A COLLABORATIVE AGENT-BOX SYSTEM

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Kunal Kishore Singh

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department/Program:
Software Engineering

March 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

A Collaborative Agent-Box System

By

KUNAL KISHORE SINGH

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Kendall E. Nygard
Chair

Dr. Saeed Salem

Dr. Nikita Barabanov

Approved:

04/05/2013	Dr. Brian M. Slator
Date	Department Chair

ABSTRACT

In this paper I present a multi-agent system that aims at demonstrating software agent's collaborative behavior when a number of intelligent agents are clustered together within one environment. This study draws inspiration from a research publication titled as Agent Collaboration and Social Networks, authored by R. Sean Bowman and Henry Hexmoor, which aims at demonstrating agent collaboration by implementing a game of agents, boxes and holes.

Similar to the strategies discussed in paper and with a few design modifications of my own I have implemented a simulation that starts with agents, boxes and holes placed in a random order on a square grid. The goal of the agents is to capture a box after having some negotiations with other interested agents, carry and drop the identified box into a hole [9]. The agents have to achieve their goal in a minimum possible time by opting for the shortest available route.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Kendall E. Nygard for his continued support through the duration of my paper and for his guidance over the period of my graduate studies at North Dakota State University. My sincere gratitude to the all the members of the supervisory committee; Dr. Saeed Salem and Dr. Nikita Barabanov without whose participation and assistance this study would not have been successful. I honestly appreciate their time and interest in this study.

Last but not least I would like to take this opportunity to express my heartfelt thanks to my family for their blessings and my friends for their constant help and motivation to successfully undertake and complete my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

A multi-agent system is comprised of intelligent agents that collaborate with each other in an environment. These agents are intelligent computer programs that use their knowledge to perform allocated tasks. They act autonomously by controlling their actions to solve specific problems [3, 6].

The aim of this paper is to demonstrate collaboration among multiple agents that try to push boxes into holes on a square grid. The paper illustrates agent's movement and communication models thereby proposing an efficient way that will assist agents to accomplish their objective. The movement model outlines the path traversed by agents to reach a known destination whereas the communication model describes what messages are exchanged, when the message exchange should take place and how frequently the messages must be sent and received.

Java Agent Development Framework (JADE) is used to implement the Agent-Box application. The format for exchanging agent messages is specified by Agent Communication Language (ACL) language that is defined by a standard organization, Foundation for Intelligent Physical Agents (FIPA). All the operations performed by an agent during its life span and that are fundamental to the development of this application are also identified in this paper.

In section 2 of this paper all the development phases leading to construction of the application are described. Section 3 identifies issues found during the testing of the application. Section 4 contains screen shots of different stages of the application and section 5 concludes the paper.

2. DEVELOPMENT PHASES

The design process guiding the development of Agent-Box collaboration application is based on Waterfall Model. The implementation of the project is broken down into four major phases namely:

Phase 1: System Requirements.

Phase 2: System Design.

Phase 3: System Implementation.

Phase 4: System Testing.

2.1. System Requirements:

The start of the application should demonstrate random placement of agents and objects (boxes and holes) on a square grid. The agents should aim to acquire a box in their line of sight by negotiating with other agents interested in the same box. The agents then must be able to carry the identified box towards a hole and finally drop the box into the hole. Once an agent drops a box into a hole it should wander randomly on the grid and look for other unassigned boxes. The end of the application should be marked by filling all the holes on the grid.

The following section will discuss all the functional, non-functional, interface and data requirements needed to build the Agent-Box application. The requirements will be detailed enough to cover all the primary and important functionality of the application.

2.1.1. *Functional Requirements:*

The requirements to capture the intended behavior of the system are outlined in Table 1 below [19]. The following headings will be used to indicate the commitment level of each requirement.

- Commit (C) – Indicates that the given requirement will be implemented.

- Target (T) – Indicates that the given requirement will be aimed for but the implementation will not be guaranteed.

Table 1: Functional requirements of the application.

REQ - ID	Description	C	T
FR#1	Objects shall be placed randomly on the grid at the start of the application.	C	
FR#1.1	Boxes shall be placed randomly on the grid at the start of the application.	C	
FR#1.2	Holes shall be placed randomly on the grid at the start of the application.	C	
FR#1.3	Agents shall be placed randomly on the grid upon clicking the 'Start Agents' button.	C	
FR#1.4	Placement of agents shall not overlap with objects placement on grid.	C	
FR#1.5	Placement of boxes shall not overlap with holes placement on grid.	C	
FR#2	All agents on the grid shall exchange information on location of boxes with other agents.	C	
FR#2.1	All agents on the grid shall exchange information on location of holes with other agents.	C	
FR#2.2	All agents on the grid shall exchange information on their current location with other agents.	C	
FR#3	All agents on the grid shall exchange information by sending notifications to other agents.	C	
FR#3.1	Agent shall send a notification to all other agents after identifying an object.	C	
FR#3.2	Agent shall send a notification to all other agents after picking a box.	C	
FR#3.3	Agent shall send a notification to all other agents after depositing a box into a hole.	C	
FR#3.4	Agent shall send a notification to all other agents on their way towards an object.	C	
FR#4	All agents on the grid shall make use of their line of sight to exchange information.	C	
FR#4.1	All agents on the grid shall be able to identify objects located up to 4 squares from their current location.	C	
FR#4.2	All agents on the grid shall be able to identify objects located in 4 directions from their current location.	C	
FR#4.3	All agents on the grid shall not be able to identify objects located on their diagonals.	C	

(continued)

Table 1: Functional requirements of the application (continued).

REQ - ID	Description	C	T
FR#5	Agent shall settle for an object at the shortest distance from its current location.	C	
FR#5.1	Agent shall send a notification regarding its interest in an object to all other agents.	C	
FR#5.2	Other agents interested in the same object shall only approach that object when they are at a shorter distance.	C	
FR#5.3	Other agents interested in the same object shall send a notification regarding their interest in that object.	C	
FR#5.4	Agent whose name comes first in alphanumeric order shall approach the object of interest if two or more agents are at equal distance from that object.	C	
FR#6	Agents shall remember the location of objects informed by other agents on the grid.	C	
FR#7	Agents shall have knowledge on objects that have been consumed by other agents on the grid.	C	
FR#8	Agent shall pick a box once it has the knowledge of its box of interest.	C	
FR#8.1	Agent shall not be able to pick another box until it deposits the box, which it is carrying, into a hole.	C	
FR#9	Agent shall carry a box towards a hole once it has the knowledge of its hole of interest.	C	
FR#10	Agent shall deposit a box into a hole.	C	
FR#11	Agent shall look for other unassigned boxes once it deposits a box into hole.	C	
FR#12	Agent shall wander randomly on the grid until it finds an unassigned box.	C	
FR#12.1	Agent shall be able to make a left turn on the grid.	C	
FR#12.2	Agent shall be able to make a right turn on the grid.	C	
FR#12.3	Agent shall be able to move in a forward direction on the grid.	C	
FR#12.4	Agent shall be able to move in a backward direction on the grid.	C	
FR#12.5	Unloaded agents shall not be able to move through squares containing holes.	C	
FR#12.6	Loaded agents shall not be able to move through squares containing another box.	C	
FR#12.7	Loaded agents shall be able to cross paths with other agents on the grid.	C	
FR#12.8	Unloaded agents shall be able to cross paths with other agents on the grid.	C	

(continued)

Table 1: Functional requirements of the application (continued).

REQ - ID	Description	C	T
FR#13	Remote Agent Management GUI shall be launched when the user clicks 'R' button on the interface.		T
FR#13.1	User shall be able to suspend an agent using the Remote Agent Management GUI.		T
FR#13.2	User shall be able to resume an agent using the Remote Agent Management GUI.		T
FR#14	Sniffer GUI shall be launched when the user clicks 'S' button on the interface.		T
FR#14.1	User shall be able to view all notifications sent by an agent using the Sniffer GUI.		T
FR#14.2	User shall be able to view all notifications received by an agent using the Sniffer GUI.		T

2.1.2. *Non-functional Requirements:*

Some of the characteristics and important quality attributes like reliability, portability, maintainability and usability that must be taken into consideration while building the application are outlined in the table below (Table 2).

Table 2: Non-functional requirements of the application.

REQ - ID	Description	C	T
NFR#1	The application shall not crash except in the event of an operating system or hardware failure.	C	
NFR#2	The application shall run on any environment supporting JDK 1.5 and above.	C	
NFR#3	The application shall run on any environment with access to a JVM.	C	
NFR#4	The application code shall be modular to permit future modifications.	C	
NFR#5	The application shall be maintained by adequate documentation of code.	C	
NFR#6	The application shall be provided with Build documentation, Source code, Packaged files, Test cases and results.	C	
NFR#7	The application usage shall not require a user to have any kind of special training.	C	

2.1.3. *Interface Requirements:*

The requirements that should be taken into consideration so as to design a user friendly interface are captured in the table below (Table 3).

Table 3: Interface requirements of the application.

REQ - ID	Description	C	T
IR#1	The number of rows in the grid shall be 15.	C	
IR#2	The number of columns in the grid shall be 15.	C	
IR#3	The number of agents on the grid shall be 10.	C	
IR#4	The agents shall be denoted using a lorry.	C	
IR#5	The agents shall be numbered for unique identification.	C	
IR#6	A red circle marked with alphabet 'H' shall depict a hole.	C	
IR#7	A solid black square shall depict a box.	C	
IR#8	A hole shall disappear from the grid once an agent deposits a box in it.	C	
IR#9	Button with text 'Agents' shall be displayed on top right section of the interface.	C	
IR#10	Button with text 'Boxes' shall be displayed on top right section of the interface.	C	
IR#11	Button with text 'Holes' shall be displayed on top right section of the interface.	C	
IR#12	Button with text 'R' shall be shall be displayed on top right section of the interface.		T
IR#13	Button with text 'S' shall be shall be displayed on top right section of the interface.		T
IR#14	A log section containing information on agent's notification to other agents shall be displayed at bottom right section of the interface.		T
IR#14.1	Users shall be able to filter the log depending on their choice of agents.		T
IR#15	Button with text 'Refresh' shall be displayed on the bottom of the log section of the interface.		T
IR#15.1	Users shall be able to update messages in the log by clicking the 'Refresh' button.		T

2.1.4. *Data Requirements:*

The requirements outlining the format and content of an agent's notification message are identified below (Table 4).

Table 4: Data requirements of the application.

REQ - ID	Description	C	T
DR#1	XML format shall be used to represent agent notification messages.	C	
DR#2	XML document shall follow W3C recommendations.	C	
DR#3	XML document containing message content shall be created.	C	
DR#4	All message content shall be contained within meaningful XML tags.	C	
DR#4.1	Meaningful XML tags shall be self-descriptive.	C	
DR#5	Message content shall include an agent's current location.	C	
DR#6	Message content shall include an object type.	C	
DR#7	Message content shall include an object's location.	C	
DR#8	Message content shall include an object's count.	C	

2.1.5. *Requirement Limitations:*

The following constraints shall be considered during design and implementation phases of the application (Table 5).

Table 5: Limitations of the application.

Limitations	Description
Hardware Limitations	None.
Architecture Limitations	None.
Language/Tool Limitations	Development will be done in JAVA. Agent-Box System will be implemented using JADE Agent development framework. Agent Communication Language will adhere to FIPA Standards (**FIPA-ACL**). Development will be done using Eclipse IDE.
Security Limitations	None.

2.2. System Design:

The following section will discuss all the architectural, interface, and component level design needed to build the Agent-Box application. It will provide a detailed view of primary components of the application so as to cover up all the functional, non-functional and interface requirements outlined in the System Requirements section. UML (Unified Modeling Language) notations will be used to model application's architecture and high-level components. Static view of the application will be depicted using a class diagram. Dynamic view will be specified by use case, activity and sequence diagrams [22].

2.2.1. *Use Case Diagrams:*

A user's interactions with the system are illustrated using use case diagrams. The figures below describe high-level system functions and actors (agents in this case) interaction with the system. The use case diagram below identifies all the actions performed by an agent during its life-cycle on the square grid. The agents are involved in a number of activities that range from performing their respective tasks to communicating information to other agents on the grid.

