Documentation.

GEOG5995M Planning for Drunks Project.

Sources.

All of the code produced for this project is my own. I have learnt skills through lectures, online courses and a number of website providing advice and suggestions regarding Python.

The data, which was read in to provide the environment, is a 300 by 300 raster file which was provided by Dr Andy Evans through the online resources for the GEOG5995M module.

The idea for this project was not mine in any way and the algorithm for the code was provided by Dr Andy Evans.

My code is also available from my Github repository from the following web address: <https://github.com/annabelelizabethwhipp>

Intention of the software.

This software intended to build a model representing drunk people leaving a pub and returning to their own home.

Each drunk has a number, 0 to 24 and has to return to their home, which is number 10 to 250. A drunk leaves the pub and moves around the environment randomly until they reach their own home. When they are home, the next agent can leave the pub and return home. This process continues until all of the drunks have left the pub and have returned to their home. When a drunk has reached their home, they remain there and do not leave again. The model runs until all the agents have returned home.

The model can be altered to run for a specific number of iterations in order to see how many drunks have returned home.

Once all the drunks have returned home and the model stops running a density map is produced. This records how many drunks have passed through a point. Each time a drunk moves through a point a value of 1 is added to the environment. The same drunk may pass through a point more than once.

Issues during development.

One of the first issues I encountered was how to make the agents know that they were not starting from their home location. I fixed this issue by using a Boolean expression. The agents are told in the agent framework that initially they are not at their home location using the following code when defining the class Agent:

self.is\_home = False

Another issue I had with this project was knowing whether agents had returned home. This problem was comprised of two elements. I initially decided to get the agents to state whether they had returned home or not using the following code:

for i in range(len(agents)):

print('agent {0}: {1}'.format(i, agents[i].is\_home))

This code tells us that for every time the model is run we want the model to print the agent number and whether they are home or not. For example, if agent 0 is home it will print:

agent 0: true

However, when the environment was printed it appeared that some agents were home, but were not informing the user they had reached their home. In order to attempt to resolve the issue I changed the code so that drunks that had returned home changed their colour to white. This then allowed me to conclude that some agents appeared to be home as they had reached a square representing a house, however they had actually just reached the home of another drunk. This issue arose while the model was set to run for 1000 iterations. This would not be an issue while the model is set to run until all the agents have returned home. I changed the colour to white using the following if else statement:

if agents[i].is\_home == True:

matplotlib.pyplot.scatter(agents[i].x,agents[i].y, c='white')

else:

matplotlib.pyplot.scatter(agents[i].x,agents[i].y)

matplotlib.pyplot.show()

Potential changes and developments.

The colour of the pub could be changed so that it can easily be identified when the environment is printed. It does not really add to the utility of the model, but may make the model more visually appealing. This could be done by changing the number of the pub, however it would need to be above one and not interfere with any of the numbers allocated to the houses. The number of the pub would need to be changed in both drunkmodel.py and drunk.plan.txt

