ADVANCED PROJECT II

Project Report

Supervisor: Prof. Dr. Agostino Merico

Contributor: Dr. Davi Tavares

Virtual Environment for Individual-Based Modeling - Part II

By Ralph Florent

January 10, 2020

Abstract

This document describes the steps taken to add new features to the past work named *Virtual Environment for Individual-Based Modeling - Part I*. As this work mainly consists of a continuation of that previous version of this Agent-Based Modeling project, the same assumptions regarding the behavioral aspects of the waterbirds within some lagoons in the tropics are considered to build a more stable, featured *virtualized* system — a simulation. Among these newly-implemented features, figure the use of a *dynamical* environment and different types of waterbird species to study their interactions with the system.

Given that the project is built upon some advanced programming techniques, this document also reports on the technical aspects, namely procedural methods (algorithms, logics), applied to carry out the current release of the project. And finally, the results are openly discussed while giving an overview of encountered impediments and their possibly corresponding solutions.

1 Introduction

The *Virtual Environment for Individual-Based Modeling - Part II* is a continuation of the first release of the virtual environment project whose purpose is to study the habitat use by waterbirds in coastal lagoons of the tropics while using the Agent-Based Modeling (ABM) techniques [1]. As mentioned in the outlook for the first version, we see space for additional features that will supposedly give a considerably fair understanding of the system, including its components. Hence, we have decided to implement new features to help conceive a dynamical system and simulate more physical characteristics of an interactive environment. But before going deep into more details concerning the newly-implemented features, namely their technical aspects, that are considered for the second release, let us first raise some relevant points about this project report.

1.1 Assumptions

Roughly speaking, the past work is conceived to achieve a *Virtual Environment (VE)* using ABM techniques to characterize the waterbirds' behaviors, adaptation, and evolution within a set of habitats with distinguishable properties [1]. Since this project is built on top of the previous carried-out project[1], this report is indeed addressed to an audience with some practical understanding to grasp specific tightly-related concepts and certain decision-making when it comes to choosing between two (2) or more options. Additionally, it is expected that the reader possesses some knowledge on the following concepts of Python[2] programming: *object-oriented programming, file system handling, plots with matplotlib[3, 4, 5], and data format (e.g., YAML).*

With that being said, it is assumed that the reader is already familiar with these concepts, and therefore, any mention of them will not be enforced through any further detailed explanation in this document. However, although that broader explanation will not be provided throughout the following sections, particular notes and references will be intentionally used as a form of guidance for further readings.

1.2 Motivation

The idea of having a VE for ABM systems is quite innovative. It attracts many social science disciplines and looks very promising from an end-user perspective. As for ecological systems, being able to digitalize a specific real-world environmental context or situation while considering all (or the major part) of its complexities and implications is an opportunity knocking for alleviating some research

[1] It is highly recommended to check out the documentation of the *Virtual Environment for Individual-Based Modeling - Part I*. Refer to Appendix C for more details on how to access this documentation.

projects — accounting for budgets, expenses, resources, etc. This project itself is a good example to illustrate the importance of having a VE for further analysis of ecological systems.

In addition to that, the realization of this project will provide the following benefits:

- **Contribution to life science**: given that computational resources and equipment can be taken advantage of for heavy calculations, having the proper tools to carry out successfully related experiments is considered as an asset to the science community.
- **Contribution to waterbirds' lifestyle**: by simulating the waterbirds' life in a virtual environment, predictive analyses can help to determine factors that cause natural resources exhaustion (food unavailability, drought) or any other similar negative consequences, and take anticipated decisions to mitigate them.
- **Intellectual Property**: not only the tool can and will contribute to a large community that supports advanced techniques to improve life science work, but the author of such a tool can also benefit from worldwide recognition and publications.

1.3 General Comments

Recalling that the following document is about the tool used to describe the waterbirds' interactions within some habitats in the tropics, this description is therefore highly technical. That is to say, considering the key concepts of the VE tool are explained in the first part of the project, the following sections dive deep down into the programming techniques used to implement it. With the purpose of providing an easy-to-follow guide to grasp the main idea behind the writing of this report, an overview of its structure is outlined below:

- **Overview**: though it is not necessarily a concern, yet it is relevant to retake some of the notions discussed in our past work to facilitate the understanding of the terms used across the document. Parts of these notions are the basic theoretical background, the previous methods to operate the VE tool, the obtained results and discussions, and the reasons for having a part two (2).
- **Features**: right after explaining the reasons for defining new goals and how to reach them, we discuss the new features considered for the VE. The most relevant features are the application of a dynamical environment (involving more environmental characteristics), the increase of the number of waterbird species, the "multiple-agent updates per processing unit" implementation, and some other techniques for further analyses.
- **Methodology**: the procedural methods used in the past suffer some breaking changes while adding the new features, and these changes are discussed in the corresponding section. Besides

that, it is important to highlight certain points taken into account to ease up the development speed and productivity from a programmer's perspective.

- **Results & Discussions**: the reported results are based on a chosen test setting. Hence, we expose and assess them while maintaining the scope closed and simple, and we provide some outlook given the impediments we have confronted.

