Warehouse Layout and Picking Simulation

New Mexico

Supercomputing Challenge

Final Report

April 4, 2012

Team 64

Los Alamos High School

Team Members:

Sudeep Dasari

David Murphy

Colin Redman

Teachers:

Lee Goodwin

Mentors:

Elizabeth Cooper

Executive Summary:

This project explores what factors contribute to increased order picking time in a specific warehouse layout. Warehouses are extremely complex and as a result an difficult portion of the supply chain to model. Of all the processes that occur inside a warehouse order picking is probably the most important as it accounts for 55% of all total costs. In order to determine the biggest contributors to inefficiencies in the picking process we created a simulation that can model the picking process in a warehouse. The simulation was an agent based model written in the Java programming language and built with the MadKit platform. Agents represented the various actors present in a warehouse, such as the pickers, conveyor segment, and the warehouse itself. warehouse layout program was also created in order to feed the simulation information related to the warehouses' layout, and a database was used to store item stock, incoming orders, and final results of the simulation. The simulation was subsequently used to determine some interesting relationships between the warehouse layout and the picking times for a pre-determined set of customer orders. The parameters changed included the number of orders allowed to be processed at one time, whether or not the picker was collecting multiple items on a picking route, how the items were distributed (sorted) on the shelves, and after our results were analyzed it was found that there were many parameters that had an influence on the picking time.

Problem Statement:

As e-commerce companies such as Amazon become the dominant way people buy goods, increasing strain has been put on getting an order to a customer as quickly, reliably, and efficiently as possible [2]. A critical part of this supply chain is the logistics of the warehouse that these online companies use to store products until they are purchased, picked, and shipped. Inarguably, the most important function in a warehouse is the picking process, which accounts for 55% of operational expenses [1]. Since under-performance in picking can lead to bad overall effectiveness and high operational costs, it is imperative that a company's warehouse is organized in such a way that it allows for the most effective order picking possible [3].

The basic principle behind order picking is that a worker, called a "picker", receives an order and picks its components off of storage shelves, after which he or she deposits the items in a tote which is placed on a conveyor belt. Figure 1 below demonstrates linear order picking across a shelf.

Figure 1- Depicts a picker moving along a pick line to pick items x and y.

There are two basic implementations of order picking, zone and discrete order picking. "In discrete order picking a single worker walks to pick all the items necessary to fulfil a single customer order" (Eisenstein, 2006). Discrete order picking is demonstrated in Figure 2. As you can see, a picker is moving throughout the warehouse

picking the various parts of an order before returning to a depot to return the fully picked order. The picker then gets the next order and repeats the process.

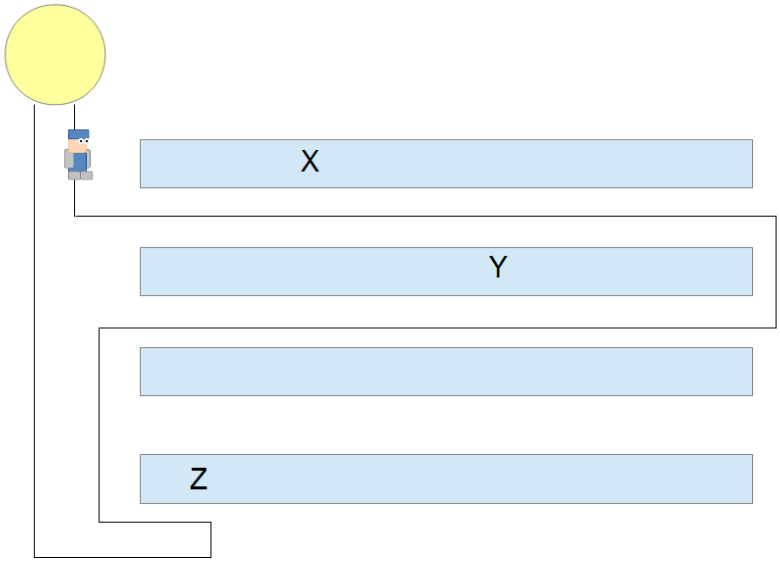


Figure 2- Depicts discrete order picking implemented in such a way that the picker starts at the depot, picks items x, y, and z, and then returns to the depot to get the next order.

In zone picking, each picker is assigned a zone and he or she picks all the items of a set of orders that occur in his or her zone. Once all the items in a zone are picked, they are put on a conveyor in units called totes, before being batched together into their separate orders and sent to a shipping area [1], [3].Figure 3 shows a picker picking all the various items in an order that fall in his row before depositing them on a conveyor.

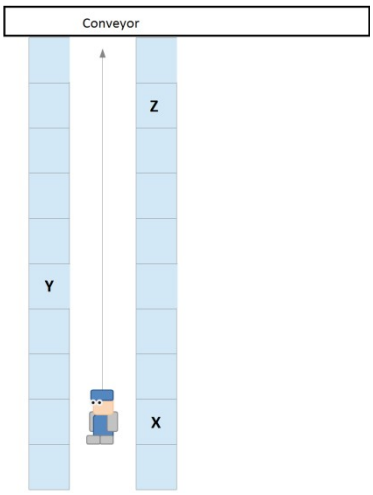


Figure 3- Depicts a picker moving in his zone, picking items x, y, and z, and then depositing them on a conveyor.

While discrete order picking has many advantages, mainly reliability and simplicity, zone picking is definitely the better organizational scheme in a warehouse due to significantly decreased travel times on smaller orders [3]. Simply put, zone picking offers faster pick times than discrete order picking, with the same reliability.

The most important thing to consider in order picking is the distance a picker walks during each order, as this directly correlates with the amount of worker hours a company must pay for [1],[3]. It is possible to cut the distance walked by configuring the layout of a warehouse, and by improving the organization of the warehouse. Configuring the layout of the facility is achieved by modifying the positions of the various conveyors, depots, and shelves. Carefully sorting the items and picker distribution based on item demand can help substantially increase the organization of the warehouse [3].

Problem Solution:

The goal of this project was to create a program that can accurately simulate the picking process in a warehouse. We wanted to find out which factors would have a greater impact on the picking times than others. We have identified several parameters within the capability of our simulation which can be adjusted in behavior space to impact the overall picking times for a set of orders in a specific warehouse layout. These parameters will be more fully explained later in this document.

To summarize, the problem solution was implemented in Java in two distinct applications. First, we developed a graphical layout program which we called the “Warehouse Layout Manager” using Java in the Eclipse Integrated Development Environment (IDE). This program was used to create the various warehouses for testing.

The simulation itself was also coded in Java and it utilized a framework called MadKit[9], a multi-agent platform based on the Agent-Group-Role (AGR) philosophy. We also up a SQL database (using HSQLDB) with tables of items and orders so that these could be kept constant from run-to-run. The output of the Warehouse Layout Manager was saved as a file that was read in by the simulator. The simulator can be run with any number of orders and a set of items stored in the warehouse. The other parameters that can be varied from run-to-run are the way the items are stored on the shelves, how they are sorted onto the shelves and how the items are picked for an order. Although we can simulate discrete picking we concentrated on zone picking for our results. However we could set whether or not the picker fetched more than one item at a time while in the shelves, or if he just picked up one item at a time and took it back to the tote. After the simulation was run we recorded the simulation steps for completing all the orders in the set and the other parameters that were set for the simulation run.

Warehouse Layout Manager:

The “Warehouse Layout Manager” is a program that is used to layout the various components of a warehouse. It was implemented using Swing, Java’s primary Graphical User Interface (GUI) toolkit, along with the Eclipse’s WindowBuilder plug-in. When starting a new warehouse layout it starts with a dialog box to input the warehouse dimensions in grid units (a

grid unit is considered to be a 2' x 2' area in the warehouse). Once the dimensions were entered the dialog box was replaced by a new window which contained a panel of three buttons as well as a grid space where various warehouse components are added. The components are the stop and start segments of a conveyor, conveyor segments (with a flow direction), item stops (special conveyor segments where a tote is ejected for picking), and shelf areas. The grids not filled with any of these objects were treated as open areas for worker movement. More specific instructions on using the Warehouse Layout Manager can be found in Appendix A. The save files utilized the unique T64 file extension which was named after our team number. The layouts were saved in zipped ArrayList objects utilizing the: java.io.FileOutputStream, java.io.ObjectOutputStream, and java.util.GZIPOutputStream. Saved files were read back into the layout manager for editing. Most importantly the saved files could also be parsed by the simulator to set up the simulation.

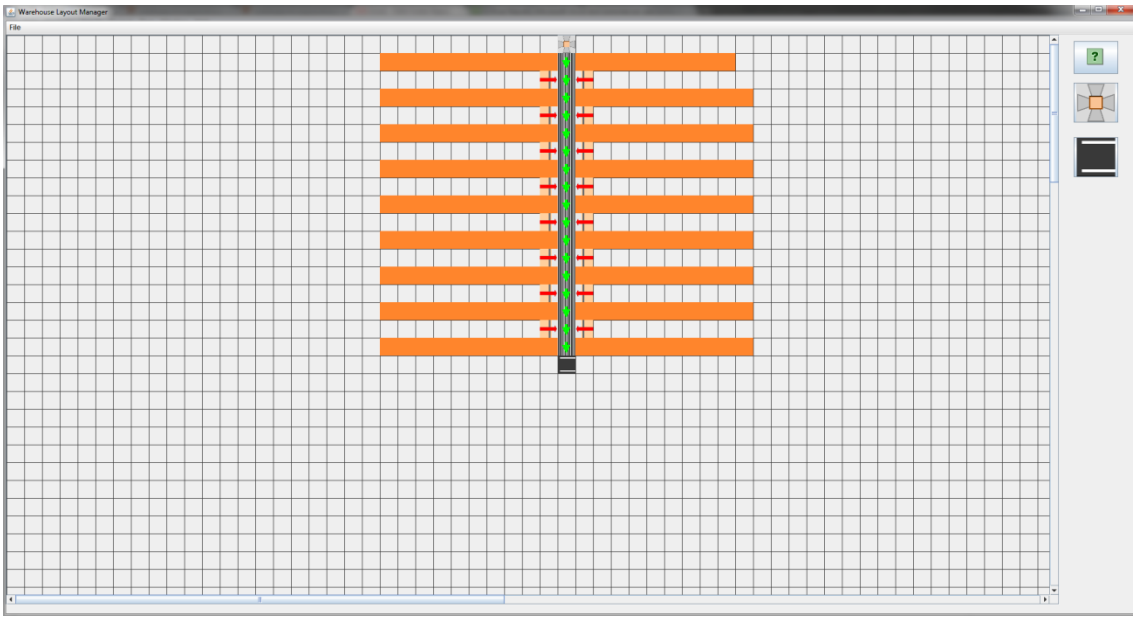


Figure 1 - This is the layout manager window displaying the layout of a demo warehouse. The orange areas are shelves and the graphics with the directional arrows are conveyor segments. Picking stations can be seen off the conveyor located near the shelves.

Database:

The SQL database had two purposes. The first was to contain pre-generated sets of items for the warehouse and orders with sets of items. The second purpose was to record results of the simulation runs.

The database has three table structures. The “item” table contains more than150,000 items, each with a unique id, and their generated popularity. A warehouse simulation used a subset of these items to stock the shelves. For example if a run was set up to use 100 items, the first 100 items were selected from the database. Likewise if a run was set up to use 500 items (for a larger warehouse), the first 500 items were selected to stock the shelves. The only other parameter stored in the items table (besides the item id) was the item popularity. Realistically in e-commerce the goal is to sell and ship items quickly so most of the items in the table were weighted to be more popular [2]. To calculate the popularity a random number was generated and weighted using the square root of the randomly generated number. This resulted in a curve shown below with the resulting popularity on the y-scale based on any randomly generated number (0-1) on the x-axis.

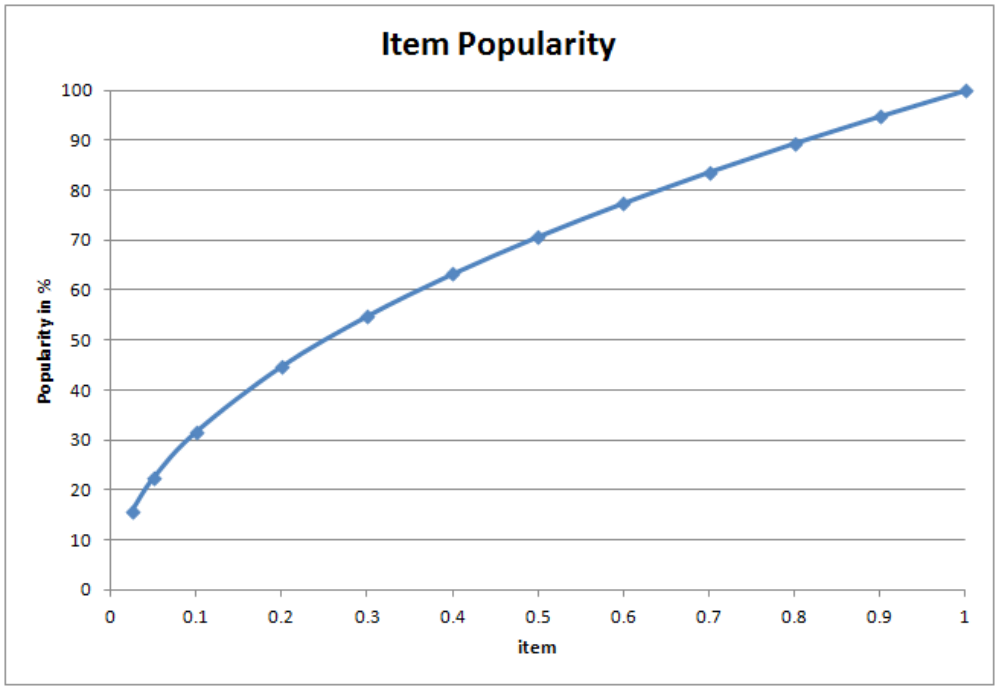


Figure 2 - The probability that an item is popular

The idea behind the popularity is that most orders will contain popular items, and the popularity can be used to stock the shelves in a more efficient manner so that the picking time is optimized. How the shelves were stocked with these items was based on simple or more complex algorithms in the simulator initialization. The most simple was to randomly assign bins to the items (assuming most were popular anyway) and the most complex was to carefully store items by popularity.

The “orders” table is actually several tables. The items in the orders had to be based on a pre-determined number of items in the warehouse so that the orders contained a subset of these items. In other words a warehouse with 100 items had orders that contained items 0-100, whereas a warehouse with 1000 items had more variety in the orders (could contain any items with ids 0-1000). So orders were pre-generated using a random number of items (0-12, with 6 being the most popular based on the average number of items in an on-line shopping cart [7].) However, we did account for outliers in our generated order, and as a result the range of orders in our database was between one and twelve. This is demonstrated in the graph below. The orders table had two columns: a unique order id and a list of items in the order. Orders were always selected from the database in the same way (same lookup) to provide consistency in the simulation runs. Any number of orders (up to 10,000) could be selected from any of the order tables. Order tables were set up for 100, 200, 300, 400, 500, 1000, 1200, 2000, 4000, and 10000 items in the warehouse.

The last table is the “results” table. Following a simulation run, a variety of information about each run is collected and placed in the database under a unique run id. A unique ID is also generated for a set of runs that are created from one behavior space run so all the results of this experiment can be compared with greater ease by simply finding all the runs that share this ID.

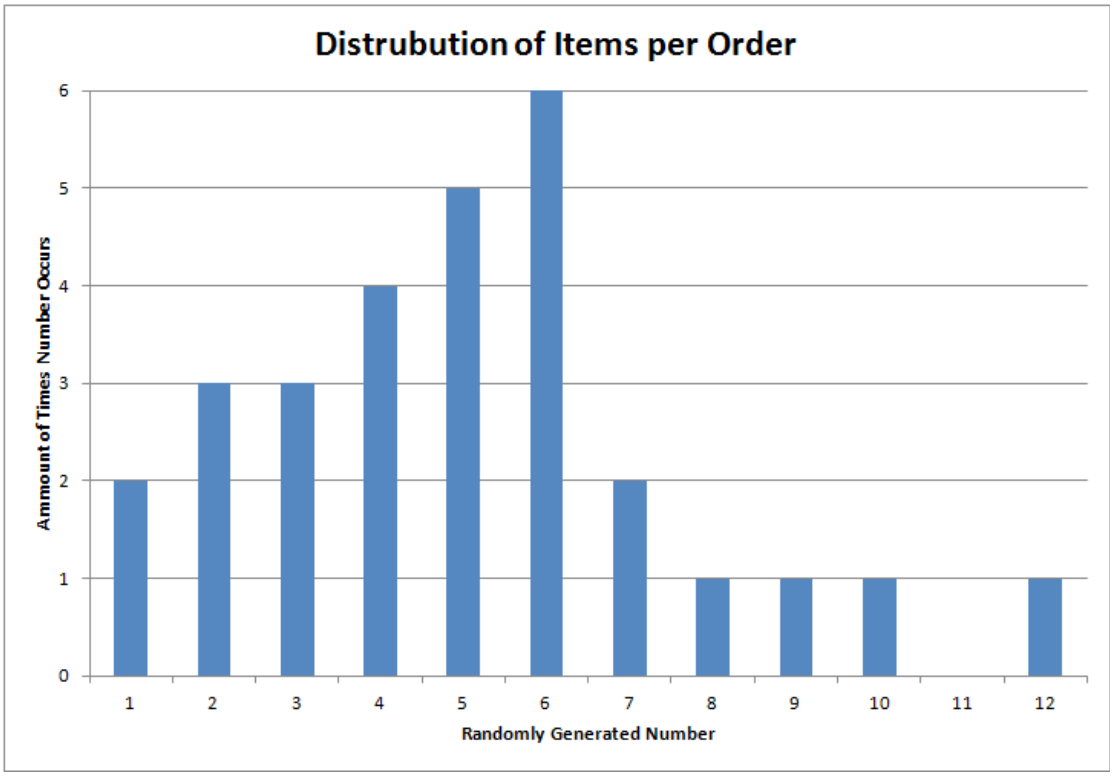


Figure 3 - This graph shows a range of up to 12 items, but most orders have around six.

Simulation:

The first step to set up a simulation is the creation of a warehouse in “Warehouse Layout Manager”. In this program the output will look similar to what is pictured below.

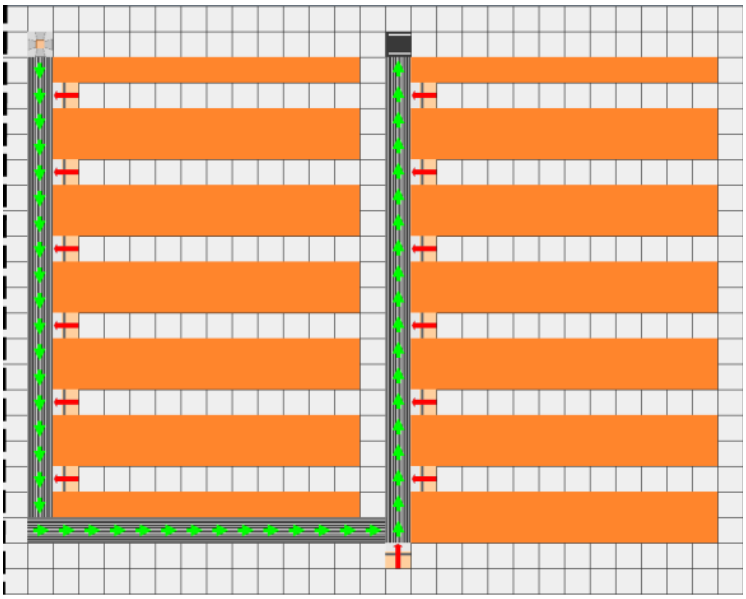


Figure 4 – How the warehouse is built in the “Warehouse Layout Manager”

This warehouse is saved as a binary file with the ".t64" extension. Once a file is created in this manner, it can be selected to be loaded into a simulation.

The simulator can be run with or without graphics, and also with command line parameters (useful for exploring behavior space) or with a panel to set specific simulation parameters. The screen shot below shows what a parameter panel looks like. The parameters that can be set include which warehouse layout to use, how many items should be stocked, which warehouse should be loaded, how many orders should be processed, and what the maximum number of orders to be processed should be in addition to several other important parameters. Once this dialog is closed the program queries the database for a list of orders the length of which corresponds to how many orders are supposed to be processed.

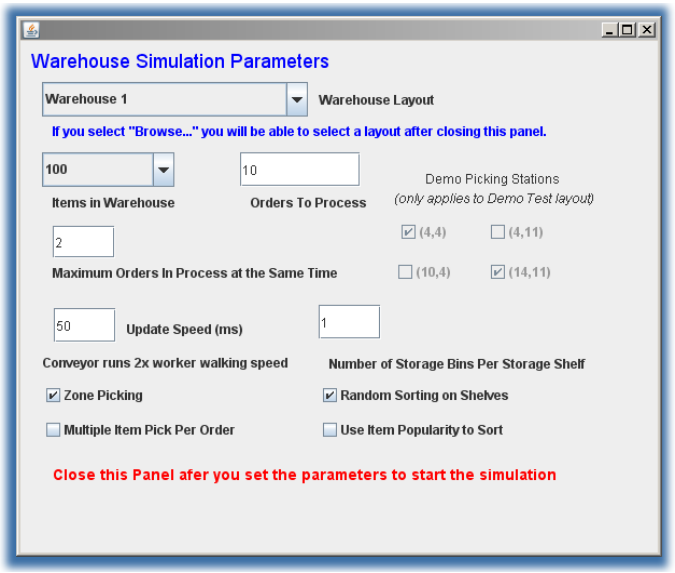


Figure 5 - The parameter panel for a single simulation run

It is also possible to run the program without these by providing the parameters directly to the simulator for the purpose of consecutive runs for an experiment, and this is the method that was used to find our results. A helper program we created called “Behavior Space” let us make many simulation runs by varying one parameter at a time in this manner. The orders table of the database is important because while we could generate random orders for each individual simulation, a database allows us to make a controlled experiment; no random parameters affect our results so we are sure they are consistent to the input we provided. Our simulator does not contain icons but is still a

direct representation of the file as it was created. Picking zones are calculated for the simulation, and that accounts for the re-coloration of the shelves and the addition of circles next to the picking stations in this screen shot (colors are randomly generated since we do not know ahead of time how many zones will be in the warehouse). A picker is represented by the circle (he moves around on the grid as he works) and the shelves that are part of the picker's “zone” or “domain” are in the same color as the picker.

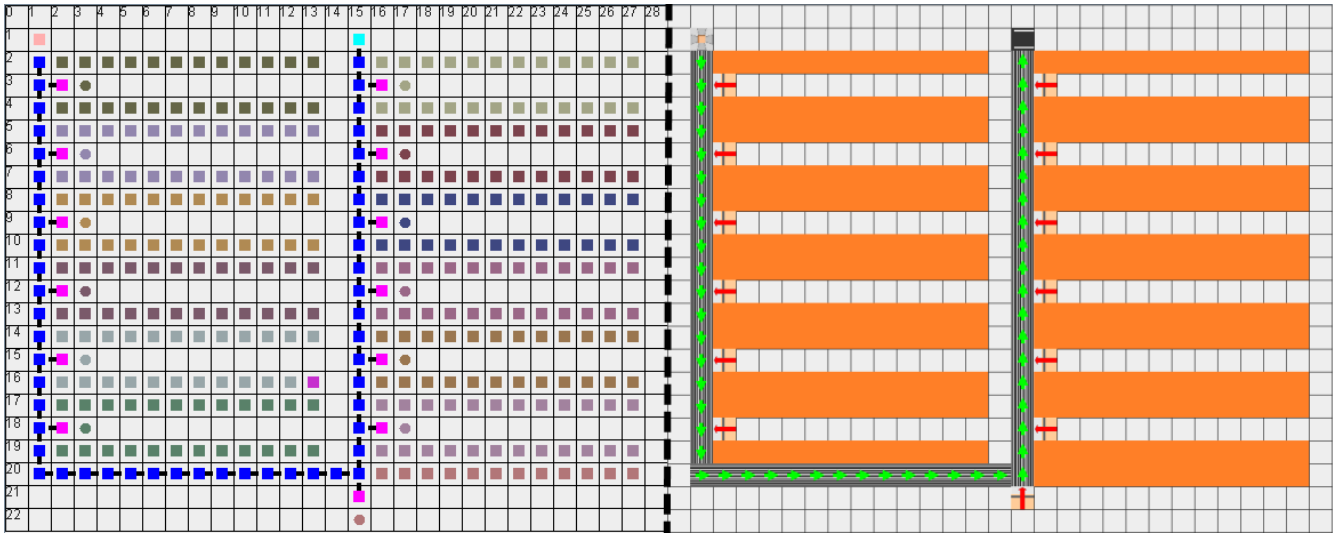


Figure 6 – The warehouse layout and resulting simulation screen

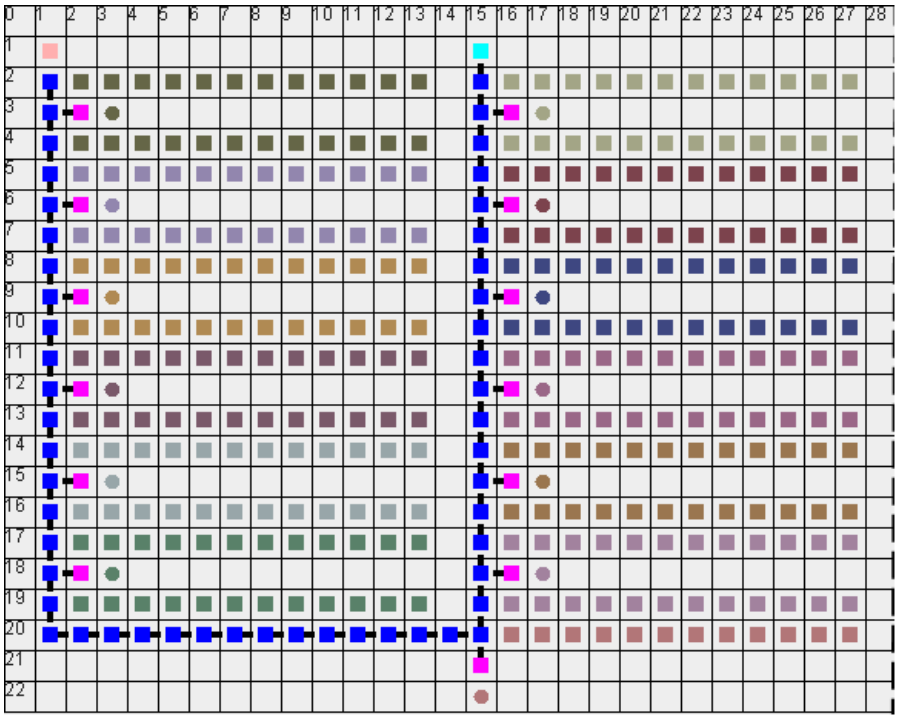
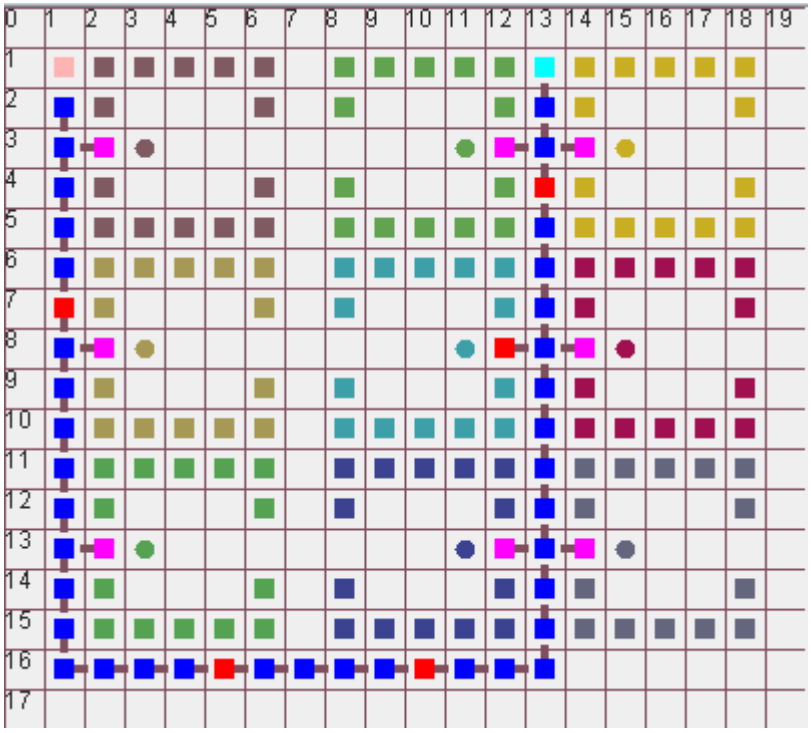


Figure 7 – Aisle Layout

Directly above is a screen shot of an example simulation setup called the “Aisle” warehouse. The conveyor that snakes through the facility is blue with a thick black line pointing in the direction it is facing and is broken into conveyor segments. The picking stations are pink with a thick black line pointing to the conveyor they are connected to. Next to each picking station a picker is generated and assigned to that station so that he will stay in that zone. Each station has the shelves with the closest walking distance added to its zone, so that when a tote that needs an item in those shelves is passing by it will be redirected so the picker can collect that item. Zones in this example are mostly aisles, so it is apparent what is closest to the station. The shelves in a zone and the picker that is assigned to that zone are colored in the same way so it is easier to see where they are. Zone colors are randomly generated. At the start of the conveyor there is always an origin, colored a pink salmon color, for the totes where they are placed onto the conveyor and at the end of the conveyor there is a place where the totes are removed from the system (the order is ready to be shipped), which is cyan. A tote on the conveyor paints that segment red.

Below is another layout we used called “Boxy” Layout.



The A* (pronounced "A star") path finding algorithm was core to our project in order to get the best paths for the purpose of the movement of pickers, calculation of walking distance for picking station zones, etc. The form of the algorithm we used was adapted to java by memoization.com (http://memoization.com/2008/11/30/a-star-algorithm-in-java/) [8] from the Maze example in *AI Application Programming* by M. Tim Jones, and was heavily modified to fit our needs. The general idea of an A* search algorithm is that it uses a heuristic to find the best next point from each point as it works out a path, where once each best point is selected it is added to the list of best path points. Once this path reaches the goal this list is saved as the path to be used. Specifically, our program generates a complete second overlay grid to be used for the path finding, where the points on the grid that are not accessible (conveyors, storage, etc.) are removed from this second grid because they cannot be part of the worker's path. Starting from the beginning point where the path needs to come from, the program looks at all adjacent points to find the one that is closest to where it needs to be, and weights it additionally based on how far the path thus far is from the start point. It then adds the point to be the "parent" of the point it just came from, and repeats the process. Once it reaches the end point the program follows the chain of points back to where it originated and forms a list of all the points, which is the result. In our program a router class is available to be

instantiated for use by any other portion of the program, and it is inside these routers that the path finding is performed. The path finding part of the simulation was by far the most time consuming part of the development. Despite the weeks spent finding the cause of of stack overflows, we are very satisfied with the resulting code and its functionality.

The Agent Model

At the heart of our simulation is the agent framework MadKit [9]. Part of the learning curve for our simulation was understanding how MadKit agents interact and communicate with each other as needed. Madkit is based on messaging between agents in the same community and groups. For our simulation all agents were in the "warehouse" community. The Warehouse agent itself belonged to several other groups so that it could be the central relay for other kinds of agents to interact. There was one Warehouse agent in the model, but there could be any number of conveyor segment agents (which communicate with each other to transfer totes) and any number of picker agents.

Item Stops (where the pickers put items into totes) were another type of agent. The role of the Item Stop was to tell a picker when it has a tote to fill and to put the tote back onto the conveyor system when it was filled with the items from its zone. Zones (called a domain in the code) are calculated based on the item bins closest to itself by walking distance (using A*). Therefore zones are actually a collection of item bins that are closest to the picking station.

End conveyor segments and Start segments (known as ToteSpawns) are subclasses of the normal conveyor segments and have different roles in the "conveyor segment" group. The ToteSpawn is responsible for initiating the order in the system. It puts a new tote on the conveyor system with an order number.

The simulation agents themselves are prompted to act by schedulers that operate once each tick (the unit of time in the simulation). Schedulers call specific methods in each of their agents each time tick. Some Agents act independently of timers, such as the central Warehouse agent. All agents can receive and send messages to either specific agents or to all agents of a certain group and role (as a

broadcast message). For example in our model the Item Stop can send a message only to its own picker to tell it to work on an order.

In the initialization stage of our simulation, after the layout is read in, all the agents are started and launched by the Warehouse. One Picker is launched for each ItemStop (and is assigned to the stop at this time). When agents are launched in the MadKit platform they call their “activate” method which sets the agent's community, group, and role. After this the agent begins to listen for messages and handles them in its “receiveMessage” method. Messages in our simulation are all subclasses of the generic MadKit Message class. The reason we made so many subclasses is that it is easy to determine what kind of message it is when received by the receiveMessage method. Schedulers are special agents that call methods in the Agent classes on a timer. The Timer scheduler is the main scheduler which calls the “Tic” method in the Warehouse to check the status of the order process.

The basic process of the simulation begins when the Warehouse sends the first “NewOrder” message to the ToteSpawn segment of the conveyor system. It responds by placing a tote with the order number on the next conveyor segment. The next conveyor segment for the ToteSpawn (and all conveyor segments) is set up during the initialization of the simulation (when the layout file is read).

If the system is set up to handle multiple orders at a time, more totes with orders are launched, up to the number allowed by the maximum allowed orders in process at one time (a simulation parameter).

The Tote continues along the conveyor until it reaches an "Item Stop" (picking station). If the order in the tote has an item in the Domain of this ItemStop then it is ejected onto the ItemStop segment. When the ItemStop receives the tote it examines the order and sends the Picker off to collect one or more items (depending on if the picker is operating in single-pick or multi-pick mode). The ItemStop monitors the tote (one of the methods called every time tick of the simulation) to see if it is full yet. If it is full it places it back on the conveyor system to head toward the End Segment (or toward another Item Stop for more items).

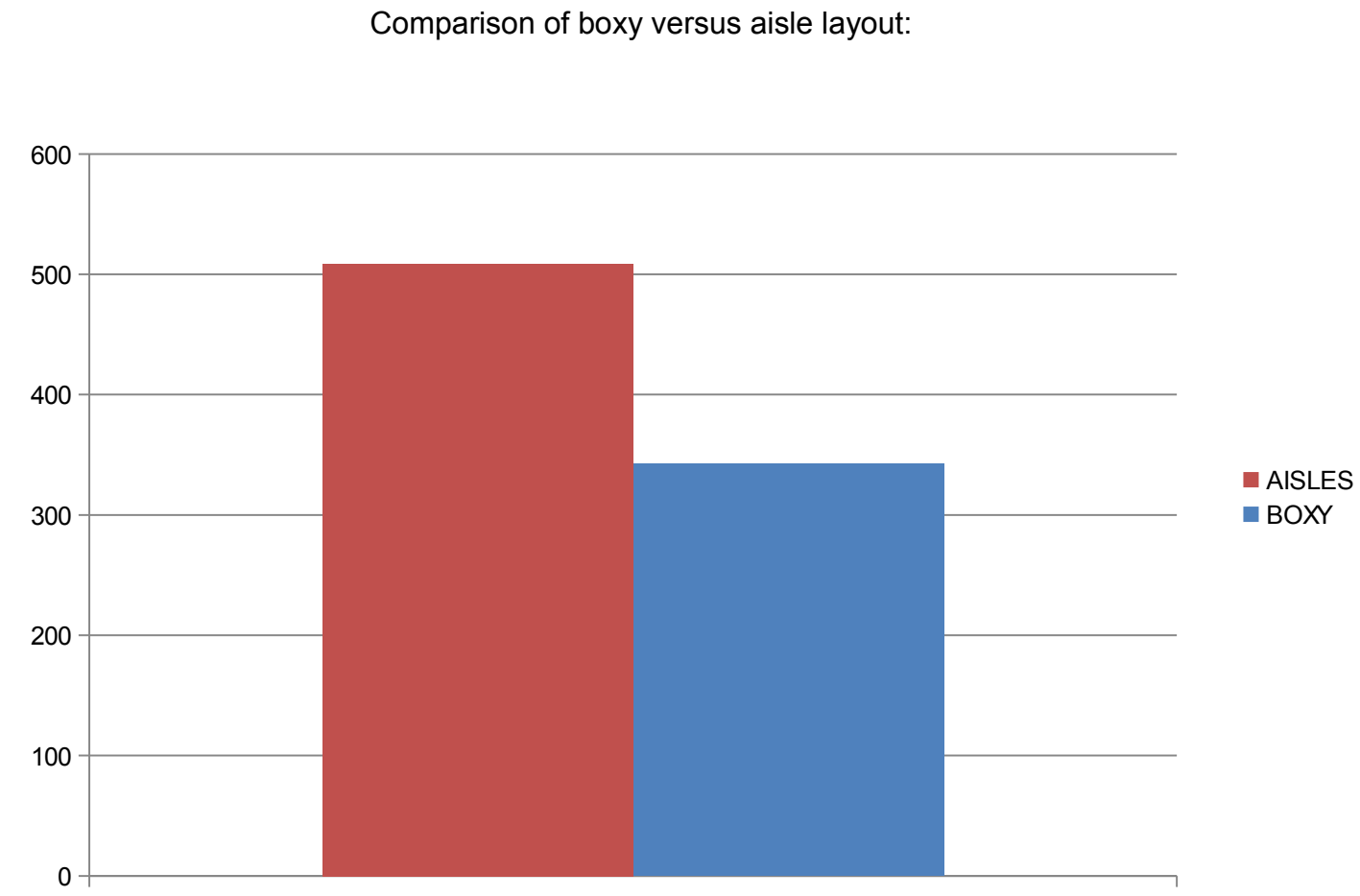
Sometimes a tote that needs to get off at an Item Stop cannot because the ItemStop already has a tote. So this Tote will go around the conveyor and become a partially filled

order. Partially filled order totes are placed back on the conveyor system by the ToteSpawn if the number of orders in process is less than the maximum allowed at one time.

