# 3.9-C 强化学习——多智能体强化学习（MARL）

## 3.9.1 从一个智能体到多个智能体

现实中的很多问题都存在合作/协作（cooperative）、竞争（competitive）的关系，例如在各类体育运动项目，竞技游戏中的拳击项目为两个选手间的竞争关系；足球项目为两个队之间的竞争关系，但在各个队内部为协作关系；而救援队的救援任务只有成员间的协作关系等。单个智能体无法实现协作和竞争关系模拟，需要建立可以之间交互的多个（甚至百千，或更多）智能体来实现（同时存在与环境间的交互），即为多智能体强化学习（multi-agents reinforcement learning，MARL）。

以伊辛（Ising）模型为例解释 MARL。

### 3.9.1.1 伊辛模型

#### 1）伊辛模型

伊辛模型（Ising（Lenz-Ising）model），以物理学家 Ernst Ising 和 Wilhelm Lenz 命名，是统计力学（statistical mechanics）中铁磁性（Ferromagnetism）的数学模型，由处于两种状态（+1 或 -1），代表原子“自旋（spins）”磁偶极矩（ magnetic dipole moments）（描述磁性强度和方向的物理量）的离散变量组成。原子的自旋被表示在一个晶格（lattice）中，每个自旋与相邻的自旋相互作用，且相邻的自旋与该自旋相一致情况的能量低于不一致的情况，但热扰乱了这种趋势，从而产生了不同结构相（structural phases）的可能性[1]。同时可以查看*计算机视觉中的马尔可夫网*部分对伊辛模型的描述。

物理学家试图研究电子如何“排列”自己，及温度如何影响这一过程（加热磁铁超过一定温度会导致其突然失去磁力）。单个电子会产生磁场，而微小的电子会影响其附近的其它电子，使电子之间朝向一个方向排列或者相反的方向。当排列的电子数量增加，磁场增大并在材料上产生一些内部应变，电子会形成簇，称为（电）畴（domains），是指自发极化方向相同的小区域，电畴与电畴之间的边界称为畴壁。例如部分电子排列为自旋向上的畴，而部分排列为相邻自旋向下的畴，在非常局部的水平上，电子通过排列来最小化它们的能量。为了对这一现象建模，避免材料中数万亿电子之间的相互作用而仅假设电子只受其最近邻域电子的影响，这与上述邻域学习的假设完全相同。

考虑一个晶格（位置）点集合。每个晶格点都有一组相邻的点形成一个d-维晶格。对于每一个晶格点k∈Λ，存在一个离散变量σk​，使得σk​∈{−1,+1}，表示该晶格点的自旋。一个自旋构型（spin configuration），σ={σk​}k∈Λ​是为每个晶格点赋予一个自旋值。对于任意两个临近的点， i,j∈Λ，存在一个相互作用Jij​。并且，点j∈Λ有一个与其交互的外部磁场（external magnetic field）hj​。自旋构型σ的能量由哈密尔顿函数（Hamiltonian function）给出， H(σ)=−∑⟨ij⟩​Jij​σi​σj​−μ∑j​hj​σj​，式中，第一个和为相邻自旋对的和（每对计算一次）。符号⟨ij⟩表示点i和j为最近邻。μ为磁矩（magnetic moment）。注意，因为电子的磁矩与自旋是反平行的，哈密尔顿函数第二项的符号应该为正，用符号为一种惯例。构型概率（configuration probability）由玻尔兹曼分布（Boltzmann distribution）给出，为参数温度的倒数β≥0： Pβ​(σ)=Zβ​e−βH(σ)​，式中，β=(kB​T)−1，归一化常数为Zβ​=∑σ​e−βH(σ)，为配分函数（partition function）（在物理学中，配分函数描述了处于热力学平衡状态系统的统计性质）。物理学中的玻尔兹曼分布即为 SoftMax 函数，可以查看*SoftMax回归（函数、归一化指数函数）*部分内容。自旋的函数可以表述为⟨f⟩β​=∑σ​f(σ)Pβ​(σ)，是f的期望值（均值）。构型概率Pβ​(σ)表示系统（在平衡状态下）在构型σ状态下的概率。

依据哈密尔顿函数H(σ)，伊辛模型可以根据相互作用的符号分类，对于点对，

，相互作用称为铁磁的（ferromagnetic）；

，相互作用称为反铁磁的（antiferromagnetic）；

，表示自旋间并无相互作用。

在铁磁的伊辛模型中，自旋倾向于一致，即相邻自旋的构型为相同符号的概率更大；在反铁磁的伊辛模型中，相邻自旋倾向于相反的符号。

也解释了点自旋与外部磁场的相互作用，

，点的自旋倾向于向正方向排列；

，点的自旋倾向于向反方向排列；

，没有外部的影响。

伊辛模型通常在没有外部磁场与晶格作用情况下检验，因此  
H(σ)可以简化为H(σ)=−∑⟨i j⟩​Jij​σi​σj​；另外，假设所有的最近邻⟨ij⟩具有相同的相互作用强度，因此可以设Jij​=J， H(σ)进一步简化为H(σ)=−J∑⟨i j⟩​σi​σj​。

#### 2）蒙特卡罗方法

蒙特卡罗（Monte Carlo，MC）方法为依赖于重复的随机抽样来获得数值结果的一类广泛计算方法。其基本的概念是使用随机性（randomness）来解决原则上可能是确定性（deterministic）的问题，通常可用于解决任何具有概率解释的问题。根据大数定律，由某个随机变量期望值描述的积分可以通过取该变量独立样本的经验均值（empirical mean）（样本均值（sample mean））来近似[2,3,4]。当变量的概率分布被参数化时，经常使用马尔可夫蒙特卡罗（ Markov chain Monte Carlo，MCMC）采样器，其核心思想是设计一个具有给定平稳概率分布的马尔可夫链模型，即在极限情况下，由 MCMC 方法生成的样本将是来自期望(目标)分布的样本[5,6]。通过遍历定理（ergodic theorem），用 MCMC 采样器随机状态的经验测度来近似平稳分布。

另外，从满足非线性演化方程的概率分布序列中生成图形（例如伊辛模型），这些概率分布流（flows）总可被解释为马尔可夫过程的随机状态分布，其转移概率（transition probabilities）取决于当前随机状态的分布，这些模型可以看作非线性马尔可夫链随机状态规律的演化。为避免概率分布流越来越高的采样复杂度，模拟这些复杂非线性马尔可夫过程的一种自然方法是对该过程的多个副本进行采样，用采样的经验测度（empirical measures）代替演化方程中随机状态的未知分布。当迭代次数足够大时（系统趋于无穷大时），这些随机经验测度将收敛于非线性马尔可夫链随机态的确定性分布，使得样本（粒子（particles）、个体（individuals）、智能体（agents）等）间的统计相互作用消失[7]。

MC 方法各有不同，但往往遵循一个特定模式：

1. 定义一个可能输入的域（domain）；
2. 从域上的概率分布随机生成输入；
3. 对输入执行确定性操作；
4. 汇总结果。

下述代码[8]为应用 MC 方法近似计算圆周率（），过程如下：

1. 假设正方形横纵坐标区间均为[-1,+1]，则其面积为；
2. 从均匀分布（uniform distribution）中生成足够多的样本点坐标，并落于正方形内；
3. 计算落于半径为1，原点为(0,0)圆内样本点的数量；
4. 落于圆内样本点数量与总共样本数量的比值为圆形和正方形区域面积比的估计值；
5. 已知正方形面积为4，及圆和正方形的比值，因此可以求出圆形面积，进而得到值。

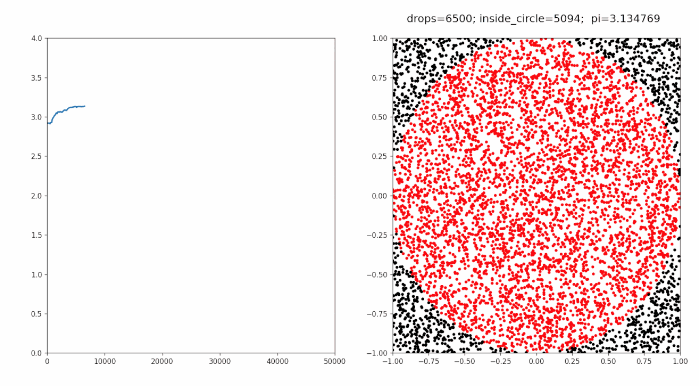
%load\_ext autoreload   
%autoreload 2   
import usda.rl as usda\_rl   
  
import numpy as np  
import torch  
from matplotlib import pyplot as plt  
  
from IPython.display import HTML  
from tqdm import tqdm

def pi\_using\_MonteCarlo(nTrials=int(10E7)):   
 # Input parameters  
 radius = 1  
 #-------------  
 # Counter for the points inside the circle  
 nInside = 0  
   
 # Generate points in a square of side 2 units, from -1 to 1.  
 XrandCoords = np.random.default\_rng().uniform(-1, 1, (nTrials,))  
 YrandCoords = np.random.default\_rng().uniform(-1, 1, (nTrials,))  
   
 for i in range(nTrials):  
 x = XrandCoords[i]  
 y = YrandCoords[i]  
 # Check if the points are inside the circle or not  
 if x\*\*2+y\*\*2<=radius\*\*2:  
 nInside = nInside + 1  
   
 area = 4\*nInside/nTrials  
 print("Value of Pi: ",area)  
   
pi\_using\_MonteCarlo()

Value of Pi: 3.14150196

定义pi\_using\_MonteCarlo\_anim()函数，实现 MC 方法计算，随采样数量增加，采样点分布和估计值的变化。

anim=usda\_rl.pi\_using\_MonteCarlo\_anim(nTrials=int(5e4))   
anim.save('../imgs/3\_9\_c/pi\_mc.gif')  
HTML(anim.to\_jshtml())



#### 3）2 维度伊辛模型的 MC 模拟

在统计学和统计物理中，Metropolis–Hastings 算法是一种从难以直接抽样的概率分布中获得随机样本序列的 MCMC 方法。该序列可用于近似分布（例如生成直方图）或计算积分（例如期望值）。Metropolis–Hastings 算法和其它 MCMC 算法通常用于从多维分布中采样，特别是高维数情况[9]。Nicholas Metropolis 等人Metropolis, Nicholas; Rosenbluth, Arianna W.; Rosenbluth, Marshall N.; Teller, Augusta H.; Teller, Edward (1 June 1953). “Equation of State Calculations by Fast Computing Machines”. The Journal of Chemical Physics. 21 (6): 1087–1092. Bibcode:1953JChPh..21.1087M.于 1953 年提出了该算法，称为 Metropolis 算法。到 1970 年，W.K. Hastings[10] 将其扩展到更一般的方法，即 Metropolis–Hastings 算法。Nicholas Metropolis 等人在其论文中描述把个粒子置于任意的构型（configuration）中，例如规则的晶格。然后按照2X→X+αξ1​;Y→Y+αξ2​ 方式移动每个粒子，式中，α为最大允许位移（此处为任意值）； 1ξ1​和2ξ2​为[-1,1]区间的随机数。在移动一个粒子之后，它同样有可能在以其原始位置为中心，边为2α的正方形内的任何地方。然后计算由粒子移动引起的系统能量变化ΔE，如果ΔE<0，即粒子移动使得系统达到较低能量的状态，因此允许粒子移动并置于新的位置；如果ΔE>0，则将以概率exp(−ΔE/kT)移动，即取一个[0,1]区间的随机数ξ，如果ξ<exp(−ΔE/kT)，将粒子移动到其新的位置。如果ξ>exp(−ΔE/kT)，则返回到原来的位置。且无论粒子是否发生了位移，系统均处于一个新的构型中。

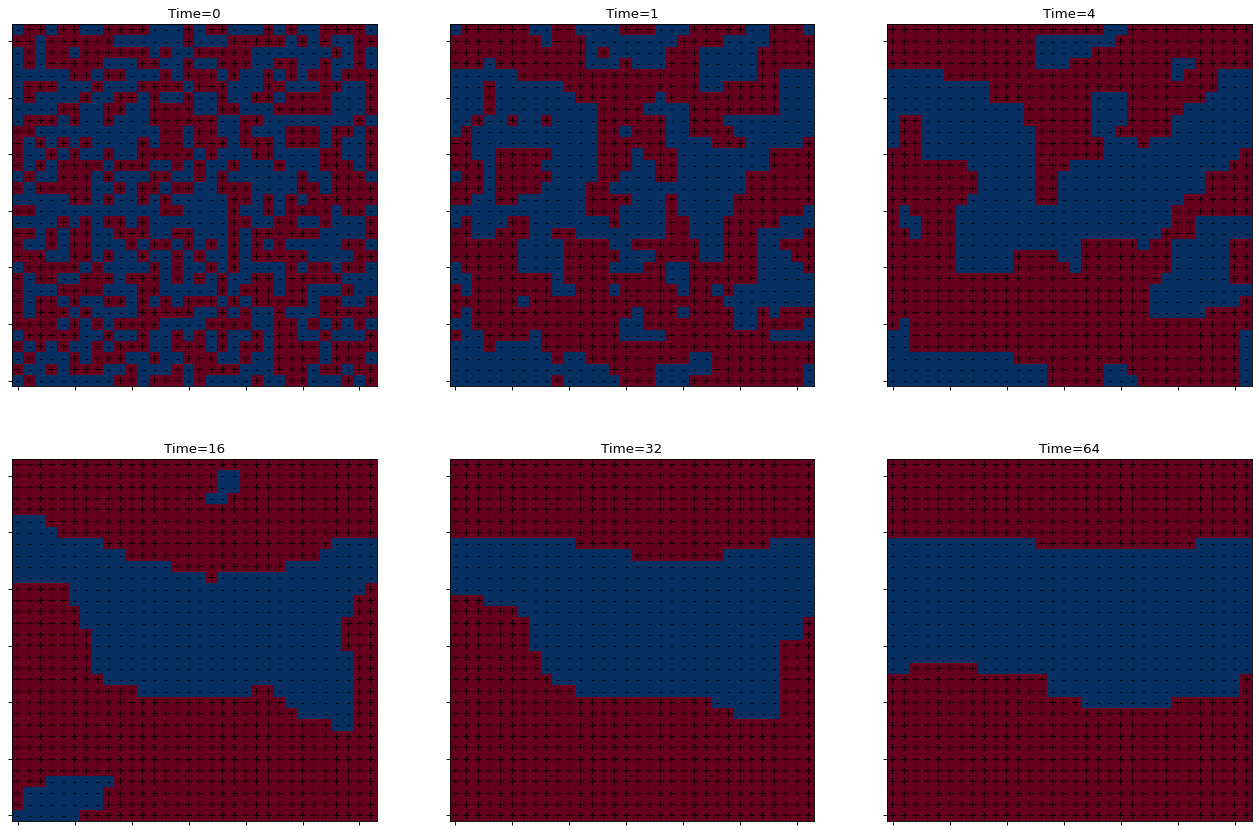
根据 Metropolis 算法，书写 2 维度伊辛模型的 MC 模拟代码[11]，其中对应 Metropolis 算法 的代码如下：

def mcmove(self, config, beta):  
 ''' This is to execute the monte carlo moves using Metropolis algorithm such that detailed balance condition is satisified'''  
 N=self.N  
 for i in range(N):  
 for j in range(N):   
 a = np.random.randint(0, N)  
 b = np.random.randint(0, N)  
 s = config[a, b]   
 nb = config[(a+1)%N,b] + config[a,(b+1)%N] + config[(a-1)%N,b] + config[a,(b-1)%N]  
 cost = 2\*s\*nb  
 if cost < 0:   
 s \*= -1  
 elif rand() < np.exp(-cost\*beta):  
 s \*= -1  
 config[a, b] = s  
 return config

