# Agents Visualization and Web GUI Development in MASS Java

Yifei Yang

A white paper

submitted in partial fulfillment of the

requirements of the degree of

Master of Science in Computer Science & Software Engineering

University of Washington Bothell

2023

Project Committee:

Prof. Munehiro Fukuda, Chair

Prof. Bill Erdly, Member

Prof. Hazeline Asuncion, Member

University of Washington Bothell

# Abstract

Agents Visualization and Web GUI Development in MASS Java

Yifei Yang

Chair of Supervisory Committee:

Prof. Munehiro Fukuda

Computing & Software Systems

Multi-Agent Spatial Simulation (MASS) is an agent-based modeling (ABM) library that supports parallelized simulation over a distributed computing cluster. Places is the collection of elements (a single element is called Place) that are dynamically allocated within the cluster. Agents is a set of execution instances that can reside on a place and migrate to any other Place with global indices. MASS UI consists of two parts: InMASS and MASS-Cytoscape. InMASS allows users to execute command line by line in an interactive way and provides users with additional features. MASS-Cytoscape enables users to visualize places and agents in Cytoscape. However, the current implementation of InMASS hacked MASS too much and became incompatible with the latest versions of MASS. The current visualization is limited to a single computing node and can present limited agents' information. Moreover, the recent MASS does not have a web interface to simplify operations. To address these problems, the goals of this project are: (1) Re-engineering the current implementation of InMASS. (2) Developing Places visualization of 2D continuous space, Binary Tree, and Quad Tree. Improve current Agents visualization. (3) Designing an all-in-one WEB GUI for InMASS design. We adopted original ideas to accomplish the first goal, re-implemented InMASS features, including dynamical loading, checkpointing/rollback, and agent history tracking, and optimized the current codebase. These modifications open up the possibility of the future expansion of InMASS and allow InMASS to serve all MASS users. The project extended current places and agents' visualization to display places in a distributed setting and the agent's number within each place; optimized the operation logic MASS control panel. These additions and optimizations made it easy to use and analyze simulations. The implementation of the web interface enables users to monitor their clusters. And it provided a basic frame for future developers to add on more practical functions.

# TABLE OF CONTENTS

# List of Figures

# Listing

# Chapter 1 Introduction

## 1.1 Overview

Agent-Based Modeling (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 behavior of the system and what governs its outcome [1]. Agent-based models are a type of microscale models [2] that simulate the simultaneous operations and interactions of multiple agents in an attempt to re-create and predict the appearance of complex phenomena. A multi-agent system (MAS or "self-organized system") is a computerized system composed of multiple interacting intelligent agents [3]. Multi-agent systems can solve mathematically complicated problems for an individual agent or monolithic system to solve [4]. The intelligence of agents may refer to methodic, functional, procedural approaches, algorithmic search, or reinforcement learning [5][6]. Agent-based modeling is related to, but distinct from, the concept of multi-agent systems or multi-agent simulation in that the goal of ABM is to search for explanatory insight into the collective behavior of agents obeying simple rules, typically in natural systems rather than in designing agents or solving specific practical or engineering problems. Our recent literature review on agent-based models and multi-agent systems shows that ABMs are used in many scientific domains, including biology, ecology, and social sciences [7].

The MASS library, a parallel-computing library for Multi-Agent Spatial Simulation, was first developed by UW Distributed System Lab (DSLab). As envisioned from its name, the design is based on multi-agents, each behaving as a simulation entity in a given virtual space [8]. Therefore, Places and Agents are crucial elements of the MASS library. Places are virtual spaces of dynamically allocated elements over a cluster of computing nodes. The representational formats of Places are variable, including grid, graph, and tree. The primary component of Places is called a Place. It is pointed to by a set of network-independent indices and is capable of exchanging information with any other Place elements. Besides, MASS helps to partition input data into multiple computing nodes to increase scalability.

Agents are a set of execution instances that can reside in a place, migrate to any other Place with available indices (thus duplicating themselves), and interact with other Agents and multiple Places [9]. Typically, there are three behaviors for agents: MIGRATE, SPAWN, and KILL. One of the essential methods in agents is manageAll. It updates each agent's status based on each of its latest migrate(), spawn(), and kill() calls. These methods are defined in the Agent base class and may be invoked from other functions through callAll. All operations are done in parallel among multi-processes/threads.

Figure 1.1 shows the MASS library model. From the model, places are mapped to threads, whereas Agents are mapped to processes. Agents are grouped into bags, and If agents are associated with a particular Place, they are allocated to the same process whose thread takes care of this Place [9].

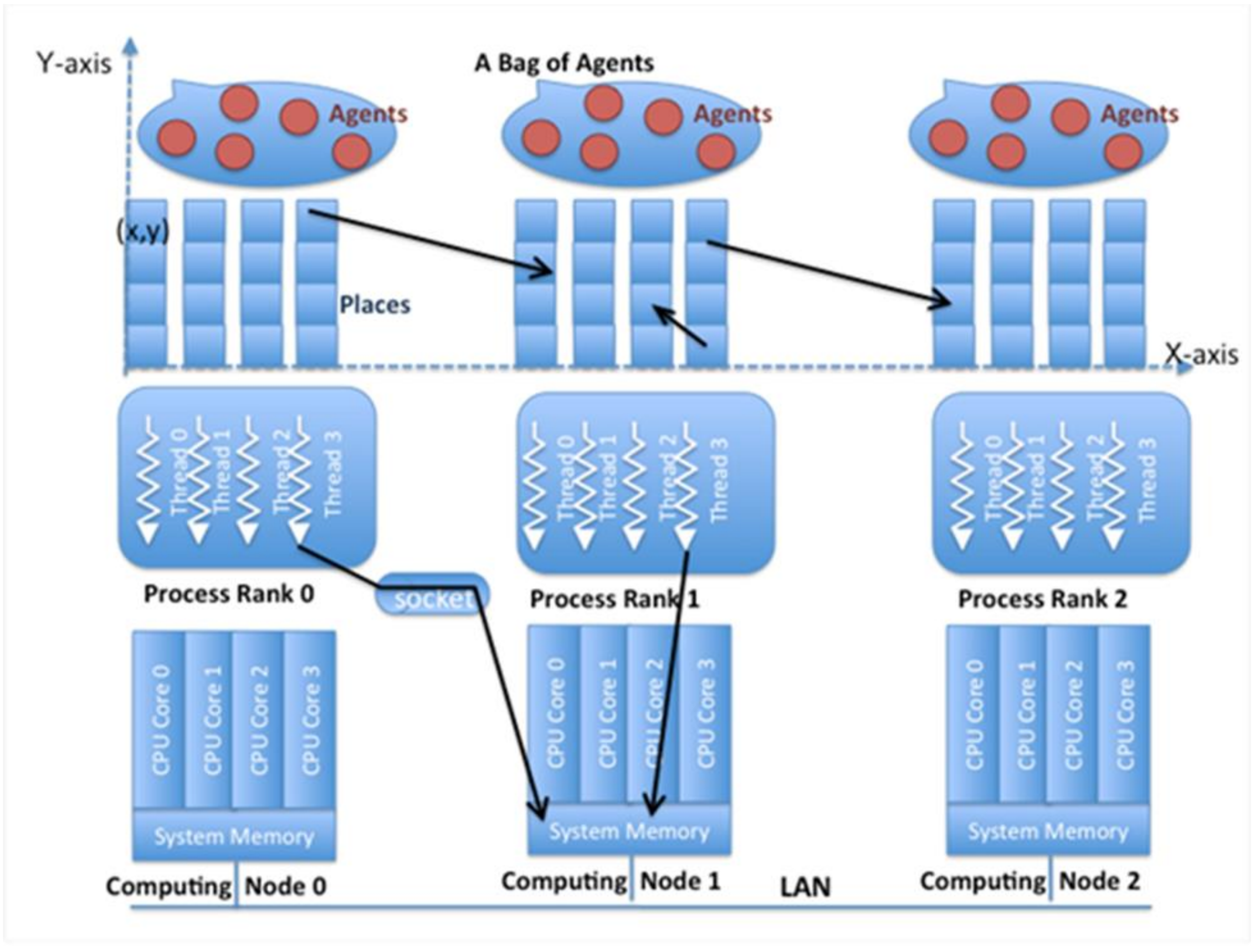


*Figure 1.1 MASS library model*

## 1.2 Motivation

The MASS library supports multiple data structures, including multi-dimensional arrays, graphs, 2D continuous space, binary trees, and quad trees. Some of these data structures are designed to speed up the calculation process of some practical problems. For example, graphs can be used in counting triangles, binary trees in range search, and quad trees in finding the closest pair of points (CPP). However, the limitation is that the simulation process is not transparent to MASS users. Furthermore, some data structures, such as binary trees and quad trees, can sometimes be very complicated to understand because there is no user manual to instruct users on how to use these data structures to solve practical problems. Therefore, it might be challenging for users to verify the results and monitor their simulations. To address the verification problem, the project implemented the visualization of these complicated structures in Cytoscape by extending the current design of MASS-Cytoscape plugins. Besides, the current agents visualization can’t display the number of agents or agentIDs in each place, making it difficult for users to understand agents’ spawn and migration in each simulation step. Another limitation is that the MASS library doesn’t have a web interface to monitor the MASS cluster and simulation. To address the monitoring problem, the project referred to some famous distributed software websites, including Spark UI [10] and HUE (Hadoop User Experience) [11]. As a result, it implemented an all-in-one Web GUI for InMASS users.

## 1.3 Project Goals

The goals of this project are:

1. **Re-engineering the current implementation of InMASS.** Many existing features associated with InMASS, including checkpointing, agent tracking, and lambda expressions, are developed based on a different version of the InMASS branch. As the development of the main branch has been left for a substantial amount of time, the original features cannot be directly merged into the latest InMASS branch, so this goal aims to facilitate InMASS’s better expansion in the future. It requires integrating InMASS’s old features and implementing some features from scratch unless they can’t be merged into the MASS develop branch.

2. **Developing place visualization of 2D continuous space, Binary Tree, and Quad Tree. Improve current Agent visualization design**. Cytoscape [20] is one of the well-known graph visualizers. And we used it to visualize MASS Java's graphs, trees, and 2D space data structures. Daniel Blashaw implemented graph visualization and visualized agents in graph vertices using different colors. However, Tianhui Nie's tree and 2D visualization implementation are limited to a single-computing execution. Therefore, it's essential to visualize places in a distributed setting and visualize the actual number of agents in each place besides displaying colors.
3. **Designing an all-in-one WEB GUI for InMASS.** Currently, a user is responsible for configuring a cluster system, running a MASS application in parallel, launching/stopping InMASS, and connecting Cytoscape to MASS. All these software tools must be operated separately. As Spark, MapReduce, and Docker, we need an all-in-one Web GUI panel that automates MASS invocation and shows the current status of MASS execution.

## 1.4 Success Metrics and Stakeholders

The Level of success of the project:

- Minimum - Finish the re-engineering work, visualization of Places over 2D space and trees. Improve agent visualization in graphs. And implement the all-in-one web interface.
- Expected – (1) Finish Usability testing by our lab members and all functional components testing to ensure everything works (2) Optimize the current MASS control panel in Cytoscape.
- Aspirational - (1) Merge and refactor existing MASS UI codebase for future development (2) Look for other visualization tools which may produce a better effect (3) Extend agent

visualization for trees and 2D continuous space (4) For the web UI, Improve the operation and maintenance process to facilitate future students to maintain.

The stakeholders of this project are mainly all MASS users. Because several DSLab members are working on benchmark applications such as Range Search and Voronoi diagram, the visualization of agents in Cytoscape can help them better understand agents' movement in their simulations. Besides, the web interface provided a new way to understand the status of their MASS clusters.

## 1.5 Paper Structure

The rest of this paper is organized with the following chapters. Chapter 2 discusses other ABM simulators and their agent visualization, and compares InMASS with them. Chapter 3 includes a summary of the work from previous students who have enabled this project's extension to the InMASS library. Chapter 4 discusses the implementation and architecture details of this project. Chapter 5 includes the verification of the results of this project. Finally, Chapter 6 concludes the project and lists recommendations for future work.

# Chapter 2 Related Work

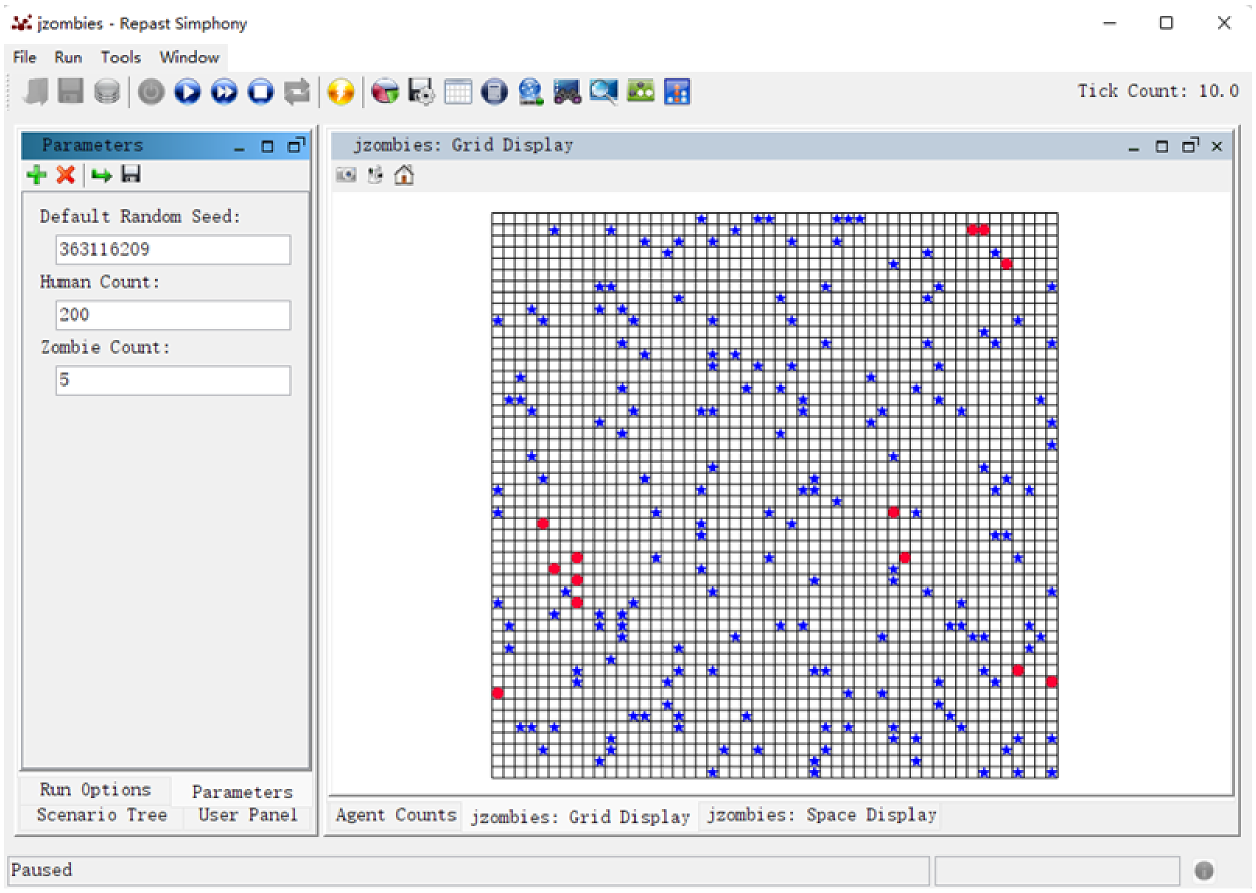
Agent-based modeling (ABM) is a powerful and helpful technique for resolving complicated problems. Various agent-based applications have recently been developed to help analyze ABM problems visually. As many similar ABM simulation systems have visualization features, this section reviews other four free and open-source modeling libraries and compares them with MASS UI. The review includes Repast Simphony, NetLogo, FLAME, and MASON.

## 2.1 Repast Simphony

Sallach, Collier, and others initially developed Repast Simphony (Recursive Porous Agent Simulation Toolkit) at the University of Chicago in 2000 [13]. Later releases of Repast have extended the system to handle large-scale agent simulation application development. It's worth mentioning that Repast is initially written in C++.

Repast Simphony is written in Java instead and is a cross-platform, Java-based, open-source framework for creating agent-based models. All its functions are integrated within the Eclipse IDE (Integrated development environment). The “Context” is the core concept and object in Repast Simphony. It can hold a collection of agents with the same properties and behaviors as a bucket without specifying how they should interact. Agents can travel through Contexts anytime and simultaneously exist in multiple Contexts. Projections mean the workplace among agents in each context. Projections can include grids, continuous spaces, networks, and GIS spaces, which allows different projections in the same context. Each projection has rules and algorithms defining interactions between agents and environments. The next concept is scheduling. Unlike MASS, where we explicitly call all the agents to move, Repast Simphony uses annotations on the movement method, such as @Watch and @ScheduledMethod, which can trigger or schedule the agents' movement. In this way, we only need to create context and put all agents with projections' spaces inside, and then press the start button in the popped-up menu. With this operation, agents can move according to our pre-set annotations' behaviors.

Specifically, Repast Simphony offers a range of features for agent movement visualization, including 2D, 3D and GIS, which allows users to track agents in real time. In addition, Repast Simphony also provides users with the ability to customize the appearance and behaviors of agents, as well as the simulation environment in which they interact. Figure 2.1 shows a visual editor interface for a zombie example. From the parameter panel, users can modify the default human count and zombie count. While Repast Simphony provides a good level of visualization for agent movement, it could be further improved by allowing users to interact with the simulation environment in real time. For example, users could have an ability to pause the simulation, to zoom in/out, or to select any individual agent to view its behavior and movement patterns.



Tick Count: 10.0

Default Random Seed: 363116209

Human Count: 200

Zombie Count: 5

*Figure 2.1: The visual editor interface of the zombie model in Repast Simphony*

*The blue stars are human. The red dots are zombies.*

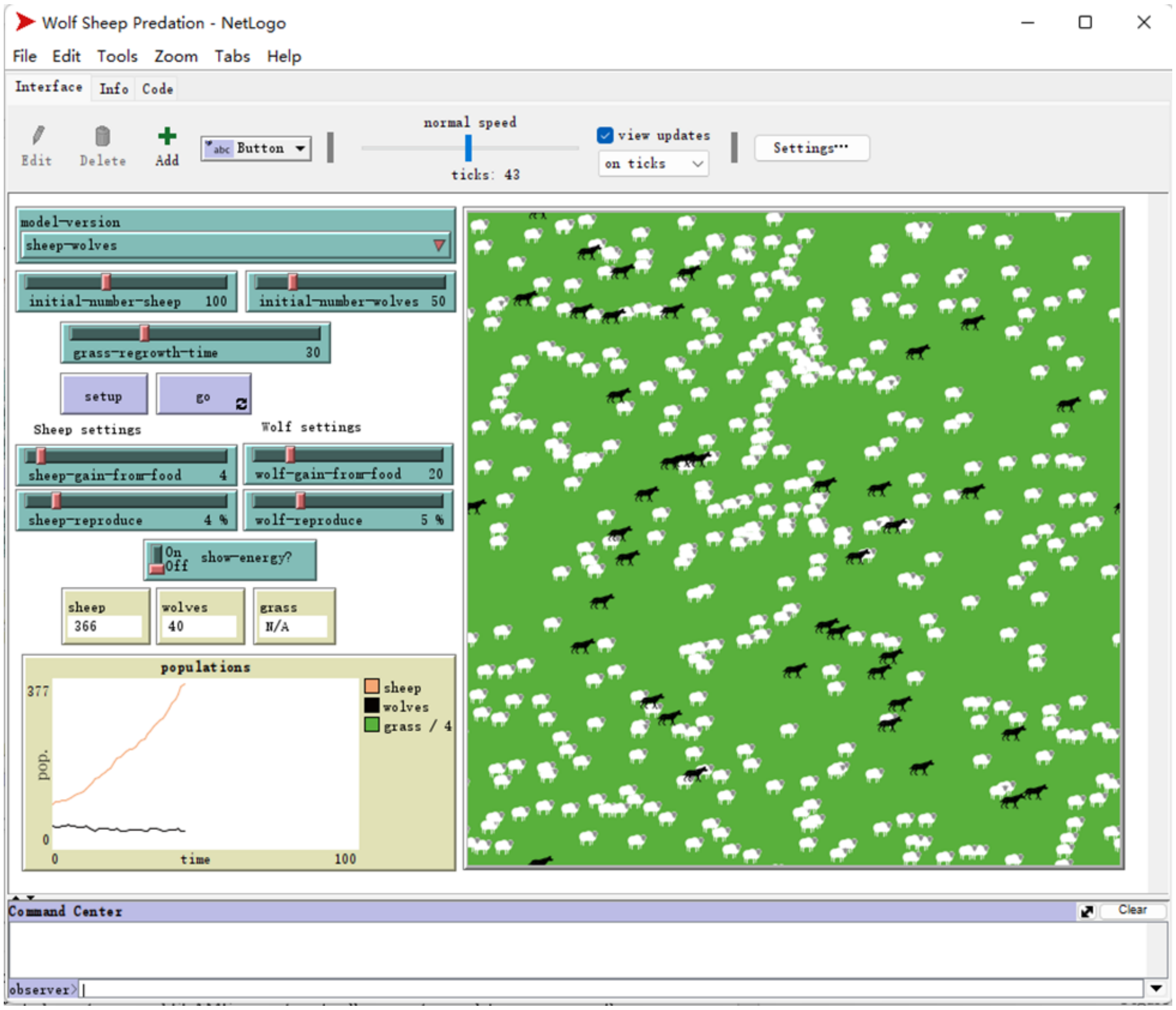
## 2.2 NetLogo

NetLogo [14], developed by Uri Wilensky, is a free, open-source, agent-based simulation environment that uses a modified version of the Logo programming language [15]. Wilensky thought that agent-based simulation environments and languages should be simple enough to allow beginners to start quickly, and that libraries should, in principle, place no limits on what experience users can do. Besides, there are lots of example simulations in its module library.

NetLogo was designed to help teaching complex concepts, and now it can be used to develop more sophisticated simulations like social and natural phenomena.

NetLogo contains four different types of agents: turtles, patches, links, and the observer. Turtles are the most common type of agents. They can be created and manipulated using NetLogo programming language, (i.e., enhanced logo). They have properties and behaviors, and can move around during the simulation. The patches are stationary and arranged in a 2D grid within the simulation. Links represent the connections between other agents in the simulation. For example, links are available in a social network simulation to connect turtles to form the network. The last type is the observer. It is responsible for defining rules, setting initial conditions, and running the simulation. Also, it supports built-in commands for users to interact directly with the four turtles, just like a simple query. Compared with MASS, turtles are like agents, and patches are like places. Links assist agents in exchanging information. The observer serves as controlling the simulation's initialization and finish.

