**Show me the way to go home... A Drunken Model**

GEOG5990M Programming for GIS Analysis: Core Skills

SI: 201045221

1. Brief Introduction to Documentation:

This document provides context of the project, the model development process, issues during model development, and any improvements that can be made. The project selected is “Show me the way to go home...”, which was developed for the second Assessment of GEOG5990M Programming for GIS Analysis: Core Skills module for the MSc GIS program.

The project was produced using Python 3.7.6 in Anaconda Spyder. The project aims to simulate drunken individuals walk back home after drinking at the town pub.

2.0 Intention of the Model

The model for this project attempts to simulate drunken individuals find their way home after drinking at a local town pub. The density routes of the drunks are recorded therefore, a town planner can evaluate the model and ensure that all the drunks are able to safely get home, as well as how the rate of alcohol consumption can affect the density routes. This model is relatively simple, there are 25 drunks within an 300x300 environment, where the pub lies centrally in the environment and the houses are near-equidistant from each other and are situated mainly near the boundaries of the environment (Figure 1). The 25 drunks move towards their respective houses, stumbling as a result of a function randomly generating decisions; the function decides whether the drunks moves randomly or moves consciously towards their house. When moving in a random unconscious manner, alcohol consumption is randomly generated to reduce the drunken individuals speed. Although the model is simple and not complex enough to be applicable to real-world problems, it forms as a foundation model which can be constantly improved and engineered towards a particular real-world problem.

A screenshot of a computer

Description automatically generated with medium confidenceFigure 1: Shows the environment of the model. Using the green/blue colormap, houses are denoted by different shades/colours. The red stars in the centre of the model depict the drunks.

To summarise, the model is separated into two python scripts, the first is the Drunk\_ABM.py script, which determines how the drunks move. The second script is the Drunk\_Model.py script, which sets the environment, finds the drunks house number and their relative position in the environment, animates the drunks moving, plots the model, plots density routes, and adds GUI functionality. The model overall can be broken down into five relatively simple steps:

1. Pulls in the drunk.plan text file using csv reader to append to the environment. Using the environment, the houses and pub locations are determined.
2. Displays the environment with the houses and pub data onto a scatter graph.
3. Creates a drunk class, where the house number and positions are appended to the drunk class.
4. Drunks move either consciously towards their house using Euclidean distances, or in an unconscious random behaviour, until the drunks reach to their respective homes.
5. Density of the routes taken by each drunk are recorded, plotted onto a scatter graph, and saved into a text file (Figure 2).

Chart, scatter chart

Description automatically generatedFigure 3: Density plot of drunks after the model had finished. Each time a drunk moves over an X and Y coordinate, that value is appended += 1.

3.0 Model Development Process and Issues

To begin the development of the model, the code used for Assessment 1 of GEOG5990M Programming for GIS Analysis: Core Skills was adapted because, the model represents an agent-based modelling scenario. Initially, the environment was produced by reading in the drunk.plan text file via the csv reader, creating 300 rows and 300 columns of integers ranging from 0 to 250. The values in the environment were in the range (1, 10, 20, 30, ... , 250) in which, values of 1 represented the town pub, and the remaining values denoted the houses. The text file was sourced via the second Assessment brief documentation. To append each drunk to a house position, the following code was used (Figure 3).

A screenshot of a computer

Description automatically generated with medium confidenceFigure 3: Python code showing how each drunk is appended to its own house number.

Understanding how to append the locations of the houses to each drunk became more of a challenge. Using knowledge from the first Assessment, looking through the environment rows and columns, if the X and Y values in the environment are the same as the input argument, the X and Y values are appended to the list. However, the drunks need one X and Y value appended to move towards in the move function and this issue became prominent. Therefore, using the print statement, the total rows and columns for each house were determined (11 rows and 11 columns per house). Therefore, to append a single X and Y value for the house to each drunk, a random number in the range from 1 to 11 was calculated for each drunk for each X and Y house value. Thus, each drunk had a random X and Y house value appended.

At the beginning of the model development process, the starting position of the drunks in the pub was problematic. The X and Y values were calculated by converting the drunk.plan.txt file to excel, and then values of 1 which indicates the pub, were recorded. The central point on the left side of the pub were chosen as the starting coordinates for all drunks (X=129, Y=149). Later during the development process, the position of the pubs could have been calculated using a similar function to the house position, where if the X and Y values at the environment were equal to 1, then the X and Y values are appended to a Pub\_Position list, which is then appended to each drunk. However, having performed the first method, the second method was disregarded.