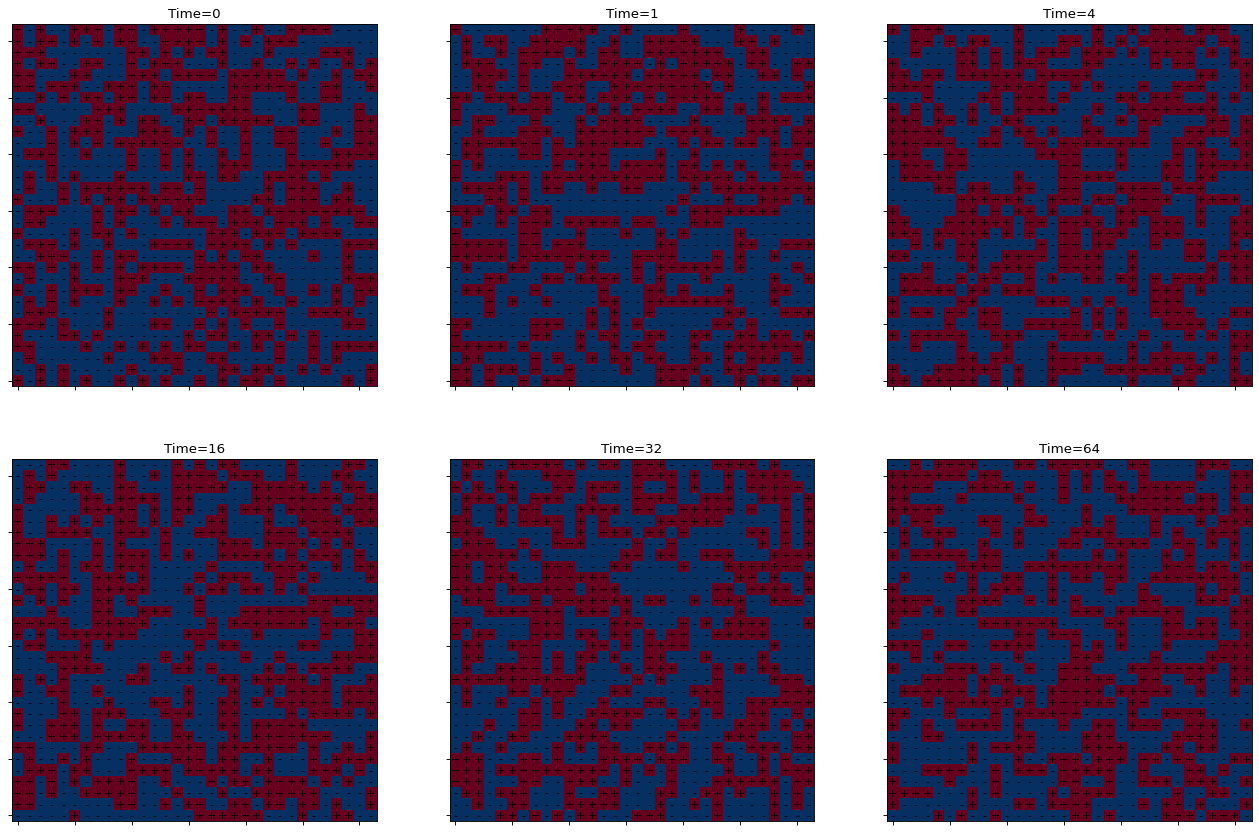
下面的试验中比较了不同温度参数temp为0.4和10的结果，当温度较低时，电子的自旋倾向于朝向一个方向，保持磁性；而当温度较高时，电子的自旋倾向于朝向相反的方向，从而失去磁性。

ising=usda\_rl.Ising\_Metropolis\_MCMC(N=32,temp=.4)  
ising.simulate(1001,idxes=[1,4,16,32,64],figsize=(20,20))

100%|███████████████████████████████████████████████| 500/500 [32:12<00:00, 3.87s/it]



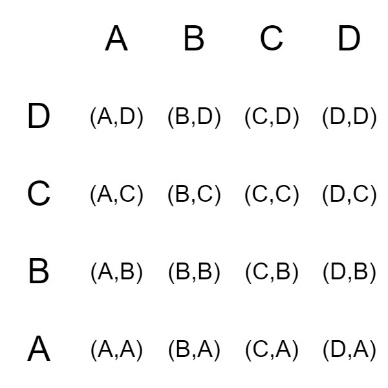
ising=usda\_rl.Ising\_Metropolis\_MCMC(N=32,temp=10)  
ising.simulate(1001,idxes=[1,4,16,32,64],figsize=(20,20))



### 3.9.1.2 伊辛模型的 MARL 模拟[12]

#### 1） 多智能体的维度问题

一个常规的 （动作价值）函数表述为，为“状态-动作”二元组对应收益的函数。当扩展到多个智能体，每个智能体都会与其它智能体交互，则智能体j的Q函数可以表述为Qj​(s,aj​,a−j​):S×Aj​×A−j​→R，式中， −j为除索引为j的智能体外的所有其它智能体。该类型的多智能体Q函数，能够保证训练收敛，从而学习到最优价值和策略函数。但是当智能体的数量很大时，则联合动作空间（ joint action-space）a−j​随着智能体的数量呈指数增长∣A∣N，例如如果一个智能体有上下左右等4个动作，那么如果有两个智能体时则动作的组合方式有42=1642=16个，为三个时就有64个，按4N的方式增长。用独热编码向量（one-hot vector）表示一个智能体的4个动作时，可表示为[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]等，那么对于4个动作的两个智能体的联合动作空间，则可一般表示为，



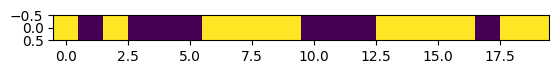
为了解决联合动作空间随智能体增加而指数级增长的问题，根据大多数环境下彼此邻近的智能体才会产生重大影响的现象，则不需要模拟环境中所有智能体的联合动作，而仅考虑每一智能体与其邻域内智能体的联合动作。即将整个联合动作空间划分为一组重叠的子空间，并只计算这些小得多的子空间值，称为邻域学习（neighborhood Q-learning）或子空间学习（subspace Q-learning ）。例如，对于有4个动作的100个智能体，其完整的联合动作空间大小为4100，这是个难以处理的数量级。而如果配置邻域大小为3，则邻域联合动作空间大小为43=6443=64，为一个可以计算的数量级。为这100个智能体的每一个均构建这样的一个邻域联合动作空间向量，并用于计算每个智能体的Q值。

2） 1 维度伊辛模型的 MARL 模拟——邻域学习

定义Ising\_MARL类，实现1维度和2维度伊辛模型的 MARL 模拟。首先创建一个二进制的数字晶格，用数字 1 表示电子自旋向上（spin-up），数字 0 表示电子自旋向下（spin-down），即一个智能体存在2个动作。因为只考虑每个电子与其邻域两个电子之间的相互作用，因此联合动作空间大小为22=4。在1维度的伊辛模型试验中，配置晶格大小为20，即为状态空间大小，随机初始化电子的自旋方向，如下。

size=(20,)  
hid\_layer=20   
epochs=200  
lr=0.001   
  
ising\_1d=usda\_rl.Ising\_MARL(size,hid\_layer,epochs,lr)  
print(ising\_1d.grid)  
plt.imshow(np.expand\_dims(ising\_1d.grid,0));

tensor([1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1],  
 dtype=torch.uint8)



“状态-动作”二元组对应收益的函数定义为每个电子与其领域两个电子之间的自旋朝向关系，如果朝向均一致则具有最大的收益；如果朝向均不同，则获得最小的收益。如下代码，

def get\_reward\_1d(self,s,a):   
 r = -1  
 for i in s:  
 if i == a:  
 r += 0.9  
 r \*= 2.  
 return r

get\_reward\_1d函数中参数s为邻域电子的自旋方向（状态）列表，在1维度伊辛模型中为对应到每个电子左右相邻的各一个电子。如果该电子位于最左或最右的晶格，则其左和右的邻域电子对应到最右和最左的晶格，即晶格是一个连续的闭环。参数为当前电子的状态。初始的收益r=-1，如果所有电子朝向一致（自旋向上后向下）有最大收益为1.6；如果邻域电子与当前电子朝向均不同，则有最小收益为-2.0；如果邻域有一个电子与当前电子朝向同，则获得收益约为-0.2。

根据邻域电子状态返回联合状态（动作），例如一个电子邻域两个电子的状态为[0,1]时（表示其中一个自旋向上，一个向下），其联合状态为[0,1,0,0]；如果为[0,0]，则为[1,0,0,0]；如果为[1,0]，则为[0,0,1,0]；如果为[1,1]，则为[0,0,0,1]。对应上述联合状态结果的计算代码如下，

def get\_substate(self,b):   
 s = torch.zeros(2)   
 if b > 0:   
 s[1] = 1  
 else:  
 s[0] = 1  
 return s  
  
def joint\_state(self,s):   
 s1\_ = self.get\_substate(s[0])   
 s2\_ = self.get\_substate(s[1])  
 ret = (s1\_.reshape(2,1) @ s2\_.reshape(1,2)).flatten()   
 return ret

根据两个邻域电子联合状态空间（大小为4），由定义(价值)函数的神经网络模型返回当前电子动作（动作空间大小为2）的价值估计值（即动作的联合概率），选择概率最大的动作作为当前电子要执行的动作。定义(价值)函数的神经网络模型如下，

def qfunc(self,s,theta,layers=[(4,20),(20,2)],afn=torch.tanh):  
 l1n = layers[0]   
 l1s = np.prod(l1n)   
 theta\_1 = theta[0:l1s].reshape(l1n)   
 l2n = layers[1]  
 l2s = np.prod(l2n)  
 theta\_2 = theta[l1s:l2s+l1s].reshape(l2n)  
 bias = torch.ones((1,theta\_1.shape[1]))  
 l1 = s @ theta\_1 + bias   
 l1 = torch.nn.functional.elu(l1)  
 l2 = afn(l1 @ theta\_2)   
 return l2.flatten()

代码的作者 Zai, A.[12] 使用Numpy结合PyTorch库完成神经网络的构建，这对理解完全用PyTorch构建的神经网络是有帮助的。

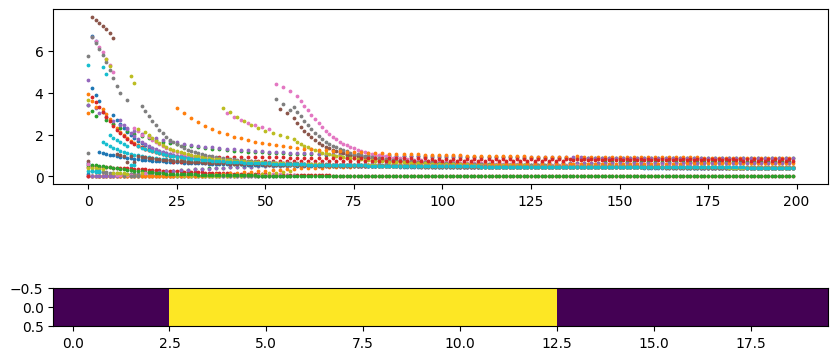
根据函数返回的价值估计值选择当前电子执行的动作，执行该动作后当前电子的状态的表述与所执行的动作一致，为自旋向上或向下。比较当前电子的状态和邻域两个电子的状态，由动作收益函数get\_reward\_1d返回收益值。用该收益值更新，并用欧几里德距离度量计算更新前后值间的距离作为损失，从而完成前向传播计算。由PyTorch库完成反向传播梯度计算，并指定学习率lr更新Q函数神经网络的参数值。训练Q函数神经网络的代码如下，

def train\_1d(self):  
 self.losses = [[] for i in range(self.size[0])]   
   
 for i in range(self.epochs):  
 for j in range(self.size[0]):   
 l = j - 1 if j - 1 >= 0 else self.size[0]-1   
 r = j + 1 if j + 1 < self.size[0] else 0   
 state\_ = self.grid[[l,r]]   
 state = self.joint\_state(state\_)   
 qvals = self.qfunc(state.float().detach(),self.params[j],layers=[(4,self.hid\_layer),(self.hid\_layer,2)])  
 qmax = torch.argmax(qvals,dim=0).detach().item()   
 action = int(qmax)  
 self.grid\_[j] = action   
 reward = self.get\_reward\_1d(state\_.detach(),action)  
 with torch.no\_grad():   
 target = qvals.clone()  
 target[action] = reward  
 loss = torch.sum(torch.pow(qvals - target,2))  
 self.losses[j].append(loss.detach().numpy())  
 loss.backward()  
 with torch.no\_grad():   
 self.params[j] = self.params[j] - self.lr \* self.params[j].grad  
 self.params[j].requires\_grad = True  
   
 with torch.no\_grad():   
 self.grid.data =self.grid\_.data

完成价值函数神经网络的训练后，打印每次迭代的损失self.losses，和最终更新后的晶格构型grid（状态估计值）如下。

ising\_1d.train\_1d()  
usda\_rl.ising\_1d\_plot(ising\_1d.losses,ising\_1d.grid,size)

tensor([0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],  
 dtype=torch.uint8) tensor(10)



