In [ ]:

```
################## seiar ###################################
def seiar(y, t, beta, psi, nu, kappa, alpha, tau, p, eta, f, epsilon, q, delta):
    S, E, I, A, R = y
    Lambda = epsilon * E + (1 - q) * I + delta * A
    dSdt = -beta * S * Lambda - psi * nu * S
    dEdt = beta * S * Lambda - kappa * E
    dIdt = p * kappa * E - alpha * I - tau * I
    dAdt = (1 - p) * kappa * E - eta * A
    dRdt = f * alpha * I + tau * I + eta * A + psi * nu * S
    return [dSdt, dEdt, dIdt, dAdt, dRdt]

class SeiarEnvironment(gym.Env):
    beta: float
    psi: float
    nu_daily_max: float
    nu_total_max: float
    kappa: float
    alpha: float
    tau: float
    p: float
    eta: float
    f: float
    epsilon: float
    q: float
    delta: float
    S0: float
    E0: float
    I0: float
    A0: float
    R0: float
    tf: float
    continuous: bool
    RepN: float
    def __init__(self, beta, psi, nu_daily_max, nu_total_max, kappa, alpha,
                 tau, p, eta, f, epsilon, q, delta,
                 S0, E0, I0, A0, R0, tf, dt, continuous, RepN):
        super(SeiarEnvironment, self).__init__()
        self.beta = beta
        self.psi = psi
        self.nu_daily_max = nu_daily_max
        self.nu_total_max = nu_total_max
        self.nu_min = 0.0
        self.kappa = kappa
        self.alpha = alpha
        self.tau = tau
        self.p = p
        self.eta = eta
        self.f = f
        self.epsilon = epsilon
        self.q = q
        self.delta = delta
        self.S0 = S0
        self.E0 = E0
        self.I0 = I0
        self.A0 = A0
        self.R0 = R0
        self.tf = tf
        self.dt = dt
        self.time = 0
        self.days = [self.time]
        self.history = [[S0, E0, I0, A0, R0]]
        self.nus = []
        self.rewards = []
        self.continuous = continuous
        self.observation_space = gym.spaces.Box(low=0, high=np.inf, shape=(5,), dtype=np.float32)
        if self.continuous:
            self.action_space = gym.spaces.Box(low=-1, high=1, shape=(1,), dtype=np.float32)
        else:
            # for dqn
            self.action_space = gym.spaces.Discrete(2)

    def reset(self):
        self.time = 0
        self.days = [self.time]
        self.state = np.array([self.S0, self.E0, self.I0, self.A0, self.R0])
        self.nus = []
        self.rewards = []
        self.history = [self.state]
        return np.array(self.state, dtype=np.float32), {}

    def action2control(self, action):
        nu = self.nu_min + (self.nu_daily_max - self.nu_min) * (action[0] + 1.0) / 2.0
        return nu

    def step(self, action):
        if self.continuous:
            nu = self.action2control(action)
        else:
            nu = self.nu_min if action == 0 else self.nu_daily_max
        self.nus.append(nu)
        S0, E0, I0, A0, R0 = self.state
        sol = odeint(seiar, [min(0, S0-nu), E0, I0, A0, R0],
                     np.linspace(0, self.dt, 101),
                     args=(self.beta, self.psi, 0,
                           self.kappa, self.alpha, self.tau,
                           self.p, self.eta, self.f, self.epsilon,
                           self.q, self.delta))

        self.time += self.dt
        new_state = sol[-1, :]
        S, E, I, A, R = new_state
        self.state = new_state

        reward = - I - nu
        if np.sum(self.nus) > self.nu_total_max:
            reward -= 1000
        reward *= self.dt

        self.rewards.append(reward)
        self.days.append(self.time)
        self.history.append([S, E, I, A, R])

        done = True if self.time >= self.tf else False
        return (np.array(new_state, dtype=np.float32), reward, done, False, {})

    @property
    def dynamics(self):
        df = pd.DataFrame(dict(
                                days=self.days,
                                susceptible=[s[0] for s in self.history],
                                infected=[s[2] for s in self.history],
                                nus=self.nus + [None],
                                rewards=self.rewards + [None])

                          )
        return df
```