

Programming Skills Coursework

Group 4

s1754385 Yixuan Li

s1702244 Yu Fu

s1722845 Zheng Guo

Nov. 03 2017

1. Introduction
   1. Project Introduction

This project aims to build a model to deal with the puma-hare problem within a full development framework by Java language. We need to solve the problem by groupwork. Finally, we will get the final generation density, the average density and the total time as the result and this documentation will illustrate how we finish this assignment.

1.2 Programming Language

Java is a generic term for the Java object-oriented programming language and the Java platform that was introduced by Sun Microsystems in May 1995. It was co-developed by James Gosling and colleagues, and was officially launched in 1995.

1.3 Revision Control

Git is an excellent distribution version control system. The version control system can keep a history of a collection of files and can roll back the file collection to another state (history state). Another state can be a different file, it can be a different file content. In a distributed version control system, everyone has a complete source code (including all the history of the source code information) and can operate on the local data. The distributed version control system does not require a centralized code repository.

1.4 Debugger

In this project, we use Eclipse to debug the code. Eclipse provides a Debug perspective which gives you a pre-configured set of views. We set a few breakpoint to test, like: read from a file, output a ppm file etc , and we use shortcut key to step through the codes.

1.5 Build Tools

In this project, we use java builder as our build tools. The Java builder builds Java programs using its own compiler (the Eclipse Compiler for Java) that implements the Java Language Specification. The Java builder can build programs incrementally as individual Java files are saved. Note that the Eclipse Compiler for Java can also be invoked using Ant as described in the using the ant javac adapter section.

1.6 Test Tools

1.6.1 Junit4 introduction

JUnit is a Java language unit testing framework. It was established by Kent Beck and Erich Gamma, and gradually became one of the most successful test system. Moreover, it has its own extended ecosystem. Most Java development environments have been integrated with JUnit as a tool for unit testing. The Junit test is a programmer's test, as is called white-box test as well, because the programmer knows the software how completes the function and completes what kind of the function. Besides, Junit is a set of framework, inherited Test Case class, you can use Junit for automatic testing.

1.6.2 System requirements

The recommended system configuration for this tutorial is as follows:

1. A system supporting either the Sun JDK 1.5.0\_09 (or later) or the IBM developer kit for Java technology 1.5.0 SR3 with at least 500MB of main memory.

[2. At](2.At) least 20MB of disk space to install the software components and examples covered.

3. The instructions in the tutorial are based on a Microsoft Windows operating system. All the tools covered in the tutorial also work on Linux and UNIX systems. (2007,IBM)

4. For the sections on running JUnit 4 in Eclipse, you need a working installation of Eclipse 3.2 or later.

2 Relative Technology

2.1 Third-Party Technology

Where to get, and how to build and install, any third-party packages needed by your code (for packages that are not already on the Physics Computational Lab machines).

We use Jfreechart to draw the line chart of average density of the result.

Before we use it, we need to install it into our project. The procedure is as followed:

1. Download the latest version of Jfreechart by the website : <https://sourceforge.net/projects/jfreechart/files/1.%20JFreeChart/1.0.19/>. Then unzip it.
2. Next step is to import the necessary JAR files into the project. If we use eclipse,

we can select the current program, right click it, select “build path- configure build path”.

3. In “libraries”, click “add external JARs”, find the path of jfreechart and then go into the lib, select “jcommon-1.0.23” and “jfreechart-1.0.19”. Add them into reference libraries.

