

**————————————————————**

Weekly Assignment in *Spatial Simulation*

**————————————————————**

Lecturer： Wallentin, Gudrun

Stu-Name： Chen, Yuzhou

Stu-number： s1104123

Major： Applied Geoinformatics

Winter semester, 2023

Assignment #1 Calendar week 40 / 2023, Winter semester Course Date: 03.10.2023

## Introduction

GAMA is an open-source platform specifically designed for developing, running, and analyzing Agent-Based Model (ABM). The task of this assignment was to get start with GAMA coding and comprehend the order of code execution based on the given model "OrderofExecution", which is logistically complicated. In order to gain profound understanding of the execution sequence, I reviewed the document and executed the given code step by step to find the answer.

## Methods

It is essential to clarify the model structure at the very beginning. There are mainly 4 code parts included in the given model: 1) global part; 2) species definition part; 3) grid definition part; 4) experiment part. Apart from these basic parts, there are also 2 action type worthy of attention, which is *init* and *reflex.* While executing the program, the order sequence may not only be determined by the priority of statement parts, but also depends on the actions. Once clicking on the Execution button, the first block of output can be seen in the *Console*. And every new click will give a new output block in the console. By comparing outputs in every step, I think the model runs as follows:

* **Click *Run*.** All members start to initiate respectively.
  1. **Experiment**, similar to the ‘main’ entrance of programming language like C & Python, this is where the program begins. It defines how a model can be executed, including two types of experiment, gui and batch.
  2. **Grid**, a special type of agent differs from regular agents(species), which can be initiated automatically at the beginning of the simulation, no need to use ‘create’ explicitly.
     + The CA grid has a size of 2\*2, indicating that CA consists of 4 instances. The code "int grid\_var <- 1" defines a property named "grid\_var" for a grid and initializes it to 1. In the "init" block, the "grid\_var" is increased to 2, and all 4 instances’ value is printed.
  3. **Global**, a way to define global elements. It represents a specific agent, called *world*. The world is always created and initialized first when a simulation is launched.
* Firstly, a "glob\_var" is initiated to 1. Then comes the init{} section.
* According to the document, *cycle* is an build-in, read-only variable, designating the current time step of simulation. The first cycle is the cycle with number 0, thus leading to an output "time step: 0".
* After that, species is created in the order of B, A, C, with the number of 3, 5 and 2, respectively. The numbers represent the size of a species, just like indicating the population of a particular species in an ecosystem. In the meantime, all the "init" block in these species are triggered, with the order of "created". For example, agent\_B is created first, so the execution goes into its init part, gaining "agent\_B\_var" a value of 1, and print all 3 members in agent\_B(for 3 lines in the output). And agent\_A, agent\_C work the same.
* Finally, ask operator come into use. It is used to specify the interaction between the instances of species and the other agents. In this model, the "init" section of the global block contains an ask operator that involves the first agent in the "agent\_A" species, which is "agent\_A[0]". At this time, variable "agent\_A\_var" already has a value of 1(because of the initiate work of agent\_A), so the code "agent\_A\_var <- agent\_A\_var + 1" will send agent\_A\_var a value of 2, thus leading to an output "Agent A variable from global: 2". But! The real change happens in next step.
  1. **Species**, a prototype of agents, defining the attributes, behavior and aspects of agents.
     + All the initiative steps related to Species have been mentioned in Step 3, for creating species is an essential part of the global statement.
* **Let the simulation step ahead 1 cycle.** All *update* and *reflex* statements are triggered. *reflex* can be written in global part and agent parts, the ones in global part named "global reflex", others named "agent reflex". All the reflex function will execute at each time step.
  1. Global reflexes take the top priority to execute, which may run with regular order, from "global\_reflex\_1" to "global\_reflex\_4".
  2. Agent reflexes follows, which contribute to specific species. Needs to be mentioned that, the grid and species have been created, so the execution order of reflexes won’t follow the order in global init part, but follow the code itself. So, the update of "agent\_A\_var" happens first, then "reflex\_A1" follows, then "reflex\_B1" follows……
* **How to rewrite the model?** My thoughts: All requests are about initialization and update, code changes needn’t be in *reflex* functions.
  1. Agent\_A should have the values 1, 2, 3, 4 and 5.

Answer: There are two approaches to tackle this problem. 1) Use "ask" operator for 5 times in global init block, give each agent in agent\_A a specific value. For instance, ask A[0]->1, ask A[1]->2, etc. 2) Use an updating variable "plus\_num" as an add-on item to "agent\_A\_var", after every initiation operation, "plus\_num" increments by 1. Or a list<int> can be used.

* 1. To increment the values of all variables with 2 units per time step. Use update facets only.

Answer: Update statements execute in every time step, and they should be added to the end of initialization expression.

## Results

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Fig 1. Initialization step (Origin) | Fig 2. Step ahead (Origin) | Fig 3. Initialization step (New) | Fig 4. Step ahead (New) |

## Discussion

1.Talk about the difference between reflex\_B1 and reflex\_C1& reflex\_C2.

Actually, both agent\_B and agent\_C achieve the same outcome (incrementing a variable and reporting it), but agent\_B combines these functions into one reflex, and agent\_C separates them into two reflexes. The practical output is the same, as reflexes within an agent execute sequentially. In my opinion, the main difference lies in the coding practice, where agent\_C follows a more functionally separated approach, which I think is a better coding practice.

2.The value of individual can differ from each other in one species. With the help of the GAMA official document, I understand now that, the initiation of agents’ value in a specific species is a process executed sequentially, not in parallel, which enables us to specify a special initialization sequence for the agents, through initialization expressions or ask statements. It can be described as asynchronous. In javascript, "async" and "await" are features used to handle asynchronous programming.

3.But still an unresolved issue. Agent\_B consists of 3 individuals, when they are initiated by the expression "agent\_B\_var <- rnd(2,6);", there are always 2 individuals holding the same value. May this problem arise from the initialization sequence or something else?