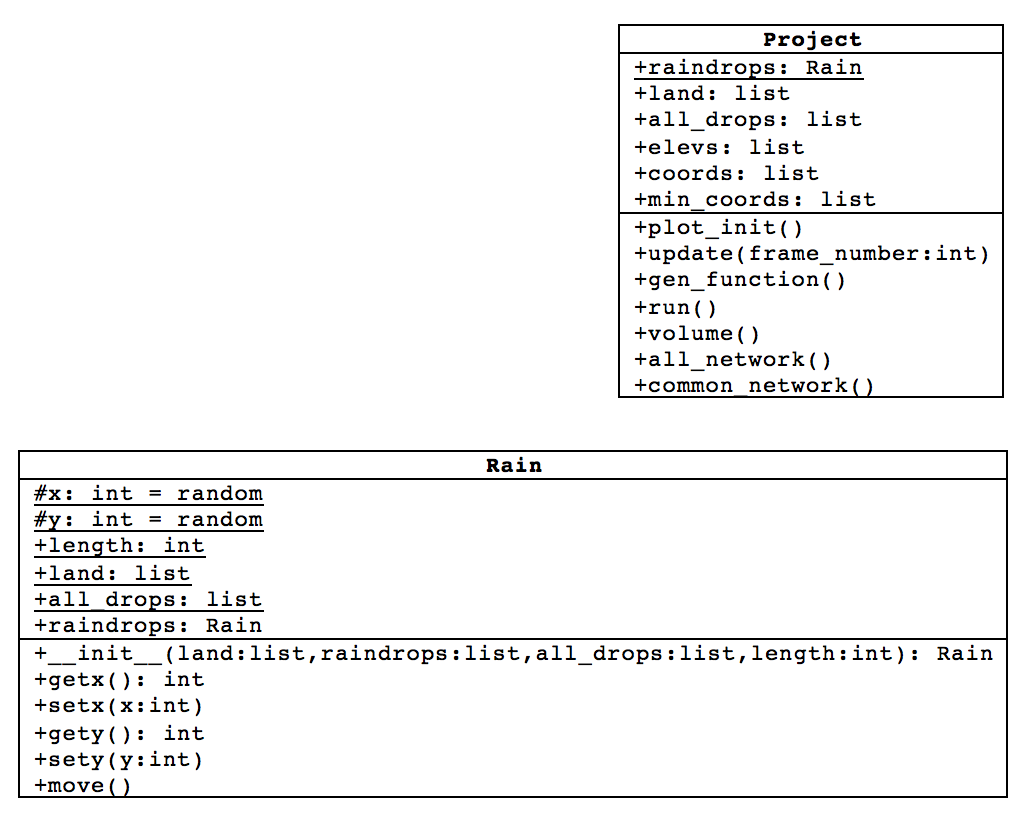
|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | | Name (optional): | |
|  | |  | |
| 200764899 | | Python Project | |
|  | |  | |
| MSc River Basin Dynamics and Management with GIS | |  | |
|  | |  | |
| GEOG5990M Programming for Geographical Information Analysis: Core Skills | | 1537 | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
|  |  | |  |
|  | | | |

**Independent Project: Ancillary Documentation**

**UML Diagram**



1

\*

**Intention of the Software**

The intention of the software is to build a program that models rainfall in a landscape. The model is designed to randomly create raindrops in the landscape, make them move downslope, and trace the paths of the raindrops to allow visualisation of the drainage network in the environment. The program also provides the user with the option to calculate the total volume of water that reaches an outlet point in the landscape.

**Issues During Development**

*Issues with range of coordinates*

Due to the set-up of the ‘move’ function in the rain framework, using the full extent of the landscape became an issue as the code required an out-of-range index to be found within the landscape data, despite this index not existing within the data. Originally, to avoid this issue, the agents were created to only have coordinates ranging between values of 1 and 98, so the ‘move’ function never tried to calculate out of range. However, this did not seem to be the best way to approach this issue as, ideally, the model should be able to operate within the entire landscape.

This issue was resolved by using a series of ‘if’ and ‘elif’ statements within the ‘move’ function to tell the program that different coordinates have different numbers of neighbouring coordinates depending on their position within the landscape, i.e. agents plotting in the middle of the landscape have eight neighbouring coordinates, agents plotting along the edges of the landscape only have five neighbours, and agents plotting at the vertices have three neighbours. This resolved the index range issue, and the model is now able to operate in the full extent of the landscape. It must be noted that the code for the ‘move’ function is lengthy and it is possible that it could be condensed, however, due to time constraints, a shorter alternative version of the code was not written for this project.

The user of the model can now also adjust the range, according to the size of the environment data, by setting the ‘length’ parameter to the desired value. However, this value must remain no larger than len(land) – 1 otherwise an ‘out-of-range’ error will occur.

*Creating a stopping condition that would work for any environmental dataset*

Initially, the model met the stopping condition when all agent coordinates were either (0,0) or (99,99). These were the coordinates of the two outlet points for the landscape data contained within ‘land.txt’. However, on another environmental dataset, these outlet points will likely be located at different coordinates, so the stopping condition did not work for other datasets.

In order to get the stopping condition working for all environments, the coordinates of the lowest elevation were found within the environmental dataset, and then these coordinates used within the stopping condition statement. The stopping condition is now activated when all raindrops meet a point of lowest elevation. If the stopping condition is not met, the model runs for the number of iterations (‘num\_of\_steps’) set by the user at the start.

