Excercise 1.
Implementing a first Application in RePast: A Rabbits Grass Simulation.

Group №: 75 - Thomas Kimble, Jules Afresne

October 1, 2019

1 Implementation

1.1 Assumptions

We work in a square grid that acts as a torus with a size defined by the user (*gridSize*).

Each cell can contain *grass* defined by an integer which can accumulate in a cell with growth. *Grass* grows randomly on the grid at *grassGrowthRate*, which is chosen by the user: $grassGrowthRate = n$ signifies that n grass will grow every step. The initial amount of grass on the grid is chosen by *numInitGrass*.

We assume that a *rabbit* is created with a random *energy* level between two chosen variables *minEnergy* and *maxEnergy*, decided by the user. The user also defines the amount of starting rabbits with the *numInitRabbits* parameter. Every step, each rabbit has four random movement directions (NESW movement). Two rabbits can not be on the same cell at once.

With each movement or step, a rabbit loses 1 energy, but when a rabbit lands on a cell with n grass it gains $5 \cdot n$ energy. Finally when a rabbits energy reaches *birthThreshold* (also defined by the user), it reproduces. In our case this means that another rabbit is created randomly on the grid with a an energy between *minEnergy* and *maxEnergy*, and the original *minEnergy* and rabbits energy is reset to between *minEnergy* and *maxEnergy*.

1.2 Implementation Remarks

We have two main boundary conditions: when a rabbit is on the edge of the grid moving outside, and when two rabbits are on a collision course in the same cell.

We use a torus map to avoid agents being lost after the edge of the grid. For this we use the following formula to allow a return to 0 if we go one move further than n, being the size of the grid, in any x or y direction (where % is the modulo operator): $newX = (newX + gridSize)\ \%\ gridSize$

To avoid collision with another rabbit when one is created or after a movement we first check whether the new cell is free. Here, a routine will return true if the rabbit is created, and false if not. If there are no cells left, the rabbit is not created and the routine will run $k = 10 \cdot gridSize$ times before giving up.

2 Results

2.1 Experiment 1: Life Cycles

2.1.1 Setting

We set the following parameters:

Grid size: 20
Number of initial rabbits: 5
Number of initial grass: 60
Grass growth rate: 1
Minimum start energy: 50
Maximum start energy: 100
Birth threshold: 175

2.1.2 Observations

We observe life cycles over time. The rabbits eat the grass to gain energy, reproduce, then the offspring eat grass to do the same. Grass levels decrease and the rabbits are left with no more food, they use up all of their energy and a large quantity proceed to die. Fewer rabbits gives more opportunity for grass growth which leads to higher quantities of grass and the cycle starts again.

We can clearly see this behaviour in Figure 1. Indeed as the amount of grass grows, the number of rabbits decreases, and the cycle repeats itself.

After leaving the experiment running for dozens of minutes we have observed that the cycle does not converge and keeps on repeating itself.

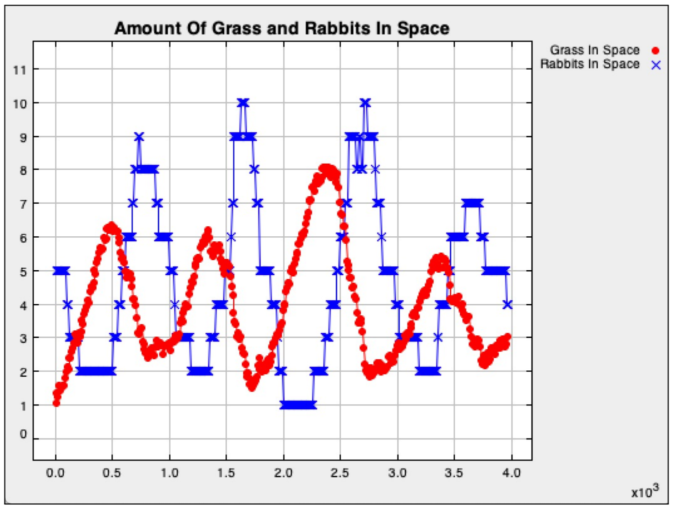


Figure 1: Blue: Number of living rabbits in the simulation space and Red: Amount of grass in the simulation space (scaled down to 2% of actual value for visualisation)

