TEC | Tecnológico de Costa Rica

Computing Engineering Department
Master of Science in Computer Science

On the Feature Space and Architecture of ABM Frameworks

A thesis submitted in partial fulfilment to opt for the degree of

Magister Scientiæ in Computer Science

Advisor:
Ignacio Trejos-Zelaya, M.Sc.

Author:
Luis Carlos Lara López

Scientific Advisors:
Santiago Núñez-Corrales, Ph.D.
José Helo-Guzmán, Ph.D.

September 2024

Escuela de Ingeniería en Computación
Unidad de Posgrado

TEC | Tecnológico de Costa Rica

ACTA DE APROBACION DE TESIS

On the Feature Space and Architecture of ABM Frameworks

Por: LARA LOPEZ LUIS CARLOS

TRIBUNAL EXAMINADOR

IGNACIO TREJOS ZELAYA (FIRMA)
Firmado digitalmente por IGNACIO TREJOS ZELAYA (FIRMA)
Fecha: 2024.11.13 11:34:23 -06'00'

MSc. Ignacio Trejos Zelaya
Profesor Asesor

JOSE ELIAS HELO GUZMAN (FIRMA)
Firmado digitalmente por JOSE ELIAS HELO GUZMAN (FIRMA)
Fecha: 2024.11.13 13:26:33 -06'00'

Dr. José E. Helo Guzmán
Profesor Lector

Santiago Nunez-Corrales
Digitally signed by Santiago Nunez-Corrales
Date: 2024.11.13 14:53:19 -06'00'

Dr. Santiago Núñez Corrales
Lector Externo

LILIANA SANCHO CHAVARRIA (FIRMA)
PERSONA FISICA, CPF-03-0257-0983.
Fecha declarada: 14/11/2024 09:57:43 AM
Esta es una representación gráfica únicamente, verifique la validez de la firma.

Dra.-Ing. Lilliana Sancho Chavarría
Presidente, Tribunal Evaluador Tesis
Programa Maestría en Computación

SINAES POST-GRADO ACREDITADO COSTA RICA

12 de Noviembre, 2024

To my beloved wife María del Mar and our children Baruc, Isaac, Maripaz, Natanael & Agnes. They are life itself.

A mi amada esposa María del Mar y a nuestros hijos Baruc, Isaac, Maripaz, Natanael & Agnes. Ellos son la vida misma.

No time, no space
Another race of vibrations
The sea of the simulation

Keep your feelings in memory
I love you, especially tonight

Franco Battiato, No time no space

Acknowledgements

First of all, thanks to the Divine Providence which allowed me to have a beautiful family, a fulfilling life, gave me the opportunity, time and energy to study my Master's degree and complete this thesis.

My full appreciation and devotion to my wife, she was always there when I needed her, took great care of the kids alone in the long hours of my research and held me together every single time I had chosen to quit.

I would like to thank professors Ignacio Trejos-Zelaya, Santiago Núñez-Corrales, and José Helo-Guzmán, for all the valuable advice they provided, and for their observations and feedback during the development of this research effort.

Also thanks to all professors from TEC, classmates, coworkers and friends who helped me complete this work. It could not have been done without all your support.

Abstract

Agent Based Modeling (ABM) is a computational paradigm for simulating the actions and interactions of autonomous agents (both individual or collective entities such as organizations or groups) to understand the behavior of a system and the underlying laws governing its outcomes. For the conclusions reached via this technique to be valid, the agent population in the ABM should be representative of – and commensurate with – the population size and rule set under study. Moreover, since the systems under study are complex and hence many of their possible behaviors are unknown, simulations should be executed several times to identify average (and representative) behaviors. From a computational perspective, most research reported in the literature has focused on the impact of performance, scalability and visual representations on the quality of the scientific outcomes obtained with ABM.

The present work details an Feature Space Maturity Model (FSMM) which includes significant sections usually overlooked in current ABM frameworks. Advances in Computer Science and the advent of more powerful hardware make it possible to define a reference architecture capable of supporting – at the framework level – robust computational properties, a set of essential features leading to correctness and fidelity to reality.

The purpose of this work is manifold. First, it introduces a maturity model for formal assessments of the feature space of ABM and evaluates the five most used ABM frameworks. Second, it proposes a macroscopic analysis of different implementations as a correctness mechanism. Third, it identifies and proposes a reference architecture based on the frameworks reviewed. Finally, it provides a Proof of Concept (POC) implementation based on the reference architecture to evaluate its quality according to the FSMM.

Contents

List of Figures

List of Tables

Acronyms

ABM Agent Based Modeling

API Application Programming Interface

CSV Comma-separated values

CV Coefficient of variation

DES Discrete-event simulation

DSL Domain Specific Language

EBM Equation-Based Modeling

FPS Frames per Second

FSMM Feature Space Maturity Model

GOL Game of Life

HPC High Performance Computing

IDE Integrated Development Environment

MEF Minimally Effective Framework

MVC Model-View-Controller

NPI Non-Pharmaceutical Interventions

OO Object-Oriented

OS Operating System

POC Proof of Concept

REPL Read-Eval-Print Loop

SD System Dynamics

SI International Unit System

SDL Specification and Description Language

SNT Social Network Theory

UI User Interface

UML Unified Modeling Language

CHAPTER 1

Introduction

"All the world's a stage. And all the men and women merely Players"

William Shakespeare, As You Like It

ABM is a computational model for simulating the actions and interactions of autonomous agents (both individual or collective entities such as organizations or groups) in order to understand the average behavior of a system and what governs its outcomes [68]. ABM differentiates itself from top-down simulation techniques like Equation-Based Modeling (EBM), Discrete-event simulation (DES) or System Dynamics (SD) because, in ABM, the models are built bottom up, defining only the agents and environment rules but not defining explicitly the behavior of the complex system. Such behavior emerges from the agents' interactions [59].

The methodological effectiveness of ABMs for complexity science [54], particularly in generative social science [26], resides in their ability to reproduce emergent behavior observed in – or expected from – real systems with relative ease [59]: ABMs abstract phenomena *bottom-up*. Top-down methods – i.e., equations-based modeling, dynamical systems, discrete event simulation – carry the burden of overly detailed specification. Obtaining meaningful results requires extensive parameterization, a process whose complexity becomes unmanageable for relatively small numbers of interacting entities – which forces the use of formal and numerical approximations. An even more striking limitation of top-down models is their opaqueness in terms of how micro-scale agent actions contribute to macro-scale observables, in stark contrast to what ABMs can achieve [81]. It is then no surprise for top-down models to be *agentized* – e.g., [85; 78; 88] - to then be able to answer questions pertaining to emergence.

A typical model-building exercise starts by specifying the structure, dynamics and interactions of autonomous agents (e.g., individuals, groups, organizations) based on prior evidence and its synthesis into a set of quantified relations. This process of abstraction results in a conceptual model stated in terms of interactions between classes of agents, and between classes of agents and their simulated environment when the latter is modeled. Naturally, the selection and prioritization of features changes the degree of representativeness of the model to what is being modeled, otherwise known as its

fidelity to reality [70], hard to assert for any given model. Since the latter may be too stringent a target, most ABMs aim to reproduce archetypal features of systems stated in terms of trajectories follow by agents and observables, as well as distributions of events during simulation time. In principle, once the abstract specification that serves as the blueprint for an ABM has solidified, the phenomenological outcome of any simulation should not depend on the particular computational implementation as determined by individual ABM frameworks despite non-functional differences such as runtime performance [3].

To produce reliable predictions, simulations in an ABM should match the target system size and rule set and should be executed several times to identify mean behaviors [10]. In order to achieve this level of efficiency, several ABM frameworks provide only limited simulation capabilities and typically use a standard programming language such as Java to implement the simulation behavior [56].

With the advance of Computer Science and the advent of more powerful hardware, we put forward that it is possible to define a reference architecture for ABM that supports, with robust computational properties, the essential features leading to correctness and fidelity to reality. ABM frameworks that follow this architecture could bring significant benefits to the research community and the general public, such as simplifying the development and maintenance of models using ABM, making them less error-prone, more accurate and credible, democratizing ABM's usage and increasing its value as a scientific tool. Some of the essential features for an ABM framework include advanced simulation functionalities, interrogation capabilities, and counterfactual support. The feature space could be further split into sub-criteria [8].

The objective of this work is manifold. First, it provides a maturity model for a systematic assessment review of the feature space. Second, it proposes an approach to validate the fidelity of the implementation to the specification in the overall picture of fidelity to reality. Also, it provides an initial reference architecture, with robust

computational properties, that ranks high in that maturity model. Finally, it creates a POC which follows the reference architecture to facilitate its evaluation. The structure of this thesis proposal is as follows. Other sections of this Introduction include a state-of-the-art ABM conceptual framework in subsection 1.2, related work in subsection 1.3, problem description in subsection 1.4, and hypothesis in subsection 1.5. Chapter 2 explains the objectives and contributions of the research, including its scope and limitations. Chapter 3 presents our Feature Space Maturity Model. Chapter 4 puts forward a Reference Architecture for Agent Based Modeling. Chapter 5 describes four experiments carried on each of five leading ABM frameworks, plus the POC ABM framework implementation. Chapter 6 exhibits and discusses the results obtained in the experimentation. The experiments' results and the experimentation process enable us to score the ABM frameworks and the POC, which is the subject of Chapter 7. Chapter 8 concludes and Chapter 9 outlines avenues for future work.

1.1 Problem description

Some of the most important deficiencies identified in ABM frameworks is the lack of both a FSMM and a reference architecture. This leads to several issues, including the inability to rank frameworks based on simulation capabilities, and the need to start from scratch the specification of future ABM frameworks. This work proposes designing an initial maturity model, and use it for creating a reference architecture and a small POC implementation to validate its quality.

The FSMM follows the facilitator-based approach of Nuñez-Corrales and Gasser [72], which is the capability of the framework to provide a desired feature, from (0) no facilitators, (1) facilitators need technical involvement for each case, (2) mature general facilitators require low technical involvement to (3) facilitators require no technical involvement. The FSMM consists of three sections: simulation features, interrogation

capability, and research supporting features, as defined previously. The sections are split into basic features. There are examples or descriptions to illustrate what it means for a framework to be assessed at some level for each feature.

The proposed reference architecture aims to achieve a high score in every feature of the FSMM while providing robust computational properties. Finally, a small POC has been developed to evaluate the reference architecture. The evaluation was performed by comparing four representative – and exemplar – models implemented in the POC versus implementations built in the other five ABM frameworks under study.

1.2 Related Work

Research works around different aspects of ABM are continuously published. Performance and scalability are some of the most studied topics [55]. Núñez-Corrales and Gasser [72] provide a scoring approach for the easiness of running social science simulations when access to cyber-infrastructure is available. They follow a systematic approach and propose an ideal framework called Minimally Effective Framework (MEF). They also scored five of the most used frameworks (which includes Swarm and Repast, also scored in this thesis) and assert that a new type of framework is needed for fully realizing the potential of agent-based modeling and simulation in Computational Social Science. While they realized the need for scoring other areas, their focus was mainly on simulations over High Performance Computing (HPC). They used the concept of facilitator, a way in the framework to achieve certain behavior and propose an approach of scoring from 0 (no facilitators) to 3 (facilitators require no technical involvement) depending on the maturity of the evaluated feature in the framework. This scoring approach has also been followed in the FSMM developed in this work.

Kravari and Bassiliades [52] present a broad review of several agent platforms and realizes the need for a scoring system applicable to ABM frameworks which includes usability, operating ability, pragmatics, and security management. Overall, the authors surveyed 24 frameworks. Its main drawback is that the survey is relatively shallow and does not provide insights for enhancements. A similar review was performed by Abar et al. [1] which surveyed more than 80 frameworks.

Works such as [89] and [96] have argued on the importance of a Domain Specific Language (DSL) for ABM. Nevertheless, a pure functional approach has limited adoption [90]. Also, the Object-Oriented (OO) approach matches better the general concept of an agent which is an entity that does things to things [21]. Montañola-Sales et al., in their 2013 review of ABM for the Social Sciences [65], confirm one of the main challenges of social simulation is to find a methodology capable of improving

communication between people related to the construction of a simulation model. They proposed Unified Modeling Language (UML), Petri Nets and Specification and Description Language (SDL) as approaches to code the logic of model [65]

1.3 Hypothesis

It is possible to identify a feature space leading to correctness and fidelity to reality in ABM frameworks. Furthermore, it is possible to define a reference architecture with robust computational properties which supports that feature space.

CHAPTER 2

Conceptual Framework

"All models are wrong, but some are useful."

George Box, Science and statistics

The use of ABM is heterogeneous. There are researchers whose research goals revolve around the design of various types of agents. In this case, the role of simulation is to validate the future operation of physical or virtual agents [10]. On the other hand, there are researchers whose goal is not agent design *per se* but rather the agent design is a means to develop simulations that can lead to a better understanding of a global or emergent phenomenon associated with complex adaptive systems [58]. The subsections in this chapter identify several concepts around ABM to validate the need for a FSMM to evaluate current frameworks and the necessity of a reference architecture which could be used as a baseline for future framework implementations.

2.1 General ABM Concepts

ABM consists of three main aspects: the agents, the environment, and the rules or interactions. Hossein et al. mention four types of agents in an increasing level of complexity [82]:

- Reflective or myopic agents, which are very simple if-then agents so that if they face situation A, they immediately do action B.
- Utility-based agents are very similar to the reflective ones, but they have a utility function that they do want to maximize in every case.
- Goal-based agents are an advanced form of utility-based function because they have a goal that dictates their actions.
- Adaptive agents are the most advanced form because they possess enough cognitive capabilities to change their actions in similar conditions, based on prior experience. E.g. if they do action A in situation B and lose some payoffs when they face situation B again, they don‘t do action A according to their prior experiences.

The environment is where the artificial social life of the agents unfolds [28]. Environments come in three different major forms:

- **Spatial environment** The spatial environment is often a 2D/3D continuous plane or discrete lattice on a sphere or toroidal topology. There are models which do not require spatial environment simulation, like pure networks simulations. Simulations where users do interact within the spatial environment are called space-aware, and are the main concern of this work.
- **Networked environment** In real-world situations, such as socio-economic settings, agents have more networked interactions than spatial (geographical) interactions. Using network structures as an ABM environment provides ample opportunities to synthesize Social Network Theory (SNT) with ABM.
- **Mixed environment** Here, both spatial and network pieces coexist in the environment.

There are five basic classes of interactions [101]:

- **Agent-self**: an agent checks its internal states and decides according to them.
- **Environment-self**: are when areas of the environment alters or changes itself.
- **Agent-agent**: are usually the most important type of action within ABMs and most of the emergent behaviors originate from this type of interaction.
- **Environment-agent**: happen when the agent manipulates or examines an area of the world in which it exists, or when the environment in some way observes or alters the agent‘s internal states
- **Environment-environment**: between different areas of the environment; these are probably the least commonly used interaction type in ABM.

2.2 Types of a ABM

There are several definitions of ABM based on the complexity of the models they can handle. It would be senseless to spend time defining and implementing a FSMM for the most basic ABM approaches (as they could be trivially modeled in any ABM framework). Macal [59] provides four definitions in increasing complexity:

- An **individual** ABM is one in which the agents in the model are represented individually and have diverse characteristics.
- An **autonomous** ABM is one in which the individual agents have internal behaviors that allow them to be autonomous, able to sense whatever condition occurs within the model at any time and to act with the appropriate behavior in response.
- An **interactive** ABM is one in which autonomous agents interact with other agents and with the environment.
- An **adaptive** ABM or dynamically constrained is one in which the interacting, autonomous agents change their behaviors during the simulation, as agents learn, encounter novel situations, or as populations adjust their composition to include larger proportions of agents who have successfully adapted. It is also possible for the rules of the environment to change during the simulation.

The aim of this work is to identify a FSMM and a reference architecture for the most complex version of ABMs, the *dynamically constrained* ones.

2.3 Complexity and emergent behavior in ABM

There is substantial literature about complexity and emergent behavior. For the purposes of this work, we will only mention two topics related to ABM: the architecture of complexity and emergent behavior [20]. First, the architecture of complexity indicates

that complexity is usually structured in a hierarchical way and, in many cases, there is a massive repetition of simpler structures [86].

Also, in most cases it seems that complex structures are achieved by starting with a simple stable state and modifying it. One way to identify a sub-system in this hierarchy is the use of nearly decomposable matrices, which are matrices that describe the relationship between elements of a system. They are organized in a way where highly interacting elements are grouped closely and poorly interacting elements are far away, whereby islands of interaction may be identified as a sub-system.

Another essential concept in "complexity science" is "emergence" or emergent behavior [6] [95]. Topics such as the emergence of cultural norms or institutions from the interactions of individuals' activities are critical and not well addressed by other competing modeling formalism. Emergence is fundamentally a multi-resolution concept with, as has been noted, micro-motives leading to macro-behaviors [84] [41]. Thus, emergence can be characterized by a measure of macroscopic behavior achieving a threshold value in a simulation built from microscopic behavior [38]. The initial approaches to ABM only generated the agent metrics for a person to analytically identify if there was an emergent behavior. While human insight should not be underestimated, the amount of data that ABM simulations may generate is vast, and most of it is useless, limiting the ability of a human to reason about emergent behavior [42].

One of the most important reasons for using ABM is the impossibility – for economical or ethical reasons – to have the critical amount of participants required to observe emergent behaviors, which could be very different from behaviors exhibited by few participants [39]. One example of this is the Ultimatum game, in which one party, the proposer, is endowed with a sum of money. The proposer is tasked with splitting it with another party, the responder. Once the proposer communicates their decision, the responder may accept it or reject it. If the responder accepts, the money is split as per the proposal; if the responder rejects, both players receive nothing. Both parties know

in advance the consequences of the responder accepting or rejecting the offer.

If the parties are played by two individual players, the proposer will usually propose a fair (50/50 or just a slightly bigger part for the proposer) out of fear of the responder rejecting the proposal (in that case both get nothing) if the responder does not feel it just [40]. Nevertheless, if parties are played by a huge amount of interconnected players, an emergent behavior of accepting a hugely unequal proposal will appear. It could be said that functional but highly unequal societies can be compared to the responder party accepting the unfair proposal.

2.4 Macroscopic Statistical Analysis and fidelity to reality

ABMs function as an *in silico* laboratory in which the researcher inputs agent characteristics, specifies initial conditions, applies rules for agent-agent interactions, and programs static or transient rules to move among model states.

For non trivial models they are many ways to implement its agents' behaviors. Researchers, who wish to maximize the understanding of a complex system as clearly as they possibly can, must introduce several behaviors for its agents, to identify which ones match better empirical macroscopic results. This approach is called a pluralistic methodology and has been mentioned by several authors [33].

Comparisons metrics for some macroscopic simulation are desired, disregarding the models' internal dynamics. For example, there could be two models to measure a pandemics contagion rate, but the first one is much more resource-consuming than the second one because it simulates human behavior in more detail. If the identified contagion rates of both models are similar, regardless of the implementation, it can be said the models are macroscopic comparable, and the second one will be desired for large-scale simulations as it is less resource-consuming.

Another important concept in ABM is fidelity to reality, which can be seen as a two blocks chain, with fidelity of the specification to reality and fidelity of the implementation to the specification. The first point is not developed in this work but there are several papers about this topic like [80].

For the second point, it is important to mention that there is no such thing as "the test" to validate (increase the confidence that an inference about a simulated process is correct for the actual specification) a implementation [44].

One way to tackle correctness and fidelity to the specification is to use macroscopic statistical analysis over multiple implementations of of the same specifications in different frameworks. To do that, a list of quantitative macroscopic observables needs to be defined which should be provided by each implementation. Then, for each implementation, the simulation can be run several times to obtain statistical behaviors of each observable.

Observables can be gathered via different approaches. An observable could be an environment value, the aggregate of an agent's values or some ratio when the simulation completes or reaches a milestone. It could also be the amount of rounds it took to reach some specific status.

When statistical behavior of an observable differs among implementations, it could be seen as as an alert for one or more of the implementations not following the specification correctly in some desired area. Likewise, consistent statistical behavior among implementations increases the confidence that all of the implementation follow the specification correctly.

In this work, macroscopic statistical analysis was used to validate that for one of the experiments, all implementation produced the same statistical behavior for three defined macroscopic observables in 10,000 executions of the Pareto experiment.

2.5 Counterfactuals in ABM

Counterfactual reasoning closely relates to ABMs. Concretely, counterfactuals provide baselines used to evaluate possible alternatives to life events that may occur or have already occurred; something that is contrary to what actually happened [79]. Based on this point of view, counterfactual support means the ability of the ABM framework to compare outcomes of simulations under different input parameters (collectively referred to as a "treatment") from any number of hypothetical scenarios [61]. A specific example of counterfactual support in ABM will be the case where a model starts with an initial setting, and at some specific point of the simulation, it branches into two different simulations, the first one following one set of settings and the second one following a different set.

One example where branching could be useful is the example where the model needs to reach some desired state before two or more rules are applied. For example, in the case of the COVID-19 contagion rate model, it might be desired to branch the model by applying different restriction rules with different levels of enforcement at different moments of the epidemic to explore the evolution of the contagion rate.

In summary, macroscopic comparisons are useful to compare desired outputs of models with different inner implementations and comparable initial states and counterfactuals are useful to compare desired outputs of models with similar inner implementations but different settings.

2.6 The need for ABM frameworks

While there are famous agent-based models like Game of Life (GOL) which can be easily coded in any non-trivial programming language [35], the facts are that many social models are so complex that implementing them from scratch will not be feasible. Most ABM share a common feature space, which makes it desirable to have general

support for modeling. One of the earlier examples of this support is the Simula 67 programming language, developed in the mid-1960s and widely used for research by the 1970s. It was the first language aimed at automating step-by-step agent simulations [71], and was released along with other useful features like objects, classes, and inheritance, yet it was a *programming language*, not a framework, and had important limitations to implement non-trivial models.

ABM frameworks were developed to reduce the semantic gap between model specifications and programming language capabilities. Over the years, numerous ABM frameworks have been created to meet various modeling needs, ranging from lightweight frameworks with limited simulation capabilities to highly sophisticated ones with custom languages tailored for simulation. Despite the availability of many ABM frameworks today, there is no agreed-upon formal specification for the ABM feature space. This lack of standardization hinders the ability to compare different frameworks effectively. Frameworks that rank higher on the FSMM are expected to facilitate more robust ABM development and achieve higher fidelity to real-world scenarios. Additionally, a FSMM could serve as a baseline for future specifications of ABM frameworks. The next section presents a list of the most critical elements of the feature space for ABM frameworks, identified during the preparatory work.

2.7 Framework limitations

Some of the most important identified deficiencies in ABM frameworks is the lack of both a FSMM and a reference architecture. This leads to several issues, including the inability to rank frameworks based on simulation capabilities, and the need to start the specification of future ABM frameworks from scratch. This work proposes designing an initial maturity model, and use it for creating a reference architecture and a small POC implementation to validate its quality.

Usually, maturity models have six maturity level per focus area [9]. Nevertheless some levels, like repeatable and managed, apply only to organizations, not to specific software systems. Therefore, our proposal for the FSMM will follow the facilitator-based approach of Núñez-Corrales and Gasser [72], which is the capability of the framework to provide a desired feature, from (0) no facilitators, (1) facilitators need technical involvement for each case, (2) mature general facilitators require low technical involvement, to (3) facilitators require no technical involvement. The FSMM consists of three sections: simulation features, interrogation capability, and counterfactual support, as defined previously. The sections are split into basic features. Examples or descriptions are provided to illustrate what it means for a framework to be assessed at some level for each feature.

The proposed reference architecture aims to achieve a high score in every feature of the FSMM while providing robust Computational properties. Finally, a small POC has been developed to evaluate the reference architecture. The evaluation consists of comparing four representative models implemented in the POC versus implementations built in the other five ABM frameworks under study.

CHAPTER 3

ABM Feature Space Maturity Model

"Humans are good at discerning subtle patterns that are really there, but equally so at imagining them when they are altogether absent"

Carl Sagan, Contact

The FSMM proposed in this work has three sections: simulation features, interrogation capabilities, and research supporting features. These features may overlap, yet an effort has been made to reduce the overlapping between them.

3.1 Simulation features

Simulation features are the most important criterion in the FSMM. The primary goal is to identify fundamental features that enable a clear implementation of models and ensure high fidelity to reality.

A standard high-level programming language like Java, without any third-party libraries, would be considered a framework scoring 0 in all categories.

The simulation section contains six features, as follows:

3.1.1 Usability

Usability was included to reflect how easy it is for an average user to install and use the framework. Although it is not formally part of the simulation category, it was added due to its significant impact on the overall use of the framework.

0. No way to install. User needs to compile from source code. Very limited documentation.
1. Difficult to install, important stability issues. Basic documentation.
2. Generally stable but some minor stability issues. Useful and well-written documentation.
3. The framework is stable and easy to install on Windows/Linux/Mac. Documentation describes all functionalities, and includes several working examples.

3.1.2 Basic framework functionality

This topic relates to the baseline simulation features which are provided by the framework and are used in a wide array of cases, like data processing and communication

with external tools. It also includes features which could help validating whether a model or simulation is acceptable for use for a specific purpose [83]. They include debugging tools like visualization, breakpoints, execute while some expression is true – all of which contribute to increase the accuracy of the model [5].

0. Basically no support of standard framework functionalities. No support for random distributions, no ability to save simulation status. No support to read, write structured data files like CSV or JSON. No ability to stop simulation.

1. Ability to read, write files. Tools to handle structured data like CSV, JSON files.

2. Ability to communicate with other tools for sending and receiving data, like http/https support.

3. Ability to run simulation for certain amount of steps or while a condition is true. Ability to pause simulation interrogate agents or the environment.

3.1.3 Locality

Agents should be space-aware and should have a direction and perspective of their environment: front, sides and back. Spaces could be continuous or discrete. The expected space shapes are planes, cylinders and torus. Any agent can be in only one point/cell at a time and is considered to be at the center of the point/cell. On their round, the agents can modify only their visible surroundings. This is useful for avoiding global effects which could affect the simulation quality. Also there could be inaccessible regions i.e. Agent can not move to that region.

0. All locality concepts needs to be implemented, agents do not have front, no support for neighborhood access. No support for space concept (neither grid nor continuous).

1. Basic locality functionality, like support to translate position to torus, tube, plane.

2. Support for spaces of both grid and continuous types.

3. Agents can move forward, backward, rotate. The space shape could be square, tube, torus. Space could be grid or continuous or both.

   A radial neighborhood of range r consist of a list of elements with the euclidean distance $\leq$ r.

   A Moore neighborhood defined on a two-dimensional square lattice and is composed of a central cell and the eight cells that surround it. An extension of this definition is the Moore neighborhood of range r. It means the set of points at a Chebyshev distance of $\leq$ r [99].

   A von Neumann neighborhood (or 4-neighborhood) is classically defined on a two-dimensional square lattice and is composed of a central cell and its four adjacent cells. An extension of this definition is the von Neumann neighborhood of range r. It means the set of points at a Manhattan distance of $\leq$ r [100].

3.1.4 Agent Management

Agent management corresponds to the ability to handle the administration of agents, including the creation, destruction and – after the agent dies – its remaining management.

0. No agent concept, agents concept needs to be implement from scratch using other concepts like structures or objects.

1. There is a basic concept of agents. Agent can be created but extra structures or logic is required to manage them. Agents need to be added to the scheduler and logic is required to move them in the grid, specially in a seamless grid.

2. Full fledged support for agent management. Agent creation adds agents to the space and an agent death removes the agent from the space. There is the concept of agent type: the ability to list all live agents by their types. Move actions move

agents in space without need for extra structures. There are abilities to search nearby agents using radial, Moore and von Neumann neighborhood approaches.

3. There is a standard way of communicating between agents in the mail-box approach [17] where the sender sends the message during its turn and the receiver will see all available messages (in its mail-box) at the beginning of its turn. There may be two types of messages: one-to-one and broadcasting.

3.1.5 Force fields

Force fields are not very common in ABM and are used only in very specific simulations [60]. Nevertheless, they are a powerful tool and it is plausible that its lack of use is not related to their benefits, but more probably to their general lack of support in well-known ABM Frameworks.

0. No force fields concept. To calculate forces extra data structures and algorithms (coding) are required.

1. Basic support of force fields concept. Ability define a force field emission point and ability to calculate distance (shortest distance in seamless grids like torus grid) and angle from any point to the force field point.

2. Agent and environment can assign and reset force fields naturally. Forces have type and force intensity can grow or decay linearly or quadratically. Multiple types of forces and each emission point could have either positive or negative values. The average of forces of the same type can be calculated in any point of the space (including seamless grids like the torus). Forces can have a radial range limit.

3. Ability to visualize forces as bump/heat maps with varying level of detail. Ability to show or hide forces by type/layer.

3.2 Interrogation capabilities

Interrogation capability is the second most important criterion in the FSMM, especially interrogation support around the concept of "emergent" behavior. Statistically, we understand *macroscopic support* as the ability to define a set of metrics that will be persisted while the rest of the simulation data will be discarded. Also, macroscopic metrics may be defined regardless of the micro-motives implementation. If sets of metrics are implemented for two or more models, they are considered macroscopic-comparable for interrogation. For this criterion, the four features considered are:

3.2.1 Visualization

Visualization for ABM, or any simulation in general, is a very important requirement, as the amount of data generated by the simulation is huge. Visualization is also useful tool for debugging the model.

0. No visualization support. Data needs to be displayed using third party libraries.
1. Limited visualization tools. Usually slow, poorly designed but functional.
2. Fast, well documented visualization. Ability to disable visualization globally during execution
3. Comprehensive and well-documented visualization support, ability to visualize both detailed and statistical data. Ability define visualization layers by agent, items and force field attributes.

3.2.2 Statistical support for model attributes

Statistical support for model attributes: It is the ability of the framework to define and track a set of metrics for a single execution and discard the rest of the data.

0. No statistical tools. All stats needs to be calculated and charted using third party tools.

1. Basic support for statistical analysis. The main expected metrics are average, moving average, standard deviation, maximum, minimum, 95 and 99 percentiles (and in general the nth percentile). Basic ability to create charts.

2. Well documented and easy way to create and configure charting (ability to define title, axis data, minimum and maximum for grid, etc).

3. Support for powerful statistics tools which behave like consumers to create correlation graphs.

3.2.3 Support for macroscopic statistical analysis

Support for macroscopic statistical analysis of observables in different implementations. With this analysis it is possible to identify specification deviations of different implementations. Identifying and fixing those deviations should, in turn, increase the overall fidelity to reality of the implementation. Macroscopic statistical analysis includes standard statistical comparisons between two or more simulations, F-test, histograms and ANOVA [66].

0. Single execution only. All stats will need to be saved manually in any execution for further comparison.

1. Execute two simulations at a time and compare macroscopic observables using and F-test in real-time.

2. Create and orchestrate multiple simulations on different runners. Compare statistical behavior of macroscopic observables using histograms or ANOVA.

3. Tools to collect, reconcile and statistically analyse the behavior of macroscopic observables of different simulations. Those results would likely be stored in files and could be created by implementations in different frameworks.

3.2.4 Nearly decomposable system view

There should be a way to identify regions of the model which high internal interactions and poor external ones.

0. No concept of nearly decomposable system view. Identification of high internal and poor external interaction needs to be implemented manually.

1. Ability to identify regions with high amount of physically close agents in any space type (torus, plane, tube).

2. Ability to identify highly interacting agents through messages.

3. Ability to visualize using tools like heat-maps to distinguish the highly interacting regions.

3.3 Research supporting features

Research supporting features consists of a set of three features. As far as we know, their support is lacking in ABM frameworks. Such support might benefit ABM frameworks to reduce the semantic gap between the specification and the implementation. They are graph support, counterfactual and utility features.

Due the novelty and complexity of implementing experiments which require these features, they will not be used as scoring topics, but are briefly described here as a basis for further research.

3.3.1 Graph support

Graph theory offers a wide range of tools and metrics for analyzing networks. These can be applied to understand various properties of the agent network, such as centrality, clustering, and path lengths, which can provide insights into the dynamics of the modeled system. [73].

Graph support in ABM makes it applicable to a wider range of fields, such as sociology, epidemiology, economics, ecology, and engineered complex systems – where network structures are fundamental. This broad applicability enhances the utility and relevance of ABM.

0. There is not support for graphs, neither logically, nor in visualization.

1. Basic graph capabilities, including weighted graph, and path finding; ability to visualize graphs and relations as arrows or lines.

2. Ability to assign agents to graph edges, not only to the grid. Ability to have graph edges following a curve. Ability to set the percentage of the edge covered by the agent.

3. Environment with Multi-grid support. Edges can move agent from between grids and grids could be stacked to simulate buildings floors, section of space stations, etc.

3.3.2 Utility features

For experiment implementation it is usually required to have a map to translate experiment types and magnitudes to implementation values, which could generate friction during the implementation. It is expected that providing a way to implement the experiment closer to the specification will lead to a higher fidelity to the specification.

0. There is not support for research supporting features, there is the need for a map to translate specification values to implementation values.

1. Basic support for units of measurement, like ability to define rounds in time scales (seconds, minutes, hours), distances in metric units (centimeters, meters, etc), and other like temperatures, light intensity, etc. Provide basic arithmetic using translation features like "3 m + 50 cm" are "3.5 m". It should also validate the data types involved in sub-expressions and throw an error for expressions like "3 s + 3 m".

2. Ability to define resources, and query them using set theory methods. This simple idea could easily represent complex behavior. For example, in the second experiment – the Pred-Prey –, defining the behavior of the wolf with the rabbits in sight may be that the wolf will walk in direction to the closest rabbit.

$$\text{wolf} = \text{self}$$

$$\text{visible} = \{r : \text{rabbits} | \text{dist}(\text{wolf}, r) \leq \text{wolf.vision}\}$$

$$\text{chosen} : \text{visible} \bullet \forall\, r : \text{visible} \bullet dist(\text{wolf}, \text{chosen}) \leq dist(\text{wolf}, r)$$

3. Ability to define an environment with several layers of grids. Each grid could have inaccessible cells, whcih means agents should not be able to move into that cell, nor used in any way during the simulation (like finding neighbors through that cell). Grids could be standard square grid or have a more complex layout like a Voronoi topology [16].