Keep in mind that for each subtopic mentioned above, we break them down into smaller points to facilitate an easy-to-understand structure and elaborate supporting details. This report is also available on GitHub under a public repository: github.com/systemsecologygroup/BirdsABM. Please refer to Appendix B to know how to walk through the repository and contribute to this project in the future.

2 Overview

The goal of the "*Virtual Environment for Individual-Based Modeling - Part I*" project was clear and simple: create a virtual environment prototype while using an ABM technique to study the habitat use by waterbirds in coastal lagoons of the tropics. Therefore, we built our assumptions on the general model concept shown below in Figure 1.

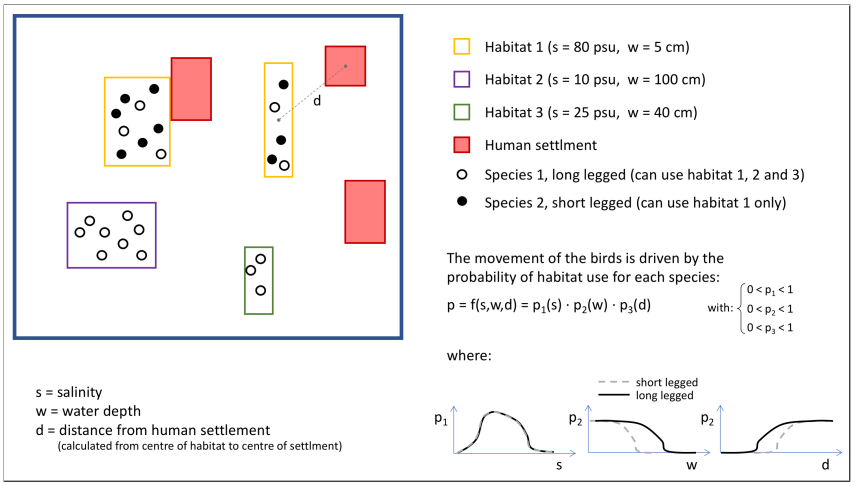

Figure 1: Model concept for the *Virtual Environment for Indivual-Based Modeling - Part I*.

As the second part of the project is built upon this first part, we retake some points that are relatively important and constitute the core elements of the second release. In the section, we provide an overview of the past work, then discuss the correlation between both the first and second versions.

2.1 Theory review

In the first part of the project, some core elements or components such as *warterbirds* and *lagoons* of the system are discussed and, subsequently, their digital representation within the prototype based on the assumptions derived from previous investigations.

Recalling that tropical coastal lagoons are shallow aquatic ecosystems located at the boundary between terrestrial and marine environments [6], the high environmental heterogeneity of coastal lagoons, in both temporal and spatial scales, provides habitats for aquatic bird species with different ecological needs [7, 8, 9]. Seven environmental variables mainly characterize these habitats, and water depth is the most crucial variable influencing the waterbird assemblage [6].

On the other hand, the aquatic birds or *waterbirds* inhabiting the coastal lagoons were, according to a survey conducted by Tavares D.C. et al. in 2015 [6], grouped into guilds[2] reflecting species' foraging habits and morphology. The six identified guilds were: diving birds (grebes), dabbling ducks (belonging to the genera Dendrocygna and Anas), large wading birds (herons, egrets, and storks), vegetation gleaners (jacanas and gallinules), fishing birds (gulls and terns) and small wading birds [11].

Agent-Based Modeling is a computational simulation framework based on intense processing and algorithmic calculations due to the fact the typical context in which the ABM is used is to study the collective behavior of a large number of components or agents [1].

2.2 Methods

The core functionality of the project is based on a programmatically-implemented coding procedure that allows us to explore certain programming techniques and choose the most convenient workflow for building the first VE prototype.

Its workflow scheme (as illustrated in Figure 2), includes some internal processes that were based on the following premises: *Initialize, Observe*, and *Update*, where:

1. *Initialize*: stands for initial conditions of the system
2. *Observe*: displays a snapshot of the current state of the system
3. *Update*: generates a new state of the system by computing some random movements of the agents (based on specific factors).

The VE prototype was mainly a digital representation of an ABM system where each component of that system relies on the interaction and interconnection with other involved elements in an organized

[2] Blondel proposed the guild concept in 2003 [10].