The tote finally reaches the end of the conveyor (End segment). If the order is complete this order process is finished and the order is removed from the system (shipped). If the tote's order is not yet filled it is put back on the conveyor to go past the item stops again.

When the orders are all complete the database record is updated with the number of steps for the simulation and other simulation parameters.

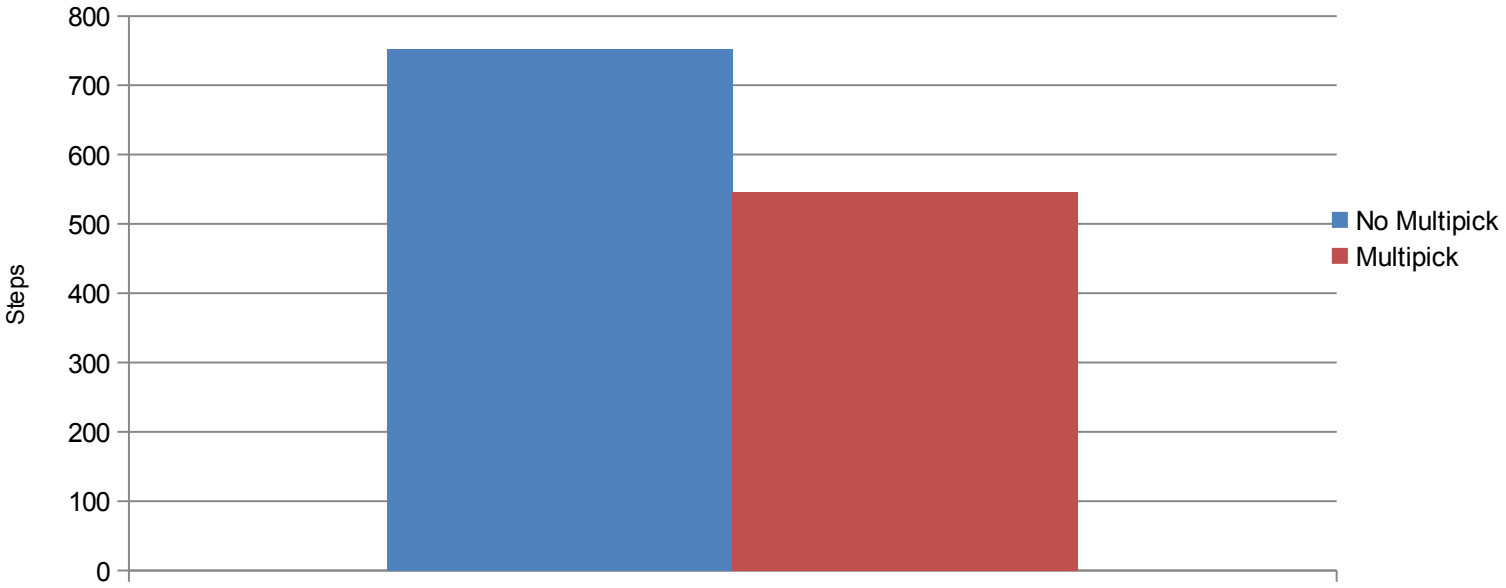
Results and Analysis:



Our simulations of the boxy warehouse ran faster than the aisle picking warehouse.

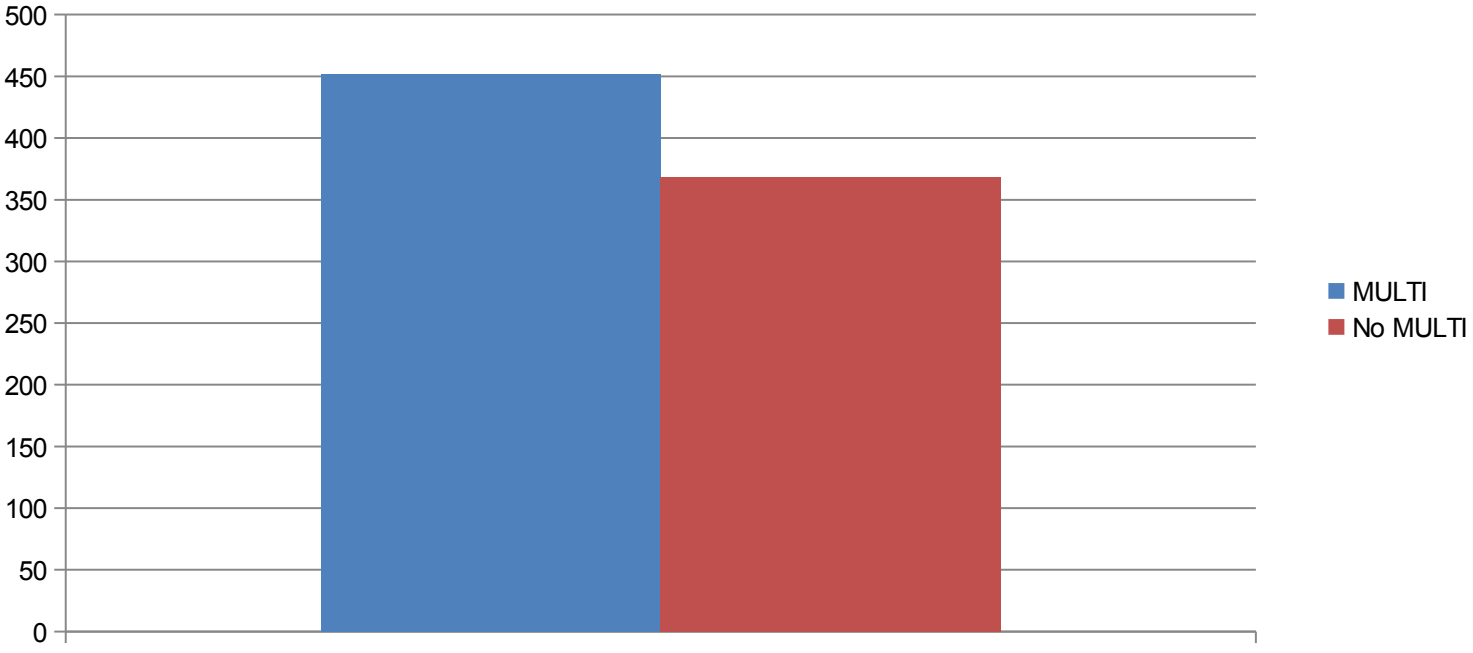
Comparisons of picking multiple items at a time versus picking one item at a time:

Aisles:



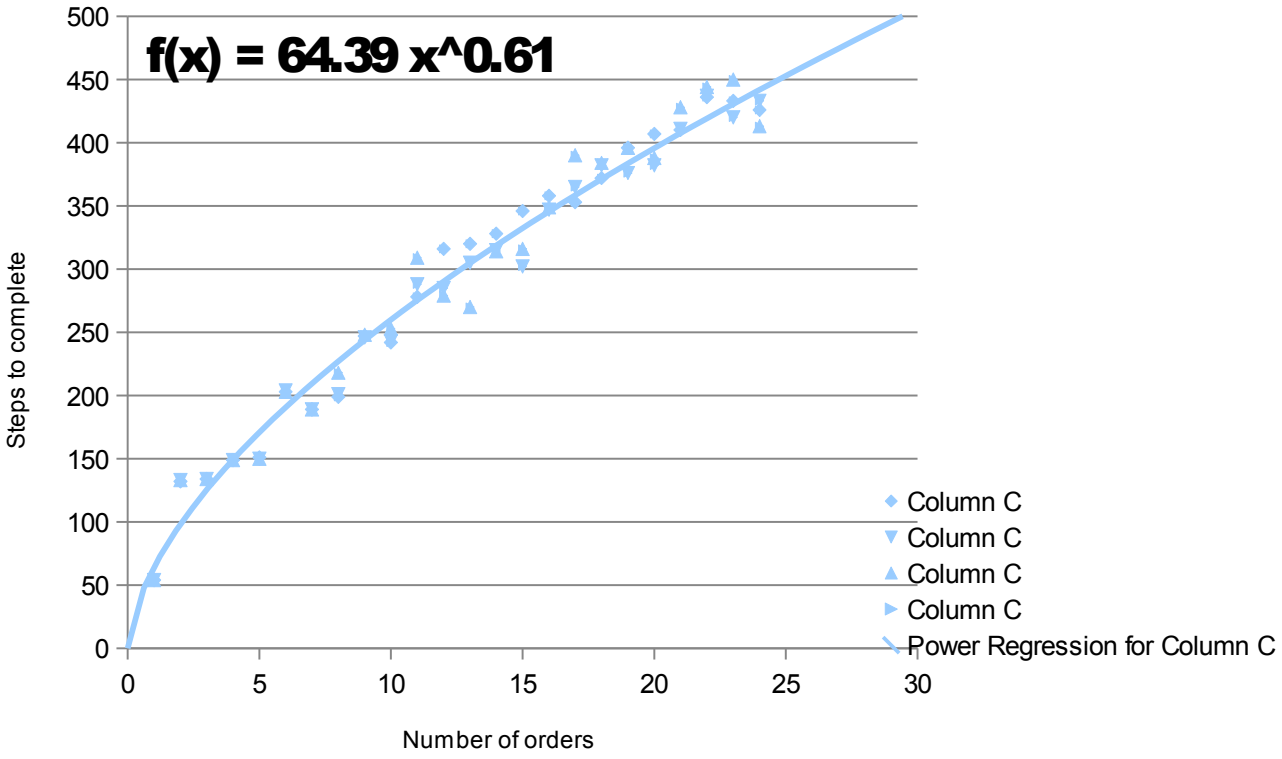
On the facility with pickers in aisle shaped zones there was a clear time saving in multipicking.

Boxy Layout:



There were fewer items in this run and so fewer total steps. There is a slightly smaller saving.

Number of orders:



This was an interesting set of runs comparing the number of orders to the amount of time that the run takes to complete. This graph was created based off of the data, and one can tell that the pick time does not increase linearly with the number of orders.

Conclusions:

From the data generated by our program we can draw some interesting conclusions. The boxy warehouse appears to be more efficient, although it is likely that this is because we reduced the number of items to 100 for both layouts and so the aisles were not full, leading to a loss of efficiency. Multi- picking is always faster than single picking, and this result makes sense because it cuts out countless unnecessary trips for the picker returning to his station in between picking up items. However, single-picking is less inefficient on a boxy layout. Again, this makes sense because the picker is never as far from the station as it is inside an aisle

where it must walk all the way back down the aisle between picks. Finally, we retrieved an interesting result when increasing the number of orders that are run through the database. The time it takes to pick for an increasing number of orders increases, of course, but the increase was not linear. At lower order quantities the increase vastly increased the time required for a complete pick, but as the order quantities increased each individual order came to make less of an impact.

The most important result of our project was the simulator itself. These results are only changing several parameters, but with slight modifications we could measure much more.

Significant Original Accomplishments:

Creating a working simulation of the picking process was a major task. It was made simpler by using the MadKit framework and implementing the Warehouse Layout Manger to generate warehouse layouts. MadKit worked great but took some planning and some trial and error to set up the agents and messages. Overall we highly recommend that any Java programmer building a simulation that involves agent interactions use MadKit because of its flexibility and adaptability to large agent-based projects.

The Warehouse Layout Manager was a significant accomplishment that helped us (and others interested in using our simulation) lay out a warehouse for testing. We think that adding this application was an excellent idea so that the warehouse configurations did not have to be hand coded for the simulation (although we used a simple “Demo Test” configuration for testing code that is hand coded and is part of the WarehouseLayout code).

Adapting the A* Maze route algorithm to path finding in the Warehouse was a straightforward idea, but took a lot of work to make it function correctly in our environment. This Route code was use extensively in the simulation.

Once all these parts were put together and the simulation was running it was beautiful to watch the pickers go to the shelves and grab items, add them to the order, watch the totes leave the picking stations and travel around the conveyors until the orders were all filled.

Acknowledgements:

First, our team would like to thank the New Mexico Supercomputing Challenge for making the project possible. Next, we would like to thank Elizabeth Cooper who was our mentor this year. She helped us with creating the simulation and proofreading the final report. Our team would also like to thank our teacher Lee Goodwin, who provided encouragement throughout the year. Many experts advised us on the development of our project and we would like to recognize their valuable input and assistance as well. John Snider answered questions we had related to the warehouse business, which he currently works in. His input was useful in cementing our grasp of the internal workings in a warehouse. We would also like to thank Jeff Grantham for giving us feedback on the interim report and those who evaluated our project at the Northern New Mexico Community College. We would like to thank Ms.Parkinson, the teacher that runs our school's writing lab, who helped grammar check our final report. Finally we thank Fabien Michel, a primary MadKit developer, for being an active supporter of our project.

Background on our Team and Experience:

Our team is composed of three freshman students from Los Alamos High School. Colin Redman has been participating in the supercomputing challenge since 2005. This is Sudeep Dasari's 5th year participating, and David Murphey's 2nd year. Experience from prior years was helpful in planning and implementing this project. Last year's project on sorting packages in a shipping facility was similar but implemented in Greenfoot (also a Java based agent system). One major difference between that project and this one was that there were less variable parameters that could be adjusted so the simulation was not as interesting.

This is the second MadKit project that Colin, our chief programmer, has produced for the New Mexico Supercomputing Challenge. The first one was three years ago, simulating a closed environment in a long voyage spacecraft. This used a previous version of MadKit (version 4.2) and although a lot of thought went into the simulation it was not very complete (too many parameters to consider). This year MadKit made a lot more sense and we used an newer updated version (version 5.0) which was easier to integrate with the simulation code since it was not designed as an IDE, but as a library.

Sudeep worked with this team last year and this year was the secondary programmer, but chief developer of the Warehouse Layout Manager. David was also with the team last year (his first year) and contributed by drawing graphics for both programs and for the illustrations in the final report and evaluation presentation. Everyone contributed to the final report but Sudeep and David did most of the work while Colin was finishing the simulation code and producing results.

Citations:

1. Eisenstein, Donald D. *ANALYSIS AND OPTIMAL DESIGN OF DISCRETE ORDER PICKING TECHNOLOGIES ALONG A LINE*. Rep. Graduate School of Business, The University of Chicago, 27 Dec. 2006. Web. 7 Mar. 2012. <http://faculty.chicagobooth.edu/donald.eisenstein/research/pick12.pdf>.

2. Ricker, Fred R., and Ravi Kalakota. *Order Fulfillment: The Hidden Key to E-Commerce Success*. Rep. Logistech, 1999. Web. 3 Mar. 2012. <http://www.logistech.us/lsi/resources/SCM9911ecomm.pdf>.

3. Koster, Rene De, The Le-Duc, and Kees Jan Roodbergen. *Design and Control of Warehouse Order Picking: A Literature Review*. Rep. Rotterdam, The Netherlands: Erasmus University, 2006. *Design and Control of Warehouse Order Picking: A Literature Review*. ScienceDirect, 25 Oct. 2006. Web. 26 Mar. 2012.

4. Vrysagotis, Vassilios, and Patapios Alexios Kontis. *Warehouse Layout Problems : Types of Problems and Solution Algorithms*. Rep. International Scientific Press, 31 Aug. 2011. Web. 13 Mar. 2012.

5. Bartholdi, John J., and Steven T. Hackman. *WAREHOUSE & DISTRIBUTION SCIENCE Release 0.95*. Rep. The Supply Chain and Logistics Insitute, 21 Aug. 2011. Web. 26 Mar. 2012.

6. "Maximize Pick Productivity." *The Avery Way. Warehouse Consultant Gets Results Fast!* Avery Way. Web. 14 Mar. 2012. <http://www.elogistics101.com/Article/MaxPickProd.htm>.

7. "Good News! The Average On-line Order Value Was up More than 10 Percent from Last Year." *Good News! The Average On-line Order Value Was up More than 10*

Percent from Last Year. Pinnacle Cart. Web. 19 Mar. 2012. <http://blog.pinnaclecart.com/2010/11/22/good-news-the-average-on-line-order-value-was-up-more-than-10-percent-from-last-year/>.

8. Admin. "A-Star Algorithm in Java." *A-Star Algorithm in Java*. Memoization, 30 Nov. 2008. Web. 28 Mar. 2012. <http://memoization.com/2008/11/30/a-star-algorithm-in-java/>.

9. MadKit http://www.madkit.org/. A Multi-Agent Development Kit by Olivier Gutknecht, Jacques Ferber, and Fabien Michel, LIRMM Montpellier France

Appendix A. Warehouse Layout Manager User Guide

A warehouse can be laid out using the following keyboard commands, mouse commands, and the buttons located on the panel next to the grid. The first button on the panel was a help button that brought up a window which displayed a help dialog coded in HTML. The second button is used to lay warehouse start points, and the third button is used to lay warehouse end points. A segment is added to the grid by clicking the button then clicking on an open spot on the grid. The arrow keys are used to lay conveyor segments in such a way that the up arrow key laid an upward facing conveyor segment, etc. To lay a conveyor segment mouse-over an open grid space, and press the corresponding arrow key. Finally, the number keys; one, two, three, and four, are used to lay item stops for pickers in a similar fashion as the conveyor segments are laid. To add an area of shelves drag the mouse across the grid.

The standard control-o and control-s commands can be used to open and save the warehouse layouts. The layouts are saved using a code that conformed to the arrangement; x-coordinate in terms of the grid, y-coordinate in terms of the grid, and the type of block in the grid. For example, the code 0010.0076.shelf, would refer to a shelf block created in a grid space, 10 to the right of the upper-right hand corner of the grid, and 76 down. The actual Java method used to create the code is located in **Appendix B**.

Appendix B. Warehouse Layout Code

The essential part of the layout code is the file format for storing the layout. The method below shows how this was done by reading one by one the components in the grid and sending them as binary objects to the file output stream.

GUI.java

```
import java.awt.Dimension;
import java.awt.EventQueue;

import javax.swing.JFrame;



public class GUI {
	/** Warehouse grid dimensions */
	public static Dimension size = new Dimension();
	public static int gridSize=30;

	public static void main(String[] args) {
		if (args.length==0){
			//no arguments, so need to run dialog
			size.width = 0;
			size.height = 0;
		} else {
			if (args.length<2) {
				System.out.println("Parameters must be width and height");
				System.exit(44);
			} else {
				size.width = Integer.parseInt(args[0]);
				size.height = Integer.parseInt(args[1]);;
			}
		}
		EventQueue.invokeLater(new Runnable(){
			public void run() {
				if (size.width <=0 || size.height<=0) {
					Dialog d=new Dialog();
					d.setVisible(true);
```

```
                    d.setSize(300, 300);
                } else {
                    StartFrame frame= new StartFrame();
                    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                }
            }});

    }


}
```

Dialog.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextArea;



public class Dialog extends JDialog {
	private StartFrame sf;
	public int x;
	public int y;
	public Dialog(){
		//x=0;
		//y=0;
		if (GUI.size.width>0 && GUI.size.height>0) { //dimensions already set
			setVisible(false);
			StartFrame frame= new StartFrame();
			frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		}
		requestFocusInWindow();
		JLabel label=new JLabel("Enter Side Dimension of Warehouse");
		label.setBounds(0, 0, 221, 68);
		JLabel labeal=new JLabel("");
		labeal.setBounds(221, 0, 221, 68);
		JLabel label1=new JLabel("Enter Width");
		label1.setBounds(0, 68, 117, 68);
		final JTextArea xarea=new JTextArea();
		xarea.setBounds(129, 49, 221, 68);
		JLabel label2=new JLabel("Enter Height");
		label2.setBounds(0, 136, 221, 68);
		final JTextArea yarea=new JTextArea();
		yarea.setBounds(129, 128, 221, 68);
		JButton button=new JButton("Ok");
		button.setBounds(0, 204, 221, 68);
		button.addActionListener(
```

```
                new ActionListener(){

                    public void actionPerformed(ActionEvent e) {
                        int cx=Integer.parseInt(xarea.getText());
                        int cy=Integer.parseInt(yarea.getText());
                        GUI.size.width=cx;
                        GUI.size.height=cy;
                        setVisible(false);
                        StartFrame frame= new StartFrame();
                        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    }
                    });
        getContentPane().setLayout(null);

        getContentPane().add(label);
        getContentPane().add(labeal);
        getContentPane().add(label1);
        getContentPane().add(xarea);
        getContentPane().add(label2);
        getContentPane().add(yarea);
        getContentPane().add(button);


    }


}
```

CButtons.java

```
import javax.swing.*;
import java.awt.event.*;


@SuppressWarnings("serial")
public class CButtons extends JButton {
	StartFrame mouseState;

	/**
	 * PathFinder
	 */
	public CButtons(final String icon, StartFrame MouseState){
		mouseState = MouseState;
		setIcon(new ImageIcon(icon));
		addActionListener(
				new ActionListener(){

					public void actionPerformed(ActionEvent e) {
						try{
							mouseState.setMouseState(icon);
						}

						catch(NullPointerException e1){
							System.out.println("oops");
						}
					}
					});



	}
}


/*
 * "BoxStart.png", "ConveyerStraight.png", "ConveyerLeft.png", "ConveyerRight.png", "ConveyerBack.png",
 			"ConveyerIntersection.png", "BoxEnd.png"
 */
```

MouseComponent.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.awt.event.MouseAdapter;
import java.awt.geom.*;

import javax.swing.*;

@SuppressWarnings("serial")
public class MouseComponent extends JComponent  implements Scrollable{

	public ArrayList<FacilityRectangle> BLOCKS;
	private FacilityRectangle BLOCK;
	public int Level = 1;
	private String code;
	private int x;
	private int y;
	private Dialog d;
	public MouseComponent(int z, int k) {
		x=z;
		y=k;

		setSize(z, k);
		BLOCKS = new ArrayList<FacilityRectangle>(200);
		addMouseListener(new MouseHandler());
		addMouseMotionListener(new MouseMotionHandler());
		addKeyListener(new KeyHandler());
	}

	String state = null;

	public void setState(String state){
		this.state = state;
	}

	public void setLevel(int i) {
		Level = i;
		repaint();
	}
	public int getLevel(){
		return Level;
	}
```

```
public void paintComponent(Graphics g){
	Graphics2D g2=(Graphics2D)g;
	double x1=0;
	double y1=0;
	while(x1<x){

		g2.draw(new Line2D.Double(x1, 0, x1, GUI.gridSize*GUI.size.width));
		x1=x1+GUI.gridSize;
	}
	while(y1<y){

		g2.draw(new Line2D.Double(0, y1, GUI.gridSize*GUI.size.height, y1));
		y1=y1+GUI.gridSize;
	}
	if(Level==1){
		for (FacilityRectangle f: BLOCKS){
			g2.setColor(Color.RED);
			//g2.drawImage (f.icon.getImage(), 0, 0, getWidth (), getHeight (), null);
			g2.drawImage (f.icon.getImage(), (int)f.rect.getX(),  (int)f.rect.getY(),GUI.gridSize, GUI.gridSize, null);

			//f.icon.paintIcon(this,g2,(int)f.rect.getMinX(),(int)f.rect.getMinY());
		}
	}

	}



public FacilityRectangle select(Point2D p) {
	if (Level == 1){
		for (FacilityRectangle f : BLOCKS) {
			if (f.rect.contains(p))
				return f;
		}
	}
	return null;

}

public void add() {
	Point2D p=this.getMousePosition();
	BLOCK = select(p);
```

```
        if (BLOCK == null){
        double X = p.getX();
        double Y = p.getY();

        double x = X % GUI.gridSize;
        double y = Y % GUI.gridSize;

        if (x != 0 || y != 0) {
                X = X - x;
                Y = Y - y;
        }

        code=makeCode(X, Y, state);
        BLOCK = new FacilityRectangle(X, Y, state, code);
        state = null;
        if (Level == 1) {
                BLOCKS.add(BLOCK);
        }

        repaint();
        }
}
public void add(String s) {
        Point2D p=this.getMousePosition();
        BLOCK = select(p);
        if (BLOCK == null){
        double X = p.getX();
        double Y = p.getY();

        double x = X % GUI.gridSize;
        double y = Y % GUI.gridSize;

        if (x != 0 || y != 0) {
                X = X - x;
                Y = Y - y;
        }
        try{
        code=makeCode(X, Y, s);
        BLOCK = new FacilityRectangle(X, Y, s, code);
        }catch(NullPointerException e){
                System.err.println("No state specified");
        }
        if (Level == 1) {
                BLOCKS.add(BLOCK);
```

```
        }

        repaint();
        }
}

public void remove(FacilityRectangle b) {
        if (b == null)
                return;
        if (Level == 1) {
                BLOCKS.remove(b);
        }

        repaint();
}

public String makeCode(double cx, double cy, String mouseComponent){
        String mecode = null;
        String xcode=null;
        String ycode=null;
        int x=(int)cx/GUI.gridSize;
        int y=(int)cy/GUI.gridSize;

        if(mouseComponent.equals("images/BoxStart.png")){
                if(x<10){
                        xcode="000"+Integer.toString(x);
                }
                else if(x<100){
                        xcode="00"+Integer.toString(x);
                }
                else if(x<1000){
                        xcode="0"+Integer.toString(x);
                }
                else{
                        xcode=Integer.toString(x);
                }

                if(y<10){
                        ycode="000"+Integer.toString(y);
                }
                else if(y<100){
                        ycode="00"+Integer.toString(y);
```

```
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"start";

    }
    else if(mouseComponent.equals("images/BoxEnd.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"end00";
    }
    else if(mouseComponent.equals("images/ConveyerStraight.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
```

```
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"cn090";
}
else if(mouseComponent.equals("images/ConveyerLeft.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
```

```
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"cn180";
    }
    else if(mouseComponent.equals("images/ConveyerRight.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"cn360";
    }
    else if(mouseComponent.equals("images/ConveyerBack.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
```

```
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"cn270";
}
else if(mouseComponent.equals("images/lift.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
```

```
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"lift0";
    }
    else if(mouseComponent.equals("images/itemdrop-down.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"iddwn";
    }
    else if(mouseComponent.equals("images/itemdrop-left.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
```

```
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"idlft";
}
else if(mouseComponent.equals("images/itemdrop-right.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
```

```
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"idrgt";
    }
    else if(mouseComponent.equals("images/itemdrop-up.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"idup0";
    }
    else if(mouseComponent.equals("images/Shelf.png")){
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
```

```
        }

        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"shelf";
}
else{
        if(x<10){
            xcode="000"+Integer.toString(x);
        }
        else if(x<100){
            xcode="00"+Integer.toString(x);
        }
        else if(x<1000){
            xcode="0"+Integer.toString(x);
        }
        else{
            xcode=Integer.toString(x);
        }
        if(y<10){
            ycode="000"+Integer.toString(y);
        }
        else if(y<100){
            ycode="00"+Integer.toString(y);
        }
        else if(y<1000){
            ycode="0"+Integer.toString(y);
        }
        else{
            ycode=Integer.toString(y);
        }
        mecode=xcode+"."+ycode+"."+"inter";
```

```
            System.out.println("moo");
        }

        return mecode;
    }


    /*
     * First Class defines what clicking mouse does Next Class defines what
     * dragging does Classes are seperated by space
     */

    private class MouseHandler extends MouseAdapter {
        public void mousePressed(MouseEvent event) {
            setVisible(true);
            requestFocusInWindow();



        }

        public void mouseClicked(MouseEvent event) {
            if(event.getClickCount()==1){
                try{
                add();
                setVisible(true);
                requestFocusInWindow();
                }
                catch(Exception e){
                    System.out.println(e);
                }
            }
            BLOCK = select(event.getPoint());
            if (BLOCK != null && event.getClickCount() >= 2) {
                remove(BLOCK);
            }
        }
    }

    private class MouseMotionHandler implements MouseMotionListener {

        public void mouseDragged(MouseEvent event) {
            add("images/Shelf.png");
        }
```

```
        public void mouseMoved(MouseEvent event) {

        }
}
private class KeyHandler implements KeyListener{

        public void keyTyped(KeyEvent arg0) {

        }

        public void keyReleased(KeyEvent arg0) {
        }

        public void keyPressed(KeyEvent agr0) {


                if(agr0.getKeyCode()==KeyEvent.VK_LEFT){
                        state="images/ConveyerRight.png";
                        add();

                }
                if(agr0.getKeyCode()==KeyEvent.VK_RIGHT){
                        state="images/ConveyerLeft.png";
                        add();
                }
                if(agr0.getKeyCode()==KeyEvent.VK_DOWN){
                        state="images/ConveyerBack.png";
                        add();
                }

                if(agr0.getKeyCode()==KeyEvent.VK_UP){
                        state="images/ConveyerStraight.png";
                        add();
                }

                if(agr0.getKeyCode()==KeyEvent.VK_4){
                        state="images/itemdrop-right.png";
                        add();

                }
                if(agr0.getKeyCode()==KeyEvent.VK_3){
```

```
                    state="images/itemdrop-left.png";
                    add();
                }
                if(agr0.getKeyCode()==KeyEvent.VK_2){
                    state="images/itemdrop-down.png";
                    add();
                }

                if(agr0.getKeyCode()==KeyEvent.VK_1){
                    state="images/itemdrop-up.png";
                    add();

            }

        }
}
    public Dimension getPreferredSize(){
        Dimension d= new Dimension(x, y);
        return d;

    }
    public Dimension getPreferredScrollableViewportSize() {
        Dimension d= new Dimension(x, y);
        return d;
    }


    public int getScrollableBlockIncrement(Rectangle visibleRect,
            int orientation, int direction) {
        return GUI.gridSize;
    }


    public boolean getScrollableTracksViewportHeight() {
        return false;
    }


    public boolean getScrollableTracksViewportWidth() {
        return false;
    }


    public int getScrollableUnitIncrement(Rectangle visibleRect,
```

```
                int orientation, int direction) {
            return GUI.gridSize;
        }

}
```

HelpPanel.java

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.net.URL;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JEditorPane;
import javax.swing.JScrollPane;


@SuppressWarnings("serial")
public class HelpPanel extends JDialog {
	public HelpPanel() {
		setTitle("Warehouse Builder Help");
		setSize(700, 600);
		getContentPane().setLayout(null);
		JEditorPane htmlPane = new JEditorPane();
		htmlPane.setEditable(false);
		htmlPane.setContentType("text/html");
		URL htmlURL = getClass().getResource("help.html");
	    try {
		htmlPane.setPage(htmlURL);
	    } catch (IOException e) {
		e.printStackTrace();
	    }
		JScrollPane scrollPane = new JScrollPane(htmlPane,
				JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
				JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED );
		scrollPane.setBounds(0, 0, 629, 472);
		getContentPane().add(scrollPane);
		//scrollPane.setViewportView(htmlPane);

		JButton close = new JButton("Close");
		close.setBounds(478, 483, 127, 38);
		close.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent event) {
		   dispose();
		}
	    });
		getContentPane().add(close);
```

```
            setModalityType(ModalityType.APPLICATION_MODAL);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }

}
```

StartFrame.java

```
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.KeyStroke;

class StartFrame extends JFrame{
	private int compx;
	private String cfile;
	private int compy;
	//private Dialog d;
	private ArrayList<String> s1;
	private JFileChooser chooser;
	private MouseComponent comp;
	private CButtons Mbutton;
	private JPanel ChooserPanel;
	private  HelpPanel helpPanel;
	private boolean saveAsAccelerator;
	@SuppressWarnings("unused")
	private String mouseState = "images/Background.png";

	public void setMouseState(String state){
```

```
        mouseState = state;
        comp.state = state;
}


public StartFrame(){
        int maxWidth=1778;
        int maxHeight=1109;
        setVisible(true);
        cfile=null;
    s1= new ArrayList<String>(200);
        setTitle("Warehouse Layout Manager");

        Toolkit toolkit = java.awt.Toolkit.getDefaultToolkit ();
        Dimension screensize = toolkit.getScreenSize();
        setSize((int)(0.75*screensize.width), (int)(0.75*screensize.height)-43);
        if (GUI.size.width*GUI.gridSize > 0.75*screensize.width ) {
                setSize((int)(screensize.width), (int)(screensize.height)-43);
        }

        chooser=new JFileChooser();
        chooser.setCurrentDirectory(new File("Saves"));


        JMenuBar menubar= new JMenuBar();
        setJMenuBar(menubar);

        JMenu Filemenu = new JMenu("File");
        menubar.add(Filemenu);




        final JMenuItem save= new JMenuItem("Save");

        if(saveAsAccelerator==false){
        save.setAccelerator(KeyStroke.getKeyStroke("ctrl S"));
        }
        save.setEnabled(false);
        save.addActionListener(
                        new ActionListener(){
```

```
                public void actionPerformed(ActionEvent arg0) {
                    if(cfile==null){

                    }
                   try {
                                for(FacilityRectangle r: comp.BLOCKS){
                                    s1.add(r.code);
                                }



                     FileOutputStream fos = new FileOutputStream(cfile);
                     GZIPOutputStream gzos = new GZIPOutputStream(fos);
                     ObjectOutputStream out = new ObjectOutputStream(gzos);
                     out.writeObject(s1);
                     out.writeObject(GUI.size.width*GUI.gridSize);
                     out.writeObject(GUI.size.height*GUI.gridSize);
                     out.flush();
                     out.close();
                     JOptionPane.showMessageDialog(save,
                                "Done Saving.");
                   }
                    catch (IOException e) {
                      System.out.println(e);
                   }
                }});
final JMenuItem newButton= new JMenuItem("New");
Filemenu.add(newButton);
newButton.setAccelerator(KeyStroke.getKeyStroke("ctrl N"));
newButton.addActionListener(
            new ActionListener(){

                public void actionPerformed(ActionEvent event) {
                    save.setEnabled(false);
                    saveAsAccelerator=true;
                int ok=JOptionPane.showConfirmDialog(
                            newButton, "Are you sure you want to make a new file. Please save first.",
                            "Confirm", JOptionPane.YES_NO_OPTION,
                            JOptionPane.YES_NO_OPTION);

                if(ok==JOptionPane.YES_OPTION){
                    comp.BLOCKS.clear();
                    cfile=null;
```

```
                        comp.repaint();
                    }


                        }
                    }

                );
            final JMenuItem open= new JMenuItem("Open");
            Filemenu.add(open);
            open.setAccelerator(KeyStroke.getKeyStroke("ctrl O"));
            open.addActionListener(
                    new ActionListener(){

                        public void actionPerformed(ActionEvent event) {
                            chooser.showOpenDialog(null);
                            String filename=chooser.getSelectedFile().getPath();
                            cfile=null;
                            save.setEnabled(false);
                            saveAsAccelerator=true;
                             try {
                                 String state=null;
                                 String sx=null;
                                 String sy=null;
                                FileInputStream fis = new FileInputStream(filename);
                                GZIPInputStream gzis = new GZIPInputStream(fis);
                                ObjectInputStream in = new ObjectInputStream(gzis);
                                s1 = extracted(in);
                                int cx=(Integer) in.readObject();
                                int cy=(Integer) in.readObject();
                                in.close();
                                    for(String s: s1){
                                        sx=s.substring(0, 4);
                                        sy=s.substring(5, 9);
                                        if(s.substring(10,15).equals("start")) state="images/BoxStart.png";
                                        else if(s.substring(10, 15).equals("end00"))state="images/BoxEnd.png";
                                        else if(s.substring(10,
15).equals("cn090"))state="images/ConveyerStraight.png";
                                        else if(s.substring(10, 15).equals("cn180"))state="images/ConveyerLeft.png";
                                        else if(s.substring(10, 15).equals("cn270"))state="images/ConveyerBack.png";
                                        else if(s.substring(10, 15).equals("cn360"))state="images/ConveyerRight.png";
                                        else if(s.substring(10, 15).equals("cn360"))state="images/ConveyerRight.png";
```

