Package Sorting Logistics

New Mexico Supercomputing Challenge

Final Report

April 6, 2011

Team 69

Los Alamos Middle School

Team Members:

Sudeep Dasari

David Murphy

Colin Redman

Teachers:

Wyatt Dumas

Mentors:

Elizabeth Cooper

Jim Redman

Executive Summary

Our project this year is to create a program that would simulate a package sorting facility and to examine factors that cause inefficiencies in the sorting times. We decided to try to simulate the huge Memphis sorting facility of FedEx. The FedEx facility is laid out with many circular conveyor systems with “run outs” between them and to where the boxes are loaded into cargo carriers. The packages come in on flights from around the world and are sorted into nine primary regions (based on population and location), which in turn are all sorted into nine other local regions (final destinations), each going out on one or more cargo planes[1]. The FedEx Memphis hub handles 150-160 incoming (and therefore outgoing) flights during the big night sort [2]. The night sort is the most critical sort for FedEx due to the limited time (about 10pm to 3am) in which the sort must be completed [1,2]. Our simulation is set up to try to sort packages in any random input order and find out how much time it takes to sort all the packages to their final destination cargo carriers. Our goal was to find out if human error and package spacing problems (collisions) would affect the results.

The simulation was created in a Java environment called Greenfoot [3]. We used Greenfoot to create a sorting facility that was similar to the FedEx Memphis hub. Greenfoot was used to layout the components of the sorting facility, which was represented by a grid in the background, with agents (actors) representing conveyers, sensors, box outlets, and box inputs. The program uses a database to store the package order used for the experiments and to store the experimental results.

We timed the sort for random package orders and assessed the efficiency of the facility we created inside the program using the time for all the boxes to be sorted to their final destination cargo carriers.

Our Team

Our team this year has three members; Colin Redman is our head programmer and has over four years of programming (Java) experience. He has been participating every year in the Supercomputing Challenge, since he was in 5th grade, so this is his fourth project. Sudeep Dasari is our project organizer and this is his third year of Supercomputing Challenge experience. David Murphy is our primary artist and writer. This is his first year doing a Supercomputing Challenge project. We all actively participated in all stages of the project including coding reviews, discussions, weekly meetings (every week except over Christmas break), research, and writing this report.

Problem Statement

We learned a lot about how FedEx and UPS sorts from our research. A YouTube video originally televised by the PBS Reading Rainbow gave us good background information [5]. Our problem is to study how sorting facilities work and how inefficiencies occur. Most shipment companies like FedEx and UPS lose millions of dollars due to inefficiencies in their sorting facilities because of a combination of weather delays, equipment breakdowns, and human errors [2]. A prime example of this is the large scale weather disaster suffered by a FedEx shipping facility in Memphis in 2004 [4]. Over a course of a few days during the Christmas rush Memphis was hit by an icy snow storm and planes that would normally take packages to and from the FedEx super hub were stopped cold when they needed to leave between 11pm and 3am. FedEx lost millions of dollars to the effects of the weather storm itself and lost even more money to the aftershock of delayed packages. The cause of the package delays were a combination of package collisions and other shipment problems. Problems like the weather disaster occurring at shipment facilities are a way of life and are unavoidable, however more efficient package movement, and

redesigned facilities will decrease the amount of money lost due to bad package handling. Every year FedEx and UPS experience similar situations and if they could sort the packages faster they may be able to avoid storms at either their sorting hub or at their final local destinations.

Before we can stop large scale problems like the large scale repercussions from poor package handling, we must first understand how companies like FedEx, and UPS lose money in a shipping facility. The most basic explanation for this is the more time a plane stays grounded at a runaway the more money is lost [1,2]. The planes can't take off without receiving every package that is sorted through the main packaging facility. The loss of money because of slow package movement is the main reason that efficient package logistics is so important. The main variable that contributes to lower package efficiency is package collisions and pileups [2]. These collisions are caused due to the crowded nature of most shipment facilities and errors in the packages spacing [1]. If a conveyor stops moving package pileups occur. When the force of the impacts are high enough, and the boxes smash into each other with enough force, as a conveyor fills up, some of the packages on a particular belt will be derailed [1]. The sorting system at FedEx relies on critical package spacing so that the machinery that reads the barcodes and signals the arms that push boxes off onto different conveyors will work correctly. The spacing of packages will become a problem if they start to pile up. The entire purpose of our project is to make the package logistics inside the shipment facility as efficient as possible. We will do this by creating a program with a set of algorithms that create a simulated facility, and move boxes around it as efficiently as possible.