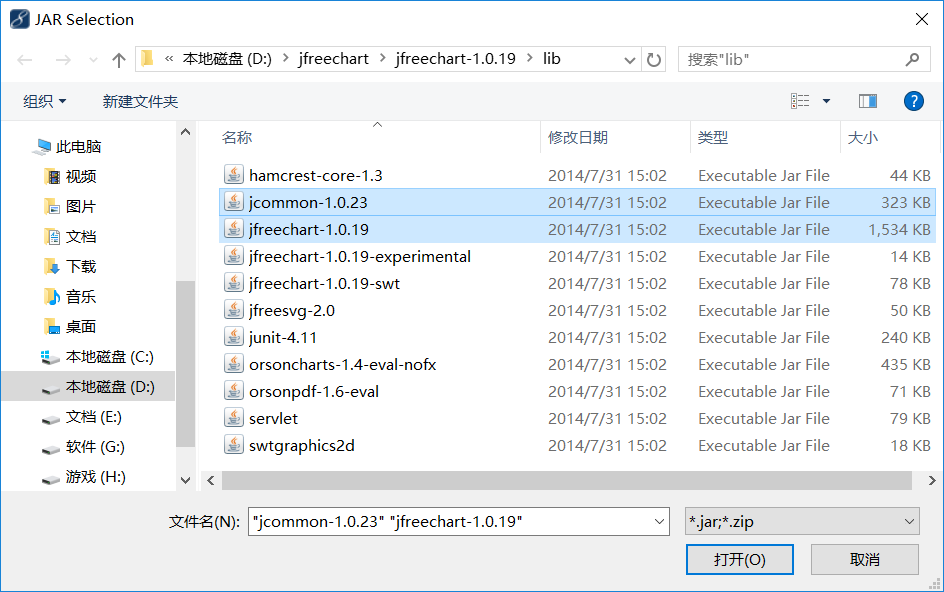


Fig.1 Third-party packages

3 Build the code

3.1 Build for main program

1. To make it a runnable file by command line from eclipse, we can find “export” button from file menu in eclipse.

2. We will come into a new window by clicking export. Then click “java” to go to a submenu.

3. Now we can see “JAR file” option. We need to pick that option and click “next” button at the bottom of the window.

4. Choose a proper destination for your JAR file by clicking browse and the click “next”.

5. The last step is to click “finish”.

3.2 Build for Junit test.

To run the Junit test on CP-lab computers, we make the whole program to a JAR file, so that it can be run from terminal. Once we built the JAR file, we built the Junit test at the same time. And the test can be run from terminal by the command introduced in the following part.

4 Run the code

4.1 Run the main program

1. Fisrtly, unzip the “Group4\_cw1.zip” using command “unzip Group4\_cw1.zip”

2. Next, go inside the folder Group4\_cw1 using “cd Group4\_cw1”

3. Last, run the script run.sh using command “sh run.sh” which runs the Junit test and main program toegther

4. After running the code, you supposed to see three results: (1)output ppm file and txt file in outputlist folder; (2)line chart of average density in outputlist folder; (3) Total running time in the terminal directly.

4.2 Run the Junit test separately

If you want to run the Junit test inside the Group4\_cw1 file individually, run it using command: “java –cp coursework1.jar org.junit.runner.JUnitCore com.test.TestProject”

5 Test the code

5.1 Test Plan

Table 1 Test Case

|  |  |  |  |
| --- | --- | --- | --- |
| Items to be Tested | Features to be Tested | Feature Type | Level of Risk |
| The user input value | The input should be an integer | User error protection | Low |
| Read from the file | Check the data read in is equals to the test data | Performance | Low |
| The width and height of the algorithm | Check width and height are not null and legal | Functionality | Low |
| The accuracy of random generating animal | Check the original density of puma and hare are right | Functionality | Low |
| The result of the algorithm | Test whether the algorithm can run successfully. | Functionality | High |
| WritePPM() | Write color array to a ppm file | Functionality | Low |
| WriteText() | Write the density array to a txt file | Functionality | Low |

5.2 Test Case

Table 2 Test Case 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case Name | Test user input | | Case No. | 001 |
| Case Version | V1.0 | | | |
| Pre Condition | No | | | |
| Post Condition | No | | | |
| Test Point | Testing Procedure | Expected Result | | Testing Result |
| When user input something from keyboard | Allow user to have an input from keyboard, and check whether the input is what we expect or not. | User input a value between 1 and 10 | | Test successfully without failure |

Table 3 Test Case 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case Name | Test the correctness of reading file | | Case No. | 002 |
| Case Version | V1.0 | | | |
| Pre Condition | No | | | |
| Post Condition | No | | | |
| Test Point | Testing Procedure | Expected Result | | Testing Result |
| Read from a test file (10x10.dat) | Test the data we get from reading the test file and compare it with the content of the test file | The test data is the same as the data reading in. | | If it is equal, it pass the test, otherwise, it fails. |