```
                                        else if(s.substring(10, 15).equals("lift0"))state="images/lift.png";
                                        else if(s.substring(10,
15).equals("cn360"))state="images/ConveyerRight.png";
                                        else if(s.substring(10, 15).equals("iddwn"))state="images/itemdrop-
down.png";
                                        else if(s.substring(10, 15).equals("idlft"))state="images/itemdrop-
left.png";
                                        else if(s.substring(10, 15).equals("idrgt"))state="images/itemdrop-
right.png";
                                        else if(s.substring(10, 15).equals("idup0"))state="images/itemdrop-
up.png";
                                        else if(s.substring(10, 15).equals("shelf"))state="images/Shelf.png";
                                        else state="images/ConveyerIntersection.png";
                                        comp.BLOCKS.add(new FacilityRectangle(Double.parseDouble(sx)*GU
I.gridSize, Double.parseDouble(sy)*GUI.gridSize,state, s));
                                    }

                                    JOptionPane.showMessageDialog(open,"Done Opening.");
                                    comp.repaint();

                                }
                                catch (Exception e) {
                                    System.out.println(e);
                                }
                            }

                        @SuppressWarnings("unchecked")
                        private ArrayList<String> extracted(
                                ObjectInputStream in) throws IOException,
                                ClassNotFoundException {
                            return (ArrayList<String>)in.readObject();
                        }
                            }


                    );




            final JMenuItem saveAs= new JMenuItem("Save As");
            Filemenu.add(saveAs);
```

```
saveAsAccelerator=true;
if(saveAsAccelerator){
	saveAs.setAccelerator(KeyStroke.getKeyStroke("ctrl S"));
}
saveAs.addActionListener(
		new ActionListener(){

			public void actionPerformed(ActionEvent arg0) {
				if(cfile==null){
				chooser.showSaveDialog(null);
				String filename=chooser.getSelectedFile().getPath()+".T64";
				cfile=filename;
				saveAsAccelerator=false;
				save.setEnabled(true);
				}
			   try {
					for(FacilityRectangle r: comp.BLOCKS){
						s1.add(r.code);
					}



				FileOutputStream fos = new FileOutputStream(cfile);
				GZIPOutputStream gzos = new GZIPOutputStream(fos);
				ObjectOutputStream out = new ObjectOutputStream(gzos);
				out.writeObject(s1);
				out.writeObject(GUI.size.width*GUI.gridSize);
				out.writeObject(GUI.size.height*GUI.gridSize);
				out.flush();
				out.close();
				JOptionPane.showMessageDialog(save,
						"Done Saving.");
			}
			 catch (IOException e) {
			    System.out.println(e);
			}
		}});
Filemenu.add(save);
comp=new MouseComponent(GUI.size.width*GUI.gridSize,GUI.size.height*GUI.gridSize);
JScrollPane scroll= new JScrollPane(comp);
if (GUI.size.width*GUI.gridSize <maxWidth){
	maxWidth = GUI.size.width*GUI.gridSize+10;
```

```
        }
        if (GUI.size.height*GUI.gridSize < maxHeight){
            maxHeight = GUI.size.height*GUI.gridSize+10;
        }
        scroll.setBounds(0, 0, maxWidth, maxHeight);
        scroll.getViewport();
        getContentPane().setLayout(null);
        getContentPane().add(scroll);

        ChooserPanel= new JPanel();
        ChooserPanel.setBounds(maxWidth+2, 0, 135, maxHeight);
        getContentPane().add(ChooserPanel);
        ChooserPanel.setLayout(null);



        String [] files= {"BoxStart.png", "BoxEnd.png"};

        for(int i=0; i<files.length; i++){
            Mbutton= new CButtons("images/"+files[i],this);
            Mbutton.setBounds(22, 81+92*i, 75, 67);
            ChooserPanel.add(Mbutton);
        }
        JButton helpButton = new JButton("");
        helpButton.addActionListener(new ActionListener() {
            public synchronized void actionPerformed(ActionEvent actionEvent) {
                if (helpPanel==null){
                    helpPanel = new HelpPanel();       }
                helpPanel.setVisible(true);
            }
        });
        helpButton.setIcon(new ImageIcon(StartFrame.class.getResource("/com/sun/java/swing/plaf/windows/icon
s/Question.gif")));
        helpButton.setBounds(22, 11, 75, 55);
        ChooserPanel.add(helpButton);

    }
    public int getCompx() {
        return compx;
    }

    public void setCompx(int compx) {
        this.compx = compx;
    }
```

```
	public int getCompy() {
		return compy;
	}

	public void setCompy(int compy) {
		this.compy = compy;
	}
}
```

FaciltyRectangle.java

```
import java.awt.Component;
import java.awt.geom.Rectangle2D;
import javax.swing.*;

@SuppressWarnings("serial")
public class FacilityRectangle extends Component {

	public FacilityRectangle(double x, double y, String state, String c){
		rect=new Rectangle2D.Double(x, y, size, size);
		icon = new ImageIcon(state);
		code=c;

	}
	String code="";
	int size=GUI.gridSize;
	Rectangle2D rect;
	ImageIcon icon;

}
```

GBCHelper.java

```
import java.awt.*;

@SuppressWarnings("serial")
public class GBCHelper extends GridBagConstraints {
	/**
	 * PathFinder
	 */
	public  GBCHelper(int gridx, int gridy){
		this.gridx=gridx;
		this.gridy=gridy;
	}
	public GBCHelper(int gridx, int gridy, int gridwidth, int gridheight){
		this.gridheight=gridheight;
		this.gridwidth=gridwidth;
		this.gridx=gridx;
		this.gridy=gridy;

	}
	public GBCHelper setAnchor(int anchor){
		this.anchor=anchor;
		return this;
	}
	public GBCHelper setWeight(double weightx, double weighty){
		this.weightx=weightx;
		this.weighty=weighty;
		return this;
	}
	public GBCHelper setPad(int ipadx, int ipady){
		this.ipadx=ipadx;
		this.ipady=ipady;
		return this;
	}
	public GBCHelper setWeight(int weightx, int weighty){
		this.weightx=weightx;
		this.weighty=weighty;
		return this;
	}
}
```

Appendix C. Warehouse Simulation Code

BehaviorSpace.java

```
package team64.madkit;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.Console;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.io.Writer;
import java.util.Arrays;
import java.util.UUID;
/**
 * This is a utility used to launch Warehouse multiple times. It sets up the
 * simulation, changing one parameter, then starts the Warehouse main.
 *
 * @author Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class BehaviorSpace {

	UUID setID = UUID.randomUUID();
	public static void main (String args[]){
		BehaviorSpace space = new BehaviorSpace();
	}
	//args in order of: WarehouseName, # of items, # of orders, max orders at one time, speed, shelf height,
MultiPick ("true"/"false"), UUID, show graphics ("true"/"false")
	public String[] toArray(String WarehouseName, int numberOfItems, int numberOfOrders,
int maxOrdersAtSameTime, int updateSpeed, int shelfHeight, boolean Multipick, UUID setID,
boolean showGraphics, String notes){
		String[] args = new String[10];
		args[0] = WarehouseName;
		args[1] = numberOfItems+"";
		args[2] = numberOfOrders+"";
		args[3] = maxOrdersAtSameTime+"";
		args[4] = updateSpeed+"";
		args[5] = shelfHeight+"";
		if (Multipick){
			args[6] = "true";
```

```
            }else{
                  args[6] = "false";
            }
            args[7] = setID.toString().replace('-', '0');
            if (showGraphics){
                  args[8] = "true";
            }else{
                  args[8] = "false";
            }
            args[9] = notes;
            return args;
      }



int simulationsRun = 0;
int simulationsLaunched = 0;

      public BehaviorSpace(){
            System.out.println("Args in order of: WarehouseName, # of items, # of orders, max orders at one time, speed, shelf height, MultiPick (true/false), UUID, showGraphics (true/false), notes, totecapacity");
            for (int x = 0; x < 3; x++){
            for (int i = 0; i < 5; i++){
                  //String[] args = toArray("large3.t64",200,10,4,5,1,true,setID);
                  //new RunThread(i) {

                        //@Override public void run() {

                              String[] args = toArray("large3.t64",100,25,5,5,1,true,setID,false,"Iterating number of orders at the same time with multipick");

                              System.out.println("Simulation "+i+" is starting with parameters " + Arrays.toString(args));
                              try {

                                    Runtime r = Runtime.getRuntime();
                                    //System.out.println("java -classpath ./team64.jar;./madkitkernel-5.0.0.14.jar; team64.madkit.Warehouse ../database/hsqldb_1_8_1_1.jar "+args[0]+" "+args[1]+" "+args[2]+" "+args[3]+" "+args[4]+" "+args[5]+" "+args[6]+" "+args[7], null, null)");
                                    final Process p = r.exec("cmd /k java -classpath ./bin;./madkitkernel-5.0.0.14.jar;./database/hsqldb_1_8_1_1.jar; team64.madkit.Warehouse "+args[0]+" "+args[1]+" "+args[2]+" "+args[3]+" "+args[4]+" "+args[5]+" "+args[6]+" "+args[7]+" "+args[8]+" "+args[9]+"", null, null);
```

```
                    //System.out.println(p);

            final BufferedReader stdOut = new BufferedReader(new InputStreamReader(p.getInputStream()));
            final BufferedReader stdErr = new BufferedReader(new InputStreamReader(p.getErrorStream()));


                    // Thread that reads std out and feeds the writer given in input
                    new Thread() {
                        @Override public void run() {

                        }
                    }.start(); // Starts now

                    // Thread that reads std out and feeds the writer given in input
                    new Thread() {
                        @Override public void run() {
                            String line;
                            try {
                                while ((line = stdErr.readLine()) != null) {
                                //            System.err.println(line );
                                }
                            } catch (Exception e) {throw new Error(e);}
                            try {
                                stdErr.close();

                            } catch (IOException e) { /* Who cares ?*/ }
                        }
                    }.start(); // Starts now

                    String line = "";
                    try {
                        while ((line) != null) {
                            line = stdOut.readLine();
                        //      System.out.println(line) ;
                            if (line.contains("Your simulation is now ready to exit for the enjoyment of the
world.")){

                                System.out.println("Simulation "+i+" is closing");
                                try {
                                    //out.flush();
                                    stdOut.close();
                                } catch (IOException e) { /* Who cares ?*/ }
                                line = null;
                            }
                        }
```

```
                        } catch (Exception e) {throw new Error(e);}
                        try {
                                //out.flush();
                                stdOut.close();
                        } catch (IOException e) { /* Who cares ?*/ }


                        // System.out.println("</ERROR>");
                        //this.wait();
                        //int exitVal = p.waitFor();
                        //System.out.println("Process exitValue: " + p.exitValue());
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }// catch (InterruptedException e) {
                // TODO Auto-generated catch block
                //e.printStackTrace();
                //catch (InterruptedException e) {
                // TODO Auto-generated catch block
                // e.printStackTrace();
                //}
        //   }
        //}.start();
        //simulationsLaunched++;
    }
    //while(simulationsRun<simulationsLaunched){
        //delay
    //}
    }
    System.exit(0);
  }

  private class RunThread extends Thread{
    int runNumber;
    public RunThread(int i){
        runNumber = i;
    }
  }
  //}
}
```

Warehouse.java

```
package team64.madkit;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.io.IOException;
import java.net.ServerSocket;
import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.HashSet;
import java.util.Hashtable;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;

import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.filechooser.FileFilter;

import madkit.kernel.Agent;
import madkit.kernel.AgentAddress;
import madkit.kernel.Madkit;
import madkit.kernel.Madkit.LevelOption;
import madkit.kernel.Madkit.Option;
import madkit.kernel.Message;
import team64.madkit.messages.GoToDestinationMessage;
import team64.madkit.messages.LunchMessage;
import team64.madkit.messages.MoveTote;
import team64.madkit.messages.NewLocationMessage;
import team64.madkit.messages.NewOrder;
```

```
import team64.madkit.messages.SetLocationMessage;
import team64.madkit.messages.ToteMessage;


/**
 * Warehouse contains the Main for running the simulation. It sets up the
 * simulation and launches all the agents
 *
 * @author Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class Warehouse extends Agent {

	/**
	 * time in milliseconds for conveyor updates. Workers work at a multiple
	 * (5x) rate
	 */
	protected static int updateSpeed = 50;
	/**
	 * Delay between repaints of the watcher
	 */
	protected static int graphicsSpeed = 50;
	/** the number of orders to process for this run */
	public static int numberOfOrders = 10;
	/** the number of items to store in the warehouse */
	public static int numberOfItemsInWarehouse = 100;
	/** the maximum number of orders we can have on the conveyor system */
	private static int maxOrdersInProcess = 2;
	/** the number of conveyor segments between totes */
	protected static int toteSpacing = 2;
	/** flag for simulation in run state */
	static boolean simulationRunning = true;
	/** frame to display while initializing */
	static JFrame waitDialog = new JFrame();
	/** option to run without the graphical display */
	protected static boolean showWarehouseGraphics;
	/**
	 * only for the Test Demo - whether or not to make a path through the
	 * shelves
	 */
	protected static boolean usingPath = true;
```

```
/**
 * This is a note in the database about how we selected items from the
 * database. It will usually be 'database asc order' or 'random'
 */
protected static String itemSetup = "database asc order";
/**
 * sort items on shelves by popularity (more popular closer to picking
 * station
 */
protected static boolean sortByPopularity = false;
/**
 * try to give each item stop an equal number of items if set to true.
 * Otherwise we fill up the shelves in each item stop then go on to the next
 * one. This only works if each item stop domain has equal access to shelves
 */
protected static boolean distributeItemsEvenly = false;

// static AgentAddress myOwnAddress=null;
/**
 * size of the warehouse in grid units (basically a grid unit holds one
 * tote, one conveyor segment, one worker, or a tote + conveyor segment)
 */
protected static int warehouseWidth;
protected static int warehouseHeight;
/** the number of pickers is generally the same as the item stops */
protected static int numberOfPickers = 0;
/** item stops in the simulation layout */
protected static int numberOfItemStops = 0;
/**
 * the number of active picking stations will be less than or equal to the
 * number of item stops. If not all shelves are filled in the warehouse then
 * an picking stations (item stop) may not be in use.
 */
protected static int activeStations = 0;
protected static ArrayList<ItemStop> itemStops;
protected static String stopListString = "";
/** set of item stop locations so we can easily find a close one when needed */
protected static HashSet<Point> itemStopLocations = new HashSet<Point>();
protected static Point workerStartingLocation;
/**
 * this is the layout file used to generate the warehouse. If null we use a
 * default test layout
 */
protected static String layoutFileName = null;
```

```
/** number of steps in the simulation from the simulation scheduler */
protected static long steps = 0;
/** step number when we launch the first order tote */
protected static long startStep = 0;
/** step number when all orders are filled */
protected static long endStep = 0;

/** set to true to signal the first order */
private boolean firstOrder = true;

/** the simulation scheduler. Each tic is a simulation step. */
protected Timer simulationScheduler;

/** random number generator */
protected Random r = new Random();

/** schedules worker actions for the simulation */
protected WorkerScheduler workerScheduler;

protected ConveyorScheduler conveyorScheduler;

/** schedules the conveyor system actions for the simulation */
protected ConveyorSystem conveyorSystem;

/** schedules the painting of the main warehouse graphics */
protected DisplayScheduler displayScheduler;
/**
 * sets up probes for the agents in the warehouse and sets up the main
 * graphical display
 */
protected WarehouseWatcher warehouseWatcher;

/** order in progres panel */
protected OrderWatcher orderWatcher;

/**
 * sets up the layout of the warehouse
 */
protected WarehouseLayout warehouseLayout = new WarehouseLayout();
/**
 * rand is used to randomly move an agent 1 grid step when needed 0=no
 * movement, 1=+1, 2=-1
```

```
 */
protected static Random rand = new Random();
/** keep a set of totes to process later that have incomplete orders */
public static List<Tote> incompleteToteList = new ArrayList<Tote>();

/** an x-y array of grid units */
private static GridUnit[][] grid = null; // the environment grid
/** database connection */
static Connection conn; // connection to the database
protected String dburl = "jdbc:hsqldb:hsql://localhost/DistriButionCenter";
protected String user = "SA";
protected String password = "";
/** the id from the "runs" table (known after we store the start timestamp) */
protected static int runID = 0;
/** the total number of simulation steps from start to finish */
protected long processSteps;

/** the hashtable of orders to pick */
static private Hashtable<Integer, List<Integer>> orders;
/** enumeration of the orders hashtable so we can iterate through it */
protected Enumeration orderIDs;
/**
 * the next order that will be processed. If this is -1 no orders have been
 * started If this is -99 then all orders are in process or finished
 */
protected static int nextOrderID = -1;
/**
 * the set of orders that are in process. If this set is empty either the
 * order process has not been started OR it is all finished
 */
protected static ArrayList<Integer> ordersInProcess = new ArrayList<Integer>();

/** hashtable of an item id and which bin it is stored in */
static private Hashtable<Integer, StorageBin> itemLocations;

// private List tasks = new ArrayList();
/** the maximum number of items for a tote */
static final int toteCapacity = 15;

/** the default number of bins in a shelf */
static int shelfHeight = 1;

protected static int zoneSingle = 1;
protected static int zoneMulti = 2;
```

```
protected static int discreteSingle = 3;
protected static int discreteMulti = 4;
public static int workerSteps;

/**
 * the picking mode, one of discrete (0), zoneSingle (1), zoneMulti (2).
 * zone = picker assigned to a station zoneMulti = picker has a list of
 * items to get then returns to tote at the item stop zoneSingle = picker
 * gets one item at a time from the shelves discrete = picker has no
 * specific area for picking items discreteMulti = picker has a list of
 * items to get then returns to tote at the item stop discreteSingle =
 * picker gets one item at a time from the shelves
 *
 * */
protected static int pickingMode = zoneSingle;

/** used for demo layout */
protected static boolean useDemoStop1;
protected static boolean useDemoStop2;
protected static boolean useDemoStop3;
protected static boolean useDemoStop4;

public static GridUnit[][] getGrid() {
	if (grid == null) {
		grid = new GridUnit[warehouseWidth][warehouseHeight];
	}
	return grid;
}

/** public access method for getting a Grid Unit in the grid */
public static GridUnit getGridUnit(int x, int y) {
	return grid[x][y];
}

protected void activate() {
	waitDialog.setVisible(false);
	getLogger().setWarningLogLevel(Level.INFO);
	setLogLevel(Level.FINE);
	createGroupIfAbsent("warehouse", "conveyorsystem", true, null);// needs
	// to be
	// in
	// this
```

```
// group
// to
// send
// messages
// to
// conveyors
createGroupIfAbsent("warehouse", "workers", true, null); // needs to be
// in this
// group to
// send
// messages
// to
// workers
createGroupIfAbsent("warehouse", "warehouse", true, null);

requestRole("warehouse", "warehouse", "warehouse");
requestRole("warehouse", "workers", "warehouse");
requestRole("warehouse", "conveyorsystem", "warehouse");

// requestRole("warehouse","conveyorsystem","manager");
setOrders(new Hashtable());
// Make the database connection ...
try {
	// System.out.println( Class.forName("org.hsqldb.jdbcDriver"));
	Class.forName("org.hsqldb.jdbcDriver").newInstance();

	conn = DriverManager.getConnection(dburl, user, password);
	System.out.println("Database connection established");
} catch (Exception ex) {
	ex.printStackTrace();
	System.out.println("Exception: " + ex.getMessage()
			+ ". Did you forget to start the database?");
	System.exit(99);
}

// the master warehouse layout
launchAgent(warehouseLayout);
// now we can launch other agents from the initialize method
warehouseLayout.initialize(true);
workerStartingLocation = warehouseLayout.entranceLocation;
itemStops = new ArrayList<ItemStop>();
itemStops = warehouseLayout.itemStops;

// we can either randomly stock shelves or apply an algorithm to stock
```

```
// in a logical manner by popularity or "on sale" items
// randomlyStockShelves(numberOfItemsInWarehouse);
// stock shelves the same way every run
stockShelves(numberOfItemsInWarehouse);
// domains must be calculated AFTER the shelves are stocked
calculateDomains();
System.out.println("getting orders from database");
getOrders(numberOfItemsInWarehouse, numberOfOrders);
// Now we are ready to start processing orders
// launch a picker for each item stop
for (int i = 0; i < warehouseLayout.itemStops.size(); i++) {
	Picker picker = new Picker();
	picker.destination = warehouseLayout.itemStops.get(i)
			.getSegmentInfo().getSegmentLocation();
	// Point behind = stop.segmentInfo.getBehindLocation();
	warehouseLayout.itemStops.get(i).picker = picker;
	picker.setPickerStopAssignment(warehouseLayout.itemStops.get(i));
	// set picker color to match domain color
	Color domainColor = warehouseLayout.itemStops.get(i).stopInfo.domainColor;
	picker.workerInfo.setWorkerColor(domainColor);
	launchAgent(picker, 10, false);
	warehouseLayout.itemStops.get(i).picker = picker;
	picker.workerInfo.setLocation(warehouseLayout.itemStops.get(i)
			.getBehindLocation());
	picker.sendMessage("warehouse", "workers", "warehouse",
			new SetLocationMessage(warehouseLayout.itemStops.get(i)
					.getBehindLocation()));
	numberOfPickers++;
}
numberOfItemStops = warehouseLayout.itemStops.size();
stopListString = "";

for (int i = 0; i < warehouseLayout.itemStops.size(); i++) {
	Point stopLocation = warehouseLayout.itemStops.get(i)
			.getSegmentInfo().getSegmentLocation();
	String stopString = "[" + stopLocation.x + "," + stopLocation.y
			+ "] ";
	stopListString = stopListString + stopString;
	itemStopLocations.add(stopLocation);
}

// System.out.println("initiate a tote");
```

```
// initiateBasicTote(toteLauncherLocation);
// ArrayList order = new ArrayList();

// order.add(30);
// orders.put(0,order);
// conveyorSystem= new ConveyorSystem();
// launchAgent(conveyorSystem,false);AddTask
// launchTotes(2);

// test only

/*
 * AgentAddress s = getAgentWithRole("warehouse", "workers", "sorter");
 * grid[0][0].addAgent(p); boolean result =
 * grid[0][0].checkOccupantCompatibility(s); result =
 * grid[0][1].checkOccupantCompatibility(s);
 *
 * AgentAddress t = getAgentWithRole("warehouse", "container", "tote");
 *
 * AgentAddress cs = getAgentWithRole("warehouse","conveyorsystem",
 * "segment");
 *
 * grid[2][2].addAgent(cs); result =
 * grid[2][2].checkOccupantCompatibility(t); result =
 * grid[2][2].checkOccupantCompatibility(s);
 */

simulationScheduler = new Timer();
if (logger != null) {
	logger.info("Started the Simulation Scheduler");
}
launchAgent(simulationScheduler, false);
workerScheduler = new WorkerScheduler();
if (logger != null) {
	logger.info("Launching worker scheduler");
}
launchAgent(workerScheduler, false);

conveyorScheduler = new ConveyorScheduler();
if (logger != null) {
	logger.info("Launching conveyor scheduler");
}
launchAgent(conveyorScheduler);
```

```
        if (showWarehouseGraphics) {
            /** set up a watcher for the warehouse with a graphical display */
            warehouseWatcher = new WarehouseWatcher(this);
            launchAgent(warehouseWatcher, true);
            /** set up a scheduler for the watcher */
            displayScheduler = new DisplayScheduler();
            launchAgent(displayScheduler, false);

            orderWatcher = new OrderWatcher();
            launchAgent(orderWatcher, true);

        }

}

/**
 * this attempts to distribute the inventory items evenly to all item stops
 *
 * @param numberOfItems
 */
protected void evenlyStockShelves(int numberOfItems) {
        itemSetup = "database asc order";
        // get items from database
        // hashtable keyed on bin slots
        setItemLocations(new Hashtable(numberOfItems));
        // precalculte a set of shelves that are closest to each item stop

        for (int s = 0; s < numberOfItemStops; s++) {

        }

        ResultSet rs = null;
        try {
            if (conn != null) {
                System.out.println("Database connection exists");
            }
            Statement st = conn.createStatement();

            String table = "Items";
            // Select items (0-99) from the database in a random order
            String sql = "SELECT id,popularity,size FROM items where id<"
                    + numberOfItems + " ORDER BY id asc";
```

```
            rs = st.executeQuery(sql); // run the query

            int stopIndex = 0;
            for (int i = 0; i < numberOfItems; i++) {
                  rs.next();
                  int itemID = rs.getInt("id");
                  int popularity = rs.getInt("popularity");
                  int size = rs.getInt("size");
                  // use the next item stop for this item
                  // ItemStop stop =

                  getItemLocations().put(itemID,
                              (StorageBin) warehouseLayout.allBins.get(i));
                  ((StorageBin) warehouseLayout.allBins.get(i)).setItemId(itemID);
                  ((StorageBin) warehouseLayout.allBins.get(i))
                              .setQuantity(((StorageBin) warehouseLayout.allBins
                                          .get(i)).getMax());
                  logger.fine("Stocked item "
                              + itemID
                              + " in bin at "
                              + ((StorageBin) warehouseLayout.allBins.get(i))
                                          .getLocation());
            }
            System.out.println("Stocked " + numberOfItems
                        + " items in the warehouse.");
      } catch (SQLException se) {
            System.out.println("SQLException: " + se.getMessage());
            se.printStackTrace();
      }

}

/**
 * distributes items into the shelves (bins) with the more popular items
 * closer to the item stop
 */
protected void stockShelvesbyPopularity(int numberOfItems) {
      // get items from database
      // hashtable keyed on bin slots
      itemSetup = "random by popularity";
      setItemLocations(new Hashtable(numberOfItems));

      ResultSet rs = null;
      try {
```

```
            if (conn != null) {
                System.out.println("Database connection exists");
            }
            Statement st = conn.createStatement();

            String table = "Items";
            // Select items (0-99) from the database in a random order
            String sql = "SELECT id,popularity,size FROM items where id<"
                    + numberOfItems + " ORDER BY RAND()";
            rs = st.executeQuery(sql); // run the query

            for (int i = 0; i < numberOfItems; i++) {
                rs.next();
                int itemID = rs.getInt("id");
                int popularity = rs.getInt("popularity");
                int size = rs.getInt("size");
                getItemLocations().put(itemID,
                        (StorageBin) warehouseLayout.allBins.get(i));
                ((StorageBin) warehouseLayout.allBins.get(i)).setItemId(itemID);
                ((StorageBin) warehouseLayout.allBins.get(i))
                        .setQuantity(((StorageBin) warehouseLayout.allBins
                                .get(i)).getMax());
                logger.fine("Stocked item "
                        + itemID
                        + " in bin at "
                        + ((StorageBin) warehouseLayout.allBins.get(i))
                                .getLocation());
            }
            System.out.println("Stocked " + numberOfItems
                    + " items in the warehouse.");
        } catch (SQLException se) {
            System.out.println("SQLException: " + se.getMessage());
            se.printStackTrace();
        }
    }

    /** distributes items into the shelves (bins) */
    protected void randomlyStockShelves(int numberOfItems) {
        // get items from database
        // hashtable keyed on bin slots
        setItemLocations(new Hashtable(numberOfItems));
        itemSetup = "database random order";
```

```
        ResultSet rs = null;
        try {
            if (conn != null) {
                System.out.println("Database connection exists");
            }
            Statement st = conn.createStatement();

            String table = "Items";
            // Select items (0-99) from the database in a random order
            String sql = "SELECT id,popularity,size FROM items where id<"
                    + numberOfItems + " ORDER BY RAND()";
            rs = st.executeQuery(sql); // run the query

            for (int i = 0; i < numberOfItems; i++) {
                rs.next();
                int itemID = rs.getInt("id");
                int popularity = rs.getInt("popularity");
                int size = rs.getInt("size");
                getItemLocations().put(itemID,
                        (StorageBin) warehouseLayout.allBins.get(i));
                ((StorageBin) warehouseLayout.allBins.get(i)).setItemId(itemID);
                ((StorageBin) warehouseLayout.allBins.get(i))
                        .setQuantity(((StorageBin) warehouseLayout.allBins
                                .get(i)).getMax());
                logger.fine("Stocked item "
                        + itemID
                        + " in bin at "
                        + ((StorageBin) warehouseLayout.allBins.get(i))
                                .getLocation());
            }
            System.out.println("Stocked " + numberOfItems
                    + " items in the warehouse.");
        } catch (SQLException se) {
            System.out.println("SQLException: " + se.getMessage());
            se.printStackTrace();
        }
    }

    /** distributes items into the shelves (bins) */
    protected void stockShelves(int numberOfItems) {
        itemSetup = "database asc order";
        // get items from database
        // hashtable keyed on bin slots
        setItemLocations(new Hashtable(numberOfItems));
```

```
        ResultSet rs = null;
        try {
            if (conn != null) {
                System.out.println("Database connection exists");
            }
            Statement st = conn.createStatement();

            String table = "Items";
            // Select items (0-99) from the database in a random order
            String sql = "SELECT id,popularity,size FROM items where id<"
                    + numberOfItems + " ORDER BY id asc";
            rs = st.executeQuery(sql); // run the query

            for (int i = 0; i < numberOfItems; i++) {
                rs.next();
                int itemID = rs.getInt("id");
                int popularity = rs.getInt("popularity");
                int size = rs.getInt("size");
                getItemLocations().put(itemID,
                        (StorageBin) warehouseLayout.allBins.get(i));
                ((StorageBin) warehouseLayout.allBins.get(i)).setItemId(itemID);
                ((StorageBin) warehouseLayout.allBins.get(i))
                        .setQuantity(((StorageBin) warehouseLayout.allBins
                                .get(i)).getMax());
                logger.fine("Stocked item "
                        + itemID
                        + " in bin at "
                        + ((StorageBin) warehouseLayout.allBins.get(i))
                                .getLocation());
            }
            System.out.println("Stocked " + numberOfItems
                    + " items in the warehouse.");
        } catch (SQLException se) {
            System.out.println("SQLException: " + se.getMessage());
            se.printStackTrace();
        }
    }

    /**
     * this calculates the shortest path between the picking station (item stop)
     * and an item bin. the item id with the shortest path is added to the
```

```
 * "domain" of the item stop
 */
public void calculateDomains() {
	// calculate domains only if there is at least one item stop AND there
	// is at least one storage bin
	activeStations = 0;
	if (warehouseLayout.itemStops.size() > 0
			&& warehouseLayout.allBins.size() > 0) {
		// for each bin find the closest item stop
		for (int i = 0; i < warehouseLayout.allBins.size(); i++) {
			StorageBin bin = warehouseLayout.allBins.get(i);
			if (bin.getItemId() > 0) { // skip item ids of -1 (no items in
												// the bin)
				Router pather = new Router();
				int bestPath = 9999;
				ItemStop bestStop = null;
				// find which stop is closest to this bin
				for (int p = 0; p < warehouseLayout.itemStops.size(); p++) {
					ItemStop stop = warehouseLayout.itemStops.get(p);
					List<Point> path = pather.getBestPath(
							bin.getLocation(),
							stop.segmentInfo.getSegmentLocation());
					pather.overlayElement[bin.getLocation().x][bin
							.getLocation().y].setStart(false);
					pather.overlayElement[stop.segmentInfo
							.getSegmentLocation().x][stop.segmentInfo
							.getSegmentLocation().y].setEnd(false);
					if (path.size() == 0) {
						System.out.println("Skipping");

					}
					// System.out.println("Is "+ path.size() +" less than "
					// + bestPath+"?");
					if (path.size() < bestPath) {
						bestPath = path.size();
						bestStop = stop;
					}

				}
				bestStop.stopInfo.domain.add(bin.getItemId());
				// set color of the grid for this bin to the domain color
				grid[bin.getLocation().x][bin.getLocation().y]
						.setStopDomainColor(bestStop.stopInfo
								.getDomainColor());
```

```
                    logger.fine("Added item " + bin.getItemId()
                                + " to domain for item stop at "
                                + bestStop.segmentInfo.getSegmentLocation().x + ","
                                + bestStop.segmentInfo.getSegmentLocation().y);

                }

            }

            for (int p = 0; p < warehouseLayout.itemStops.size(); p++) {
                ItemStop stop = warehouseLayout.itemStops.get(p);
                if (stop.stopInfo.domain.size() > 0) {
                    activeStations++;
                }
                logger.fine("Item Stop at "
                            + stop.segmentInfo.getSegmentLocation().x + ","
                            + stop.segmentInfo.getSegmentLocation().y + " has "
                            + stop.stopInfo.domain.size() + " items in its domain");
            }
            /*
             * //set the domain color for each bin for (int p = 0; p <
             * warehouseLayout.itemStops.size(); p++){ ItemStop stop =
             * warehouseLayout.itemStops.get(p); Color domainColor =
             * stop.stopInfo.domainColor; for (int b = 0; b <
             * warehouseLayout.allBins.size(); b++){ StorageBin bin =
             * warehouseLayout.allBins.get(b); int itemID = bin.getItemId(); if
             * (itemID >-1 && stop.stopInfo.domain.contains(itemID)){
             * bin.setBinColor(domainColor); } }
             */

        }
    }

    /**
     * read a number of orders from the pre-generated order database table ...
     * which table depends on number of items
     */
    protected void getOrders(int numberOfItems, int numberOfOrders) {

        String orderTableName = "OrdersFor" + numberOfItems + "Items";
        ResultSet rs = null;
        try {
```

```
            // System.out.println(conn);
            Statement st = conn.createStatement();

            String table = "Items";
            // orders in a specific sequence
            String sql = "SELECT id,qty,items FROM " + orderTableName
                        + " where id<" + numberOfOrders + " ORDER BY ID ASC";
            // orders in a random sequence (but still same orders as above)
            // String sql = "SELECT id,qty,items FROM " + orderTableName +
            // " where id<" + numberOfOrders + " ORDER BY RAND()";
            rs = st.executeQuery(sql); // run the query

            for (int i = 0; i < numberOfOrders; i++) {
                  rs.next();
                  int orderID = rs.getInt("id");
                  int qyt = rs.getInt("qty");
                  String orderList = rs.getString("items");
                  // make the list of item Integers...
                  List<Integer> order = new ArrayList();
                  String[] items = orderList.split(",");
                  for (int item = 0; item < qyt; item++) {
                        order.add(Integer.parseInt(items[item]));
                  }
                  orders.put(orderID, order);
                  logger.fine("Order " + orderID + " contains items " + orderList);
            }
      } catch (SQLException se) {
            System.out.println("SQLException: " + se.getMessage());
            se.printStackTrace();
      }
      orderIDs = orders.keys();
}

/**
 * This is called by the Timer scheduler. This checks if a new tote needs to
 * be spawned and keeps track of the orders in process.
 */
protected void orderProcessor() {

      if (firstOrder == true) {
            recordStart();
            firstOrder = false;
      }
      if (ordersInProcess.size() < maxOrdersInProcess && spawnerExists()
```

```
                && !orders.isEmpty()) {
            if (orderIDs.hasMoreElements()) {
                nextOrderID = (Integer) orderIDs.nextElement();
                ordersInProcess.add(nextOrderID);
                sendMessage("warehouse", "conveyorsystem", "spawner",
                        new NewOrder(nextOrderID));
            } else {
                if (!incompleteToteList.isEmpty()) {
                    sendMessage("warehouse", "conveyorsystem", "spawner",
                            new ToteMessage(incompleteToteList.get(0)));
                    ordersInProcess.add(incompleteToteList.get(0).toteInfo
                            .getOrderNumber());
                    incompleteToteList.remove(0);
                }
            }
        }
    }

    /** checks to see if the tote spawner has been launched yet */
    protected boolean spawnerExists() {

        if (getAgentsWithRole("warehouse", "conveyorsystem", "spawner") != null) {
            return true;
        }
        return false;
    }

    /** runs continuously while the Warehouse is running */
    protected void live() {
        boolean color = false;
        while (simulationRunning) {
            pause(updateSpeed);
            if (getOrders().isEmpty() && ordersInProcess.isEmpty()) {
                recordEnd();
            }
        }

    }

    /** returns a GridUnit with an agent of this specific AgentAddress */
    private GridUnit findAgent(AgentAddress address) {
        for (int i = 0; i < warehouseWidth; i++) {
```

```
            for (int p = 0; p < warehouseHeight; p++) {
                if (grid[i][p].getOccupants().contains(address)) {
                    return grid[i][p];
                }
            }
        }
        return null;
}

