University of Hamburg

Project Thesis

Prey-Predator-Simulator

Author:
Eugen Betke

Supervisor:
Prof. Dr. Thomas Ludwig
Julian Kunkel

August 24, 2013

Contents

Abstract

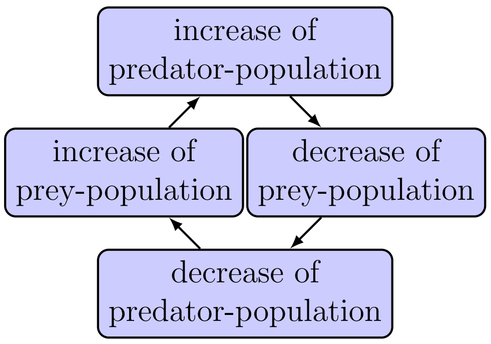
Simulators have found a wide application in science. They have established in almost all fields of research and help to answer to some of the open key questions. They are often used to simulate problems were other methods have been proven to be too complicated in some situations. Another strength of simulations is that the additional complexity can often be easily integrated in the simulated model.

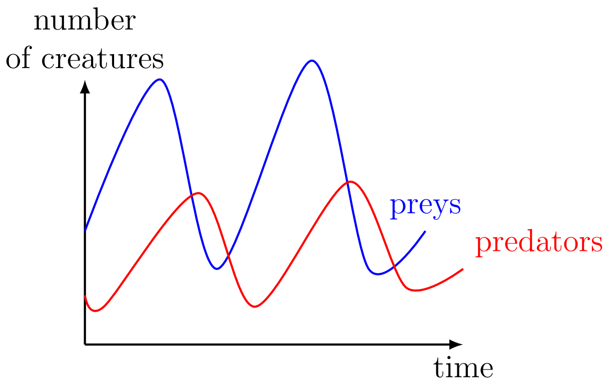
The main objective of this project is to develop a multi-core capable simulator for the prey-predator model. The simulator shall have the ability to visualize the world, to show statistics about the population development and allow users to control the most important parameters. The simulator doesn't comply with a special prey-predator model, but implements the model, that is developed in this project.

The programming languages used in this project are C and C++. The project benefits from open source software, particularly gtkmm for graphical user interface and pthreads for the implementation of the multi-core capable simulation.

1 Introduction

The prey-predator model shows the growth of two interdependent populations. The interdependency arises because the preys serve as food for the predators and because of this interdependency the growth of one population is influenced by the size of the other population. For example the simulation of such a model can end up in a kind of vicious circle as illustrated in the Fig. 1(a): (1) decrease of prey-population causes (2) decrease of predator-population causes (3) increase of prey-population causes (4) increase of predator-population causes (1)...(2)...(3)... Typically this ends in an endless fluctuation like sketched in Fig. 1(b). An other outcome of the simulation can be that all predators die off. It is also possible that all creatures die.


increase of predator-population
increase of prey-population
decrease of prey-population
decrease of predator-population
(a) Vicious circle



(b) Fluctuation populations

Figure 1: Prey-predator model

Such simulations show that the populations in the real world can change naturally even when humans do not interfere. Social scientists also use similar simulations to study human behaviour and community dynamics.

2 Definition of the project

The primary goal of this project is the implementation of a prey-predator model in a multi-core capable software simulator. The simulator should have the following components:

1. Simulation core
2. Visualization of the world
3. Statistics
4. User interface

Simulation core is the most important component of the simulator. It implements the model and is the only multi-core capable part of the simulator. The second component is the visualisation of the world, so that a user can easier observe and understand the events. The third component deals with statistics about the population in the world. It also shows the history of the simulation. Finally there is a user interface, where the user can change the properties of the objects and the simulation. The components that are not multi-core capable can be switched on and off, so that the simulation core can use as much processor as possible.

3 Modeling

This section gives an overview over the main concepts of the prey-predator model used by this simulator.

3.1 World

The world is represented as a two dimensional hex grid of fields (Fig. 3.1). The main advantage of such a representation is that each field has the same distance to all of its neighbours. This makes the simulation a little bit closer to the real world. Another property of this world is that nobody can enter or leave it, as if the world is surrounded by a wall.

3.2 Field

Fields are the main building blocks of the world. In a hex grid they can have two to six neighbours, depending on their position, i.e. the inner fields have always six neighbours (Fig. 3.4), the fields in the corners have two or three neighbours and the field on the border between the corners have four neighbours.

Every field contains one instance of a plant and can contain several preys and predators to the same time. The intensity of the green colour shows how many plants are available on the field. A white coloured field means, that there are no plants on the field and full green colour means, that the maximal amount of plants is available and no further plants can grow. The number on the top of the cell shows how many preys are on the field and the number below shows how many predators are on the field.