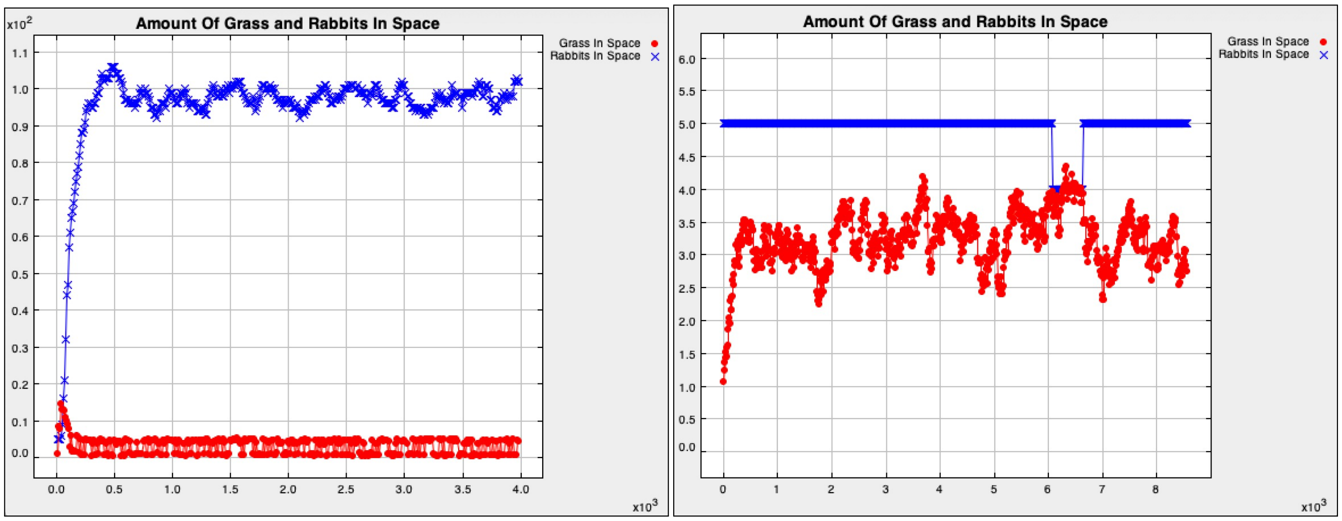
2.2 Experiment 2: Stability

2.2.1 Setting

After testing we have noticed two main types of stable situations: Grass Stable (Figure 2a) and Rabbit Stable (Figure 2b).

If we consider the life cycle parameters as the base, we can obtain a Grass Stable situation by raising the *grassGrowthRate* parameter (from 10 to 100 for example).

However a Rabbit Stable situation is obtained by raising the *minEnergy*, *minEnergy* and therefore the *birthThreshold* parameters (to 500, 1000 and 1500 respectively for example). Finally if we raise the *birthThreshold* parameter (to 2000 for example) agents survive but in no particularly stable way.



(a) Grass Stable Situation (b) Rabbit Stable Situation

Figure 2: Blue: Number of living rabbits in the simulation space and Red: Amount of grass in the simulation space (scaled down to 2% of actual value for visualisation)

2.2.2 Observations

In a Grass Stable situation, the amount of grass always stays constant, while the amount of rabbits is governed by the grass growth rate. Higher the growth rate, higher the amount of rabbits.

In a Rabbit Stable Situation the grass varies around a certain level, but the amount of rabbits in the space stays constant with time. The rabbits have more energy to start and take longer to reproduce, but the food is sufficient for them to survive. Even if a rabbit dies, more food enables another one to reproduce keeping the number stable. The opposite also applies.

2.3 Experiment 3: Extinction

2.3.1 Setting

Again, if we consider the life cycle parameters as the base, extinction is observed in a few situations. By lowering the *birthThreshold* to the *maxEnergy* limit, by reducing the *grassGrowthRate* parameter, by raising the initial amount of rabbits *numInitRabbits* or the initial amount of grass *numInitGrass*.

2.3.2 Observations

If we lower the textitbirthThreshold to the *maxEnergy* limit rabbits will reproduce very fast, overpopulating the space. All the grass is eaten and the rabbits have nothing left. We can compensate this by raising the *grassGrowthRate*.

However if we reduce the *grassGrowthRate*, the grass has no time to grow before it is eaten by the rabbits. They then starve, similarly to the previous situation.

Raising *numInitRabbits* or *numInitGrass* without touching bthe other parameters have a similar effect. With a high amount of starting rabbits, all the grass is eaten fast, energy is wasted and the rabbits die. With a high initial amount of grass, the rabbits have the opportunity to reproduce fast, overpopulating the space as explained previously.