SIMULATION DESIGN

The environment in our simulation is a two-dimensional array of cells modelled against the progression of time.

Time is expressed as ‘generations’ which represent a discrete step in time.

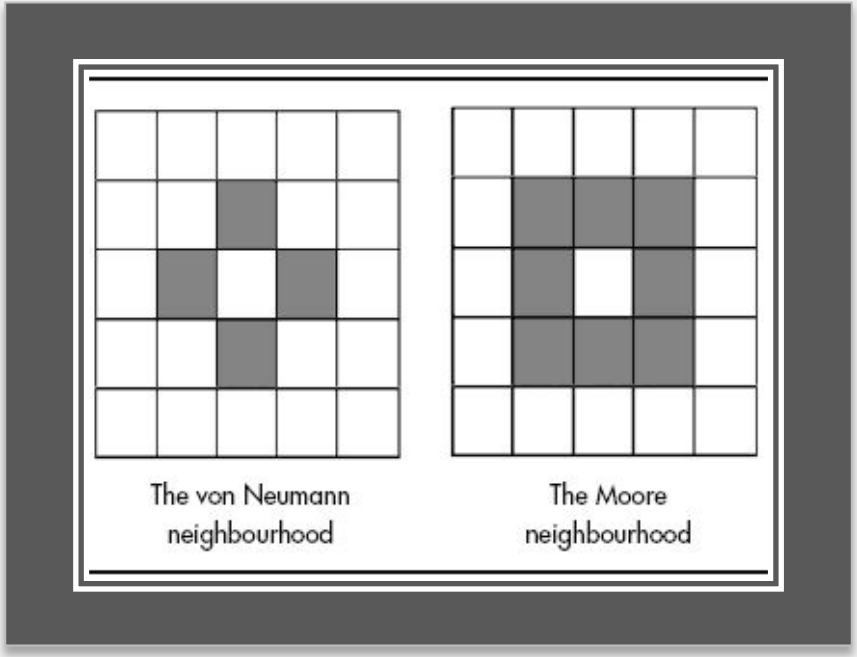
In each new generation, the state of every cell in the environment is updated according to a set of rules.

Each cell is square in shape.

SIMULATION DESIGN

Each cell has a Moore neighbourhood consisting of its immediate cell neighbours.

Each cell can have one of three states: empty (Black), prey (green) and predator (red).



ASSUMPTIONS

All predators are identical to one another, there are no differences between individuals.

All prey are identical to one another, there are no differences between individuals.

Prey can only die to predation , there are no other forms of death for them.

Predators can only die of starvation, there are no other forms of death for them.

ASSUMPTIONS

It is assumed that prey have an unlimited supply of food and do not suffer from any other variables which may kill them (disease, accidents, other predators etc). This means that the prey population should grow exponentially in the absence of predators and fill up all available space. This falls in line with the Lotka-Volterra model.

The predator and prey exist in a vacuum, no other entities are present in this system.

Every cycle is the same and executes the same set of rules. Time-related cycles such as day/night do not exist.

When predators eat prey, prey change to an empty space which the predator then fills with another predator automaton.

Attempt-1 Rules

Prey (1)	**Predator (2)**	**Blank (0)**
if (predNeighbors > 0) next[x][y] = 2; **//Eaten** else next[x][y] = board[x][y]; **//Stasis**	if (preyNeighbors < 1) next[x][y] = 0; **//Starvation** else next[x][y] = board[x][y]; **//Stasis**	if (predNeighbors > 0 || preyNeighbors == 0) next[x][y] = 0; **//Remain Blank** else if (preyNeighbors > 0) next[x][y] = 1; **//Breed prey**