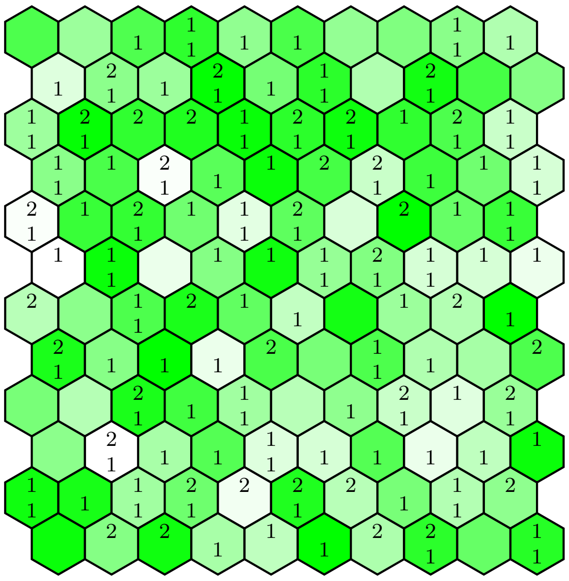


Figure 2: Example of a world. Intensity of green indicates how much plants are on the field. The number on the top of a field shows how many preys and the number on the bottom shows how many predators are on the field.

3.3 Living organisms

Before going further we should clarify some terms.

Organisms Organisms is a generic term for plants, preys and predators.

Creatures Creatures is a generic term for preys and predators.

Energy All organisms in the world have some quantity of energy, which is represented as a positive natural number and limited by an upper bound. Depending on the environment conditions, the energy can increase and decrease. How or when this happens is discussed later in detail (see further chapters). Energy models the nutritional value of the body, which is often measured in kcal in the real world.

Satiation Satiation is the opposite of hunger. Similar to the energy it is represented as a positive natural number and is also limited by an upper bound.

3.3.1 Plants

A plant is an organism that can not die, even if preys eat everything on the field the plant can still regenerate. We assume that plants get a constant amount of energy from the sun (not explicitly modelled) and therefore they are growing constantly during the simulation.

3.3.2 Creatures

In the simulation fat bodies would have a lot of energy and a slim body less energy. Young creatures are small and would have a relatively small upper bound, but the upper bound grows, when a creatures becomes older until some limit is reached. When creatures do not get enough food they become slimmer and slimmer, until they die of starvation. Satiation is in the first place a motivation for looking for food and it can change the behaviour of the creatures. Creatures die when satiation decreases to some level, even if they have a lot of energy. Analogy: In the real world, we can feed an animal until it becomes fat. When suddenly stopped being fed, the animal dies in about a week, but it is still fat.

Typically, preys move to a field that contains more plants. They don't move, if they find on the current field more plants than on the adjacent fields. This behaviour changes drastically, if a prey meets a predator. In this case it tries to escape in a random adjacent field. It doesn't matter how many plants there are.

The preys have a small visibility range and are not able to see predators in the adjacent fields. Therefore it can happen that they move to a field where a hungry predator is waiting for them. This is a bad situation for a prey, but it doesn't always mean the end. It can survive, if other preys are located on the field and are eaten by the predators first. Preys can also survive, if the predators are not hungry. In this case they don't touch the preys.

As already mentioned, a predator doesn't kill preys, if it is not hungry, even if it finds preys on the same field. A predator has the ability to see, if preys are on the adjacent fields. If it is hungry and doesn't find food on the current field, it moves to the adjacent field were a prey is located. If a predator is hungry and it finds no preys, it moves to a random adjacent field.

3.4 Concept of the food chain

As already mentioned, each field contains a plant. They all together serve as a food source for the preys. Preys, in turn, serve as food source for the predators. Thus we obtain a short food chain that is illustrated in Fig. 3.4.

$$\begin{tikzcd}
\boxed{\text{Plant}} & \boxed{\text{Prey}} \arrow[l] & \boxed{\text{Predator}} \arrow[l]
\end{tikzcd}$$

Figure 3: Food chain: predators eat preys and preys eat plants.

Let's take a closer look at the act of eating, that is relatively simple and is always following a common scheme. As described above, all organisms have some amount of energy (e.g. in the real world this energy could be measured in kcal). When a creature is eating, it takes a certain amount energy from its food and transfers it to his body.

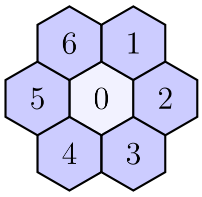


Figure 4: 6 Neighbours of a field 0.

3.5 Simulation

In the cycle-based simulation each field runs at the beginning of each cycle an instance of the process that is illustrated in Fig. 5. A cycle is finished, when all fields finish their processes, then a new cycle can be started. Unfortunately, the processes of different fields can influence each other. For example misbehaviour can occur when a process moves some preys to an adjacent field, where the movement operation is not executed. The adjacent field executes the movement operation and can move these preys again. In this way these preys can be moved twice. To avoid such misbehaviour some control structures are needed.

