## 2.9.1 NetLogo，Mesa和Repast4Py

### 2.9.1.1 智能体模型（Agent-Based Models，ABM）

当人们努力将地理系统及其过程的本质简化和提炼成表征物质世界简单但强大的定律过程时，世界自身的丰富性和多样性则大部分被定义掉了，往往留下即明显又平庸的简单概念。当研究者意识到这些限制，需要从更基本的层面上表征地理系统，开始试图寻找某种基本元素或单位（可以表征一个地方地理特征或某类问题的基本因素），出现了从聚合到非聚合、从群体到个人、从宏观到微观这可察觉的转变[1]2。在20世纪70年代，初步形成了复杂性研究的高潮，其标志是自组织理论的研究。普利高津、哈肯、艾根断言复杂性是物质世界自组织运动的产物，坚持以自组织为基本概念揭示复杂性的本质和来源。20世纪80 年代是复杂性研究的初创阶段，1984年成立的[圣菲研究所( Santa Fe Institute)](https://www.santafe.edu/)①专门从事复杂性研究，1986 年I·普里高津和G·尼科里斯出版了专著《探索复杂性》[2]。复杂性研究从传统科学领域拓展到生物学、经济学、人工智能、计算机科学、生命科学等领域，出现蝴蝶效应、非线性、临界、协同、混沌、自组织和涌现等新思想。而复杂性科学与计算机科学互为推动，复杂适应系统理论之父约翰•霍兰德（John Henry Holland）在其研究遗传算法之初使用计算机进行数学模型的建立，而遗传算法也成为了人工智能开发的重要理论基础。

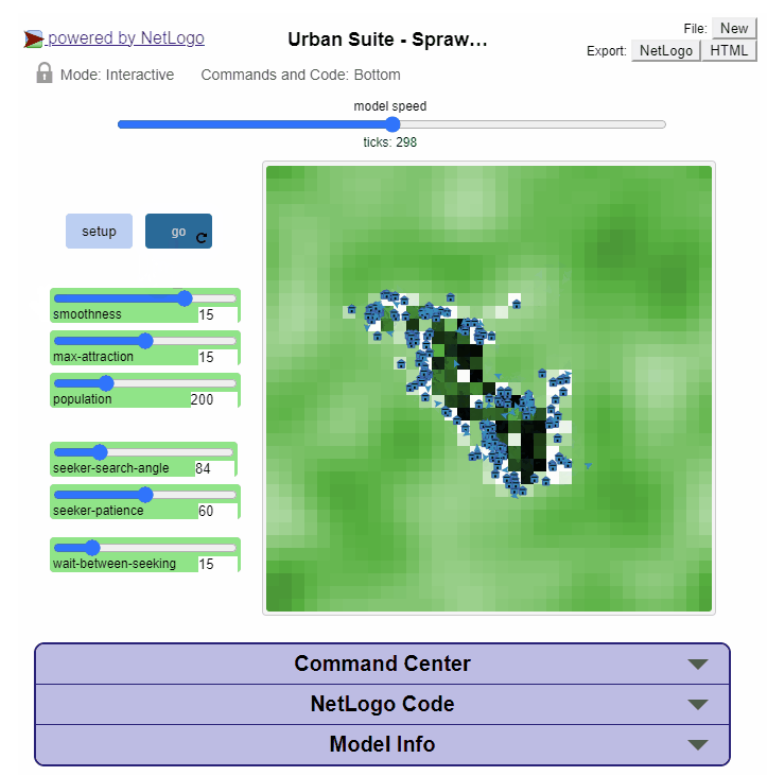
科学中理论发展的常规过程是从观察开始，从这些数据中归纳出一些理论，然后提出一些假设，并在不同时间和不同地点通过其它一些独立的观察集合进行验证。ABM则构建了一种建模风格，具有反映世界丰富性的能力，可以很好的解释城市、区域、全球系统本身及所有物质世界组成的进化和变化的空间结构，但这不同于经典的实验过程，虽然智能体行为的结果可能是可测试的，但与代理行为相关的ABM模型中指定的大部分内容，及行为的过程是不可测试的。因此，对于ABM模型而言模型测试的过程更详细，尤其可信度测试，例如在许多不同初始条件集下运行模型进行实验、模型参数敏感性测试和应用传统算法的最大化拟合优度。同时，应用ABM的另一观点是模型本身不再是为预测而建立，而是为一般的科学探索提供信息，及利益相关者之间关于未来可能会发生什么的任何辩论[1]5。

ABM构建需要模拟/建模系统，包括单独软件或编程语言的支持库等。目前存在有为数不少的ABM模拟系统，例如[Swarm](http://www.swarm.org/wiki/Swarm_main_page)②，[MASON](https://cs.gmu.edu/~eclab/projects/mason/)③（基于Java），[Repast](https://repast.github.io/)④（基于Java版、基于C++版和基于Python版），[StarLogo](https://education.mit.edu/project/starlogo-tng/)⑤（游戏性，节点可视化编程，网页版[StartLogo NOVA](https://www.slnova.org/)⑥），[NetLogo](http://ccl.northwestern.edu/netlogo/index.shtml)⑦（基于Java），[AgentCubes](https://agentsheets.com/)⑧（游戏性，节点可视化编程，商业——面向儿童编程教学），[AnyLogic](https://www.anylogic.com/)⑨(商业——面向工业)，和[Mesa](https://mesa.readthedocs.io/en/stable/index.html)⑩(Python库)，[AgentPy](https://agentpy.readthedocs.io/en/latest/)⑪，[PythonABM](https://pythonabm.readthedocs.io/en/latest/index.html)⑫等，其中基于Python开发的ABM库开始增多，结合Python自身庞大的库体系，适应于更为广泛的应用。综合ABM模拟系统是否在持续维护、持续开发；是否拥有广泛的用户群，并有更多的贡献者；是否有系统的说明手册或教程，及大量示范模型；是否可以结合到地理空间数据模拟；是否支持或拥有Python版本等，筛选出NetLogo、Repast和Mesa三个ABM模拟系统，其中，NetLogo为独立的可编程软件，含有大量已有示例模型，使用易于学习包括StarLogo和StarLogoT的多代理建模语言，但是因为自身多代理建模语言的局限性，这不仅在于自身语言的结构，同样在于自身仅用于ABM建模，因此很难拓展应用已有大量的研究来构建新模型；Repast看到了Python拥有庞大用户群。持续的发展势头，和大量以Python代码形式存在的已有知识库（例如各类数据处理库、地理空间信息分析库和机器（深度）学习库等）的优势，2022年10月释放了Repast for Python（Repast4Py）1.0版，并在2022年12月释放了2.0版，依托Repast自身已有的积累（模型和用户、社区）和对庞大模型的拓展支持、多线程计算方式、及对更复杂模型的弹性构建都使得Repast4Py具有更广阔的发展潜力；因为Python的发展潜力，依托Python库开发的ABM模拟系统越来越多，其中典型的是Mesa库，具有相对较多的代码贡献者，并在持续维护和更新，ABM模拟系统构建的代码结构易于理解，且类似NetLogo可视化智能体的运动，这为Mesa带来了更多潜在的用户。NetLogo、Repast4Py和Mesa均提供了系统的说明手册、教程，方便使用者学习。