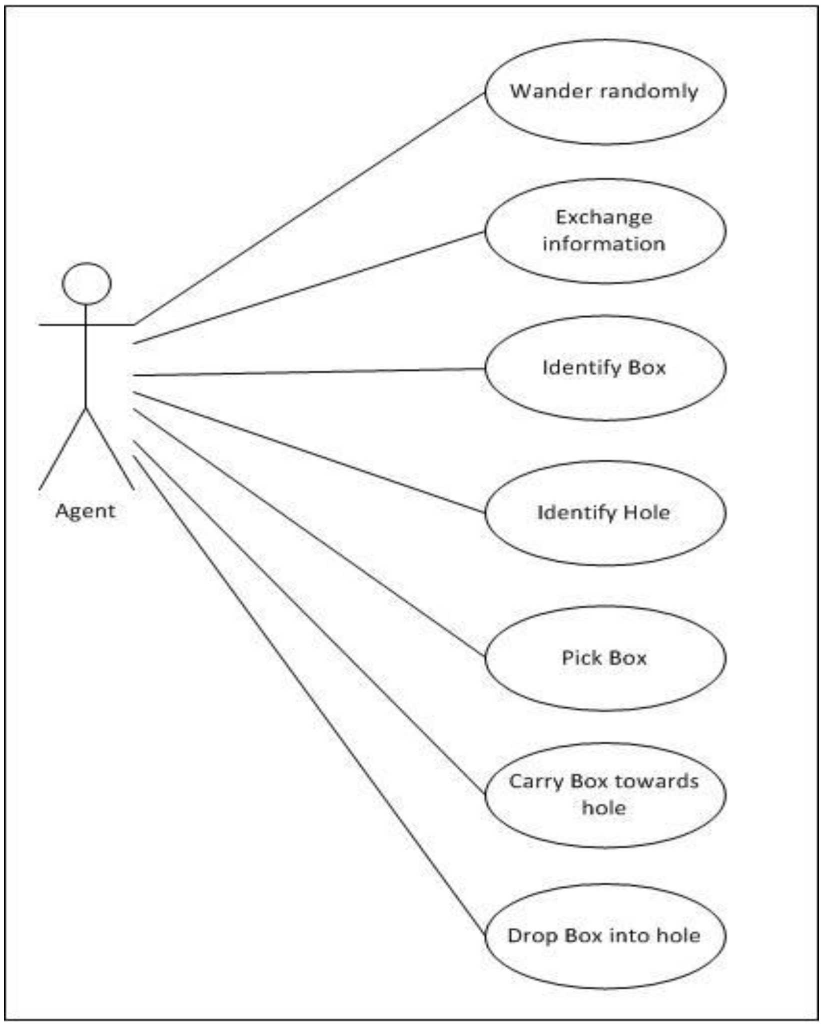
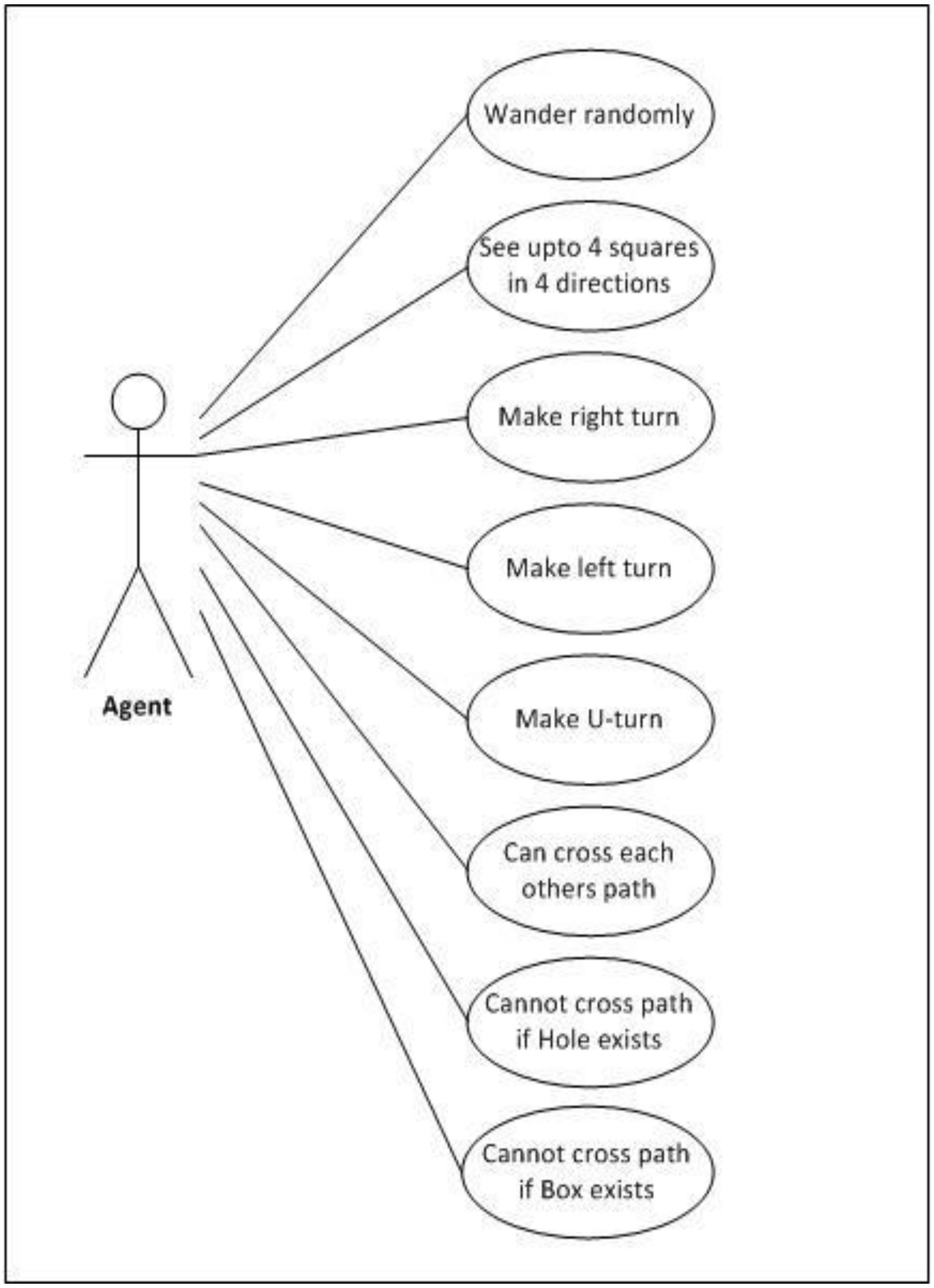


Figure 1: Use case diagram showing agent actions.

The agent's movement on a grid is restricted and they can only make certain maneuvers to reach their object of interest. All agent movements are pointed out in the diagram below.


Wander randomly
See upto 4 squares in 4 directions
Make right turn
Agent
Make left turn
Make U-turn
Can cross each others path
Cannot cross path if Hole exists
Cannot cross path if Box exists
Figure 2: Use case diagram showing an agent's navigation.

All active agents on the grid need to constantly communicate with one another by exchanging meaningful messages. The content of the messages should be informative enough to guide an agent to their object of interest.

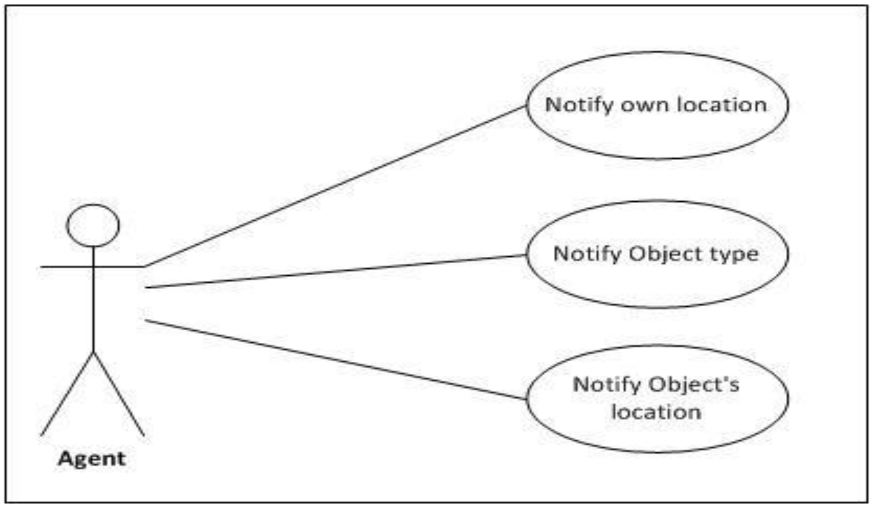


Figure 3: Use case diagram showing an agent's notification.

The agents shall have a memory that will keep track of an objects location and count. The diagram below highlights an agent's awareness.

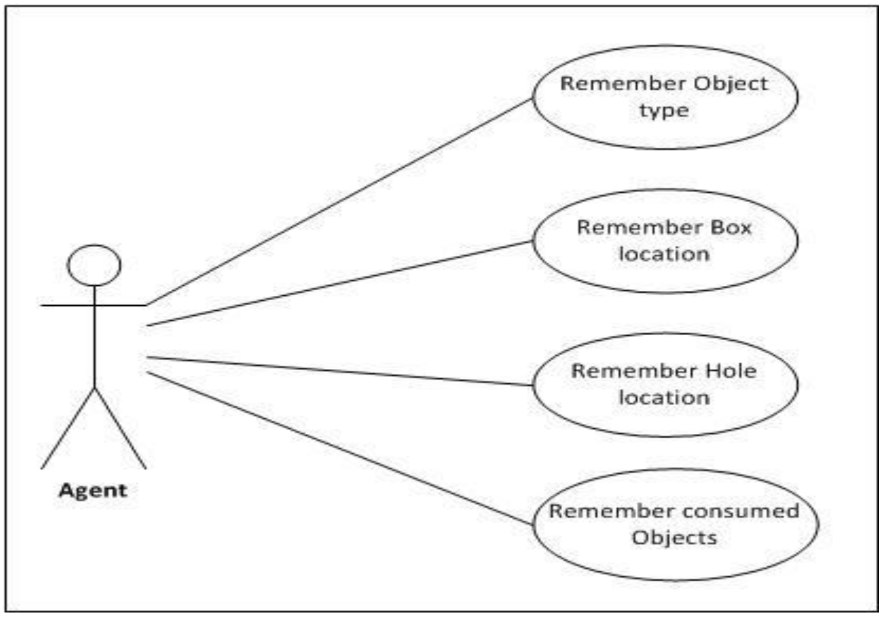


Figure 4: Use case diagram showing an agent's memory.

2.2.2. *Activity Diagrams:*

The diagrams below depict stepwise activities, actions and flow of control of different components of Multi-Agent application. Agents on the grid carry out number activities from the moment they are active and until they die. The activity diagram below identifies all the movements, the order and the approach followed by agents to accomplish their objective.