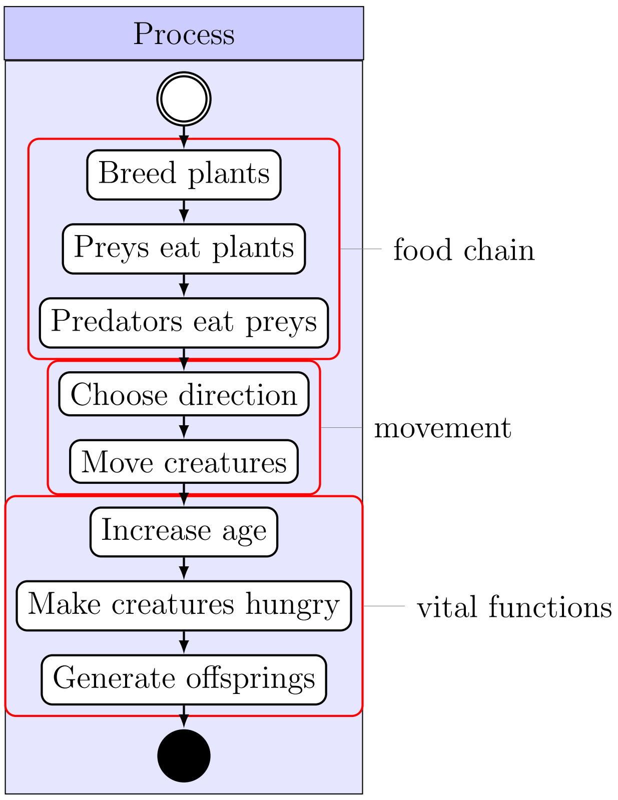


Figure 5: Simulation process

4 Implementation

4.1 Simulation Core

Simulation core is the most important part of the simulator. It creates an abstract representation of the world and determines the behaviour of creatures. This is also the only part that can not be disabled during the simulation and needs to take into account the whole world. Therefore it could be a good idea to apply some performance-enhancing methods.

To improve the responsiveness of the GUI, the GUI and the simulation core are executed in different threads. Simulation core in turn creates several sub-threads, that do the work in parallel and increase the performance of the simulator.

4.1.1 Internal representation of the world

The hex grid of the world uses a somewhat strange and non-intuitive coordinate system (Fig. 6). The fields in a column are one below the other, but there is a lot of space between them. The fields in a row seems to be arranged not in a single, but in two rows. Nevertheless, the grid has a well defined grid structure and each field can be uniquely identified. This makes this coordinate system suitable for our purposes. We will later see the advantage of this system, when we distribute the fields among the threads.

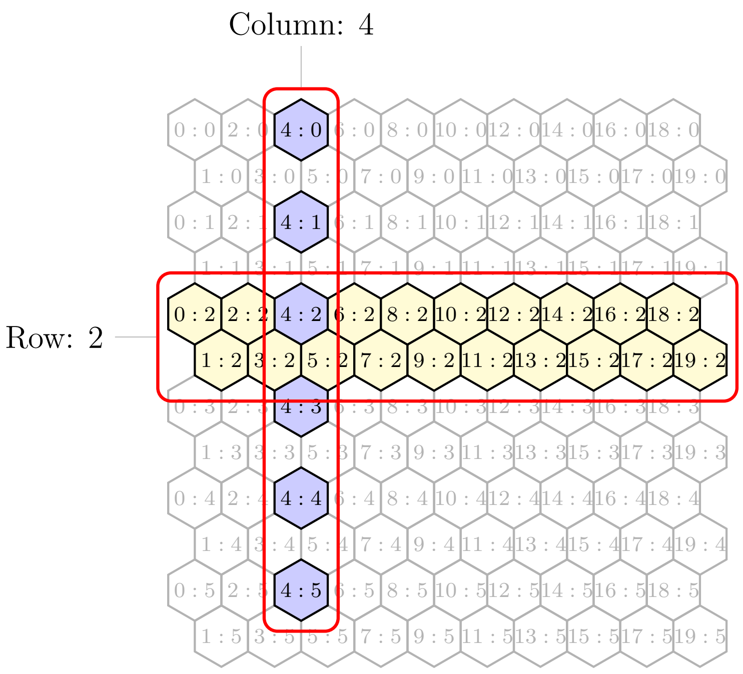


Figure 6: Coordinate system of hex grid

4.1.2 Phase model

The simulation process described in the previous chapter is used by the simulation core in a modified way (Fig. 7). The operations of the process are combined to phases, whereby a phase needs to be completed by all threads, before the execution of the next phase can be started. Such a subdivision allows a synchronisation of the processes on different fields and helps to avoid the misbehaviour of the simulation.

The model has some additional operations that are executed in the “phase 0”. These operations were integrated for convenience purposes. They make it possible to change the configuration of organisms at runtime. The price for this convenience is a small overhead on each simulation cycle. The overhead we get compared to the whole process is small and can be neglected.