NetLogo also includes a range of built-in commands and functions for working with agents, networks, and spatial data. Figure 2.2 shows a visual editor interface for the sheep-wolf predator model. From the interface, users can change initial parameters by dragging the red button. The graph in the right bottom corner shows the population trend of two agents groups: sheep and wolves. However, it's difficult to see which direction the sheep and wolf models are moving. NetLogo could improve its ability to animate agent movement using other graphical elements like arrows. In addition, NetLogo is written in logo and its visualization tools may not be as compatible with other programming languages, (e.g., Java) or simulation platforms, which can limit its usefulness in some contexts.



normal speed
ticks: 43
view updates: on ticks

model-version: sheep-wolves
initial-number-sheep 100
initial-number-wolves 50
grass-regrowth-time 30
sheep-gain-from-food 4
wolf-gain-from-food 20
sheep-reproduce 4 %
wolf-reproduce 5 %
show-energy? Off
sheep 366
wolves 40
grass N/A

Figure 2.2: The visual editor interface of the wolf predation in NetLogo

## 2.3 MASON

MASON (Multi-Agent Simulator Of Neighborhoods) is an open-source, Java-based simulation toolkit for building and running agent-based models [16]. It is designed to support a large number of agents efficiently on a single machine. Fields in MASON are commonly used to model the environment or spatial properties. Fields provide a convenient and efficient way to represent spatial relationships and enable agents to interact with their environment. Agents can access the field to read or modify the values at specific locations, allowing them to perceive and respond to the spatial attributes in their environment. MASON supports the following features：

1. MASON models are separated from visualization, which means users can run simulations in CLI (command line interface).
2. MASON models are serializable to disk so that users can checkpoint the state of the simulation, mid-run, to a file on the disk. The simulation can restart from this checkpoint even if it's on a different machine, or under visualization versus running on the command line, or even under a different visualization. This is because all models related to simulation implement the serializable interface.
3. The MASON model is entirely encapsulated within a single class: a subclass of sim.engine.SimState which the user designs. The purpose is to give users a place to store any and all elements that is necessary for the simulation.
4. MASON replaces some of Oracle's internal classes to improve performance.

The last thing about MASON is that it has another version called Distributed MASON. The idea is like our current MASS implementation. This is because one single machine's computing power is limited in some cases. Hence, MASON considers cutting up fields into regions and hands each region to a process. The region of the field that a given process is responsible for is known as its partition. Each partition takes care of its simulation. Like MASS, agents can travel through different machines and inevitably need to access data or agents in another process. But MASON mentions that agents' inter-process communication can be prolonged, especially when exchanging serialized data with a remote agent. Given that considerations, MASON suggests limiting the simulation to one machine.

Fields can also be visualized to provide a graphical representation of the spatial data during the simulation. MASON provides a wide range of 2D and 3D visualization for user-provided models. Figure 2.3 shows 2D visualization of MASON. And MASON provides some built-in shapes for the underlying models including square, hexagon, compass, etc. However, MASON's user manual is over 450 pages, and it can be complex and require a steep learning curve to understand the extra visualization.

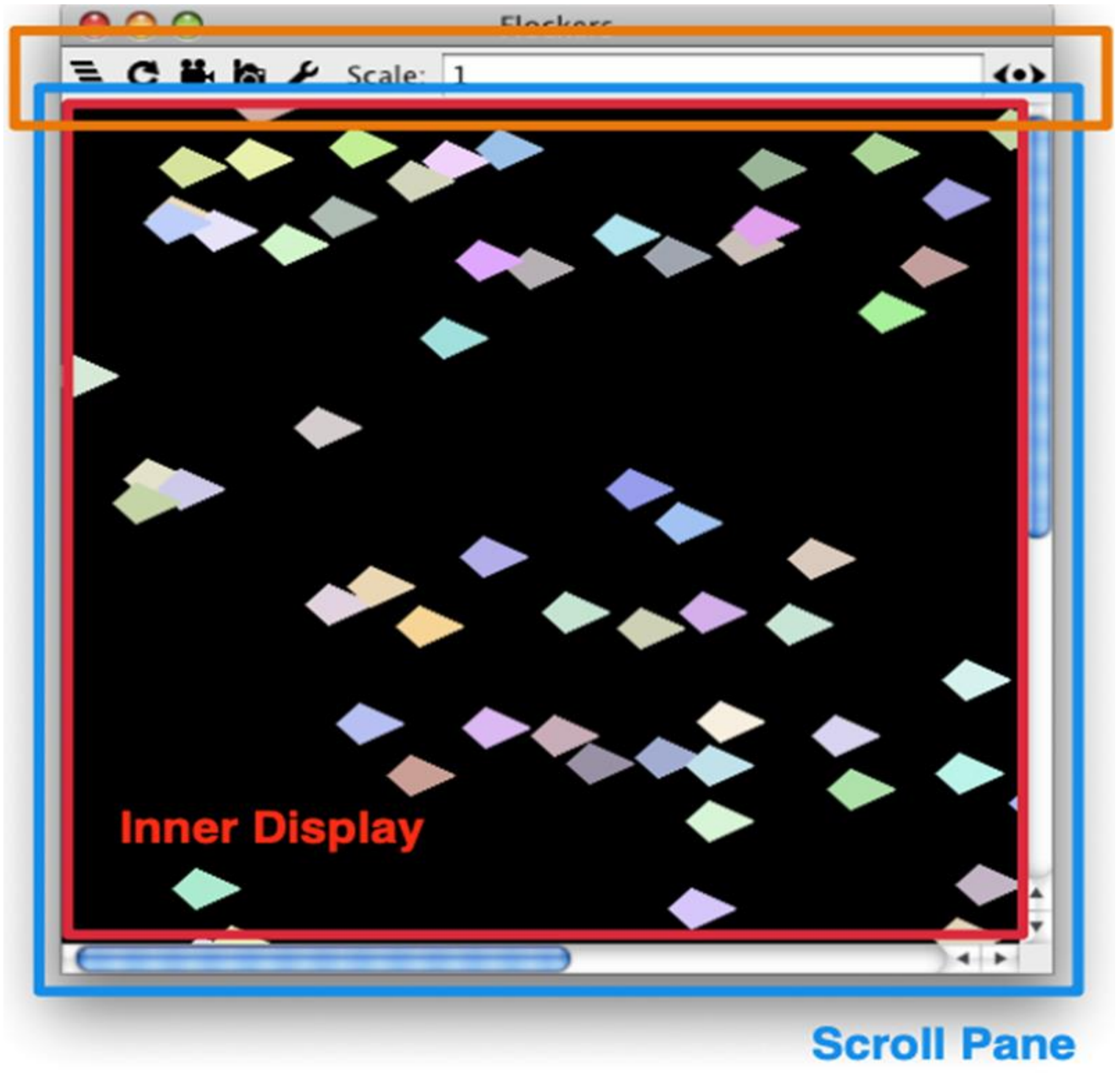


*Figure 2.3: MASON's 2D visualization*

## 2.4 FLAME

FLAME (Flexible Large-scale Agent Modeling Environment) [17], written in C, is a modeling and simulation platform that can be run on high-performance computers (HPCs). In FLAME, Agents are first-class citizens, and FLAME can automatically generate simulation programs after we define agent-based models. The model has two parts: agents' functions (written in C) and configuration file (written in XML). The XML file specifies attributes, including agent names, local variables, functions, and messages.

Figure 2.4 shows the life cycle of each agent. Agents will execute a series of transition functions from the start to the end state. In these functions, agents can exchange messages by reading incoming or writing outgoing messages. Because of agent models' parallel execution, communication must be synchronous when a particular message type is involved. Besides, filtering messages is needed because of broadcast communication. Compared with MASS, the agents of both sides have some similarities. Both agents have their memory, are independent, and can send or broadcast messages. The difference is that agents in FLAME can't travel through processes. In addition, an XML file manages the essential part of the FLAME model, which may increase complexity when different agents are involved. Sometimes it may not be intuitive enough for users. The advantage of FLAME is that it mainly focuses on simulations with many agents. They perform better in that aspect, primarily since it's written in C. However, this might represent a problem for the modelers, given that C requires a certain experience and expertise.

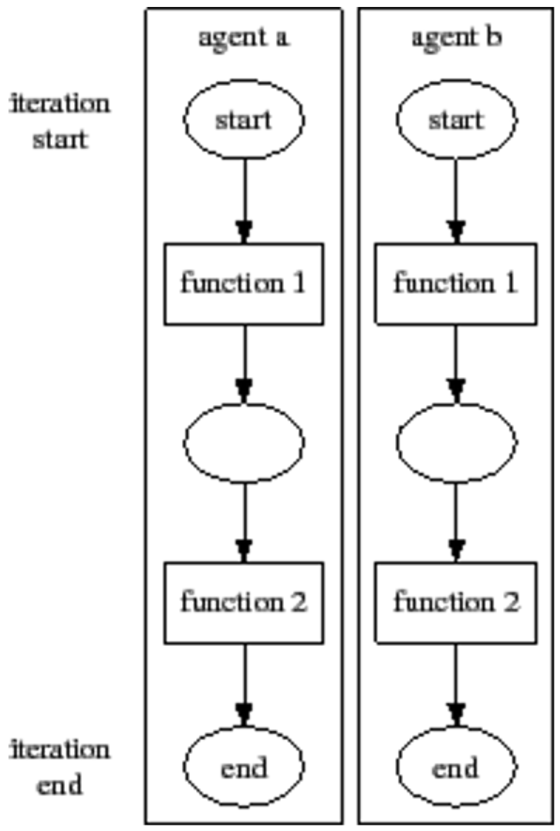


*Figure 2.4: FLAME Process*

As for the visualization, NetLogo and Repast can produce output in real time. FLAME on the other hand, requires that a model is initially run for a specific number of iterations to produce a textual output. This textual output can be later visualized by utilizing the FLAME visualizer. Therefore,the models of NetLogo and Repast are directly executable in visual simulation, while Flame models can be visualized with the use of an external tool.

## 2.5 Summary

All previous four simulators can't interact with the simulation in real time. Interactive computation has been implemented in MASS within the InMASS branch. As for parallelization, Repast Simphony, NetLogo, and FLAME only support single execution. Although MASON's distribution feature can support multiple machines, it's not recommended according to the official guidance.

Instead, MASS can fully run the simulation in parallel.  As for the learning curve and flexibility, MASS uses third-party software, (i.e., Cytoscape) to visualize both agents and places. Within the Cytoscape, users can easily change the output layer format for better observation. Besides, MASS supports tracking any agent's movement and records agents' migration histories. This path can be displayed in the Cytoscape.

# Chapter 3 InMASS and Previous GUI Work

This section offers a summary of previous work that includes five basic components. First, we talk about the InMASS and two features associated with it. Second, we talk about the reason why we choose Cytoscape as the visualization tool. Third, we talk about the previous graph visualization. Fourth, we talk about the previous tree visualization. Figth, we talk about Agent Visualization. In the last, we talk about the MASS control panel.

## 3.1 InMASS

Nasser Alghamdi [18] was the first developer of InMASS. InMASS provides an interactive environment for users to communicate with their MASS clusters. Users can execute statements line by line instead of hard coding them into a single jar package. That means users can determine a specific implementation of Place and Agent during the simulation. The benefit is that users can change and query the current state of the simulation at run time. Therefore, InMASS provides a dynamic and flexible solution for each simulation. To achieve better performance and minimize the impact on the codebase, Daniel Blashaw [20], the second developer of InMASS, enhanced simulation controls of InMASS in the following two aspects:

1. Management of dynamically created classes.
2. Choice of Memory and disk space for maintaining agents' history.

On top of that, InMASS uses JShell [19] to achieve interactive operations. JShell is a Java Shell tool, an interactive tool for learning the Java programming language and prototyping Java code. For example, Read Evaluate Print Loop (REPL), an interactive interpreter runs statements, declarations, and expressions as they are typed in the console or a terminal and displays the results immediately.  Besides, the current InMASS supports the following features: checkpoint, rollback, lambda expressions, and agent tracking. All these features are combined to make InMASS an excellent choice for users with needs for dynamic operations at run time.

### 3.1.1 Checkpoint/Rollback

Checkpointing allows users to take an on-going execution snapshot of the current MASS cluster. InMASS provides three strategies for checkpointing. They are InMemoryStoreStrategy, InDiskStoreStrategy, and InTemporaryDiskStoreStrategy. As the names imply, users can checkpoint the current state of the simulation in memory, or temporarily in the disk, or permanently in the disk. All functional calls after this point will be recorded in InMASS. When users want to rollback to a certain call x, the simulation will roll back to the checkpointing state first, and then re-execute calls until the call x. The goal of this feature is to make the simulation controllable and fault tolerant.

### 3.1.2. Agent History Tracking

Daniel Blashaw first developed the agent tracking API. The feature aims to make agent movement traceable, as agents may spawn child agents to expand the search radius and the parent agent may be killed after spawning child agents. Thus, it’s not possible for each agent to record places it has traveled to. To resolve this, the tracking feature instead has each place record the IDs of agents residing on it. So, the information of agent path will not be affected when an agent is killed. To get all agents’ complete paths, the agent tracking API helps users collect path information from each place and combine them together. In some benchmark applications such as Triangle Counting (TC) [24], showing the full agents’ paths is useful to better understand their behaviors.

## 3.2 Cytoscape

Cytoscape is an open-source software for visualizing and analyzing complex networks. It is widely used by researchers, biologists, and data scientists to explore and understand the complex relationships and interactions within biological systems, social networks, and other complex systems [20]. There are three main reasons of choosing Cytoscape as this project's visualization tool:

First, the Cytoscape desktop application is written in Java, which means it can be used on various operating systems, including Windows, Mac, and Linux. However, Cytoscape does not currently support running on Macs with M1 (Silicon Valley) chips. If you are using an M1 Mac and want to run Cytoscape, one possible way is to use the build script from their GitHub web page (https://github.com/cytoscape/cytoscape) to build a new Cytoscape because all related codes are open source.

Second, Cytoscape's key strengths are its flexibility and extensibility. It is designed to use various data sources and file formats, including SIF (Simple Interaction Format), GML, XGMML, and GraphML. To cooperate with these data formats, Cytoscape offers a user-friendly interface and a range of powerful features for working with networks, including tools for importing and exporting data and visualizing layout networks. On top of visualization, it allows users to customize the appearance of their network and highlight specific features or patterns.

Third, Cytoscape offers a range of plugins that can enhance the current application by adding new capabilities. These plugins are all managed by App Manager and can be developed by users or third-party developers. One of the powerful plugins is called StringApp, which may be familiar to researchers in the biological field. This supports importing needed data directly from the online STRING database and visualizing it locally.

## 3.3 Graph Visualization

Justin Gilroy [21] implemented MASS's graph data structure and maintenance features. Besides, he was the first developer to visualize graphs constructed over multiple computing nodes, using Cytoscape. To conveniently import graphs into the simulation, Justin included several formats, including HIPPIE and MATSim, in his implementation.

However, there are some limitations to his design. Cytoscape is written in Java, and the desktop application runs on JVM. Memory usage would be an obvious problem when introducing a large graph. To address this problem, Daniel Blashaw introduced the idea of the partial graph. When users want to display a large graph in Cytoscape, they can choose to display n-neighbors of the vertex they are most interested in, and the information around that vertex can be more specific. It can save much more time than loading the whole graph.

## 3.4 Tree Visualization

Yuna Guo [22] was the first student who implemented tree structures for the MASS library. The structures include binary tree, quad tree, and 2D continuous space. The binary tree structure allows the user to construct a unique binary tree using a bunch of two-dimensional point coordinates at each compute node. Although the graph structure can be used as a binary tree sometimes, the binary tree structure is mainly designed to provide a more intuitive programming model and facilitate simulations that are specific to binary trees, such as the RangeSearch benchmark application [24]. The quad tree is used in the closest pair of points (CPP) benchmark application [24].

Tianhui Nie [23] was the first student who visualized places of 2D continuous space, binary tree, and quad tree. She implemented the BinaryTreeNodelModel and QuadTreeNodelModel to help transfer binary-tree and quad-tree Places data to Cytoscape. As for 2D continuous space visualization, she managed to map quad-tree Places data into 2D space using the tree index to calculate the length and width of each grid in 2D space.

## 3.5 Agent Visualization

Daniel Blashaw first implemented agent visualization for the TriangleCounting application in Cytoscape. To successfully transfer agents' history path data to Cytoscape, Daniel enhanced the MASS- Cytoscape plugin and added a new task called "Import Agents" to integrate all agents' path data into a single table within Cytoscape. Besides, he implemented two modes of displaying agents: the first mode is "Heat Map" where each vertex will change the shade of its color according to the number of agents residing on it, and the second one is "Agent Path" where only the selected agent's path will be highlighted.

## 3.6 MASS Control Panel

The MASS Control Panel was first created by Daniel Blashaw where users can change certain parameters to import graphs and agents. It is composed of the following four main plugins: import-network, export-network, import-agent, and mass-agents. The import-network plugin is responsible for importing graphs from InMASS to Cytoscape, while the export-network plugin is responsible for exporting graph networks. The import-agent plugin is responsible for import agents' path data from InMASS, and the mass-agent plugin is responsible for creating MASS-Control Panel and integrating the functions of the previous three plugins. Overall, the MASS Control Panel provides all InMASS users with a user-friendly GUI.

Tianhui Nie further enhanced the control panel by adding features that support importing other structures, including binary tree, quad tree, and 2D continuous space.

## 3.7 Summary

All previous students have made significant contributions to the development of InMASS. Nasser Alghamdi first implemented InMASS and its checkpoint/rollback features, while Daniel Blashaw later enhanced InMASS with dynamic loading and choice of different storage strategies. However, their work is based on the old version of MASS, and to work with the latest version of MASS, InMASS and all related features need re-engineering. Additionally, Daniel Blashaw visualized graph places and agents in Cytoscape. But agents within each vertex are only displayed using the shade of colors instead of dots or agentIDs. Tianhui Nie was the first to implement the visualization of binary tree, quad tree, and 2D continuous Space. However, her visualization can only work with a single computing node, and the current MASS Control Panel is not user-friendly for importing other structures. Users must manually input the import type, and the inner logic still needs improvement.

# Chapter 4 MASS UI Enhancement

This chapter shows the architecture and features of the new MASS UI: (1) Enhanced InMASS, (2) Enhanced MASS-Cytoscape, and (3) Web GUI.

## 4.1 Enhanced InMASS

To provide the main functionality of InMASS, we have leveraged Nasser Alghamdi and Daniel Blashaw’s work on InMASS. We re-implemented dynamic loading and extended existing features including checkpoint/rollback and agent history tracking to adapt to MASS. In addition, we also optimized the code structure to improve readability.

### 4.1.1 Dynamic Loading

The previous implementation relies on passive dynamic loading, where remote nodes only receive specific class definitions from the master node when creating a Places or Agents object. In contrast, our project uses active dynamic loading, where class definitions are passed to remote nodes when a Places or Agents object is created on the master node. To facilitate this process, we develop a new message type, "CLASS_INITIALIZATION", as shown in Figure 4.1. This message includes a new value, "bc.getByteClass", where "bc" is an object of the ByteCode class that is added to encapsulate the class defined on the master node. To transfer this definition between computing nodes, we use a byte array to store the serialized classes.

Upon receiving the initialization message, remote nodes deserialize the ByteCode object and store it into a local classLoader called InMASSClassLoader, which is added to enhance the default classLoader for InMASS users. The InMASSClassLoader fetches the passed class definitions and is used in the SimpleObjectFactory that manages all object creation. Within the InMASSClassLoader class, there are two variables: byteClassStore and classStore. The byteClassStore maps class names to the corresponding ByteCode objects, while classStore stores all deciphered class objects. After remote nodes store the class definition in their local InMASSClassLoader, they send ACK messages back to the master node, which confirms it with the barrierAllSlave method.

In step 6, the host node begins sending the PLACE_INITIALIZATION message to all remote nodes. Remote nodes load their local InMASSClassLoader and use the overridden method findClass to fetch class definitions from the local classStore maps assigned values in step 3. As a result, remote nodes create local Places and sent an ACK message back to the master node.

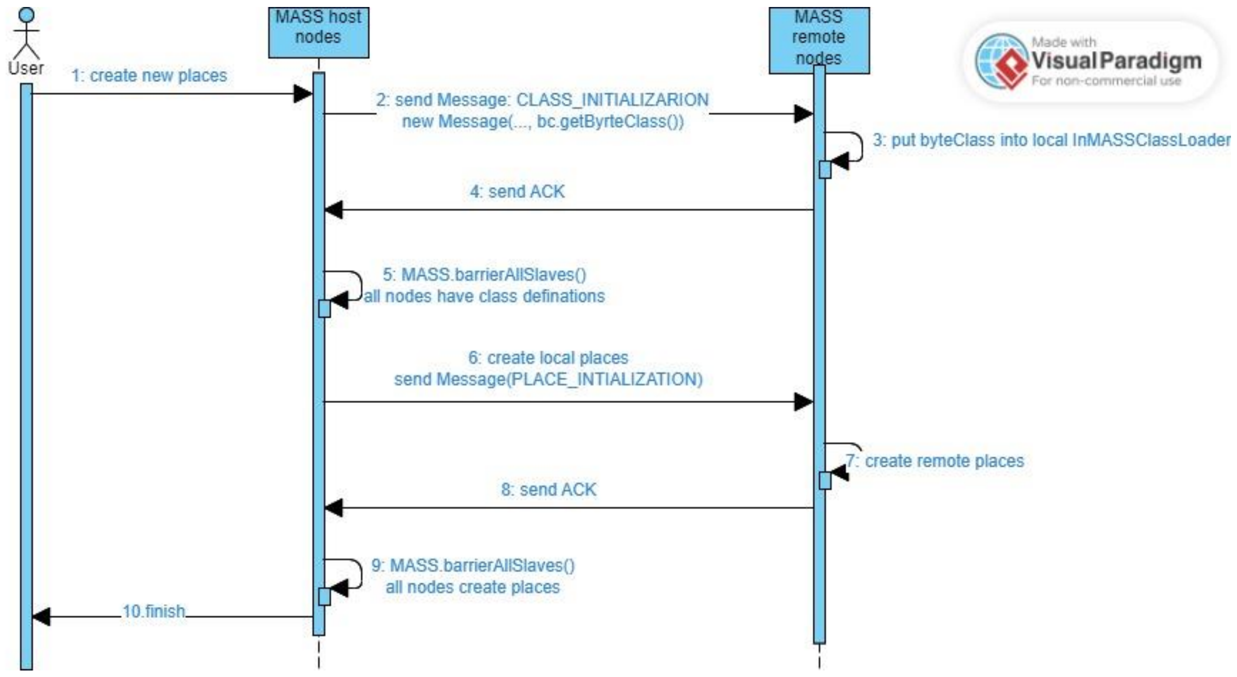

User
MASS host nodes
MASS remote nodes
Made with Visual Paradigm For non-commercial use
1: create new places
2: send Message: CLASS_INITIALIZARION new Message(..., bc.getByrteClass())
3: put byteClass into local InMASSClassLoader
4: send ACK
5: MASS.barrierAllSlaves() all nodes have class definations
6: create local places send Message(PLACE_INTIALIZATION)
7: create remote places
8: send ACK
9: MASS.barrierAllSlaves() all nodes create places
10.finish

*Figure 4.1: The Sequential Diagram of the New Dynamic Loading Feature*

To address deserialization issues with agents that can migrate across different computing nodes, we add a new class, InMASSObjectInputStream, that is only used in deserialization. This class interact directly with InMASSClassLoader to retrieve the agent class definition, allowing remote nodes to deserialize these agent objects successfully.

