Programming for Geographical Information Analysis: Core Skills Assessment 2

Word count:

1510

Student number:

200924183

Contents

Introduction

Agent-based modelling has seen increasing use in contemporary society, given its potential to effectively model real-world scenarios depending on its inputs. Agent-based modelling and simulation or (ABMS) for short focuses on the autonomous generation of “agents” within a theoretical space (Macal and North, 2010).

It has seen use in a variety of contexts such as modelling of infectious diseases, consumer behaviour and the stock market (Macal and North, 2010). This application will look at the conundrum of getting 25 drunk agents/people to traverse a 300x300m environment only stopping when they reach their home.

A typical structure of an agent-based model is:

- Agents and their associated attributes and behaviours
- The method of interaction between agents
- The agent environment and how agents traverse and interact with said environment (Macal and North, 2010).

Code used was primarily adapted from teaching material available at the University of Leeds and utilised in assessment 1, although this has been improved and adapted where possible using resources such as W3schools (W3schools, 2021) and Stack overflow (Stack Overflow, 2021).

Key requirements

The completion of following requirements was necessary in order to create the desired application outlined in the introduction.

1. Read in 300x300 environment grid which contained the location of a pub and 25 homes.
2. Display pub on a map
3. Create agents which represented drunks.
4. Crete identification numbers which matched the drunks’ homes i.e., 10-250.
5. Create code to stop agent after the desired outcome is reached.
6. Create code which will track the movement of agents and produce a density map.
7. Display the drunks on a map to illustrate the outcome of the code.
8. Write the density of the new environment to a text file following the traversal of the drunks on their way home.

Extras

To enhance the model further the decision was made to add a GUI found under the file learning.py, this would simply show the process of agents moving around the environment and leaving a value of 1 as they travelled to map the density. This code was used heavily in the testing process as it offered a visualisation of what the code was doing and eventually led to a revision of the original prototype.

The GUI would do the following:

1. Run the code at the press of a button.
2. Show the animation in real time at lower iterations.
3. Display an exit button to end the GUI.

Design

In terms of design the application was relatively simplistic as the code would rely on outputs created by matplotlib in order to present the results. After the running of the final_model.py code a series of plots would appear (plot pane for Spyder 5.0.3) depicting the layout of pubs, pubs with the drunks returning home and a density map.

Another two sets of code, named proto.py and learning.py were created to help compartmentalise the overarching themes of the model. Learning.py focuses on showing the user how the drunks move as well as the effect this has on the density. This was added as a personal decision as it helps to conceptualise how the code is causing a change of agent/drunk position and the effect this has on the environment.

Proto.py was the initial rendition of the drunk's model which was left in to help give some idea of the development process which has occurred. Drunkframework.py contained the Drunk class which contained an array of "self-parameters", including "move" which naturally moved the agents around the grid.

Development

The first goal in the development of the drunk's application was to read in the data, so that there was an understanding for the context of the project. The squares observable in figure 1 depict 25 homes, all with a unique set of numbers which represent the name of the house. The house numbers ranged from 10-250 perfectly matching the 25 drunks.

Creating the lists

The drunks would be created in the form of an empty list. The base code for the formation of the drunks can be viewed in figure 1. Along with some of the other lists created in this investigation, namely drunks, density, environment, and row list. Each would play a key role in holding both the input and in the case of 'density' the output data.

```
17  environment = []
18  #drunks adapted from agents from GUI's practical replacing "agents"
19  drunks = []
20  #density is an empty list which will track agent movement independent o
21  density= []
22  #specifies number of drunks/agents
23  num_of_drunks = 25
24  #outlines the number of iterations the line 64-78 code will undergo
25  num_of_iterations = 100
```

