PARIS LODRON UNIVERSITÄT SALZBURG

MID TERM REPORT

SPATIAL SIMULATION

Submitted by :
Rabina Twayana
I2313887

Submitted to :
Prof. Gudrun Wallentin

14 Dec 2024

TABLE OF CONTENTS

INTRODUCTION **1**

WHAT'S THE ORDER? **2**

WORKING WITH LISTS **5**

ON THE MOVE **6**

GRAGING COWS **7**

PROJECT PROPOSAL **8**

VISUALIZE AND EXPORT RESULTS **9**

DISCUSSION AND CONCLUSION **10**

REFERENCES **11**

INTRODUCTION

Spatial simulation refers to the process that uses computational models to simulate and analyze the complex systems over time and space in the spatial context. It plays significant role in understanding and predicting the behaviour of spatially explicit systems that are influenced by spatial variability (O'Sullivan & Perry, 2013). Spatially explicit simulation is leveraged for wide range of application that benefits researchers and decision makers in diverse fields such as ecology, biology, hydrology, epidemiology, urban planning, resource management, disaster management, transportation, land-use change, wildfire prevention, etc. within a geographic or conceptual space. It encompasses a variety of models and approaches, each with its own strengths and weaknesses. For instance, equation-based approaches include Spatial System Dynamics, which id based on feedback loops and stock-and-flow diagrams to model complex interactions over time and Partial Differential Equation (PDE) Models, which simulate continuous physical processes over time and space using methods like Finite Difference, Finite Element, or Finite Volume (Wallentin, 2024),. Probability-based models involve Markov Chains, which predict future states based on current states and transition-probability matrix that is derived from statistical data (Wallentin, 2024). Rule-based approaches includes Agent-Based Models (ABMs), which simulate individual agents' behaviors and interactions accounting heterogeneity of the entities within a spatial environment (Taillandier et al., 2010), and Cellular Automata, divide space into discrete cells and processes and evolves in heterogeneous landscapes (Zenil & Martinez, 2024). These models are often combined, such as integrating ABMs with Cellular Automata, to capture the complexity of spatial systems and study emergent behaviors and dynamics in various scenarios.

Within this course, six assignments were completed to build expertise in ABMs using GAMA platform. GAMA platform is very powerful and easy-to-use open-source tool for modeling and simulating spatially explicit agent-based models. The high level programming language GAML(**GA**ma **M**odeling **L**anguage) is used for devolping the agent based models. It integrates GIS data, supports multi-level modeling, and offers advanced visualization and analysis tools, making it a versatile tool for GIS based applications (Taillandier et al., 2019).

The assignments included **Initialization and execution order** which emphasized on sequence of agents initialization and execution along with learning and following best coding practices. **Working with lists** involved analyzing lion population age demographics using list data types. The **On the move** module different movement rules are implemented and visualized for different species. **Grazing cows** focuses on simulating interactions between cows and grass, exploring steady-state dynamics. A **project proposal** was developed to work on real application. Finally, **visualizing and exporting results** focused on creating dashboards with maps and graphs to interpret simulation outcomes and export results in various formats.

These assignments collectively emphasize initializing agents and environments, defining behaviors, analyzing system dynamics, and presenting and exporting results effectively and implementing this knowledge in real world projects.

All the models developed for the assignments and datasets used, are available in this github repository: https://github.com/rabinatwayana/Spatial-Simulation-Assignments

WHAT'S THE ORDER?

INTRODUCTION

The study focuses on understanding the order of initialization and execution of different parts of a GAMA model. The initial task involved executing the provided execution_order.gaml model which aimed to understanding the initialization order of grid and species (agents), their execution order while performing the simulation, and understanding how the values are assigned in variables and updated in each step/iteration.

The subsequent task focused on restructuring the model code to enhance readability and conciseness. The task aimed for refactoring code in a concise and readable format and exploring alternative way to define the values for agents and cells and finally implementing logic to update the value of each agent by incrementing it by 2 at every step.

METHOD

For the first task, the provided model was imported and executed in GAMA. For the second task, the same model were updated by changing the name of agents and reflexes and removed the non-required code blocks. New species names **range_agent**, **random_agent** and **float_agent** were given based on the type of data they hold. The range_agent were initialized based on agent's self index , random_agents were assigned with the randon value generated using the **rnd** function between 2 and 6, and float_agent started with the uniform value of 0.0. For cell species, built-in attributes **grid_x** was used to assign the corresponding x-coordinate value. To increase the values of all variables by 2 units per time step, **update** command was applied in all species including grid which updates the values in every simulation iteration.