3.3.3 Counterfactual support

Counterfactual is a relatively new idea for ABM and – as far as we know – its support is nonexistent in ABM frameworks. This is the ability in the framework to define two or more execution of the same model with different initial parameters, or to branch a new executions from a previous one with new set of parameters. Then it would be possible to perform some macroscopic comparisons among those executions.

Model branching for multiple possible worlds generation: This is the ability to describe a trigger where a simulation will split into two or more simulations, each of them with the same initial history but with new parameters. Because of the risk of branch explosion, control settings, like max amount of branches, should be provided.

0. Single model during whole simulation. Model branching at the beginning or during execution needs to be completely implemented from zero.

1. Ability to save a simulation status then, in a different process read, pre-process and run the updated model.

2. Ability to orchestrate multiple executions in a list of runners with their initial set of parameters.

3. Ability to branch a model when some rule has been activated (triggered). Ability to manage executions and orchestrate executions in new runners. Ability to remove macroscopic similar executions (those having similar macroscopic behavior).

CHAPTER 4

Experiments on Frameworks

"Divide each difficulty into as many parts as is feasible and necessary to resolve it."

René Descartes, Discourse on Method

This chapter starts by describing the representative ABM frameworks to be evaluated, followed by a detailed description of each experiment performed during our research.

4.1 Frameworks to be evaluated

The research approach followed resembles a waterfall. There were three main stages: research around the feature space needed to create the FSMM, then designing a reference architecture with robust Computational Properties, and finally, the evaluation of the reference architecture performed via the POC and *in-silico* experiments.

During this research, five ABM frameworks were chosen to be evaluated according the Feature Space Maturity Model because of their usage share in ABM related research, documentation quality, historical importance, and ease of access. Swarm was one of theABM Frameworks initially considered for evaluation. Swarm [18] was released on November 1997 and its last stable release (version 2.4.1) is of April 2009. Due to the high number of incompatibilities, we decided to drop it early in the research.

The ABM frameworks selected for this research are: NetLogo, MASON, Repast, Mesa, and Agents.jl. A short description of each one follows.

4.1.1 NetLogo

NetLogo is a multi-agent programmable modeling environment. It is used by hundreds of students, teachers, and researchers worldwide. It was developed for domain experts without a programming background to model related phenomena [101]. It was first released on 1999 and its most recent stable version was 6.3.0 released on 8 December 2021.

4.1.2 MASON

MASON is a fast discrete-event multi-agent simulation library core written in Java which is a well known and powerful programming language [37]. It is designed to be the foundation for large custom-purpose Java simulations, and also to provide sufficient functionality for many lightweight simulation needs [56]. MASON Stands for Multi-Agent Simulator Of Neighborhoods (or Networks). It is intended to provide a core of facilities useful for the Social Sciences as well as to other agent-based modeling fields such as Artificial Intelligence and Robotics. The latest release of MASON was version 20 on August, 2019 and requires the agents to be developed in Java. Simulations need to extend from the class sim.engine.SimState; being just a library, it requires steps management to be handled by the developer.

4.1.3 Repast

The Repast Suite is a family of advanced, free, and open-source agent-based modeling and simulation platforms that have been under continuous development for over 20 years. Its most used version is Repast Simphon 2.10.0, released on 22 December 2022. Repast is a richly interactive and easy-to-learn Java-based modeling toolkit that is designed for use on workstations and small computing clusters [69].

Repast uses Groovy – a wrapper over the Java programming language [7] usually distributed as a Plug-In for the Eclipse Integrated Development Environment (IDE) Open Source Platform [15]. While not explicitly mentioned in their documentation, it appears that Repast re-implements all of NetLogo's calls.

4.1.4 Mesa

Mesa is a modular ABM framework for building, analyzing, and visualizing agent-based models. Mesa is written in Python [93] and keeps its modeling, analysis, and

visualization components separate – though they are intended to work together. Most models consist of one class to represent the model itself; one class (or more) for agents; a scheduler to handle time (order in which the agents act), and possibly a space for the agents to inhabit and move through [50].

There is a module which allows Mesa to run as a server and the UI can be displayed in a standard browser like Firefox [67].

4.1.5 Agents.jl

Agents.jl is a framework for ABM fully implemented in Julia, a modern and powerful programming language [13] for high-performance computing. It's a free, open source, transparent, intuitive and simple-to-learn high quality software, with extensive documentation. It is modular and has a function-based design. It supports many types of space: arbitrary graphs, regular grids, continuous space, or even instances of Open Street Map. [22].

4.2 Experiments

During this work, four experiments were performed, each one covering different combinations of features used in ABM. The experiments are: Sugarscape, Pred-Prey, Pareto Principle, and Covid propagation model.

4.2.1 Sugarscape Model

Sugarscape is a model for artificially intelligent agent-based social simulation following some or all rules presented by Joshua M. Epstein & Robert Axtell in their book *Growing Artificial Societies* [27]. There are several variants but all Sugarscape models include the agents (inhabitants), the environment (a two-dimensional grid), and the rules governing the interaction of the agents with each other and the environment.

The original model is based on a 51x51 cell grid, where every cell can contain different amounts of sugar (or spice). In every step, agents look around, find the closest cell filled with sugar, move, and metabolize. They can leave pollution, die, reproduce, inherit sources, transfer information, trade or borrow sugar, generate immunity, or transmit diseases - depending on the specific scenario and variables defined at the set-up of the model. Sugar in simulation could be seen as a metaphor for resources in an artificial world through which the examiner can study the effects of social dynamics such as evolution, marital status, and inheritance on populations [27].

What is modeled?

As mentioned, this model may grow huge, so the basic model is a 150x200 torus grid with 150 agents of both sexes (75 males and 75 females) randomly scattered in the grid. Agents can overlap in the grid and can choose to reproduce, which always succeeds. Offspring are considered adults immediately after being spawned and will have in average half of their parents values but same energy as the mother.

Agents consume sugar of just one grid per round and have the ability to look for mates and sugar based on sight which is the same as speed (i.e. in their turn the can walk as far as they can see), consume sugar based on metabolism, and have a growing probability of dying of natural causes (also the agent could also die if its sugar is less than its metabolism).

Therefore the agents have 5 main static parameters, normalized for simplicity

- Max-Energy: A value from 0 to 1, indicates the max allowed energy the agent can consume. It is assigned randomly using the Normal curve, average 0.9 and standard deviation 0.1 .

- Metabolism: The energy the agent consumes per round. It is assigned randomly using the Normal curve, with average 0.2 and standard deviation of 0.1.

- Range: a value which indicates how much the agent can see and move in the grid. It is assigned randomly using the Normal curve, with average 10 and standard deviation of 1.0.

- Mate-need-grow-rate: A value which indicates the grow of the need to mate per round. It is assigned randomly using the Normal distribution, average is 0.1 and standard deviation is 0.01 .

- Death-grow-rate: A value from 0 to 0.1 It indicates the grow of the natural death rate. It is assigned randomly using the Normal distribution, average in 0.09 and with standard deviation 0.01 .

Those four parameters are floating-point values and are to be assigned randomly to the first agents and be averaged for offspring.

Therefore, the agents have 3 counter parameters, which accumulate during execution

- Curr-Energy: A value from 0 to Max-Energy: It shrinks per round based on the metabolism and grows based in the energy consumed in the patch. New agents start with Max-Energy, and for spawned agents, the start of Curr-Energy/2 of their mother.

- Mate-need: A value from 0 to 1. It indicates the need to mate per round. It starts in 0 and grows by Mate-need-grow-rate. It is reset to 0 if the agent mates successfully.

- Natural-death-rate: A value from 0 to 1. It indicates the probability of death, it grows by turn based on the death-grow-rate.

There is a difference in the use of Mate-need: for male agents it indicates the need to search for a female to mate. For females, it indicates their acceptance of a mating proposal.

The environment is divided in patches and each patch has 3 parameters

- Max-Energy: A value from 0 to 1, indicates the max allowed energy an agent can consume in this patch. It is assigned randomly using normal curve, average 0.9 and standard deviation 0.1 .

- Grow-Rate: A value from 0 to 1. The energy the path grows per round. It is assigned randomly using the Normal curve, with average 0.04 and standard deviation of 0.001.

- Curr-Energy: A values from 0 to Max-Energy. It grows by grow-rate and shrinks if consumed by and agent.

There is an important concept of mating decisions which is used for both males and females, but in different ways. The mating decision threshold is based on the amount of stored energy (Curr-Energy / Metabolism) and the Mate-need value. The probability to choose mating grows with the mate-need and stored energy.

$$Mate\_probability = Stored\_Energy * Mate\_need$$

There is a small difference in males and female behaviors The rules of the agents are as follows:

- At the beginning of a round, a male agent will decide whether he looks for a female to mate with, based on the mating probability. If mating is chosen, the male agent will look for any available female within his range of action and, if found, he will go and propose mating; if rejected, he will feed from that patch if available, but if consummated, he will skip feeding as courtesy for the impregnated female. In the case where no female is found, the male agent will choose a random direction and will go that way as far as he can, in pursuit of a female, once he reaches there, he will feed from that patch. If he chooses feeding instead of mating, it will follow the standard feeding process.

- The female accepting of rejecting a mate offer is also based on the mating decision.
- At the beginning of a round of an impregnated female, she will spawn a new agent with its 5 parameters set based on her own and the father's parameters, using a Normal curve of avg 0.5 and 0.1 of standard deviation and its sex chosen at random. Then it will consume half of its Curr-Energy. After that, it will follow the standard feeding process. A non-impregnated female agent will just follow the standard feeding process.

The standard feeding process consists of choosing the patch with the highest amount of sugar; if there is a tie, the agent will go to the first one found, get there and consume sugar until Curr-Energy reaches Max-Energy.

Sugar will be refilled at a fixed rate in each grid but will not accumulate if unused.

What is monitored?

Being an evolution approach it is interesting to measure the average of agents parameters, it is expected that Max-Energy, Range grow while Metabolism and Death-grow-rate shrink in average.

The Mate-need-grow-rate is more tricky as in both extreme values could cause population collapse. A high value could cause death by starvation in both males searching for females and in females giving birth. A value too low will cause a lack of new spawned agents.

Which criteria of the FSMM can this experiment help evaluate?

Modeling the Sugarscape experiment helps evaluate the following features of the simulation criterion:

- Usability
- Basic framework functionality

- Locality
- Agent management

Modeling the Sugarscape experiment helps evaluate the interrogation capabilities criterion as per the following features:

- Visualization
- Statistical support for model attributes.

4.2.2 PredPrey Model

The Lotka–Volterra equations, also known as Predator-Prey (Pred-prey for short) equations, are a pair of first-order nonlinear differential equations, frequently used to describe the dynamics of biological systems in which two species interact, one as a predator and the other as prey. The model makes several assumptions that might not be valid. In general, the approach can apply to "organic systems" using a plant species and a herbivorous animal species as an example.

The advantage of this model is that both the behavior of the predators and prey are related to a single set of equations, shown next:

$$\frac{dx}{dt} = \alpha x - \beta xy$$
$$\frac{dy}{dt} = \delta xy - \gamma y$$

where:

x is the number of prey (for example, rabbits), y is the number of some predator (for example, wolves); $\frac{dy}{dt}$ and $\frac{dx}{dt}$ represent the instantaneous growth rates of the two

populations; t represents time, and parameters α, β, δ, γ are positive real numbers describing the interaction of the two species.

What is modeled?

As mentioned, the model grows quickly, so the basic model is a 500x500 torus grid with 25 wolves and 250 rabbits. Both wolves and rabbits reproduce asexually. Rabbits reproduce exactly after 15 rounds and predators reproduce after a successful hunt. If in its turn a predator can reach a rabbit, it will kill (eat) it by going next to it.

There is no overlapping restriction for the grid and both wolves and rabbits could see any agent in a radius of 20 units around them and move a maximum distance of 10 units. Wolves will die if they do not eat in ten rounds (time to live goes to 10 after eating a rabbit and decreases each turn the wolf does not eat). There is no metabolism for rabbits, their only mission is to escape wolves.

Both wolves and rabbits will be guided by forces. Forces are appliedby agents only in their visible radius.

Rabbits have four forces

- Desired force: A force of defined random intensity from 0 to 1 which continuously changes each round (up to 45^{o}, from 22.5^{o} left to 22.5^{o} right).
- Flock force: A force to be near to its flock. It grows linearly to the euclidean distance of each rabbit and has a initial intensity of 1.
- Personal space force: A force against being to near to other rabbits. It falls linearly to the euclidean distance and has a initial intensity of 40.
- Fear force: A force against the predators. Its intensity is 200 and doesn't fall and is 0 if there is no visible wolf.

Wolves have three different behaviors

- No rabbit in sight: When there is no rabbit in vision range, the wolf will walk their maximum distance in random direction in search for rabbits.
- Rabbits in sight but not one in reach: When there are one or more rabbits in sight but not in walking distance, the wolf will walk in direction to the closest rabbit.
- At least one rabbit within reach: When there is a rabbit in walking distance, the wolf will go next to rabbit and eat it. The rabbit dies and the time to live of the wolf goes back to the maximum of 10 turns.

What is the goal of the simulation?

We seek to evaluate several criteria of the FSMM concept, specially forces in the ABM frameworks and try to validate the amount of predators and preys that follow the Lotka–Volterra equations. Data collected per round will be the amount of live wolves and rabbits and their relative percentages to the total population.

Which criteria of the FSMM can this experiment help evaluate?

Modeling the Pred-prey experiment helps evaluate the following features of the simulation criterion:

- Usability
- Basic framework functionality
- Locality
- Agent management
- Force fields

Modeling the Pred-prey experiment helps evaluate the interrogation capabilities criterion on these features:

- Visualization
- Statistical support for model attributes.
- Nearly decomponsable system view

4.2.3 Confounding galore in Pandemic times

The global impact of COVID-19 has been profound, and it is still a public health threat even with the current generation of vaccines. The Imperial College of London developed a model around the so-called Non-Pharmaceutical Interventions (NPI) – aimed at reducing contact rates in the population and thereby reducing transmission of the virus. Their results drove the behavior or several countries during the peak of the pandemic. They conclude that the effectiveness of any one intervention in isolation is likely to be limited, requiring multiple interventions to be combined to have a substantial impact on transmission [31].

Another main difficulty in addressing the pandemic was the significant amount of misleading information around the disease [32]. Due the global impact of social networks, this experiment focuses on modeling three main options on a pandemic, the best case with only good NPI and first generation of vaccines, one with worst case, with only contagion-increasing measures, and one where users decide to use different options and influence others.

What is modeled?

This model can grow quickly, so the basic model is a list of 100x100 tiles. Each tile has one agent. On every round, each agent has and initial 21% probability to get sick. Each agent has one measure to lower its probability in 10% to become sick (using a face mask) and two measures to increase its probability to become sick in 10% (say being less careful by believing it is immune due to wearing amulets), and a measure which

does not have any effects (say drinking a glass of orange juice daily).

So the minimum probability to get sick is 1% and the maximum is 21%, yet agents do not know a-priori which measures are useful and which measures are useless. Agents do not die and only get sick for the duration of the round.

Each agent has an initial trust level for each measure from 0% to 100%, where 0% means total lack of trust and 100% means total trust, and chooses to use the measure depending on random chance being lower of trust level.

Each agent also has a list of 10 forwarding message friends and can have 0 or more receiving message friends. The messages contains trust % for each of the available measures.

At the beginning of the turn, each agent consumes all its incoming messages together with the incoming friends' trust level, if any, then re-balances its trust levels using a weighted average. A weighted average is a average where some data points contribute more than others [12]. The simulation will be run twice, the trust contribution of the influencers in the first round will be only 2% of the weighted average and in the second round it will be 3% .

Then, the agent decides which measure it will use for the round. Then its sickness probability and sickness status are calculated.

If the agent gets sick, it again re-balances its trust level by lowering its trusts by 20% in the measure it is using and increases its trusts by 1% for the measures it is not using. In the case they do not get sick, they increase by 10% their trusts for the measures they are using and lower by 1% the trusts for the measures they are not using.

Finally, at the end of its round, the agent forwards its new trust level to its 10 forwarding friends.

The simulation is run for 1200 rounds, and the data to be plotted is the average trust level of each of the four measures.

For this experiment, the initial trust level and influencers for the agents should be

collected from a http call to a Web service which sends a JSON file with the trust level for the 10,000 participating agents. Also at the end of a simulation, each framework should send a JSON file indicating the average % of agents which got sick per round.

The format for the incoming JSON message is:

```
[
  {
    "inactiveReinforce": 1,
    "activeReinforce": 10,
    "juice": x,
    "amulet": y,
    "mask": z
    "influences": [
      {
        "agent": id,
        "power": power0to1
      }, ...

    ]
  },...
]
```

x, y, z and w are integers from 0 to 100, the value agent is an id of the influencer and the power is the power of the influencer over the agent.

The format for the outgoing JSON message is

```
{
  "framework":x,
  "sickRatio": [y1,y2,...,y1250]
}
```

Here, x is the framework name and each y is the sick ratio per round.

The color schema of the agent is thus: the more brightness the less the chance of getting sick. If the agent chooses the risk lowering options and did not choose the risk increasing option it will be very bright, and the other-way around it will be very dark. Choosing the effect-less measurement will change the tint from green to orange.

The srever for this experiment was implemented here `http://localhost:9000/stats?size=SIZE&influencerCount=INFLUENCER_COUNT‘`, where SIZE is the amount of agents and INFLUENCER_COUNT indicates how many agents each agent will influence.

What is the goal of the simulation?

The goal of the simulation is to understand the basics of confounding variables in infection dynamics. It is also used to exercise the ability of the frameworks to communicate with other tools using the standard http protocol.

Which criteria of the FSMM can this experiment help evaluate?

Modeling the Confounding galore experiment helps evaluate the following features from the simulation criterion:

- Usability
- Basic framework functionality
- Agent management

From the interrogation capability criterion, these are the features to be evaluated:

- Visualization
- Statistical support for model attributes

4.2.4 Economic Wealth Inequity (Pareto Principle)

The Pareto principle states that for many outcomes, roughly 80% of consequences come from 20% of causes (the "vital few"). Other names for this principle are the 80/20 rule, the law of the vital few, or the principle of factor sparsity. It is based on the works of Italian economist Vilfredo Pareto, who wrote about the 80/20 connection while at the University of Lausanne. In his first work, Pareto showed that approximately 80% of the land in Italy was owned by 20% of the population [75]. An example is in the 1992 United Nations Development Program Report, which showed that the distribution of global income is very uneven, with the richest 20% of the world's population receiving 82.7% of the world's income which follows Pareto's principle [92].

This inequality in the wealth distribution seems to emerge inevitably in interacting environments even in initial equitable starting conditions due to the statistical mechanics of populations and Gini coefficient [102]. The Gini coefficient measures the inequality among the values of a frequency distribution, such as levels of wealth [36].

What will be modeled?

As mentioned, this model could grow huge, so the basic model will be similar to a Monopoly game. It will be a 100x100 grid with 100 agents. Each tile has a cost to buy (in coins) and will bring a rent for the owner if another agent steps in. Each agent starts with no coins but will collect a static number of 75 coins per turn.

The rent per tile is calculated from a random number between 1 and 50 coins, and the cost of purchasing will be that value multiplied by ten. Those values do not change during a simulation execution.

Tiles can only belong to one agent, and during its turn, the agent first will collect the static 75 coins, then it will move to another tile in a random position. If the tile is not owned, and the agent has enough money to buy it, it will buy it. If the tile is owned and the agent can pay the rent, it will pay it to the owner of the tile.

Finally if the agent cannot pay the rent, it will have to pay an initial amount using its available coins, and then, pay the rest by giving the tile owner some tiles it owns. It could be that agent's tiles cannot match the missing rent, then the agent will have to surrender all its tiles, but the rest of the rent will be forgiven. There is also the case were the list of surrendered tiles will exceed the cost of the missing rent pay, but the owner will not pay the change.

Calculating the list of tiles to pay the rest of the rent, where the excess is minimal, is a optimization variant of the subset sum problem, which is a known NP-hard problem [51], therefore a quick (greedy) approximating algorithm will be used. First the agent tiles are sorted based on its cost from cheaper to more expensive. Iteratively, they tiles

will be added to an initially empty bag until the amount is reached or exceeded; thus, the cheaper tiles will be removed while the amount exceeds the rent cost.

After the tile is purchased, or the rent is paid, if the agent has enough coins, it can upgrade its properties by paying the current cost, this duplicates both the cost and the rent. The agents will upgrade from their cheaper to the more expensive properties. Agents do not die and no more agents are added during the simulation. The goal of each agent is to buy as many tiles as it can.

This experiment does have a natural end when there are no more available tiles (for buying(. This experiment should be run a thousand times with varying values for the tiles.

What is the goal of the simulation? The goal of the simulation is twofold, the first is to confirm the Pareto principle at the end of the simulation. Data collected contains the final ratio of both coins and land owned by each agent.

Secondly, we want to validate fidelity to the specification of the five implementations through Macroscopic statistical analysis. For each framework, three macroscopic observable values will be collected for 10 000 executions. They are the amount to rounds to complete the experiments, the percentage of coins of the top 20% richer agents, and the percentage of tiles owned by the top 20% land owner agents.

These values will be plotted in three histograms. With one series for each framework, we aim to validate they were implemented identically and exercise the idea of statistical validation of (otherwise) indistinguishable models.

Which criteria of the FSMM can this experiment help evaluate?

Modeling the Pareto experiment help evaluate the following features from the simulation criterion

- Usability
- Basic framework functionality

From the interrogation capability criterion, the Pareto experiment aims to evaluate the following features:

- Visualization
- Statistical support for model attributes
- Support for macroscopic statistical analysis of compatible models

4.2.5 Correlating experiments and feature scoring

The experiments were designed in such a way for them to be useful for scoring the features in the FSMM. This relationship is depicted in the following table.

Experiment	Sugarscape	PredPrey	Pareto	Pandemics
Usability	✓	✓	✓	✓
Basic framework functionalities	✓	✓	✓	✓
Locality	✓	✓		
Agent management	✓	✓		✓
Force fields		✓		

Table 4.1: Simulation features exercised per experiment

Experiment	Sugarscape	PredPrey	Pareto	Pandemics
Visualization	✓	✓	✓	✓
Statistical support for model attributes	✓	✓	✓	✓
Macroscopic statistical analysis			✓	
Nearly decomposable System view		✓		

Table 4.2: Interrogation capabilities exercised per experiment

The relationships between experiments and criteria of the FSMM are illustrated in the following Venn diagrams.

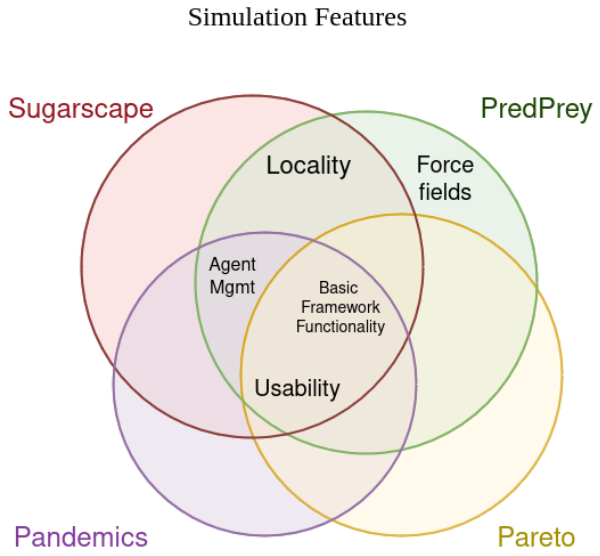

Figure 4.1: Simulation Features exercised by experiment

Similarly for each interrogation capability the Venn diagram is

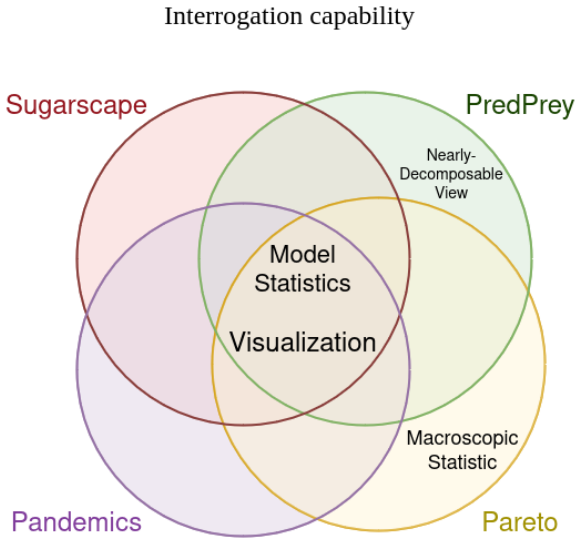

Figure 4.2: Interrogation Capabilities Features exercised by experiment

CHAPTER 5

Results & Discussion

"Done is better than perfect. "

Sheryl Sandberg, Lean In: Women, Work, and the Will to Lead

5.1 Sugarscape Model

This section presents the results of the Sugarscape experiment implementation. An analysis of each criterion per experiment, per framework, is provided.

As mentioned in the experiment descriptionm only three graph per simulation were included in this document. Other graphs where used during the development and debugging of the simulation.

5.1.1 NetLogo

Sugarscape as defined in the experiment is implemented in NetLogo in this URL `https://gitlab.com/msc_tesis/NetLogo/-/blob/main/Sugarscape/`.

The implementation required a single experiment NetLogo file which contains both the User Interface (UI) and behavior specs. This was straightforward due **Basic framework functionality** already provided by the framework, with only two types of agents (breeds in NetLogo jargon), males and females, and heavy use of provided **Locality** and **Agent management** functionality. The **Visualization** of the agents was also available since the beginning; it can be disabled to speed up simulation.

Debugging was done using several tools including graphs. As defined in the experiments, only **Model statistics** are shown in the final report; these include average agent energy, live count and average patch energy .

The most challenging aspect was the semantics difference from the usual OO approach. In general, the **Usability** of the tool was really good.

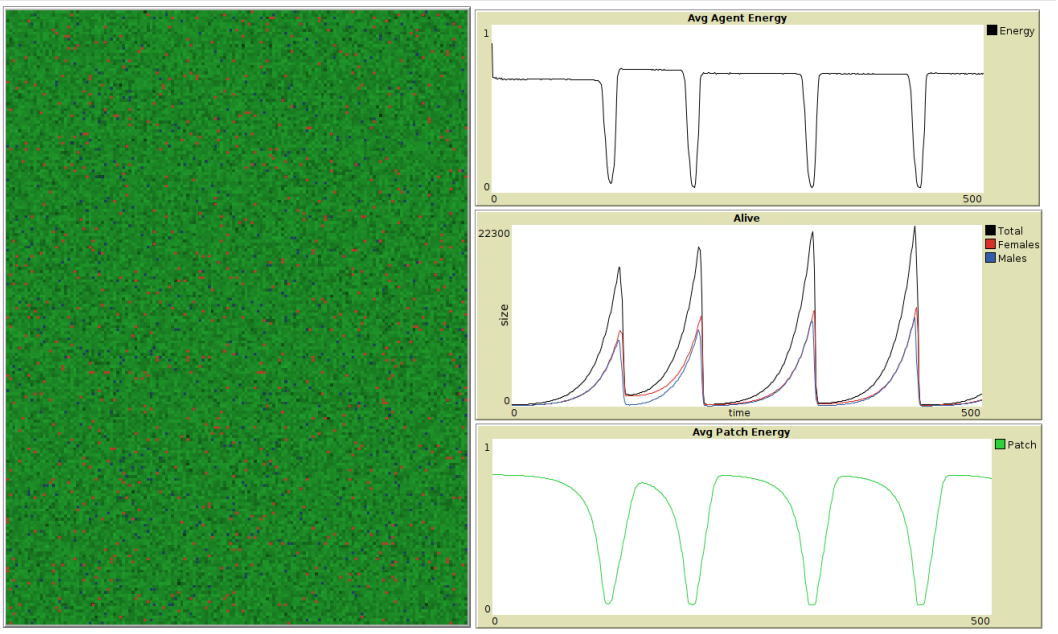

Figure 5.1: Sugarscape on NetLogo

5.1.2 MASON

Sugarscape as defined in the experiment is implemented in MASON in this URL `https://gitlab.com/msc_tesis/MASON/-/tree/main/Sugarscape`

The experiment was implemented in nine standard Java classes using the MASON jar file libraries. Four classes were utilities, implementing missing **Agent management** functionalities, three for agents behaviors, one for the environment and a final one encapsulating 'boiler plate' code for the UI.

The implementation required several utility classes. The minimalist approach of the framework lacks of tools for **Locality**. Tough not rigorously measured, the execution was probably the fastest one, which helped with the **Usability** and takes advantage of the ample support for the Java environment to implement **Basic framework functionality**. As the MASON framework is just a Java library, the experiment was developed in the Netbeans IDE[4].

Visualization capabilities were acceptable out of the box. **Model statistics** including average agent energy, live count and average patch energy – as shown below – were straightforward to implement.

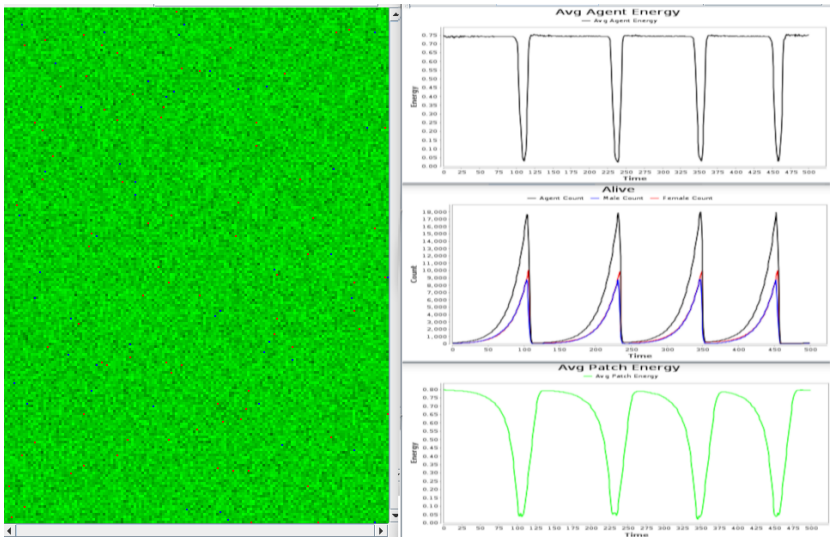

Figure 5.2: Sugarscape over MASON

5.1.3 Repast

Sugarscape as defined in the experiment is implemented in Repast in this URL `https://gitlab.com/msc_tesis/Repast/-/tree/main/Sugarscape`.

Sugarscape was implemented using a customized Eclipse version which works as the IDE for **Basic framework functionality**. Eight Groovy files were created, one utility, three for the agents, one for the environment and the final three were 'boiler plate' code for configuration and UI specification. Repast generates boiler plate code in Java to handle the simulation. It creates the code during file creation and code change. The main drawback of Repast is **Usability**, because the generated code may get stale quickly and affect its underlying Eclipse Platform. A full project clean was required every hour or so to avoid IDE crashes, also **Visualization** speed was noticeably slow.

Given that Repast follows NetLogo's approach, most of the logic was taken from the NetLogo version, updating the code to follow Groovy's syntax. As in NetLogo, **Locality** and **Agent management** was handled by the framework.

Model statistics were slightly harder to handle than in NetLogo for Repast requires a data producer and a graph description; the UI to define them failed from time to time.

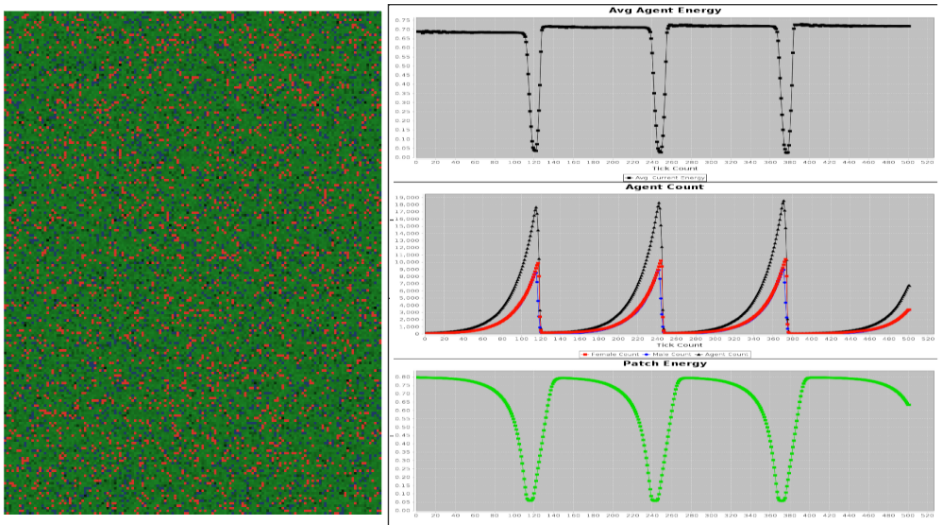

Figure 5.3: Sugarscape over Repast

5.1.4 Mesa

Sugarscape as defined in the experiment is implemented in Mesa in this URL `https://gitlab.com/msc_tesis/Mesa/-/tree/main/Sugarscape`.

As Mesa is just a collection ofPpython classes, the experiment was implemented in the Community version of IntelliJ[47]. It was implemented using five Python files. They were one utility file to implement **Locality** functionality, one file for the agent behavior, one for the patch behavior, one for the simulation and **Agent management** and one to specify the UI.

Even while distributed in five files, this was the implementation which required the smallest amount of code (less than 15 kb) and it benefits from the huge support for the Python environment to implement **Basic framework functionality**.

The main **Usability** drawback was that the convoluted connection between the UI and the back-end makes the simulation**Visualization** very slow when the UI is enabled. **Model statistics** including average agent energy, live count and average patch energy – as shown – were easy to implement.