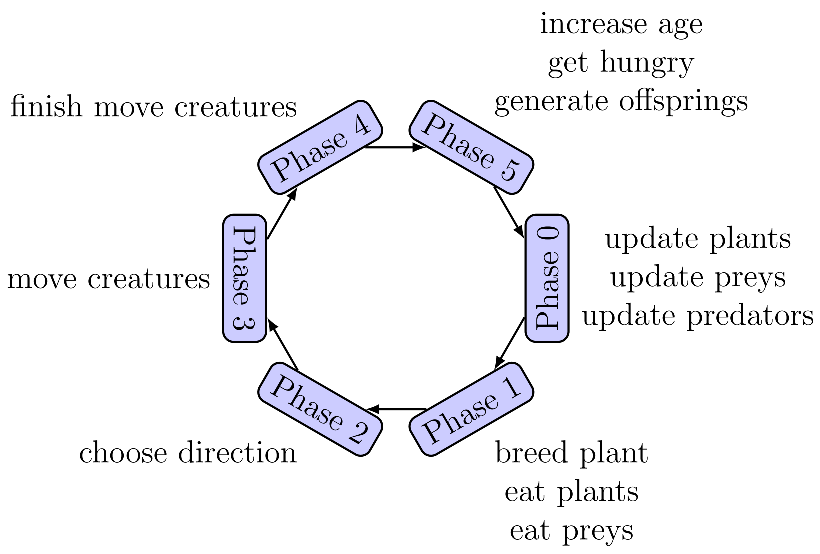


Figure 7: Phase model.

As already mentioned the simulation core is a performance critical part of the simulator and therefore it was made with multi-core processing in mind. Here it means that each phase is splitted over several threads. Each thread in turn is assigned to a number of consecutive columns. Fig. 8 shows how a 20×5 hex grid is divided among four threads. In this case each thread gets 5 columns. In general, an algorithm computes for each thread an interval and tries to distribute the columns uniformly as far as possible. Of course this works best with slim columns, such as they were defined in the previous section. By the way, this is also an explanation why this strange coordinate system was chosen.

Let’s take a look at the thread functions. They all have a common structure, that is shown in the pseudo-code in the Listing 1. There are dependencies between the cycles, therefore the Each thread processes a part of abstract world.

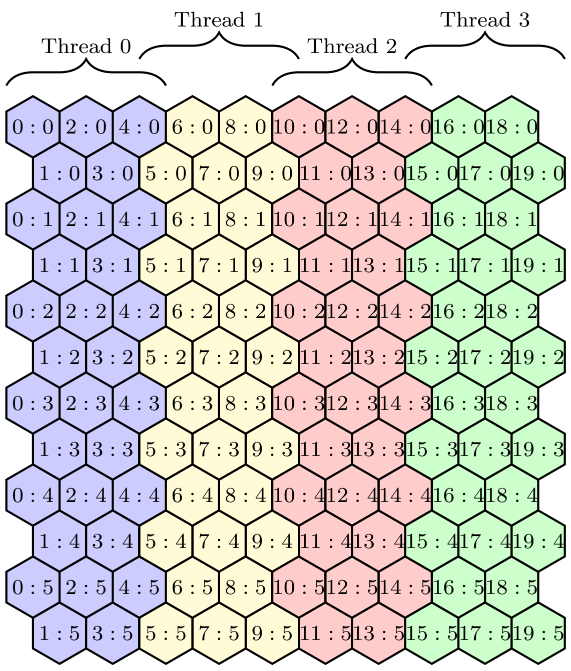


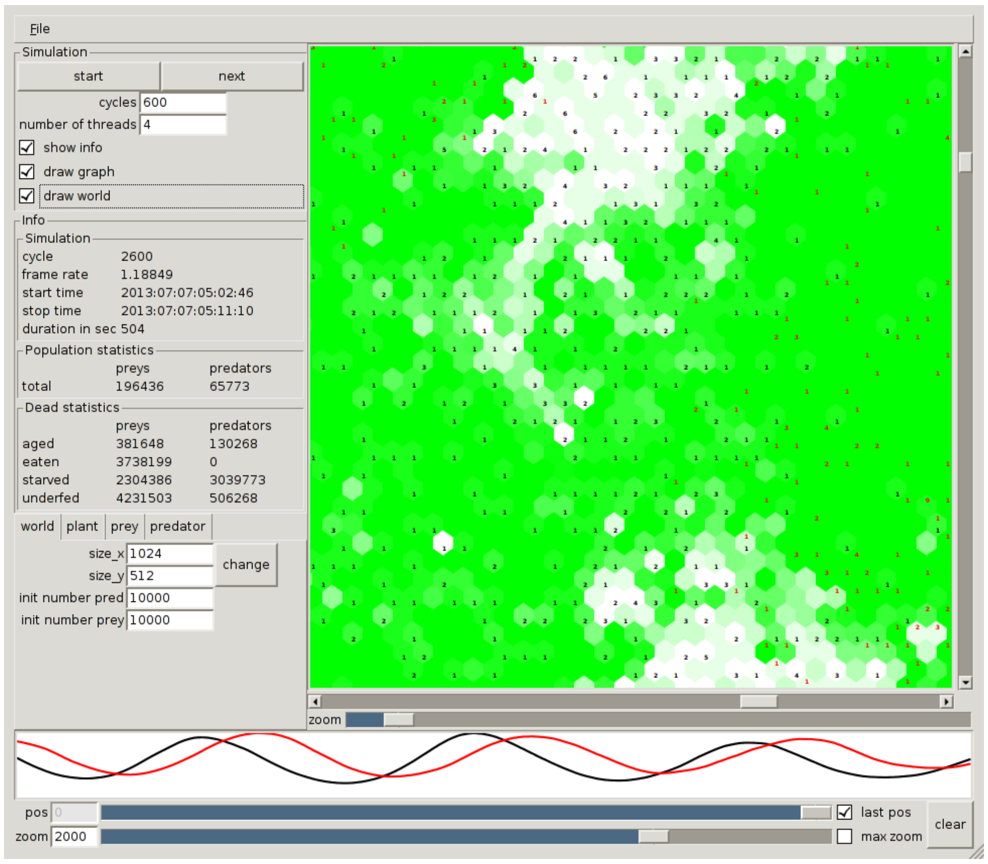
Figure 8: Thread intervals

```
void* CAbstractWorld::phase_x(void* param) {
    SThreadParam* tp = (SThreadParam*) param;
    CAbstractWorld* aw = tp->abstract_world;
    SInterval interval = aw->calc_interval(tp->thread_id);

    for (int x = interval.a; x <= interval.b; x++) {
        for (int y = 0; y < aw->get_size_y(); y++) {
            aw->get_field(x, y)->some_operation();
        }
    }

    pthread_mutex_lock(&aw->mutex_x);
    aw->counter_x++;
    if (aw->counter_x == aw->num_of_threads) {
        aw->counter_x = 0;
        pthread_cond_broadcast(&aw->cond_var_x);
    }
    else {
        while (pthread_cond_wait(&aw->cond_var_x, &aw->mutex_x) != 0);
    }
    pthread_mutex_unlock(&aw->mutex_x);
}
```

