Introduction

The purpose of this project was to create a program to allow a user to simulate hare and puma populations within a defined landscape where the respective populations were modelled by the following partial differential equations:

where represents the hare population, represents the puma population, and , , , , , and are all constants that can be altered by the user before the start of the simulation. The simulation modelled these continuous equations using a discrete approximation over a landscape partitioned into squares.

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 had created the Population, GridMap, and TestDriver classes at that point that would eventually be presented in the final code for the project. To guide our efforts Colum drew up a list of tasks that the program must perform to meet the project requirements. This list guided the group’s efforts and kept everyone on task.

Group Organization and Task Allocation

We met at least once a week to discuss and review where we were with the project and talk through any design or coding issues we had come across.

We chose to have a quite flexible and fluid allocation of tasks allocation. We are on three completely separate degree programs and have very different sets of courses and other coursework and commitments so we decided we should try to work around this as much as possible. This meant that we knew that some weeks certain people would be able to do less work than others and we allocated the work for that week accordingly but tried to make sure that that every person did a fairly equal share of the work across the entire time we worked on the project.

For the final project, Sarah wrote the code for the program to read in and print data, as well as the classes to hold the hare and puma population characteristics. Colum wrote the main method, the code to hold information on the landscape and the hare and puma populations at a given point in time, as well as the algorithm to update the population for each time step. All group members contributed to the final report, which Colum brought together.

Design

In the basic design of our program, and in keeping with the principles of object-oriented programming, we organized the program by the basic data it holds and the functions that it performs. The program has classes to hold the characteristics of the puma and hare populations, a representation of the landscape, the number of animals across the map at any point in time, and methods to read and print data files.

Colum and Xiao both started the project by writing the base code independently before it was decided that the project would continue by integrating a few of Xiao’s methods into Colum’s initial code. Colum created the GridMap, Population, and TestDriver classes.

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

The Population class has two double arrays to hold hare and puma population densities across the landscape. Of interest in the Population class is the algorithm in updatePop method, which uses a discrete approximation of the above differential equations to update the hare and puma populations 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 return the sum of the adjacent squares’ populations.

In our first draft of our code information about hares and pumas was integrated into the “Population” class. We found this rather undesirable in the long term as we felt that the methods in the “Population” class should be able to deal with any predator/ prey system that followed the same structure of equation. To this extent Sarah created the “Animal” superclass to hold the general properties of an animal including type (predator/prey), birth rate, diffusion rate, mortality rate, and predation rate. The superclass ensures that only an animal of type “predator” is assigned a predation rate. The use of a superclass rather than an interface was to allow explicit coding of the methods all Animal objects can have. The set methods for the various rates ensure that the user cannot enter a negative rate. This meant that the classes for “Hare” and “Puma” that defined the properties of these particular animals were quite bare and only included some specific constructors. A more complicated structure of an “Animal” superclass with “prey” and “predator” subclasses were considered, but we decided that this only made the project unnecessarily more complex since the Animal superclass was more than sufficient.

The PrintingMethods class contains a number of static methods to print different information from a population to a file. The printingDenstityFile method prints the average densities of at a given time for the predator in the 1st column and for the prey in the 2nd column to a file specified by the user. This allows the user to view the densities of the two animals side by side on the same graph and see how the two populations interact. Average density was defined as being the number density in a square relative to the total animal population. This means that if the user manages to print a file where at a given time the densities of both populations are “1” there is a bug in the code.

The PrintingMethods class has a method to convert a double value into an RGB integer value. This normalized the number by dividing by the maximum possible and multiplying by the maximum RGB value as specified by the user. In this project this means that the population of a given square is scaled by the total animal population and multiplied by 245 and adding 10, to ensure any square with some population is at least slightly colored. So a square with no population would be completely white with a value of (255,255,255) and a square with the entire population would be all black with a value of (0, 0, 0).

There is also a method to print the density grid to a plain ppm file, which creates a single color ppm file. Each animal is represented with a separate color to make it easier for the user to watch the individual animals and changes. The magic number 'P3' is on the first line, the next line is blank, next maximum RGB value, the next line contains the dimensions of the population grid. The rest of the file is a matrix with the name number of rows as the population grid and three times the number of columns. As the RGB values are separated into three columns per one column of the grid each square is represented by one pixel.

The method to create the matrix is separate to the print method the matrices to represent the densities of hares and pumas can be printed separately and we can also print the combined maps of their densities. The only trick to the addition is to make sure that adding 255 is the same as adding 0 (white added to white is still white).

Reading in initial conditions can be done from an input text file or from the user entering them from command line prompts in the ReadingMethods class. The readInitialValuesFromFile method only deals with setting up the initial conditions by setting proportions of hares and pumas across the landscape.

