An Incremental Enhancement of MASS GUI

Luo Leng

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

December 27, 2024

Project Committee:

Prof. Munehiro Fukuda, Ph.D., Chair

Prof. Annuska Zolyomi, Ph.D.,Member

Prof. Robert Dimpsey, Ph.D., Member

Abstract

An Incremental Enhancement of MASS GUI
Luo Leng

Chair of the Supervisory Committee:

Dr. Munehiro Fukuda

Computing & Software Systems

This research enhances the Multi-Agent Spatial Simulation (MASS) framework's application usability and development workflow, addressing the challenge of fragmented workflow in distributed computing simulations. The project transforms multiple separate interfaces—InMASS for simulation execution, Cytoscape for visualization, and a web-based monitoring interface—into a unified platform, reducing operational complexity by consolidating three separate tools into one integrated system. Key contributions include enhanced agent visualization in quadtree-based 2D space, implementation of octree data structure for 3D space simulation, and development of interactive MASS simulation capabilities within Cytoscape. The integration leverages Cytoscape's OSGi framework and JShell's REPL features to encapsulate web browser and CLI functionalities as plugins, enabling seamless interaction between components while reducing development complexity and learning curve. Notable technical implementations include comprehensive agent tracking functionality, SSH-tunneled remote access capabilities, and real-time cluster monitoring features. Compared to existing open-source agent-based modeling systems, our enhanced MASS framework offers distributed computing capabilities, while maintaining an intuitive, integrated development environment. The results show that time spent on common simulation tasks (modifying agent behaviors, adjusting input, and configuring output) was reduced by 60% while enhancing user experience in developing and monitoring distributed agent-based simulations.

TABLE OF CONTENTS

Abstract ..... **I**
TABLE OF CONTENTS ..... **II**
LIST OF FIGURES ..... **III**
LIST OF TABLES ..... **IV**
LIST OF LISTINGS ..... **V**
Chapter 1. INTRODUCTION **1**
1.1 Project Overview 1
1.2 Goals/achievements 2
Chapter 2. Background **4**
2.1 MASS 4
2.2 Web-GUI 8
2.3 Cytoscape 9
2.4 Agent Tracking and visualization 10
2.5 Challenge 11
Chapter 3. Related Work **13**
3.1. Repast Simphony 13
3.2. NetLogo 15
3.3 Distributed MASON 17
Chapter 4. MASS GUI Development and Enhancement **20**
4.1 Integration Design 20
4.2 InMASS 23
4.3 Web GUI 29
4.4 Cytoscape 32
Chapter 5. Evaluation of MASS Functionality and Usability **39**
5.1 Comparison Between Current and Previous Workflows 39
5.2 Evaluation based on user feedback 43
Chapter 6. Conclusion **53**
6.1 Summary of Achievements 53
6.2 Limitations 54
6.3 Future Work 54
REFERENCES **56**
Appendix A: Setting Up and Running InMASS GUI Applications **58**
Appendix B: InMASS User Experience Survey Result **71**
Appendix C: Survey Template **75**
Appendix D: Class diagram for Closest Pair Points Implementation **78**
Appendix E: System Initialization Sequences **79**

LIST OF FIGURES

Figure 2.1: MASS Model [1]…… 6
Figure 2.2: InMASS welcome screen…….……8
Figure 2.3: Previous Web GUI……..……..……..9
Figure 2.3: MASS-Cytoscape……..……..10
Figure 3.1: Zombies simulation in Repast Simphony [12]……14
Figure 3.2 The NetLogo Predator-Prey simulation Wolf-Sheep-Grass [15]…….……16
Figure 3.3: Distributed MASON interface showing worker configuration[17]……18
Figure 4.1 : Component Diagram for MASS Cytoscape……...21
Figure 4.2: Activity diagram for MASS Cytoscape……...22
Figure 4.3 Class Hierarchy and Relationships in InMASS System …….…27
Figure 4.4: integrated Web GUI with Cytoscape built-in browser……29
Figure 4.5: SSH Command Line plugin……34
Figure 4.6: Class Diagram of SSH-CLI Plugin…….……36
Figure 4.7: 3D space visualization in Cytoscape……...38
Figure 5.1: Previous workflow of InMASS development and visualization…….…40
Figure 5.2: Current workflow of InMASS development and visualization……...41

LIST OF TABLES

Table 4.1: List of Cytoscape Plugins and Their Usage……..……33
Table 5.1: Comparison between original workflow and current workflow……...……42
Table 5.2: Survey Questions for InMASS Evaluation …….……..…44
Table 5.3: System Information and Installation Experience (Q1&Q2)...…….……45
Table 5.4: Distribution of Participants' Experience with Other ABM Tools (Q3).....46
Table 5.5: Performance Comparison between MASS Cytoscape and Other Tools (Q3).....47
Table 5.7: Desired Simulation Scenarios (Q4)……...……...47
Table 5.6: Interface Preferences (Q6).……...48
Table 5.7: Feature Preferences for Web Based Interface (Q7).....49
Table 5.8: Most Valuable Aspects of InMASS (Q8).....50
Table 5.9: Areas for Improvement (Q9).....50
Table 5.10: Additional Suggestions from User Feedback (Q10).....51

LIST OF LISTINGS

Listing 2.1: InMASS initialization using JavaShellToolBuilder…….7
Listing 4.1: Implementation of agent tracking analysis…….24
Listing 4.2: Quadtree's Agents and Places initialization …….25
Listing 4.3: RESTful endpoints used in Web GUI…….……...32

ACKNOWLEDGEMENTS

I sincerely thank **Professor Munehiro Fukuda** for his invaluable guidance, insightful suggestions, and patient, hands-on instruction throughout the development of this white paper and project. His mentorship has been instrumental in navigating the academic environment and the research process. His expertise and support not only shaped the direction of this work but also deepened my understanding of research methodologies.

I am also deeply thankful to the members of **DSLab** for their constructive feedback, technical assistance, and collaborative spirit. Their contributions during lab meetings and informal discussions were essential in refining key aspects of this research and greatly enriched my overall experience.

Chapter 1. INTRODUCTION

1.1 Project Overview

Big Data computing, which emerged prominently more than a decade ago, involves using clusters of computing nodes to process and analyze vast amounts of data [1]. This technique leverages powerful tools like MapReduce and Spark, which excel at processing large volumes of unstructured data and streaming text-based information at scale [2]. These tools facilitate the automatic parallelization of data processing tasks, which allows for efficient handling of large datasets across multiple computing nodes. However, they often face challenges with processing in-memory data structures such as graphs and typically lack capabilities for incremental data modifications and visualization.

Agent-based modeling (ABM) is a distinct computational research approach that enables researchers to create complex models based on the interactions of individual agents within a simulated environment [3]. Each agent operates autonomously with its own set of rules and behaviors, interacting with other agents and the environment. This bottom-up modeling approach is incredibly versatile and can simulate a wide range of phenomena, from biological systems to economic markets and social behaviors. In the context of Big Data, ABM is particularly useful because it can help uncover emergent patterns and behaviors within large datasets that traditional analytical methods might miss. This makes ABMS invaluable in fields that require a deep understanding of dynamic and complex systems, such as epidemiology, urban planning, and ecosystem management.

The Multi-Agent Spatial Simulation (MASS) library, developed by the Distributed Systems Laboratory at the University of Washington Bothell. This advanced parallel computing framework is designed to simplify the creation of ABM in distributed computing

environments[4]. The MASS library utilizes - Places and Agents to facilitate structured data analysis and large-scale simulations across computing node clusters. Places are organized in a multidimensional array to represent the simulation environment, allowing Agents to interact with and autonomously migrate between these locations. The framework ensures efficient parallel execution through multithreaded communication processes, connecting via secure channels and TCP sockets, to enable robust data exchange and method invocation between Places and Agents.

This project primarily aims to significantly enhance and integrate the Graphical User Interface (GUI) of the MASS. Previously, the application featured three main components: a Web GUI for monitoring cluster performance, InMASS for executing simulations, and MASS-Cytoscape for visualizing places and agents. The MASS library accommodates various data structures, such as graphs, 2D continuous spaces, binary trees, and quad trees, which are tailored to expedite calculations for specific practical problems—graphs for triangle counting, binary trees for range searching, and quad trees for identifying the closest pair of points (CPP). Currently, the visualization of these data structures is a fragmented process. DsLab students and developers must manually transfer data from the MASS application to Cytoscape, a complex and multi-step procedure. This project seeks to simplify this data transfer, enhance user interaction, and add new functionalities to enrich users' understanding of their simulations.

1.2 Goals/achievements

The main goals of my project enhance and enrich the GUI functionalities within the MASS framework. The four key objectives include:

1. **Enhanced Agent Visualization:** Enrich the visualization of agents within the MASS framework. Instead of representing agents as simple dots on a vertex, the goal is to visualize them with their IDs. It is also necessary to visualize agents within a quad

tree-based 2D space to see their interactions and distribution. This enhancement will help users better understand how agents travel between places.

2. **Octree / 3D Space Visualization:** Allow MASS to construct an octree for 3D space representation and then project this visualization in Cytoscape. This improvement introduces templates and insights for a data structure not present in the current application.
3. **MASS Computation Directed from Cytoscape:** Allow users to initiate agent migration/computation and implement agent-based algorithms directly within the interface. This feature should streamline the MASS computation process.
4. **Integrated GUI:** The goal is to integrate InMASS, Cytoscape, MASS, and the web menu into one single website. Such integration will simplify the workflow and enhance usability.

Chapter 2. Background

This project builds upon the efforts of various DSLab graduate students whose foundational work and academic contributions have been crucial for its smooth progression. Their innovations include the development of inMASS for easier execution of MASS statements and simulations, enhancements to monitoring features for improved user control over simulations and cluster performance, and the incorporation of trees and continuous spaces into the MASS Java Library. They also integrated Cytoscape with the MASS library and introduced several valuable tools. The summary below outlines the previous work that forms the foundation of this project.

2.1 MASS

The MASS library is a parallel-computing framework designed for Multi-Agent Spatial Simulation. Central to the library are the concepts of Places and Agents[6].

2.1.1 Places, Agents, and SmartAgent

- Places: Places represents a multi-dimensional distributed array framework that spans across a cluster of computing nodes. This array structure, which could be 2D or 3D, is partitioned and distributed among multiple computing nodes for parallel processing. A Place is an individual element within the Places array structure. Each Place has a unique network-independent index (like coordinates in a grid) that identifies its position within the global array. Places can communicate with each other regardless of their physical location in the cluster, enabling seamless information exchange across the network.
- Agents: Agents, the execution instances that can reside in a Place, migrate to another, or interact with multiple Places and other Agents, thereby duplicating themselves. Key agent behaviors include migrate( ), spawn( ), and kill( ), each moving the calling agent,

spawning its children, and terminating itself, which are managed through the manageAll method that updates agents' statuses based on their actions.

- Smart agent: We introduced the Smart agent class [7] to facilitate high-level agent migration functions, which extends the foundational Agent class, incorporating automated navigation methods and properties. This setup not only simplifies the code required to define agent behaviors by using automated methods but also improves simulation performance through better lifecycle management of agents. Users can create custom agent classes for specific actions and extend SmartAgent to utilize these advanced navigation capabilities.

Figure 2.1 shows the MASS library architecture. At its core, the system utilizes multi-threaded communication where processes are forked across multiple cluster nodes. These nodes communicate with each other through SSH-tunneled TCP connections, represented by the socket connections between nodes. Each node in the cluster spawns a number of threads equal to its available CPU cores, which operate on the node's system memory. For Places, the system employs a vertical striping mechanism where the multi-dimensional array of Places is partitioned into vertical sections, with each stripe being statically mapped to specific threads. Each thread consistently processes its assigned portion of the Places array. In contrast, Agent distribution follows a more dynamic approach where Agents are allocated to processes based on their proximity to the Places that the process maintains. These Agents can be executed by any of the multiple threads within their assigned process, enabling flexible, location-aware scheduling. This approach allows MASS to optimize both computational efficiency and system flexibility while maintaining effective load balancing across the cluster.

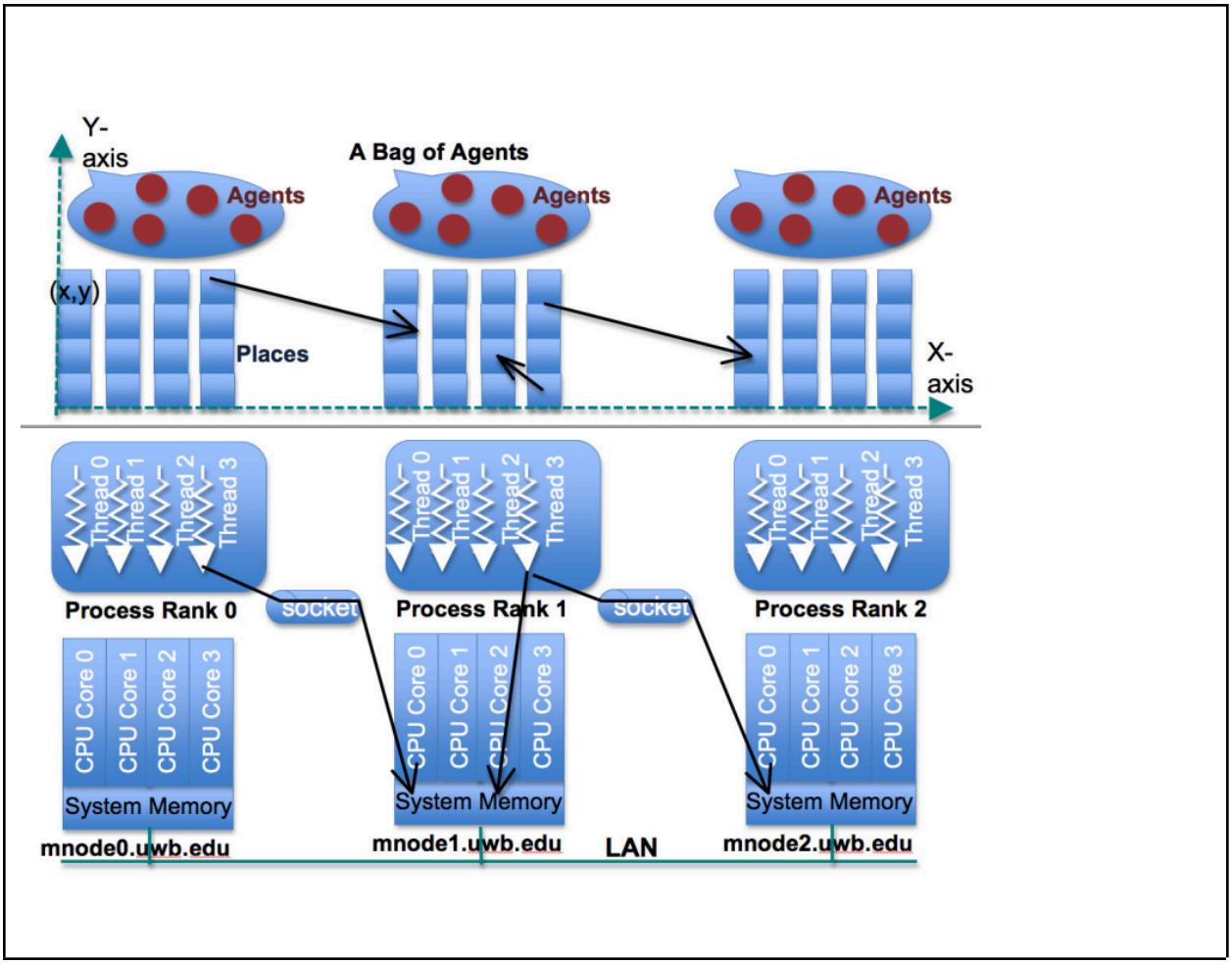


Figure 2.1: MASS Model [1]

2.1.2 InMASS

Interactive MASS (InMASS) is a powerful enhancement to the MASS Java library. It was initially developed [2] and later enhanced in [6] and [7]. At its core, InMASS wraps around JShell, Java's interactive REPL (Read-Evaluate-Print Loop) tool. This enables users to execute MASS simulations through a command-line interface line-by-line, similar to Python's interactive mode. Users can dynamically modify and query simulation states at runtime, without needing to stop or recompile the program.

The system manages configurations through a properties file and handles distributed execution through nodes.xml configuration. It supports both single-node and cluster modes for flexible deployment. Key features include dynamic class management, checkpoint capabilities, and real-time agent tracking. InMASS is particularly useful for scientific simulations and distributed computing applications. It bridges Java's compiled nature with the flexibility of interpreted languages, making real-time modification and monitoring possible in distributed environments.

Listing 2.1 (lines 1-10) demonstrates the configuration and launch process of the JShell environment with essential parameters (lines 5- 7) including startup path, class path, and runtime flags for both single-node and cluster deployments. Figure 2.2 welcome screen after successful initialization showing the JShell environment configuration details, including JAR path, nodes path for cluster configuration, startup parameters, and confirmation of MASS initialization followed by the JShell prompt.

Listing 2.1: InMASS initialization using JavaShellToolBuilder

```
1  // start jshell
2  try {
3      JavaShellToolBuilder builder = JavaShellToolBuilder.builder();
4      // enable InMASS flag, enable self attach agent, & suppress warning
5      builder.run(...arguments:"--startup", startupPath, "--class-path", jarPath, "-R-DIr
6          "-R-Djdk.attach.allowAttachSelf=true", "-R--add-modules java.se --add-exports j
7          System.getProperty(key:"single") != null ? "-R-Dlocal" : "-R-Dcluster");
8  } catch (Exception e) {
9      e.printStackTrace();
10 }
```

Figure 2.2: InMASS welcome screen

2.2 Web-GUI

The WEB GUI serves as an essential tool for users to monitor their cluster's performance in real-time. It includes a cluster monitoring panel for tracking the status of each machine, with indicators to show if a machine is running or has been terminated unexpectedly [6]. The interface also allows users to monitor the functional calls made to the cluster, an extension of the checkpoint/rollback feature. This feature provides the user with a view of all calls along with their function IDs, aiding in identifying the specific steps for potential rollbacks. The rollback operation is also simplified for a streamlined user experience within InMASS. Figure 2.3 showcases the primary interface of the WEB GUI, offering essential insights into cluster performance for users. The interface's upper area features a cluster monitoring panel, providing real-time updates on each cluster machine's status. The panel below presents the functional calls made by users to the cluster. This functionality builds upon the checkpoint/rollback feature, allowing users to see all made calls along with their associated function IDs.

Figure 2.3: Previous Web GUI

2.3 Cytoscape

Cytoscape is an open-source Java platform used by data scientists and academics from a range of fields for the display and analysis of scientific graphs including biological networks [8]. It offers a user-friendly interface for data import/export and visualization while supporting a variety of data sources and file types. Users can adjust network appearances and highlight particular patterns thanks to customization capabilities. It is adaptable and expandable using plugins, which are controlled by the App Manager and created by users or outside developers [8]. Figure 2.4 displays the InMASS user panel as integrated within Cytoscape. On the application's left side, the InMASS user panel is featured, enabling users to link Cytoscape with Dslab machines running InMASS applications and to configure the visualization layout. On the application's right side, the visualization of agents and places from the InMASS application is presented.

The OSGi (Open Services Gateway initiative) framework serves as a fundamental component in Cytoscape's architectural design [9] and its integration capabilities with tools such as InMASS. This Java-based framework implements a dynamic component system that

facilitates remote installation, initiation, termination, updates, and uninstallation of applications without system reboot requirements. Within Cytoscape's ecosystem, OSGi enables modular architecture, thereby enhancing platform extensibility through various plugins and applications.