RESULTS

```
CA variable: 2
CA variable: 2
CA variable: 2
CA variable: 2
time step:0
global variable: 1
Agent_B variable: 1
Agent_B variable: 1
Agent_B variable: 1
Agent_A variable: 1
Agent_A variable: 1
Agent_A variable: 1
Agent_A variable: 1
Agent_A variable: 1
Agent_C variable: 1
Agent_C variable: 1
Agent A variable from global: 2
```

a

```
time step:0
global variable: 2
grid var 0: 73
Agent_A variable: 3
Agent_A variable: 2
Agent_A variable: 2
Agent_A variable: 2
Agent_A variable: 2
Agent_B variable: 2
Agent_B variable: 2
Agent_B variable: 2
Agent_C variable: 2
Agent_C variable: 2
CA variable: 75
CA variable: 4
CA variable: 4
CA variable: 4
```

b

Figure 1: Task 1(a)Initialization, (b)Iteration 1

```
CA variable x-coordinate: 0
CA variable x-coordinate: 1
CA variable x-coordinate: 0
CA variable x-coordinate: 1
Range agent init value: 1
Range agent init value: 2
Range agent init value: 3
Range agent init value: 4
Range agent init value: 5
Random agent init value: 2
Random agent init value: 6
Random agent init value: 6
Float agent init value: 0.0
Float agent init value: 0.0
```

a

```
time step:0
Range agent value: 3
Range agent value: 4
Range agent value: 5
Range agent value: 6
Range agent value: 7
Random agent value: 4
Random agent value: 8
Random agent value: 8
Float agent value: 2.0
Float agent value: 2.0
CA variable: 2
CA variable: 3
CA variable: 2
CA variable: 3
```

b

Figure 2: Task 2 (a)Initialization, (b)Iteration 1

Figure 1 represent the outputs obtained from the provided model and Figure 2 represents the outputs obtained after restructuring and updating the code.

DISCUSSION

This task was very helpful for understanding the execution order of initialization and simulations. Observing the order in which the species are created, how different initial value can be set to an individual agent, knowing the availability of in-built attributes like grid_x, use of looping, and modify the paricular agents by indexing are the major takeaway from this assignment. The questions are discussed below.

1. Initialise the Model

The output of the initialization of model is presented in Figure 1 Task 1 (a).

a) Why has the „CA variable“ a value of 2 in time step 0?

“CA” variable has value 2 in time step 0 because the variable “grid_var” of type interger(int) is initialized and assigned with value 1 initially and then updated by adding 1 to the initial value of itself i.e. **grid_var <- grid_var** + 1; in the “init” function that is executed in initialization process.

b) Why is the exact same information displayed several times for the agents and the CA?

The exact same information displayed several times for the agents and the CA depends upon the number of instances (agents) defined while creating species and the number of cells (width,height) defined while creating grid. In this example:

- create agent_B number: 3;
- create agent_A number: 5;
- create agent_C number: 2;
- grid CA **width**: 2 **height**: 2 {}

Each instance (agent) in case of species and each cell in case of grid will be assigned with same or differnet properties during the initialization process. This is why, agent_A, agent_B, and agent_C are displayed 5, 3, and 2 times respectively. In case of grid, number of cells equals to width*height, thus CA is displayed 4 times.

c) Why is the “Agent_A variable from global” different from the Agent_A variables reported before?

Initially, instances of species **agent_A** are created and initialized with value 1 in variable **agent_A_var**. Thus, In this step, all the agent of species holds the value 1. Following this in global section, only the first instance of “Agent_A” species is called and value is increased by 1. Thus, the value is printed 2 in this step. This is why *“Agent_A variable from global” different from the Agent_A variables.*

d) What is the order of Agent variables reported? Can this order be explained?

The order in which agents variables are reported is Agent_B, Agent_A and Agent_C. This order is based on the sequence in which the agents of species are created during global initialization (init) function. For example, if creation of Agent_C is defined first, the Agent_C variable will be reported first and so on.

e) What is the order of execution for the initialisation?

The order of execution of initialization is creation of cells of grid and then agent of species. In a nutshell, if the species of type “grid” exists in the model, agents of this species are created and finally the “init” statement of the global section is executed which includes the creation of agents of different species.

