PARIS
LODRON
UNIVERSITÄT
SALZBURG

Mid-term report of *Spatial Simulation*

Lecturer: Wallentin, Gudrun

Stu-Name: Chen, Yuzhou

Stu-number: s1104123

Major: Applied Geoinformatics

Winter semester, 2023

Contents

Background introduction

The report 1 dives into the sequence of code execution in "OrderofExecution" model with the aim of understanding how initiation and reflex actions impact it, which involves executing the code step-by-step and checking the order in which global definitions, species and grids are processed. This investigation sheds light on the importance of code running and action influence in Agent-Based Modeling of GAMA platform, providing insights into model control for me.

The report 2 discusses the development of an agent-based model named "Ass2CowsModel" that focuses on simulating the behavior of cow group. Each cow starts with a random age, and then engages in activities like grazing, vocalizing and moving at every step. The model keeps track of their life cycles with changing colors, aiming to enhance my understanding of GAMA coding structure and the order in which tasks are executed while constructing and analyzing behavior models for various species.

The report 3 presents a model based on GAMA that focuses on simulating the movement behaviors of 3 herbivorous species: cows, sheep and goats. The model aims to understand how these species interact within ecosystems by analyzing their movements, which involves creating a scenario with these species, each with their unique ways of moving. By utilizing built-in capabilities for modeling movement and geometry, the model visualizes how these animals perceive and interact with their surroundings based on factors, like speed, preferred directions and sensing abilities, thus demonstrating the patterns and potential interactions among these animals in their environments.

The report 4 focuses on simulating the movement and behavior of cows grazing on the Vierkaser pasture using GAMA. The fenced area consists of 80% shrubland with no vegetation and 20% grazeland divided into sections, each with varying levels of maximum grass. The target is to position the cows within the grazeland and allow them to freely roam around the fenced area. At each time step the cows will search for the available patch of grass to graze on continuously regrowing it until it reaches its maximum capacity.

The report 5 concentrates on creating a Unified Modeling Language (UML) for the “CattleGrazeModel” model before. When it comes to UML diagrams, they are representations that use nodes and edges to show how different parts of a system are connected. Considering the cattle have actions such as identifying spots, moving around, grazing, which may interact with grass, and acquiring energy, this assignment aims to understand the basics of UML notations and build a model using an UML Activity diagram with agent-based components to represent the interactions between the cattle and grazeland ecosystem through a workflow.

The report 6 tries to present a pre-designed model aiming to simulate the growth of the city of Salzburg from 1830 until now and helps me to do research on scenarios concerning with how road can influence the city growth. This research topic is important in various fields, including urban development, the traffic, society, and economics. Therefore, the model’s UML is created with an integrated urban index system that combines factors like slope, city center, roads and water bodies to predict how Salzburg will expand. Several validation techniques are proposed to ensure the accuracy and reliability of the simulation results by cross-referencing with real-world data.

1# Order of Execution

1.1 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. The answer was find by executing the given code step by step.

1.2 Methods

There are mainly 4 code parts included in the given model: 1) Global part; 2) Species definition; 3) Grid definition; 4) Experiment part. Apart from these basic parts, there are also 2 action type, 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. 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; 2) **Grid**, a special type of agent differs from regular agents(species), which can be initiated automatically at the beginning of the simulation The CA grid has a size of 2*2, indicating that CA consists of 4 instances. In the initiation, the variable named "grid_var" is declared, assigned with 1 and it increases; 3) **Global**, a way to define global elements. According to the document, 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, where an build-in, read-only variable named *cycle* exists, designating the current time step of simulation. After that, species is created in the order of B, A, C with the number of 3, 5 and 2, respectively. In the meantime, all the "init" block in these species are triggered, with the order of "created". Finally, *ask* operator come into use. It is defined 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. But! The real change happens in next step; 4) **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 (Fig 1; Fig 3).

- Let the simulation step ahead 1 *cycle*. All *update* and *reflex* statements are triggered in each time step. *reflex* can be written in global part and agent parts. **Global reflexes** take the top priority to execute, which may run with sequential order. **Agent reflexes** follows, which contribute to specific species. Needs to be mentioned, 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 (Fig 2; Fig 4).

- How to rewrite the model? My thoughts: All requests are about initialization and update, code changes needn't be in *reflex* functions.

Question: 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.

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

Answer: Update statements execute in every step, which should be added to the end of initialization.