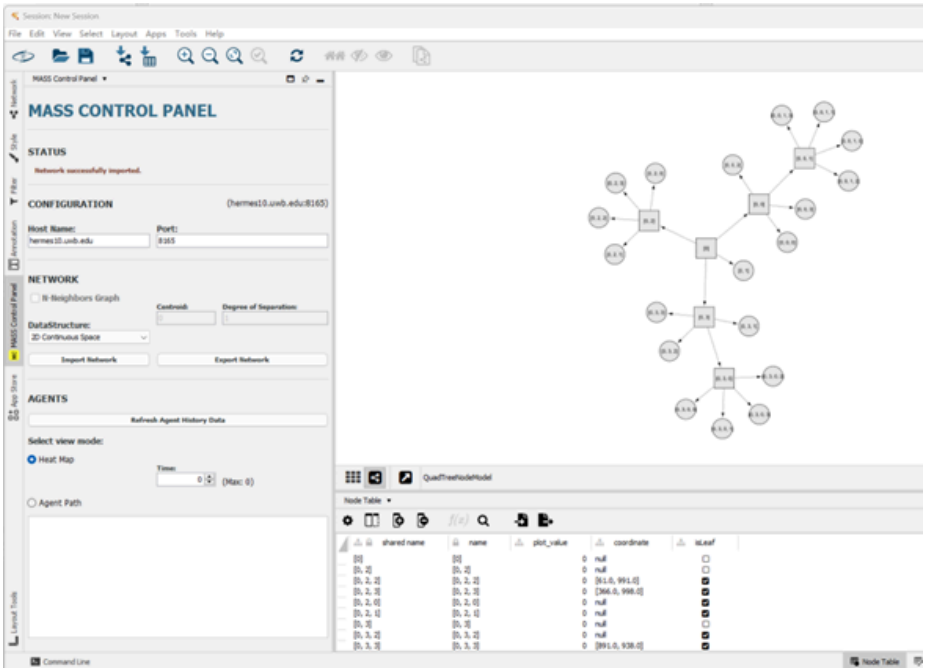


Figure 2.4: MASS-Cytoscape

2.4 Agent Tracking and visualization

MASS agent-tracking API was developed and announced in 2022 [10].Three new classes to capture the agent visitation data within MASS Java library: AgentHistoryCollection, AgentHistoryModel which is later renamed as AgentItinerary, and AgentHistoryManager[5]. AgentItinerary is responsible for recording the complete journey of a single agent. It contains agent's identity information, visit history and agent status information, which enables complete tracking of the agent path. AgentHistoryManager is responsible for handling and maintaining

historical records of all Agents. It stores all Agents' journey records through itineraryMap and filters which Agent types need to be tracked through the classNames collection. When an Agent performs key operations (such as moving, spawning, or dying), Place calls manage methods to record these events. AgentHistoryCollection is the container class that summarizes all Agents' historical records through a list of AgentItineraries. The main purpose of this class is to provide a unified interface to access all historical data and facilitate the transmission of this data in distributed systems. When an experiment ends, we can obtain behavioral records of all Agents through AgentHistoryCollection, thereby performing visualization.

2.5 Challenge

The previous work on MASS GUI enhanced the MASS library's usability and simplified the simulation development process. However, as mentioned in the introduction section, these functionalities were not well integrated, resulting in a fragmented workflow that required users to manage 4-5 different interfaces simultaneously.

The workflow complexity can be broken down as follows:

1. Users needed a terminal window on their local machine for SSH access to remote machines, which had to remain active for file transfers and log monitoring.
2. A second window was required for remote machine access and JShell session management. Due to remote display limitations, users could only access either the command-line interface or the JShell session at a time, requiring a complete session termination to switch between them. The JShell session needed to remain active until simulation data was transferred to Cytoscape.
3. A third window ran the Cytoscape application for visualizing simulation properties and results.

4. A fourth window hosted the web-based GUI monitoring feature in a web browser.
5. Code modifications required a fifth window for editing, along with using the second window for file management and simulation re-runs.

This fragmented workflow made it challenging for users to maintain focus on their primary development tasks. Moreover, when combined with Java's complex compilation and JAR packaging requirements, developers were forced to dedicate a significant portion of their time and attention to managing these various interfaces and technical processes rather than focusing on the actual simulation development work.

Chapter 3. Related Work

While many Agent-Based Modeling (ABM) simulation systems include visualization features, this project sets itself apart by aiming to deliver a more streamlined, user-friendly, and interactive experience within the MASS framework. The subsequent sections will detail the specific features of various ABM systems and compare them with the proposed enhancements to the MASS GUI.

3.1. Repast Simphony

Repast Simphony is a widely used open-source ABM environment that builds upon the Repast 3 library [11]. It was designed with a focus on well-factored abstraction, which allows user model components to be plain, unadorned Java objects that are accessible to and replaceable with external software.

Regarding third-party model integration, Repast Simphony's plugin architecture is designed to be open and extensible. The system includes built-in support for various external analysis tools. For visualization capabilities, Repast Simphony offers both 2D and 3D visualization modules that provide interactive viewing of running models. As shown in Figure 3.1, the interface features a user control panel for parameter adjustment (such as number of humans, zombies, and gestation period) and a scenario tree for model configuration. The main visualization window displays agents in a 2D space with different colors representing various agent states, and white lines indicating agent interactions or relationships. The tick counter at the top tracks simulation progress, while the simulation can be controlled through buttons like "Setup," "Go Once," and continuous "Go." The 2D visualization is implemented using OpenGL via the JOGL API library, which provides excellent performance for displaying large numbers of agents. The 3D visualization supports complex spatial relationships and includes features for

camera control, zooming, and rotation, enabling real-time observation and analysis of agents' behaviors and interactions. These capabilities are enhanced by a highly interactive and customizable user interface for dynamic simulation control and experimentation.

However, compared to the MASS library, Repast Simphony is limited to single-machine multithreading. While it supports parallel execution, it cannot distribute computation across multiple machines, thus limiting the scale of simulation.

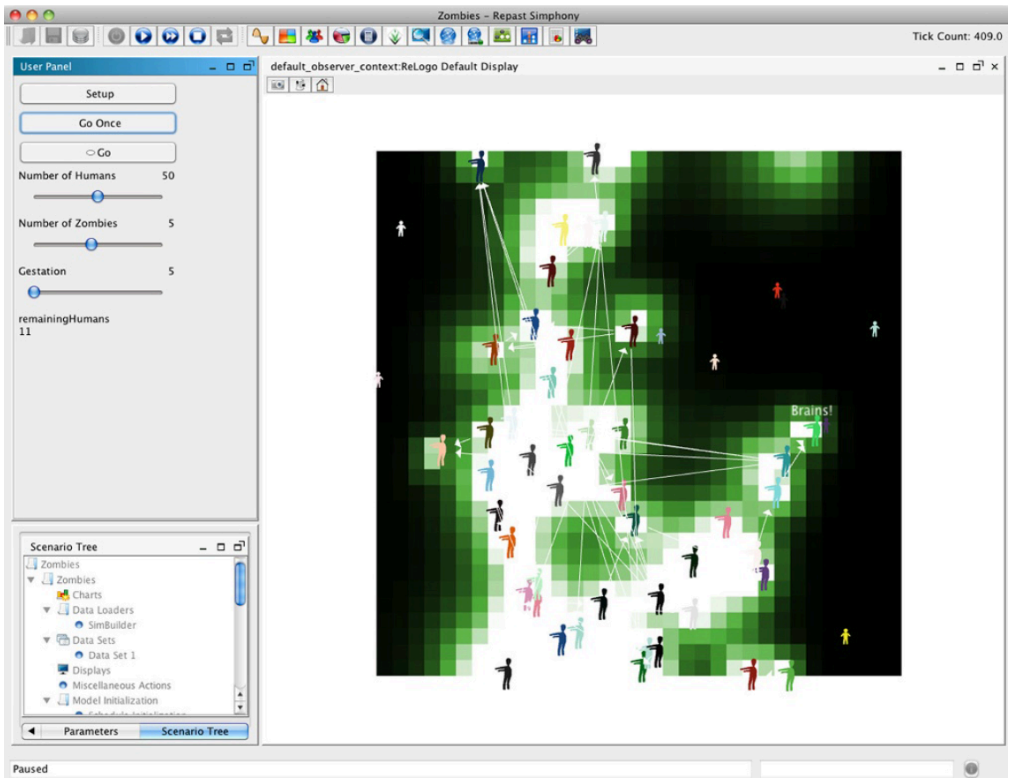


Figure 3.1: Zombies simulation in Repast Simphony [12]

3.2. NetLogo

NetLogo, developed by Uri Wilensky, is a free and open-source agent-based simulation environment based on the Logo programming language [13]. While initially designed for teaching complexity concepts, it has evolved to support sophisticated simulations of social and natural phenomena. The environment operates through four agent types: mobile turtles (primary agents), stationary patches (forming a 2D grid world), links (connections between agents), and an observer (controlling simulation execution).

As demonstrated in Figure 3.2, NetLogo provides an intuitive interface for spatial visualization and simulation control. The Wolf-Sheep Predation model showcases NetLogo's interactive capabilities, featuring adjustable parameters such as initial population numbers, reproduction rates, and energy gains for both predator (i.e, wolves) and prey (i.e, sheep) species, as well as environmental settings like grass regrowth time. The interface includes real-time population graphs tracking the dynamic relationships between wolves, sheep, and grass over time, while the main visualization window displays the spatial distribution and interaction of agents across the environment. The simulation can be controlled through basic commands like "setup" and "go," with additional options for speed adjustment and visualization preferences.

Following Wilensky's philosophy, NetLogo balances accessibility with capability - it's designed to be approachable for beginners while supporting advanced users through extensive built-in commands for agent, network, and spatial operations. However, its Logo-based architecture can present compatibility challenges with other programming languages like Java, potentially limiting its integration with other simulation platforms[14]. NetLogo Web enables users to run and create simulations directly in web browsers without installation, primarily

serving as an educational platform for teaching ABM concepts and agent behavior algorithms. However, several key constraints affect its capabilities: performance is restricted to approximately 1,000 agents maximum, and it offers limited compatibility with structured data input files. These constraints make it more suitable for educational purposes and small-scale demonstrations rather than complex research simulations.

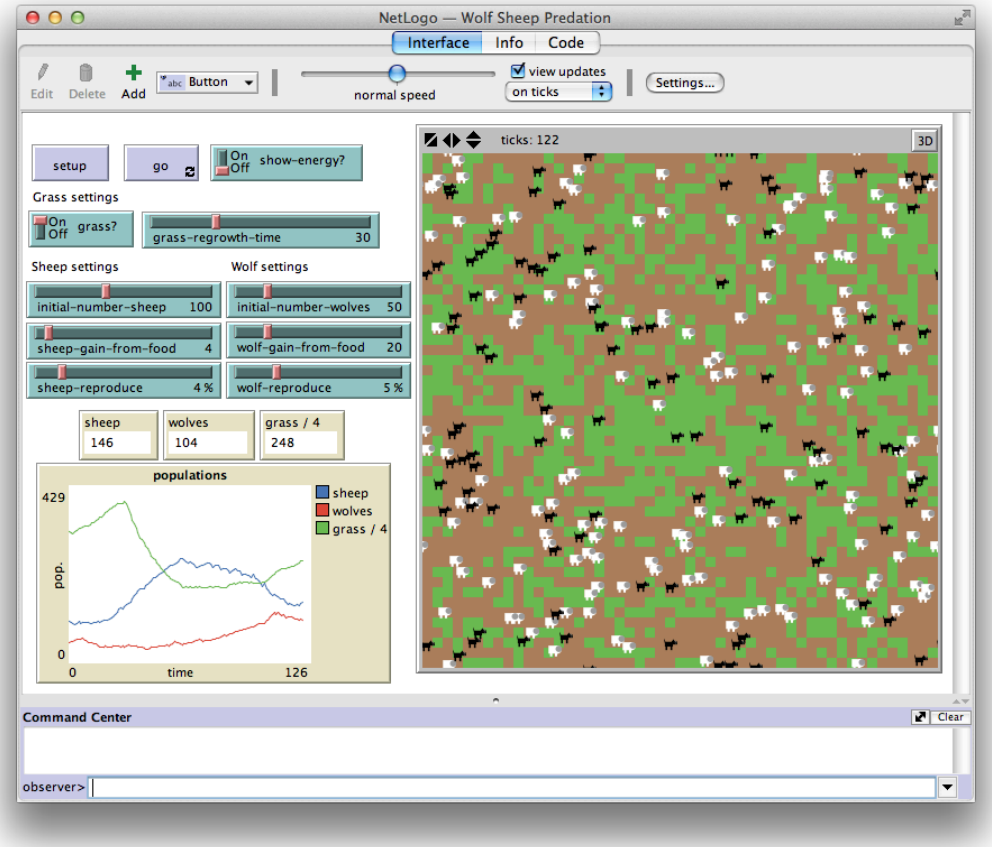


Figure 3.2 The NetLogo Predator-Prey simulation Wolf-Sheep-Grass [15]

3.3 Distributed MASON

MASON employs a Model-View-Controller (MVC) architecture that strictly separates simulation and visualization components [16]. The core simulation engine uses a real-valued time schedule for agent management, allowing agents to be scheduled for future actions, interact with their environment, and move through space. Agents can exist either as physical entities in space or as abstract computational units, with support for various field structures including 2D/3D spaces, networks, and grids.

The visualization system includes a control console and multiple display windows for real-time data representation, supporting both 2D/3D rendering and GIS integration through GeoMASON. Models are fully serializable and self-contained, enabling parallel execution across multiple threads. MASON's design emphasizes modularity, extensibility, and high performance, making it suitable for large-scale simulations involving millions of agents.

Distributed MASON, built upon the original MASON framework, employs a parallel discrete event simulation architecture using Logical Processes. As shown in Figure 3.3, the D.MASON interface provides configuration options for distributed simulation, including worker node management and region partitioning. The interface allows users to specify key parameters such as the number of regions (set to 4 in the example), maximum distance (10), width and height dimensions (= 10,400), and the total number of agents (= 15,0000). Workers can be connected through IP address and port configurations, with the system supporting both horizontal and square partitioning strategies for load distribution. It utilizes OpenMPI for communication and implements sophisticated load balancing strategies, including local, global, and hierarchical approaches. Distributed MASON remains in prototype stage, as stated on its official website. Its

unresolved performance issues and developmental bugs currently restrict its practical use in research applications.

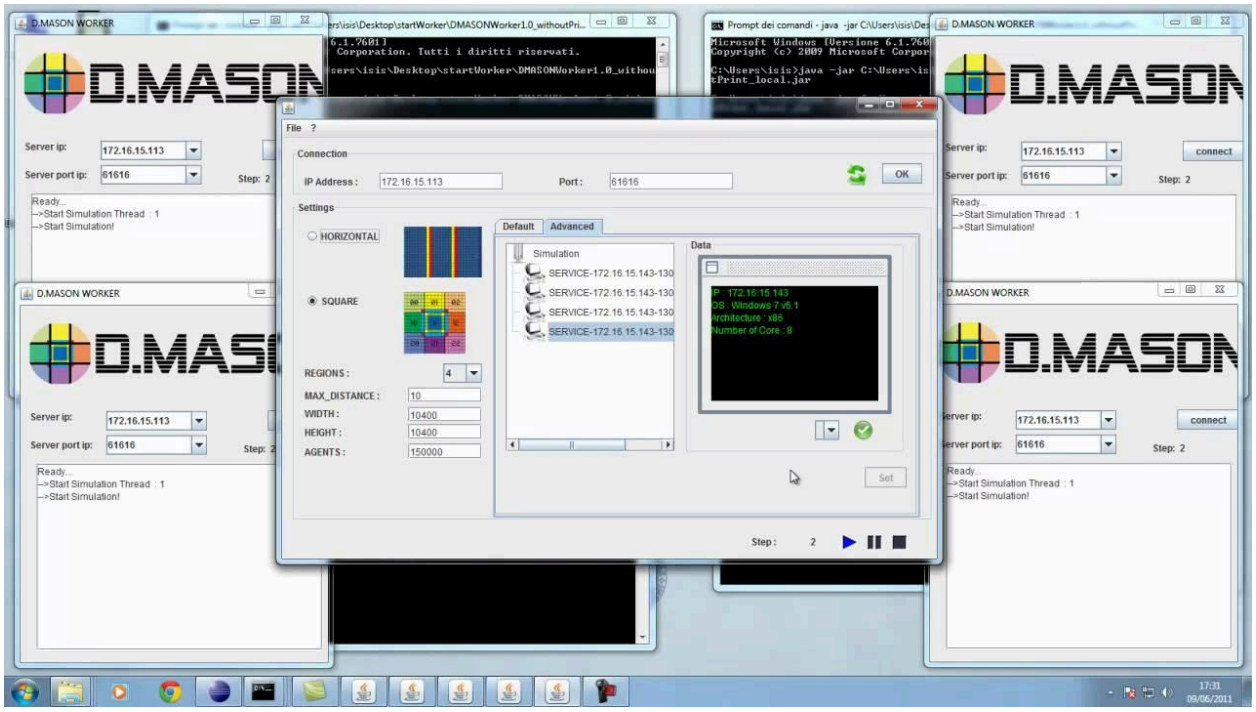


Figure 3.3: Distributed MASON interface showing worker configuration[17]

3.4 Summary

Examining Repast Simphony, NetLogo, and Distributed MASON highlights the current state of agent-based modeling visualization tools - each offering distinctive advantages yet facing noteworthy technical limitations. Repast Simphony provides sophisticated visualization but is limited to single-machine execution. NetLogo offers excellent educational value and web accessibility but struggles with large-scale simulations. Distributed MASON, despite its promising architecture, remains in the early prototype stage with reliability concerns. Also, with the use of OpenMPI, it provides a learning curve. These limitations highlight why MASS combined with Cytoscape presents an attractive solution. MASS provides true distributed

computing capabilities that overcome the single-machine limitations of Repast and NetLogo, while avoiding the stability issues of Distributed MASON. Cytoscape was chosen as our visualization platform because it offers extensive plugin support, Strong network visualization capabilities, real-time interaction features and cross-platform compatibility.

However, we can learn valuable lessons from these systems:

- From Repast Simphony: The importance of well-factored abstraction and extensible plugin architecture.
- From NetLogo: The value of user-friendly interfaces and interactive parameter modification.
- From Distributed MASON: The benefits of clear separation between simulation and visualization components.

Chapter 4. MASS GUI Development and Enhancement

This section introduced some major architectural design of the applications. Combining the enhancements we implemented in the MASS Java library, MASS Cytoscape, and Web GUI, in alignment with previously established goals. Building on the implementations by earlier students, we refactored the code to boost performance and introduced new functionalities. Ultimately, we integrated these three independent applications into a single, integrated application, enhancing the user experience for MASS users

4.1 Integration Design

To address the challenges of fragmented InMASS application usage, we designed and implemented an all-in-one solution to optimize user experience. By design, users can now utilize Cytoscape as their primary interface for most operations, eliminating the need to switch between multiple applications. Since version 3.0, Cytoscape has adopted the OSGi framework, which provides a dynamic module system for Java applications. This framework enables both Cytoscape core functions and external plugins to evolve independently without affecting unrelated functionalities.

Leveraging Cytoscape's OSGi framework, we were able to integrate all existing features into Cytoscape Desktop while preserving their original design and implementations. OSGi's modular architecture allowed us to efficiently encapsulate web browser and CLI functionalities as Cytoscape plugins, providing embedded browsing and direct command execution capabilities within the main application.

The OSGi framework significantly simplified our development process by providing:

- A proven modular architecture for Java applications

- Clear specifications for plugin development
- Robust service management capabilities
- Flexible module communication mechanisms

Figures 4.1 and 4.2 illustrate the architecture and workflow of MASS Cytoscape. Figure 4.1 shows the component diagram, representing the structural organization of MASS Cytoscape. The system consists of two main parts: the Cytoscape Application and the Remote Workspace. Within the Cytoscape Application, there are four essential plugins: MASS Control Panel, SSH-CLI, Web GUI, and Visualization Canvas. The Remote Workspace contains DSLab machines running the InMASS application, which includes various simulation applications such as Closest Pair Points, Triangle Counting, and etc.