Attempt-1 Result

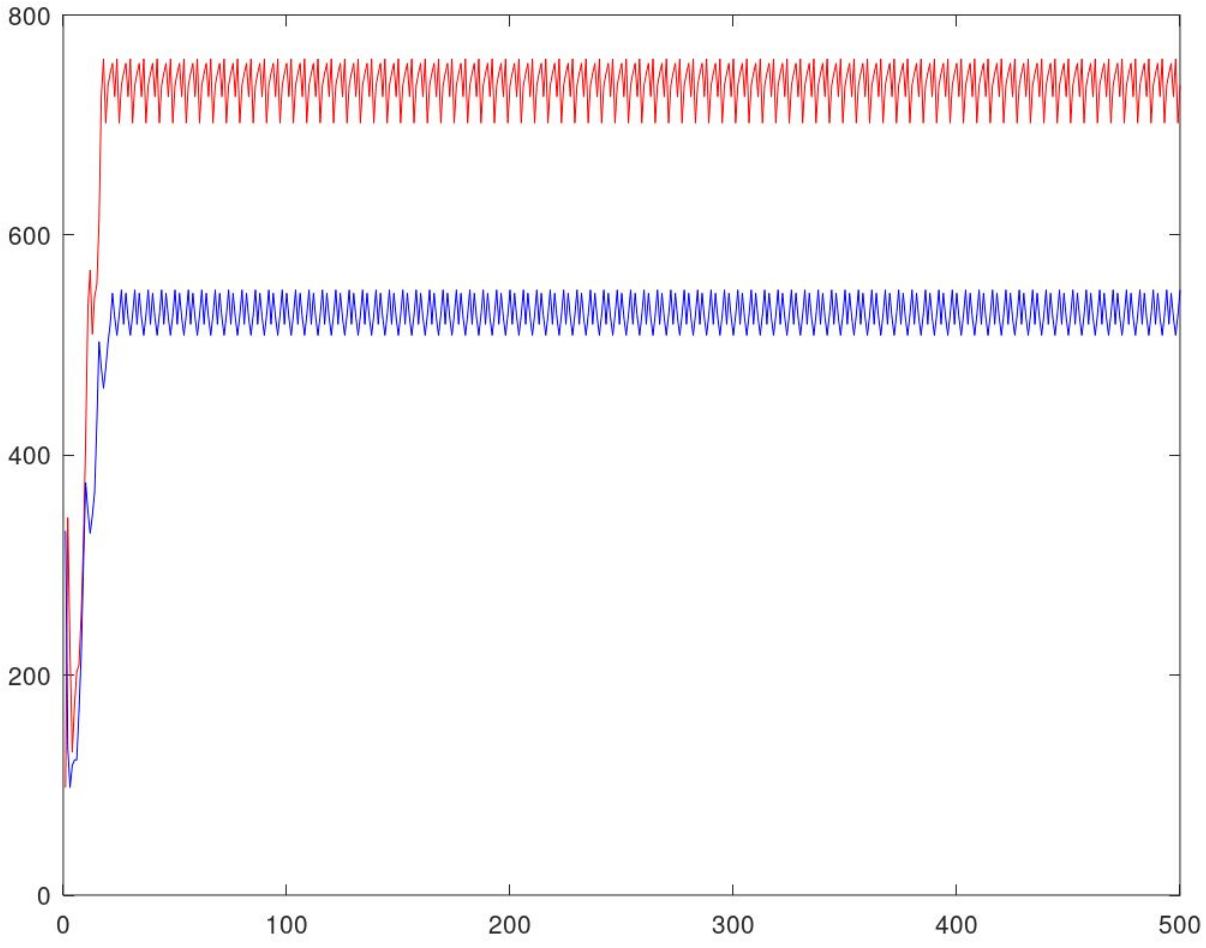
giving the prey some form of detection and evasion or

giving the predators a limit to their hunger so they will not hunt the prey to extinction

Attempt-1 Result

giving the prey some form of detection and evasion or

giving the predators a limit to their hunger so they will not hunt the prey to extinction



Changes made

Added a stochastic Monte Carlo picker method to determine the probability that a predator successfully catches its prey. Using random function, chance of predator catching prey is implied. It represents the a in aNP of Lotka-Volterra model. it provided a way for prey to escape the predators.

```
float r = random(0,1);

if (r > catchRate) rtn = 1;

    else rtn = 2;
```

Attempt-2 Rules

Prey (1)	**Predator (2)**	**Blank (0)**
if (predNeighbors > 0) next[x][y] = eatEscapePicker(); **//Hunted** else next[x][y] = board[x][y]; **//Stasis**	if (preyNeighbors >0) next[x][y] = 0; **//moving** else if (preyNeighbors < 1) next[x][y] = 0; **//Starvation** else next[x][y] = board[x][y]; **//Stasis**	if (predNeighbors >= 2 && preyNeighbors >= 1) next[x][y] = 2; **//Breed pred** else if (preyNeighbors >= 2) next[x][y] = 1; **//Breed prey** else next[x][y] = 0; **//Statis**