1.3 Results

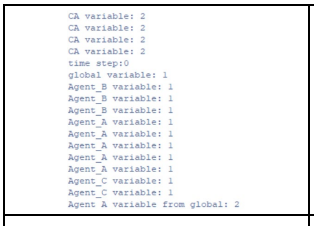

Fig 1. Initialization (Origin)

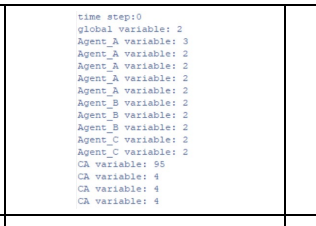

Fig 2. Step ahead (Origin)

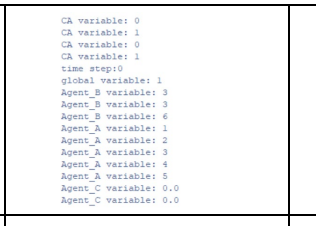

Fig 3. Initialization (New)

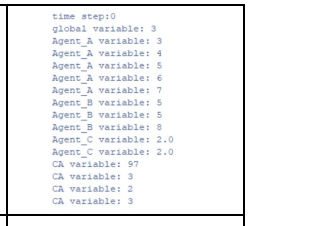

Fig 4. Step ahead (New)

1.4 Discussion

1.Talk about the differences between reflex_B1 and reflex_C1 or reflex_C2.

Both agent_B and agent_C increment a variable and report it, but agent_B combines these functions into one reflex, and agent_C separates them into two. The practical output is the same, as reflexes within an agent execute sequentially. In my view, the main difference lies in the coding practice, where agent_C follows a more functionally separated approach, which is a better coding practice.

2.The value of individuals can differ from each other in one species. The initiation of agents' value in a species is executed sequentially, not in parallel, which enables us to specify a special initialization sequence for the agents. It can be described as asynchronous in Javascript.

3.In the previous work, there was a problem with the expression "agent_B_var <- rnd(2,6)", there are always 2 individuals holding the same value when initiation(like in Fig 3)! The probability of happening equals (3*5*4+1*5)/125=52%, which is surprisingly high.

2# Cows behavior model

2.1 Introduction

GAMA has wide-ranging applications in modeling complex ecological systems and various aspects of biological behavior. The task of this assignment was to simulate the behavior patterns and growth status of cows associated with the time steps. In order to better depict the typical characteristics of this species, I designed a model called “Ass2CowsModel”.

2.2 Methods

Given the basic information in the instructions, cows’ initial state, behavior patterns, growth changes, removal of old members, and increase of new members should all be taken into account. There is a code provided that has 1 global part and 1 species called “cows”. At the very beginning of the code, global part is defined to do initiation works, which indicates the birth, age and the ready status of the 30 cows, where the age was randomly assigned by “age<-rnd(0,7);”; a “cows_num” variable was used to receive the quantity of cows rather than using a fixed number in the create statement, which may enhance the portability of the program. When it comes to the “species” definition for cows, the first thing is to add “skills:[moving]” outside the curly brackets, so that “wander” action can be triggered, combing with speed and amplitude attributes. Another actions cows would take in a new time step are as follows:1) grow older(that’s normal); 2) shout, like “moooooooo” ; 3) age_report, reporting current age of an individual; 4) die_catch, if older than 15 years old, with a fresh-baby cow coming into birth.

The conclusion is that *age growth* must be the first change in a new time step through “update:age+1;”. Then *die_catch* follows to judge the live, birth or death of the cows agents in a new year, which may have a profound impact on the cows’ number and the following output. Then reflexes named *shout*, *move*, *age_report* come into use, which show the actions of cow agents individually. So why these 3 specific reflexes arranged to the end of the section? Because all these outputs are determined only at the time when cows’ growth status are settled. And it is really a challenge to code the *die_catch* reflex, which needs to identify a dying cow and create a new one based on it. That means a fresh-baby cow with age 0 will “come into birth” when a cow reach 15 years old. In other words, it can also be regarded as the appearance of a 1 year old cow in the next year, when the dying cow last year truly passed away. This can help to maintain the overall cows population unchanged. *if* control structure is taken to find proper cows, use *create* and *ask* statement to create and initialize a new agent(with all action copies inside *ask* to ensure the new one act as normal), and finally “do die;” for dying one. When it comes to visualization, there are 2 parts requiring to be set: The *aspect* inside *species* format setting, where the color on the age of 5. And the *display* set in *experiment* part, using “output{display map{species cows aspect:default;}}” according to the document (Fig 5-9).