Table 4 Test Case 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case Name | The width and height of the algorithm | | Case No. | 003 |
| Case Version | V1.0 | | | |
| Pre Condition | No | | | |
| Post Condition | No | | | |
| Test Point | Testing Procedure | Expected Result | | Testing Result |
| Read width and height | Test whether height or width is null. If so, report failure.  Test whether height or width is illegal value. If so, report failure. | The height and width are both right. | | If they are null or illegal(over 2000 or below 0), report failure. If they are right, test successfully. |

Table 5 Test Case 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case Name | The accuracy of random generating animal | | Case No. | 004 |
| Case Version | V1.0 | | | |
| Pre Condition | No | | | |
| Post Condition | No | | | |
| Test Point | Testing Procedure | Expected Result | | Testing Result |
| Read array of puma and hare density | Test the value in each array is between 0 and 5. | Both are satisfied with the standard. | | They both are between 0 and 5. |

Table 6 Test Case 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case Name | The result of the algorithm | | Case No. | 005 |
| Case Version | V1.0 | | | |
| Pre Condition | No | | | |
| Post Condition | No | | | |
| Test Point | Testing Procedure | Expected Result | | Testing Result |
| Test whether the algorithm can run successfully. | Test whether the algorithm function can run smoothly. | It will return the successful flag(1). | | Flag = 1  Test without mistake |

Table 7 Test Case 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case Name | TesOutputPPM | | Case No. | 006 |
| Case Version | V1.0 | | | |
| Pre Condition | No | | | |
| Post Condition | After testing 005, please go to test 006 | | | |
| Test Point | Testing Procedure | Expected Result | | Testing Result |
| Test WriteText() method | Create a 10 by 10 2-dimensional matrix.  Write this matrix using WriteText() method to a txt file.  Read average density to variable “a1”.  Read the data from this txt file back to a 10 by 10 array called “text”.  Compare an defined double value with “a1”.  Compare this matrix and array “text” to see if they are equal using assertArrayEquals(). | A1 equals to the value we defined. And the matrix we defined is totally equals to the array we read from txt file. | | A1 equals to the value we defined and the matrix equals to array “text”. |

Table 8 Test Case 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case Name | TesOutputText | | Case No. | 007 |
| Case Version | V1.0 | | | |
| Pre Condition | No | | | |
| Post Condition | No | | | |
| Test Point | Testing Procedure | Expected Result | | Testing Result |
| Test WritePPM() method | Create a 3 by 3 2-dimensional array called “matrixInit”.  Write this array using WritePPM() method to a ppm file.  Read this ppm file back to a 3 by 9 array called “text”.  Compare this array “text” and another array we defined before called “matrixTest” to see if they are equal using assertArrayEquals(). | “matrixTest” we defined is totally equals to array “text” we read from ppm file. | | “matrixTest” equals to array “text”. |

5.3 Test result

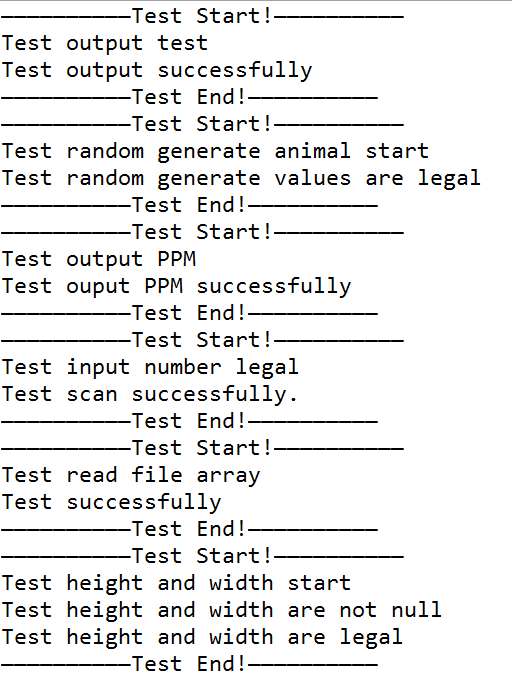


Fig.2 Result you will see after testing

6 Summary of key design decisions and reasons

6.1 Input

The input part of the code is designed to read data from a dat file which contains the height, the width(these two are in the first line) and the data(2D array, the other part of the file).