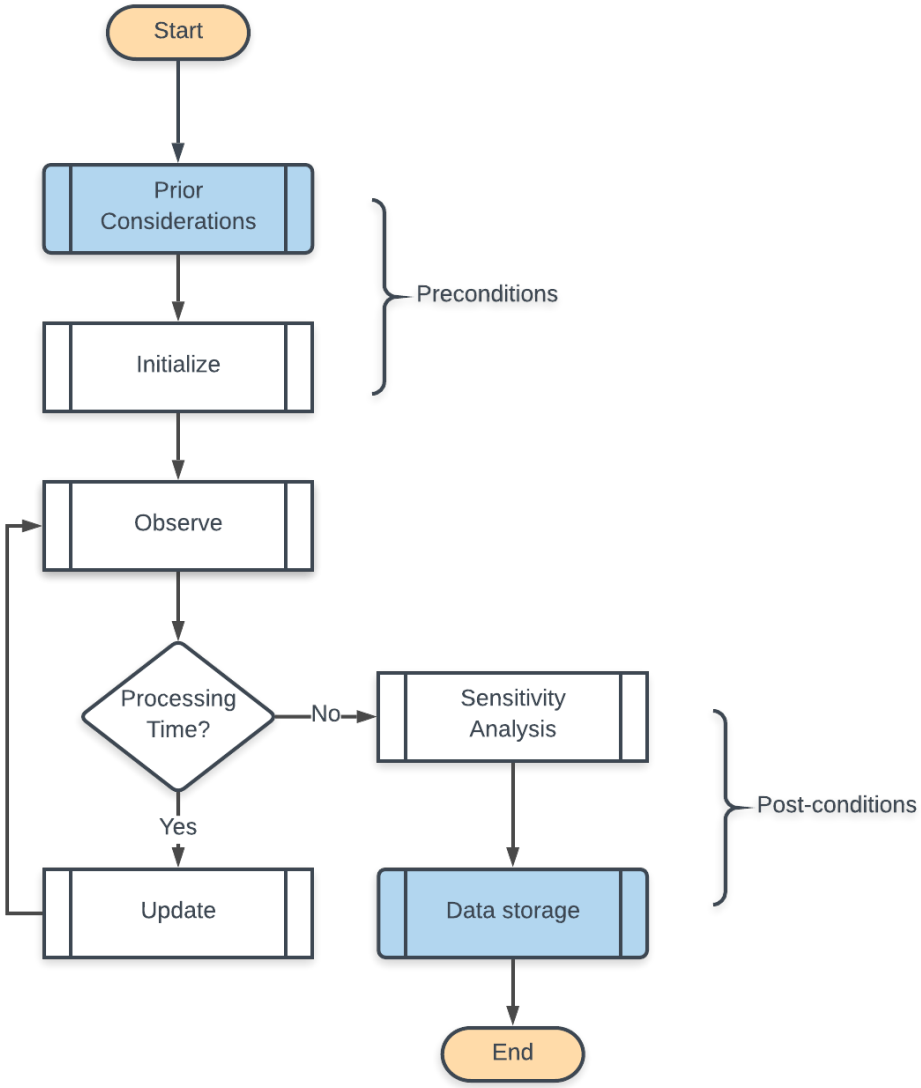

Figure 2: Workflow diagram
(credits: made with *Lucidchart*)

flow. The diagram in Figure 2 was used as a visual aid expressing the intended performing flow of the components within the system.

2.3 Results

After conducting two successfully (2) pilot tests, we combined them to shape a final prototype that represents the model concept illustrated in Figure 1.

Although this version of the VE simulation only considers certain factors that would interpret much

closer to the waterbirds' lifestyle inhabiting the coastal lagoons of the tropics, yet the results were fairly satisfying. Let us retake some key notions (terms and concepts) used to ease up the interpretations of those results.

- **Habitat**: a patch-focused drawing representation of the lagoons, rectangularly shaped and based on additional design settings.
- **Agent**: categorized as *short-legged* or *long-legged*, a drawing representation of the waterbirds.

It should be recalled that each run of the main script (available in Appendix D) would generate different results since we did not handle a seed-oriented initial state (see Table 1) for the components.

Initial Conditions	
Total of long-legged waterbirds	20
Total of short-legged waterbirds	20
Processing times	100
Threshold for	$1 * e^{-7}$
Areas for short-legged waterbirds	`one`
Areas for long-legged waterbirds	`one, two, three`
Habitat one (big)	s = 80psu, w = 5cm, f = 0.3
Habitat one (small)	s = 80psu, w = 5cm, f = 2.56
Habitat two	s = 10psu, w = 5cm, f = 6.41
Habitat three	s = 25psu, w = 40cm, f = 11.53

Table 1: Default values and parameters for the VE prototype's initial conditions.

As we do not intend to describe every detail of the results here, please refer to the documentation of the first version (see Appendix C) for further explanation.

2.4 Outlook

Note that in our previous work running the script would only generate a single-point dataset for each probability distribution function (PDF) over time. That is because, at that time, we did not consider a one-unit time processing where an *update* action would be performed on every agent. Meanwhile, we discussed in the analysis of the results on how to achieve a data set by tuning the habitats' characteristics and other influent factors over time. For instance, simulating a water depth reduction by including rainfall influence or tweaking the food availability factors is an excellent example of how to achieve a multi-point dataset for each PDF.

In addition to that, all the operations were executed in memory (CPU + RAM). No actual file dumping was being done. That was an exhausting condition for expensive computations and could even turn into memory leaks, among other consequences.

3 Additional Features

As discussed in previous sections, the first part of the project leaves a lot of rooms for improvement. As a matter of fact, that is the reason why we have decided to come up with additional features and augment the functional capabilities for both end-users and developers. Thus, we introduce in this section the set of chosen features and their importance across the application.

For the second version of the project, we have selected a set of features carefully to gradually integrate into the first version as we maintain the same workflow scheme displayed in Figure 2. These additional features are:

1. *File structure*
2. *More types of agents*
3. *Multiple-agent updates per processing unit*
4. *Dynamical environment.*

Next, we detail each one of them and their importance for the project.