Attempt-2
Result

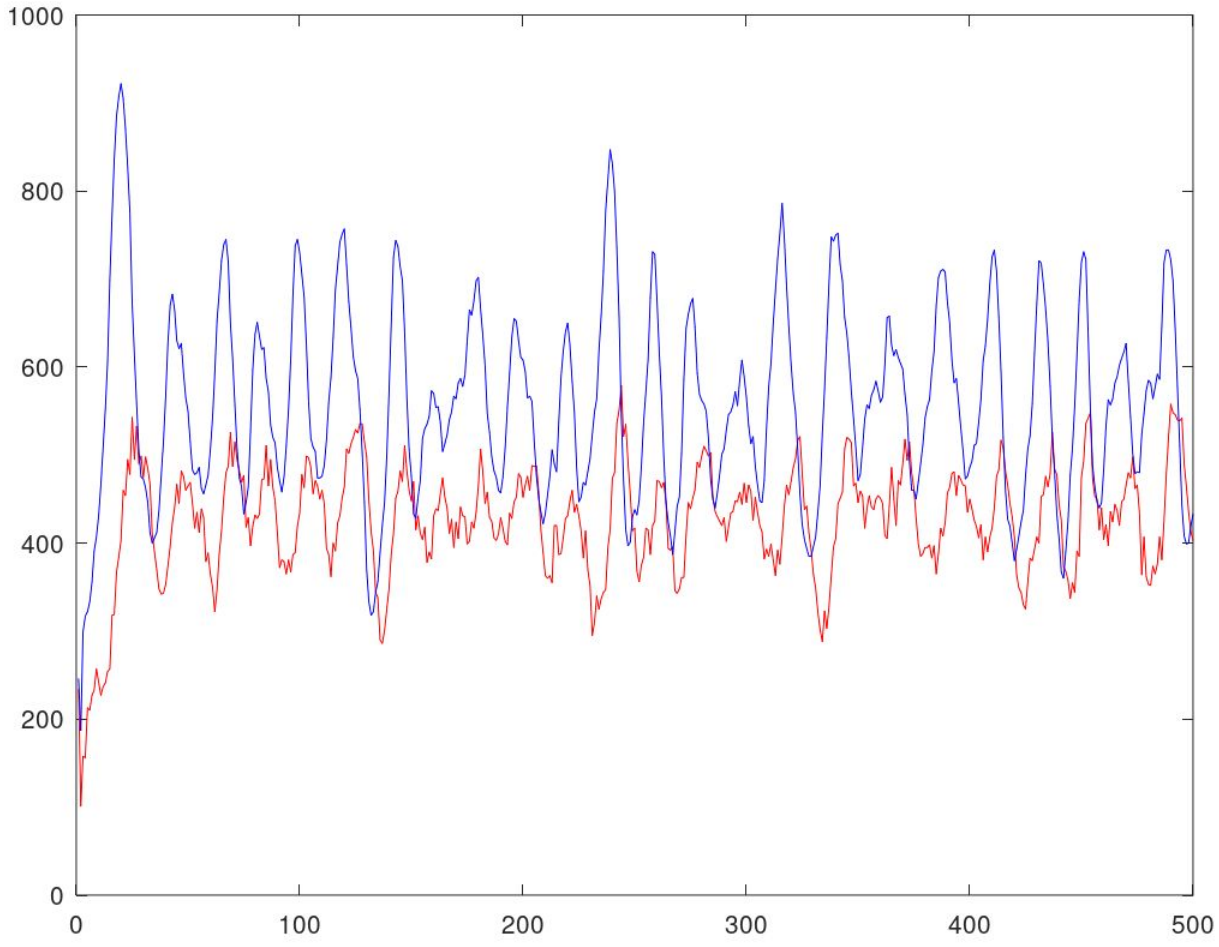
presents the accurate
representation of the Lotka-Volterra
model produced by this simulation,

Catch rate - 0.33

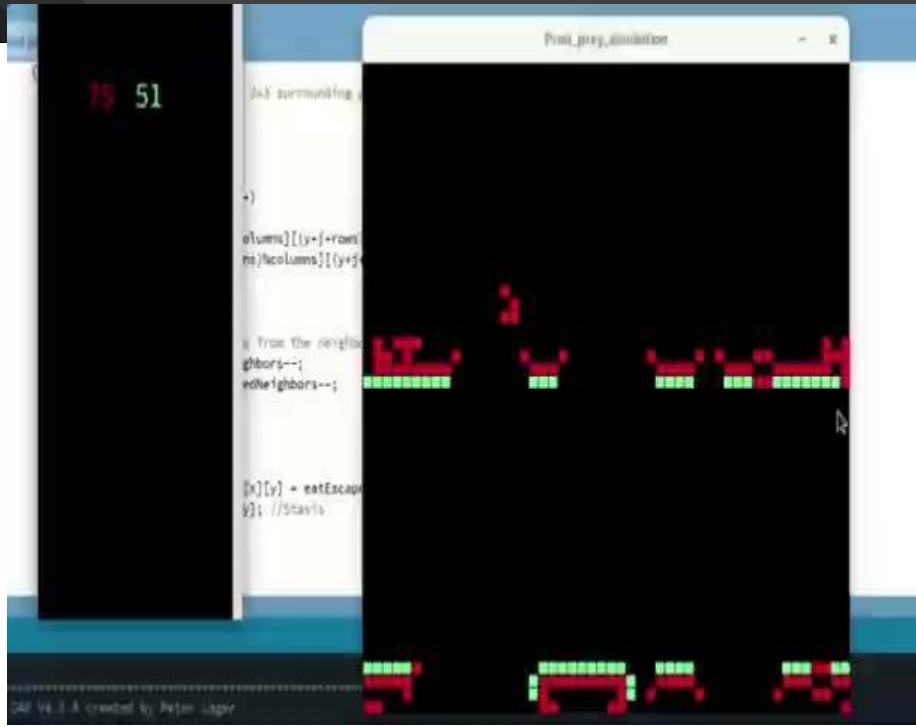
Attempt-2 Result

presents the accurate representation of the Lotka-Volterra model produced by this simulation,