The class used to get those data is BufferReader. It has two ways to read data: first choice is read character by character while the second choice is read line by line. It is clearly to see that read line by line run quicker than former one. The data we get from a line is in String type and we have to separate this String into the data format we need and make it easy approach by main function.

We store the height and width into an array. The details is: we change data type from String to String array to integer array. It is one line of data, we only needs to read once and get result.

We store the data(the content of landscape) into an 2D array. To make this works, we change data type from String to String array to integer array to 2D integer array. As there are lines of data, we use a condition test to check the data that we read from is available( there still have something to be read), and when we finish read the whole file, we get the data and return to main.

6.2 Algorithm Section

6.2.1 initialize the animal information

First two animal classes (hare and puma) have been declared to store the information of these. In the 2 classes, they contain the basic static statistics information (like birthrate, predationRate and so on), the location on the map and the density value. And then, with the help of the map architecture(where contains land and water), we give all the land nodes of pumas and hares some random values which are between 0 and 5 so that all the land nodes have the initial values for algorithm.

6.2.2 make the model of algorithm

Next step is to achieve the algorithm. In order to get the right model of puma and hare, we need to calculate every node's neighbor first because accroding to the formula [picture of it], [sum of 4 nerghibours] is a necessary part and the number of neighbor is another essential part. The core part of the algorithm is how to get the correct number of neighbor. Our method is to calculate different situations of the location for the map

1. When we find the current node is one of the 4 vertex nodes (in the red circles), the neighbour number of current node is 2, and then detect how many water nodes the neighbours contain. If there is no water node, the final neighbour number is 2. If one water node, the neighbour number is 1. If two water node, the number is 0. After that we the calculate the sum of the density of the neighbours for the current node.



Fig.3 Map model analysis

1. When we find the current node is on the boundary but not the vertex ones (in the ovals), the neighbour number is 3, and then dealing with the water node just like part(1) do. Last, we calculate the sum of the density of the neighbour nodes.



Fig.4 Map model analysis

When we find the current node is in the middle part of the map which means it is not on the border lines (in the shadow part), the number of neighbours is 4. Next, the water node should be considered and the calculation will be done finally.



Fig.5 Map model analysis

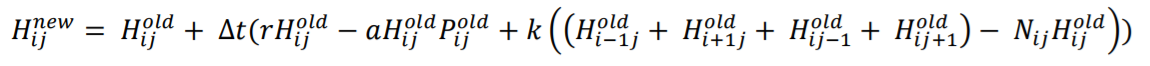
6.2.3 calculate the next generation of puma and hare

After we get the number of the neighbors and the value respectively, we have the result of the next formula:





We can calculate the result of density of next generation using the formula (NO.)as followed.





In the 2 formulas, we can use the value of parameters in table(No.):

Table 9 Parameters list

|  |  |
| --- | --- |
| parameters | values |
| r: the birth rate of hares | 0.08 |
| a: the predation rate at which pumas eat hares | 0.04 |
| b: the birth rate of pumas per one hare eaten | 0.02 |
| m: the puma mortality rate | 0.06 |
| k: the diffusion rates for hares | 0.2 |
| l: the diffusion rates for pumas | 0.2 |
| Δt: the size of the time step | 0.4 |

6.3 Output

The class WriteToPPM has to output the density array of hares and pumas to ppm files every T cycle. The first step is to read the result array which is given by calculation function for the hares and pumas. Then we need to a convert the amounts of the two creature to a certain color, and save the RGB text information to a ppm file which can be displayed by a certain software colorful. After we receive a 2-dimension array, every element in the result array is specified to a certain color. Then we create a new color array for each pixel which store its RGB value.

In ppm file, we need to write the information at begging of the file for other software to display the ppm file to a colorful picture properly. The program write “P3” at fist line to indicate the file encoding type. Then we write the size of matrix wrote in this file. And we need to put the range of RGB value to the next line. After all above, we can write the color matrix to this file.