Originally, the model had the drunks move randomly in the environment. However, this method was redundant because, the majority of the drunks never returned home within iterations range of 1000. Consequently, the move function needed to have an element where the drunks are consciously moving in the direction of their house. Using Assessment 1 code, the Euclidean distance between variables using Pythagorean theorem became useful. This distance function was amended to the project, where the X and Y values of the house location calculated in the house position function were used, along with the current X and Y values of the drunks. The concept for the move function was if the drunk moved, would the new distance between the drunk and the house position be greater or less than the previous distance. Implementing this concept took the longest of the model development process. The resultant method was comparing the current distance with the potential distance if the drunk moved right and if the drunk moved down and if the new distance was greater, then the drunk moved up/left, otherwise it moved down/right. During the process, the drunk needed an element of random movement to simulate drunken individuals with cognitive impairment. The move function added a random element, where if the randomly generated number was below 0.5, the drunk moves consciously, otherwise the drunks move in a random direction. An additional issue that arouse was drunks moving past the environment (greater than 300 or less than 0). To combat this issue, IF statements were created, where X and Y values could not increase or decrease past 299 or 0 respectively.

Animating the drunks was relatively simple however, making the drunks stop when reaching home or stopping the model when all drunks found their way home, became challenging. To stop the drunks when reaching home, a True statement was added, where if the house number of the drunk equals their X and Y coordinates, the drunk stops moving. Thus, the drunk stops moving whenever they reach the house (each coordinate within the house had the same house number value) not the X and Y house location values that were appended. To further discriminate drunks that returned home, the colour of the marker depicting the drunk turns black rather than red (Figure 4), using the similar method of when the house number equals the drunks X and Y coordinates.

A picture containing chart

Description automatically generatedFigure 4: All the drunks turn black when returned home.

The final process in building the model was constructing the GUI interface (Figure 5). A simple run and close function were created, where the density of drunks were plotted upon closing the model. Buttons were produced using the command of both of these functions however, using inspiration of examples of agent-based models, more GUI features could be included to allow more user interactivity. One feature is to change the number of iterations, allowing an individual to know how many drunks get home after a set number of iterations. Using various examples on Stack Overflow and [Python Tutorial](https://www.pythontutorial.net/tkinter) , the Tkinter slider widget for the number of iterations was added to the GUI interface, with a range from 1 to 500 iterations, with a function determining the default value at 400. After finalising the iteration slider, other features that was considered were Tkinter slider widgets to control the speed when moving consciously towards home (Sober\_Speed) and controlling the value of alcohol consumption that determines the speed of the drunks during random movements. However, when attempting to produce these, errors stating these variables were not defined (due to existing in the move function in the separate Drunk\_ABM.py file). These features were also left out the final model because, changing the values of these A screenshot of a computer

Description automatically generated with medium confidencevariables does not provide any significant improvement by itself to the model.

Figure 5: GUI interface when running the model.

4.0 Improvements

Within the move function, alcohol consumption could be randomly generated and fixed for the entire iteration run for each drunk, and the rate of alcohol consumption determines the rate of conscious moves rather than the speed of the drunk. For instance, if the alcohol consumption is low, then when random.random() < 0.9, the drunk moves towards their house, and >0.9 the drunk moves randomly however, if the alcohol consumption is high, then when random.random() < 0.4, the drunk moves home, otherwise they move randomly. Therefore, drunks who have a higher alcohol consumption moves more sporadically within the environment.

Adding barriers to the environment allows for more complex scenarios for the routes taken by the drunks. For instance, adding in roads along with walls or vegetation. These barriers could be coded similar to how the drunks cannot exceed the extent of the environment. However, if the environment were to be amended with barriers, the environment could be adjusted to better represent real-life towns, filled with streets, barriers, and moving the houses into more realistic patterns.

Several improvements can be made to the visualisation of the model. Primarily, each drunk can be designated a colour and their respective house would also be the same colour. Additionally, the pub in the model is obscured, with the colourmap unable to differentiate the pub to the background, so adding in a colour function could assign a distinct colour for the pub to become detectable in the model.

5.0 General Sources

The model was initially developed upon pre-existing code and skills established from the first Assessment of GEOG5990M Programming for GIS Analysis: Core Skills, which illustrated an Agent-Based Model, where agents move randomly within an environment, eating the environment as they move. The Assessment 1 code provided a foundation for this project, where instead of moving randomly in the environment, the agents/drunks have some conscious decision-making in walking towards particular sets of X and Y values. The Assessment 1 code was particularly helpful for providing help in stopping the model when reaching the threshold of iterations, creating functions to read in data or find house positions, as well as producing the GUI interface. Other sources used were the websites Stack Overflow, which helped diagnosing errors in the code, and [Python Tutorial](https://www.pythontutorial.net/tkinter), which supported the development of the GUI interface.

Word Count: 1952 words