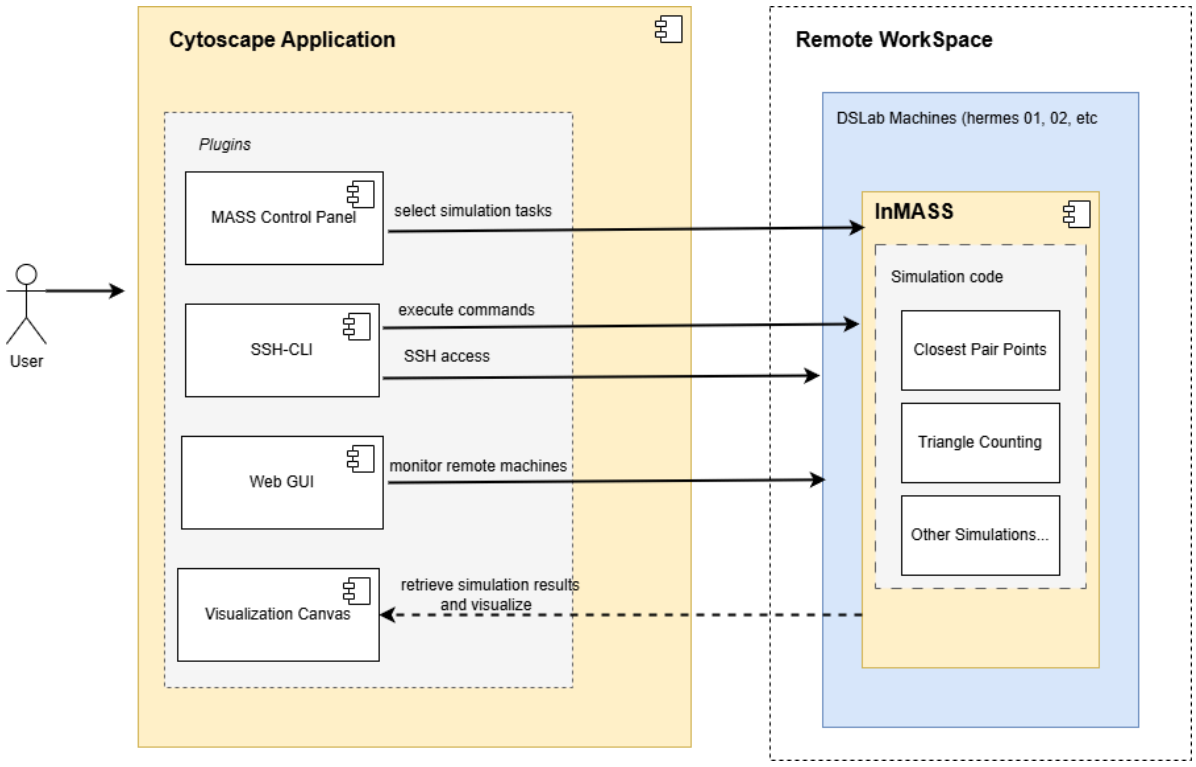


Figure 4.1 : Component Diagram for MASS Cytoscape

Figure 4.2 presents the activity diagram, demonstrating the workflow of user interactions and system operations. To begin, the user opens Cytoscape (1), which initiates the parallel activation of three core plugins. The user interacts with the MASS Control Panel (2) to configure remote machine settings, while simultaneously, SSH-CLI (3) prepares for command execution, and the Web GUI (4) automatically starts in the background. The MASS Control Panel enables users to input remote machine information and establish SSH connections by creating a socket in MASSConfig. Through SSH-CLI, users execute commands to start running InMASS programs and benchmark applications such as the Closest Pair Points (5). While the simulation is running, the Web GUI actively monitors the cluster status (6). After the simulation is completed, users use the Control Panel to retrieve simulation data and information from DSlab Machines (7), and the Visualization Canvas generates the visual representation of the results.

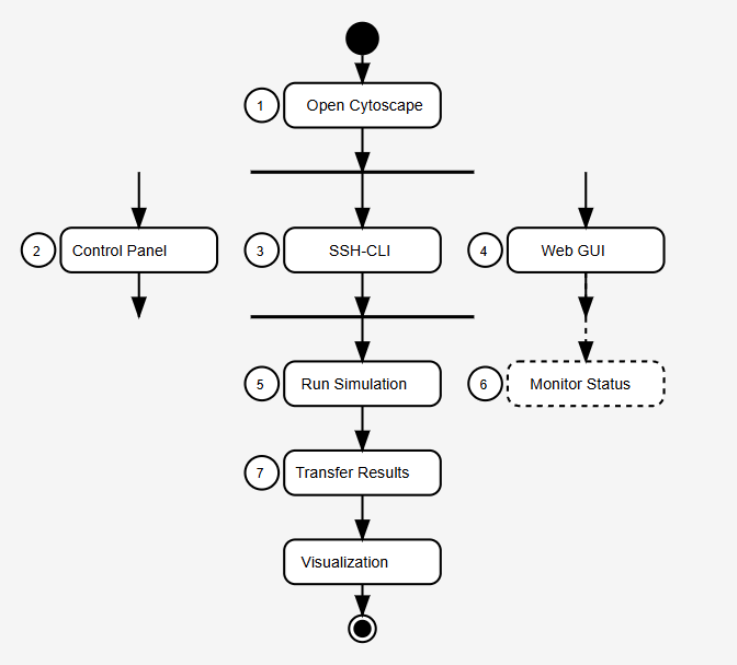


Figure 4.2: Activity diagram for MASS Cytoscape

4.2 InMASS

Our enhancements include optimizing quadtree agent visualization, reproducing agent path tracking, and refining the checkpoint and rollback functionalities. Additionally, we introduced a new Octree class within the InMASS library specifically for simulating the closest pair of points. To enhance visualization features, we established multiple listeners between InMASS and Cytoscape, ensuring efficient transfer of necessary agent information between the applications.

4.2.1 Quadtree Agent visualization

Quadtree-based agent visualization is implemented for the closest pair of points problem. The simulation begins with user inputs including point coordinates, place sizes, and granularity parameters. The system generates agents at initial point locations, which then propagate through space by spawning child agents in adjacent positions. These child agents inherit information from their parents and follow specific spawn-and-kill patterns during spatial exploration. The simulation terminates upon identifying the closest pair through continuous evaluation of agents from different sources within shared subspaces. To capture the complete agent evolution process, we implemented comprehensive agent tracking functionality.

As shown in Listing 4.1, the agent tracking system implements monitoring through the analyzeTrackingResults method (line 1). This functionality operates at two levels: individual agent movement tracking and population distribution analysis. At the individual level, the system maintains an AgentHistoryCollection (line 3) that records each agent's complete itinerary, iterating through all agents (lines 6-19). For each agent, it captures temporal information via getTime() and spatial position via getPlaceLinearIndex() (lines 11-12). The tracking system also monitors agent status, recording whether agents remain active through isAlive() and if they

successfully completed their objectives via isSuccessful() (lines 17-18). At the population level, the system maintains a temporal map of agent distributions across different places using nested data structures Map<Long, Map<String, Integer>> (line 22). This allows for detailed analysis of agent population dynamics over time, tracking how many agents occupy each vertex at different iterations of the simulation (lines 24-28), with the specific agent counts printed for each place.

Listing 4.1: Implementation of agent tracking analysis

```
1  private void analyzeTrackingResults(QuadTreePlaces places) {
2    // Retrieve tracking history
3    AgentHistoryCollection collection = (AgentHistoryCollection) places.getAgentHistory();
4
5    System.out.println("\nAgent Movement Analysis:");
6    for (AgentItinerary itinerary : collection.getItineraries()) {
7      System.out.print(itinerary.getAgentName() + "\t");
8
9      // Print complete path
10     for (AgentItinerary.Visit v : itinerary.getHistory()) {
11       System.out.print("[t=" + v.getTime() +
12                        ",p=" + v.getPlaceLinearIndex() + "] ");
13     }
14     System.out.println();
15
16     // Print success/termination status
17     System.out.println("Active: " + itinerary.isAlive() +
18                        " Success: " + itinerary.isSuccessful());
19   }
20
21   // Print agent population history
22   Map<Long, Map<String, Integer>> agentNumHistory = places.getAgentNumHistory();
23   System.out.println("\nAgent Population History:");
24   agentNumHistory.forEach((iteration, distribution) -> {
25     System.out.println("Iteration " + iteration + ":");
26     distribution.forEach((vertexId, count) ->
27       System.out.println("  Place " + vertexId + ": " + count + " agents"));
28   });
29 }
```

Listing 4.2: Quadtree's Agents and Places initialization

```
1  QuadTreePlaces places = new QuadTreePlaces(1, ClosestPairPlace.class.getName(),
2      dimensions, input_filename, granularityEnhancer, initArgs);
3
4  // Initialize agent tracking
5  if (enableAgentTracking) {
6      // Register agent class for tracking
7      places.callAll(-2, ClosestPairAgent.class.getName());
8      MASS.getLogger().debug("Initialized agent tracking for ClosestPairAgent");
9  }
10
11 // Create initial agents with tracking enabled
12 double[] dummy_originalCoordinates = new double[dimensions];
13 Arrays.fill(dummy_originalCoordinates, -1.0);
14 ClosestPairAgentArgs init_argument = new ClosestPairAgentArgs(dummy_originalCoordinates)
15 Agents agents = new Agents(2, ClosestPairAgent.class.getName(), init_argument, places);
```

As illustrated in Listing 4.2, agent tracking is initialized in the ClosestPair class's main method prior to agent creation. This is accomplished by setting enableAgentTracking flag (line 5) and registering the ClosestPairAgent class for monitoring through places.callAll() (line 7). The system logs the initialization of tracking functionality for ClosestPairAgent (line 8).

After tracking initialization, the code proceeds to create initial agents (lines 11-15). First, it initializes a dummy coordinate array filled with -1.0 values (lines 12-13). These coordinates are then used to create initialization arguments for the ClosestPairAgent (line 14). Finally, the Agents collection is instantiated with these parameters, connecting it to the QuadTreePlaces structure (line 15).

The tracking code integrates with the QuadTreePlaces structure, which is initialized with specific parameters including dimensions and granularity settings (lines 1-2). This setup ensures that the tracking system can monitor agents as they navigate through the distributed space. This initialization allows the system to track:

1. How Agents moved through the QuadTree structure

2. The pattern of Agent spawning and termination
3. The efficiency of closest pair discovery
4. The load distribution across different regions of the space

4.2.2 Octree Implementation

Before diving into the simulation, it's necessary to understand the hierarchy and relationships of the MASS system. As illustrated in Figure 4.3, the system architecture consists of two main hierarchical structures: the agent hierarchy and the place hierarchy.

For agent class hierarchy (shown in the right branch of Figure 4.3), the major classes include Agent, AgentsBase, Agents, OctTreeAgent, and ClosestPairAgent. Agent is the foundational abstract class that defines basic agent properties and behaviors. Each Agent contains essential attributes including a unique id and current location (place), while providing fundamental lifecycle management methods: spawn() for agent creation, kill() for termination, and migrate() for movement.

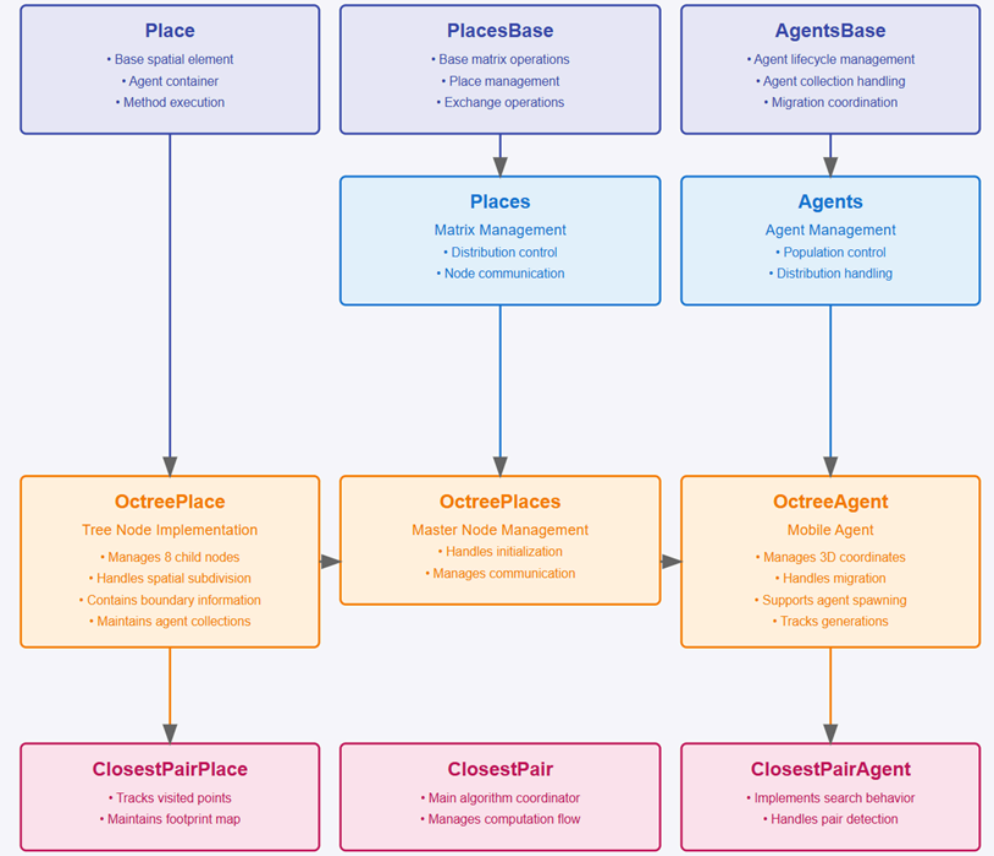


Figure 4.3 Class Hierarchy and Relationships in InMASS System

Building upon this foundation, AgentsBase manages collections of Agent instances and handles lifecycle operations, implementing crucial parallel computing mechanisms through methods such as callAll() for parallel agent operations and manageAll() for state change management. The Agents class inherits from AgentsBase, offering a more accessible interface for applications by encapsulating group creation and management operations, including load balancing and distribution strategies.

For place class hierarchy (illustrated in the left and middle columns of Figure 4.3), the system is built on Place as the fundamental spatial unit, maintaining spatial coordinates and managing agent interactions. PlacesBase supervises Place collections, implementing spatial

partitioning and management operations. The Places class extends this functionality with matrix management capabilities including distribution control and node communication. OctTreePlace provides spatial management through various getter methods for accessing boundary information, dimensional data, and maintaining the octree structure through specialized child getters (getTopLeftFront(), getBottomRightBack(), etc.).

Specialized agent implementations include OctTreeAgent, designed specifically for octree spatial structures with enhanced 3D space movement capabilities, and ClosestPairAgent, which extends OctTreeAgent to solve the closest pair problem. The ClosestPairAgent implements essential operation constants (MIGRATE, SPAWN_AGENTS, PROPAGATE, etc.) and methods for managing agent lifecycle (killDuplicates(), killParent()) and pair detection (collectPairs()). The agent maintains state through originalCoordinates and provides interfaces for movement control through migrate() and state reporting via printAgent(). The complete class structure and implementation details are provided in Appendix D.

The ClosestPair class serves as the computation coordinator, managing initialization parameters (input_filename, dimensions, granularityEnhancer) and providing core methods like computeClosestPair() and calculateInitialBoundary(). Working in conjunction, ClosestPairPlace extends OctTreePlace with specialized mechanisms for tracking visited points (IS_VISITED) and managing footprint distributions through footPrintMap data structures.

In the context of finding closest pairs in 3D space, the system uses octree spatial partitioning and agent propagation strategies. The process initiates with recursive spatial division using OctTreePlaces. Agent propagation alternates between 26 directions (Moore neighborhood) in odd generations and 6 directions (von Neumann neighborhood) in even generations, with

parent agents being cleared to maintain efficiency. The algorithm determines sub-space division based on distance calculations and boundary dimensions, utilizing the PairOfPoints class to maintain coordinate pairs and source agent information for distance calculations and comparison.

During execution, it checks for closest pairs by identifying agents from different source points within the same subspace. When potential pairs are detected, their Euclidean distance is calculated and compared against the current minimum. The process continues through agent spawning, movement, and cleanup cycles until the closest pair is identified, utilizing a footprint tracking mechanism to prevent redundant checks within sub-spaces.

4.3 Web GUI

The web GUI, shown in Figure 4.4, has been enhanced to provide cluster monitoring and control capabilities through integration with Cytoscape. Our new implementation addresses several limitations in the previous version and introduces new functionalities for remote machine monitoring.

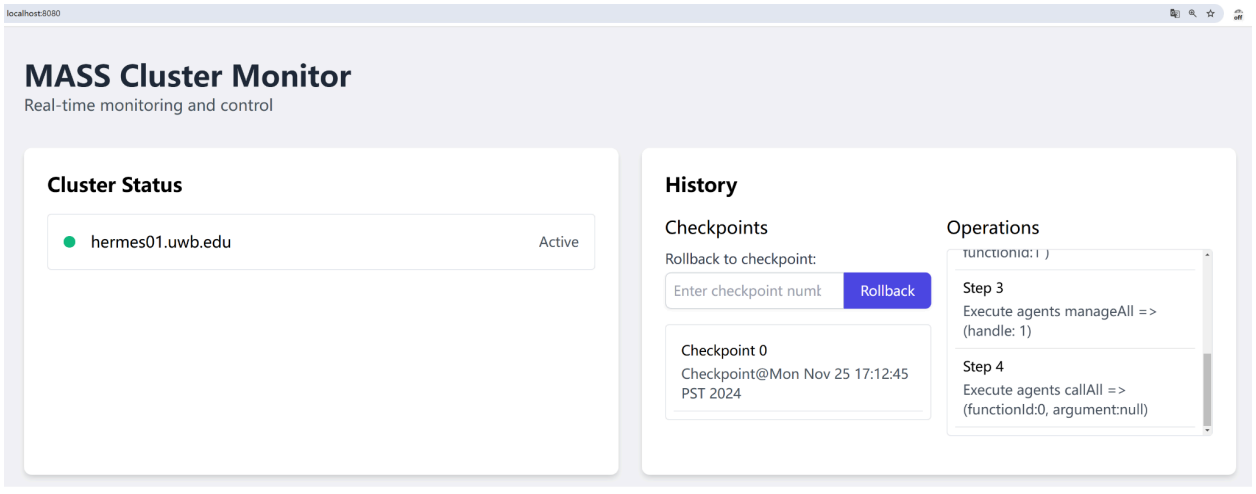


Figure 4.4: integrated Web GUI with cytoscape's built-in browser

The system architecture maintains its Vert.x-based backend and React.js frontend structure, with visual improvements and functionality extensions to both components. During the InMASS startup process, the web server initialization follows a systematic approach. As shown in Listing 4.3, the initialization sequence begins by importing essential components from the MASS library. This includes:

- Core MASS functionalities through the wildcard import (line 2)
- Logging utilities via LogLevel class (line 3)
- The Vert.x framework core component (line 4)
- The MyVerticle component from MASS router package (line 5)

When InMASS begins execution, it first loads necessary configurations and initializes the MASS library along with the logging system. Following this initialization phase, the system creates a Vert.x instance that serves as the foundation for the web server. Through Vert.x's deploy method, this instance deploys a MyVerticle component, which establishes the HTTP server and configures various endpoints for system monitoring and control.

Listing 4.3: Web GUI Initialization

```
1  // import statement
2  bufferedWriter.write(str:"import edu.uw.bothell.css.dsl.MASS.*;\n");
3  bufferedWriter.write(str:"import edu.uw.bothell.css.dsl.MASS.logging.LogLevel;\n");
4  bufferedWriter.write(str:"import io.vertx.core.Vertx;\n");
5  bufferedWriter.write(str:"import edu.uw.bothell.css.dsl.MASS.router.MyVerticle;\n");
```

On the backend, several RESTful endpoints have been enhanced for system monitoring and control. As shown in Listing 4.3, the "/calls" endpoint (lines 1-10) retrieves execution history by accessing MHistory's getInfos() method (line 5) and returns a formatted JSON array

of recorded operations using encodePrettily() (line 9). The "/rollback/:step" endpoint (lines 12-20) implements state restoration functionality by parsing the step parameter (lines 16-17) and invoking MASS.rollback() with the specified checkpoint (line 18), enabling precise system state management. Additionally, while not shown in Listing 4.3, the "/sync" endpoint provides real-time cluster status monitoring through MASS.getClusterStatus(). Both endpoints in Listing 4.3 follow proper HTTP protocol by setting appropriate content-type headers - "application/json" for /calls (line 3) and "text/plain" for /rollback (line 14).