/** returns 0, +1 or -1 for a random grid movement */
public static int getRandomStep() {
        int step = rand.nextInt(3);
        return step - 1;
}

private AgentAddress getPickerAtLocation(int x, int y) {
        this.sendMessage("warehouse", "workers", "pickers", new Message());
        return null;

}

private List<Point> addPoints(List<List<Point>> points, List<Point> list) {
        for (int i = 0; i < points.size(); i++) {
            List<Point> innerList = points.get(i);
            for (int p = 0; p < innerList.size(); p++) {
                list.add(innerList.get(p));
            }
        }
        // System.out.println("added list = " +list.size());
        return list;
}

/**
 * Returns an array list of valid grid units surrounding the grid unit at
 * Point p
 */
public static List<Point> getSorrounding(Point p) {
        List<Point> open = new ArrayList<Point>();
        open.add(new Point(p.x + 1, p.y));
        open.add(new Point(p.x, p.y + 1));
        open.add(new Point(p.x + 1, p.y + 1));
        open.add(new Point(p.x - 1, p.y));
        open.add(new Point(p.x - 1, p.y - 1));
        open.add(new Point(p.x, p.y - 1));
```

```
        open.add(new Point(p.x + 1, p.y - 1));
        open.add(new Point(p.x - 1, p.y + 1));

        // System.out.println("valid open size = " + validOpen.size());
        return open;
    }

    public static List<Point> getAdjacent(Point p) {
        List<Point> open = new ArrayList<Point>();
        open.add(new Point(p.x + 1, p.y));
        open.add(new Point(p.x, p.y + 1));
        open.add(new Point(p.x - 1, p.y));
        open.add(new Point(p.x, p.y - 1));

        // System.out.println("valid open size = " + validOpen.size());
        return open;
    }

    private void launchPickers(int numberOfPickers) {
        if (logger != null)
            logger.info("Launching " + numberOfPickers + " pickers");

        for (int i = 0; i < numberOfPickers; i++) {
            launchAgent(new Picker(), 10, false);
        }
    }

    /** End Agent */
    protected void end() {
        simulationRunning = false;
        // pickersList=null;
        // sortersList=null;
        if (logger != null) {
            logger.info("Stopping Warehouse Simulation");
        }
        // stop all the schedulers
        conveyorScheduler.end();
        workerScheduler.end();
        displayScheduler.end();
    }

    /** called by framework to set up graphics for this Agent */
```

```
	public void setupFrame(JFrame frame) {
		JPanel panel = new JPanel();
		panel.setBounds(0, 0, 240, 280);
		panel.add(new JLabel(
				"Please wait while the simulation is initialized..."));
		frame.setTitle("Please Wait...");
		frame.add(panel);
		frame.setSize(500, 350);
		waitDialog = frame;
	}

	protected void lunch() {
		broadcastMessage("warehouse", "workers", "picker",
				new GoToDestinationMessage(new Point(warehouseWidth - 1, 2)));
	}

	public void receiveMessage(Message m) {
		getLogger().setWarningLogLevel(Level.INFO);
		setLogLevel(Level.FINE);
		// System.out.println("Warehouse got a message: " + m);
		if (m != null) {
			if (logger != null) {
				logger.finest("Warehouse got a Message from " + m.getSender()
						+ " says: " + m.toString());
			}

			// used for testing - make workers go to the lunchroom
			if ((m instanceof LunchMessage)) {
				lunch();
			}

			// this message is sent by the conveyors to request a tote movement
			if ((m instanceof MoveTote)) {
				handleMoveToteMessage(m);
			}

			// this message is sent constantly by the pickers when moving
			else if ((m instanceof NewLocationMessage)) {
				AgentAddress address = m.getSender();
				logger.finest("New Location Message Received by Warehouse, reply will be sent back to
sender.");
				Point oldLocation = ((NewLocationMessage) m).getOldLocation();
				Point newLocation = ((NewLocationMessage) m).getNewLocation();
				logger.finer("Worker moving from " + oldLocation + " to "
```

```
                        + newLocation);
                // make sure location request is within warehouse bounds
                boolean locationInBounds = checkBounds(newLocation);
                if (locationInBounds) {
                    GridUnit newLoc = grid[newLocation.x][newLocation.y];
                    GridUnit oldLoc = grid[oldLocation.x][oldLocation.y];
                    if (!newLoc.isBlocked()) {
                        oldLoc.removeAgent(address);
                        newLoc.addAgent(address);
                        // send back message confirming move
                        sendReply(m, new NewLocationMessage(oldLocation,
                                newLocation));
                    } else {// could not move
                        // send back message with old location (agent did not
                        // move)
                        logger.severe("Agents not compatible (move not completed) at "
                                + oldLocation + " to " + newLocation);
                        sendReply(m, new NewLocationMessage(oldLocation,
                                oldLocation));
                    }
                } else {// new location was not in the bounds of the warehouse
                    // send back message with old location (agent did not
                    // move)
                    logger.fine("New Location was not in the warehouse bounds (move not completed) at "
                            + oldLocation + " to " + newLocation);
                    sendReply(m, new NewLocationMessage(oldLocation,
                            oldLocation));
                }
            }
            // used when warehouse objects are first set into the Grid
            else if (m instanceof SetLocationMessage) {
                AgentAddress address = m.getSender();
                logger.finer("Set Location Message Received by Warehouse.");
                Point location = ((SetLocationMessage) m).getlocation();
                GridUnit loc = grid[location.x][location.y];
                loc.addAgent(address);
            } else {// no handler for this kind of message
                System.out.println("Warehouse got a strange message from "
                        + m.getSender());
            }
        }
    }
```

```
/** handler for the MoveTote message */
protected void handleMoveToteMessage(Message m) {
	Point p2 = ((MoveTote) m).getDestination();
	MoveTote mt = (MoveTote) m;
	Tote tote = mt.getTote();

	if (tote == null) {
		System.out.println("tote is null, cannot move tote");
	}

	try {
		int orderNumber = tote.toteInfo.orderNumber;
		if (getOrders().get(orderNumber).isEmpty()) {
			logger.finest("tote has a complete order ...");
			// keep tote moving along to the End segment
			if (grid[p2.x][p2.y].containsConveyor()) {
				sendMessage((grid[p2.x][p2.y].occupants.get(0)),
						new ToteMessage(((MoveTote) m).getTote()));
			}
			return;
		}

		else if (orderNumber != -1
				&& (tote.toteInfo.filled < Warehouse.toteCapacity)) {
			// check for an item stop next to the segment where this tote
			// is...
			List<Point> neighbors = getAdjacent(((MoveTote) m).getFrom());// checks
											// neighbors
											// of
											// the
											// destination
			ArrayList<Point> intersection = new ArrayList<Point>(neighbors);
			intersection.retainAll(itemStopLocations);
			if (!intersection.isEmpty()) {
				// stop is next to us so we need to check if the orde
				// contains an item in this stop's domain
				// which stop is this?
				Point stopLocation = intersection.get(0); // there should be
										// only one
				// get the the item stop at this location
				ItemStop stop = getItemStopAtLocation(stopLocation);
				if (stop != null) {// it should not be null, but better
							// check
```

```
                        if (getNextItemAtStop(orderNumber, stop) != -1
                                && stop.tote == null) {
                            // send tote to the item stop if there is no tote
                            // there already
                            sendMessage(
                                    (grid[stopLocation.x][stopLocation.y].occupants
                                            .get(0)), new ToteMessage(tote));
                            return;
                        }
                        // send the tote to the next conveyor segment, this stop
                        // is not in our order
                        else if (grid[p2.x][p2.y].containsConveyor()) {
                            sendMessage((grid[p2.x][p2.y].occupants.get(0)),
                                    new ToteMessage(tote));
                            return;
                        }
                    }
                }
            }

        } catch (NullPointerException e) {
            System.err.println(e);
            System.err.println(((MoveTote) m).getTote());
            System.out
                    .println("Warehouse NPE in message handler .. THIS IS THE ONE ^^^^^^^^^^^^^^^^^^^^");
        }
        if (grid[p2.x][p2.y].containsConveyor()) {
            sendMessage((grid[p2.x][p2.y].occupants.get(0)), new ToteMessage(
                    tote));
        }

}

/** provides access to lunchroom layout */
protected Point getLunchroomLocation() {
    return warehouseLayout.lunchroomLocation;
}

/** provides access to lunchroom layout */
protected Dimension getLunchroomSize() {
    return new Dimension(warehouseLayout.lunchroomWidth,
            warehouseLayout.lunchroomDepth);
```

```
}

/**
 * convenience method to check if a location point is OK (within the
 * warehouse bounds)
 */
private boolean checkBounds(Point location) {
	if ((location.x > Warehouse.warehouseWidth - 1) || (location.x < 0)) {
		return false;
	}
	if ((location.y > Warehouse.warehouseHeight - 1) || (location.y < 0)) {
		return false;
	}

	return true;
}

/** startup per instructions in the Madkit kernal API */
public static void startupSimulation() {
	// executeThisAgent(args, 1, true);
	String[] argss = { LevelOption.agentLogLevel.toString(), "FINE",
			Option.launchAgents.toString(),// gets the -- launchAgents
							// string
			Warehouse.class.getName() + ",false,1;" };
	Madkit.main(argss);
}

static String notes;

/**
 * @param args
 */

public static void main(String[] args) {
	// args in order of: WarehouseName, # of items, # of orders, max orders
	// at one time, speed, shelf height, MultiPick ("true"/"false")
	if (args.length > 0) {
		// if these trys catch anything it will simple revert to the
		// defaults as already set
		layoutFileName = args[0];
		System.out.println("filename of warehouse is " + layoutFileName);
		try {
			numberOfItemsInWarehouse = Integer.parseInt(args[1]);
		} catch (Exception e) {
```

```
}
try {
	numberOfOrders = Integer.parseInt(args[2]);
} catch (Exception e) {

}
try {
	maxOrdersInProcess = Integer.parseInt(args[3]);
} catch (Exception e) {

}
try {
	updateSpeed = Integer.parseInt(args[4]);
} catch (Exception e) {

}
try {
	shelfHeight = Integer.parseInt(args[5]);
} catch (Exception e) {

}
try {
	if (args[6].equals("true")) {
		pickingMode = zoneMulti;
	} else {
		pickingMode = zoneSingle;
	}

} catch (Exception e) {

}
try {
	setID = args[7];
} catch (Exception e) {

}
try {
	if (args[8].equals("true")) {
		showWarehouseGraphics = true;
	} else {
		showWarehouseGraphics = false;
```

```
            }

        } catch (Exception e) {

        }
        try {
            notes = args[9];
        } catch (Exception e) {

        }
        startupSimulation();

    } else {
        final JFrame setup = new SetupFrame();
        setup.setName("SetupFrame");
        setup.setSize(600, 500);
        final ParameterPanel pp = new ParameterPanel();
        setup.add(pp);
        setup.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        // create custom close operation
        setup.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                maxOrdersInProcess = Integer
                        .parseInt(pp.txtMaxOrdersInProcess.getText());
                updateSpeed = Integer.parseInt(pp.txtUpdateSpeed.getText());
                numberOfOrders = Integer.parseInt(pp.textFieldOrders
                        .getText());
                shelfHeight = Integer.parseInt(pp.txtShelfHeight.getText());
                numberOfItemsInWarehouse = (int) ((Integer) pp.comboBoxItems
                        .getSelectedItem());
                useDemoStop1 = pp.chkLoc1.isSelected();
                useDemoStop2 = pp.chkLoc2.isSelected();
                useDemoStop3 = pp.chkLoc3.isSelected();
                useDemoStop4 = pp.chkLoc4.isSelected();

                if (pp.comboBox.getSelectedItem() == null) {
                    layoutFileName = "Demo Test";
                } else {
                    layoutFileName = (String) pp.comboBox.getSelectedItem();
                }

                boolean useZone = pp.chckbxZone.isSelected();
                boolean useMulti = pp.chckbxMulti.isSelected();
                if (useZone && useMulti) {
```

```
        pickingMode = zoneMulti;
    } else if (useZone && !useMulti) {
        pickingMode = zoneSingle;
    } else if (!useZone && useMulti) {
        pickingMode = discreteMulti;
    } else if (!useZone && !useMulti) {
        pickingMode = discreteSingle;
    }

    if (layoutFileName.equals("Browse...")) {
        JFileChooser chooser = new JFileChooser();

        FileFilter filter = new FileFilter() {

            public boolean accept(File f) {
                String ext = "";
                String s = f.getName();
                int i = s.lastIndexOf('.');

                if (i > 0 && i < s.length() - 1) {
                    ext = s.substring(i + 1).toLowerCase();
                }
                if (ext.toLowerCase().equals("t64")) {
                    return true;
                }
                return false;
            }

            public String getDescription() {
                // TODO Auto-generated method stub
                return "Team 64 Warehouse Layout Files";
            }

        };
        chooser.setFileFilter(filter);
        chooser.setCurrentDirectory(new File("./"));
        chooser.showDialog(pp, "Open Layout File (*.t64)");
        layoutFileName = chooser.getSelectedFile().getPath();
    }
    setup.dispose();
    // start the database
    Runtime r = Runtime.getRuntime();
```

```
                try {
                    r.exec("org.hsqldb.Server");
                } catch (IOException e1) {
                    // error could be that it is running already or some
                    // other problem
                    // check if port 9001 is open
                    try {
                        ServerSocket srv = new ServerSocket(9001);
                        srv.close();
                        srv = null;
                    } catch (java.net.BindException e3) {
                        // socket is in use, good...
                        System.out.println("HSQLDB already running (OK)");
                    } catch (IOException e2) {
                        // really is not running :(
                        System.out
                                .println("Could  not start the HSQLDB Server");
                        System.exit(99);
                    }
                }

                startupSimulation();
            }
        });

        setup.setVisible(true);
    }
    /*
     * try{ updateSpeed = Integer.parseInt(JOptionPane.showInputDialog
     * ("What speed would you like to run at?",50)); }catch(Exception e){
     * updateSpeed = 50; } try{ numberOfOrders =
     * Integer.parseInt(JOptionPane.showInputDialog
     * ("How many orders should be processed?",10)); }catch(Exception e){
     * numberOfOrders = 10; } try{ numberOfItemsInWarehouse =
     * Integer.parseInt(JOptionPane.showInputDialog
     * ("How many items should be placed in the warehouse?",100));
     * }catch(Exception e){ numberOfItemsInWarehouse = 100; } try{
     * shelfHeight = Integer.parseInt(JOptionPane.showInputDialog
     * ("How many bins should be in each shelf?",1)); }catch(Exception e){
     * shelfHeight = 1; } try{ maxOrdersInProcess =
     * Integer.parseInt(JOptionPane.showInputDialog
     * ("How many orders should be processed at the same time?",1));
     * }catch(Exception e){ maxOrdersInProcess = 1; }
     *
```

```
         * //waitDialog.add(new
         * JLabel("Please wait while the simulation is initiated..."));
         * //waitDialog.setSize(400, 100); //waitDialog.setVisible(true);
         */

    }

    // results in order of: steps,

    public void sendEndToListeners() {
        System.out.println("Trying to run end listeners");
        for (BehaviorSpace space : listeners) {
            System.out.println("Running end listener");
        }
    }

    ArrayList<BehaviorSpace> listeners = new ArrayList<BehaviorSpace>();

    public void addEndListener(BehaviorSpace space) {
        System.out.println("Adding end listener");
        listeners.add(space);
    }

    /**
     * a method called every simulation tic by Timer so we can increment a
     * counter
     */
    public void tic() {
        steps++;
    }


/** public access to the orders */
    public static Hashtable<Integer, List<Integer>> getOrders() {
        return orders;
    }

    /** public access the orders */
    public static void setOrders(Hashtable<Integer, List<Integer>> orders) {
        Warehouse.orders = orders;
    }
```

```
/** get the itemLocations hashtable */
public static Hashtable<Integer, StorageBin> getItemLocations() {
	return itemLocations;
}

/** set the itemLocations hashtable */
public static void setItemLocations(
		Hashtable<Integer, StorageBin> itemLocations) {
	Warehouse.itemLocations = itemLocations;
}

/**
 * removes all orders from the order list and ordersInProcess list. This
 * means all orders are complete.
 */
public static void removeOrderFromLists(int orderNumber) {
	orders.remove((Object) orderNumber);
	ordersInProcess.remove((Object) orderNumber);
	System.out.println("ORDER " + orderNumber + " IS COMPLETE");
}

/** when a tote has an item added to it, it is removed from the order */
public static void removeItemFromOrder(int orderNumber, int itemID) {
	List<Integer> orderList = orders.get(orderNumber);
	if (orderList != null) {
		if (orderList.isEmpty()) {
			// the next time botherworker is called we will send the tote
			// away
			System.out.println("Order list is empty for order "
					+ orderNumber + ". This order is complete");

		} else {
			orderList.remove((Object) itemID);
			// logger.fine("removed item " + itemID +
			// " from order "+orderNumber);
		}
	}
}

/** gets the next item in the order */
public static int getNextItem(int orderNumber) {
	List orderList = orders.get(orderNumber);
	if (orderList.isEmpty()) {
		return -1;
```

```
        }
        return (Integer) orderList.get(0);
    }

    /** the picker needs to know which item is next in the order */
    public static int getNextItemAtStop(int orderNumber, ItemStop is) {
        // System.out.println(
        // "stop domain before selecting next item "+is.stopInfo.domain);
        int itemToReturn = -1;
        List<Integer> orderList = orders.get(orderNumber);
        // System.out.println(orderList);

        if (orderList != null) {
            if (orderList.isEmpty()) {
                itemToReturn = -1;
            } else {
                int nextItem = (Integer) orderList.get(0);
                if (is.stopInfo.domain.contains((Object) nextItem)) {
                    itemToReturn = nextItem;
                }
            }
        } else {
            itemToReturn = -1;
        }
        // System.out.println("Next item to pick --->> " + itemToReturn +
        // " stop domain "+is.stopInfo.domain);
        return itemToReturn;
    }

    /**
     * the whole list of items for the Item Stop. Had to make sure we returned a
     * copy of the items, not the items themselves
     */
    public static ArrayList<Integer> getItemsAtStop(int orderNumber, ItemStop is) {
        List orderList = orders.get(orderNumber);
        ArrayList<Integer> ordersForThisStop = new ArrayList<Integer>();
        if (orderList != null) {
            if (orderList.isEmpty()) {
                return ordersForThisStop; // will be empty
            }

            for (int i = 0; i < orderList.size(); i++) {
```

```
                int nextItem = (Integer) orderList.get(i);
                if (is.stopInfo.domain.contains(nextItem)) {
                    ordersForThisStop.add(new Integer(nextItem));
                }
            }
            return ordersForThisStop;
        }
        return null; // only sends back null if warehouse orders is null
}

/** gets the location of the bin that has this item with itemID */
public static Point getItemLocation(int itemID) {
        StorageBin bin = getItemLocations().get(itemID);
        Point itemLocation = bin.getLocation();
        return itemLocation;
}

/** returns the item stop agent at the specified location */
public ItemStop getItemStopAtLocation(Point loc) {
        ItemStop stop = null;
        for (int x = 0; x < warehouseLayout.itemStops.size(); x++) {
            if (warehouseLayout.itemStops.get(x).segmentInfo
                    .getSegmentLocation().equals(loc)) {
                stop = warehouseLayout.itemStops.get(x);
                // System.out.println("found a stop");
                x = warehouseLayout.itemStops.size();
            }
        }
        return stop;
}

/**
 * this is called exactly one time per simulation to log the staring step
 * from when the first order tote is launched
 */
public static void recordStart() {
        Statement st;
        ResultSet rs = null;
        // store start step in the database
        startStep = steps;
        // store the timestamp and get the runID from the database
        long timeMillis = System.currentTimeMillis();
        Timestamp startTime = new Timestamp(timeMillis);
        try {
```

```
            st = conn.createStatement();
            String sql = "insert into xruns (rundate) values ('" + startTime
                        + "')";
            System.out.println(sql);
            st.executeQuery(sql);
        } catch (SQLException e) {
            e.printStackTrace();
            System.out
                        .println("Could not start a simulation run because couuld not record start time in database");
            System.exit(99);
        }
        // get the runid
        try {
            st = conn.createStatement();
            rs = st.executeQuery("select id from xruns where rundate='"
                        + startTime + "'");
            rs.next();
            runID = rs.getInt("id");
        } catch (SQLException e) {
            e.printStackTrace();
            System.out
                        .println("Could not start a simulation run because couuld not get run id from the database");
            System.exit(99);
        }

}

static String setID = null;

/**
 * this is called exactly one time per simulation to log the ending step
 * when all orders are filled
 */
public static void recordEnd() {
        endStep = steps;
        // store the timestamp and get the runID from the database
        long timeMillis = System.currentTimeMillis();
        Date startTime = new Date(timeMillis);
        try {
            Statement st = conn.createStatement();
            String sql = "update xruns set steps = " + (endStep - startStep)
                        + " where id=" + runID;
```

```
System.out.println(sql);
st.executeQuery(sql);
// number of pickers is number of item stops

sql = "update xruns set pickers = " + numberOfPickers
            + " ,stations =" + numberOfItemStops + " where id=" + runID;
System.out.println(sql);
st.executeQuery(sql);

sql = "update xruns set orders = " + numberOfOrders + " ,items ="
            + numberOfItemsInWarehouse + " where id=" + runID;
System.out.println(sql);
st.executeQuery(sql);

sql = "update xruns set systemMax = " + maxOrdersInProcess
            + ", speed=" + updateSpeed + ",shelfheight=" + shelfHeight
            + "  where id=" + runID;
System.out.println(sql);
st.executeQuery(sql);

if (pickingMode == zoneSingle || pickingMode == zoneMulti) {
      sql = "update xruns set zone = true where id=" + runID;
} else {
      sql = "update xruns set zone = false where id=" + runID;
}
System.out.println(sql);
st.executeQuery(sql);

if (pickingMode == zoneMulti || pickingMode == discreteMulti) {
      sql = "update xruns set multi = true where id=" + runID;
} else {
      sql = "update xruns set multi = false where id=" + runID;
}

System.out.println(sql);
st.executeQuery(sql);

sql = "update xruns set itemsorting = '" + itemSetup
            + "',distributeEvenly = " + distributeItemsEvenly
            + " where id=" + runID;
System.out.println(sql);
st.executeQuery(sql);

sql = "update xruns set activestations = " + activeStations
```

```
                + " where id=" + runID;
        System.out.println(sql);
        st.executeQuery(sql);

        String pathAdd = "";
        if (layoutFileName == null || layoutFileName == "Demo Test") {
            layoutFileName = "default";
            if (usingPath) {
                pathAdd = " with path";
            }
        }

        // if the layout file has a path, remove the path for storing in
        // database
        File layout = new File(layoutFileName);
        layoutFileName = layout.getName();
        sql = "update xruns set layout = '" + layoutFileName
                + "' where id=" + runID;
        System.out.println(sql);
        st.executeQuery(sql);

        sql = "update xruns set graphics = " + showWarehouseGraphics
                + " where id=" + runID;
        System.out.println(sql);
        st.executeQuery(sql);

        String notes = "stops " + stopListString + pathAdd;
        sql = "update xruns set notes = '" + notes + "' where id=" + runID;
        System.out.println(sql);
        st.executeQuery(sql);
        if (setID != null) {

            sql = "update xruns set RUNSETID = '" + setID + "' where id="
                    + runID;
            System.out.println(sql);
            st.executeQuery(sql);
        }

    } catch (SQLException e) {
        e.printStackTrace();
        System.out
                .println("FAILURE: Could not record end data in database");
```

```
                System.exit(99);
            }

            // now it will end the simulation
            System.out
                    .println("Your simulation is now ready to exit for the enjoyment of the world.");
            System.exit(0);

        }

}
```

WarehouseLayout.java

- ```
  * Warehouse layout initializes the areas and systems of the warehouse.
  * It can be done manually or by reading a file from the Warehouse Layout Designer.
  * This uses the Warehouse.grid for the layout template. The constructor does a lot
  * of the work and needs to be called before any schedulers are started.
  * (c) 2012 Team 64 New Mexico Supercomputing Challenge
  */
  package team64.madkit;

  import java.awt.Color;
  import java.awt.Point;
  import java.awt.Rectangle;
  import java.io.File;
  import java.io.FileInputStream;
  import java.io.ObjectInputStream;
  import java.util.ArrayList;
  import java.util.Hashtable;
  import java.util.List;
  import java.util.Random;
  import java.util.zip.GZIPInputStream;

  import javax.swing.JFileChooser;
  import javax.swing.filechooser.FileFilter;

  import madkit.kernel.AbstractAgent;
  import team64.madkit.messages.SetLocationMessage;

  /**
   * @author team64 New Mexico Supercomputing Challenge
   * @2012
   *
   */
  public class WarehouseLayout extends AbstractAgent  {

      /**
       * size of the warehouse in grid units (basically a grid unit holds one
       * tote, one conveyor segment, one worker, or a tote + conveyor segment)
       */
      protected  int warehouseWidth;
      protected  int warehouseHeight;
      protected  Random r=new Random();
  ```
  ```

```
        protected  int lunchroomWidth;
        protected  int lunchroomDepth;
        protected  Point lunchroomLocation;
        protected  Point entranceLocation;
        protected  Point toteLauncherLocation;

        protected Hashtable <Integer,StorageBin> itemLocations;
        protected ArrayList <StorageBin> allBins = new ArrayList<StorageBin>();
        protected ArrayList <ConveyorSegment> conveyorSegments = new ArrayList<ConveyorSegment>();
        protected ArrayList <ItemStop> itemStops = new ArrayList<ItemStop>(); //a separate list of just the
item stops

        protected Point demoStop1 = new Point(4,4);
        protected Point demoStop2 = new Point (10,4);
        protected Point demoStop3 = new Point(4,11);
        protected Point demoStop4 = new Point (14,11);
        protected boolean useStop1 = true;
        protected boolean useStop2 = true;
        protected boolean useStop3 = false;
        protected boolean useStop4 = false;


        /** Generic Constructor */
        void WarehouseLayout() {

        };

        /** only needed so that this can be launched and launch the warehouse agents in the layout */
        public void activate() {

        }

        /** Start layout with a layout file or use defaults if no file */
        public  void initialize(boolean inputFile) {

                if (Warehouse.layoutFileName.equals("Demo Test")) {
                        //use test defaults
                        if (Warehouse.usingPath) {
                                setupWarehouseWithTestDefaultsAndPathway();
                        } else {
                                setupWarehouseWithTestDefaults();
                        }
                }else if (Warehouse.layoutFileName.equals("Warehouse 1")){
```

```
                Warehouse.layoutFileName="large3.T64";
            }
            else if (Warehouse.layoutFileName.equals("Warehouse 2")){
                Warehouse.layoutFileName="large3.T64";
            }
            else if (Warehouse.layoutFileName.equals("Warehouse 3")){
                Warehouse.layoutFileName="large3.T64";
            }
            if (!Warehouse.layoutFileName.equals("Demo Test")){
                importGrid(Warehouse.layoutFileName);
            }

            System.out.println("Initialized Warehouse Layout");
        }

        public void setupGrid(){
            Warehouse.warehouseWidth = getWarehouseWidth();
            Warehouse.warehouseHeight = getWarehouseHeight();
            Warehouse.getGrid();
            for (int column = 0; column < warehouseWidth; column++) {
                for (int row = 0; row < warehouseHeight; row++) {
                    Warehouse.getGrid()[column][row] = new GridUnit();
                }
            }
        }

        public void placeShelf(Point p){
            for (int i = 0; i < Warehouse.shelfHeight; i++){
                StorageBin bin = new StorageBin(100, p);
                Warehouse.getGrid()[p.x][p.y].addBin(bin);
                bin.setItemId(-1);//indicate no item in the bin yet
                allBins.add(bin);
            }
        }

        /** returns whether or not the point is in the lunchroom */
        public  boolean inLunchRoom(Point p){
            //creates an arbitrary rectangle that can be checked to contain a point
            Rectangle lunch =
    new Rectangle(lunchroomLocation.x,lunchroomLocation.y,lunchroomWidth,lunchroomDepth);
            if (lunch.contains(p)){
                return true;
```

```
        }
        return false;
    }

    /** Creates the conveyor segments in specified locations and directions */

    private List<ConveyorSegment> setupConveyorSystem() {
        List<ConveyorSegment> conveyorSegments = null;

        return conveyorSegments;
    }


    /** sets up the default conveyor system */
    private  void createDefaultConveyorCircle() {

        // create, launch, and configure a bunch of conveyor segments
        //conveyor segment needs to know which one is its NEXT link
        //public static int right=0;
        //public static int up=1;
        //public static int left=2;
        //public static int down=3;

        Point location = new Point (0,0);
        for (int x = 2; x < 20; x++) {
            location = new Point(x, 3);
            addConveyorSegment(ConveyorSegment.right,location);
            location = new Point(x, 10);
            addConveyorSegment(ConveyorSegment.left,location);
        }

        location = new Point(20, 10);
        addConveyorSegment(ConveyorSegment.left,location);

        location = new Point(1,3);
        addConveyorSegment(ConveyorSegment.right,location);

        for (int y = 4; y < 10; y++){

            location = new Point(20, y);
            addConveyorSegment(ConveyorSegment.up,location);

            if (y!=8&&y!=7&&y!=6){
                location = new Point(1, y);
```

```
                    addConveyorSegment(ConveyorSegment.down,location);
                }
            }

            location = new Point(20, 3);
            addConveyorSegment(ConveyorSegment.up,location);

            location = new Point(1, 10);
            addConveyorSegment(ConveyorSegment.down,location);

            //for all the conveyor segments, automatically find the next
            //segment in the flow direction


        }

        /** instantiates and launches a conveyor segment at the specified location */
        protected void addConveyorSegment(int direction,Point location){
            ConveyorSegment cs = new ConveyorSegment(direction);
            cs.segmentInfo.setSegmentLocation(location);
            cs.segmentInfo.setNextSegmentLocation(cs.findNextLocation());
            launchAgent(cs);
            //This sends a message to the warehouse to add this to the grid
            cs.sendMessage("warehouse","conveyorsystem","warehouse",
new SetLocationMessage(new Point(cs.segmentInfo.getSegmentLocation().x,cs.segmentInfo.getSegmentLocation(
).y)));
            conveyorSegments.add(cs);
        }


        /** arrange the storage bins for default warehouse setup */
        private void placeBinsWhereOpen(){
            for (int i = 0; i < warehouseWidth; i=i+1){
                for (int p = 0; p < warehouseHeight; p++){
                    if (Warehouse.getGrid()[i][p].occupants.isEmpty()){
                        try{
                            if(!inLunchRoom(new Point(i,p)) && Warehouse.getGrid()[i][p].path==false){
                                if ((Warehouse.getGrid()[i+1][p].occupants.isEmpty()&&
                                        Warehouse.getGrid()[i-1][p].occupants.isEmpty()&&
                                        Warehouse.getGrid()[i][p+1].occupants.isEmpty()&&
                                        Warehouse.getGrid()[i][p-1].occupants.isEmpty()&&
                                        Warehouse.getGrid()[i+1][p-1].occupants.isEmpty()&&
```

```
                                    Warehouse.getGrid()[i-1][p-1].occupants.isEmpty()&&
                                    Warehouse.getGrid()[i-1][p+1].occupants.isEmpty()&&
                                    Warehouse.getGrid()[i+1][p+1].occupants.isEmpty())){
                                if ((Warehouse.getGrid()[i+1][p].bins.isEmpty()&&
                                        Warehouse.getGrid()[i-1][p].bins.isEmpty()
                                        )){
                                    if ((Warehouse.getGrid()[i][p+1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i][p-1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i+1][p-1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i-1][p-1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i-1][p+1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i+1][p+1].bins.isEmpty())){

                                        placeShelf(new Point (i,p));
                                    }
                                }else{
                                    if ((Warehouse.getGrid()[i][p+1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i][p-1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i+1][p-1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i-1][p-1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i-1][p+1].bins.isEmpty()&&
                                            Warehouse.getGrid()[i+1][p+1].bins.isEmpty())){

                                        StorageBin bin = new StorageBin(100, new Point (i,p));
                                        Warehouse.getGrid()[i][p].addBin(bin);

                                        bin.setItemId(-1);//indicate no item in the bin yet
                                        placeShelf(new Point (i,p));
                                    }

                                }
                            }
                        }
                    } catch (ArrayIndexOutOfBoundsException e){

                    }
                }
            }
        }
    }

    /** Creates a path of layout that cannot contain shelves */
    private void makeAPathThroughShelves(Point start, Point end){
```

```
        //Lets use A* (why not)
        Router pathGenerator = new Router();
        List <Point> path = pathGenerator.getBestPath(start,end);

        for (int p = 0; p < path.size(); p++){
            int x = path.get(p).x;
            int y = path.get(p).y;
            if (!Warehouse.getGrid()[x][y].isBlocked()){
                Warehouse.getGrid()[x][y].addPath();
            }
        }
}

/** import warehouse layout with a filename */
public void importGrid(String filename){
  try {
            String state=null;
            ArrayList<String> s1= new ArrayList<String>(200);
            int sx=0;
            int sy=0;
            System.out.println(filename);
            FileInputStream fis = new FileInputStream(filename);
            GZIPInputStream gzis = new GZIPInputStream(fis);
            ObjectInputStream in = new ObjectInputStream(gzis);
            s1=(ArrayList<String>) in.readObject();
            int cx=(Integer) in.readObject();
            int cy=(Integer) in.readObject();
            in.close();
            int maxX = 00;
            int maxY = 00;
            for(String s: s1){
                sx=(int) Double.parseDouble(s.substring(0, 4));
                sy=(int) Double.parseDouble(s.substring(5, 9));
                if (sx>maxX){
                    maxX = sx+0;
                }
                if (sy>maxY){
                    maxY = sy+0;
                }
            }

            setWarehouseWidth(maxX+2);
```

```
setWarehouseHeight(maxY+2);
setupGrid();
entranceLocation = new Point(0, warehouseHeight / 2);
for(String s: s1){
        sx=(int) Double.parseDouble(s.substring(0, 4));
        sy=(int) Double.parseDouble(s.substring(5, 9));

        if(s.substring(10,15).equals("start")){
                state="images/BoxStart.png";
                System.out.println("miracle");
                conveyorSegments.add(this.initiateBasicSpawn(new Point(sx,sy)));

        }
        else if(s.substring(10, 15).equals("end00")){
                state="images/BoxEnd.png";
                conveyorSegments.add(this.initiateBasicEnd(new Point(sx,sy)));
        }
        else if(s.substring(10, 15).equals("cn090")){
                state="images/ConveyerStraight.png";
                this.addConveyorSegment(3, new Point(sx,sy));

        }
        else if(s.substring(10, 15).equals("cn180")){
                state="images/ConveyerLeft.png";
                this.addConveyorSegment(2, new Point(sx,sy));
        }
        else if(s.substring(10, 15).equals("cn270")){
                state="images/ConveyerBack.png";
                this.addConveyorSegment(1, new Point(sx,sy));
        }
        else if(s.substring(10, 15).equals("cn360")){
                state="images/ConveyerRight.png";
                this.addConveyorSegment(0, new Point(sx,sy));
        }
        else if(s.substring(10, 15).equals("iddwn")){
                state="images/itemdrop-down.png";
                conveyorSegments.add(this.initiateBasicStop(new Point(sx,sy), 1));
        }
        else if(s.substring(10, 15).equals("idlft")){
                state="images/itemdrop-left.png";
                conveyorSegments.add(this.initiateBasicStop(new Point(sx,sy), 2));
        }
        else if(s.substring(10, 15).equals("idrgt")){
                state="images/itemdrop-right.png";
```

```
                    conveyorSegments.add(this.initiateBasicStop(new Point(sx,sy), 0));
                }
                else if(s.substring(10, 15).equals("idup0")){
                    state="images/itemdrop-up.png";
                    conveyorSegments.add(this.initiateBasicStop(new Point(sx,sy), 3));
                }
                else if(s.substring(10, 15).equals("shelf")){
                    state="images/Shelf.png";
                    placeShelf(new Point (sx,sy));

                }
                else {
                    state="images/ConveyerIntersection.png";
                }



            }

        }
        catch (Exception e) {
            System.out.println(e);
            //
            e.printStackTrace();
        }
}


public void importGrid(){
    JFileChooser chooser=new JFileChooser();
    FileFilter filter = new FileFilter(){

        @Override
        public boolean accept(File f) {
             String ext = "";
            String s = f.getName();
            int i = s.lastIndexOf('.');

            if (i > 0 &&  i < s.length() - 1) {
                ext = s.substring(i+1).toLowerCase();
            }
           if (ext.equals("t64")){
```

```
                return true;
            }
            return false;
        }

        @Override
        public String getDescription() {
            // TODO Auto-generated method stub
            return "*.t64";
        }

};
chooser.setFileFilter(filter);
chooser.showOpenDialog(null);
String filename=chooser.getSelectedFile().getPath();
try {
        String state=null;
        ArrayList<String> s1= new ArrayList<String>(200);
        int sx=0;
        int sy=0;
        FileInputStream fis = new FileInputStream(filename);
        GZIPInputStream gzis = new GZIPInputStream(fis);
        ObjectInputStream in = new ObjectInputStream(gzis);
        s1=(ArrayList<String>) in.readObject();
        int cx=(Integer) in.readObject();
        int cy=(Integer) in.readObject();
        in.close();
        int maxX = 00;
        int maxY = 00;
        for(String s: s1){
            sx=(int) Double.parseDouble(s.substring(0, 4));
            sy=(int) Double.parseDouble(s.substring(5, 9));
            if (sx>maxX){
                maxX = sx+0;
            }
            if (sy>maxY){
                maxY = sy+0;
            }
        }

        setWarehouseWidth(maxX+2);
        setWarehouseHeight(maxY+2);
        setupGrid();
        entranceLocation = new Point(0, warehouseHeight / 2);
```

```
for(String s: s1){
        sx=(int) Double.parseDouble(s.substring(0, 4));
        sy=(int) Double.parseDouble(s.substring(5, 9));

        if(s.substring(10,15).equals("start")){
                state="images/BoxStart.png";
                System.out.println("miracle");
                conveyorSegments.add(this.initiateBasicSpawn(new Point(sx,sy)));

        }
        else if(s.substring(10, 15).equals("end00")){
                state="images/BoxEnd.png";
                conveyorSegments.add(this.initiateBasicEnd(new Point(sx,sy)));
        }
        else if(s.substring(10, 15).equals("cn090")){
                state="images/ConveyerStraight.png";
                this.addConveyorSegment(3, new Point(sx,sy));

        }
        else if(s.substring(10, 15).equals("cn180")){
                state="images/ConveyerLeft.png";
                this.addConveyorSegment(2, new Point(sx,sy));
        }
        else if(s.substring(10, 15).equals("cn270")){
                state="images/ConveyerBack.png";
                this.addConveyorSegment(1, new Point(sx,sy));
        }
        else if(s.substring(10, 15).equals("cn360")){
                state="images/ConveyerRight.png";
                this.addConveyorSegment(0, new Point(sx,sy));
        }
        else if(s.substring(10, 15).equals("iddwn")){
                state="images/itemdrop-down.png";
                conveyorSegments.add(this.initiateBasicStop(new Point(sx,sy), 1));
        }
        else if(s.substring(10, 15).equals("idlft")){
                state="images/itemdrop-left.png";
                conveyorSegments.add(this.initiateBasicStop(new Point(sx,sy), 2));
        }
        else if(s.substring(10, 15).equals("idrgt")){
                state="images/itemdrop-right.png";
                conveyorSegments.add(this.initiateBasicStop(new Point(sx,sy), 0));
```

```
                    }
                    else if(s.substring(10, 15).equals("idup0")){
                        state="images/itemdrop-up.png";
                        conveyorSegments.add(this.initiateBasicStop(new Point(sx,sy), 3));
                    }
                    else if(s.substring(10, 15).equals("shelf")){
                        state="images/Shelf.png";
                        placeShelf(new Point (sx,sy));

                    }
                    else {
                        state="images/ConveyerIntersection.png";
                    }



                }

            }
            catch (Exception e) {
                System.out.println(e);
                //
                e.printStackTrace();
            }
        }


    /** Launches the item stops (where the totes stop to get filled) */
    protected  ItemStop initiateBasicStop(Point p, int direction){
        ItemStop stop = new ItemStop(direction);
        itemStops.add(stop);
        stop.segmentInfo.setSegmentLocation(p);
        stop.segmentInfo.setNextLocation(stop.findNextLocation());
        //random color but not too light and not too dark
        stop.stopInfo.setDomainColor(new Color(r.nextInt(130)+50,r.nextInt(130)+50,r.nextInt(130)+50));
        launchAgent(stop);

        //This sends a message to the warehouse to add this to the grid
        stop.sendMessage("warehouse","conveyorsystem","warehouse",
new SetLocationMessage(new Point(stop.segmentInfo.getSegmentLocation().x,stop.segmentInfo.getSegm
entLocation().y)));
        conveyorSegments.add(stop); //add the stop so that we can generically check for it
```

```
            return stop;
        }


        protected  End initiateBasicEnd(Point p){
            End end = new End(3);
            end.segmentInfo.setSegmentLocation(p);
            //ends do not have a next loctation...?
            //Since this is an "End" segment, we really need the Start segment location for the next
            // NOOOO -- end.segmentInfo.setNextLocation(end.findNextLocation());
            launchAgent(end);
            //get the "next location" for the end segment based on what the End class does to get this
            end.getNextLocation();
            //This sends a message to the warehouse to add this to the grid. There is a chance that the
            //spawner does not exist yet, but if so the next location will remain null and we will check
            //later for this information
            end.sendMessage("warehouse","conveyorsystem","warehouse",
new SetLocationMessage(new Point(end.segmentInfo.getSegmentLocation().x,end.segmentInfo.getSegmentLocati
on().y)));
            conveyorSegments.add(end); //add the segment so that we can generically check for it
            return end;
        }

        protected  ToteSpawn initiateBasicSpawn(Point p){
            System.out.println("Spawner Made!");
            ToteSpawn spawn = new ToteSpawn(3);
            spawn.segmentInfo.setSegmentLocation(p);
            List <Point> neighbors = Warehouse.getAdjacent(p);
            for (int i = 0; i < neighbors.size(); i ++){
                if (!Warehouse.getGrid()[neighbors.get(i).x][neighbors.get(i).y].occupants.isEmpty()){
                    if (Warehouse.getGrid()[neighbors.get(i).x]
[neighbors.get(i).y].occupants.get(0).getRole().equals("segment")){
                        spawn.segmentInfo.setNextLocation(neighbors.get(i));
                    }
                }
            }

            launchAgent(spawn);

            //This sends a message to the warehouse to add this to the grid
            spawn.sendMessage("warehouse","conveyorsystem","warehouse",
new SetLocationMessage(new Point(spawn.segmentInfo.getSegmentLocation().x,spawn.segmentInfo.getSegment
```

```
Location().y)));
            conveyorSegments.add(spawn); //add the segment so that we can generically check for it
            return spawn;
        }



        /** convenience method to check if a location point is OK */
        private boolean checkBounds(Point location) {
            if ((location.x>warehouseWidth-1) || (location.x<0)) {
                return false;
            }
            if ((location.y>warehouseHeight-1) || (location.y<0)) {
                return false;               }

            return true;
        }


        private  void setupWarehouseWithTestDefaults() {
            System.out.println("warehouse using test defaults");
            setWarehouseWidth (25);
            setWarehouseHeight(25);
            setupGrid();
            entranceLocation = new Point(0, warehouseHeight / 2);
            lunchroomWidth = 4;
            lunchroomDepth = 4;
            lunchroomLocation = new Point(warehouseWidth- lunchroomWidth, 0);

            toteLauncherLocation = new Point (1,3);

            createDefaultConveyorCircle();


            useStop1 = Warehouse.useDemoStop1;
            useStop2 = Warehouse.useDemoStop2;
            useStop3 = Warehouse.useDemoStop3;
            useStop4 = Warehouse.useDemoStop4;

            if (useStop1) {
                ItemStop itemStop = initiateBasicStop(new Point(4,4), 3);
                conveyorSegments.add(itemStop);
            }
```

```
        if (useStop2) {
            ItemStop itemStop2 = initiateBasicStop(new Point(10,4), 3);
            conveyorSegments.add(itemStop2);
        }

        if (useStop3) {
            ItemStop itemStop3 = initiateBasicStop(new Point(4,11), 3);
            conveyorSegments.add(itemStop3);
        }


        if (useStop4) {
            ItemStop itemStop4 = initiateBasicStop(new Point(14,11), 3);
            conveyorSegments.add(itemStop4);
        }


        ToteSpawn spawn = initiateBasicSpawn(new Point(1,6));
        conveyorSegments.add(spawn);

        End end = initiateBasicEnd(new Point(1,8));
        conveyorSegments.add(end);

        placeBinsWhereOpen();
    }

    private  void setupWarehouseWithTestDefaultsAndPathway() {
        System.out.println("warehouse using test defaults and a path through shelves");
        setWarehouseWidth (25);
        setWarehouseHeight(25);
        setupGrid();
        entranceLocation = new Point(0, warehouseHeight / 2);
        lunchroomWidth = 4;
        lunchroomDepth = 4;
        lunchroomLocation = new Point(warehouseWidth- lunchroomWidth, 0);

        toteLauncherLocation = new Point (1,3);

        createDefaultConveyorCircle();


        useStop1 = Warehouse.useDemoStop1;
```

```
        useStop2 = Warehouse.useDemoStop2;
        useStop3 = Warehouse.useDemoStop3;
        useStop4 = Warehouse.useDemoStop4;

        if (useStop1) {
            ItemStop itemStop = initiateBasicStop(new Point(4,4), 3);
            conveyorSegments.add(itemStop);
        }

        if (useStop2) {
            ItemStop itemStop2 = initiateBasicStop(new Point(10,4), 3);
            conveyorSegments.add(itemStop2);
        }

        if (useStop3) {
            ItemStop itemStop3 = initiateBasicStop(new Point(4,11), 3);
            conveyorSegments.add(itemStop3);
        }


        if (useStop4) {
            ItemStop itemStop4 = initiateBasicStop(new Point(14,11), 3);
            conveyorSegments.add(itemStop4);
        }


        End end = initiateBasicEnd(new Point(1,8));
        conveyorSegments.add(end);

        ToteSpawn spawn = initiateBasicSpawn(new Point(1,6));
        conveyorSegments.add(spawn);

        Point pathStart = new Point(10,12);
        Point pathEnd = new Point(10,22);


        //place path where shelves cannot be put
        makeAPathThroughShelves(pathStart,pathEnd);

        placeBinsWhereOpen();

    }

    /** convenience method for warehouse setup */
```

```
    private  void setWarehouseHeight(int wh) {
        warehouseHeight = wh;

    }

    /** convenience method for warehouse setup */
    private void setWarehouseWidth(int ww) {
        warehouseWidth = ww;

    }


    /** convenience method for warehouse setup */
    public int getWarehouseWidth() {
        return warehouseWidth;
    }

    /** convenience method for warehouse setup */
    public int getWarehouseHeight() {
        return warehouseHeight;
    }

    /** This reads the Warehouse Layout save file into the various layout items */
    private void readInputFile(String inputFileName) {
        setupGrid();
        //parse the file and do similar actions to setting up default warehouse
        //do not launch the conveyor agents here, but set all their grid locations
        //add all conveyor type agents to the conveyorSegments list

    }

}
```

