University of Edinburgh
EPCC

Programming Skills Coursework
Project Report
Group 7

Contents

1 Introduction

1.1 The Task & Mathematics

The aim of this project is to implement sequential version of a two-dimensional predator-prey (Pumas/Hares) model with spatial diffusion using C++. A key part of the project is to produce "best practise" scientific code and utilise the skills gained throughout the course Programming Skills such as version control, build tools and testing framework. Without further ado we introduce the governing coupled system of partial differential equations that govern the densities of hares and pumas in a landscape.

$$\frac{\partial H}{\partial t} = rH - aHP + k\left(\frac{\partial^2 H}{\partial^2 x} + \frac{\partial^2 H}{\partial^2 y}\right) \tag{1.1}$$

$$\frac{\partial P}{\partial t} = bHP - mP + l\left(\frac{\partial^2 P}{\partial^2 x} + \frac{\partial^2 P}{\partial^2 y}\right) \tag{1.2}$$

where H is the density of hares and P the density of pumas. The coefficient r is the birth rate of hares, a the predation rate coefficient (the rate at which pumas eat hares), b the reproduction rate of pumas per one hare eaten and m the puma mortality rate. k and l are the diffusion rates for the two species, hares and pumas respectively.

To convert this continuous problem to one of a discrete nature suitable for a computer to solve we divide space and time into discrete quantities in order to approximate a solution. Time is modelled by successive, discrete, time-steps, of size Δt. The numbers of hares and pumas in the landscape at next time step depend on the numbers present in the current time-step. So, the new number of hares, H, at time $t + \Delta t$, in grid square (i, j), is given by:

$$H_{ij}^{\text{new}} = H_{ij}^{\text{old}} + \Delta t\left(rH_{ij}^{\text{old}} - aH_{ij}^{\text{old}}P_{ij}^{\text{old}} + k\left(\left(H_{i-1j}^{\text{old}} + H_{i+1j}^{\text{old}} + H_{ij-1}^{\text{old}} + H_{ij+1}^{\text{old}}\right) - N_{ij}H_{ij}^{\text{old}}\right)\right) \tag{1.3}$$

N_{ij} is the number of "dry" (i.e. non-water) grid squares out of the four neighbouring squares (i, j) (to the north, south, east and west). It is assumed that hares cannot move diagonally nor can they swim.

The new number of pumas, P, at time $t + \Delta t$, in grid square (i, j), is given by:

$$P_{ij}^{\text{new}} = P_{ij}^{\text{old}} + \Delta t\left(bP_{ij}^{\text{old}}H_{ij}^{\text{old}} - mP_{ij}^{\text{old}} + l\left(\left(P_{i-1j}^{\text{old}} + P_{i+1j}^{\text{old}} + P_{ij-1}^{\text{old}} + P_{ij+1}^{\text{old}}\right) - N_{ij}P_{ij}^{\text{old}}\right)\right) \tag{1.4}$$

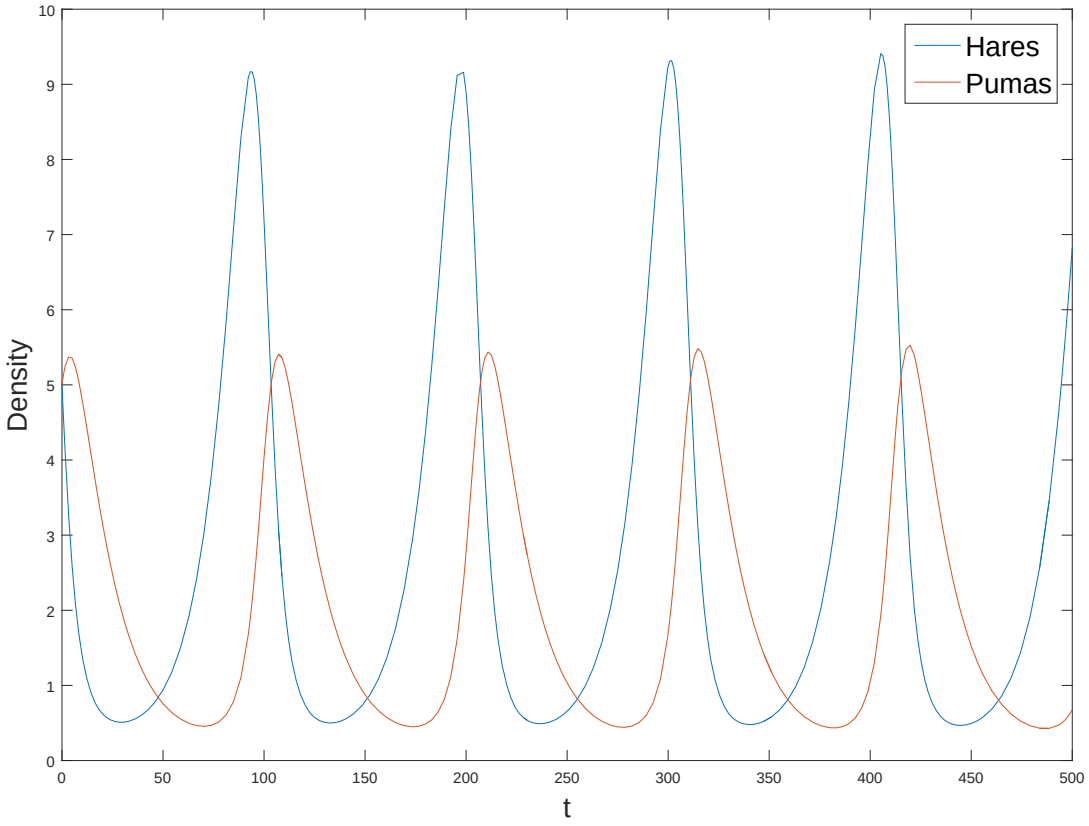
As a reference it is interesting to consider the simplified version of equations (1.1) and (1.2) which drop the addition of diffusion coefficients. These equations then become the classic Lotka-Volterra equations,