2.3 Results

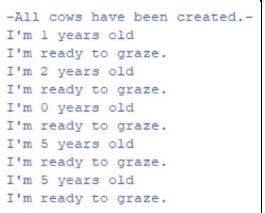

Fig 5. Initialization

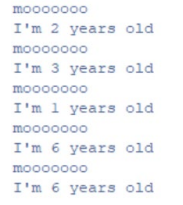

Fig 6. First step

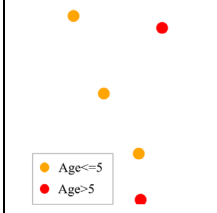

Fig 7. GUI of 1 step

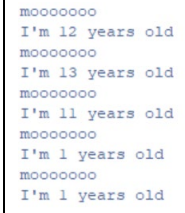

Fig 8. New Gene.

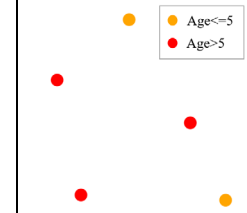

Fig 9. GUI of new Gene.

* Gene. stands for generation

2.4 Discussion

With the help of the practice on the model construction with “cows”, a much deeper insight is get with the execution order and the coding structure of GAMA.

From my point of view, the relationship between species and agents can be regarded just as the relationship between class and objects in ***Java***, which separately reflexes the generality and the specificity of a normal object in the world. Also, the *init{}* part matches with the self-defined *constructor*, while the *reflex* statements match with *methods*. Nevertheless, the main difference lies in the number of instances, which can be fixed in *.gaml* but not *Java*, so that we can get access to the specific instance(object) with *ask* facet and order number in GAMA.

3# Species Movement model

3.1 Introduction

GAMA offers built-in statements enabling simulation of species' movement behaviors to comprehend biological interactions within ecosystems. The task of this assignment is to track the movement of 3 species and visualize their action neighborhoods associated with the time steps. In order to better depict the typical movement and covering area, a model named "Ass3-MovementModel" is designed.

3.2 Methods

There are 3 typical intrinsic types of movement in GAMA coding, including *wander*, *move* and *goto*, which can be called straightly by *do* statement. These movement can't be visualized to action neighborhood without the help of Geometry data type, which contains point, line, or poly vectors. There is a code provided that has 1 global part, 3 species and 1 experiment part. At the beginning, a global part is defined to initiate cows, sheep and goats with individual number. When it comes to the definition of species, the first thing is to add "skills:[moving]" at the start line, so that *do* actions can be triggered. Each species consists of two parts: *reflex* and *aspect*. Where *reflex* is used to define actions, specify the range of movement or perception, and assign values to geometry features. *Aspect* is used to define GUI drawing rules, implemented through draw statements, which support direct drawing of geometry features. In this assignment, each move pattern matches with a movement behavior of a species: 1) cows do wander; 2) sheep do move; 3) goats do goto, leaving different covering areas compared to each other. **Cows** *wander* within a limited range of 90° with a speed of 2, using amplitude facet to specify. "cow_area1" stands for the action area with a shape of sector, which can't be defined directly by geometry, so *intersection* is used to create a logical AND sector region by circle and cone like "circle(speed) intersection cone(heading-45,heading+45);", where heading stands for the current moving direction. "cow_area2" works the same. Then *aspect* follows to define the way to draw both agents themselves and neighborhoods. **Sheep** *move* continuously to the south and can smell wolves at a distance of 3. According to the document, heading angle starts from the east and increases in a clockwise direction, so south is "heading:90.0". Move action generate a straight line from start to end, so it can be tracked using line feature with a length equal to the speed. Similarly, the position of wolf can be defined as a point, just in-line with the sheep, using "point(self.location+{0,-3});" is enough. **Goats** *goto* the origin. Unlike the others, *goto* needs a target facet while being called, thus indicating a fixed direction for the goats. So in every time step, goats move in a straight line, which can be create using "circle(speed)" to cut the line that runs from current position to the origin. In visualization, the predefined *aspect* types of each species can be called in experiment part and properties like "transparency" can be customized (Fig 10-12).