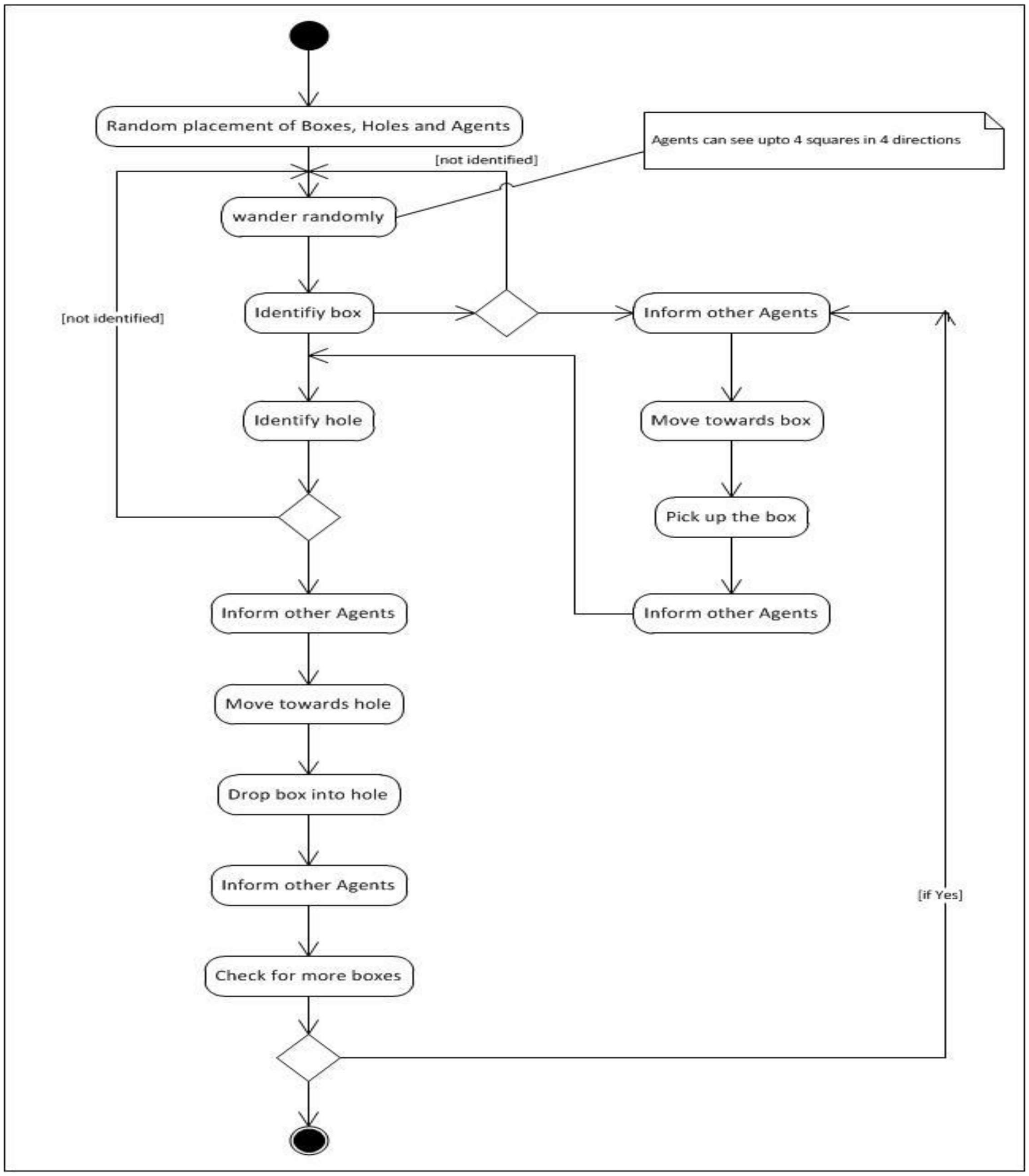


Figure 5: Activity diagram showing agent's movement on the grid.

The agents remain in constant touch with other agents by sending and receiving notification messages. Messages are exchanged at all times during an agent’s activity.

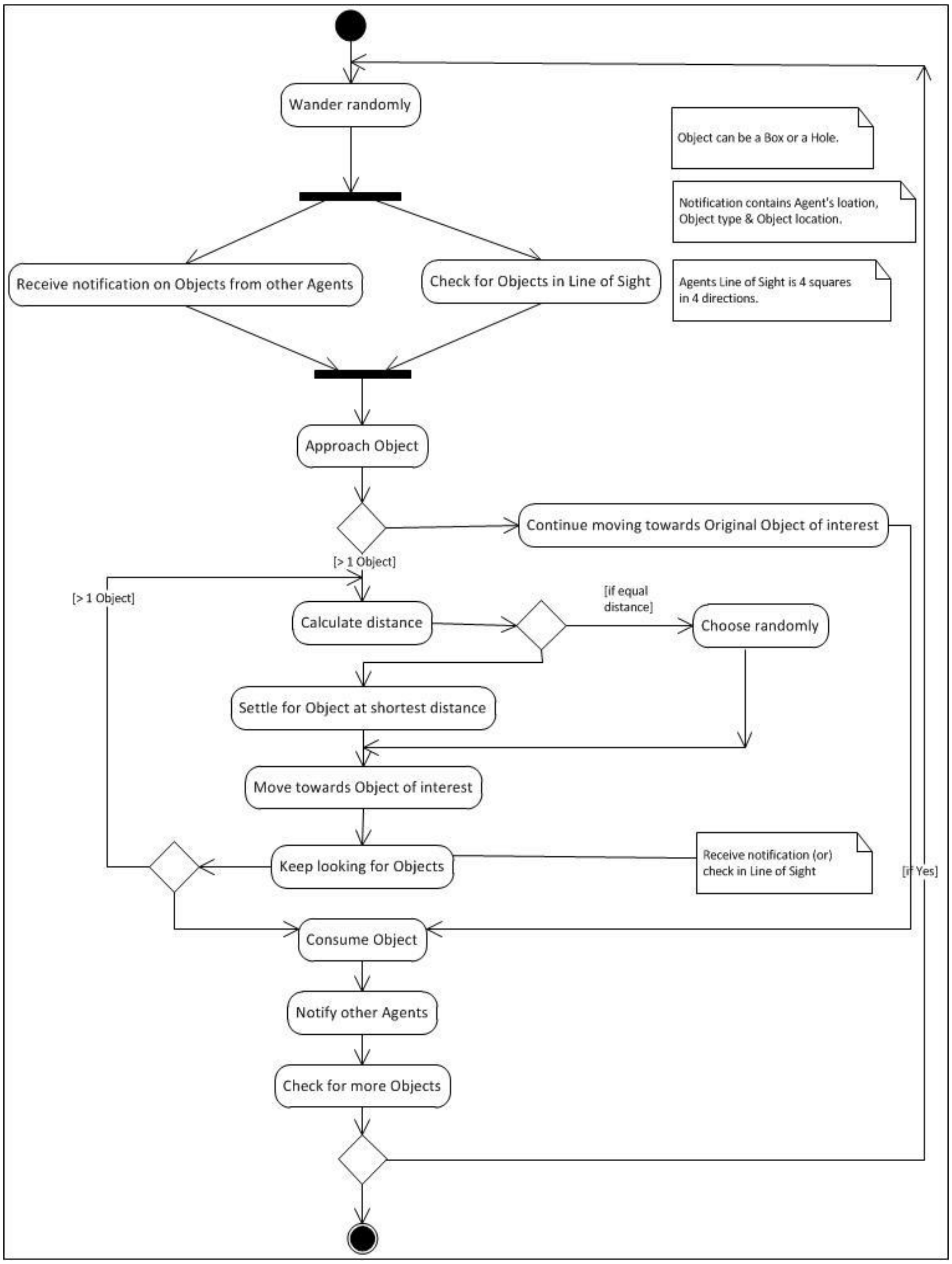

Wander randomly
Object can be a Box or a Hole.
Notification contains Agent's loation, Object type & Object location.
Receive notification on Objects from other Agents
Check for Objects in Line of Sight
Agents Line of Sight is 4 squares in 4 directions.
Approach Object
Continue moving towards Original Object of interest
[> 1 Object]
[> 1 Object]
Calculate distance
[if equal distance]
Choose randomly
Settle for Object at shortest distance
Move towards Object of interest
Keep looking for Objects
Receive notification (or) check in Line of Sight
[if Yes]
Consume Object
Notify other Agents
Check for more Objects

Figure 6: Activity diagram showing agent’s movement and communication.

A number of agents may qualify for an object. In such cases the concerned agents have to negotiate with one another before approaching their object of interest. The diagram below depicts the approach adopted when more than one agent compete for an object.

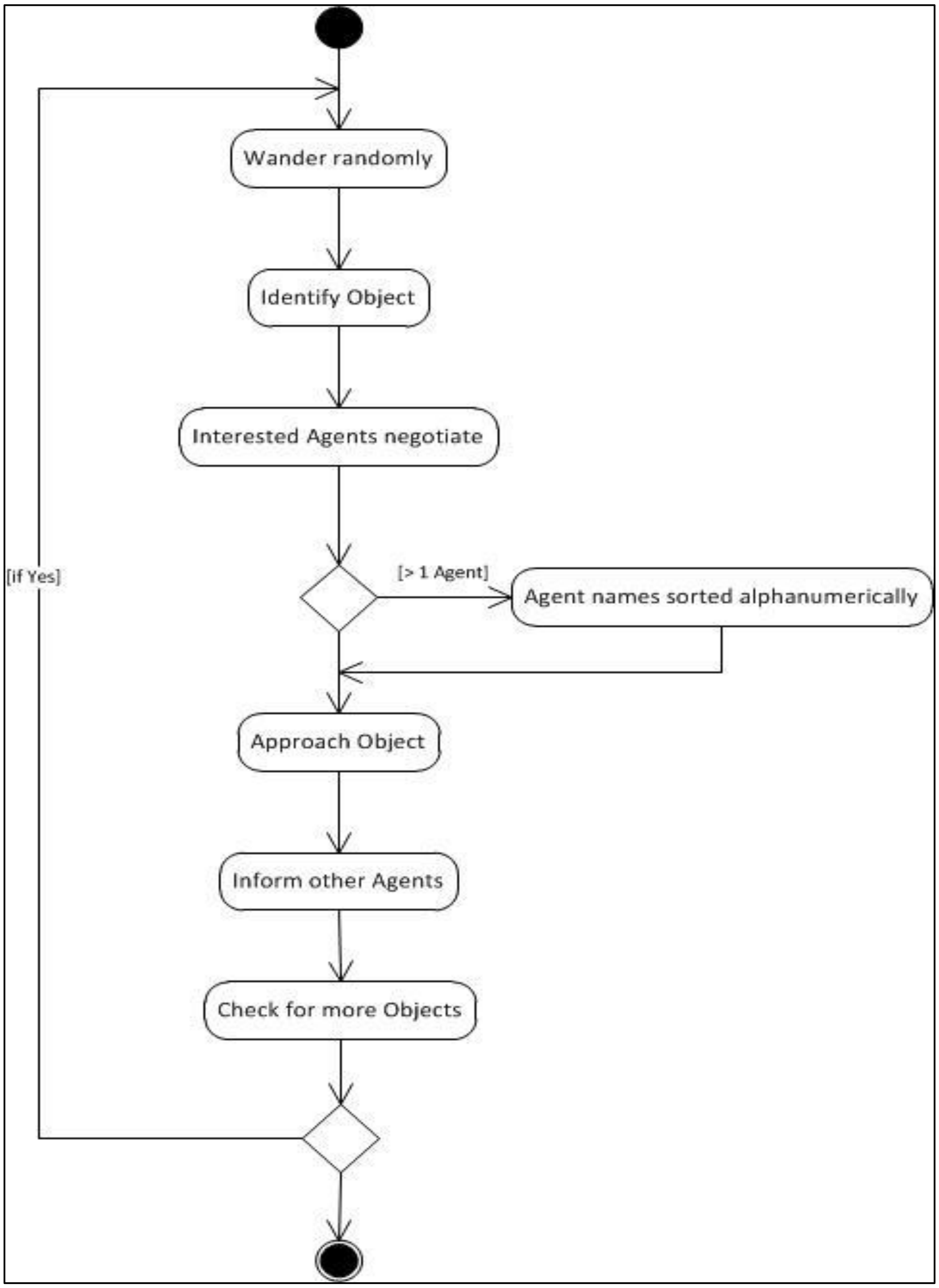


Figure 7: Activity diagram showing an agent’s negotiation.

2.2.3. *__Class Diagram:__*

The diagram below shows different classes, attributes and methods of Agent-Box application. Relationships among classes are depicted using arrows. Details about each Class that comprise of all important methods, method signatures (parameters, data and return types) are provided in corresponding class tables (Table 6 to Table 16) in section 2.2.6.