Problem Solution

In order to solve this problem we started with plain Java without a graphical system but found that that we had trouble displaying the simulation. Then we tried NetLogo but we could not manage to find a way to make it handle the complexity of the problem. We found out how to add Java extensions to NetLogo but we still had problems with this solution. So we finally created a facility simulation in a Java environment called

Greenfoot [3]. Greenfoot is a live Java object world in which you can create multiple objects (agents) and tell them to do certain tasks. Each object can be separately coded in Java, but the environment takes care of the agent interactions. To simulate the FedEx facility we set up the conveyor system so we had multiple inputs (boxes placed into the sorting system) and multiple outputs (cargo carriers to various final destinations). We created a facility using multiple agents like conveyor tracks, nodes, inputs, outputs, and boxes. The facility represented by a grid in the background, with agent icons representing conveyers, box outlets, and box inputs. This facility was arranged in a nine by nine configuration. Because of our research we understood that the FedEx facility was laid out with circular conveyor systems, similar to (but much bigger than) the baggage carrousels in airports. FedEx has over 300 miles of conveyor belts in its Memphis sorting hub [6]. The main conveyor sorts to nine regional destinations, which in turn are all circular conveyor systems with eight to nine outputs (to the cargo carriers). We simulated this by making the circular conveyors straight, but assumed that a box that did not make it to its final destination on that conveyor would go around again (be put back to the start of the conveyor it is currently on). This is how we simulated a circular conveyor system.

In our original simulation the boxes' destination was set at random. For our final simulation each destination had the same number of boxes. In other words, if we were sorting 1200 boxes, and we had 40 final destinations, there would be 30 boxes randomly arranged for each destination in the experimental set of 1200 boxes. Our final simulation runs used 4050 boxes, 50 for each of the 81 destinations. A database was used to keep track of these box orders and destinations. The experiments (random box orders) were set up prior to running the boxes through the simulation and one set of these randomly ordered boxes was used at a time to run the simulation experiments. The results of each experiment were stored in the database along with which set of boxes were used to determine this result. Time results were compared for the same number of boxes sorted regardless of the order in which the boxes were sorted.

As an example of an experimental data set, the boxes are represented by a regional code and an local code. The range of regional codes was 1-9, and likewise the local codes are 1-9. In a simple system that only had 3 regional and 3 local destinations, and 18 packages (two for each destination) the list of packages (the “experiment”) could be something like: 21,22,13,21,33,33,11,12,23,32,11,12,13,22,23,31,31,32 where “21” would mean regional sort “2” and local sort “1”. In our final simulation we had 4050 boxes, 50 for each of the 81 destinations, leading to a random list of 4050 of these box destination, much larger than the simple example shown above.

The simulation starts by selecting one set of boxes at random from the database. Each box (agent) has knowledge of its destination. In a real sorting facility scanners are used to read the barcode destination on the box. The boxes then follow the route laid out by the conveyor belts to separate nodes where scanners (agents) tell the box whether or not it should exit to a different conveyor for local destination sorting. Each node then acted like a junction which changed the direction of the boxes’ route that lead to another set of conveyers and nodes, or an exit. The boxes traveled throughout this system until they reach their respective exits. When all boxes for a certain destination were fully sorted, the time was recorded and stored in the database.