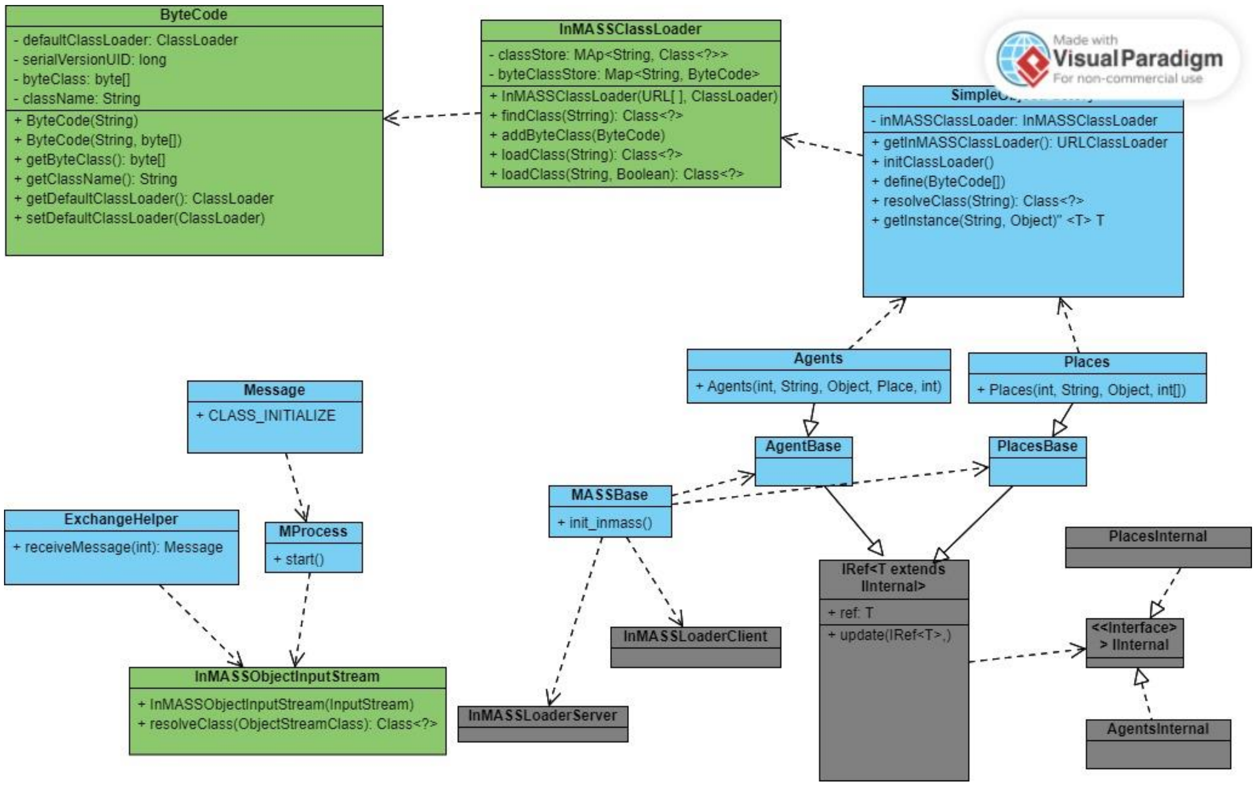

ByteCode
- defaultClassLoader: ClassLoader
- serialVersionUID: long
- byteClass: byte[]
- className: String
+ ByteCode(String)
+ ByteCode(String, byte[])
+ getByteClass(): byte[]
+ getClassName(): String
+ getDefaultClassLoader(): ClassLoader
+ setDefaultClassLoader(ClassLoader)
InMASSClassLoader
- classStore: MAp<String, Class<?>>
- byteClassStore: Map<String, ByteCode>
+ InMASSClassLoader(URL[ ], ClassLoader)
+ findClass(Strring): Class<?>
+ addByteClass(ByteCode)
+ loadClass(String): Class<?>
+ loadClass(String, Boolean): Class<?>
Made with Visual Paradigm For non-commercial use
- inMASSClassLoader: InMASSClassLoader
+ getInMASSClassLoader(): URLClassLoader
+ initClassLoader()
+ define(ByteCode[])
+ resolveClass(String): Class<?>
+ getInstance(String, Object)" <T> T
Agents
+ Agents(int, String, Object, Place, int)
Places
+ Places(int, String, Object, int[])
Message
+ CLASS_INITIALIZE
AgentBase
PlacesBase
MASSBase
+ init_inmass()
ExchangeHelper
+ receiveMessage(int): Message
MProcess
+ start()
PlacesInternal
IRef<T extends IInternal>
+ ref: T
+ update(IRef<T>,)
<<Interface>> IInternal
InMASSLoaderClient
InMASSObjectInputStream
+ InMASSObjectInputStream(InputStream)
+ resolveClass(ObjectStreamClass): Class<?>
InMASSLoaderServer
AgentsInternal

*Figure 4.2: Class Diagram of the New Dynamic Loading Feature*

Due to the complexity of the MASS library, this diagram only shows affected classes. Classes marked in green means new classes. Classes marked in blue means modified classes. Classes marked in gray means removed classes.

### 4.1.2 Checkpointing/Rollback

To minimize complexity and minimize disruption to the existing MASS implementation, our approach aims to simplify the design. We remove the previous implementation of encapsulating all internal fields within two new classes (AgentsInternal and PlacesInternal). Instead, we adopt the original idea of using placesMap and agentsMap to store a snapshot of the simulation for checkpointing. The placesMap stores a PlacesBase object for each created places object, identified by a handle ID. Similarly, agentsMap stores an AgentsBase object for each created agent object, also identified by a handle ID. During checkpointing and rollback operations, these base classes are serialized or deserialized. To ensure proper serialization, we mark certain fields (objectFactory, eventDispatcher, and clock) as transient since they either cannot or do not need to be serialized. We use the MState object to maintain a snapshot of the checkpointed placesMap and agentsMap. During deserialization, the MState object helps us recover the old maps. For AgentsBase deserialization, we have to deserialize it twice since the Agents class, which extends AgentsBase, adds a new field called "localAgents". The first deserialization is used to update the inner values of the AgentsBase object, while the second one is required to replace the current localAgents with the old version. This necessitates a forced conversion of the AgentsBase type to the Agent type.

### 4.1.3 Agent History Tracking

In the previous tracking feature, agents record their location and iteration time when manageAll is invoked. The agents have three behaviors: migration, spawning, and killing. During migration, the destination place records the current time and the upcoming agent. During spawning, the current place records the time and the spawned agents and further records the mapping between child and parent agents. However, in most simulations, the agent's location changes during migration, making it unnecessary to record the agent's location during spawning. To address this issue, we add a new parameter, "shouldAddHistory," to the recording function. This Boolean value is set to true when agents are migrating and false when they are spawning. In the latter case, the current place only records the mapping between child and parent agents.

We also optimize the synchronization of the iteration among all computing nodes. For the agent initialization, all computing nodes first set their local iteration to zero. The flag value, shouldIncrement, is set to true when the initialization of agents is done. Figure 4.3 illustrates the iteration synchronization order in agents manageAll among multiple nodes. In step 1, we add the iteration variable to the AGENTS_MANAGE_ALL message, which is the first synchronization step to ensure that every computing node has the same iteration value as the master node at the beginning of manageAll. In step 2, both the master node and remote nodes execute manageAll and update agent location and history. In step 3, the master node creates an array to retrieve the updated iteration from remote nodes. In step 4, the master node updates the array with iteration values sent from remote nodes through barrierAllSlaves. We extend the barrierAllSlaves function to not only receive the number of agents used to update the localAgents array but also the iteration values from remote nodes. Step 5 is the second synchronization step that updates the iteration value with the maximum value calculated from the returned array on the master node and ensures that the current iteration value is the latest value for the next manageAll.

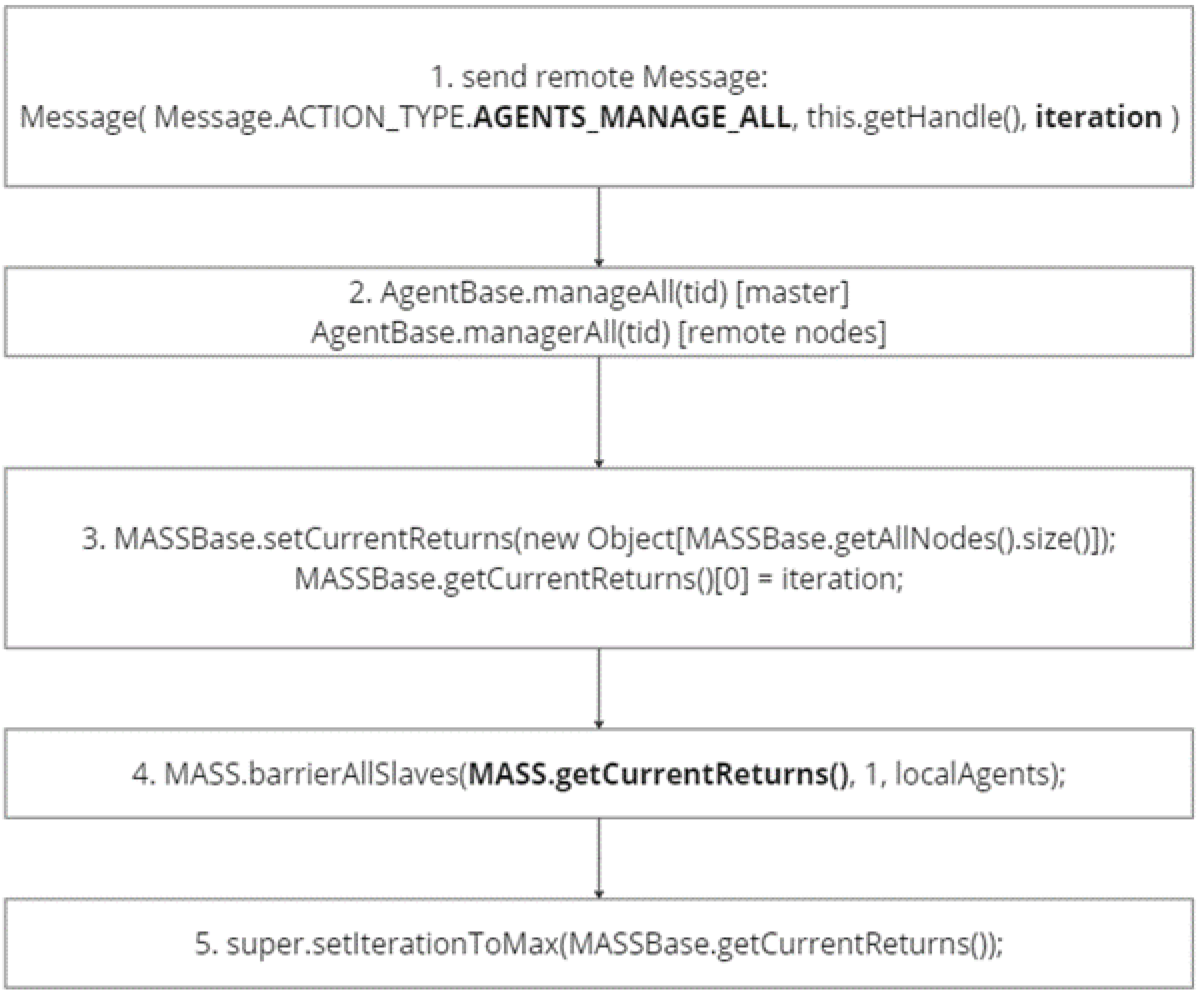

1. send remote Message:
Message( Message.ACTION_TYPE.**AGENTS_MANAGE_ALL**, this.getHandle(), **iteration** )
2. AgentBase.manageAll(tid) [master]
AgentBase.managerAll(tid) [remote nodes]
3. MASSBase.setCurrentReturns(new Object[MASSBase.getAllNodes().size()]);
MASSBase.getCurrentReturns()[0] = iteration;
4. MASS.barrierAllSlaves(**MASS.getCurrentReturns()**, 1, localAgents);
5. super.setIterationToMax(MASSBase.getCurrentReturns());

*Figure 4.3: Iteration Synchronization Order of Agents ManageAll*

## 4.2 Enhanced MASS-Cytoscape

The implementations are to expand and optimize the three plugins: import-network, import-agent, and mass-agents. Figure 4.4 shows an overview of MASS-Cytoscape architecture. On the left side, we have a user space where users can interact with both InMASS and Cytoscape. On the right side, we demonstrate import classes and components of Cytoscape as the client and InMASS as the server.

The general execution flows are:

(1) A user starts InMASS and executes a simulation with creating both places and agents.

(2) A user creates a CytoscapeListener instance using places object.

a) The listener initiates a thread to create a socket server listening to default port 8165
b) The listener initiates a HashMap to register all processors. The key is a string describing the function of the processor. The value is a processor that implements Supplier functional interface. The processor invokes a certain function within created places and returns the obtained result.

(3) A user executes multiple function calls (agents spawning, migration, and manageAll).

(4) A user opens the Cytoscape desktop application and accesses the MASS Control Panel.

(5) A user changes the default configuration including host name, port, data structure.

(6) A user imports a network of the created places.

a) Cytoscape reads hostname and port, establishes socket connection with InMASS.
b) Cytoscape sends a request string based on the selected data structure to InMASS.
c) InMASS receives the request string and parses it with registered processors. The processor executes its registered function and sends returned data back to Cytoscape.
d) Cytoscape receives the data and updates its node and edge table.

(7) Cytoscape creates a new network view, and the user can see rendered places from node and edge tables.

(8) A user imports agents history data.

a) Cytoscape reads hostname and port, establishes socket connection with InMASS.

b) Cytoscape sends two request strings based on the selected data structure to InMASS. The first string is to request agents' full path data. The second string is to request agents' historical count.

c) InMASS receives the request strings separately and parses it with registered processors. The processor executes the registered function, and then sends path and historical count data back to Cytoscape.

d) Cytoscape receives agents' path data and imports the data into a newly created table called AgentHistory.

e) Cytoscape receives agents' historical count data and imports the data into a newly created table called AgentNumHistory.

(9) A user chooses either mode from two view modes:

a) Heat Map: A user changes time value from 0 to maximum iteration time that is calculated from the AgentHistory table. Agents are displayed as dots within each visualized place.

b) Agent Path: A user selects any agent name from the scroll panel. The corresponding agent path is highlighted. The colors of involved nodes and edges are distinct from others.

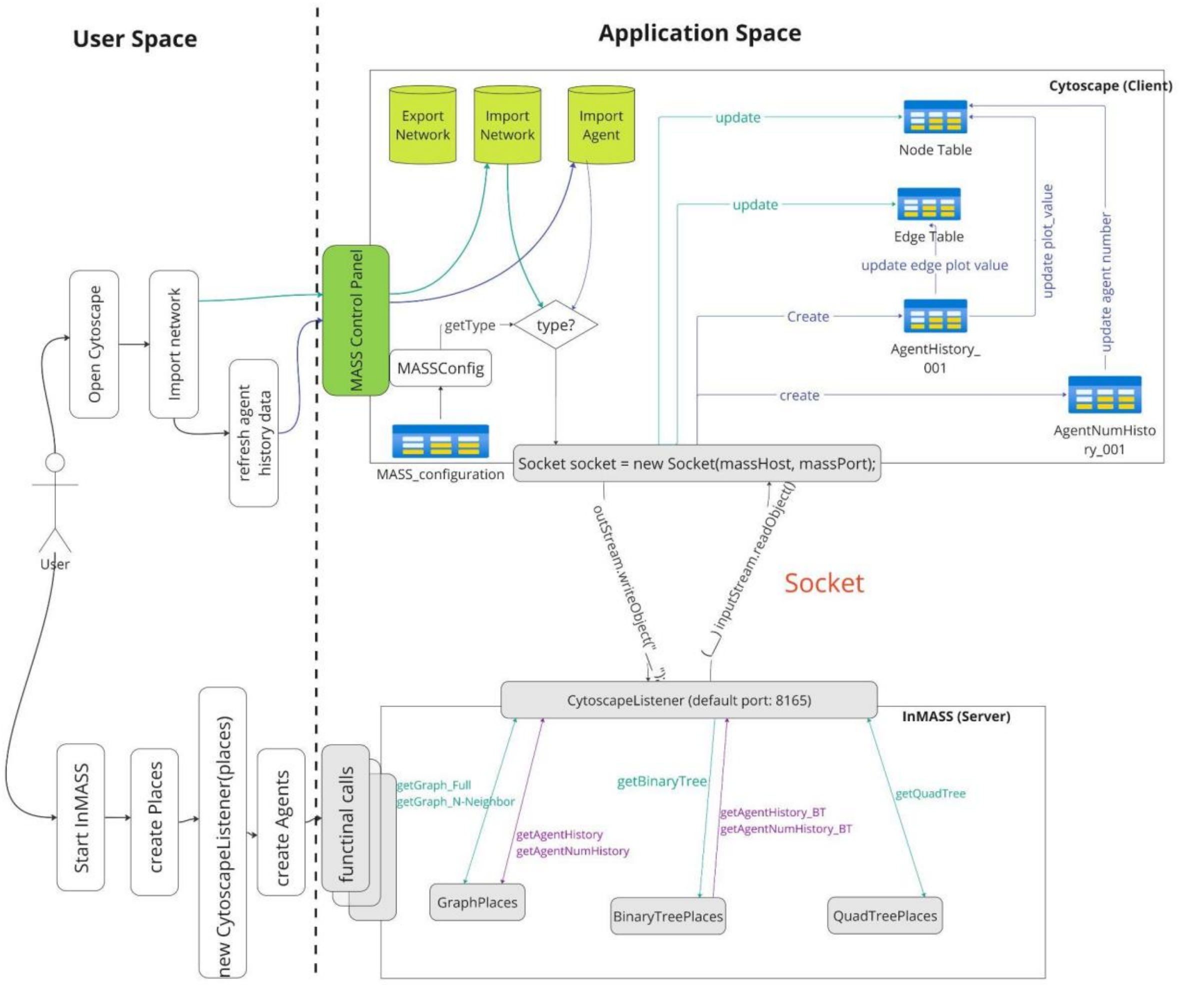


*Figure 4.4: MASS-Cytoscape Integration System Architecture*

The left side is the user space where users can run the simulation within InMASS and open Cytoscape to visualize created places and agents. The right side is the application space where we only demonstrate important components, classes, and methods. The three green cylinders are MASS-Cytoscape plugins. Arrows marked in green indicate the flow when importing network (visualize places). It is managed by the import-network plugin. Arrows marked in purple indicate the flow when importing agents (visualize agents). It is managed by the import-agent plugin. Within InMASS, methods marked in green are used to return data for Places visualization. Methods marked in purple are used to return data for Agents visualization. After retrieving the data, the plugins then create or update related table information.

## 4.2.1 Enhanced Place Visualization

### a. Binary Tree Places

To enable the visualization of binary trees in a distributed setting, we introduce a new message type, "BINARY_TREE_GET_PLACES," which facilitates the retrieval of binary tree information from remote nodes. Given the recursive nature of binary trees, we only request the root node of each remote binary tree.

As illustrated in Figure 4.5, upon receiving a request string from Cytoscape, the master node invokes the transferBinaryTree function, which converts the class of the component node from BinaryTreePlace to BinaryTreeModel. This step aims to simplify the binary tree data to include only the necessary information for visualization. Next, the master node adds the converted tree node rootA' to a list. In the getRemoteBinaryTrees function, the master node sends a "BINARY_TREE_GET_PLACES" message to all remote nodes and appends the fetched tree nodes rootB' and rootC' to the list. Finally, a list of binary tree nodes data is sent to Cytoscape for visualization.

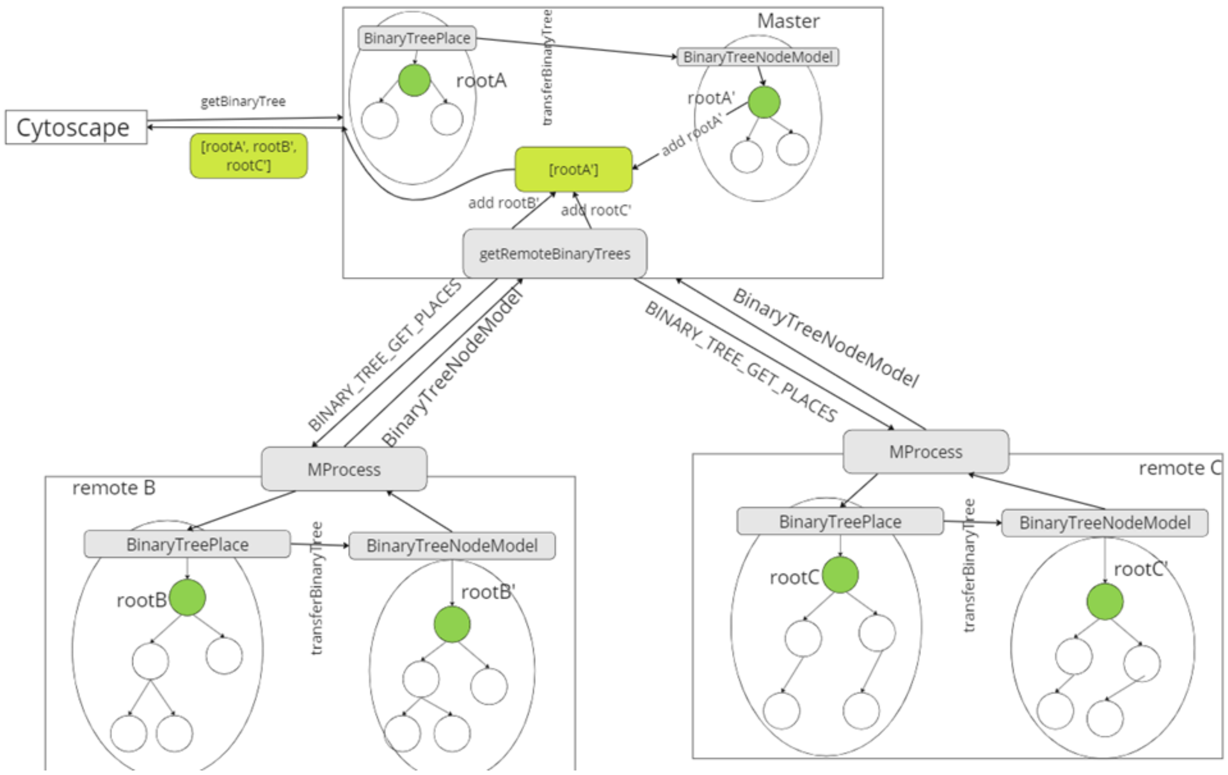


*Figure 4.5: The General Process of getBinaryTree Function*

### b. Quad Tree Places

Similarly, for quad trees, we introduced a new message type, "QUAD_TREE_GET_PLACES," to retrieve information from remote nodes. We also simplify the original QuadTreePlace class by introducing the QuadTreeModel class.