Listing 1: Thread function

4.2 User interface

The GUI (Fig. 9) is written in gtkmm [1]. It is divided in to three parts. On the left side you find control/info panel. The visualization of the world is placed right to the panel and at the bottom you find a history of the population development.



```
cycles 600
number of threads 4
[✓] show info
[✓] draw graph
[✓] draw world
```

```
Info
Simulation
cycle            2600
frame rate       1.18849
start time       2013:07:07:05:02:46
stop time        2013:07:07:05:11:10
duration in sec  504
```

```
Population statistics
          preys      predators
total     196436     65773
```

```
Dead statistics
          preys      predators
aged      381648     130268
eaten     3738199    0
starved   2304386    3039773
underfed  4231503    506268
```

```
size_x 1024
size_y 512
init number pred 10000
init number prey 10000
```

```
pos 0
zoom 2000
[✓] last pos
[ ] max zoom
```

Figure 9: GUI of the prey-predators simulator.

When the simulator is closed it saves many parameters and settings in an INI file, so that when the simulator is started again, it restores almost completely the state of the previous run. This behaviour improves the usability the program enormously and was achieved with the help of the iniParser library [2].

4.2.1 Control/Info panel

On the top of the control/info panel are control elements for simulation. There you can define the size of the simulation run and number of threads and start and stop the simulation. The GUI checks if the the values are valid and sets them automatically to the nearest possible value, if they are not already. There you can also activate and deactivate the serial components of the simulator. These are the info panel, the visualization of the world and and the visualization of the graph. But even when the visualization of the graph is deactivated, the statistics will still be recorded.

The info block below show information about the current population of the world. Below you find some death counters that register four kinds of death: aging, kill, underfed and starvation.

The simulation itself can be customized in the block of the bottom of the control/info panel. Most of the settings there can be changed at runtime / during the simulation. The setting that are not runtime capable are disable after the start button is pressed. The block itself consists of four parts: world, plants, preys and predators. In the world tab you can define the size of the world and the initial number of the creatures. In the plants, preys and predators tabs you can define the reproduction and vital functions.

4.2.2 World

The visualization of the world is very similar to the model, described in the previous section. Therefore a detailed description will be omitted.

4.2.3 Statistics

The history of the population development is shown at the bottom of the GUI. In the drawing area the development of the populations is visualized by the two lines. The red line represents the predator-population and the black line prey-population. Both lines are automatically scaled, so that you can better see the fluctuations of the populations. If you need the exact values, look for the export feature, that is described later in this section.

You can scroll through the history with the two scales below the drawing area. With the pos-scale you decide from which cycle the graph should be drawn. Right to the pos-scale you find and entry, where you can enter the precise value. Left to the scale you find a check box. When activated, the graph will automatically move forward and show you the last values.

Below the pos-scale you find the zoom-scale. By moving this scale you can determine the range that shall be displayed. Left to the zoom-scale you find an entry, where you can enter the precise zoom value. Right to the zoom-scale you find a check box. When activated, the whole history will be drawn in the drawing area and automatically update, when the simulation is running.