#### 3） 2维度伊辛模型的 MARL 模拟——MF-Q [13]

在1维度伊辛模型中，如果不采取领域学习的方式，而是计算当前电子（智能体）之外所有电子的联合动作（状态），那么其大小将为。虽然领域学习大幅度减小了联合状动作空间的大小，但是如果增加维度，例如对于2维度的伊辛模型，其邻域电子个数为8，则其联合动作空间将为；继续增加维度到3维空间，领域电子数将为26，联合动作空间大小则为，这使得计算又变得困难。

邻域方法在伊辛模型中起作用的原因是因为电子的自旋受其最近邻域磁场的影响最大。如果将邻域各电子的联合动作空间，考虑到各单个电子自旋的影响转化为邻域所有电子自旋和，这将避免联合动作空间大小的影响。例如在1维度的伊辛模型中，邻域两个电子的动作分别为[1,0]和[0,1]（独热编码向量），那么其和为[1,1]。进一步将和除以所有邻域电子数，归一化结果为[0.5,0.5]，则是邻域电子动作的概率分布。同上述过程，Yaodong Yang等人[Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, & Jun Wang. (2020). Mean Field Multi-Agent Reinforcement Learning.]提出了平均场（Mean Field） MARL，对应到Q学习算法为平均场学习（mean field Q-learning ，MF-Q），公式表述为  
Qj(s,a)=Nj1​∑k∈N(j)​Qj(s,aj,ak)，式中， N(j)是索引值为j的智能体其邻域智能体索引集，大小Nj=∣N(j)∣。

在2维伊辛模型模拟试验中，由 SoftMax 归一化指数函数策略，根据 价值函数的估计值计算动作分布概率，从而选择概率最大的动作，定义softmax\_policy方法代码如下，

def softmax\_policy(self,qvals,temp=0.9):   
 soft = torch.exp(qvals/temp) / torch.sum(torch.exp(qvals/temp))   
 action = torch.multinomial(soft,1)   
 return action

输入参数temp为温度参数。当温度趋近于无限大时， SoftMax 返回的联合动作概率分布趋向于均匀分布，概率差异最小化，最终导致电子自旋方向倾向于互相取反，磁性消失；当温度趋近于无限小时，概率差异最大化，导致电子自旋倾向于向上或向下，与邻域电子大多数的自旋方向相一致，从而保持磁性。

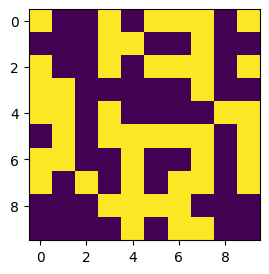
动作收益函数在2维中变为当前电子采取动作后的状态与邻域电子平均场向量之间的差异，例如如果当前电子执行动作后的状态为[0,1]，假设邻域电子状态平均场向量为[0,1]，则距离计算结果的距离为 0.986；如果平均场向量为[0.25,0.75]，那么计算结果的距离为0.848，具体实现的函数get\_reward\_2d代码如下，

def get\_reward\_2d(action,action\_mean):   
 r = (action\*(action\_mean-action/2)).sum()/action.sum()   
 return torch.tanh(5 \* r)

首先随机初始化2维伊辛模型的晶格构型，状态大小为，如下结果。

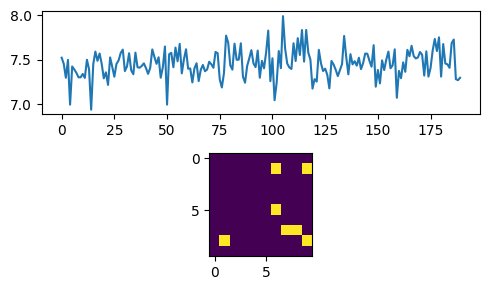
size=(10,10)  
hid\_layer = 10   
epochs = 200  
lr = 0.0001   
num\_iter = 3  
  
ising\_2d=usda\_rl.Ising\_MARL(size,hid\_layer,epochs,lr)   
print(ising\_2d.grid)  
  
fig,ax = plt.subplots(figsize=(3,3))  
ax.imshow(ising\_2d.grid)  
print(ising\_2d.grid.sum())

tensor([[1, 0, 0, 1, 0, 1, 1, 1, 0, 1],  
 [0, 0, 0, 1, 1, 0, 0, 1, 0, 0],  
 [1, 0, 0, 1, 0, 1, 1, 1, 0, 1],  
 [1, 1, 0, 0, 0, 0, 0, 1, 0, 0],  
 [1, 1, 0, 1, 0, 0, 0, 0, 1, 1],  
 [0, 1, 0, 1, 1, 1, 1, 1, 0, 1],  
 [1, 1, 0, 0, 1, 0, 0, 1, 0, 1],  
 [1, 0, 1, 0, 1, 0, 1, 1, 0, 1],  
 [0, 0, 0, 1, 1, 1, 1, 0, 0, 0],  
 [0, 0, 0, 0, 1, 0, 1, 1, 0, 0]], dtype=torch.uint8)  
tensor(48)



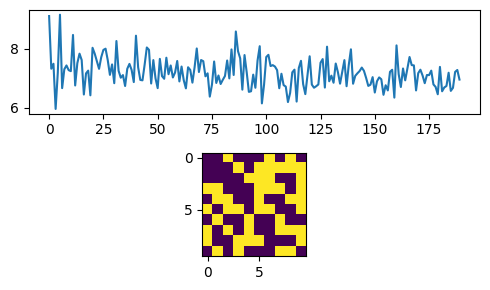
配置温度参数为temp=0.4，温度较低，电子的自旋倾向于朝向一个方向，保持磁性。

ising\_2d.train\_2d(num\_iter,temp=0.4)  
usda\_rl.ising\_2d\_plot(ising\_2d.losses,ising\_2d.grid)



配置温度参数为temp=10，温度较高，电子的自旋倾向于朝向相反的方向，从而失去磁性。

ising\_2d.train\_2d(num\_iter,temp=10)  
usda\_rl.ising\_2d\_plot(ising\_2d.losses,ising\_2d.grid)



## 3.9.2 MARL 环境集成和算法集成

### 3.9.2.1 环境和算法集成

#### 1）环境集成

在多个智能体的 RL 环境中，环境集成的库较为多样，如下表格罗列了主要或具有代表性的一些库环境。

| 序号 | 库名称 | 说明 | 图示 |
| --- | --- | --- | --- |
| 1 | [PettingZoo](https://pettingzoo.farama.org/)① | 为解决一般 MARL 问题，Python 化的简单环境集成库。包括有多种环境，实用程序工具，可以自定义环境。其中[AEC API](https://pettingzoo.farama.org/api/aec/)②支持基于顺序回合的环境（sequential turn based environments），[Parallel API](https://pettingzoo.farama.org/api/parallel/)③支持并行多线程计算。 | Atari：基于 [Arcade（街机）学习环境](https://github.com/Farama-Foundation/Arcade-Learning-Environment)④，其在现代 RL 发展中起到重要作用。    Butterfly：为[Farama](https://farama.org)⑤创造的具有挑战性的场景。所有环境都需要高度的协调，需要学习突发行为来实现最优策略。因此，目前学习这些环境非常具有挑战性。    Classic：经典流行的回合制游戏，大多数具有竞争性。    MPE（Multi Particle Environments）：多粒子环境，是一组面向通信环境（communication oriented environment），其中粒子智能体可以移动、通信、看到彼此、推断彼此，并于固定地标交互的环境。这些环境来源于[OpenAI’s MPE](https://github.com/openai/multiagent-particle-envs)⑥SISL：由斯坦福智能系统实验室（Stanford Intelligent Systems Laboratory，SISL）创建，包括3个协作式多智能体基准环境。 |
| 2 | [Jumanj](https://github.com/instadeepai/jumanji) ⑦ | 用[JAX（Autograd and XLA）](https://github.com/google/jax) ⑧编写的可扩展的 RL 环境。Jumanji 正在帮助引领 RL 领域硬件加速的研究和开发，其高速环境支持更快的迭代和大规模实验，同时降低了复杂性。 |  |
| 3 | [robotic-warehouse（RWARE）](https://github.com/semitable/robotic-warehouse)⑨ | 多机器人仓库根据现实世界的应用模拟了一个由机器人移动和交付所需货物的仓库。 |  |
| 4 | [VMAS（Vectorized Multi-Agent Simulator）](https://github.com/proroklab/VectorizedMultiAgentSimulator) ⑩ | 为矢量化多智能体模拟器，是一个为有效的 MARL 基准测试而编写设计的矢量化框架，由 PyTorch 编写的矢量化 2D 物理引擎和一组具有挑战性的多机器人环境组成。VMAS 模拟不同形状的智能体和地标，并支持旋转、弹性碰撞、关节（joints）和自定义重力。为了简化模拟，使用了完整的运动模型，并可以自定义诸如雷达等传感器，且支持智能体之间的通信，可以按批次执行模拟，无缝扩展到千万级别的并行环境中。 VMAS 具有与[OpenAI Gym](https://github.com/openai/gym)⑪、[RLlib](https://docs.ray.io/en/latest/rllib/index.html)⑫和 [torchrl](https://github.com/pytorch/rl)⑬等相兼容的接口，可以与更广泛的 RL 算法集成，其灵感来自于[OpenAI’s MPE](https://github.com/openai/multiagent-particle-envs)⑥。 |  |
| 5 | [MAgent2](https://github.com/Farama-Foundation/MAgent2)⑭ | 格网世界（gridworld）中大量像素智能体在战斗或者其它竞争性场景中相互作用交互。 |  |
| 6 | [AI-Economist](https://github.com/salesforce/ai-economist)⑮ | 为一个用于模拟含有智能体和政府的社会经济行为和动态，弹性、模块化和可组合的环境框架。 | [14] |
| 7 | [Nocturne](https://github.com/facebookresearch/nocturne) ⑯ | 为一个 2D，部分观察（partially observed）的驾驶模拟器。使用 C++ 构建以提供计算速度，并导出为 Python 库。Nocturne 设计的目的是处理来自[Waymo Open Dataset（Waymo 开源数据集）](https://github.com/waymo-research/waymo-open-dataset) ⑰的交通场景，并可以拓展到其它驾驶数据集。Nocturne可以训练自动驾驶汽车的控制器来解决 Waymo 数据集中的各种任务，并将其作为基准来评估设计的控制器。 |  |
| 8 | [Unity ML-Agents](https://github.com/Unity-Technologies/ml-agents) ⑱ | [Unity](https://unity.com)⑲机器学习智能体工具包（Unity Machine Learning Agents Toolkit，ML-Agents）是将游戏和模拟作为环境用于智能体训练的开源项目。ML-Agents 提供有基于 PyTorch 库实现的先进算法，使游戏开发人员和爱好者能够轻松训练2D、3D和 VR/AR 游戏的智能体。研究人员还可以使用其提供的简单易用的 Python API，用RL、模拟学习（imitation learning）、神经进化（neuroevolution），或者任何其它方法训练智能体。这些经过训练的智能体可用于游戏开发的过程，也可以用于任何相关的领域场景。 |  |
| 9 | [Neural MMO](https://github.com/openai/neural-mmo) ⑳ | Neural MMO 由 OpenAI 发布，用于 MARL 大型多智能体游戏环境。该平台在一个持久且开放的任务中支持大量、可变数量的智能体。许多智能体和物种的纳入可以带来更好的探索、形成不同生态位（divergent niche formation）和更高的综合能力。 |  |

#### 2）算法集成

下述列出的 RL 算法含有适合单个智能体同时适合多个智能体，及仅适合单个或多个智能体的算法，算法收集于[CleanRL](https://github.com/vwxyzjn/cleanrl)㉑、[Tianshou (天授)](https://github.com/thu-ml/tianshou)㉒、[RLlib](https://docs.ray.io/en/latest/rllib/rllib-algorithms.html)㉓和[Stable-Baselines3（SB3）](https://stable-baselines3.readthedocs.io/en/master/)㉔等几个 RL 算法集成库。

| 序号 | 算法 | 论文 | 备注 |
| --- | --- | --- | --- |
| 1 | A2C，Advantage Actor-Critic（优势动作评价） | Asynchronous Methods for Deep Reinforcement Learning[15] |  |
| 2 | A3C，Asynchronous Advantage Actor-Critic | Asynchronous Methods for Deep Reinforcement Learning[15] |  |
| 3 | AlphaZero，Single-Player Alpha Zero | Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm[16] |  |
| 4 | APPO，Asynchronous Proximal Policy Optimization | Proximal Policy Optimization Algorithms[17] |  |
| 5 | ARS，Augmented Random Search | Simple random search provides a competitive approach to reinforcement learning[18] |  |
| 6 | BanditLinUCB，Linear Upper Confidence Bound | Linear Upper Confidence Bound Algorithm for Contextual Bandit Problem with Piled Rewards[19] |  |
| 7 | BC，Behavior Cloning | Exponentially Weighted Imitation Learning for Batched Historical Data[20] |  |
| 8 | CQL，Conservative Q-Learning | Conservative Q-Learning for Offline Reinforcement Learning[21] |  |
| 9 | CRR，Critic Regularized Regression | Critic Regularized Regression[22] |  |
| 10 | C51，Categorical DQN | A Distributional Perspective on Reinforcement Learning[23] |  |
| 11 | DDPG，Deep Deterministic Policy Gradient（深度确定性策略梯度） | Continuous control with deep reinforcement learning[24] |  |
| 12 | APEX\_DDPG（Ape-X），Distributed Prioritized Experience Replay | Distributed Prioritized Experience Replay[25] |  |
| 13 | DreamerV3 | Mastering Diverse Domains through World Models[26] |  |
| 14 | Dreamer | Dream to Control: Learning Behaviors by Latent Imagination[27] |  |
| 15 | DQN，Deep Q Network（深度Q网络） | Human-level control through deep reinforcement learning[28] |  |
| 16 | APEX\_DQN（Ape-X），Distributed Prioritized Experience Replay | Distributed Prioritized Experience Replay[25] |  |
| 17 | DDQN，Double DQN（双网络深度Q学习） | Deep reinforcement learning with double q-learning[29] |  |
| 18 | ES，Evolution Strategies | Evolution Strategies as a Scalable Alternative to Reinforcement Learning[30] |  |
| 19 | GAE，Generalized Advantage Estimator（广义优势函数估计器） | High-dimensional continuous control using generalized advantage estimation[31] |  |
| 20 | HER，Hindsight Experience Repla | Hindsight Experience Replay[32] |  |
| 21 | IMPALA，Importance Weighted Actor-Learner Architecture | IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures[33] |  |
| 22 | LeelaChessZero，MultiAgent LeelaChessZero | Lc0[34] |  |
| 23 | MAML，Model-Agnostic Meta-Learning | Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks[35] |  |
| 24 | MARWIL，Monotonic Advantage Re-Weighted Imitation Learning | Exponentially Weighted Imitation Learning for Batched Historical Data[20.] |  |
| 25 | MB-MPO，Model-Based Meta-Policy-Optimization | Model-Based Reinforcement Learning via Meta-Policy Optimization[36] |  |
| 26 | PG，Policy Gradient（策略梯度） | Policy gradient methods for reinforcement learning with function approximation[Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Advances in Neural Information Processing Systems 12, [37] |  |
| 27 | PPO，Proximal Policy Optimization（近端策略优化） | Proximal policy optimization algorithms[17] |  |
| 28 | PPG，Phasic Policy Gradient | Phasic Policy Gradient[38] |  |
| 29 | PER，Prioritized Experience Replay（优先级经验重放） | Prioritized experience replay[39] |  |
| 30 | Qdagger | Reincarnating Reinforcement Learning: Reusing Prior Computation to Accelerate Progress[40] |  |
| 31 | RND，Random Network Distillation | Exploration by Random Network Distillation[41] |  |
| 32 | R2D2，Recurrent Replay Distributed DQN | Recurrent Experience Replay in Distributed Reinforcement Learning[42] |  |
| 33 | SAC，Soft Actor-Critic（软动作评价） | Soft actor-critic algorithms and applications[43] |  |
| 34 | SlateQ | SLATEQ: A Tractable Decomposition for Reinforcement Learning with Recommendation Sets[44] |  |
| 35 | TD3，Twin Delayed DDPG（双延迟深度确定性策略梯度） | Addressing function approximation error in actor-critic methods[45] |  |
| 36 | QMIX，QMIX Monotonic Value Factorisation | QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning[46] |  |
| 37 | MADDPG，Multi-Agent Deep Deterministic Policy Gradient | Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments[47] |  |
| 38 | Parameter Sharing | Cooperative Multi-Agent Control Using Deep Reinforcement Learning[48] |  |
| 39 | Fully Independent Learning | Multi-Agent and Hierarchical[49] |  |
| 40 | Shared Critic Methods | Implementing a Centralized Critic[50] |  |
| 41 | Curiosity (ICM: Intrinsic Curiosity Module) | Curiosity-driven Exploration by Self-supervised Prediction[51] |  |