To improve the visual representation of quad trees, we add two new fields to the QuadTreeModel class: isLeaf and coordinates. The coordinates field is used as the default label for the visualized places, while the isLeaf field distinguishes between leaf nodes (which correspond to actual values) and non-leaf nodes (which correspond to value ranges used to split the quad tree).

To achieve this effect, we leverage one of Cytoscape's mapping functions, DiscreteMapping. Specifically, we specify "NODE_SHAPE" as the selected style and "isLeaf" as the mapping column. The function returns a key-value pair map, where the keys are the actual values in the isLeaf column, and the values are pre-defined node shapes in Cytoscape. As shown in Listing 4.1, we set the shape of leaf nodes to circle and non-leaf nodes to rectangle.

*Listing 4.1: Code snippet of the Style Mapping for Quad Tree visualization*

```
// NODE SHAPE (style functions)
DiscreteMapping nodeShapeMapping = (DiscreteMapping)
```

```
      discreteVisualMappingFunctionFactory.createVisualMappingFunction("isLeaf
                                        Boolean.class, BasicVisualLexicon.NODE_SHAPE);
nodeShapeMapping.putMapValue(true, NodeShapeVisualProperty.ELLIPSE);
nodeShapeMapping.putMapValue(false, NodeShapeVisualProperty.RECTANGLE);
style.addVisualMappingFunction(nodeShapeMapping);
```

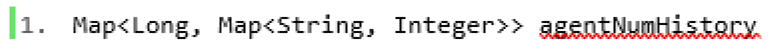
## 4.2.2 Enhanced Agent Visualization

### a. Graph Agents

To improve visualization clarity, it is more effective to represent agents as dots on each place rather than visualizing their IDs. The number of dots on each place corresponds to the number of agents present. To track the number of agents at each iteration time, the project leverages the

MASS agent tracking API to record each agent's travel history. This history consists of two parts: the iteration time and the linear index of visited places. To store this information, a global HashMap named "agentNumHistory" is created, as shown in Listing 4.2. The "Long" type in the map represents the iteration time, which increments by one each time agents invoke the "manageAll" function. The "String" type represents a place's linear index, and the "Integer" type represents the number of agents.

Listing 4.2: Code snippet of agentNumHistory type



```
Map<Long, Map<String, Integer>> agentNumHistory
```

To transfer the "agentNumHistory" data to Cytoscape, a new function named "getAgentNumHistory" is created, as shown in Figure 4.6. The function checks if the "agentNumHistory" map already exists in the system and caches the previous query result to improve performance. If it is the first query, the "callAll" method is invoked. To collect agents' path history, the tracking API involves three general steps: merge, propagate parent history, and sort the history by iteration time for each agent. To get agent number history, only the first step is included in the implementation. After merging all agents' itineraries, an aggregation is performed to assign values to the "agentNumHistory" map. This map is a shallow copy of the "agentNumHistory" map in MASSBase, which is the final return value. To ensure that there is only one "agentNumHistory" in a simulation, this step is crucial.

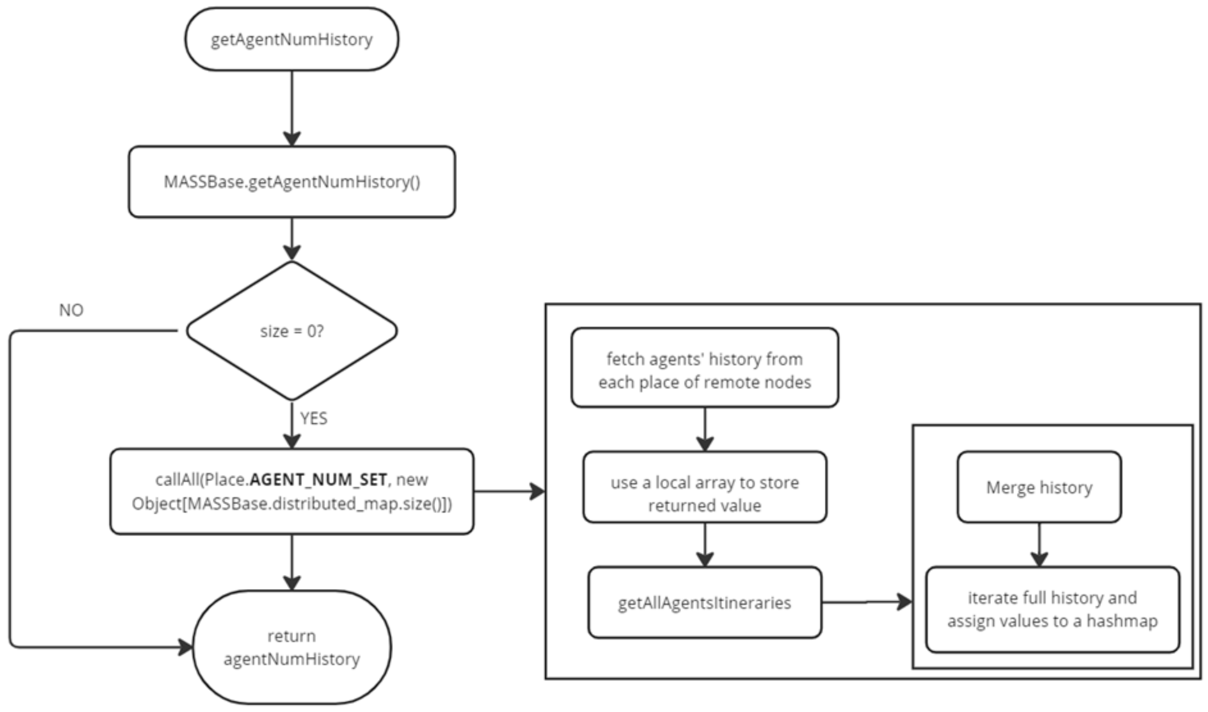

getAgentNumHistory
MASSBase.getAgentNumHistory()
size = 0?
NO
YES
callAll(Place.**AGENT_NUM_SET**, new Object[MASSBase.distributed_map.size()])
return agentNumHistory
fetch agents' history from each place of remote nodes
use a local array to store returned value
getAllAgentsItineraries
Merge history
iterate full history and assign values to a hashmap

*Figure 4.6: The Flowchart of Agent Number Tracking API*

In Cytoscape, an additional function is created to receive the returned "agentNumHistory" map. A new column called "Number_of_Agents" is added to the imported node table to store the number of agents. The required type is "String," indicating that four agents will be displayed as four dots (….) on that place. As shown in Figure 4.4, an extra table named "agentNumHistory" is created when importing agents. When a user changes the iteration time, the plugin queries the "agentNumHistory" table with the time, calculates the correct number of agents, transforms it into dots, and updates the "Number_of_Agents" field. This way, the user can see agents as dots when switching the default displaying label to "Number_of_Agents."

### b. Binary Tree Agents

To facilitate the visualization of agents in binary tree places, our implementation extended the agent history tracking API and agent number tracking API for all binary tree related classes. Notably, the global identifier for each binary tree place is a two-dimensional array, which is different from the graph agent visualization where each vertex has a unique global linear index. As illustrated in Figure 4.7, the first coordinate of the array identifies the compute node where the current binary tree place is located, while the second coordinate distinguishes different binary tree places within the same computing node.

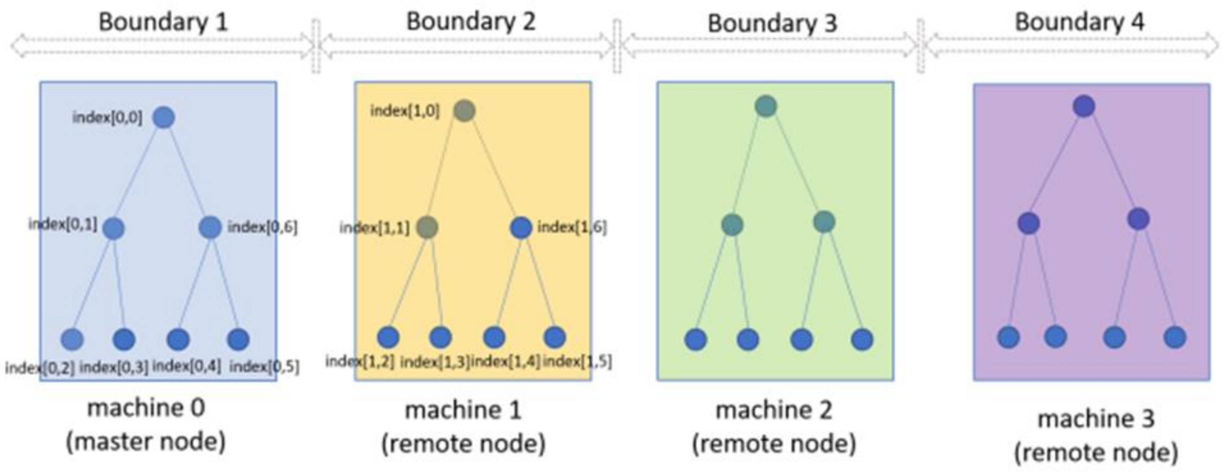


*Figure 4.7: Binary Tree Places Models*

However, the current agent tracking API only supports passing the linear index of a place, which poses a challenge in calculating the global linear index for each binary tree place across multiple computing nodes. To address this, we implement a new way of calculating the global linear index by creating a new two-dimensional array to store the size of binary tree nodes. The first coordinate represents the maximum number of computing nodes, and the second coordinate represents the maximum number of binary tree nodes among all computing nodes. To fetch the size of binary tree nodes from each remote node during initialization, we implemented a new message type

"BINARY_TREE_PLACES_NODES." We calculate the maximum size of all returned values, assign the value of host size to the first coordinate, and use the MatrixUtilities class's getLinearIndex function to calculate the global linear index. Finally, we send the constructed size array to all remote nodes and replace the size array used in the MatrixUtilities class, automatically calculating the linear index when an agent records its current location.

Moreover, we implement two callAll methods for binary tree places to retrieve agent history from remote places. The first method enables agent history tracking for all nodes, while the second method returns an object array to store the retrieved history data. The rest of the processing is similar to that of graphs.

## 4.3 Web GUI

The system architecture consists of three distinct components: the front-end, back-end, and data. React.js is used in the front-end to facilitate rendering and communication with the back-end through Axios requests. Vert.x serves as the web framework in the back-end and runs a Verticle that listens to port 8080. Upon starting InMASS and opening a JShell session, a Vert.x instance initializes and deploys a MyVerticle in the container using Vert.x's deploy method. The Verticle then creates an http server and multiple routers to process the requests and returns data to the front-end. The data server leverages memory space, where all required data can be found in memory variables. Once the simulation ends or the JShell session closes, all resources are released.

Figure 4.8 depicts the system's architecture diagram, where React.js manages front-end page rendering and user actions. Users interact with the front-end by sending Axios requests to the Vert.x server. Upon receiving the HTTP request, MyVerticle identifies the corresponding router to process it. The router queries the memory space to retrieve the data, converts it into a JSON string, and returns the string data to the front-end.

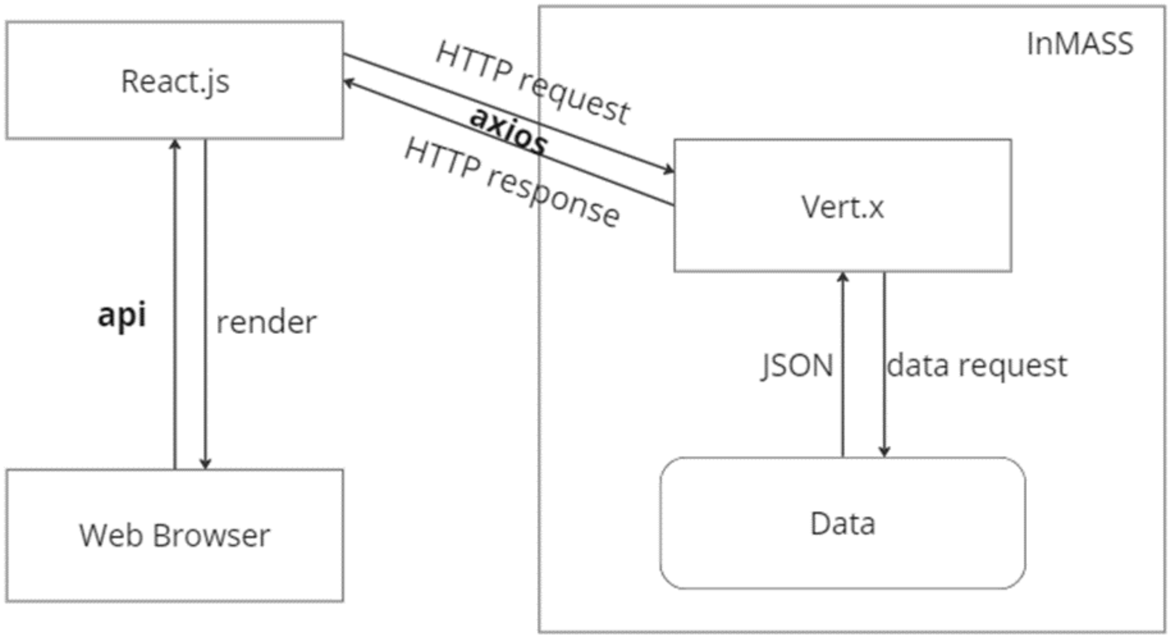


*Figure 4.8: The Architecture Diagram of the WEB GUI*

The WEB GUI uses three primary APIs. The first API, "/sync," is used to query the status of the entire cluster to determine if all nodes are in "running" status or if some nodes are in "terminate" status. We develop a new method called "getClusterStatus" to obtain this data. In this method, the master node initially sends ACK messages to remote nodes. If the remote node is still running, it responds to the master node with an ACK message. The master node then receives the returned information, encapsulates it into Status objects, and returns it as a list. The second API, "/calls," retrieves all functional calls made after a user checkpoints the simulation. The third API, "/rollback/:step," provides a "rollback" button after each functional call in the WEB GUI, enabling users to return to a specific step. The path's step variable is computed from the index of that specific call in the entire call list.

# Chapter 5 Evaluation

In this chapter, we (1) verified the following features of the re-engineered InMASS: Dynamic loading, checkpoint/rollback, and agent history tracking; (2) demonstrated enhanced Places visualization of binary tree, quad tree, and 2D continuous space in a distributed setting; enhanced Agents visualization of graph and binary tree; optimized MASS control panel; and (3) evaluated our Web GUI design.

## 5.1 Re-engineering Work

The project re-implements the following three features of the original InMASS such as dynamic loading, checkpoint/rollback, and agent tracking API; optimizes the code structure to enhance readability. The three features are tested on the Hermes computing cluster. In the cluster, this project mainly uses three machines (hermes01.uwb.edu ~ hermes03.uwb.edu).

### 5.1.1 Verification of Dynamic loading

To verify the dynamic loading feature, the project utilizes the QuickStart application. The QuickStart application creates 27 places in 3D dimensions, with each computing node managing nine of them. Nine agents are initially located on hermes01.uwb.edu and move along the x-axis, advancing one step per iteration. Only the Matrix and Nomad classes are initialized, each from Places and Agents, on hermes01.uwb.edu. The simulation terminates when all agents migrate to herme03.uwb.edu, demonstrating that the Agent and Place classes are available on the other two remote nodes, and that the remote inMASSClassLoaders are able to successfully load the classes.

### 5.1.2 Verification of Checkpoint and Rollback

To verify the checkpoint/rollback feature, the project still uses the QuickStart application discussed in section 5.1.1, while also adding more code and functions to the sample program. Figure 5.1 illustrates the entire simulation process, which includes the checkpointing of the system immediately after creating agents and places. Subsequently, all agents execute two migrations from hermes01.uwb.edu to hermes03.uwb.edu. After the migrations, the rollback function is utilized to revert the system state to time 0, when the agents and places are initially created. The MASS resets all agents back to herme01.uwb.edu before calling for another round of agent migration, this time from herme01.uwb.edu to hermes02.wub.edu. The success of this migration serves as a verification of the checkpoint/rollback feature.

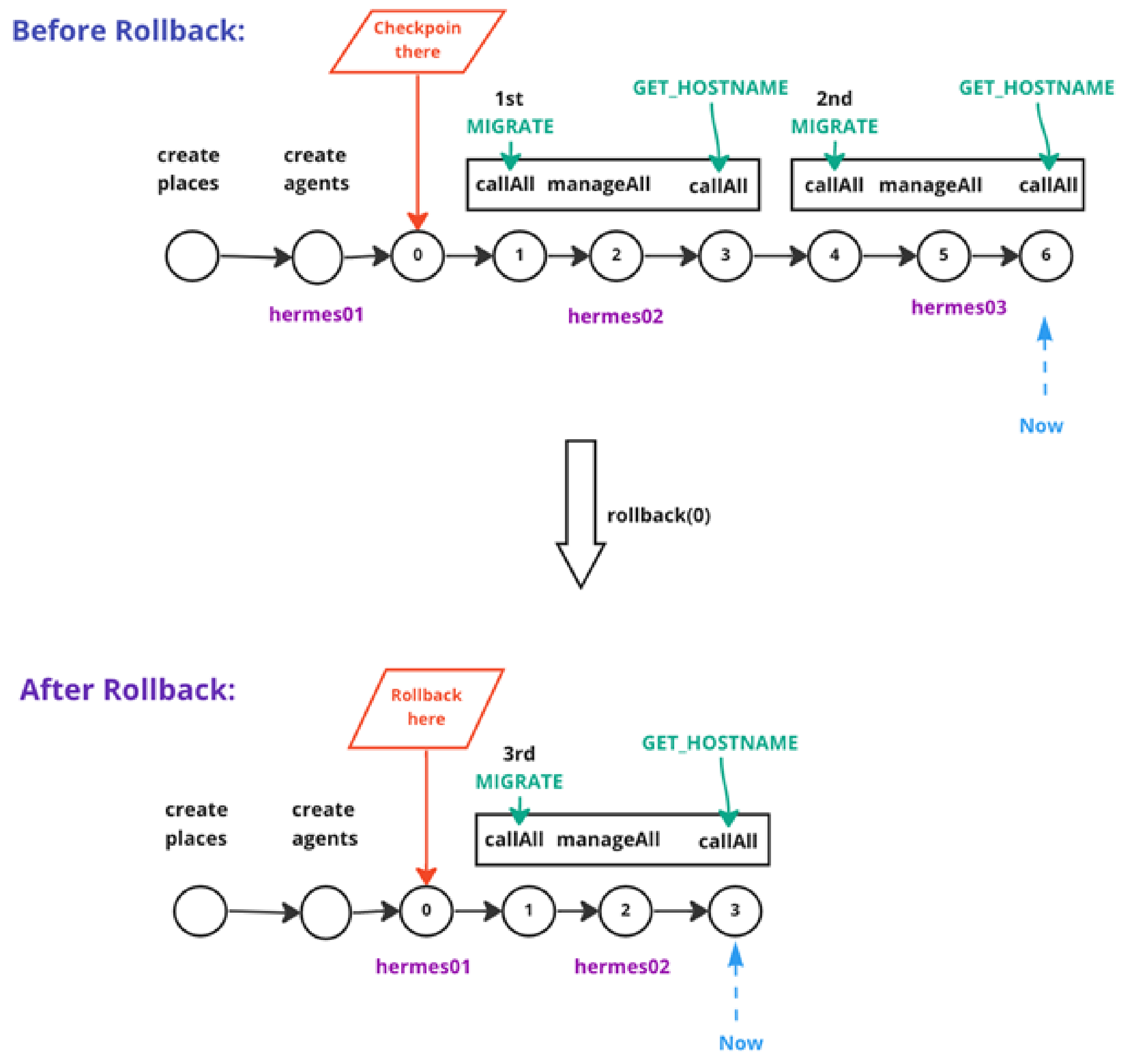


*Figure 5.1: The whole process of quickStart Application (Rollback version)*

### 5.1.3 Verification of Agent History Tracking

To verify this feature, the project uses the triangle counting benchmark application, as shown in Figure 5.2, where the visualization of imported nodes indicates only two triangles in the graph.