SetupFrame.java (used by Main in Warehouse)

```
package team64.madkit;

import javax.swing.JFrame;

public class SetupFrame extends JFrame {

}
```

ParameterPanel.java

```
/** Panel to change simulation defaults on startup */
package team64.madkit;

/**
 * @author team64 New Mexico Supercomputing Challenge
 * @2012
 *
 */

import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFormattedTextField;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class ParameterPanel extends JPanel {
	protected JFormattedTextField txtMaxOrdersInProcess;
	protected JFormattedTextField txtUpdateSpeed;
	protected JFormattedTextField txtShelfHeight;
	protected JFormattedTextField textFieldOrders;
	protected JComboBox comboBoxItems; //number of items in the warehouse
	protected JComboBox comboBox; //warehouse layout
	protected JCheckBox chckbxZone;
	protected JCheckBox chckbxMulti;
	protected JCheckBox chkRandomSorting;
	protected JCheckBox chckbxUsePopularity;
	protected JCheckBox chkLoc1;
	protected JCheckBox chkLoc2;
	protected JCheckBox chkLoc3;
	protected JCheckBox chkLoc4;
	private boolean enableChange=false;
	private JLabel lblNewLabel_7;


	public ParameterPanel() {
```

```
setLayout(null);

JLabel lblNewLabel = new JLabel("Warehouse Simulation Parameters");
lblNewLabel.setForeground(Color.BLUE);
lblNewLabel.setFont(new Font("Arial", Font.PLAIN, 18));
lblNewLabel.setBounds(10, 0, 353, 38);
add(lblNewLabel);

comboBox = new JComboBox();

comboBox.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {
		if (enableChange) {
			if (!comboBox.getSelectedItem().equals("Demo Test")) {
				chkLoc1.setEnabled(false);
				chkLoc2.setEnabled(false);
				chkLoc3.setEnabled(false);
				chkLoc4.setEnabled(false);
			}else {
				chkLoc1.setEnabled(true);
				chkLoc2.setEnabled(true);
				chkLoc3.setEnabled(true);
				chkLoc4.setEnabled(true);
			}
		}

	}
});
comboBox.setToolTipText("select a pre-configured warehouse or use default for the test layout");
comboBox.setBounds(20, 38, 247, 32);
comboBox.addItem(new String("Demo Test"));
comboBox.addItem(new String("Warehouse 1"));
comboBox.addItem(new String("Warehouse 2"));
comboBox.addItem(new String("Warehouse 3"));
comboBox.addItem(new String("Browse..."));
add(comboBox);

JLabel lblNewLabel_1 = new JLabel("Warehouse Layout");
lblNewLabel_1.setBounds(277, 43, 123, 22);
add(lblNewLabel_1);

chckbxZone = new JCheckBox("Zone Picking");
chckbxZone.setBounds(20, 316, 133, 23);
chckbxZone.setSelected(true);
```

```
add(chckbxZone);

 comboBoxItems = new JComboBox();
comboBoxItems.addItem(new Integer(100));
comboBoxItems.addItem(new Integer(200));
comboBoxItems.addItem(new Integer(300));
comboBoxItems.addItem(new Integer(400));
comboBoxItems.addItem(new Integer(500));
comboBoxItems.addItem(new Integer(1000));
comboBoxItems.addItem(new Integer(1200));
comboBoxItems.addItem(new Integer(2000));
comboBoxItems.addItem(new Integer(4000));
comboBoxItems.addItem(new Integer(10000));

comboBoxItems.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {
	}
});
comboBoxItems.setBounds(20, 103, 123, 32);
add(comboBoxItems);

JLabel lblNewLabel_2 = new JLabel("Items in Warehouse");
lblNewLabel_2.setBounds(30, 138, 123, 22);
add(lblNewLabel_2);

JLabel lblOrdersToProcess = new JLabel("Orders To Process");
lblOrdersToProcess.setBounds(214, 138, 123, 22);
add(lblOrdersToProcess);

txtMaxOrdersInProcess = new JFormattedTextField(2);
txtMaxOrdersInProcess.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {
	}
});
txtMaxOrdersInProcess.setBounds(30, 171, 58, 30);
add(txtMaxOrdersInProcess);
txtMaxOrdersInProcess.setColumns(10);

JLabel lblNewLabel_3 = new JLabel("Maximum Orders In Process at the Same Time");
lblNewLabel_3.setToolTipText("Orders in the conveyor system at one time");
lblNewLabel_3.setBounds(30, 203, 302, 22);
add(lblNewLabel_3);
```

```
txtUpdateSpeed = new JFormattedTextField(50);
txtUpdateSpeed.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {
	}
});
txtUpdateSpeed.setColumns(10);
txtUpdateSpeed.setBounds(31, 247, 58, 32);
add(txtUpdateSpeed);

JLabel lblUpdateSpeedms = new JLabel("Update Speed (ms)");
lblUpdateSpeedms.setToolTipText("Orders in the conveyor system at one time");
lblUpdateSpeedms.setBounds(99, 255, 161, 22);
add(lblUpdateSpeedms);

txtShelfHeight = new JFormattedTextField(1);
txtShelfHeight.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {
	}
});
txtShelfHeight.setColumns(10);
txtShelfHeight.setBounds(277, 244, 58, 32);
add(txtShelfHeight);

JLabel lblNumberOfShelves = new JLabel("Number of Storage Bins Per Storage Shelf");
lblNumberOfShelves.setToolTipText("Orders in the conveyor system at one time");
lblNumberOfShelves.setBounds(287, 287, 269, 22);
add(lblNumberOfShelves);

chckbxMulti = new JCheckBox("Multiple Item Pick Per Order");
chckbxMulti.setBounds(20, 348, 230, 23);
add(chckbxMulti);

chkRandomSorting = new JCheckBox("Random Sorting on Shelves");
chkRandomSorting.setSelected(true);
chkRandomSorting.setBounds(277, 316, 256, 23);
add(chkRandomSorting);

chckbxUsePopularity = new JCheckBox("Use Item Popularity to Sort");
chckbxUsePopularity.setBounds(277, 348, 247, 23);
add(chckbxUsePopularity);

JLabel lblNewLabel_4 = new JLabel("Conveyor runs 2x worker walking speed");
lblNewLabel_4.setBounds(21, 290, 246, 14);
```

```
add(lblNewLabel_4);

textFieldOrders = new JFormattedTextField(10);
textFieldOrders.setColumns(10);
textFieldOrders.setBounds(204, 103, 112, 32);
add(textFieldOrders);

JLabel txt = new JLabel("Close this Panel afer you set the parameters to start the simulation");
txt.setForeground(Color.RED);
txt.setFont(new Font("Arial", Font.BOLD, 14));
txt.setBounds(30, 378, 503, 47);
add(txt);

JLabel lblNewLabel_5 = new JLabel("Demo Picking Stations");
lblNewLabel_5.setFont(new Font("Arial", Font.PLAIN, 12));
lblNewLabel_5.setBounds(376, 120, 142, 14);
add(lblNewLabel_5);

JLabel lblNewLabel_6 = new JLabel("(only applies to Demo Test layout)");
lblNewLabel_6.setFont(new Font("Arial", Font.ITALIC, 12));
lblNewLabel_6.setBounds(347, 138, 206, 14);
add(lblNewLabel_6);

chkLoc1 = new JCheckBox("(4,4)");
chkLoc1.setSelected(true);
chkLoc1.setBounds(350, 165, 58, 23);
add(chkLoc1);

chkLoc2 = new JCheckBox("(10,4)");
chkLoc2.setSelected(false);
chkLoc2.setBounds(347, 202, 60, 23);
add(chkLoc2);

chkLoc4 = new JCheckBox("(14,11)");
chkLoc4.setSelected(true);
chkLoc4.setBounds(433, 202, 85, 23);
add(chkLoc4);

chkLoc3 = new JCheckBox("(4,11)");
chkLoc3.setSelected(false);
chkLoc3.setBounds(433, 165, 85, 23);
add(chkLoc3);
```

```
            lblNewLabel_7 = new JLabel("If you select \"Browse...\" you will be able to select a layout after closing this panel.");
            lblNewLabel_7.setForeground(Color.BLUE);
            lblNewLabel_7.setBounds(30, 76, 503, 14);
            add(lblNewLabel_7);
            enableChange=true;
        }
}
```

Router.java

```
/*
 *Router.java is class based on A* (A Star) which
 *returns the best path from a start point to an end point
 * (c) 2012 Team 64 New Mexico Supercomputing Challenge
 */
package team64.madkit;

import java.awt.Point;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.logging.Level;

import javax.swing.JFrame;

import madkit.kernel.Agent;
import madkit.kernel.Message;
import team64.madkit.messages.RequestStatusMessage;
import team64.madkit.messages.StatusMessage;
import team64.madkit.notused.DirectionQuery;
import team64.madkit.notused.Directions;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class Router{

	/** overlay the grid to calculate paths */

	private List<WarehouseGridOverlay> opened = new ArrayList<WarehouseGridOverlay>();
	private List<WarehouseGridOverlay> closed = new ArrayList<WarehouseGridOverlay>();
	private List<WarehouseGridOverlay> bestList = new ArrayList<WarehouseGridOverlay>();

	/** The overlay uses same coordinates as the warehouse grid but
	 * contains information needed to find the path
	 */
	WarehouseGridOverlay[][] overlayElement = null;
```

```
	/** testing reversed paths */
	public boolean reversedPath=false;
	/** busy is true if currently finding a path - DO NOT DISTURB!*/
	public boolean busy=false;

	public Router(){
		resetPather ();

	}

	public void resetPather (Point p, Point p2){
		overlayElement[p.x][p.y].setStart(false);
		overlayElement[p2.x][p2.y].setEnd(false);
		opened = new ArrayList<WarehouseGridOverlay>();
		closed = new ArrayList<WarehouseGridOverlay>();
		bestList = new ArrayList<WarehouseGridOverlay>();

		overlayElement = new WarehouseGridOverlay[Warehouse.warehouseHeight]
[Warehouse.warehouseWidth];

		for (int row = 0; row < Warehouse.warehouseHeight; row++) {// rows
			for (int column = 0; column < Warehouse.warehouseWidth; column++) {// columns
				overlayElement[column][row] = new WarehouseGridOverlay(column,row);
				overlayElement[column][row].setStart(false);
				overlayElement[column][row].setEnd(false);
			}
		}
	}
	public void resetPather (){
		opened = new ArrayList<WarehouseGridOverlay>();
		closed = new ArrayList<WarehouseGridOverlay>();
		bestList = new ArrayList<WarehouseGridOverlay>();

		overlayElement = new WarehouseGridOverlay[Warehouse.warehouseWidth]
[Warehouse.warehouseHeight];
		for (int row = 0; row < Warehouse.warehouseHeight; row++) {// rows
			for (int column = 0; column < Warehouse.warehouseWidth; column++) {// columns
				overlayElement[column][row] = new WarehouseGridOverlay(column,row);
				overlayElement[column][row].setStart(false);
				overlayElement[column][row].setEnd(false);
			}
		}
	}
```

```
/**
 * get best Path using A* (AStar). Modified from Maze example in AI
 * Application Programming, M. Tim Jones (Charles River Media Programming,
 * 1995) C code implemented in Java from
 * http://memoization.com/2008/11/30/a-star-algorithm-in-java/
 *
 * @param start
 * @param end
 * @return
 */
protected List<Point> getBestPath(Point start, Point end) {
	resetPather ();
busy = true;
	WarehouseGridOverlay gridStart;
	WarehouseGridOverlay goal;
	bestList.clear();
	opened.clear();
	closed.clear();
	//make new local copies of the points or they get changed for the next run
	Point thisStart = new Point(start.x,start.y);
	Point thisEnd = new Point(end.x,end.y);

	//System.out.println("Finished adjancent");
	gridStart = overlayElement[thisStart.x][thisStart.y];
	gridStart.setStart(true);
	gridStart.setEnd(false);
	goal = overlayElement[thisEnd.x][thisEnd.y];
	goal.setEnd(true);
	goal.setStart(false);


	//calculate adjacencies (must do this after setting the start and end points)
	for (int i = 0; i < Warehouse.warehouseHeight; i++) {// rows
		for (int j = 0; j < Warehouse.warehouseWidth; j++) {// columns
			//overlayElement[j][i].clearAdjacencies();
			//overlayElement[j][i].setParent(null);
			overlayElement[j][i].calculateNewAdjacencies(overlayElement);
		}
	}
```

```
Set<WarehouseGridOverlay> adjacencies = gridStart.getAdjacencies();

//System.out.println("got adjancent");
for (WarehouseGridOverlay adjacency : adjacencies) {
	adjacency.setParent(gridStart);
	//gridStart.setChild(adjacency);
	if (adjacency.isStart() == false) {
		opened.add(adjacency);
	}
}//System.out.println("Finished opened");



//System.out.println(adjacencies);
while (opened.size() > 0) {
	WarehouseGridOverlay best = findBestPassThrough(goal);
	//System.out.println("Best is: " + best.getColumn() + ","+best.getRow());
	//System.out.println("found best pass through");
	opened.remove(best);
	closed.add(best);
	if (best.isEnd()) {
		//System.out.println("Best is the goal!");
		// fill list with the path before returning
		//System.out.println(start+" to "+end);
		populateBestList(goal);
		overlayElement[thisStart.x][thisStart.y].setStart(false);
		overlayElement[thisEnd.x][thisEnd.y].setEnd(false);
		List<Point> adjacents = Warehouse.getAdjacent(thisEnd);
		for (int i = 0; i < adjacents.size(); i++){
			WarehouseGridOverlay adjacent = overlayElement[adjacents.get(i).x][adjacents.get(i).y];
			//adjacent.clearAdjacencies();
			adjacent.getAdjacencies().remove(overlayElement[thisStart.x][thisStart.y]);
		}

		//convert from overlay grid locations to a list of points for the path
		overlayElement[thisStart.x][thisStart.y].setStart(false);
		overlayElement[thisEnd.x][thisEnd.y].setEnd(false);
		//System.out.println("returning bestList of size " +bestList.size());

		return generateBestPathPoints();
	} else {
		Set<WarehouseGridOverlay> neighbors = best.getAdjacencies();
		//check if any of the neighbors IS the parent of the best and remove that neighbor!!!! or get stack
overflow
```

```
                /*   for (WarehouseGridOverlay neighbor : neighbors) {
                        if (neighbor.getParent()!= null) {
                            if (neighbor == best.getParent()) {
                               System.out.println("neighbor of the best is best's parent === BAD");
                               best.removeAdjacency(neighbor);
                            }

                        }
                    }*/


                    //if one of the neighbors of best is the END point, then we are finished
                    if (neighbors.contains(goal)) {
                        //System.out.println("Neighbors contains the goal!");
                        opened.remove(goal);
                        closed.remove(goal);
                        opened.add(0, goal);
                        goal.setParent(best);
                        // fill list with the path before returning
                        //System.out.println(thisStart+" to "+thisEnd);
                        populateBestList(goal);
                        //print out the path
                        //reset the overlays
                        List<Point> adjacents = Warehouse.getAdjacent(end);

                        //overlayElement[thisStart.x][thisStart.y].setStart(false);
                        //overlayElement[thisEnd.x][thisEnd.y].setEnd(false);
/*                      for (int i = 0; i < adjacents.size(); i++){
                            WarehouseGridOverlay adjacent = overlayElement[adjacents.get(i).x][adjacents.get(i).y];
                            //adjacent.clearAdjacencies();
                            adjacent.getAdjacencies().remove(goal);
                        }*/
                        //convert from overlay grid locations to a list of points for the path
                        //System.out.println("returning bestList of size " + bestList.size());
                        return generateBestPathPoints();
                    } else {
                    for (WarehouseGridOverlay neighbor : neighbors) {
                        if (opened.contains(neighbor)) {
                            WarehouseGridOverlay tmp = new WarehouseGridOverlay(neighbor.getColumn(),
neighbor.getRow());
                            tmp.setParent(best);
                            //best.setChild(tmp);
```

```
                        if (tmp.getPassThrough(goal) >= neighbor.getPassThrough(goal)) {
                            continue;
                        }
                    }

                    if (closed.contains(neighbor)) {
                        WarehouseGridOverlay tmp = new WarehouseGridOverlay(neighbor.getColumn(),
neighbor.getRow());
                        tmp.setParent(best);
                        //best.setChild(tmp);
                        if (tmp.getPassThrough(goal) >= neighbor.getPassThrough(goal)) {
                            continue;
                        }
                    }
                    //if (neighbor.getChild()!=null&&best.getParent()!=null){
                    //if(!neighbor.getChild().equals(best.getParent())){


                        neighbor.setParent(best);
                        //best.setChild(neighbor);

                        opened.remove(neighbor);
                        closed.remove(neighbor);
                        opened.add(0, neighbor);
                    }
                /*  }else{
                        neighbor.setParent(best);
                        best.setChild(neighbor);

                        opened.remove(neighbor);
                        closed.remove(neighbor);
                        opened.add(0, neighbor);
                    }}*/

            }
            }
        }//should not get here (it will be null)
        System.out.println("BAD!");
        return null;

    }


    //intended to check list if already added and if so return -- should solve stack overflow because it will stop things
```

```
from running through more than once.
    private List<WarehouseGridOverlay> added = new ArrayList<WarehouseGridOverlay>();
    private void populateBestList(WarehouseGridOverlay overlay) {
        bestList.add(overlay);
        //added.add(overlay);
        if (overlay.getParent().isStart() == false) {
            //System.out.println("Parent of overlay is: "+ overlay.getParent().getColumn()
+","+overlay.getParent().getRow());
            //System.out.println(overlay.getParent()+" 999999999999999999999 "+overlay);
            //System.out.println(overlay.getParent().toPoint()+" 999999999999999999999 "+overlay.toPoint());
            populateBestList(overlay.getParent());
        }
        return;
    }

    /** Generates a List with the path points */
    private List<Point> generateBestPathPoints(    ) {
        List<Point> pointList = new ArrayList<Point>();
        if (reversedPath==false) {
            for (int i = 0; i<bestList.size(); i++) {
                int row = bestList.get(bestList.size()-i-1).getRow();
                int column = bestList.get(bestList.size()-i-1).getColumn();
                //System.out.println("Column,Row "+column+" "+row);
                pointList.add(new Point(column,row));
            }
        } else { //for the reversed case
            for (int i = 0; i<bestList.size(); i++) {
                int row = bestList.get(i).getRow();
                int column = bestList.get(i).getColumn();
                //System.out.println("Column,Row "+column+" "+row);
                pointList.add(new Point(column,row));
            }
        }
        busy = false;
        return pointList;
    }

    private WarehouseGridOverlay findBestPassThrough(WarehouseGridOverlay goal) {
        WarehouseGridOverlay best = null;
        for (WarehouseGridOverlay overlay : opened) {
            if (best == null || (overlay.getPassThrough(goal) < best.getPassThrough(goal))) {
                best = overlay;
```

```
                }
            }
            return best;
        }


        protected void end() {
            opened = null;
            closed = null;

            bestList = null;
            overlayElement = null;

        }
        /** not currently used */
        public void setupFrame(JFrame frame) {
            frame.setSize(500, 350);
        }


}
```

WarehouseGridOverlay.java

```
package team64.madkit;

import java.awt.Point;
import java.util.HashSet;
import java.util.Set;


/** Overlays a specific grid unit of the warehouse for finding paths */
public class WarehouseGridOverlay {

	private boolean start;
	private boolean end;
	private int row;
	private int column;

	private double localCost; // cost of getting from this grid unit to the goal
	private double parentCost; // cost of getting from parent square to this node
	private double passThroughCost;// cost of getting from the start to the goal

	private Set<WarehouseGridOverlay> adjacencies = new HashSet<WarehouseGridOverlay>();

	private WarehouseGridOverlay parent;
	//private WarehouseGridOverlay child;

	public WarehouseGridOverlay(int column,int row) {
		this.row= row;//rows
		this.column = column;//columns
	}

/** returns true if this gridUnit is the start of the path */
	public boolean isStart() {
		return start;
	}

	public Point toPoint (){
		return new Point(column, row);
	}

/** set to true if this gridUnit is the start of the path */
	public void setStart(boolean start) {
```

```
            this.start = start;
        }

/** returns true if this gridUnit is the end of the path */
        public boolean isEnd() {
            return end;
        }

        /** set to true if this gridUnit is the end of the path */
        public void setEnd(boolean end) {
            this.end = end;
        }


/** gets the column for this grid overlay */
        public int getColumn() {
            return column;
        }

/**sets the column for this grid overlay */
        public void setColumn(int column) {
            this.column = column;
        }

        /** gets the row for this grid overlay */
        public int getRow() {
            return row;
        }

        /**sets the row for this grid overlay */
        public void setRow(int row) {
            this.row = row;
        }


        /** returns a set of GridUnits that can be entered from this one */
        public Set<WarehouseGridOverlay> getAdjacencies() {
            return adjacencies;
        }

        /** sets a set of GridUnits that can be entered from this one */
        public void setAdjacencies(Set<WarehouseGridOverlay> adjacencies) {
```

```
        this.adjacencies = adjacencies;
    }


    /** gets the parent of this WarehouseGridOverlay */
    public WarehouseGridOverlay getParent() {
        return parent;
    }

    /** sets the parent of this WarehouseGridOverlay */
    public void setParent(WarehouseGridOverlay parent) {
        this.parent = parent;
    }

/*  public void setChild(WarehouseGridOverlay child) {
        this.child = child;
    }*/

    public void clearAdjacencies (){
        adjacencies.clear();
    }

    /** adds valid adjacent grid units to our adjacent list.
     * valid adjacent grid units do not have any occupants.
     */
    public void calculateAdjacencies(WarehouseGridOverlay[][] elements) {
        //there are a few cases to take care of. 1) check if the grid we are on is a block
        // blocker = conveyor or bin
        // 2) if the the adjacencies are blocked (whether or not the grid is a block)
        //and 3) if this is the END itself
      //FIRST take care of the case where this grid is NOT a blocker
        if (!Warehouse.getGrid()[column][row].isBlocked()){
            int bottom = row + 1; //grid row below us
            //int top = row - 1; //grid row above us
            //int left = column - 1; //grid column to the left of us
            int right = column + 1; //grid column to the right of us

            if (bottom < Warehouse.warehouseHeight) {
                //this code only applies for the upperleft to lowerright maze problem
                /*  if (elements[column][bottom].isEnd()){
                elements[column][bottom].addAdjacency(this);
              this.addAdjacency(elements[column][bottom]);
```

```
                }
                    */

                    if(!Warehouse.getGrid()[column][bottom].isBlocked() || elements[column][bottom].isEnd() ){
                        //ok to move here (nothing to the bottom of us in the grid or if it is blocked, it is the END
point)
                        elements[column][bottom].addAdjacency(this);
                        this.addAdjacency(elements[column][bottom]);
                        //System.out.println(this.column+":"+this.row);
                    } else {
                        //even though the bottom is blocked we need to add ourselves as its adjacency
                        elements[column][bottom].addAdjacency(this);
                    }
                }
                if (right < Warehouse.warehouseWidth) {
                    /*  if (elements[right][row].isEnd()){
                    elements[right][row].addAdjacency(this);
                   this.addAdjacency(elements[right][row]);
                }*/
                    if(!Warehouse.getGrid()[right][row].isBlocked() || elements[right][row].isEnd()) {
                        //ok to move here (nothing in the grid to our right) or to the RIGHT is the END point
                        elements[right][row].addAdjacency(this);
                        this.addAdjacency(elements[right][row]);
                        //System.out.println(this.column+":"+this.row);
                    } else {
                        //even though the right is blocked we need to add ourselves as its adjacency
                        elements[right][row].addAdjacency(this);
                    }
                }
            }//SECOND take care of the caes where the grid is a blocker (but we still need to set its adjacencies),
            //also 3) if this is the end (and it is a blocked grid unit)
             else {
                    //we still need to calculate adjacencies for the blocked grids to get a path (but do not put blocked
grid into the right,bottom adjacencies
                    int bottom = row + 1; //grid row below us
                    int right = column + 1; //grid column to the right of us

                    if (bottom < Warehouse.warehouseHeight) {
                        if(!Warehouse.getGrid()[column][bottom].isBlocked() || elements[column][bottom].isEnd()){
//ok to move here (nothing to the bottom of us in the grid)
                            //elements[column][bottom].addAdjacency(this);
                            this.addAdjacency(elements[column][bottom]);
                            //System.out.println(this.column+":"+this.row);
                        }
```

```
                }

                if (right < Warehouse.warehouseWidth) {
                    if(!Warehouse.getGrid()[right][row].isBlocked() || elements[right][row].isEnd()) {//ok to move here
(nothing in the grid to our right)
                        //elements[right][row].addAdjacency(this);
                        this.addAdjacency(elements[right][row]);
                        //System.out.println(this.column+":"+this.row);
                    }
                }



                if (this.isEnd()) {
                    //add the grid units to the right and bottom to ourselves if they are not blockers
                    if(!Warehouse.getGrid()[right][row].isBlocked()) {
                        elements[right][row].addAdjacency(this);
                    }
                    if(!Warehouse.getGrid()[column][bottom].isBlocked()) {
                        elements[column][bottom].addAdjacency(this);
                    }
                }
            }
        }

        /** adds valid adjacent grid units to our adjacent list.
         * valid adjacent grid units do not have any occupants.
         */
        public void calculateNewAdjacencies(WarehouseGridOverlay[][] elements) {

            //add all non-blocker grid points to the adjacencies
          //if the adjacency is the end point, also add it (even if it is a blocker)
                int bottom = row + 1; //grid row below us
                int right = column + 1; //grid column to the right of us
                int top = row - 1; //grid row above us
                int left = column - 1; //grid column to the left of us
                //System.out.println("Getting adjacencies for "+column+","+row);
                adjacencies.clear();
                setParent(null);

                if (bottom < Warehouse.warehouseHeight) {
                    if(!Warehouse.getGrid()[column][bottom].isBlocked() || elements[column][bottom].isEnd()){ //ok to move
```

```
here (nothing to the bottom of us in the grid)
                        this.addAdjacency(elements[column][bottom]);
                    }
                }
                if (bottom < Warehouse.warehouseHeight && right < Warehouse.warehouseWidth) {
                    if(!Warehouse.getGrid()[right][bottom].isBlocked() || elements[right][bottom].isEnd()){ //ok to move
here (nothing to the bottom of us in the grid)
                        this.addAdjacency(elements[right][bottom]);
                    }
                }
                if (bottom < Warehouse.warehouseHeight && left >= 0) {
                    if(!Warehouse.getGrid()[left][bottom].isBlocked() || elements[left][bottom].isEnd()){ //ok to move
here (nothing to the bottom of us in the grid)
                        this.addAdjacency(elements[left][bottom]);
                    }
                }
                if (top >= 0) {
                    if(!Warehouse.getGrid()[column][top].isBlocked() || elements[column][top].isEnd()){ //ok to move
here (nothing to the bottom of us in the grid)
                        this.addAdjacency(elements[column][top]);
                    }
                }
                if (top>=0 && right < Warehouse.warehouseWidth) {
                    if(!Warehouse.getGrid()[right][top].isBlocked() || elements[right][top].isEnd()){ //ok to move here
(nothing to the bottom of us in the grid)
                        this.addAdjacency(elements[right][top]);
                    }
                }
                if (top>=0 && left >= 0) {
                    if(!Warehouse.getGrid()[left][top].isBlocked() || elements[left][top].isEnd()){ //ok to move here
(nothing to the bottom of us in the grid)
                        this.addAdjacency(elements[left][top]);
                    }
                }

                if (right < Warehouse.warehouseWidth) {
                    if(!Warehouse.getGrid()[right][row].isBlocked() || elements[right][row].isEnd()) {//ok to move here
(nothing in the grid to our right)
                        this.addAdjacency(elements[right][row]);
                    }
                }
                if (left >= 0) {
                    if(!Warehouse.getGrid()[left][row].isBlocked() || elements[left][row].isEnd()) {//ok to move here
(nothing in the grid to our right)
```

```
                    this.addAdjacency(elements[left][row]);
                }
            }


/*          if (this.isEnd()) {
                //add the grid units to the right and bottom to ourselves if they are not blockers
                if(!Warehouse.getGrid()[right][row].isBlocked()) {
                    elements[right][row].addAdjacency(this);
                }
                if(!Warehouse.getGrid()[column][bottom].isBlocked()) {
                    elements[column][bottom].addAdjacency(this);
                }
            }*/

    // }
}

public void addAdjacency(WarehouseGridOverlay gridOverlay) {
    adjacencies.add(gridOverlay);
}

public void removeAdjacency(WarehouseGridOverlay gridOverlay) {
    adjacencies.remove(gridOverlay);
}

public double getPassThrough(WarehouseGridOverlay goal) {
    if (isStart()) {
        return 0.0;
    }
    /*if (goal.getChild()!=null){
        System.out.println("goal child not null");
    if(goal.getChild().equals(parent)){
        return 9999;
    }
    }*/
    return (getLocalCost(goal) + getParentCost());
}

public double getLocalCost(WarehouseGridOverlay goal) {

    if (isStart()) {
```

```
                return 0.0;
            }
            localCost = 1.0 * (Math.abs(column - goal.getColumn()) + Math.abs(row - goal.getRow()));
            return localCost;
        }

        public double getParentCost() {

            if (isStart()) {
                return 0.0;
            }

            if (parentCost == 0.0) {
                parentCost = 1.0 + .5 * (parent.getParentCost() - 1.0);
            }

            return parentCost;
        }

/*      public WarehouseGridOverlay getChild() {
            return child;
        }*/

        public String toString() {
            return "("+column+","+row+") prnt:"+this.getParent().column+","+this.getParent().row;
        }

}
```