3.1 A new file structure

Though it should not be seen as a proper[3] feature for the project, yet it remains an essential factor for the refactoring of the code implementation. Renewing the file structure of the project is the first

[3]It occurs that there are many definitions of a software feature. However, this action seems more like a refactoring than a new capability to the project. Proposing a new approach for file organization is yet questionable as it does not represent a distinguishable characteristic/capability of the project.

step considered before anything.

Recall that the first code implementation was done in a single Python file (basically, a Jupyter Notebook). This implementation was proper because it follows a standard programming workflow, which implies that the code should be at least *architected, standardized, structured, scalable, collaborative, documented*, and so forth [1, 12]. However, that is not the best approach to upkeep as we are adding more features and therefore increasing the degree of complexity of the project. With that, restructuring the file organization by breaking the main code into small chunks of code plays an essential role in project maintenance and scalability. The open-source community names this decision: "a near-term view of implementation and a long-term vision". Observe in the scheme below the compliant's folder and the new file structure used for the current application.

```
<project-root>
├── dist
├── docs
├── graphs
├── samples
├── src
│   ├── notebooks
│   │   ├── agent.py
│   │   ├── config.py
│   │   ├── config.yml
│   │   ├── constants.py
│   │   ├── core.py
│   │   ├── habitat.py
│   │   ├── helpers.py
│   │   ├── main.py
│   │   └── ...
│   ├── tests
│   ├── requirements.txt
│   ├── setup.py
│   └── ...
└── ...
```

Being out of the scope of this document, we will not discuss here other related topics such as *file structure convention, file naming, single responsibility principle*, among others. These topics are quite engaging and require further readings. Please feel free to refer to the *Coding Style Guide* for Python programming at python.org/dev/peps/pep-0008 or google.github.io/styleguide/pyguide.html for more details.

Note that, besides the file structure, we also follow an application structure commonly called LIFT:

- L: locate the code quickly
- I: identify the code at a glance

- F: keep the flattest structure possible
- T: try to be DRY — Don't Repeat Yourself *(avoid being so DRY that it sacrifices readability).*

Needless to say, having a good file structure along with a LIFT structure makes the project more robust and undoubtedly scalable. Now that we have a clear understanding of the file structure let us see how we integrate the actual features with it.

3.2 More than two types of agents

In the previous version of the project, we hardcoded two (2) types of agents: *short-legged* and *long-legged* waterbirds. Recall, as we discussed in the Overview section, that in the real world, there are more than just two categories of waterbirds. This short- and long-legged way of categorizing the agents was a simple approach as we intend to firstly create a pilot test and then built upon it something more complex. So, based on the assumption that the belly of birds cannot touch the water because it would contradict the homeostasis theory[4], we involve a set of four (4) bird species, with legs sizes of 5 cm (small shorebirds), 10 cm (big shorebirds), 30 cm (small herons) and 60 cm (big herons). As for their virtual representation in the VE, we discuss the implementation in the Methodology section.

3.3 Multiple-agent updates per processing unit

As for now, we can go a bit more technical. The previous implementation was lacking the functional capability of processing all the agents separately for a single update. That is to say, for n processing times, only n agents were randomly selected and processed[5] with the possibility of replacement. For instance, the same bird may get selected and processed five times in a setting constituted of ten (10) agents and five (5) processing times.

This new feature, *Multiple-agent updates per processing unit*, refers to adding the capability of processing every single agent for every processing unit.

Important: *Be aware that we use the term* process *in lieu of* update *when referring to moving an agent randomly to another position. There is a little nuance, and we want to keep a clear cut. It occurs that the term "process" is more global and avoids the confusion that an agent will always move. Au contraire, an agent will move if and only if the conditions for moving to another place are met. Thus, the term "process" means to try to update.*

[4] Birds would lose temperature rapidly.

[5] Processed: update an agent geometric position if the criteria for moving it were met.

3.4 Make the environment dynamical

Finally, we opt for using a dynamical environment, that is, by varying specific characteristics of the physical environment (PE) of the VE. One of the most relevant environmental factors in the VE setting is the water depth. By tweaking this parameter, we can simulate ecological changes in the VE over time and observe how the waterbirds evolve with those changes. Why? Because in real-life situations, the waterbirds' interactions and evolution are highly correlated to physical environmental changes.

Simulating an environment change within the virtual environment is a bit tricky and requires to tweak some variables to obtain the desired results. So, we provide more details on this specific case scenario in the Methodology section.

4 Methodology

The methods used in the first version are still applicable for this version of the project, except that we employ some key changes without losing the main focus. So, we describe in the following section what has been changed and how these changes are implemented.

4.1 The application

What is happening now when the application runs? Recall the workflow scheme in Figure 2 that indicates the three (3) main steps *Initialize, Observe, Update* used in the first part of the VE simulation. The idea is still the same, except that we include now more internal processes. As illustrated in line number 3 of Listing 2, the core methods imported to operate the simulation are exactly the same. In other words, the application runs and does the following:

1. first, initializes some internal configurations (we detail more about it later);
2. next creates m number of habitats and n number of agents;
3. then snapshots the current state of the virtual environment;
4. then runs the update process per time unit t;
5. finally, repeats the bullet points 3 and 4 for p times.

We omit for now the *post-condition* operations (scripts) since they are not part of the main idea but simply helper functionalities for future analyses.