3.3 Results

Fig 10. Result of Time Step 2

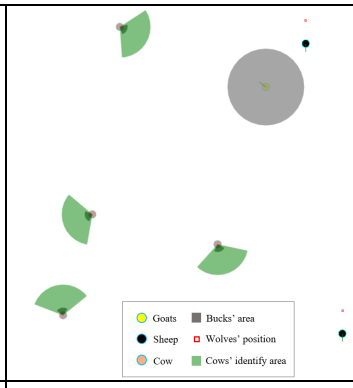

Fig 11. Result of Time Step 4

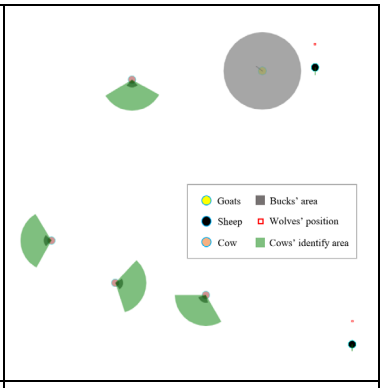

Fig 12. Result of Time Step 7

3.4 Discussion

1. Setting the aspect statement is akin to configuring the font family in Python chart drawing. In GAMA, this feature enables users to systematically customize a graphical style of graphics and call it in the experiment part using key-value pairs of aspect facet, where the value represents the name of the family. This offers a modular form of reuse for displaying various phenomenon in the same species.
2. How to define wolf area? Some debate arouses. It is better to use "point" rather than using "line area" to identify the perception neighborhood, because 3 is like an alarming distance for the sheep, which may become useless when the wolf approaches the sheep, within 3 meters.
3. In this assignment, sheep move in a *due direction*. What if there is an *angular deviation* on the direction? The vector "{0,speed}" should be calculated according to the specific angle with trigonometric functions.

4# Cows Graze model

4.1 Introduction

A variety of built-in statements and solutions provided by GAMA can support the construction of the interactive movement simulation of the agents. The task of this assignment is to learn the basic knowledge and operations of *Geometry* type and *field*, and to simulate the spatial interaction among several cows considering the distribution of grass biomass in the Vierkaser pasture area. To better depict the whole region and cows' movement, a model named "Ass4-CowGrazeModel" is designed.

4.2 Methods

Given the basic requirements in the instructions, there are 3 essential constituents to be taken into account: **1) The based field region**, including both graze area and shrubland; **2) The biomass field region**, indicating the nutrition value; **3) The moving cows**, which always identifying the harvest spot of grass as the target. The field areas mentioned above can't be visualized without the help of Geometry data, which can be defined from external files like *.geojson*. There is a code provided presenting a global part, 1 species (cow) and 1 experiment part.

At the beginning, the global part is defined to: 1) Import *geojson* files, using *file* type to catch the value; 2) Convert *file* into *envelop* or *geometry* by homonymous operators, using *geometry* type to receive the value; 3) Initialize biomass field and attributes, including the initiation of area range using "field biomass <- field(ceil(shape.width/5), ceil(shape.height/5),0.0)", regrowing height in each time step and the initial height of each kind of the grazeland using *loop*; 4) Give birth to 5 cows with random locations, using built-in variable *location* and method *any_location_in()* to achieve; 5) Define the process of biomass growth running in each time step, which can be done by *loop s* statement like "loop s over: biomass cells_in <my_geom>{}", a proprietary iteration method for *field*. It is worth noting that the range of grazeland is a combination of 4 different grass covers, so its *geometry* should be obtained by summing up the geometries of the four kinds of grass. At the same time, these 4 kinds of grasses varies in biomass contents, so they should be looped and assigned respectively. When it comes to the definition of cow *species*, the key issue is how to identify the field cell with the maximum biomass within the range. A *reflex* named *graze* is defined to solve this problem. Firstly, a range variable named *range* and a *geometry point* named *best_spot* needs to be declared while initializing, indicating the moving range and next destination for cow's moving. Secondly, to get the explicit position of the spot, we should form an expression as the logic follows: 1) Get the moving neighborhood of the cows, containing the position of the grid where the cow is now and the maximum range it can move to in next step; 2) Collect the cells' value within the neighborhood, using *collect* operator; 3) Maximize the biomass in reach, considering both the range and the biomass contents, using *with_max_of* operator and built-in variable named *z*. Based on the comprehensive analysis above, the expression is as follows, where "shuffle()" works to shuffle the order of the list of centroid point from *collect* operator "best_spot <- shuffle(((biomass cells_in (self + range)) collect (each.centroid))) with_max_of each.z;" Then, let the cows *do move* to their destination towards the *best_spot* and eat grass using minus. In the experiment part, *field* needs to be visualized by *mesh* statement with green color to make it clear (Fig 13-15).