$$\frac{\partial H}{\partial t} = rH - aHP$$

$$\frac{\partial P}{\partial t} = bHP - mP$$

which have a lovely oscillating solution with the default parameters for our project $r = 0.08, a = 0.04, b = 0.02, m = 0.06$ and a, uniform, initial density of 5 hares and 5 pumas. See figure 1.1 for this behaviour.

Figure 1.1: Lotka-Volterra Solution



2 User Manual

2.1 Compile and Install

2.1.1 Source Code Folder

The source code folder was created in order to distinguish the source code, libraries and all the other files needed to compile. It is structured as following:
PROJECT_HOME_FOLDER

\src	The folder for source code
\include	The folder for header file
\lib	The folder for for third party or shared library
\build \bin	The folder for executable file
\obj	The folder for object file
\testbin	The folder for executable unitest file
\cfg	The folder for configuration file
\dist	The for packed program
\test	The folder for source code of unitest
\tools	The folder for source code of tools
\doc	The folder for documents
\ppmfolder	The path of output ppm file
makefile	The makefile for compile and installation

2.1.2 Compile Code

All the operations of this program is executed by make command with a makefile. To compile the code, should be at the root folder of the project, a Makefile is prepared there. Afterwards, type: `make all` to compile the program. In compiling, the .cpp file in `./src` folder will be automatically compiled. All the object files will automatically be saved at `./build/obj`, and all the excutable binary file will be directely shipped to `./build/bin ./`

2.1.3 Installation

After compiling, use `make install` to install the program to your system's `HOME\PScoursework1` folder.
We use different folders to set up the product environment.
Our project's product folder is as below:

HOME \PScoursework1

\bin	The folder for executable file
\cfg	The folder for configuration file
\ppm_output	The folder for ppm style outputfile

The user can add the bin folder to it's own PATH environment variable for convenience.

2.2 Usage of Program

The program runs through the command line and the user can set all the various parameters via the command line, all the parameters is listed:

```
./caldensity -cfgfile <filename> -mapfile <filename>- ppmfolder <ppm_filefolder> -loops <looptimes>
-show <0/1> -interval <intervaltimes> -r <value> -a <value> -m <value> -k <value> -l <value>
-dt <value>
```

2.2.1 Detailed parameter description:

File path and name

mapfile	the path of the land map file
cfgfile	the path of the config file COMPULSORY
pumafile	the path of the file of puma's initialize density
harefile	the path of the file hare's initialize density
ppmfolder	the path for the ppm files to be saved

Key parameters and switches

loops	the total loop times (little t)
show	1 or 0 (1:display_changes_of_density, 0:don't display)
interval	the interval times for output (big T)
r	the birth rate of hares
a	the predation rate of pumas
b	the birth rate of pumas per one hare eaten
k	the diffusion rate for hares
l	the diffusion rate for pumas
dt	the size of time step

The -pumafile and -harefile are optional parameters and the system will randomly generate densities if these parameters are not given.

2.3 Configuration file

The configuration file exists for ease of use when choosing parameters. **Note** that, command line arguments have higher priority. In the configuration file, the "-" is not necessary, just use the name of parameter, like "loops". An example of configuration file is listed below:

r = 0.8
m = 0.06
k = 0.2
a = 0.04
b = 0.02
dt= 0.4
l = 0.2
loops = 500
interval = 10
show = 0
mapfile = $HOME/PScourse1/cfg/map.dat
ppmfolder = $HOME/PScourse1/ppm_output

Note that, because the relative path of the file depends on the user's current folder, **it is highly recommended** to use the absolute path in the configuration file.The program support $HOME environment parameter.

