Introduction

In our group we had three members and we all indicated a preference to work in Java and were assigned to work together.

Planning

In our first meeting we briefly touched upon our previous programming experience and what each member of the group was best suited to contribute. We decided to take a more open approach to planning and Colum and Xiao immediately started writing some base code for the project to present at our second meeting. At the second meeting it was decided that the group would expand upon Colum’s code for the rest of the project. Colum created the Population, GridMap, and TestDriver classes at that point that would eventually be presented in the final code for the project.

Colum wrote Population class was created to hold two two-dimensional arrays to hold hare and puma population densities across the landscape and it hold various other kinds of information on the simulated population. Of interest in the Population class is the updatePop method, which updates the hare and puma populations every for every time step of size . The updatePop method takes in Puma and Hare objects as parameters, which hold information on the hare and puma populations, and makes use of several other methods such as the getAdjHarePops and getAdjPumaPops methods, which both take in the coordinates of a particular square in the landscape and return the sum of the adjacent squares’ populations. It is in the updatePop method that the differential equations described in the assignment are approximated.

In the GridMap class Colum wrote the code for the program to hold a two-dimensional array representing the landscape. The GridMap class holds a representation of the landscape in 1’s and 0’s as well as several methods to collect information about the landscape. The GridMap class has a isDry method to tell whether a given square is dry or not as well as the getDryNeighbors method which returns the number of dry neighbors adjacent to a given square. These two methods are both called upon elsewhere in the program including in the Population class.

In order to meet all the requirements for how the program is supposed to run as described in the assignment Colum drew up a list of steps for the TestDriver class to follow so that the program performs like it’s supposed to. The TestDriver class follows this list of steps and starts with taking in a file and reads the first line of the data file, which contains the dimensions of the landscape. The program then instantiates a GridMap object called landscape by feeding the file and the landscape dimensions into a GridMap constructor, which reads the file and creates a representation of the landscape in the program. Then TestDriver class creates a Population object called population using a constructor that takes in the GridMap object landscape as an argument so that two two-dimensional arrays can be created to hold information on the hare and puma populations. The program then proceeds to ask the user if he/she wants to modify the default hare and puma population characteristics such as the birth, mortality, and predation rates. The population characteristics are stored in a Puma object and a Hare object for each respective population. The Hare and Puma classes hold these population characteristics and are called upon later in the simulation. After determining the population characteristics, the user can change the number of time steps in the simulation, which is held as a field in the Population class. When the number of time steps changes the field must change as well and this calculation is performed in the background automatically without notifying the user. The user then chooses how often the program prints to the data file before deciding how to initialize the simulation. The user has two options of how to start the simulation: he/she can choose to assign hare and puma populations to particular squares or he/she can choose to assign random populations across the map between 0 and 5.

, creating Population and GridMap objects, taking in new values for the differential equations, asking the user how he/she wants to initialize the simulation, getting a path extension to print the data output, and finally running the simulation before asking the user if he/she wants to run another simulation.

* Colum wrote up the list of steps that the program should perform
* We let it grow organically (questionable thing to say)
* We initially wanted to use a dual coordinate system, but we soon got rid of this idea because it was too complicated and was unnecessary for writing the program.
* We took an open approach to planning

Division of Labour

In our first meeting we briefly touched upon our previous programming experience and what each member of the group was best suited to contribute.

* Organic as well (not the best way to do it)

Programming Language

* We worked in Java and it seems like it was a robust enough language for us to work in.
* Java is a very versatile language and was very useful
* Because Java is so robust we were able to use it on 3 different OS’s.

Revision Control

* Colum set up a GitHub repository, which is where we worked from.

Build Tools

* Xiao wrote an ANT file

Debugging

* There was some confusion using N\_x and N\_y as fields because it was difficult to remember, which one was columns and which one rows. This lead to some problems later on that Colum rectified using the debugger in his IDE.

Performance Tests and Analysis

* Xiao

Conclusion

* Needed better design planning and division of labour. I’ve never done a programming group project before and I wasn’t sure how we should divide our work up between us.

Further Work