### 3.9.2.2 水世界（waterworld\_v4）[52]

#### 1）水世界的环境

[SISL 环境](https://pettingzoo.farama.org/environments/sisl/)㉕由斯坦福智能系统实验室（Stanford Intelligent Systems Laboratory，SISL）创建，包括3个协作式多智能体基准环境，为Multi-Agent Walker、Pursuit Evastion和Waterworld，其作为“基于深度强化学习的协同多智能体控制（Cooperative multi-agent control using deep reinforcement learning）”的一部分发布，代码最初发布于[MADRL](https://github.com/sisl/MADRL)㉖。

水世界（Waterworld）模拟智能体（archea）在它们的环境中移动并试图存活。这些被称为追捕者（pursuers）的智能体试图在吞噬食物（food）时避免漂浮在水中的毒物（poison），其中追捕者为智能体（为带有放射状传感器的紫色圆表示），而食物（为绿色圆表示）和毒物（为最小的红色圆点表示）属于环境。环境的输入参数可以配置追捕者是否协作共同吞噬食物，从而建立即协作又竞争的关系。同时，收益可以全局的分配给所有的追捕者，也可以局部的分配给特定的追捕者。环境是一个二维的连续空间，每个追铺着都有一个由位于区间[0,1]的 和表示的位置，但不能够越过环境中间最大的一个灰绿色圆形障碍。追捕者通过选择一个推力向量来增加它们当前移动的速度，其外部的传感器可以读取临近对象的速度和方向，可用于环境导航。

* 观测空间（Observation Space）

依赖于环境的输入参数，每个智能体的观测（状态）空间大小为一个长度大于4的向量。观测空间的总大小为每个智能体的传感器与其特征数的乘积加上两个表明是否与食物和毒物分别碰撞的元素。如果参数speed\_features为True，则一个传感器的特征数为8，否则为5，那么观测空间的大小为，观测向量元素的取值范围为[-1,1]。例如，如果有5个追捕者，5个食物和10个毒物，每个追捕者都有30个范围有限的传感器（黑色放射线）用于检测邻近的食物和毒物，从而产生 242 个元素向量，为观测空间关于环境的计算值，代表了每个智能体上传感器感知到的距离和速度。当传感器在其范围内没有感知到任何对象时，速度报告为0，距离报告为1。

* 动作空间（Action Space）

智能体有一个连续的动作空间，对应于水平和垂直推力表示为一个大小为2的向量。值的范围取决于参数pursuer\_max\_accel配置的大小。动作值必须在[-pursuer\_max\_accel, pursuer\_max\_accel]之间。如果动作值大小超过了这个区间，则按比例缩小到[pursuer\_max\_accel]。

* 收益（Rewards）

当多个智能体（依赖于参数n\_coop的配置，协作共同吞噬食物的智能体数量）捕获到食物时，每个智能体都会获得一个food\_reward收益（此时，食物不会被破坏）；如果探触到食物则收到encounter\_reward收益；如果探触到毒物则收到poison\_reward收益；并对每个动作都有一个thrust\_penalty x ||action||收益，其中||action||为动作速度的欧几里得范数（euclidean norm）。收益分配有全局收益（global rewards）和局部（本地）收益（local rewards），根据参数local\_ratio配置的比例进行全局和局部收益的分配。如果为全局收益（比例为1 - local\_ratio）则将收益应用于每个智能体；如果为局部收益（比例为local\_ratio）则将分配到的收益仅应用于其行为产生收益的代理。

* 参数（Arguments）

waterworld\_v4.env(n\_pursuers=5, n\_evaders=5, n\_poisons=10, n\_coop=2, n\_sensors=20,sensor\_range=0.2,radius=0.015, obstacle\_radius=0.2, n\_obstacles=1,obstacle\_coord=[(0.5, 0.5)], pursuer\_max\_accel=0.01, evader\_speed=0.01,poison\_speed=0.01, poison\_reward=-1.0, food\_reward=10.0, encounter\_reward=0.01,thrust\_penalty=-0.5, local\_ratio=1.0, speed\_features=True, max\_cycles=500)

n\_pursuers：追捕者（智能体）的数量；

n\_evaders：食物的数量；

n\_poisons：毒物的数量；

n\_coop：必须同时探触到食物才能吞噬食物的追捕者数量

n\_sensors：所有追捕者传感器的数量；

sensor\_range：传感器树突长度；

radius：追捕者表示的圆形基本半径大小，食物为其2倍，毒物为其3/4；

obstacle\_radius：障碍物的半径大小；

obstacle\_coord：障碍物的位置，如果配置为None，则为随机位置；

pursuer\_max\_accel：追捕者最大加速度（最大动作大小）；

pursuer\_speed：追捕者速度；

evader\_speed：食物速度；

poison\_speed：毒物速度；

poison\_reward：追捕者吞噬了毒物的收益；

food\_reward：追捕者吞噬了食物的收益；

encounter\_reward：追捕者碰撞到食物的收益；

thrust\_penalty：用于惩罚较大动作负面收益的比例因子；

local\_ratio：所有智能体全局和局部收益的分配比例；

speed\_features：是否切换到追捕者传感器探测到其它对象（毒物、食物和其它追捕者）的速度；

max\_cycles：完成最大迭代后，所有智能体返回done。

从PettingZoo库读入waterworld\_v4水世界环境。并查看动作空间、观测（状态）空间、智能体等相关信息。其环境应用程序接口（Application Programming Interface，API）标准同由[OpenAI](https://openai.com)㉗建立的[gymnasium](https://github.com/Farama-Foundation/Gymnasium)㉘库的环境 API 标准，用于 RL 算法和环境之间的通信。

%load\_ext autoreload   
%autoreload 2   
import usda.rl as usda\_rl  
  
from pettingzoo.sisl import waterworld\_v4  
from IPython.display import HTML  
  
import glob  
import os  
import time  
  
import supersuit as ss  
from stable\_baselines3 import PPO  
from stable\_baselines3.ppo import MlpPolicy  
import pygame  
  
import matplotlib  
matplotlib.rcParams['animation.embed\_limit'] = 2\*\*128

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

从打印信息结果来看，动作空间大小为2，值域为[-1.0,1.0]的连续空间；观测空间大小为 242，值域为的连续空间；智能体（追捕者）数量为2。在metadata元数据中可知渲染模式render\_modes有human和rgb\_array两种方式，同时提供了环境名称，是否可以并行计算和渲染时每秒帧数等信息。

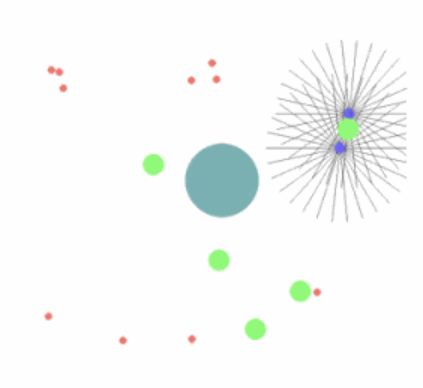
env=waterworld\_v4.env(n\_pursuers=2,render\_mode='rgb\_array')  
env.reset()  
print(f'action spaces:\n{env.action\_spaces}\nagents:\n{env.agents}\nenv.observation spaces:\n{env.observation\_spaces}\nmax num agents:\n{env.max\_num\_agents}\nmetadata:\n{env.metadata}')

action spaces:  
{'pursuer\_0': Box(-1.0, 1.0, (2,), float32), 'pursuer\_1': Box(-1.0, 1.0, (2,), float32)}  
agents:  
['pursuer\_0', 'pursuer\_1']  
env.observation spaces:  
{'pursuer\_0': Box(-1.4142135, 1.4142135, (242,), float32), 'pursuer\_1': Box(-1.4142135, 1.4142135, (242,), float32)}  
max num agents:  
2  
metadata:  
{'render\_modes': ['human', 'rgb\_array'], 'name': 'waterworld\_v4', 'is\_parallelizable': True, 'render\_fps': 15}

通过循环迭代所有智能体，并从各智能体的动作空间中随机采样获得一个随机动作的集合，并打印动画如下。

frames = []  
  
env.reset()  
for agent in env.agent\_iter(): # Yields the current agent. Needs to be used in a loop where you step() each iteration.  
 frames.append(env.render())  
 observation, reward, termination, truncation, info = env.last() # Returns observation, cumulative reward, terminated, truncated, info for the current agent  
 if termination or truncation:  
 action = None  
 else:  
 action = env.action\_space(agent).sample() # this is where you would insert your policy  
 env.step(action)  
env.close()

anim=usda\_rl.plot\_animation(frames,interval=100)  
anim.save(filename="../imgs/3\_9\_c/waterworld\_v4.gif")  
HTML(anim.to\_jshtml())



查看观测空间向量。

print(f'{observation.shape}\n{observation}')

(242,)  
[1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 0.9893761  
 0.90091896 0.8615445 0.8615445 0.62730205 0.387694 0.2896998  
 0.23960806 0.21219195 0.19817767 0.193847 0.19817767 0.21219195  
 0.23960806 0.2896998 0.387694 0.62730205 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 1. 1. 1. 1. 1. 1.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0.  
 0. 0. ]

查看收益和动作空间向量。

action = env.action\_space(agent).sample()   
print(f'{reward}\n{action}')

-0.452107404985053  
[-0.7013988 0.15334858]

#### 2）MARL 算法训练环境智能体

[PettingZoo环境库说明文件](https://pettingzoo.farama.org/)㉙提供了应用多种 MARL 算法集成库于其环境智能体训练的途径，包括CleanRL、Tianshou、Ray’s RLlib、LangChain和Stable-Baselines3 (SB3)等。下面迁移代码[53]仅以应用SB3的PPO算法训练水世界环境智能体为例。试验中应用了并行环境计算（parallel\_env），并用[SuperSuit](https://pypi.org/project/SuperSuit/)㉚库创建矢量化环境。定义train\_butterfly\_supersuit函数完成训练。

def train\_butterfly\_supersuit(env\_fn, steps: int = 10\_000, seed: int | None = 0, save\_root=None,num\_cpus=15,\*\*env\_kwargs):  
 # Train a single model to play as each agent in a cooperative Parallel environment  
 env = env\_fn.parallel\_env(\*\*env\_kwargs)  
 env.reset(seed=seed)  
 print(f"Starting training on {str(env.metadata['name'])}.")  
 env = ss.pettingzoo\_env\_to\_vec\_env\_v1(env)  
 env = ss.concat\_vec\_envs\_v1(env, 8, num\_cpus=num\_cpus, base\_class="stable\_baselines3")  
 # Note: Waterworld's observation space is discrete (242,) so we use an MLP policy rather than CNN  
 model = PPO(  
 MlpPolicy,  
 env,  
 verbose=3,  
 learning\_rate=1e-3,  
 batch\_size=256,  
 )  
  
 model.learn(total\_timesteps=steps)  
 if save\_root is not None:  
 model.save(os.path.join(save\_root,f"{env.unwrapped.metadata.get('name')}\_{time.strftime('%Y%m%d-%H%M%S')}"))  
 print("Model has been saved.")  
 print(f"Finished training on {str(env.unwrapped.metadata['name'])}.")  
 env.close()

训练过程中打印训练信息，完成后将训练好的模型保存至本地磁盘。

env\_fn=waterworld\_v4  
env\_kwargs=dict()  
save\_root='../models/'  
  
train\_butterfly\_supersuit(env\_fn, steps=10E4, seed=0, save\_root=save\_root,\*\*env\_kwargs)

Starting training on waterworld\_v4.  
Using cuda device  
------------------------------  
| time/ | |  
| fps | 1395 |  
| iterations | 1 |  
| time\_elapsed | 23 |  
| total\_timesteps | 32768 |  
------------------------------  
------------------------------------------  
| time/ | |  
| fps | 1193 |  
| iterations | 2 |  
| time\_elapsed | 54 |  
| total\_timesteps | 65536 |  
| train/ | |  
| approx\_kl | 0.0048262174 |  
| clip\_fraction | 0.0374 |  
| clip\_range | 0.2 |  
| entropy\_loss | -2.79 |  
| explained\_variance | -0.00408 |  
| learning\_rate | 0.001 |  
| loss | 8.14 |  
| n\_updates | 10 |  
| policy\_gradient\_loss | -0.00035 |  
| std | 0.973 |  
| value\_loss | 12.1 |  
------------------------------------------  
------------------------------------------  
| time/ | |  
| fps | 1127 |  
| iterations | 3 |  
| time\_elapsed | 87 |  
| total\_timesteps | 98304 |  
| train/ | |  
| approx\_kl | 0.0051830206 |  
| clip\_fraction | 0.0414 |  
| clip\_range | 0.2 |  
| entropy\_loss | -2.75 |  
| explained\_variance | 0.355 |  
| learning\_rate | 0.001 |  
| loss | 6.55 |  
| n\_updates | 20 |  
| policy\_gradient\_loss | -0.000364 |  
| std | 0.954 |  
| value\_loss | 13.5 |  
------------------------------------------  
-----------------------------------------  
| time/ | |  
| fps | 1080 |  
| iterations | 4 |  
| time\_elapsed | 121 |  
| total\_timesteps | 131072 |  
| train/ | |  
| approx\_kl | 0.006278244 |  
| clip\_fraction | 0.0538 |  
| clip\_range | 0.2 |  
| entropy\_loss | -2.71 |  
| explained\_variance | 0.351 |  
| learning\_rate | 0.001 |  
| loss | 8.7 |  
| n\_updates | 30 |  
| policy\_gradient\_loss | -0.000652 |  
| std | 0.931 |  
| value\_loss | 16.5 |  
-----------------------------------------  
Model has been saved.  
Finished training on waterworld\_v4.