A key enhancement addresses the previous limitation where monitoring was restricted to the local machine hosting the web server. The new implementation enables remote machine monitoring through integration with Cytoscape's built-in browser. This is particularly crucial for applications running on lab machines that lack graphical display capabilities.

To enable remote monitoring, we developed a Cytoscape plugin that establishes an SSH tunnel between client devices and lab machines. The tunnel forwards requests from the local port (default: 8080) to the corresponding port on the remote machine where the web service is running. When requests are made, the web server processes them through these endpoints (Listing 4.3, lines 1-20) and returns necessary resources through the established tunnel, allowing Cytoscape's built-in browser to render the web application's frontend.

The enhanced implementation maintains the original RESTful API structure while ensuring proper functionality of all endpoints. The system now effectively handles:

- Remote machine status monitoring through the "/sync" endpoint
- Operational history tracking via the "/calls" endpoint
- System state management using the fixed "/rollback/:step" endpoint

Once initialized, the server remains active throughout the JShell session, processing requests and maintaining system state until the session is terminated.

Listing 4.3: RESTful endpoints used in Web GUI

```
router.route("/calls").handler(ctx -> {
    HttpServerResponse response = ctx.response();
    response.putHeader("content-type", "application/json");

    List<String> infos = MHistory.getInstance().getInfos();
    JsonArray jsonArray = new JsonArray();
    infos.forEach(jsonArray::add);

    response.end(jsonArray.encodePrettily());
});

router.route("/rollback/:step").handler(ctx -> {
    HttpServerResponse response = ctx.response();
    response.putHeader("content-type", "text/plain");

    String step = ctx.request().getParam("step");
    int to = Integer.parseInt(step);
    MASS.rollback(to);
    response.end("Rollback to step: " + to);
});
```

4.4 Cytoscape

We developed plugins for 3D visualization, network export, MASS-CLI, and web-GUI in Cytoscape. These additions are designed to improve the visualization of InMASS simulations and streamline the integration of applications.1. Table 1 provides a summary of the plugins created to facilitate InMASS simulation visualization in Cytoscape.

Table 4.1: List of Cytoscape Plugins and Their Usage

Name of Plugin	Usage
3D-Visulization	Renders agents and places in 3D space
export-network	Sends Cytoscape networks to MASS
import-agent	Retrieves and store agent history information
import-network	Simplifies in-memory network from MASS and rebuilds in Cytoscape
mass-agents	Creates InMASS-Control Panel and integrates other plugins
ssh-CLI	Facilitates remote access to Web GUI and remote machines, provides terminal interface for smoother simulation development

4.4.1 SSH-command line interface

As illustrated in Figure 4.5, the SSH Command Line plugin implements a comprehensive terminal emulation within Cytoscape, featuring a dual-purpose interface that seamlessly integrates regular SSH commands with JShell session interactions. The core implementation centers around the SSHCommandLine class, which, as shown in the Figure 4.6, contains essential components like outputArea, commandField, and commandHistory for handling terminal functionality through custom enhancements and JSch library integration. The CyActivator class orchestrates the entire system by creating and managing the relationships between SSHCommandPanel, SSHServiceManager, and QuickStartServer components. The

interface processes input through a dedicated outputThread that monitors the SSH stream, implementing custom text processing for ANSI escape sequences and control characters.

The command input area functions as a multi-line text editor with two distinct operational modes: Remote Interface Mode for immediate command execution using Enter key, and JShell Development Mode utilizing Shift+Enter for multi-line code input. The SSHServiceManager class manages these operations through its connection handling methods and credential management functions. The plugin implements sophisticated features including command history navigation (managed by navigateHistory() method), intelligent tab completion, and directory tracking through the SSHConnectionManager's updateCurrentDirectory() functionality.

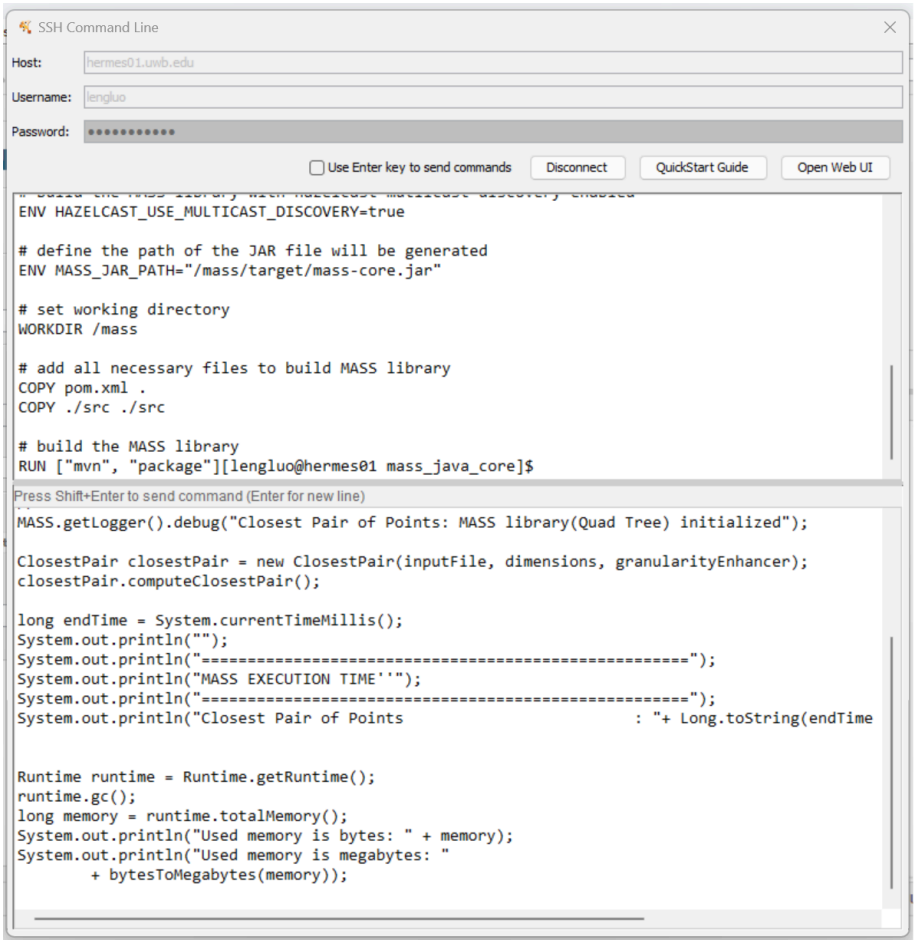


Figure 4.5: SSH Command Line plugin

The integration with JShell's REPL (Read-Eval-Print Loop) makes development more fluid and interactive. When users input code through the CLI, JShell categorizes the input into distinct code snippets based on their type (imports, variable declarations, class definitions, etc.). This feature, combined with JShell commands like /save, /open, and /reset, enables efficient code testing and modification without requiring a complete Java application compilation. For example, developers can:

1. Write and test partial functionalities through the multi-line input box
2. Use /save to persist working code snippets to .jsh files
3. Execute /reset to clear the session state
4. Utilize /open to reload saved functionalities into a fresh session

This workflow, coupled with InMASS's debug messaging and logging system, streamlines the development process by eliminating the need for frequent recompilation and context switching between local, remote, and JShell environments. The implementation includes real-time output processing with specialized UTF-8 character handling and buffer management, alongside robust session management for JShell integration. This integrated approach enables developers to write, test, and modify simulation code efficiently, particularly beneficial for MASS simulation development.

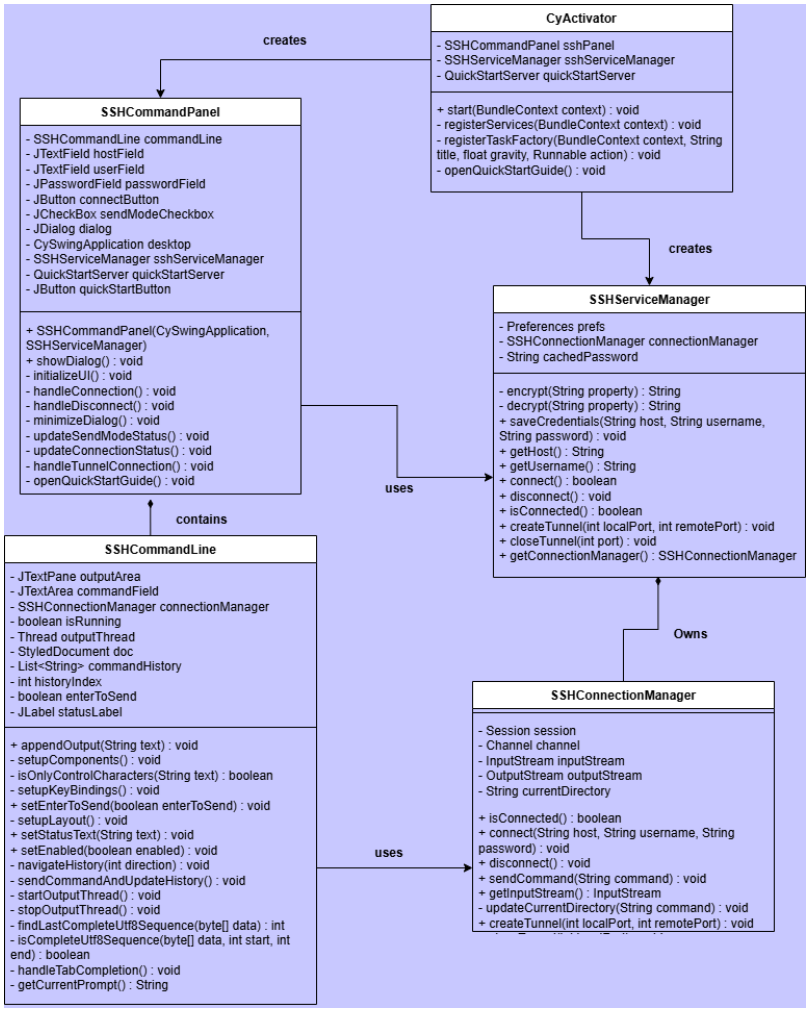


Figure 4.6: Class Diagram of SSH-CLI Plugin

4.4.2 Web GUI

On the Cytoscape side, we implemented multiple classes to enable secure remote access to the Web GUI. The SSHCommandPanel implements the "Open Web UI" button that triggers tunnel creation, utilizing encoded passwords stored in memory to establish secure connections. The SSHConnectionManager handles the actual tunnel creation using JSch library, while the tunnel management system automatically handles cleanup when SSH sessions terminate.

The plugin maintains active tunnels throughout SSH sessions, with automated lifecycle management tied to the connection state. Working in conjunction with the Disconnect button, our

implementation provides comprehensive connection management through three key mechanisms:

1. Connection State Management - handles SSH session termination and UI state reset
2. Tunnel Cleanup - ensures proper closure of port forwarding tunnels
3. Window Closing Handler - manages graceful shutdown of active connections

This integrated approach effectively prevents connection issues and resource leaks, particularly addressing problems that could arise from inactive web sessions or occupied ports. When users disconnect, the system automatically releases all resources, closes tunnels, and resets the connection state, ensuring clean termination of remote sessions and maintaining system stability.

4.4.3 3D space

We implemented a 3D space visualization plugin for MASS-Cytoscape by extending the existing Cy3D plugin. The visualization leverages OpenGL for rendering, taking advantage of its open-standard specification that enables both 2D and 3D graphics programming without requiring additional software installation.

In the visualization, octree nodes are represented as blue cubes in 3D space. When the simulation concludes, users can select individual places to inspect agent information and states within that specific location. The visualization implements a camera entity containing coordinates and movement operations, enabling users to zoom in/out, rotate views, and navigate the 3D space effectively to better locate and examine specific places.

As shown in Figure 4.7, our implementation visualizes a 5x5x5 cube structure, demonstrating the three-dimensional representation of the octree data structure. Each smaller

cube within this 5x5x5 framework represents a distinct place in 3D space, providing users with a clear spatial understanding of the simulation environment. While the current implementation successfully visualizes the basic 3D structure, accurately rendering complete agent paths in 3D space remains a technical challenge that we plan to address in future developments. This visualization enables users to better understand the spatial relationships and agent distributions within the MASS simulation space.

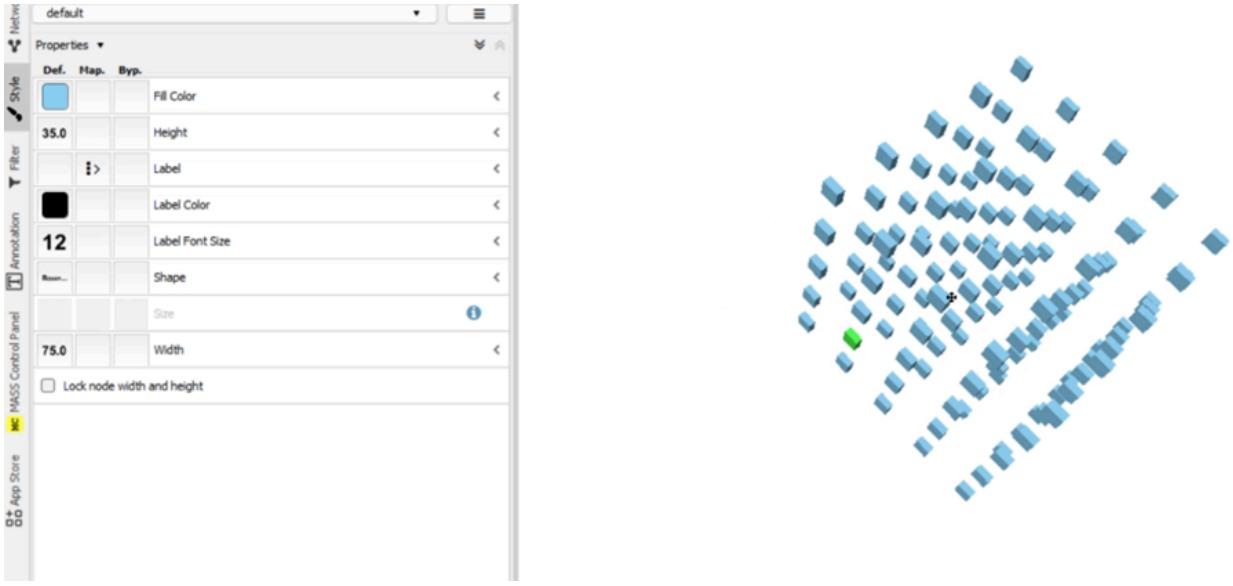


Figure 4.7: 3D space visualization in Cytoscape

Chapter 5. Evaluation of MASS Functionality and Usability

This section evaluates the functionality and usability of the MASS GUI, examining aspects such as the execution performance of InMASS, the functionality of the Web GUI, the usability of Cytoscape, and includes discussions on the findings and implications of these evaluations.

5.1 Comparison Between Current and Previous Workflows

The original MASS Cytoscape application operates through a complex sequential workflow requiring multiple coordinated components. Users must begin by opening two separate terminals: one for activating the InMASS application and another for launching the Web GUI server. The Web GUI terminal must remain active throughout the entire session, as its closure would terminate the connection between the web interface and the application. Once these components are running, users can execute their simulations through the InMASS application. After simulation gets completed, the process continues with opening Cytoscape, importing the simulation data, and using Cytoscape's visualization capabilities to analyze the results.

The code modification process, illustrated in Figure 5.1, introduces additional workflow steps. When code changes are needed, users must terminate the InMASS application through the terminal, modify the source code, recompile the JAR file, move the compiled JAR to the designated directory, and relaunch InMASS to verify their changes. Each compilation cycle consumes approximately 30 seconds, and any compilation errors force users to repeat the entire process. The workflow complexity increases further with the need for an additional terminal to manage file operations and unterminated remote machines.

This development environment creates bottlenecks through its disconnected components and manual processes. The need to coordinate multiple terminals, handle files manually, and wait

for compilation extends development time, particularly during phases of rapid code iteration where developers need to test multiple changes. The requirement to cycle through the complete modification process for each code change, combined with the risk of terminal connection disruptions, makes the development and testing process more time-consuming than necessary.

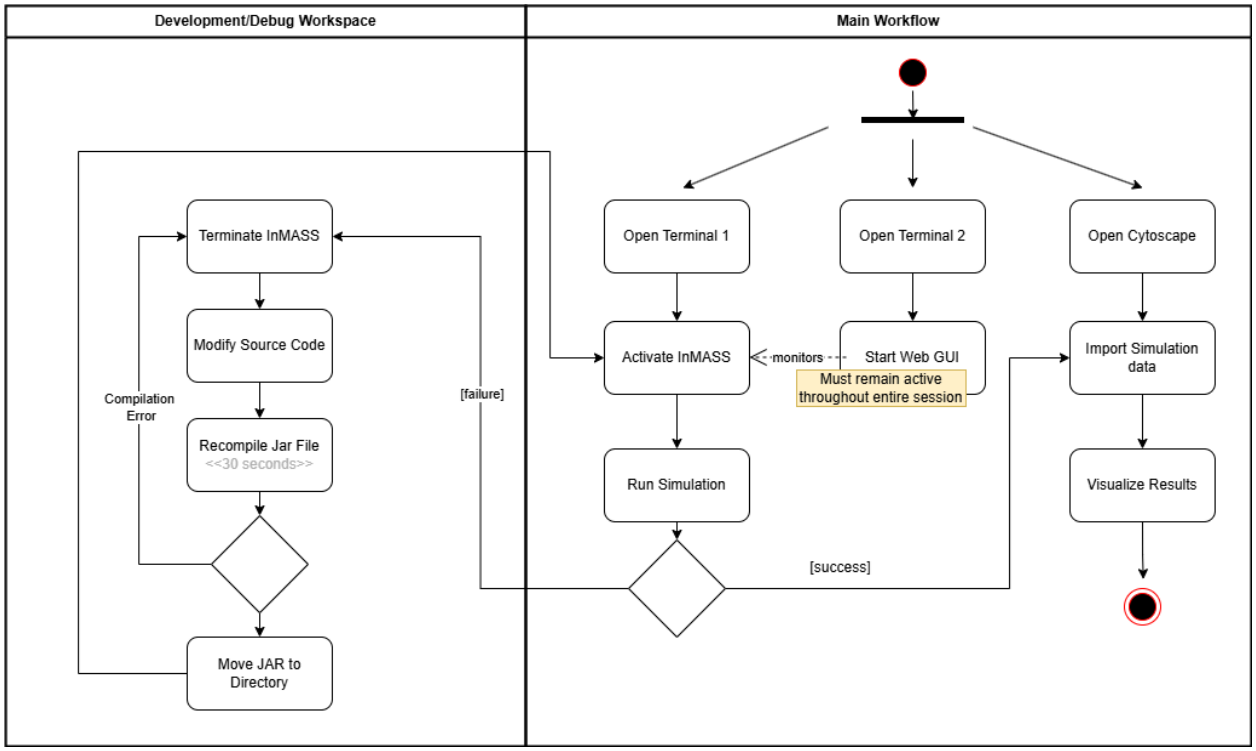


Figure 5.1: Previous workflow of InMASS development and visualization

The current workflow, shown in Figure 5.2, offers a streamlined development experience through integrated tools within Cytoscape. Users begin by opening Cytoscape and connecting to remote machines using the CLI plugin, which automatically initializes the Web GUI server. This integration eliminates the need for manual Web GUI management, as it is now coupled with the SSH session lifecycle. Through the CLI plugin, users can activate InMASS and execute simulations while maintaining full access to development features. This terminal interface

provides advanced capabilities including code snippet management through .jsh files, enabling persistent development without the need for constant recompilation. The system incorporates robust error handling with clear feedback messages and secure credential storage, ensuring session continuity.