### 2.9.1.2 NetLogo，Mesa和Repast4Py 模拟系统比较

#### 1) NetLogo的模型Sprawl Effect

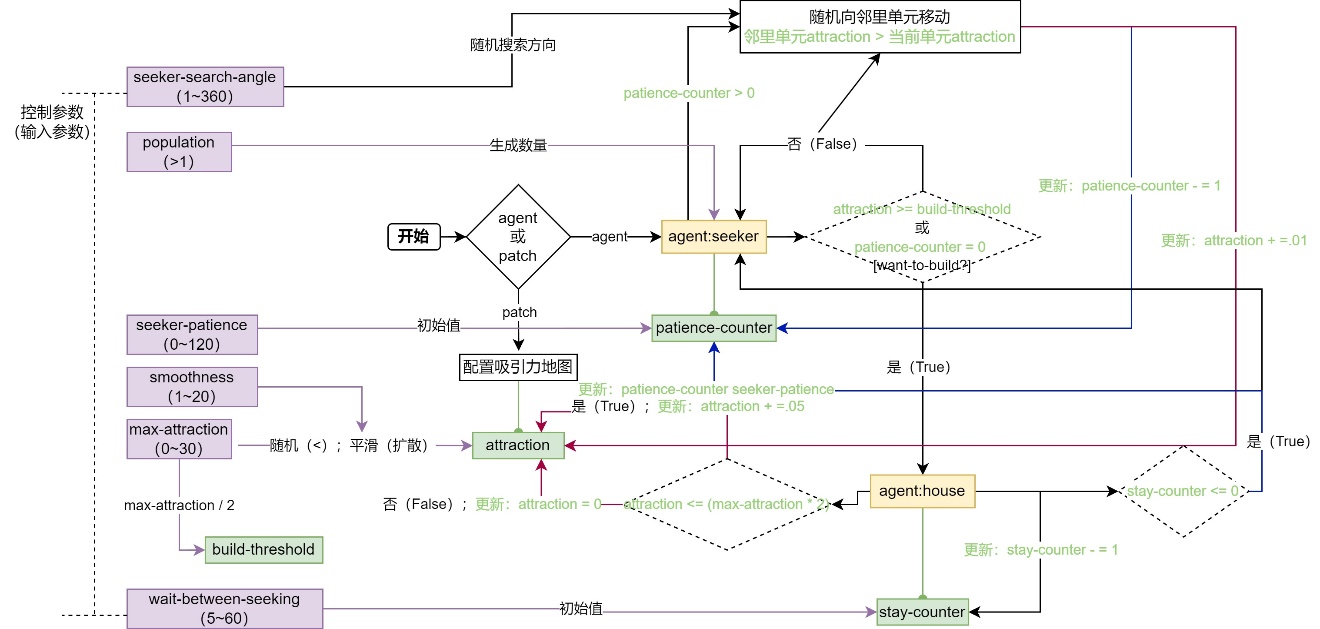
选择NetLogo模型库中的“Sprawl Effect”[3]智能体模型来说明ABM在城市空间规划设计领域一般应用模式，并比较不同的模拟系统。“Sprawl Effect”模型展示了城市扩张的一个简单模式，可以探索影响城市蔓延的因素，及影响过程。虽然模型条件简单，无法实现对现实城市发展更多细节上的描述，但是可以说明、模拟和展示某些行为和土地利用模式间的关系。



“Sprawl Effect”模型的说明文件解释了该模型的工作机制，同时可以结合NetoLogo的代码更深入的理解这个实现过程。该模型首先通过to setup-patches程序块构建了吸引力地图（opography of attractiveness）（较亮的网格单元具有较高的吸引力值attraction；较暗的网格单元则值较低）。吸引力地图是通过set attraction ( random max-attraction )语句随机生成小于给定最大吸引力值random max-attraction的随机数，并通过repeat和diffuse相结合的方法，使得每个单元都从各个邻近单元获得其0.4（40%，该值可以根据分析情况调整）吸引力值的1/8，同时邻近单元的吸引力相应的减少对应的值，保持整个分析网格所有单元的吸引力值之和不变。模型中的智能体代表居住人口，且包含两种状态，一个是seeker搜寻者，另一个是house房屋。

在seeker状态下，定义to turn-toward-attraction程序来根据邻近单元的吸引力值确定搜寻的方向。给定搜寻角度seeker-search-angle，在左右搜寻角度内采样吸引力值attraction，比较吸引力值与当前位置值的大小，及搜索范围内左右值的大小，确定搜寻方向，具体的搜寻方向为确定的搜寻方向内的一个随机值，即执行rt random seeker-search-angle或lt random seeker-search-angle语句，然后超该方向前进0.5步（fd 0.5）。迭代每一步时，seeker还需要决定是否要在当前网格单元上定居，成为house状态，通过定义to-report want-to-build?程序，比较随机获取的一个小于给定最大吸引力的值random attraction与build-threshold参数（为最大吸引力值max-attraction的一半）返回一个布尔值（True或者False），同时确定patience-counter参数是否为0，也返回一个布尔值。是否要定居的条件是随机从上述返回的两个布尔值中随机抽取一个结果，如果为True，则seeker转换为house状态。这里模型假设了一种情况，如果两个布尔值中随机抽取的一个结果为False，虽然不会定居，即将seeker转变为house，但是会增加该搜索单元的吸引力，执行set attraction attraction + .01增加吸引力值；另一条规则是，土地的吸引力值不会一直增加，在一块土地上进行过多的活动会降低吸引力，因此当一个house状态单元的吸引力执行set attraction attraction + .05语句不断增加，达到最大吸引力的2倍时，则将该单元的吸引力重置为0。虽然这种将单元（土地）吸引力突然变为0的做法不太现实，但不断被占用的土地的吸引力随时间推移而降低的想法是可能的。

为了方便观察智能体的行为，绘制流程图如下，其中紫色框为控制参数（输入参数），给出了参考取值范围；黄色框为智能体的两种状态seeker和house；绿色框为影响因子，其中attraction因子属于`patch对象，patient-counter因子属于智能体seeker对象，stay-counter因子属于house对象，而build-threshold为全局变量；同时用线的颜色标示了因子更新的流向行为，例如patience-counter有2个流向行为更新，attractin有3个流向行为更新，stay-counter因子有1个流向行为更新。



“Sprawl Effect”模型的NetLogo代码摘录如下：

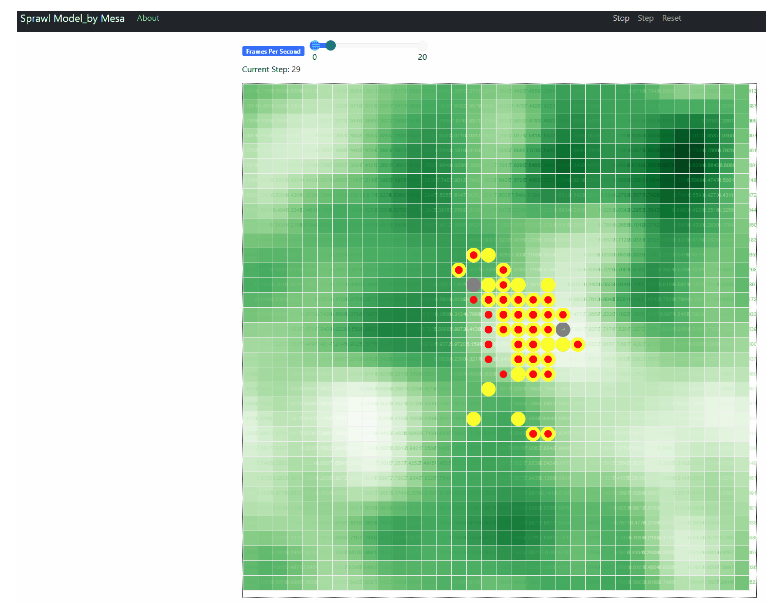
globals [ build-threshold ]  
patches-own [ attraction ]  
  
breed [ houses house ]  
breed [ seekers seeker ]  
  
houses-own [ stay-counter ]  
seekers-own [ patience-counter ]  
  
to setup  
 clear-all  
 setup-patches  
 setup-turtles  
 set build-threshold floor (max-attraction / 2)  
 reset-ticks  
end  
  
to setup-patches  
 ask patches  
 [  
 set attraction ( random max-attraction )  
 ]  
 repeat smoothness  
 [  
 diffuse attraction .4  
 ]  
 ask patches  
 [  
 set pcolor scale-color green attraction 2.5 10  
 ]  
end  
  
to setup-turtles  
 create-seekers population  
 [  
 set color sky  
 set shape "default"  
 set patience-counter seeker-patience  
 set size .75  
 ]  
 ask turtles  
 [  
 setxy 0 0  
 ]  
 ;; if you want to start the turtles at the most attractive location on the map,  
 ;; uncomment the following line  
 ; ask turtles [ move-to max-one-of patches [ attraction ] ]  
end  
  
to go  
 ask seekers  
 [  
 ifelse (want-to-build?)  
 [  
 set breed houses  
 set shape "blue-house"  
 set stay-counter wait-between-seeking  
 ]  
 [  
 if (patience-counter) > 0  
 [  
 turn-toward-attraction  
 fd 0.5  
 set patience-counter patience-counter - 1  
 set attraction attraction + .01  
 ]  
 ]  
  