Statistics can also be exported to the a CSV file, which is widely used and can be imported by various programs, i.e. Libreoffice. This file contains exact values of population on each cycle and corresponding death statistics. An example of a CSV file is given in the listing 2. The first line contains column names and the following lines are filled with data. To export the data click on “File → Export to CSV”.

```
preys total;preys aged;preys killed;preys starved;preys underfed;preds total↩
    ;preds aged;preds killed;preds straved;preds underfed
2884;2;17;5;17;356;0;0;0;0
2894;3;38;11;36;362;0;0;2;1
2916;7;50;15;47;367;0;0;3;2
2937;10;63;23;62;376;0;0;3;3
2970;12;76;30;75;380;1;0;3;3
3007;14;87;38;89;387;3;0;4;3
3027;17;113;46;99;394;4;0;4;4
3039;22;129;54;113;392;5;0;6;4
3053;24;151;61;124;402;5;0;6;5
3093;25;172;71;137;406;6;0;7;6
3108;28;193;82;149;408;7;0;8;6
3123;33;205;94;162;415;7;0;8;6
```

Listing 2: An example of a CSV file.

5 Experiments

During the experiments only world parameters and number of threads were changed. These are the most interesting parameters for this project, because they have direct and comprehensive impact on the performance. Although the change of plants, preys and predators parameters can also significantly influence the performance, they were kept constant. It is very difficult to predict how exactly performance is influenced. The main purpose of these parameters is to control the behaviour of the populations. All relevant parameters are shown in the figure 10.

Parameter	Range
size x	128 – 1024
size y	32 – 256
init number pred	100 – 10000
init number prey	100 – 10000

(a) World

Parameter	Value
max energy	900
growth rate	0.1

(b) Plants

Parameter	Value
birth rate	0.04
energy consumption	200
satiation consumption	100
max age	1000
max satiation	1000
max energy	1000

(c) Preys

Parameter	Value
birth rate	0.04
energy consumption	100
satiation consumption	30
max age	1000
max satiation	1000
max energy	1000

(d) Predators

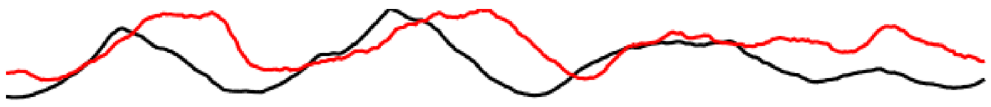
Figure 10: Simulation parameters.

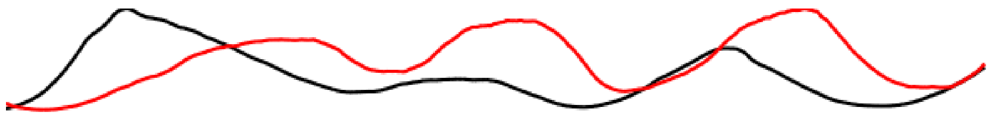
5.1 Model

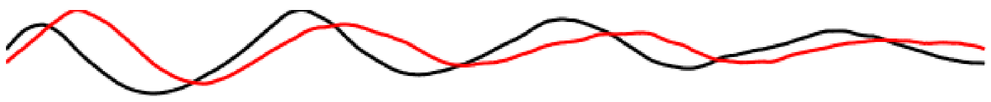
This section takes a closer look at the implementation of the model. The results of the experiments should be compared with the theory of prey-predator model.

5.1.1 Dependency between populations

As already discussed in the introduction, there is an dependency between two population. The special and maybe the most interesting case of fluctuating populations could be verified by several simulation runs. Some of them are shown in the Fig. 11. The lines are running similar to the lines shown in the Fig. 1(b). That also means that the populations behave in the same manner as was described by the viscous circle.


(a) 128 × 64 grid


(b) 256 × 128 grid


(c) 512 × 256 grid

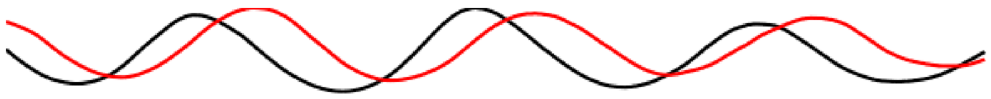

(d) 1024 × 512 grid

Figure 11: Fluctuation of populations during 2000 simulation cycles.

The conclusion of these simulation runs may be the following: if preys and predators live on the same area and neither of the populations die off, then both populations are fluctuating. The other cases where both populations die off or only preys survive can be easily induced and reproduced, but they are out of interest and will not be examined further.

5.1.2 Large vs. small worlds

The size of the world is also an important criteria for survivability of populations. In worlds smaller than 128 × 64 fields it is difficult to find parameters, that keep both populations alive. This problem becomes less important in large worlds.

5.2 Performance

Another interesting aspect of the simulator is the performance. To measure the performance we've run different simulations on our testing system (Tab. 1) under different configurations and noted the results. The intention of this performance test is find out how performance increases when we run a simulation on several processor cores. We also guess that the performance can be strongly affected by the grid size. Therefore we ran some simulation with different grid sizes. The simulation parameters stayed the same for all tests. They are listed in the table 10(a).

The most important components of our testing system are listed in the table 1. The simulations were run on Arch-Linux with 3.9.6-1-ARCH kernel. Besides we've ensured that the hyper-threading technology was disabled during the simulation runs.

