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

Group №48: Boubacar Camara, Ivan Snozzi

September 29, 2020

## 1 Implementation

### 1.1 Assumptions

In addition to the initial assignement requirements, we added the following assumptions to our model:

- Rabbits are born with 20 points as initial energy.
- Rabbits energy decreases by one at each simulation tick.
- New rabbits are randomly placed in a cell which does not contain another rabbit.
- Rabbits try to move in a random direction ($UP, DOWN, LEFT, RIGHT$) at each simulation tick. If all cells accessible from valid directions are already occupied, the rabbit does not move.
- Rabbits reproduction costs them a predefined amount of energy.
- At any time of the simulation, there can be at most $GridSize * GridSize$ rabbits. Hence, a rabbit cannot reproduce if this threshold is reached.
- Each grid cell contains at most $maxGrassGrowth$ grass unit.
- At most $GrassGrowthRate$ grass units spread on the grid at each simulation tick.
- A cell is colored in gray if it contains a rabbit. Otherwise, the cell is brown if empty, and green if it contains grass. The green color changes linearly depending on the number of grass units in the cell. The lower is the number of grass units, the darker will be the green. Similarly the higher is the number of grass units the lighter will be the green.

### 1.2 Implementation Remarks

Our RePast implementation of the simulation is highly inspired from the tutorial referenced in the assignment description. Hence we will focus here on a few implementation details.

To randomly place a new rabbit in the grid, we first check that the number of living rabbits is below the number of grid cells. If not we abort the operation. Otherwise, we sample a random position $(x, y)$ and place the rabbit in the corresponding cell if it is not occupied by another rabbit. Otherwise we reiterate this process until we find a free cell. We execute at most $10 * GridSize * GridSize$ iterations and abort the operation if this threshold is reached.

We use a similar methodology to spread grass in the grid, with the only difference that we check the total number of grass units in the simulation space and the number of grass units in the sampled cell to make sure that adding the grass unit does not break the assumption concerning the maximum number of grass units per cell.

However, in order to randomly move rabbits on the grid, we use a methodology which is orthogonal to the iterative sampling described before. We modeled directions by adding a private enum DIRECTION to the RabbitSimulationAgent class. This enum contains four values, one for each of the possible directions. To sample a random direction, we first aggregate all valid directions in a List. A direction is invalid if it leads to a collision with another rabbit. If the List is non-empty, we generate a random index to select the random direction. Otherwise the rabbits does not move.

Since the grid has no borders (it is a torus) we use the modulo operator to make sure that rabbits new coordinates are within the grid and reset to the opposite side when the rabbit moves through one of the grid sides. More precisely we use the $Math.floorMod(n, moduloBase)$ function to compute the correct coordinate. Compared to Java built-in modulo operator, this function outputs a positive value when $modulobase$ is positive. As an example, $-1\%20 == -1$ but $Math.floorMod(-1, 20) == 19$ which is the expected index when the rabbit move through either the left or upper grid side.

## 2 Results

For all experiments we launched the simulation and plot the populations evolution for 1000 ticks.

### 2.1 Experiment 1

This experiment is the result of launching the simulation with the default parameters.

#### 2.1.1 Setting

*BirthThreshold*=30, *GrassGrowthRate*=30, *GridSize*=20, *MaxGrassGrowth*=20, *NumInitGrass*=50, *NumInitRabbits*=100

#### 2.1.2 Observations

This setting models a situation where both populations of grass and rabbits survive. Both populations plots follow a sinusoïdal pattern and are shifted with respect to each other. In other words, a local maximum in the rabbits population is related to a local minimum in the number of grass units and vice-versa. Oscillations amplitude tends to reduce over time. This could stabilize the ratio between the number of rabbits and grass units when time goes to infinity.

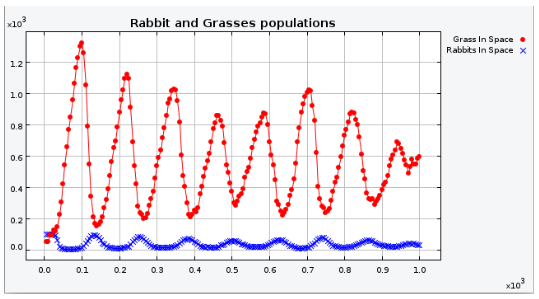

Figure 1: Plot of the rabbits (blue) and grasses (red) populations with default parameters over time

### 2.2 Experiment 2

For the second experiment we decreased the value of the *grassGrowRate* variable from 30 to 10.

### 2.2.1 Setting

*BirthThreshold*=30, *GrassGrowthRate*=**10**, *GridSize*=20, *MaxGrassGrowth*=20, *NumInitGrass*=50, *NumInitRabbits*=100

### 2.2.2 Observations

Decreasing the value of the *grassGrowRate* variable from 30 to 10 caused the extinction of the rabbits. From the plot [2] we can see how after the death of the rabbits the population of the grasses kept growing linearly till each plants reached its *MaxGrassGrowth*.

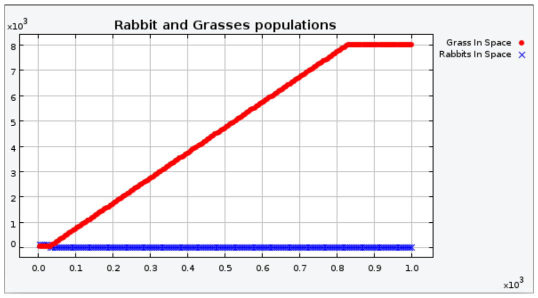

Figure 2: Plot of the rabbits (blue) and grasses (red) populations over time with *grassGrowRate* value modified to 10

## 2.3 Experiment 3

In the last experiment we modified the *maxGrassGrowth* value to 1.

### 2.3.1 Setting

*BirthThreshold*=30, *GrassGrowthRate*=30, *GridSize*=20, *MaxGrassGrowth*=**1**, *NumInitGrass*=50, *NumInitRabbits*=100

### 2.3.2 Observations

Once the initial rabbits population decreased the rabbits left in the simulation didn't die because they had enough grass to eat but they could not reproduce because the energy they could get from the grasses was just enough to compensate the energy they used to move. With those settings both grasses and rabbits population survived and reached a stable number.

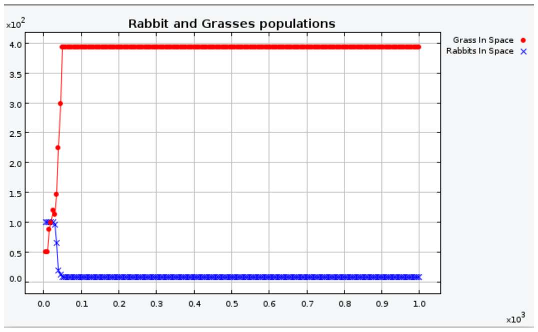

Figure 3: Plot of the rabbits (blue) and grasses (red) populations over time with *maxGrassGrowth* value modified to 1