 ]  
 ask houses  
 [  
  
 ifelse attraction <= (max-attraction \* 2)  
 [ set attraction attraction + .05 ]  
 [ set attraction 0 ]  
  
 set stay-counter stay-counter - 1  
 if (stay-counter) <= 0  
 [  
 set breed seekers  
 set patience-counter seeker-patience  
 set shape "default"  
 ]  
 ]  
 ask patches [ set pcolor scale-color green attraction 2.5 10 ]  
 tick  
end  
  
to-report want-to-build?  
 report random attraction >= build-threshold or patience-counter = 0  
end  
  
to turn-toward-attraction  
 let ahead [attraction] of patch-ahead 1  
 let myright [attraction] of patch-right-and-ahead seeker-search-angle 1  
 let myleft [attraction] of patch-left-and-ahead seeker-search-angle 1  
 ifelse ((myright > ahead) and (myright > myleft))  
 [  
 rt random seeker-search-angle  
 ]  
 [  
 if (myleft > ahead)  
 [ lt random seeker-search-angle ]  
 ]  
end  
  
  
; Copyright 2007 Uri Wilensky.  
; See Info tab for full copyright and license.

#### 2) Mesa重写模型Sprawl Effect

ABM是一种建模方式，不管是何种模拟系统都具有共同相似的核心逻辑，但是在流程结构设计、方法调用方式和内容数量上会不同。Mesa相比NetLogo而言已经具有了ABM的核心逻辑，也能够可视化运行过程，并收集过程数据建立分析图表，但是功能方法不及NetLogo丰富，这可能存在多种原因，1是，Mesa模拟系统的搭建远晚于NetoLogo，系统仍旧在社区的贡献下处于开发期或调整更新期；2是，因为以Python为平台，很多方法完全可以借助已有库或者自行编写完成，避免Mesa过于冗余，而保持精简。

通过比较Mesa和NetLogo的代码书写方式，以Python编程语言为依托的Mesa或者类似的Python库更容易拓展，也更容易解决NetoLogo很多无法处理的问题，例如融入大数据处理、机器（深度）学习等内容。就Mesa而言，其模拟系统结构清晰，容易构建智能体模型，但仍有些代码方法调用晦涩的部分，例如NetLogo对patch的处理异常丰富，但是Mesa对应的网格Grid目前拥有的功能有限，例如无法使用Grid构建吸引力地图，而是通过构建智能体agent\_house的方式实现，可能这与开发者的目的有关，例如Grid只是承载智能体的基底；再者，NetLogo中有现成的方法可以非常方便的获取智能体所在单元和任何指定单元，指定智能体类型的相关属性值，但在Mesa中实现，需要通过get\_cell\_list\_contents方法一次获取所有单元中包含的智能体对象，并通过循环判断智能体是否为某个类型智能体后再读取属性值，因为这个功能会频繁使用，因此略显繁复的操作增加了代码书写的负担；对于Mesa的可视化界面，虽然可以完成智能体的迭代过程，但是如果增大数据量（智能体数量等），演示的速度会滞涩。

即使Mesa存在很多不足，也不能确保其未来开发的程度，但是当前版本足可以用于探索城市空间相关研究内容。下图为用Mesa重写Sprawl Effect模型后，可视化运行的结果。其中对于吸引力地图（绿色变化单元，智能体agent\_patch）只是显示了初始值，并未书写吸引力地图更新后的变化值地图，但是在每一单元上文字标识了吸引力值；黄色圆形为智能体房屋（agent\_house）对象，如果变为灰色，则为stay\_counter因子小于0的状态；红色圆形为智能体搜索者agent\_seeker对象。



在保持原模型基本算法不变的条件下，适应Mesa模拟系统做出几点调整，

一个是，NetLogo可以非常方便的转换智能体类型，例如seeker对象和house对象，但Mesa中目前没有发现有对应set breed houses的方法，因此选择了构建两类智能体agent\_seeker和agent\_house，对应seeker对象和house对象；

二是，吸引力地图用智能体对象构建，在生成地图数据时，定义了diffuse\_attractiveness()和generate\_attraction\_colors()函数实现，使用的方法是自定义卷积核，执行二维卷积，并保持卷积前后所有单元吸引力值和不变；

三是，对应to turn-toward-attraction，定义了turn\_toward\_attraction()函数，其逻辑有所调整，是随机选择邻里8个单元中吸引力值均高于当前单元吸引力值的单元为智能体seeker移至单元，如果没有高于当前单元的邻里单元，则保持不动。原模型是根据指定方向角度，满足吸引力值大于当前值情况下，随机获取一个指定角度值内的角度值；

四是，控制参数（输入参数）的配置上，使用了argparse库，而没有构建按钮、滑条等可视化方式。

* 调入库和配置控制参数

import mesa  
from random import randrange,choice  
import numpy as np  
from scipy.signal import convolve2d  
from matplotlib import pyplot as plt, colors  
import math  
import itertools   
import argparse  
from tqdm import tqdm  
  
parser=argparse.ArgumentParser(prog='Urban Sprawl\_mesa(python)version',description='转换NetLogo的Sprawl Effect为Python 的Mesa版本')  
parser.add\_argument('--width', default=34, type=int,help='模拟网格宽')  
parser.add\_argument('--height', default=34, type=int,help='模拟网格高')  
parser.add\_argument('--max\_attraction', default=15, type=int,help='吸引力最大值，取值范围 0~30')  
parser.add\_argument('--smoothness', default=15, type=int,help='卷积（光滑）次数，取值范围 1~20')  
parser.add\_argument('--share', default=.4, type=float,help='用于卷积核的分配因子（对应NetLogo版的diffuse参数），取值范围 0~1')  
parser.add\_argument('--population\_seeker', default=200, type=int,help='智能体搜寻着（agent\_seeker）初始化数量，取值范围 1~750')  
parser.add\_argument('--seeker\_patience', default=60, type=int,help='智能体搜寻者（agent\_seeker）的寻找耐心程度，取值范围 0~120')  
parser.add\_argument('--wait\_between\_seeking', default=15, type=int,help='智能体搜寻者（agent\_seeker）的寻找耐心程度，取值范围 5~60')  
parser.add\_argument('--agent\_house\_uniqueIDs', default=list(range(0,100000)), type=list,help='用于智能体房屋（agent\_house）的ID标识')  
  
args=parser.parse\_args([])

* 构建智能体类

class agent\_patch(mesa.Agent):  
 '''用于构建吸引力地图的智能体'''  
 def \_\_init\_\_(self,pos,unique\_id,attraction,color,model):  
 super().\_\_init\_\_(unique\_id,model)  
 self.pos=pos  
 self.attraction=attraction  
 self.color=color  
   
 def step(self):  
 pass   
   
class agent\_house(mesa.Agent):  
 '''智能体——房屋（house）'''  
 def \_\_init\_\_(self,pos,unique\_id,model):  
 super().\_\_init\_\_(unique\_id,model)  
 self.pos=pos   
 self.stay\_counter=args.wait\_between\_seeking  
 self.dropped=0  
   
 def house\_state\_update(self):   
 '''更新智能体—房屋所在位置的吸引力值'''  
 this\_cell=self.model.grid.get\_cell\_list\_contents([self.pos])   
 turtle\_attraction=[i for i in this\_cell if type(i) is agent\_patch][0]   
 if turtle\_attraction.attraction <= self.model.max\_attraction\*2:  
 turtle\_attraction.attraction+=.05  
 else:   
 turtle\_attraction.attraction=0   
   
 self.stay\_counter-=1  
 if self.stay\_counter <= 0:   
 # self.model.grid.remove\_agent(self)  
 self.dropped=1  
   
 def step(self):   
 self.house\_state\_update()  
   