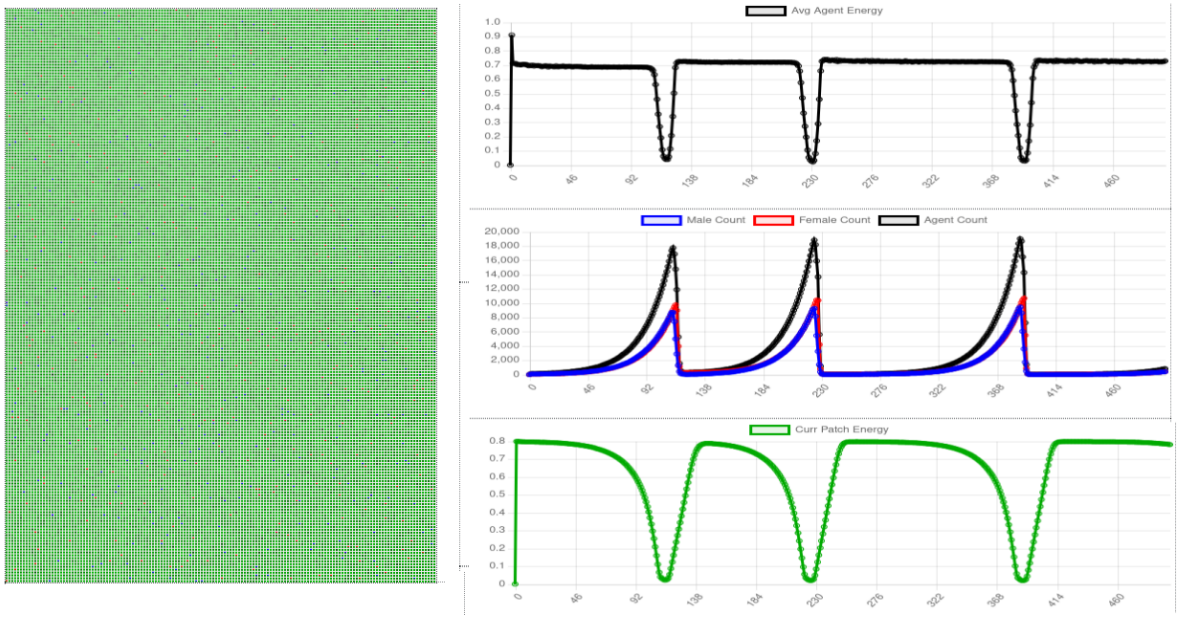

Figure 5.4: Sugarscape over Mesa

5.1.5 Agents.jl

Sugarscape as defined in the experiment is implemented in Agents.jl in this URL `https://gitlab.com/msc_tesis/Agents.jl/-/tree/main/Sugarscape`.

It was implemented using five Julia files. They were two utility files, including **Locality** logic, one file for the agent behavior, one for the simulation and one to specify the UI. As a Julia package, it benefits from the support for the Julia environment to implement **Basic framework functionality**, including the Microsoft Visual Studio Code [64] using the Julia extension [49] which supports debugging.

There were two major **Usability** drawbacks. The first was that the startup time was very slow (about 2 minutes). This was somehow mitigated using a Operating System (OS) dependent cache (which takes about 5 minutes to create) and using an interactive command-line (called Read-Eval-Print Loop (REPL)), with some limitations (a change in the structure specs will require a new session). The other drawback is the lack of ability to specify multiple axis for the **Model statistics** graph using the default Application Programming Interface (API).

Visualization capabilities were limited and there was a bug in **Agent management**, which breaks the framework when the agent size is a function (instead of static value) and agents are removed during the experiment.

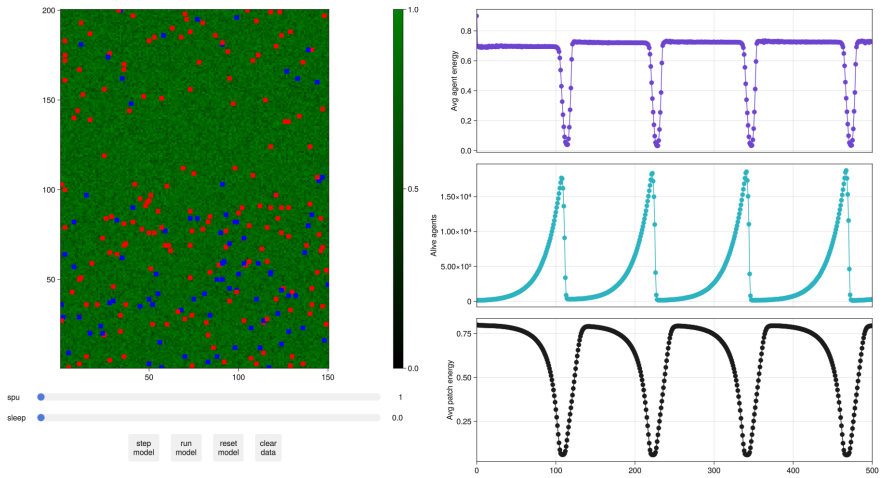

Figure 5.5: Sugarscape over Agents.jl

5.2 PredPrey Model

This section presents the results of the PredPrey experiment implementation, along with an analysis of each criterion per experiment for each framework.

While in most of the cases, there were wolves and rabbits at the end of the 1000 rounds, there were four possible outcomes for this experiment

- Population collapse where wolves eat all rabbits and die of starvation
- All Wolves die of starvation but some rabbits survive and reproduce continuously
- Wall stability where wolves and rabbits form two columns, the wolves one chasing the rabbit ones. This formation allowed continuous growth of both types of agents.
- Fireworks, where packets of rabbits grew and were eventually attacked by wolves

For the two latter outcomes, which were the most common ones, there were sections of highly interacting agents which should have been detected, but no framework provided **Nearly decomposable system view** capabilities.

5.2.1 NetLogo

PredPrey, as defined in the experiment, is implemented in NetLogo in this URL `https://gitlab.com/msc_tesis/NetLogo/-/blob/main/PredPrey/`.

The implementation required a single experiment file which is a NetLogo specs and contains both the UI and behavior specs, specially **Force fields** logic as it is not provided by the framework.

The implementation was straightforward with only two types of agents (called breeds in NetLogo jargon), rabbits and wolves using **Locality** and **Agent management** functionality already provided. Only two **Model statistic** graphs were added, the alive count and the percentage of rabbits vs wolves.

Visibility capabilities were used, specially the direction of agents using the triangle shape. The most challenging **Usability** aspect was that the semantics is different from the usual OO approach. A major drawback is the limited amount of debugging tools, which may slow down bigger simulation endeavours, yet for this experiment the provided **Basic framework functionality** was good enough.

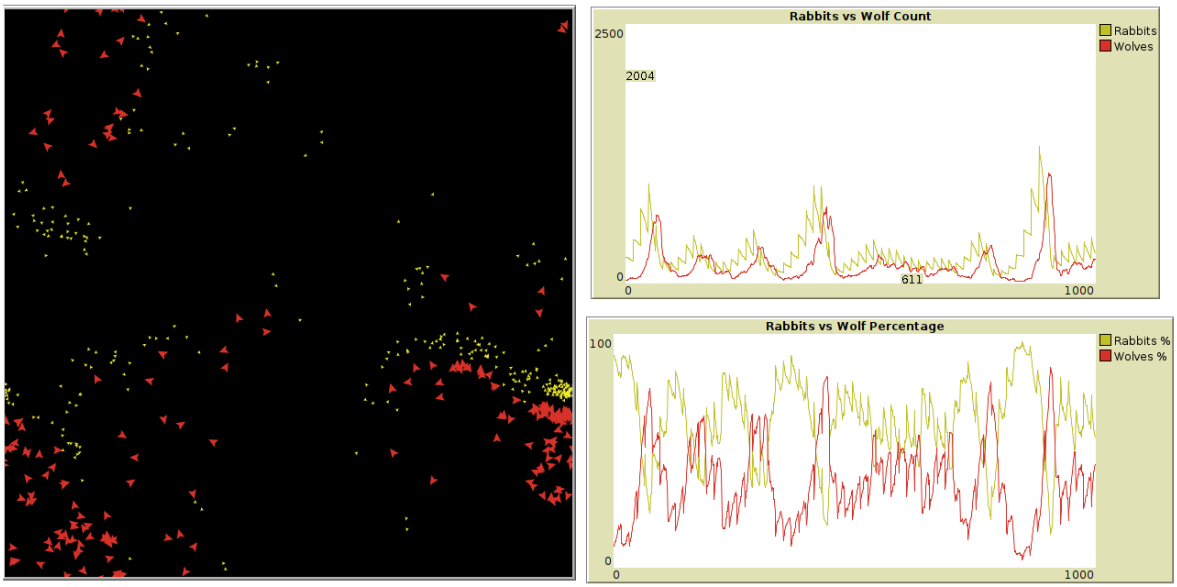

Figure 5.6: PredPrey over NetLogo

5.2.2 MASON

PredPrey, as defined in the experiment, is implemented in MASON in this URL `https://gitlab.com/msc_tesis/MASON/-/tree/main/PredPrey`.

The experiment was implemented in nine standard Java classes using the MASON jar file libraries. Four classes were utilities, including vector functions for **Force fields**, three for agents behaviors (the base agent class, one for the rabbits and another one for wolves), one for the environment and a the final one was 'boiler plate' code for the UI. Stability of the framework over Netbeans was good for the whole experiment, which confirms MASON **Usability**.

The implementation required several utility class events to complete **Locality** functionality, due to the minimalist approach of the framework, yet it has the advantage of the vast support for the Java environment to implement **Basic framework functionality**. One interesting **Agent management** limitation is that the removeSteppable method does not remove the agent until the full round is complete, therefore, a boolean rabbit variable isDead was required to avoid using a death prey.

Only two **Model statistic** graphs were added, the alive count and the percentage of rabbits vs wolves. A limitation for **Visibility** was the missing capability to draw a shape, like a triangle, to show which direction the agent was facing.

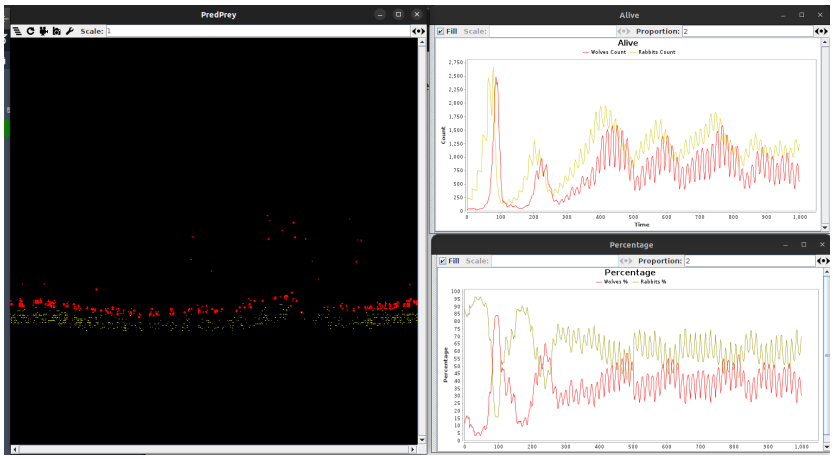

Figure 5.7: PredPrey over MASON

5.2.3 Repast

PredPrey, as defined in the experiment, is implemented in Repast in this URL `https://gitlab.com/msc_tesis/Repast/-/blob/main/PredPrey/`.

The implementation required eight Groovy files and two Java files, including a vector class to handle **Force fields**.

Because of Repast origins, most of the **Basic framework functionality** was implemented based on the NetLogo version, updating the code to follow Groovy's syntax. Also, as in NetLogo, **Locality** and **Agent management** was handled by the framework.

There were several **Usability** issues, for example, Repast creates helper classes during coding which needs to be re-synchronized using the clean/rebuild option from time to time. When using Java 11, several random incompatibilities started to appear at execution time; the only solution was to move the framework to Java 17.

Visibility performance was very poor compared to the NetLogo/MASON version. As with previously mentioned frameworks, only two **Model statistic** graphs were added, the alive count and the percentage of rabbits vs wolves.

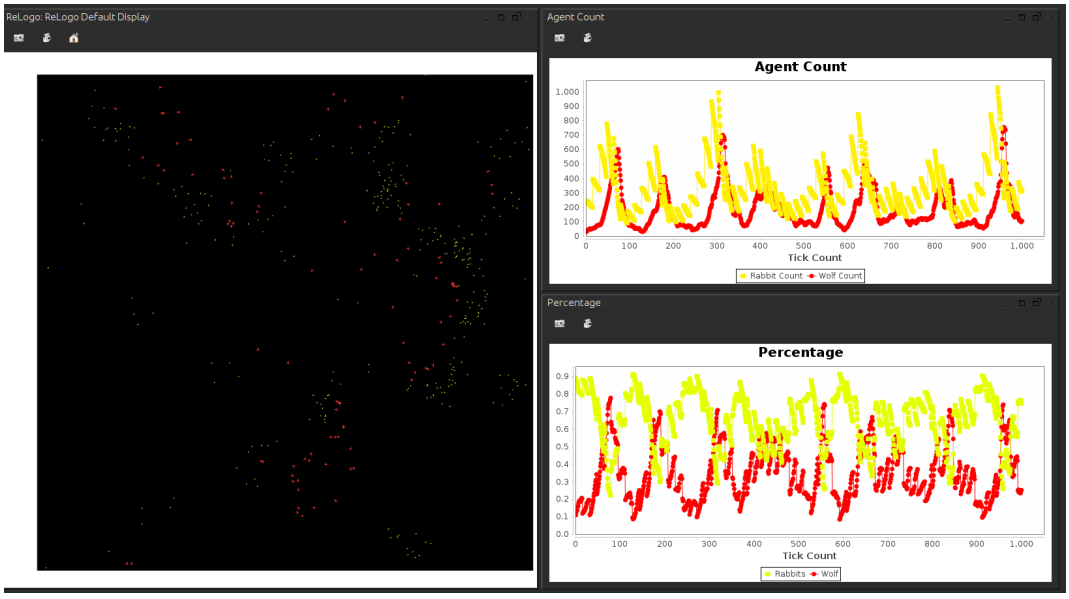

Figure 5.8: PredyPrey over Repast

5.2.4 Mesa

PredPrey, as defined in the experiment, is implemented in Mesa in this URL `https://gitlab.com/msc_tesis/Mesa/-/blob/main/PredPrey/`.

The implementation required 5 files, including a vector utility to simulate the **Force fields**, one file for **Locality** functionality, one for both agents behavior, one for the simulation and **Agent management**, and finally one for UI specs. As a Python library, it benefits from the support for the Python environment to implement **Basic framework functionality**

The implementation was straightforward, with only two types of agents: rabbits and wolves. Only two **Model statistic** graphs were added, the alive count and the percentage of rabbits vs wolves.

The most challenging **Usability** aspect was the incredibly slow speed of the UI in the browser which makes testing really difficult. Also **Visibility** functionality was limited and getting an acceptable screen rendering required manual adjustment, including zooming out on a scaled canvas – worked only on Google Chrome.

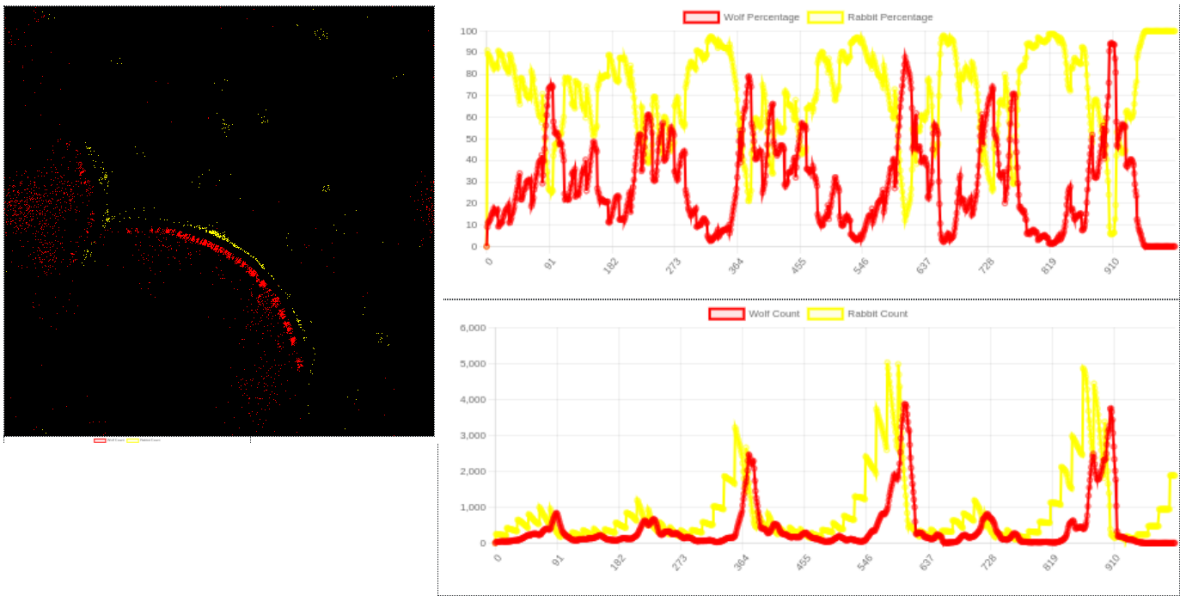

Figure 5.9: PredPrey over Mesa

5.2.5 Agents.jl

PredPrey, as defined in the experiment, is implemented in Agents.jl in this URL `https://gitlab.com/msc_tesis/Agents.jl/-/tree/main/PredPrey`

The experiment was implemented in four standard Julia files using the Agents.jl libraries, including a class to handle **Locality** capabilities. It used the complex number data type to simulate the **Force fields**. As a Julia package, it benefits from the support for the Julia environment to implement **Basic framework functionality**.

There were several **Usability** issues during the implementation, including the lack of backward compatibility of system images from Julia 1.9 and 1.10 and some bugs in **Agent management** when the size is a function and agents are removed. Also **Visibility** performance was very poor compared to NetLogo/MASON.

It was impossible to visualize **Model statistic** on two axes with the provided API.

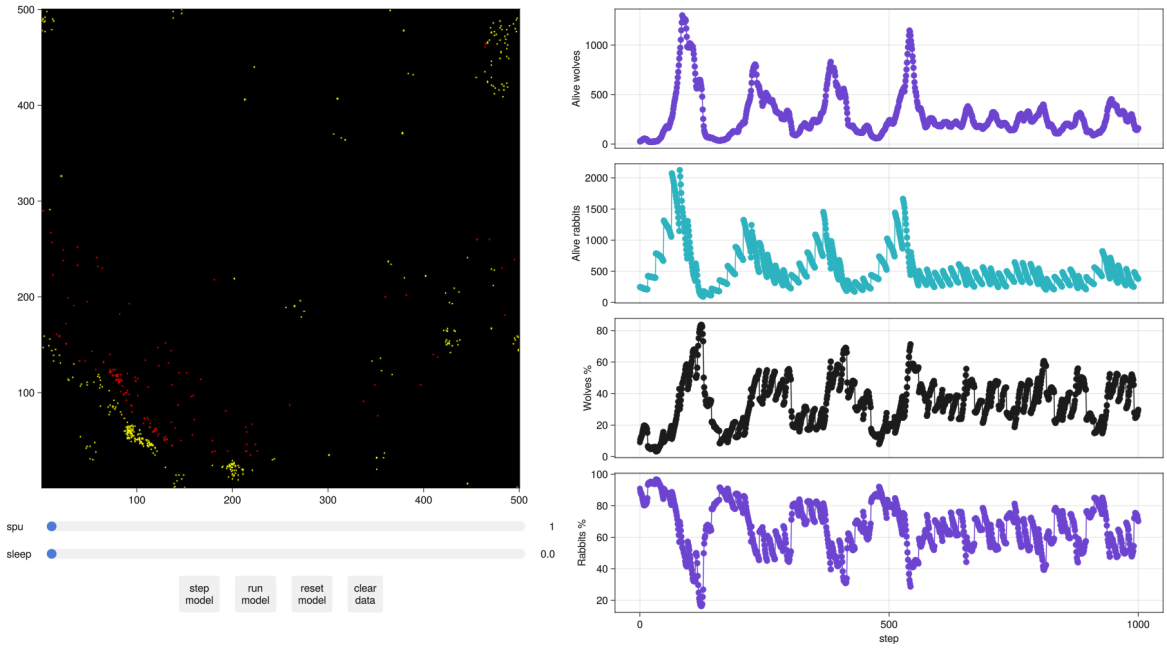

Figure 5.10: PredPrey over Agents.jl

5.3 Pandemics Model

In this section the results of the Pandemics experiment implementation, along with analysis of each criterion per experiment, for each framework, is provided.

5.3.1 NetLogo

The COVID-19 pandemic, as defined in the experiment, is implemented in NetLogo in this URL `https://gitlab.com/msc_tesis/NetLogo/-/blob/main/Pandemics/`.

The implementation required a single experiment file which is a NetLogo specs and contains both the UI and behavior specs. This was straightforward due **Basic framework functionality** being already provided by the frameworkd, using the standard NetLogo turtles. The **Visualization** of the agents was also available since the beginning, with the ability to disable it to speed up simulation.

The most challenging aspect was that the semantics is different from the conventional OO approach. In general, the **Usability** of the tool was really good. One limitation was that the http requests were poorly documented and the post call did not support more than 1500 sets of values, even using the multipart structure which should support several megabytes of data.

We used the provided **Agent management** capabilities to locate one user per cell in the grid but had to implement the required mailbox approach.

The following figure shows the two rounds of the simulation on NetLogo with 2% and 3% of agent trust provided by influencers.

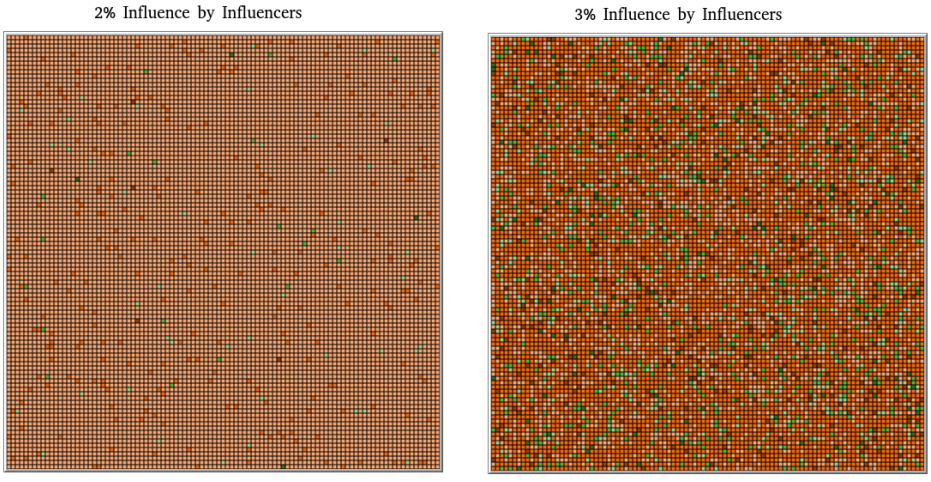

Figure 5.11: Pandemics over NetLogo

We are interested in reviewing **Model statistics**, including the sick ratio and trust level differences among the two rounds. In the first round, influencers provide a 2% agent trust, whereas in the second round, they only offer an additional percent point of trust. Interestingly, despite this expected small difference, in the second round agents do not discover that the amulet increases the risk of getting sick.

The following result charts depict data generated from a plot utilizing HTTP-exported data from NetLogo.

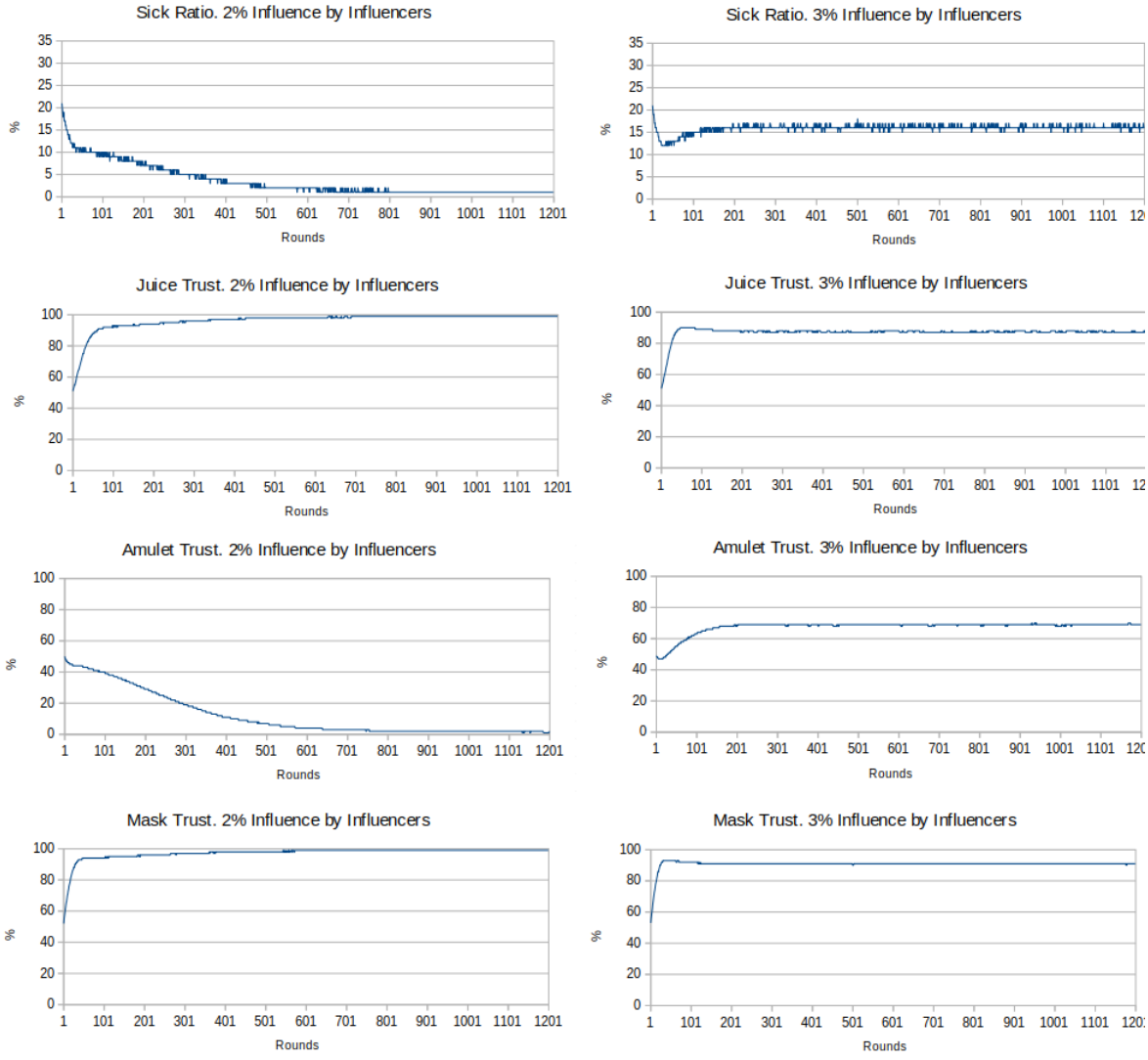

Figure 5.12: Pandemics over NetLogo using reported data

5.3.2 MASON

The COVID-19 pandemic, as defined in the experiment, is implemented in MASON in this URL `https://gitlab.com/msc_tesis/MASON/-/blob/main/Pandemics/`.

The implementation required seven experiment files, three for logic, one for the UI and for handling multipart post. This was straightforward due to **Basic framework functionality** being already provided by the Java environment, using known libraries like those for JSON handling. The **Visualization** of the agents was also available since the beginning, being completely fluid even for 10000 agents.

There wasn't any particular technical challenge and in general the **Usability** of the tool was really good. One limitation was the http multipart post functionality was lacking from the standard Java libary, so it was built for the Java client.

We used the provided **Agent management** capabilities to locate one user per cell in the grid but had to implement the required mailbox approach.

The first chart shows the two rounds of the simulation over MASON with 2% and 3% of agent trust provided by influencers.

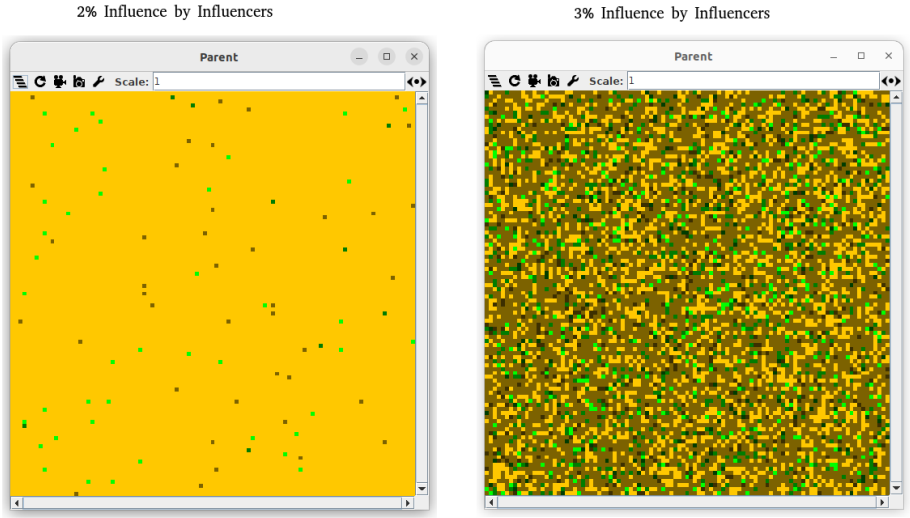

Figure 5.13: Pandemics over MASON

Similar to NetLogo, we are interested in reviewing **Model statistics**, including sick ratio and trust level differences among the two rounds. In the first round, influencers provide a 2% agent trust, whereas in the second round, they only offer an additional percent point of trust. Interestingly, despite this expected small difference, in the second round the agents do not discover that the amulet increases the risk of getting sick.

The following result charts depict data generated from a plot utilizing HTTP-exported data from MASON.

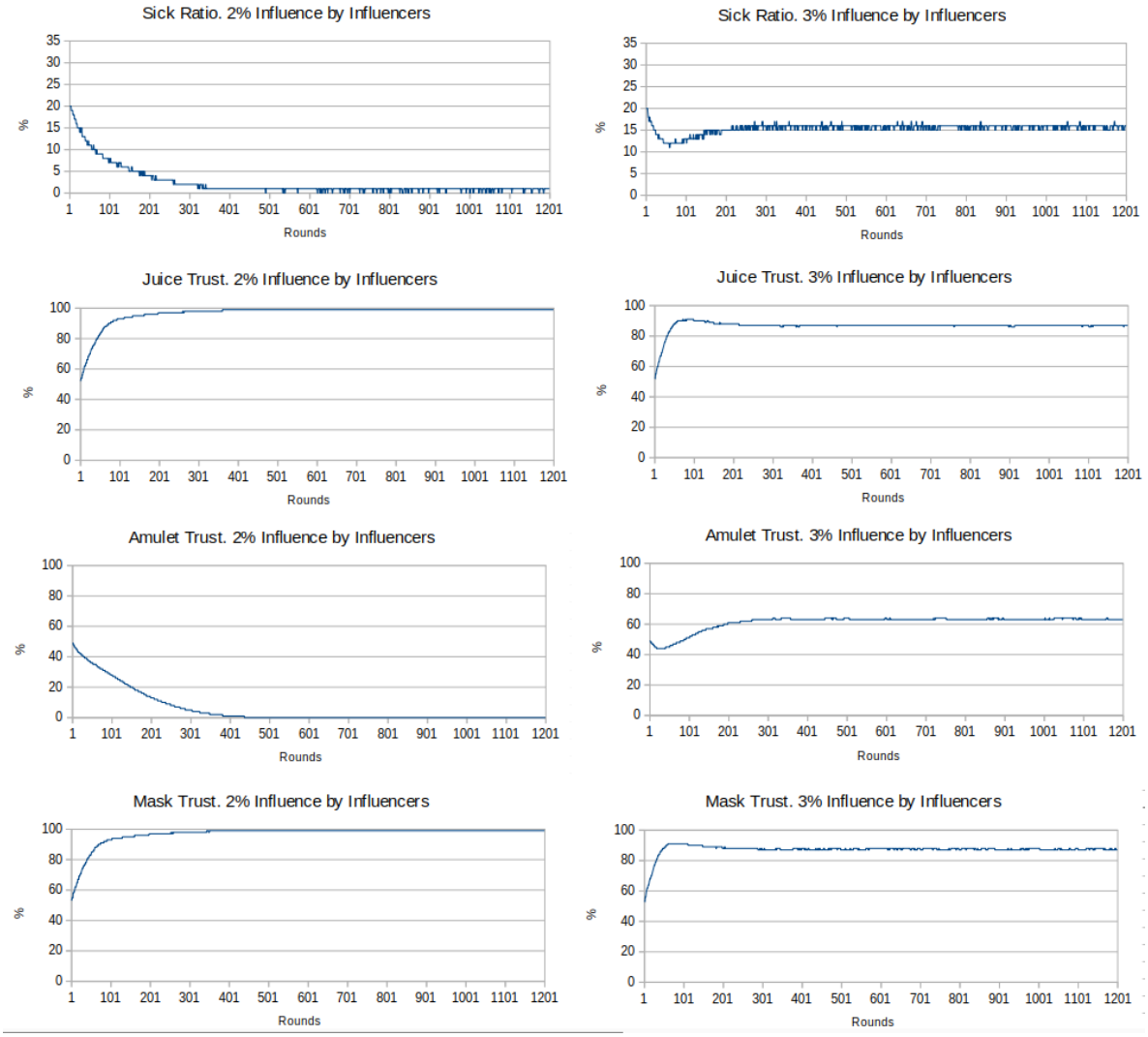

Figure 5.14: Pandemics over MASON using reported data

5.3.3 Repast

The COVID-19 pandemic, as defined in the experiment, is implemented in Repast in this URL `https://gitlab.com/msc_tesis/Repast/-/blob/main/Pandemics/`.