4.2 External configuration

Although it implies the utilization of intensive programming to implement it, we proceed this time by adding a configuration file to load the external setup for the PE setting to run or simulate. The two files in charge of such a task are *config.yml* and *config.py* and are located under the *src/notebooks* folder. For best practice and human-readability reasons [13], the *config.yml* file is used, among other suitable options, to pass an application profile along with the parameters required by the system to operate. The profile is useful for meta operations like file system handling, for example. In contrast, the parameters are a way to load the core values such as habitat and/or agent information for the simulation. For example, dynamically loading the types of agents is made quite practical with the external setup. That is, an array of agents with the format shown in Listing 1 is expected and will be internally handled and represented as-is in the VE.

```
agents:
    - type: 30cm # unique identifier to categorize an agent
    quantity: 7  # total of agents of this kind to create
    color: '#7C79A2' # color representation of this type of agent
    label: 30cm-legged # descriptive label for the plots
    fn: # runnable function (definition, dependencies, arguments)
        def: 'lambda x: -10.89 * math.log(x) + 44.71'
        deps:
            - 'import math'
        args:
            - x
    habs: # types of habitats allowed for use
        - 1
        - 2
```

Listing 1: Example of data format for loading a specific type of agent

On the other hand, the *config.py* file is applied to load the main configuration in memory and apply it when it is necessary. This file also serves as a basis to set some constants values as they are frequently used across the application.

4.3 File system handling

The initial setup involves additional settings besides loading the external configuration. These settings are the file system handling, the operating system status check, and the Python environment (kernel). They help to prepare a sanitized workspace to run the application without crossing some surprising exceptions (e.g., memory leaks or overflow) at the time of execution.

```python
import config
import constants
from core import initialize, observe, update
# omit other imports for post-conditions

# main entry point for the application
def application():
    # pre-conditions
    config.init() # initialize internal config for the app
    time = 0 # define stopwatch for the process

    # process for t times
    print('=> START: Running simulation for waterbirds ABM')
    habitats, agents = initialize()
    observe(habitats, agents, time)

    for time in range(1, constants.PROCESSING_TIME):
        agents = update(habitats, agents, time)
        observe(habitats, agents, time)
    print('=> END: Running simulation for waterbirds ABM')
    # omit post-conditions

application() # run application
```

Listing 2: Main entry point of the virtual environment

The application is made smart enough to create, remove, and check the existence of files within the scope of the project. The folders *samples* and *graphs* are used for this purpose and contain respectively snapshots and plots files. Keep in mind that, for the file system handling, specific use cases (e.g., user permissions) are not taken into account.

4.4 The constants

The constants are useful when it comes to avoiding magic strings and numbers. As mentioned before, the loaded configuration is part of the constants used throughout the application. Besides that, we use constants for core elements (e.g., key identifiers for the static habitats), directory and file paths, in-memory storage, and default values.

4.5 The helpers

As we mention in the previous sections, the core functionalities *Initialize, Observe, Update* include some internal operations. For reusability reasons, we use helper functions to handle these operations such as random points generation, euclidean distance, gif maker, and so on.

4.6 The *Agent* class definition

In the first place, the *Agent* class was defined such that it accepts on-the-fly attributes. Normally, we kept it simple:

- ***type***: the category name of the waterbird species;
- ***x, y***: the x- and y-coordinate of the agent's position within the VE.

For this implementation, we define properly the attributes for the class and add the `color` and `name` attributes. These represent respectively the color of an agent's specific type and a unique identifier for data tracking. Moreover, we make the ***x***- and ***y***-coordinate attributes private and combine them into a single private property called ***point*** — point(x, y), a 1-dimensional tuple of shape (2,) representing a geometric position in a Cartesian plane. Being a private property, we provide the methods *get_point()* and *set_point((x, y))* acting respectively as *getter* and *setter* for such a property.

4.7 The *Update* algorithm

Roughly speaking, the instructions, as referred in algorithm 1, define a synchronous approach for allocating randomly a new position x, y to an agent if the criteria for moving are met. This was the basic idea of the *update* algorithm.

Algorithm 1: Update (VE Part I)

Data: Given a randomly selected agent

Result: Update an agent's position when doable

Randomly choose a new destination within an "acceptable" habitat;

Compute the probability of that new destination use for this agent ;

if *the calculated probability complies with the threshold* **then**

    move the selected agent to that new destination;

else

    maintain the same position;

end

Recalling that this algorithm above is defined in its most simplistic model, we had already anticipated that some changes would be necessary to make the virtual environment dynamical. As a result, to implement additional functionalities like *multiple-agent update per processing unit*, we break down the general idea into 2 pieces (or callable functions): *update* and *update_one*.

- ***update***: (short for update all) is an iteration over all the existing agents and apply some changes;

- ***update_one***: is the same *update* procedure defined in algorithm 1.

Important: *For this version of the VE prototype, we will refer to* `update` *as* `update_all` *to avoid confusion.*

4.8 The criteria for allocating new locations

The requirements for designating new locations to an agent are defined by the following characteristics: water depth, lagoon salinity, food availability, and minimal distance to human settlements. These characteristics are used to calculate the probability of moving to a new allowed destination (habitat), along with a chosen threshold[6]. In other words, the movement is considered adequate when the overall probability surpasses the chosen threshold.