GridUnit.java

```
** One basic grid unit that makes up the grid that the warehouse is laid out on */
package team64.madkit;

import java.awt.Color;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import madkit.kernel.AgentAddress;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */

final public class GridUnit
{
	/** domain color, if set. If not set will default to light gray*/
	protected Color stopDomainColor = Color.lightGray;
	/** a collection of agents occupying the grid unit */
	protected List<AgentAddress> occupants = null;

	/** a layered set of storage bins  = a shelf unit */
	protected List<StorageBin>bins = new ArrayList<StorageBin>();

	/** Agent Addresses */
	public  List <AgentAddress> getOccupants(){
		return occupants;
	}

	/** if set to true this is a PATH, which means NO SHELVES */
	protected boolean path=false;

	/** gets a conveyor segment at this location, or null if there are none */
	public AgentAddress getConveyorSegment(){
		Iterator i = occupants.iterator();
		while(i.hasNext()){
			AgentAddress current = (AgentAddress) i.next();
			if ( current.getGroup().equals("conveyorsystem")){
```

```
                return current;
            }
        }
        return null;
}

//for stops only
//List <Integer> domain  = new ArrayList<Integer>();


//only used when in search algorithms
GridUnit parent;

//only used when in search algorithms
int distance = 9999;

public boolean isBlocked(){
        if (containsBins()||containsConveyor()){
            return true;
        }
        return false;
}

public boolean containsBins (){
        if (!bins.isEmpty()){
            return true;
        }
        return false;
}

/** gets the bin at the specified layer */
public StorageBin getBinAt (int binLayer){
        if (!bins.isEmpty()){
            return (bins.get(binLayer));
        }
        return null;
}

public int getBinLayers(){
        return bins.size();
}

public void addBin(StorageBin bin){
        bins.add(bin);
```

```
    }

    public void addPath(){
        path = true;
    }


    public boolean addItem(int id, int quantity){
        if (!containsBins()){
            return false;
        }
        Iterator <StorageBin> iter = bins.iterator();
        while(iter.hasNext()){
            StorageBin bin = iter.next();
            if (bin.getItemId() == id){
                bin.addItem(quantity);
                return true;
            }
        }
        return false;
    }

//  return of 0 means no items; 1 means not enough; two is true
    public int containsItem(int id, int quantity){
        if (!containsBins()){
            return 0;
        }
        Iterator <StorageBin> iter = bins.iterator();
        while(iter.hasNext()){
            StorageBin bin = iter.next();
            if (bin.getItemId() == id){
                if (bin.getQuantity()>=quantity){
                    return 2;
                }
                return 1;
            }
        }
        return 0;
    }

    public List <Integer> getBinItems(){
        if (!containsBins()){
```

```
                return null;
            }
            List<Integer>items = new ArrayList<Integer>();
            Iterator <StorageBin> iter = bins.iterator();
            while(iter.hasNext()){
                StorageBin item = iter.next();
                items.add(item.getItemId());
            }
            return items;
        }

        public GridUnit() {
            occupants = new ArrayList<AgentAddress>();
        }

        public boolean containsConveyor(){
            Iterator i = occupants.iterator();
            while(i.hasNext()){
                AgentAddress current = (AgentAddress) i.next();
                if ( current.getGroup().equals("conveyorsystem")){
                    return true;
                }
            }
            return false;
        }

    /** returns true if this is an item stop conveyor segment */
        public boolean containsItemStop() {
            Iterator i = occupants.iterator();
            while(i.hasNext()){
                AgentAddress current = (AgentAddress) i.next();
                if ( current.getRole().equals("itemstop")){
                    return true;
                }
            }
            return false;
        }



        public List<AgentAddress> getPickers (){
            List <AgentAddress> found = new ArrayList<AgentAddress>();
            Iterator<AgentAddress> i = occupants.iterator();
            while(i.hasNext()){
```

```
            AgentAddress current = (AgentAddress) i.next();
            if ( current.getRole().equals("Picker")){
                found.add(current);
            }
        }
        return found;

}

/*public void activate()
{
        getLogger().setWarningLogLevel(Level.INFO);
        setLogLevel(Level.FINEST);
        createGroupIfAbsent("warehouse","warehouse",true,null);
        requestRole("warehouse","warehouse", "grid",null);
}

public void receiveMessage(Message m) {
        ObjectMessage message = (ObjectMessage)m;
        if (m != null) {
            if(logger != null) {
                logger.info("Grid Unit got an object Message from " + message.getSender());
            }
        }

}*/

/** returns the color of the domain if set */
public Color getStopDomainColor(){
        return stopDomainColor;

}

/** sets the color of the grid objects if they are in the stop domain */
public void setStopDomainColor(Color stopDomainColor){
        this.stopDomainColor = stopDomainColor;

}

/** add an agent to the occupied set */
protected void addAgent (AgentAddress agent) {
        occupants.add(agent);
```

```
            if (agent.getRole().equals("conveyorsegment")){
                System.out.println("Added conveyor segment to grid " + this.toString());
            }
        }

        /** remove an agent from the occupied set */
        protected void removeAgent (AgentAddress agent) {
            if (occupants.size()>0){
                occupants.remove(agent);
            }
        }

        /** checks if there is an occupant if the candidate can
         * also occupy the grid unit. Returns true if the spot is empty or
         * compatible, false otherwise.
         * Rules are:
         * 1) if the class is a worker, nothing else can occupy the spot
         * 2) if the class is a tote, only a conveyor segment can occupy the spot as well
         * 3) if the class is a conveyor segment, only a tote can occupy the spot as well
         *
         */
        public boolean checkOccupantCompatibility(AgentAddress agent) {
            if (containsBins()){
                return false;
            }
            if (occupants.size()>0) { //only check if there are occupants already
                // Iterating over the elements in the set
                if (occupants.size()>1){
                //   return false;//Definitely cannot have more than 2 things in this spot
                }
                //otherwise we have to get the occupant's group and see if we can put agent in this spot too
                //only way to get the occupant is to iterate
                String agentGroup = agent.getGroup();
                Iterator<AgentAddress> it = occupants.iterator();
                while (it.hasNext()) {
                    // Get group of
                    String occupierGroup = it.next().getGroup();
                    if (agentGroup.equals("workers")){
                        return false; //no matter what the agent is it cannot occupy with a Worker
                    } else {
                        if (agent.getRole().equals("tote")){
                            //if the agent is a tote, and the occupier is a conveyor segment return true, otherwise
return false
                            if(occupierGroup.equals("segment")) {
```

```
                                return true;
                            } else {
                                return false;
                            }
                        }
                    }

                }
            }
            //if the occupants set is null then there is no question the new agent can occupy
            return true;

        }




/*
        *//** gets the AgentAddress of the worker in this gridUnit. This will return null if no worker is here.  *//*
        public AgentAddress getWorker() {
            return worker;
        }

        *//** sets the AgentAddress of the worker (if any) in this gridUnit.*//*
        public  void setWorker(AgentAddress worker) {
             this.worker = worker;
        }

        *//** gets the AgentAddress of the converyorSegment in this gridUnit. This will return null if no conveyor is here. *//*
        public AgentAddress getConveyorSegment() {
            return conveyorSegment;
        }

        *//** sets the AgentAddress of the conveyorSegment (if any) in this gridUnit.  *//*
        public  void setConveyorSegment(AgentAddress conveyorSegment) {
             this.conveyorSegment = conveyorSegment;
        }

        *//** gets the AgentAddress of the tote in this gridUnit. This will return null if no tote is here.*//*
        public AgentAddress getTote() {
            return tote;
```

```
    }

    *//** sets the AgentAddress of the tote (if any) in this gridUnit.  *//*
    public  void setTote(AgentAddress tote) {
        this.tote = tote;
    }

    *//** gets the AgentAddress of the storage bin in this gridUnit. This will return null if no storage bin is here. *//*
    public AgentAddress getStorageBin() {
        return storageBin;
    }

    *//** sets the AgentAddress of the storage bin (if any) in this gridUnit.  *//*
    public  void setStorageBin(AgentAddress storageBin) {
        this.storageBin = storageBin;
    }

*/
    /** for testing *//*
    public static void main(String[] args) {
        GridUnit[][] environment = null; //the environment grid
        environment  = new GridUnit[10][10];
        Picker aPicker = new Picker();
        aPicker.activate();
        AgentAddress p = aPicker.getAgentWithRole("warehouse", "worker", "picker");
        Sorter aSorter = new Sorter();
        aPicker.activate();
        AgentAddress s = aPicker.getAgentWithRole("warehouse", "worker", "sorter");

        environment[0][0].addAgent(p);
        boolean result = environment[0][0].checkOccupantCompatibility(s);
        result = environment[0][1].checkOccupantCompatibility(s);
        Tote aTote = new Tote();
        aTote.activate();
        AgentAddress t = aTote.getAgentWithRole("warehouse", "container", "tote");
        ConveyorSegment aCS = new ConveyorSegment(2);
        aCS.activate();
        AgentAddress cs = aCS.getAgentWithRole("warehouse","conveyorsystem", "segment");

        environment[2][2].addAgent(cs);
        result = environment[2][2].checkOccupantCompatibility(t);
        result = environment[2][2].checkOccupantCompatibility(s);
    }*/
```

```
}
```

Workers.java

```
/*
 * Worker.java - A worker of any type in the warehouse. Workers have a role of either a sorter, picker, or packer.
 * */

package team64.madkit;

import java.awt.Color;
import java.awt.Point;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;

import madkit.kernel.AbstractAgent;
import madkit.kernel.Message;
import madkit.message.ObjectMessage;
import team64.madkit.messages.NewLocationMessage;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class Workers extends AbstractAgent
{

	/** if this worker is doing its job then working will be true */
	//boolean working = true;
	/** worker is idle if it does not have an item to pick and is waiting for a task */
   protected static int idle=1;
   /** worker has an item in its possession */
   protected static int hasItem=2;
   /** worker is getting an item */
   protected static int gettingItem=3;

	//Vector <Task> tasks = new Vector<Task>();
	static float minimumWage=6.50f; //so we can calculate the cost of using the workers
	protected float hourlyCost=minimumWage;

	/** workerInfo contains the location, destination, etc. of the worker */
	WorkerInformation workerInfo= new WorkerInformation();
   /** all workers have a state. This is one of idle, gettingItem, or hasItem */
	protected int state;
```

```
        List <Point> route = new ArrayList <Point> (); //route will not be empty if a route has been calculated


        /*public int estimateDistance(){
                int taskTimes = 0;
                //check time estimates for all but the one being worked on
                for (int i = 1; i < tasks.size(); i++){
                        Task current = tasks.get(i);
                        taskTimes = taskTimes + current.getDirections().size();
                }
                return taskTimes;
        }*/


        /*
        protected void taskFinished(){
                tasks.remove(0);
                if (tasks.isEmpty()){
                        state=idle;
                }
        }

        protected void pickUpDone(){
                tasks.get(0).toPickup = false;
        }
*/
        public void activate(){
        getLogger().setWarningLogLevel(Level.INFO);
        setLogLevel(Level.FINEST);
        requestRole("warehouse","workers", "idle",null);

        }



/** used by workers that are milling around (if they do not have any work orders) */
        public Point moveRandomly() {
                int randStep = Warehouse.getRandomStep();
                if(logger != null) {
                        logger.finest("random step for x direction is " + randStep);
                }
                int xadd = randStep;
```

```
        int yadd = Warehouse.getRandomStep();
        int newx = xadd + workerInfo.location.x;
        if (newx < 0) {
            newx=0;
        }
        if (newx > Warehouse.warehouseWidth) {
            newx = Warehouse.warehouseWidth;
        }

        int newy = yadd + workerInfo.location.y;
        if (newy < 0) {
            newy=0;
        }
        if (newy > Warehouse.warehouseHeight) {
            newy = Warehouse.warehouseHeight;
        }
        return new Point(newx,newy);
        //don't set the new location until we get reply from warehouse about the change
        //workerInfo.setLocation(new Point(newx,newy));
    }

    /**returns a very basic path from start to destination without regard to obstacles */
    protected List<Point> getBasicPath(Point start, Point destination) {
        List<Point> basicPath = new ArrayList<Point>();
        //equation of the line that goes through start and destination

        double slopey = (1.0f*(destination.y-start.y))/(1.0f*(destination.x-start.x));
        double slopex = (1.0f*(destination.x-start.x))/(1.0f*(destination.y-start.y));
        //if there are more rows (y steps) than columns (x steps) then step by y, otherwise step by x
        boolean stepy = false;
        if (Math.abs(destination.y-start.y) > Math.abs(destination.x-start.x)) {
            stepy = true;
        }
        if (stepy) {
            int lowy;
            int endy;
            //find lowest y value and start from there
            if (start.y<destination.y) {
                lowy=start.y;
                endy=destination.y;
            } else {
                lowy = destination.y;
                endy = start.y;
            }
```

```
            for (int y=lowy;y<endy;y++) {
                int x = (int) ((slopex*y) + (start.x-slopex*start.y));
                basicPath.add(new Point(x,y));
            }
        } else {//step in x direction
            int lowx;
            int endx;
            //find lowest x value and start from there
            if (start.x<destination.x) {
                lowx=start.x;
                endx=destination.x;
            } else {
                lowx = destination.x;
                endx = start.x;
            }
            for (int x=lowx;x<endx;x++) {
                int y = (int) ((slopey)*x + (start.y-slopey*start.x));
                basicPath.add(new Point(x,y));
            }
    }

        return basicPath;
}


@SuppressWarnings("unchecked")
protected Message getBestRoute(List<Message> routes) {
        ObjectMessage<Integer> best = (ObjectMessage<Integer>) routes.get(0);
        for(Message m : routes){
            ObjectMessage<Integer> distance = (ObjectMessage<Integer>) m;
            if(best.getContent() > distance.getContent()){
                best = distance;
            }
        }
        return best;
}

public void receiveMessage(Message m) {
        if (m != null) {
            if(logger != null) {
                logger.info("General Worker got a Message from " + m.getSender()+ " says: " +m.toString());
            }
```

```
                if ((m instanceof NewLocationMessage)) { //a reply from the Warehouse about the request to move
locations
                    Point p2 = ((NewLocationMessage) m).getNewLocation();
                    //could be the same as the old location, but put it into the new location in WorkerInfomation
anyway
                    workerInfo.setLocation(p2);
                }
            }

        }

        /** Called by a property probe for a Worker. */
        class WorkerInformation {

            /** different workers have different colors */
            protected Color workerColor;
            /** the point in the warehouse where the worker is currently */
            private Point location = null; //where the worker is in the warehouse
            /** the point in the warehouse that the worker is traveling to. If null there is no destination and the work is
idle. */
            private Point destination = null; //where the worker is going in the warehouse

            public WorkerInformation() {
                workerColor = Color.magenta;
                //location = new Point(Workers.startx,Workers.starty);
                location = new Point(Warehouse.workerStartingLocation);
            }

            /**
             * @return the worker's Color
             */
            public Color getWorkerColor() {
                return workerColor;
            }


            /**
             * @param workerColor the workerColor to set
             */
            public void setWorkerColor(Color workerColor) {
                this.workerColor = workerColor;
            }

            /**Gets the location of the worker in the warehouse
```

```
         * @return the location
         */
        public Point getLocation() {
            return location;
        }

        /**Sets the location of the worker in the warehouse. Should only be called from a reply
         * to sendMessage("warehouse","warehouse","warehouse",new NewLocationMessage(this.location, location))
         * @param location the location of the worker
         */
        public void setLocation(Point location) {
            //do not set the location until we get a reply from the warehouse with the new or old location
            this.location = location;
        }

    }


}
```

Picker.java

```
package team64.madkit;

import java.awt.Color;
import java.awt.Point;
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.logging.Level;

import madkit.kernel.Message;
import team64.madkit.messages.AddTask;
import team64.madkit.messages.ColorMessage;
import team64.madkit.messages.LocationMessage;
import team64.madkit.messages.LocationQueryMessage;
import team64.madkit.messages.NewLocationMessage;
import team64.madkit.notused.DirectionQuery;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class Picker extends Workers {
	//int orderNumber = 0; // 0 means it has not been assigned to an order
	/** the current item the picker has been asked to get for the tote*/
	int itemFetch = -1;
	/** a list of items to fetch (picked in this order) generated by the Item Stop.
	 * This can be used for discrete picking if only one item is
	 * added to this list at a time.
	 */
	ArrayList <Integer> fetchList = null;

	/** we need a copy of the original list so when we get back to the tote we can add all the items */
	ArrayList <Integer> originalFetchList =  null;
			//new ArrayList <Integer>();

	//boolean assignedToStation = false;
	protected Point destination;
```

```
	//boolean hasItem = false;
	protected Tote tote;
	protected ItemStop myPickingStation;
	protected Router myRouteFinder = new Router();

	public Picker() {
		workerInfo = new WorkerInformation();
		workerInfo.setWorkerColor(Color.orange);
		state = idle;
	}


/** sets the  item stop (picking station) that this worker is assigned to */
	protected void setPickerStopAssignment(ItemStop stop) {
		myPickingStation = stop;
	}

	public void activate()
	{
		getLogger().setWarningLogLevel(Level.INFO);
		setLogLevel(Level.FINE);
		createGroupIfAbsent("warehouse","workers",true,null);
		requestRole("warehouse","workers", "worker",null);
		requestRole("warehouse","workers", "picker",null);

	}

	public void reorderFetchList(){
		// key is distance, item
		Hashtable <Integer, Integer> itemDistance = new Hashtable<Integer, Integer>();
		List <Integer> newList = new ArrayList <Integer>();
		for (int i = 0; i < fetchList.size(); i++){
			int distance = myRouteFinder.getBestPath(myPickingStation.segmentInfo.getSegmentLocation(),
Warehouse.getItemLocations().get(fetchList.get(i)).getLocation()).size();
			itemDistance.put(distance,fetchList.get(i));
		}
		Set <Integer> distances = itemDistance.keySet();
		Object[] distancesArray = distances.toArray();
		Arrays.sort(distancesArray);
		for (int i = 0; i < distancesArray.length; i++){
			newList.add(itemDistance.get(Integer.parseInt(""+distancesArray[i])));
		}
```

```
    }

    /** called by the worker scheduler */
    public void move()
    {
        if(logger != null) {
            if (tote!=null) {
                logger.finer("Scheduler has said that this picker worker should move to work on order " +
tote.toteInfo.orderNumber);
            } else {
                logger.finer("Scheduler has said that this picker worker should wait for an order");
            }
            logger.finer("Route is empty? " + route.isEmpty());
            if (state == idle ) {
                logger.finer("Worker is idle" );
            }
        }
        Point myLocation = workerInfo.getLocation();
        if (state != idle && itemFetch!=-1 ) {
            if (route.isEmpty()) {
/*                  if (itemFetch==-1){ //not assigned to get anything
                        state = idle;
                        return;
                    }*/
                    myRouteFinder = new Router();
                    //get a route
                    //if getting an item need to get a route from where we are to the item bin
                    if (state==gettingItem) {
                        Point binLocation = Warehouse.getItemLocation(itemFetch);
                        route = myRouteFinder.getBestPath(myLocation, binLocation);
                    } else {//returning to item stop
                        route = myRouteFinder.getBestPath(myLocation,
myPickingStation.segmentInfo.getSegmentLocation());
                    }
                    if (state==gettingItem) {
                        logger.finer("Route was empty so now we have a new one to get an item.  Route is size " +
route.size());
                    }
                    if (state==hasItem) {
                        logger.finer("Route was empty so now we have a new one to return to item stop Route is
size " + route.size());
                    }
                    if (state==gettingItem) {
                        logger.finer("Route to get item " + itemFetch+" is from " +myLocation+" to
```

```
"+Warehouse.getItemLocation(itemFetch));
                    }
                    if (state==hasItem) {
                        logger.finer("Route to put item " + itemFetch+" into tote is " +myLocation+" to "+
myPickingStation.segmentInfo.getSegmentLocation());
                    }
                } else {//we have a route, so go go go ....
                    Point nextPoint = route.get(0);
                    route.remove(0);
                    GridUnit newLoc = Warehouse.getGridUnit(nextPoint.x,nextPoint.y);
                    //there is a chance that the starting position is the first position in the route...
/*                  while (workerInfo.getLocation()==nextPoint) {
                        if (route.size()>0) {
                            route.remove(0);
                        }
                        nextPoint=route.get(0);

                    }*/
                    boolean blocked = newLoc.isBlocked();
                    if (state==gettingItem && atBinLocation()) {
                        //check if whole fetch list is empty

                        if (fetchList.isEmpty()) {
                            state=hasItem;
                        } else {

                            itemFetch = fetchList.get(0);
                            fetchList.remove(0);
                            state=gettingItem;
                        }
                        return;
                    }
                    else if (state==hasItem && atItemStop()){
                        //we have arrived at an item stop
                        if (tote!=null) { //did the tote leave???
                            tote.toteInfo.addToContents(originalFetchList);
                        }
                        resetParamsAfterDeliveringItem();
                        return;
                    }
                    if (!blocked){
                        //move to the next location
```

```
                        sendMessage("warehouse","workers","warehouse",new NewLocationMessage(workerInfo.g
etLocation(),nextPoint));
                    } else {
                        //if the path is blocked, find a new route the next time we move
                        route.clear();
                    }

                }


            } else {//picker is idle, needs an item to fetch
                //the picker will get a new item the next time it is bothered in the itemstop's "botherpicker" method
            }

        }


        /**returns true if the worker is at or near the bin location */
        protected boolean atBinLocation() {
            List adjacent = Warehouse.getSorrounding(workerInfo.getLocation());
            if (adjacent.contains(Warehouse.getItemLocation(itemFetch))) {
                return true;
            }
            return false;
        }

        /**returns true if the worker is at or near the item stop */
        protected boolean atItemStop() {
            List adjacent = Warehouse.getSorrounding(workerInfo.getLocation());
            if (adjacent.contains(myPickingStation.getSegmentInfo().segmentLocation)) {
                return true;
            }
            return false;
        }


        /** clears all items and sets the worker idle */
        protected void resetParamsAfterDeliveringItem(){
                state = idle;
                itemFetch = -1;
                fetchList.clear();
                originalFetchList.clear();
                //tote = null; set tote to null after it is filled at this itemstop
                route.clear();
```

```
    }



/*  *//** wait for your tote, find a path to the item, go get it, fill the tote, put the tote back on the conveyor *//*
    protected void handlePickingRequest() {

    }*/

    /** gets whether or not this picker is already assigned to a station */
    protected boolean isAssigned() {
        if (myPickingStation==null){
            return false;
        } else {
            return true;
        }
    }



    /** this will be a message to go pick one or more items from the storage shelves */
    public void receiveMessage(Message m) {
        if (m != null) {
            if(logger != null) {
                logger.finest("Picker got a Message from " + m.getSender()+ " says: " +m.toString());
            }

            else if ((m instanceof ColorMessage)){
                this.workerInfo.setWorkerColor(((ColorMessage) m).getColor());
                //setNextConveyorLink(aa);
            }
            //this is used when an item stop gets a tote with an order, the picker is informed
            else if ((m instanceof AddTask)){
                if ((((AddTask) m).getLocation()).equals(destination)){
                    fetchList = new ArrayList <Integer>();
                    fetchList = (((AddTask) m).getItems());
                    //make a copy so that we keep the original list to add to the tote later
                    originalFetchList =  new ArrayList <Integer>();
                    for (int i =0;i<fetchList.size();i++) {
                        originalFetchList.add(fetchList.get(i));
                    }
                    tote = ((AddTask) m).getTote();
```

```
                        //working = true;
                        itemFetch = fetchList.get(0);
                        state = gettingItem;
                    }else{

                        //sendMessage(m.getSender(), new RetryPick((((AddTask) m).location),(((AddTask)
m).item),((AddTask) m).tote));
                    }
                    //setNextConveyorLink(aa);
                }
                if ((m instanceof LocationQueryMessage)) {
                    LocationMessage message = new LocationMessage(this.workerInfo.getLocation());
                    this.sendReply(m, message);
                    logger.finest("Location Query Recieved");
                }
                if ((m instanceof NewLocationMessage)) { //a reply from the Warehouse about the request to move
locations
                    if(((NewLocationMessage) m).isMove()){
                    Point p2 = ((NewLocationMessage) m).getNewLocation();
                    //could be the same as the old location, but put it into the new location in WorkerInfomation
anyway
                    workerInfo.setLocation(p2);
                    } else {
                        //we are lost or blocked! send a message to find a path to the end of our route (the
destination)
                        sendMessage("warehouse","workers","pathfinder",
new DirectionQuery(workerInfo.getLocation(),route.get(route.size())));
                        route.clear();

                    }
                }


            }
        }
}
```

ConveyorSegment.java

```
/*
 * ConveyerSegment.java - One segment of a conveyor system. It moves the tote, if any, to the next segment when commanded to move.
 *   One segment can hold one tote.
 *   There are different types of conveyor segments: regular segment (moves at a constant speed),curved segment (changes direction vector),
 *    timer segment (can change speed to space totes), transfer segment (it can push totes off the conveyor),
 *    and sensor segment (it tells the system where the tote is when it passes).
 *
 */

package team64.madkit;

import java.awt.Color;
import java.awt.Point;
import java.util.List;
import java.util.logging.Level;

import madkit.kernel.AbstractAgent;
import madkit.kernel.AgentAddress;
import madkit.kernel.Message;
import madkit.message.ActMessage;
import team64.madkit.messages.MoveTote;
import team64.madkit.messages.NewLocationMessage;
import team64.madkit.messages.ToteMessage;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class ConveyorSegment extends AbstractAgent
{
	//these are direction constants for the conveyor segments
	public static int right=0;
   public static int up=1;
   public static int left=2;
   public static int down=3;

	boolean toteMoved = false;
```

```
protected Tote tote=null; //if this conveyor segment has a tote, this is the tote's id
//AgentAddress nextConveyorSegmentAddress = null; //so we know where the tote goes next
protected SegmentInformation segmentInfo ;

/** Generic Constructor */
public ConveyorSegment() {
	segmentInfo = new SegmentInformation();
}

public ConveyorSegment(int direction)
{
	segmentInfo = new SegmentInformation();
	segmentInfo.direction = direction;
	//segmentInfo.setSegmentLocation(location);
}

public void activate() {
	setLogLevel(Level.FINE);
	requestRole("warehouse","conveyorsystem", "segment",null);
	//segmentInfo.setSegmentLocation(segmentInfo.getSegmentLocation());
	//Warehouse.grid[segmentInfo.getSegmentLocation()]

}



public void toteNotMoved(){
	toteMoved = false;
}

public void moveTote() {
	if(!toteMoved){
		if(logger != null) {
			logger.finest("Scheduler has said to move my tote to the next segment if I have one...");
		}
		if (tote!=null) {
			//System.out.println("shifting tote");
			shiftToteTo(segmentInfo.getNextLocation());
			tote = null;
			segmentInfo.setHasTote(false);
		}
	}
}
```

```
    public void ejectTote() {
        if(logger != null) {
            logger.finest("Scheduler has said that if I get a tote to eject it ...");
        }

    }

    private void shiftToteTo(Point p){
        //System.out.println("shifting tote "+ tote);
        List <AgentAddress> aa = getAgentsWithRole("warehouse", "warehouse", "manager");
        AgentAddress warehouse = aa.get(0);
        sendMessage(warehouse,new MoveTote(segmentInfo.getSegmentLocation(),p,tote));
    }

    public void receiveMessage(Message m) {
        if (m != null) {
            if(logger != null) {
                logger.finer("Conveyor Segment got a Message from " + m.getSender()+ " says: " +m.toString());
            }
            String senderGroup = m.getSender().getGroup();
            String senderRole =  m.getSender().getRole();
            //check if it is an object message
            if ((m instanceof ToteMessage)) {
                Tote tote = ((ToteMessage) m).getTote();
                addToteToConveyor(tote);
                logger.finer("adding tote with order " + tote.toteInfo.getOrderNumber() + " to conveyor");
            }
            else
                if ((m instanceof NewLocationMessage)) { //a reply from the Warehouse about the request to set the
location
                    Point p2 = ((NewLocationMessage) m).getNewLocation();
                    if(logger != null) {
                        logger.finer("Warehouse has said to set new location to: " + p2);
                    }
                    //could be the same as the old location, but put it into the new location in WorkerInfomation anyway
                    segmentInfo.setSegmentLocation(p2);
                }
                else if ((m instanceof ActMessage)) {
                    if (tote !=null) {
                        this.shiftToteTo(segmentInfo.getNextLocation());
                        tote=null;
```

```
                    }
                }
        }//m is null??
}

/* sets the next conveyor segment to transfer a tote */



public void addToteToConveyor(Tote tote) {
        if (this.tote==null) {//this means we do not already have a tote sitting on us
                //System.out.println("tote id is null; adding new tote");
                this.tote = tote;
        } else { //we have a tote already so we have to eject our existing tote to the next conveyer segment
                //System.out.println("tote id is not null; adding new tote and shifting existing down");
                this.shiftToteTo(segmentInfo.getNextLocation());
                this.tote=tote;
        }
        if (this.tote!=null){
                segmentInfo.setStateColor(Color.green);
                segmentInfo.setHasTote(true);
                //System.out.println("tote id is not null; setting has tote");
        }
        else {
                segmentInfo.setStateColor(Color.white);
                segmentInfo.setHasTote(false);
                //System.out.println("tote id is null; setting not has tote");
        }
        toteMoved = true;
}

/** this automatically tries to find the next conveyor segment in the flow direction.
 * This should only be used at startup (when the conveyor system is layed out
 * because at runtime it is possible to dynamically change the next segment for redirecting
 * totes.
 *      right=0;
 *  up=1;
 *  left=2;
 *  down=3;
 **/
public Point findNextLocation(){
        Point location = (Point) segmentInfo.getSegmentLocation().clone();
        int direction = segmentInfo.direction;
        Point nextLocation = new Point (0,0);
```

```
            // System.out.println("looking for next location with direction "+direction+" at "+ location);

            //in unit circle the points are in the form (cos a, sin a); converts direction to radians and adds (cos a, sin a);
            nextLocation = new Point (location.x + (int) Math.cos((Math.PI*direction)/2), location.y+
(int) Math.sin((Math.PI*direction)/2));

            return  nextLocation;
        }


        public SegmentInformation getSegmentInfo(){
            return segmentInfo;
        }



}
```

SegmentInformation.java

```
/** Contains data for a conveyor segment agent */

package team64.madkit;
/** this is a holder for all segment data */
import java.awt.Color;
import java.awt.Point;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */

class SegmentInformation {
	/** this segment's grid location */
	protected Point segmentLocation; //a location to draw the segment representation
	/** the next segment grid locaton. This is where a tote will go when moved */
	protected Point nextSegmentLocation;
	protected Color stateColor;
	protected Color fontColor;
	protected boolean hasTote=false;
	protected int direction;

	public Point getNextSegmentLocation() {
		return nextSegmentLocation;
	}



	public void setNextSegmentLocation(Point nextSegmentLocation) {
		this.nextSegmentLocation = nextSegmentLocation;
	}



	public Color getFontColor() {
		return fontColor;
	}



	public void setFontColor(Color fontColor) {
```

```
        this.fontColor = fontColor;
}



public int getDirection() {
        return direction;
}

public void setDirection(int direction) {
        this.direction = direction;
}

public SegmentInformation() {
        stateColor = Color.white;
        //segmentLocation = new Point (0,0);
        segmentLocation=new Point(0,0);
        nextSegmentLocation = new Point (0,0);
}



/**
 * @return the stateColor
 */
public Color getStateColor() {
        return stateColor;
}

/**
 * @param stateColor the stateColor to set
 */
public void setStateColor(Color stateColor) {
        this.stateColor = stateColor;
}

/**
 * @return the segmentLocation
 */
public Point getSegmentLocation() {
        return segmentLocation;
}
```

```
	/**
	 * sets the next segment to pass the tote to
	 */
	public void setNextLocation(Point nextSegmentLocation) {
		this.nextSegmentLocation = nextSegmentLocation;
	}

	/**
	 * @return the segment to pass the tote to
	 */
	public Point getNextLocation() {
		return nextSegmentLocation;
	}

	/**
	 * @param segmentLocation the segmentLocation to set
	 */
	public void setSegmentLocation(Point segmentLocation) {
		this.segmentLocation = segmentLocation;
	}

	/**
	 * @return whether or not this segment has a tote on it
	 */
	public boolean getHasTote() {
		return hasTote;
	}

	/**
	 * @param sets whether or not this segment has a tote on it
	 */
	public void setHasTote(boolean hasTote) {
		this.hasTote = hasTote;
	}
}
```