定义eval函数，用已训练的模型进行模拟，模拟过程中对应智能体收集每一时刻的收益，计算各自收益累积值如下。

def eval(env\_fn, latest\_policy\_path, num\_games: int = 100, render\_mode: str | None = None, \*\*env\_kwargs):  
 # Evaluate a trained agent vs a random agent  
 env = env\_fn.env(render\_mode=render\_mode, \*\*env\_kwargs)  
 print(f"\nStarting evaluation on {str(env.metadata['name'])} (num\_games={num\_games}, render\_mode={render\_mode})")  
   
 model = PPO.load(latest\_policy\_path)  
 rewards = {agent: 0 for agent in env.possible\_agents}  
 # Note: We train using the Parallel API but evaluate using the AEC API  
 # SB3 models are designed for single-agent settings, we get around this by using he same model for every agent  
 frames=[]  
 for i in range(num\_games):  
 env.reset(seed=i)  
   
 for agent in env.agent\_iter():  
 if render\_mode=='rgb\_array':  
 frames.append(env.render())  
 obs, reward, termination, truncation, info = env.last()  
 for agent in env.agents:  
 rewards[agent] += env.rewards[agent]  
 if termination or truncation:  
 break  
 else:  
 act = model.predict(obs, deterministic=True)[0]  
  
 env.step(act)  
 env.close()  
 avg\_reward = sum(rewards.values()) / len(rewards.values())  
 print("Rewards: ", rewards)  
 print(f"Avg reward: {avg\_reward}")  
   
 if render\_mode=='rgb\_array':  
 return frames

默认的各收益值为poison\_reward=-1.0, food\_reward=10.0, encounter\_reward=0.01,thrust\_penalty=-0.5，根据已训练模型计算的各智能体累积收益值应该为一个大于0的数值。

# Evaluate 10 games (average reward should be positive but can vary significantly)  
latest\_policy\_path=r'../models/waterworld\_v4\_20231004-205936.zip'  
eval(env\_fn, latest\_policy\_path, num\_games=10, render\_mode=None, \*\*env\_kwargs)

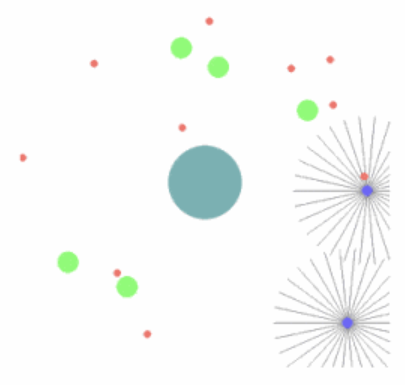
Starting evaluation on waterworld\_v4 (num\_games=10, render\_mode=None)  
Rewards: {'pursuer\_0': 706.6802023271017, 'pursuer\_1': 444.42363687866117}  
Avg reward: 575.5519196028814

将模拟的过程保存为动画，可以观察到追捕者和食物、毒物及障碍物之间的互动行为。因为默认配置n\_coop=2，因此仅当两个追捕者都探触到食物时，食物才被吞噬（消失）。

# Watch 2 games  
latest\_policy\_path=r'../models/waterworld\_v4\_20231004-205936.zip'  
env\_fn=waterworld\_v4  
env\_kwargs=dict()  
frames=eval(env\_fn, latest\_policy\_path, num\_games=2, render\_mode="rgb\_array", \*\*env\_kwargs)

Starting evaluation on waterworld\_v4 (num\_games=2, render\_mode=rgb\_array)  
Rewards: {'pursuer\_0': 171.0708689165037, 'pursuer\_1': 90.4392274241714}  
Avg reward: 130.75504817033755

anim=usda\_rl.plot\_animation(frames,interval=100)  
anim.save(filename="../imgs/3\_9\_c/waterworld\_v4.gif")  
HTML(anim.to\_jshtml())



## 3.9.3 自定义 MARL 环境——MPE\_realworld

[PettingZoo](https://pettingzoo.farama.org/tutorials/custom_environment/1-project-structure/)㉛库提供了自定义 MARL 环境的 API，并且为一个并行环境（parallel environment），这意味着每个智能体可以同时动作。自定义环境的文件结构如下（可以同时从 [PettingZoo 库的 GitHub 代码仓库](https://github.com/Farama-Foundation/PettingZoo)㉜下载源码查看其已有环境的文件结构），

Custom-Environment  
├── custom-environment  
 └── env  
 └── custom\_environment.py  
 └── custom\_environment\_v0.py  
├── README.md  
└── requirements.txt

* /custom-environment/env ，存储环境的位置，可含有任何辅助函数模块；
* /custom-environment/custom\_environment\_v0.py，导入所定义环境的模块，可以通过文件名，例如v0等显示的表明环境版本；
* /README.md，描述环境的 MD 文件（可选）；
* /requirements.txt，依赖库列表文件，包含PettingZoo库。

[Multi-Agent Particle Environment，MPE](https://github.com/openai/multiagent-particle-envs)⑥ 由 OpenAI 开发，目前嵌入到 PettingZoo库的 [MPE 环境包](https://pettingzoo.farama.org/environments/mpe/)㉝下，为一个简单的多智能体粒子世界，具有连续的观测空间和离散的动作空间，及一些基本的模拟物理（simulated physics）。该环境的一些基本概念有，

* Landmarks（地标）：为环境中不可被控制的静态圆形对象，根据设计的环境不同，地标的含义也会不同，例如可以为固定的目的地，障碍物等；
* Visibility（可见性）：当一个智能体对另一个智能体可见时，另一个智能体则包含该智能体的相对位置、速度等信息。如果该智能体暂时的隐藏起来，则另一个智能体包含该智能体的位置、速度等信息配置为0；
* Communication（通信）：环境中，某些智能体可以将消息作为其动作的一部分广播（broadcast），该消息将被传输到允许查看该消息的每个智能体；
* Color（颜色）：由于所有的智能体都为圆形，因此人们需要通过其颜色来识别观察智能体。智能体不能观测到颜色；
* Distances（距离）：智能体之间和智能体与地标之间的距离通常用于智能体的收益。

MPE 的终止结束条件是达到max\_cycles参数配置的循环次数，默认次数为 25。

观测空间（Observation Space）：

观测向量的组成通常包含有智能体的位置和速度，其它智能体的相对位置和速度，地标的相对位置，地标和智能体的类型，从其它智能体获得的通信信息等。如果一个智能体对另一个智能体不可见，则另一个智能体的观测空间中则不含有该智能体，因此不同的智能体可能有不同的观测空间大小。

动作空间（Action Space）：

* 离散动作空间（Discrete action space）：包括移动和通信两部分动作，对于移动为上下左右4个基本方向和静止不动，计5个值。对于通信可以在2到10个环境相关的通信选项之间进行选择，这些选项将消息广播给所有可以听到它的智能体；
* 连续动作空间（Continuous action space）：包括移动和通信两部分动作，对于移动可以在4个基本方向上输入一个0.0到1.0之间的速度，其中相对速度（例如左和右）则被加在一起。可以通信的智能体可以在其访问的环境中的每个通信通道上输出连续值。

### 3.9.3.1 simple\_realworld

原 MPE 中的 Simple环境只有一个智能体，一个代表目的地的地标。该智能体计算其到地标的距离（欧几里得距离），并取反作为收益。该环境主要用于 MPE 库的代码调试。基于 MPE 环境架构，如果将其用于现实世界中情景模拟，需要加载一个类似土地覆盖（landcover，LC）的地图（可以包含反应不同信息的多个地图），用于描述真实世界的场景；同时，场地的大小不再是 -1 到 +1 的一个正方形地块，应该可以反应实际地图的大小范围。除了智能体之间及和地标之间的相互作用，应该也包括智能体和地图之间的交互。

对于地图的加载使用一个至少2维的数组（定义变量名为plat）用于表示每一个栅格单元的数值，该数值根据分析内容进行配置。如果大于2维，则表示每一个栅格单元包含多个垂直叠加平行的值（3个维度）或者具有继承关系的嵌套值（大于3个维度），表示该位置的不同的多个信息。为了观察不同栅格值，对于分类数据传入一个颜色字典，可以根据分类值的颜色观察值的分布（定义变量名为plat\_colors）。如果每一个分类对智能体有不同的影响，则可以传入一个智能体受地图影响的收益值（定义变量名为plat\_rewards）。在定义的simple\_world（基于simple）环境中，实际从地图获取的收益值为传入地图收益值与其最大值的比值，将其归一化。

地图的大小以栅格单元大小的方式表述，对于一个表示栅格的2维数组，每个数值即为一个栅格单元的值，如果一个地块的高空分辨率低，即一个栅格单元很大，则数组形状（shape）会很小；反之，高空分辨率高，栅格单元变小，数组形状则会变大。MPE 环境智能体和地标的位置信息位于 -1 到 +1 的区间内，为两个维度（x和y坐标值），在定义的SimpleEnv类下的draw绘制环境的方法下，给出了智能体和地标[-1,+1]区间位置到画布（地图）位置坐标的一个转化代码，因此可以在不改变原 MPE [-1,+1]阈值情况下，通过定义地图坐标到位置信息和位置信息到地图坐标的转化函数plat\_coordi2p\_pos和p\_pos2plat\_coordi实现不同坐标空间的变换。地图的坐标由2维数组对应的索引表示。

智能体和地图之间的交互通过智能体所处位置地图自身具有的值，或者传入对应类别的地图收益值plat\_rewards，又或者邻域栅格的值与其所具有的某种函数关系确定。智能体的初始化位置为随机位置，而地标是从分类值为1的区域内随机选择一个栅格单元所在的位置。定义的simple\_realworld环境，类似simple环境仅为测试实现上述内容关系的自定义环境代码所用。并将其迁移到USDA库，位于其下的mpe\_realworld包下。具体调用过程如下。

%load\_ext autoreload   
%autoreload 2   
import usda.mpe\_realworld as usda\_mpe  
from usda.mpe\_realworld.mpe import simple\_realworld  
from usda import datasets as usda\_datasets  
import usda.rl as usda\_rl   
  
import matplotlib.pyplot as plt  
import mapclassify  
import matplotlib  
from IPython.display import HTML  
import supersuit as ss  
  
from stable\_baselines3 import PPO  
from stable\_baselines3.ppo import MlpPolicy

用generate\_categorical\_2darray生成一个用于测试simple\_realworld环境的随机分类样本数据（地图），具体可查看*标记距离*部分的解释。

size=200  
X\_,\_=usda\_datasets.generate\_categorical\_2darray(size=size,sigma=7,seed=77)  
X=X\_[0].reshape(size,size)\*size  
X\_BoxPlot=mapclassify.BoxPlot(X)  
y=X\_BoxPlot.yb.reshape(size,size)  
y=y[:100,:]

配置打印地图的分类颜色。

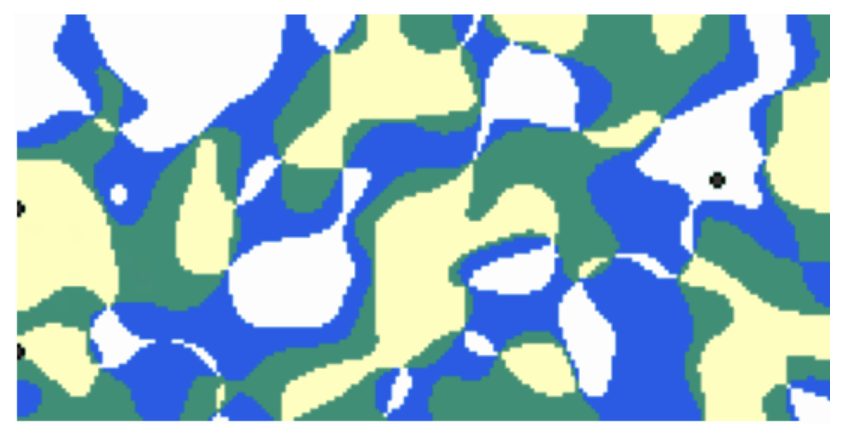
levels = list(range(1,10))  
clrs = ['#FFFFFF','#005ce6', '#3f8f76', '#ffffbe', '#3a5b0d', '#aaff00', '#e1e1e1','#F44336','#eeeee4']   
clrs\_dict={1:'#FFFFFF',2:'#005ce6',3:'#3f8f76',4:'#ffffbe'}  
cmap, norm = matplotlib.colors.from\_levels\_and\_colors(levels, clrs,extend='max')

与PettingZoo库一样，可以直接调入环境，并通过env方法建立环境。在传入的参数中可以看到增加了plat、plat\_colors和plat\_rewards等3个参数，用于表示真实世界中的地图等信息。为了验证所建立的环境运行无误，通过迭代一个循环（默认为 25），渲染打印过程，观察是否正确的加载了地图，智能体和地标位置是否正确，智能体是否发生的移动等。

env=simple\_realworld.env(render\_mode="rgb\_array",  
 plat=y,  
 plat\_colors=clrs\_dict,  
 plat\_rewards={1:-1,2:-3,3:1,4:2},  
 )  
  
env.reset()   
frames=[]  
for agent in env.agent\_iter():  
 frames.append(env.render())  
 observation, reward, termination, truncation, info = env.last()   
 if termination or truncation:  
 action = None  
 else:  
 # this is where you would insert your policy  
 action = env.action\_space(agent).sample()  
 env.step(action)  
env.close()