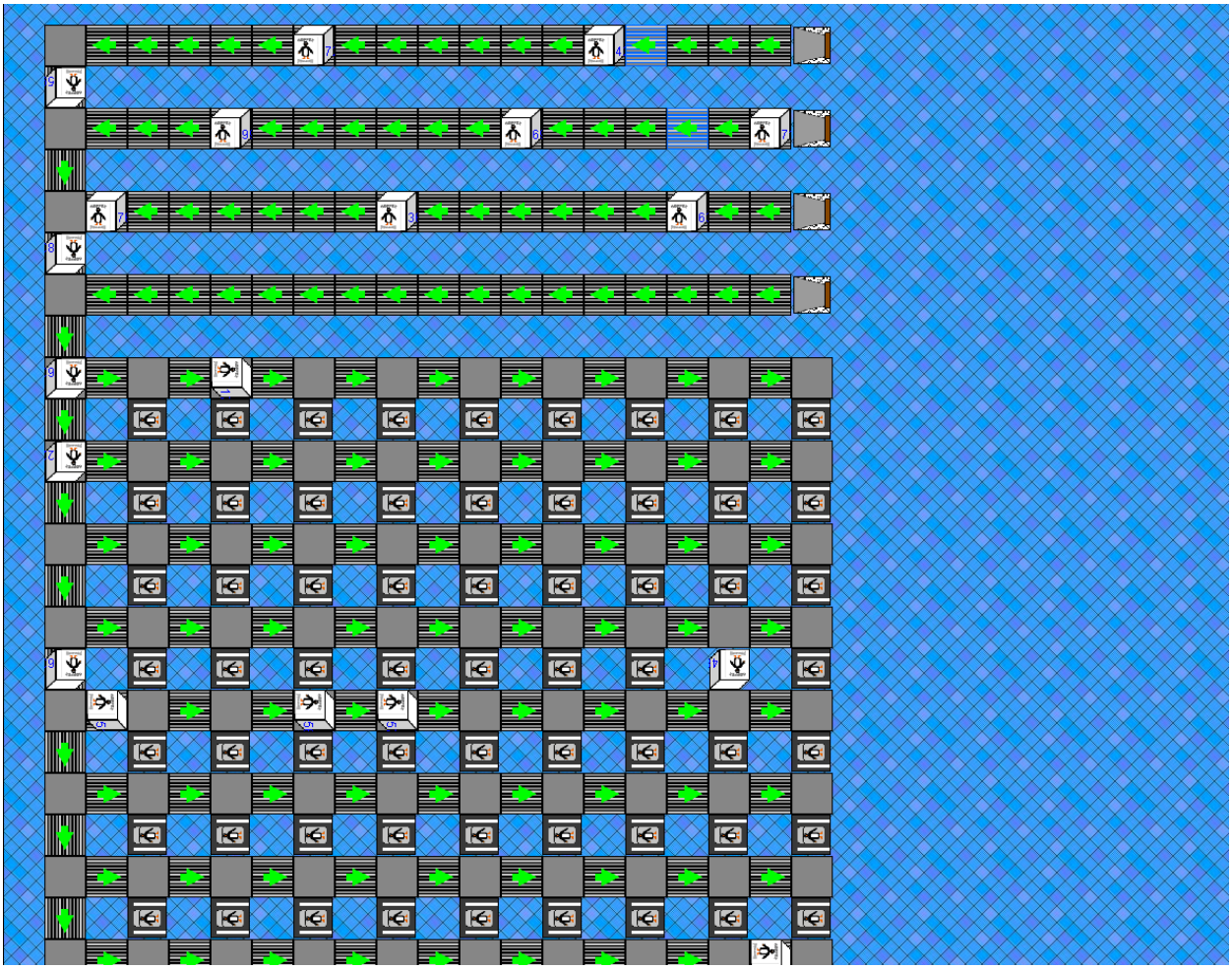

Figure 1. Part of the final simulation system showing 4 separate inputs and part of the multiple output system.

In reality humans normally handle the boxes twice during the whole sort [2,5]. The first time is when the boxes are added to the system (represented by the inputs in our simulation) and the last time when the boxes are loaded on their final cargo carriers. In the real FedEx system the operators take the boxes from the input cargo carriers and place them on the input conveyors at certain increments in time, (for example 4-5 seconds apart). The real FedEx system relies on this package spacing. The most likely place for human error is in the inputs. If the spacing is wrong in the real system the boxes will not scan correctly (barcode is not read correctly). Occasionally a box will be placed in the system face down so the bar code cannot be read at all [1,2,5]. If this is the case the box will not know its

destination in our simulation and must be “handled” again and placed back on the conveyor any other way other than face down.

One other way that package sorting is delayed is when the “run out” conveyors to the final destinations (cargo carriers) become so full that the conveyor system cannot handle any more boxes. The sort is delayed for this destination. We did not simulate this condition but it would be something that we might want to consider in the future.

Another factor that affects the sorting facility efficiency is that some cargo planes have longer distances to travel to get their sorted packages to their final destinations. For example, planes to the east coast and west coast should be allowed to leave first, or as soon as their packages are sorted and loaded. Some destinations, such as those to the mid-west, can be delayed in favor of letting these flights out first. This was not taken into account in our simulation but we might want to look at the effect of prioritizing the package sort so that if any delays or slowdowns during the sort occur they impact the higher priority destinations less.