ItemStop.java

```
/*
 * ItemStop.java - One segment of a conveyor system. It receives the tote if there
 * is an item that needs to be fetched for the order in this item stop's domain.
 *
 */

package team64.madkit;

import java.awt.Color;
import java.awt.Point;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;

import madkit.kernel.AgentAddress;
import madkit.kernel.Message;
import team64.madkit.messages.AreYouAssignedMessage;
import team64.madkit.messages.MoveTote;
import team64.madkit.messages.ToteMessage;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class ItemStop extends ConveyorSegment
{
	Picker picker;
	protected StopInformation stopInfo;



	public ItemStop(int direction)
	{
		super(direction);
		stopInfo = new StopInformation();
		segmentInfo.direction = direction;
		//segmentInfo.setSegmentLocation(location);
	}
```

```
	public void activate() {
		setLogLevel(Level.FINE);
		requestRole("warehouse","conveyorsystem", "itemstop",null);
		//segmentInfo.setSegmentLocation(segmentInfo.getSegmentLocation());
		//Warehouse.grid[segmentInfo.getSegmentLocation()]

	}


	public Point getBehindLocation(){
		Point location = (Point) segmentInfo.getSegmentLocation().clone();
		// System.out.println("looking for next location with direction "+direction+" at "+ location);

		//in unit circle the points are in the form (cos a, sin a); converts direction to radians and adds (cos a, sin a);
			location = new Point (location.x - (int) Math.cos((Math.PI*segmentInfo.direction)/2), location.y -
(int) Math.sin((Math.PI*segmentInfo.direction)/2));



		//System.out.println("got next location with direction "+direction+" at "+ location);
		return location;
	}

	public void toteNotMoved(){
		toteMoved = false;
	}



	/** Called from conveyor scheduler . Find a picker to work here or tell the picker to get to work .
	 * This should only happen if the worker is idle.
	 */
	public void botherPicker(){
		if (picker == null) {
			//get a PICKER to work here
			List  <AgentAddress> pickers = new ArrayList <AgentAddress>();
			pickers = getAgentsWithRole("warehouse", "workers", "pickers");
			//find a picker that is not assigned
			broadcastMessage("warehouse", "workers", "pickers", new AreYouAssignedMessage());

		}
		if ( (picker.state == Workers.idle)  &&  tote!=null) {
```

```
                int orderToPick = tote.toteInfo.getOrderNumber();
                picker.tote = tote;
                picker.fetchList = (ArrayList<Integer>) Warehouse.getItemsAtStop(orderToPick,this);
                picker.reorderFetchList();
                //if picker is only doing one item at a time, only give him one item in the list
                if ((Warehouse.pickingMode == Warehouse.zoneSingle || Warehouse.pickingMode ==
Warehouse.discreteSingle) && picker.fetchList!=null && !picker.fetchList.isEmpty()) {
                        //only need one item in the list
                        ArrayList <Integer> oneItemList = new ArrayList<Integer>();
                        oneItemList.add(picker.fetchList.get(0));
                        picker.fetchList = oneItemList;
                }
                if (picker.fetchList!=null && !picker.fetchList.isEmpty()) {
                        picker.itemFetch = picker.fetchList.get(0);
                }
                else {
                        picker.itemFetch = -1;
                }
                //deep copy the item list into original item list (needed so that these items can be added the tote later)
                picker.originalFetchList = new ArrayList <Integer>();
                for (int i =0;i<picker.fetchList.size();i++) {
                        picker.originalFetchList.add(picker.fetchList.get(i));
                }
                picker.state = Workers.gettingItem;
                //the next time the "move" method is called for the Picker it should get a route and go get this item

                }
        }

    /** Called from conveyor scheduler. It checks to see if the order
     * in the tote is complete for this station
     */
    public void checkToteStatus(){
        if (tote != null) {
                if (Warehouse.getNextItemAtStop(tote.toteInfo.orderNumber,this)==-1){
                        //send the tote out of here!!
                        logger.fine("Pushing tote off");
                        shiftToteTo(segmentInfo.getNextLocation());
                        tote = null;
                        picker.tote = null;
                        segmentInfo.setHasTote(false);
```

```
                }
            } //if tote is null tell the spawner
            if (tote==null) {
                //don't worry, the Timer will get the Warehouse order processer to send a tote with an order
            }
        }

        /** used to move the tote out of the item stop */
        private void shiftToteTo(Point p){
            sendMessage("warehouse","conveyorsystem","warehouse",new MoveTote(segmentInfo.getSegmentLocati
on(),p,tote));
        }

        /** receives messages from the warehouse and other agents in the system */
        public void receiveMessage(Message m) {
            if (m != null) {
                if(logger != null) {
                    logger.finer("Item Stop got a Message from " + m.getSender()+ " says: " +m.toString());
                }
                //check what kind of message this is...
                if ((m instanceof ToteMessage)) {
                    Tote tote = ((ToteMessage) m).getTote();
                    List <Integer> order = Warehouse.getOrders().get(tote.toteInfo.getOrderNumber());
                    logger.fine("We got a message that a new tote has arrived with an order (number
"+tote.toteInfo.getOrderNumber()+") that has "+order.size()+" items to fetch in it");
                    addToteToConveyor(tote);
                    //tell picker about the tote
                    picker.tote = tote;
                    picker.state = Workers.idle; //worker is idle until he has an item to fetch
                }    else {
                    System.out.println("Item Stop got a strange message from "+ m.getSender());
                }
            }//m is null??
        }

/** this is how a tote gets put on the Item Stop and how it leaves when filled */
        public void addToteToConveyor(Tote tote) {
            if (this.tote==null) {//this means we do not already have a tote sitting on us
                //System.out.println("tote id is null; adding new tote");
                this.tote = tote;
                logger.fine("tote on conveyor");
            } else { //if we have a tote already and it has an unfilled order,
                //then add the new tote to the expansion area
```

```
                //shiftToteTo(segmentInfo.getNextLocation());
                //this.tote=tote;
            }
            if (tote!=null){
                segmentInfo.setStateColor(Color.green);
                segmentInfo.setHasTote(true);
                //System.out.println("tote id is not null; setting has tote");
            }
            else {
                segmentInfo.setStateColor(Color.white);
                segmentInfo.setHasTote(false);
                //System.out.println("tote id is null; setting not has tote");
            }
            toteMoved = true;
        }


/** external access to stop information */
        public StopInformation getStopInfo(){
            return stopInfo;
        }

        /** contains all the data for the stop, especially information on the picking domain */
        class StopInformation   {
            /** colors for the domains */
            protected Color[] colorList = {Color.GREEN,Color.WHITE,Color.PINK,Color.YELLOW,new Color(180,180,255),
Color.CYAN, Color.gray,new Color(180,255,255)};
            /** Color to paint the stop domains and workers that work in the domain */
            protected Color domainColor=Color.white;

            /** Domain is a set of item ids that are closest to the item stop */
            ArrayList <Integer> domain  = new ArrayList<Integer>();


            public StopInformation() {
                super();
                segmentInfo.stateColor = Color.white;
            }

            public StopInformation(int colorIndex) {
                super();
                this.domainColor = colorList[colorIndex];
```

```
        }

        public StopInformation(Color domainColor) {
            super();
            this.domainColor = domainColor;
        }

        /**
         * @return the domainColor
         */
        public Color getDomainColor() {
            return domainColor;
        }


        /**
         * @param domainColor the domainColor to set
         */
        public void setDomainColor(Color domainColor) {
            this.domainColor = domainColor;
        }


    }

}
```

ToteSpawn.java

```
/** ToteSpawn.java - One segment of a conveyor system. It generates new totes for the
 * simulation when a new order is received.
 */

package team64.madkit;

import java.awt.Point;
import java.util.logging.Level;

import madkit.kernel.Message;
import madkit.message.ActMessage;
import team64.madkit.messages.LocationMessage;
import team64.madkit.messages.LocationQueryMessage;
import team64.madkit.messages.MoveTote;
import team64.madkit.messages.NewLocationMessage;
import team64.madkit.messages.NewOrder;
import team64.madkit.messages.ToteMessage;



/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */

public class ToteSpawn extends ConveyorSegment
{

	/** Constructor */
	public ToteSpawn(int direction)
	{

		segmentInfo.direction = direction;

	}

	public void activate() {
		setLogLevel(Level.FINE);
		requestRole("warehouse","conveyorsystem", "spawner",null);	}
```

```
/** called when a New Order message is received from the Warehouse*/
	public void spawnTote(int orderNumber) {

		if(logger != null) {
			logger.finer("Scheduler has said to make a tote if there are more orders");
		}

		Tote tote = new Tote(orderNumber);
		//put the tote onto the conveyor system
		Point next = segmentInfo.getNextLocation();
		//send the tote to the next location
		sendMessage(Warehouse.getGrid()[next.x][next.y].getConveyorSegment(),new ToteMessage(tote));
	}

	public void spawnTote(Tote tote) {

		if(logger != null) {
			logger.finer("Scheduler has said to make a tote if there are more orders");
		}

		//put the tote onto the conveyor system
		Point next = segmentInfo.getNextLocation();
		//send the tote to the next location
		sendMessage(Warehouse.getGrid()[next.x][next.y].getConveyorSegment(),new ToteMessage(tote));
	}

	/** this automatically tries to find the next conveyor segment in the flow direction.
	 * This should only be used at startup (when the conveyor system is layed out
	 * because at runtime it is possible to dynamically change the next segment for redirecting
	 * totes.
	 *      right=0;
	 *  up=1;
	 *  left=2;
	 *  down=3;
	 **/
	public Point findxNextLocation(){
		Point location = (Point) segmentInfo.getSegmentLocation().clone();
		Point nextLocation = new Point(0,0);
		int direction = segmentInfo.direction;
		boolean notfound = true;
		//find the nearest regular conveyor segment
		//start looking in the direction, but if not found, start looking elsewhere
```

```
            // System.out.println("looking for next location with direction "+direction+" at "+ location);

            //in unit circle the points are in the form (cos a, sin a); converts direction to radians and adds (cos a, sin a);
          while (notfound) {
                  nextLocation = new Point (location.x + (int) Math.cos((Math.PI*direction)/2), location.y+
(int) Math.sin((Math.PI*direction)/2));
          }
            return nextLocation;
      }


      public void ejectTote() {
            if(logger != null) {
                  logger.finer("Scheduler has said that if I get a tote to eject it ...");
            }

      }

      private void shiftToteTo(Point p){
            //System.out.println("shifting tote "+ tote);
            sendMessage("warehouse","conveyorsystem","warehouse",new MoveTote(segmentInfo.getSegmentLocation(),p,t
ote));
      }

      public void receiveMessage(Message m) {
            if (m != null) {
                  if(logger != null) {
                        logger.finer("Tote Spawn got a Message from " + m.getSender()+ " says: " +m.toString());
                  }

                  //New Order message is sent by the Warehouse to start a new order
                  if ((m instanceof NewOrder)) {
                        int orderNumber = ((NewOrder) m).getOrder();
                        //spawn a tote with this order
                        spawnTote(orderNumber);
                  }
                  if ((m instanceof ToteMessage)) {
                        Tote tote = ((ToteMessage) m).getTote();
                        //spawn this tote
                        spawnTote(tote);
                  }
                  //end segment may send this message to find the tote spawner
```

```
                if ((m instanceof LocationQueryMessage)) {
                    //if the End segment is sending this then send back the NEXT segment, not our own location
                    if (m.getSender().getRole() == "end") {
                        sendMessage(m.getSender(),new LocationMessage(segmentInfo.getNextSegmentLocation
()));
                    } else {
                        sendMessage(m.getSender(),new LocationMessage(segmentInfo.getSegmentLocation()));
                    }
                }
                /*if ((m instanceof NewLocationMessage)) { //a reply from the Warehouse about the request to set the
location
                    Point p2 = ((NewLocationMessage) m).getNewLocation();
                    if(logger != null) {
                        logger.finer("Warehouse has said to set new location to: " + p2);
                    }
                    //could be the same as the old location, but put it into the new location in WorkerInfomation
anyway
                    segmentInfo.setSegmentLocation(p2);
                } */
                else if ((m instanceof ActMessage)) {
                    if (tote !=null) {
                        shiftToteTo(segmentInfo.getNextLocation());
                        tote=null;
                    }
                }
            }//m is null??
        }

        /* sets the next conveyor segment to transfer a tote */

}
```

End.java

```
/* End is a special segment of a conveyor system. This is where the tote exits with a full order */


package team64.madkit;

import java.awt.Color;
import java.awt.Point;
import java.util.logging.Level;

import madkit.kernel.Message;
import madkit.message.ActMessage;
import team64.madkit.messages.LocationMessage;
import team64.madkit.messages.LocationQueryMessage;
import team64.madkit.messages.MoveTote;
import team64.madkit.messages.NewLocationMessage;
import team64.madkit.messages.NewOrder;
import team64.madkit.messages.ToteMessage;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class End extends ConveyorSegment{

/** we need a phantom holding area for tote that arrived here
 *
 */

	public End(int direction)
	{
		segmentInfo.direction = direction;
		segmentInfo.nextSegmentLocation = null;
		//segmentInfo.setSegmentLocation(location);
	}

	public void activate() {
		setLogLevel(Level.FINE);
		requestRole("warehouse","conveyorsystem", "end",null);
		//segmentInfo.setSegmentLocation(segmentInfo.getSegmentLocation());
```

```
            //Warehouse.grid[segmentInfo.getSegmentLocation()]

        }



        public void toteNotMoved(){
            toteMoved = false;
        }

        public void moveTote() {
            if(!toteMoved){
                if(logger != null) {
                    logger.finer("Scheduler has said to move my tote to the next segment if I have one...");
                }
                if (tote!=null) {
                    //System.out.println("shifting tote");
                    shiftToteTo(segmentInfo.getNextLocation());
                    tote = null;
                    segmentInfo.setHasTote(false);
                }
            }
        }

        public void ejectTote() {
            if(logger != null) {
                logger.finer("Scheduler has said that if I get a tote to eject it ...");
            }

        }

        private void shiftToteTo(Point p){
            //System.out.println("shifting tote "+ tote);
            sendMessage("warehouse","conveyorsystem","warehouse",new MoveTote(segmentInfo.getSegmentLocati
on(),p,tote));
        }

        /** The next location for an end segment is kinda tricky.
        * For our model we will put the next location for the tote start (tote spawner) to
        * be the next location. The next location would be used if a tote ended up in the End
        * Segment but the order was not filled yet.
        */
        protected void getNextLocation() {
            sendMessage("warehouse","conveyorsystem", "spawner",new LocationQueryMessage());
```

```
    }

/** message handler for End Segment */
    public void receiveMessage(Message m) {
        if (m != null) {
            if(logger != null) {
                logger.finer("End Segment got a Message from " + m.getSender()+ " says: " +m.toString());
            }

            if ((m instanceof LocationMessage)) {
                //if this is from a Tote Spawner segment then this is the NEXT LOCATION for us
                if (m.getSender().getRole() == "spawner") {
                    segmentInfo.setNextLocation(((LocationMessage) m).getPoint());
                }
                return;
            }

            else if ((m instanceof ToteMessage)) {
                Tote tote = ((ToteMessage) m).getTote();
                int orderNumber = tote.toteInfo.orderNumber;
                if(Warehouse.getOrders().get(orderNumber)!=null){
                if (Warehouse.getOrders().get(orderNumber).isEmpty()){
                    Warehouse.removeOrderFromLists(orderNumber);
                    //System.out.println("romving");
                    //if order is complete destroy the tote
                    tote=null;

                }else {
                    //order is not complete so put the tote back on the conveyor system
                    Warehouse.ordersInProcess.remove((Object)orderNumber);
                    Warehouse.incompleteToteList.add(tote);
                    this.tote = null;
                }
                }

                /*              //check if ALL orders are finished
                if (Warehouse.getOrders().isEmpty()) {
                    //          Warehouse.recordEnd();
                }*/

            } else if
            ((m instanceof NewLocationMessage)) { //a reply from the Warehouse about the request to set the
```

```
location
                    Point p2 = ((NewLocationMessage) m).getNewLocation();
                    if(logger != null) {
                        logger.finer("Warehouse has said to set new location to: " + p2);
                    }
                    //could be the same as the old location, but put it into the new location in WorkerInfomation
anyway
                    segmentInfo.setSegmentLocation(p2);
                    return;
                }
                else if ((m instanceof ActMessage)) {
                    if (tote !=null) {
                        shiftToteTo(segmentInfo.getNextLocation());
                        tote=null;
                    }
                    return;
                }
                else {//no handler for this kind of message
                    System.out.println("End Segment got a strange message from "+ m.getSender());
                }
            }//m is null??
        }

        /* sets the next conveyor segment to transfer a tote */



        public void addToteToConveyor(Tote tote) {
            if (this.tote==null) {//this means we do not already have a tote sitting on us
                //System.out.println("tote id is null; adding new tote");
                this.tote = tote;
            } else { //we have a tote already so we have to eject our existing tote to the next conveyer segment
                //System.out.println("tote id is not null; adding new tote and shifting existing down");
                shiftToteTo(segmentInfo.getNextLocation());
                this.tote=tote;
            }
            if (this.tote!=null){
                segmentInfo.setStateColor(Color.green);
                segmentInfo.setHasTote(true);
                //System.out.println("tote id is not null; setting has tote");
            }
            else {
                segmentInfo.setStateColor(Color.white);
                segmentInfo.setHasTote(false);
```

```
                //System.out.println("tote id is null; setting not has tote");
            }
            toteMoved = true;
        }

}
```

StorageBin.java

```
/** basic shelf bin. A stack of these makes up a storage unit. */
package team64.madkit;

import java.awt.Color;
import java.awt.Point;
/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class StorageBin {

	private int quantity = 0;
	private int max;
	private int id = -1;
	private Point location;
	protected Color binColor = Color.lightGray; //default color for an unfilled bin

	/** Constructor */
	public StorageBin(int max, Point p) {
		super();
		this.max = max;
		location = p;
	}

	/** gets the bin location in the grid */
	public Point getLocation() {
		return location;
	}

	/** set the bin location in the grid */
	public void setLocation(Point location) {
		this.location = location;
	}

/** gets the number of items in the bin */
	public int getQuantity() {
		return quantity;
	}

	/** sets the number of items in the bin */
	public void setQuantity(int quantity) {
```

```
        this.quantity = quantity;
}

/** gets the maximum capacity */
public int getMax() {
        return max;
}

/** sets maximum capacity */
public void setMax(int max) {
        this.max = max;
}

/** gets the id of the item in the bin
 * -1 indicates no item
 */
public int getItemId() {
        return id;
}

/**sets the item id of the item in the bin.
 * If no item is in the bin this is set to -1
 */
public void setItemId(int id) {
        this.id = id;
}

/** add more items to the bin */
public void addItem(int quantity){
        this.quantity = this.quantity+quantity;
}



/**sets the bin Color */
public void setBinColor(Color binColor) {
        this.binColor = binColor;
}

/**gets the bin color */
public Color getBinColor() {
        return binColor;
```

```
    }


}
```

Tote.java

```
/** Totes go around on conveyers until filled with an order. They are used by workers to add
 * or remove items.
 * Totes can be moved by the conveyors or by workers.
 */

package team64.madkit;

import java.awt.Color;
import java.awt.Point;
import java.util.ArrayList;
import java.util.List;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */

public class Tote{

	static int capacity = 100; //the capacity of the tote is a fixed value (100)
	ToteInformation toteInfo;


	/** generic constructor */
	public Tote (){
		toteInfo = new ToteInformation();
	}

	/** construct a tote with an order number */
	public Tote(int orderNumber) {
		toteInfo = new ToteInformation();
		toteInfo.orderNumber = orderNumber;
	}


/** Constructor */
	class ToteInformation {
		int orderNumber = -1; // -1 means it has not been assigned to an order
		/** state color is used to show if a tote has anything in it or if it is empty */
```

```
	protected Color stateColor;
	List<Integer> contents = new ArrayList<Integer>(); //hashtable contains the order line item (key) and quantity of these in the tote
	int filled  = 0; //size of the items
	//final int max = 1;
	//Point location = new Point(0,0); //where the tote is in the warehouse


	/** Tote data. */
	public ToteInformation() {
		stateColor = Color.green;
	}

	/** gets the color to draw the tote in the graphics.
	 * @return the stateColor
	 */
	public Color getStateColor() {
		return stateColor;
	}


	/** sets the state color for the tote
	 * @param stateColor the stateColor to set
	 */
	public void setStateColor(Color stateColor) {
		this.stateColor = stateColor;
	}

	/**Gets the location of the tote in the warehouse
	 * @return the location
	 */
	//public Point gettLocation() {
	//	return location;
	//}

	/**Sets the location of the tote in the warehouse
	 * @param location the location of the tote
	 */
	//public void setLocation(Point location) {
	//	this.location = location;
	//}

	/** gets the order number for the tote. Does not have to be unique if more thatn one tote is needed for an order
```

```
         */
        public int getOrderNumber() {
            return orderNumber;
        }

        /** sets the order number for the tote.
         * a zero order number means the tote is available for any other job.
         */
        public void setOrderNumber(int orderNumber) {
            this.orderNumber = orderNumber;
        }

        /** adds one or more items that were fetched to the tote and
         * removes these items from the order
        */
        public void addToContents(ArrayList <Integer> items){
            for (int i=0;i<items.size();i++) {
                int itemID = items.get(i);
                contents.add(itemID);
                Warehouse.removeItemFromOrder(orderNumber,itemID);
                filled++;
            }

        }

    }

}
```

Scheduler Agents

Timer.java

```
/*
 * Tics every simulation step so we have a way to measure the number of simulation steps.
 * This timer calls two methods every tic - orderProcessor in Warehouse and tic in Warehouse.
 */
package team64.madkit;

import java.util.logging.Level;

import madkit.action.SchedulingAction;
import madkit.kernel.Message;
import madkit.kernel.Scheduler;
import madkit.message.SchedulingMessage;
import madkit.simulation.GenericBehaviorActivator;



/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class Timer extends Scheduler{



	private GenericBehaviorActivator <Warehouse>ticker;
	private GenericBehaviorActivator <Warehouse>orderProcessor;

	/** generic constructor */
  Timer(){
	}

  public void activate()
	{
		super.activate();
		getLogger().setWarningLogLevel(Level.INFO);
		setLogLevel(Level.FINE);
		requestRole("warehouse","warehouse","timer",null);
		ticker = new GenericBehaviorActivator<Warehouse> ("warehouse","warehouse","manager","tic");
		orderProcessor = new GenericBehaviorActivator<Warehouse>
```

```
("warehouse","warehouse","manager","orderProcessor");
            addActivator(ticker);
            addActivator(orderProcessor);
            setDelay(Warehouse.updateSpeed*3);
            //auto starting myself the agent way
            receiveMessage(new SchedulingMessage(SchedulingAction.RUN));
      }

            @Override
      protected void checkMail(Message m) {
            if (m != null) {

            }
            super.checkMail(m);
      }

            protected void end() {
                  //leaveRole("warehouse","workers","scheduler");
                  setDelay(99999);
                  if(logger != null)
                        logger.info("Workers are finished");
                  }


}
```

WorkerScheduler.java

```
*
 * Makes the workers move on their tasks every second
 */
package team64.madkit;

import java.util.logging.Level;

import madkit.action.SchedulingAction;
import madkit.kernel.AbstractAgent;
import madkit.kernel.Message;
import madkit.kernel.Scheduler;
import madkit.message.SchedulingMessage;
import madkit.simulation.GenericBehaviorActivator;



/**
 *
 * @author Team64
 */
public class WorkerScheduler extends Scheduler{


	private GenericBehaviorActivator<Picker> pickers;
	//private GenericBehaviorActivator<Sorter> sorters;
   private int humanfactor = 2;// humans move slower than the conveyors
	/** generic constructor */
	WorkerScheduler(){
	}

	public void activate()
	{
		super.activate();
		getLogger().setWarningLogLevel(Level.INFO);
		setLogLevel(Level.INFO);
		requestRole("warehouse","workers","scheduler",null);
		pickers = new GenericBehaviorActivator<Picker>("warehouse","workers","picker","move");
		//sorters = new GenericBehaviorActivator<Sorter>("warehouse","workers","sorter","move");
		addActivator(pickers);
		//addActivator(sorters);
		setDelay(Warehouse.updateSpeed*humanfactor);
		//auto starting myself the agent way
```

```
            receiveMessage(new SchedulingMessage(SchedulingAction.RUN));
        }

            @Override
        protected void checkMail(Message m) {
            if (m != null) {

            }
            super.checkMail(m);
        }

            protected void end() {
                //leaveRole("warehouse","workers","scheduler");
                setDelay(99999);
                if(logger != null)
                    logger.info("Workers are finished");
                }


}
```

ConveyorScheduler.java

```
/*
 * Makes the conveyor segments move totes to the next segment in the conveyor system
 */
package team64.madkit;

import java.util.logging.Level;

import team64.madkit.ItemStop.StopInformation;

import madkit.action.SchedulingAction;
import madkit.kernel.Message;
import madkit.kernel.Scheduler;
import madkit.message.SchedulingMessage;
import madkit.simulation.GenericBehaviorActivator;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class ConveyorScheduler extends Scheduler
{


	private GenericBehaviorActivator<ConveyorSegment> conveyorSegmentActivator;
	protected int conveyorSpeed = Warehouse.updateSpeed; //time in milliseconds for conveyor segment
movement


	/** generic constructor */
	ConveyorScheduler(){

	}

	public void activate() {

		super.activate();
		getLogger().setWarningLogLevel(Level.INFO);
		setLogLevel(Level.FINE);
		requestRole("warehouse","conveyorsystem","scheduler",null);
		conveyorSegmentActivator =
new GenericBehaviorActivator<ConveyorSegment>("warehouse","conveyorsystem",
```

```
"segment","moveTote");
            addActivator(new GenericBehaviorActivator<ConveyorSegment>("warehouse","conveyorsystem",
"segment","toteNotMoved"));
            addActivator(conveyorSegmentActivator);

            addActivator(new GenericBehaviorActivator<ConveyorSegment>("warehouse","conveyorsystem",
"itemstop","botherPicker"));
            addActivator(new GenericBehaviorActivator<ConveyorSegment>("warehouse","conveyorsystem",
"itemstop","checkToteStatus"));

            setDelay(conveyorSpeed);
            //auto starting myself the agent way
            receiveMessage(new SchedulingMessage(SchedulingAction.RUN));
      }

            @Override
      protected void checkMail(Message m) {
            if (m != null) {

            }
            super.checkMail(m);
      }

            protected void end() {
            //leaveRole("warehouse","conveyorsystem","scheduler");
            setDelay(9999999);
            if(logger != null)
                  logger.info("Conveyors are shut down");
            }



}
```

DisplayScheduler.java

```
/*
 * Used by the WarehouseWatcher to schedule updates to the graphical display
 */
package team64.madkit;

import java.util.logging.Level;

import madkit.action.SchedulingAction;
import madkit.kernel.Message;
import madkit.kernel.Scheduler;
import madkit.message.SchedulingMessage;
import madkit.simulation.GenericBehaviorActivator;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class DisplayScheduler extends Scheduler
{


	private GenericBehaviorActivator<Warehouse> warehouseViewerActivator;



	/** generic constructor */
	DisplayScheduler(){

	}

	public void activate() {

		super.activate();
		getLogger().setWarningLogLevel(Level.INFO);
		setLogLevel(Level.FINE);
		requestRole("warehouse","warehouse","scheduler",null);
		/** this sets up the way to call "updateDisplay" for the watcher when the scheduler ticks */
		warehouseViewerActivator =
new GenericBehaviorActivator<Warehouse>("warehouse","warehouse","watcher","updateDisplay");
		addActivator(warehouseViewerActivator);
		warehouseViewerActivator =
```

```
new GenericBehaviorActivator<Warehouse>("warehouse","warehouse","orderwatcher","updateDisplay");
            addActivator(warehouseViewerActivator);
            setDelay(Warehouse.graphicsSpeed);
            //auto starting myself the agent way
            receiveMessage(new SchedulingMessage(SchedulingAction.RUN));
      }

            @Override
      protected void checkMail(Message m) {
            if (m != null) {

            }
            super.checkMail(m);
      }

            protected void end() {
                  setDelay(99999);
            //leaveRole("warehouse","warehouse","scheduler");
            if(logger != null)
                  logger.info("Display scheduler is shut down");
            }


}
```

Watcher Agents (mostly used to draw graphics)

WarehouseWatcher.java

```
/*
 *Draws the warehouse graphic when probed by the DisplayScheduler
 */
package team64.madkit;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.util.List;
import java.util.logging.Level;

import javax.swing.JFrame;
import javax.swing.JLayeredPane;
import javax.swing.JScrollPane;

import madkit.kernel.AgentAddress;
import madkit.kernel.Watcher;
import madkit.simulation.PropertyProbe;
import team64.madkit.ItemStop.StopInformation;
import team64.madkit.Workers.WorkerInformation;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class WarehouseWatcher extends Watcher {


	private WarehouseGraphics display;
	//private ReceivingDockPanel receivingDockArea;
	private LunchRoomArea lunchroom;
	protected static PropertyProbe<ConveyorSegment, SegmentInformation> ppConveyors;
	protected static PropertyProbe<Workers,WorkerInformation> ppWorkers;
	protected static PropertyProbe<ItemStop, StopInformation> ppStops;
```

```
    protected static PropertyProbe<End, SegmentInformation> ppEnds;
    protected static PropertyProbe<ToteSpawn, SegmentInformation> ppSpawns;
    protected boolean drawGrid = true;
    protected boolean drawNumbers = true;
    protected boolean drawDomain =false;
    private Warehouse warehouse;


    public WarehouseWatcher(Warehouse warehouse) {
   this.warehouse = warehouse;
  // lunchroom = new LunchRoomArea();
    }

    @Override
    protected void activate()
    {
        super.activate();
        getLogger().setWarningLogLevel(Level.INFO);
        setLogLevel(Level.FINE);
        requestRole("warehouse","warehouse","watcher");
        //set up a property probe that watches for changes in the warehouse's conveyor(s)
        ppConveyors = new PropertyProbe<ConveyorSegment, SegmentInformation>("warehouse","conveyorsystem",
"segment","segmentInfo");
        ppStops = new PropertyProbe<ItemStop, StopInformation>("warehouse","conveyorsystem",
"itemstop","segmentInfo");
        ppEnds = new PropertyProbe<End, SegmentInformation>("warehouse","conveyorsystem", "end","segmentInfo");
        ppSpawns = new PropertyProbe<ToteSpawn, SegmentInformation>("warehouse","conveyorsystem",
"spawner","segmentInfo");
        //set up a property probe that watches for changes in the warehouse's workers
        ppWorkers = new PropertyProbe<Workers,WorkerInformation>("warehouse","workers","worker","workerInfo");
        //set up a property probe that watches for changes in the warehouse's totes

        addProbe(ppConveyors);
        addProbe(ppWorkers);
        addProbe(ppStops);
        addProbe(ppEnds);
        addProbe(ppSpawns);
        //List <AgentAddress> aa = getAgentsWithRole("warehouse", "warehouse", "manager");
        //Warehouse.myOwnAddress = aa.get(0);


    }
```

```
    @Override
    protected void end() {
        removeProbe(ppConveyors);
        removeProbe(ppStops);
        removeProbe(ppWorkers);
        removeProbe(ppEnds);
        removeProbe(ppSpawns);
        //sendMessage("warehouse","conveyorsystem", "launcher", new KernelMessage(KernelAction.EXIT));
        //sendMessage("warehouse","conveyorsystem", "scheduler", new
SchedulingMessage(SchedulingAction.SHUTDOWN));//stopping the scheduler
        leaveRole("warehouse","warehouse","watcher");
    }

    public void updateDisplay()
    {
        if(! (display.isVisible() && isAlive())){
            return;
        }
        if(logger != null) {
            logger.finest("Updating warehouse display ...");
        }
        display.repaint();
    }

    /** used by a watcher to paint the worker graphics */
    private void paintWorkers(Graphics g) {
        if (ppWorkers !=null) {
            List<Workers> workerList = ppWorkers.getShuffledList();
            int gridSize = WarehouseGraphics.gridSize;
            for (int i=0;i<ppWorkers.size();i++) {
                WorkerInformation worker=workerList.get(i).workerInfo;
                Color c = worker.getWorkerColor();
                g.setColor(c);
                Point loc = worker.getLocation();
                //need to convert the grid location to a graphical gridunit location
                Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);
                g.fillOval(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);
                if(logger != null) {
                    logger.finest("Painting a worker at x:" + graphicLocation.x + " y:"+graphicLocation.y);
                }
            }
        }
    }
```

```
/** used by a watcher to paint the grid units with bins */
private void paintShelves(Graphics g) {
	GridUnit[][] grid = Warehouse.getGrid();
	int gridSize = WarehouseGraphics.gridSize;
	g.setColor(Color.red);
	for (int i = 0; i < Warehouse.warehouseWidth; i++){
		for (int p = 0; p < Warehouse.warehouseHeight; p++){
			if(grid[i][p].containsBins()){
				Color domainColor = grid[i][p].getStopDomainColor();
				StorageBin bin = grid[i][p].getBinAt(0);
				if ( bin!=null) {
					g.setColor(domainColor);
				} else {
					g.setColor(Color.red);
				}
	//			if(grid[i][p].getBinItems().isEmpty()){
				// Color fullColor = g.getColor();
				// Color newColor = fullColor.darker();
				// g.setColor(newColor);
			//  }

				Point graphicLocation= WarehouseGraphics.convertGridToDisplay(i, p);
				g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);
				if(logger != null) {
					logger.finest("Painting a Bin at x:" + graphicLocation.x + " y:"+graphicLocation.y);
				}

			}
		}
	}
}


/** used by a watcher to paint the conveyor systems graphics */
private void paintConveyors(Graphics g) {
	if (ppConveyors!=null){
		List<ConveyorSegment> segmentList = ppConveyors.getShuffledList();
		//System.out.println(segmentList);
		int gridSize = WarehouseGraphics.gridSize;
		for (int i=0;i<ppConveyors.size();i++) {
			SegmentInformation segment=segmentList.get(i).segmentInfo;
```

```
                    Point loc = segment.getSegmentLocation();
                    //need to convert the grid location to a graphical gridunit location
                    Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);
                    Point nextLocation = segment.getNextLocation();
                    Point nextLocationgraphics=
WarehouseGraphics.convertGridToDisplay(nextLocation.x,nextLocation.y);
                    Graphics2D g2d = (Graphics2D) g;
                    g2d.setStroke(new BasicStroke(4));
                    g2d.setColor(Color.black);
                    Point midpoint = new Point((((nextLocationgraphics.x+graphicLocation.x)/2)+gridSize/2),
(((nextLocationgraphics.y+graphicLocation.y)/2)+gridSize/2));
                    g2d.drawLine((graphicLocation.x+(gridSize/2)),(graphicLocation.y+
(gridSize/2)),midpoint.x,midpoint.y);
                    g2d.setStroke(new BasicStroke(1));
                    if (segment.getHasTote()){
                        g.setColor(Color.red);
                        //System.out.println("tote id is true; setting color to red");
                    }else{
                        //System.out.println("tote id is false; setting color to black");
                        g.setColor(Color.blue);
                    }
                    g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);

                    if(logger != null) {
                        logger.finest("Painting a conveyor segment at x:" + graphicLocation.x + "
y:"+graphicLocation.y);
                    }
                }
            }

            if (ppEnds!=null){
            List<End> endList = ppEnds.getShuffledList();
            int gridSize = WarehouseGraphics.gridSize;
            for (int i=0;i<ppEnds.size();i++) {
                SegmentInformation segment=endList.get(i).segmentInfo;
                Point loc = segment.getSegmentLocation();
                //need to convert the grid location to a graphical gridunit location
                Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);


                if (segment.getHasTote()){
                    g.setColor(Color.red);
                    //System.out.println("tote id is true; setting color to red");
                }else{
```

```
            //System.out.println("tote id is false; setting color to black");
            g.setColor(Color.cyan);
        }
        g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);

        if(logger != null) {
            logger.finest("Painting a conveyor segment at x:" + graphicLocation.x + " y:"+graphicLocation.y);
        }
}
}

if (ppSpawns!=null){
        List<ToteSpawn> spawnList = ppSpawns.getShuffledList();
        int gridSize = WarehouseGraphics.gridSize;
        for (int i=0;i<ppSpawns.size();i++) {
            SegmentInformation segment=spawnList.get(i).segmentInfo;
            Point loc = segment.getSegmentLocation();
            //need to convert the grid location to a graphical gridunit location
            Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);


            if (segment.getHasTote()){
                g.setColor(Color.red);
                //System.out.println("tote id is true; setting color to red");
            }else{
                //System.out.println("tote id is false; setting color to black");
                g.setColor(Color.pink);
            }
            g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);

            if(logger != null) {
                logger.finest("Painting a conveyor segment at x:" + graphicLocation.x + " y:"+graphicLocation.y);
            }
        }
        }


if (ppStops!=null){
        List<ItemStop> segmentList = ppStops.getShuffledList();
        //System.out.println(segmentList);
        int gridSize = WarehouseGraphics.gridSize;
        for (int i=0;i<ppStops.size();i++) {
```

```
                    StopInformation stop=segmentList.get(i).stopInfo;
                    SegmentInformation segment=segmentList.get(i).segmentInfo;
                    Point loc = segment.getSegmentLocation();

                    //need to convert the grid location to a graphical gridunit location
                    Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);
                    Point nextLocation = segment.getNextLocation();
                    Point nextLocationgraphics=
WarehouseGraphics.convertGridToDisplay(nextLocation.x,nextLocation.y);
                    Graphics2D g2d = (Graphics2D) g;
                    g2d.setStroke(new BasicStroke(4));
                    g2d.setColor(Color.black);
                    Point midpoint = new Point((((nextLocationgraphics.x+graphicLocation.x)/2)+gridSize/2),
(((nextLocationgraphics.y+graphicLocation.y)/2)+gridSize/2));
                    g2d.drawLine((graphicLocation.x+(gridSize/2)),(graphicLocation.y+
(gridSize/2)),midpoint.x,midpoint.y);
                    g2d.setStroke(new BasicStroke(1));
                    if (segment.getHasTote()){
                        g.setColor(Color.red);
                        //System.out.println("tote id is true; setting color to red");
                    }else{
                        //System.out.println("tote id is false; setting color to black");
                        g.setColor(Color.magenta);
                    }
                    g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);
                    Color c = stop.getDomainColor();
                    g.setColor(c);
                    if (drawDomain){ //drawn lines from the stop to its item bins in the
domain
                        for(int p = 0; p < stop.domain.size(); p++){
                            Point domainBinCoord =
Warehouse.getItemLocations().get(stop.domain.get(p)).getLocation();
                            Point domainBinGraphicCoord =
WarehouseGraphics.convertGridToDisplay(domainBinCoord);
                            //Integer id = segment.domain.get(p);
                            g.drawLine(graphicLocation.x, graphicLocation.y, domainBinGraphicCoord.x,
domainBinGraphicCoord.y);
                        }
                    }
                    if(logger != null) {
                        logger.finest("Painting a conveyor segment at x:" + graphicLocation.x + "
y:"+graphicLocation.y);
                    }
                }
```

```
        }
}



/** paint the lunchroom area */
private void paintLunchRoom(Graphics g) {
        g.setColor(lunchroom.fillColor);
        g.fillRect(lunchroom.x,lunchroom.y,lunchroom.width,lunchroom.height);
        g.drawLine(800,800,40,40);
}

/** paint the best path -- for testing */
private void paintPath(Graphics g) {
        g.setColor(Color.magenta);
        //List<Point> path=Warehouse.theBestPath;

        //for (int i=0;i<path.size();i++) {
        //   Point start = WarehouseGraphics.convertGridToDisplay(path.get(i).x, path.get(i).y);
        //  // Point end =  WarehouseGraphics.convertGridToDisplay(path.get(i+1).x, path.get(i+1).y);
        //System.out.println("Paint Point  " + start );
        //   g.fillOval(start.x+15,start.y+15,10,10);
        // }
        g.setColor(Color.orange);
        g.drawLine(800,800,40,40);
}




@Override
public void setupFrame(JFrame frame) {
        display = new WarehouseGraphics(){
                /** need to override the default so that updated information is used */
                protected void paintComponent(Graphics g) {
                        super.paintComponent(g);
                        paintShelves(g);
                        paintConveyors(g);
                        paintWorkers(g);
```

```
                //paintLunchRoom(g);

                //paintPath(g);
                if(logger != null) {
                    logger.finest("Graphics is " + g);
                }
                if (drawGrid) {
                    g.setColor(Color.black);
                    for (int column=0;column<Warehouse.warehouseWidth;column++) {
                        //draw a square the gridSize x gridSize
                        for (int row=0;row<Warehouse.warehouseHeight;row++) {
                            g.drawRect(column*gridSize,row*gridSize,gridSize,gridSize);
                        }
                    }
                }
                if (drawNumbers){
                    g.setColor(new Color(32,32,32));
                    for (int column=0;column<Warehouse.warehouseWidth;column++) {

                        for (int row=0;row<Warehouse.warehouseHeight;row++) {
                            if (column==0) {
                                g.drawString(""+row, 0,row*gridSize+10);
                            }
                            if (row==0) {
                                g.drawString(""+column, column*gridSize,10);
                            }
                        }
                    }

                }
            }
        };
        JLayeredPane jlp = new JLayeredPane();
        JScrollPane jsp = new JScrollPane(jlp);
        jlp.setPreferredSize(WarehouseGraphics.convertGridToDimension());
        jsp.setViewportView(jlp);

        jlp.add(display,0);
        jlp.add(new Canvas());

        //JScrollPane jsp = new
JScrollPane(display,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTAL_SCROLLBAR
_AS_NEEDED);
        //jsp.setViewportView(display);
```

```
            //frame.add(jlp);

            //display.add(lunchRoomArea);
            //frame.add(receivingDockArea);
            frame.getContentPane().add(jsp,BorderLayout.CENTER);
            frame.setVisible(true);
            frame.setPreferredSize(new Dimension(800, 800));
        }

        /** parameters for the lunchroom */
        public class LunchRoomArea {

            public Color fillColor = new Color(255,0,255,20); //light purple
            private int x = warehouse.getLunchroomLocation().x;
            private int y = warehouse.getLunchroomLocation().y;
            private int width = warehouse.getLunchroomSize().width;
            private int height = warehouse.getLunchroomSize().height;
            public Point location = new Point (x*WarehouseGraphics.gridSize,y*WarehouseGraphics.gridSize);
            public Dimension size =
new Dimension(width*WarehouseGraphics.gridSize,height*WarehouseGraphics.gridSize);
            public LunchRoomArea() {

            }

        }

}
```