Catch rate - 0.33



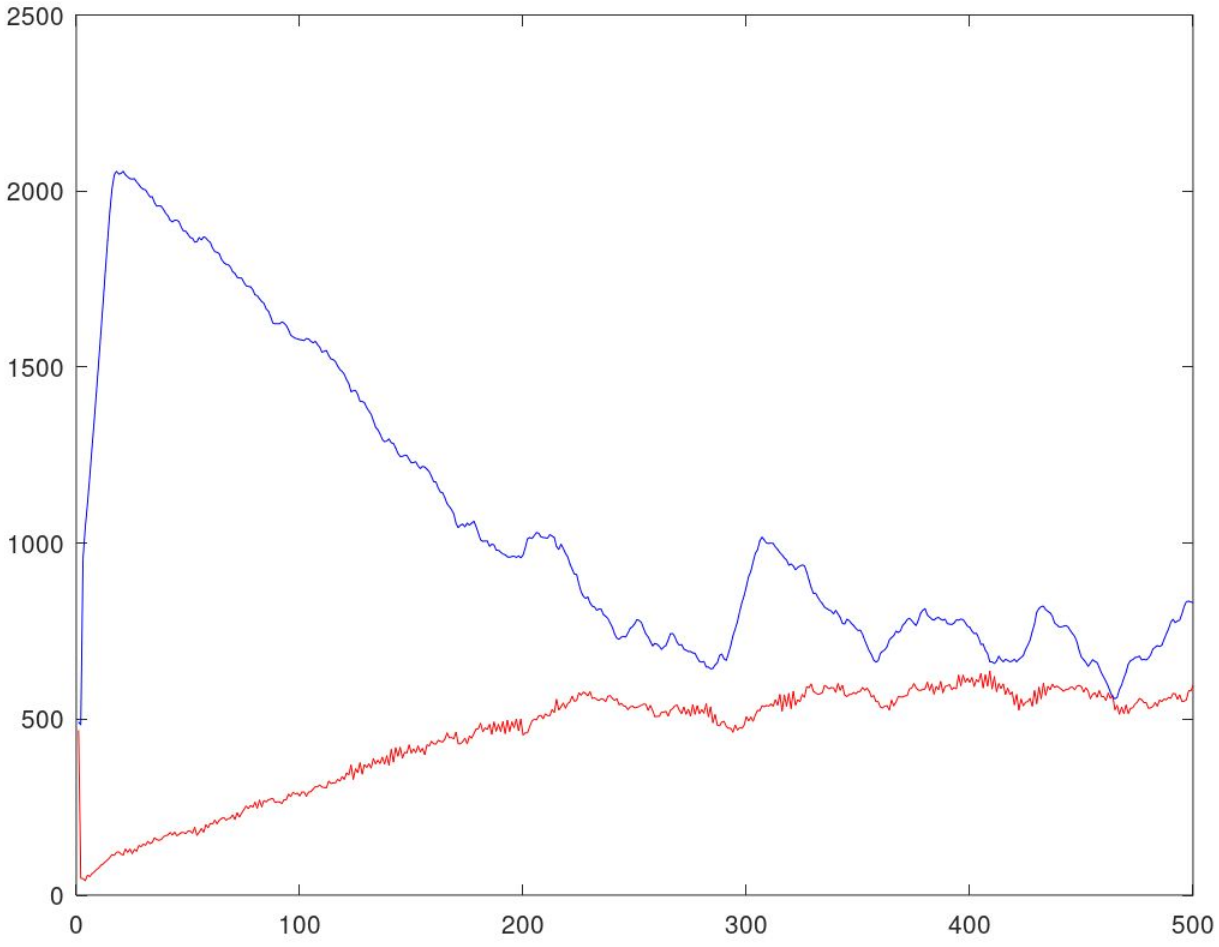
Other Results



Result

With very low catch rate, prey can escape easily.