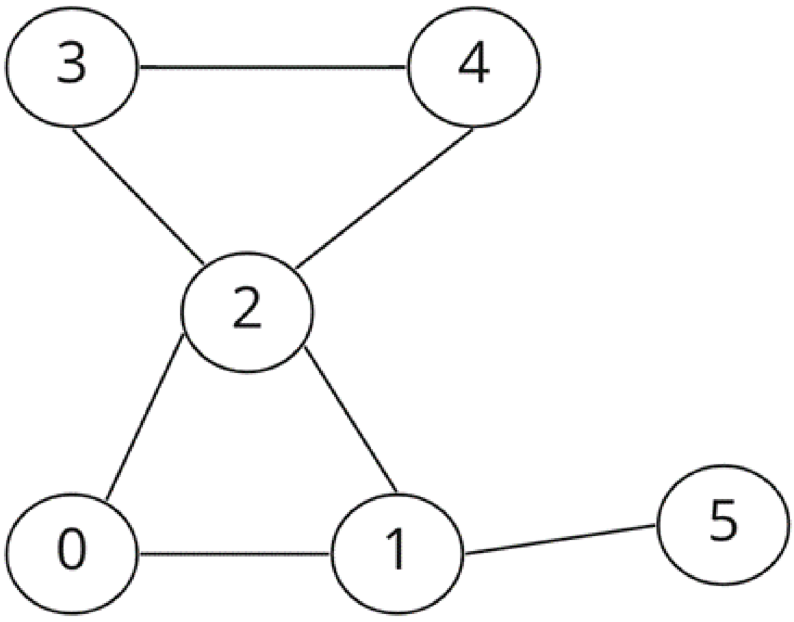
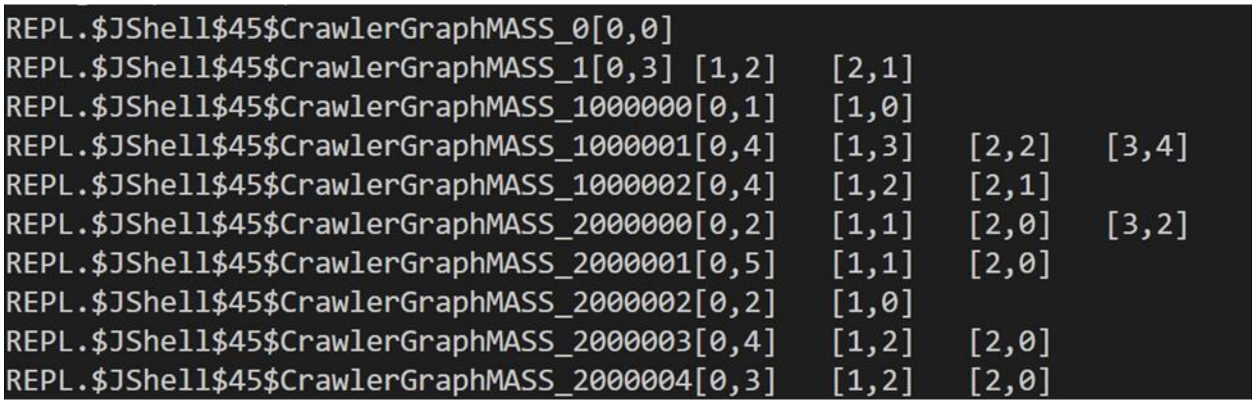


*Figure 5.2: Graph Nodes' Visualization in Triangle Counting Benchmark Application*

In this application, agents are initialized on each vertex and move three steps. During the first two steps, agents migrate only to neighboring nodes with lower indices. In the final step, agents try to migrate back to their original vertex. The number of agents that complete all steps represents the number of triangles. Figure 5.3 presents the printed result of all agents' history, where only two agents (agent 1000001 and agent 2000000) travel back to the original vertex, with paths of [0,4] [1,3] [2,2] [3,4] and [0,2] [1,1] [2,0] [3,2], respectively. These results confirm the existence of two triangles in the graph and validate the functionality of the agent history tracking API.



```
REPL.$JShell$45$CrawlerGraphMASS_0[0,0]
REPL.$JShell$45$CrawlerGraphMASS_1[0,3] [1,2]   [2,1]
REPL.$JShell$45$CrawlerGraphMASS_1000000[0,1]   [1,0]
REPL.$JShell$45$CrawlerGraphMASS_1000001[0,4]   [1,3]   [2,2]   [3,4]
REPL.$JShell$45$CrawlerGraphMASS_1000002[0,4]   [1,2]   [2,1]
REPL.$JShell$45$CrawlerGraphMASS_2000000[0,2]   [1,1]   [2,0]   [3,2]
REPL.$JShell$45$CrawlerGraphMASS_2000001[0,5]   [1,1]   [2,0]
REPL.$JShell$45$CrawlerGraphMASS_2000002[0,2]   [1,0]
REPL.$JShell$45$CrawlerGraphMASS_2000003[0,4]   [1,2]   [2,0]
REPL.$JShell$45$CrawlerGraphMASS_2000004[0,3]   [1,2]   [2,0]
```

*Figure 5.3: Printed traces of all created agents*

## 5.2 Places and Agents Visualization

In this section, we present the visualization of places and agents as well as the optimized MASS control panel. The visualization is implemented under a distributed setting (using hermes01.uwb.edu ~ hermes03.uwb.edu) and is verified using various benchmark applications. The following subsections describe the visualization of different types of places and agents.

### 5.2.1 Places Visualization

#### a. Binary Tree Places

The visualization of binary tree places is implemented and verified using the RangeSearch benchmark application. The implementation is tested on three computing nodes (hermes01.uwb.edu ~ hermes03.uwb.edu). The partition algorithm of Binary Tree Places is used to divide 16 points into 8 binary tree places on hermes01.uwb.edu, another 4 points on hermes02.uwb.edu, and the remaining 4 points on hermes03.uwb.edu. Figure 5.4 shows the visualization result, where each binary tree represents a computing node.

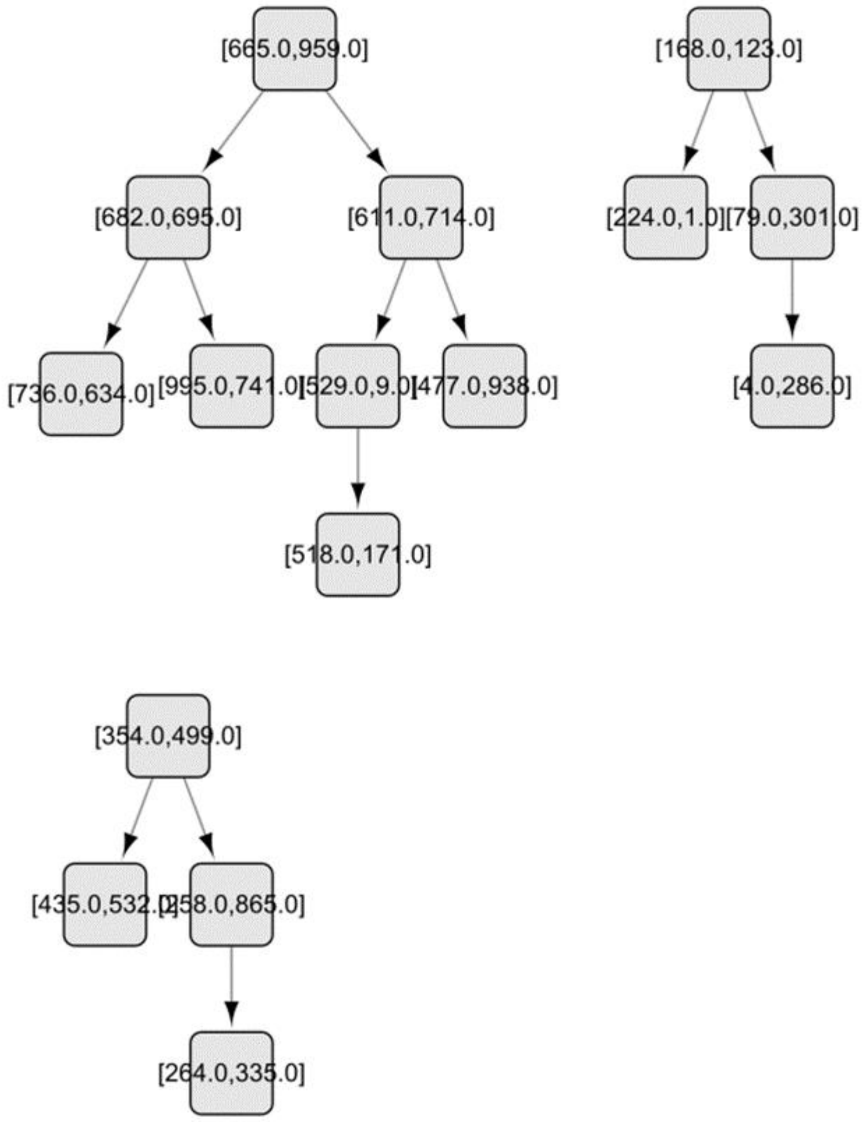


*Figure 5.4: Binary Tree Places Visualization*

### b. Quad Tree Places

The visualization of quad tree places is implemented and verified using the CPP benchmark application. The implementation is tested on two computing nodes (hermes01.uwb.edu ~ hermes02.uwb.edu). The partition algorithm divides all 16 points into two groups, where each group constructs a single quad tree. Figure 5.5 shows the visualization result, where the quad tree on the top stores 9 points, and the quad tree below stores the other 7 points.

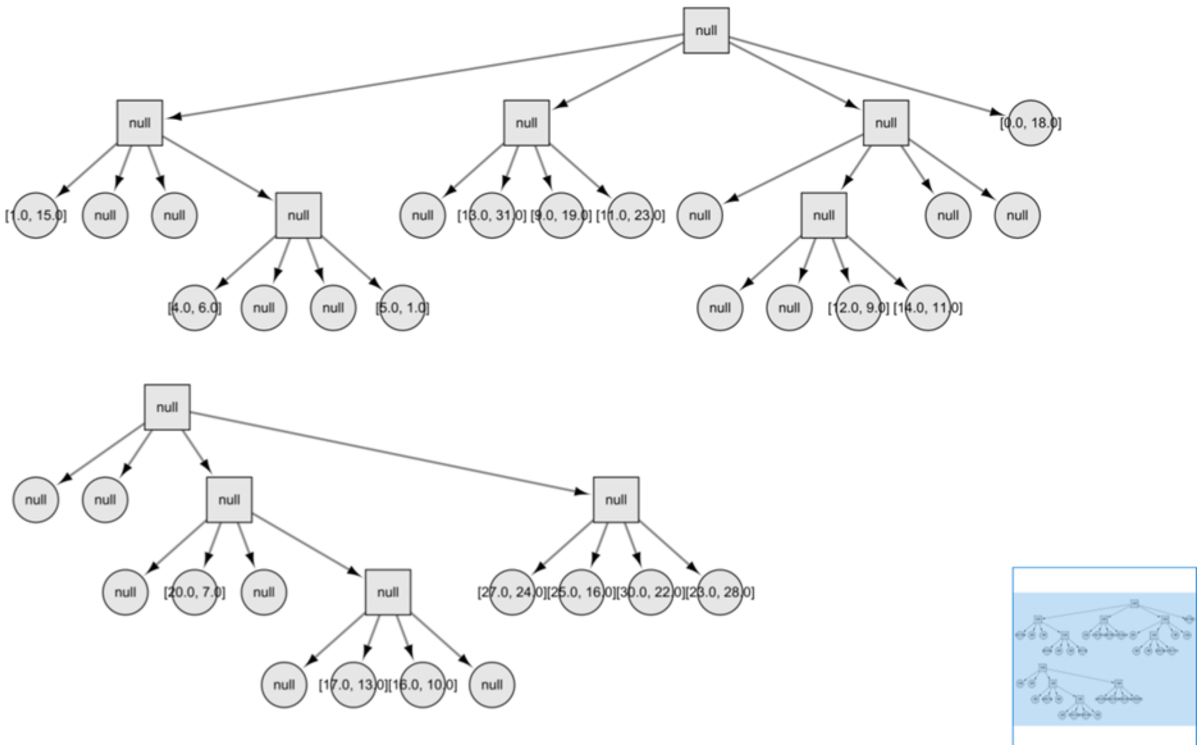


*Figure 5.5: Quad Tree Places Visualization*

### c. 2D Continuous Space

The visualization of 2D continuous space is implemented using the same data as quad tree. We use two machines (hermes01.uwb.edu ~ hermes02.uwb.edu) and pre-calculate the position of each 2D space to avoid overlapping. The width and length of the previously visualized 2D space are used to calculate the position of the next visualized 2D space. Figure 5.6 shows the final visualization result, where the visualized 2D spaces are arranged in anti-diagonal positions, and each place is shown using the global tree index.

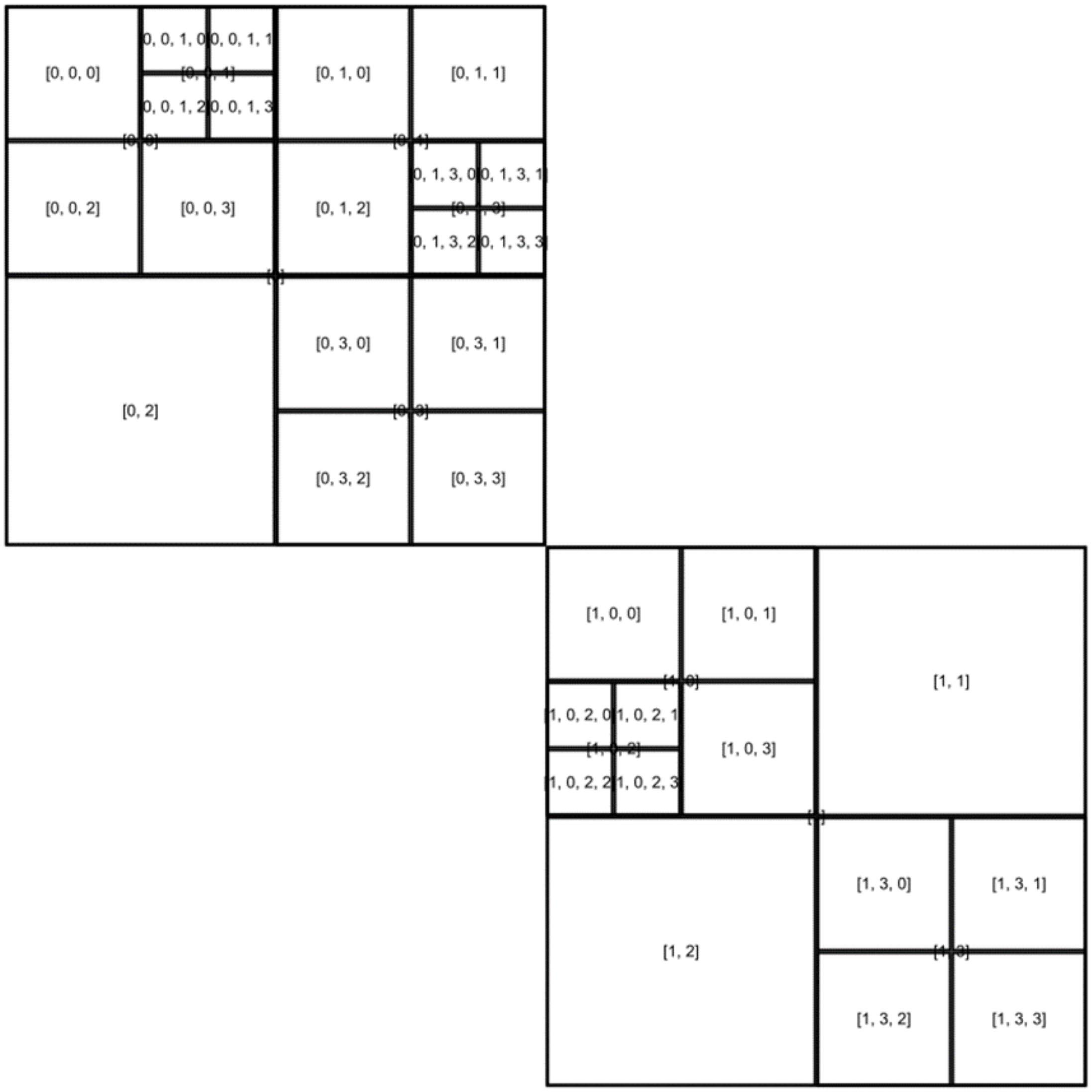


*Figure 5.6: 2D Continuous Space Places Visualization*

### 5.2.2 Agents Visualization

#### a. Graph Agents

This project visualizes the agents in graphs using the TriangleCounting application. The initial distribution of agents is shown in Figure 5.7, where each vertex has only one agent. The agents then migrate to the neighboring vertices with a lower index. Figure 5.8 shows the agents' visualization after the simulation is finished, where only two agents reside on vertex 2 and 4. The visualization of agents in graphs is verified by the agents' trace shown in section 5.1.3.

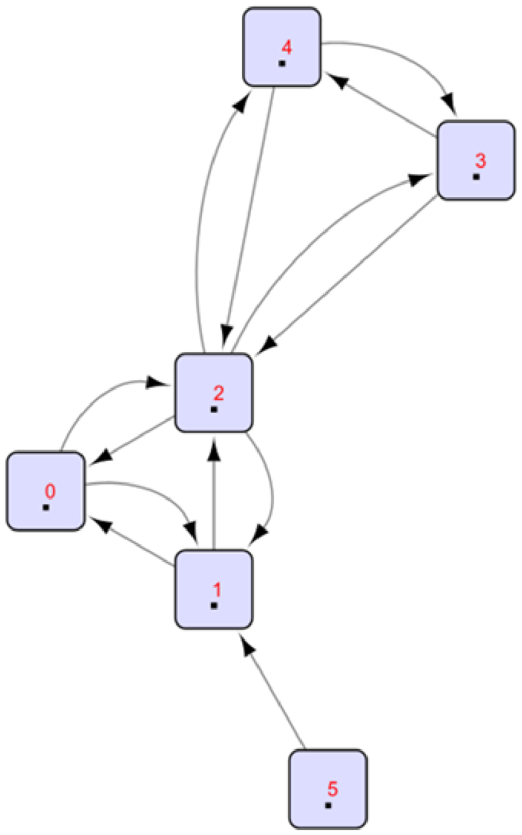


*Figure 5.7: Agents visualization at the beginning of the TriangleCounting application*

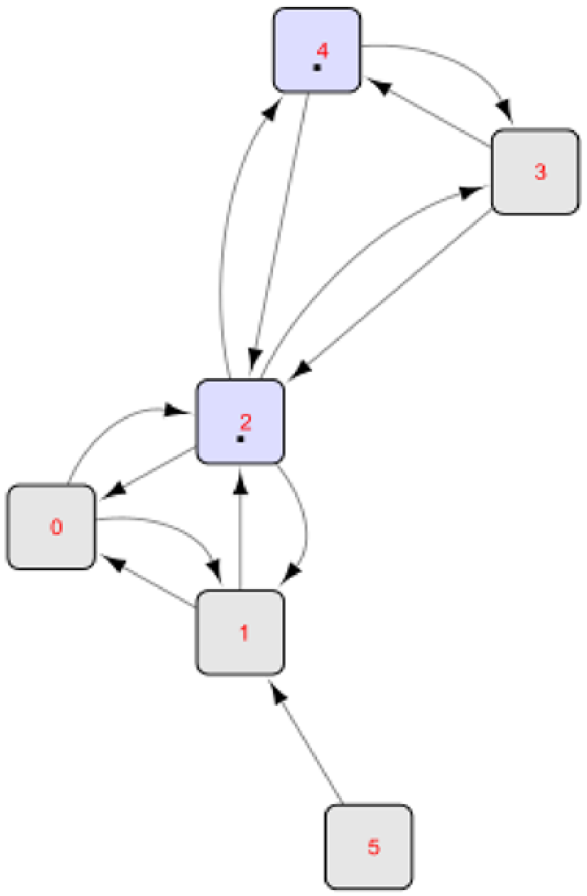


*Figure 5.8: Agents visualization at the end of the TriangleCounting application*

### b. Binary Tree Agents

Section 5.2.1 discussed the use of the RangeSearch application to visualize places in this project. As for agents visualization, figure 5.9 presents the initial visualization result. It shows that two agents have been initialized on the root node of each binary tree. The red coordinate denotes the boundary of the current binary tree location. To construct the binary tree, the algorithm divides the first level using the x coordinate, the second level using the y coordinate, and the third level using the x coordinate again. The entire binary tree is divided alternatively in x and y coordinates.

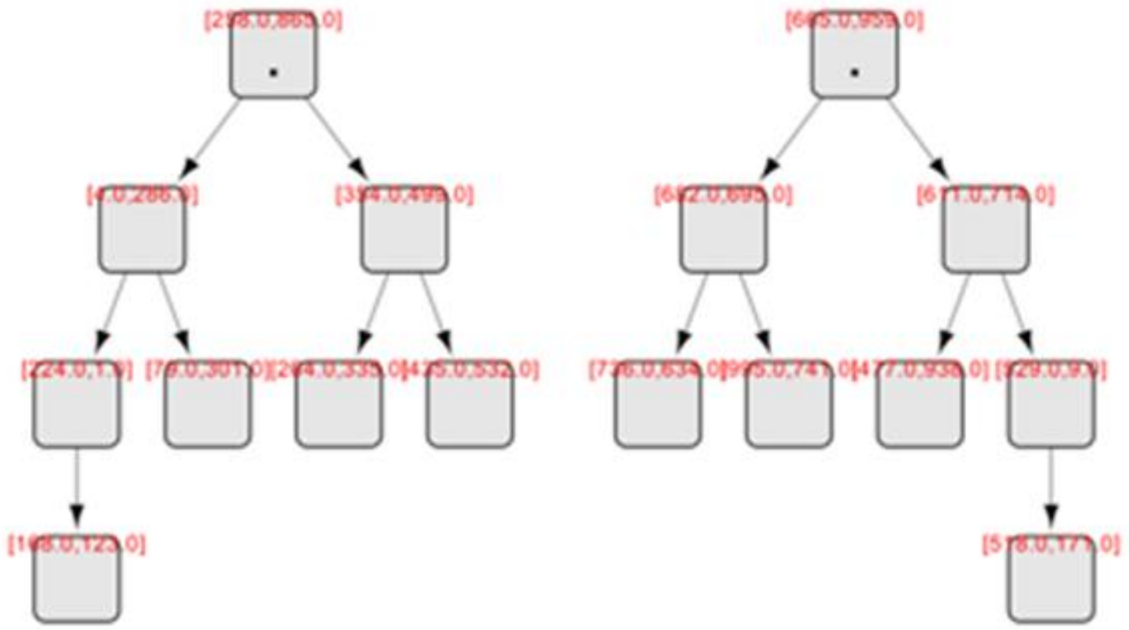


Figure 5.9: Agents visualization at the beginning of the RangeSearch application

Figure 5.10 presents the visualization result after the search has stopped. The binary tree on the right only contains one agent, indicating that the search stopped earlier in the left binary tree as the remaining nodes were outside the search boundary, and all agents within that tree had terminated themselves.