打印 MARL 环境测试结果。

anim=usda\_rl.plot\_animation(frames,interval=100,figsize=(7,5))  
anim.save(filename="../imgs/3\_9\_c/simple\_realworld\_rnd.gif", writer="pillow")  
HTML(anim.to\_jshtml())



训练simple\_realworld环境智能体的策略网络模型仍然使用了Stable-Baselines3 (SB3)库的 PPO 算法，具体代码同水世界的具体训练过程。这里有意识的将地标所在的地图分类（值为1）的收益配置为 -1，从而使得智能体很难最终接近地标位置。

使用了并行环境parallel\_env方法，用[SuperSuit](https://pypi.org/project/SuperSuit/)㉚库（包装器）包装（wrap）MARL 环境进行环境转化预处理，再传入 SB3 模型。

env=simple\_realworld.parallel\_env(  
 render\_mode="rgb\_array",  
 plat=y,  
 plat\_colors=clrs\_dict,  
 plat\_rewards={1:-1,2:-3,3:1,4:2},  
 )  
  
env = ss.pettingzoo\_env\_to\_vec\_env\_v1(env)  
env = ss.concat\_vec\_envs\_v1(env, 8, num\_cpus=12, base\_class="stable\_baselines3")  
  
model = PPO(  
 MlpPolicy,  
 env,  
 verbose=3,  
 learning\_rate=1e-3,  
 batch\_size=256,  
)

Using cuda device

通过训练过程打印的相关参数值可以估计相关参数变化（例如损失 loss）的程度所花费的时间步（total\_timesteps）。

model.learn(total\_timesteps=3E5)  
  
simple\_world\_PPO\_path=r'../models/simple\_world\_PPO.zip'  
model.save(simple\_world\_PPO\_path)

------------------------------  
| time/ | |  
| fps | 637 |  
| iterations | 1 |  
| time\_elapsed | 51 |  
| total\_timesteps | 32768 |  
------------------------------  
-----------------------------------------  
| time/ | |  
| fps | 585 |  
| iterations | 2 |  
| time\_elapsed | 111 |  
| total\_timesteps | 65536 |  
| train/ | |  
| approx\_kl | 0.011204008 |  
| clip\_fraction | 0.153 |  
| clip\_range | 0.2 |  
| entropy\_loss | -1.6 |  
| explained\_variance | 0.00964 |  
| learning\_rate | 0.001 |  
| loss | 60.7 |  
| n\_updates | 10 |  
| policy\_gradient\_loss | -0.013 |  
| value\_loss | 183 |  
-----------------------------------------  
-----------------------------------------  
| time/ | |  
| fps | 566 |  
| iterations | 4 |  
| time\_elapsed | 231 |  
| total\_timesteps | 131072 |  
| train/ | |  
| approx\_kl | 0.014367534 |  
| clip\_fraction | 0.211 |  
| clip\_range | 0.2 |  
| entropy\_loss | -1.52 |  
| explained\_variance | 0.682 |  
| learning\_rate | 0.001 |  
| loss | 25 |  
| n\_updates | 30 |  
| policy\_gradient\_loss | -0.0246 |  
| value\_loss | 60 |  
-----------------------------------------  
-----------------------------------------  
| time/ | |  
| fps | 560 |  
| iterations | 6 |  
| time\_elapsed | 350 |  
| total\_timesteps | 196608 |  
| train/ | |  
| approx\_kl | 0.013886394 |  
| clip\_fraction | 0.201 |  
| clip\_range | 0.2 |  
| entropy\_loss | -1.38 |  
| explained\_variance | 0.798 |  
| learning\_rate | 0.001 |  
| loss | 8.99 |  
| n\_updates | 50 |  
| policy\_gradient\_loss | -0.0247 |  
| value\_loss | 17.8 |  
-----------------------------------------  
------------------------------------------  
| time/ | |  
| fps | 556 |  
| iterations | 10 |  
| time\_elapsed | 588 |  
| total\_timesteps | 327680 |  
| train/ | |  
| approx\_kl | 0.0074738376 |  
| clip\_fraction | 0.0812 |  
| clip\_range | 0.2 |  
| entropy\_loss | -1.19 |  
| explained\_variance | 0.704 |  
| learning\_rate | 0.001 |  
| loss | 4.19 |  
| n\_updates | 90 |  
| policy\_gradient\_loss | -0.00438 |  
| value\_loss | 8.73 |  
------------------------------------------

读入保存的已训练模型，用该模型（价值函数）预测多个智能体下一步的动作，总共进行了10次模拟，并记录过程，打印查看动画。

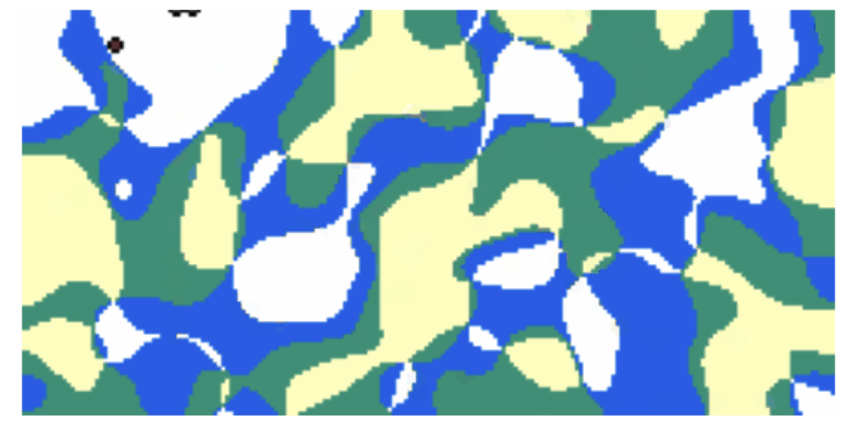
env=simple\_realworld.env(  
 render\_mode="rgb\_array",  
 plat=y,  
 plat\_colors=clrs\_dict,  
 plat\_rewards={1:-2,2:-5,3:0,4:1},  
 )  
  
simple\_world\_PPO\_path=r'../models/simple\_world\_PPO.zip'  
model = PPO.load(simple\_world\_PPO\_path)  
  
rewards = {agent: 0 for agent in env.possible\_agents}  
frames=[]  
for i in range(10):  
 env.reset(seed=i)  
   
 for agent in env.agent\_iter():  
 frames.append(env.render())  
 obs, reward, termination, truncation, info = env.last()  
 for agent in env.agents:  
 rewards[agent] += env.rewards[agent]  
 if termination or truncation:  
 break  
 else:  
 act = model.predict(obs, deterministic=True)[0]  
  
 env.step(act)  
   
env.close()  
avg\_reward = sum(rewards.values()) / len(rewards.values())  
print("Rewards: ", rewards)  
print(f"Avg reward: {avg\_reward}")

Rewards: {'agent\_0': -172.60241248297925, 'agent\_1': -156.38614996027982}  
Avg reward: -164.49428122162954

从动画中智能体的运动轨迹可以观察到，对应分类1（白色，-1）、2（蓝色，-3）、3（绿色，1）和4（黄色，2）的颜色和分类地图给予的收益，大多数情况下，智能体尽量避开了收益为负值的分类区域蓝色（2）和白色（1），而尽量在正值黄色（2）和绿色（3）下移动，并尽可能的向地标靠近。

anim=usda\_rl.plot\_animation(frames,interval=100,figsize=(7,5))  
anim.save(filename="../imgs/3\_9\_c/simple\_realworld\_PPO.gif")  
HTML(anim.to\_jshtml())

MovieWriter ffmpeg unavailable; using Pillow instead.



### 3.9.3.2 纳沃纳广场的人行为模拟

从[OpenStreetMap（OSM）](https://www.openstreetmap.org/#map=18/41.89895/12.47073) ㉞下载纳沃纳广场（piazza\_navona，(12.472986°, 41.898889°)） OSM 数据，采用*OSM数据与核密度估计*部分阐述的方法处理 OSM 数据，提取way区域（Polygon）数据和node兴趣点（Point）数据。区域数据提取建筑字段building的信息，点数据提取设施amenity字段信息，并分别转化为同一栅格单元大小的栅格数据，且合并波段为一个文件，用于 MPE\_realworld 的地图参数plat的输入。

获得了真实世界的地图信息，在用PettingZoo库建立navona\_v1环境时， 利用这些信息对（多个）智能体的影响建立对应的收益函数。而后调用navona\_v1环境，配置相关参数，利用SB3库提供的算法，完成网络模型训练。

#### 1）数据准备

在处理纳沃纳广场区域的 OSM 数据时，主要考虑的问题是如何将矢量数据转化为 MPE\_realworld 可以读取的数组形式，并可以叠加任意多层信息，而栅格数据满足试验需求。

* 从 OSM 到 GeoDataFrame 格式数据

%load\_ext autoreload   
%autoreload 2   
import usda.data\_process as usda\_dp  
import usda.geodata\_process\_opt as usda\_geoproces  
  
import geopandas as gpd  
from shapely.geometry import LineString, MultiPoint, Polygon  
import os  
import shutil  
from osgeo import gdal  
import rioxarray as rxr  
import numpy as np  
import matplotlib.pyplot as plt

使用定义的osmHandler类，完成 OSM 数据的处理，并读取为 GeoDataFrame 格式数据，包括区域数据和点数据。

piazza\_navona\_fn='../data/piazza\_navona.osm'  
  
osm\_handler=usda\_dp.osmHandler()   
osm\_handler.apply\_file(piazza\_navona\_fn,locations=True)  
  
epsg\_wgs84=4326  
osm\_columns=['type','geometry','id','version','visible','ts','uid','user','changeet','tagLen','tags']  
osm\_node\_gdf=gpd.GeoDataFrame(osm\_handler.osm\_node,columns=osm\_columns,crs=epsg\_wgs84)  
osm\_node\_gdf.head(3)

|  | **type** | **geometry** | **id** | **version** | **visible** | **ts** | **uid** | **user** | **changeet** | **tagLen** | **tags** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | node | POINT (12.47228 41.89919) | 25388168 | 6 | True | 2011-09-18 16:47:51+00:00 | 430115 | Emistrac | 9334851 | 0 | {} |
| **1** | node | POINT (12.47225 41.89980) | 25388169 | 5 | True | 2011-08-14 00:27:17+00:00 | 430115 | Emistrac | 9011017 | 0 | {} |
| **2** | node | POINT (12.47270 41.89981) | 25388170 | 6 | True | 2011-08-14 00:27:17+00:00 | 430115 | Emistrac | 9011017 | 0 | {} |

osm\_area\_gdf=gpd.GeoDataFrame(osm\_handler.osm\_area,columns=osm\_columns,crs=epsg\_wgs84)  
osm\_area\_gdf.head(3)

|  | **type** | **geometry** | **id** | **version** | **visible** | **ts** | **uid** | **user** | **changeet** | **tagLen** | **tags** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | area | MULTIPOLYGON (((12.47270 41.89981, 12.47270 41... | 8494276 | 64 | True | 2023-08-21 19:40:23+00:00 | 19233464 | secondaryhighway | 140198197 | 13 | {'name': 'Piazza Navona', 'name:es': 'Plaza Na... |
| **1** | area | MULTIPOLYGON (((12.47649 41.89856, 12.47649 41... | 47681262 | 48 | True | 2013-12-12 10:47:31+00:00 | 904963 | ty000 | 19410566 | 3 | {'building:part': 'yes', 'height': '28.3', 'ro... |
| **2** | area | MULTIPOLYGON (((12.47404 41.89839, 12.47404 41... | 51221130 | 27 | True | 2023-09-20 14:05:07+00:00 | 12500589 | tg4567 | 141515915 | 6 | {'building': 'yes', 'building:levels': '4', 'n... |

区域和点数据通常范围不同，因此用clip方法裁切到同一大小。

minx, miny, maxx, maxy = osm\_area\_gdf.geometry.total\_bounds  
envelope = gpd.GeoDataFrame([[Polygon([(minx, miny), (minx, maxy), (maxx, maxy), (maxx,miny), (minx, miny)])]], columns=['geometry'],crs=osm\_area\_gdf.crs)  
osm\_node\_clipped\_gdf=osm\_node\_gdf.clip(envelope)

从tags字段中提取building建筑信息。

osm\_area\_gdf['buildinginfo']=osm\_area\_gdf.tags.apply(lambda x:x['building'] if 'building' in x.keys() else 'None')

从经纬度坐标经投影转化到米制坐标。

Rome\_epsg=32633  
osm\_area\_gdf.to\_crs(Rome\_epsg,inplace=True)

同样的方式处理点数据。

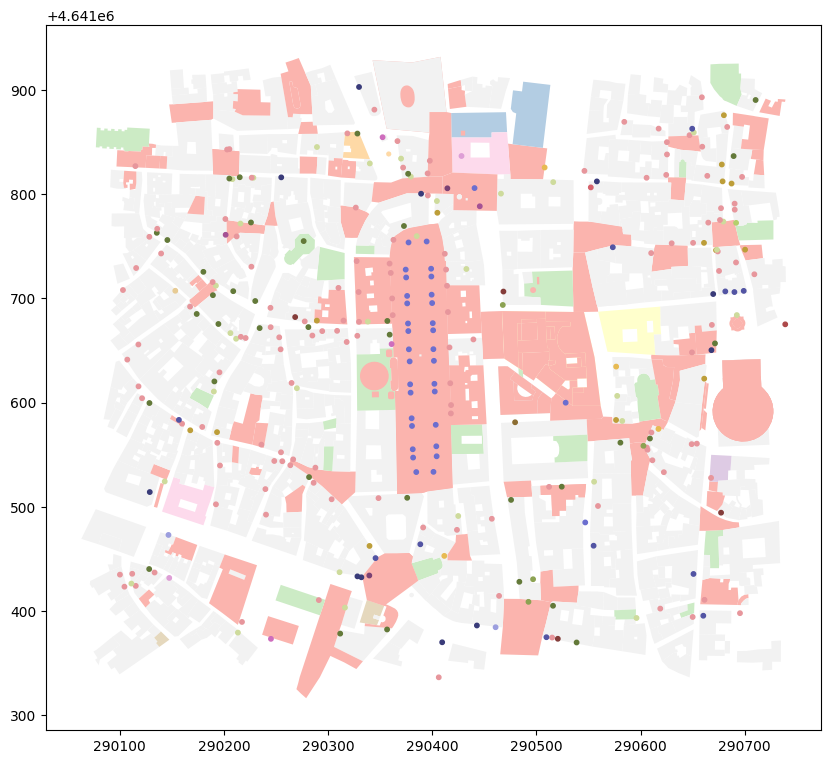
osm\_node\_clipped\_gdf['amenity']=osm\_node\_clipped\_gdf.tags.apply(lambda x:x['amenity'] if 'amenity' in x.keys() else 'None')

osm\_node\_clipped\_gdf=osm\_node\_clipped\_gdf[osm\_node\_clipped\_gdf.amenity!='None']

osm\_node\_clipped\_gdf.to\_crs(Rome\_epsg,inplace=True)