Component	Manufacturer	Model
CPU	Intel	Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz
Mainboard	Supermicro	X9SAE
Graphic adapter	Nvidia	NVS300
RAM	Hynix	HMT351U6CFR8C-PB ($4 \times 4GB$)

Table 1: Testing system.

5.2.1 Data

As discussed in the previous section we run simulation with a different number of threads. Precisely we are interested in results produced by simulations with 1, 2, 3 and 4 threads. Simulations with more than 4 threads would assign several threads to one core and therefore they are out of interest, because we don't expect a significant increase of performance. We also run simulation on different grid sizes. From our point of view worlds simulations with worlds smaller than 128×32 are not really interesting. There is not enough room for population and they can easily die. Therefore we begin to simulate a 128×32 world and double the axis in the next simulations. We've also decided to keep the number of processed fields constant. That means when we double the axis we quarter the number of cycles. We start with 32000 cycles in the 128×32, then we run 8000 cycles on the 256×64 world, 2000 cycles on the 512×128 world and 500 cycles on the 1024×256 world. The results of the test runs are summarized in the table 5.2.1.

5.2.2 Acceleration

In an ideal case the performance would rise linearly to the number of threads, but unfortunately our implementation has some drawbacks that prevent it from running optimal. There are two main reasons for the performance losses. Firstly the simulator runs serial code during the simulation. For example each time a cycle is completed serial code needs to be executed to gather some statistics. Secondly to avoid misbehaviour of the simulation the threads must be synchronized. We have already discussed that without synchronization prey and predators could jump several times

	1024×256 grid 500 cycles		512×128 grid 2000 cycles		256×64 grid 8000 cycles		128×32 grid 32000 cycles	
Threads	Duration in [sec]	Accel.	Duration in [sec]	Accel.	Duration in [sec]	Accel.	Duration in [sec	Accel.
1	678	1.00	660	1.00	646	1.00	632	1.00
2	346	1.96	340	1.94	345	1.87	340	1.86
3	239	2.84	241	2.74	246	2.63	243	2.60
4	192	3.53	192	3.44	199	3.25	195	3.24

in one cycle. Synchronization implies waiting times for threads that have finished their job first.

The results of test runs are visualized in the Fig. 12. We can see there that the acceleration factor becomes better with the increasing world size. This property seems to be independent from the number of threads. This can be easily explained by the fact, that the size of the serial code is constant after each cycle and when we increase world size, we also increase the run length of threads. In this way total execution time of serial code is reduced. The same applies to the synchronization overhead and waiting times are also reduced.

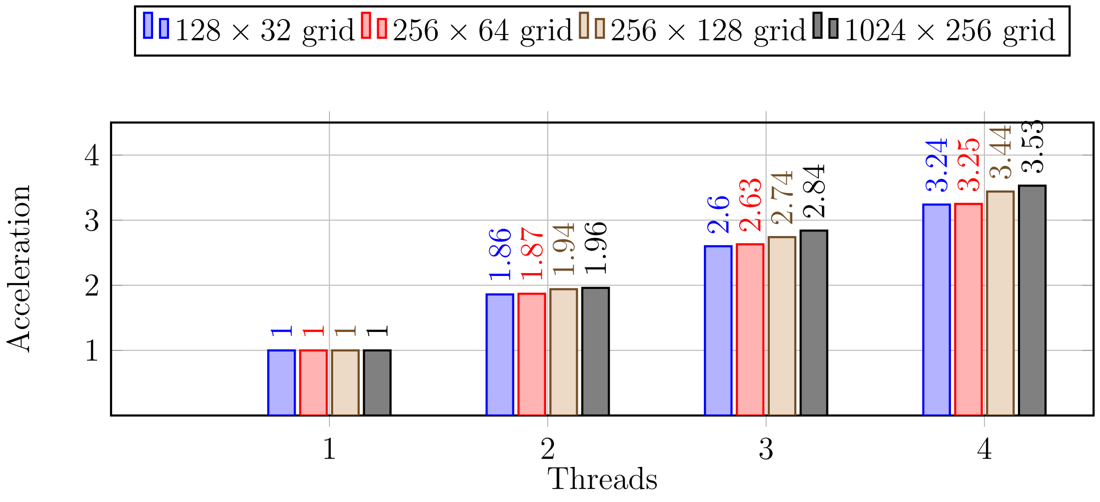


Figure 12: Acceleration

5.2.3 Efficency

In the last section we saw, that the performance increases, when we increased the number of threads. Now we take a look how efficient the simulator uses the resources of the system. We assume that efficiency is maximal when we run the simulator only with one thread. In such a configuration no synchronisation between threads is necessary and no thread will wait for other threads.

In Fig. 13 the data was reordered so that we can see how the number of threads accelerates the simulation. This property seems to be independent from the size of the world. Utilisation of the processor seems to be higher when we run the simulation on bigger worlds. Nevertheless the efficiency decreases when we increase the number of threads. We can directly see this in the Fig. 14. The percentages shows how many computing resources the simulator uses compared to the ideal case.