Because of these criteria, we use internally a set of helpers to compute specific operations like the minimal distance to the human settlements, for example. Observe the lines 22, 23, 24, 35 in Listing 3 the respective helper functions *gen_rand_point(), which_habitat(), compute_dist(), eval_fn()*. Keep in mind that the calculations in lines 36, 37, 38 are statically set, and therefore, overly apply to any type of agents. That is because the specific functions for these probability calculations are still missing.

[6]Valid for sensitivity analysis by tuning its value.

```python
# omit imports and helpers
def update_one(habitats, agent):
    """ Update agent in one unit of time
    d: distance between the current habitat and the closest human settlement
    w: water depth of the current habitat
    s: salinity of the current habitat
    f: food availability in the current habitat
    """
    human_settlements = [h for h in habitats if h.id == C.HUMAN_SETTLEMENT]
    prob = 0.0

    for ag_cnf in C.CNF_AG: # for each category of agent (e.g., 15cm legged)
        restricted_habs = [] # this agent can use certain areas only

        for _type in ag_cnf['habs']:
            for hab in habitats:
                if hab.type == _type:
                    restricted_habs.append(hab) # are these limited areas
                    break

        if agent.type == ag_cnf['type']: # do's and dont's specific to this agent
            _x, _y = gen_rand_point(restricted_habs, 'in')
            _habitat = which_habitat((_x, _y), restricted_habs)
            _d = compute_dist(_habitat, human_settlements)
            min_index = _d.index( min(_d) ) # consider minimal distance

            # specific characteristics (props) of the selected habitat
            d = _d[min_index] # minimal distance to human settlement
            w, s, f = _habitat.props.values() # water depth, salinity, food

            # this agent knows a specific way to compute certain operations
            w_meta_fn = C.get_agentp(agent.type, 'fn')

            # compute the probability of moving to this habitat
            prob_w = eval_fn(w_meta_fn, w) # evaluate, compute the probability
            prob_d = -0.0013 * d**2 + 0.0074 * d - 0.0001 # missing eval_fn
            prob_s = 0.00006 * s**2 + 0.0002 * s + 0.0004 # missing eval_fn
            prob_f = (0.00673 * f**2) - (0.002936 * f) + 0.5 # missing eval_fn
            prob = prob_s * prob_w * prob_d * prob_f

        if prob > C.THRESHOLD:
            agent.set_point((_x, _y)) # allocate new position to the agent
    return agent, _habitat
```

Listing 3: Update agent in one unit of time

5 Results & Discussions

To test out the second version of the VE prototype, we prepare a setting with these values shown in Table 2. Obviously, this setting is configured using the YAML file, *config.yml*, which, in turn, is loaded via the *config.py* for some future usage.

Setting for Testing		
Processing times	25	
Threshold	$1 * e^{-7}$	
Waterbirds species	5 cm	5 in Habitat 1
	10cm	4 in Habitat 1
	30cm	7 in Habitat 1, 2
	60cm	5 in Habitat 1, 2, 3
Rainfall distribution	time: 1-10	20 units
	time: 11-20	200 units
	time: 21-30	600 units
	time: 31-40	30 units
	time: 41-50	0

Table 2: Values and parameters used to load the configuration for the initial setup of the VE prototype. The results are shaped in accordance with these preset values, but not necessarily the same due to the randomness the initial state.

Note that there is no external configuration format for the habitats yet. That is because the code implementation is tightly coupled to the existing design (and geometry) initially set. Automating the design setting for the VE prototype would add more complexity to the current virtual environment. Hence, it can be tackled down in a future release of the prototype. Similarly, be aware that the specific probability functions used to compute the overall probability of moving an agent are not normalized yet. That is the reason why the chosen threshold is so relatively low. Otherwise, the agent would never get updated. Fair tuning values for the threshold should be in the range of two (2) decimals

(e.g., 0.10; 0.25; 0.50, etc.).

The feature, *multiple-agent updates per processing unit*, makes the VE computations expensive and time-consuming. Given the handheld device[7] used to try out the setting, we choose a considerably fair processing time value, which is twenty-five (25) to run our tests. That is, for a total number of twenty-one (21) agents, the *update_one* function will be executed $21 * 25$ times. That is just the tip of the iceberg. Deep down within the *update_one()* function is the *gen_rand_one()* called to generate a randomly "acceptable" position. Depending on the habitat's capacity, this *gen_rand_one()* function can be quite a time- and memory-consuming due to its endless search for new positions.

Now that we have explained the reasons for our values and parameter choices let us present the obtained results that correspond to the chosen setting.