4.3 Results

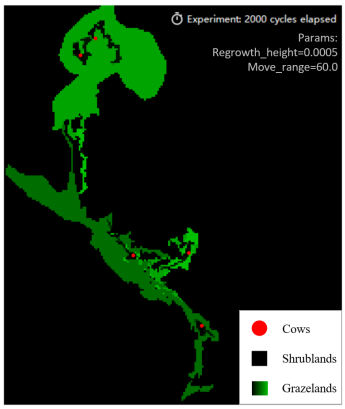

Fig 13. Regrowth_height = 0.0005

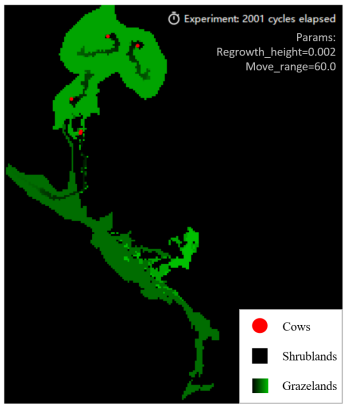

Fig 14. Regrowth_height = 0.002

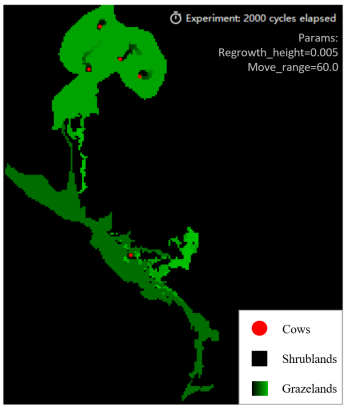

Fig 15. Regrowth_height = 0.005

4.4 Discussion

1. Additional issues considered in my code: The *logistic model* of growth. *Growth rate next step= $k \cdot x \cdot (max - x)$.*

2. From this coding task, some differences are find between ***field*** and ***grid*** which can express geographic raster. Field acts as a light-weight alternative to grids, as they hold spatialized discrete values without having to build agent, which can be defined in global section. So that, the field can also cover the whole environment like grid. According to the document, *field* was added in GAMA 1.9.1, which performs better than *grid* in larger sizes (above 500*500).

5# UML of Cattle Graze model

5.1 Introduction

A Unified Modeling Language (UML) diagram works as a partial representation of a system model, which contains graphical nodes connected with edges. The task of this assignment is to learn the basics of the UML notations and design a conceptual model using UML Activity diagram with agent-based components. To better visualize the conceptual model of the cattle-grazeland ecosystem, a UML diagram is designed in the form of a sequential flow chart, which named "CattleGrazingEcosystem".

5.2 Methods

There are 2 species that interact with each other in the ecosystem. Considering that grass, measured by biomass-content, occupies the lowest level in the food chain, it functions as a producer, while cattle make a living by consuming grass, establishing a producer-consumer relationship between these two. In general, cattle and grazeland agents in our ecosystem have their own characteristics and behavioral patterns, which can be represented by *Action* components in UML notation. These behaviors, depending on the execution mode, will produce different results, which are commonly distinguished by *Decision Nodes* in UML notation, and even, other characteristics will produce corresponding positive/negative feedbacks on the results determined by *Decision Nodes*. In fact, *Decision Node* is also used as *If* conditions in a normal flowchart. For biomass Object, the most important characteristic is biomass-content. Considering that plants in nature have a maximum growth height, if the grass is still at the maximum height, it will *Maintain* the current condition, but if there is a partial loss, it will trigger the *Grow* behavior, which will be limited by the *Logistic Growth Model*, with the following formula: $dN/dt=r*N*(1-N/K)$, where N is the number of populations, t is the time, r is the intrinsic growth rate and K stands for the max capacity. As a result, the line graph shows an S-shaped curve. When it comes to the definition of cattle, there are 4 sequential actions that cattle would take as follows:**1) identify spot**; **2) move**; **3) graze**; **4) gain energy**. While cattle are searching for a spot, they will always obtaining the optimal intake. Once the cattle have found the best spot, it will move towards it and start grazing. During this time, the physical state of the cattle cannot be ignored, where a characteristic named *Energy* is used to define in the UML notation: if the cattle does not find a suitable spot, it will be starved, leading to a greater grass intake in subsequent *Graze* action and vice versa. After recognizing all characteristics and actions of both *Biomass* and *Cattle* species, we need to add the *Control Flow*, an arrow-shaped component, to connect all existing nodes (Fig 16-17).