2. After the Simulation Step Ahead 1 Cycle

The output of the execution of first simulation is presented in Figure 1 Task 1 (b).

a) What is the order of execution at the first step? Does it differ from the initialisation?

The order of execution differs from the initialization order. The order of execution begins with the execution of reflex functions in the global section, followed by agents section which are defined in the sequence of Agent_A, Agent_B and Agent_C respectively and finally the grid species is executed.

b) Why is the order of agents different, now?

The order of agents in this step is different because after the global section, species Agent_A is defined, followed by Agent_B and Agent_C and finally the grid.

c) Why does one of the Agent_A variables have a value of 3, instead of value 2?

The Agent_A variables have a value of 3, instead of value 2 because only the first instance (agent) of species Agent_A is called and updated the value to 2 during the initialization process in global section previously and in this cycle the value is increased by 1 which became 3.

d) Why does one of the CA variables now differ from the others?

The one of the CA variables differs from the others because in reflex function (global_reflex_4) of the global section, only the first instance (CA[0]) of grid species is assigned with random integer value between 0 and 99 before executing the reflex function of grid. Therefore, this is due to the random value assignment for the first grid cell, however, the rest of the cells executed normally and have same values. In every cycle, the random value will be assigned for CA[0] which can not be predicted.

e) Have a look at how reflexes represent the behaviour of Agents in agent_B and agent_C: Is there any difference in the output? What is the better code design?

The value update logic and "write" action are defined in same reflex function for species **Agent_B** whereas defined in separate reflex functions for species Agent_C. Despite this difference, there is no change in outputs. For the concise and readable code, the reflex funcion of species Agent_C could be defined similar to Agent_B or follow the approach similar to Agent_A by using the "update" statement to update the value and reflex function to make a print.

3. Separate functionalities and give more meaningful names to the reflexes (Task 2)

To summerize the output as shown in Figure 2 Task 2 (a), range agents are initialized with the value from 1 to 5, random agents are provides with the random value between 2 and 6 and float agents are defined with constant value 0.0 and grid species are defined with corresponding x-coordinate value. The result of first iteration is represented in Figure 2 Task 2 (b), in which all the values are increased by 2.

WORKING WITH LISTS

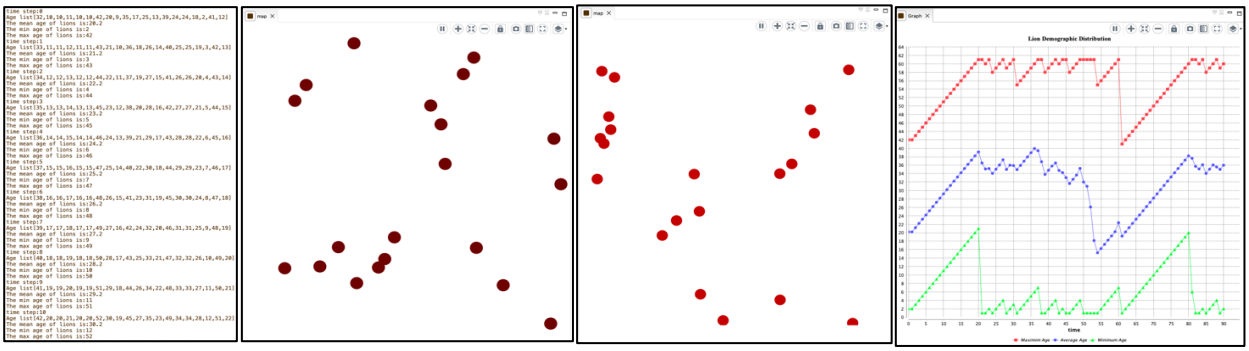
INTRODUCTION

The task involves reporting the demographic data particularly mean, minimum and maximum age of the lion population in each simulation step and visualizing the lion population in shades of red color based on mean age of lions. Furthermore, to enhance the model's realism, lions maturity threshold was integrated in which lions were programmed to die upon exceeding 60 years and just before each death, a baby lion was created to ensure the continuity of the population. The main aim of the task is to develop a deeper understanding of the declaration and manipulation of data type "**list**" in GAMA model.