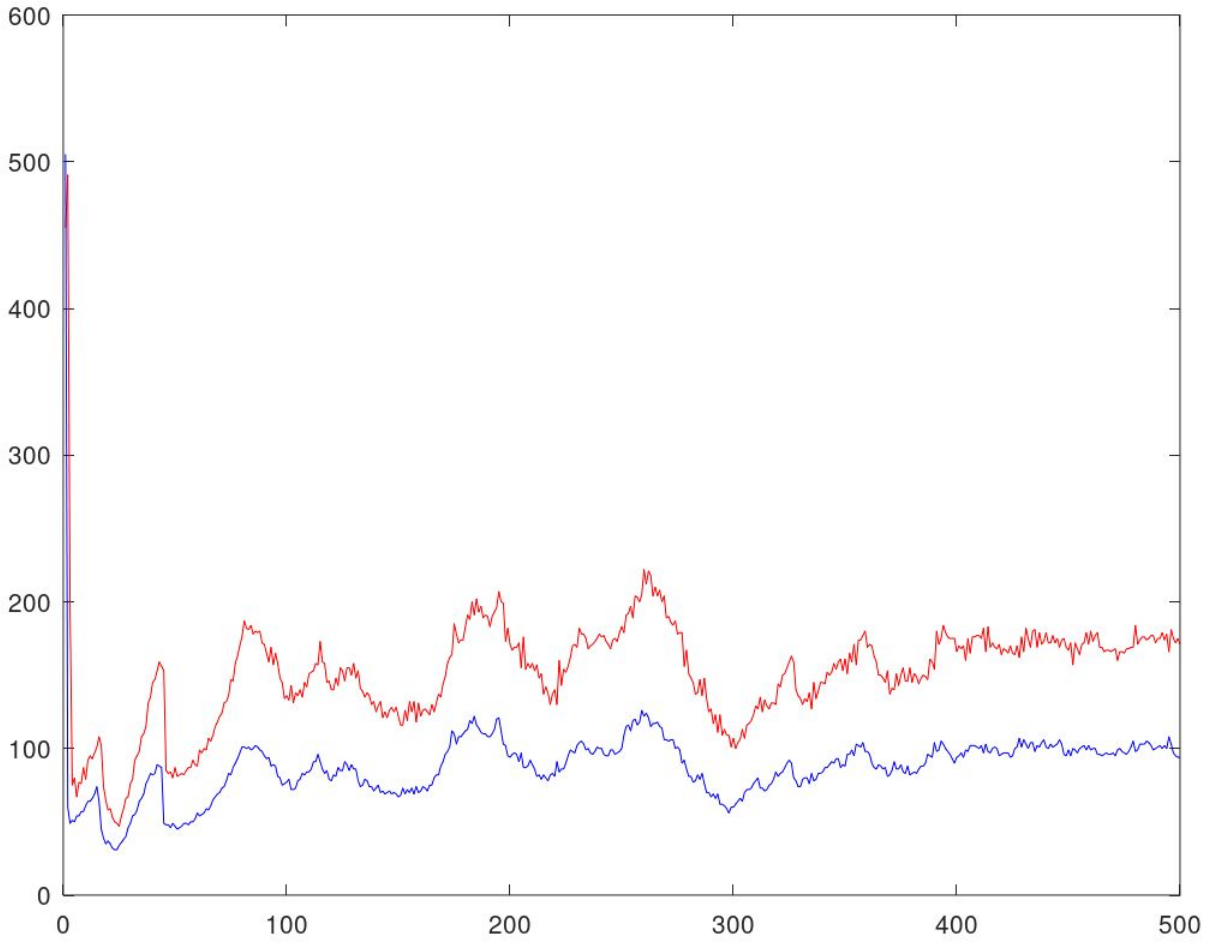
Catch rate - 0.05

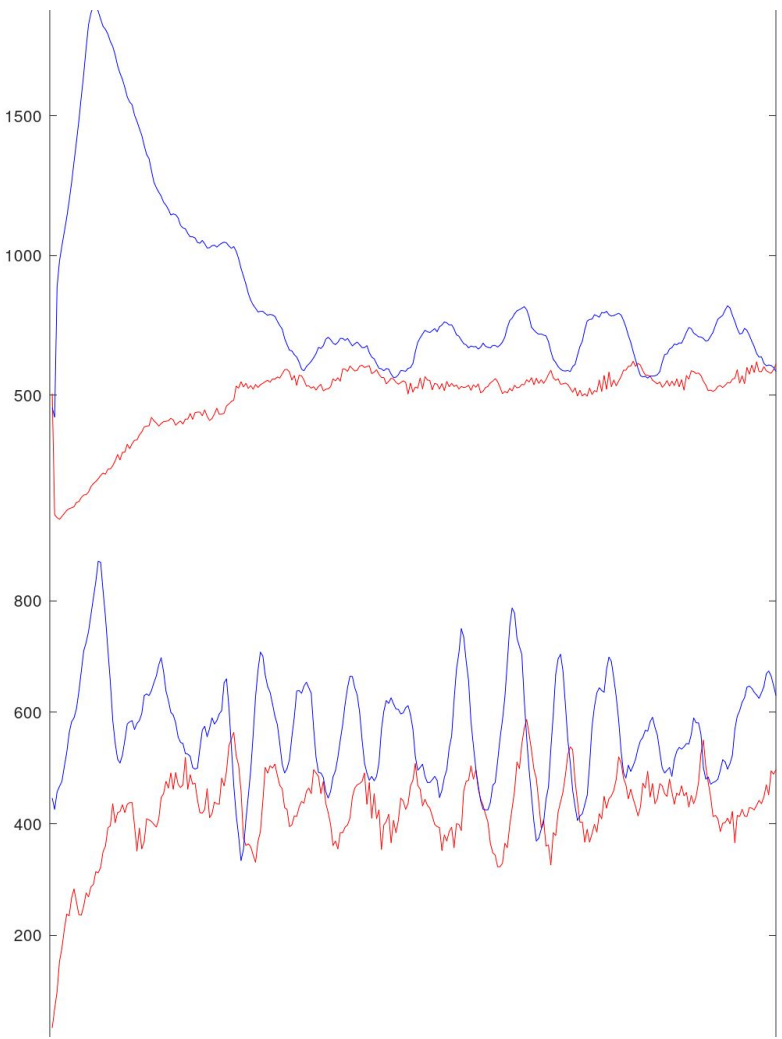


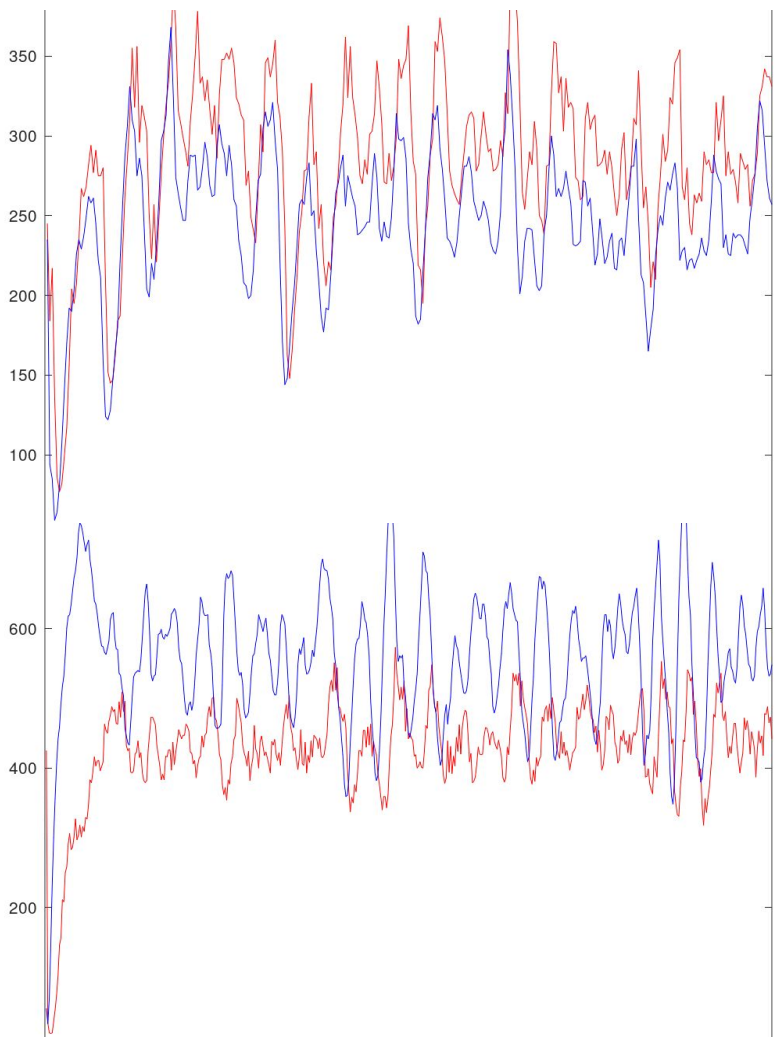
Result

With very high catch rate, prey can escape easily.

Catch rate - 0.95




1500
1000
500
800
600
400
200


350
300
250
200
150
100
600
400
200

Features of Simulation

Prey population is typically higher than the predator population.

The system is stable.

The system exhibits oscillation.

The prey curve leads the predator curve.

The predator population tracks the peaks of the prey population.

Elementary cellular automaton pattern