class agent\_seeker(mesa.Agent):  
 '''智能体——搜寻着（seeker）'''  
 def \_\_init\_\_(self,pos,unique\_id,model):  
 super().\_\_init\_\_(unique\_id,model)  
 self.pos=pos  
 self.foundit=0  
 self.patience\_counter=args.seeker\_patience   
   
 def seeking(self):  
 '''智能体seeker每步（tick）行为'''  
 if self.want\_to\_build():  
 self.foundit=1  
 self.generate\_agent\_house()  
 else:  
 if self.patience\_counter>0:  
 neighbors\_filtered=self.turn\_toward\_attraction()  
 if neighbors\_filtered:  
 neighbors\_choice=choice(neighbors\_filtered)  
 self.model.grid.move\_agent(self,neighbors\_choice)  
 self.patience\_counter-=1  
   
 this\_cell=self.model.grid.get\_cell\_list\_contents([self.pos])  
 turtle\_attraction=[i for i in this\_cell if type(i) is agent\_patch][0]  
 turtle\_attraction.attraction+=.01  
   
 def want\_to\_build(self):  
 '''同NetLogo版的 to-report want-to-build? 部分'''  
 this\_cell=self.model.grid.get\_cell\_list\_contents([self.pos])  
 turtle\_patch=[i for i in this\_cell if type(i) is agent\_patch][0]   
 selection\_lst=[turtle\_patch.attraction >= self.model.build\_threshold, self.patience\_counter==0]  
   
 return choice(selection\_lst)  
   
 def generate\_agent\_house(self):  
 '''在智能体seeker当前位置，生成一个智能体房屋（agent\_house）'''  
 turtle\_house=agent\_house(self.pos,f"house\_{args.agent\_house\_uniqueIDs.pop()}",self.model)   
 self.model.schedule.add(turtle\_house)   
 self.model.grid.place\_agent(turtle\_house, self.pos)   
   
 def turn\_toward\_attraction(self):  
 '''根据当前位置单元和邻里8个单元的吸引力值，选择大于当前位置吸引力值的邻里单元为智能体seeker的移至位置单元'''  
 neighbors=[i for i in self.model.grid.get\_neighborhood(self.pos, moore=True,include\_center=True,radius=1)]  
 attraction\_neighbors=[]  
 for xy in neighbors:  
 this\_cell=self.model.grid.get\_cell\_list\_contents([xy])  
 turtle\_attraction=[i for i in this\_cell if type(i) is agent\_patch][0]   
 attraction\_neighbors.append(turtle\_attraction.attraction)  
   
 local\_attraction=attraction\_neighbors.pop(4) # 智能体当前位置值  
 neighbors.pop(4)   
 neighbors\_filter=[i>local\_attraction for i in attraction\_neighbors]   
 neighbors\_selection=list(itertools.compress(neighbors,neighbors\_filter))  
  
 return neighbors\_selection  
   
 def step(self):   
 self.seeking()

* 模型初始化

class sprawl\_model(mesa.Model):  
 '''初始化模型'''  
 def \_\_init\_\_(self,  
 width=args.width,  
 height=args.height,  
 max\_attraction=args.max\_attraction,  
 smoothness=args.smoothness,  
 share=args.share,  
 population\_seeker=args.population\_seeker):  
   
 self.max\_attraction=max\_attraction  
 self.build\_threshold=math.floor(max\_attraction/2)  
   
 # 定义模拟网格为MultiGrid类型，每个单元格（cell）可以包含多个智能体对象  
 self.grid=mesa.space.MultiGrid(width,height,torus=False)  
 # 定义调度器（scheduler）类型为RandomActivation。每一步以随机顺序激活一个智能体，且每一步都会重新随机排序。  
 # 等同于NetLogo的'ask agents'，且通常为默认行为  
 self.schedule=mesa.time.RandomActivation(self)   
  
 # 以智能体方式，构建吸引力地图（topography of attractiveness ）。首先建立随机吸引子值2d数组（矩阵）  
 attractiveness=self.diffuse\_attractiveness(width,height,self.max\_attraction,smoothness,share)   
 # 生成吸引力地图颜色值（对应吸引力二维矩阵值）  
 attraction\_colors=self.generate\_attraction\_colors(attractiveness,reverse=True)  
 # 生成吸引力地图（即智能体agent\_patch）  
 self.setup\_agent\_patch(attractiveness,attraction\_colors)   
 # 初始化智能体搜寻者（seeker）  
 self.setup\_agent\_seeker(population\_seeker,pos=(width//2,height//2))   
  
 def diffuse\_attractiveness(self,width,height,max\_attraction,smoothness,share):   
 '''用二维卷积的方式实现NetLogo中的diffuse方法；用循环的方式实现NetLogo中的repeat方法，完成吸引力地图二维矩阵值的随机生成'''  
 attraction=np.random.randint(0,max\_attraction,size=(width,height),dtype=np.uint32)  
 share\_percent=share/8  
 kernel=np.array([[share\_percent,share\_percent,share\_percent],  
 [share\_percent, 1-share,share\_percent],  
 [share\_percent,share\_percent,share\_percent]])   
 diffusion\_attraction=np.copy(attraction)  
 i=0  
   
 while i<smoothness:  
 diffusion\_attraction=convolve2d(diffusion\_attraction,kernel,mode='same',boundary='wrap')  
 i+=1  
  
 return diffusion\_attraction   
  
 def generate\_attraction\_colors(self,array,cmap=plt.cm.Greens,reverse=False):  
 '''根据矩阵值对应生成颜色值'''  
 norm=colors.Normalize() # vmin，vmax配置为默认值  
 color=[colors.to\_hex(c) for c in cmap(norm(array.reshape(-1)))]  
 if reverse:  
 color.reverse()  
 return np.reshape(color,array.shape)   
  
 def setup\_agent\_patch(self,attraction,colors):  
 '''以智能体形式生成吸引力地图'''  
 i=0  
 for \_, x, y in self.grid.coord\_iter():  
 pos=(x,y)  
 turtle\_attraction=agent\_patch(pos,f"attraction\_{i}",attraction[x,y],colors[x,y],self)  
 self.schedule.add(turtle\_attraction)   
 self.grid.place\_agent(turtle\_attraction,pos)  
 i+=1  
   
 def setup\_agent\_seeker(self,population,pos):  
 '''初始化智能体搜寻者（seeker）'''  
 for i in range(population):  
 turtle\_seeker=agent\_seeker(pos,i,self) #f"seeker\_{i}"  
 self.schedule.add(turtle\_seeker)   
 self.grid.place\_agent(turtle\_seeker,pos)   
  
 def step(self):  
 # self.datacollector.collect(self)   
 self.schedule.step()

* 行为模拟可视化参数配置

def agent\_portrayal(agent):  
 seekers\_colors={0:'blue',1:'red'}  
 dropped\_colors={0:'yellow',1:'grey'}  
   
 if agent is None:  
 return   
 if type(agent) is agent\_patch:  
 portrayal={  
 "Shape": "rect",  
 "Color": agent.color,  
 "Filled": "true",  
 "Layer": 0,  
 "w": 1,  
 "h": 1,  
 "text":agent.attraction,  
 # "text\_color":'grey',  
 }   
   
 elif type(agent) is agent\_seeker:  
 portrayal = {  
 "Shape": "circle",  
 "Color": seekers\_colors[agent.foundit],  
 "Filled": "true",  
 "Layer": 2,  
 "text": agent.patience\_counter,  
 "r": 0.5,  
 "scale": 2,  
 }   
   
 elif type(agent) is agent\_house:  
 portrayal = {  
 "Shape": "circle",   
 "Color": dropped\_colors[agent.dropped],  
 "Filled": "true",  
 "Layer": 1,  
 "text": agent.stay\_counter,  
 "r": 1,  
 "scale": 2,  
 }   
   
 return portrayal

* 模拟运行

因为输入参数以argparse库方式全局配置，因此并未在mesa.visualization.ModularServer下以字典形式传入参数。

grid=mesa.visualization.CanvasGrid(agent\_portrayal,args.height,args.width,1000,1000)  
server=mesa.visualization.ModularServer(  
 sprawl\_model,  
 [grid],  
 "Sprawl Model\_by Mesa"  
 )  
server.launch(port=8526)