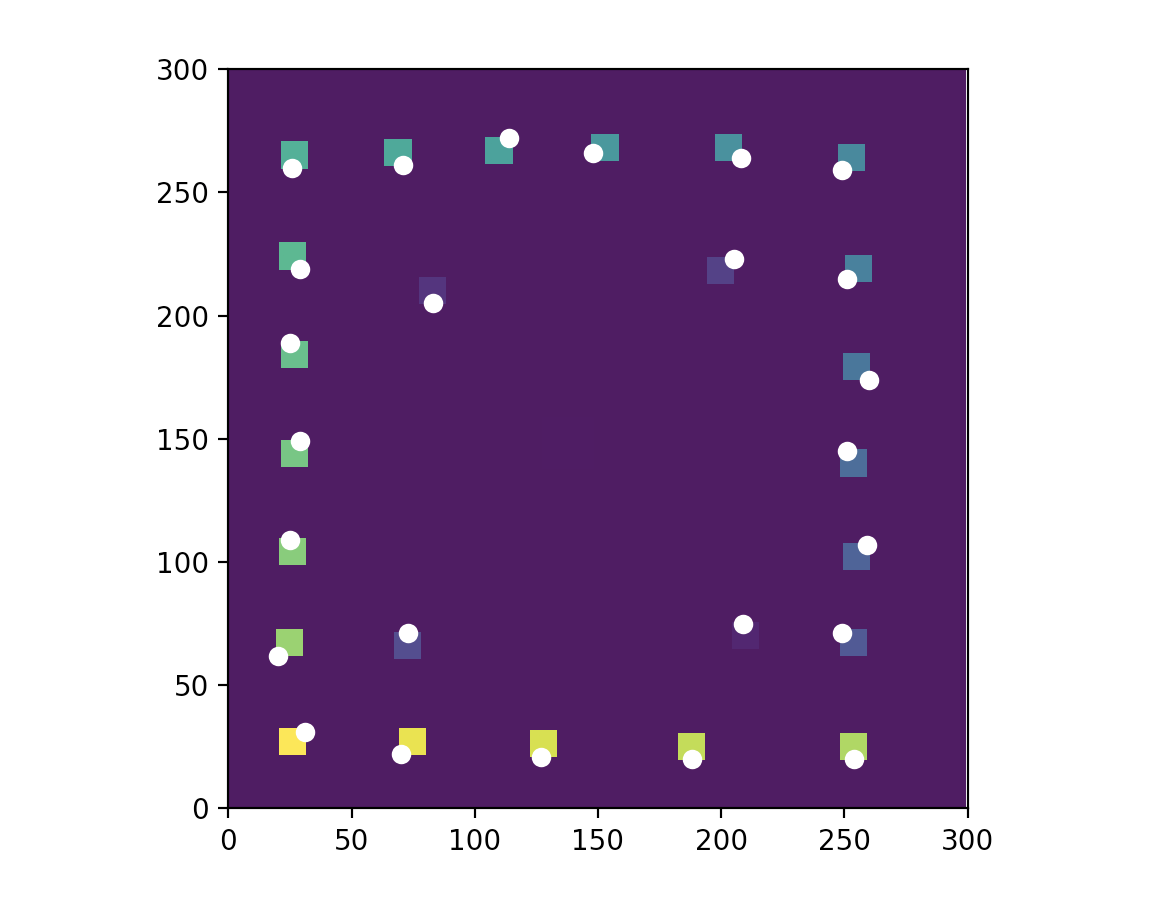
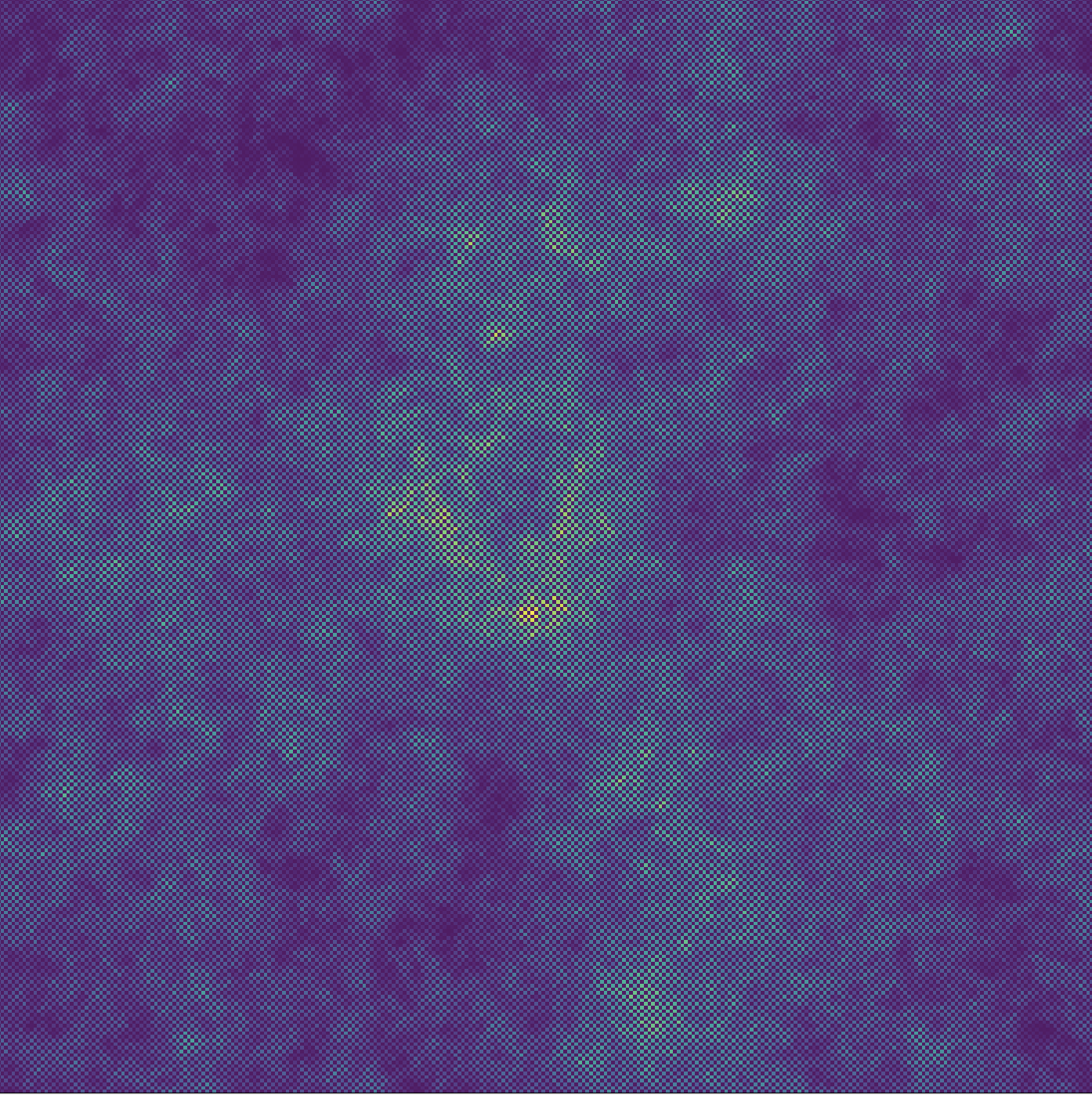
Anyone running the model has the option to run a specific number of iterations or to run the model until the drunks all reach their home.

