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

Group №16: Léopold Bouraux, Vincent Rinaldi

October 1, 2019

1 Implementation

1.1 Assumptions

For the achievement of this application, in addition to the different constraints that were imposed, we made the following assumptions in order to build our world model :

- The grass amount, in each cell, has no upper bound, and its value increases continuously as long as it is incremented at a simulation tick.
- The color of a cell, containing grass, turns into an even brighter green if its grass amount value increases.
- If a rabbit, appearing as a white or gray cell, goes to a grass cell, appearing in green, the rabbit gets an amount of energy equal to the grass amount value that the grass cell was containing.
- When a rabbit goes on a grass cell, the grass amount value of the grass cell is immediately set to 0, and the grass cell turns back to black, until its grass amount value is again increased, which will make it turn green again.
- The energy amount value of a rabbit decreases by 1 at each step, even if it gains some energy at the same moment, increasing the energy amount value of a rabbit at each simulation tick by *GrassAmountValueOfActualCell - 1*.
- A rabbit having an energy amount value strictly less than 10 appears as a gray cell, otherwise it appears as a white cell.
- When the energy amount of a rabbit reaches the birth threshold value, making a new rabbit appear in the world model, the rabbit that just made birth sees its energy amount value immediately decreased by a predefined amount, represented by the variable *birthDamage*, if the difference is strictly lower than the value of the variable *birthThreshold*, and otherwise, it will be set to *BirthThreshold - 1*.
- The upper bound of the total number of living rabbits in the model is equal to the total number of cells in our model, equivalent to *ModelSizeX * ModelSizeY*, meaning that, when we want to add a rabbit, we first have to select an empty cell (such that the cell selection is random), and the number of cell selection attempts is upper bounded.
- After having chosen the next cell where a given rabbit should go, if this cell is already occupied by an other rabbit, the former rabbit won't move instead, but will still see its energy amount value decreased by 1, and won't be able to collect the energy from the cell it is staying on, in the case the grass amount value of this cell is increased at the next simulation tick, which enforces the fact that, in order to retrieve the energy contained in a grass cell, the rabbits have to explicitly make a move from an other cell to this cell.

1.2 Implementation Remarks

When executing our program, the very first step is to call the *tear down* routine (counter-intuitively named *setup*) in order to perform a cleanup by resetting the whole simulation to nothing, and getting it ready to be started again.

The *buildSchedule* method is a bit specific. In order to manage the different actions to take at each simulation tick, we defined 3 local classes. The first one is *RabbitsGrassSimulationStep*, which lets every agents move by one cell in one of the *N, S, E, W* directions, and spreads around randomly an amount of energy (grass) equivalent to the predefined *grassGrowthRate* value. It then gets rid of the dead agents (rabbits having an energy value equal to 0) and adds a new rabbit to the world model for each living rabbit having an energy value greater or equal than the predefined *birthThreshold* (and then removes some energy from those living rabbits equal to the predefined *birthDamage* value). This sequence of operations is done at each simulation tick. The two other local classes, *RabbitsGrassSimulationCountLiving* and *RabbitsGrassSimulationUpdateRabbitsAndGrassInSpace*, operate at every 10 simulation ticks. The former simply reports the current number of living rabbits and the current amount of grass in the world model (those data are used for our population plot), whereas the latter updates our plot.

Concerning the way we handle the different borderline cases, concerning the amount of grass spread around the world model, there is no upper bound condition, since we decided that each cell could contain an infinite amount of grass. However, we can't place a new rabbit freely. After selecting a cell randomly from the world model, we have to check if this cell is occupied. If it is not the case, then our rabbit is normally added to the world model on this cell. Otherwise, we select again randomly a new cell and check again if the cell is already occupied or not. After $10 * ModelSizeX * ModelSizeY$ failed attempts, we consider that the grid is overloaded and we don't add the agent to the world model.

Finally, about the movement of the rabbits, since our world model is a torus, if a rabbit reaches the border of the world model and goes out of bounds, it will teleport to the opposite side of the world model. This means that, at each simulation tick, since we choose randomly the next direction each living rabbit will take between *N, S, E, W*, we have to make sure that the new coordinates *(x, y)* fit into the world model boundaries. This is done thanks to the mathematical formula $(NewCoordinateX + ModelSizeX) \bmod ModelSizeX$ (the same goes for the coordinate y). If a rabbit tries to move to a cell which is already occupied, it doesn't move instead, only during this simulation tick (we don't try again to pick randomly a new direction since a rabbit can be surrounded by 4 other rabbits, one in each direction). However, its energy value still decreases by 1, and he will not be able to retrieve the energy from the cell he is stuck on, in the case some grass would be spread on this cell during the same simulation tick.

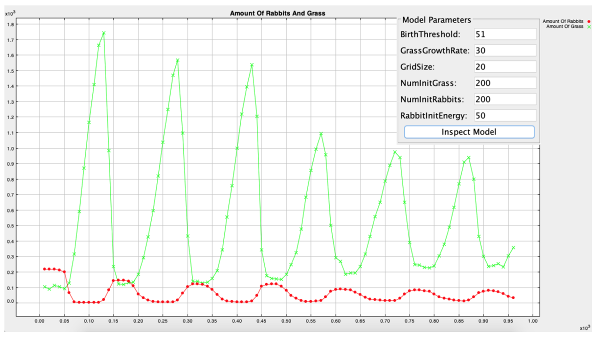
2 Results

Each of the graphs below represents the current total amount of grass in the world model (green) and the current total number of living rabbits (red) as function of the number of simulation ticks that have passed.