Interface starting at http://127.0.0.1:8526  
  
Socket opened!  
{"type":"reset"}  
{"type":"get\_step","step":1}  
{"type":"get\_step","step":2}  
{"type":"get\_step","step":3}  
{"type":"get\_step","step":4}

#### 3) Repast4Py 重写模型 Sprawl Effect

[Repast Suite](https://repast.github.io/)④是一系列免费开源的智能体模型建模和模拟系统，这些系统已经持续开发了20多年，包括Repast Simphony，为一个交互丰富且易于学习基于Java的建模工具包，专为在工作站和小型计算集群上使用；Repast for High Performance Computing （Repaset HPC），是一个精简且以专家应用为中心基于C++的分布式建模工具包，旨在用于大型计算机集群和超级计算机；[Repast for Python （Repast4Py）](https://repast.github.io/repast4py.site/guide/user_guide.html#_footnotedef_3)⑬为Repast Suite的新成员，以Repaset HPC为基础开发的基于Python的分布式建模工具，是Python包，旨在为来自不同科学领域的研究人员提供更容易的入口，以应用大规模分布式ABM方法。其利用[Numba](https://numba.readthedocs.io/en/stable/user/5minguide.html)⑭、Numpy和PyTorch包及Python C API创建可扩展的建模系统。

ABM通常会包含大量智能体（代理），而各个智能体在每个时间步（tick）或以某个频率执行某些行为。完成迭代循环模拟的时间取决于智能体的数量和执行行为的复杂性，Repast Suite通过将智能体分布到并行运行的多个进程中，每个进程仅对群体的一个子集执行自己的循环，以减小模拟时间，可以创建更大智能体群体和更复杂的行为，这有益于地理空间信息相关的探索。

Repast4Py可以在 Linux, macOS和Windows上运行，但前提是安装了有效的[MPI（Message Passing Interface）](https://en.wikipedia.org/wiki/Open_MPI)⑮并支持[mpi4py](https://mpi4py.readthedocs.io/en/stable/)⑯。Repast4Py在Linux上开发和测试，因此对于Windows用户建议在Windows系统安装Linux子系统（ Windows Subsystem for Linux ，WSL）。WSL的安装方法在Microsoft官网有说明，为[Install Linux on Windows with WSL](https://learn.microsoft.com/en-us/windows/wsl/install)⑰。Repast4Py的安装则在其官网也给出了详细的说明，通常在Linux（Ubuntu）终端中执行sudo apt install mpich和env CC=mpicxx pip install repast4py完成安装。

虽然Repast4Py提供了ABM信息打印和输出的相关模块repast4py.logging，但目前Repast4Py提供的方法中还未涉及智能体模拟运行的可视化途径，而可视化是查看智能体是否执行行为，根据行为调整参数和行为规则，及分析智能体空间分布或其属性值分布情况来做出相关研究判断或作为研究结果的重要条件。目前并不确定Repast4Py开发团队在未来的开发计划中是否加入可视化模块，因此对于当前的Repast4Py，可以自行构建可视化代码（简单的或复杂的），这包括两种方式，其一是记录模拟过程的智能体位置数据和其属性数据，保存至本地磁盘，模拟结束后可视化模拟过程；其二是与NetoLogo和Mesa方法同，模拟过程中实现可视化。本次实验选择了简单的第一种方法，模拟结束后再可视化模拟过程。

将Sprawl Effect模型用Repast4Py完成Python书写，与Mesa类似，这不仅包括ABM模拟系统代码结构上的类似，任何ABM模拟系统均具有相同的内核；同样，因为构建同一智能体行为规则，因此智能体部分基本上可以迁移Mesa完成的代码，因为二者均是基于Python编程语言的库。Repast4Py和Mesa最大的区别是，Repast4Py对分布计算的支持；目前Mesa支持可视化，而Repast4Py尚未开发；且Repast4Py通常在Linux系统下允许，而Mesa可以直接在Windows下Python解释器中运行（例如Anaconda）。

除了NetoLog，Mesa和Repast4Py的比较，这里引入了*Agent-Based Models of Geographical Systems*一书中*Designing and Building an Agent-Based Model*[1]152一章对Swarm、RePast、Mason和NetLogo的比较表格。因为该书出版于2012年，基于Python的ABM库鲜见，Mesa还未出现，RePast也尚未构建Repast4Py的Python包，但这个比较表格可以看到非Python下ABM工具使用的情况。从中可以看到RePast在多项指标中都有不错的成绩，而Repast4Py的加入，会使得RePast仍旧保持主流，且吸引更多的用户选择Repast4Py为ABM的模拟系统。

|  | Swarm | RePast | Mason | NetLogo |
| --- | --- | --- | --- | --- |
| Licence | GPL | GPL | GPL | Free, but not open source |
| Documentation | Patchy | Limited | Improving, but limited | Good |
| User base | Diminishing | Large | Increasing | Large |
| Modelling language(s) | Objective-C, Java | Java, Python | Java | NetLogo |
| Speed of execution | Moderate | Fast | Fastest | Moderate |
| Support for graphical user interface development | Limited | Good | Good | Very easy to create, using “point and click” |
| Build-in ability to create movies and animations | No | Yes | Yes | Yes |
| Support for systematic experimentations | Some | Yes | Yes | Yes |
| Ease of Learning and Programming | Poor | Moderate | Moderate | Good |
| Ease of Installation | Poor | Moderate | Moderate | Good |
| Link to geographical Information System | No | Yes | Yes | Yes |

Sprawl Effect模型的Repast4Py版，为Windows的WSL下完成书写，仅有一个rank（一个线程（process）），即未使用分布式模拟，但代码中保留了使用分布式的痕迹（但未书写restore\_agent方法，用于智能体从一个线程移动到另一个时，即重新创建智能体所需转移数据）。使用的Python解释器为Linux下安装的Spyder。模拟代码位于一个模块下。

* 导入模块和配置控制参数

控制参数的配置上分成了两部分，一部分使用argparse方法传入；另一部分定义params字典，作为参数传入模型。

import sys  
import math  
import numpy as np  
from typing import Dict, Tuple  
from mpi4py import MPI  
from dataclasses import dataclass  
  
import numba  
from numba import int32, int64  
from numba.experimental import jitclass  
  
from repast4py import core, space, schedule, logging, random  
from repast4py import context as ctx  
from repast4py.parameters import create\_args\_parser, init\_params  
  
from repast4py.space import ContinuousPoint as cpt  
from repast4py.space import DiscretePoint as dpt  
from repast4py.space import BorderType, OccupancyType  
  
from scipy.signal import convolve2d  
from random import randrange,choice  
from matplotlib import pyplot as plt, colors  
import argparse  
  
parser=argparse.ArgumentParser(prog='Urban Sprawl\_repast4Py(python)version',description='转换NetLogo的Sprawl Effect为Python 的Mesa版本')  
parser.add\_argument('--max\_attraction', default=15, type=int,help='吸引力最大值，取值范围 0~30')  
parser.add\_argument('--smoothness', default=15, type=int,help='卷积（光滑）次数，取值范围 1~20')  
parser.add\_argument('--share', default=.4, type=float,help='用于卷积核的分配因子（对应NetLogo版的diffuse参数），取值范围 0~1')  
parser.add\_argument('--seeker\_patience', default=60, type=int,help='智能体搜寻者（agent\_seeker）的寻找耐心程度，取值范围 0~120')  
parser.add\_argument('--wait\_between\_seeking', default=15, type=int,help='智能体搜寻者（agent\_seeker）的寻找耐心程度，取值范围 5~60')  
  
args=parser.parse\_args([])

* 网格邻居查找器

迁移Repast4Py提供的GridNghFinder方法，该方法是一个是使用NumPy和Numba库快速计算相邻网格单元位置的类。NumPy是科学计算包，提供基于多维数组和矩阵的支持，及在这些数组上运行快速和优化的数学函数；Numba是Python的即时编译器，可以将某些类型的Python函数和类编译为优化的本机代码，绕过较慢的Python解释器，提升计算效率。