In WriteToPPM() method, there are 3 parameters. The first one “resultArray” is the density results of either hare or puma in a 2-dimension array. “Animal” is to specify which animal density we are writing. Hare uses pink color, puma uses blue. T is the time interval that it needs to output a ppm file.

We also need to record the density of two creatures every T generations. So that we have another class called WriteToText() which writes the density array and average density value at this generation without converting to a color array to a txt file. There are four parameters. “resultArray” is the same with previous method to record the density. “average” is the average density of one animal at present. “animal” is to indicate which animal we are recording. “T” tells us the time interval that we output the txt file. And we make the average density of every T generations to a line chart which uses a third-party library called jfreechat and jcommon.

7 Reference

Eclipse. (2014). Eclipse documentation - Archived Release

Available:https://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Fconcepts%2Fconcept-java-builder.htm Last accessed 01.11.2017.

IBM. (2007). Jump into JUnit4.

Available:<https://www.ibm.com/developerworks/java/tutorials/j-junit4/index.html>. Last accessed 01.11.2017.

8 Appendix

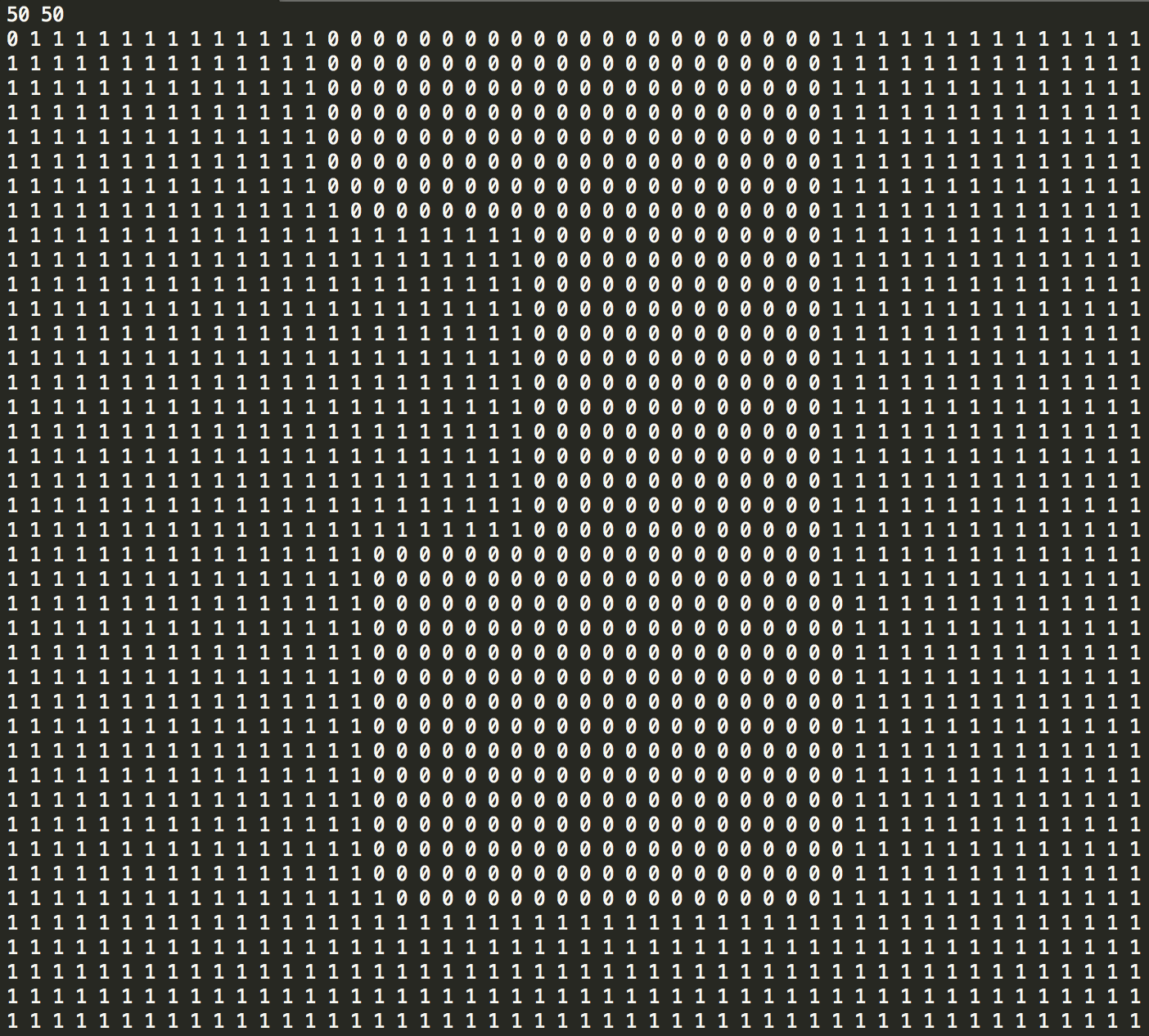


Fig.6 50\*50.dat map file

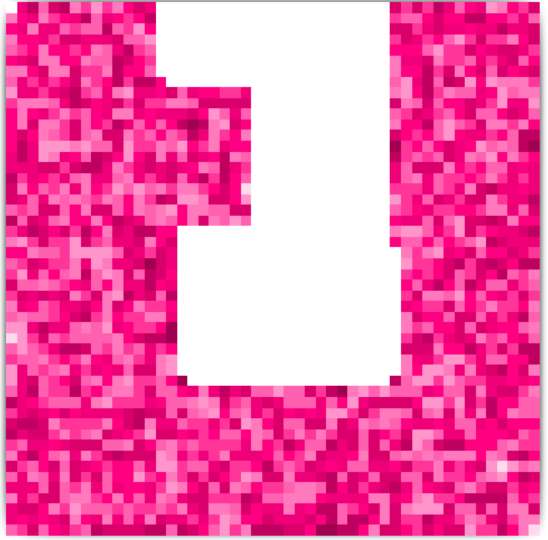


Fig.7 First hare generation result

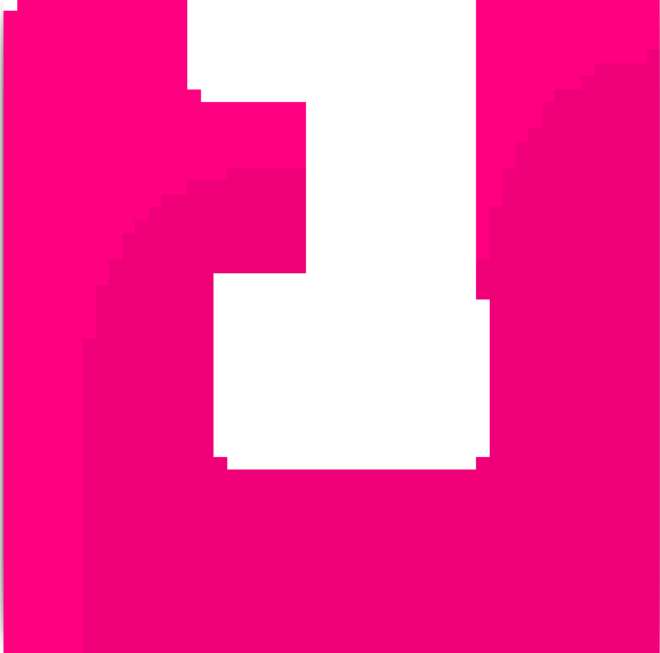


Fig.8 Last hare generation result(1250 generations)