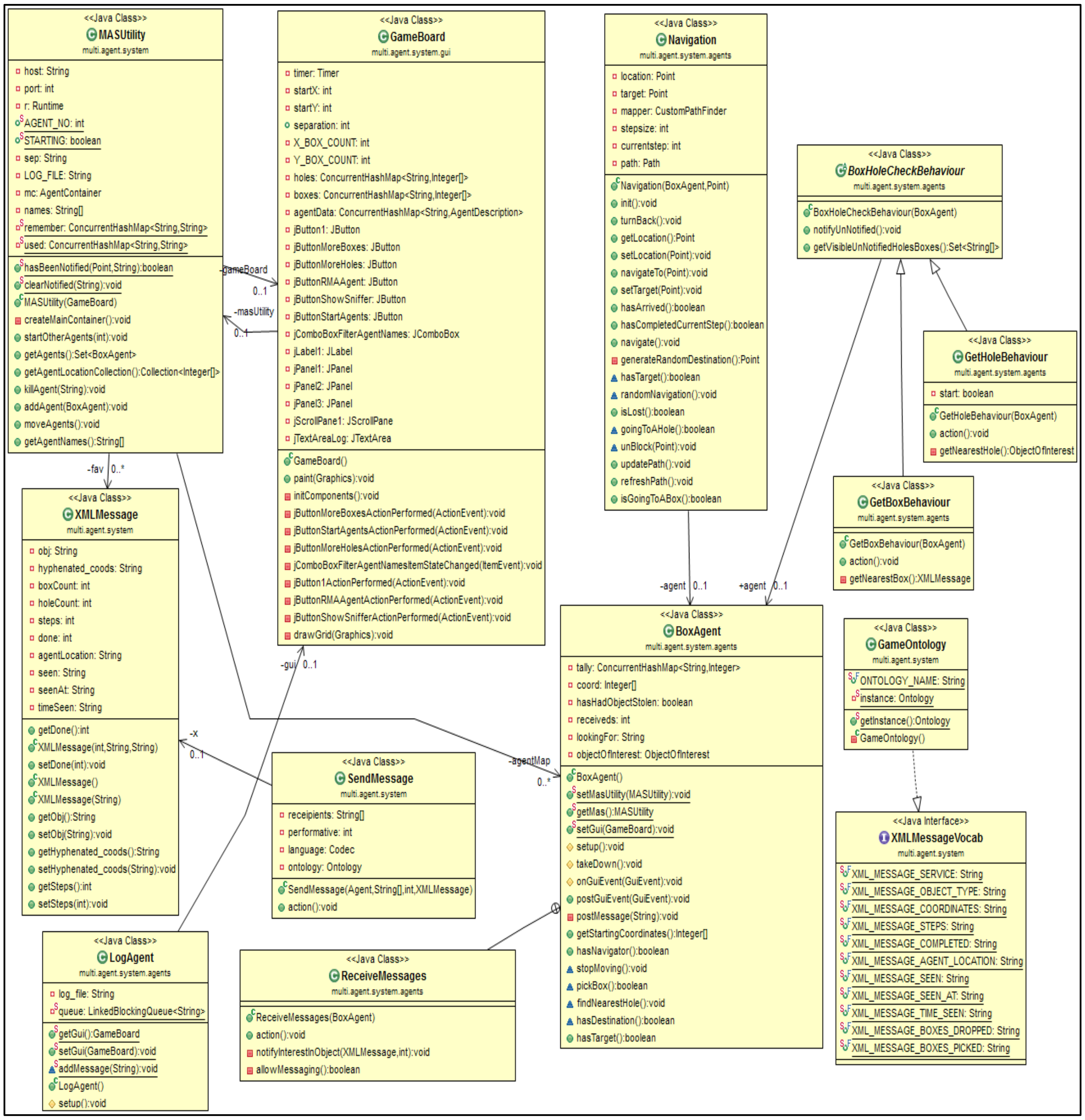


Figure 8: Class diagram showing different classes and their relationships.

2.2.4. *Sequence Diagram:*

Interactions between agents of the application showing sequence of messages exchanged between them are depicted in the event diagram below. Once an agent acquires an object, a notification message containing an object's type, an object's location and the agent's location should be issued to all other agents.

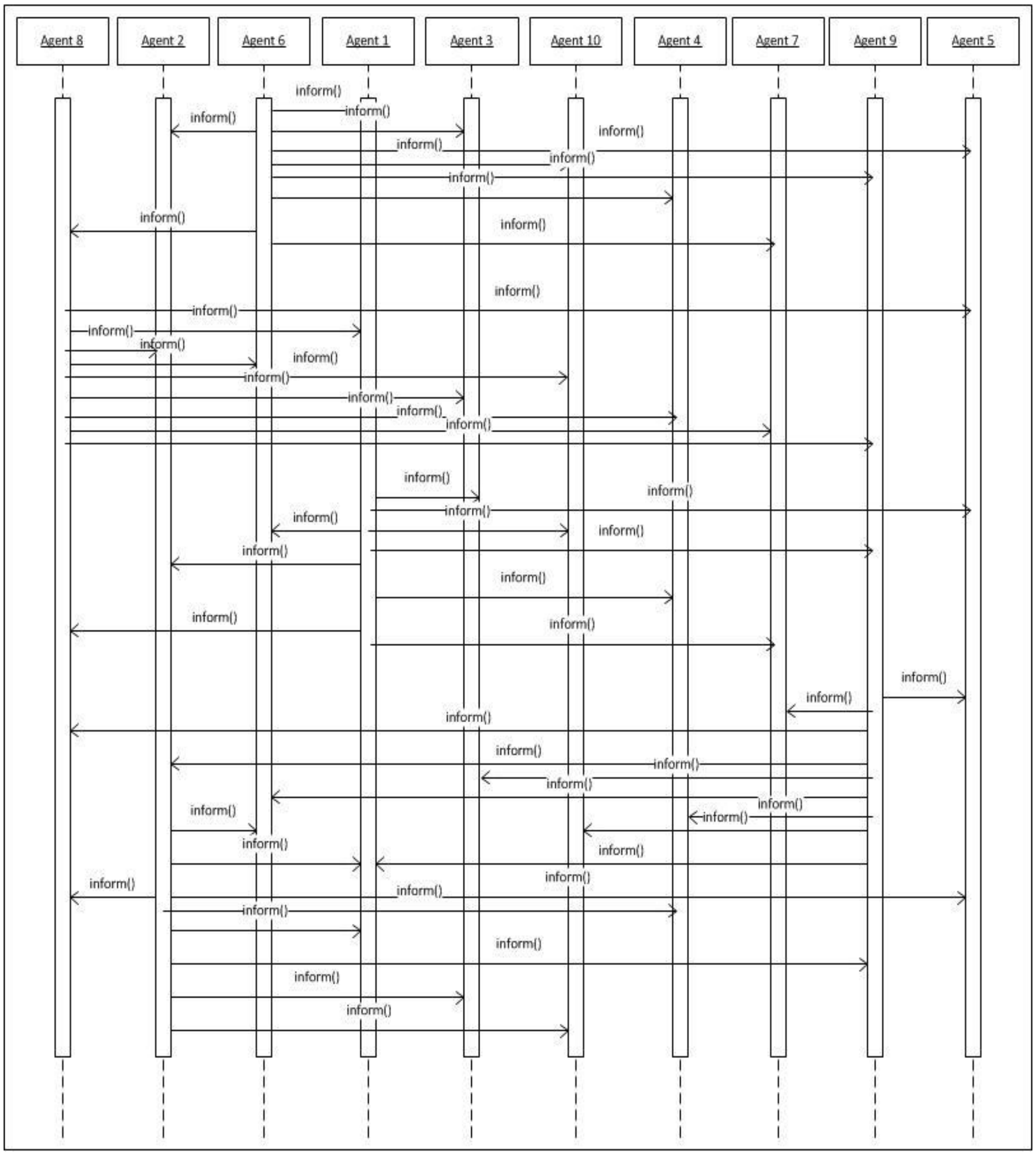


Figure 9: Event diagram showing exchange of messages between agents.

2.2.5. *Agent's Life-Cycle:*

The diagram below displays operations performed by an agent during its life span i.e. from the moment an agent is active until it is terminated (no more registered with the Agent Management System) [2].

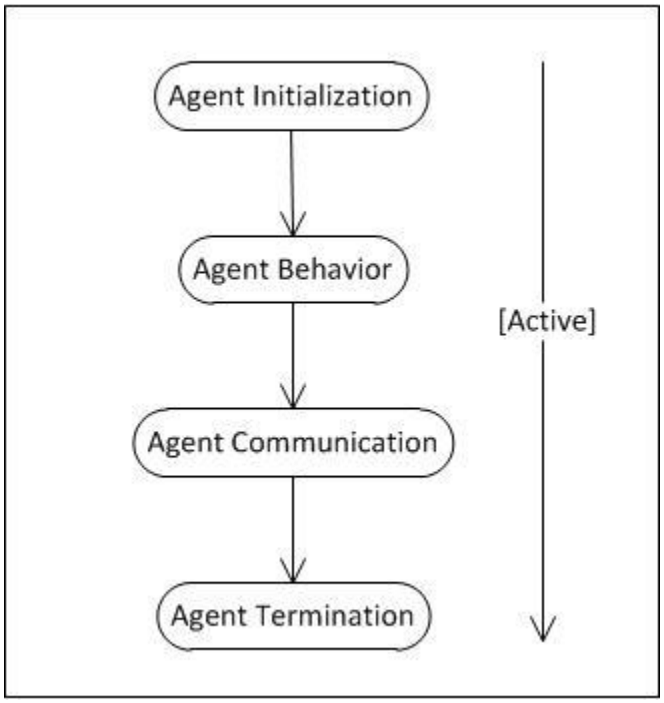


Figure 10: Diagram showing an agent's life-cycle.

2.2.5.1. *Agent Operations:*

A brief description of all the operations identified in the above figure is provided below. Along with the description, names of **JADE API's** to be referenced, which are fundamental to the development of Agent-Box application, are also outlined.

1) Agent Initialization: An agent can be created by extending the *jade.core.agent class* and implementing the *setUp() method* of the class. An agent identifier (AID), object of *jade.core.aid class*, uniquely identifies all the agents on a platform [2].

2) Agent Behavior: The task of an agent is implemented by the Behavior class i.e. by extending *jade.core.behaviours.behaviour class*. A behavior object is created and then agent executes this task by calling the *addBehaviour() method* of the Agent class [1, 2].

3) Agent Communication: The format for exchanging agent messages is specified by ACL language which adheres to FIPA standards. The format contains certain fields that must be exchanged for agents to communicate with one another. In order to send messages to another agent, an object of the *ACLmessage class* needs to be created, then the required fields need to be filled out and finally the *method send()* has to be called. *Method receive()* is called by an agent when it wants to pick received messages from the message queue [1, 2].

4) Agent Termination: An active agent on a platform can be terminated by calling the *doDelete() method* of the *jade.core.agent class*. Once an agent is terminated, *method takeDown()* of the same class is called to complete agent clean-up operations [2].

2.2.6. *Low-Level Design Details:*

This section contains detailed design information that includes illustration of all the important classes, methods and method signatures (parameters passed, data and return types) of the Agent-Box Application.

2.2.6.1. *GameBoard Class:*

This Class handles all the events that are triggered as a result of user operation i.e. it contains action listeners registered to different UI components. Other methods that deal with layout (drawing of lines and images), generation of coordinates for holes, boxes and agents are also defined within this class. The table below provides details on specific methods of the GameBoard class (Table 6).

Table 6: Methods and method signatures of class GameBoard.

Method Name	Description	Return Type	Parameter	Data Type
drawGrid()	To create horizontal and vertical grid lines.	None	g	Graphics
drawHoles()	To display holes on the grid.	None	g	Graphics
drawBoxes()	To display boxes on the grid.	None	g	Graphics
drawAgents()	To display agents on the grid.	None	g	Graphics
initializeHoles ()	To generate random coordinates for placement of holes.	None	none	none
initializeBoxe s()	To generate random coordinates for placement of boxes.	None	none	none
generateNew AgentCoordin ate()	To generate random coordinates for placement of agents.	None	none	none

2.2.6.2. *MASUtility Class:*

This Class contains methods related to creation of Main and Agent Containers, creation of all the System Agents as well as creation of RMA and Sniffer Agents. The table below provides details on specific methods of the MASUtility class (Table 7).

Table 7: Methods and method signatures of class MASUtility.