METHOD

The **LionDemographic** model was developed to analyze the demographic dynamics of a lion population in each simulation step. To store age of lions, empty list global variable **age_list** was declared initially that holds the integer data type. In the global **init** function, 20 lion species were created and each assigned with a random age between 1 and 40, and these ages were subsequently added to the **age_list**. The **reflex** get_older function for lion's species was created where lions age is increased by 1 and updated age was added to **age_list**. In **reflex** lion_maturity function, if the lions age exceeded 60, a new lion was created before the death of respective old lion. The built-in functions **mean**, **min** and **max** were used to calculate the mean, minimum and maximum age among the species respectively. To ensure the updated age is added in list variable **age_list** in each iteration, the list was reset to empty within the reflex function in the global section. To visualize the lion's population in red shades, the mean age value was type casted into integer and multiplied by 5 to ensure that shades do not appear too dark (close to black).

RESULTS



a b c d

Figure 1: (a): Age Lists and Statistics, (b) Map of Simulation Step 1, (c) Map of Simulation Step 90, (d) Lion Demographic Graph

The figures illustrate output of the lion demographic model. Figure 1(a) shows the initial random ages assigned to lions and updated age list and demographic statistics in subsequent iterations. The Figure 1 (c) is the map of lion population after 90 step, which highlights the changes like removal of matured lion, addition of newborn lion and fluctuation of red shades reflecting the average age changes compared to step 1 (Figure 1 (b)). From the graph shown in Figure 1 (d), it can be analyzed that average age increases until lions maturity and started fluctuating afterwards.

DISCUSSION

The lion demographic model effectively demonstrated the use of list variable and its manipulation in GAMA along with the use of **if-else** conditional statement to manage the flow in model and use of in-built function to compute the statistics from the data list. However, the visualization based on the intensity of red color corresponding to the mean age of lion population appears to have limited variability, resulting in minimal fluctuations in color shades.

ON THE MOVE

INTRODUCTION

The task involved creating and assigning different movement rules to various animal species that includes cows, sheep, and goats, and visualize their action neighborhood. The main aim of the task was to develop a deeper understanding of movement behavior of the species and visualize where an agent potentially could be in the next time step.

METHOD

The **model** **movement** was developed with agents that encompasses 5 **cows**, 3 **sheep** and 2 **goats** within **init** block of **global** section. All species were created with **skills**: [**moving**] but with different movement properties. To move cows with correlated random walk at a speed of 2 and 90 degrees, the command **do** **wander** **speed**: 2.0 **amplitude**: 90.0; was defined in reflex move_cow function. For the sheep, the command **do** **move** **speed**: 1.0 **heading**: 90.0; was implemented in **reflex** move_sheep function to move them consistently towards south at speed of 1. To enable goats to walk slowly towards the origin with speed of 0.5, the command **do** **goto** **target**: {0.0, 0.0} **speed**: 0.5; was defined in **reflex** move_goat function. To visualize the action neighbourhood of agents based on their movement properties, action areas were defined for cows [**circle**(2.0) **intersection** **cone**(**heading** - 45, **heading** + 45);], sheep [**line**(*self*.**location**, *self*.**location** + {0.0, 1.0});] and goats [**line**(*self*.**location**, {0.0, 0.0}) **intersection** **circle**(0.5);].

Finally, all the species and action neighborhood geometries were mentioned under display settings of the experiment and visulaised in the map.