叠加矢量的区域和点数据，查看数据处理结果是否正确。

ax=osm\_area\_gdf.plot(column='buildinginfo',figsize=(10,10),cmap='Pastel1')  
osm\_node\_clipped\_gdf.plot(column='amenity',ax=ax,markersize=10,cmap='tab20b');



* 从 GeoDataFrame 到栅格数据

矢量区域的建筑信息和点数据的设施信息均为字符串，在转化为栅格数据前需要进行整数编码。在编码时，需要注意不使用数值 0，而是从1顺序编号，方便后续地图打印。

area\_encoding=dict(enumerate(osm\_area\_gdf.buildinginfo.unique()))  
area\_encoding={v:k+1 for k,v in area\_encoding.items()}  
area\_encoding

{'None': 1,  
 'yes': 2,  
 'church': 3,  
 'basilica': 4,  
 'apartments': 5,  
 'residential': 6,  
 'school': 7,  
 'retail': 8,  
 'public': 9,  
 'commercial': 10,  
 'bridge': 11,  
 'hotel': 12,  
 'temple': 13,  
 'kiosk': 14}

仅提取所需信息列数据，包括编码列和几何列。

osm\_area\_gdf['b\_id']=osm\_area\_gdf.buildinginfo.apply(lambda x:area\_encoding[x])  
osm\_area\_selection\_gdf=osm\_area\_gdf[['geometry','b\_id']]  
osm\_area\_selection\_gdf.head(2)

|  | **geometry** | **b\_id** |
| --- | --- | --- |
| **0** | MULTIPOLYGON (((290359.243 4641740.674, 290359... | 1 |
| **1** | MULTIPOLYGON (((290669.458 4641593.029, 290669... | 1 |

同样处理点数。

node\_encoding=dict(enumerate(osm\_node\_clipped\_gdf.amenity.unique()))  
node\_encoding={v:k+1 for k,v in node\_encoding.items()}  
node\_encoding

{'cafe': 1,  
 'pharmacy': 2,  
 'restaurant': 3,  
 'bar': 4,  
 'bureau\_de\_change': 5,  
 'bank': 6,  
 'charging\_station': 7,  
 'drinking\_water': 8,  
 'atm': 9,  
 'fast\_food': 10,  
 'taxi': 11,  
 'ice\_cream': 12,  
 'monastery': 13,  
 'bench': 14,  
 'fountain': 15,  
 'place\_of\_worship': 16,  
 'bicycle\_parking': 17,  
 'clock': 18,  
 'library': 19,  
 'toilets': 20,  
 'vending\_machine': 21,  
 'pub': 22,  
 'nightclub': 23,  
 'theatre': 24,  
 'post\_box': 25,  
 'telephone': 26,  
 'university': 27,  
 'post\_office': 28,  
 'college': 29}

osm\_node\_clipped\_gdf['amen\_id']=osm\_node\_clipped\_gdf.amenity.apply(lambda x:node\_encoding[x])  
osm\_node\_selection\_gdf=osm\_node\_clipped\_gdf[['geometry','amen\_id']]  
osm\_node\_selection\_gdf.head(2)

|  | **geometry** | **amen\_id** |
| --- | --- | --- |
| **7292** | POINT (290538.610 4641369.689) | 1 |
| **6717** | POINT (290520.572 4641373.243) | 2 |

定义的rasterize函数，为从磁盘空间读写 SHP 格式数据， 并写入到临时磁盘存储位置。因此首先将 GeoDataFrame 数据以 SHP 格式写入到本地磁盘；在用rasterize函数转化为栅格后，需要从临时存储空间中复制该文件到目标存储文件夹中，使用shutil.copy的方法。

piazza\_navona\_shp\_root=r'../data/piazza\_navona\_osm'  
  
osm\_area\_selection\_gdf.to\_file(os.path.join(piazza\_navona\_shp\_root,'piazza\_navona\_osm\_area.shp'))  
osm\_node\_selection\_gdf.to\_file(os.path.join(piazza\_navona\_shp\_root,'piazza\_navona\_osm\_node.shp'))

area\_path=usda\_geoproces.rasterize(os.path.join(piazza\_navona\_shp\_root,'piazza\_navona\_osm\_area.shp'),'b\_id',cellSize=1,NoData\_value=-9999,dtype='int32')  
shutil.copy(area\_path, piazza\_navona\_shp\_root)  
node\_path=usda\_geoproces.rasterize(os.path.join(piazza\_navona\_shp\_root,'piazza\_navona\_osm\_node.shp'),'amen\_id',cellSize=1,NoData\_value=-9999,dtype='int32')  
shutil.copy(node\_path, piazza\_navona\_shp\_root)  
print(f'area\_path={area\_path}\nnode\_path={node\_path}')

area\_path=C:\Users\richie\AppData\Local\Temp\tmpsaejolrx.tif  
node\_path=C:\Users\richie\AppData\Local\Temp\tmptuvxajgu.tif

将转化的区域和点两个栅格数据合并为多个波段的一个文件。

area\_node=gdal.BuildVRT("", [area\_path,node\_path], separate=True) # Build an in-memory VRT  
area\_node\_fn=os.path.join(piazza\_navona\_shp\_root,'navona\_area\_node.tif')  
gdal.Translate(area\_node\_fn, area\_node);

查看由 OSM 转化获得的最终栅格文件，确定数据处理正确。

navona\_area\_node=rxr.open\_rasterio(area\_node\_fn)  
navona\_area\_node

<xarray.DataArray (band: 2, y: 615, x: 676)>  
[831480 values with dtype=int32]  
Coordinates:  
band (band) int32 1 2  
x (x) float64 2.901e+05 2.901e+05 ... 2.907e+05 2.907e+05  
y (y) float64 4.642e+06 4.642e+06 ... 4.641e+06 4.641e+06  
spatial\_ref int32 0  
Attributes:  
\_FillValue: -9999  
scale\_factor: 1.0  
add\_offset: 0.0

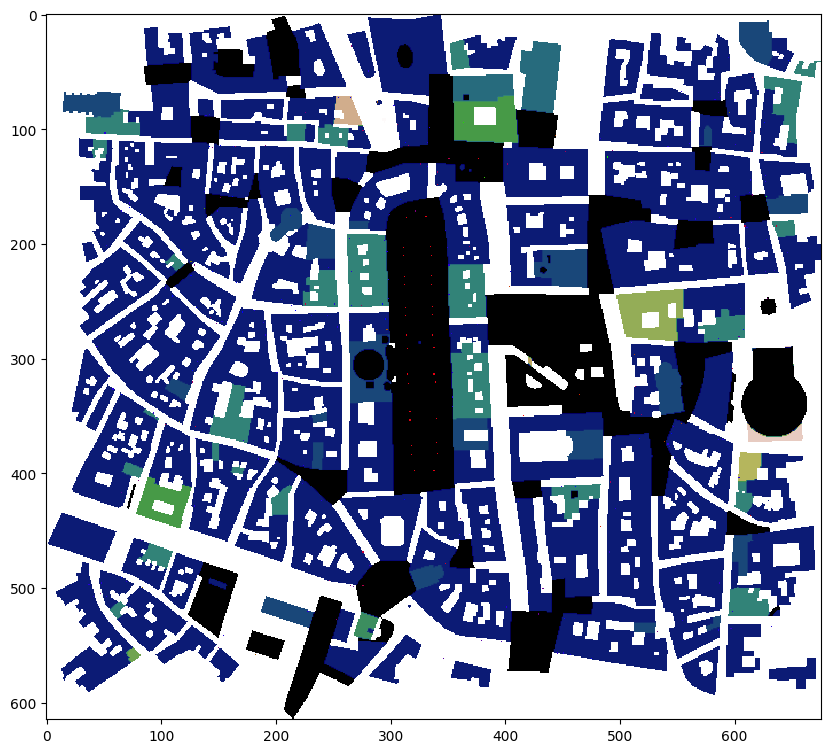
为一个三维数组，包含叠加的两个波段信息，建筑信息和设施信息。转化时的空值区域默认填充为数值 -9999。

plat=navona\_area\_node.data  
plat.shape

(2, 615, 676)

打印建筑和设施的栅格数据，确定两个波段数据对位。

fig, ax=plt.subplots(figsize=(10,10))  
masked\_navona\_area=np.ma.masked\_where(navona\_area\_node[0] == -9999, navona\_area\_node[0])  
ax.imshow(masked\_navona\_area,cmap='gist\_earth')  
masked\_navona\_node=np.ma.masked\_where(navona\_area\_node[1] == -9999, navona\_area\_node[1])  
ax.imshow(masked\_navona\_node,interpolation='none',cmap='brg')  
plt.show()



#### 2）基于 MARL 的人行为模拟

* 建立环境 navona\_v1

基于simple\_realworld环境，建立navona\_v1，主要是处理地图和智能体之间的收益关系，这里定义了三个收益函数，一个为plat\_area\_rewards方法，主要处理智能体和建筑环境的关系。根据提供的plat\_rewards，智能体当前时刻（位置）的收益对应用地类型的收益值，如果智能体位于广场、道路，或者公共建筑等开放空间则具有较高的收益，位于一般居住建筑内则为负值；第二个为plat\_node\_rewards方法，主要处理智能体和设施之间的关系。给定搜索范围参数nodes\_radius，计算智能体当前位置搜索范围内设施的数量，给定一个比例因子缩放值后作为收益；第三个为agents\_group\_rewards方法，主要处理智能体之间的团聚关系。给定距离参数group\_dis，如果其它智能体与当前智能体的距离位于该距离内则认为是聚集的，计算满足距离要求的智能体数量，并应用泊松分布（poisson）返回值作为收益值。应用泊松分布的目的是表达适当的积聚人数更容易维持，当人数很少或者较多时，集聚的人数都比较容易发生变化，倾向于到一个比较容易维持的人数。总的收益为上述3个收益值的和。

%load\_ext autoreload   
%autoreload 2   
import usda.mpe\_realworld as usda\_mpe  
from usda.mpe\_realworld.mpe import navona\_v1  
import usda.utils as usda\_utils  
import usda.rl as usda\_rl   
  
import matplotlib.pyplot as plt  
import mapclassify  
import matplotlib  
from IPython.display import HTML  
import supersuit as ss  
  
from stable\_baselines3 import PPO  
from stable\_baselines3.ppo import MlpPolicy  
import os  
import rioxarray as rxr  
import numpy as np

The autoreload extension is already loaded. To reload it, use:  
 %reload\_ext autoreload

读入建筑和设施的地图数据，其中将建筑数据的空值 -9999 配置为 15，是避免 navona\_v1 环境打印地图时颜色配置缺失。并建立建筑对应的颜色值。

piazza\_navona\_shp\_root=r'C:\Users\richie\omen\_richiebao\omen\_github\USDA\_special\_study\data\piazza\_navona\_osm'  
area\_node\_fn=os.path.join(piazza\_navona\_shp\_root,'navona\_area\_node.tif')  
navona\_area\_node=rxr.open\_rasterio(area\_node\_fn)  
plat=np.stack(navona\_area\_node.data,axis=2)  
plat[:,:,0][plat[:,:,0]==-9999]=15   
  
plat\_colors=usda\_utils.cmap2hex('Pastel1',16)  
print(plat\_colors)   
cmap, norm = matplotlib.colors.from\_levels\_and\_colors([i+1 for i in list(plat\_colors.keys())],list(plat\_colors.values()),extend='max')

{0: '#fbb4ae', 1: '#fbb4ae', 2: '#b3cde3', 3: '#b3cde3', 4: '#ccebc5', 5: '#decbe4', 6: '#decbe4', 7: '#fed9a6', 8: '#fed9a6', 9: '#ffffcc', 10: '#e5d8bd', 11: '#e5d8bd', 12: '#fddaec', 13: '#fddaec', 14: '#f2f2f2', 15: '#f2f2f2'}

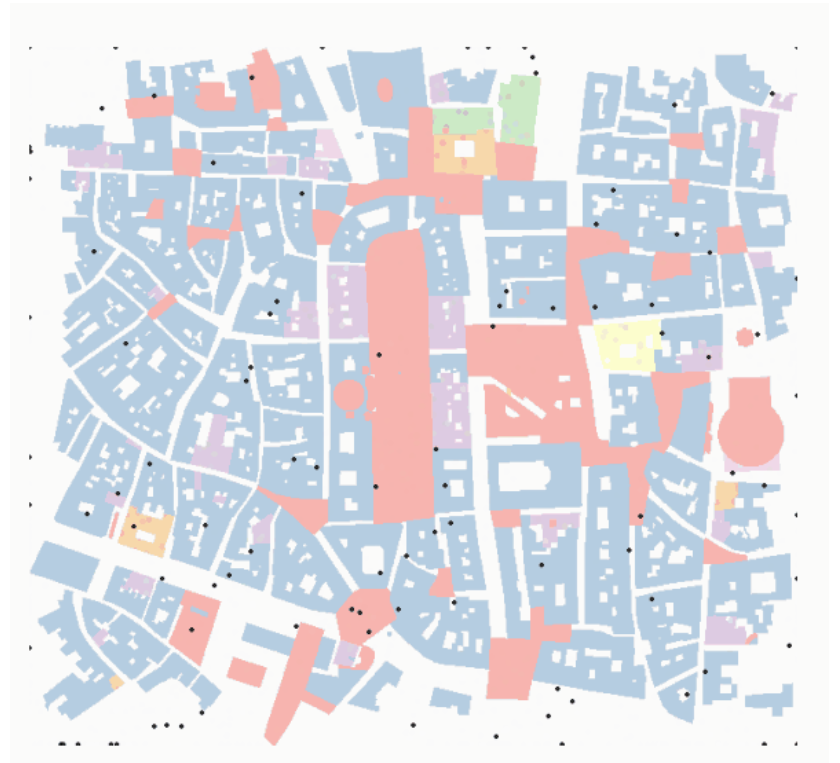
配置建筑对应智能体影响的收益值。

plat\_rewards={  
 1:5,  
 2:-5,  
 3:1,  
 4:1,  
 5:-5,  
 6:-1,  
 7:0.5,  
 8:1,  
 9:1.5,  
 10:1,  
 11:1.5,  
 12:0.5,  
 13:1,  
 14:1,  
 15:0}