5.3 Results

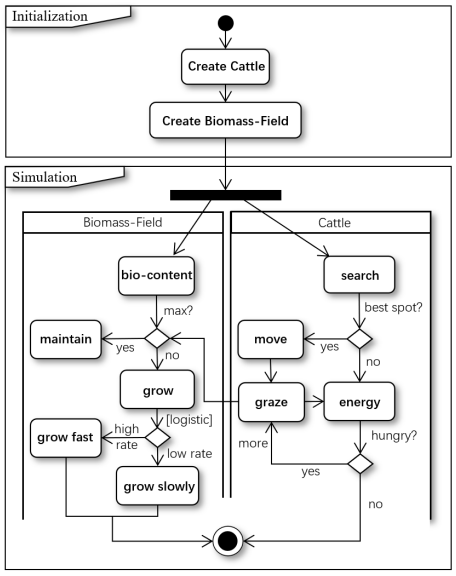

Fig 16. ***Old version*** of UML diagram

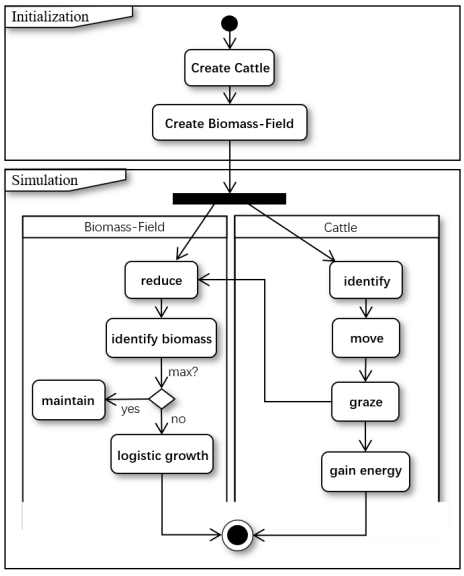

Fig 17. ***Revised version*** of UML diagram

5.4 Discussion

While drawing the UML of an ecosystem model with species interactions, we can view each species as an abstract Class in Java. So that, the core components of the UML notation are the attributes and methods of the Objects, and the judgement condition corresponding to the control flow is mainly distributed in the *reflex* functions of the species. That's why GAMA recommends to write each action in a different *reflex* function while programming.

6# UML of Urban Growth model

[This document was collaborated by Yuzhou Chen and Haoyu Cao.]

6.1 Introduction

Urban growth is the spatial growth of city limits over time. The study of urban growth is of great significance in several fields, including urban development, planning, environment, society and economy. Therefore, being able to accurately predict urban growth trends has become a great challenge for current society. Agent-based spatial simulation methods have considerable potential for modeling urban growth and support the input of raster and vector geographic data. In order to discover the urban expansion pattern of the city of Salzburg, we design a scenario that takes into account a series of factors affecting urban expansion, such as surface slope, city center, roads, and water bodies (Kumar et al., 2021). In addition, we show the specific simulation process of the methodology by means of UML modeling. The data inputs, simulation process, and data outputs are clear in the UML diagrams, and presenting the simulation as UML diagrams helps in the subsequent development of the urban growth program.

6.2 Methods

Considering the requirements in the instructions, our project aims to simulate the urban growth based on the theory of an integrated urban index evaluation system. There are mainly 6 essential constituents to be taken into account: **1) The land use plots**, indicating the extent of urban expansion, which is the main task of this simulation; **2) The road**, including different road types, guiding the extension of the city in a sector model; **3) The slope**, which indicates the degree of land surface relief, is a limiting factor for urban expansion, especially in hilly and mountainous areas; **4) The city centers**, consisting of real or hypothetical transportation hub locations, are indicative of the multi-center pattern that may result from urban development; **5) The water regions,** no urban growth in these areas; **6) Neighbors of non-urbanized plots**, which are the ranges dynamically delineated in the simulation to evaluate the contribution of the level of urbanization within the neighborhood of an image element to the central image element. At each simulated time step, the model calculates factor-weighted composite scores for unurbanized points and filters the range of urbanization for the next time step on the basis of the scores and additional judgment conditions (Fig 1). This will result in a long time series simulation of urban growth in Salzburg city (Fig 18-19).

When it comes to the *Input Data* of this project, all the datasets above can be imported into the GAMA model in the type of raster or vector files. So, the *land use plots* can be derived from the land use type map in Salzburg, 1830; the *road* can be a line vector stored in a shapefile with an attribute table indicating the road type (or maybe speed); the *slope* can be calculated by FABDEM in 30m; the *city centers* may be assumed by ourselves depending with some support evidence; the *water regions* can be derived from some basic geodata for Europe.