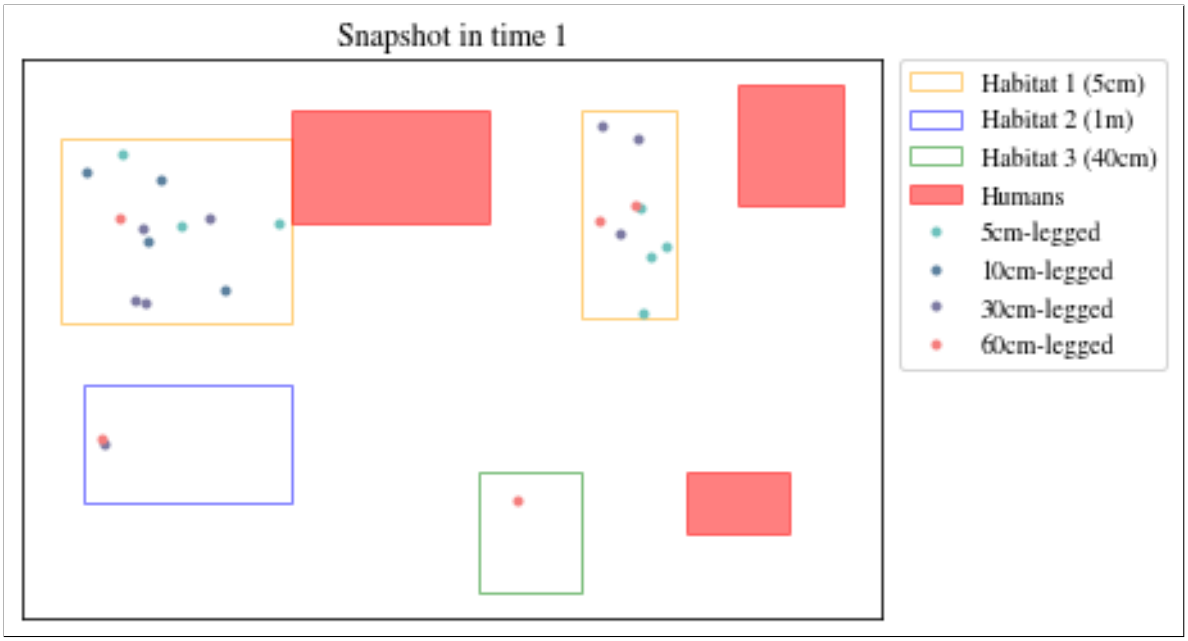

Figure 3: Example of a snapshot sample. Taken at the very first processing time, this snapshot displays the physical environment setting and distribution of agents within the simulation.

In Figure 3, we present an example of a snapshot when running the VE simulation. This snapshot is taken in time one (1) during the execution, which is the initial state of the components within the VE. Then, for every processing unit, a new snapshot will be generated and saved. Recall that this preview illustrated in Figure 3 is simply a visual aid for observations. But, in reality, we use a dictionary-based data format to hold in memory a summary of each state after processing, which is later on used for other purposes such as generating plots, data tracking, and sensitivity analysis.

The graph shown in Figure 4 is actually a graphical representation of that summary mentioned

[7]Laptop with limited computational resources (RAM + CPU).

above. For example, each subplot displays the distribution of the waterbirds' frequency in each lagoon.

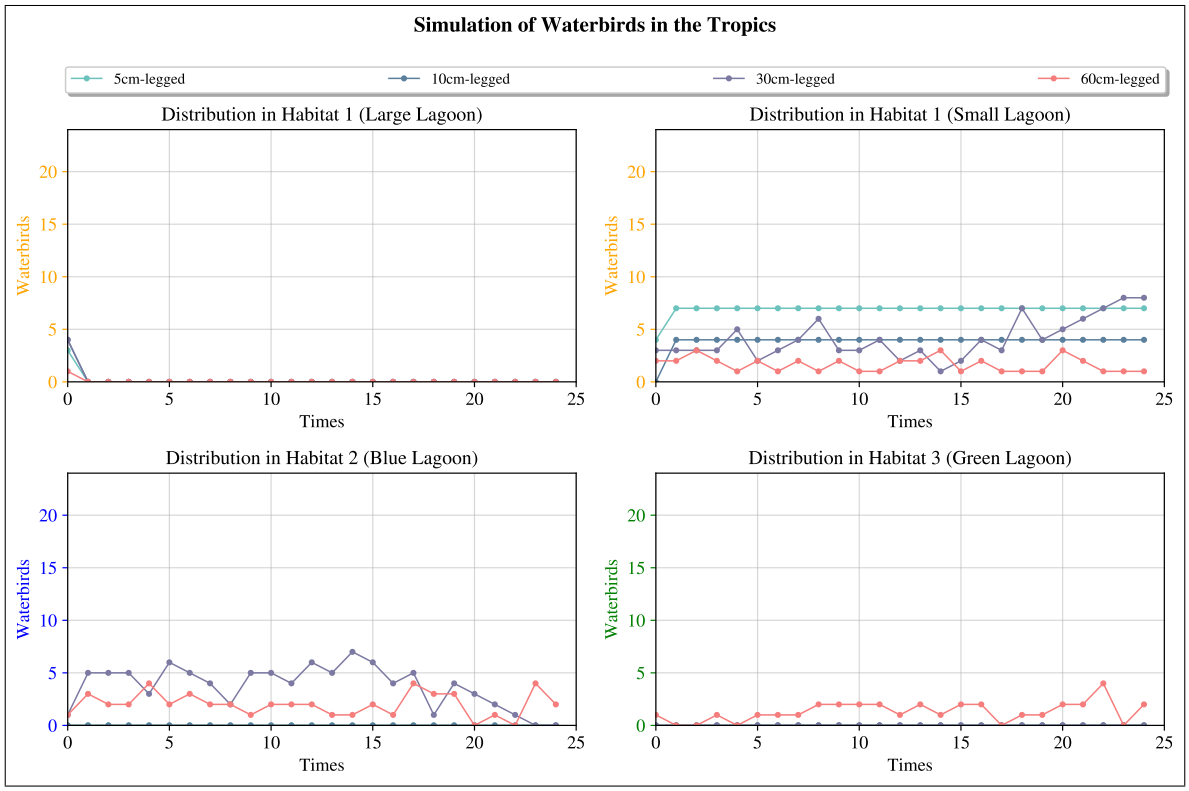

Figure 4: Summary of the simulation of the waterbirds in the tropics. This simulation corresponds to the setting used in Table 2.