The code could be built upon to stop the drunk retracing their steps. There are issues with preventing a drunk from following or crossing a path they have previously taken or from going to a particular x,y coordinate, as they made need to do so to reach their home. If the drunks are prevented from doing this, it may result in some agents becoming stuck in one of the four corners of the environment or never reaching their home. If the model is set to run until all the drunks are home, as is the case in this version of the code, the model would not finish running.

For this project I chose not to implement a torus. A torus would allow agents to exit on one side on the environment, then come back in on the opposite side. I chose not to utilise this function as it would be more realistic to have the drunks, reach the limits of the environment and have to remain within them.

Examples.

*Figure One* is an example of the environment when all agents have returned home. *Figure Two* is an example of a density map produced when the model was run until all the agents returned home.

**

*Figure One*

*Figure Two*

Code for drunkmodel.py.

Created on Tue Nov 14 13:31:06 2017

@author: annabelwhipp

Assessment 2: Planning for drunks

Completed as part of Programming for Social Science: Core Skills

GEOG5995M

"""

# Start of imports

import matplotlib

import matplotlib.pyplot

import DrunkAgentFramework

import csv

import matplotlib.animation

# Start of creating lists

environment=[]

mapenvironment=[]

agents=[]

num\_of\_agents= 25

num\_of\_iterations=10000

# Reading in environment data. The environment is a single 300 # by 300 raster

# file which represents 25 houses and 1 pub. The pub is # represented by 1s and

# the houses are numbered 10-250 (increasing by an increment of # 10 each time).

# Empty spaces are represented by 0s.

f=open ("drunk.plan.txt")

reader=csv.reader (f, quoting = csv.QUOTE\_NONNUMERIC)

# Defining variable figure

fig= matplotlib.pyplot.figure(figsize=(7,7))

# Defining variable axes

ax= fig.add\_axes([0,0,1,1])