RESULTS

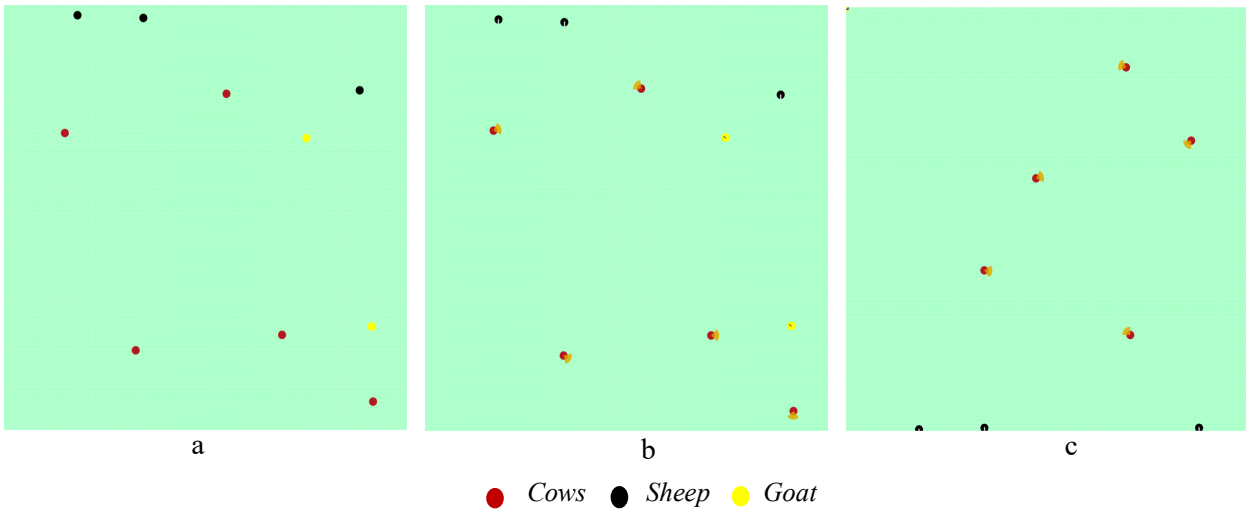


Figure 1: (a) Model Initialization, (b) Simulation Step , (c) Simulation Step 100

Goat agents that have their destination at origin, stops the movement when they reach their target that is at top left corner of the map window. Furthermore, sheep continously headed towards the North and stop when they reached the northernmost boundary. In contrast, cows wander randomly with no definite endpoint, thus their movements have no end.

DISCUSSION

The movement model effectively demonstrate the use of different movement rules that can be defined in GAMA along with visualization of their action neighborhood. This signifies the control of the behaviour of an agents based on the assigned movement rules such as direction, destination, speed, movement type and so on in the model.

GRAGING COWS

INTRODUCTION

The task involves creating cows model that grazes on the Vierkaser grassland on Untersberg. The main aim of the task is to develop a deeper understanding the interaction between agents (cows) and grid (grass) species and fitting the parameters in such a way that steady state evolves within the system.

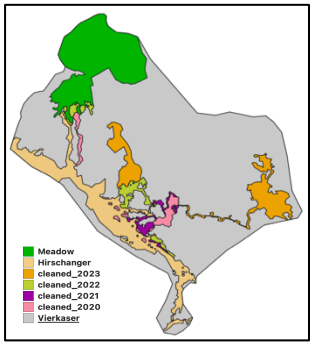


Figure 1: Study Area Map (Vierkaser Grassland)

METHOD

The **model** **GrazingCows** was developed that comprises 12 **cows** agent that are randomly within grassland using a geometry files provided in .geojson file format of the Vierkaser property along different types of grassland (meadow, hirschanger, cutback areas of 2020, 2021, 2022 and 2023) in **init** block of **global** section. Then species **grid** **grass** was created with cell size of 5 m and assigned with an intial **biomass** value equal to half of the threshold value. This was based on its intersection with grassland types, with provided maximum biomass values as follows: 6 for meadows and cutback areas from 2021-2023, 4 for Hirschanger, and 7 for cutback areas from 2020. Cells that don't intersect with any grassland type were assigned 0. In each iteration biomass values were increased by 0.1 and stopped when it reached the threshold. The cow agents are defined with **action_radius** of 10m and **wander** with **speed** of 5m within grassland. The mean biomass value was calculated in **reflex** **update_mean_biomass** function in the global section which is displayed in graph. The cow grazing behaviour was defined in **reflex** **cow_graze** function where best spot is find out within action area and reduce biomass by 0.1.