2.1 Experiment 1

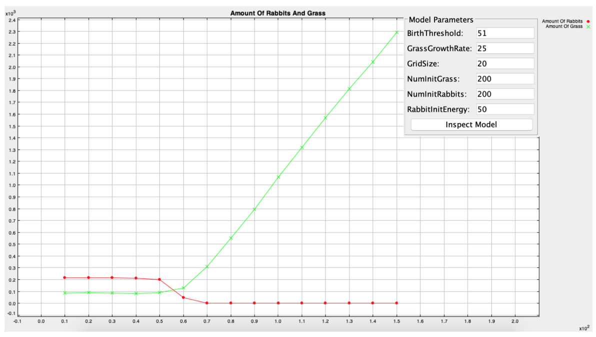
2.1.1 Setting

Basically, we set the initial amount of grass and rabbits to 200, the initial energy of the rabbits to 50, and the birth threshold just one unit above. The birth damage value has been fixed to $birthThreshold/3$. For this experiment, we will only modify the grass growth rate at each simulation tick : it is first set to 30 (Fig.1a), then we decrease it to 25 (Fig.1b).



Model Parameters
BirthThreshold: 51
GrassGrowthRate: 30
GridSize: 20
NumInitGrass: 200
NumInitRabbits: 200
RabbitInitEnergy: 50

(a) GrassGrowthRate set to 30



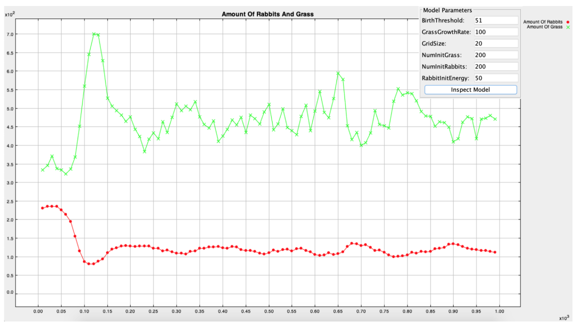
Model Parameters
BirthThreshold: 51
GrassGrowthRate: 25
GridSize: 20
NumInitGrass: 200
NumInitRabbits: 200
RabbitInitEnergy: 50

(b) GrassGrowthRate set to 25

2.1.2 Observations

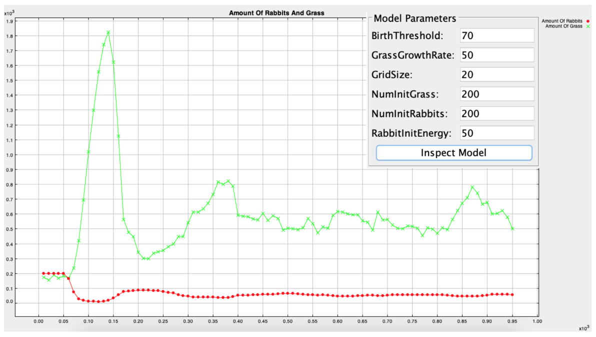
On Fig.1a, after 60 simulation ticks, the number of rabbits starts to drop drastically. Indeed, since rabbits move randomly, they are likely to stay in the same area for a few simulation ticks, causing in some regions a scarcity of grass if lots of rabbits operate there. This scarcity of energy leads to deaths, decreasing the rabbits population, and therefore increasing the total amount of grass, since less rabbits are collecting energy. Then, when the amount of grass becomes too abundant, the few remaining rabbits can gain energy more quickly, and thus reproduce a lot, causing then the grass to strongly decrease, and so on, creating a cycle. Thus, we can observe an alternating pattern between the two curves that seems to fade and stabilize over time (the two values tend to converge). However, as Fig.1b is showing, if the grass growth rate is too low, the rabbits are not feed quickly enough, making all of them to die fast, and then causing the amount of grass to increase linearly and infinitely over time.

2.2 Experiment 2



Model Parameters
BirthThreshold: 51
GrassGrowthRate: 100
GridSize: 20
NumInitGrass: 200
NumInitRabbits: 200
RabbitInitEnergy: 50

(a) GrassGrowthRate set to 100



Model Parameters
BirthThreshold: 70
GrassGrowthRate: 50
GridSize: 20
NumInitGrass: 200
NumInitRabbits: 200
RabbitInitEnergy: 50

(b) BirthThreshold set to 70

2.2.1 Setting

Now we will keep the same parameters, except that, we will first set the grass growth rate to 100, and then we will set the grass growth rate to a basic value, such as 50, but also increase the birth threshold to 70.

2.2.2 Observations

On Fig.2a, we notice that with a very high grass growth rate, the number of rabbits seems to stay constantly at a higher value since there is enough grass at each simulation tick to keep their energy stable. In addition, the two values seems to converge more quickly than in the first experiment. On Fig.2b, a higher birth threshold seems to slow down the reproduction process. The increase in the number of rabbits when the amount of grass is huge is less important than before, and the rabbits population looks like to converge faster.