Despite the new changes brought up to this version of the simulation, there is still room for improvement. Roughly speaking, we can enumerate the following flaws:

- the units need to be double-checked and standardized;
- the probability functions need to be double-checked and normalized;
- the real-world values must map correctly the values used in the virtual environment;
- the habitats' capacity is not taken into account, yet it remains an essential criterion for moving;
- the current code implementation is not unit-tested;
- the application is partially documented;
- among other considerations.

6 Conclusion

In this document, the main changes performed on the first version of the virtual environment have been highlighted and described. These changes are the main aspects that characterize the second version of this project. This version of the simulation is technically more robust and partly automated through external configurations.

This VE prototype is now capable of simulating a dynamical environment along with different waterbird species. Meanwhile, it lacks some implementation regarding the probability functions that are specific to the waterbird species and the characteristics (salinity, food) of the given physical environment.

A test setting is chosen to try out the core functionalities of the prototype. The simulation works as expected and the test results surely prove it. Unfortunately, no conclusion concerning the waterbirds' evolution or interactions can be drawn yet as the assumptions from the coastal lagoons are still in their analysis stage.

References

[1] Ralph Florent. "Advanced Project I - Virtual Environment for Individual-Based Modeling." In: (June 2019).

[2] Python Software Foundation. *Welcome to Python.org*. Mar. 2019. URL: https://www.python.org/.

[3] The Matplotlib development team. *matplotlib.pyplot.plot*. Mar. 2019. URL: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html. (Accessed: 22.12.2019).

[4] The Matplotlib development team. *matplotlib.patches.Patch*. Mar. 2019. URL: https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.patches.Patch.html. (Accessed: 22.12.2019).

[5] The Matplotlib development team. *matplotlib.path*. Mar. 2019. URL: https://matplotlib.org/3.1.0/api/path_api.html. (Accessed: 22.12.2019).

[6] Davi Castro Tavares et al. "Environmental and anthropogenic factors structuring waterbird habitats of tropical coastal lagoons: implications for management." In: *Biological Conservation* 186 (2015), pp. 12–21.

[7] YAA NTIAMOA-BAIDU et al. "Water depth selection, daily feeding routines and diets of waterbirds in coastal lagoons in Ghana." In: *Ibis* 140.1 (1998), pp. 89–103.

[8] Mariano Paracuellos and José L Tellería. "Factors affecting the distribution of a waterbird community: the role of habitat configuration and bird abundance." In: *Waterbirds* (2004), pp. 446–453.

[9] Davi Castro Tavares and Salvatore Siciliano. "An inventory of wetland non-passerine birds along a southeastern Brazilian coastal area." In: *Journal of Threatened Taxa* 5.11 (2013), pp. 4586–4597.

[10] Jacques Blondel. "Guilds or functional groups: does it matter?" In: *Oikos* 100.2 (2003), pp. 223–231.

[11] Davi Castro Tavares and Salvatore Siciliano. "Temporal variation in the abundance of waterbird species in a coastal lagoon in the northern Rio de Janeiro state." In: *Biotemas* 27.1 (2014), pp. 121–132.

[12] Smashing Magazine. *The Nine Principles Of Design Implementation*. Aug. 2017. URL: https://www.smashingmagazine.com/2017/08/nine-principles-design-implementation/. (Accessed: 22.12.2019).

[13] Martin Thoma. *Configuration files in Python*. Jacobs University Bremen. July 2014. URL: https://martin-thoma.com/configuration-files-in-python/. (Accessed: 03.01.2020).

Appendix A Online repository

All the code implemented during the execution of the prototype described in this report is available on the GitHub repository github.com/systemsecologygroup/BirdsABM.

Appendix B Understand the repository

It is highly recommended to check and read the markdown files (e.g., *README.md*) to encourage further understanding of every part of the repository. It is a self-sufficient, self-explanatory repository where every taken step is detailed with sustainable reasons.

We also try to follow the best practices by using the recommendations of the open-source community. For example, observe how we use the commit messages, the file structures, the naming conventions, and so forth.

Finally, we use up-to-date tools and software so that we can take advantage of their fully-available features.

Appendix C Access VE Part I

The documentation for the first version of the virtual environment prototype is publicly available at github.com/systemsecologygroup/BirdsABM/dist under the name *ve-ibm.draft@ju.pdf*. Consider reading it to catch the story behind how and why it all started.

Appendix D The Python Scripts

The Python scripts for this version of the virtual environment prototype are publicly available at github.com/systemsecologygroup/BirdsABM/src/notebooks starting from the *main.py* file.