Method Name	Description	Return Type	Parameter	Data Type
createMainC Ontainer()	To create a Main container using an instance of JADE runtime.	None	none	none
startOtherAge nts()	To create System Agents within the Agent Container.	None	no	int
createRMAA gent()	To create a new RMA agent to monitor agents via a GUI	None	none	none
startSniffer()	To create a new Sniffer agent.	None	none	none

2.2.6.3. *GameOntology Class:*

This Class contains schema that define language and vocabulary used by agents to communicate with one another. This class implements XMLMessageVocab interface that creates and defines vocabulary used for agent communication. The table below provides details on specific methods of the GameOntology class (Table 8).

Table 8: Methods and method signatures of class GameOntology.

Method Name	Description	Return Type	Parameter	Data Type
add()	To add a schema to the Ontology and associate it with Class XMLMessage.	none	as	AgentActionSchema

2.2.6.4. *XMLMessage Class:*

This Class contains methods for creating XML formatted messages and for parsing an XML document. The object of this class is passed to the constructor of the SendMessage class. The table below provides details on specific methods of the XMLMessage class (Table 9).

Table 9: Methods and method signatures of class XMLMessage.

Method Name	Description	Return Type	Parameter	Data Type
aString()	To return a document containing XML formatted messages.	String	none	none
XMLMessage ()	To parse the XML document to extract the values from respective tags.	none	xmlEncodedString	String

2.2.6.5. *SendMessage Class:*

This Class describes behavior for an agent operation using method action(). It extends Class OneShotBehavior and implements the action() method to create a new message of the ACLMessage class. The table below provides details on specific methods of the SendMessage class (Table 10).

Table 10: Methods and method signatures of class SendMessage.

Method Name	Description	Return Type	Parameter	Data Type
action()	To run an agent's behavior. Receivers, Language, Ontology and content of the message are added and then sent.	none	none	none

2.2.6.6. *BoxAgent Class:*

This Class contains methods for agent initializations, agent terminations and agent clean-up operations. Other implemented methods are related to agent's picking up a box, dropping into a hole and sending notifications to other agents. The table below provides details on specific methods of the BoxAgent class (Table 11).

Table 11: Methods and method signatures of class BoxAgent.

Method Name	Description	Return Type	Parameter	Data Type
setup()	Contains agent initializations and startup code.	none	none	none
takedown()	Contains agents clean up operations.	none	none	none
notifyGoingForBox()	Called when an agent settles for the nearest box and wants to send a message to all the other agents.	none	Point Steps	Point int
notifyBoxPicked()	Called when an agent has picked up a box and wants to send a message to all the other agents.	none	Location	Point
notifyGoingForHole()	Called when an agent settles for the nearest hole and wants to send a message to all the other agents.	none	Point Steps	Point int
notifyDroppedBox()	Called when an agent has dropped a box into a hole and wants to send a message to all the other agents.	none	Location	Point

2.2.6.7. *ReceiveMessages Class:*

This Class describes behavior for an agent operation using method action(). It extends Class CyclicBehavior and implements the action() method to create a new message of the ACLMessage class. The table below provides details on specific methods of the ReceiveMessages class (Table 12).

Table 12: Methods and method signatures of class ReceiveMessages.

Method Name	Description	Return Type	Parameter	Data Type
action()	Runs an agent's behavior after receiving and parsing the XML message content.	none	none	none
notifyInterestInObject()	Called when an agent reads the message and shows interest in object.	none	x length	XMLMessage int

2.2.6.8. *BoxHoleCheckBehavior Class:*

This is an abstract class that extends Class OneShotBehavior of Agent class. It contains methods that send notification to system agents on objects that have not yet been identified. The table below provides details on specific methods of the BoxHoleCheckBehavior class (Table 13).

Table 13: Methods and method signatures of class BoxHoleCheckBehavior.

Method Name	Description	Return Type	Parameter	Data Type
notifyUnnotified()	Sends a notification message, containing an unidentified object's location, to all the other agents on the grid.	none	none	none
getVisisbleUnNotifiedHolesBoxes()	This method stores information on unidentified boxes and holes location.	none	none	none

2.2.6.9. *GetBoxBehavior Class:*

This Class implements the action() method that deals with identification of the closest box. The agent then sends a message, about its choice of a box, to all the other agents. This class extends BoxHoleCheckBehavior class. The table below provides details on specific methods of the GetBoxBehavior class (Table 14).

Table 14: Method and method signatures of class GetBoxBehavior.

Method Name	Description	Return Type	Parameter	Data Type
action()	This method calculates the coordinates of the nearest box.	none	none	none

2.2.6.10. *GetHoleBehavior Class:*

This Class implements the action() method that deals with identification of the closest hole. The agent then sends a message, about its choice of a hole, to all the other agents. This class extends BoxHoleCheckBehavior class. The table below provides details on specific methods of the GetHoleBehavior class (Table 15).

Table 15: Method and method signatures of class GetHoleBehavior.

Method Name	Description	Return Type	Parameter	Data Type
action()	This method calculates the coordinates of the nearest hole.	none	none	none

2.2.6.11. *Navigation Class:*

This Class deals with agent's movement on the grid. The agent's can navigate to a point if they know the location or can randomly move on the grid if they are unaware of an object's location. An object of this class is passed to the constructor of the BoxAgent class. Each agent has a navigation object that tells that is responsible for making sure the agent follows this path. The table below provides details on specific methods of the Navigation class (Table 16).

Table 16: Methods and method signatures of class Navigation.

Method Name	Description	Return Type	Parameter	Data Type
navigateTo()	This method allows an agent to move towards an identified point.	none	p	Point
randomNavig ation()	This method allows agents to wander randomly on the grid.	none	none	none
setTarget()	This method allows for navigation towards a targeted object.	none	p	Point
navigate()	This method allows already loaded agents to move towards their destination if they are aware of the object's location and the path.	none	none	none
goingToAHol e()	This is a Boolean method that returns true if an agent is moving towards a targeted hole.	Boolean	none	none
isGoingToAB ox()	This is a Boolean method that returns true if an agent is moving towards a targeted box.	Boolean	none	none

2.2.7. *Design Rationale:*

Some of the important decisions made during the design process and reasons driving those decisions are briefly described below.

2.2.7.1. *Concurrent Hash Map:*

Concurrent Hash Map {ConcurrentHashMap<String, Integer[]>()}will be used to initialize holes, boxes and agents on the grid. Since all active agents on the grid are threads and will simultaneously access and modify maps; the program should be thread safe while allowing for concurrent modifications.

2.2.7.2. *A-Star Algorithm:*

This search algorithm will be used for grid traversal. The A-star library has method that takes input as two positions and plots the shortest path between them avoiding obstacles if any. It is simple to implement and is known for its performance.

2.2.7.3. *Ontology:*

The Ontology that contain agent schemas, will be defined as a Singleton object. This will allow only one Ontology object to be shared among different agents on the same Java Virtual Machine.

2.2.8. *Additional Design Considerations:*

System Agent monitoring (temporarily suspension of grid agents or resumption of grid agents after a certain period of time) using a Remote Agent Management GUI and displaying agent notifications via a Sniffer GUI will be some of the main challenges.

2.2.8.1. *Performance Considerations:*

1. The time difference between messages sent, received and processed by an agent will play a crucial role as the negotiations between agents may see messages being exchanged to the tune of 1000 in a few seconds.
2. The launch of JADE's Remote Management GUI and Sniffer GUI may slow down the application considerably.

2.2.9. *User Interface Design:*

The figure below represents the initial prototype of user interface. The interface is designed in a way to keep a user's interaction simple and efficient. Boxes shall be represented using solid square images, holes shall be indicated with a letter 'H' and agents shall be identified uniquely on the grid. There will be buttons to add more boxes, holes or agents. The 'Log' section of the interface shall display all the messages exchanged between agents.

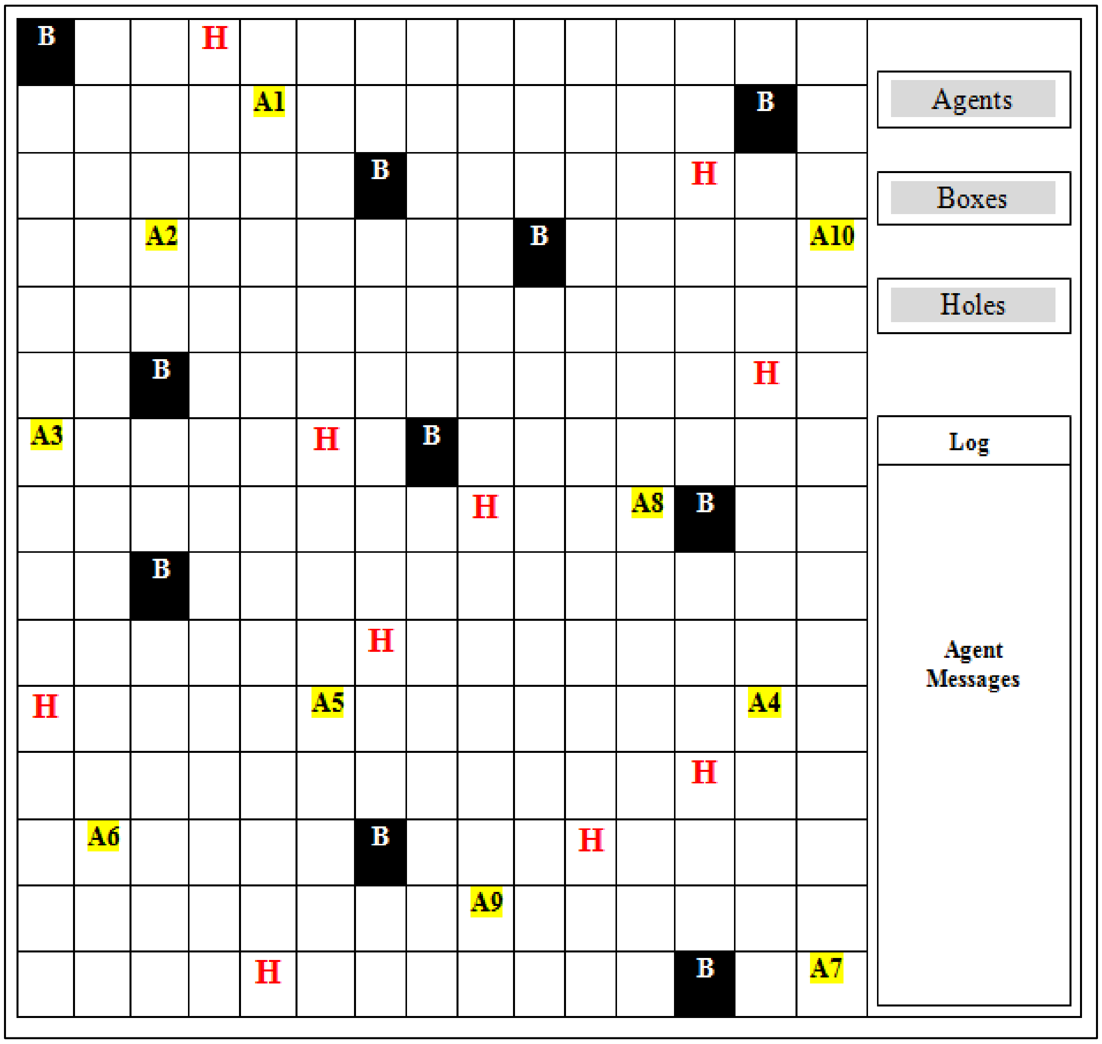


Figure 11: Initial user interface prototype.

2.3. System Implementation:

In this phase the actual source code will be written and system will be built from the design prototype specified in the previous section. The goal of this phase is to translate and implement the design into program code in a manner that will help to reduce testing and maintenance efforts later.

The source code will follow an efficient and maintainable coding style and will adhere to Sun Java coding recommendations [20]. Some of the conventions that will be adopted are laid out below.

1. There will be consistent indentation through all the program statements so as to make the code more readable. 4 spaces shall be used to indent.
2. Classes, Methods and Variables shall have meaningful names and will convey their intent of use. Variable and Method names will be mixed case and will start with a lower case.
3. Classes related to one another will be bundled under one package.
4. Methods will perform one specific task and reasonable number of lines of code will be written.
5. Block and Inline comments will be used to describe a method's purpose.