Figure 1. Lists

Reading data

Code below outlines the process of both reading data and appending the data into rows. This will prevent errors such as "list index out of range" from occurring. The process of appending was also done for the density list so that a density map could be exported as a file following the completion of this code.

```
34    f = open('drunk.txt', newline='')
35    #Note that the correct directory must be navigated to in the terminal else the full file p
36    reader = csv.reader(f, quoting=csv.QUOTE_NONNUMERIC)
37
38    #Used for testing purposes to ascertain the lay of the environment
39    #matplotlib.pyplot.xlim(0, 300)
40    #matplotlib.pyplot.ylim(0, 300)
41    #matplotlib.pyplot.imshow(environment)
42
43    for row in reader:
44        rowlist =[]
45        for value in row:
46            rowlist.append(value)
47        environment.append(rowlist)
48    f.close()
49
50    #Code on lines 46-50 appends the density list output to a 300x300 grid, this code is needd
51    #to prevent the error "IndexError: list index out of range"
52    for i in range(300):
53        rowlist = []
54        for j in range(300):
55            rowlist.append(0)
56        density.append(rowlist)
57    #matplotlib.pyplot.imshow(environment) run this in isolation to check the environment is
58    #correct
```

Figure 2. Reading data

Assigning identification numbers

As there were 25 different houses it was necessary to assign an identification number so that the drunks could be stopped upon reaching their desired destination. The code seen in figure 3 is a for loop which iterates through the 25 drunks assigning them an identification value of 10-250 by multiplying 1-25 by 10. The print identification was an internal check to ensure the code was running as intended.

```
43    ## Make drunks and assign them with an identification number.
44    for i in range(num_of_drunks):
45        identification = ((1+i)*10)
46        print(identification) #this should print 10-250 giving each of the drunks an identification number, later to be matched up with houses
47        drunks.append(drunkframework.Drunk(environment, drunks, identification))
```

Figure 3. Drunk identification.

Getting drunk's home

Getting the drunks home was done with a combination of a framework which can be viewed in the source code and a for/while loop. In the initial iteration of getting the drunks home the function track was made to add a value of 1 to the environment as the agent traversed it. The general gist of the code presented in figure 4 is to get the drunks to move around until their identification numbers matched their house number. Under the for/while loop in figure 4 is a snippet of code which would write a new density file into the working directory.

```
#This is is supposed to work whereby if the co-ordinates of stilldrunk match their identif
#In the prototype density of the environment changed throughout the iterations, as such th
#often stop in areas which were not their home. The work around this was seperating the pr
#and move through the creation of the density list. Track is left in but commented.
for i in range (num_of_drunks):
    stilldrunk = drunks[i]
    for j in range(num_of_iterations):
         while environment [stilldrunk._y][stilldrunk._x] != stilldrunk.identification:
                density[drunks[i]._y][drunks[i]._x]+=1
                drunks[i].move()
                #drunks[i].track() omitted from the final iteration of the application

#saves density list (see lines 68 to 73)
with open('density.txt', 'w', newline='') as f:
    csvwriter = csv.writer(f, delimiter=',', quoting=csv.QUOTE_NONNUMERIC)
    for row in density:
        csvwriter.writerow(row)

#lines 79 to 90 serve the purpose of display the density and drunks in relation
#to their finishing position within the environment
```

Figure 4. Getting drunks home/writing data

Plotting drunks

The code in figure 5 below simply plotted the density map, drunks/agents, and environment as a by-product of the code which came before it. To this end it could be assured that the drunks had made it home. Figures 6 and 7 depict the graphical outputs expressed through matplotlib.

```
87   #lines 79 to 90 serve the purpose of display the density and drunks in relation
88   #to their finishing position within the environment
89
90   matplotlib.pyplot.xlim(0, 300)
91   matplotlib.pyplot.ylim(0, 300)
92   matplotlib.pyplot.imshow(density)
93
94   matplotlib.pyplot.xlim(0, 300)
95   matplotlib.pyplot.ylim(0, 300)
96   matplotlib.pyplot.show(drunks)
97
98
99   matplotlib.pyplot.xlim(0, 300)
00   matplotlib.pyplot.ylim(0, 300)
01   matplotlib.pyplot.imshow(environment)
02
03   #Code below just prints we're home for each of the 25 agents following a resolution of
04   #the code
05
06   for i in range(num_of_drunks):
07           matplotlib.pyplot.scatter(drunks[i]._x, drunks[i]._y)
08           print("we're home!")
09
```