2.4 Density Display

When the switch "-show" is opened, two dynamic graphs of density will appear. Note that user should use X11 display forwarding to connect to the server. The user can also run it in any local machine that supports X11.

3 Tests

Every team member was responsible for creating the tests that corresponded to his work.
From the project's source folder (after running make all & make install) the user can also do: make test, in order to check that everything works as desired. Aside the console output, the results are also saved in: "source directory"/build/testbin/

For the main part of the program, we chose to test that:

- Test all the file name and path parameter using right and wrong value to verify the process logic
- Using $HOME in each path parameter to test if the program can replce the environment variable.
- Input numbered parameters using right and wrong and boundary value to test if can get the proper feedback from program
- Test if the PPM file can work output well in indicated location.
- Test the total time that the program output is correct or not.
- Test the mapfile with different size , to test it's capability and process ability.

- Test the config file read using right or wrong values in it.
- Test the program in display mode (isshow = 1) to see if the graphy is correct or not, and if it acordinate with our output.
- Do the integrate test and non-functional test, set a large map size and large loop count, run the program to see if it will crash.

For the landscape class, it was deemed necessary to test:

- If the same file read into pumas and hares is read in correctly and the two density matrices are identical.
- If two different file sizes are provided e.g. a map of size 50 by 50 and an initial density file of 100 by 100, the constructor throws an exception.
- Test that a density path is not required and two random densities are generated
- Wrong parameters are handled e.g. wrong value of r
- Wrong parameters are handled e.g. wrong value of dt
- The array corresponding to the number of neighbours
- Check that hares and pumas have the correct values after 1, 10 and 20 iterations
- the functions returning the averages of hares and pumas.

For the map generator class, we decided to test:

- Extremely small size of the map
- White noise output (no interpolation)
- Input with illegal values

4 Design Considerations

4.1 Project Size and Dependencies

After our initial meeting we agreed that the project did not require a particularly object-orientated approach. It was agreed that we would have two major classes: one to model the landscape and the evolution of the densities and one to take care of the display aspect of the project i.e. creating an animated display of the evolution of the densities. The main file would then run the simulation and link the two classes as can be seen from the class diagram in figure 4.1.

