self.mass_cart + self.mass_pole
        self.length = 0.7 # actually half the pole length
        self.pole_mass_length = self.mass_pole * self.length

    def simulate(self, action, state_tuple):
        """
        Simulation dynamics of the cart-pole system

        Parameters
        ----------
        action : int
            Action represented as 0 or 1
        state_tuple : tuple
            Continuous vector of x, x_dot, theta, theta_dot

        Returns
        -------
        new_state : tuple
            Updated state vector of new_x, new_x_dot, nwe_theta, new_theta_dot
        """
        x, x_dot, theta, theta_dot = state_tuple
        costheta, sintheta = cos(theta), sin(theta)
        # costheta, sintheta = cos(theta * 180 / pi), sin(theta * 180 / pi)

        # calculate force based on action
        force = self.physics.force_mag if action > 0 else (-1 * self.physics.force_mag)

        # intermediate calculation
        temp = (force + self.pole_mass_length * theta_dot * theta_dot * sintheta) / self.mass
        theta_acc = (self.physics.gravity * sintheta - temp * costheta) / (self.length * (4/3 - self.mass_pole *
costheta * costheta / self.mass))

        x_acc = temp - self.pole_mass_length * theta_acc * costheta / self.mass

        # return new state variable using Euler's method
        new_x = x + self.physics.tau * x_dot
        new_x_dot = x_dot + self.physics.tau * x_acc
        new_theta = theta + self.physics.tau * theta_dot
        new_theta_dot = theta_dot + self.physics.tau * theta_acc
        new_state = (new_x, new_x_dot, new_theta, new_theta_dot)

        return new_state

    def get_state(self, state_tuple):
        """
        Discretizes the continuous state vector. The current discretization
        divides x into 3, x_dot into 3, theta into 6 and theta_dot into 3
        categories. A finer discretization produces a larger state space
        but allows for a better policy

        Parameters
        ----------
        state_tuple : tuple
            Continuous vector of x, x_dot, theta, theta_dot

        Returns
        -------
        state : int
```

```
            Discretized state value
        """
        x, x_dot, theta, theta_dot = state_tuple
        # parameters for state discretization in get_state
        # convert degrees to radians
        one_deg = pi / 180
        six_deg = 6 * pi / 180
        twelve_deg = 12 * pi / 180
        fifty_deg = 50 * pi / 180

        total_states = 163
        state = 0

        if x < -2.4 or x > 2.4 or theta < -twelve_deg or theta > twelve_deg:
            state = total_states - 1 # to signal failure
        else:
            # x: 3 categories
            if x < -1.5:
                state = 0
            elif x < 1.5:
                state = 1
            else:
                state = 2
            # x_dot: 3 categories
            if x_dot < -0.5:
                pass
            elif x_dot < 0.5:
                state += 3
            else:
                state += 6
            # theta: 6 categories
            if theta < -six_deg:
                pass
            elif theta < -one_deg:
                state += 9
            elif theta < 0:
                state += 18
            elif theta < one_deg:
                state += 27
            elif theta < six_deg:
                state += 36
            else:
                state += 45
            # theta_dot: 3 categories
            if theta_dot < -fifty_deg:
                pass
            elif theta_dot < fifty_deg:
                state += 54
            else:
                state += 108
        # state += 1 # converting from MATLAB 1-indexing to 0-indexing
        return state

    def show_cart(self, state_tuple, pause_time):
        """
        Given the `state_tuple`, displays the cart-pole system.

        Parameters
        ----------
        state_tuple : tuple
            Continuous vector of x, x_dot, theta, theta_dot
        pause_time : float
            Time delay in seconds

        Returns
        -------
        """
        x, x_dot, theta, theta_dot = state_tuple
        X = [x, x + 4*self.length * sin(theta)]
        Y = [0, 4*self.length * cos(theta)]
        plt.close('all')
        fig, ax = plt.subplots(1)
```

```
        plt.ion()
        ax.set_xlim(-3, 3)
        ax.set_ylim(-0.5, 3.5)
        ax.plot(X, Y)
        cart = patches.Rectangle((x - 0.4, -0.25), 0.8, 0.25,
                         linewidth=1, edgecolor='k', facecolor='cyan')
        base = patches.Rectangle((x - 0.01, -0.5), 0.02, 0.25,
                         linewidth=1, edgecolor='k', facecolor='r')
        ax.add_patch(cart)
        ax.add_patch(base)
        x_dot_str, theta_str, theta_dot_str = '\\dot{x}', '\\theta', '\\dot{\\theta}'
        ax.set_title('x: %.3f, $%s$: %.3f, $%s$: %.3f, $%s$: %.3f'\
                                %(x, x_dot_str, x_dot, theta_str, theta, theta_dot_str, x))
        plt.show()
        plt.pause(pause_time)

class Physics:
    gravity = 9.8
    force_mag = 10.0
    tau = 0.02 # seconds between state updates
```