WarehouseWatcher.java

```
*
 *Draws the warehouse graphic when probed by the DisplayScheduler
 */
package team64.madkit;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
```

```
import java.awt.Point;
import java.util.List;
import java.util.logging.Level;

import javax.swing.JFrame;
import javax.swing.JLayeredPane;
import javax.swing.JScrollPane;

import madkit.kernel.AgentAddress;
import madkit.kernel.Watcher;
import madkit.simulation.PropertyProbe;
import team64.madkit.ItemStop.StopInformation;
import team64.madkit.Workers.WorkerInformation;

/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class WarehouseWatcher extends Watcher {


	private WarehouseGraphics display;
	//private ReceivingDockPanel receivingDockArea;
	private LunchRoomArea lunchroom;
	protected static PropertyProbe<ConveyorSegment, SegmentInformation> ppConveyors;
	protected static PropertyProbe<Workers,WorkerInformation> ppWorkers;
	protected static PropertyProbe<ItemStop, StopInformation> ppStops;
	protected static PropertyProbe<End, SegmentInformation> ppEnds;
	protected static PropertyProbe<ToteSpawn, SegmentInformation> ppSpawns;
	protected boolean drawGrid = true;
	protected boolean drawNumbers = true;
	protected boolean drawDomain =false;
	private Warehouse warehouse;


	public WarehouseWatcher(Warehouse warehouse) {
	this.warehouse = warehouse;
	// lunchroom = new LunchRoomArea();
	}

	@Override
	protected void activate()
	{
```

```
            super.activate();
            getLogger().setWarningLogLevel(Level.INFO);
            setLogLevel(Level.FINE);
            requestRole("warehouse","warehouse","watcher");
            //set up a property probe that watches for changes in the warehouse's conveyor(s)
            ppConveyors = new PropertyProbe<ConveyorSegment, SegmentInformation>("warehouse","conveyorsystem",
"segment","segmentInfo");
            ppStops = new PropertyProbe<ItemStop, StopInformation>("warehouse","conveyorsystem",
"itemstop","segmentInfo");
            ppEnds = new PropertyProbe<End, SegmentInformation>("warehouse","conveyorsystem", "end","segmentInfo");
            ppSpawns = new PropertyProbe<ToteSpawn, SegmentInformation>("warehouse","conveyorsystem",
"spawner","segmentInfo");
            //set up a property probe that watches for changes in the warehouse's workers
            ppWorkers = new PropertyProbe<Workers,WorkerInformation>("warehouse","workers","worker","workerInfo");
            //set up a property probe that watches for changes in the warehouse's totes

            addProbe(ppConveyors);
            addProbe(ppWorkers);
            addProbe(ppStops);
            addProbe(ppEnds);
            addProbe(ppSpawns);
            //List <AgentAddress> aa = getAgentsWithRole("warehouse", "warehouse", "manager");
            //Warehouse.myOwnAddress = aa.get(0);


        }

        @Override
        protected void end() {
            removeProbe(ppConveyors);
            removeProbe(ppStops);
            removeProbe(ppWorkers);
            removeProbe(ppEnds);
            removeProbe(ppSpawns);
            //sendMessage("warehouse","conveyorsystem", "launcher", new KernelMessage(KernelAction.EXIT));
            //sendMessage("warehouse","conveyorsystem", "scheduler", new
SchedulingMessage(SchedulingAction.SHUTDOWN));//stopping the scheduler
            leaveRole("warehouse","warehouse","watcher");
        }

        public void updateDisplay()
        {
```

```
        if(! (display.isVisible() && isAlive())){
            return;
        }
        if(logger != null) {
            logger.finest("Updating warehouse display ...");
        }
        display.repaint();
    }

    /** used by a watcher to paint the worker graphics */
    private void paintWorkers(Graphics g) {
        if (ppWorkers !=null) {
            List<Workers> workerList = ppWorkers.getShuffledList();
            int gridSize = WarehouseGraphics.gridSize;
            for (int i=0;i<ppWorkers.size();i++) {
                WorkerInformation worker=workerList.get(i).workerInfo;
                Color c = worker.getWorkerColor();
                g.setColor(c);
                Point loc = worker.getLocation();
                //need to convert the grid location to a graphical gridunit location
                Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);
                g.fillOval(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);
                if(logger != null) {
                    logger.finest("Painting a worker at x:" + graphicLocation.x + " y:"+graphicLocation.y);
                }
            }
        }
    }

    /** used by a watcher to paint the grid units with bins */
    private void paintShelves(Graphics g) {
        GridUnit[][] grid = Warehouse.getGrid();
        int gridSize = WarehouseGraphics.gridSize;
        g.setColor(Color.red);
        for (int i = 0; i < Warehouse.warehouseWidth; i++){
            for (int p = 0; p < Warehouse.warehouseHeight; p++){
                if(grid[i][p].containsBins()){
                    Color domainColor = grid[i][p].getStopDomainColor();
                    StorageBin bin = grid[i][p].getBinAt(0);
                    if ( bin!=null) {
                        g.setColor(domainColor);
                    } else {
                        g.setColor(Color.red);
                    }
```

```
//                  if(grid[i][p].getBinItems().isEmpty()){
                        // Color fullColor = g.getColor();
                        // Color newColor = fullColor.darker();
                        // g.setColor(newColor);
                    //  }

                        Point graphicLocation= WarehouseGraphics.convertGridToDisplay(i, p);
                        g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);
                        if(logger != null) {
                            logger.finest("Painting a Bin at x:" + graphicLocation.x + " y:"+graphicLocation.y);
                        }

                    }
                }
            }
        }

    /** used by a watcher to paint the conveyor systems graphics */
    private void paintConveyors(Graphics g) {
        if (ppConveyors!=null){
            List<ConveyorSegment> segmentList = ppConveyors.getShuffledList();
            //System.out.println(segmentList);
            int gridSize = WarehouseGraphics.gridSize;
            for (int i=0;i<ppConveyors.size();i++) {
                SegmentInformation segment=segmentList.get(i).segmentInfo;
                Point loc = segment.getSegmentLocation();
                //need to convert the grid location to a graphical gridunit location
                Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);
                Point nextLocation = segment.getNextLocation();
                Point nextLocationgraphics= WarehouseGraphics.convertGridToDisplay(nextLocation.x,nextLocation.y);
                Graphics2D g2d = (Graphics2D) g;
                g2d.setStroke(new BasicStroke(4));
                g2d.setColor(Color.black);
                Point midpoint = new Point((((nextLocationgraphics.x+graphicLocation.x)/2)+gridSize/2),
(((nextLocationgraphics.y+graphicLocation.y)/2)+gridSize/2));
                g2d.drawLine((graphicLocation.x+(gridSize/2)),(graphicLocation.y+(gridSize/2)),midpoint.x,midpoint.y);
                g2d.setStroke(new BasicStroke(1));
                if (segment.getHasTote()){
                    g.setColor(Color.red);
                    //System.out.println("tote id is true; setting color to red");
                }else{
```

```
                        //System.out.println("tote id is false; setting color to black");
                        g.setColor(Color.blue);
                    }
                    g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);

                    if(logger != null) {
                        logger.finest("Painting a conveyor segment at x:" + graphicLocation.x + "
y:"+graphicLocation.y);
                    }
                }
            }

            if (ppEnds!=null){
            List<End> endList = ppEnds.getShuffledList();
            int gridSize = WarehouseGraphics.gridSize;
            for (int i=0;i<ppEnds.size();i++) {
                SegmentInformation segment=endList.get(i).segmentInfo;
                Point loc = segment.getSegmentLocation();
                //need to convert the grid location to a graphical gridunit location
                Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);


                if (segment.getHasTote()){
                    g.setColor(Color.red);
                    //System.out.println("tote id is true; setting color to red");
                }else{
                    //System.out.println("tote id is false; setting color to black");
                    g.setColor(Color.cyan);
                }
                g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);

                if(logger != null) {
                    logger.finest("Painting a conveyor segment at x:" + graphicLocation.x + " y:"+graphicLocation.y);
                }
            }
            }

            if (ppSpawns!=null){
                List<ToteSpawn> spawnList = ppSpawns.getShuffledList();
                int gridSize = WarehouseGraphics.gridSize;
                for (int i=0;i<ppSpawns.size();i++) {
                    SegmentInformation segment=spawnList.get(i).segmentInfo;
                    Point loc = segment.getSegmentLocation();
                    //need to convert the grid location to a graphical gridunit location
```

```
                Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);


                if (segment.getHasTote()){
                    g.setColor(Color.red);
                    //System.out.println("tote id is true; setting color to red");
                }else{
                    //System.out.println("tote id is false; setting color to black");
                    g.setColor(Color.pink);
                }
                g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);

                if(logger != null) {
                    logger.finest("Painting a conveyor segment at x:" + graphicLocation.x + " y:"+graphicLocation.y);
                }
            }
            }


        if (ppStops!=null){
            List<ItemStop> segmentList = ppStops.getShuffledList();
            //System.out.println(segmentList);
            int gridSize = WarehouseGraphics.gridSize;
            for (int i=0;i<ppStops.size();i++) {
                StopInformation stop=segmentList.get(i).stopInfo;
                SegmentInformation segment=segmentList.get(i).segmentInfo;
                Point loc = segment.getSegmentLocation();

                //need to convert the grid location to a graphical gridunit location
                Point graphicLocation= WarehouseGraphics.convertGridToDisplay(loc.x, loc.y);
                Point nextLocation = segment.getNextLocation();
                Point nextLocationgraphics= WarehouseGraphics.convertGridToDisplay(nextLocation.x,nextLocation.y);
                Graphics2D g2d = (Graphics2D) g;
                g2d.setStroke(new BasicStroke(4));
                g2d.setColor(Color.black);
                Point midpoint = new Point((((nextLocationgraphics.x+graphicLocation.x)/2)+gridSize/2),
(((nextLocationgraphics.y+graphicLocation.y)/2)+gridSize/2));
                g2d.drawLine((graphicLocation.x+(gridSize/2)),(graphicLocation.y+(gridSize/2)),midpoint.x,midpoint.y);
                g2d.setStroke(new BasicStroke(1));
                if (segment.getHasTote()){
                    g.setColor(Color.red);
                    //System.out.println("tote id is true; setting color to red");
```

```
                    }else{
                        //System.out.println("tote id is false; setting color to black");
                        g.setColor(Color.magenta);
                    }
                    g.fillRect(graphicLocation.x+gridSize/4,graphicLocation.y+gridSize/4,gridSize/2 ,gridSize/2);
                    Color c = stop.getDomainColor();
                    g.setColor(c);
                    if (drawDomain){ //drawn lines from the stop to its item bins in the
domain
                        for(int p = 0; p < stop.domain.size(); p++){
                            Point domainBinCoord =
Warehouse.getItemLocations().get(stop.domain.get(p)).getLocation();
                            Point domainBinGraphicCoord =
WarehouseGraphics.convertGridToDisplay(domainBinCoord);
                            //Integer id = segment.domain.get(p);
                            g.drawLine(graphicLocation.x, graphicLocation.y, domainBinGraphicCoord.x,
domainBinGraphicCoord.y);
                        }
                    }
                    if(logger != null) {
                        logger.finest("Painting a conveyor segment at x:" + graphicLocation.x + "
y:"+graphicLocation.y);
                    }
                }

            }
        }



    /** paint the lunchroom area */
    private void paintLunchRoom(Graphics g) {
        g.setColor(lunchroom.fillColor);
        g.fillRect(lunchroom.x,lunchroom.y,lunchroom.width,lunchroom.height);
        g.drawLine(800,800,40,40);
    }

    /** paint the best path -- for testing */
    private void paintPath(Graphics g) {
        g.setColor(Color.magenta);
        //List<Point> path=Warehouse.theBestPath;

        //for (int i=0;i<path.size();i++) {
        //   Point start = WarehouseGraphics.convertGridToDisplay(path.get(i).x, path.get(i).y);
```

```
        //  // Point end =  WarehouseGraphics.convertGridToDisplay(path.get(i+1).x, path.get(i+1).y);
        //System.out.println("Paint Point  " + start );
        //   g.fillOval(start.x+15,start.y+15,10,10);
        // }
        g.setColor(Color.orange);
        g.drawLine(800,800,40,40);
}




@Override
public void setupFrame(JFrame frame) {
        display = new WarehouseGraphics(){
                /** need to override the default so that updated information is used */
                protected void paintComponent(Graphics g) {
                        super.paintComponent(g);
                        paintShelves(g);
                        paintConveyors(g);
                        paintWorkers(g);

                        //paintLunchRoom(g);

                        //paintPath(g);
                        if(logger != null) {
                                logger.finest("Graphics is " + g);
                        }
                        if (drawGrid) {
                                g.setColor(Color.black);
                                for (int column=0;column<Warehouse.warehouseWidth;column++) {
                                        //draw a square the gridSize x gridSize
                                        for (int row=0;row<Warehouse.warehouseHeight;row++) {
                                                g.drawRect(column*gridSize,row*gridSize,gridSize,gridSize);
                                        }
                                }
                        }
                        if (drawNumbers){
                                g.setColor(new Color(32,32,32));
                                for (int column=0;column<Warehouse.warehouseWidth;column++) {
                                        for (int row=0;row<Warehouse.warehouseHeight;row++) {
                                                if (column==0) {
```

```
                                        g.drawString(""+row, 0,row*gridSize+10);
                                    }
                                    if (row==0) {
                                        g.drawString(""+column, column*gridSize,10);
                                    }
                                }
                            }

                        }
                    }
                };
                JLayeredPane jlp = new JLayeredPane();
                JScrollPane jsp = new JScrollPane(jlp);
                jlp.setPreferredSize(WarehouseGraphics.convertGridToDimension());
                jsp.setViewportView(jlp);

                jlp.add(display,0);
                jlp.add(new Canvas());

                //JScrollPane jsp = new
JScrollPane(display,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,JScrollPane.HORIZONTAL_SCROLLBAR
_AS_NEEDED);
                //jsp.setViewportView(display);
                //frame.add(jlp);

                //display.add(lunchRoomArea);
                //frame.add(receivingDockArea);
                frame.getContentPane().add(jsp,BorderLayout.CENTER);
                frame.setVisible(true);
                frame.setPreferredSize(new Dimension(800, 800));
        }

        /** parameters for the lunchroom */
        public class LunchRoomArea {

                public Color fillColor = new Color(255,0,255,20); //light purple
                private int x = warehouse.getLunchroomLocation().x;
                private int y = warehouse.getLunchroomLocation().y;
                private int width = warehouse.getLunchroomSize().width;
                private int height = warehouse.getLunchroomSize().height;
                public Point location = new Point (x*WarehouseGraphics.gridSize,y*WarehouseGraphics.gridSize);
                public Dimension size =
new Dimension(width*WarehouseGraphics.gridSize,height*WarehouseGraphics.gridSize);
                public LunchRoomArea() {
```

```
            }

        }

}
```

OrderWatcher.java

```
/*
 * A Watcher for changes in the warehouse orders
 */
package team64.madkit;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.logging.Level;

import javax.swing.JFrame;

import madkit.kernel.Watcher;
import madkit.simulation.PropertyProbe;


/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */

public class OrderWatcher extends Watcher {


	private OrderPanel display;


	protected static PropertyProbe<Warehouse, Warehouse> ppOrders;


	public OrderWatcher() {

	}

	@Override
	protected void activate()
	{
		super.activate();
		getLogger().setWarningLogLevel(Level.FINEST);
		setLogLevel(Level.FINE);
```

```
            requestRole("warehouse","warehouse","orderwatcher");
            //set up a property probe that watches for changes in the warehouse's orders(s)
            ppOrders = new PropertyProbe<Warehouse, Warehouse>("warehouse","warehouse",
"warehouse","ordersUpdate");
            addProbe(ppOrders);
      }

      @Override
      protected void end() {
            removeProbe(ppOrders);
            leaveRole("warehouse","warehouse","orderwatcher");    }

      public void updateDisplay() {
            if(! (display.isVisible() && isAlive())){
                  return;
            }
            if(logger != null) {
                  logger.finest("Updating orders display ...");
            }
            int oip = Warehouse.ordersInProcess.size();
            String orderList = "";
            if (oip>0) {
                  for (int i=0;i<oip;i++) {
                        orderList=orderList+","+Warehouse.ordersInProcess.get(i);
                  }
                  orderList = orderList.substring(1,orderList.length());
            }
            display.textArea.setText(oip+" Order(s) are in process ("+orderList+")\n");
            for (int i =0;i<Warehouse.numberOfItemStops;i++){
                  ItemStop stop = Warehouse.itemStops.get(i);
                  if (stop.tote==null) {
                        display.textArea.append("Item Stop at " + stop.segmentInfo.segmentLocation.x + ","+
stop.segmentInfo.segmentLocation.y + " does not have a tote\n");
                  } else {
                        int orderNumber =stop.tote.toteInfo.orderNumber;
                        int itemCount = Warehouse.getItemsAtStop(orderNumber,stop).size();
                        display.textArea.append("Item Stop at " + stop.segmentInfo.segmentLocation.x + ","+
stop.segmentInfo.segmentLocation.y + " has " + itemCount+" items to fetch for order " +  orderNumber+"\n");
                  }
            }
      }
```

```
        /** test */
        private void paintLunchRoom(Graphics g) {
                g.setColor(Color.red);
                g.fillRect(0,0,30,30);
        }


        @Override
        public void setupFrame(JFrame frame) {
                display = new OrderPanel();
                frame.getContentPane().add(display);
                frame.setVisible(true);
                frame.setPreferredSize(new Dimension(300, 300));
        }



}
```

OrderPanel.java

```
package team64.madkit;

import java.awt.Font;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;

public class OrderPanel extends JPanel {
	JTextArea textArea;

	public OrderPanel() {
		setSize(358,390);
		setLayout(null);


		JLabel lblNewAbel = new JLabel("Order Monitor");
		lblNewAbel.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 16));
		lblNewAbel.setBounds(10, 0, 205, 39);
		add(lblNewAbel);

		textArea = new JTextArea();
		textArea.setEditable(false);
		textArea.setLineWrap(true);
		textArea.setBounds(20, 50, 310, 256);
		add(textArea);
	}
}
```

Messages

AddTask.java

```
package team64.madkit.messages;

import java.awt.Point;
import java.util.ArrayList;
import java.util.List;

import team64.madkit.Tote;

import madkit.kernel.AgentAddress;
import madkit.kernel.Message;

/**
 * @author team64
 * @2012
 *
 */
public class AddTask extends Message {
   /** list of items for the worker to deal with */
      protected ArrayList <Integer> items;
      /** this is the location of the 0th (first) item in the list */
      protected Point location;
      /** the tote for the task */
      Tote tote;


      public AddTask(Point location, ArrayList items, Tote tote) {
            this.location = location;
            this.items =  items;
            this.tote = tote;
      }

/** gets the items list */
      public ArrayList getItems() {
            return items;
      }

/** sets the items list */
      public void setItem(ArrayList items) {
            this.items = items;
      }
```

```
/** gets the location of the first item on the list*/
	public Point getLocation() {
		return location;
	}

/** sets the location of the FIRST item in the list */
	public void setLocation(Point location) {
		this.location = location;
	}

	/** gets the tote agent*/
	public Tote getTote() {
		return tote;
	}




}
```

ColorMessage.java

```
package team64.madkit.messages;

import java.awt.Color;

import madkit.kernel.Message;

/**
 * @author team64
 * @2012
 *
 */

public class ColorMessage extends Message {


	protected Color color;


	public ColorMessage(Color color) {

		this.color = color;
	}


	public Color getColor() {
		return color;
	}


	public void setColor(Color color) {
		this.color = color;
	}




}
```

GoToDestinationMessage.java

```
package team64.madkit.messages;
/** Message sent to the worker by the warehouse. Worker knows where
 * it is and calculates a route to the destination.
 *
 */


import java.awt.Point;
import java.util.List;

import madkit.kernel.Message;


public class GoToDestinationMessage extends Message {
private Point destination;

public GoToDestinationMessage(Point destination) {
	super();
	this.setDestination(destination);
}
/** gets the destination point */
public Point getDestination() {
	return destination;
}

/** sets the destination point */
public void setDestination(Point destination) {
	this.destination = destination;
}


}
```

LocationMessage.java

```
package team64.madkit.messages;

import java.awt.Point;

import madkit.kernel.AgentAddress;
import madkit.kernel.Message;


public class LocationMessage extends Message {

	private Point point;

	public LocationMessage(Point point) {
		super();
		this.setPoint(point);
	}
/** access point */
	public Point getPoint() {
		return point;
	}
/** set point */
	public void setPoint(Point point) {
		this.point = point;
	}




}
```

LocationQueryMessage.java

```
package team64.madkit.messages;

import java.awt.Point;

import madkit.kernel.AgentAddress;
import madkit.kernel.Message;


public class LocationQueryMessage extends Message {



}
```

LunchMessage.java (used for testing)

```
package team64.madkit.messages;

import java.awt.Point;
import java.util.List;

import madkit.kernel.Message;


public class LunchMessage extends Message {


public LunchMessage() {

}




}
```

MoveTote.java

```
package team64.madkit.messages;

import java.awt.Point;

import madkit.kernel.Message;
import team64.madkit.Tote;
/**
 * @author  Team 64 New Mexico Supercomputing Challenge
 * @2012
 *
 */
public class MoveTote extends Message {
	private Point from;
	private Point destination;
	private Tote tote;

	public MoveTote(Point from, Point destination,Tote tote) {
		super();
		this.setFrom(from);
		this.setDestination(destination);
		this.setTote(tote);

	}

	/** gets the destination for this message */
	public Point getDestination() {
		return destination;
	}
	/** sets the destination point of the item */
	public void setDestination(Point destination) {
		this.destination = destination;
	}

	/** gets the tote agent address  */
	public Tote getTote() {
		return tote;
	}

	/** sets the tote agent address */
	public void setTote(Tote tote) {
```

```
        this.tote = tote;
    }

    /** gets the origin point of the tote */
    public Point getFrom() {
        return from;
    }

    /** sets the origin point of the tote */
    public void setFrom(Point from) {
        this.from = from;
    }

}
```

NewLocationMessage.java

```
package team64.madkit.messages;
/** Message sent to the warehouse by a worker, and warehouse replies with the same type of message.
 * The worker is requesting a change of location, while the warehouse replies with a valid destination.
 *
 */


import java.awt.Point;
import java.util.List;

import madkit.kernel.Message;


public class NewLocationMessage extends Message {
private Point oldLocation;
private Point newLocation;
private boolean move = true;

public NewLocationMessage(Point oldLocation, Point newLocation) {
	super();
	this.setOldLocation(oldLocation);
	this.setNewLocation(newLocation);
}

/** gets the old location (where the agent was before this move) */
public Point getOldLocation() {
	return oldLocation;
}

/** sets the old location (where the agent was before this move) */
public void setOldLocation(Point oldLocation) {
	this.oldLocation = oldLocation;
}

/** gets the new location (where the agent is going) */
public Point getNewLocation() {
	return newLocation;
}

/** sets the new location (where the agent is going) */
```

```
public void setNewLocation(Point newLocation) {
	this.newLocation = newLocation;
}

/** gets whether or not this is a move, otherwise new location is the same as the old location */
public boolean isMove() {
	return move;
}

/** sets whether or not this is a move */
public void setMove(boolean move) {
	this.move = move;
}

/*public NewLocationMessage(List <Point> directions) {
	super();
	this.directions = directions;
}
*/

}
```

NewOrder.java

```
/** This message is used when a new order is
 * received at a tote spawn so that the tote is
 * started with the correct order number
 */
package team64.madkit.messages;

import madkit.kernel.Message;

public class NewOrder extends Message {

	/** the order number for the tote */
	private int order;


	public NewOrder(int order) {
		super();
		this.setOrder(order);
	}

/** gets the order number for the tote that has the order*/
	public int getOrder() {
		return order;
	}

/** sets the order number for the tote that has the order*/
	public void setOrder(int order) {
		this.order = order;
	}

}
```

SetLocationMessage.java

```
package team64.madkit.messages;
/** Message sent to the warehouse any static object in the warehouse to notify the grid that it is in that location
 * (c) 2012 Team 64 New Mexico Supercomputing Challenge
*/



import java.awt.Point;

import madkit.kernel.Message;


public class SetLocationMessage extends Message {
private Point location;


public SetLocationMessage(Point location) {
	super();
	setlocation(location);

}

/** gets the location  */
public Point getlocation() {
	return location;
}

/** sets the location */
public void setlocation(Point location) {
	this.location = location;
}


}
```

ToteMessage.java

```
package team64.madkit.messages;

/** this is a message sent by a tote to any other agent */

import team64.madkit.Tote;
import madkit.kernel.AgentAddress;
import madkit.kernel.Message;


public class ToteMessage extends Message {

	protected Tote tote;

	public ToteMessage(Tote tote) {
		this.tote = tote;
	}

	/** gets the tote */
	public Tote getTote() {
		return tote;
	}

	/** sets the tote id */
	public void setTote(Tote tote) {
		this.tote = tote;
	}



}
```

Database Setup Programs

Items Generator

```
package team64.madkit.setup;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Random;

import javax.swing.JOptionPane;

public class ItemsGenerator {

	Connection conn = null;
   Statement st = null;


	/**
	 * @param args
	 */
	public static void main(String[] args) {
		@SuppressWarnings("unused")
		ItemsGenerator gen = new ItemsGenerator();

	}

	public ItemsGenerator (){

		String dburl = "jdbc:hsqldb:hsql://localhost/DistriButionCenter";
	String user = "SA";
	String password = "";

	try {
		//System.out.println( Class.forName("org.hsqldb.jdbcDriver"));
	   Class.forName("org.hsqldb.jdbcDriver").newInstance();

	   conn = DriverManager.getConnection(dburl, user, password);
	   System.out.println("Database connection established");
	} catch (Exception ex) {
		ex.printStackTrace();
	   System.out.println("Exception: " + ex.getMessage()
			+ ". Did you forget to start the database?");
	}
	int itemsToAdd;
	try{
	itemsToAdd = Integer.parseInt(JOptionPane.showInputDialog("How many active items would you like in the
database?"));
	} catch (NumberFormatException ex){
		System.err.println("Item Generation Aborted!");
		return;
	}
	try {

	   ResultSet rs = null;

	   st = conn.createStatement();
	   //System.out.println(st.toString());
	   // st.execute("CREATE TABLE Packages (id identity,Packages LONGVARCHAR, destinations Integer,total
```

```
BIGINT)");
         // st.execute("insert into packages (packages,destinations,total) values ('3,2,1,3,3,2,2,3,1,1,1,2',3,12)");



        // st.executeQuery("UPDATE ITEMS SET ACTIVE = FALSE");
        // rs.next();

         Random rand = new Random();
         for (int i = 0; i < itemsToAdd; i++){
            int supplier = (int) (Math.random()*10L)+1;
            double popularityBase = (rand.nextFloat());
            //double p1 = Math.pow(popularityBase,0.1)+1;
            //double p2 = Math.log(p1);
            //int popularity = (int)(Math.log(Math.pow(popularityBase,0.1)+1)*100);
            double p1 = Math.pow(popularityBase,0.5);
            int popularity = (int)(p1*100);
            int sizeIndex = (int) (Math.random()*26L);
            int[] possSizes = {5,5,5,5,5,5,5,5,10,10,10,10,10,10,10,25,25,25,25,33,33,33,50,50,100};
            st.executeQuery("insert into items (Size, Active, Suppliercsl,popularity) values ("+possSizes[sizeIndex]+", true,
"+supplier+", "+popularity+")");

         }
         st.close();
         System.out.println("Wrote " + itemsToAdd + " items to the database");
      } catch (Exception ex) {

         System.out.println("SQLException: " + ex.getMessage());
         ex.printStackTrace();
      }

      }



}
```

OrderListGenerator.java

```
package team64.madkit.setup;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Random;

import javax.swing.JOptionPane;

/** This tests setting up a list of orders for the simulation run.
 * It reads from the items table of the database to get the items for the
 * order.
 *
 * note: CREATE TABLE orders (id identity, qty smallint, items LONGVARCHAR);
  */


public class OrderListGenerator {

    Connection conn = null;
   Statement st = null;


    /**
     * @param args
     */
    public static void main(String[] args) {
        @SuppressWarnings("unused")
        OrderListGenerator gen = new OrderListGenerator();

    }

    public OrderListGenerator (){

        String dburl = "jdbc:hsqldb:hsql://localhost/DistriButionCenter";
    String user = "SA";
    String password = "";
    ResultSet rs = null;

    try {
        //System.out.println( Class.forName("org.hsqldb.jdbcDriver"));
       Class.forName("org.hsqldb.jdbcDriver").newInstance();

       conn = DriverManager.getConnection(dburl, user, password);
       System.out.println("Database connection established");
    } catch (Exception ex) {
        ex.printStackTrace();
       System.out.println("Exception: " + ex.getMessage()
            + ". Did you forget to start the database?");
    }
    //generate the order list based on the number of items in the warehouse
    int numberOfItems;
    try{
        numberOfItems = Integer.parseInt(JOptionPane.showInputDialog("How many different items in the
warehouse for this run?"));
    } catch (NumberFormatException ex){
        System.err.println("Order Generation Aborted!");
        return;
    }

    int numberOfOrders=10000;
```

```
        try {

            //create the table based on the number of items in warehouse
                String tableName = "ordersFor"+numberOfItems+"items";

            st = conn.createStatement();

            st.execute("CREATE TABLE " + tableName + " (id identity,qty smallint,items longvarchar);");


            Random rand = new Random();
            for (int i = 0; i < numberOfOrders; i++){
                int[] possOrderSizes = {1,1,2,2,2,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6,6,6,7,7,8,9,10,12};
                int orderSizeIndex = (int) (Math.random()*28L);
                int orderSize = possOrderSizes[orderSizeIndex];
                //generate the order list
                String orderList="";
                if (orderSize>1) {
                for (int j=0;j<orderSize;j++) {
                        //get a random item from the item table
                        String sql = "SELECT id FROM items where id<" + numberOfItems + " ORDER BY RAND()LIMIT 1";
                        rs = st.executeQuery(sql); // run the query
                    rs.next();
                    int itemID = rs.getInt("id");
                        orderList = orderList+","+itemID;
                }
                //strip the extra comma from the orderlist
                orderList=orderList.substring(1);
                String sql = "insert into " + tableName + " (qty,Items) values ("+orderSize+ ",'" + orderList+"')";
                st.executeQuery(sql);
                } // one item order
                else {
                        //get a random item from the item table
                        String sql = "SELECT id FROM items where id<" + numberOfItems + " ORDER BY RAND()LIMIT 1";
                        rs = st.executeQuery(sql); // run the query
                    rs.next();
                    int itemID = rs.getInt("id");
                    //store this in the order table
                    sql = "insert into " + tableName + " (qty,Items) values ("+orderSize+ ",'" + itemID+"')";
                    st.executeQuery(sql);
                }



            }
            st.close();
            System.out.println("Wrote " + numberOfOrders + " orders to the database");
        } catch (Exception ex) {

            System.out.println("SQLException: " + ex.getMessage());
            ex.printStackTrace();
        }

        }


}
```