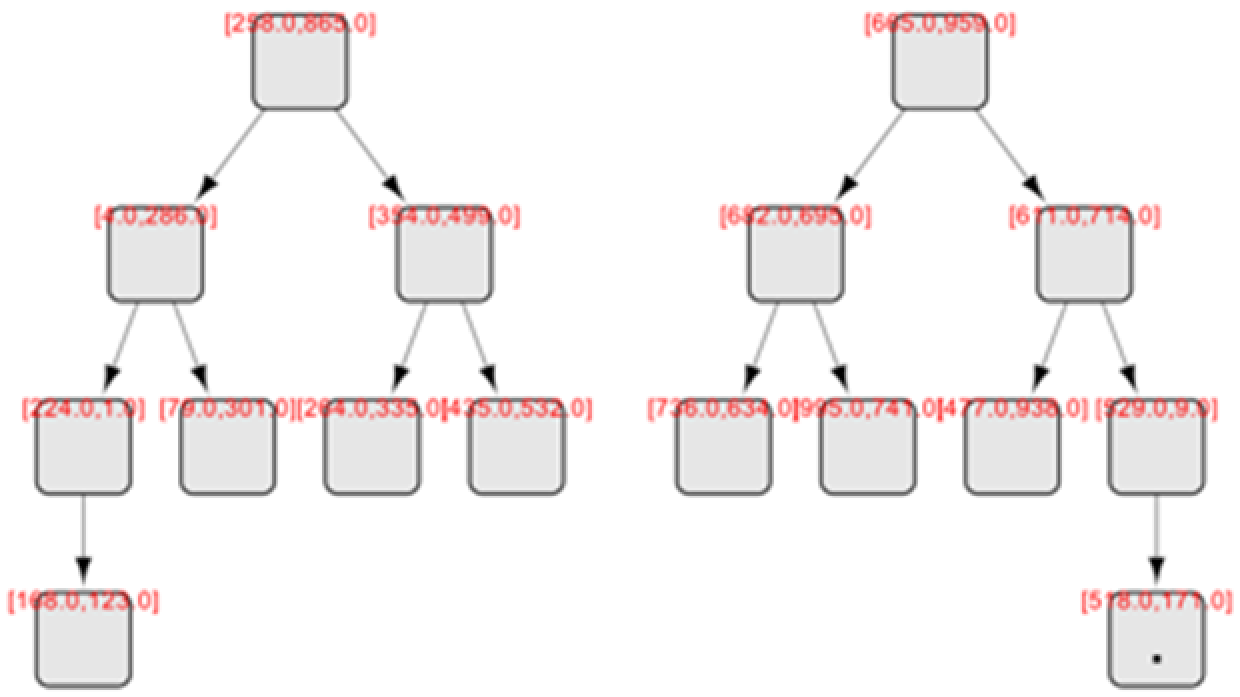


*Figure 5.10: Agents visualization at the end of the RangeSearch application*

### c. Latency

In addition to the original dataset, this project uses 1000 and 10000 points to test the visualization of binary tree agents. Because the number of agents at each moment is pre-calculated before passing to Cytoscape, and because MASS has efficient messaging, the latency from agents on different nodes to visualization is negligible. But the rendering time of places in Cytoscape increases with the number of created places. On the contrary, the rendering of agents in each place is very fast and the users can’t see significant delays. However, when the number of places is large, the current screen cannot accommodate all the visualized places at a size that can be seen clearly. This may affect the final effect of the agent visualization.

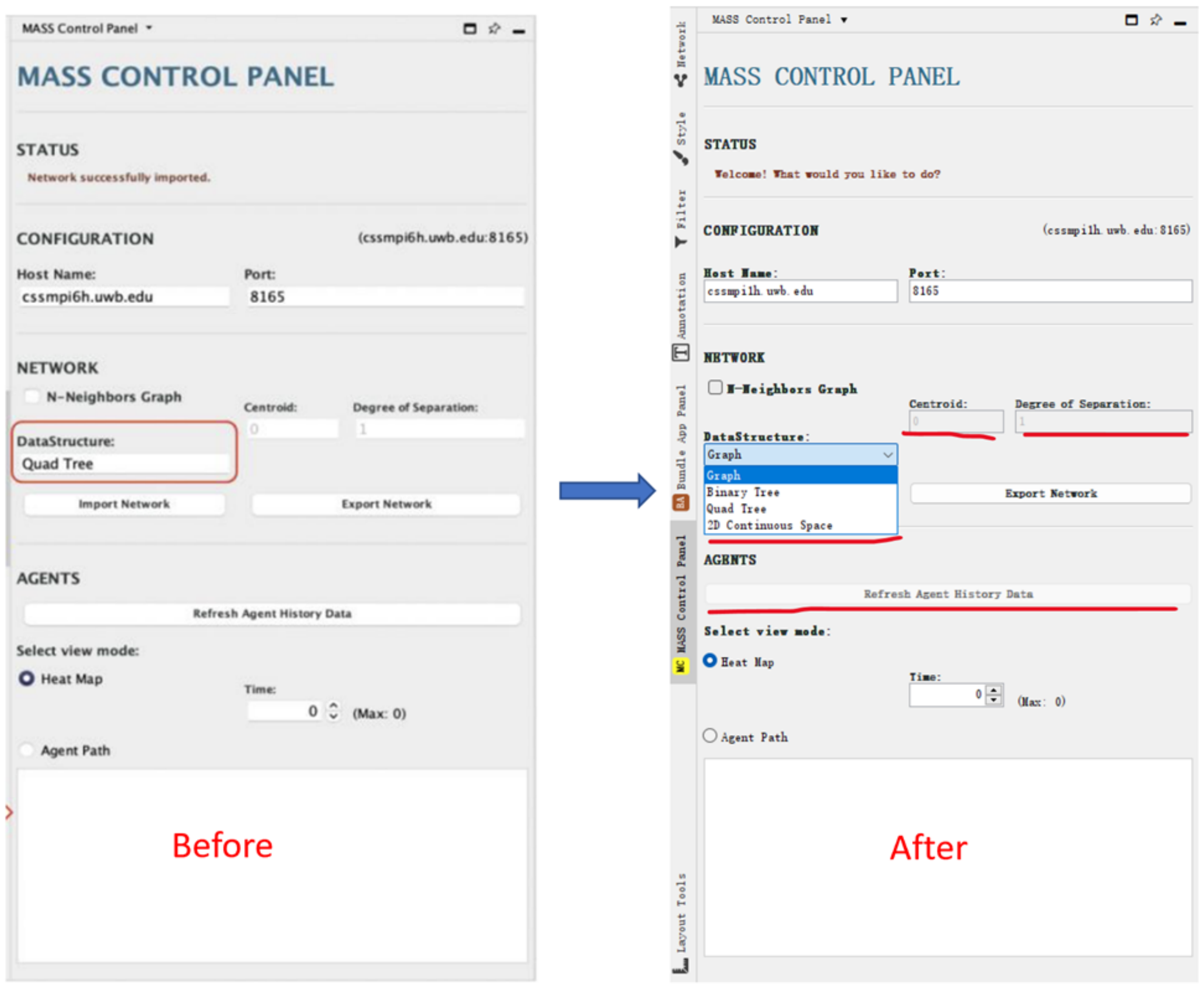
### 5.2.3 MASS Control Panel Optimization

Figure 5.11 compares the optimized panel, which includes three critical enhancements aimed at enhancing the overall user experience. First, the data structure input box has been replaced with a select box, offering users a choice of four supported data structures: Graph, Binary Tree, Quad Tree, and 2D Continuous Space. This upgrade simplifies the selection process, allowing users to choose their preferred data structure with ease.

Second, some parameters on the panel, such as N-Neighbors Graph, centroid, and Degree of Separation, are only applicable to graph networks. To prevent confusion and streamline the user experience, this project hides these options when users import a non-graph network.

Finally, this project optimizes the import agents operation logic. As discussed in section 4.2.2, MASS-Cytoscape performs two functions when importing agents' history data: importing agents' history path data and importing the number of agents' history data. However, this operation can only function after importing network places. To optimize the button logic and streamline the user experience, this project has ensured that the button is only activated after importing the network.

By implementing these enhancements, this optimized panel ensures that users can effortlessly navigate and utilize the features offered by MASS-Cytoscape.



Before:

Network successfully imported.

CONFIGURATION (cssmpi6h.uwb.edu:8165)

Host Name: cssmpi6h.uwb.edu Port: 8165

Centroid: 0 Degree of Separation: 1

DataStructure: Quad Tree

Time: 0 (Max: 0)

After:

Welcome! What would you like to do?

CONFIGURATION (cssmpi1h.uwb.edu:8165)

Host Name: cssmpi1h.uwb.edu Port: 8165

Centroid: 0 Degree of Separation: 1

DataStructure: Graph

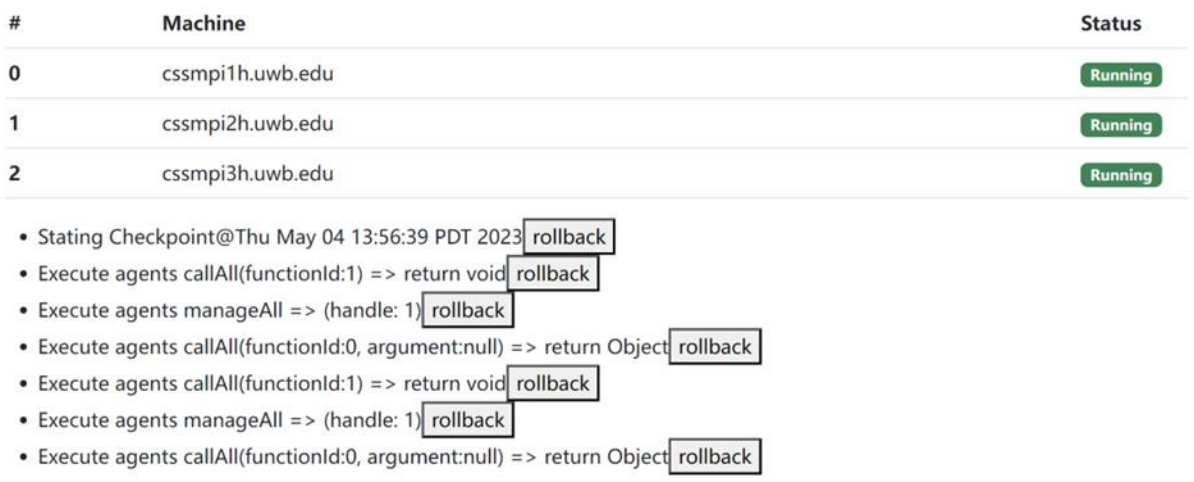
Time: 0 (Max: 0)

*Figure 5.11: Optimized MASS Control Panel*

## 5.3 Web GUI

Figure 5.12 displays the main page of the WEB GUI, which provides users with critical insights into their cluster's performance. The top section of the interface contains the cluster monitoring panel, where users can obtain real-time updates on the status of each machine in the cluster. The machines' status is categorized into two groups: running and terminated. In the event of an unexpected shutdown, the machine's status will transition from running to terminated.

The panel below displays the functional calls that users make to the cluster. This feature is an extension of the checkpoint/rollback feature, which enables users to view all the calls with their respective function IDs. This feature assists users in identifying the specific step they need to roll back to with ease. Additionally, the rollback button is designed to streamline users' operations within InMASS.



| # | Machine | Status |
|---|---|---|
| 0 | cssmpi1h.uwb.edu | Running |
| 1 | cssmpi2h.uwb.edu | Running |
| 2 | cssmpi3h.uwb.edu | Running |

- Stating Checkpoint@Thu May 04 13:56:39 PDT 2023 rollback
- Execute agents callAll(functionId:1) => return void rollback
- Execute agents manageAll => (handle: 1) rollback
- Execute agents callAll(functionId:0, argument:null) => return Object rollback
- Execute agents callAll(functionId:1) => return void rollback
- Execute agents manageAll => (handle: 1) rollback
- Execute agents callAll(functionId:0, argument:null) => return Object rollback

Figure 5.12: All-in-one WEB GUI

# Chapter 6 Conclusion

In ABM simulations, the interaction between their underlying system and its users can significantly increase usability and controllability. In this project, we enhanced the design of InMASS to make it compatible with the latest MASS, which is critical for future InMASS developers to stay aligned with MASS developers. The project also implemented visualization of Places in multiple computing nodes, which underlines the distributed nature of our MASS system. The improvement of Agent visualization makes it possible to see the actual number of agents residing on each place and facilitates users in monitoring the distribution of agents at each iteration during the whole simulation. The last implementation of this project is the Web GUI, which allows users to monitor the state of the whole distributed cluster and simplify the rollback operation.

This project can be optimized and further expanded in the following three aspects:

1. Optimize the WEB GUI by designing a built-in terminal. Currently, InMASS users run the simulation using the console. The WEB GUI only addresses the monitoring problem. In that case, it would be better to combine them together. So, users can run the simulation and monitor the cluster all in a single WEB interface.
2. Extend agents visualization for quad tree and 2D continuous space in Cytoscape. Current visualization is extended for both graph and binary tree using triangle counting and range search benmark application. The next step should be visualizing quad tree and 2D space agents using finding the closest pair of points and voronoi diagram benchmark application.
3. Develop a MASS's own visualization tool. If Cytoscape can't visualize quad tree or 2D space agents, or the visualization result is not ideal. It's necessary to develop a visualization tool for MASS only.

4. Perform usability tests. The current users of InMASS are only limited to DSLab members. To get accurate feedback and remove bias, it’s better to collect feedback from more than one hundred users outside of DSLab groups. The questions are:
    a. What features do you think are necessary for visualization?
    b. What do you think of the startup process for visualization in MASS compared with other ABM simulators in terms of difficulty and latency?
    c. Does MASS support all data structures that you are interested in?
    d. As for places visualization, are these places correctly located in the corresponding positions?
    e. Compare the latency of visualizing both places and agents.
    f. As for agents visualization, compare the appearance of agents.
    g. Are agents’ movement static or dynamic?
    h. Is it possible to observe or trace single agents' movement?
    i. Can you know the agents' moving directions?
    j. Compare the MASS control panel with other ABM simulators’ editor interface.

# BIBLIOGRAPHY

[1] Grimm, Volker, and Steven F. Railsback. Individual-based modeling and ecology. Princeton university press, 2005.

[2] Gustafsson, Leif; Sternad, Mikael (2010). "Consistent micro, macro, and state-based population modelling". Mathematical Biosciences. 225 (2): 94–107. doi:10.1016/j.mbs.2010.02.003. PMID 20171974.

[3] Hu, J.; Bhowmick, P.; Jang, I.; Arvin, F.; Lanzon, A., "A Decentralized Cluster Formation Containment Framework for Multirobot Systems" IEEE Transactions on Robotics, 2021.

[4] Hu, J.; Turgut, A.; Lennox, B.; Arvin, F., "Robust Formation Coordination of Robot Swarms with Nonlinear Dynamics and Unknown Disturbances: Design and Experiments" IEEE Transactions on Circuits and Systems II: Express Briefs, 2021.

[5] Hu, J.; Bhowmick, P.; Lanzon, A., "Group Coordinated Control of Networked Mobile Robots with Applications to Object Transportation" IEEE Transactions on Vehicular Technology, 2021.

[6] Wiering, M. A. (2000). "Multi-agent reinforcement learning for traffic light control". Machine Learning: Proceedings of the Seventeenth International Conference (Icml'2000): 1151–1158. hdl:1874/20827.

[7] Niazi, Muaz; Hussain, Amir (2011). "Agent-based Computing from Multi-agent Systems to Agent-Based Models: A Visual Survey" (PDF). Scientometrics. 89 (2): 479–499. arXiv:1708.05872. doi:10.1007/s11192-011-0468-9. hdl:1893/3378. S2CID 17934527. Archived from the original (PDF) on October 12, 2013.

[8] M.Fukuda, Parallel-Computing Library for Multi-Agent Spatial Simulation in Java, 2010

[9] DSLab, "MASS Java Manual," 2016. [Online]. Available: https://depts.washington.edu/dslab/MASS/docs/MASS%20Java%20Technical%20Manual.pdf

[10] Apache Spark, “Spark Web UI – Understanding Spark Execution,” 2023. [Online]. Available: https://sparkbyexamples.com/spark/spark-web-ui-understanding/

[11] Cloudera, “Hue - The open-source SQL Assistant for Data Warehouses,” 2023. [Online]. Available: https://gethue.com//posts/

[12] Shannon P, Markiel A, Ozier O, et al. Cytoscape: a software environment for integrated models of biomolecular interaction networks[J]. Genome research, 2003, 13(11): 2498-2504.

[13] North, Michael J, Nicholson T Collier, Jonathan Ozik, Eric R Tatara, Charles M Macal, Mark Bragen, and Pam Sydelko. 2013. “Complex Adaptive Systems Modeling with Repast Simphony.” Complex Adaptive Systems Modeling 1 (1): 3. https://doi.org/10.1186/2194-3206-1-3.

[14] Wilensky, U. & Stroup, W., 1999. HubNet. http://ccl.northwestern.edu/netlogo/hubnet.html. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[15] Abelson, Hal, Nat Goodman, and Lee Rudolph. "Logo manual." (1974).

[16] Luke, Sean, et al. "MASON: A Java multi-agent simulation library." Proceedings of Agent 2003 Conference on Challenges in Social Simulation. Vol. 9. No. 9. 2003.

[17] L. Chin, D. Worth, C. Greenough, S. Coakley, M. Holcombe and M. Kiran, "FLAME: an approach to the parallelisation of agent-based applications", 2012.

[18] Nasser. Alghamdi, "Supporting Interactive Computing Features for MASS Library: Rollback and Monitoring System" Depts.washington.edu, 2020. [Online]. Available: http://depts.washington.edu/dslab/MASS/reports/NasserWhitePaper_sp20.pdf

[19] Cheng F, Cheng F. jshell[J]. Exploring Java 9: Build Modularized Applications in Java, 2018: 57-65.

[20] Daniel. Blashaw, "Interactive Environment to Support Agent-Based Graph Programming in MASS Java," Depts.washington.edu, 2021. [Online]. Available: http://depts.washington.edu/dslab/MASS/reports/DanielBlashaw_whitepaper.pdf

[21] Gilroy J, Paronyan S, Acoltzi J, et al. Agent-navigable dynamic graph construction and visualization over distributed memory[C]//2020 IEEE International Conference on Big Data (Big Data). IEEE, 2020: 2957-2966.

[22] Y. Guo, "Construction of Agent-navigable Data Structure from Input Files", Depts.washington.edu, 2021. [Online]. Available: https://depts.washington.edu/dslab/MASS/reports/YunaGuo_whitepaper.pdf.

[23] Tianhui. Nie, "Visualization of 2D Continuous Spaces and Trees in MASS Java," Depts.washington.edu, 2022. [Online]. Available: http://depts.washington.edu/dslab/MASS/reports/TianhuiNie_whitepaper.pdf

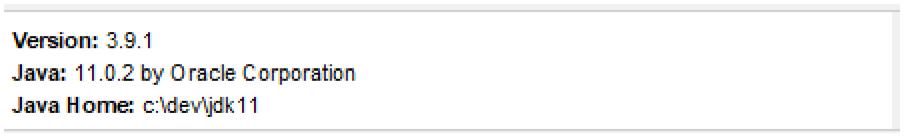
[24] Fukuda M, Gordon C, Mert U, et al. An agent-based computational framework for distributed data analysis[J]. Computer, 2020, 53(3): 16-25.

# Appendix A: Developer Guide

## A1. Cytoscape Installation

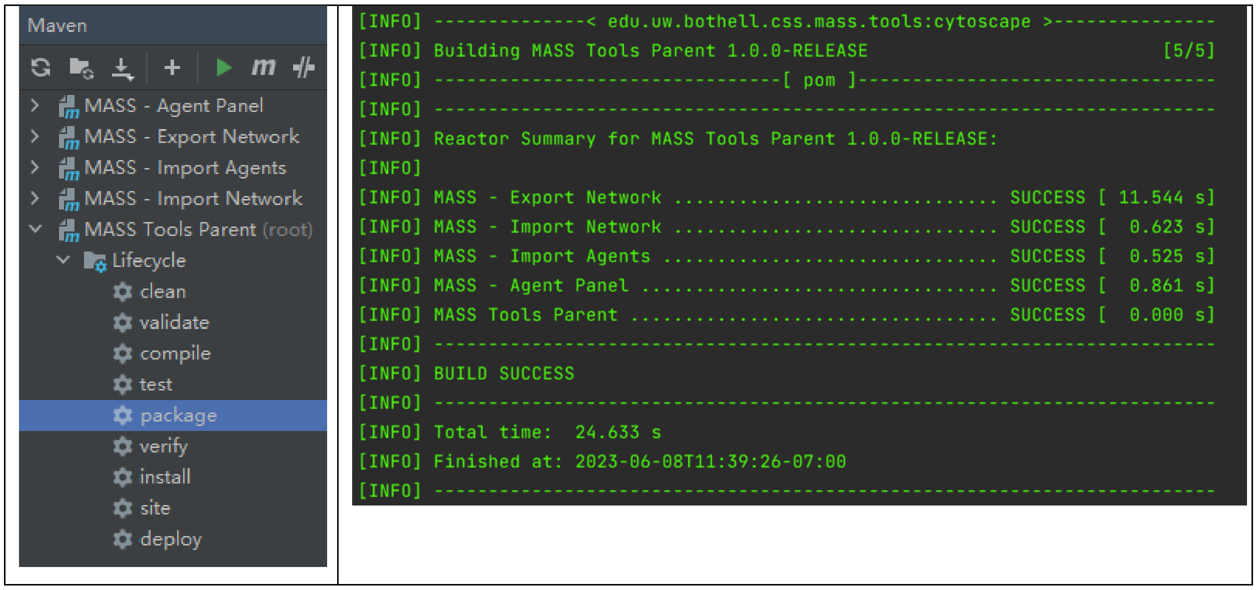
The following installation steps are base on Windows 11, but it should be similar on other platforms. If you want to check the instructions based on Mac OS platform. Please refer to the developer guide of Tianhui Nie's whitepaper.

1. **Download JDK 11 and set JAVA_HOME environment**.
2. **Download and install Cytoscape**. Download Cytoscape from its official website. In this project, we use Cytoscape verison 3.9.1.