# Environment file csv into rows

for row in reader:

rowlist = []

for item in row:

rowlist.append(item)

#print(len(rowlist))

environment.append(rowlist)

print(type(environment), len(environment))

#print (environment)

# The next section of code is for setting up mapenvironment. The mapenvironment

# is creating an environment to map the density plot. Each time an agent passes

# through a point the value of 1 is added to the environment. This allows a

# density map to be produced. A new environment is created to plot the density

# as the 'houses' and 'pub' have already been assigned values, for example 10,

# 20, 30. This would interupt the procedure of adding 1 to an empty environment

# Creating the size of the environment

height=len(environment)

width=len(environment[0])

# Adding zeros to the mapenvironment

for i in range(height):

rowlist = []

for j in range(width):

rowlist.append(0)

mapenvironment.append(rowlist)

# checking the mapenvironment has been made

print(height,width)

# Looking through the environment to find the location of the pub

for y, row in enumerate (environment):

for x, value in enumerate (row):

if value==1:

xpub=x

ypub=y

print('got pub location')

# Make i agents, append coordinates to list and then plot initial positions

for i in range(num\_of\_agents):

number = (i+1)\*10

agents.append(DrunkAgentFramework.Agent(environment, agents, number, xpub,

ypub))

print('made agents')

# Take the ith agent and perform the following

for i in range (num\_of\_agents):

# When the agent hasn't reached home, move and add 1 to the # mapenvironment

# with each iteration

while agents[i].is\_home==False:

agents[i].move()

mapenvironment[agents[i].y][agents[i].x]+=1

# Stop moving if the agent gets to their home

if agents[i].house\_number==agents[i].environment[agents[i].y][agents[i].x]:

agents[i].is\_home=True

# Print the mapenvironment. This produces a density map.

matplotlib.pyplot.imshow(mapenvironment)

matplotlib.pyplot.show()

#for i in range (num\_of\_agents):

# matplotlib.pyplot.scatter(agents[i].x,agents[i].y)

# matplotlib.pyplot.imshow(environment)

# This tells the user which agents have reached their home. A true or false is

# provided for each agent. Each agent has a number. If it says true the agent

# has returned to their home. If false the agent has not reached their home.

for i in range(len(agents)):

print('agent {0}: {1}'.format(i, agents[i].is\_home))

# Start of displaying the agents statically on the environment

matplotlib.pyplot.xlim(0,300)

matplotlib.pyplot.ylim(0,300)

matplotlib.pyplot.imshow(environment)

for i in range (num\_of\_agents):

# If the agent arrives at their home plot them as white,

# if not they are plotted with their original colour

if agents[i].is\_home == True:

matplotlib.pyplot.scatter(agents[i].x,agents[i].y, c='white')

else:

matplotlib.pyplot.scatter(agents[i].x,agents[i].y)

matplotlib.pyplot.show()

# Writing the new mapenvironment(density map)to a csv file

f2= open ('mapenvironmentout.csv', 'w', newline= '')

writer = csv.writer (f2, delimiter = ' ')

for row in mapenvironment:

writer.writerow(row)

f2.close

Agent framework (DrunkAgentFramework.py).

# Imports

import random

# Define a class of agent in an environment

# The agents are aware of the environment, agents, the numbers allocated to

# the houses, the X and Y coordinates of the pubs and whether they have reached

# their home or not. The agents begin knowing that they are not at home.

class Agent ():

def \_\_init\_\_(self, environment,agents, number, xpub, ypub):

self.environment = environment

self.agents = agents

self.house\_number = number

self.x = xpub

self.y = ypub

self.is\_home = False

# Moving agents in the environment.

def move (self):

if random.random() < 0.5:

newx=self.x+1

if newx<len(self.environment) and newx>0:

self.x = newx

else:

newx=self.x-1

if newx<len(self.environment) and newx>0:

self.x = newx

if random.random() < 0.5:

newy=self.y+1

if newy<len(self.environment) and newy>0:

self.y = newy

else:

newy=self.y-1

if newy<len(self.environment) and newy>0:

self.y = newy