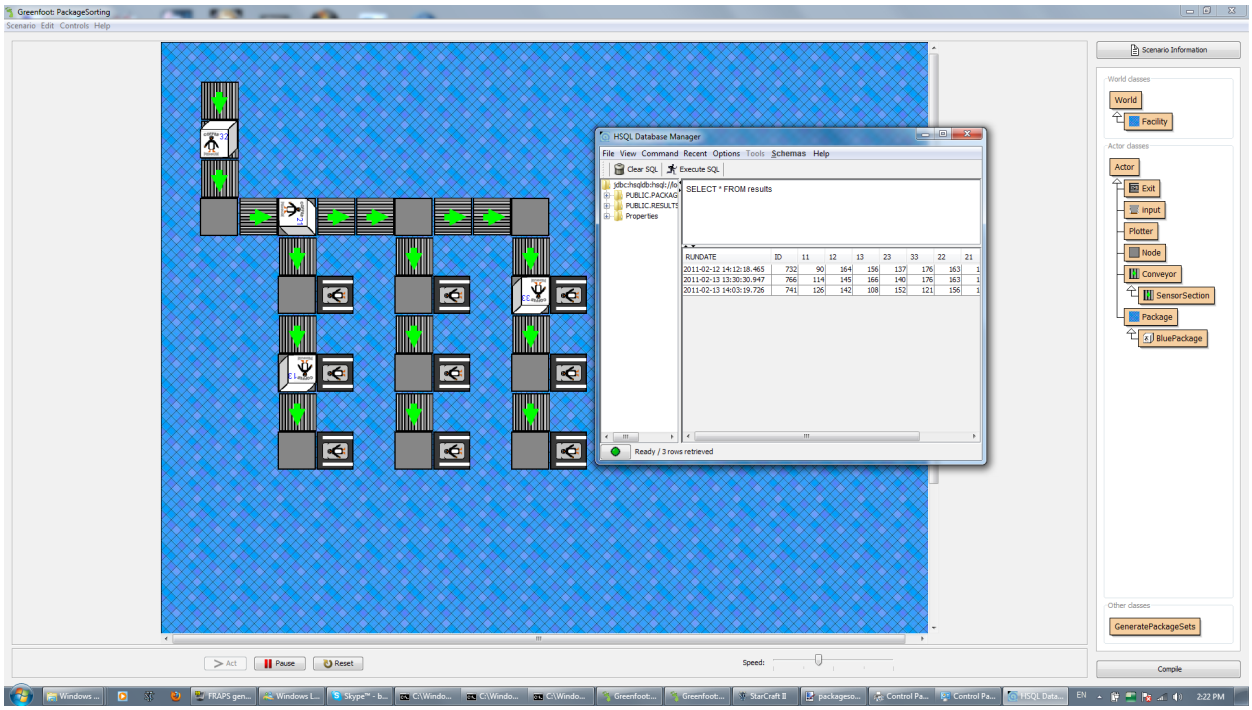

Figure 2. Image of one of our earliest simple models. This has one input and only 9 outputs (9 final destinations).

Code Summary

The conveyor, sensors (nodes), inputs and outputs of the simulation had to be hand coded for placement in the system so that the boxes had realistic routes through the system. This was very time consuming and the whole team participated in getting this task done. The box was the active agent object that contained data related to its current position and its final destination. If a box were the last one put into the system for a specific destination it also had a flag to indicate that the timer was to be stopped for the destination when it reached the output node (so the total time to sort all boxes to that destination was calculated). The conveyor agent moves a box from one side to the next (directional). The sensor nodes were at the critical locations where a box needed to be “scanned” to send it to a different conveyor if necessary.

Before the simulations were run, a separate Java program was used to add experiments to the database. Various numbers of boxes were used for the experiments as we developed and tested the simulation. We wanted to keep the box orders in the database in case we wanted to run the same experiment later with different parameters such as conveyor delays, weather problems, human errors, scanner errors, etc. Hundreds of experiments were set up in the database, each with a unique “experimenta” id. The results table records also contained the experiment number so we could keep the time results together with the time for the sorts, as well as the number of inputs, destinations, and total number of boxes sorted. Only time results for the same number of boxes, inputs, and destinations were compared.

The database we used runs as a Java application from the command line. This database is called HSQLDB, (HyperSQL DataBase), and is the leading SQL relational database engine written in Java [7]. It is used as the back-end for Open Office and many other open source commercial products. The main benefit for us is that it uses standard SQL, is Java

based (so we could access it from our program), and is portable (easy to move from one system to another).