Code reviews and inspections will be performed during the entire development life-cycle of Agent-Box application. Multiple passes through the code, keeping in mind the requirement and design specifications, will be made. These reviews will help in early detection of issues and potential problems [7]. The defects will be analyzed and corrected before moving onto the next phase.

2.4. System Testing:

The following section shall discuss the testing approach that will be adopted to validate the quality of Agent-Box application. Validation will be done against outlined specification and will ensure that application fulfills the intended purpose.

2.4.1. *Testing Approach:*

The functionality of the application shall be examined using the Black-box testing strategy. Adequate number of Test Cases shall be created to cover all the implemented functional and non-functional requirements of the application. Sufficient GUI testing (covering all user clicks) shall be done to ensure application interface is in accordance with the laid out specifications.

2.4.2. *Test Items:*

Acceptable number of Black-box test cases will be identified to test the implemented functionalities. The features to be tested are as follows:

1. Agent's placement.
2. Agent's notification.
3. Agent's visibility.
4. Agent's negotiation.
5. Agent's navigation.
6. Interface testing.

2.4.3. *Test Cases:*

The following table lists all set of conditions as well as adequate number of test cases necessary to test all the implemented functionalities of the application (Table 17).

Table 17: Black-box test cases table.

Test ID	Purpose	Pre-Condition	Input	Expected Output	Pass
TC#1	Ensure random placement of Agents on the grid.	Application is launched. 'Start Agents' button is active.	User clicks 'Start Agents' button.	Upon clicking the 'Start Agents' button, Agents should be placed randomly on the grid.	X
TC#2	Ensure placement of Agents do not overlap with placement of boxes on the grid.	Application is launched. 'Start Agents' button is active.	User clicks 'Start Agents' button.	Upon clicking the 'Start Agents' button, Agents should be placed in different coordinates than the coordinates of the boxes.	X

(continued)

Table 17: Black-box test cases table (continued).

Test ID	Purpose	Pre-Condition	Input	Expected Output	Pass
TC#3	Ensure placement of Agents do not overlap with placement of holes on the grid.	Application is launched. 'Start Agents' button is active.	User clicks 'Start Agents' button.	Upon clicking the 'Start Agents' button, Agents should be placed in different coordinates than the coordinates of the holes.	X
TC#4	Ensure that notification contains sending agent's name.	All Agents are active. Agent sees an object in its line of sight.	User clicks 'Start Agents' button.	User should be able to view the notification containing agent's name in the 'Log' section of the interface.	X
TC#5	Ensure that notification contains type of object identified by an agent.	All Agents are active. Agent sees an object in its line of sight.	User clicks 'Start Agents' button.	User should be able to view the notification along with the object type in the 'Log' section of the interface.	X
TC#6	Ensure that notification contains information on object's location.	All Agents are active. Agent sees an object in its line of sight.	User clicks 'Start Agents' button.	User should be able to view the notification along with the object location information in the 'Log' section of the interface.	X
TC#7	Ensure that notification contains path taken by an agent to reach its destination.	All Agents are active. Agent is aware of its destination.	User clicks 'Start Agents' button.	User should be able to view the notification along with the path traversed by an agent to reach its destination in the 'Log' section of the interface.	X
TC#8	Ensure an agent is able to send a notification to all other agents after identifying an object.	All Agents are active. Agent sees an object in its line of sight.	User clicks 'Start Agents' button.	User should be able to view the notification in the 'Log' section of the interface.	X
TC#9	Ensure an agent is able to send a notification to all other agents after picking up a box.	All Agents are active. Agent sees a box in its line of sight.	User clicks 'Start Agents' button.	User should be able to view the notification in the 'Log' section of the interface.	X

(continued)

Table 17: Black-box test cases table (continued).

Test ID	Purpose	Pre-Condition	Input	Expected Output	Pass
TC#10	Ensure an agent is able to send a notification to all other agents after dropping a box into a hole.	All Agents are active. Agent sees a hole in its line of sight.	User clicks 'Start Agents' button.	User should be able to view the notification in the 'Log' section of the interface.	X
TC#11	Ensure an agent is able to send a notification to all other agents, when it sees an object, on its way towards another targeted object.	All Agents are active. Agent sees an object in its line of sight.	User clicks 'Start Agents' button.	User should be able to view the notification in the 'Log' section of the interface.	X
TC#12	Ensure that an agent can only see an object up to 4 squares from its current location.	Agent is active. Agent identifies an object in its line of sight.	User clicks 'Start Agents' button.	User should be able to view a notification in the 'Log' section as soon as an agent sees an object within a distance of 4 squares from its current location.	X
TC#13	Ensure that an agent cannot see an object beyond 4 squares from its current location.	Agent is active. Object is located at a distance greater than 4 squares.	User clicks 'Start Agents' button.	User should not be able to view any notification in the 'Log' section.	X
TC#14	Ensure that an agent can only see an object in 4 directions from its current location.	Agent is active. Agent identifies an object in its line of sight.	User clicks 'Start Agents' button.	User should be able to view a notification in the 'Log' section as soon as an agent sees an object in any of the 4 directions from its current location.	X
TC#15	Ensure that an agent cannot see an object except for its line of sight from its current location.	Agent is active. Object is located in a square that does not fall in agent's line of sight.	User clicks 'Start Agents' button.	User should not be able to view any notification in the 'Log' section.	X

(continued)

Table 17: Black-box test cases table (continued).

Test ID	Purpose	Pre-Condition	Input	Expected Output	Pass
TC#16	Ensure that an agent approaches an object at the shortest distance from its current location.	Agent is active. Agent has knowledge of location of all objects in its line of sight.	User clicks 'Start Agents' button.	User should be able to view notification in the 'Log' section on agent's interest in an object at the shortest distance.	X
TC#17	Ensure that if two or more agents qualify for an object then agent whose name comes first in alphanumeric order shall approach that object.	Agent is active. More than one agents are interested in an object.	User clicks 'Start Agents' button.	User should be able to view notification in the 'Log' section on an agent that will qualify in case of a tie. The agent which comes first in alphanumeric order should approach the object.	X
TC#18	Ensure that an unloaded agent is able to pick up a box once it has knowledge of its box of interest.	Agent is active. Agent identifies a box. Agent informs all other agents about its interest in a box.	User clicks 'Start Agents' button.	User should be able to view notification in the 'Log' section after an agent picks up the box in which it was interested.	X
TC#19	Ensure that an already loaded agent shall not be able to pick another box until it deposits the first box, which it is carrying, into a hole.	Agent is active. Agent has picked up a box. Agent sees another box in its line of sight.	User clicks 'Start Agents' button.	User should not be able to view a notification in the 'Log' section about a loaded agent picking up a box.	X
TC#20	Ensure that an agent carries a box towards a hole once it identifies a hole.	Agent is active. Agent has picked up a box. Agent has identified a hole.	User clicks 'Start Agents' button.	User should be able to view a notification in the 'Log' section that displays information on the location of a hole.	X

(continued)

Table 17: Black-box test cases table (continued).

Test ID	Purpose	Pre-Condition	Input	Expected Output	Pass
TC#21	Ensure that an agent deposits a box into a hole once it has knowledge of the hole.	Agent is active. Agent has picked up a box. Agent has identified a hole.	User clicks 'Start Agents' button.	User should be able to view a notification in the 'Log' section that displays information on agent's path once an agent deposits a box into a hole.	X
TC#22	Ensure that an agent wanders randomly on the grid looking for other unassigned boxes.	Agent is active. Agent is unloaded.	User clicks 'Start Agents' button.	User should be able to see that unloaded agents wander randomly on the grid in search of other unassigned boxes.	X
TC#23	Ensure that an agent is able to make a left turn on the grid, depending upon the knowledge of objects.	Agent is active.	User clicks 'Start Agents' button.	User should be able to see that an agent can make a left turn on the grid.	X
TC#24	Ensure that an agent is able to make a right turn on the grid, depending upon the knowledge of objects.	Agent is active.	User clicks 'Start Agents' button.	User should be able to see that an agent can make a right turn on the grid.	X
TC#25	Ensure that an agent is able to move forward on the grid, depending upon the knowledge of objects.	Agent is active.	User clicks 'Start Agents' button.	User should be able to see that an agent can move in a forward direction on the grid.	X
TC#26	Ensure that an agent is able to move backward on the grid, depending upon the knowledge of objects.	Agent is active.	User clicks 'Start Agents' button.	User should be able to see that an agent can move in a backward direction on the grid.	X

(continued)

Table 17: Black-box test cases table (continued).

Test ID	Purpose	Pre-Condition	Input	Expected Output	Pass
TC#27	Ensure that unloaded agents cannot move through squares containing holes.	Agents are active. An Unloaded Agent encounters a hole in its direction of movement.	User clicks ‘Start Agents’ button.	User should be able to see that unloaded agents take another path if they encounter a hole in their original path of movement.	X
TC#28	Ensure that already loaded agents cannot move through squares containing another box.	Agents are active. A loaded Agent encounters a box in its direction of movement.	User clicks ‘Start Agents’ button.	User should be able to see that loaded agents take another path if they encounter a box in their original path of movement.	X
TC#29	Ensure that Loaded Agents are able to cross one another’s path on the grid.	Agents are active. A loaded agent encounters another Loaded Agent in its direction of movement.	User clicks ‘Start Agents’ button.	User should be able to see that loaded agents are able to cross each other’s path while moving on the grid.	X
TC#30	Ensure that unloaded agents are able to cross paths with other agents on the grid.	Agents are active. Unloaded agents encounter other agents in their direction of movement.	User clicks ‘Start Agents’ button.	User should be able to see that unloaded agents are able to cross paths with all other agents on the grid.	X
TC#31	Ensure that a Remote Agent Management GUI is launched when user clicks ‘R’ button on the interface.	‘R’ button must be active.	User clicks ‘R’ button.	A Remote Management Agent GUI is launched.	X

(continued)

Table 17: Black-box test cases table (continued).

Test ID	Purpose	Pre-Condition	Input	Expected Output	Pass
TC#32	Ensure that a Sniffer GUI is launched when user clicks ‘S’ button on the interface.	‘S’ button must be active.	User clicks ‘S’ button.	A Sniffer GUI is launched.	X
TC#33	Ensure that user is able to filter messages of a particular agent using the ‘Filter for’ combobox.	‘Filter for’ combobox shows all agents.	User selects for his/her choice of agents from the combobox.	User should be able to see all the messages related to an agent depending upon his/her selection from the combobox.	X
TC#34	Ensure that a user is able to reset messages in the ‘Log’ section of the interface by clicking the ‘Refresh’ button.	‘Refresh’ button is active.	User clicks ‘Refresh’ button.	User should be able to see updated messages in the ‘Log’ section when he/she clicks the ‘Refresh button.’	X
TC#35	Ensure that the application does not crash due to programming errors.	Application is launched.	User starts the game by clicking ‘Start Agents’ button.	User should not encounter any errors until the game ends.	X
TC#36	Ensure that the application runs correctly when transported to other systems.	Other systems must have JDK 1.5 or above installed.	User starts the game by clicking ‘Start Agents’ button.	User should not encounter any errors until the game ends.	X

3. ISSUES

The issues found during testing of Agent-Box application are identified below.

1) JADE uses a queuing mechanism to send and receive messages. The messages are not sent immediately or processed immediately upon reception as they are placed in a queue. The time difference between message sent by an agent and message processed by a receiving agent may cause delay.
2) The untimely reception on negotiating messages may cause one agent to change direction a few times due to becoming both qualified and disqualified for an object of interest within a short span of time.
3) If an agent was interested in an object but lost it out on the criteria to another agent, it then treats that square as empty and continues on a random motion, which may sometime include a path through the same object.

4. SCREEN CAPTURES

The section below contains screen shots of the Agent-Box application. The images display different stages from launch of the application till the end of the game. Screen shots of JADE’s Remote Agent Management GUI and Sniffer Agent GUI are also provided.