@numba.jit((int64[:], int64[:]), nopython=True)  
def is\_equal(a1, a2):  
 return a1[0] == a2[0] and a1[1] == a2[1]  
  
spec=[  
 ('mo', int32[:]),  
 ('no', int32[:]),  
 ('xmin', int32),  
 ('ymin', int32),  
 ('ymax', int32),  
 ('xmax', int32)  
]  
  
@jitclass(spec)  
class GridNghFinder:  
  
 def \_\_init\_\_(self, xmin, ymin, xmax, ymax):  
 self.mo = np.array([-1, 0, 1, -1, 0, 1, -1, 0, 1], dtype=np.int32)  
 self.no = np.array([1, 1, 1, 0, 0, 0, -1, -1, -1], dtype=np.int32)  
 self.xmin = xmin  
 self.ymin = ymin  
 self.xmax = xmax  
 self.ymax = ymax  
  
 def find(self, x, y):  
 xs = self.mo + x  
 ys = self.no + y  
  
 xd = (xs >= self.xmin) & (xs <= self.xmax)  
 xs = xs[xd]  
 ys = ys[xd]  
  
 yd = (ys >= self.ymin) & (ys <= self.ymax)  
 xs = xs[yd]  
 ys = ys[yd]  
  
 return np.stack((xs, ys, np.zeros(len(ys), dtype=np.int32)), axis=-1)

* 定义日志数据结构

用于logging.create\_loggers方法构建日志的数据结构。

@dataclass  
class Info:  
 """  
 Dataclass used by repast4py aggregate logging to record the info from agent houses after each tick.  
 """  
 house\_total: int = 0  
 seeker\_total: int = 0

* 构建智能体类

基本同Mesa构建智能体的方法，只是根据位置提取单元智能体对象的方法略有差异，例如Mesa中使用如下语句，使用type类型方式判断属于哪类智能体。

this\_cell=self.model.grid.get\_cell\_list\_contents([self.pos])  
turtle\_patch=[i for i in this\_cell if type(i) is agent\_patch][0]

Repast4Py使用语句示例如下，使用的是Patch.TYPE方法，因为Repast4Py在定义智能体时指定了类型并用super().\_\_init\_\_(id=a\_id, type=Seeker.TYPE, rank=rank)传入到了父类core.Agent下，，因此通过get\_agents返回的单元智能体包括该属性，通过obj.uid[1]方式获取：

objs=self.grid.get\_agents(self.pt)  
agent\_attraction=[obj for obj in objs if obj.uid[1]==Patch.TYPE][0]

比较Mesa和Repast4Py定义智能体的差异，Mesa父类mesa.Agent初始化时传入的两个参数为unique\_id和model，因此Mesa的智能体可以直接调用模型model中定义的方法和变量，但未标识智能体类型；而Repast4Py父类core.Agent初始化时传入了三个参数，为id、type和rank，因此Repast4Py标识了智能体类型，且传入了用于多线程的rank标识，但是并未传入model模型，因此在Repast4Py中调用模型方法和变量时，将model类即class Model的实例化对象转换成了全局变量，方便调用。

为了保持代码的清晰，Mesa和Repast4Py均只在step(self)中调用定义的智能体行为函数，而不书写具体行为代码。

class Patch(core.Agent):  
   
 TYPE=0  
   
 def \_\_init\_\_(self, a\_id: int, rank: int, attraction: float):  
 super().\_\_init\_\_(id=a\_id, type=Patch.TYPE, rank=rank)  
 self.attraction=attraction   
   
class Seeker(core.Agent):  
 """The Seeker Agent  
  
 Args:  
 a\_id: a integer that uniquely identifies this Seeker on its starting rank  
 rank: the starting MPI rank of this Seeker.  
 """  
  
 TYPE=1  
  
 def \_\_init\_\_(self, a\_id: int, rank: int):  
 super().\_\_init\_\_(id=a\_id, type=Seeker.TYPE, rank=rank)  
 self.infected=False  
 self.foundit=0  
 self.patience\_counter=args.seeker\_patience   
   
 def seeking(self):  
 '''智能体seeker每步（tick）行为'''  
 self.grid=model.grid  
 self.pt=self.grid.get\_location(self)   
 self.objs=self.grid.get\_agents(self.pt)  
  
 if self.want\_to\_build():  
 self.foundit=1  
 self.generate\_agent\_house()  
 else:  
 if self.patience\_counter>0:  
 neighbors\_filtered=self.turn\_toward\_attraction()  
 if neighbors\_filtered:  
 neighbors\_choice=choice(neighbors\_filtered)  
 next\_pt=dpt(neighbors\_choice[0], neighbors\_choice[1])  
 self.grid.move(self, next\_pt)  
  
 for obj in self.grid.get\_agents(next\_pt):  
 if obj.uid[1]==Patch.TYPE:  
 obj.attraction+=.01   
   
 def want\_to\_build(self):   
 '''同NetLogo版的 to-report want-to-build? 部分'''  
 patch=[obj for obj in self.objs if obj.uid[1]==Patch.TYPE][0]   
 selection\_lst=[patch.attraction >= model.build\_threshold, self.patience\_counter==0]  
   
 return choice(selection\_lst)  
   
 def generate\_agent\_house(self):  
 '''在智能体seeker当前位置，生成一个智能体房屋（house）'''  
 house=House(model.house\_id,model.rank)  
 model.context.add(house)  
 model.move(house, self.pt.x, self.pt.y)   
   
 model.house\_id+=1  
   
 def turn\_toward\_attraction(self):   
 '''根据当前位置单元和邻里8个单元的吸引力值，选择大于当前位置吸引力值的邻里单元为智能体seeker的移至位置单元'''  
 local\_attraction=[obj.attraction for obj in self.objs if obj.uid[1]==Patch.TYPE][0]  
  
 nghs=model.ngh\_finder.find(self.pt.x,self.pt.y)  
 at=dpt(0,0)  
 attraction\_neighbors=[]  
 for ngh in nghs:  
 at.\_reset\_from\_array(ngh)  
 for obj in self.grid.get\_agents(at):  
 if obj.uid[1]==Patch.TYPE:  
 attraction\_neighbors.append([ngh,obj.attraction])   
   
 neighbors\_filter=[i for i in attraction\_neighbors if i[1]>local\_attraction]   
 neighbors\_selection=[i[0] for i in neighbors\_filter]  
   
 return neighbors\_selection   
   
 def step(self):  
 self.seeking()   
  
class House(core.Agent):  
   
 TYPE=2  
   
 def \_\_init\_\_(self, a\_id: int, rank: int):  
 super().\_\_init\_\_(id=a\_id, type=House.TYPE, rank=rank)  
 self.stay\_counter=args.wait\_between\_seeking  
 self.dropped=0  
   
 def house\_state\_update(self):   
 '''更新智能体—房屋所在位置的吸引力值'''  
 grid=model.grid  
 pt=grid.get\_location(self)   
 patch=[obj for obj in grid.get\_agents(pt) if obj.uid[1]==Patch.TYPE][0]  
  
 if patch.attraction <= args.max\_attraction\*2:  
 patch.attraction+=.05  
 else:  
 patch.attraction=0  
  
 self.stay\_counter-=1  
 if self.stay\_counter <= 0:  
 self.dropped=1   
   
 def step(self):  
 self.house\_state\_update()

* 模型初始化

Repast4Py和Mesa模型初始化时，均需要定义网格（grid）、调度器（scheduler）、初始化智能体（agent），及可选项定义日志（logging），只是二者的方法名和方法的输入参数、调用方式会存在差异。因为Repast4Py没有可视化模块，因此仅初始化了吸引力地图值，而没有转换为颜色值，但增加了日志功能用于数据的记录。在数据记录和保持至本地的方法中，一种是使用了Repast4Py提供的logging.create\_loggers方法，记录智能体house每一步总数量和seeker的总数量，形式如下：

tick,house\_total,seeker\_total  
2,102,200  
3,196,200  
4,287,200  
5,305,200  
6,412,200  
7,516,200  
8,581,200  
9,676,200  
10,718,200