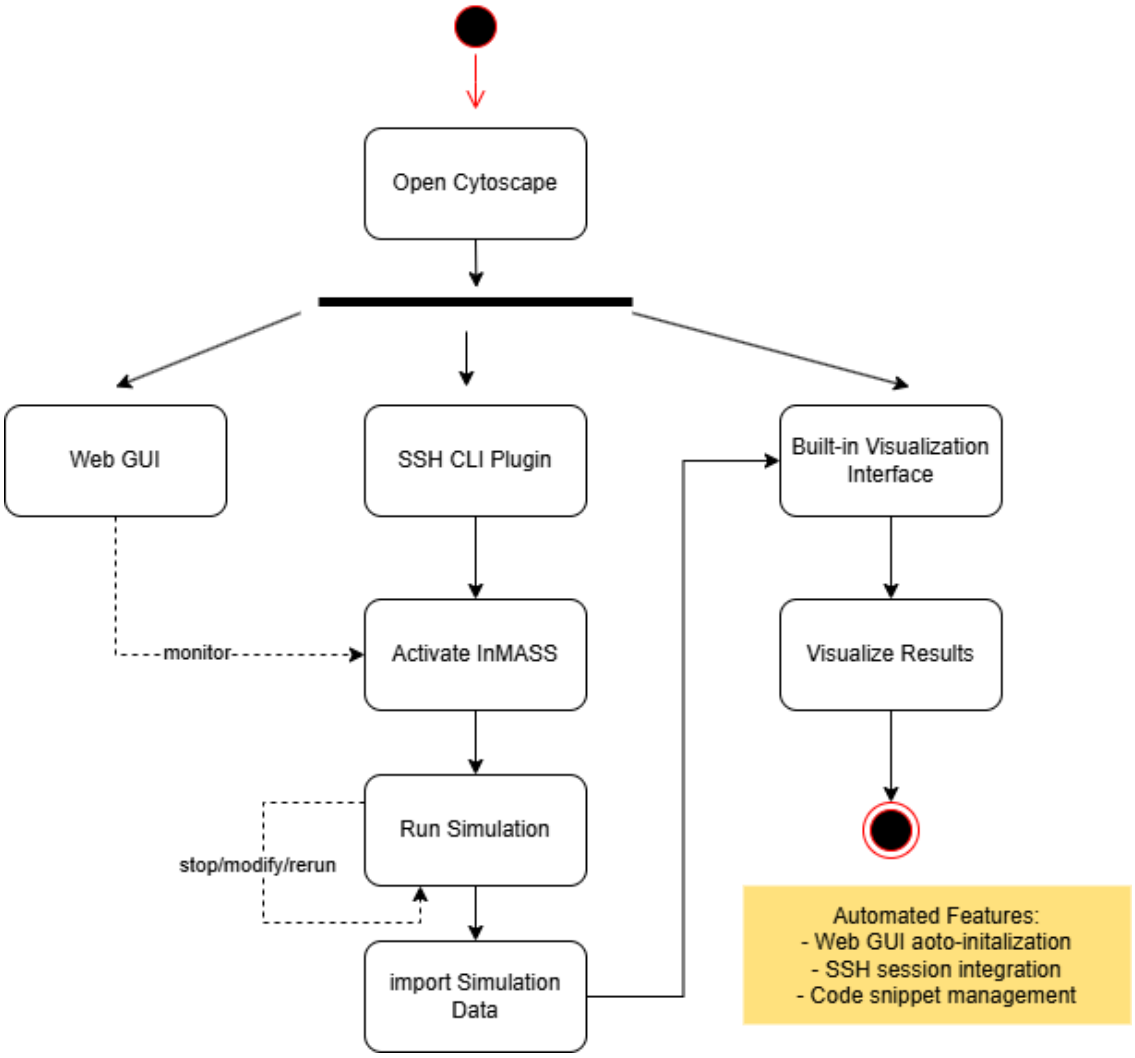


Figure 5.2: Current workflow of InMASS development and visualization

Key improvements in this workflow include:

- Automated Web GUI lifecycle management
- Persistent development environment through .jsh files
- Integrated error handling and feedback system

- Seamless switching between administration and development tasks
- Direct monitoring of simulation status through the Web GUI
- Ability to modify and rerun simulations without session termination

This integrated approach enhances the development experience by eliminating the need for manual terminal management and reducing compilation cycles, particularly beneficial for MASS simulation development where maintaining development context is crucial.

5.1.1 Time evaluation

As shown in Table 2, the original workflow requires approximately 2.5 minutes to complete a debug cycle, excluding simulation time. The debug cycle includes multiple manual steps: initial InMASS activation (15 seconds), code modification requiring four distinct actions like locating folders and files (1 minute), compilation and JAR file management (1 minute), and various system restart operations. Each modification cycle forces developers to spend significant time on operational tasks rather than actual development work.

Table 2 also reveals how the current workflow reduces this overhead to approximately 1 minute - a 60% improvement in efficiency. Instead of the original's complex code modification process, developers now simply edit code snippets and save them as .jsh files (30 seconds). The entire compilation and JAR file management step is eliminated, and system cleanup is streamlined through a simple reset command (5-7 seconds) rather than requiring full terminal management.

Table 5.1: Comparison between original workflow and current workflow

Step	Original workflow	Time	Current workflow	Time
1. Initial Setup	Activate InMASS	15 sec	Activate InMASS	15 sec
2. Simulation	Run simulation	*	Run simulation	*

3. Stop Simulation	Terminate simulation using MASS.finish()	1sec	Terminate simulation using MASS.finish()	1sec
4. Cleanup	Quit Jshell	3sec	Reset Jshell session using /reset	5-7 sec
5. Code Modification	1. Locate folder 2. Open code editor 3. Locate code 4. Make changes	1 min	1.Edit code snippet 2. Save as jsh file	30 sec
6. Rebuild	Recompile and move JAR files	1 min	Not needed	-
7. Restart	Activate InMASS	15 sec	Not needed	No need
8. verify change	Run simulation	*	Load jsh file	5 sec
Total Fixed Time	-	~ 2.5 min	-	~ 1 min
* Simulation time varies based on input files but is consistent between workflows				

5.2 Evaluation based on user feedback

The evaluation collected responses from 10 participants: five DSLab students directly involved in MASS projects, four students from CSS 534 (Parallel Programming in Grid and Cloud) class, and one Computer Science student outside of both groups. The survey, as shown in Table 5.2, consisted of 10 questions covering system information, user experience, performance comparison, add-on simulation requirements, and interface preferences.

The survey questions were structured to enable quantitative and qualitative evaluation through multiple complementary approaches. For performance assessment, comparative metrics using five-point scales were employed to measure startup time and ease of use relative to existing ABM simulators. Installation experience was similarly quantified using a standardized scale, enabling calculation of mean difficulty scores and experience distribution. Feature

prioritization was achieved through multiple-select questions about simulation scenarios and interface features, allowing frequency analysis of user requirements.

To ensure broad applicability, the survey collected platform distribution data through operating system categorization, providing quantifiable metrics of system compatibility across different user environments. Interface preferences were captured through direct selection questions, yielding clear distributions to guide design decisions. While several questions were open-ended, their responses were structured for systematic categorization, enabling analysis of most-cited features, user visualization preferences, and key improvement areas.

This methodological combination of structured ratings, multiple-choice selections, and categorizable open-ended responses provides a foundation for quantitative analysis of InMASS's effectiveness, user satisfaction, and potential enhancements. The complete survey responses and raw data are provided in Appendix B.

Table 5.2: Survey Questions for InMASS Evaluation

No.	**Question**	**Response Format**
1	What operating system are you using?	Multiple choice: Windows/macOS/Linux/Other
2	How would you rate the difficulty of following the installation instructions?	Scale: Very Easy to Very Difficult
2a	If you experienced any difficulties, please describe them:	Open-ended response
3a	Compared to other ABM simulators - Startup Time	Scale: Much faster to Much slower
3b	Compared to other ABM simulators - Ease of Use	Scale: Much easier to Much harder
4	Which simulation scenarios would you be interested in seeing implemented in InMASS?	Multiple select with 9 options

5	What types of agent/place visualizations would be most useful for your work?	Open-ended response
6	Which user interface design would you prefer?	Single select: 3 options
6a	Please explain your choice:	Open-ended response
7	If you selected a web-based interface, which features would be most important?	Multiple select with 6 options
8	What aspects of InMASS do you find most valuable?	Open-ended response
9	What aspects of InMASS could be improved?	Open-ended response
10	Do you have any additional suggestions or comments?	Open-ended response

The system demonstrates cross-platform compatibility, with five respondents using Windows and five using macOS. For installation difficulty in Question 2, six respondents rated it as "Easy," four as "Moderate". The installation process currently requires manual plugin installation, involving copying JAR files into the Cytoscape apps directory, suggesting that future releases would benefit from uploading these plugins to the Cytoscape Apps store for simpler installation procedures.

Table 5.3: System Information and Installation Experience (Q1&Q2)

Category	**Count (n=10)**	**Percentage**
Q1. Operating System		
Windows	5	50%
macOS	5	50%
Q2. Installation Difficulty		
Easy	6	60%
Moderate	4	40%
Q2a. Issues		

Port Conflicts	2	20%
Cytoscape Task Issues	2	20%
VPN/SSH Access Issues	1	10%
Multiple checkpoints conflict	1	10%
Minior usability issues	2	20%
No Issue Reported	2	20%

In Question 2 about specific difficulties, two users reported port 8080 conflicts affecting both local and remote machine configurations. Users reported difficulties in modifying Cytoscape's port settings for the web UI, and some found that even after attempting to change node configurations in nodes.xml, these modifications failed to persist through system resets. Two users encountered Cytoscape Task window issues, and one experienced VPN/SSH access problems. Three users reported no installation issues.

Question 3 addressed performance comparisons with existing ABM platforms. Only three respondents had prior ABM experience, while seven had no prior experience. Among the three experienced users, startup time ratings varied: one reported "Much faster," one "Slightly faster," and one "Slightly slower." Two users rated InMASS as "Slightly easier" to use and one rated "Slightly harder."

Table 5.4: Distribution of Participants' Experience with Other ABM Tools (Q3)

Experience Level	**Count (n=10)**	**Percentage**
Prior ABM tool experience	3	30%
No Prior Experience	7	70%

Table 5.5: Performance Comparison between MASS Cytoscape and Other Tools (Q3)

Performance Metrics	**Count (n =3)**	**Percentage**
Q3a. Startup Time		
Much Faster	1	33%
Slightly Faster	1	33%
Slightly Slower	1	33%
Q3b. Ease of Use		
Slightly Easier	2	66%
Slightly harder	1	33%

Question 4 about desired simulation scenarios revealed that Path Finding was the most requested feature (5 users, 50%), followed by Flocking/Swarming Behavior (4 users, 40%). Network Community Detection and Collision Detection each received interest from three users (30%). Two users (20%) each expressed interest in Load Balancing and Spatial Clustering. This distribution indicates strongest user interest in movement and network analysis simulations.

Table 5.6: Desired Simulation Scenarios (Q4)

Simulation Type	**Number of Respondents Interested**
Path Finding	5
Network Community Detection	3
Collision Detection	3
Flocking/Swarming Behavior	4
Load Balancing Scenarios	2
Spatial Cluster	2
KNN	1
Custom: graph Database queries	1

Questions 6 and 7 focused on interface preferences, revealing two distinct perspectives on system architecture. Of the ten respondents, six favored transitioning to a complete web-based interface, citing workflow disruptions from context switching and suggesting that modern frameworks could enhance functionality while reducing development time. These users emphasized how eliminating interface fragmentation would create a more cohesive experience for MASS-based operations.

Two respondents preferred maintaining the current hybrid design (Cytoscape + web services), while two expressed no preference. Those favoring the hybrid approach noted that converting Cytoscape's visualization canvas would require substantial development effort, particularly given its reliance on desktop-specific graphics libraries and layout algorithms.

Table 5.6: Interface Preferences (Q6)

Interface Preference	**Count (n=10)**	**Percentage**
Web-based Interface	6	60%
Current Hybrid Design	2	20%
No preference	2	20%

For users who preferred a web-based interface, we further investigated their desired features through a multiple-select question. As shown in Table 5.7, built-in data analysis tools and export/import capabilities emerged as the most requested features, with 60% of respondents (6 out of 10) selecting these options. The strong preference for export/import capabilities reflects users' need for convenient simulation management - specifically, the ability to save simulation states, share configurations with other users, and rerun previous simulations without manual reconfiguration. This aligns with typical research workflows where experiments often need to be

reproduced or shared across team members. Built-in data analysis tools received equal priority, suggesting users value integrated analysis capabilities within the same interface. Integrated documentation and real-time collaboration features were selected by 40% of participants (4 out of 10), indicating moderate interest in collaborative and learning-support features. Custom visualization layout with 30% of respondents (3 out of 10) selecting this option. These results suggest that users prioritize practical data handling capabilities over interface customization, with a balanced interest in documentation and collaborative features.

Table 5.7: Feature Preferences for Web Based Interface (Q7)

Feature Preference	**Count (n=10)**	**Percentage**
Integrated documentation	4	40%
Real-time collaboration	4	40%
Custom visualization layout	3	30%
Built-in data analysis tools	6	60%
Export/import capabilities	6	60%

Based on responses to Questions 8-10, users identified several valuable aspects of InMASS. The visualization capabilities were frequently highlighted, particularly for their educational value in explaining ABM concepts. Users specifically valued the ability to track agent movements and visualize data structures. Development features received positive feedback, with users appreciating the ability to modify code without recompilation, utilize checkpoints/rollbacks, and work with JShell as an alternative to traditional Java development. The distributed computing aspects of InMASS were noted as valuable, with users praising the Web GUI for monitoring remote machines, the customized CLI, and the relatively straightforward multi-machine configuration process. The demo simulations were also

highlighted for effectively demonstrating both basic functions and more specialized capabilities of the system.

Table 5.8: Most Valuable Aspects of InMASS (Q8)

Category	**Features Mentioned**
Visualization Capabilities	- Simulation visualization for education/explanation - Agent movement tracking - Graph structure visualization
Development Features	- Code modification without recompiling - Checkpoints/rollbacks - JShell integration
Distributed Computing	- Multi-machine monitoring - Web GUI node tracking - Simple configuration
Demo and Examples	- Quick start applications - Specialized simulations - Basic function demonstrations

Regarding areas for improvement (Table 5.9), interface usability emerged as a significant concern. Users found the Cytoscape UI overly complex with too many buttons and unclear functionality. The inconsistency between pop-up windows for the Web GUI/CLI and built-in Cytoscape panels was noted as making workflow management difficult. Documentation needs were emphasized, with users requesting better explanation of features, and comprehensive guides with detailed sample programs. Accessibility improvements were suggested, including the ability to modify simulation parameters directly in Cytoscape without source code changes, and making the system more approachable for users outside the lab. Performance concerns were raised specifically regarding slow initialization when working with multiple machines.

Table 5.9: Areas for Improvement (Q9)

Category	Improvement Suggestions
User Interface	- Simplify Cytoscape UI (too many buttons/options)

	- More descriptive interface - Inconsistent window management between CLI and MASS Control Panel
Documentation	- Better functionality explanations - Sample programs with detailed explanations
Accessibility	- Easier parameter modification without source code changes - More accessible for users outside the lab - Simpler command interface
Performance	- Slow initialization on multiple machines

For question 10, many respondents indicated "None" or expressed satisfaction without additional suggestions, while others provided specific feature requests focusing primarily on expanding graph application capabilities and improving documentation. The suggestion for comparative demos indicates user interest in understanding InMASS's capabilities relative to other ABM platforms.

Table 5.10: Additional Suggestions from User Feedback (Q10)

Category	Suggestions
Graph Applications	- Provide visualization tools to monitor agent interactions and partition performance
Comparative Features	- Provide demos from other ABM applications for comparison
Documentation	- Need better documentation for InMASS applications - Need sample program with functionality explained
Survey Format	- Survey could have been easier as a Google Form

The survey results demonstrate that InMASS successfully achieves its core objectives of making MASS simulations more accessible and manageable, with particular strengths in visualization capabilities, flexible development features, and distributed computing functionality.

Cross-platform compatibility is strong, showing even usage across Windows and macOS platforms.

User feedback highlighted several key improvement areas. Most notably, 60% of users preferred transitioning to a fully web-based interface to reduce context switching and create a more cohesive experience. Built-in data analysis tools and export/import capabilities emerged as the most requested features. Path Finding (50%) and movement-based simulations were identified as the most desired simulation capabilities.

The evaluation reveals a clear roadmap for future development, focusing on interface consolidation, documentation enhancement, and improved accessibility while maintaining the system's technical features.

Chapter 6. Conclusion

This research has enhanced the Multi-Agent Spatial Simulation (MASS) framework by integrating previously fragmented interfaces into a unified, user-friendly platform. The project's achievements can be summarized in three key areas:

6.1 Summary of Achievements

1. Integration and Workflow Optimization: The project successfully unified InMASS, Cytoscape, and the web-based monitoring interface into a cohesive single platform, marking an advancement in usability. Through the implementation of an embedded SSH-CLI within Cytoscape, users no longer need to coordinate multiple terminal windows, thereby optimizing their workflow efficiency. We further developed a seamless connection between simulation execution and visualization components through the OSGi framework, establishing a more systematic user interface.

2. Enhanced Visualization Capabilities: The project advanced the selection of visualization by implementing sophisticated agent tracking in quadtree-based 2D space, while concurrently developing octree data structure visualization for 3D space simulations. Within Cytoscape, we implemented interactive MASS computation capabilities, enabling users to conduct and analyze simulations in real-time.

3. Improved Monitoring and Control: The project enhanced remote access capabilities through the implementation of SSH-tunneled connections for cluster monitoring, enabling researchers to conduct oversight of their simulations remotely. The enhanced web GUI now facilitates real-time performance tracking across multiple machines,

providing users with comprehensive visibility into their distributed systems. Furthermore, the integration of checkpoint and rollback functionalities within the Cytoscape interface has established a more robust simulation management system.

6.2 Limitations

Despite these achievements, several limitations require consideration. The visualization component encounters constraints when rendering complex 3D space simulations, particularly in managing overlapping agents. Current implementation limits the quantity of agents that can be effectively visualized. The integrated application may experience computational overhead during large-scale simulations, while real-time monitoring generates substantial network traffic.

Through user feedback, several operational limitations were identified. Users still need to navigate multiple interfaces within Cytoscape, fragmenting the overall process. Multiple users reported conflicts as the system's services depend on fixed ports, which are commonly occupied by other development tools. An adjustable port configuration is needed to ensure compatibility with users' existing development environments. Additionally, the current documentation structure has resulted in knowledge gaps that create steep learning curves for new users.

6.3 Future Work

Building on our current work and user feedback, several potential directions for future research emerge. For enhanced visualization, the system will implement filtering mechanisms for selective agent visualization and support for custom agent behavior visualization. Performance

optimization will focus on improving network communication efficiency through compressed data transfer protocols and optimized data streaming mechanisms.

We will implement two major enhancements to improve user experience. First, a "click-and-play" interface will replace current command-line operations, making the system more accessible while maintaining advanced functionality for experienced users. Second, we will develop a unified web-based interface, consolidating all tools into a single platform based on strong user preference for streamlined workflows. This redesign will include adjustable port configuration to ensure compatibility with various development environments.

To support user adoption, we will create comprehensive documentation including interactive tutorials, video guides, and example simulations. We will also expand our library of pre-built applications with essential algorithms for graph analysis and spatial computing.

This project has improved the MASS framework's usability while maintaining its distributed computing capabilities. Through these planned improvements, we aim to enhance the system's technical capabilities while making it more accessible to both new users and experienced researchers in the distributed computing community.