The implementation required five experiment files, plus hundreds of automatically created maintenance files. The main files we the agent Groovy file, the simulation Groovy file, and the UI specs files. There were two helper java files, a required utility file and one for handling multipart post. It was straightforward due to **Basic framework functionality** being already provided by the Java environment, using known libraries like those for JSON handling. The **Visualization** of the agents was also available since the beginning, being fluid even for 10000 agents.

The main challenge for the **Usability** of the tool is the fact that there is a need to keep cleaning the environment to avoid stale autogenerated Java classes and that reporting specs is not performed through code but in the UI which can get convoluted. Another limitation was that the http multipart post functionality was lacking from the standard Java library, so it was built for the Java client.

We used the provided **Agent management** capabilities to locate one user per cell in the grid but had to implement the required mailbox approach.

The first chart shows the two rounds of the simulation over Repast with 2% and 3% of agent trust provided by influencers.

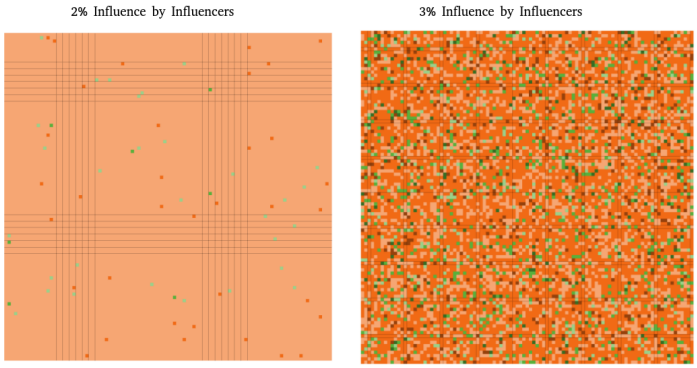

Figure 5.15: Pandemics over Repast

Similar to NetLogo and MASON, we are interested in reviewing we are interested in reviewing two **Model statistics**: the sick ratio and trust level differences among the two rounds. In the first round, influencers provide a 2% agent trust, whereas in the second round, they only offer an additional percent point of trust. Interestingly, despite this expected small difference, in the second round agents do not discover that the amulet increases the risk of getting sick.

The following result charts depict data generated from a plot utilizing HTTP-exported data from Repast.

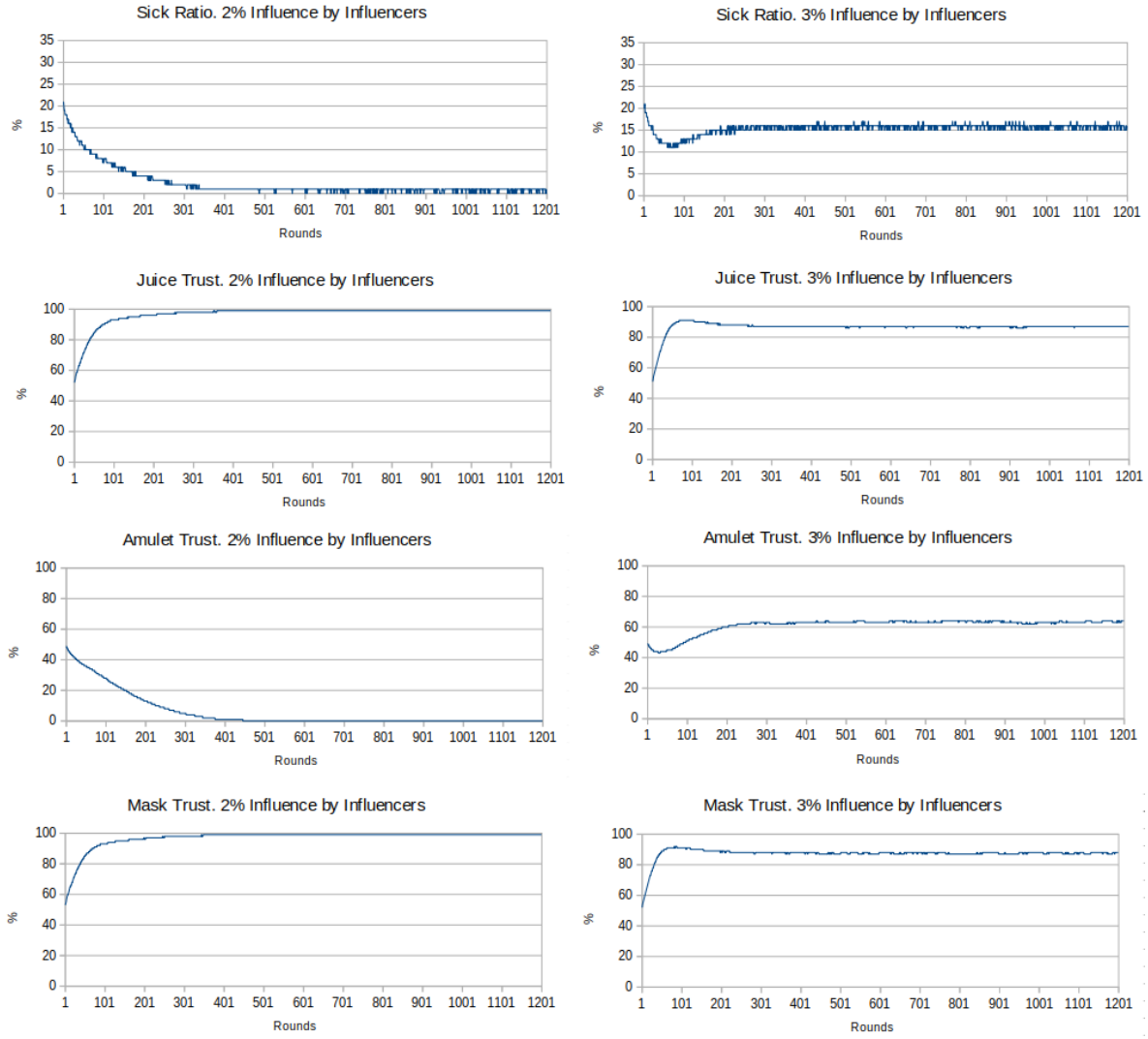

Figure 5.16: Pandemics over Repast using reported data

5.3.4 Mesa

The COVID-19 pandemic, as defined in the experiment, is implemented in Mesa in this URL `https://gitlab.com/msc_tesis/Mesa/-/blob/main/Pandemics/`.

The implementation required seven experiment files, three for logic, one for the UI. It was straightforward due to **Basic framework functionality** being already provided by the Python environment, using known libraries like those for JSON handling. There were no communication capabilities between agents which is a part of the **Agent Management** topic. The **Visualization** of the agents was also available since the beginning, being fluid even for 10000 agents.

There wasn't any technical challenge. In general, the **Usability** of the framework was really good. The speed of display is slow, therefore only the final step was displayed.

The first chart shows the two rounds of the simulation on Mesa with 2% and 3% of agent trust provided by influencers.

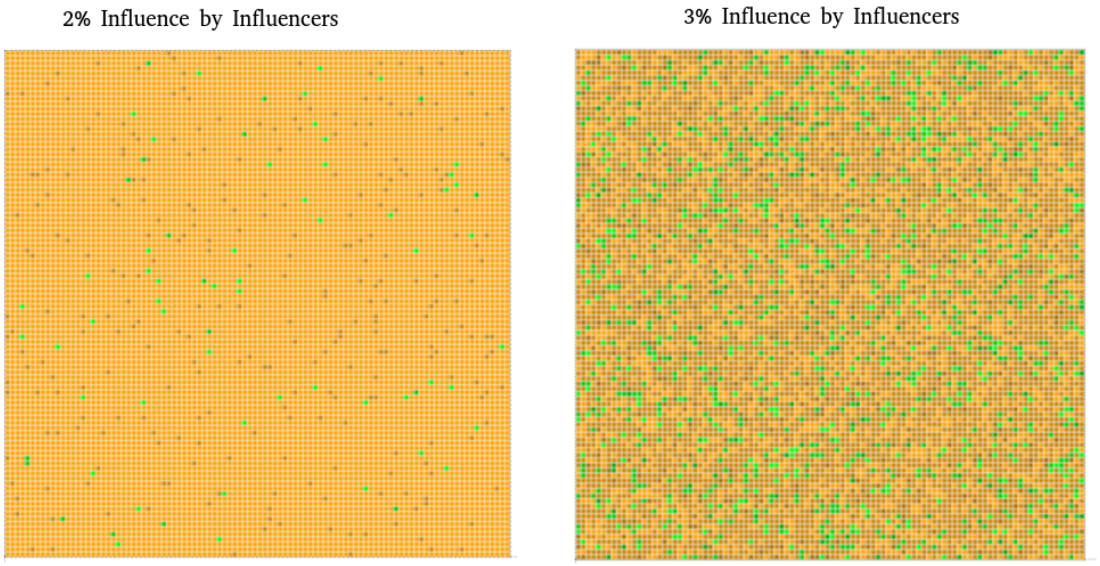

Figure 5.17: Pandemics over Mesa

Similar to the other frameworks, we are interested in reviewing the sick ratio and trust level differences among the two rounds. In the first round, influencers provide a 2% agent trust, whereas in the second round, they only differ in one additional percent point of trust. Interestingly, despite this small difference, in the second round agents do not discover that the amulet increases the risk of getting sick.

The following result charts depict data generated from a plot utilizing HTTP-exported data from Mesa.

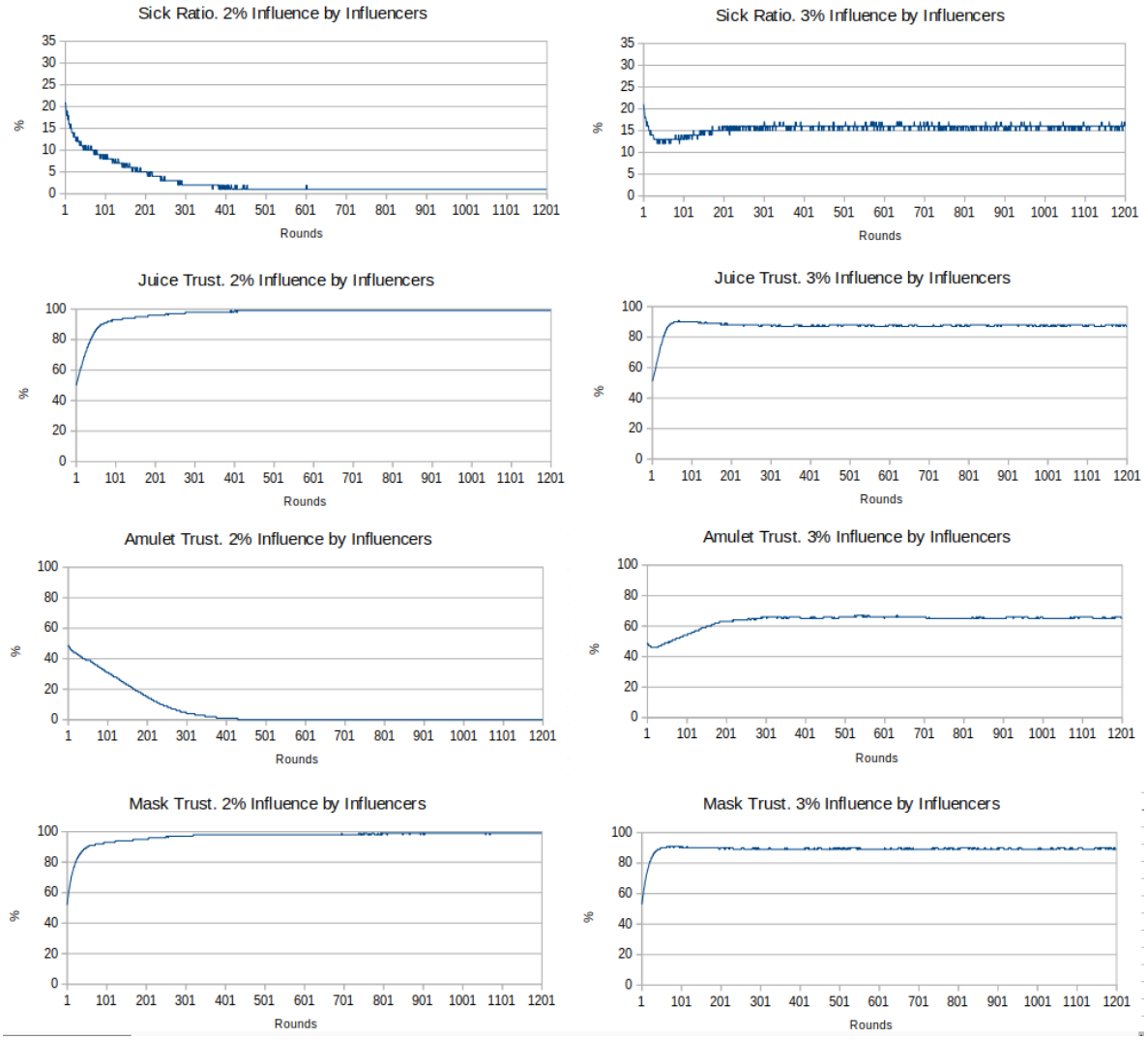

Figure 5.18: Pandemics over Mesa using reported data

5.3.5 Agents.jl

The COVID-19 pandemic, as defined in the experiment, is implemented in Agents.jl in this URL `https://gitlab.com/msc_tesis/Agents.jl/-/blob/main/Pandemics/`.

The implementation required five experiment files, three for logic, one for the UI and one to handle command line simulation. The implementation had to be written two times as the Makie UI support couldn't handle all users. So for the UI shown in the following section, we used a smaller 85x85 grid (instead of the 100x100 expected grid). The reports were done using a command-line only version with the expected grid size. **Basic Framework Functionality** was provided by the Julia environment, including using known libraries like JSON & HTTP. The **Visualization** was really slow even when using a version that was only 64% the size of the other frameworks

Provided **Agent Management** capabilities like placing agents in the coordinate were not used due to memory issues in Julia. The main **Usability** challenge was the unexpected Julia memory issue which was not present in previous experiments.

The first chart shows the two rounds of the simulation over Julia, with a smaller grid (85x85) than with the other implementations (100x100) due the memory issues with Julia. The results of the 1200 steps with 2% and 3% of agent trust were provided by GUI-less execution of the experiment with the full grid.

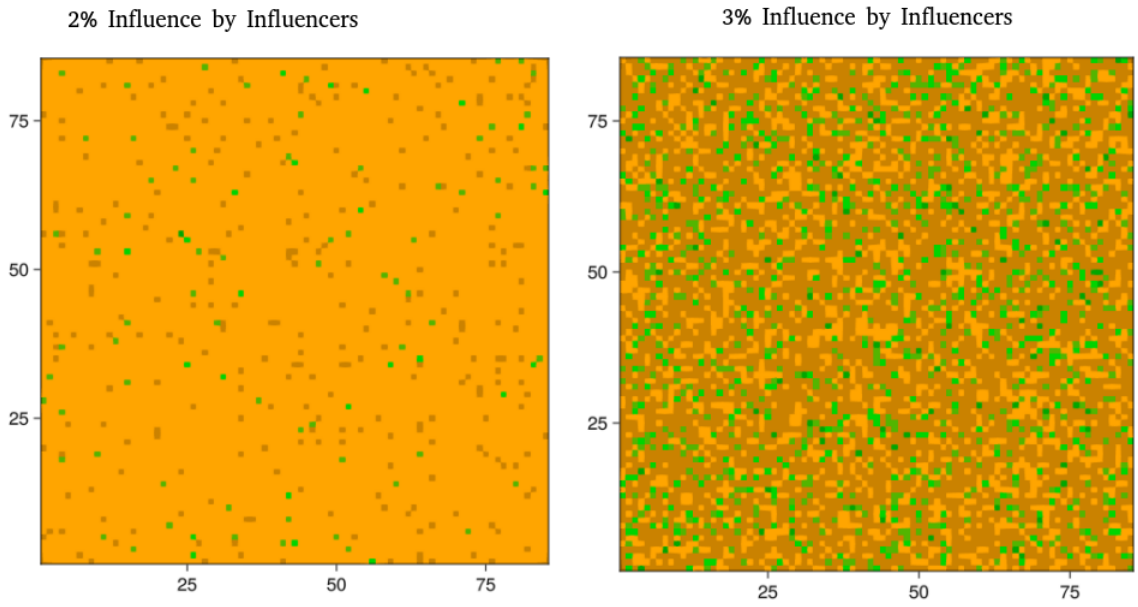

Figure 5.19: Pandemics over Agents.jl

We are interested in reviewing **Model statistics**, including the sick ratio and trust level differences among the two rounds. In the first round, influencers provide a 2% agent trust, whereas in the second round, they offer an additional percent point of trust. Interestingly, despite this small difference, in the second round agents do not discover that the amulet increases the risk of getting sick.

The following result charts depict data generated from a plot utilizing HTTP-exported data from Agents.jl.

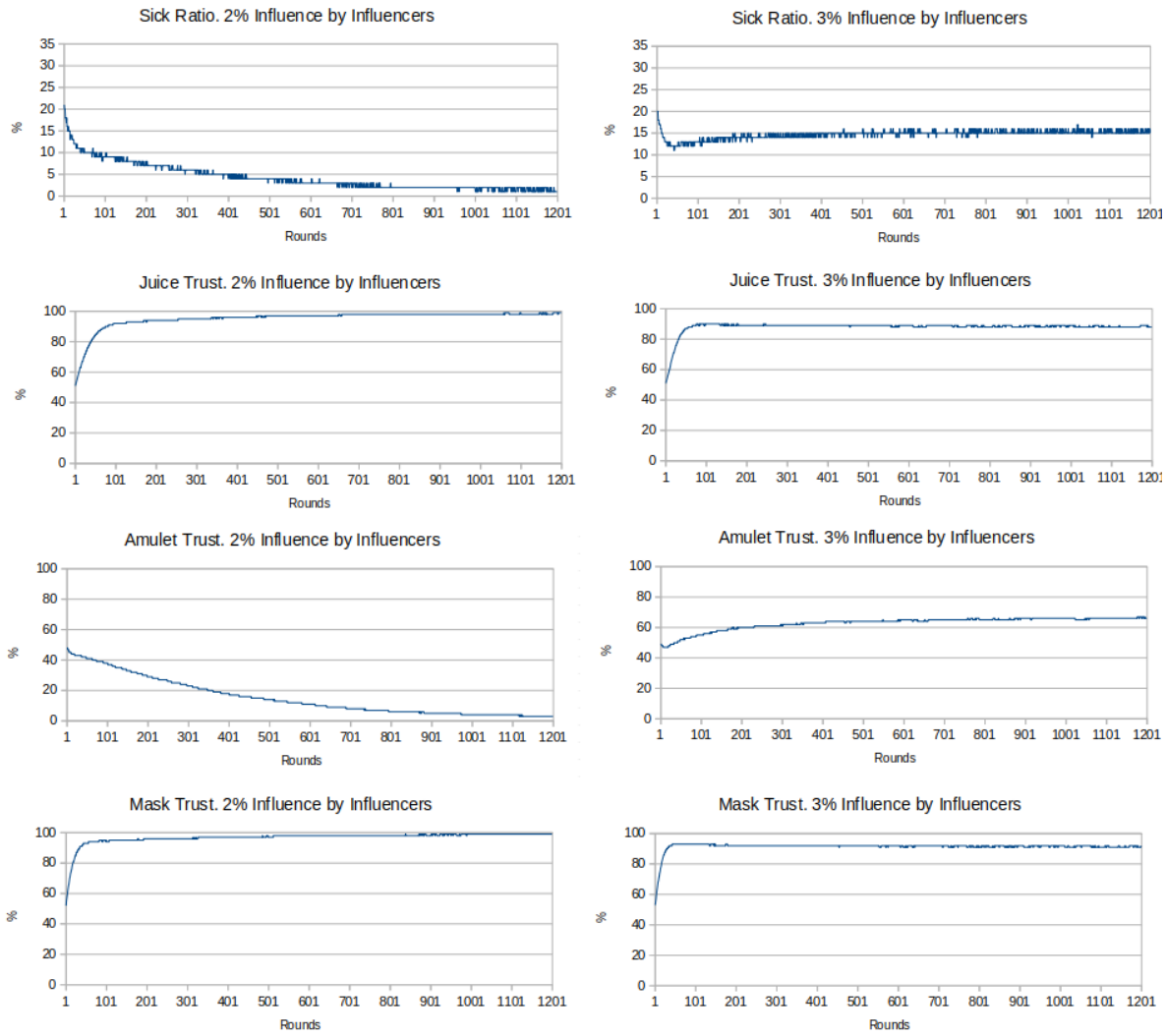

Figure 5.20: Pandemics over Agents.jl using reported data

5.4 Pareto Model

This section shows the results of the Pareto experiments, along with analysis of each criterion per experiment, for each framework.

For this experiment, due its quick run and natural stop, the simulations were run 10000 times to collect macroscopically comparable data and to validate that all implementations of the experiments are statistically identical.

At the end of the experiments, we perform an analysis of the macroscopic observables – for each framework.

5.4.1 NetLogo

Pareto, as defined in the experiment, is implemented in NetLogo in this URL `https://gitlab.com/msc_tesis/NetLogo/-/blob/main/Pareto/`.

The implementation required a single experiment file which contains both the UI and behavior specs. No external add-ons were required as NetLogo provided all required **Basic Framework Functionality**.

The implementation was straightforward, with only one type of agent. **Visualization** was very fast, and for the **Model statistic** only the cumulative curve in decreasing order (the Pareto curve) for the owned tiles and coins was added.

A Comma-separated values (CSV) file data for the **Macroscopic statistic analysis** was created using provided file saving functionality. Each line contains the three macroscopic observables defined in the specification. The simulation was run 10000 times. As NetLogo does not support multiple executions, in order to simulate the Pareto model all executions were done in a loop which reset the values. For this experiment the **Usability** was good enough. As with previous experiments, the main challenge in NetLogo was its convoluted syntax.

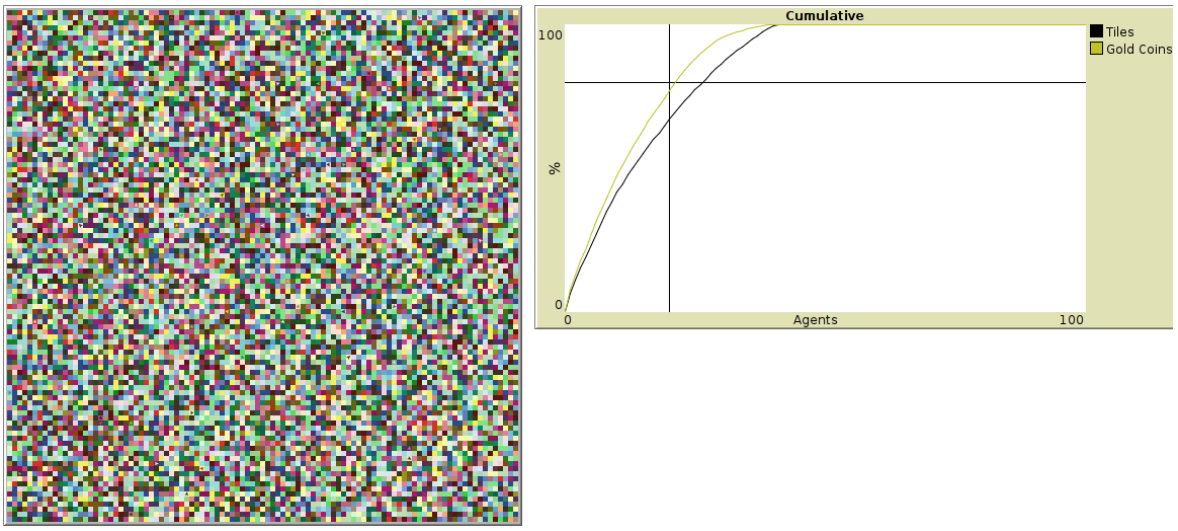

Figure 5.21: Pareto over NetLogo

5.4.2 MASON

Pareto, as defined in the experiment, is implemented in MASON in this URL `https://gitlab.com/msc_tesis/MASON/-/blob/main/Pareto/`.

The implementation required five Java files, which contains both the UI and behavior specs. No external jar files were required as Netbeans provided the **Basic framework functionality**.

The implementation was straightforward, with only one type of agent. **Visualization** was very fast, and for the **Model statistic** only the cumulative curve in decreasing order (the Pareto curve) for the owned tiles and coins was added. There was a minor limitation to draw the 20/80 lines, so two series were added to draw those lines.

A CSV file data for the **Macroscopic statistic analysis** using standard Java file system support. Each line containing the three macroscopic observables defined in the specification. The simulation was run 10000 times. Because of Java following the OO paradigm, it was very natural to create and execute the simulation objects independently. If desired, it is simple to modify each simulation execution independently. For this experiment the **Usability** was good, especially the speed of headless executions.

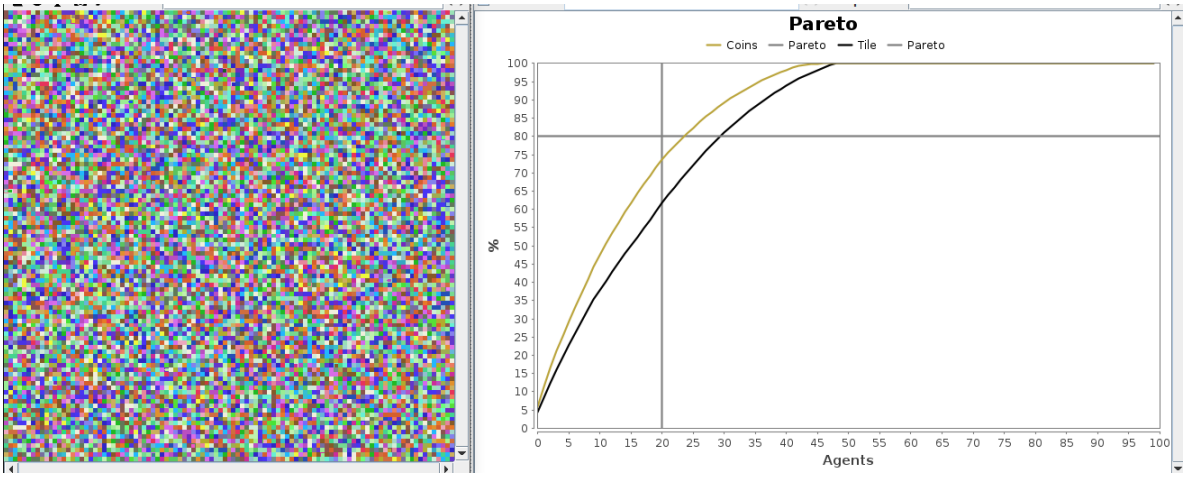

Figure 5.22: Pareto over MASON

5.4.3 Repast

Pareto, as defined in the experiment, is implemented in Repast in this URL `https://gitlab.com/msc_tesis/Repast/-/blob/main/Pareto/`.

The implementation required five Groovy files, plus 100+ files created by the Eclipse framework. **Basic framework functionality**, like debugging capabilities, were provided by Eclipse, whereas **Visualization** of not time-based graphs are not possible in the current version Repast. One **Model statistic** graph required to visually identify the Pareto behavior was the cumulative of coins and tiles. Yet, in order to create it, the data was stored in a CSV file and then post-processed as a graph in a LibreOffice Calc worksheet [34], a third party tool.

A CSV file data was produced for the **Macroscopic statistic analysis** using the standard Java file system support. Each line contains the three macroscopic observables defined in the specification. The simulation was run 10000 times. For this experiment the **Usability** was very poor, as it was required to clone a version of a previous experiment, given that the Eclipse plugin started experiencing issues creating new simulation projects from scratch. While Repast seems to provide the functionality to run multiple iterations at the same time, the documentation was lacking and the behavior seemed intricate. In order to perform the simulation, all executions were done in a loop which reset the values, and a checkbox to run 10000 times was added to the UI.

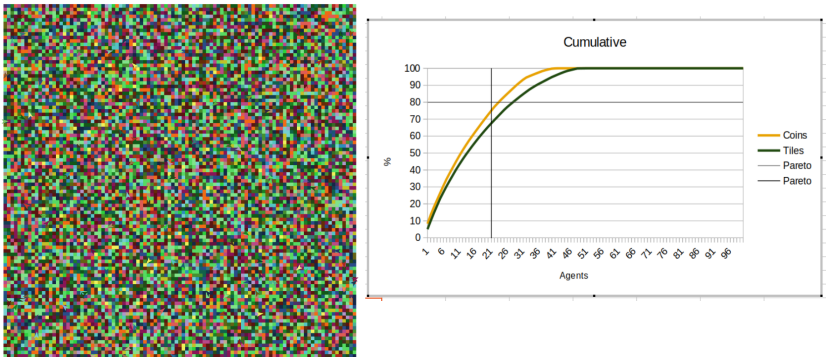

Figure 5.23: Pareto over Repast

5.4.4 Mesa

Pareto, as defined in the experiment, is implemented in Mesa in this URL `https://gitlab.com/msc_tesis/Mesa/-/blob/main/Pareto/`.

The implementation required four Python files, including the Pareto main file, one for the tile, a visualization helper and one for the agent. **Basic framework functionality**, like debugging capabilities, were provided by IntelliJ. **Visualization** of graphs which are not time-based are not possible in the current version Mesa. One **Model statistic** graph required to visually identify the Pareto behavior was the cumulative of coins and tiles; in order to create it, the data was stored in a CSV file and then post-processed as a graph in a LibreOffice Calc sheet [34], a third party tool.

A CSV file data was generated for the **Macroscopic statistic analysis** using standard Python file system support. Each line contains the three macroscopic observables defined in the specification. The simulation was run 10000 times. Given the class-based OO being present in the Python programming language, it was really natural to create and execute the simulation objects. If desired, it is uncomplicated to modify each simulation execution independently.

For this experiment the **Usability** was poor for the UI, as it was too slow, but acceptable during the headless 10000 iterations.

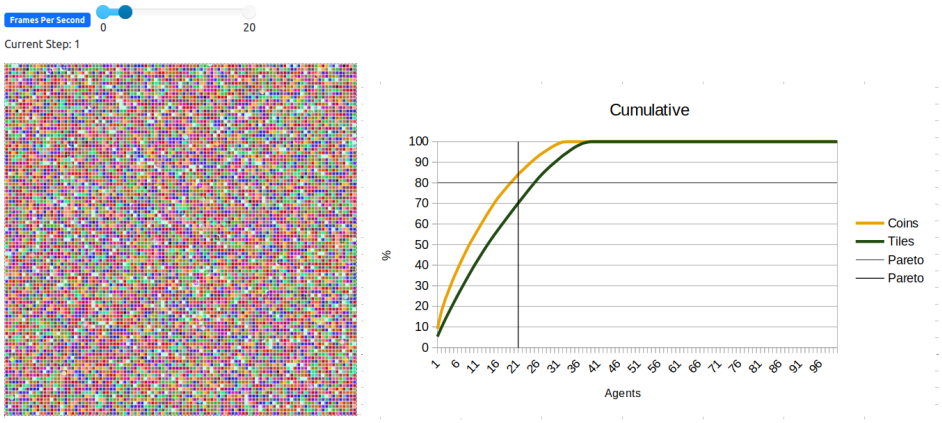

Figure 5.24: Pareto over Mesa

5.4.5 Agents.jl

Pareto, as defined in the experimen,t is implemented in Agents.jl in this URL `https://gitlab.com/msc_tesis/Agents.jl/-/blob/main/Pareto/`.

The implementation required several Julia files, which contains both the UI and behavior specs. No external Julia modules files were required as MS Code provided **Basic framework functionality** – such as debugging.

The implementation was straightforward, with only one type of agent. **Visualization** was very fast. As with Repast and Mesa, only time aggregate graphs are allowed in Agents.jl. The expected **Model statistic** was the cumulative curve in decreasing order (the Pareto curve) for the owned tiles and coins, and had to be saved in a file to be plotted afterward by a third party tool.

A CSV file data for the **Macroscopic statistic aAnalysis** was created, using Julia's standard file system support. Each line contains the three macroscopic observables defined in the specification. The simulation was run 10000 times. Because of its functional approach, it was natural to create and execute simulations in Julia. If desired, it is easy to modify each simulation execution independently. For this experiment the **Usability** was good, specially the speed of headless executions.

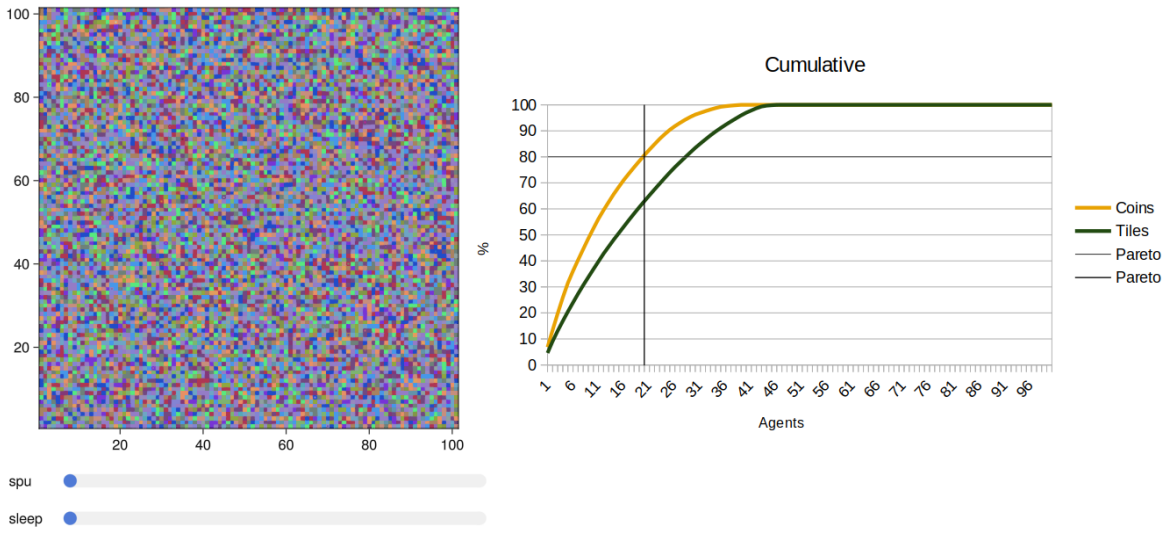

Figure 5.25: Pareto over Agents.jl

5.5 Macroscopic Statistical Analysis & Specification Fidelity

Regarding the Pareto simulation, for each execution, three macroscopic observables were included in a CSV file. These files where then processed by a script to create histograms, where each series represents a framework.