另一种方式是，使用np.save方法，执行一步（tick）就将一步所有智能体house的位置（坐标）转换为数组形式存储至本地磁盘文件下。不管是使用哪一种方式，在模拟运行存入数据时，文件都是打开的，因此需要定义at\_end()函数，传入到self.runner.schedule\_end\_event(self.at\_end)调度器中，当模拟解释时，关闭所有打开的文件。

本次实验仅有一个线程，但保留了多线程操作的代码，执行多线程的操作同样适用于单线程。

class Model:  
 '''初始化模型'''  
 def \_\_init\_\_(self, comm, params):   
 self.build\_threshold=math.floor(args.max\_attraction/2)  
   
 self.comm=comm  
 self.context=ctx.SharedContext(comm)  
 self.rank=self.comm.Get\_rank() # Get the rank that is executing this code, the current process rank  
   
 self.runner=schedule.init\_schedule\_runner(comm)  
 self.runner.schedule\_repeating\_event(1, 1, self.step)   
 self.runner.schedule\_stop(params['stop.at'])  
 self.runner.schedule\_end\_event(self.at\_end)   
   
 # BoundingBox(xmin=0, xextent=34 ymin=0, yextent=34, zmin=0, zextent=0)  
 box=space.BoundingBox(0, params['world.width'], 0, params['world.height'], 0, 0)  
 self.grid=space.SharedGrid('grid',   
 bounds=box,   
 borders=BorderType.Sticky,   
 occupancy=OccupancyType.Multiple,  
 buffer\_size=2,   
 comm=comm)   
 self.context.add\_projection(self.grid)   
 self.space=space.SharedCSpace('space',   
 bounds=box,   
 borders=BorderType.Sticky,   
 occupancy=OccupancyType.Multiple,  
 buffer\_size=2,   
 comm=comm,   
 tree\_threshold=100)   
 self.context.add\_projection(self.space)   
 self.ngh\_finder=GridNghFinder(0, 0, box.xextent, box.yextent)  
   
 self.info=Info()  
 loggers=logging.create\_loggers(self.info, op=MPI.SUM, rank=self.rank)  
 self.data\_set=logging.ReducingDataSet(loggers, self.comm, params['info\_file'])  
   
 self.house\_pts\_set=open(params['house\_pts\_file'],'wb')  
   
 world\_size=comm.Get\_size() # Get the number of process ranks over which the simulation is distributed  
 total\_seeker\_count=params['seeker.count']  
 pp\_seeker\_count=int(total\_seeker\_count / world\_size)  
  
 if self.rank < total\_seeker\_count % world\_size:  
 pp\_seeker\_count += 1   
   
 attractiveness=self.diffuse\_attractiveness(params['world.width'],params['world.height'],args.max\_attraction,args.smoothness,args.share)   
 with open(params['attraction\_file'],'wb') as f:  
 np.save(f,attractiveness)  
   
 local\_bounds=self.space.get\_local\_bounds()   
   
 x\_lst=range(0,local\_bounds.xextent,1)  
 y\_lst=range(0,local\_bounds.yextent,1)  
 xy\_meshgrid=np.stack(np.meshgrid(x\_lst,y\_lst),axis=-1)  
 for i,xy in enumerate(xy\_meshgrid.reshape(-1,2)):  
 patch=Patch(i,self.rank,attractiveness[xy[0],xy[1]])  
 self.context.add(patch)  
 self.move(patch, xy[0],xy[1])  
   
 for i in range(pp\_seeker\_count):  
 seeker=Seeker(i, self.rank)  
 self.context.add(seeker)  
 x=local\_bounds.xextent//2  
 y=local\_bounds.yextent//2  
 self.move(seeker, x, y)  
   
 self.house\_id=1   
  
 def step(self):  
 '''执行智能体的step方法'''  
 tick=self.runner.schedule.tick   
 self.log\_info(tick)  
 for seeker in self.context.agents(Seeker.TYPE):  
 seeker.step()   
   
 for house in self.context.agents(House.TYPE):  
 house.step()   
   
 def log\_info(self,tick):  
 '''配置日志文件，及模拟运行时打印信息行'''  
 try:  
 num\_agents=self.context.size([Seeker.TYPE, House.TYPE])   
 self.info.house\_total=num\_agents[House.TYPE]  
 self.info.seeker\_total=num\_agents[Seeker.TYPE]  
 pt\_house\_lst=[]  
 for house in self.context.agents(House.TYPE):  
 pt\_house=self.grid.get\_location(house)  
 pt\_house\_lst.append([pt\_house.x,pt\_house.y])   
 self.data\_set.log(tick)   
   
 pt\_house\_lst\_unique=list(set(tuple(sub) for sub in pt\_house\_lst))  
 pt\_house\_array=np.array(pt\_house\_lst\_unique)  
 np.save(self.house\_pts\_set,pt\_house\_array)  
   
 # Do the cross-rank reduction manually and print the result  
 if tick % 1 == 0:   
 house\_total=np.zeros(1, dtype='int64')  
 seeker\_total=np.zeros(1, dtype='int64')   
 self.comm.Reduce(np.array([self.info.house\_total], dtype='int64'), house\_total, op=MPI.SUM, root=0)  
 self.comm.Reduce(np.array([self.info.seeker\_total], dtype='int64'), seeker\_total, op=MPI.SUM, root=0)  
 if (self.rank == 0):  
 print("Tick: {}, house\_total: {}, seeker\_total: {}".format(tick, house\_total[0], seeker\_total[0]),flush=True)   
 except:  
 print('No agent House~')   
  
 def at\_end(self):  
 '''模拟结束时，关闭日志文件'''  
 self.data\_set.close()  
 self.house\_pts\_set.close()  
  
 def move(self, agent, x, y):  
 '''定义智能体在离散网格和连续网格下移动的方法。注意，本次未使用连续网格'''  
 self.space.move(agent, cpt(x, y))  
 self.grid.move(agent, dpt(int(math.floor(x)), int(math.floor(y))))  
  
 def diffuse\_attractiveness(self,width,height,max\_attraction,smoothness,share):   
 '''用二维卷积的方式实现NetLogo中的diffuse方法；用循环的方式实现NetLogo中的repeat方法，完成吸引力地图二维矩阵值的随机生成'''  
 attraction=np.random.randint(0,max\_attraction,size=(width,height),dtype=np.uint32)  
 share\_percent=share/8  
 kernel=np.array([[share\_percent,share\_percent,share\_percent],  
 [share\_percent, 1-share,share\_percent],  
 [share\_percent,share\_percent,share\_percent]])   
 diffusion\_attraction=np.copy(attraction)  
 i=0   
 while i<smoothness:  
 diffusion\_attraction=convolve2d(diffusion\_attraction,kernel,mode='same',boundary='wrap')  
 i+=1  
  
 return diffusion\_attraction   
  
 def generate\_attraction\_colors(self,array,cmap=plt.cm.Greens,reverse=False):  
 '''根据矩阵值对应生成颜色值'''  
 norm=colors.Normalize() # vmin，vmax配置为默认值  
 color=[colors.to\_hex(c) for c in cmap(norm(array.reshape(-1)))]  
 if reverse:  
 color.reverse()  
 return np.reshape(color,array.shape)   
   
 def run(self):  
 self.runner.execute()

* 模型实例化及模拟运行

注意实例化的模型model，定义为全局变量，方便调用模型的方法或变量。

model=None  
  
def run(params: Dict):  
 """Creates and runs the urban sprawl Model.  
  