REFERENCES

[1] University of Washington, "MASS Java Manual," 2016. [Online]. Available: https://depts.washington.edu/dslab/MASS/docs/MASS%20Java%20Technical%20Manual.pdf

[2] N. Alghamdi, "Supporting Interactive Computing Features for MASS Library: Rollback and Monitoring System," University of Washington, Bothell, WA, 2020.

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

[4] M. Fukuda, C. Gordon, U. Mert, and M. Sell, "An agent-based computational framework for distributed data analysis," IEEE Computer, vol. 53, no. 3, pp. 16-25, 2020.

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

[6] Y. Yang, "Agents Visualization and Web GUI Development in MASS Java," 2024. [Online]. Available: https://depts.washington.edu/dslab/MASS/reports/YifeiYang_whitepaper.pdf

[7] V. Mohan et al., "Automated Agent Migration over Distributed Data Structures," in Proceedings of the 15th International Conference on Agents and Artificial Intelligence, Feb. 2023, pp. 363-371.

[8] P. Shannon et al., "Cytoscape: a software environment for integrated models of biomolecular interaction networks," Genome Research, vol. 13, no. 11, pp. 2498-2504, 2003.

[9] M. J. North et al., "Complex Adaptive Systems Modeling with Repast Simphony," Complex Adaptive Systems Modeling, vol. 1, no. 1, pp. 3, 2013.

[10] D. Blashaw and M. Fukuda, "An Interactive Environment to Support Agent-based Graph Programming," in Draft Paper, Division of Computing and Software Systems, University of Washington Bothell, 2022. [Online]. Available: https://faculty.washington.edu/mfukuda/papers/icaart22.pdf

[11] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with Repast Simphony," Complex Adaptive Systems Modeling, Mar. 2013.

[12] "Zombies - Repast Symphony ReLogo Execution," Repast Suite Gallery. [Online]. Available: https://repast.github.io/gallery/relogo_execution.png

[13] S. Tisue and U. Wilensky, "NetLogo: Design and Implementation of a Multi-Agent Modeling Environment," in SwarmFest, Ann Arbor, May 2004, pp. 9-11.

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

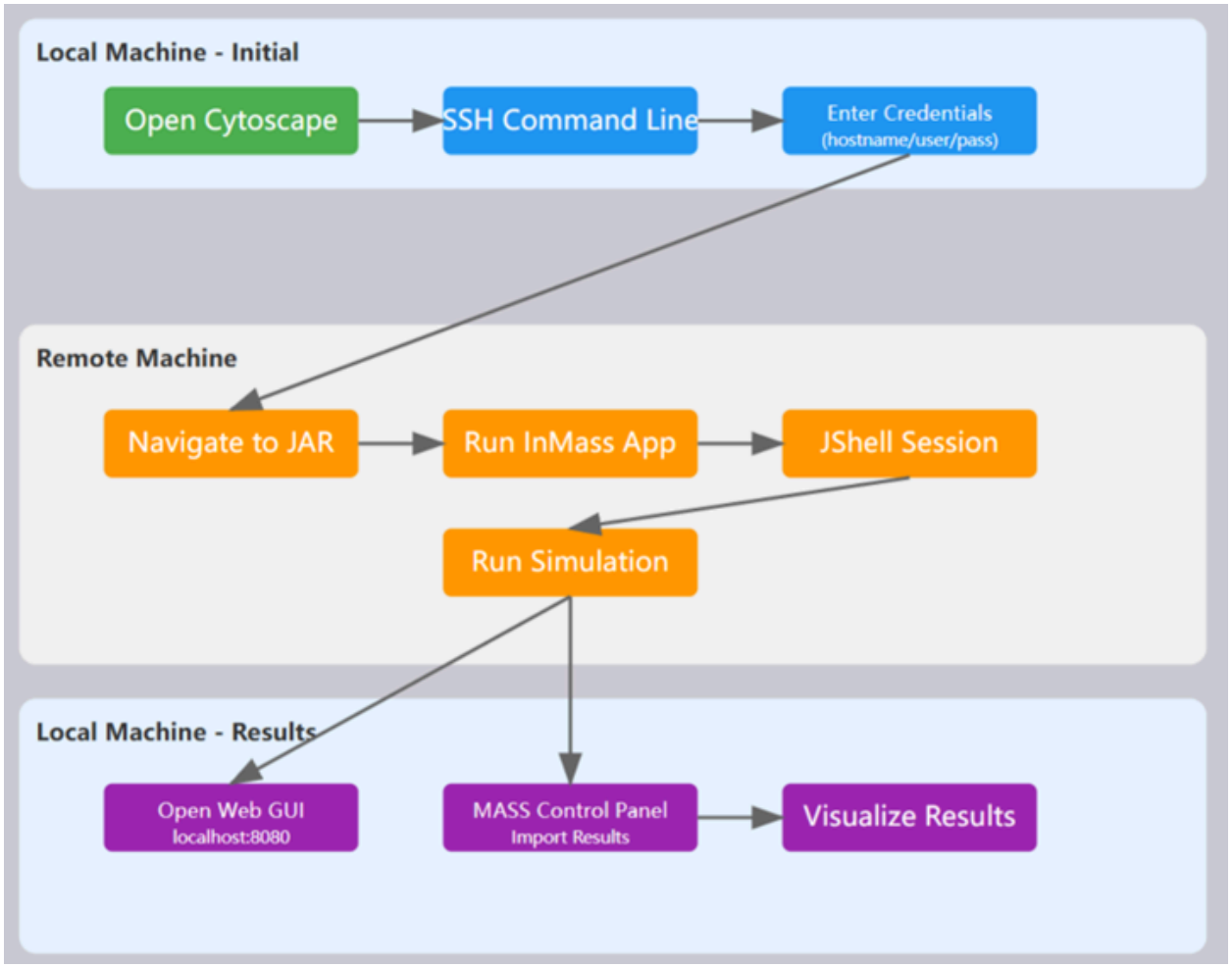
[15] "The NetLogo Predator-Prey simulation (Wolf-Sheep-Grass)," ResearchGate. [Online]. Available: https://www.researchgate.net/figure/The-NetLogo-Predator-Prey-simulation-Wolf-Sheep-Grass_fig1_222712675

[16] S. Luke, C. Cioffi, L. Panait, K. Sullivan, and G. Balan, "MASON: A Multiagent Simulation Environment," Simulation, vol. 81, pp. 517-527, 2005.

[17] "Distributed MASON (D.MASON) interface showing worker configuration," GIS Agents Blog, Mar. 2012. [Online]. Available: https://www.gisagents.org/2012/03/distributed-mason.html

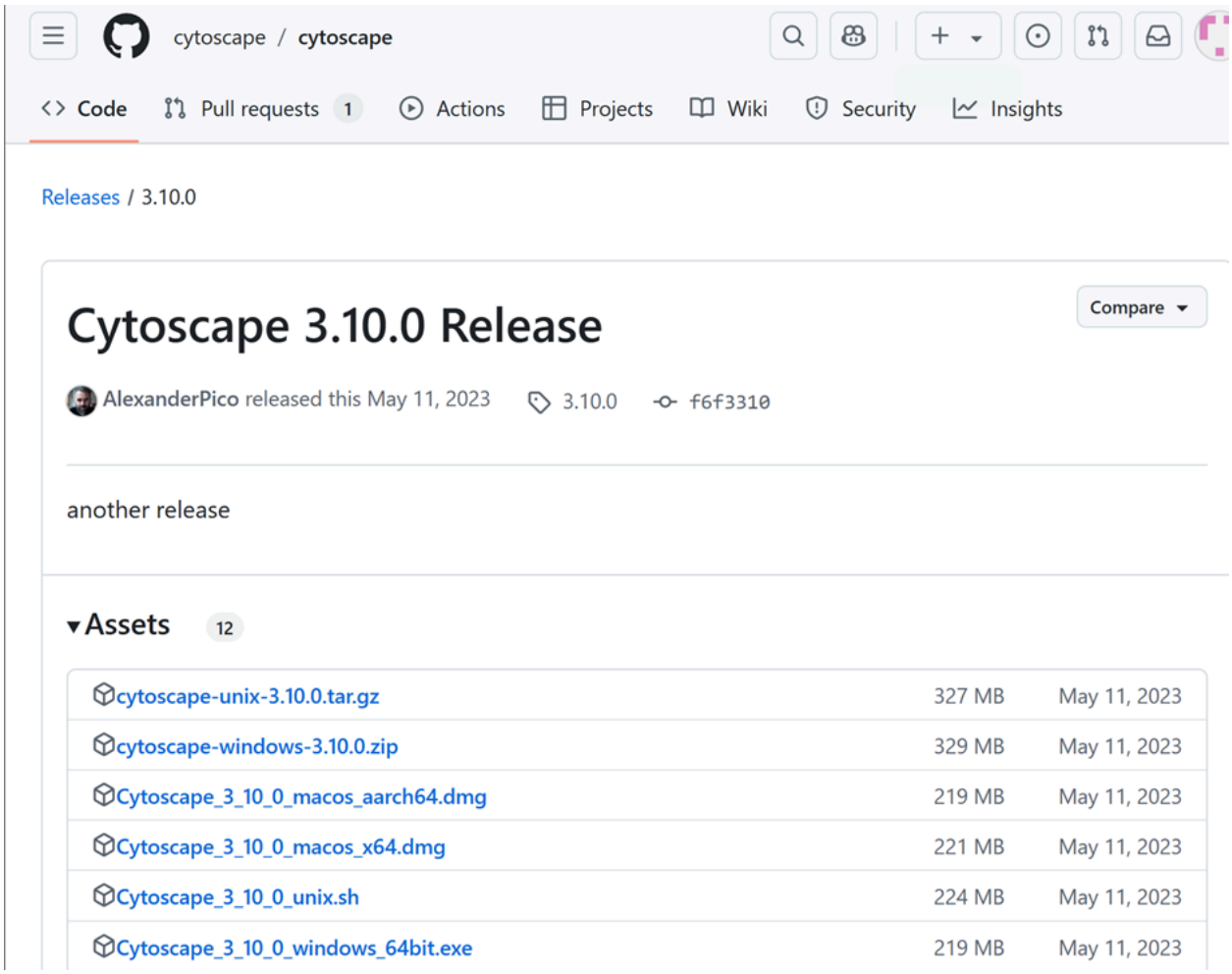
Appendix A: Setting Up and Running InMASS GUI Applications

Workflow

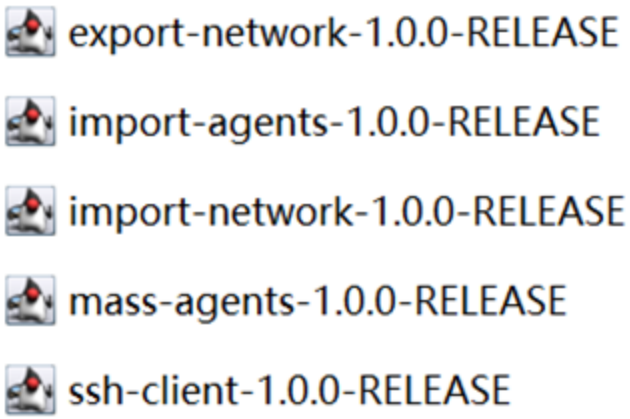


Setup Instructions

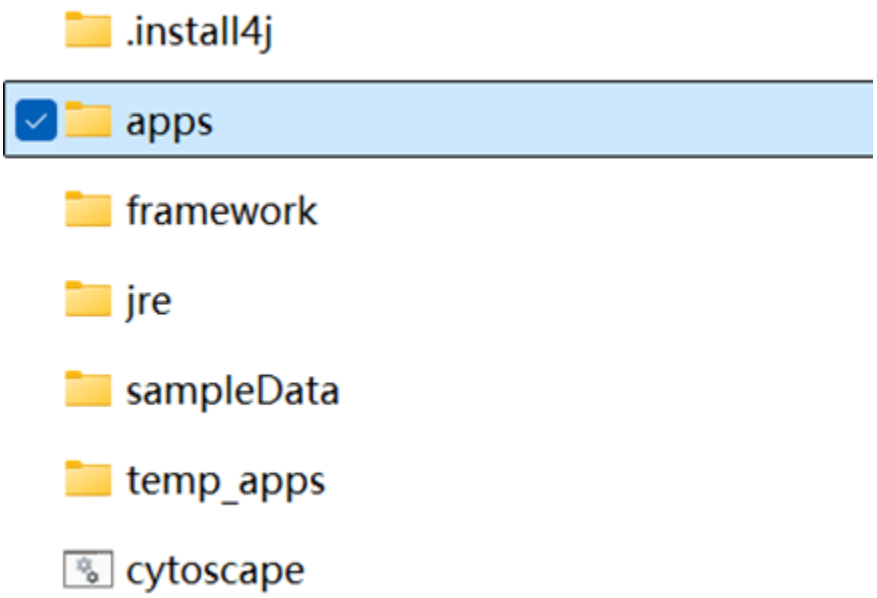
1. Download and install Cytoscape 3.10.0:
    - Get it from: https://github.com/cytoscape/cytoscape/releases/3.10.0/



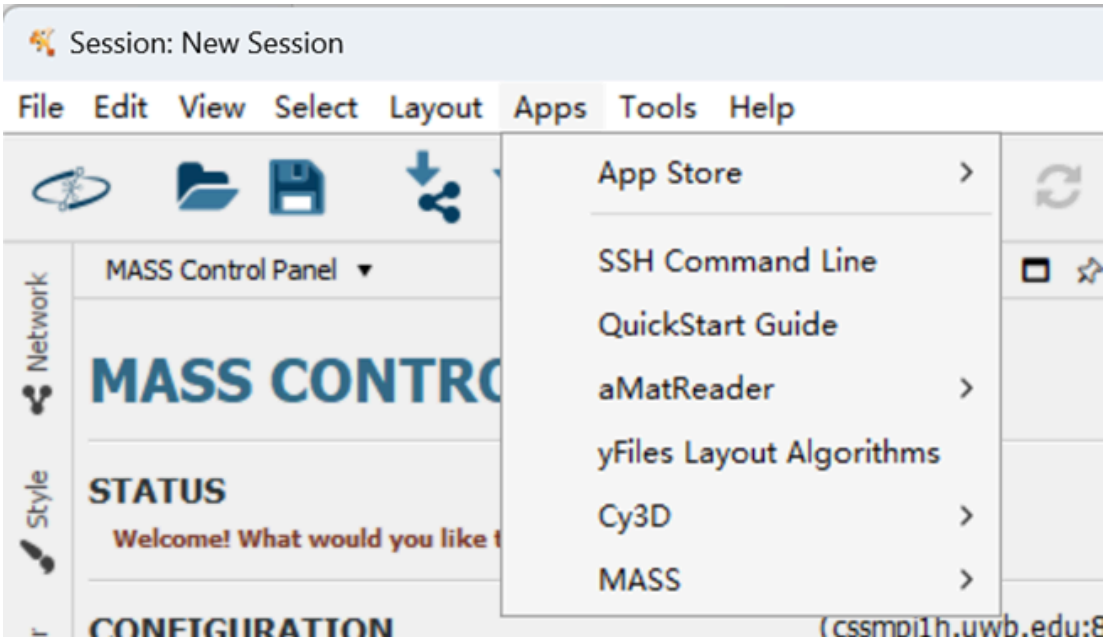
2. Install plugins from provided folder to Cytoscape:
    - Find the cytoscape_plugin folder in Luo_leng_mASS_GUI_Demo directory



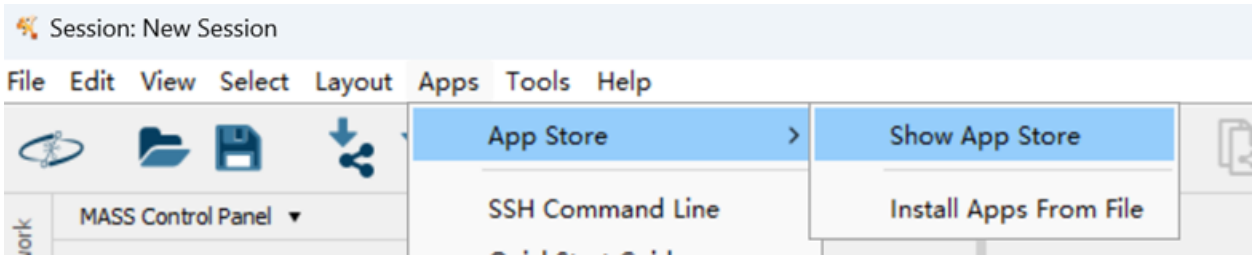
- Copy all .jar files from this folder to your Cytoscape installation's apps folder

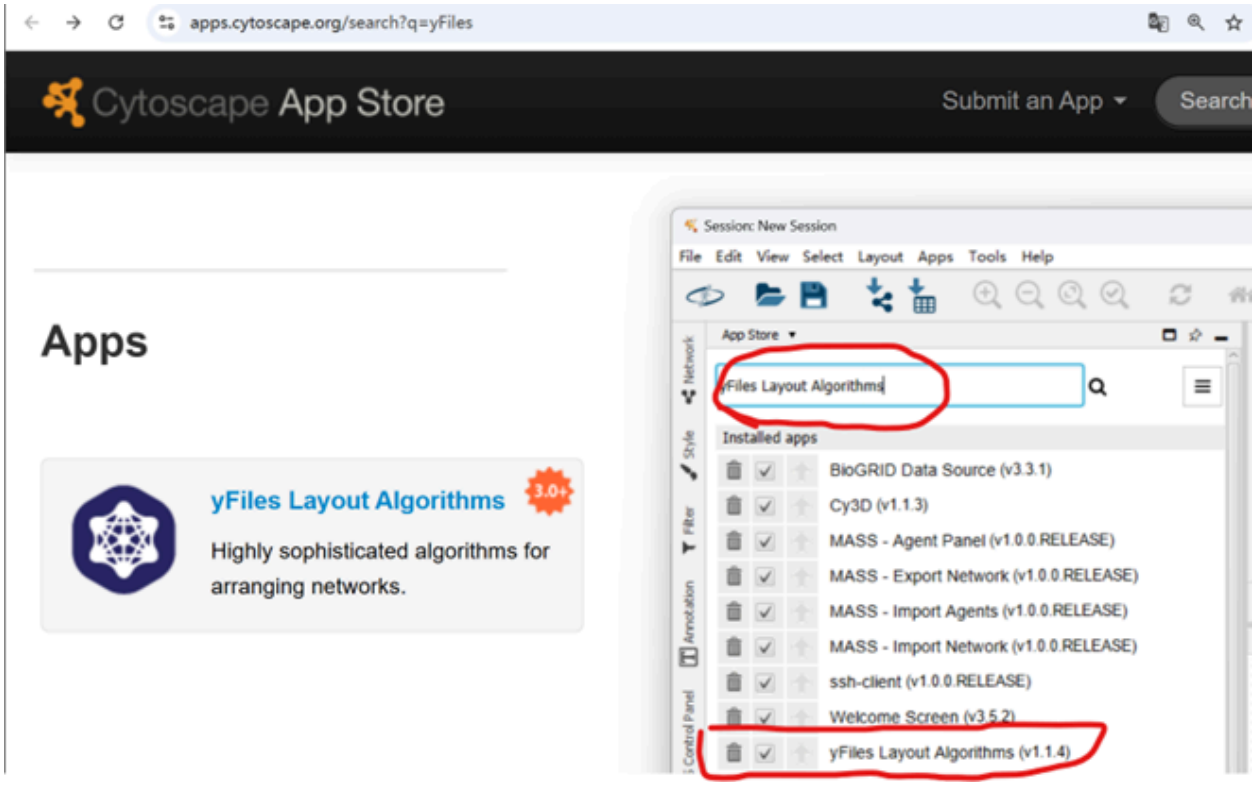


- Launch Cytoscape to verify the plugins are properly recognized, you should see SSH Command Lin, MASS under Apps Menu