Figure 5. Code for plots

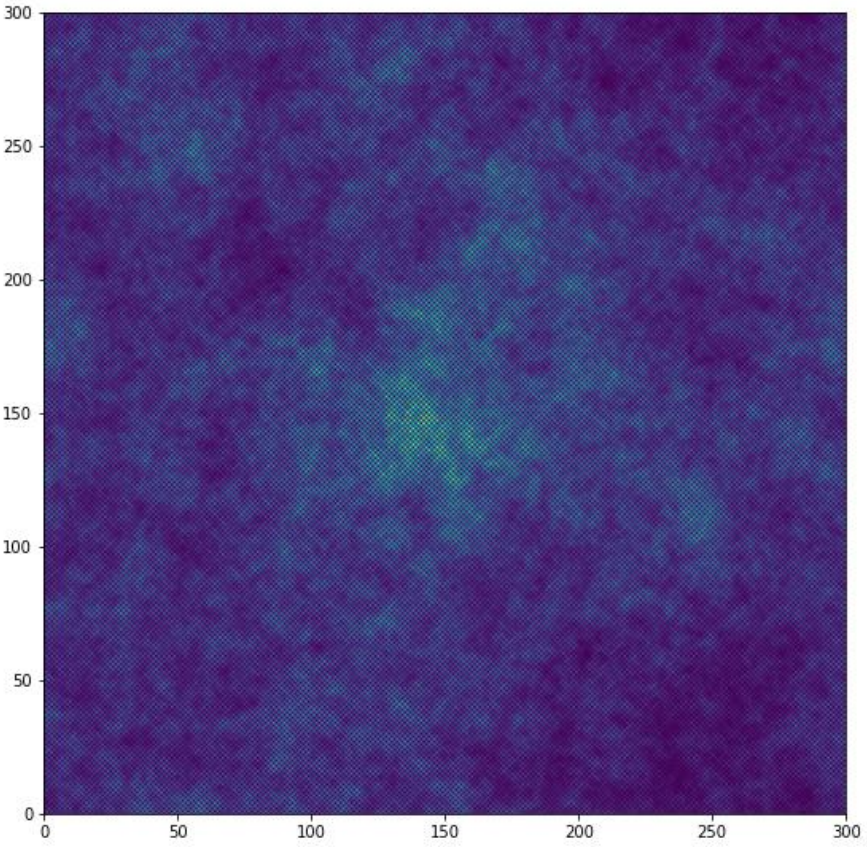

Figure 6. Density map example

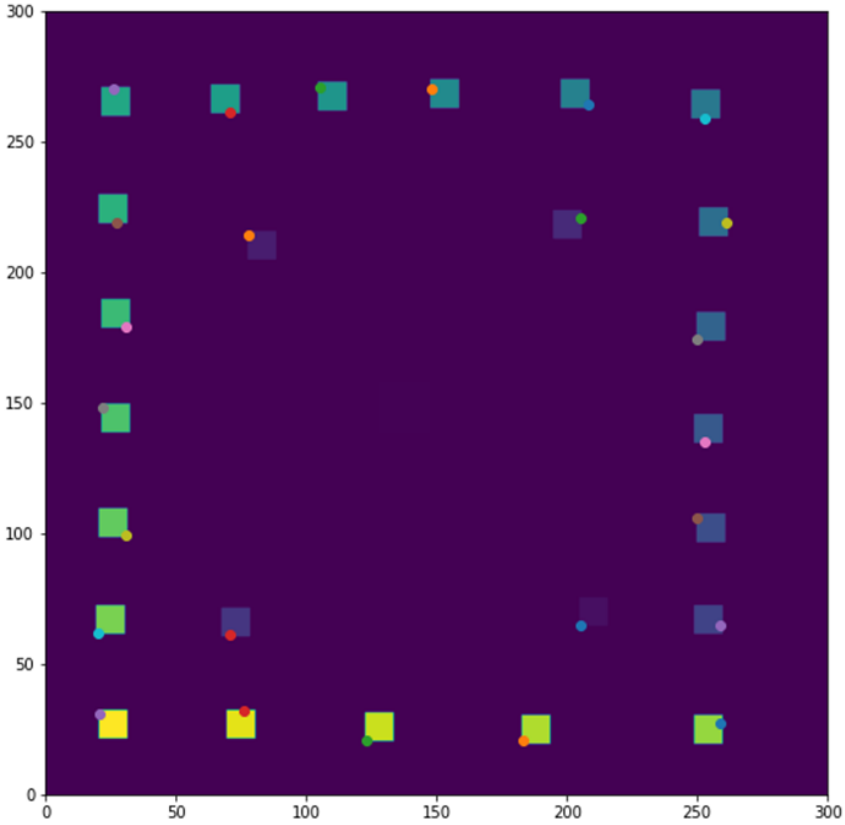

Figure 7. Drunks home example

Tests

1. Print statements after list creations were filled to make sure they contained logical results.
2. Tried various iterations and stopping conditions for loops to assess efficiency/functionality.
3. Printed plots at regular intervals to see if code was working correctly.
4. Checked outputs of written CSV files to make sure the environment was updating for density map.
5. Made use of the animation in learning.py to understand how the code was changing with the addition or removal of certain parameters.
6. Ensured code ran on different software, to date it has been tested on a Jupyter notebook and Spyder version 5.0.4

Problems encountered

As this code was primarily adapted from assessment 1 it was initially straightforward to get the 25 drunks traversing the environment. However, the first major obstacle encountered was related to getting the agents to stop on the correct node. This was related to the rationale in getting the agents to stop. Initially a function named track was created which had the goal of adding a value of 1 to the environment each time a move was moved across the environment. While this worked perfectly in creating the density it led to drunks stopping thinking they had arrived home when in actuality drunk 150 had landed on a node which had been passed 150 times for example. Many different solutions were considered with the primary one being to prevent the drunks from retracing their steps. An animation of the code revealed that the issue was that the tracking of density and movement were nested within the same loop. As a result of this finding an empty list was created in order to hold the values of density in a separate list to that of the environment.