RESULTS

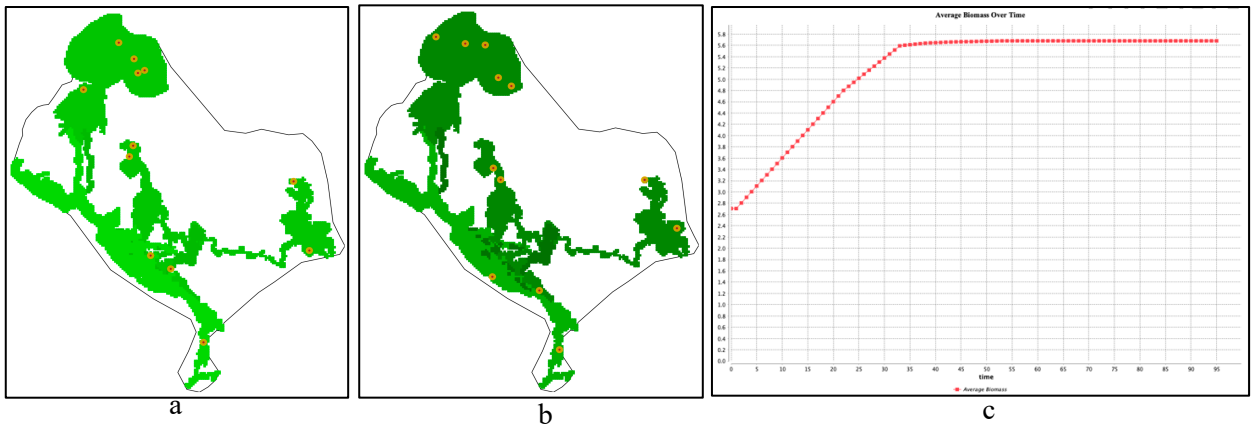


Figure 2: (a) Initial State, (b) Steady State, (c) Average Grassland Biomass Over Time

Figure 2(a) represents the initial state of the system where the grassland biomass defined as half of the biomass threshold based on intersection with grassland type. Figure 2(b) represents the steady state of the system where the grass grows to the maximum and cows on average eat as much as it regrow. Additionally, Figure 2(c) shows the system reaching a steady state after approximately 40 steps, with both the biomass increment and biomass decrement due to cow grazing set at a value of 0.1 per step.

DISCUSSION

The results indicate that the system stabilizes after a certain number of steps, as shown in Figure 2(c), where a steady state is achieved after around 40 steps. This suggests that the system reaches an equilibrium point where the biomass produced by the grass is in balance with the consumption by cows. However, these results could have been influenced by several assumptions within the model, including the initial biomass content and the constant values for biomass growth and grazing.

PROJECT PROPOSAL

TITLE

Geographical Agent-Based Modeling for Rainfall-Runoff Simulation: A Case Study of the Small Catchment Area of Bagmati River, Nepal

INTRODUCTION

In recent years, Nepal has experienced increasing rainfall in many regions, leading to flooding even in areas with traditionally low precipitation. Understanding streamflow and runoff patterns is crucial to anticipate the peak river overflows that cause such flooding. The Bagmati River, a key contributor to urban flooding in Kathmandu during the monsoon, highlights the need to study the relationship between rainfall and water levels. This project focuses on a portion of the Bagmati catchment area to develop and validate a runoff model that examines this critical relationship. Using precipitation sensor data and river water level measurements, the research aims to address the question: *How accurately can a runoff model simulate real-world hydrological dynamics through an agent-based modelling approach?* The primary objective is to create an agent-based model capable of simulating runoff dynamics for the Bagmati River and validate its accuracy by comparing simulated outcomes with observed data. This work seeks to advance our understanding of flood dynamics and improve predictive capabilities in similar contexts.

METHOD

2.1 UML Diagram: The preliminary UML diagram is outlined and will be refined as per the changes or updates in parameters. If soil type, infiltration values, saturation point. (if not available, assumption or reference from other areas will be taken)

2.2 Data Preparation and Parameter Finalization:

- **Input data:** 30m SRTM DEM downloaded from USGS, Slope generation from DEM, River network and sub-catchment digitised in Google Earth, Ground-based hydrological station data (Precipitation and Water level) from the website of Department of Hydrology and Meteorology (DHM) in Nepal, Soil type from National Agricultural Research Council(NARC), and Satellite image from USGS for terrain texture.
- **Parameters**: Precipitation, Water level

2.3 Development: The model will be developed using the GAMA platform.

2.4 Validation: The model will be validated by comparing the output water level from the model with the observed water level by using statistical Root Mean Squared Error (RMSE) metrics.

2.5 Scenarios: Different scenarios will be tested including high-intensity, medium-intensity and low-intensity rainfall events.

DISCUSSION

The model's scope could be broader if infiltration is considered. However, to simplify, a fully saturated state i.e. 100% water runoff might be considered which may affect the accuracy. The use of low-resolution DEM data and the limited number of ground precipitation stations may also reduce the model's accuracy, as varying precipitation patterns across the area cannot be fully represented. The development of the UML diagram helped in conceptualizing the overall workflow of the model. The expected insight includes an understanding of how rainfall interacts with the terrain and contributes to streamflow dynamics. The incorporation of a spatial perspective provides additional value by identifying rain flow patterns, soil characteristics, and terrain-specific behaviours which enhances the model's contextual applicability.