one-dimensional array of cells, each of which holds a single binary value, either 0 or 1.

The automaton is given an initial configuration, and then progresses through other configurations in a sequence of discrete time steps.

At each step, all cells are updated simultaneously.

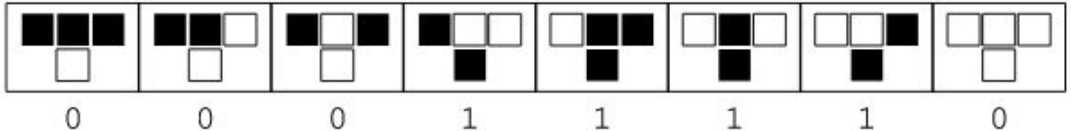
A pre-specified rule determines the new value of each cell as a function of its previous value and of the values in its two neighboring cells.

current automaton contents

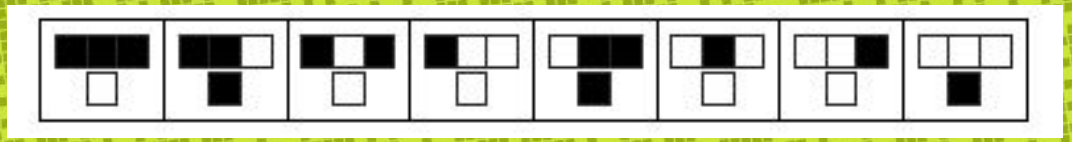
rule 30 (00011110)