The launch of the application shall display boxes and holes placed randomly on the grid. A solid black square depicts a box and a red circle marked with an alphabet ‘H’ depicts a hole. The agents will appear upon clicking the ‘Start Agents’ button.

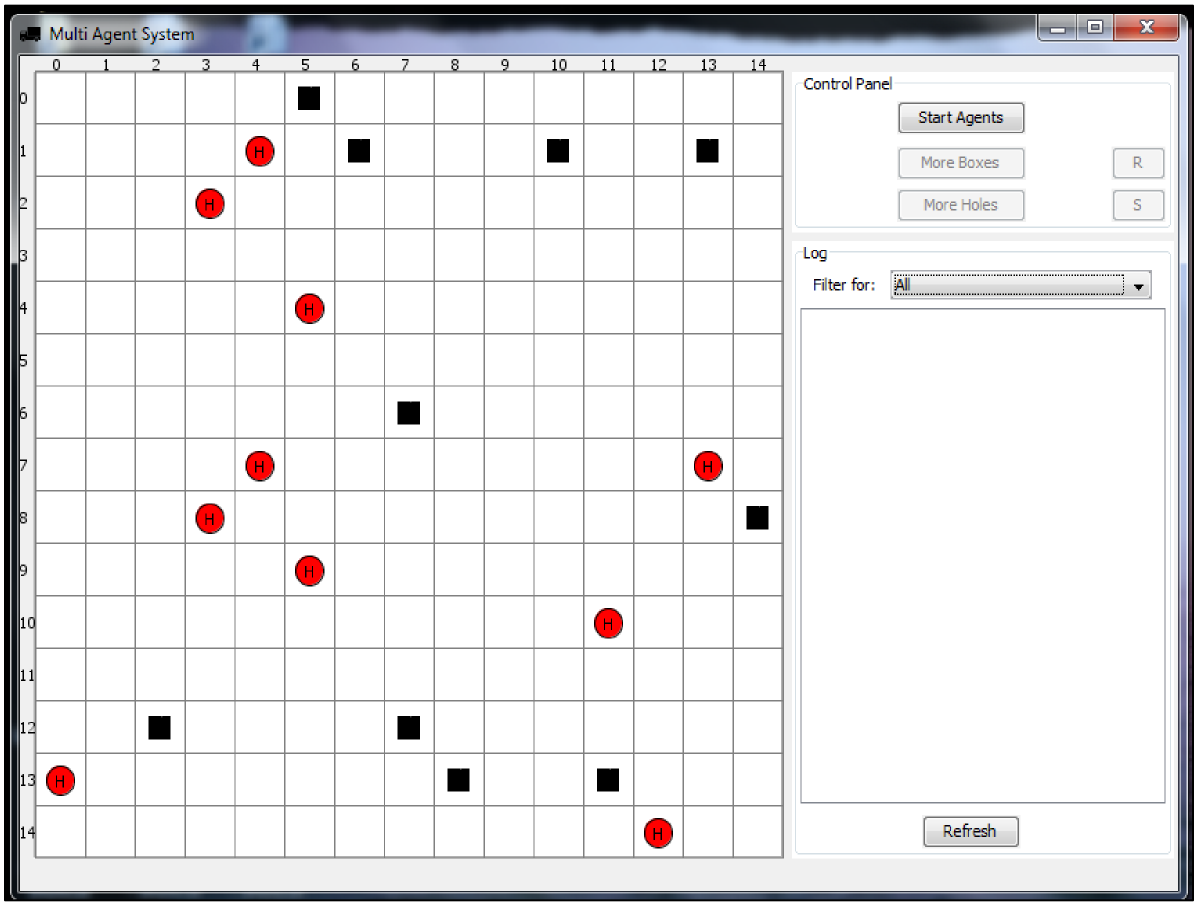
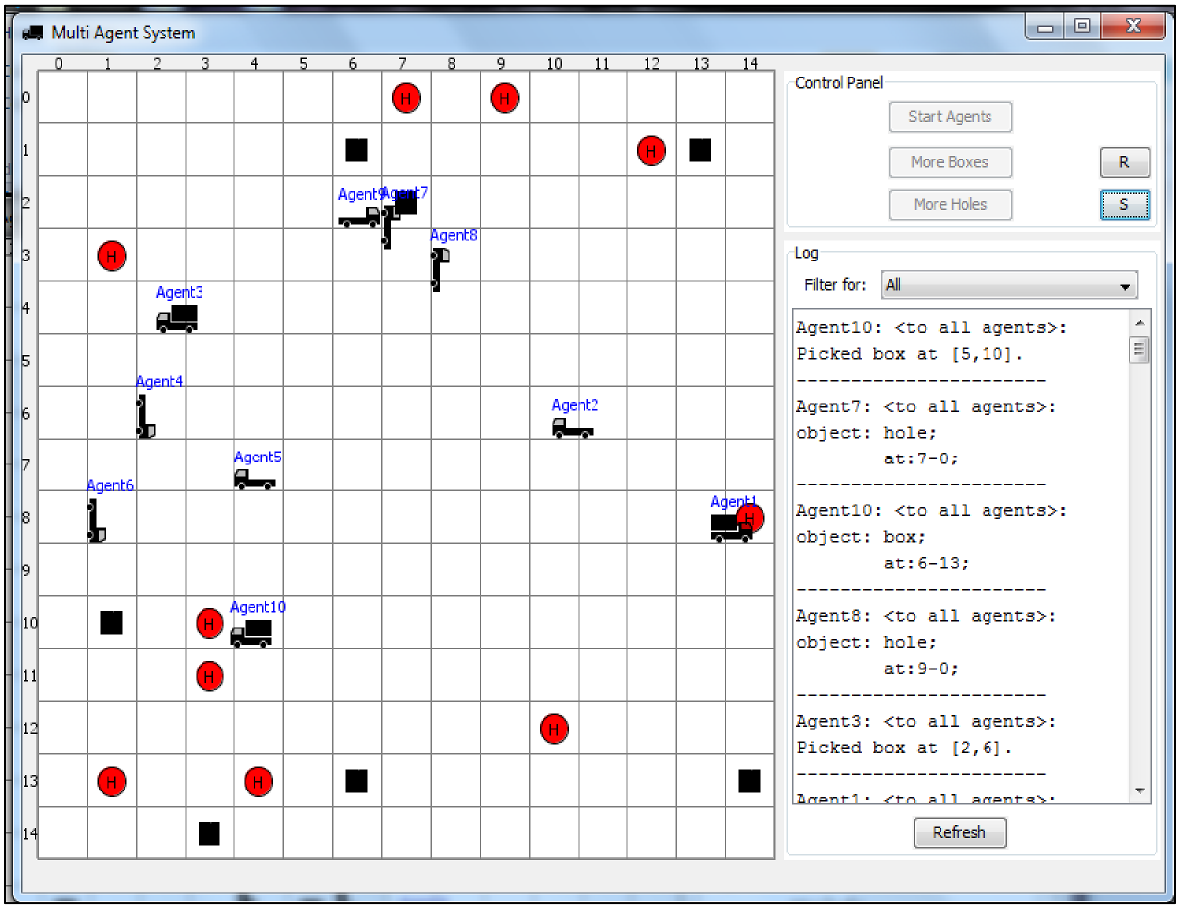


Figure 12: Image displaying first stage after the application is launched.

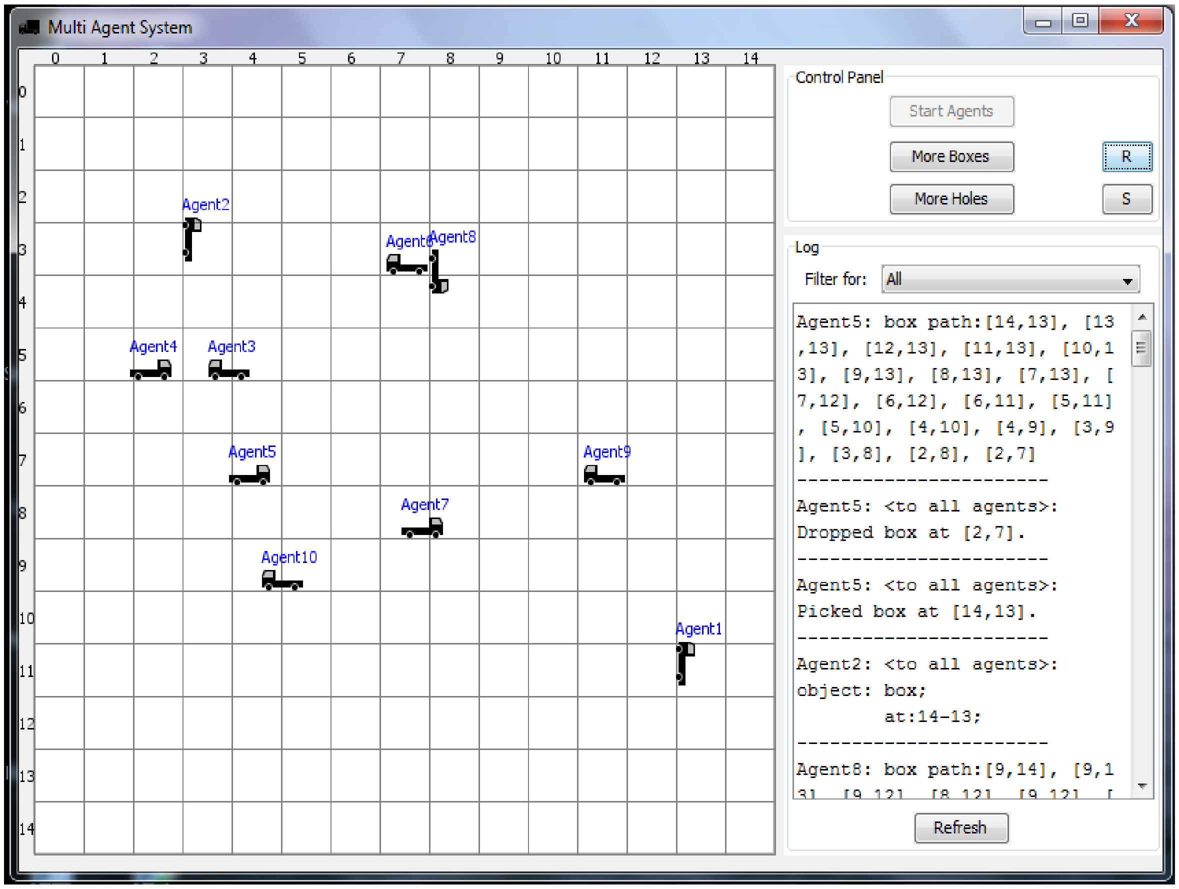
The agents are depicted using a lorry and are uniquely numbered. Once an agent picks a box, it remains loaded until it drops the box into a hole. The hole disappears from the grid upon obtaining a box. Agents wander randomly if they do not have knowledge of objects.



```
Agent10: <to all agents>:
Picked box at [5,10].
-----------------------
Agent7: <to all agents>:
object: hole;
        at:7-0;
-----------------------
Agent10: <to all agents>:
object: box;
        at:6-13;
-----------------------
Agent8: <to all agents>:
object: hole;
        at:9-0;
-----------------------
Agent3: <to all agents>:
Picked box at [2,6].
-----------------------
Agent1: <to all agents>:
```

Figure 13: Image displaying a stage after user starts the game.

