```
import random
from itertools import combinations
import numpy as np
from scipy.special import comb


class AoI_Trade(object):

    def __init__(self, seed=0, user_num=30, beta=0.5, slot=1, **kwargs):
        self.user_req_ind = None
        self.Last_State = None

        U# ATTENTION
        self.AoI_Max = 20
        self.Fail_Prob_Pro = 1 - np.power((1 - Fail_Prob), self.Tu)  # failure probability 失败概率
        self.E = np.zeros(self.N)
        self.C = self.Com_Factor * self.Cost_Upper_Bound  # 开销

        self.User_Req_Prob = np.zeros((self.N, self.N + 1))
        self.User_Req_Prob[:, 1:(self.N + 1)] = User_Req_Prob_Matrix.reshape(-1, 1) * Sensor_Popularity #
        self.User_Req_Prob[:, 0] = 1 - np.sum(self.User_Req_Prob[:, 1:self.N + 1],
                                              axis=1)
        # print(self.User_Req_Prob)
        self._action_space_fill()  # fill action space 填补动作空间
        # print("----------------------------------------------动作空间----------------------------------------")
            # print(self.Action_Space)  # 标记一下动作空间----------------------------------------------
        self.Action_Row = np.zeros(self.N, dtype=np.int64)  # index in action space of an action 动作空间中的索引

    def _action_space_fill(self):
        """
        fill the action space using 0 or 1 使用 0 或 1 填充动作空间
        """
        characters = np.arange(self.N)  # 传感器数量(0,1,...,9)
        indices = []
        for i in np.arange(self.M + 1):  # 支持更新的最大传感器数量(0,1,...,5)
            for item in combinations(characters, i):  # combinations 排列组合
                indices.append(item)

        for j in range(self.Actual_Action_Num):
```

1. self.Action_Space[j, indices[j]] = 1# 模拟用户的请求行为，并返回一个表示用户请求的指示器数组

c) def _user_request_ind(self):
   i. # 模拟用户请求行为
   ii. user_req_ind_ = np.zeros(self.N, dtype=np.int64) # 存储每个用户的请求指示器。
   iii. sensor_seq = np.arange(0, self.N + 1, 1) for i in range(self.N):
      1. # print(self.User_Req_Prob[i, :])
      2. user_req_ind_[i] = np.random.choice(sensor_seq, p=self.User_Req_Prob[i, :]) # 从 sensor_seq 中随机采 1 个样
   iv. return user_req_ind_ # 根据系统的状态和请求情况进行更新，计算额外的开销和更新次数，并限制状态空间中 AoI 值的上边界。

d) # 执行动作并更新系统状态，同时计算奖励、开销和下一个状态空间
e) def _step(self, action):
   i. # 检查给定的动作 action 是否在动作空间中存在.any()有一个为 True 即为 True，否则触发异常
   ii. assert ((np.arange(self.Actual_Action_Num) == action).any()) == True # .any() 有一个为 True 即为 True，否则触发异常
   iii. self.Action_Row = self.Action_Space[action, :] # 取动作空间中对应 action 的那一行
   iv. # print("----------------------------------action row----------------------------------")
   v. # print("Action_Row:", self.Action_Row)
   vi. self._update_state() # 更新缓存和用户状态
   vii. self.AoI_Table = np.zeros((self.N, self.N, self.T + 1)) # 用户的数量，传感器的数量，一般步长的持续时间
   viii. # 将 self.Last_State 中的用户 AoI 值复制到 self.AoI_Table 的第一个时隙中。
   ix. # print(self.Last_State)
   x. # 选择了除第一行之外的所有行，即选择了用户的状态行，表示了上一个状态的用户的状态空间
   xi. self.AoI_Table[:, :, 0] = self.Last_State[1:self.N + 1, :]
   xii. for i in range(self.N):
      1. for j in range(self.N):
         a) init_aoi = int(self.AoI_Table[i, j, 0]) # 初始 AOI 的值
         b) self.AoI_Table[i, j, 1:self.T] = np.linspace(init_aoi + 1, init_aoi + self.T - 1, self.T - 1,
            i. endpoint=True) # 返回数之间均匀分布的值
         c) self.AoI_Table[i, j, -1] = self.State_Space[1 + i, j]
   xiii. self.AoI_Table = np.minimum(self.AoI_Table, self.AoI_Max)
   xiv. # 计算每个传感器的平均 AoI 值 sensors，通过对 self.AoI_Table 的第三个维度求和并除以 self.T 得到。
   xv. sensors = np.sum(self.AoI_Table[:, :, 1:], axis=2) / self.T # (N,K)

xvi. # 计算每个用户的平均 AoI 成本 users，通过对 sensors 的第二个维度求和并除以 self.N 得到。
xvii. users = np.sum(sensors, axis=1) / self.N # (N,) users = Δnt 用户的平均 AoI 成本
xviii. aoi_cost = self.B1 * self.AoI_Cost
xix. energy_cost = self.B2 * self.Energy_Cost
xx. #ATTENTION
xxi. extra_cost =self.B3 * self.Extra_cost
xxii. extra_update_times = self.Extra_update_times
xxiii. #ATTENTION self.Total_Cost = aoi_cost + energy_cost
xxiv. self.Total_Cost = aoi_cost + energy_cost + extra_cost #
xxv. # TODO 添加一个资源分配的回报值
xxvi. self.reward = - self.Total_Cost # real reward 实际回报
xxvii. self.Last_State = self.State_Space.copy()

xxviii. return self.State_Space.copy(), (self.reward, aoi_cost, energy_cost, extra_cost, extra_update_times)

f) # 返回重置的状态空间
g) def reset(self):
i. # 初始态
ii. #ATTENTION self.State_Space = np.zeros((self.N + 1, self.N))
iii. self.State_Space = np.random.randint(20, 100, size=(self.N + 1, self.N)) # 状态空间初始化

iv. self.Last_State = self.State_Space.copy()
v. return self.State_Space.copy() # 返回重置的状态空间

h) # 模拟用户请求
i) def simulation_user_request(self): # 模拟用户请求
i. self.user_req_ind = self._user_request_ind()
ii. return self.user_req_ind.copy() # 返回模拟的用户请求

j) def step(self, action):
i. return self._step(action)

6. if __name__ == '__main__':
a) env = AoI_Trade()