Using this approach, we were able to identify few deviations of some implementations with respect to the specification – which were not apparent *a priori* – like sorting the of the tiles based on price when settling an outstanding debt or improving a tile.

5.5.1 Histogram

After performing the necessary corrections, the histograms show that the macroscopic observables of all versions are statistically indistinguishable, which supports the argument that all implementations follow the specification. Some highlights for each histogram are mentioned below.

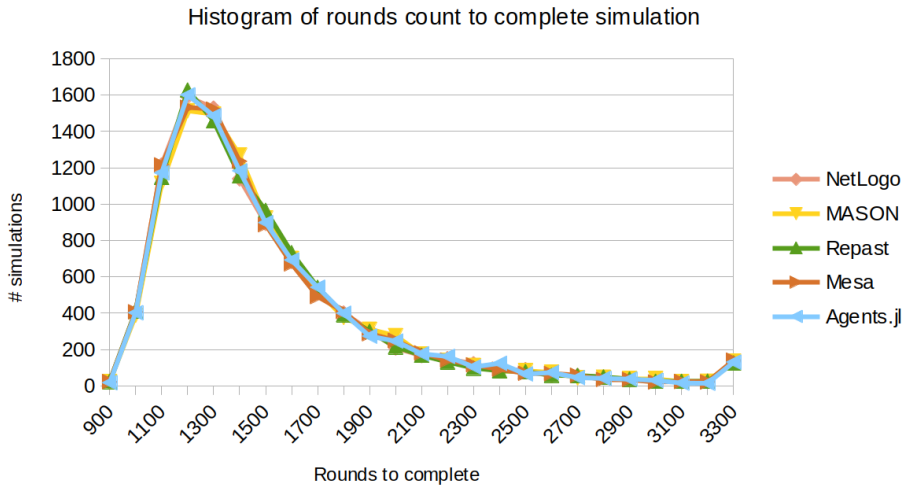

Figure 5.26: Histogram for rounds

Notice the tail after 2500 rounds. This is due the fact that each tile needs to be purchased to complete the simulation, but agents can only reach a tile by chance, which

could create iterations with more than 5000 steps.

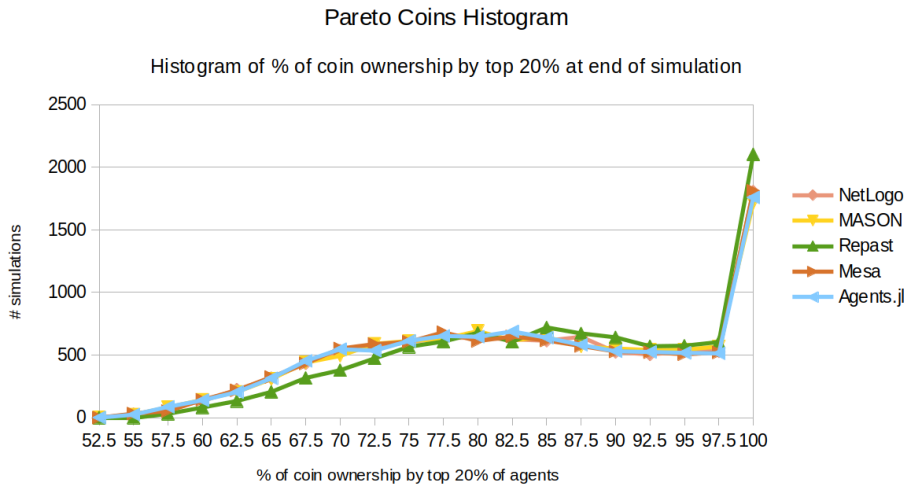

Figure 5.27: Histogram for coins ownership by top 20% of agents

Coins ownership are clearly around 80% but there is a hump of scenarios where the 20% or less of the agents owned the 100% of the coins. This could be caused by the rule of settlement, where even when an agent had an outstanding debt of one coin it needs to surrender a whole tile. Then the agents which hoard most of tiles will reap more of the rent benefits.

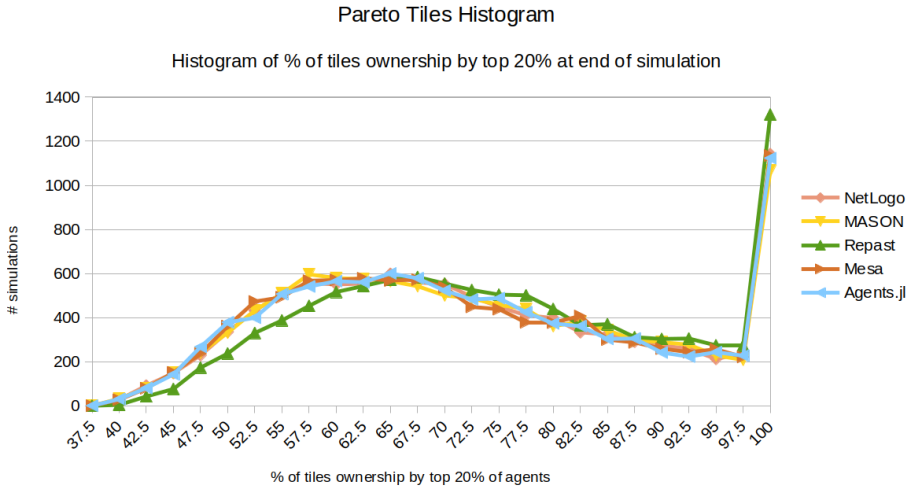

Figure 5.28: Histogram for tiles ownership by top 20% of agents

Tiles ownership are clearly around 70-80% but there is a hump in scenarios where

the 20% or less of the agents owned the 100% of the tiles. Similar to coin ownership, this could be caused by the hard rule of settlement where even when an agent had a outstanding debt of one coin it needs to surrender a whole tile, which seems to lead to some agents hoarding most of tiles in some executions.

5.5.2 Coefficient of Variation

In order to compare the macroscopic of the observables, we calculated the coefficient of variation for each framework. The Coefficient of variation (CV) is a standardized measure of dispersion of a probability distribution or frequency distribution. [29]

The CV is defined as the ratio of the standard deviation σ to the mean μ:

$$CV = \frac{\sigma}{\mu}. \tag{5.1}$$

For the five frameworks, the calculated CV for the three histograms (round count to completion, top 20% ownership of coins, top 20% of tiles ownership) is shown in the following table

Engine	Rounds to finish	Pareto Coins	Pareto Tiles
NetLogo	0.011	0.016	0.027
MASON	0.010	0.016	0.027
Repast	0.011	0.015	0.025
Mesa	0.011	0.016	0.028
Agents.jl	0.010	0.016	0.028

Table 5.1: Coefficient of variation for three chosen macroscopical observables

As CV is a unit-less measure, it is particularly useful for comparing variability across datasets with different units or scales. In the example, it shows almost identically values for almost all implementation with a small difference in Repast, specially the CV of the Pareto Tiles.

This difference can be seen in the histogram as well, and could be due to a small difference in the model or in the random number generators of Repast.

5.5.3 Fréchet Distance

The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along monotonic (not moving backwards) curves [24]. The formula for the continuous Fréchet distance is

$$F(A, B) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \{d(A(\alpha(t)), B(\beta(t)))\} \tag{5.2}$$

The discrete Fréchet distance is calculated using the above formulae for all points of two curves. The Fréchet distance between axis/frameworks of the three histograms (round count to completion, top 20% ownership of coins, top 20% of tiles ownership) were calculated. While there are techniques to approach Fréchet distance efficiently [25], the general formula was used, because of the few number of points in the curves.

The implementation of the discrete Fréchet distance is located here `https://gitlab.com/msc_tesis/StatisticalComparison`.

Fréchet distance for rounds, largest values are marked with *.

	NetLogo	MASON	Repast	Mesa	Agents.jl
NetLogo	0.0	68.0	87.0*	44.0	68.0
MASON	68.0	0.0	114.0*	72.0	95.0
Repast	87.0*	114.0*	0.0	130*.0	102.0*
Mesa	44.0	72.0	130.0*	0.0	97.0
Agents.jl	68.0	95.0	102.0*	97.0	0.0

Table 5.2: Fréchet distance for histograms of rounds to complete experiment

Fréchet distance for histogram of top 20% coin ownership, largest values are marked with *.

	NetLogo	MASON	Repast	Mesa	Agents.jl
NetLogo	0.0	89.0	299.0*	38.3	46.0
MASON	89.0	0.0	388.0*	97.0	43.0
Repast	299.0	388.0*	0.0	291.0*	345.0*
Mesa	38.32	97.0	291.0*	0.0	54.0
Agents.jl	46.0	43.0	345.0*	54.0	0.0

Table 5.3: Fréchet distance for histograms of top 20% coins ownership

Fréchet distance for histogram of top 20% tile ownership, largest values are marked with *.

	NetLogo	MASON	Repast	Mesa	Agents.jl
NetLogo	0.0	75.0	180.0*	33.1	41.6
MASON	75.0	0.0	255.0*	69.0	57.0
Repast	180.0*	255.0*	0.0	186.0*	198.0*
Mesa	33.1	69.0	186.0*	0.0	31.0
Agents.jl	41.6	57.0	198.0*	31.0	0.0

Table 5.4: Fréchet distance for histograms of top 20% tiles onwership

It clearly shows a difference in Repast implementation. It is interesting to notice that rounds also seem to exhibit the same behavior, because

- Agents do not expire and there is not concept of debts.
- The rent per tile is calculated from a random number between 1 and 50 coins.
- At the beginning of the round each agent collects 75 coins.

These rules indicate that the number of rounds should be very close if there is regularity in:

- The basic experiment setup and behavior are consistent among implementations.
- The underlying pseudo-random generation.

If the issue is the first point, it indicates the difficulty to develop simulations in Repast, as the code was checked several times without identifying any difference against the others implementations.

In the other case, it points at deficiencies in Repast's pseudo-random generation mechanism.

5.5.4 Anova and F-Test

Both ANOVA and F-test provide a p-value [45]. In null-hypothesis significance testing, where the null hypothesis is $H_0 : \mu_1 = \mu_2$, the p-value is the probability of obtaining test results at least as extreme as the result actually observed, under the assumption that the null hypothesis is correct [14].

A very small p-value means that such an extreme observed outcome would be very unlikely under the null hypothesis, and the null hypothesis is rejected, so that means are significantly different.

The ANOVA was applied to all the values in the histograms (round count to completion, top 20% ownership of coins, top 20% of tiles ownership) and F-Tests were applied the each pair of histograms.

The usual notation of * is followed

* $0.01 < p < 0.05$

** $0.001 < p < 0.01$

*** $p < 0.001$

The results are shown in the following tables.

Rounds to completion

The p-value for the ANOVA was 0.180 which is more than 0.05 which fail to reject null hypothesis, means are not significantly different. Nevertheless for F-Test between frameworks the results is

	NetLogo	MASON	Repast	Mesa	Agents.jl
NetLogo	-	0.291	0.275	0.893	0.376
MASON	0.291	-	0.027*	0.224	0.852
Repast	0.275	0.027*	-	0.331	0.041*
Mesa	0.893	0.224	0.331	-	0.297
Agents.jl	0.376	0.852	0.041*	0.297	-

Table 5.5: P-value for F-test for histograms of rounds to complete experiment

Here it can be seen that for rounds, the results of Repast are statistically significantly different for Agents.jl and MASON.

Top 20% of coin ownership

The p-value for the ANOVA was $< 10^{-5}$ which rejects the null hypothesis, and indicates the means for the histograms of the top 20% of coin ownership, are significantly different. The F-Tests between frameworks the results are

	NetLogo	MASON	Repast	Mesa	Agents.jl
NetLogo	-	0.860	$< 10^{-5}$***	0.669	0.459
MASON	0.860	-	$< 10^{-5}$***	0.800	0.571
Repast	$< 10^{-5}$***	$< 10^{-5}$***	-	$< 10^{-5}$***	$< 10^{-5}$***
Mesa	0.669	0.800	$< 10^{-5}$***	-	0.755
Agents.jl	0.459	0.571	$< 10^{-5}$***	0.755	-

Table 5.6: P-value for F-test for histograms of top 20% of coin ownership

Here it can be seen that for the top 20% of coin ownership, the results of Repast are statistically significantly different from all the other frameworks.

Top 20% of tiles ownership

The p-value for the ANOVA was$< 10^{-5}$ which rejects null hypothesis, and indicates the means of the histograms for the op 20% of tile ownership, are significantly different. The F-Tests between frameworks the results are

	NetLogo	MASON	Repast	Mesa	Agents.jl
NetLogo	-	0.423	$< 10^{-5}$***	0.494	0.388
MASON	0.423	-	$< 10^{-5}$***	0.909	0.949
Repast	$< 10^{-5}$***	$< 10^{-5}$***	-	$< 10^{-5}$***	$< 10^{-5}$***
Mesa	0.494	0.909	$< 10^{-5}$***	-	0.859
Agents.jl	0.388	0.949	$< 10^{-5}$***	0.859	-

Table 5.7: P-value for F-test for histograms of top 20% of tiles ownership

Here it can be seen that for the top 20% of tiles ownership, the results of Repast are statistically significantly different from all the other frameworks.

These results keep indicating a consistency issue in Repast which could be caused by a underlying problem related to pseudo-random number generation or in the experiment setup and behavior, which was has not been identified even after several code reviews.

5.6 Discussion

The first version of this project contained several more features for each section. After a rigorous review, they were simplified and streamlined to avoid overlapping. The nine identified features where chosen so as to minimize their overlap and make it feasible to score them independently.

In this work four experiment were proposed to exercise each framework. The set of experiments covers several common aspects of ABM in general, but left out some topics. One important topic which could be included in a updated version of the FSMM is graph support. Graphs can be implemented in any language, and their visualization might be useful for developing certain types of ABM models.

About the chosen frameworks, both MASON and NetLogo left a good impression after experimentation. Mesa and Repast left a troublesome impression due to speed and stability. Especifically, the MASON framework appears to be far more stable and fast compared to the other ones.

While NetLogo was overall the most capable framework, it was challenging to write some behaviors, especially because the only available data structure was the LISP style list and not common ones like maps. Additionally, the product was designed for the simulation to be in a single file which could be challenging for bigger projects.

Agents.jl was somewhat disappointing because it has some basic deficiencies in rendering. Also, the simulation uses GlMakie which hid some failures and makes it hard to debug. Another issue was that starting simulation required at least 3 minutes of prepossessing in a high-end laptop. Starting with a 'precooked' environment decreased this time but did not remove it. Other frameworks started almost immediately.

For Mesa the cumbersome integration between the web front-end and the back-end server made the simulation and development very slow. It was so slow to the point that some experiments, only the last image (screenshot) was rendered. That was not an issue for the other frameworks.

In the case of Repast, the challenge was that it was a Plug-In for Eclipse IDE which has had stability issues. It also created lots of helper files which were need for simulation but have to be recreated continuously as the source code changed. At one point the whole IDE stopped working to create new projects; to create new projects it was required to start from a previous project folder.

CHAPTER 6

Feature Space Maturity Scoring

"To push anything back into the past is equivalent to reducing it to its simplest elements. Traced as far as possible in the direction of their origins, the last fibres of the human aggregate are lost to view and are merged in our eyes with the very stuff of the universe."

Pierre Teilhard de Chardin, The Phenomenon of Man

6.1 Simulation Features

The following shows the scoring for the two subsections of the FSMM which were evaluated for each of the five frameworks.

Remember the scoring was based per item in a scale from 0 to 3 following the facilitator approach from [72].

The scores for the simulation features of the frameworks is based on their performance on the four experiments. Categories of the FSMM are mapped to the scores obtained by each framework; the results are shown in the following table.

Category	NetLogo	MASON	Repast	Mesa	Agents.jl
Usability	3	3	1	2	1
Basic Framework Functionalities	3	3	3	3	3
Locality	3	2	3	2	3
Agent Management	2	2	2	2	1
Force Fields	0	0	0	0	1
Summary	11	10	9	9	9

Table 6.1: Simulation Features Score

6.2 Interrogation Capabilities

The scores for the interrogation features of the FSMM for the frameworks is are based on the four experiments. The results are shown in the following table.

Category	NetLogo	MASON	Repast	Mesa	Agents.jl
Visualization	3	3	2	2	1
Statistical support for model attributes	3	3	3	3	3
Macroscopic statistical analysis	2	2	2	2	2
Nearly decomposable System view	0	0	0	0	0
Summary	8	8	7	7	6

Table 6.2: Interrogation capabilities Score

Based on those scores, we can create a quadrant-like graph with the top right being the best scored and the bottom left being the worst scored. This quadrant shows the superiority of NetLogo and MASON over the other freely available frameworks.

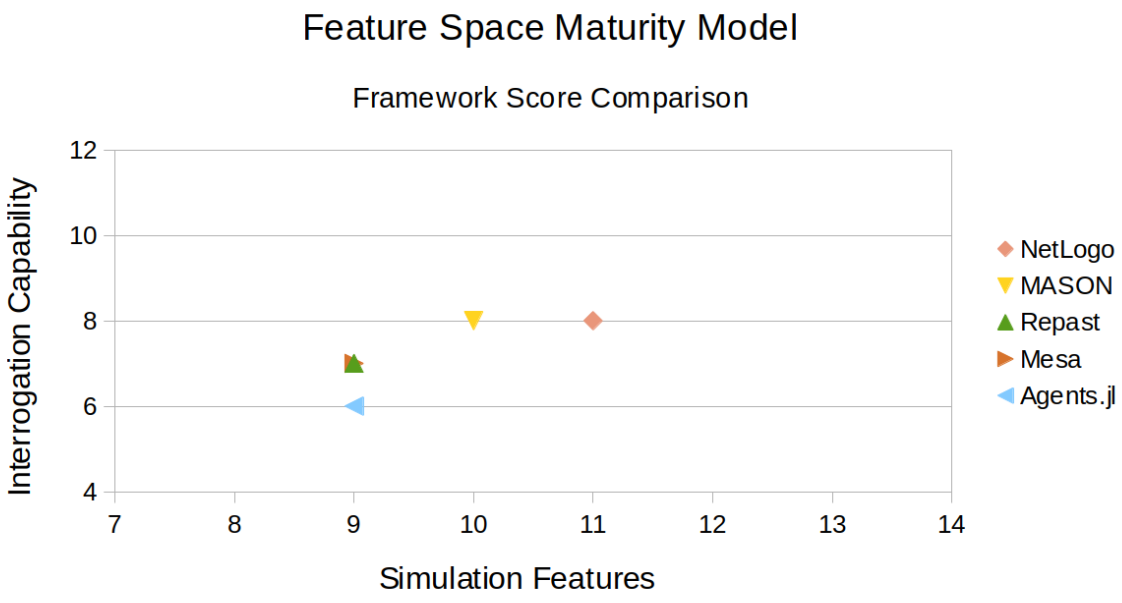

Figure 6.1: FSMM Score

CHAPTER 7

Architecture of ABM Frameworks

" All the efforts of the human mind cannot exhaust the essence of a single fly."

Thomas Aquinas

Software architecture is critical in ABM because it defines the structural organization of a system, guiding how different components interact and ensuring the system's scalability, maintainability, and performance. In ABM, the architecture helps manage the complexity of simulating large numbers of autonomous agents with unique behaviors and interactions. Without a well-designed architecture, the ABM framework can suffer from inefficiencies, poor performance, and difficulties in integrating new features or scaling to more complex simulations.

A well-structured software architecture impacts several quality attributes of an ABM system. For instance, scalability is essential when simulating thousands or even millions of agents, and a poorly designed architecture may lead to performance bottlenecks as the system grows. Modularity and maintainability are also critical because they allow developers to update or replace parts of the simulation without affecting the entire system. Furthermore, reusability of components – such as agent behavior libraries or environmental models – can be enhanced with a flexible architecture, making it easier to apply the framework to different simulation contexts.

The architecture also influences the reliability and efficiency of an ABM system. Clear communication paths between agents and a robust management of agent lifecycles are essential to ensure the accuracy of the simulation results. Poorly designed architectures can lead to issues such as deadlock, inefficient resource usage, or inaccurate modeling of agent interactions, which ultimately degrade the overall quality of the simulation.

In this chapter, we first present summaries of the architectures of the five ABM Frameworks studied in this research, then we state a set of architectural principle, and finally posit a proposed reference architecture for ABM.

7.1 Studied Architectures

This section briefly summarizes the architecture of the five studied ABM Frameworks.

7.1.1 NetLogo

NetLogo adopted the basic design principle from the Logo language of "low threshold, no ceiling". Low threshold means new users, including those who never programmed before, should find it easy to get started. No ceiling means the language shouldn't be limiting for advanced users.

Efficiency is always a vital goal for NetLogo as a multi-agent system development framewok, since many researchers would want to do large numbers of long model runs with as many agents as they can.

NetLogo is a complete system, not just a library. Just as important as the program itself is the materials it comes with. Developers devoted almost as much development effort to the Models Library as to the NetLogo software system.

The Models Library contains more than 140 pre-built simulations that can be explored and modified.

All of the models include an explanation of the subject matter, the rules of the simulation and suggestions for activities, experiments, and possible extensions. To aid learning and encourage good programming practice, the code for the simulations is well commented and as elegantly written as developers could make it [91].

7.1.2 MASON

MASON had several design goals from the very beginning. First, MASON was designed to have a small, high-performance, self-contained simulation core so that many models could be run in parallel, or could involve up to millions of agents. Second, MASON was designed to produce guaranteed identical results regardless of architecture when possible. Third, MASON was created with a Model-View-Controller (MVC) architecture with complete separation between the model and the visualization, and with model serialization. Fourth, as it came from the robotics community, MASON was meant to support a wide range of visualization facilities, including both 2D and 3D

support. Fifth, MASON was designed to be very easily modified and extended.

Roughly following an MVC architecture, MASON can be broken into two pieces. The first part is the model (the simulation proper) and the second part is the visualization. Unless one chooses to have model objects display themselves, the model and visualization are entirely separated, enabling model serialization and the removal or reinstatement of the visualization mid-run [57].

This separation of logic and visualization allowed to quickly run the Pareto experiment in parallel without the need of a UI.

MASON is not a executable program but a library. Yet, this is not a major issue, due the plethora of Java editing tools available like NetBeans or IntelliJ. Also this library approach allows the tool to be quickly reused in virtualized or containerized environments like Docker [23].

7.1.3 Repast

Repast architecture was supposedly designed to be flexible, extensible, and easy to use, allowing for the development of complex agent-based simulations. The key components of the Repast architecture include:

Agents: The core of any ABM. In Repast, agents are objects that represent entities in the model. They have states, behaviors, and interactions with other agents and the environment. Agents are typically implemented as Java or C# classes, depending on the version of Repast being used.

Environment: The space in which agents operate. Repast supports various types of environments, including grids, continuous spaces, and networks (graphs). The environment provides the context for agent interactions and movement.

Scheduler: Manages the simulation's execution timeline. The scheduler handles the timing of agent actions and events, ensuring that the simulation progresses in a coherent and controlled manner. It can be used to schedule actions at specific times or

with specific intervals.

Data Collection and Analysis: Repast includes tools for collecting and analyzing data generated during the simulation. This can include logging agent states, interactions, and other relevant metrics. The collected data can be used for post-simulation analysis and visualization.

Visualization: Provides graphical representations of the simulation state and agent behaviors.

Repast uses a DSL called ReLogo based on Logo and using the Groovy dynamic language. This was chosen supposedly due the ease of use of the Logo programming language and its associated programming idioms [74].

Repast is not an executable program but an Eclipse plug-in. While this should be an advantage in theory, in practice Repast is constrained by the stability or instability of Eclipse releases – which recently has shown some instabilities.

7.1.4 Mesa

Mesa is an Agent-Based Modeling Framework with tries to fill the missing ABM for Python.

Mesa allows users to quickly create agent-based models using built-in core components (such as agent schedulers and spatial grids) or customized implementations; visualize them using a browser-based interface; and analyze their results using Python's data analysis tools.

Designing a new framework from the ground up also allows the developers to implement features not found in existing frameworks. For example, other ABM frameworks tend to use a single agent activation regime by default; in Mesa, there are several agent schedulers and require the modeler to specify which one should be used [62].

7.1.5 Agents.jl

Agents.jl is a pure Julia framework for ABM. The basic ideas of the Agents.jl architecture are:

- **Agents** are defined as Julia structs or mutable structs, which can hold any attributes needed for the simulation. Agent behavior is encapsulated in functions that define how agents interact with each other and the environment.
- Julia handles three **spaces**.
  - **GridSpace**: A discrete grid where agents occupy specific cells. It can be one-dimensional, two-dimensional, or multi-dimensional.
  - **ContinuousSpace**: A continuous space where agents can move freely in any direction. It allows for more granular control over agent positioning.
  - **GraphSpace**: A space based on graph structures, allowing agents to occupy nodes and move along edges. Useful for modeling networks and other relational structures.
- The **model** in Agents.jl is a container that holds the agents, the space, and any additional parameters or data structures needed for the simulation.
- Julia has a **Discrete Event Scheduler** which handles the execution of agent actions. It allows for scheduling actions at specific times or in specific sequences and includes tools for collecting data during the simulation, such as agent states, interactions, and aggregate statistics.

Agents.jl is not and executable program but rather a Julia library. Editing needs to be done by a third party program. Currently the most important Julia editing tool is Visual Studio Code by Microsoft.

7.2 Proposed Architecture Principles

If the Feature Space describes which features should the ABM have, the Computational Properties describe how they should be implemented. The Computational Properties of an ABM framework architecture should enable the ease of use of the framework, understood as the capability of a user to readily and successfully perform tasks with a product without the need for advanced explanation and an instruction manual [77]. In the case of ABM frameworks, the concept could be refined further as the ability to develop or update a model in the most natural way without having to struggle with the underlying the framework's implementation.

The principles that we kept in mind when developing the architecture were

- **Stability & Maintainability**: Stability means that the framework can be used without constant crashing. Maintainability is an important quality attribute for almost every software product. Maintenance is considered an expensive phase in the software development life cycle as it consumes most of the effort allocated to the software system [97]. Similarly, models coded in ABM frameworks most probably will be modified and refined during their lifetime. While maintainability has many facets, this thesis will only focus on the tools that the framework provides to create a maintainable model, including tools that help create comments and semantic validations. In the provided FSMM this topic relates to Usability.

- **Readability**: According to Popper, science becomes possible because a thesis can be linguistically presented and thus critically assessed, becoming an object so it can be tested or falsified by others [76]. Similarly, an implementation of a model should be easily readable for its quality and intent to be correctly assessed. The most important aspect for good readability in ABM are the rules of both the agent and the environment. In the provided FSMM this is topic also relates to **Usability**.

- **Implementation**: The complexity of implementing a model in a ABM framework should be similar to the complexity of describing the model in a formal way. The fewer artifacts and boilerplate code required from the underlying technology the better. In the provided FSMM this topic relates to **Basic framework functionality**.

- **Syntax expressiveness**: due to its expressiveness power, text is the standard way to code agent and environment behavior. This behavior language should have enough expressive power to support the simulation requirements in a natural way and avoid negative impact in the code quality [11]. Expressiveness also measures how much can be expressed in the ABM framework without changing its internal implementation. In general, there should be a way to implement any new arbitrary complex behavior without having to change the source code of the ABM [30]. In the provided FSMM this topic relates to the **Agent management** and the **Locality** simulation features.

- **Visualization**: The framework should be able to present a graphical representation of the model with minimal configuration from the developer [53]. While visualizations of very large models might be too expensive and of limited value, they are one of the most useful tools in the development and debugging of a model.

- **Service Oriented**: Due to the raise of cloud computation and the nature of the ABM simulations, which are in most the cases computationally intensive (and costly) and required to be run several times, it is expected the framework to be used as collection of services.

Based on those five principles, the next section develops a proposed reference architecture.

7.3 Reference Architecture

The following reference architecture is defined for 2-dimensions, discrete time ABM frameworks with a homogeneous, single-layered grid. While 3-dimensions simulation share many similarities, they are much more resource consuming andm for many cases, they may be modeled in a 2-dimensions framework. While this architecture could be used as a initial research point for continuous-time frameworks, it is not intended for them.

For the sake of clarity and simplicity only two types of composite structures are described: a sorted list (called array) and unsorted list (called set). There is a special case of Matrix for the grid which may be seen as a array of array of patches. In both cases they do not allow duplicates and use the ¡¿ notation to indicate the type of the components.

It is expected that the architecture should have at least these 3 main top-level types of entities: the environment, the entities and the relations. The entities can be split in three sets: the patches, the agents and the vertices. The relations can be split also in two: logic relations and directed edges. Therefore there are 6 main types of entities:

- Environment
- Entity: Patch
- Entity: Agent
- Entity: Vertex
- Abstract relation: Relation
- Abstract relation: Directed Edge

While the description follows an OO approach, it should be general enough to be used by non-OO implementations. Implementation details are purposefully left vague,

as developers should be able to decide about them. Description and interaction among the defined types are explained in the following paragraphs:

7.3.1 Environment

There is only one environment per simulation and there should be ways to define custom behavior for the environment step. Environments need to have a grid of patches which could have one of three topologies: *plane*, or *cylinder* (wrapped horizontally) or a *torus* shape (wrapped both horizontally and vertically).

There should be a list of drawing layers, and each entity should be assigned to a single layer. It should be able to request the environment to create an image for the requested drawing layers with a provided scale.

7.3.2 Patch

A patch is a building block for the model. The most common shapes are squares. While some ABM frameworks allow using a different morphology for patches (like triangle or hexagon) their value is limited, and behavior provided by those non-standard morphologies should be covered with square patches plus directed edges.

A patch should be considered a 1×1 tile. For an environment with a grid width of w and a height of h there should be $w \times h$ patches. A patch can contain an unlimited amount of agents and vertices. There should be a way to interrogate the patch to collect both its agents and vertices.

There is a subtle case when an agent is on an edge, but not (in itself) on the patch. The agent's coordinates are bounded with reference to a place within the patch. Thus, there should also be a way to identify agents *above* the patch (by virtue of them being on an edge).

There should be a way to assign custom behavior and attributes to the patch and, given a patch, there should be a way to identify its neighbors using the environment

topology and one of the three neighbors metrics (radial, Moore or von Neumann). For visualization purposes there should be a way to define the color of a patch.

7.3.3 Agent

Agents are the foundations of ABM. An agent is considered dimensionless but for visualisation purposes, there should be a way to assign it a shape, color and size.

There should be two ways to create a agent: the environment spawns it or a parent agent hatches it. It is expected that when a parent hatches a child, the child location matches the parent location.

An agent needs to be on a patch or on a edge. There should be a way to identify where the agent is, its xy coordinates and, if it is in a edge, the position in the edge (0 in source, 1 in destination). As mentioned above, there should be a way to collect agents on or above a patch.

There should be a way to assign custom behavior and attributes to the agent, and there should be a way to define the default behavior entry point of each model step.

There should be a way for agents to die. After an agent dies, all of its framework-defined relations should be dropped and it should not be accessible using the framework's API.

7.3.4 Vertex

Vertices are considered dimensionless. For visualisation purposes, there should be a way to assign them shape, color and size. They are the end points (source or destination or both) of edges. There should be a way to assign custom behavior and attributes to a vertex. They can be created, moved or destroyed by the environment.

A vertex needs to be assigned to a patch with some specific coordinates but they could be modified during the simulation. Once a vertex is destroyed, all of its framework-defined relations should be dropped and it should not be accessible via the framework's

API.

7.3.5 Relation

A relation is a way in which several entities link to each other. They usually connect distinct entities but some associate an entity with itself. The arity of a relation is the number of entities it connects. The direction of a relation is the order in which the elements are related to each other. The converse of a relation carries the same information and has the opposite direction [43].

For this ABM framework architecture the arity of a relation will be always 2 (an origin and a destination), it should have a name and could be bidirectional or unidirectional. While it might be useful in some cases to simplify the architecture there could not be two or more relations with the same name, origin and destination.

7.3.6 Directed edge

Directed edges, or edges for short, are a special type of unidirectional relation, only connecting vertices and could be considered part of the topology as agents can be removed from the patches grid and added to an edge.

There should be a way to add custom attributes to edges, and there should be a well-known weight attribute which is useful in several cases to indicate some cost of using that edge – e.g. to traverse the edge from origin vertex will be 100% of the cost, but traversing from the middle will be 50%. Usually the cost is expected to be linear to the distance but other implementations could choose a different cost function.

7.3.7 UML diagrams

The following diagABM.