 Args:  
 params: the model input parameters  
 """  
 global model  
 model=Model(MPI.COMM\_WORLD, params)  
 model.run()

* 模拟运行

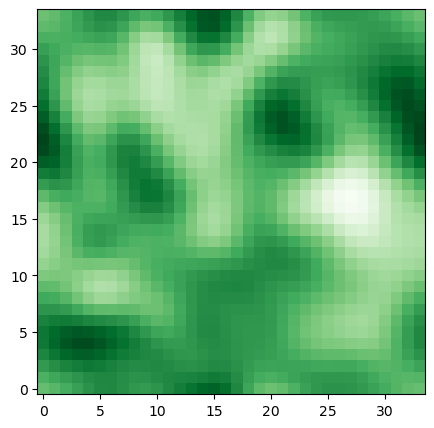
为了后续模拟过程的可视化再现，保存了吸引力地图数据attraction\_file和智能体house每一步的位置数据house\_pts\_file。将在WSL下的数据文件复制到Widows系统下的其它保存位置，可以在Linux终端下执行explorer.exe .打开Linux文件位置进行复制。

if \_\_name\_\_ == "\_\_main\_\_":  
 params={  
 'random.seed': 42,  
 'stop.at': 5.0,   
 'seeker.count': 20,  
 'world.width': 34,  
 'world.height': 34,  
 'run.number': 1,  
 'info\_file': 'output/agent\_house.csv',  
 'house\_pts\_file': 'output/house\_pts.npy',  
 'attraction\_file': 'output/attraction.npy',  
 }  
 run(params)

* 智能体模拟结果可视化

这里仅是简单的可视化运行结果数据，使用matplotlib库提供的animation方法动态显示模拟过程。首先打印了吸引力地图，观察数值的分布情况。

import matplotlib.pyplot as plt  
  
params={  
 'world.width': 34,  
 'world.height': 34,  
 'house\_pts\_file': 'data/ABM/house\_pts.npy',  
 'n': 1000,  
 'figsize': (5,5),  
 'attraction\_file': 'data/ABM/attraction.npy',  
 }  
  
x\_lst=range(0,params['world.width'],1)  
y\_lst=range(0,params['world.height'],1)  
x\_array,y\_array=np.meshgrid(x\_lst,y\_lst)   
  
with open(params['attraction\_file'], 'rb') as f:   
 attraction=np.load(f)   
fig, ax = plt.subplots(figsize=params['figsize'])   
  
cmap=plt.cm.get\_cmap('Greens')  
ax.pcolormesh(x\_array,y\_array,attraction,cmap=cmap.reversed())   
plt.show()



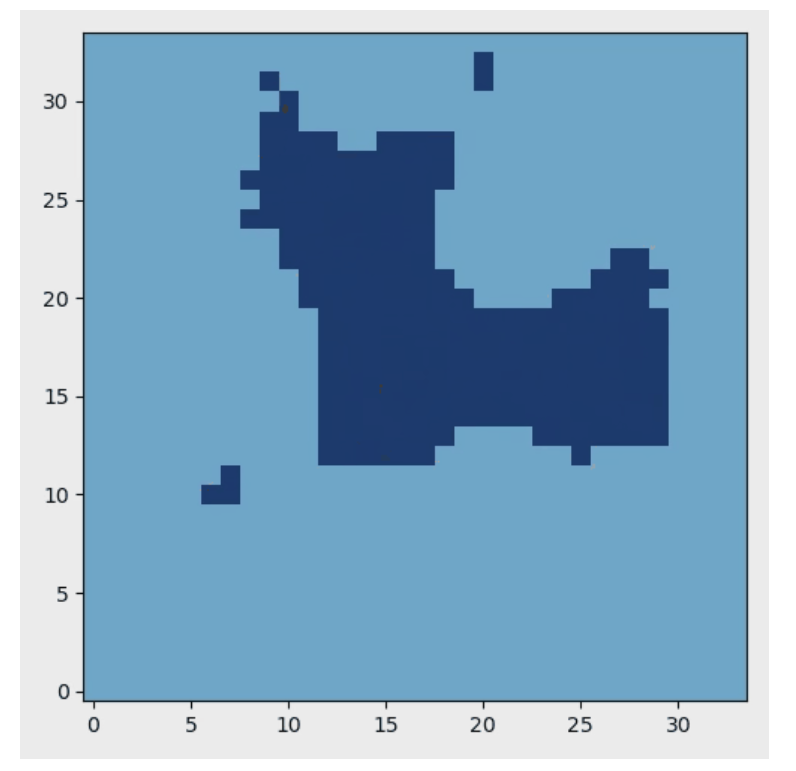
np.save方法分批存储的方式，在np.load读取数据时，需要多次按存储批次依次读取，因此使用了循环的方式读取。同时，计算了存入（读取）批次数，用于动画配置参数。

def get\_grid\_agent(params):   
 '''读取np.save()方法分批存入的智能体house位置数据，对应到二维网格位置，并转换为列表形式'''  
 x\_lst=range(0,params['world.width'],1)  
 y\_lst=range(0,params['world.height'],1)  
 x\_array,y\_array=np.meshgrid(x\_lst,y\_lst)   
 z\_array\_lst=[]  
 with open(params['house\_pts\_file'], 'rb') as f:   
 i=0  
 while i<params['n']:  
 try:  
 z\_array=np.zeros(x\_array.shape,dtype=np.int64)   
 z\_mask=np.load(f,allow\_pickle=True)  
 for m in z\_mask:   
 z\_array[m[0],m[1]]=1   
 z\_array\_lst.append(z\_array)  
 i+=1  
 except:  
 break  
 return z\_array\_lst,i  
z\_array\_lst,tick\_num=get\_grid\_agent(params)  
print(tick\_num)

599

再现Sprawl Effect的ABM模拟过程。

import numpy as np  
from matplotlib import pyplot as plt, animation  
from matplotlib import colors  
  
z\_array=np.zeros(x\_array.shape,dtype=np.int64)   
  
fig, ax=plt.subplots(figsize=params['figsize'])   
pcolormesh=ax.pcolormesh(x\_array,y\_array,z\_array,vmin=-1, vmax=1, cmap='Blues')   
  
def animate(i):  
 pcolormesh.set\_array(z\_array\_lst[i].flatten())  
anim=animation.FuncAnimation(fig, animate,frames=tick\_num,interval=100)   
HTML(anim.to\_html5\_video())



注释（Notes）：

① 圣菲研究所( Santa Fe Institute)，（<https://www.santafe.edu>）。

② Swarm，（<http://www.swarm.org/wiki/Swarm_main_page>）。

③ MASON，（<https://cs.gmu.edu/~eclab/projects/mason>）。

④ Repast，（<https://repast.github.io>）。

⑤ StarLogo，（<https://education.mit.edu/project/starlogo-tng>）。

⑥ StartLogo NOVA，（<https://www.slnova.org>）。

⑦ NetLogo，（<http://ccl.northwestern.edu/netlogo/index.shtml>）。

⑧ AgentCubes，（<https://agentsheets.com>）。

⑨ AnyLogic，（<https://www.anylogic.com>）。

⑩ Mesa，（<https://mesa.readthedocs.io/en/stable/index.html>）。

⑪ AgentPy，（<https://agentpy.readthedocs.io/en/latest>）。

⑫ PythonABM，（<https://pythonabm.readthedocs.io/en/latest/index.html>）。

⑬ Repast for Python （Repast4Py），（<https://repast.github.io/repast4py.site/guide/user_guide.html#_footnotedef_3>）。

⑭ Numba，（<https://numba.readthedocs.io/en/stable/user/5minguide.html>）。

⑮ MPI（Message Passing Interface），（<https://en.wikipedia.org/wiki/Open_MPI>）。

⑯ mpi4py，（<https://mpi4py.readthedocs.io/en/stable>）。

⑰ Install Linux on Windows with WSL，（<https://learn.microsoft.com/en-us/windows/wsl/install>）。

参考文献（References）:

[1] Heppenstall, A. J., Crooks, A. T., See, L. M., & Batty, M. (Eds.). 2012. Agent-Based Models of Geographical Systems. Springer Netherlands. https://doi.org/10.1007/978-90-481-8927-4.（Perspectives on Agent-Based Models and Geographical Systems）

[2] G.尼科里斯，I.普利高津著.罗久里,陈奎宁译. 探索复杂性[M].四川：四川教育出版社.2010.4.

[3] Felsen, M. and Wilensky, U. (2007). NetLogo Urban Suite - Sprawl Effect model. http://ccl.northwestern.edu/netlogo/models/UrbanSuite-SprawlEffect. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.