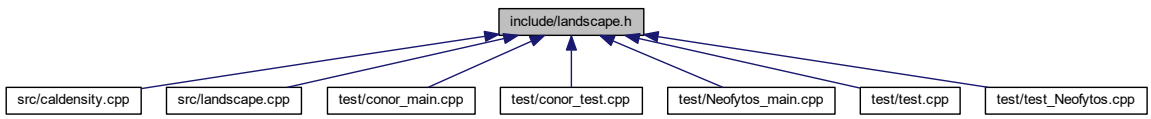


Figure 4.1: Labdscape Class Inheritance Diagram

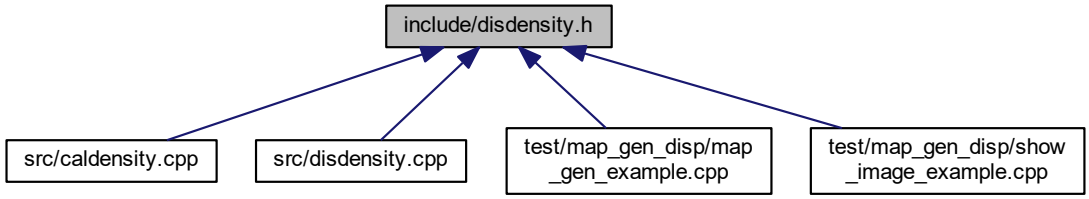


Figure 4.2: Class Inheritance Diagram

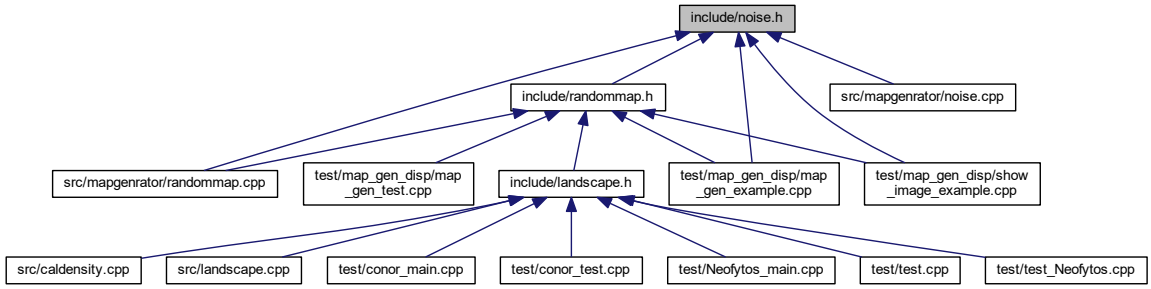


Figure 4.3: Class Inheritance Diagram

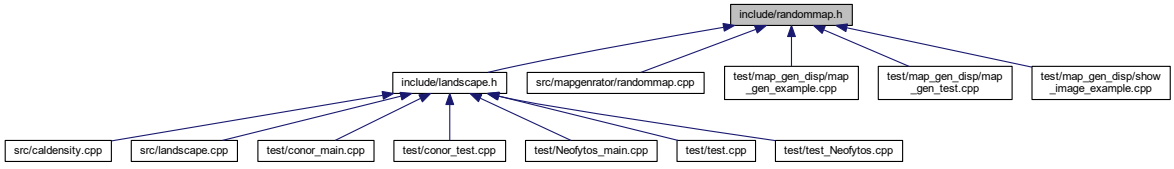


Figure 4.4: Class Inheritance Diagram

4.2 Design choices

The framework for the project was developed quickly after our first meeting and these are some of the key decisions that were made.

- Firstly we agreed on a directory structure and set up a remote repository hosted on GitHub containing this structure which allowed us to employ git as our version control.
- Secondly we agreed to use the Google C++ code style along with Doxygen to annotate our code. This ensured a consistent look across the project and allowed for easy compilation of the project documentation.
- We concluded that using a Makefile and make was suitable for building our executable and we also agreed that we would use the Google Test framework for creating our unit tests.

5 Version Control and Team Cooperation

5.1 Version Control

5.1.1 Tools and Service

Git was used as a tool for source code version control and more specifically Github, as the version control service provider.

5.1.2 Version Strategy

The team used master branch for the tested version of code. Dev branch was created in order to merge work after testing. Every team member created his own branch at local computer, to be able to work independently.

During the project, the team members pushed their work more than 130 times (fig 5.1)

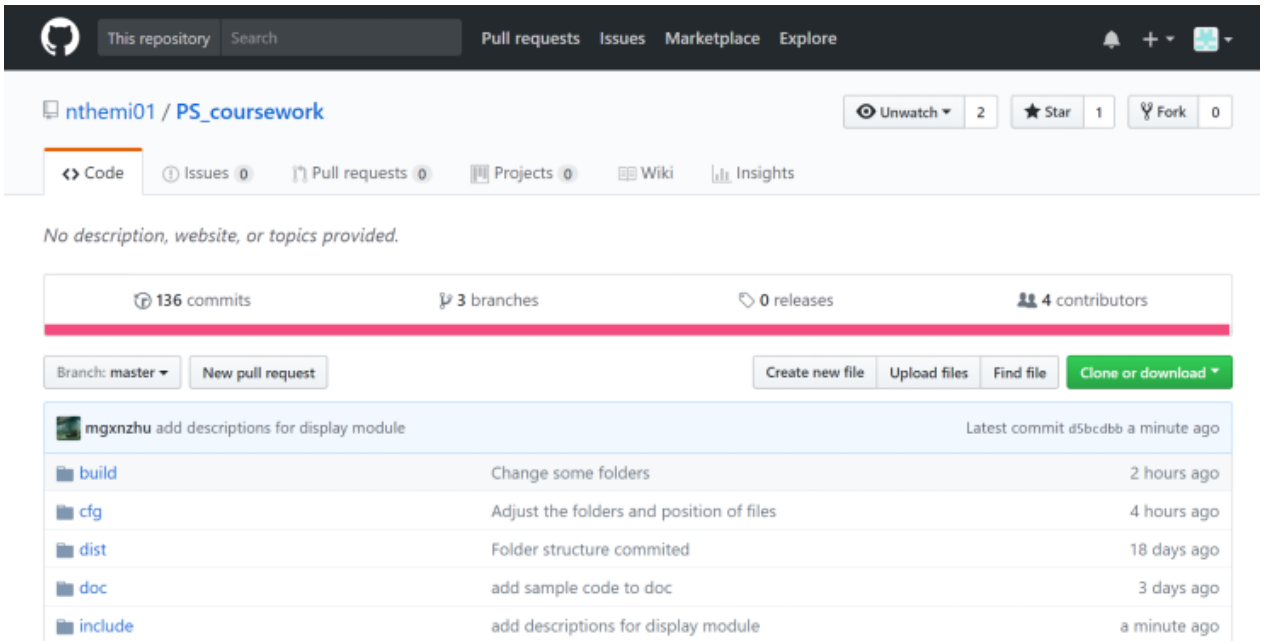


Figure 5.1: Image showing the number of pushes the team made to git

5.2 Communication

The team used Slack(www.slack.com) to communicate, exchange documents, codes and discuss problems.

5.2.1 Bug tracing

The team used the ISSUE function of Github to trace bugs, although most of our bugs were solved by discussion, as can be seen on figure 5.2.

Figure 5.2: Image illustrating the use of slack in order to resolve a real bug