The user can also chose to run the simulation just with “ant run” when they do this they will be lead through a series of questions that allow them to change an individual values one and at time. They can also choose where they want the initial populations of hares and pumas to be very precisely by setting the population in an individual square.

Finally, the main method in the TestDriver class pulls all these other separate classes together and starts by calling the ReadingMethods class. The program prompts the user to either feed in a file a of initial population parameters or to enter them manually through the terminal as well as a destination to print the data from the simulation. After creating Puma and Hare objects using information from the ReadingMethods class the simulation starts

Programming Language

We all indicated a preference to work in Java given our previous experience working in that language and we found that Java was a more than versatile enough language to write a robust program to handle the simulation. It is interesting to note that the language was also flexible enough for our group members to use work on the project across several different operating and systems and IDE’s without significant issues.

Revision Control

We agreed to use a GitHub repository for our project, which is where we stored our work. Colum set up the GitHub repository and added Sarah and Xiao on as collaborators to the project. GitHub proved to be an easy to use tool for our project and we did not encounter any major issues and were able to painlessly recover a file that one of our group members had accidentally deleted.

Build Tools

For this project, the Ant was chosen as the build tool and the build.xml file has been generated to build the simulation.

Just like “Makefile” for the C language, Ant is an efficient tool widely used to build projects. It has a variety of functions, such as compiling classes of Java, defining tags and configuration files, and generating web packages to submit to tomcat. It uses xml to describe targets and set properties for each task.

The biggest challenge with Ant is that the syntax can be quite cumbersome in the build file. However, it can be handled through the internet due to its widespread use. [clarification]

For this project, some targets are used to implement different functions such as init, compile, run, and clean. The targets of “init” and “clean” were implemented to make and remove build directories, “compile” and “compileTests” to compile source and test code, “run” and “runTests” to execute classes, and “jar” and “jarTests” to create jar files. The use of MacroDef not only reduces duplication of effort, but improves the efficiency and the reusability of code, and guarantees ease of maintenance. The test results can be written to XML and HTML, which allows the user to view them through a web browser.

Therefore, build tool can be performed as “write once run many”. It can manage code more reasonable and keep the structure of project more extensible. [clarification]

Testing Framework

We made sure that we were all involved in testing the code. Unit tests were created for the various classes we created to make sure that we were getting correct values for certain methods or that exceptions were being thrown as we wished. We tried to make sure we had unit tests for things we knew from the outset and occurred to us while writing and testing the code that the user might try to do wrong that might make the code fail. We acknowledged that we would not find everything or even manage to think through all the possibilities in the beginning. We accepted that certain methods are best tested through running the entire code and viewing the results. While the methods may run as expected and we do not know the exact nature of the results it would have been difficult to create unit tests for these but we could look for certain trends and properties in the outputs to test that the program was working as we expected.

Debugging

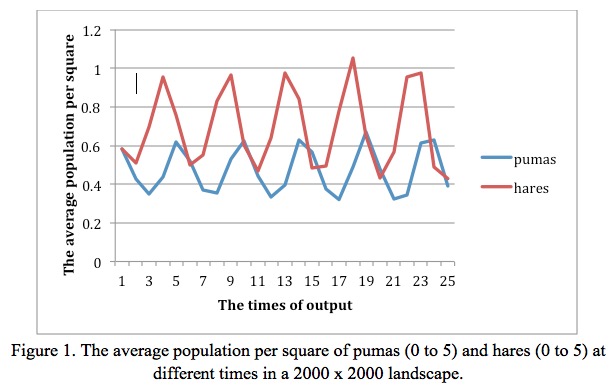
We initially encountered some bugs in the GridMap class that were mostly due to confusion involving the notation. Colum initially wrote the GridMap class and named the fields for the columns and rows as ny and nx to conform to the notation in the assignment of and . The names of these fields are not very intuitive Xiao confused the two when she expanded upon the code in one of the constructors; Colum later identified this bug and corrected the situation by introducing new field names.

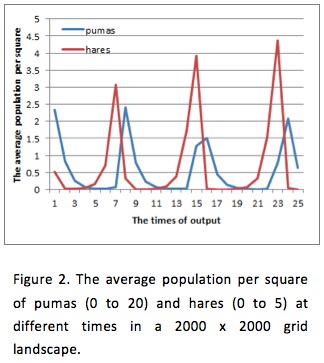
We later encountered bugs in the Population, PrintMethods, and ReadingMethods classes involving the updatePop method, which updates the populations, the code for reading in files, and printing to files. We all took part in searching the code for problem areas and identifying bugs using IDE debuggers and by manually running the code and examining the output.