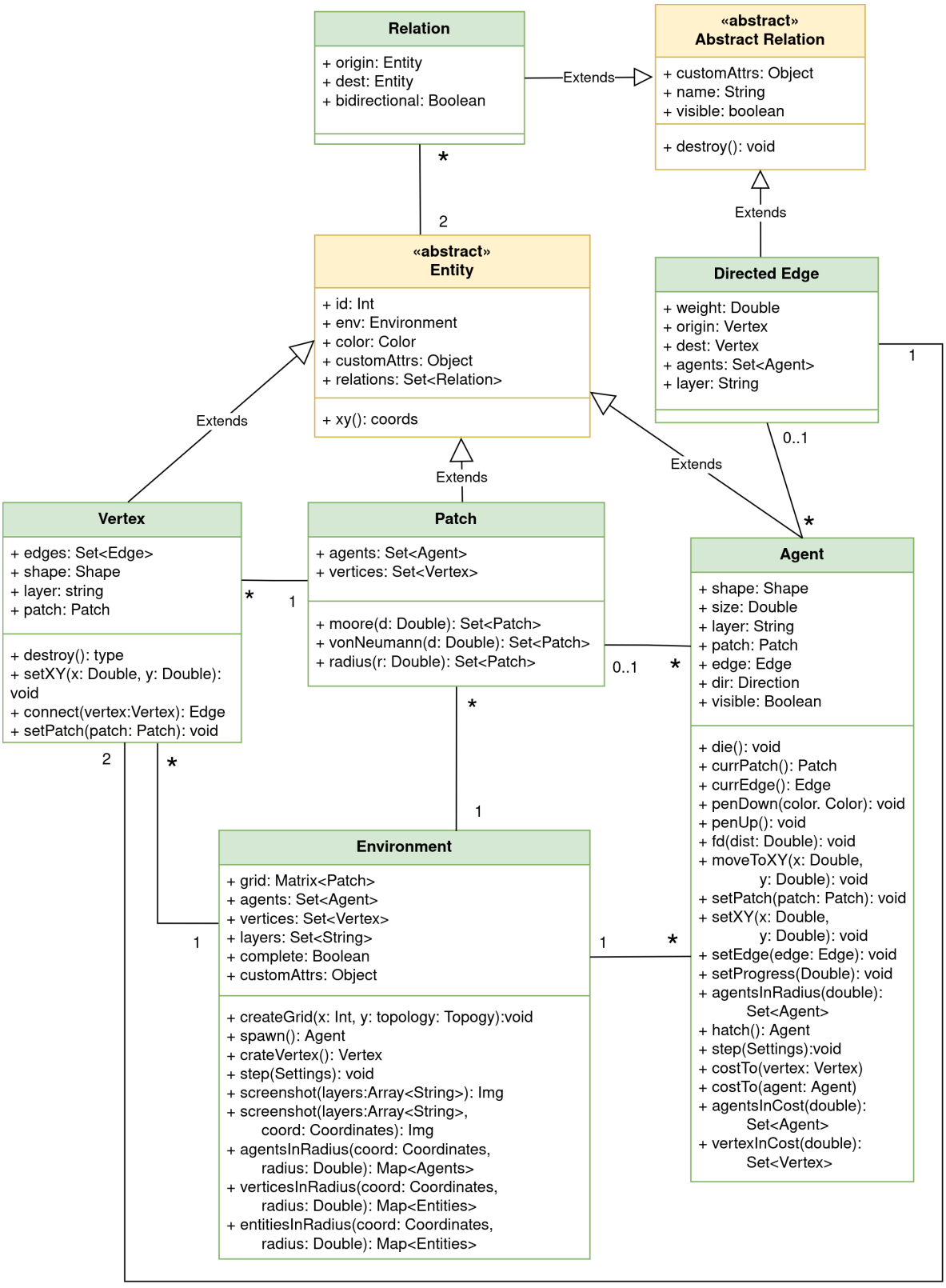

Figure 7.1: UML Diagram of top ABM elements

The diagram that follows presents a services architecture for an ABM Framework. It consists of six initial services, colored in green, four external services, colored in blue, with the final UI colored in orange. The internal services in bottom-up order are:

- Repository Service: A simple service which can process repository request (list files of a folder, read & write files, etc.). This service should have a back-end client to a distributed version control system.
- Container provider Service: This service should be able to create virtual machines which will run the ABM Framework.
- ABM Framework: This service is the actual framework detailed above.
- Executions Manager: This service should keep track of running simulations and request the Container provider service to start and stop machines.
- Request Broker Service: This service works like a reverse proxy [63], it is a server that sits in front of web servers and forwards requests from the UI to the rest of the services.

The external services are services which don't need to be developed but can use already existing products, with varying customizations. They are:

- Static Files Manager: A simple resources container which request static resources (like html, js, json files) as required.
- Version Control System: A Distributed version control system that tracks versions of files. It is often used to control source code by of a development team. The most used one currently is Git.
- Container: A container is similar to a virtual machine which has its operating system, except that unlike virtual machines they don't simulate the entire computer,

but rather create a sand boxed environment that pretends to be a virtual machine. The most used one currently are Docker containers [23].

- Container Manager: A tool which can manage (create, destroy, scale) containers. Currently the most used one is Kubernetes.

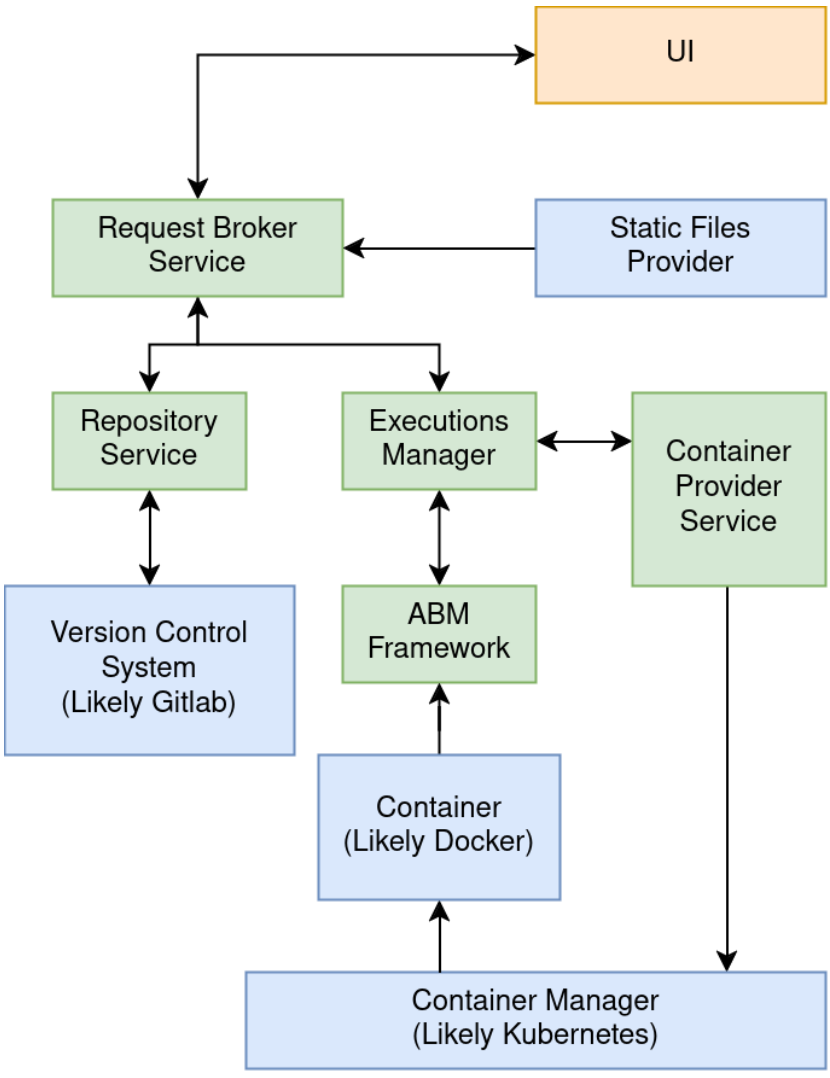

Figure 7.2: Architecture of the framework as services

7.4 Complete Entity Type Description

The following tables describe attributes and methods which should be implemented by each instance type. We describe the attributes, not because they should be visible, but as supporting documentation for the implementation. It is up to the developers to identify the best way to present the attributes, depending on the case and chosen computer language, it could be using setter or getter methods, public attributes, fully private only for internal use, etc.

The final field in the attributes indicates whether the attribute is not expected to change. For container structures this means the container is not expected to change, but its content still could change.

7.4.1 Environment

The following table describes the required environment attributes:

Name	Type	Final	Usages
grid	Matrix<Patch>	Yes	Grid used to locate agents.
agents	Set<Agent>	Yes	List of live agents.
vertices	Set<Vertex>	Yes	List of valid vertices.
layers	Set< String>	Yes	List of visibility layers.
complete	Boolean	No	Indicate if a simulation reached completion criteria.
customAttrs	Object	No	Simulation specific attributes.

Table 7.1: Required attributes for the environment type

The environment provides the methods with the InRadius suffix (*entities*InRadius, and its siblings methods) to identify all entities inside a radius of a xy point. They include agents off the grid and over an edge but which still are inside that radius. To identify all entities in the grid inside a radius, the valid method is *entities*InRadius for a given entity class.

The following table describes the required environment methods:

Name	Input	Output	Description
createGrid**	x: Int, y: Int, t: Topology, Patch sub-type	void	Creates a grid with square topology of x by y. Topology would not wrap (a plane), or could wrap vertically (cylinder) or could wrap both vertically and horizontally (torus).
spawn**	Agent sub-type	Agent	Creates an agent.
createVertex**	Vertex sub-type	Vertex	Creates a vertex.
step*	Settings	void	Trigger steps in all its agents and also its own step behavior.
screenshot	layers: Array<String>	Img	A rendered image of the selected layers for the whole grid.
screenshot	layers: Array<String> coord: Coordinates	Img	A rendered image of the selected layers for the selected subgrid.
agentsInRadius	coord: Coordinates, radius: Double	Set <Agent>	Finds all agents which lay around coord with a radial distance $\leq$ radius whether they are in grid of in a edge.
verticesInRadius	coord: Coordinates, radius: Double	Set <Agent>	Finds all vertices which lay around coord with a radial distance $\leq$ radius whether they are in the grid or at an edge.

Table 7.2: Required methods for the environment type (1/2)

patchesInRadius	coord: Coordinates, radius: Double	Set <Agent>	Finds all patches which its center lay around coord with a radial distance ≤ radius.
entitiesInRadius	coord: Coordinates, radius: Double	Set <Agent>	Finds all entities which lay around coord with a radial distance ≤ radius whether they are in grid of in a edge.

Table 7.3: Required methods for the environment type (2/2)

Notes:

* There should be a way to add custom behavior for the environment for the step method.

** If there are ways to define sub types of patches, vertices or agents, there should be a way to define the sub-type at the moment of creating them.

7.4.2 Entity

The following table describes the required attributes by all sub types of entity:

Name	Type	Final	Usages
id	Int	Yes	Unique identification number.
env	Environment	Yes	Parent environment.
color	Color	No	Color used for rendering.
customAttrs	Object	No	Simulation specific attributes.
outgoingRels	Set<Relation>	Yes	List of outgoing relations for this entity.
incomingRels	Set<Relation>	Yes	List of incoming relations for this entity.

Table 7.4: Required attributes for all sub types of entity

The entity provides the method *entities*InRadius (and its siblings methods) to identify all entities in the grid inside a radius of a xy point. They do not include agents off the grid and over an edge but which still are inside that radius. To identify

those, the valid method is *entities*InRadius from the environment class.

The following table describes the required methods by all sub types of entity:

Name	Input	Output	Description
xy	none	Coordinates	The coordinates of the entity in the grid.
agentsInRadius*	coord: Coordinates, radius: Double	Set <Agent>	Finds all agents which lay around coord with a radial distance $\leq$ radius if they are in grid.
verticesInRadius*	coord: Coordinates, radius: Double	Set <Agent>	Finds all vertices which lay around coord with a radial distance $\leq$ radius if they are in grid.
patchesInRadius*	coord: Coordinates, radius: Double	Set <Agent>	Finds all patches which its center lay around coord with a radial distance $\leq$ radius.
entitiesInRadius*	coord: Coordinates, radius: Double	Set <Agent>	Finds all entities which lay around coord with a radial distance $\leq$ radius if they are in grid.
distanceTo**	dest: Entity	Double	Distance from this entity to dest entity.

Table 7.5: Required methods for all sub types of entity

Notes:

* If the entity is outside the grid (only agents could be) the method will throw an exception.

** If either this entity or the dest entity is outside the grid (only agents could be) the method will throw an exception.

7.4.3 Patch

The following table describes the required attributes for the patches:

Name	Type	Final	Usages
agents	Set<Agent>	Yes	List of agents inside the patch.
vertices	Set<Vertex>	Yes	List of vertices inside the patch.

Table 7.6: Required attributes for all sub types of entity

The following table describes the required methods for the patches:

Name	Input	Output	Description
moore	size: Int	Set<Patch>	List of patches with Moore distance $\leq$ size.
vonNeumann	size: Int	Set<Patch>	List of patches with von Neumann distance $\leq$ size.
radius	size: Double	Set<Patch>	List of patches where distance to its center $\leq$ size.

Table 7.7: Required methods for the patch type

7.4.4 Agent

The following table describes the required attributes for the agents:

Name	Type	Final	Usages
shape	Shape	No	A shape for rendering.
size	Double	No	The size for rendering.
layer	String	No	A rendering layer. If the requested screenshot did not request the agent's layer, the agent will not be rendered.
patch	Patch	No	The patch where the agent is located, if any.
edge	Directed Edge	No	The vertex where the agent if located, if any.
dir	Direction	No	The direction of the agent.
visible	Boolean	No	If the agent should be rendered, even if the agent's layer was requested.

Table 7.8: Required attributes for the agent type

The following table describes the required methods for the agents:

Name	Input	Output	Description
die	none	void	Agent is removed from live agents. From the scheduler. All its relations are dropped.
currPatch	none	Patch void	The current patch or null if the agent is located in an edge
currEdge	none	Edge void	The current edge of null if the agent is located in the grid
penDown	color: Color	void	Agent will leave a trace when moving (by the fd, moveToXY methods). It should be rendered on the agent layer. There could be and optional render color and line thickness.
penUp		void	Agent will stop leaving a trace when moving.
fd	dist: Double	void	The agent moves forward by some distance. Will stop if it reaches a topology boundary. For agent in edge max distance is 1.
moveToXY	coord: Coordinates	void	Move agent to the to XY position Can also be used if the agent is on an edge to move it back to the grid. It is important to mention it should be the shortest distance which could be achieved by wrapping in some topology.
setPatch	patch: Patch	void	Teleport agent to the center of the patch. Can also be used if the agent is on an edge, to put it back to the grid. This method does not leave a trace, even with penDown.

Table 7.9: Required methods for the agent type (1/2)

Name	Input	Output	Description
setXY	coord: Coordinates	void	Teleport agent to the xy position in the grid. Can also be used if the agent is at an edge to put it back to the grid. It is important for this method not to leave a trace even with penDown.
setEdge	edge: Edge	void	Moves agent off the grid or other edge and puts it in the defined edge, in the origin vertex.
setProgress*	progress: Double	void	Set the progress of this agent in the edge. 0 means agent is at the origin vertex, 1 means at the destination vertex.
hatch	agent sub-type *	Agent	Create a new agent in the same spot of the original agent.
step***	settings: Object	void	Triggers the agent step behavior.
costTo**	dest:	Double	Calculate lowest cost to reach dest entity from the current position of the agent in the edge.
agentsInCost*	maxCost: Double	Set <Agent>	List of agents which have a distance cost lower than maxCost
vertexInCost*	maxCost: Double	Set <Agent>	List of vertices which have a distance cost lower than maxCost

Table 7.10: Required methods for the agent type (2/2)

Notes:

* If the agent is on the grid, not in an edge, this method will fail.

** If the agent or the dest entity are on the grid, not in an edge, this method will fail.

*** There should be a way to add custom behavior for the step method in agents.

7.4.5 Vertex

The following table describes the required attributes for the vertices:

Name	Type	Final	Usages
originOf	Set<Edge>	Yes	The list of edges which have this vertex as origin.
destOf	Set<Edge>	Yes	The list of edges which have this vertex as destination.
shape	Shape	No	A shape for rendering.
layer	String	No	A rendering layer. If the requested screenshot did not request the vertex's layer, the agent will not be rendered.
patch	Patch	No	The patch where the vertex is located.

Table 7.11: Required attributes for all sub types of entity

The following table describes the required methods for the vertices:

Name	Input	Output	Description
destroy	none	void	Destroy this vertex, including incoming and outgoing relations and directed edges.
setXY	coord: Coordinates	void	Sets the xy in position in the grid.
connect	dest: Vertex	Edge	Creates a directed edge with this vertex as the origin and dest as the destination

Table 7.12: Required methods for the vertices

7.4.6 Abstract Relation

The following table describes the required attributes for all sub types of abstract relation:

Name	Type	Final	Usages
name	String	Yes	A name for this relation.
visible	Boolean	No	Toggle for this relation to be rendered.
customAttrs	Object	No	Simulation-specific attributes.

Table 7.13: Required attributes for all sub types of abstract relation

The following table describes the required methods for all sub types of abstract relation:

Name	Input	Output	Description
destroy	none	void	Destroy this abstract relation

Table 7.14: Required methods for all sub types of abstract relation

7.4.7 Relation

The following table describes the required attributes for relations:

Name	Type	Final	Usages
origin	Entity	Yes	The origin of this relation.
dest	Entity	Yes	The destination of this relation.
bidirectional	Boolean	Yes	If the direction is bidirectional, i.e. There is another relation with same name but with switched origin and destination values. When changing values in one relation, it changes values in the other, and if one relation is dropped, the other gets dropped automatically.

Table 7.15: Required attributes for all relations

7.4.8 Directed Edge

The following table describes the required attributes for directed edges:

Name	Type	Final	Usages
weight	Double	No	The weight of the edge. Used to define a traverse cost.
origin	Vertex	Yes	The origin vertex.
dest	Vertex	Yes	The destination vertex.
agents	Set<Agents>	Yes	All agents in this edge.
layer	String	No	The drawing layer when rendering.

Table 7.16: Required attributes for all relations

7.4.9 Global Methods

Global methods are useful to implement required behavior not related to the interaction in the simulation. The following table describes some required global methods:

Name	Input	Output	Description
processReq	req: HttpRequest	HttpResponse	Process a http request and returns a http response
addTest	name: String behavior: Function	void	Adds a test which will be execute later.
runTests	none	results report	Run all provided tests and returns a success/failures report
readFile	name: String	binary	Reads a file and returns its contents.
writeFile	name: String data: binary	void	Writes contents to the specified file.
deleteFile	name: String	void	Deletes a file.

Table 7.17: Required attributes for all Global Methods

CHAPTER 8

AB-X: POC for the Reference Architecture

'To Avoid Criticism, Say Nothing,
Do Nothing, Be Nothing."

Aristotle

8.1 AB-X Design

AB-X is a POC of the defined reference architecture. It is located here `https://gitlab.com/msc_tesis/AB-X/`. The following subsections describe its design and the current implementation status.

8.1.1 Overview

For AB-X, several goals were considered, including: match the proposed architecture as closely as possible, create a product which followed the architectural goals, keep the framework simple and maintainable, have a simple yet powerful syntax, make it efficient in execution and amenable to develop it very quickly.

Based on those requirements the following design decisions were taken:

- The simulation language should have a simple syntax, basic typing support and admitting operator overloading.
- The framework has to be developed over a well-supported, mature technology.
- The POC should be fully service-oriented and every behavioral aspect should be modifiable using code.
- The presentation layer should be web-based.
- There has to be support to mathematical concepts such as sets and complex numbers – out of the box.
- Useful examples are as important as the framework itself.

Based on those premises, the language to be used for the simulation framework will be a slightly modified version of JavaScript. JavaScript is a high-level, just-in-time compiled language that conforms to the ECMAScript standard [46]. The two

modifications to be made on JavaScript's core are: the ability to validate functions input and output types, and the ability to perform custom classes operator overloading.

The framework was developed as two services: an editing service and an execution service. The architecture states the idea of an execution broker as a third service; however, due the scope of this POC, this service was not developed as part of this research. The editing service is in charge of also managing the execution service.

The baseline technology was Java using the GraalVM JDK. GraalVM is a advanced JDK which supports native JavaScript very efficiently and allows for operator overloading [87]. On top of it, the underlying technology of the services was Spring Boot. Spring Boot is an open-source Java-based framework used to create stand-alone, production-grade services or applications quickly and with minimal configuration [98].

For the web presentation layer three main libraries were used. The main UI layout and functionality was done using EasyUI. EasyUI is a collection of user interface components based on jQuery, designed to simplify the development of web applications [48]. The Ace editor, an embeddable code editor written in JavaScript, was used for text highlighting – it can be easily embedded in any web page and JavaScript application [2]. Finally, for the charts, Charts.JS was used: a simple yet flexible JavaScript charting library for the modern web [19].

When a user wants to run a model, they first select a runner (in this POC the only runner is localhost), then they need to select the baseline model file. Then they may choose two types of execution: *standard* (enforces typing protection but is resource consuming) or *performance* (low in resource consumption but does not validate types).

While users can create several environments in the same execution (for example to execute unit test) there is only on environment which is sync-up with the UI and is accessed through the global methods setEnv, getEnv and stepEnv which respectively sets, gets and process one step of the global environment.

Finally users need to choose which type of execution they want:

- Evaluate: will process the chosen file. If in the file the user sets the global environment, then the user could use the buttons to execute step by step or continuously.

- Run simulation: will perform the Evaluate process and then it will start a thread which triggers the stepEnv method continuously. A stepEnv without the global environment or with a completed global environment will return false and will stop the continuous execution thread.

- Execute tests: will perform the Evaluate process, and then it will do a runTests method which should run all defined tests and provide a report of success and failures.

There is the concept of a paused session, where there is no thread which keeps running the global stepEnv method. The UI provides the functionality to pause or terminate a session when the simulation is running continuously. A paused simulation can be also terminated, restarted, or execute just one step. A terminated simulation will release all allocated resources.

There is also the ability to interrogate the global environment using getEnv() method plus some query over the environment. The result is transformed to string and presented in the UI.

The following image shows the workflow of simulation in AB-X in the three possible execution types. The blue elements are the simulation type, the purple ones are processing sections and the yellow ones are decision sections.

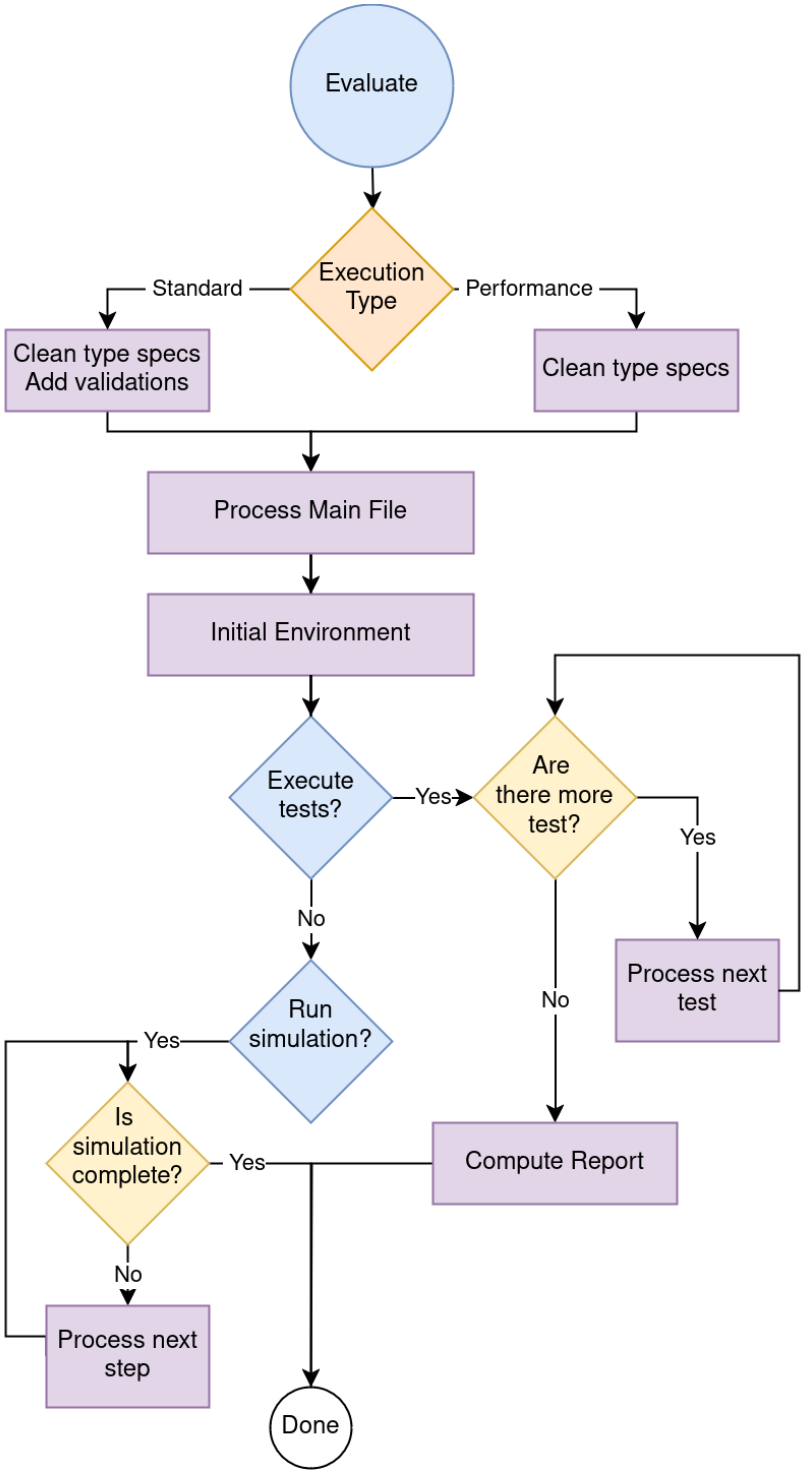

Figure 8.1: Execution screen showing simulation running

It is worth mentioning that AB-X implements a research-supporting feature which was proposed but not evaluated in the FSMM and therefore not added to the original reference architecture. This feature is the basic support for units of measurement including distance (meter based) and time (second based) and plane angles (radian based) out of the box. Examples of units were added in the companion documentation and in the coding snippets section.

8.1.2 Main simulation classes

The AB-X simulation service the AB-X Framework's main component. This service is in charge of preprocessing description files and contains all simulation logic. The following subsections describe the main classes of this framework.

StringProcessor.java & TypingSupport.java

While AB-X syntax is JavaScript based, AB-X framework supports typed variables for inputs and output. These classes allow the typing by consuming original files and output post-processed files which will have type support removed and optional validation.

This transformer maintains exact lines so debugging becomes easier. For example, this input:

```
1 function isZero(element: Number):Boolean{
2     return element === 0 ;
3 }
```

When *type validations* is enforced, it becomes:

```
1 function isZero(element){assertValidInstances([[element,Number
   ]]);return _assertType().setType(Boolean).evaluate = (
   function (){
2     return element === 0 ;
3 }).apply(this,arguments);}
```

Yet, the user never sees this temporarily generated file. In the case of errors being

detected, the user will see the error in the expected line. Suppose the isZero is in file Test.js, then if invoked with a string "hi" the error will be shown:

```
hi is not of function Number()
  at <js> assertIsInstanceOf(Utils.js:135:4300-4337)
  at <js> assertValidInstances(Utils.js:143:4455-4498)
  at <js> x(Test.js:1:42-76)
```

If *performance* is requested, it becomes:

```
function isZero(element){
    return element === 0 ;
}
```

Entity.js

Is a JavaScript class which matches closely the entity super type in the reference architecture. It is a constructor: it expects the environment and has the methods to create, get and drop relations with other Entities.

It has a method to define the xy which needs to be implemented by sub-classes.

ABMAgent.js

It is a JavaScript class which matches closely the Agent type defined in the reference architecture. It extends from Entity as expected from the OO paradigm. The constructor also allows to identify the class subtype which is used in the grid to enable searching by agent type.

Subclasses of ABMAgent.js are expected to override behavior of two main methods, the setup for initialization purposes and step which is called automatically by the framework in each round.

ABMPatch.js

This is a JavaScript class which matches closely the Patch type defined in the reference Architecture. A simulation can extend from this class to add custom methods.

ABMVertex.js

This is a JavaScript class which matches the vertex type defined in the reference Architecture, but it is yet to be completed. The missing features are related to edge management.

ABMEdge.js

This is a JavaScript class which matches the vertex type defined in the reference Architecture, but it is yet to be completed. The novel approach is related to the ability to remove an agent from the grid and add it to the edge and the opposite: moving from the edge to the grid.

ABMGrid.js & Grid.java

While the initial reference architecture does not explicitly split environment vs grid, there are several grid specific functionalities in the environment, which can be isolated in a single class. In AB-X that grid functionality is split between ABMGrid.js and ABMGrid.java. The first is a JavaScript class which matches the grid functionality without the neighbor searching capability; this was written in ABMGrid.java to speed it up.

ABMEnv.js

This is a JavaScript class which matches the Environment type defined in the reference Architecture. Forward thinking on the ability to have multilayered grids, the entire grid capability was designed in the grid classes described above. This class can be extended by a simulation and it is expected to override the setup method with custom simulation behavior.

ABMChart.js

Chart is a container which is filled with data from the simulation. It does some preprocessing of the data before sending it to the UI where it is rendered by the Chart.js library. There are two subtypes: one for the time/round and one for histograms. Both charts expect a function which provides data for the chart. The time based chart will collect data after each round but for histogram and custom chart the simulation needs to call sampleHistograms and sampleCharts respectively.

Example of use of Charts are shown in the following coding snippets section.

8.1.3 Coding snippets

The are over 80 examples provided out of the box with AB-X, in addition to the four FSMM experiments. They range from set examples to unit test to full simulation like the Inner Solar System and GOL example. The following examples can be found in the source code and were chosen to identify key sections of the AB-X product.

Add a simple text with one assertion:

```
addTest("Expected Success", ()=>{
    Assertions.assertTrue(true);
});
```

A simple agent which walks 2 units and rotates 25^{o} degrees each round

```
class WalkingAgent extends ABMAgent{
    setup(){
        this.counter = 0;
        this.aliveStrokeSteps = 100;
        this.color = namedColor("green");
        this.pen.width=4;
        this.penDown();
        this.shape = "delta";
    }
    step(){
        this.rotateDeg(25);
        this.fw(2);
    }
}
```

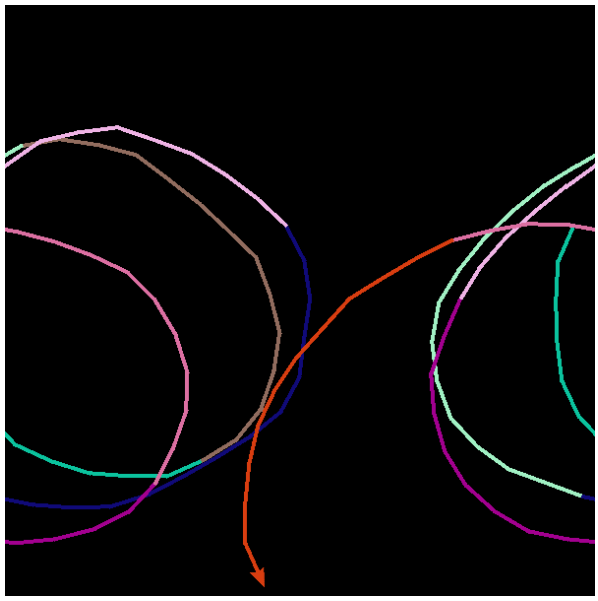

Figure 8.2: Figure shows the view-port of the agent leaving a stroke and also keep changing color

A planet agent, which will rotate certain degrees around a star. Using type and International Unit System (SI) units.

```
enableSIDistance(true);
enableSIAngle(true);
enableSITime(true);
let systemSize=500*Gm;
class Planet extends ABMAgent{
    setup(label:String, dist:UnitSystem,size:Number,
          angularVelocity:UnitSystem, color):Planet{
        this.label = label;
        this.setLocation(systemSize/2+dist,systemSize/2);
        this.angularVelocity = angularVelocity;
        this.size = size;
        this.color = color;
        this.penDown();
        return this;
    }
    rotate(sun:Star, time:UnitSystem){
        const angle = this.angularVelocity*time;
        this.rotateAround(sun,angle)
    }
}
```

The environment describing the setup method is creating each planet agent.

```
class Space extends ABMEnv {
    sun;
    setup(){
        this.patches().each(patch=>
            patch.color=namedColor("black"));
        this.sun = this.spawn(Star).setup("Sun");
        const fullCircle = 360*deg;
        this.sun.createRelation("planet", this.mercury = this.
           spawn(Planet).
        setup("Mercury", 52.2*Gm,2,fullCircle/(88*day),
           namedColor("gray")));
        this.sun.createRelation("planet", this.venus = this.
           spawn(Planet).
        setup("Venus", 107*Gm,5,fullCircle/(225*day),namedColor(
           "cyan")));
        this.sun.createRelation("planet", this.venus = this.
           spawn(Planet).
        setup("Earth", AU,5,fullCircle/(365*day),namedColor("
           blue")));
        this.sun.createRelation("planet", this.mars = this.spawn
           (Planet).
        setup("Mars", 208*Gm,3,fullCircle/(687*day),namedColor("
           red")));
    }

    step(delta:UnitSystem){
        this.sun.starStep(delta);
    }
}
```

Calling the environment and setting it as a global environment

```
space = new Space().setupEnv(
    {w:systemSize,h:systemSize,"delta":day})
setEnv(space);
setScale(2)
```

The graphical output of this code will generate a realistic representation of the orbits of the inner solar system around the Sun.