the next generation of the automaton

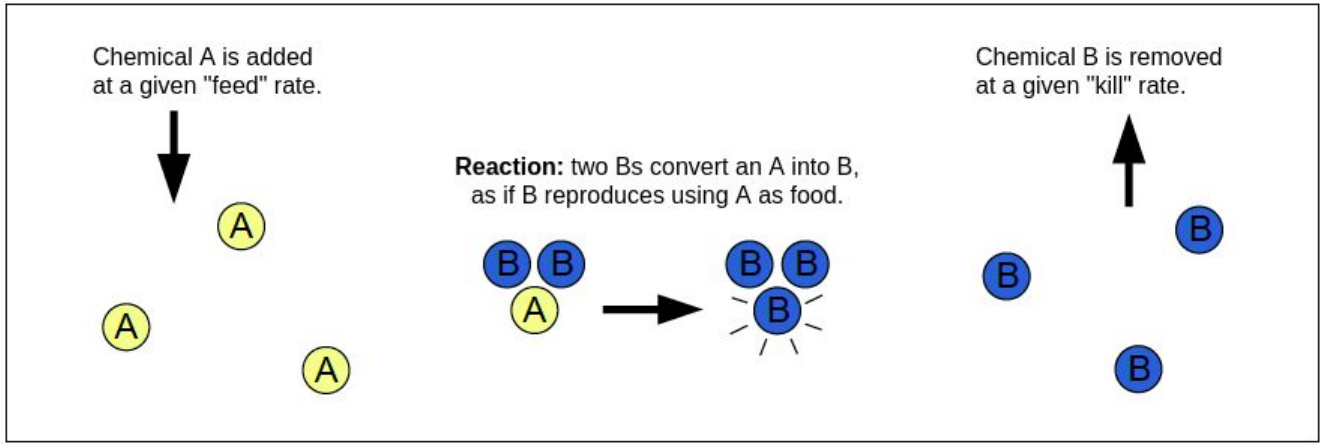
Rule - 30
00011110 = 30



Rule - 73



Reaction-Diffusion



Gray Scot model

In this models, we have two chemicals A and B which are distributed across a grid of size N. The symbol A_{ij} represents the concentration of chemical A at grid coordinates (i, j) (similar for B).

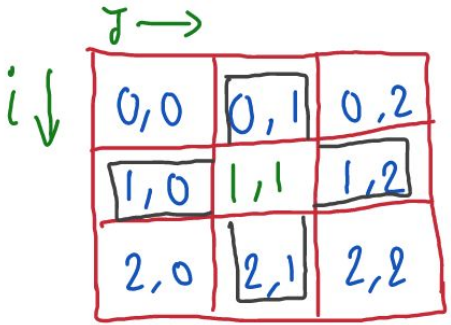
The discrete reaction-diffusion update equations are

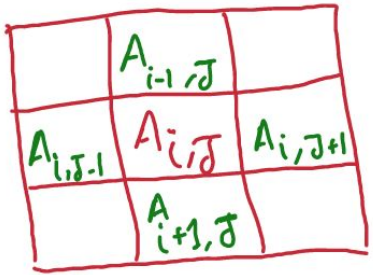
$$A_{ij}(t+1) = A_{ij}(t) + \left[D_A(\nabla^2 A)_{ij} - A_{ij}B_{ij}^2 + f(1 - A_{ij})\right] \times \Delta t$$

$$B_{ij}(t+1) = B_{ij}(t) + \left[D_B(\nabla^2 B)_{ij} + A_{ij}B_{ij}^2 - (k+f)B_{ij}\right] \times \Delta t$$

Discrete Laplacian

$$(\nabla^2 A)_{i,j} = A_{i,j-1} + A_{i,j+1} + A_{i-1,j} + A_{i+1,j} - 4A_{i,j}$$





$\triangle = 1$
grid difference

Discrete Laplacian

```
def discrete_laplacian(M):

L = -4*M
L += np.roll(M, (0,-1), (0,1)) #right neighbor
L += np.roll(M, (0,+1), (0,1)) #left neighbor
L += np.roll(M, (-1,0), (0,1)) #top neighbor
L += np.roll(M, (+1,0), (0,1)) #bottom neighbor

return L
```

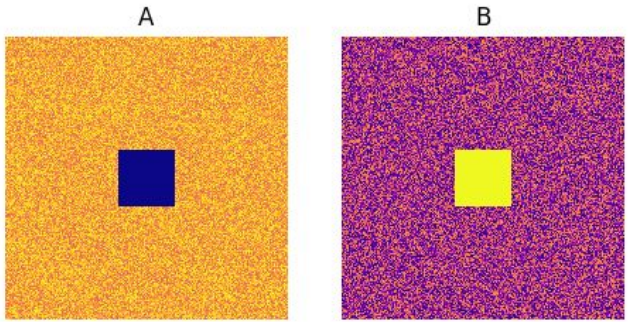
numpy.roll shifts all elements in a certain direction,
with periodic boundary conditions.

Initial conditions

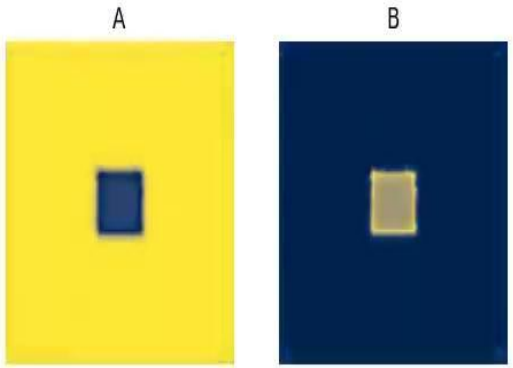
We start with a configuration where on every grid cell, there's a lot of chemical A, so the concentration is high.

Let's assume there's only a bit of B everywhere.