- Open Show App Store and download yFiles plugin



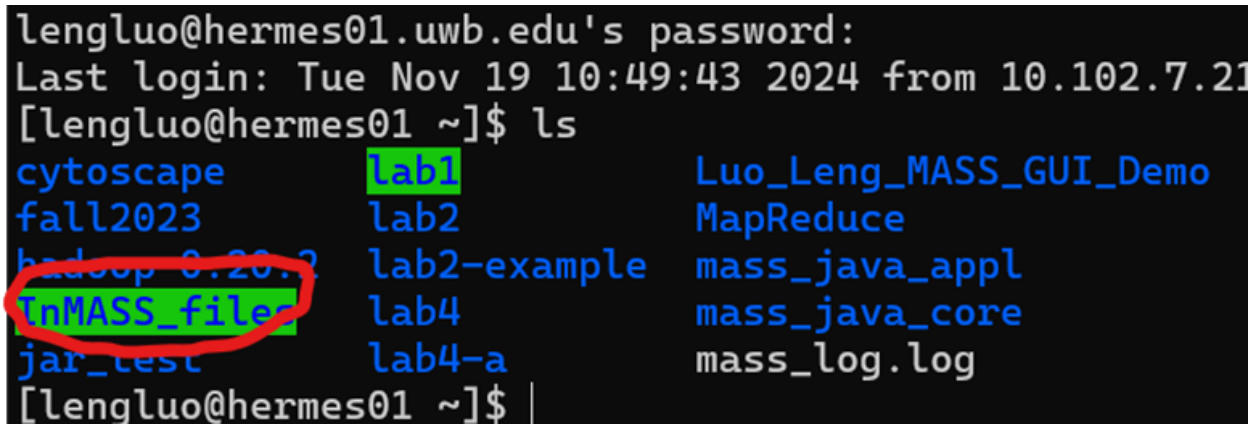


3. Copy files to remote machine:
    - Use SCP to copy the InMASS_files folder:

      Example scp -r InMASS_files username@hermes01.uwb.edu:~

```
PS E:\11_04_MASS\Luo_Leng_MASS_GUI_Demo> cd .\InMASS_files\
PS E:\11_04_MASS\Luo_Leng_MASS_GUI_Demo\InMASS_files> scp -r InMASS_files username@hermes01.uwb.edu:~
```

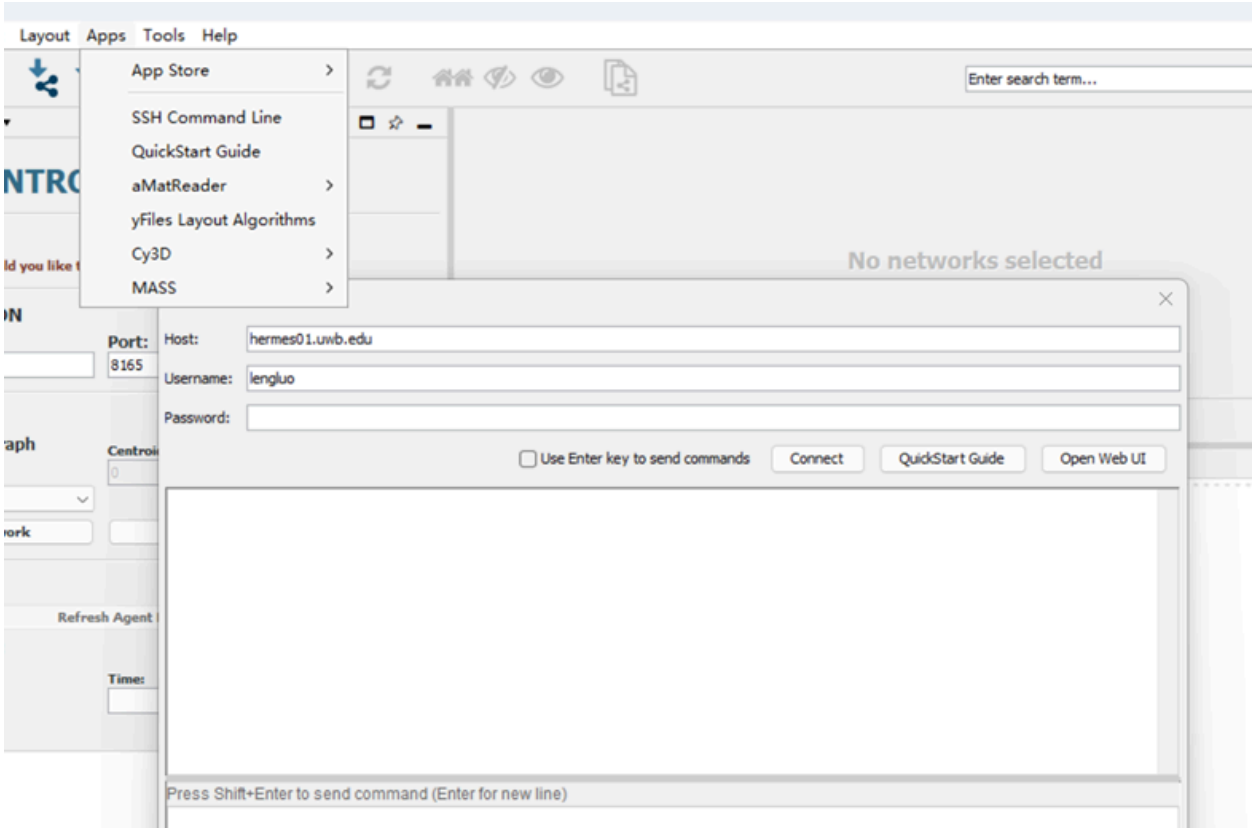
    - Connect to your remote machine



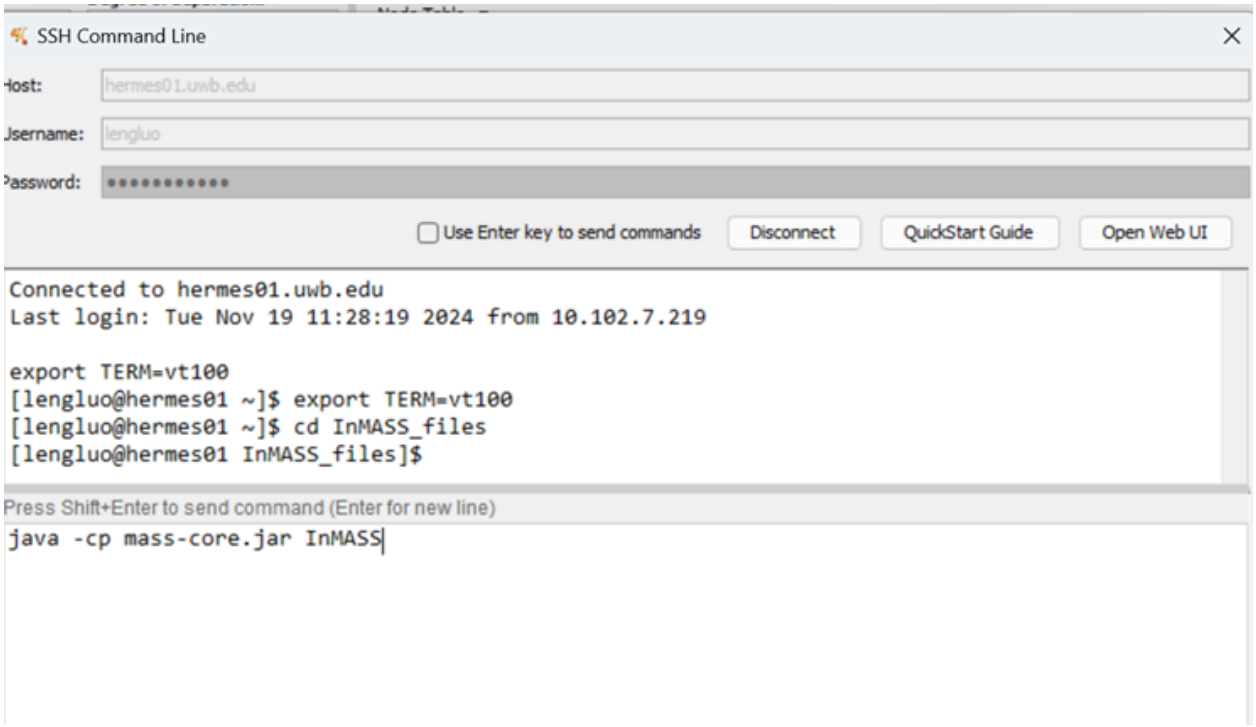
- Navigate to InMASS_files and modify nodes.xml as needed

Running the Application

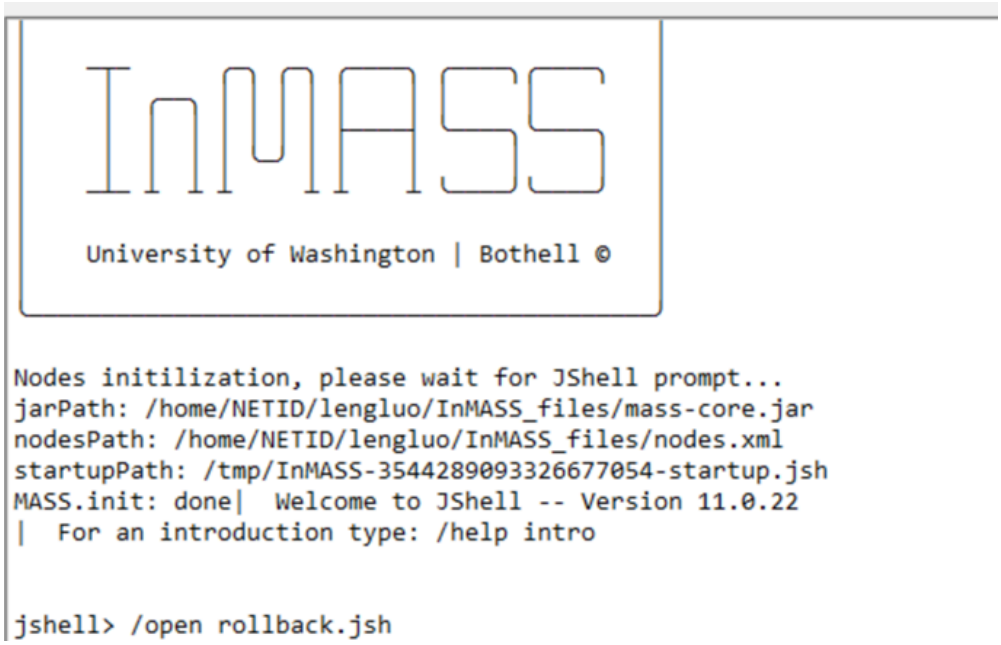
1. In Cytoscape:
   - Go to Apps menu
   - Click SSH Command Line
   - Enter your Host, Username, and Password
   - Click Connect



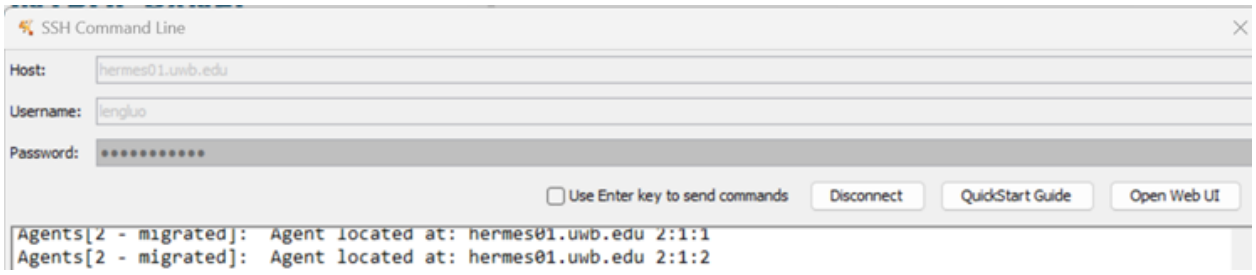
2. Start InMASS:
    - Enter: java -cp mass-core.jar InMASS
    - Wait for the jshell prompt

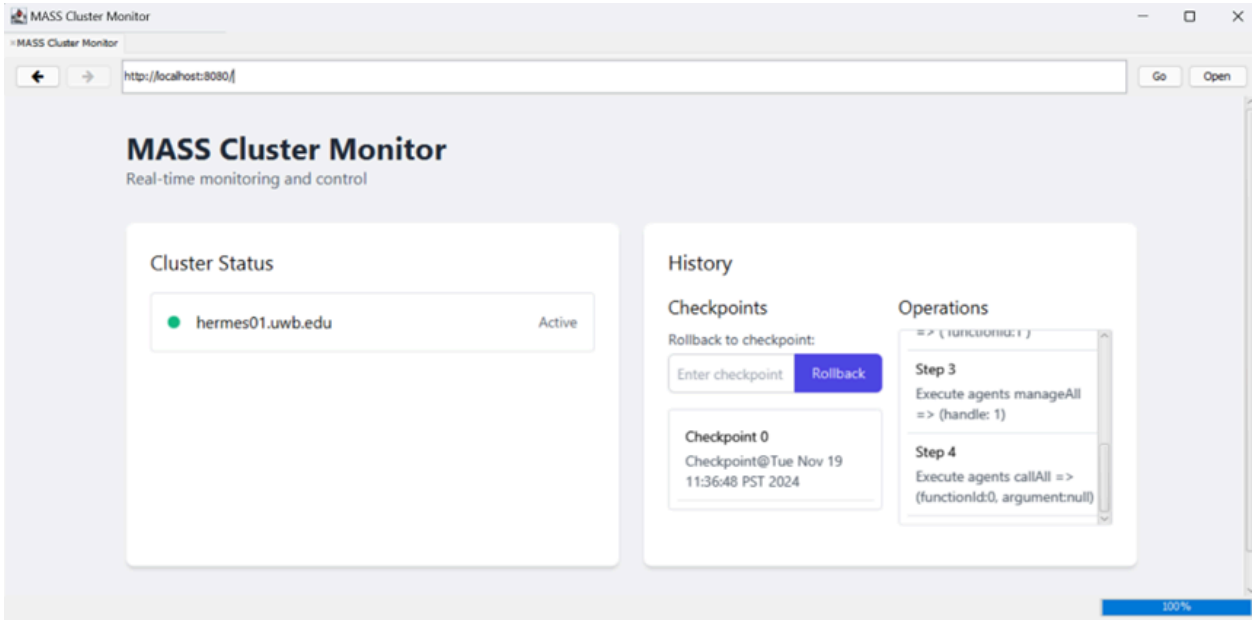


- Run: /open rollback.jsh

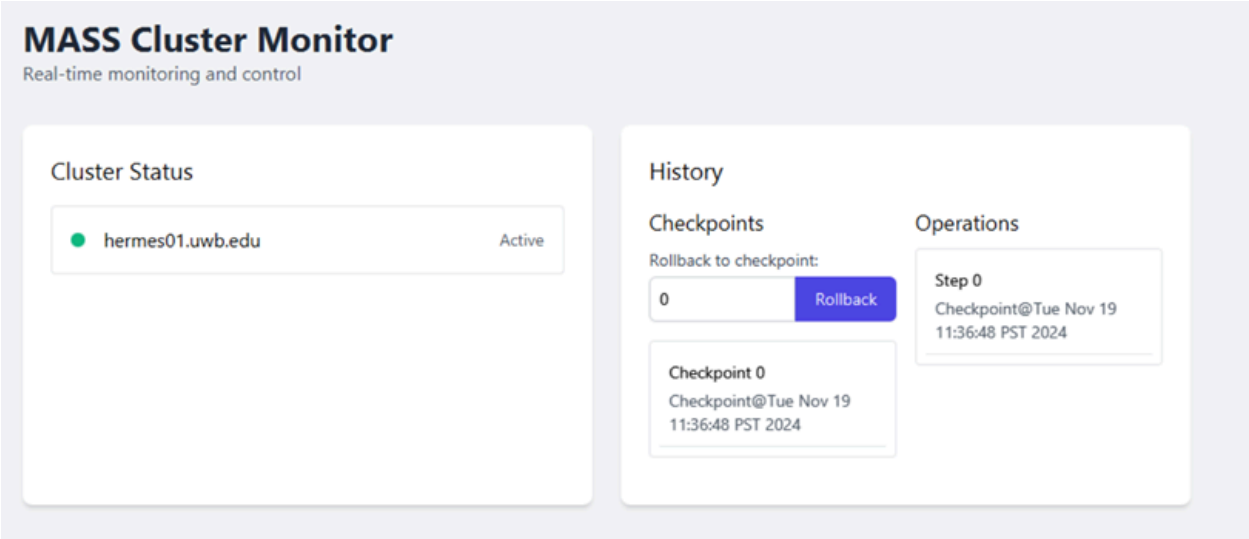


- Click "Open Web GUI" then access localhost:8080

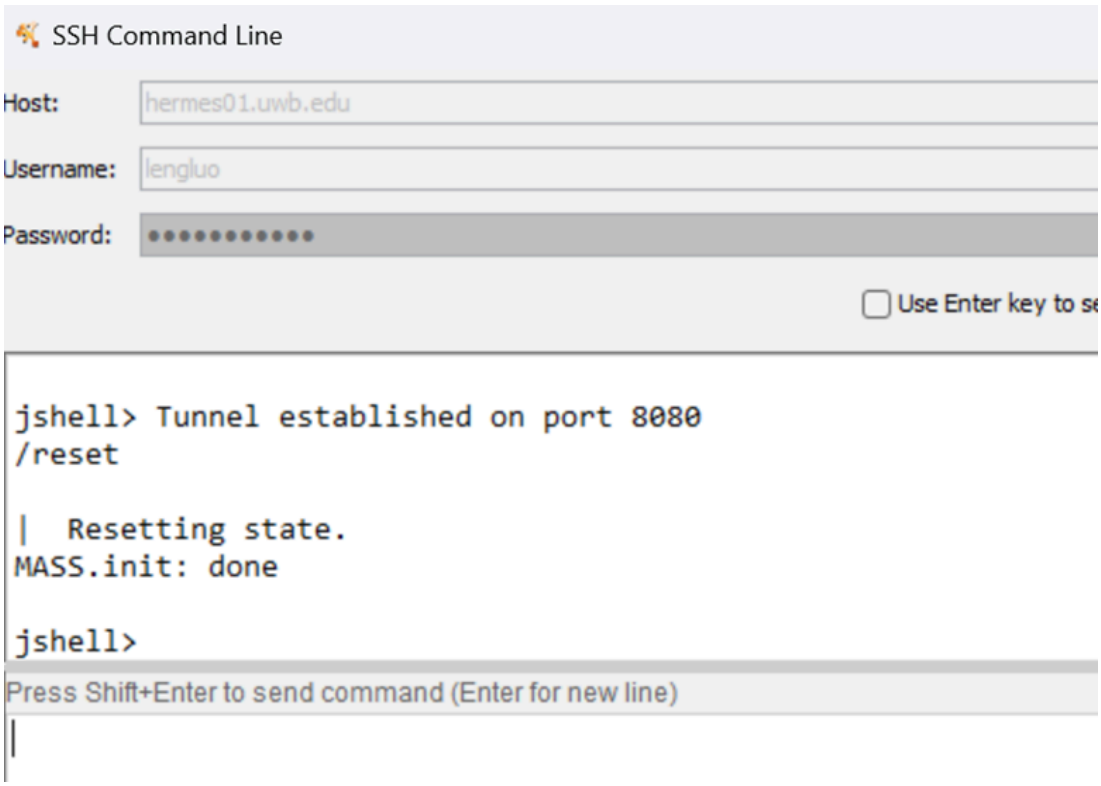




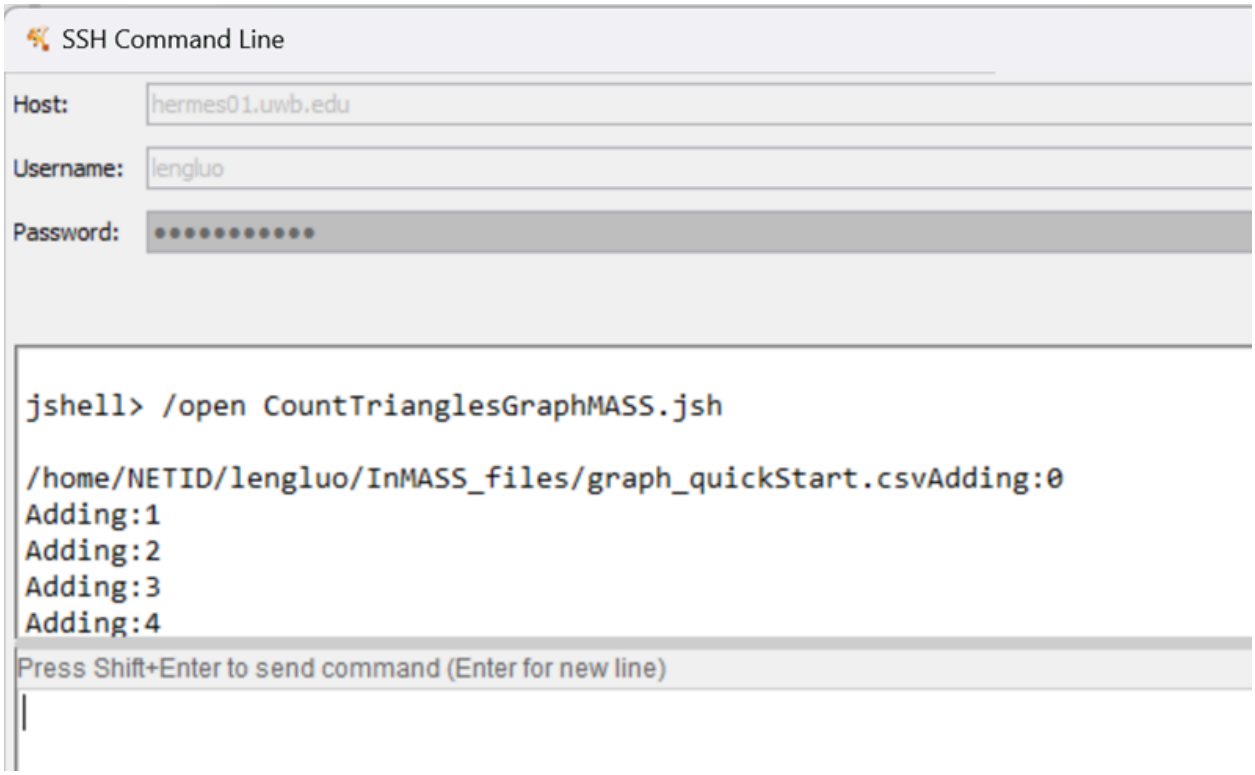
3. Managing simulations:
    - Enter 0 to roll back to the checkpoint