The database is run from a jar which contains its JDBC driver and other supporting files. The specific database to run is set up in the server.properties file and the database is run from the hsqldb_1_8_1_1.jar.

```
set classpath=./hsqldb_1_8_1_1.jar;

java org.hsqldb.Server
```

SQL commands can be entered using the HSQLDB database manager which we used to set up and maintain the tables and to look at the results. The separate java program (GeneratePackageSets.java) was used to set up the experiments. It wrote the package orders to the Packages table of the database (along with the number of destinations and total number of packages). The experimental results data was stored in the Results table of the database and was written from the Greenfoot simulation code when the simulation was finished. The Greenfoot code also did database lookups to get the experimental starting conditions (the box orders) for specific conditions (number of inputs, destinations and number of boxes to sort).

When the simulation starts it opens a socket connection to the database and reads a random set of data for the experiment based on how many inputs, destinations, and packages are to be sorted. Then the simulation is started by taking packages one at a time from the package list and sending them into the input nodes. The next package is placed on the conveyor as soon as there is space (the input node is empty) until all packages in the experimental list are gone.

As the packages travel down the conveyor they will reach a sensor node that asks the package where it is going (if it needs to be redirected onto another conveyor system). The package knows its destination and this value is compared to the destinations accessible from the node. If the package needs to be put on the new conveyor system it is transferred to

that system. As destination nodes are passed on the system, the same process is applied. If a package needs to turn off to its final destination it will do that. When a package is correctly sorted to its output (cargo carrier) it is removed from the system. The time for the individual package to the final cargo carrier is not important. The only time that is recorded is the total time to sort the first to last package for the destination. When all packages are sorted the time results are recorded to the database for each destination. In real time it took about 5 minutes of computer time to run the 4050 boxes through the simulation at the fastest Greenfoot speed.

The code we wrote for the experimental set up and Greenfoot simulation is in Appendix A. The database table schemas are shown in Appendix B.

Results

We were able to program the simulation in Greenfoot and use the HSQLDB database to store our experimental parameters and results. We collected data from many experimental runs. The raw data (time it took to sort to each of the 81 destinations) was stored in the database. Data was extracted from the database for sample runs and this data was analyzed using a spreadsheet by “binning” or finding the number of destinations that took a certain range of time to complete. For example, a set of raw data for a 3 regional, 3 local (9 total destinations) might look like:

