Predator Prey Model

Names Here

November 2011

Abstract

Contents

Chapter 1

Introduction

In this task we worked as a group to implement a 2D, sequential predator-prey algorithm with spatial diffusion in Java. We tried to design the program in such a way as to take advantage of Java's object-orientated, modular nature. Due to the large size of our group we also decided to develop a GUI for the program, although the code was designed in such a way as to be usable from the command line as well. ——is it?——

There is always some compromise between readability and performance in coding, and we tried to keep this balance in mind when designing our code. We made best use of class structure whilst still keeping our implementation of the given algorithm as efficient as possible.

The predator prey algorithm implemented in this project crudely models the interaction between population densities of different animals, specifically Hares and Pumas. Each population has a self-interaction coefficient (birth for Hares, death for Pumas) and a coefficient describing how it interacts with other species' populations.

These populations exist on a 'world' consisting of land and water, and are also able to diffuse across land squares with a rate determined by a diffusion coeffient. Thus, each population has N+1 coefficients which determine it's behaviour, where N is the number of animals.

Chapter 2

Model

$H_{ij}^{new} = H_{ij}^{old} + \Delta t(C_1 H_{ij}^{old} + C_2 H_{ij}^{old} P_{ij}^{old} + l(H_{i+1j}^{old} + H_{i-1j}^{old} + H_{ij+1}^{old} + H_{ij-1}^{old} - H_{ij}^{old}))$

for k=1:NumberOfAnimals
if (k=ThisAnimal) then
$N_k^{new} = N_k^{old} + \Delta t C_k(k) N_k^{old}$
else
$N_k^{new} = N_k^{old} + \Delta t C_k(k) N_k^{old} N_{ThisAnimal}^{old}$
end if
end

where

$$C_{Hare} = \begin{pmatrix} H & PH \end{pmatrix}$$

Chapter 3

Design & Implementation

3.1 Code

3.1.1 Structure

3.1.2 Algorithm

3.1.3 Input/Output

3.1.4 GUI

3.2 Tools

3.2.1 SVN

3.2.2 Makefile

3.2.3 Unit Testing

Every class of the project form a functional unit according to the design elaborated at the beginning of the
has been tested using the Java JUnit framework
The libraries necessary to the tests are thus accessed from the test cases using Java import statements and extending the TestCase superclass in the class declaration:

Listing 3.1: Test case headers

```
1 import org.junit.Before;
2 import org.junit.Test;
3 import junit.framework.TestCase;
```

```
4
5 public class TestAnimal extends TestCase {
6 ...
```

Then the test cases, for each class, follow the normal directives in use for Unit testing with JUnit. The most relevant methods were submitted to thorough tests, where their properties, return values and correct functioning were evaluated by comparing the actual results they provide with the expected ones. These procedures were implemented by test methods in each test case using the format required by the JUnit standards.

Every test case included a setup() method that builds the initial object and its different parameters. Directives like @Before and @Test for every method declaration were thus inserted in the code were appropriate. The functioning of a method is then assessed comparing the expected results with the actual ones with JUnit testing methods such as AssertEquals(), AssertNotNull(), etc.

Listing 3.2: Use of JUNit directives in test cases

```
1 @Before
2 public void setUp() {
3     ...
4     testAnimal = new Animal(numbAnimals);
5     testAnimal.setName("Puma");
6     testAnimal.setDiffCo(diffCoIn);
7     ...
8 @Test
9 public void testAnimal() {
10    assertNotNull(testAnimal);
11    assertNotNull(testAnimal2);
12 }
13 ...
```

Chapter 4

Performance Analysis

4.1 Testing

4.2 Analysis

Chapter 5

Conclusions

Chapter 6

Group Evaluation