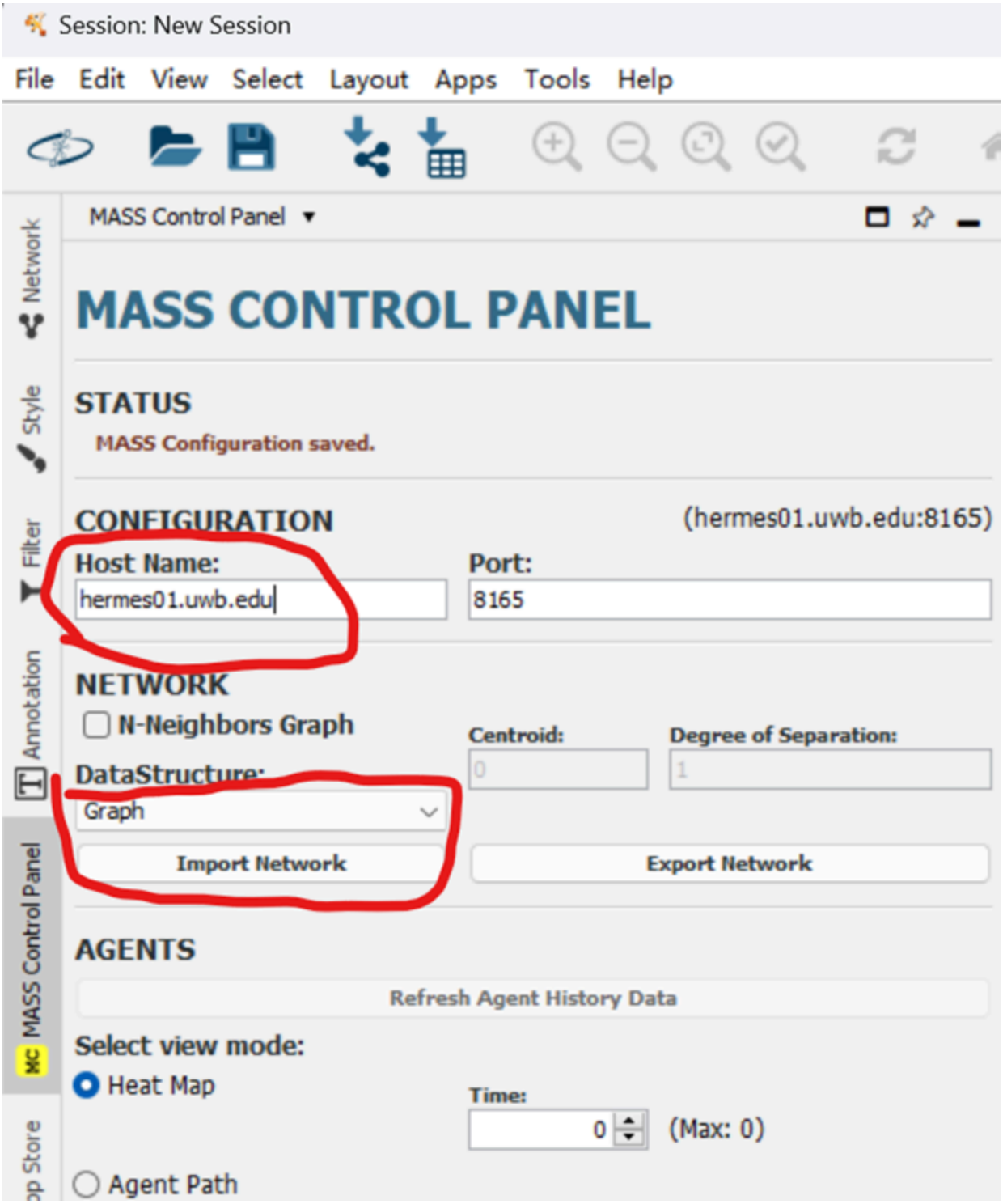
    - Use /reset in Jshell to prepare for a new simulation



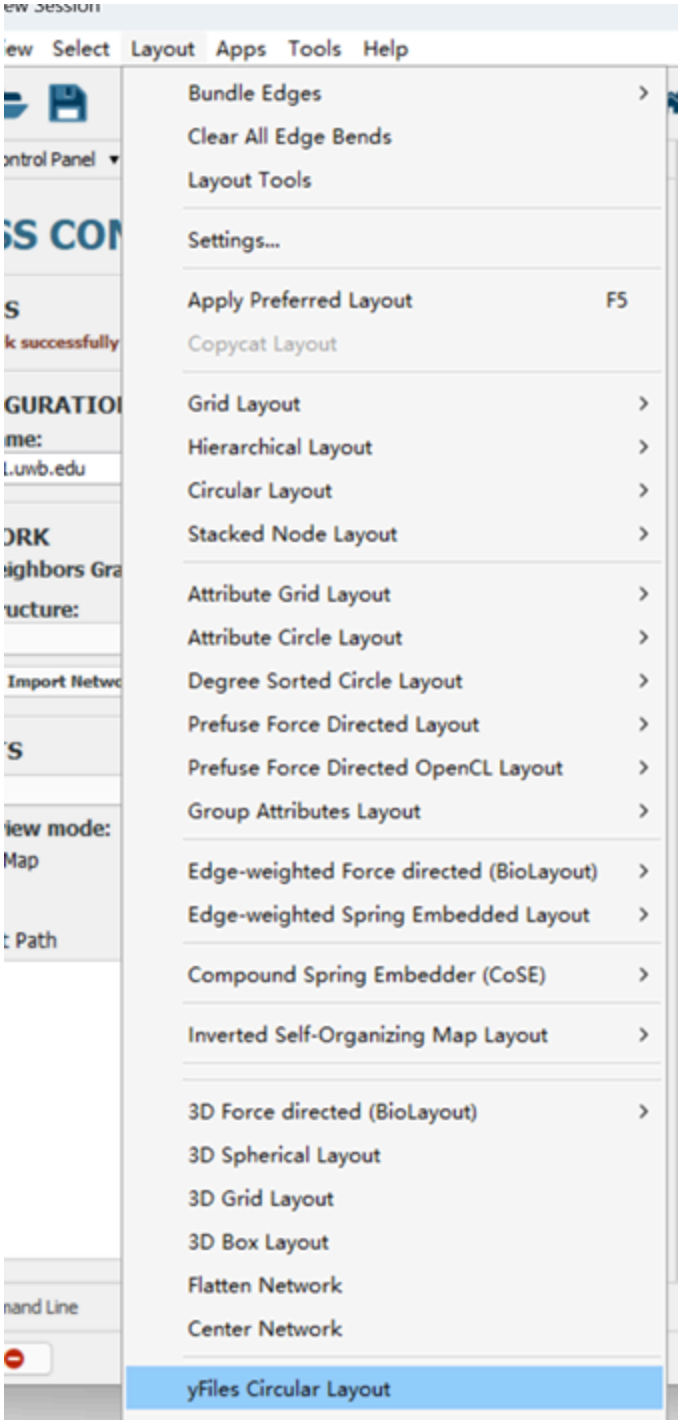
Run /open CountTrianglesGraphMASS.jsh to start simulation



4. Visualizing results:
    - In Cytoscape, change host name, then select Graph and import Network



- go to Layout Menu then select yFiles Circular layout. Or simply hold the nodes to arrange their location


ew Session
iew Select Layout Apps Tools Help
Bundle Edges
Clear All Edge Bends
Layout Tools
Settings...
Apply Preferred Layout F5
Copycat Layout
Grid Layout
Hierarchical Layout
Circular Layout
Stacked Node Layout
Attribute Grid Layout
Attribute Circle Layout
Degree Sorted Circle Layout
Prefuse Force Directed Layout
Prefuse Force Directed OpenCL Layout
Group Attributes Layout
Edge-weighted Force directed (BioLayout)
Edge-weighted Spring Embedded Layout
Compound Spring Embedder (CoSE)
Inverted Self-Organizing Map Layout
3D Force directed (BioLayout)
3D Spherical Layout
3D Grid Layout
3D Box Layout
Flatten Network
Center Network
yFiles Circular Layout

Appendix B: InMASS User Experience Survey Result

System Information and Installation Experience

ID	Operating System	Installation Difficulty	Issues
1	macOS	Moderate	Port 8080 conflict Daunting setup process for naive end-user
2	Windows	Easy	Port 8080 conflict
3	Windows	Easy	Infinite loading in "Cytoscape Task" window
4	macOS	Easy	Cytoscape crashed while attempting to use the yFile circular layout for visualization
5	macOS	Easy	Not specified
6	macOS	Moderate	Had difficulty locating Cytoscape Apps folder Web GUI button has usability issue when clicking the button twice
7	Windows	Easy	Blank Cytoscape Task Window when importing network in MASS Control Panel
8	Windows	Moderate	Not part of DSlab, need to borrow user account to access remote machines
9	macOS	Easy	Not specified
10	Windows	Moderate	Rollback does not work when multiple checkpoints occur

Performance Comparison with Other ABM Simulators

ID	Startup Time	Ease of Use	Prior ABM Experience
1	Haven’t used Other simulators	Haven’t used Other simulators	No
2	Haven’t used Other simulators	Haven’t used Other simulators	No
3	Haven’t used Other	Haven’t used Other	No

	simulators	simulators	
4	Haven’t used Other simulators	Haven’t used Other simulators	No
5	Haven’t used Other simulators	Haven’t used Other simulators	No
6	Haven’t used Other simulators	Haven’t used Other simulators	No
7	Slightly slower	Slightly harder	Yes
8	Slightly faster	Slightly easier	Yes
9	Haven’t used Other simulators	Haven’t used Other simulators	No
10	Much faster	Slightly easier	Yes

Desired Simulation Scenarios

Simulation Type	**Number of Respondents Interested**
Path Finding	**5**
Network Community Detection	**3**
Collision Detection	**3**
Flocking/Swarming Behavior	**4**
Load Balancing Scenarios	**2**
Spatial Cluster	**2**
KNN	**1**
Custom: graph Database queries	**1**

UI Preferences and Required Features

ID	**Preferred Interface**	**Key Reason for Preference**
1	Complete web-based interface	A more unified interface is needed

2	Complete web-based interface	Current design contains too much context switching
3	Complete web-based interface	Everything in one place
4	No preference	Not specified
5	No preference	Not specified
6	Current hybrid design	More comfortable with traditional desktop application layout
7	Current hybrid design	Heavy data processing concern with complete web-based interface More comprehensive features within Cytoscape
8	Complete web-based interface	More user-friendly user interface with pred-defined models
9	Complete web-based interface	Access with no installation required and provides simpler navigation
10	Complete web-based interface	Lesser installation & has ease of use

Most Requested Web Interface Features

- Built-in data analysis tools (6 respondents)
- Export/import capabilities (6 respondents)
- Integrated documentation (4 respondents)
- Real-time collaboration (4 respondents)
- Custom layout design (3 respondents)

Key Valuable Aspects Mentioned

1. Visualization capabilities:
    - Distributed graph algorithms
    - Agent movements and interactions
    - Graph structure visualization
    - Real-time simulation monitoring
2. Development features:
    - Code modification without recompiling
    - Checkpoints/rollbacks
    - JShell integration
3. Distributed computing:

- Multi-machine support
- Active node monitoring
- Simplified configuration

Common Improvement Suggestions

1. Documentation and Usability:
    - Better documentation for applications
    - Clearer functionality explanations
    - More intuitive command interface
2. Interface Improvements:
    - Streamlined workflow
    - Simplified Cytoscape UI
    - Consistent window management
3. Technical Enhancements:
    - Better support for non-lab members
    - Improved multi-machine initialization
    - More pre-built graph applications

Appendix C: Survey Template

The following survey template was used to evaluate the usability, performance, and integration of the InMASS system. Participants were asked to complete specific tasks using the system and then provide feedback based on their experiences.

InMASS User Experience Survey

System Information

1. What operating system are you using?
    - Windows
    - macOS
    - Linux
    - Other (please specify): _____

Installation & Setup Experience

2. How would you rate the difficulty of following the installation instructions?
    - Very Easy
    - Easy
    - Moderate
    - Difficult
    - Very Difficult

2a. If you experienced any difficulties, please describe them:

Performance & Comparison

3. Compared to other Agent-Based Modeling (ABM) simulators you've used, how would you rate InMASS in terms of:

a) Startup Time:

- Much faster
- Slightly faster

- About the same
- Slightly slower
- Much slower
- Haven't used other ABM simulators

b) Ease of Use:

- Much easier
- Slightly easier
- About the same
- Slightly harder
- Much harder
- Haven't used other ABM simulators

Simulation Interests & Requirements

4. Which simulation scenarios would you be interested in seeing implemented in InMASS? (Select all that apply)
    - K-Nearest Neighbors
    - Collision Detection
    - Path Finding
    - Network Community Detection
    - Spatial Clustering
    - Range Queries
    - Load Balancing Scenarios
    - Flocking/Swarming Behavior
    - Other (please specify):
5. What types of agent/place visualizations would be most useful for your work? (Open-ended response)

User Interface Preferences

6. Which user interface design would you prefer?
    - Complete web-based interface
    - Current hybrid design (Cytoscape + web services)
    - No preference

6a. Please explain your choice:

7. If you selected a web-based interface, which features would be most important? (Select all that apply)
    - Integrated documentation
    - Real-time collaboration
    - Custom layout design
    - Built-in data analysis tools
    - Export/import capabilities
    - Other (please specify): _____

General Feedback

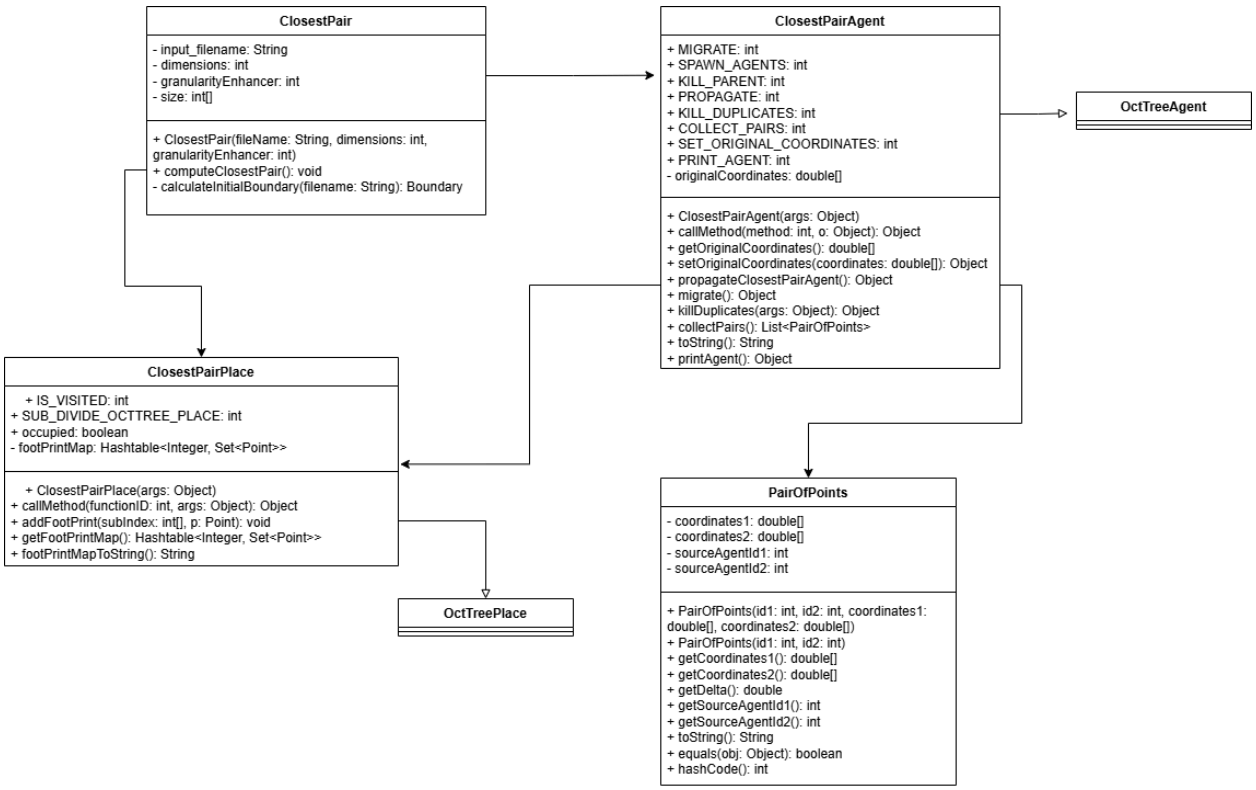
8. What aspects of InMASS do you find most valuable?

9. What aspects of InMASS could be improved?

10. Do you have any additional suggestions or comments?

Appendix D: Class diagram for Closest Pair Points Implementation

The diagram shows the relationships between the main classes: ClosestPair, ClosestPairAgent, ClosestPairPlace, PairOfPoints, and their interactions with OctTreeAgent and OctTreePlace.



Appendix E: System Initialization Sequences

The diagram shows the interactions between User, InMASS, JShell, MASS Framework, Vert.x Server, and Cytoscape Plugin components during system startup and runtime operation. This sequence includes the configuration loading process, MASS framework initialization, server deployment, and the establishment of monitoring channels through SSH tunneling.