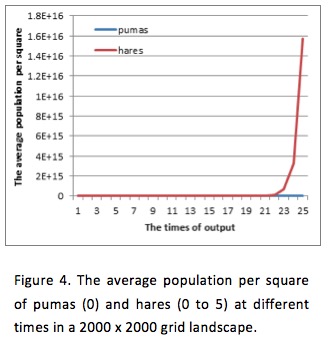
It is interesting to note that on more than one occasion we were all able to identify the same bugs while using debuggers in different IDE’s.

Performance Tests and Analysis

To optimize our project the first step in profiling was to measure speed and efficiency of the program. Timing by hand was the method we chose to reduce impact on overall code performance. Through adding some timing calls, the results showed that the updating part cost the most in CPU time, which is exactly what we expected. To improve performance, the logic of the calculation the new number of animals was optimized. For example, the program does not need to examine the water portion of the landscape in its calculations, which increases the program’s runtime unnecessarily. Therefore, the project only examines the land portion of the landscape in the updatePop algorithm. The experiment was conducted with varying sizes of landscapes, which indicated that larger sizes increased CPU time.

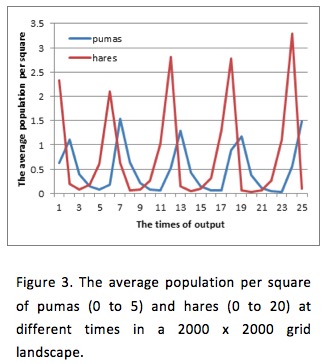


In regards to the compiler, JVM of IBM can conduct mathematical computation very quickly. In addition, JVM of BEA has the best performance in handling a large number of threads and network sockets. However, Sun’s JVM usually handles whole business logic the best, and it has been used for our project due for that reason. Although Java does not have such many flags to improve compiling performance compared with the C language, VM supports HotSpot compilation, which introduces the cache mechanism to store binary codes temporarily generated by Java codes with high frequency of being executed. The garbage collection is embedded by the virtual machine automatically, and that kind of mechanism can collect memory that was not used by object and clear the fragments of memory.

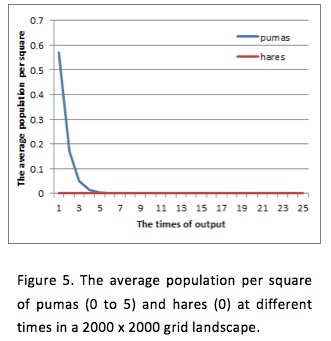


From the experiments we conducted, we can draw several conclusions.

Firstly, if the initial numbers of hares and pumas obey the natural laws, the trend of changing in pumas is always following the trend of changing in hares.

This is in line with the ecological laws of nature. To a certain extent, an increase in the number of prey will cause an increase in the number of predators, but both the growth and reduction in number is not on the same period of time. The time of maximum number of prey is not same with the maximum number of predator. The reason is that the time of maximum number for predators is the greatest inhibition to prey.

Secondly, when the number of pumas and hares start from different levels from the natural theory, for example, the density of pumas takes random values from 0 to 20 and hares from 0 to 5 (Figure 2), and the reverse situation (Figure 3). From the results of the experiment, we find that the initial setting of percentage or the gap of predators and preys will not influence the global trend in the change of pumas and hares, which means on that overall, the number of pumas is always changed by the number of hares. However, from Figure 2 and Figure 3, there are also some differences from Figure 1. For the circumstance of more pumas, the number of preys will drop dramatically due to the reason of pumas, and meanwhile pumas will also decrease in this period of time, while in the latter case, the large number of preys will lead to the increase of the number of predators.

Thirdly, when setting the densities of pumas and hares to 0 respectively, the number of prey grows exponentially in the absence of the presence of predator (Figure 4) while the number of predator decrease exponentially in the absence of prey (Figure 5).

Fourthly, the “gif” file was created from many .ppm files in output to show the results clearly. It indicates that no matter how uneven the initial densities of pumas and hares are, they will shortly distribute evenly across the landscape and the densities or the numbers will rise and drop circularly.

Finally, the situation that non-water grid squares is discontinues was considered, but it caused the non-real birth and death.

Conclusion

This program read data from .dat file and produced .ppm files with different densities of the animals. When users run this program, they can use a default input file that includes default values or input some values by themselves.

In this project, git and GitHub were used to merge our code appropriately for revision control. Ant was chosen as build tool to build and deploy our code automatically. Test units were used to test each class and ensured correctness. We debugged the program using IDE debuggers when we finished every function. Timing by hand was used to profile and the experiment was conducted in different inputs to get results for analysis.

Future work

The speed and efficiency of the program would be vastly improved with the use of Java Threads.

We have tried several kinds of input methods for users to choose, and in the future they are expected to be more suitable and intelligent for users.

The output approaches can be modified to generate more meaningful .ppm and .gif files, which can be more explicit not only for users but also for our data analysis and testing.