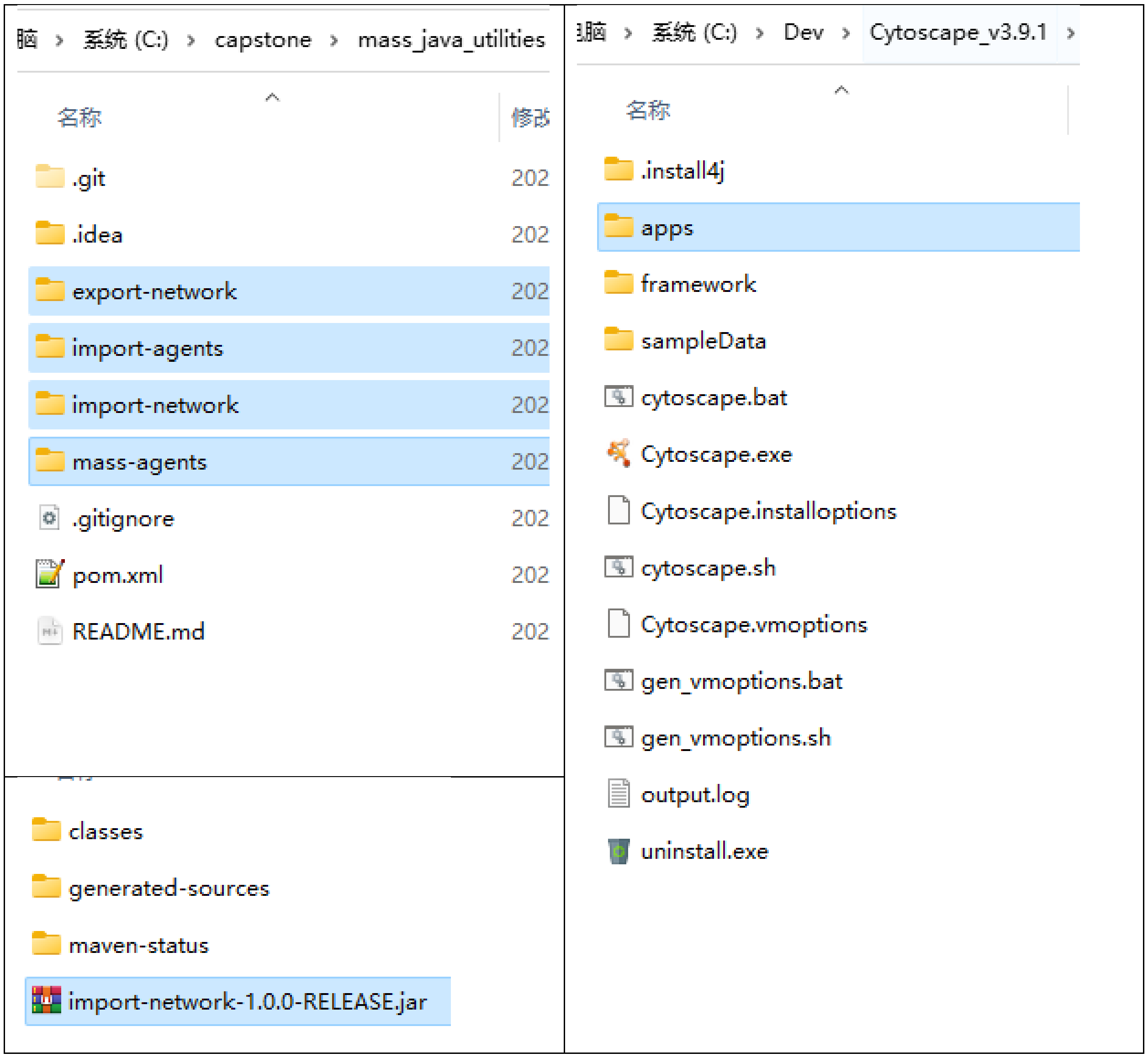
**Version:** 3.9.1
**Java:** 11.0.2 by Oracle Corporation
**Java Home:** c:\dev\jdk11

3. **Download MASS-Cytoscape plugins from Bitbucket**. Please download the mass_java_utilities repo from Bitbucket. This project is based on **yifei/dev** branch
4. **Compile and install MASS-Cytoscape plugins**.
   a. Use IDEA or terminal to compile the utilities package

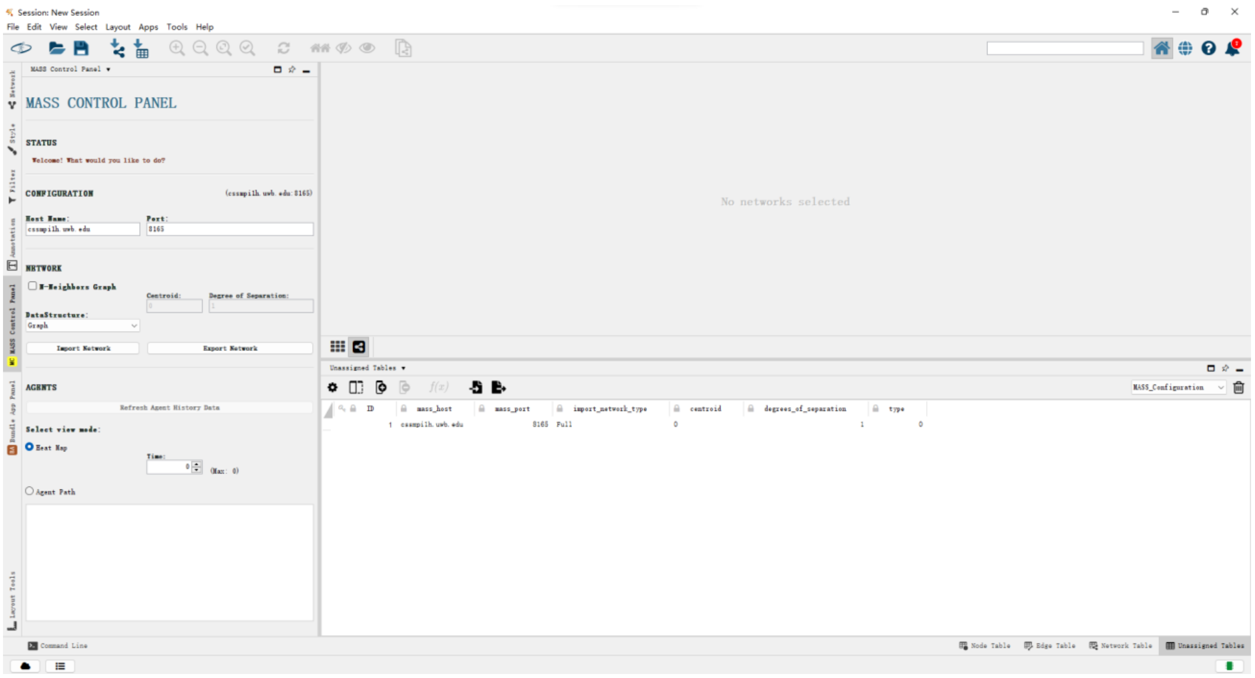


```
[INFO] -------------< edu.uw.bothell.css.mass.tools:cytoscape >---------------
[INFO] Building MASS Tools Parent 1.0.0-RELEASE                          [5/5]
[INFO] --------------------------------[ pom ]---------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for MASS Tools Parent 1.0.0-RELEASE:
[INFO]
[INFO] MASS - Export Network .............................. SUCCESS [ 11.544 s]
[INFO] MASS - Import Network .............................. SUCCESS [  0.623 s]
[INFO] MASS - Import Agents ............................... SUCCESS [  0.525 s]
[INFO] MASS - Agent Panel ................................. SUCCESS [  0.861 s]
[INFO] MASS Tools Parent .................................. SUCCESS [  0.000 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  24.633 s
[INFO] Finished at: 2023-06-08T11:39:26-07:00
[INFO] ------------------------------------------------------------------------
```

   b. Move compiled jar files to Cytoscape app folder



c. Reopen Cytoscape. The MASS plugins have been installed to the Cytoscape. MASS control panel can be found.



MASS CONTROL PANEL

STATUS

Welcome! What would you like to do?

CONFIGURATION (cssmpi1h.uwb.edu:8165)

Host Name: cssmpi1h.uwb.edu
Port: 8165

NETWORK

☐ N-Neighbors Graph

Centroid: 0
Degree of Separation: 1

DataStructure: Graph

Import Network | Export Network

AGENTS

Refresh Agent History Data

Select view mode:

● Heat Map

Time: 0 (Max: 0)

○ Agent Path

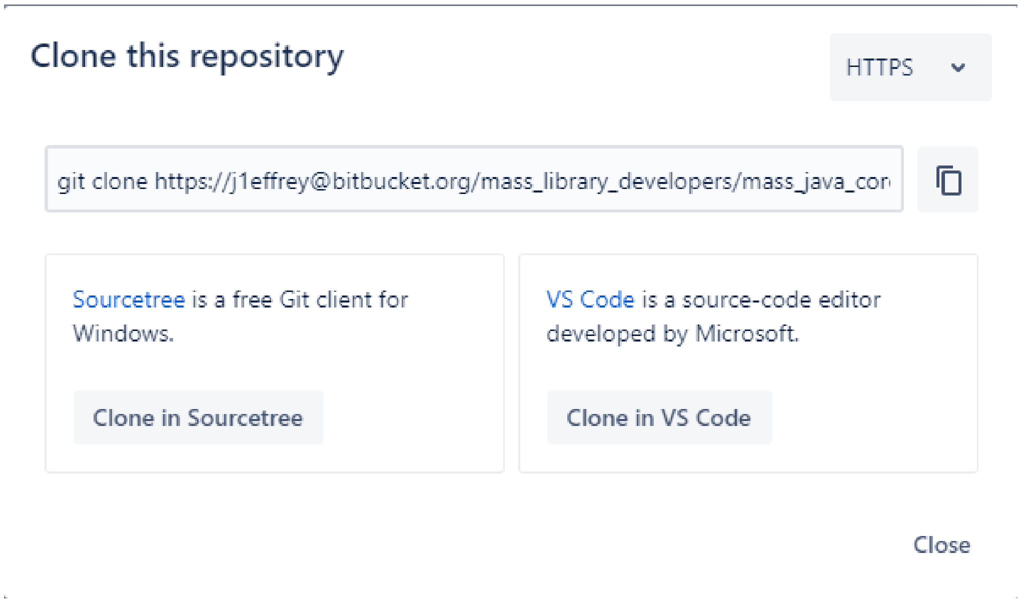
No networks selected

MASS_Configuration

| ID | mass_host | mass_port | import_network_type | centroid | degrees_of_separation | type |
|---|---|---|---|---|---|---|
| 1 | cssmpi1h.uwb.edu | 8165 | Full | 0 | 1 | 0 |

## A2. Compile and Run InMASS

1. **Download the MASS library**. Download the mass_java_core package from BitBucket. In this project, we finished the work using **InMASS** branch.
2. **Clone the repo** on a UWB virtual machine. Or use the scp command to copy the code from local to virtual machine.

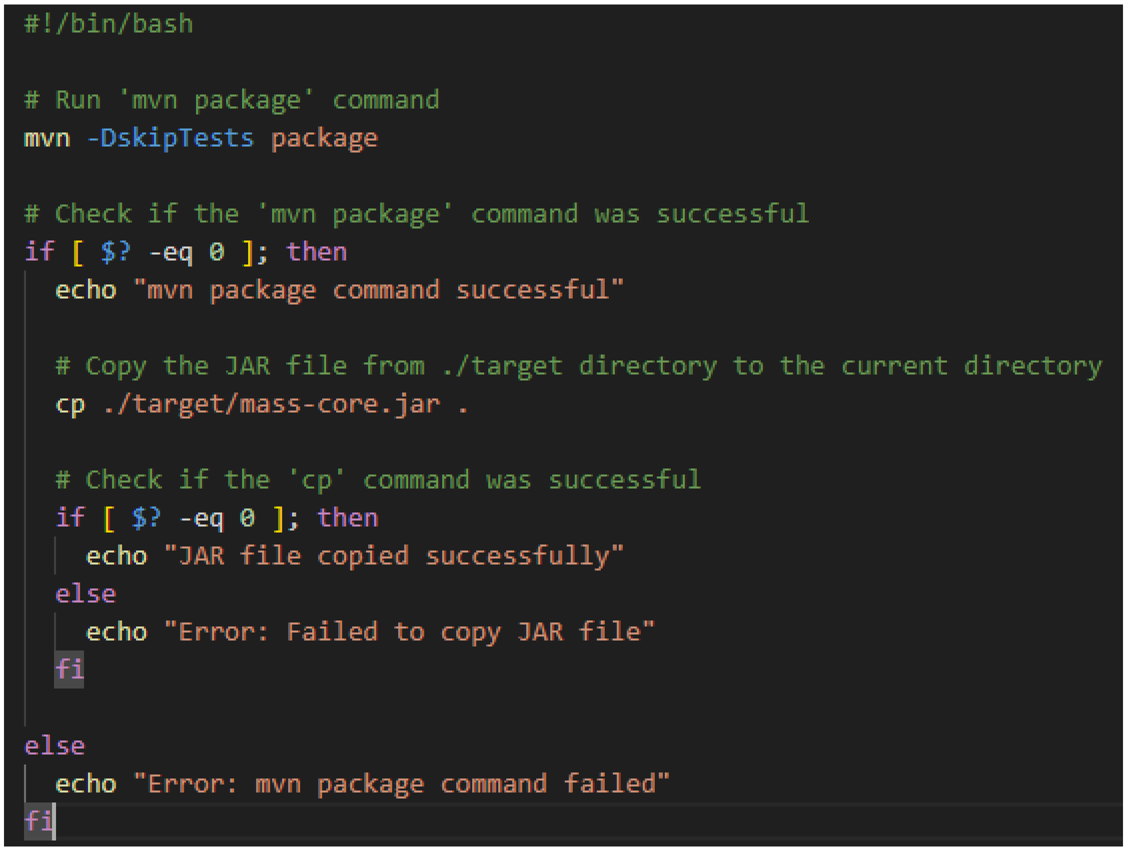


```
git clone https://j1effrey@bitbucket.org/mass_library_developers/mass_java_cor
```

Sourcetree is a free Git client for Windows.

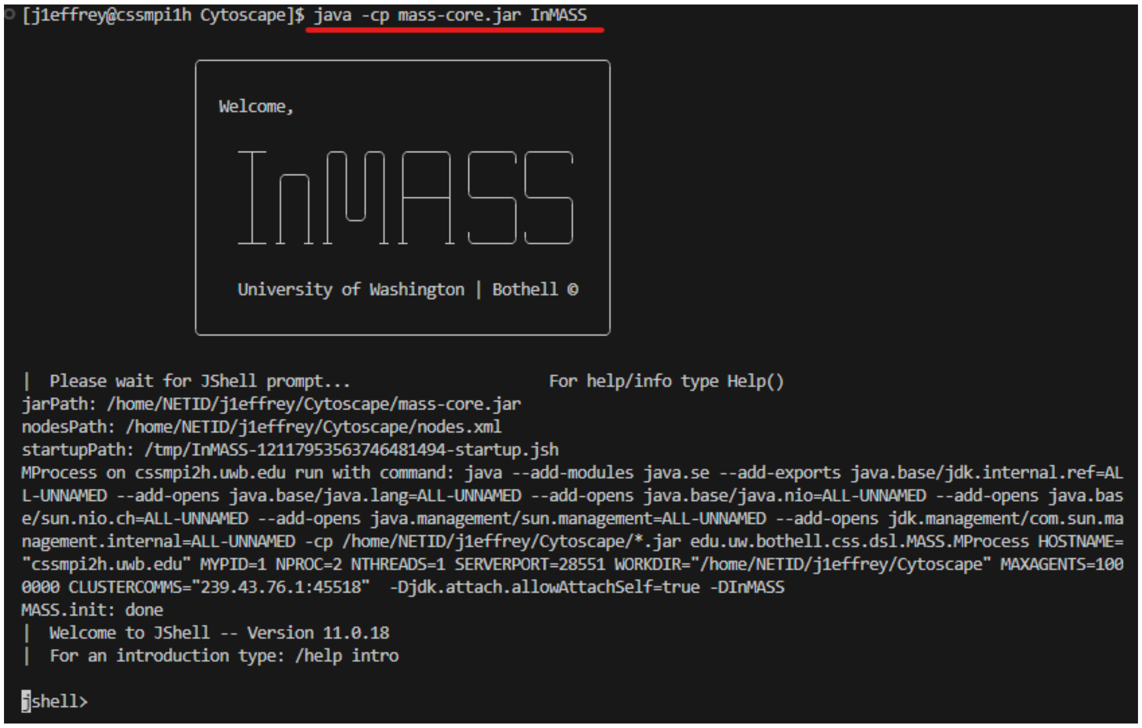
VS Code is a source-code editor developed by Microsoft.

3. **Compile the mass_java_core package**. Login to the virtual machine, find the mass_jave_core package, and run the build script (build.sh) to compile the package.



```bash
#!/bin/bash

# Run 'mvn package' command
mvn -DskipTests package

# Check if the 'mvn package' command was successful
if [ $? -eq 0 ]; then
  echo "mvn package command successful"

  # Copy the JAR file from ./target directory to the current directory
  cp ./target/mass-core.jar .

  # Check if the 'cp' command was successful
  if [ $? -eq 0 ]; then
    echo "JAR file copied successfully"
  else
    echo "Error: Failed to copy JAR file"
  fi

else
  echo "Error: mvn package command failed"
fi
```

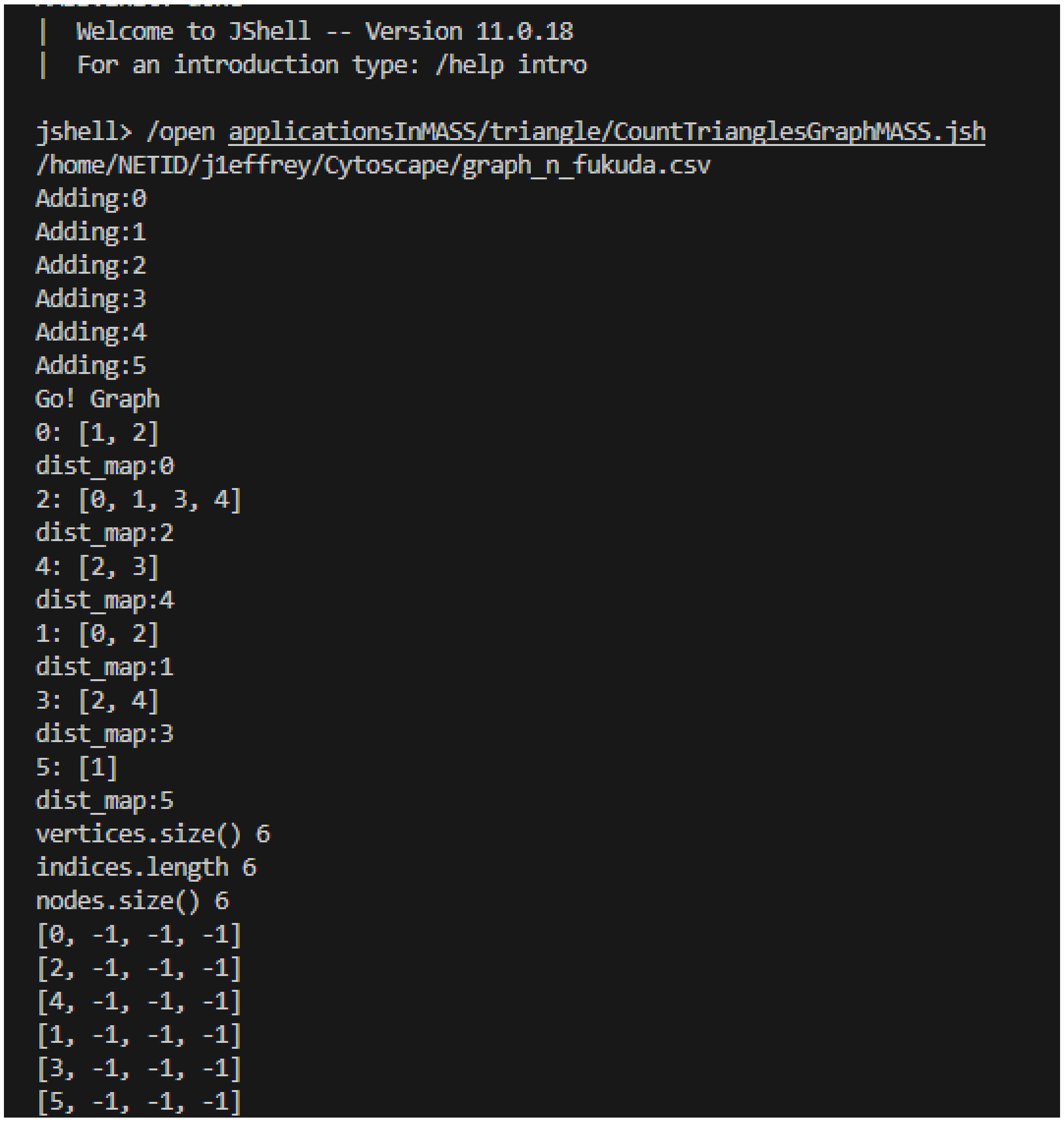
4. **Run InMASS**. Run this command to run InMASS platform.



```
[j1effrey@cssmpi1h Cytoscape]$ java -cp mass-core.jar InMASS

              Welcome,

                InMASS

                University of Washington | Bothell ©

|  Please wait for JShell prompt...                  For help/info type Help()
jarPath: /home/NETID/j1effrey/Cytoscape/mass-core.jar
nodesPath: /home/NETID/j1effrey/Cytoscape/nodes.xml
startupPath: /tmp/InMASS-12117953563746481494-startup.jsh
MProcess on cssmpi2h.uwb.edu run with command: java --add-modules java.se --add-exports java.base/jdk.internal.ref=AL
L-UNNAMED --add-opens java.base/java.lang=ALL-UNNAMED --add-opens java.base/java.nio=ALL-UNNAMED --add-opens java.bas
e/sun.nio.ch=ALL-UNNAMED --add-opens java.management/sun.management=ALL-UNNAMED --add-opens jdk.management/com.sun.ma
nagement.internal=ALL-UNNAMED -cp /home/NETID/j1effrey/Cytoscape/*.jar edu.uw.bothell.css.dsl.MASS.MProcess HOSTNAME=
"cssmpi2h.uwb.edu" MYPID=1 NPROC=2 NTHREADS=1 SERVERPORT=28551 WORKDIR="/home/NETID/j1effrey/Cytoscape" MAXAGENTS=100
0000 CLUSTERCOMMS="239.43.76.1:45518"  -Djdk.attach.allowAttachSelf=true -DInMASS
MASS.init: done
|  Welcome to JShell -- Version 11.0.18
|  For an introduction type: /help intro

jshell>
```

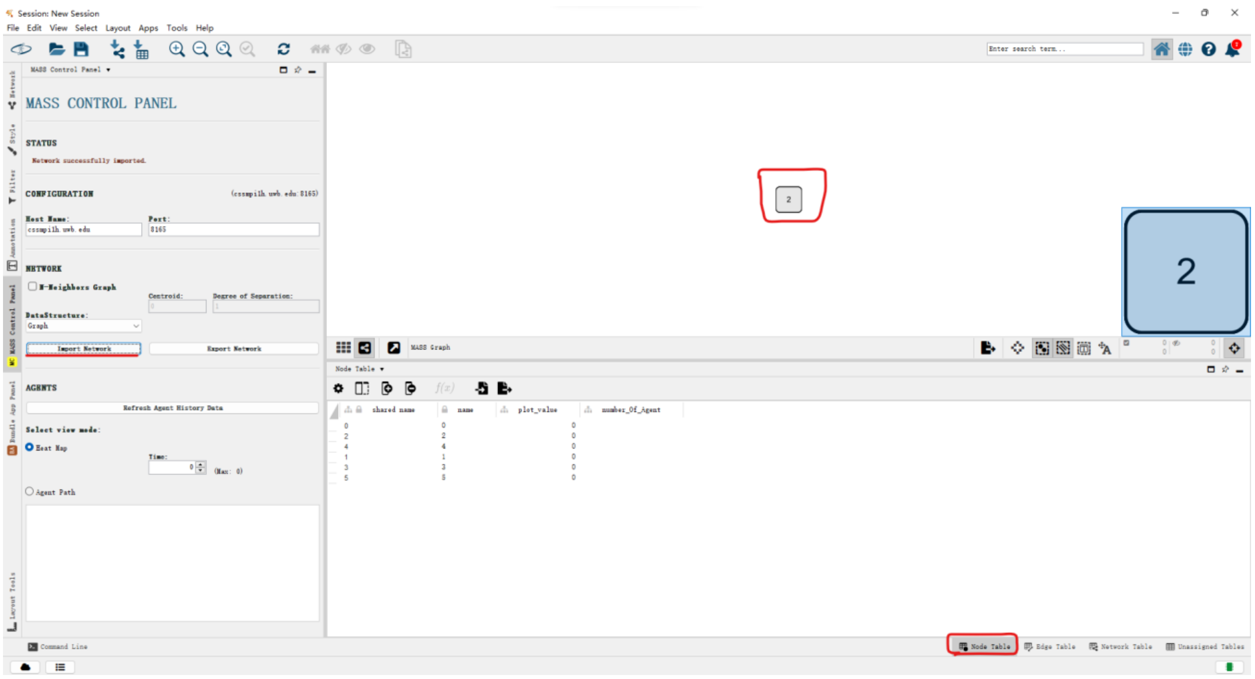
## A3. Run a visualization sample!