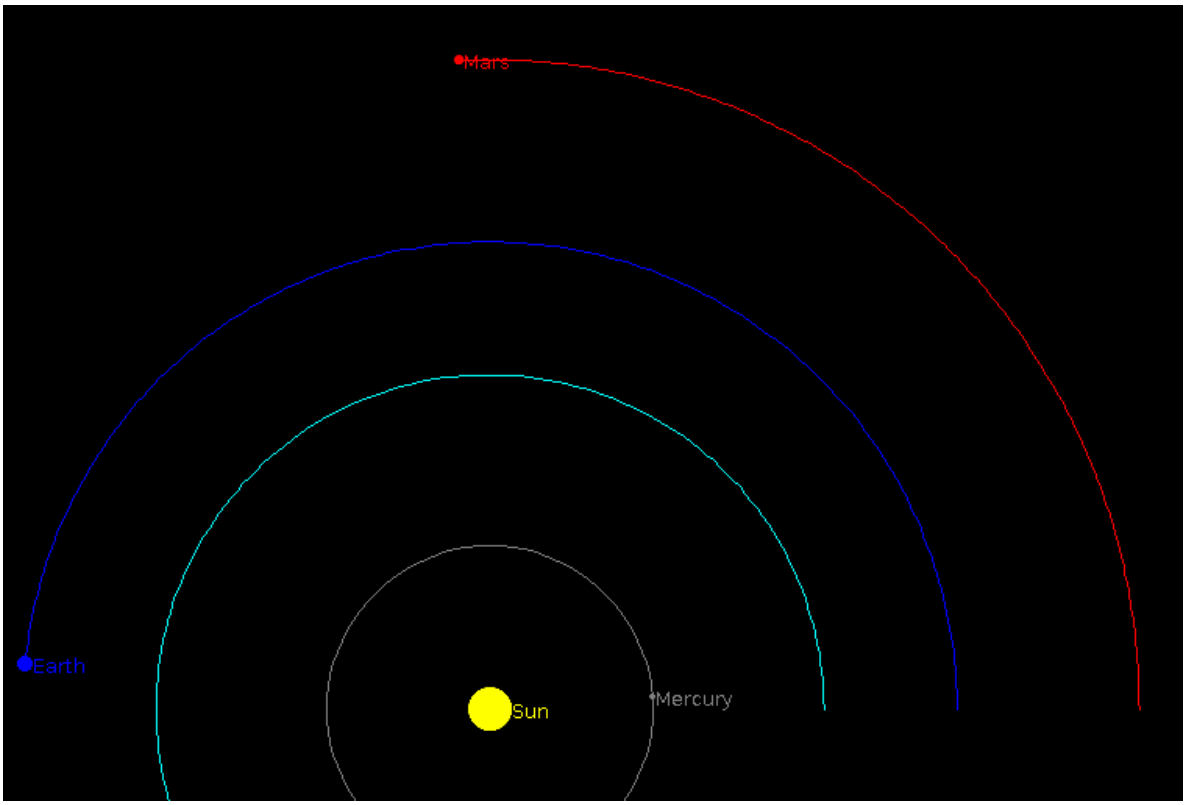

Figure 8.3: Image shows the view-port generated by AB-X with realistic planet orbits

Creating a simulation video using standard 24 Frames per Second (FPS) cinematic rate

```
include("{script}/model.js");
const env = new ModelEnv();
env.setupEnv(
    {w: 80, h: 40, gridType:"torus",agents:10});
writeImg("{script}/Result.png",env.getImg(10,["baseline"]));
startVideoRecording("{script}/Result.mp4","24");
for (let i = 0; i < 240;++i) {
    env.envStep();
    appendImg(env.getImg(10, ["baseline"]));
}
closeVideoStream();
```

Code for handling a histogram chart with 25 slots

```
class HistogramExampleEnv extends ABMEnv {
    setup(specs: Object) {
        this.addHistogram("Normal 100 samples per step", ()=>{
            const values = [];
            for (let i = 1; i <= 100;++i){
                values.push( randomNormal(100,50));
            }
            return values;
        },"addAll").setSlots(25).setXAxis("Normal Curve").
           setLimits(-100,300);
    }
    step(){
        this.sampleHistograms();
        this.setPaused(true);
    }
}
setEnv(new HistogramExampleEnv().setupEnv({}));
```

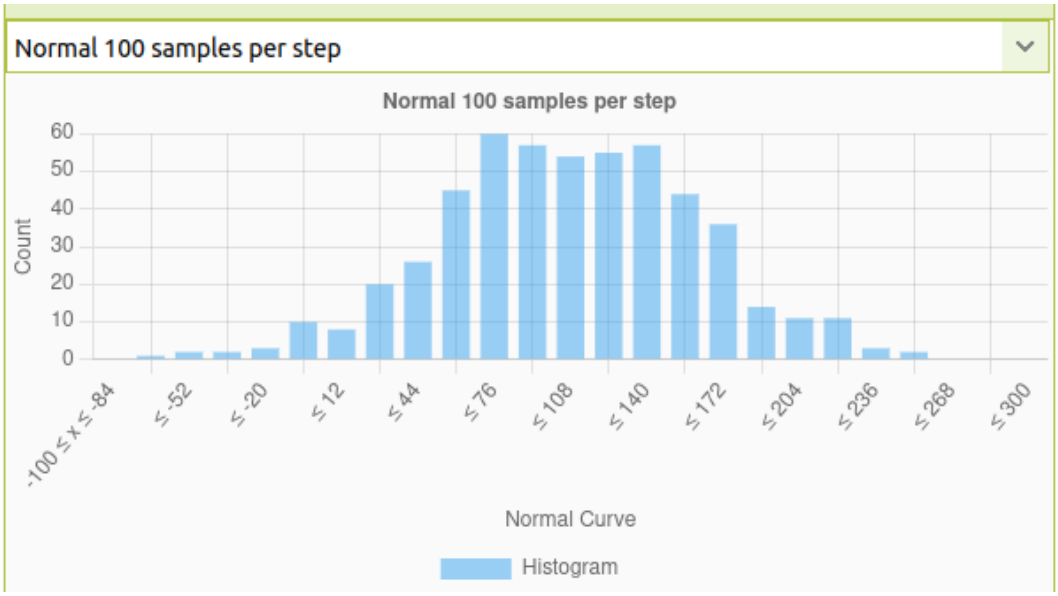

Figure 8.4: Histogram rendered after several steps

The following section of code shows how to add custom lines to any graph, in order to identify boundary or other important values.

```
setup(specs: Object):void {  ...
    this.addChart("Always 100", { "randData":(): Array=> {
            const values = [];
            for (let i = -lim; i <= lim; ++i) {
                values.push(-i+random());
            }
            return values;
        }
    },"replace", ():Array=>{
        const values = [];
        for (let i = -lim; i <= lim; ++i) {
            values.push(i);
        }
        return values;
    }).setXAxis("Rand Values").
        setXLimits(-lim-1,lim+1,10).
        setYLimits(-lim-1,lim+1,10).
        addLine([0.5,0,0.5,1],namedColor("black")).
        getSeries("randData").setColor(namedColor("green"))
}
```

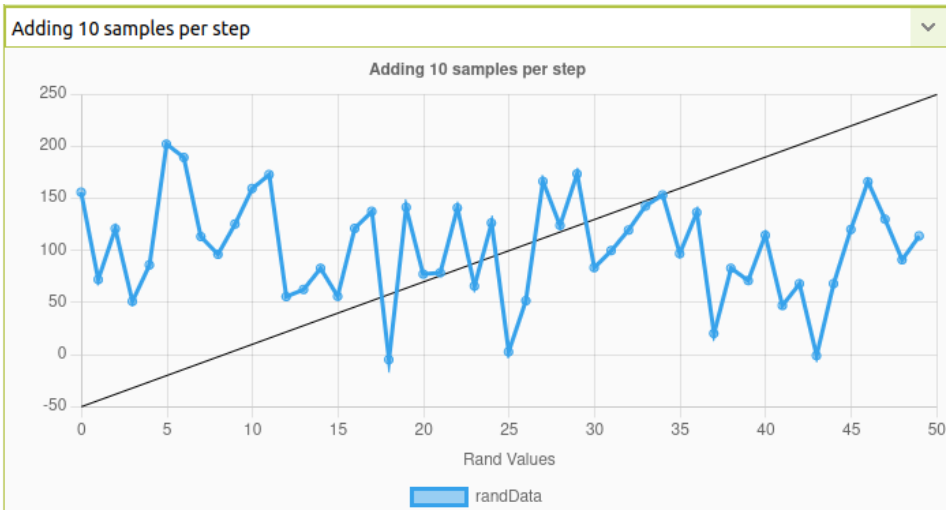

Figure 8.5: This figure shows a custom chart with a line decoration added programatically

There are many more illustrative examples which come with AB-X. The snippets shown here were included to describe both the power and simplicity of the syntax.

8.1.4 UI screenshots

The following image shows an editing page with the default work folder pointing to a folder containing all the scripts.

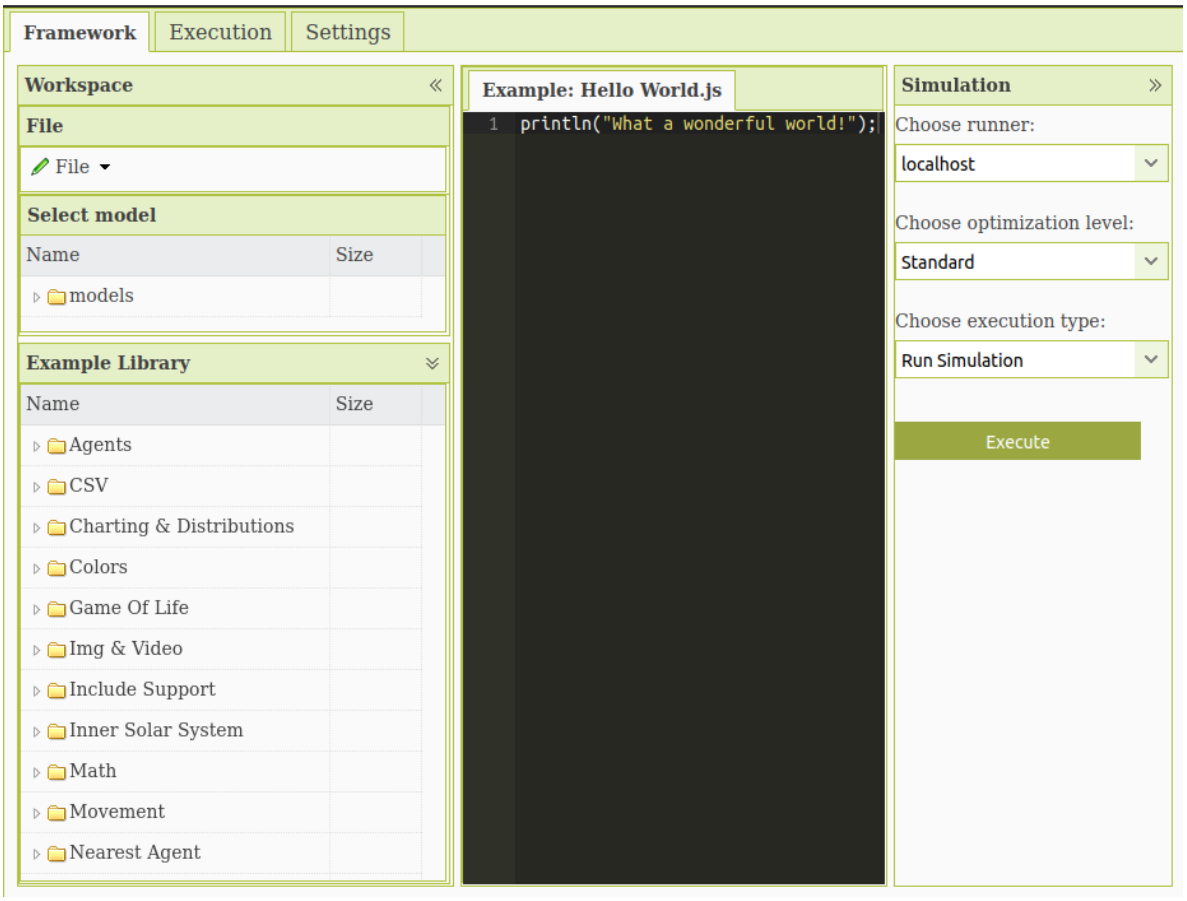

Figure 8.6: Figure showing the editing screen

The following image shows the execution of the PredPrey experiment in the execution screen. On the bottom left there is a text box which allows the user to input a query and JavaScript and get the Text response.

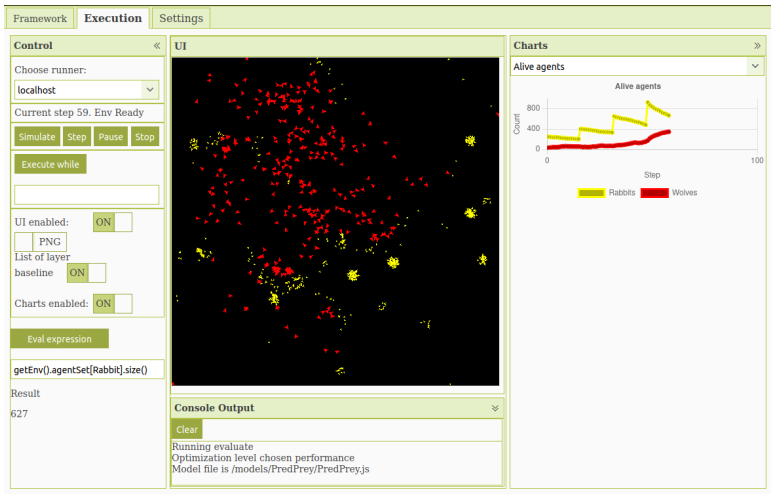

Figure 8.7: Execution screen showing simulation running

The following image shows the results of some unit test, with statistics of successes and failures.

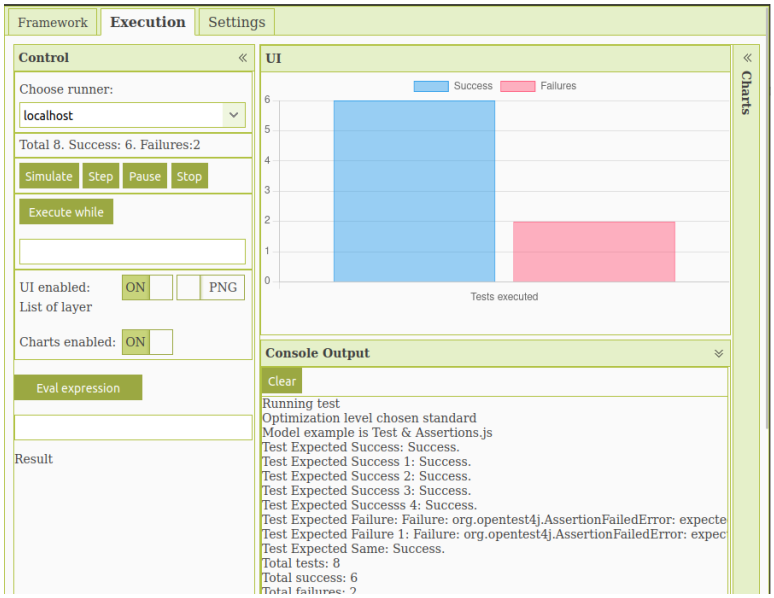

Figure 8.8: Execution screen showing unit test results

8.2 AB-X Results

This subsection describes the implementation of the four experiments requested for the scoring of the FSMM: Sugarscape, Pred-Prey, Pandemic and Pareto. Only the graphs requested for the experiment were included in this document. Additional graphs where used during development and debugging of each simulation – but are not shown here.

For the Pareto experiment, the macroscopic statistical analysis was performed, this time using the facilities provided in the AB-X implementation.

8.2.1 Sugarspace in AB-X

Sugarscape, as defined in the experiment, is implemented in AB-X in this URL `https://gitlab.com/msc_tesis/abx-fsmm/-/tree/main/Sugarscape`.

The implementation used four source files and an extra for video recording. It was straightforward due to the **Basic framework functionality** being already provided by the AB-X framework, with a basic agent called SgAgent, with two subclasses – for males and females. There was heavy use of the provided **Locality** and **Agent management** functionality. The **Visualization** of the agents was also available since the beginning, with the ability to disable it to speed up simulation.

Statistical Support including average agent energy, where added easily, including live count and average patch energy, which are shown in the final report.

The UI was smooth except when agent count exceeded 10000. Yet, even then it took about two seconds to complete a step. The **Usability** of the tool provided a good experience, as the framework responded quickly, it facilitated debugging using the *standard* option during development, and it ran fast with the *performance* option.

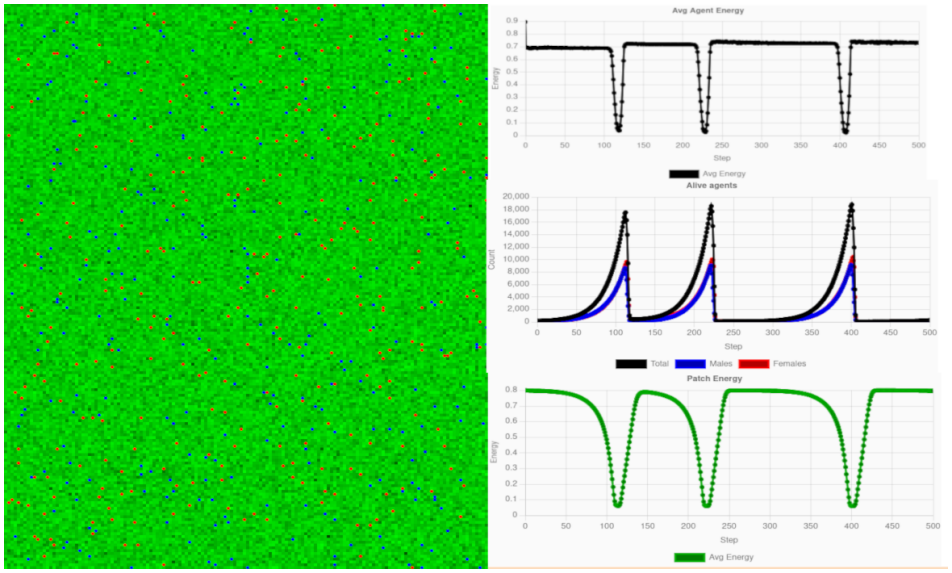

Figure 8.9: Sugarscape over AB-X

8.2.2 PredPrey in AB-X

PredPrey, as defined in the experiment, is implemented in AB-X in this URL `https://gitlab.com/msc_tesis/abx-fsmm/-/tree/main/PredPrey`.

The implementation required four files: one for rabbits, one for wolves, one for the environment and UIm and one for the simulation specs. **Force fields** logic was readily implemented using provided **Locality** and complex number support.

The implementation was straightforward, with only two types of agents using the **Agent management** functionality provided. Only two **Model statistics** graphs were added, the alive count and the percentage of rabbits vs wolves.

Visualization capabilities were used, using the delta shape for showing the direction of agents . **Basic framework functionality** was good, including the method to get agents in radius by class and nearest agent functionality. **Usability** was particularly good, as the framework responded quickly, and it was easy to debug using the *standard* option and it ran fast with the *performance* option.

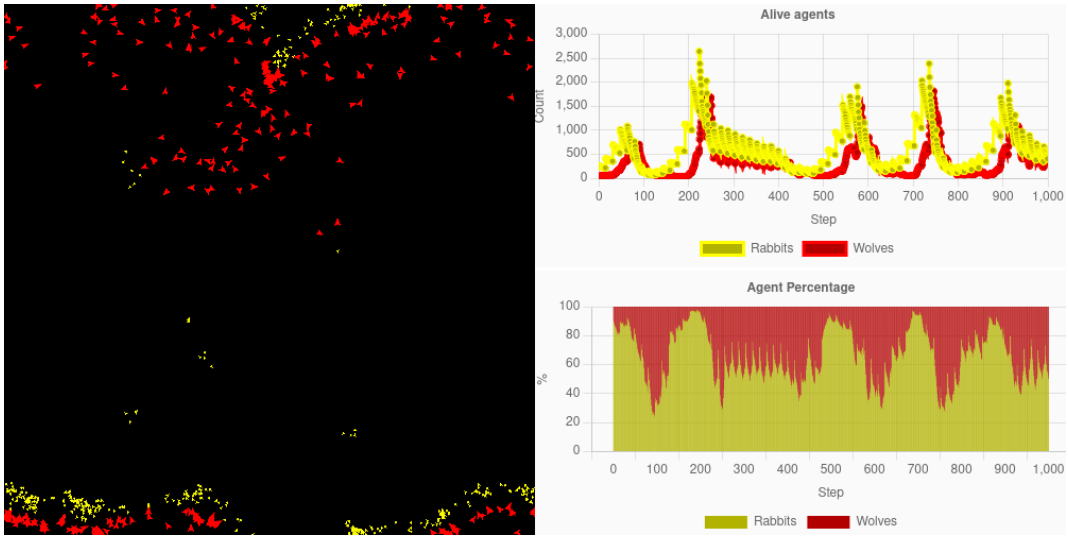

Figure 8.10: PredPrey over AB-X

8.2.3 Pandemics in AB-X

Pandemics, as defined in the experiment, is implemented in AB-X in this URL `https://gitlab.com/msc_tesis/abx-fsmm/-/tree/main/Pandemics`.

The implementation required four experiment files, two for logic, and one for each influence power level. It was straightforward due to the **Basic framework functionality**, including its http capabilities. The **Visualization** of the agents was also available since the beginning, being completely fluid even for 10000 agents.

There wasn't any technical challenge. In general, the **Usability** of the tool was very satisfactory. We used the provided **Agent management** capability of relations to handle the mailbox trust approach.

The first chart shows the two rounds of the simulation on AB-X with 2% and 3% of agent trust provided by influencers.

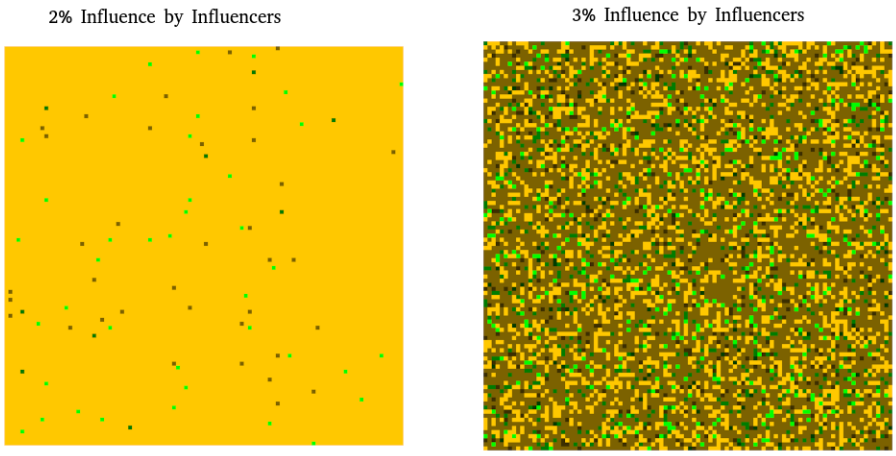

Figure 8.11: Pandemics over AB-X

Similar to NetLogo, we are interested in reviewing **Model statistics**, including sick ratio and trust level differences among the two rounds. In the first round, influencers provide a 2% agent trust, whereas in the second round, they only offer an additional percent point of trust. Interestingly, despite this apparently small difference, in the second round agents do not discover that the amulet increases the risk of getting sick.

The following result charts depict data generated from a plot utilizing HTTP-exported data from AB-X.

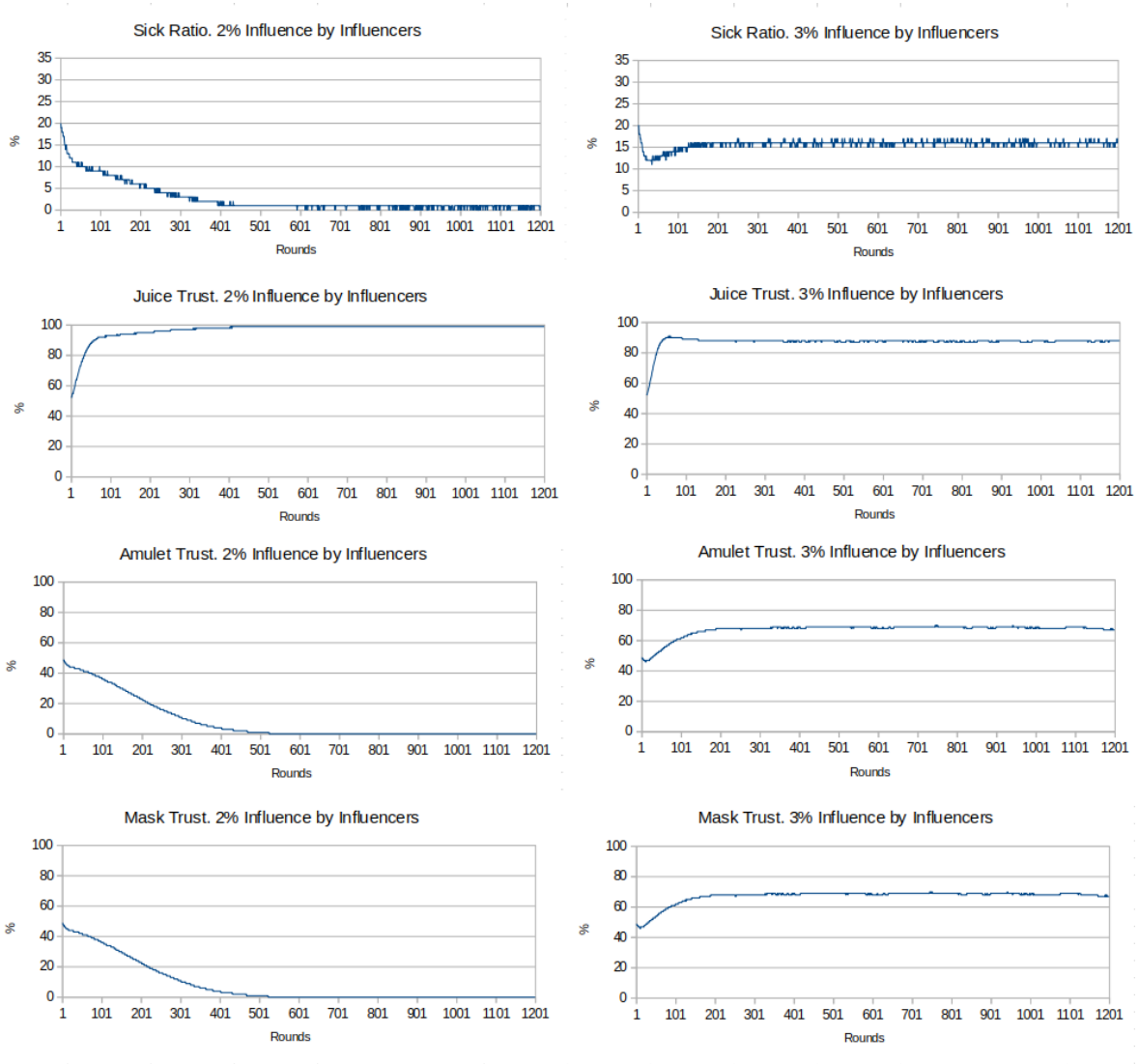

Figure 8.12: Pandemics over AB-X using reported data

8.2.4 Pareto in AB-X

Pareto, as defined in the experiment, is implemented in AB-X in this URL `https://gitlab.com/msc_tesis/abx-fsmm/-/tree/main/Pareto`.

The implementation required four JavaScript files which contains both the UI and behavior specs. There was a testing file added. No external files were required as AB-X provided the **Basic framework functionality**.

The implementation was straightforward with only one type of agent called Player. **Visualization** was very fast, and for the **Model statistic** only the cumulative curve in decreasing order (the Pareto curve) for the owned tiles and coins was added. It was simple to add guide lines in AB-X.

A CSV data file was generated for the **Macroscopic statistical analysis** using standard AB-X file system support. Each line contains the three macroscopic observables defined in the specification. The simulation was ran 10000 times. Because of AB-X's OO paradigm, it was natural to create and execute the simulation objects independently. If desired, it is straightforward to modify each simulation execution independently. For this experiment, AB-X's **Usability** was good, especially its ability to create tests.

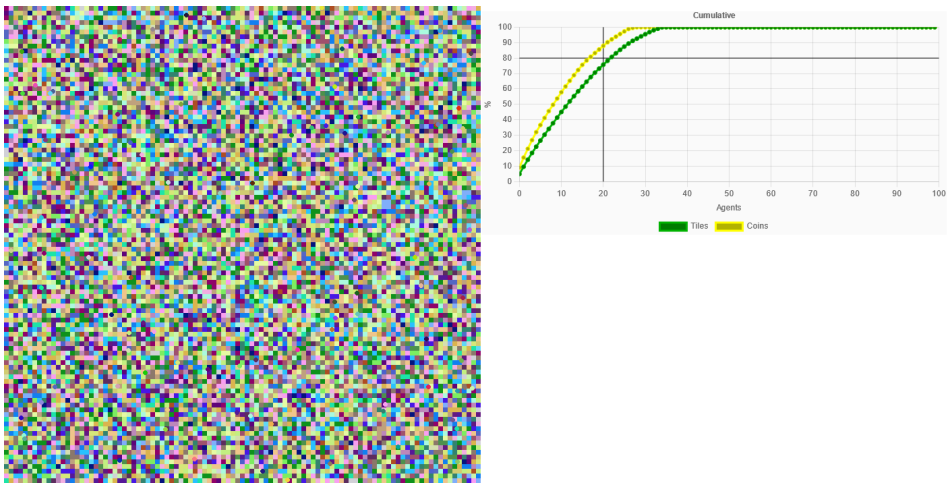

Figure 8.13: Pareto on AB-X

8.3 Macroscopic Statistical Analysis & Specification Fidelity

8.3.1 Histogram

The histogram depicted on Figure 8.14 shows that the macroscopic comparables of AB-X versus all other versions behave statistically identical, which supports the goal of all implementations following the specification.

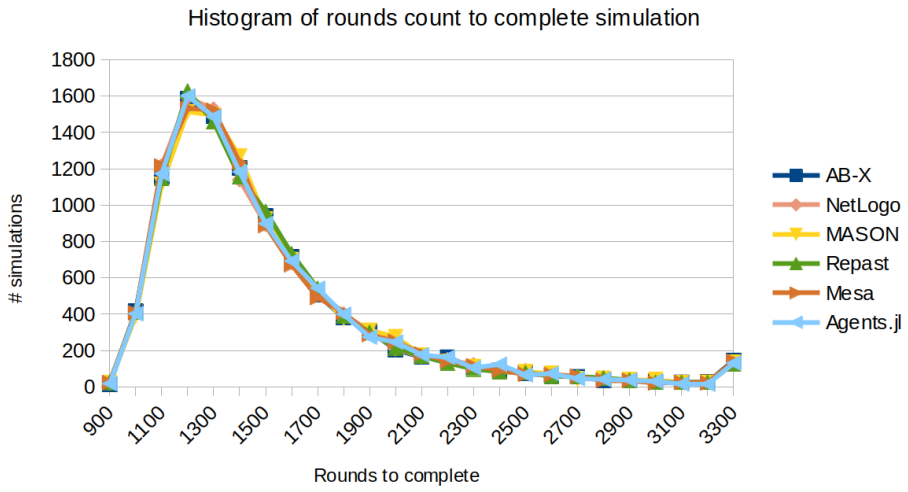

Figure 8.14: Histogram for rounds including AB-X

Similar to other frameworks, AB-X exhbits a tail after 2500 rounds. This is because each tile needs to be purchased to complete the simulation, but agents can only reach a tile by chance, which could create iterations with more than 5000 steps.

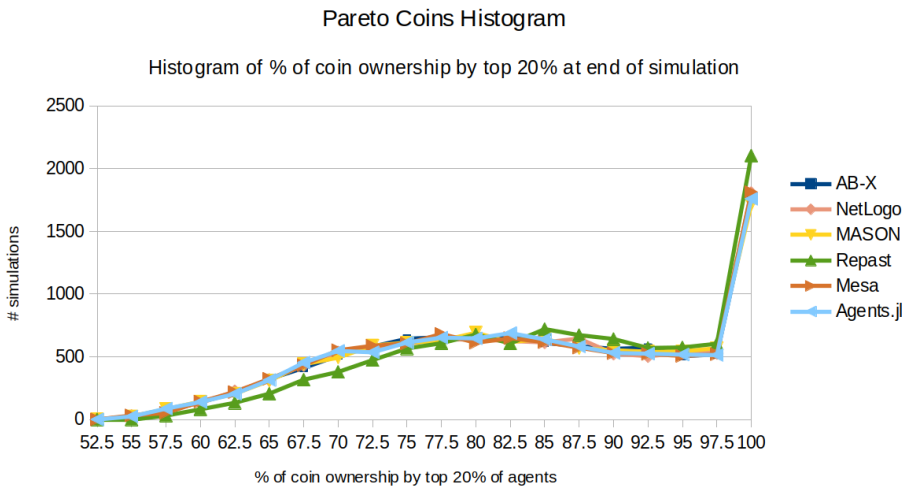

Figure 8.15: Histogram for coins including AB-X

As expected, in AB-X implementation coins ownership are clearly around 80% and it also has a hump where the 20% or less of the agents owned the 100% of the coins. This hump was also observed in the previous implementations.

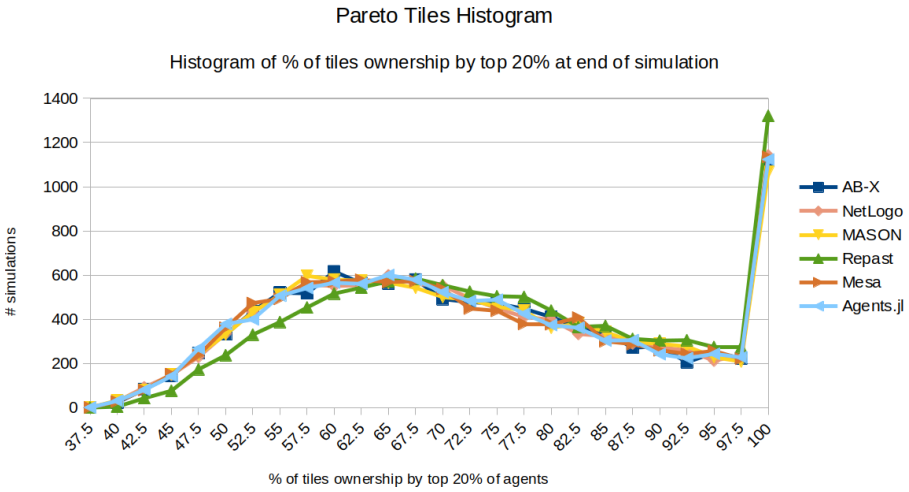

Figure 8.16: Histogram for tiles including AB-X

Similar to previous implementations, tiles ownership are clearly around 70-80%, and again, there is the hump where the 20% or less of the agents owned the 100% of the tiles.