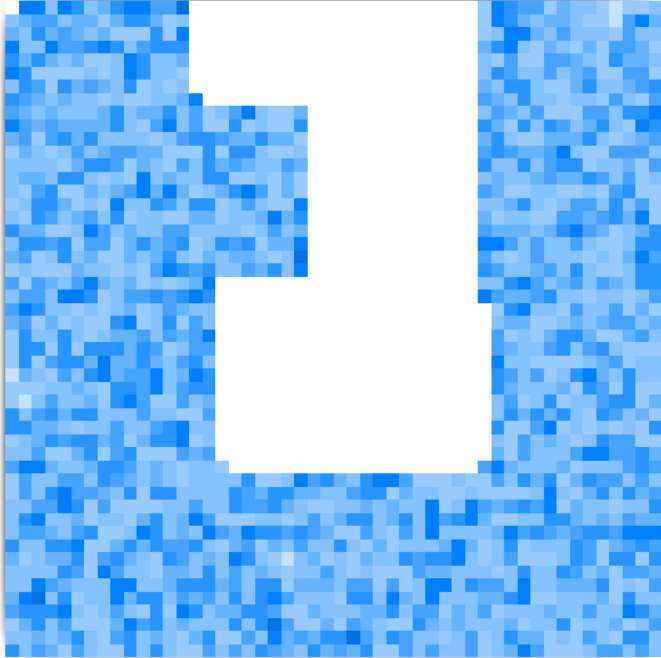


Fig.9 First puma generation



Fig.10 Last puma generations(1250 generations)

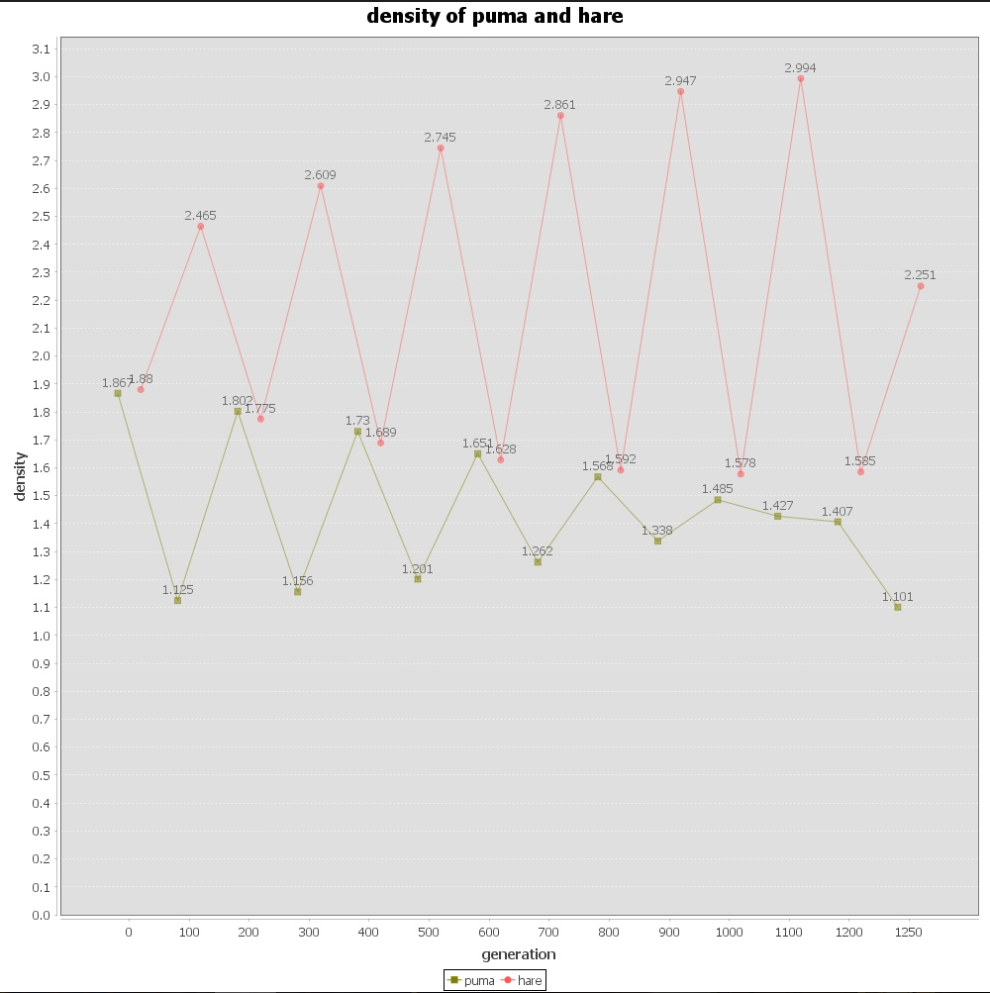


Fig.11 Average density line chart