1. Go into the **applicationInMASS** folder and run the triangle sample.

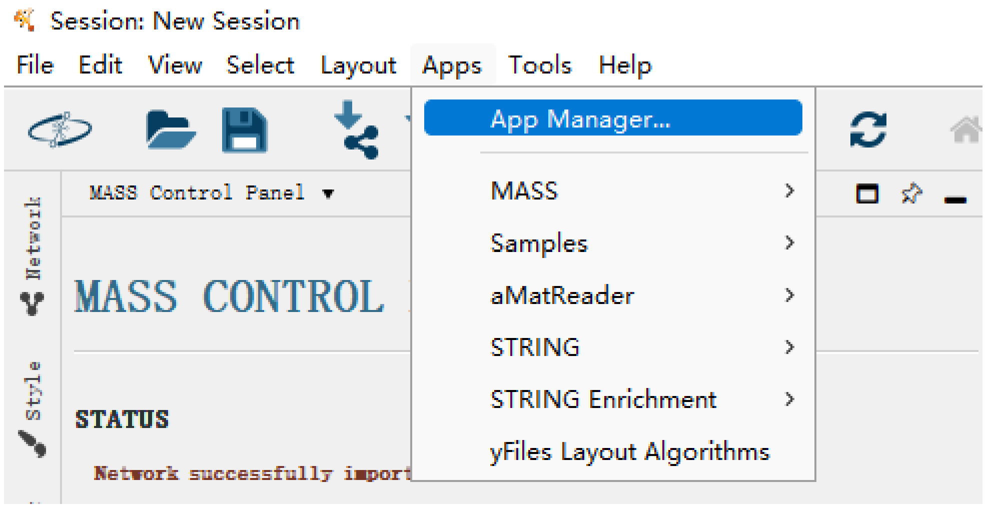


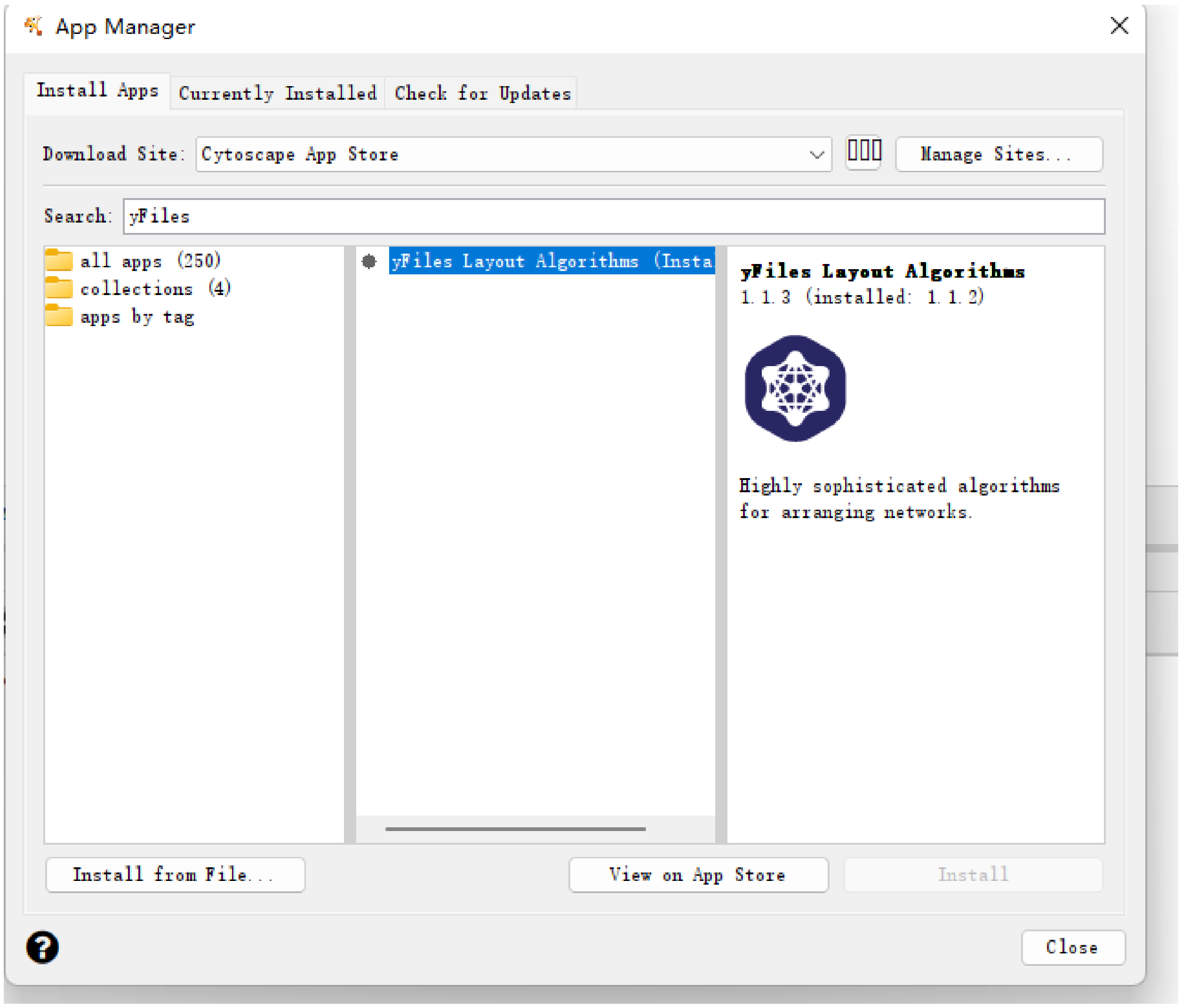
```
|  Welcome to JShell -- Version 11.0.18
|  For an introduction type: /help intro

jshell> /open applicationsInMASS/triangle/CountTrianglesGraphMASS.jsh
/home/NETID/j1effrey/Cytoscape/graph_n_fukuda.csv
Adding:0
Adding:1
Adding:2
Adding:3
Adding:4
Adding:5
Go! Graph
0: [1, 2]
dist_map:0
2: [0, 1, 3, 4]
dist_map:2
4: [2, 3]
dist_map:4
1: [0, 2]
dist_map:1
3: [2, 4]
dist_map:3
5: [1]
dist_map:5
vertices.size() 6
indices.length 6
nodes.size() 6
[0, -1, -1, -1]
[2, -1, -1, -1]
[4, -1, -1, -1]
[1, -1, -1, -1]
[3, -1, -1, -1]
[5, -1, -1, -1]
```

2. **Open Cytoscape and directly import graph network**



3. Expand the graph. This step requires to install an external plugin (**yFiles algorithm**) in the App Manager.





Download Site: Cytoscape App Store

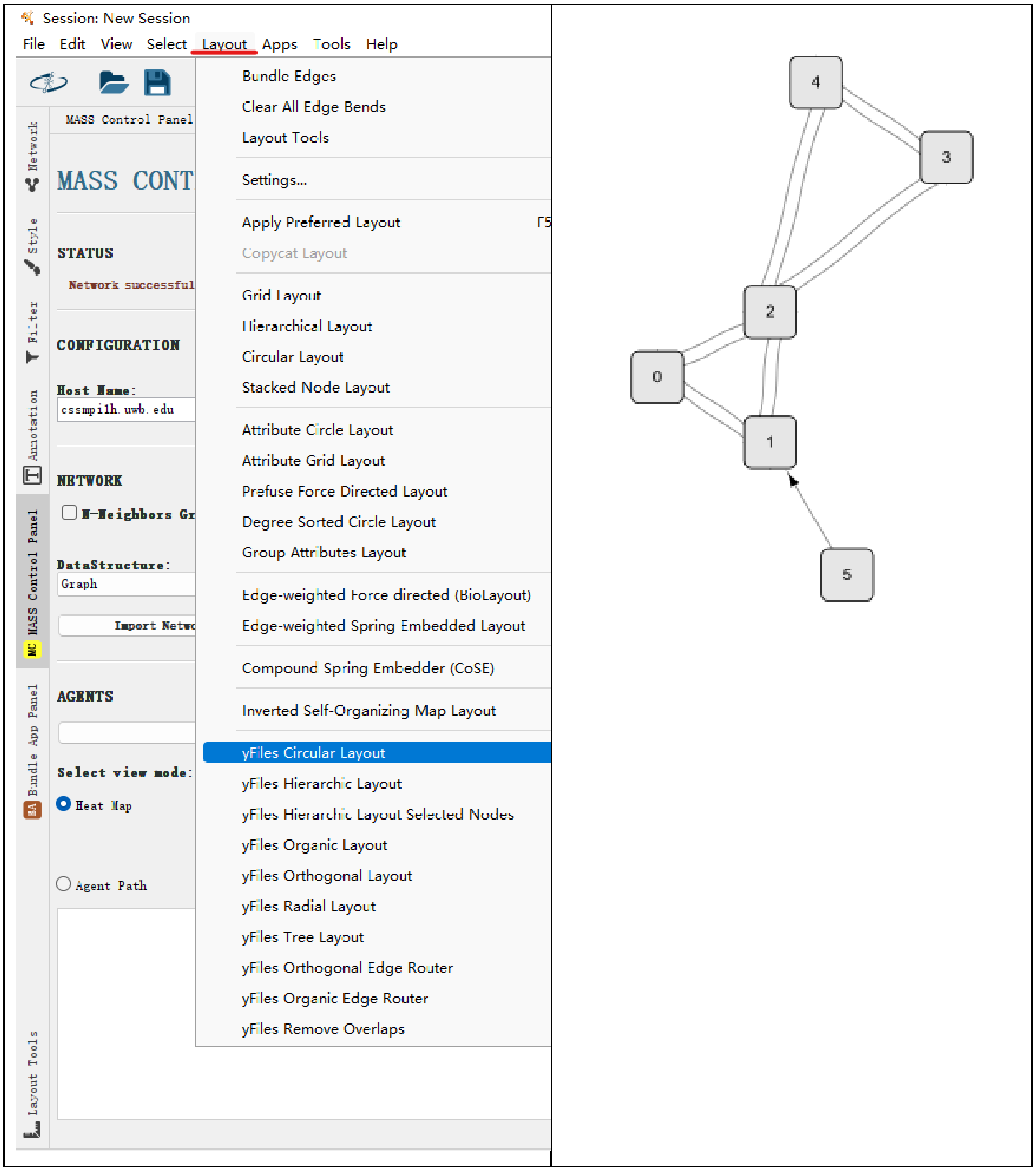
Search: yFiles

yFiles Layout Algorithms (Insta

**yFiles Layout Algorithms**
1.1.3 (installed: 1.1.2)

Highly sophisticated algorithms for arranging networks.

4. After successful installation, use the **yFile Circular Layout** to re-arrange the graph visualization. (When importing binary/quad tree network, use **yFile Tree Layout** instead)



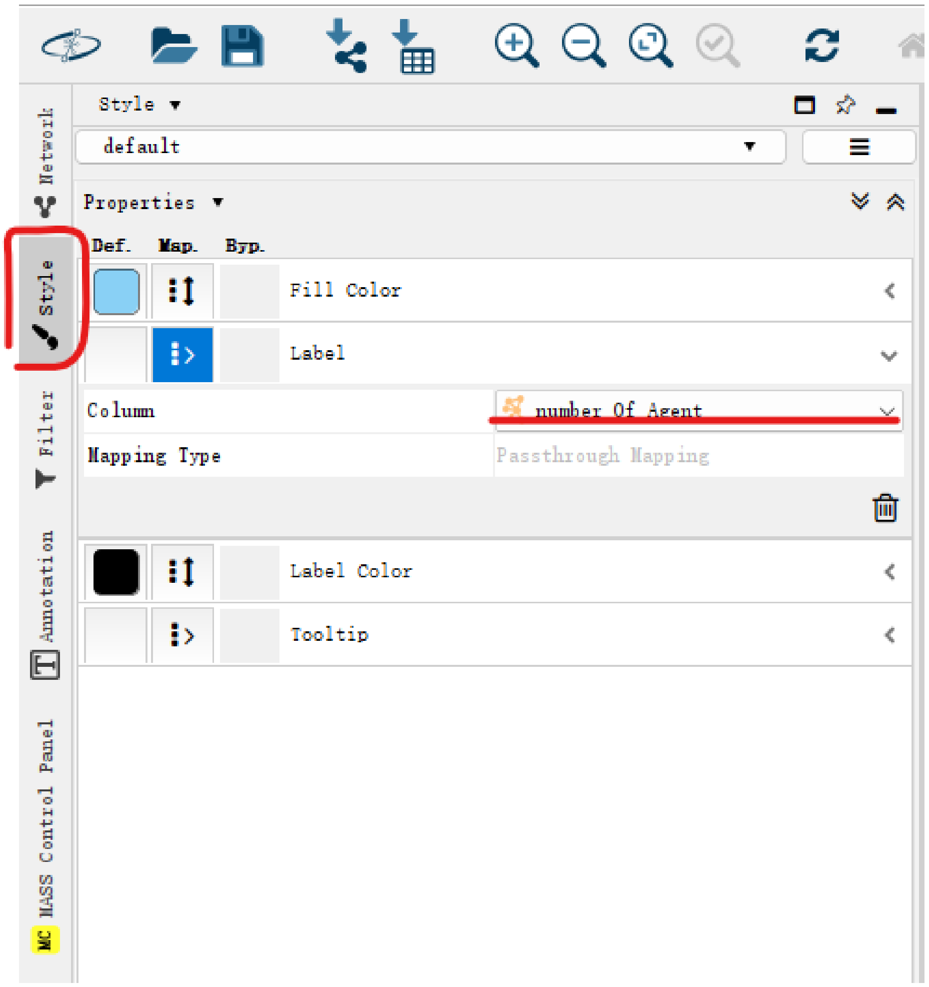
Network successful

Host Name: cssmpi1h.uwb.edu

DataStructure: Graph

Select view mode: [✓ Heat Map]

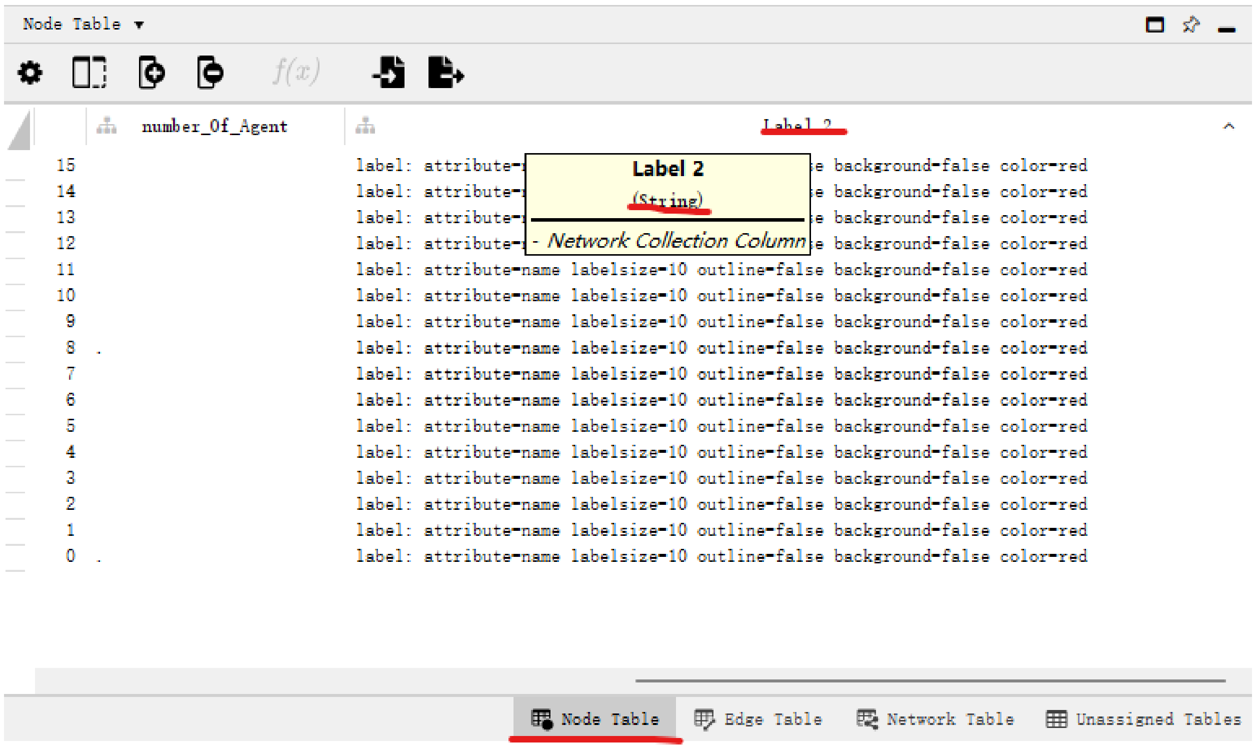
5. Import Agents (Refresh Agent History Data).
6. **Display agents as dots on each place**. Set default display label to the column **number_Of_Agent**



Column: number Of Agent

Mapping Type: Passthrough Mapping

7. Display original place ID label
   a. Add a new column “Label 2” to the Node Table, type is String. Apply value “**label: attribute=name labelsize=10 outline=false background=false color=red**” to the entire column.

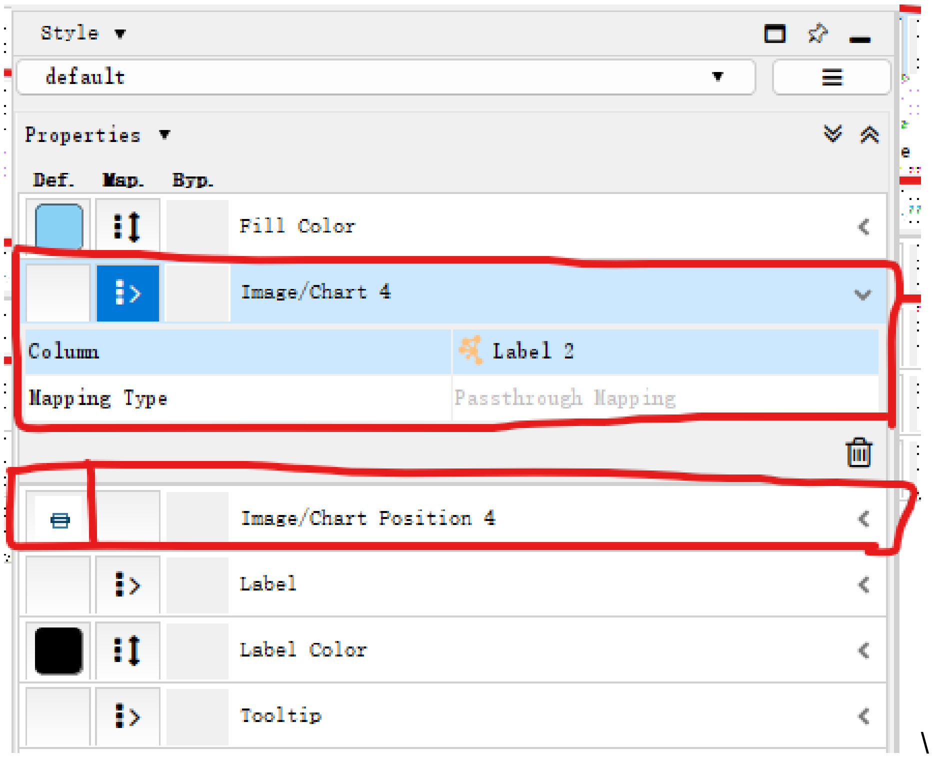


Label 2
(String)
- Network Collection Column

| | number_Of_Agent | Label 2 |
|---|---|---|
| 15 | | label: attribute= … e background=false color=red |
| 14 | | label: attribute= … e background=false color=red |
| 13 | | label: attribute= … e background=false color=red |
| 12 | | label: attribute= … e background=false color=red |
| 11 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 10 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 9 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 8 | . | label: attribute=name labelsize=10 outline=false background=false color=red |
| 7 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 6 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 5 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 4 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 3 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 2 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 1 | | label: attribute=name labelsize=10 outline=false background=false color=red |
| 0 | . | label: attribute=name labelsize=10 outline=false background=false color=red |

b. In the style panel, add a new property "**Image/Chart 4**" and select **Label 2** as Column name, and **Passthrough Mapping** as Mapping Type.

c. Add a new property "**Image/Chart position 4**" and use the left box to adjust location to avoid overlap.



Fill Color
Image/Chart 4
Column: Label 2
Mapping Type: Passthrough Mapping
Image/Chart Position 4
Label
Label Color
Tooltip

## A4. Run the WEB GUI

1. Download the MASS library. Switch branch to yifeiyang/gui.
2. Compile the core package.
3. Run a non- terminating simulation.
4. Visit local 8080 port.