从动作空间中随机采样，测试环境是否正常。

env=navona\_v1.env(  
 render\_mode="rgb\_array",  
 plat=plat,  
 plat\_colors=plat\_colors,  
 plat\_rewards=plat\_rewards,  
 agents\_num=100,  
 nodes\_radius=30,  
 group\_dis=0.5,  
 )  
  
env.reset()   
frames=[]  
for agent in env.agent\_iter():  
 frames.append(env.render())  
 observation, reward, termination, truncation, info = env.last()   
  
 if termination or truncation:  
 action = None  
 else:  
 action = env.action\_space(agent).sample()  
  
 env.step(action)  
env.close()

anim=usda\_rl.plot\_animation(frames,interval=300,figsize=(10,10))  
anim.save(filename="../imgs/3\_9\_c/navona\_v1\_rnd.gif", writer="pillow")  
HTML(anim.to\_jshtml())



* 训练模型

env=navona\_v1.parallel\_env(  
 render\_mode="rgb\_array",  
 plat=plat,  
 plat\_colors=plat\_colors,  
 plat\_rewards=plat\_rewards,  
 agents\_num=10,  
 nodes\_radius=30,  
 group\_dis=0.5,  
 )  
  
env = ss.pettingzoo\_env\_to\_vec\_env\_v1(env)  
env = ss.concat\_vec\_envs\_v1(env, 1, num\_cpus=12, base\_class="stable\_baselines3")  
  
model = PPO(  
 MlpPolicy,  
 env,  
 verbose=3,  
 learning\_rate=1e-3,  
 batch\_size=2, #256  
)

Using cuda device

model.learn(total\_timesteps=1) #3E5  
  
navona\_v1\_PPO\_path=r'../models/navona\_v1\_PPO.zip'  
model.save(navona\_v1\_PPO\_path)

env=navona\_v1.env(  
 render\_mode="rgb\_array",  
 plat=plat,  
 plat\_colors=plat\_colors,  
 plat\_rewards=plat\_rewards,  
 agents\_num=10,  
 nodes\_radius=30,  
 group\_dis=0.1,  
 )  
  
# navona\_v1\_PPO\_path=r'../models/navona\_v1\_PPO.zip'  
navona\_v1\_PPO\_path=r'C:\Users\richie\omen\_richiebao\omen\_temp\navona\_v1\_PPO.zip'  
model = PPO.load(navona\_v1\_PPO\_path)  
  
rewards = {agent: 0 for agent in env.possible\_agents}  
frames=[]  
for i in range(3):  
 env.reset(seed=i)  
   
 for agent in env.agent\_iter():  
 frames.append(env.render())  
 obs, reward, termination, truncation, info = env.last()  
 for agent in env.agents:  
 rewards[agent] += env.rewards[agent]  
 if termination or truncation:  
 break  
 else:  
 act = model.predict(obs, deterministic=True)[0]  
  
 env.step(act)  
   
env.close()  
avg\_reward = sum(rewards.values()) / len(rewards.values())  
print("Rewards: ", rewards)  
print(f"Avg reward: {avg\_reward}")

Rewards: {'agent\_0': -12.914863333463302, 'agent\_1': -93.76327083344947, 'agent\_2': 4.834605833198741, 'agent\_3': -13.763270833449354, 'agent\_4': 148.43672916655026, 'agent\_5': -41.31380166678734, 'agent\_6': -103.13078833360264, 'agent\_7': -161.1318500002787, 'agent\_8': -39.7138016667874, 'agent\_9': -227.91380166678744}  
Avg reward: -54.03741133348566

matplotlib.rcParams['animation.embed\_limit'] = 2\*\*128  
  
anim=usda\_rl.plot\_animation(frames,interval=100,figsize=(7,5))  
anim.save(filename="../imgs/3\_9\_c/navona\_v1\_PPO.gif")

HTML(anim.to\_jshtml())

计算结果（待）

注释（Notes）：

① PettingZoo，（<https://pettingzoo.farama.org/>）。

② AEC API，（<https://pettingzoo.farama.org/api/aec/>）。

③ Parallel API，（<https://pettingzoo.farama.org/api/parallel/>）。

④ Arcade（街机）学习环境]，（<https://github.com/Farama-Foundation/Arcade-Learning-Environment>）。

⑤ Farama，（<https://farama.org>）。

⑥ OpenAI’s MPE，（<https://github.com/openai/multiagent-particle-envs>）。

⑦ Jumanj，（<https://github.com/instadeepai/jumanji>）。

⑧ JAX（Autograd and XLA），（<https://github.com/google/jax>）。

⑨ robotic-warehouse（RWARE），（<https://github.com/semitable/robotic-warehouse>）。

⑩ VMAS（Vectorized Multi-Agent Simulator），（<https://github.com/proroklab/VectorizedMultiAgentSimulator>）。

⑪ OpenAI Gym，（<https://github.com/openai/gym>）。

⑫ RLlib，（<https://docs.ray.io/en/latest/rllib/index.html>）。

⑬ torchrl，（<https://github.com/pytorch/rl>）。

⑭ MAgent2，（<https://github.com/Farama-Foundation/MAgent2>）。

⑮ AI-Economist，（<https://github.com/salesforce/ai-economist>）。

⑯ Nocturne，（<https://github.com/facebookresearch/nocturne>）。

⑰ Waymo Open Dataset（Waymo 开源数据集），（<https://github.com/waymo-research/waymo-open-dataset>）。

⑱ Unity ML-Agents，（<https://github.com/Unity-Technologies/ml-agents>）。

⑲ Unity，（<https://unity.com>）。

⑳ Neural MMO，（<https://github.com/openai/neural-mmo>）。

㉑ CleanRL，（<https://github.com/vwxyzjn/cleanrl>）。

㉒ Tianshou (天授) ，（<https://github.com/thu-ml/tianshou>）。

㉓ RLlib-algorithms，（<https://docs.ray.io/en/latest/rllib/rllib-algorithms.html>）。

㉔ Stable-Baselines3（SB3），（<https://stable-baselines3.readthedocs.io/en/master/>）。

㉕ SISL 环境，（<https://pettingzoo.farama.org/environments/sisl/>）。

㉖ MADRL，（<https://github.com/sisl/MADRL>）。

㉗ OpenAI，（<https://openai.com>）。

㉘ gymnasium，（<https://github.com/Farama-Foundation/Gymnasium>）。

㉙ PettingZoo 环境库说明文件，（<https://pettingzoo.farama.org/>）。

㉚ SuperSuit，（<https://pypi.org/project/SuperSuit/>）。

㉛ PettingZoo 自定义环境，（<https://pettingzoo.farama.org/tutorials/custom_environment/1-project-structure/>）。

㉜ PettingZoo 库的 GitHub 代码仓库，（<https://github.com/Farama-Foundation/PettingZoo>）。

㉝ MPE 环境包，（<https://pettingzoo.farama.org/environments/mpe/>）。

㉞ OpenStreetMap（OSM），（<https://www.openstreetmap.org/#map=18/41.89895/12.47073>）。

参考文献（References）:

[1] Ising model-Wikipedia, <https://en.wikipedia.org/wiki/Ising_model>

[2] Metropolis, Nicholas; Rosenbluth, Arianna W.; Rosenbluth, Marshall N.; Teller, Augusta H.; Teller, Edward (1 June 1953). “Equation of State Calculations by Fast Computing Machines”. The Journal of Chemical Physics. 21 (6): 1087–1092. Bibcode:1953JChPh..21.1087M. doi:10.1063/1.1699114. ISSN 0021-9606. OSTI 4390578. S2CID 1046577.

[3] Hastings, W. K. (1 April 1970). “Monte Carlo sampling methods using Markov chains and their applications”. Biometrika. 57 (1): 97–109. Bibcode:1970Bimka..57…97H. doi:10.1093/biomet/57.1.97. ISSN 0006-3444. S2CID 21204149.

[4] Liu, Jun S.; Liang, Faming; Wong, Wing Hung (1 March 2000). “The Multiple-Try Method and Local Optimization in Metropolis Sampling”. Journal of the American Statistical Association. 95 (449): 121–134. doi:10.1080/01621459.2000.10473908. ISSN 0162-1459. S2CID 123468109.

[5] Spall, J. C. (2003). “Estimation via Markov Chain Monte Carlo”. IEEE Control Systems Magazine. 23 (2): 34–45. doi:10.1109/MCS.2003.1188770.

[6] Hill, Stacy D.; Spall, James C. (2019). “Stationarity and Convergence of the Metropolis-Hastings Algorithm: Insights into Theoretical Aspects”. IEEE Control Systems Magazine. 39: 56–67. doi:10.1109/MCS.2018.2876959. S2CID 58672766.

[7] Monte Carlo method-Wikipedia, <https://en.wikipedia.org/wiki/Monte_Carlo_method>

[8] Value of Pi using Monte Carlo – PYTHON PROGRAM, <https://www.bragitoff.com/2021/05/value-of-pi-using-monte-carlo-python-program/>

[9] Metropolis–Hastings algorithm-Wikepedia, <https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm>

[10] Hastings, W.K. (1970). “Monte Carlo Sampling Methods Using Markov Chains and Their Applications”. Biometrika. 57 (1): 97–109. Bibcode:1970Bimka..57…97H. doi:10.1093/biomet/57.1.97. JSTOR 2334940. Zbl 0219.65008.

[11] Simulating the Ising model, <https://github.com/rajeshrinet/compPhy/tree/master/ising/>

[12] Zai, A., & Brown, B. (2020). Deep reinforcement learning in action. Manning Publications.

[13] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, & Jun Wang. (2020). Mean Field Multi-Agent Reinforcement Learning.

[14] Stephan Zheng, Alexander Trott, Sunil Srinivasa, David C. Parkes, & Richard Socher. (2021). The AI Economist: Optimal Economic Policy Design via Two-level Deep Reinforcement Learning.

[15] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, 1928–1937. 2016. URL: http://proceedings.mlr.press/v48/mniha16.html.

[16] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, & Demis Hassabis. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. CoRR, 2017. URL: http://arxiv.org/abs/1707.06347, arXiv:1707.06347.

[18] Horia Mania, Aurelia Guy, & Benjamin Recht. (2018). Simple random search provides a competitive approach to reinforcement learning.

[19] Huang, H.T. (2016). Linear Upper Confidence Bound Algorithm for Contextual Bandit Problem with Piled Rewards. In Advances in Knowledge Discovery and Data Mining (pp. 143–155). Springer International Publishing.

[20] Wang, Q., Xiong, J., Han, L., sun, p., Liu, H., & Zhang, T. (2018). Exponentially Weighted Imitation Learning for Batched Historical Data. In Advances in Neural Information Processing Systems. Curran Associates, Inc..20

[21] Aviral Kumar, Aurick Zhou, George Tucker, & Sergey Levine. (2020). Conservative Q-Learning for Offline Reinforcement Learning.

[22] Ziyu Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, & Nando de Freitas. (2021). Critic Regularized Regression.

[23] Marc G. Bellemare, Will Dabney, & Rémi Munos. (2017). A Distributional Perspective on Reinforcement Learning.

[24] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 2016. URL: http://arxiv.org/abs/1509.02971.

[25] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, & David Silver. (2018). Distributed Prioritized Experience Replay.

[26] Danĳar Hafner, Jurgis Pasukonis, Jimmy Ba, & Timothy Lillicrap. (2023). Mastering Diverse Domains through World Models.

[27] Danĳar Hafner, Timothy Lillicrap, Jimmy Ba, & Mohammad Norouzi. (2020). Dream to Control: Learning Behaviors by Latent Imagination.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015. URL: https://doi.org/10.1038/nature14236, doi:10.1038/nature14236.

[29] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA, 2094–2100. 2016. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389.

[30] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, & Ilya Sutskever. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning.

[31] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 2016. URL: http://arxiv.org/abs/1506.02438.

[32] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, & Wojciech Zaremba. (2018). Hindsight Experience Replay.

[33] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, & Koray Kavukcuoglu. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures.

[34] Lc0, <https://github.com/LeelaChessZero/lc0/>

[35] Chelsea Finn, Pieter Abbeel, & Sergey Levine. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.

[36] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, & Pieter Abbeel. (2018). Model-Based Reinforcement Learning via Meta-Policy Optimization.

[37] NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999, 1057–1063. 1999. URL: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.

[38] Karl Cobbe, Jacob Hilton, Oleg Klimov, & John Schulman. (2020). Phasic Policy Gradient.

[39] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 2016. URL: http://arxiv.org/abs/1511.05952.

[40] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, & Marc G. Bellemare. (2022). Reincarnating Reinforcement Learning: Reusing Prior Computation to Accelerate Progress.

[41] Yuri Burda, Harrison Edwards, Amos Storkey, & Oleg Klimov. (2018). Exploration by Random Network Distillation.

[42] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, & Will Dabney (2018). Recurrent Experience Replay in Distributed Reinforcement Learning. In International Conference on Learning Representations.

[43] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. CoRR, 2018. URL: http://arxiv.org/abs/1812.05905, arXiv:1812.05905.

[44] Ie, E., Jain, V., Wang, J., Narvekar, S., Agarwal, R., Wu, R., Cheng, H.T., Chandra, T., & Boutilier, C. (2019). SLATEQ: A Tractable Decomposition for Reinforcement Learning with Recommendation Sets. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (pp. 2592–2599). AAAI Press.

[45] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, 1582–1591. 2018. URL: http://proceedings.mlr.press/v80/fujimoto18a.html.

[46] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, & Shimon Whiteson. (2018). QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning.

[47] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, & Igor Mordatch. (2020). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments.

[48] Gupta, M. (2017). Cooperative Multi-agent Control Using Deep Reinforcement Learning. In Autonomous Agents and Multiagent Systems (pp. 66–83). Springer International Publishing.

[49] Multi-Agent and Hierarchical, <https://docs.ray.io/en/latest/rllib/rllib-env.html#multi-agent-and-hierarchical>

[50] Implementing a Centralized Critic, <https://docs.ray.io/en/master/rllib/rllib-env.html#implementing-a-centralized-critic>

[51] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, & Trevor Darrell. (2017). Curiosity-driven Exploration by Self-supervised Prediction.

[52] Waterworld, <https://pettingzoo.farama.org/environments/sisl/waterworld/>

[53] SB3: PPO for Waterworld, <https://pettingzoo.farama.org/tutorials/sb3/waterworld/>