8.3.2 Coefficient of Variation

For the six frameworks, the calculated CV for the three histograms (round count to completion, top 20% ownership of coins, top 20% of tiles ownership) is shown in the following table.

Framework	Rounds to finish	Pareto Coins	Pareto Tiles
AB-X	0.010	0.016	0.027
NetLogo	0.011	0.016	0.027
MASON	0.010	0.016	0.027
Repast	0.011	0.015	0.025
Mesa	0.011	0.016	0.028
Agents.jl	0.010	0.016	0.028

Table 8.1: Coefficient of variation, including AB-X results, for three chosen macroscopically observables

As CV is a unit-less measure, it is particularly useful for comparing variability across datasets with different units or scales. The example shows nearly identical values for almost all implementations, including AB-X – with a small difference in Repast, specially in the CV of the Pareto Tiles.

8.3.3 Fréchet Distance

The discrete Fréchet distance is calculated using a formula explained in Chapter 6. The Fréchet distances between axis/frameworks of the three histograms (round count to completion, top 20% ownership of coins, top 20% of tiles ownership) were calculated. While there are techniques to approach Fréchet distance efficiently [25], the general formula was used, because of the small number of points in the curves.

Fréchet distance for rounds follows. Largest values are marked with *.

	AB-X	NetLogo	MASON	Repast	Mesa	Agents.jl
AB-X	0.0	43.0	49.0	73.0*	65.0	60.0
NetLogo	43.0	0.0	68.0	87.0*	44.0	68.0
MASON	49.0	68.0	0.0	114.0*	72.0	95.0
Repast	73.0*	87.0*	114.0*	0.0	130*.0	102.0*
Mesa	65.0	44.0	72.0	130.0*	0.0	97.0
Agents.jl	60.0	68.0	95.0	102.0*	97.0	0.0

Table 8.2: Fréchet distance for histograms of rounds to complete experiment

Fréchet distance for histogram of top 20% coin ownership follows. Largest values are marked with *.

	AB-X	NetLogo	MASON	Repast	Mesa	Agents.jl
AB-X	0.0	39.0	50.0	338.0*	47.0	42.0
NetLogo	39.0	0.0	89.0	299.0*	38.3	46.0
MASON	50.0	89.0	0.0	388.0*	97.0	43.0
Repast	338.0*	299.0	388.0*	0.0	291.0*	345.0*
Mesa	47.0	38.32	97.0	291.0*	0.0	54.0
Agents.jl	42.0	46.0	43.0	345.0*	54.0	0.0

Table 8.3: Fréchet distance for histograms of top 20% coins ownership

Fréchet distance for histogram of top 20% tile ownership follows. Largest values are marked with *.

	AB-X	NetLogo	MASON	Repast	Mesa	Agents.jl
AB-X	0.0	30.0	54.0	201.0	38.08	44.2
NetLogo	30.0	0.0	75.0	180.0*	33.1	41.6
MASON	54.0	75.0	0.0	255.0*	69.0	57.0
Repast	201.0*	180.0*	255.0*	0.0	186.0*	198.0*
Mesa	38.1	33.1	69.0	186.0*	0.0	31.0
Agents.jl	44.2	41.6	57.0	198.0*	31.0	0.0

Table 8.4: Fréchet distance for histograms of top 20% tiles onwership

The values clearly show that AB-X results are similar to the previous implementations, excepting the unidentified differences in Repast's implementation.

8.3.4 ANOVA and F-Test

As mentioned in chapter 6, both ANOVA and F-test provide a p-value [45]. In null-hypothesis significance testing, where the null hypothesis is $H_0 : \mu_1 = \mu_2$, the p-value is the probability of obtaining test results at least as extreme as the result actually observed, under the assumption that the null hypothesis is correct [14]. Here, *extreme* means further away from what would be expected under the null hypothesis.

The ANOVA was applied to all the values in the histograms now also including the AB-X implementation (round count to completion, top 20% ownership of coins, top 20% of tiles ownership) and the F-Test was applied the each pair of histograms.

The usual notation of * is followed:

* $0.01 < p < 0.05$

** $0.001 < p < 0.01$

*** $p < 0.001$

The results are shown in tables below.

Rounds to completion

The p-value for the ANOVA was 0.273, that is more than 0.05, which fails to reject the null hypothesis and means there are not significant differences. Nevertheless, for F-Test between frameworks the results were below 0.05.

	AB-X	NetLogo	MASON	Repast	Mesa	Agents.jl
AB-X	-	0.997	0.268	0.258	0.892	0.353
NetLogo	0.997	-	0.291	0.275	0.893	0.376
MASON	0.268	0.291	-	0.027*	0.224	0.852
Repast	0.258	0.275	0.027*	-	0.331	0.041*
Mesa	0.892	0.893	0.224	0.331	-	0.297
Agents.jl	0.353	0.376	0.852	0.041*	0.297	-

Table 8.5: P-value for F-test for histograms of rounds to complete experiment

Here it can be seen that for rounds, the results of Repast are statistically significantly different for Agents.jl and MASON.

Top 20% of coin ownership

The p-value for the ANOVA was $< 10^{-5}$ which rejects null hypothesis, and indicates the means for the histograms of the top 20% of coin ownership, are significantly different. The F-Tests between frameworks the results are

	AB-X	NetLogo	MASON	Repast	Mesa	Agents.jl
AB-X	-	0.870	0.733	$< 10^{-5}$***	0.552	0.363
NetLogo	0.870	-	0.860	$< 10^{-5}$***	0.669	0.459
MASON	0.733	0.860	-	$< 10^{-5}$***	0.800	0.571
Repast	$< 10^{-5}$***	$< 10^{-5}$***	$< 10^{-5}$***	-	$< 10^{-5}$***	$< 10^{-5}$***
Mesa	0.552	0.669	0.800	$< 10^{-5}$***	-	0.755
Agents.jl	0.363	0.459	0.571	$< 10^{-5}$***	0.755	-

Table 8.6: P-value for F-test for histograms of top 20% of coin ownership

Here it can be seen that for the top 20% of coin ownership, the results of Repast are statistically significantly different for all the other frameworks.

Top 20% of tiles ownership

The p-value for the ANOVA was$< 10^{-5}$ which rejects null hypothesis, and indicates the means of the histograms for the op 20% of tile ownership, are significantly different. The F-Tests between frameworks the results are

	AB-X	NetLogo	MASON	Repast	Mesa	Agents.jl
AB-X	-	0.777	0.603	$< 10^{-5}$***	0.687	0.560
NetLogo	0.777	-	0.423	$< 10^{-5}$***	0.494	0.388
MASON	0.603	0.423	-	$< 10^{-5}$***	0.909	0.949
Repast	$< 10^{-5}$***	$< 10^{-5}$***	$< 10^{-5}$***	-	$< 10^{-5}$***	$< 10^{-5}$***
Mesa	0.687	0.494	0.909	$< 10^{-5}$***	-	0.859
Agents.jl	0.560	0.388	0.949	$< 10^{-5}$***	0.859	-

Table 8.7: P-value for F-test for histograms of top 20% of tiles ownership

Here it can be seen that for the top 20% of tiles ownership, the results of Repast are statistically significantly different for all the other frameworks.

These results keeps indicating a consistency issue in Repast which could be caused by a underlying issue related to pseudo-random generation or in the experiment setup and behavior which has not being identified even after several code reviews.

8.4 AB-X FSMM Scoring

The following are the scoring for the two subsectionss of the FSMM which were evaluated for each of the five frameworks and AB-X.

8.4.1 Simulation Features

The score for the simulation features of the FSMM for the frameworks based on the four experiments are shown in the following table.

Category	NetLogo	MASON	Repast	Mesa	Agents.jl	AB-X
Usability	3	3	1	2	1	3
Basic Framework Functionalities	3	3	3	3	3	3
Locality	3	2	3	2	3	3
Agent Management	2	2	2	2	1	2
Force Fields	0	0	0	0	1	2
Summary	11	10	9	9	9	13

Table 8.8: Simulation Features Score including AB-X

8.4.2 Interrogation Capabilities

The score for the interrogation features of the FSMM for the frameworks based on the four experiments are shown in the following table.

Category	NetLogo	MASON	Repast	Mesa	Agents.jl	AB-X
Visualization	3	3	2	2	1	3
Statistical support for model attributes	3	3	3	3	3	3
Macroscopic statistical analysis	2	2	2	2	2	3
Nearly decomposable System view	0	0	0	0	0	1
Summary	8	8	7	7	6	10

Table 8.9: Interrogation capabilities Score including AB-X

Based on those scores, we can create a quadrant-like graph with the top right being the best scored and the bottom left being the worst scored. In this quadrant we see that AB-X performed better in the FSMM. It is worth noting that the FSMM scoring did not include research supporting features due to the almost universal lack of support in current frameworks. Yet, some of those features (such as utility features) were

purposefully added in AB-X. Had those features been included in the scoring, it is expected AB-X should have scored higher.

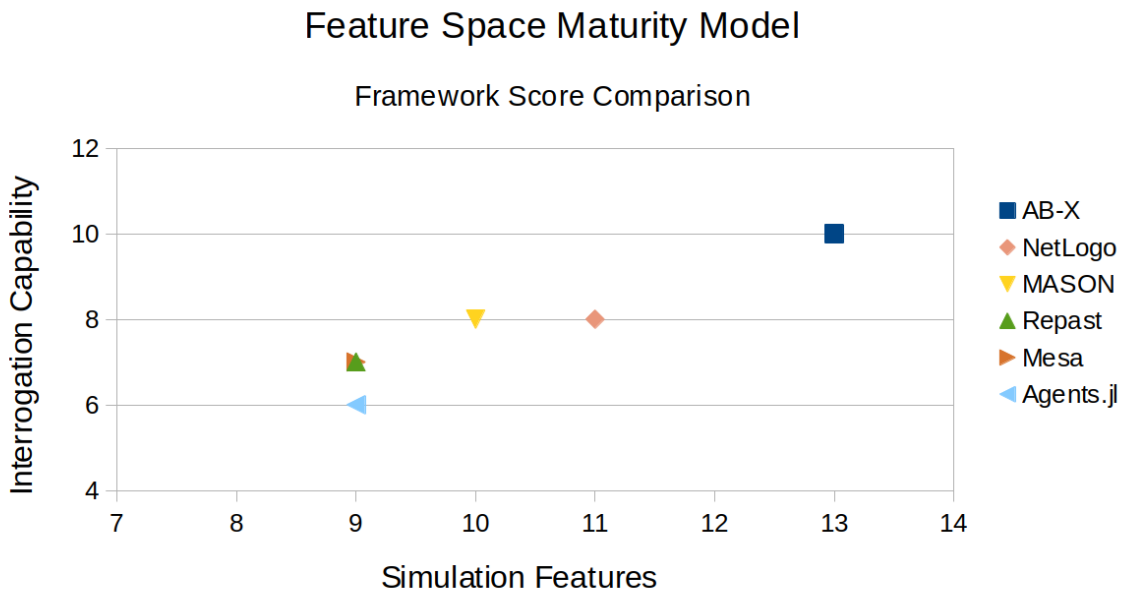

Figure 8.17: FSMM including AB-X

8.5 AB-X Future features

The following list present a list of improvements (in decreasing order of importance) which are required to transform the AB-X POC into a full fledged product:

- Finalize Vertex-Edge support: Vertex-Edge support was not in the featured experiments for scoring the FSMM. One of the most important reasons was that none of the ABM framework studied seems to support it directly and therefore requesting it would not make much sense. Nevertheless, the idea of taking agents outside of the square grid and putting them on the edges would facilitate simulating several behaviors without the need for a full 3D framework, as elsewhere in this document. One of the main forward-looking improvements for AB-X would be to fully implement the Vertex-Edge support as described in the architecture chapter.

- Two important missing services are the Containers and the Executions manager. Those needs are handled by the default service which limits the execution to the local computer. Once completed, those services will allow AB-X to be used in a distributed environment like a University or research center.

- Repository support: Another missing service is the repository service which should provide the ability to select an arbitrary location in the local disk, or to save resources in a folder backed by a versioning system. Right now, files are saved in a folder relative to the product location. While this can be used for work, most of the enterprise-ready products are expected to have this support implemented by default.

- Optimizations: While parts of the grid were developed in Java to avoid the cost to have them in pure JavaScript, there are still many improvements which can be done, specially in the neighbor algorithm which are very slow when compared to NetLogo's.

- Editor capabilities: Currently, AB-X handles only JavaScript syntax, nevertheless the underlying ACE editor could handle other text files. Also the UI is expected to handle other file types, like showing images, videos or the ability to download binary files.

- Editor customization: While AB-X UI is green with a black editor, its underlying libraries jEasyUI and ACE support varied color schemes: this may be quickly added to the UI.

- Headless charting support: Charts are generated in the UI by the JS libraries which allows interactive use by the developers but sometimes it might be desirable to have charts generated without the need of the UI.

CHAPTER 9

Conclusions

'The last will be first, and the first last."

Jesus of Nazareth, Matthew 20:16

Reflecting on the work developed in this project, we can split the conclusions into two realms: the technical one and the simulation one. The technical conclusions will comprise the proposed FSMM and the proposed architecture.

Regarding the FSMM we can conclude:

- While use of ABM has slowed somewhat in recent years, this research technique has important advantages over other top-down simulation techniques like EBM, DES or SD, due to is superior visibility and its ability to incorporate contrasting behavior.
- To score an ABM Framework, performance alone is not enough and a more comprehensive approach should be considered.
- We proposed our FSMM as an initial approach to score ABM frameworks. We knew that some topics in the FSMM would overlap, as they are interrelated concepts. An effort was made to minimize overlapping.
- After performing the experiments we reckon that the three most important topics in the FSMM were usability, basic framework functionalities, and visualization.
- While visibility and the other six topics can be implemented, usability and basic framework functionalities are foundational capabilities and need to be provided by the framework themselves.
- The proposal for our FSMM should be able to cover the spectrum from a high level computer language to a highly robust, fully-fledged ABM framework.
- There were important weaknesses in usability and visualization in the Repast, Mesa and Agents.jl frameworks, which is of concern because they are commonly used ABM frameworks.
- Only NetLogo and MASON scored highest in the top 3 items of the FSMM, which points to their current popularity.

- We proposed a statistical method to validate the functional equivalence of two or more implementation of the same experiment specification. Using it, we verified that for one of the proposed validation experiments (the Pareto experiment) five of our six implementations are statistically equivalent.

- Using statistical tools, we observed that the Repast implementation did not behave as the other implementations. Even after several hours of debugging we couldn't figure out the reason for its slight but statistically confirmed difference in behavior.

- Many architectural approaches might not be able to reach facilitator level 3 in all topics of the FSMM.

- The FSMM could be used as a starting research point for the development of new ABM Frameworks because it could help identify desired features and avoid overlooking required ones.

- An architecture approach may be chosen so as to simplify its implementation.

- While not commonly used today, proposed research supporting features might become highly relevant in the future of ABM frameworks as they furnish ways to define complex behavior in a powerful declarative way.

On the architectural approach for the FSMM we can conclude:

- The reference architecture was designed to provide insights on some of the most common challenges in ABM simulation and to cover most of the current needs regarding ABMs.

- The core principles (stability, maintainability, readability, syntax expressiveness, visualization and service oriented) laid out in this document should allow for an iterative ABM framework implementation in reasonable periods of time by a small team (the POC was developed in four months by one developer).

- The proposed reference architecture provides a high-level reference for future ABM framework architecture or implementations. It defines the system structure, its principles, and should help guiding the development process, ensuring that key architectural decisions are consistent, easy to understand and maintain.

- The proposed reference architecture was designed to open a door to future enhancement for ABM Frameworks. One example is the concept of allowing agents off the grid into a vertex. This could open the door for realistic, complex non- adjacent topologies for location-aware simulations.

- AB-X is a POC of the ABM reference architecture. While incomplete, it fared better (obtained higher marks) than other evaluated ABM frameworks, which strongly corroborates the main tenet of our research: that it is possible to build an architecture with robust computational properties which supports an ABM feature space that helps develop simulations whose correctness can be ascertained while achieving high fidelity to reality.

For ABM in general we can conclude:

- ABM Frameworks are powerful tools for simulating and analyzing complex systems composed of interacting agents. Their main advantage resides in the relative ease of applying their ability to reproduce emergent behavior observed in or expected from real systems.

- ABM Frameworks allow to shrink the semantic gap between their specification and their implementation. One way to close this gap is by providing a set of essential features, leading to correctness and fidelity to reality, as well as other benefits to the research community, including the ability for no-programmers to develop or validate their simulations.

- ABM simulation is a valid research tool in multiple science fields such as Economics, Sociology, Epidemiology, and Environmental Science, which allow capturing complex

interactions, exploring multiple scenarios with flexibility, develop counterfactual outcomes – therefore it is valuable to promote and democratize them.

CHAPTER 10

Future work

‘Men at some time are masters of their fates. The fault, dear Brutus, is not in our stars, But in ourselves, that we are underlings.”

William Shakespeare, Julius Caesar. Act 1

While performing our research we identified some important themes for future work on the FSMM, including:

- The approach for the FSMM is a simple score from 0 to 3 depending on the facilitator level. While a powerful approach, an alternative is to research a multidimensional table using a Maturity Model based on the Focus Area [94]. This approach could give finer granularity in certain aspects and would be highly relevant in scoring the three identified foundational features.

- Each experiment provides a list of features it is intended to score. Nevertheless, it should be possible to improve the experiment documentation to make them more robust for scoring the FSMM. For example, each experiment description might provide a table or rubric detailing a minimum set of expected features to achieve a certain score.

- Using macroscopic analysis of statistical observables helps to identify differences between multiple implementations of a simulation. For this work, four experiments were implemented in five different frameworks, though only the Pareto experiment was subject to the process of collecting observables for statistical analysis. It is desirable to apply similar macroscopic analyses to identify deviation in the remaining experiments and, if found, correct them.

- It is interesting to continue research to identify why out of the 5 implementations of the Pareto experiment only Repast produced a statistically different results.

- Each of the experiments, even small, produced interesting results. One of the most striking behaviors observed was in the Pandemics experiment. There, the change in one percent point in the influencers increased the trust in a counter-productive measure (wearing the amulet). It will be interesting to compare this experiment with similar problems.

- While not scored in this document, the defined research supporting features might be highly important in the future as they close the semantic gap between specification and implementation. A future version of the FSMM scoring should include those features in some way; this may require adding new experiments or modifying some of the provided experiments to explicitly score them according to the evolved FSMM.

About the reference architecture and its POC implementation it is worth to consider:

- There are several topics related to the POC in the previous AB-X Future features section, one them is the limited support of directed edges. It might be useful to complete the feature, based on the reference architecture.

- While not split in the reference architecture it becomes apparent in the development of the POC than the environment and the grid are separate entity types which could come apart in a refinement of the reference architecture.

- One interesting idea is for agents to jump off the grid into a vertex. This could be further developed by allowing a generalization of the topology into a more complex one – which would be useful for space-aware simulations, while avoiding the cost of a full 3D environment.

- An alternate topology would be a homogeneous multilayered grid where some tiles of layers could be connected by edges. This topology would enable more natural simulation of buildings. Yet another topology would be a heterogeneous multilayered one which could be used to simulate a complex of buildings, a section of a city or even a full one, space stations, and battleground fields – among others.

References

[1] Abar, S., Theodoropoulos, G. K., Lemarinier, P., and O'Hare, G. M. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33.

[2] Ajax.org (2023). *ACE (Ajax.org Cloud9 Editor)*. `https://ace.c9.io/`.

[3] Antelmi, A., Cordasco, G., D'Ambrosio, G., De Vinco, D., and Spagnuolo, C. (2022). Experimenting with agent-based model simulation tools. *Applied Sciences*, 13(1):13.

[4] Apache-Software-Foundation (2000). Netbeans. https://netbeans.apache.org/front/main/.

[5] Balci, O. (1997). Verification validation and accreditation of simulation models. In *Proceedings of the 29th Conference on Winter Simulation*, WSC '97, page 135–141, USA. IEEE Computer Society.

[6] Bankes, S. C. (2002). Agent-based modeling: A revolution? *Proceedings of the National Academy of Sciences*, 99(suppl_3):7199–7200.

[7] Barclay, K. A. and Savage, W. J. (2007). Groovy programming : an introduction for java developers.

[8] Batool, A., Bashir, M., Babar, M., Sohail, A., and Ejaz, N. (2021). Effect or program constructs on code readability and predicting code readability using statistical modeling. *Foundations of Computing and Decision Sciences*, 46:127–145.

[9] Becker, J., Knackstedt, R., and Poeppelbuss, J. (2009). Developing maturity models for it management. *Business & Information Systems Engineering*, 1:213–222.

[10] Bellifemine, F. L., Caire, G., and Greenwood, D. (2008). *Developing Multi-Agent Systems with JADE*. John Wiley.

[11] Berger, E., Hollenbeck, C., Maj, P., Vitek, O., and Vitek, J. (2019). On the impact of programming languages on code quality. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 41:1 – 24.

[12] Bevington, P. and Robinson, D. (2003). *Data Reduction and Error Analysis for the Physical Sciences*. McGraw-Hill Education.

[13] Bezanson, J., Karpinski, S., Shah, V. B., and Edelman, A. (2012). Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*.

[14] Brenneman, J. (2020). Experimental design. *J. Qual. Technol.*, 52(4):423–424.

[15] Burnette, E. (2005). Eclipse ide pocket guide.

[16] Burrough, P. A., McDonnell, R. A., and Lloyd, C. D. (2015). *Principles of geographical information systems*. Oxford University Press, London, England, 3 edition.

[17] Cao, J., Feng, X., Lu, J., and Das, S. (2002). Mailbox-based scheme for mobile agent communications. *Computer*, 35(9):54–60.

[18] Catalin Balan, G. (2009). MASON: A Java Multi-Agent Simulation Library.

[19] Chart.js Team (2023). *Chart.js: Simple yet flexible JavaScript charting for designers & developers*. `https://www.chartjs.org/`.

[20] Chibbaro, S., Rondoni, L., and Vulpiani, A. (2022). *Probability, Typicality and Emergence in Statistical Mechanics*, pages 339–360. Springer International Publishing, Cham.

[21] Crooks, A. and Metcalf, S. (2021). Introduction to agent-based models.

[22] Datseris, G., Vahdati, A. R., and DuBois, T. C. (2022). Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity. *SIMULATION*, 0(0):003754972110688.

[23] Docker (2013). Docker: Accelerated, containerized application development. Available at https://www.docker.com/, Accessed on: 2024-06-27.

[24] Dowson, D. and Landau, B. (1982). The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455.

[25] Eiter, T. and Mannila, H. (1994). Computing discrete frechet distance.

[26] Epstein, J. M. (2012). *Generative social science: Studies in agent-based computational modeling*. Princeton University Press.

[27] Epstein, J. M. and Axtell, R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. Brookings Institution Press.

[28] Epstein, J. M. and Axtell, R. L. (2006). Artificial societies and generative social science. *Artificial Life and Robotics*, 1:33–34.

[29] Everitt, B. S. (1998). *Cambridge dictionary of statistics*. Cambridge University Press, Cambridge, England.

[30] Felleisen, M. (1991). On the expressive power of programming languages. *Science of Computer Programming*, 17(1):35–75.

[31] Ferguson, N. (2020). Report 9: Impact of non-pharmaceutical interventions (NPIs) to reduce COVID19 mortality and healthcare demand.

[32] Ferreira Caceres, M. M., Sosa, J. P., Lawrence, J. A., Sestacovschi, C., Tidd-Johnson, A., Rasool, M. H. U., Gadamidi, V. K., Ozair, S., Pandav, K., Cuevas-Lou, C., Parrish, M., Rodriguez, I., and Fernandez, J. P. (2022). The impact of misinformation on the COVID-19 pandemic. *AIMS Public Health*, 9(2):262–277.

[33] Feyerabend, P. K. (2010). *Against Method*. Verso Books, London, England, 4 edition.

[34] Foundation, T. D. (2020). Libreoffice calc.

[35] Gardner, M. (1970). Mathematical games: the fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223(October):120–123.

[36] Gini, C. (1921). Measurement of inequality of incomes. *The Economic Journal*, 31(121):124–126.

[37] Gosling, J., Joy, B., Steele, G. L., Bracha, G., and Buckley, A. (2014). *The Java Language Specification, Java SE 8 Edition*. Addison-Wesley Professional, 1st edition.

[38] Gupta, S. and Ray, A. (2009). Statistical Mechanics of Complex Systems for Pattern Identification. *Journal of Statistical Physics*, 134(2):337–364.

[39] Harari, Y. N. (2017). *Homo Deus: A Brief History of Tomorrow*. HarperCollins.

[40] Harsanyi, J. C. (1961). On the rationality postulates underlying the theory of cooperative games. *The Journal of Conflict Resolution*, 5(2):179–196.

[41] Hoel, E. P. (2017). When the map is better than the territory. *Entropy*, 19(5).

[42] Hoel, E. P., Albantakis, L., and Tononi, G. (2013). Quantifying causal emergence shows that macro can beat micro. *Proceedings of the National Academy of Sciences*, 110(49):19790–19795.

[43] Hood, P. M. (2004). *Aristotle on the Category of Relation*. University Press of America, Lanham, MD.

[44] Horn, R. L. V. (1971). Validation of simulation results. *Management Science*, 17(5):247–258.

[45] Howell, D. C. (2002). *Statistical methods for psychology*. Wadsworth Publishing Company.

[46] International, E. (2020). Ecmascript® 2025 language specification.

[47] Jetbrains (2000). Intellij. idea. https://www.jetbrains.com/help/idea/discover-intellij-idea.html.

[48] jQuery EasyUI Team (2023). *jQuery EasyUI*. `https://www.jeasyui.com/`.

[49] JuliaLang (2020). Julia extension for microsoft visual studio code. `https://marketplace.visualstudio.com/items?itemName=julialang.language-julia`. Accessed November 7, 2023.

[50] Kazil, J., Masad, D., and Crooks, A. (2020). Utilizing Python for agent-based modeling: The Mesa framework. In Thomson, R., Bisgin, H., Dancy, C., Hyder, A., and Hussain, M., editors, *Social, Cultural, and Behavioral Modeling*, pages 308–317, Cham. Springer International Publishing.

[51] Kleinberg, J. and Tardos, E. (2005). *Algorithm Design*. Pearson, Upper Saddle River, NJ.

[52] Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18.

[53] Lange, P., Weller, R., and Zachmann, G. (2016). Graphpool: A high performance data management for 3D simulations. *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*.

[54] Li Vigni, F. (2021). Regimes of evidence in complexity sciences. *Perspectives on Science*, 29(1):62–103.

[55] Lorig, F., Dammenhayn, N., Müller, D.-J., and Timm, I. (2015). Measuring and comparing scalability of agent-based simulation frameworks.

[56] Luke, S. (2019). *Multiagent simulation and the MASON library*. George Mason University (CC).

[57] Luke, S., Simon, R., Crooks, A., Wang, H., Wei, E., Freelan, D., Spagnuolo, C., Scarano, V., Cordasco, G., and Cioffi, C. (2019). The mason simulation toolkit: Past, present, and future.

[58] Macal, C. and North, M. (2007). Agent-based modeling and simulation: Desktop ABMS. *Proceedings - Winter Simulation Conference*, pages 95–106.

[59] Macal, C. M. (2016). Everything you need to know about agent-based modelling and simulation. *Journal of Simulation*, 10(2):144–156.

[60] Maclay, G. J. and Ahmad, M. (2021). An agent based force vector model of social influence that predicts strong polarization in a connected world. *PLoS One*, 16(11):e0259625.

[61] Marshall, B. D. L. and Galea, S. (2014). Formalizing the role of agent-based modeling in causal inference and epidemiology. *Am J Epidemiol*, 181(2):92–99.

[62] Masad, D. and Kazil, J. (2015). Mesa: An agent-based modeling framework.

[63] MDN Web Docs (2023). Proxy servers and tunneling. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Proxy_servers_and_tunneling`. Accessed: 2024-07-26.

[64] Microsoft Corporation (2015). Microsoft visual studio code. `https://code.visualstudio.com/`. Accessed November 7, 2023.

[65] Montañola-Sales, C., Rubio-Campillo, X., Cela-Espin, J.-M., Casanovas, J., and Kaplan-Marcusán, A. (2014). Overview on agent-based social modelling and the use of formal languages. *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*, pages 333–377.

[66] Montgomery, D. (2008). *Design and Analysis of Experiments*. John Wiley & Sons.

[67] Mozilla-Foundation (2003). Mozilla firefox. Accessed November 7, 2023.

[68] Niazi, M. (2011). Agent-based computing from multi-agent systems to agent-based models: A visual survey: Scientometrics: Vol 89, no 2.

[69] North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling*, 1(1):3.

[70] Núñez-Corrales, S., Friesen, M., Mudigonda, S., Venkatachalapathy, R., and Graham, J. (2021). In-silico models with greater fidelity to social processes: towards abm platforms with realistic concurrency. In *Proceedings of the 2020 Conference of The Computational Social Science Society of the Americas*, pages 155–169. Springer.

[71] Nygaard, K. and Dahl, O.-J. (1978). *The Development of the SIMULA Languages*, page 439–480. Association for Computing Machinery, New York, NY, USA.

[72] Núñez-Corrales, S. and Gasser, L. (2018). Scalable social simulation: Evaluation of current frameworks and a new approach. In *Proceedings of the 2018 International Conference on Social Computing, Behavioral-Cultural Modeling, & Prediction and Behavior Representation in Modeling and Simulation (SBP-BRiMS 2018)*.

[73] of Mathematics, E. (2024). Graph theory.

[74] Ozik, J., Collier, N. T., Murphy, J. T., and North, M. J. (2013). The relogo agent-based modeling language. In *2013 Winter Simulations Conference (WSC)*, pages 1560–1568. IEEE.

[75] Pareto, V. (1964). *Cours D'économie Politique*. Librairie Droz.

[76] Popper, K. (1972). *Objective Knowledge: An Evolutionary Approach*. Number p. 727 in Objective Knowledge: An Evolutionary Approach. Clarendon Press.

[77] Privitera, M. B. (2005). *What is easy of use?* Industrial Designers Society of America.

[78] Robertson, D. A. (2016). Agent-based models and behavioral operational research. *Behavioral Operational Research: Theory, Methodology and Practice*, pages 137–159.

[79] Roese, N. J. (1997). Counterfactual thinking. *Psychological Bulletin*, 121:133–148.

[80] Roza, M., Voogd, J., Jense, H., and Gool, P. (2011). Fidelity requirements specification: A process oriented view.

[81] Sabzian, H., Shafia, M. A., Bonyadi Naeini, A., Jandaghi, G., and Sheikh, M. J. (2018a). A review of agent-based modeling (abm) concepts and some of its main applications in management science. *Interdisciplinary Journal of Management Studies (Formerly known as Iranian Journal of Management Studies)*, 11(4):659–692.

[82] Sabzian, H., Shafia, M. A., Bonyadi Naeini, A., Jandaghi, G., and Sheikh, M. J. (2018b). A review of agent-based modeling (ABM) concepts and some of its main applications in management science. *Iranian Journal of Management Studies*, 11(4):659–692.

[83] Sargent, R. (2000). Verification, validation and accreditation of simulation models. In *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*, volume 1, pages 50–59 vol.1.

[84] Schelling, T. C. (1980). *Micromotives and macrobehavior*. Norton.

[85] Seagren, C. W. (2011). Examining social processes with agent-based models. *The Review of Austrian Economics*, 24:1–17.

[86] Simon, H. A. (1991). *The Architecture of Complexity*, pages 457–476. Springer US, Boston, MA.

[87] Sipek, M., Mihaljevic, B., and Radovan, A. (2019). Exploring aspects of polyglot high-performance virtual machine graalvm.

[88] Stevenson, J. C. (2021). Agentization of two population-driven models of mathematical biology. In *Conference of the Computational Social Science Society of the Americas*, pages 176–189. Springer.

[89] Thaler, J. (2020). *Investigating the use of pure functional programming for agent-based simulation*. PhD thesis, University of Nottingham, UK.

[90] Thaler, J., Altenkirch, T., and Siebers, P.-O. (2018). Pure functional epidemics: An agent-based approach. In *Proceedings of the 30th Symposium on Implementation and Application of Functional Languages*, IFL 2018, page 1–12, New York, NY, USA. Association for Computing Machinery.

[91] Tisue, S. (2004). Netlogo: Design and implementation of a multi-agent modeling environment.

[92] Todaro, M. P. (1992). Review of Human Development Report 1992 by United Nations Development Programme. *Population and Development Review*, 18(2):359–363.

[93] Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[94] van Steenbergen, M., Bos, R., Brinkkemper, S., van de Weerd, I., and Bekkers, W. (2010). The design of focus area maturity models. In Winter, R., Zhao, J. L., and Aier, S., editors, *Global Perspectives on Design Science Research*, pages 317–332, Berlin, Heidelberg. Springer Berlin Heidelberg.

[95] Varley, T. F. (2020). Causal emergence in discrete and continuous dynamical systems.

[96] Vendrov, I., Dutchyn, C., and Osgood, N. D. (2014). Frabjous: A declarative domain-specific language for agent-based modeling. In Kennedy, W. G., Agarwal, N., and Yang, S. J., editors, *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 385–392, Cham. Springer International Publishing.

[97] Vern, R. and Dubey, S. K. (2014). Evaluating the maintainability of a software system by using fuzzy logic approach. *International Journal of Information Technology and Computer Science*, 7:67–72.

[98] Walls, C. (2016). *Spring Boot in Action*. Manning Publications Co.

[99] Weisstein, E. W. (n.d.a). Moore neighborhood. From MathWorld–A Wolfram Web Resource.

[100] Weisstein, E. W. (n.d.b). von neumann neighborhood. From MathWorld–A Wolfram Web Resource.

[101] Wilensky, U. and Rand, W. (2015). *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. The MIT Press. The MIT Press, Cambridge.

[102] Yakovenko, V. M. and Rosser, J. B. (2009). Colloquium: Statistical mechanics of money, wealth, and income. *Reviews of Modern Physics*, 81(4):1703–1725.