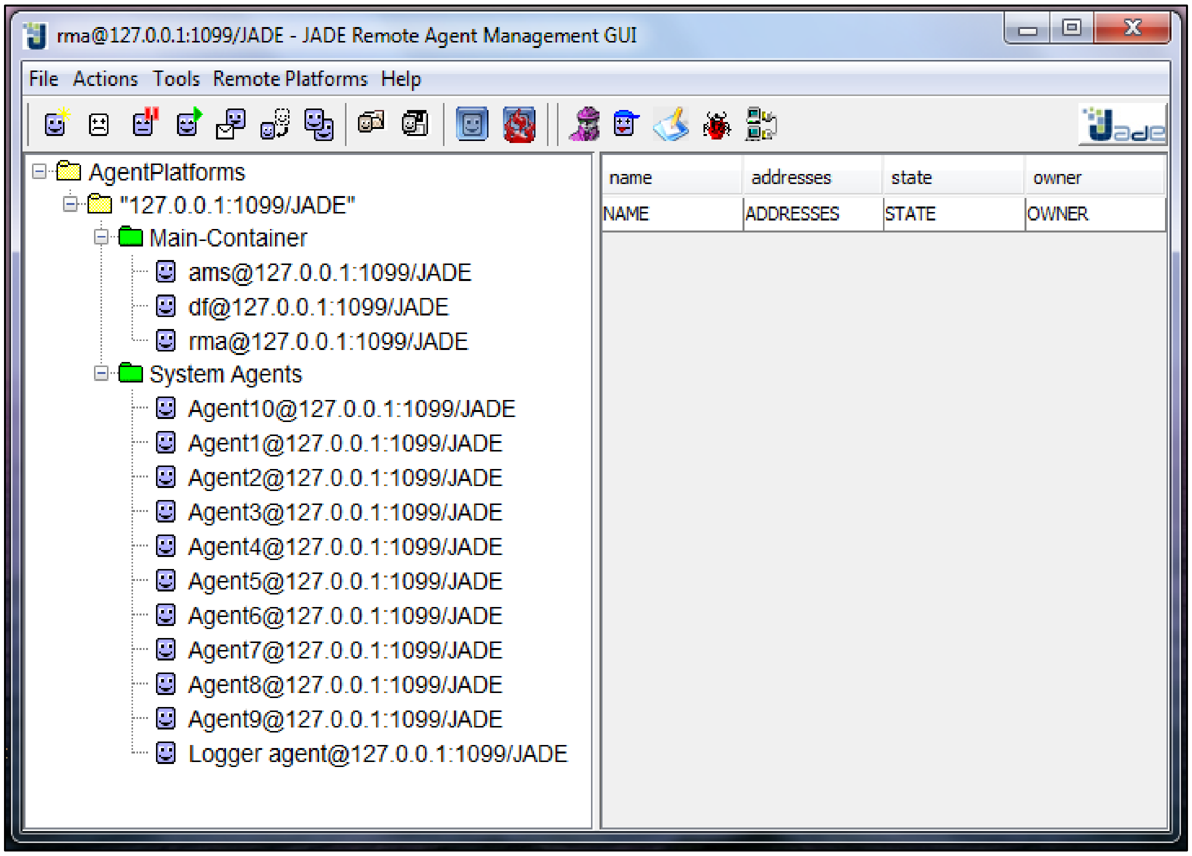
The messages exchanged between agents are displayed in the ‘Log’ section of the interface. A user can filter the log depending upon his or her choice of agents. The buttons ‘More Boxes’ and ‘More Holes’ become highlighted when all the objects are consumed. Upon clicking these buttons, boxes and holes are placed randomly on the grid and the game begins again. The agents wander randomly and remain active until the close of the application.



```
Agent5: box path:[14,13], [13
,13], [12,13], [11,13], [10,1
3], [9,13], [8,13], [7,13], [
7,12], [6,12], [6,11], [5,11]
, [5,10], [4,10], [4,9], [3,9
], [3,8], [2,8], [2,7]
-----------------------
Agent5: <to all agents>:
Dropped box at [2,7].
-----------------------
Agent5: <to all agents>:
Picked box at [14,13].
-----------------------
Agent2: <to all agents>:
object: box;
        at:14-13;
-----------------------
Agent8: box path:[9,14], [9,1
```

Figure 14: Image displaying end of the game.

The instance of a JADE run time environment is also known as a Container and it contains all the agents. The first container that is active on a platform is the Main Container. All other containers, non-main containers, must register with the Main Container. Along with other active agents, the Main Container also holds three other agents that are automatically created by JADE at the start of Main Container. These agents are the Remote Management Agent (RMA), the Agent Management System (AMS) and the Directory Facilitator agent (DF) [2].


rma@127.0.0.1:1099/JADE - JADE Remote Agent Management GUI
File Actions Tools Remote Platforms Help

```
AgentPlatforms
  "127.0.0.1:1099/JADE"
    Main-Container
      ams@127.0.0.1:1099/JADE
      df@127.0.0.1:1099/JADE
      rma@127.0.0.1:1099/JADE
    System Agents
      Agent10@127.0.0.1:1099/JADE
      Agent1@127.0.0.1:1099/JADE
      Agent2@127.0.0.1:1099/JADE
      Agent3@127.0.0.1:1099/JADE
      Agent4@127.0.0.1:1099/JADE
      Agent5@127.0.0.1:1099/JADE
      Agent6@127.0.0.1:1099/JADE
      Agent7@127.0.0.1:1099/JADE
      Agent8@127.0.0.1:1099/JADE
      Agent9@127.0.0.1:1099/JADE
      Logger agent@127.0.0.1:1099/JADE
```

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

Figure 15: Image displaying Remote Agent Management user interface.

Agent interactions showing sequence of messages exchanged are captured using Sniffer tool provided by the JADE framework. The screen image below displays agent notification messages recorded by the Sniffer tool.

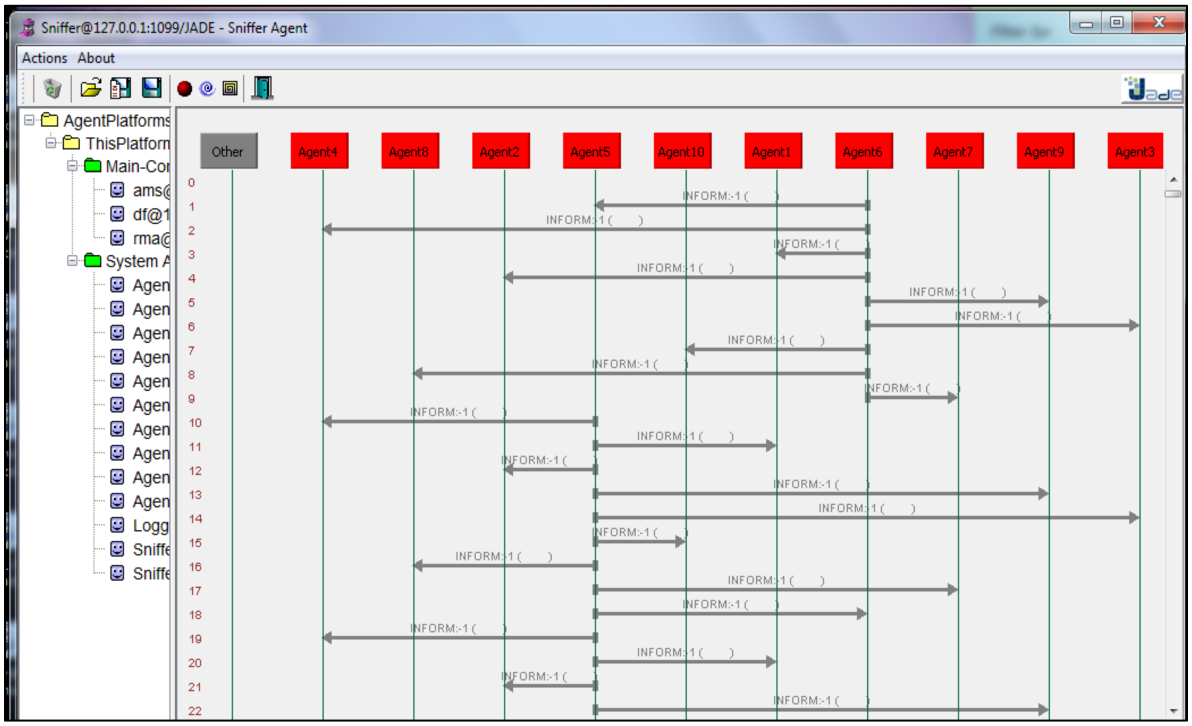


Figure 16: Image displaying messages exchanged between agents.

5. CONCLUSION

The multi-agent Agent-Box application has been successfully designed, implemented and tested to meet all the outlined specifications. The application effectively demonstrates collaboration techniques employed by intelligent agents to accomplish their objective within an environment. The agents not only carry out their duties autonomously but also adequately communicate with one another to achieve their specific goals.

The Agent-Box application analyzes behavior and intelligence of multiple agents by allowing them to communicate with each other and by varying their awareness. The agents are able to communicate at all the times; after identifying an object, after picking up a box and after dropping a box into a hole. The agents have an added awareness as they are able to see objects in their line of sight at four square distances from their current location and in all four directions. The agents negotiate with other agents before they settle for an object of their interest. They then traverse the shortest path to reach that object. All these operations and activities are carried out by agents so that they are able to make independent decisions and achieve their common goal without any user intervention. The application shows that agents on the grid carry out their task efficiently and fill all the holes in a minimum possible time. This Agent-Box application successfully demonstrates that agents are smart and capable enough to accomplish their objective without any assistance by an outside source.

6. REFERENCES

[1] Bellifemine, F. Caire, G. Trucco, T. Rimassa, G. JADE Programmer's Guide, (2010).

[2] Caire, G. JADE Tutorial: JADE Programming for Beginners, (2009).

[3] Software agent: http://en.wikipedia.org/wiki/Software_agent, (retrieved on September 15th, 2012).

[4] Java Agent Development Framework: an Open Source platform for peer-to-peer agent based applications: http://jade.tilab.com/, (retrieved on October 10th, 2012).

[5] JADE v4.3.0 API: http://jade.tilab.com/doc/api/index.html, (retrieved on October 10th, 2012).

[6] Autonomous Agents and Multi-Agent Systems for Healthcare: http://www.openclinical.org/agents.html, (retrieved on September 24th, 2012).

[7] Strategies for Effective Code Reviews: http://www.basilv.com/psd/blog/2007/strategies-for-effective-code-reviews, (retrieved on December 20th, 2012).

[8] Java Agent Development Framework: http://en.wikipedia.org/wiki/Java_Agent_Development_Framework, (retrieved on September 24th, 2012).

[9] Bowman, R. S. Hexmoor, H. Agent Collaboration and Social Networks, Proceedings of Knowledge Intensive Multiagent Systems (KIMAS), 211–214, Boston, MA, (2005).

[10] Nwana, H. S. Software Agents: An Overview, The Knowledge Engineering Review, 11, 205–244, (1996).

[11] JADE Tutorial and Primer: http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html, (retrieved on October 10th, 2012).

[12] JADE LOGGING SERVICE: http://jade.tilab.com/doc/tutorials/logging/JADELoggingService.html, (retrieved on October 28th, 2012).

[13] AStar class: http://www.robotacid.com/PBeta/AILibrary/AStar/AStar.html, (retrieved on October 10th, 2012).

[14] Hayes-Roth, B. Brownston, L. Gent, R. V. Multiagent Collaboration in Directed Improvisation, Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), Menlo Park, California, AAAI Press, 148–154, (1995).

[15] Nwana, Hyacinth S. Nduma, Divine T. A Perspective on Software Agents Research, The Knowledge Engineering Review, 14, 1–18, (1999).

[16] Bordini, R. H. Dastani, M. Winikoff, M. Current Issues in Multi-Agent Systems Development, Post-proceedings of the Seventh Annual International Workshop on Engineering Societies in the Agents World, volume 4457 of Lecture Notes in Artificial Intelligence, 38–61, (2006).

[17] The Foundation for Intelligent Physical Agents: http://www.fipa.org/, (retrieved on September 24th, 2012).

[18] Java 2 Platform Standard Edition 5.0 API Specification: http://docs.oracle.com/javase/1.5.0/docs/api/, (retrieved on October 10th, 2012).

[19] Montero, F. Navarro, E. ATRIUM: Software Architecture Driven by Requirements, Proceedings 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'09), IEEE Press, (2007).

[20] Java Programming Style Guidelines: http://geosoft.no/development/javastyle.html, (retrieved on October 16th, 2012).

[21] Cortese, E. Quarta, F. Vitaglione, G. Scalability and Performance of JADE Message Transport System, AAMAS Workshop, Bologna, (2002).

[22] Unified Modeling Language: http://en.wikipedia.org/wiki/Unified_Modeling_Language, (retrieved on March 24th, 2013).

[23] Waterfall Model: http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Waterfall_model.html, (retrieved on September 15th, 2012).