6.3 Validation patterns

Check that the model arithmetic logic is correct. As the simulation proceeds, it is almost impossible for incorrect logic to produce correct results. Therefore, it is crucial to ensure that the model's logic is correct, and this can be done by means of a single-step adjustment program to ensure that the model does not make errors. The results should be checked at every fixed number of rounds of the model, and common errors are extreme results and unchanging or recurring results.

Parameter sensitivity test. Changing the value of a variable while holding the values of other variables constant and observing the change in the simulation results. If the change in the simulation results caused by the change in the amount of change is stable, it is an indication that it is making a valid contribution to the model. Whereas a slight change in the variable causes a large change in the simulation results, it means that the variable is too sensitive under the current parameters and further consideration should be given to the assignment of the parameters.

Comparison with luminous remote sensing or OSM data. The OSM data may provide a more direct representation outcome for us to validate the simulation results. And the luminous remote sensing raster records the distribution and brightness of surface lights, which can be regarded as reflecting the scale and economic status of the city. Therefore, the night-light remote sensing data can be used as validation data to verify the results. Moreover, multiple periods of night light remote sensing data are available, which means that the model results can be validated multiple times in multiple periods.

Calibration and validation with existing land use data. Land use data is a more accurate man-made data compared to the night light remote sensing, in which the urban land can be completely equated to the area where the urban buildings are located, therefore, the land use data can not only be used to check the model results, but also to adjust the parameters of the model by comparing it with the actual urban area in the early stage of the simulation. In general, the goal of this process is to adjust the parameters in

such a way that the sum of the differences between the simulation results and the actual results at the same locations in the raster values is minimized.

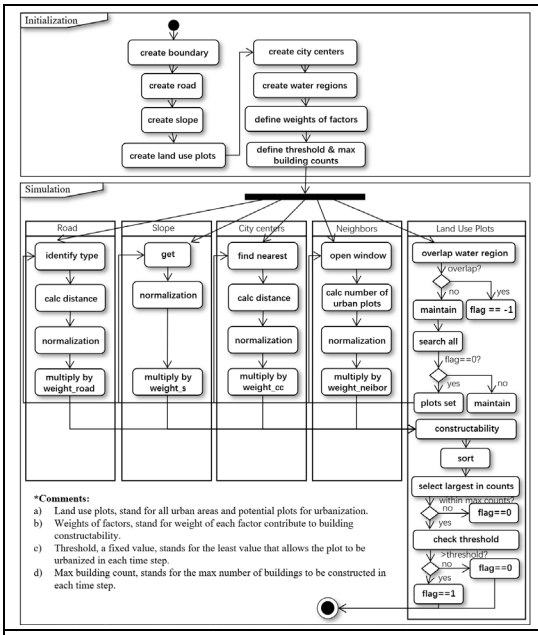

Fig 18. ***Old version*** of UML: Urban Growth Model

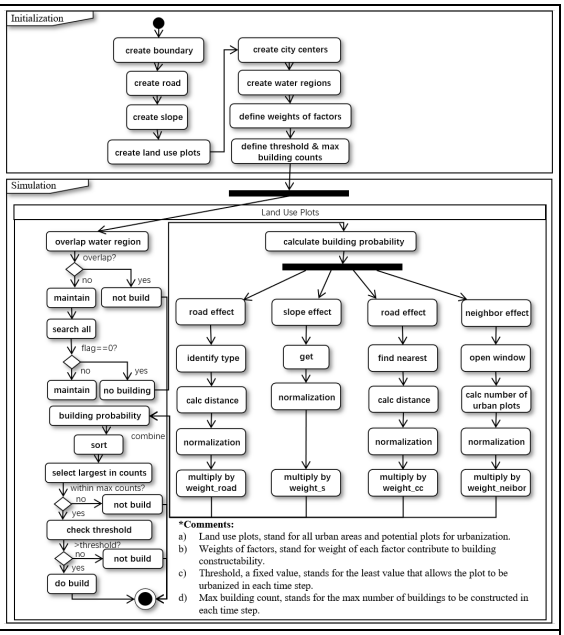

Fig 19. ***New version*** of UML: Urban Growth Model

6.4 Result

Scenarios: To explore the performance of the model in different contexts, we chose to conduct multiple sets of experiments by setting the value of the weights of a certain variable and the weights of the rest of the variables to 0, and finally compare the experimental results. For example, when exploring the effect of slope on urban growth, we set the weights of all other factors to 0, and then adjusted the factor weights of slope to observe whether there is a significant trend in the change of the results.