VISUALIZE AND EXPORT RESULTS

INTRODUCTION

The task involves creating dashboard with 2D map, charts and exporting the results obtained from the Vierkaseralm virtual pasture model. The main objective of this task is to explore different ways of visualizing the results from the model aiming to enhance the interpretability and presentation of the data.

METHOD

The experiment main_experiment was setup using the type: gui, that includes four display window: display map_2d, display "chart_1", display "chart_2" and display "chart_3". In the map_2d display, all the grassland boundaries were plotted along with the cow agents and grass grid. The charts were generated using the chart statement. In chart_1 display window, to visualize the biomass distribution (mean, minimum and maximum) of grassland overtime, the chart of **type: series** was defined provided with data and value for respective statistics. The total grass eaten by cows during each iteration was stored in **total_grass_eaten** variable. To plot the average grass eaten by cows in each iteration in chart_2 display window, chart of **type: series** with **value** average **total_grass_eaten** by cows was plotted. Finally, in the chart_3 window, the relationship between the average available grass per cell in the grassland and the mean grass eaten per cow was plotted. This was done using an chart **type: xy**, with the average available grass per cell plotted on the x-axis and the average grass eaten by cows plotted on the y-axis. The **reflex** save_data function in experiment saves the **cycle**, **mean_biomass**, **min_biomass**, **max_biomass** data to the defined file path in append mode in each step.

RESULTS

Figure 7 represents the dashboard that consists of maps and charts. Map_2d represents cows and pasture lands with varying biomass. Chart 1 represents the average, minimum and maximum biomass in each time step. Chart 2 represents the average grass eaten by cows in each time step. Finally, Chart 3 represents the average grass available per cell vs mean grass eaten by cows.

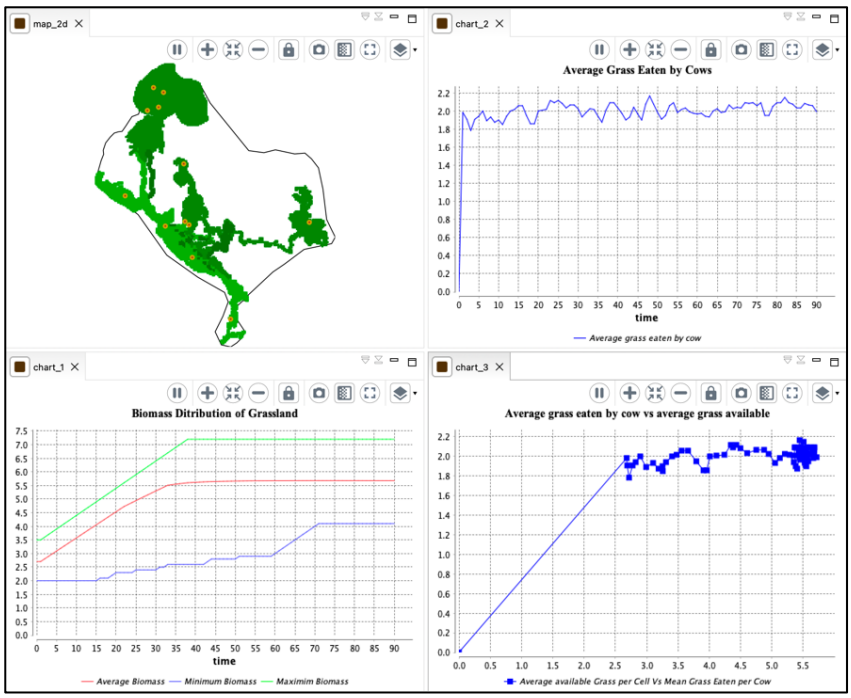


Figure 1: Dashboard with Map and Charts

DISCUSSION

The visualizations created in this study effectively presented the outcomes of the Vierkaseralm virtual pasture model. These graphical visualizations enabled a clearer understanding of the grassland dynamics overtime due to random cow grazing behaviour and limit to the growth of grass. One challenge encountered in this study was ensuring that the data accurately represented the dynamics of the virtual pasture. The results could have been affected by the assumptions made within the model, such as same biomass growth and grazing value.