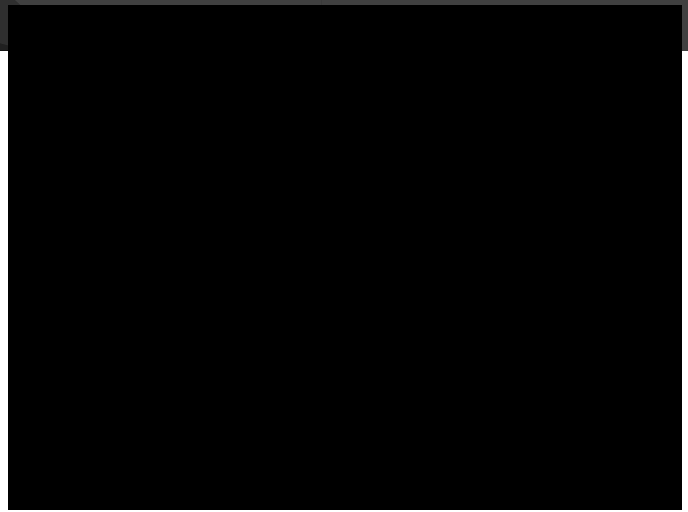
We disturb with a square in the center of the grid.



Patterns

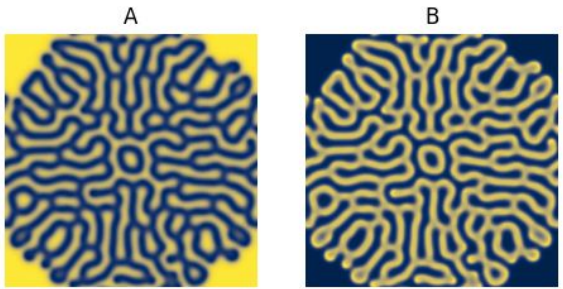


DA, DB,f, k = 0.16, 0.08, 0.060,0.062

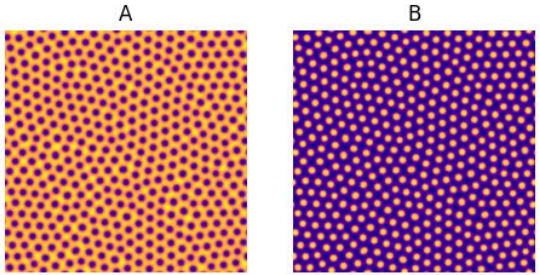


DA, DB,f , k= 0.14, 0.06, 0.035, 0.065

Patterns

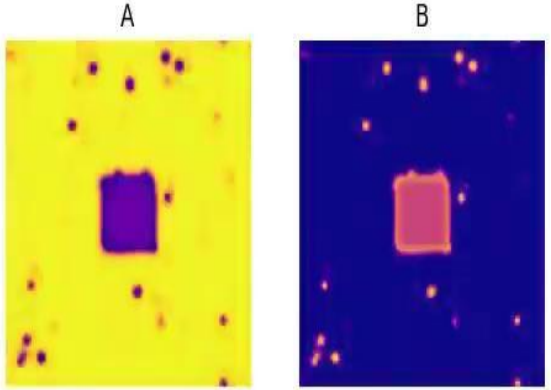


DA, DB,f, k = 0.16, 0.08, 0.060,0.062



DA, DB,f , k= 0.14, 0.06, 0.035, 0.065

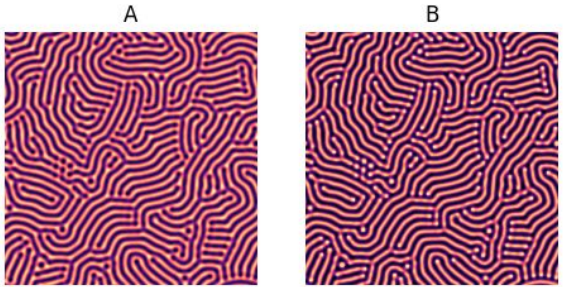
Patterns



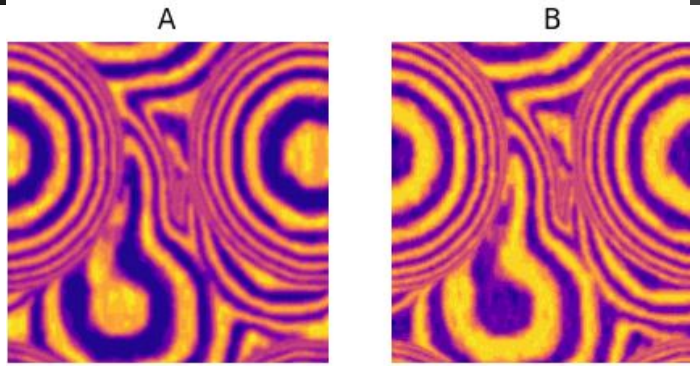
DA, DB, f, k = 0.15, 0.034, 0.035, 0.057

DA, DB, f , k= 0.16, 0.09, 0.050, 0.055

Patterns



DA, DB, f, k = 0.15, 0.034, 0.035, 0.057



DA, DB, f , k= 0.16, 0.09, 0.050, 0.055