Further problems occurred with trying to optimise the route of the drunks so that they could get home in the most efficient way possible. The main planning here was to manually create a path for each of the 25 agents using something similar to the work of Agent Moves (2021) which focuses on moving agents in Minecraft (Cooper and Chiaradia, 2020). The idea would have been to get agents to have more autonomy choosing the optimised route similar to the algorithm employed in network analyses. Unfortunately, due to time constraints and current knowledge the completion of this was not possible.

Optimisation/future work

1. The creation of autonomous optimised routes could have been implemented to allow agents to arrive to their destination quicker and thus in less iterations.
2. Fully interactive GUI could be implemented to help with user engagement. Currently the outputs provided by the code are either presented in the terminal or the plot pane. For users who are not as well versed in the python programming language an interactive GUI could be used. (Note this may require a powerful PC.)

3. Creation of police agents which send drunks directly home if they interact with each other, possibly incorporate features which let the user know if the drunks arrived at home via the police or by themselves.
4. Addition of obstacles which cannot be traversed such as lakes or busy roads, possibly incorporate some code whereby the drunks will avoid such obstacles if encountered.

References

Cooper, C. and Chiaradia, A., 2020. sDNA: 3-d spatial network analysis for GIS, CAD, Command Line & Python. *SoftwareX*, 12, p.100525.

Macal, C. and North, M., 2010. Tutorial on agent-based modelling and simulation. *Journal of Simulation*, 4(3), pp.151-162.

Microsoft MakeCode. 2021. *Agent Moves*. [online] Available at: <https://minecraft.makecode.com/tutorials/python/agent-moves> [Accessed 21 May 2021].

Stack Overflow. 2021. *Stack Overflow - Where Developers Learn, Share, & Build Careers*. [online] Available at: <https://stackoverflow.com/> [Accessed 20 May 2021].

W3schools.com. 2021. *W3schools*. [online] Available at: <https://www.w3schools.com/python/python_operators.asp> [Accessed 20 May 2021].