According to experience, the undulation of the terrain will greatly affect the process of urban sprawl, therefore, in the final results, we firstly reclassified the operation based on the slope, and secondly counted the number of buildings in different slope intervals, if the number of buildings in low slope intervals is high, then it is in line with our perception of urban growth.

6.5 Discussion

While discussing the issue of urban expansion under the influence of multiple factors, it's important to acknowledge that certain cities may also undergo the process of de-urbanization, characterized by a population shift away from urban areas towards suburban or rural regions.

6.6 Reference

Kumar V, Singh VK, Gupta K, Jha AK. 2021. Integrating Cellular Automata and Agent-Based Modeling for Predicting Urban Growth: A Case of Dehradun City. Journal of the Indian Society of Remote Sensing 49 : 2779–2795. DOI: 10.1007/s12524-021-01418-2

Discussions and conclusions

The provided tasks contain many patterns or instances of complex systems in the environment, mainly concentrating on the spatio-temporal changes, distribution, and interactions among various species. After finishing these specific tasks and generate reports based on my individual work, some achievements and deep insights about the spatially explicit modeling process are gained.

Firstly, the original task helped me to get the basic knowledge about the grammar of GAMA language in the form of .gaml. While comparing the coding hierarchical structure of .gaml file with some programming or scripting languages, such as Python, JavaScript and Java, some elementary problems mainly related to the data types are solved, which can be found in the document of GAMA. And after completing the 2nd task on modeling animal behaviors, I started to generate a deeper understanding on the 'function' concept in GAMA coding, which may fall into different working pattens, like *action*, *reflex*, *ask* and other built-in movements, just as object methods and self-defined functions in Java or OOP coding pattern in JavaScript. They accurately represent behaviors and growth patterns of the species over time providing insights into ecological systems and the simulation of biological behavior.

In the next task, things become more complex. That is, our understanding of how a species moves through space is not limited to where it is currently located. We are now starting to consider the trajectory of the species and the extent to which a specific behavior influences its movement. When running simulations, it's crucial to focus not only on where the species is located but also, but on its behavioral characteristics. Graphically representing these characteristics in an abstract way may help simulate their behavior accurately. During the coding process, obtaining trajectories for these behaviors might involve intersecting geometries, which adds complexity to the model but also makes it more realistic. For example, while determining areas where wolves exist, using point based identification is recommended instead of drawing lines, which improves the accuracy of sensing distances. This investigation demonstrates the complexities involved in modeling and underscores the need, for precise parameterization and definitions to simulate ecological behaviors within virtual environments.

Moreover, the geometry variable type and field operations are used to simulate the relationship between grazing cows in a spatial context in the following task. It incorporates a growth model to accurately depict the regrowth rates of grass making the simulation more realistic. It explains how two important species, grass (measured by its biomass) as producers and cattle as consumers have interactions that create a producer consumer relationship in a space or ecosystem, which was never mentioned in the previous exercise. Apart from this relationship, there are still some other specific relationships in the nature, such as Predation and Prey, Symbiosis, Competition, Facilitation, Indirect Interactions etc., which can also be simulated with the idea of spatial interaction, as long as they live in the same space. Through this analysis, the benefits of utilizing fields for modeling are discovered in GAMA, providing valuable insights into optimizing simulations for larger scale scenarios and improving efficiency when representing complex spatial dynamics, within simulated environments.

After building UML diagrams for ecosystem models in the 5th and 6th, each species can be represented as a Class. This viewpoint emphasizes the importance of attributes, methods and control flow that correspond to functions of species in UML notation. All species contained in a space container will not only continue their own life cycle, but also interact with other species concurrently. This idea tells the relationship between UML notation and programming paradigms demonstrating how UML diagrams can accurately represent intricate ecosystem interactions and act as a blueprint for implementing agent-based models in GAMA.

Based on the lessons from aforementioned tasks and outcomes, my own project is embarked, which involves studying the expansion of Salzburg city. The study on how cities expand under factors is a complex task that reveals the intricate dynamics shaping urban growth, where multiple elements may contribute to the growth with an unknown ratio. So that, abstracting the real scenario into a conceptual model takes the top priority to simplify the influencing factors in the process of urban change, and based on this, the interaction between the factors and urban expansion can be revealed, which is really crucial for coming up with strategies for urban development.