

# Predator Prey Model

Names Here

November 2011

## Abstract

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Model</b>	<b>3</b>
<b>3</b>	<b>Design &amp; Implementation</b>	<b>4</b>
3.1	Code . . . . .	4
3.1.1	Structure . . . . .	4
3.1.2	Algorithm . . . . .	4
3.1.3	Input/Output . . . . .	4
3.1.4	GUI . . . . .	4
3.2	Tools . . . . .	4
3.2.1	SVN . . . . .	4
3.2.2	Makefile . . . . .	4
3.2.3	Unit Testing . . . . .	4
<b>4</b>	<b>Performance Analysis</b>	<b>5</b>
4.1	Testing . . . . .	5
4.2	Analysis . . . . .	5
<b>5</b>	<b>Conclusions</b>	<b>7</b>
<b>6</b>	<b>Group Evaluation</b>	<b>8</b>

# 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 coefficient. Thus, each population has  $N+1$  coefficients which determine its 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

## Chapter 4

# Performance Analysis

### 4.1 Testing

### 4.2 Analysis

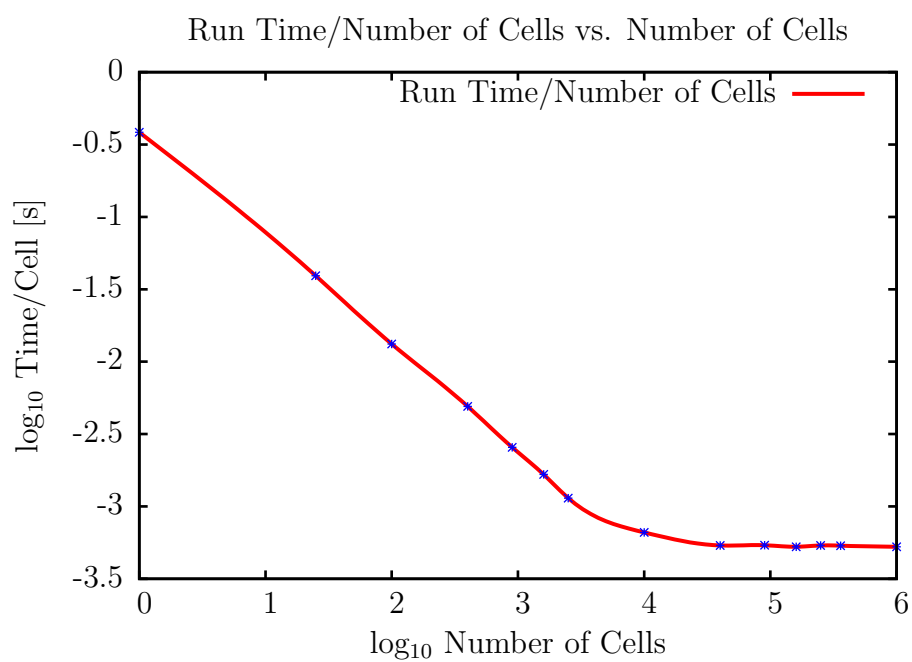


Figure 4.1: With greater than  $\sim 10000$  cells the run time scales linearly with the number of cells, as expected. Below this, overhead begins to dominate.



# Chapter 5

## Conclusions

## Chapter 6

### Group Evaluation