6 Summary

In this project a prey-predator model was implemented with multi-processing in mind. It consists of four main parts: control/info panel, visualization of the world,

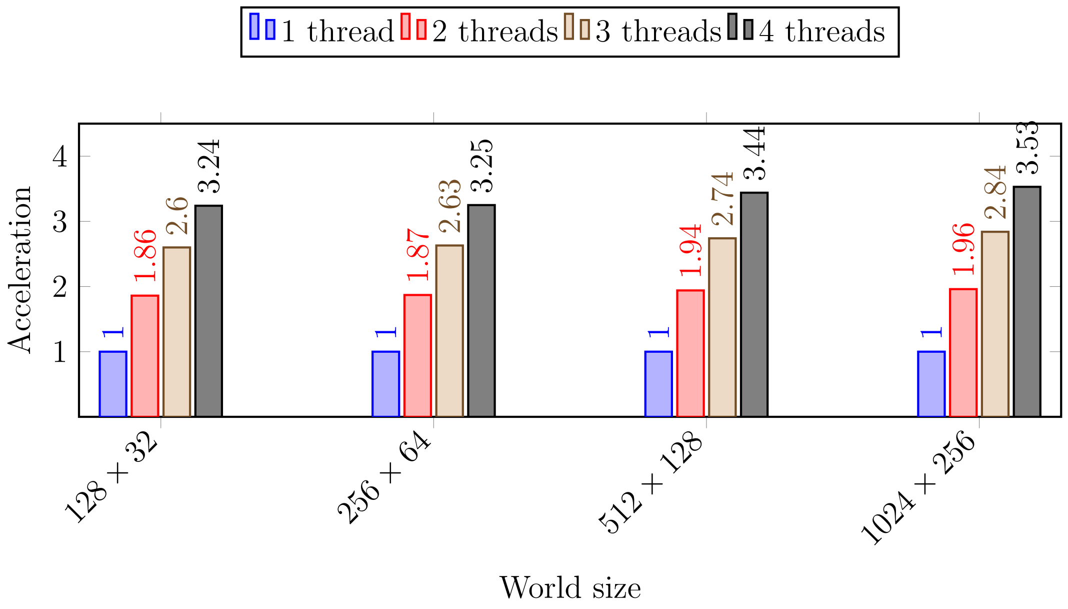


Figure 13: Acceleration of simulation, ordered by size of the world.

visualization of the population history and simulation core, whereby the simulation core is the only multi threading capable part. Other parts can be disabled during the simulation to reduce the execution of serial code.

The results show that the implementation works best with large worlds and with a number of threads equal to the number of processor cores. (The situation where the number of threads exceeds the number of cores was not tested.)

References

[1] E. Cumming, "gtkmm," Aug. 2013.

[2] N. Devillard, "iniparser: stand-alone ini parser library in ansi c," Apr. 2012.

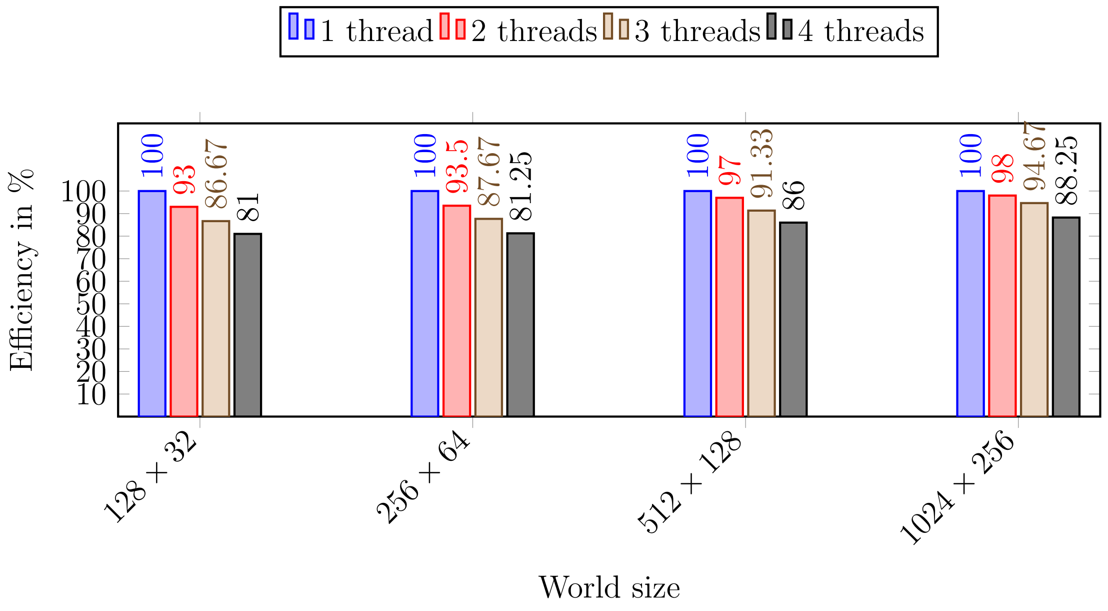


Figure 14: Efficiency.