DISCUSSION AND CONCLUSION

Spatially explicit simulation is a pivotal tool to understand and analyze complex systems that are influenced by spatial heterogeneity and dynamic interactions (Wu & David, 2002). By incorporating the spatial dimension, these models provide a framework to explore how individual entities, environmental conditions, and their mutual interactions lead to emergent patterns and behaviors. The flexibility of spatially explicit models enable researchers, spatial scientists and decision makers to simulate real-world phenomena which supports addressing challenges under varying scenarios.

Agent-based modeling (ABM) within a spatial context is a commonly used modelling approach which has the ability to capture individual-level heterogeneity and collective dynamics. A study carried out by (Husselmann & Hawick, 2011) found that by defining agents with distinct attributes and behaviors, spatial ABMs has significant capability to determine the influence of localized interactions to broader system-level outcomes. Subsequently, this capability contributes to study complex adaptive systems where interaction and feedback mechanism drive changes over time. Furthermore, the utilization of the geospatial datasets in ABM has significantly enhanced the prediction precision and accuracy of simulation because it allows for the integration of real-world constraints and behavioral patterns (Brown et al., 2005).

Through the completion of technique driven tasks during course, significant progress has been made in exceling spatially explicit modeling techniques. From understanding initialization and execution orders in GAMA to exploring agent-specific behaviors like grazing and movement, each task has contributed to building a robust foundation in modeling systems in spatially explicit environments. Implementing agent-environment interactions, such as grazing dynamics and demographic processes, highlighted the importance of spatial heterogeneity in shaping system stability and emergent behavior. Finally, the utilization of lists and built-in functions in GAMA enhances the ability to track dynamic variables, while visualization and export tasks offer knowledge and skills to effectively communicate model outcomes. In addition to this, development of UML diagrams in project developement phase helps in understanding the workflow, decision points and interaction between agents and environment.

These achievements collectively demonstrate the practical application of spatial modeling concepts and the ability to adapt techniques to simulate real-world scenarios. While substantial progress has been made, future work could focus on incorporating additional complexities, such as dynamic environmental conditions, agent learning, and multi-agent interactions. Overall, spatially explicit modeling has proven to be an invaluable approach for understanding complex systems, and the skills acquired so far lay a solid foundation for addressing more intricate challenges in the future.

In a nutshell, agent-based modeling in a spatially explicit environment, particularly through the GAMA platform, has revolutionized the field of geoinformatics. It offers powerful tools for understanding and predicting complex spatially explicit systems, with significant implications for various real-world applications. As technology continues to advance, we can expect even more sophisticated and accurate models that will further our understanding of complex systems.

REFERENCES

Brown, D. G., Riolo, R., Robinson, D. T., North, M., & Rand, W. (2005). Spatial process and data models: Toward integration of agent-based models and GIS. *Journal of Geographical Systems*, *7*(1), 25-47. https://doi.org/10.1007/s10109-005-0148-5

Husselmann, A., & Hawick, K. (2011). *Spatial Agent-based Modelling and Simulations - A Review*.

O'Sullivan, D., & Perry, G. L. W. (2013). Spatial Simulation Models: What? Why? How? In *Spatial Simulation* (pp. 1-28). Wiley Online Library. https://doi.org/https://doi.org/10.1002/9781118527085.ch1

Taillandier, P., Gaudou, B., Grignard, A., Huynh, Q.-N., Marilleau, N., Caillou, P., Philippon, D., & Drogoul, A. (2019). Building, composing and experimenting complex spatial models with the GAMA platform. *GeoInformatica*, *23*(2), 299-322. https://doi.org/10.1007/s10707-018-00339-6

Taillandier, P., Vo, D.-A., Amouroux, E., & Drogoul, A. (2010, 2010). *GAMA : bringing GIS and multi-level capabilities to multi-agent simulation* European Workshop on Multi-Agent Systems, Paris, France. https://hal.science/hal-00691400

Wallentin, G. (2024). *UNIGIS module: Spatial Simulation*. UNIGIS Salzburg. https://unigis-salzburg.github.io/Opt_Spatial-Simulation/index.html

Wu, J., & David, J. L. (2002). A spatially explicit hierarchical approach to modeling complex ecological systems: theory and applications. *Ecological Modelling*, *153*(1), 7-26. https://doi.org/https://doi.org/10.1016/S0304-3800(01)00499-9

Zenil, H., & Martinez, G. J. (2024). Cellular automata. In *Scholarpedia*.