*Appending the correct values to the ‘all\_drops’ list*

Much difficulty was encountered when attempting to append all raindrop positions to a list (‘all\_drops’). After attempting to append the values in the main ‘project.py’ model script, it was discovered through use of print statements that by the end of the model run, only the final raindrop positions were being appended to this list as the x and y coordinates were protected with a property attribute in the Rain Framework. After much deliberation, it was decided that the best place for appending to the list in the code was as part of the ‘move’ function in the Rain Framework. This allowed the coordinates of the neighbouring coordinate with the lowest elevation to be appended to the list as these coordinates are not protected by a property attribute. The coordinates in the ‘all\_drops’ list can now be used to plot the paths of the raindrops in order to view the drainage network of the landscape.

*Sinks in the landscape*

In the model, many raindrops become stuck in a sink in the landscape as they as no longer able to move downstream. The larger the environment that is used, the bigger an issue this is. As a result, bigger environments require more raindrops to fall onto the landscape to better visualise the drainage networks. An improvement for the model, which has not been implemented, could be to measure the amount of rain that has become trapped in the landscape.

*Enabling and disabling menu items in the GUI*

During initial set-up, the menu items in the GUI were all enabled upon running the ‘project.py’ script. This was problematic as the calculation of the total water volume was not possible until the model itself had been run. Therefore, if the ‘Calculate…’ > ‘Water volume’ option was selected before ‘Run model’, the model was encounter an error. The solution to this problem was to disable the ‘Calculate…’ sub-menu until ‘Run model’ had been selected. Upon consulting the lecture notes, it became clear that ‘entryconfig’, within the ‘TkInter’ module, was the required function to disable ‘Calculate…’ until ‘Run model’ had been selected. After further assistance from a website (ActiveState, 2004), the problem was resolved, and calculation of the total water volume is no longer possible before the model has been run. The ‘Drainage network’ sub-menu is also disabled until ‘Run model’ has been selected. Ideally, these sub-menu options would not be available until the animation had finished running, but a solution to this was not found during the development of this program.

*Possible future development*

Possible future development for the software could include the calculation of the volume of water at each outlet point (if more than one exists on the landscape). This development was attempted but unfortunately the model only, currently, calculates the total volume of water across all outlet points. Furthermore, the model could potentially be programmed to delineate watersheds and calculate the drainage area in the future. This would be useful for more quantitative analysis of river environments.

**Thought Processes Going into the Software Design**

Simplicity was the key aim for the software design. It is intended to be easy to use and to clearly display the model outputs. Initially, the GUI only contained one menu option: ‘Run model’. This ran the model in full, with all capabilities processed at once. However, it was decided that the software should provide the user with options. Therefore, the ‘Run model’ menu item simply runs the animation. A sub-menu ‘Calculate…’ was later added to provide the user with the option to perform calculations on the output data provided by the model. The software currently only provides one calculation option ‘Water volume’. However, there is scope to extend the calculations options in the future. Another sub-menu was added, ‘Drainage network’, to provide the user with the option to display the drainage network of all raindrops in the model (‘Show whole network’), or display only the pathways taken by three or more raindrops (‘Show common network’). It was deemed important to provide the user with options so that the software could be used to provide only the information that the user wants to see. It also makes clear the capabilities of the model.

**Software Development Process Followed**

Before development of the software, a list of steps by which to approach writing the code was made to breakdown the software into small chunks to attempt. Once each step was attempted, print statements were used to verify that the code was operating correctly before moving onto the next step. Only one step was attempted at any one time to ensure the code was always working as intended. A degree of trial and error was used during the software development to attempt to work out how to approach each chunk of code. This proved very beneficial as it enabled a better understanding of the importance of ordering and structuring code correctly.

During the development of the software, the ‘land.txt’ environment data was used as it is simple and it enabled clear determination of the functioning of the code. The ‘land.txt’ landscape was created artificially in Python. For reference, this Python script, ‘land.py’, has been provided. ‘in.txt’ is data for a real environment, and is much better for visualising the drainage networks in the program.

**General Sources Used**

In addition to the lecture notes for the module GEOG5990M Programming for Geographical Information Analysis: Core Skills, the following sources were consulted:

* ActiveState. 2004. *[TkInter-Discuss] Tip: How to Enable/Disable Menu Items.* [Online]. [Accessed 29December 2017]. Available from: <http://code.activestate.com/lists/python-tkinter-discuss/204/>
* O’Callaghan, J. F. and Mark, D. M. 1984. The Extraction of Drainage Networks from Digital Elevation Data. *Computer Vision, Graphics, and Image Processing.* **28**, pp.323-344.
* Stack Overflow. 2017a. *How to get item’s position in a list?.* [Online]. [Accessed 27 December 2017]. Available from: <https://stackoverflow.com/questions/364621/>
* Stack Overflow. 2017b. *Find and list duplicates in a list?.* [Online]. [Accessed 3 January 2018]. Available from: <https://stackoverflow.com/questions/9835762/>
* Stack Overflow. 2017c. *Finding multiple occurrences of a string within a string in Python.* [Online]. [Accessed 1 January 2018]. Available from: <https://stackoverflow.com/questions/3873361/>

Sources are provided as comments within the code to make it clear which code was written with guidance from a website.