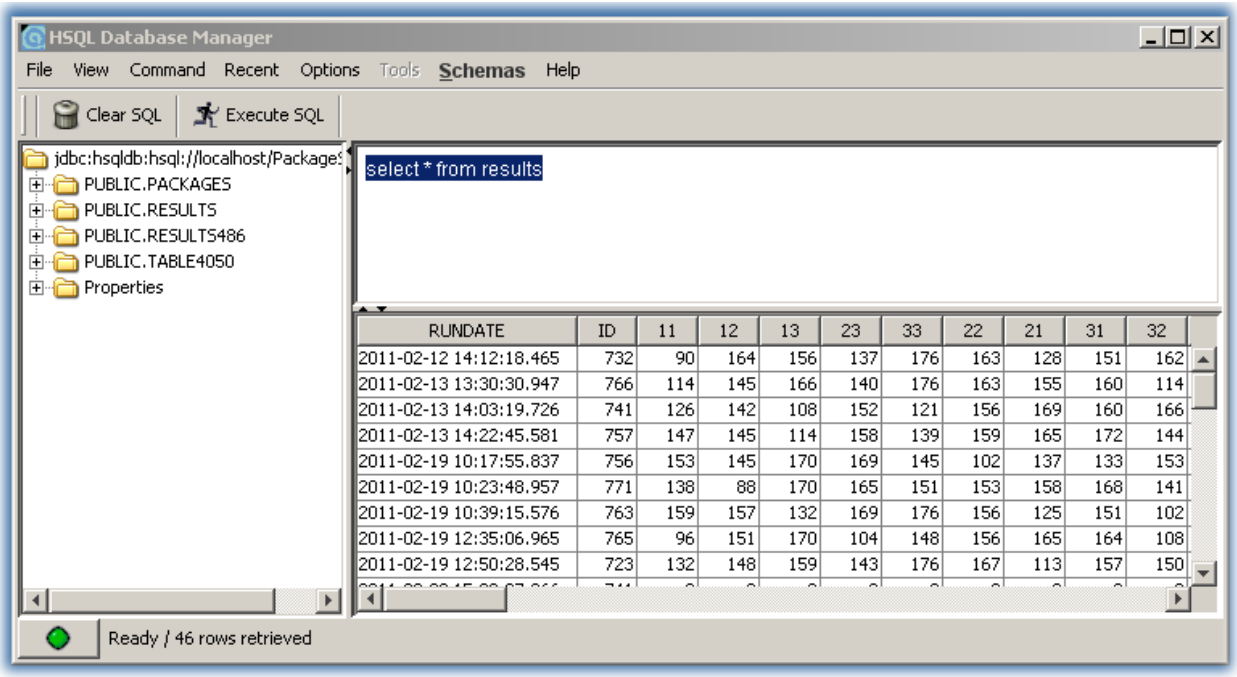

Figure 3. A set of experiments stored in the database for 9 destinations.

In our final simulation we had 81 columns of data, so it was much more complex, a table with 81 columns instead of the 9 shown above.

Time in the Greenfoot simulation was equivalent to Greenfoot “tics”. Each tic the boxes that are on the conveyors moved to a different conveyor section or were scanned and redirected to a new conveyor section, until they finally reached their exit. When all boxes for a certain exit (destination) were sorted, the total time (Greenfoot tics) was recorded for the database record. So the count of how many destinations completed for each 50 tic interval was calculated for these sample runs and the data examined.

Shown below are a few examples for experimental id’s are; 833, 1664, 969, and 951. The y-axis is the number of destinations completed in the time interval represented on the x-axis. Each bar represents a 50 time

tic interval.

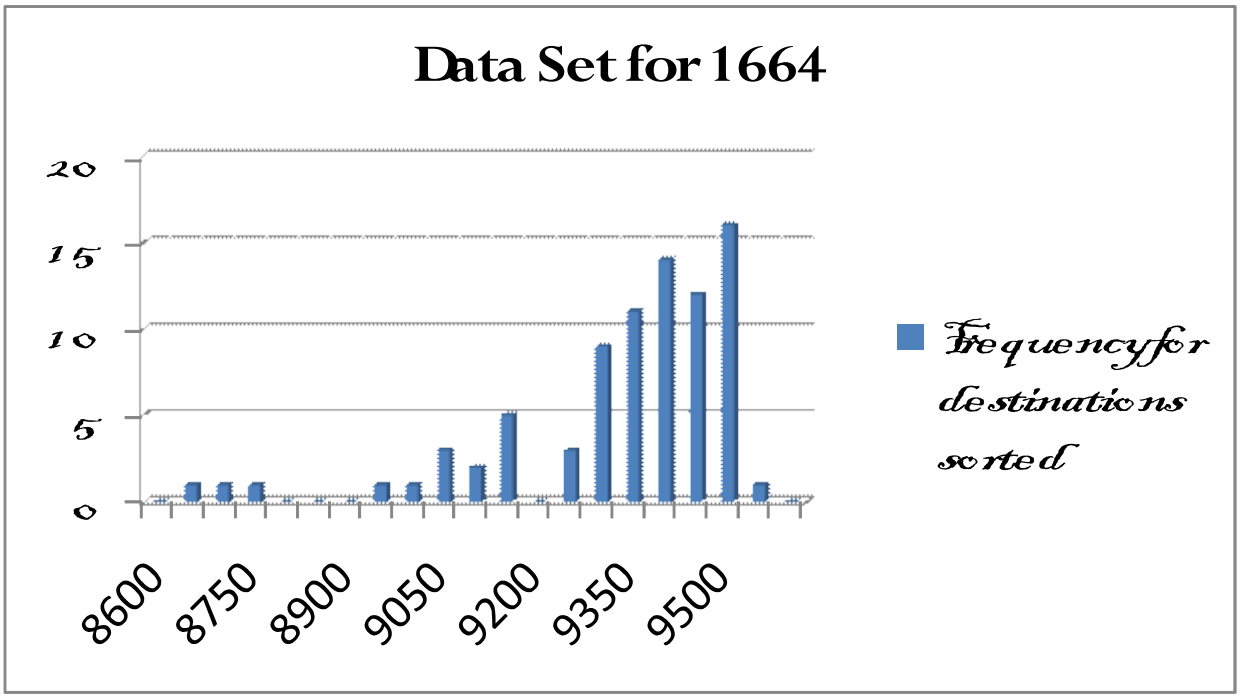

Figure 4- Frequency Graph of data set 1664, our median data set.

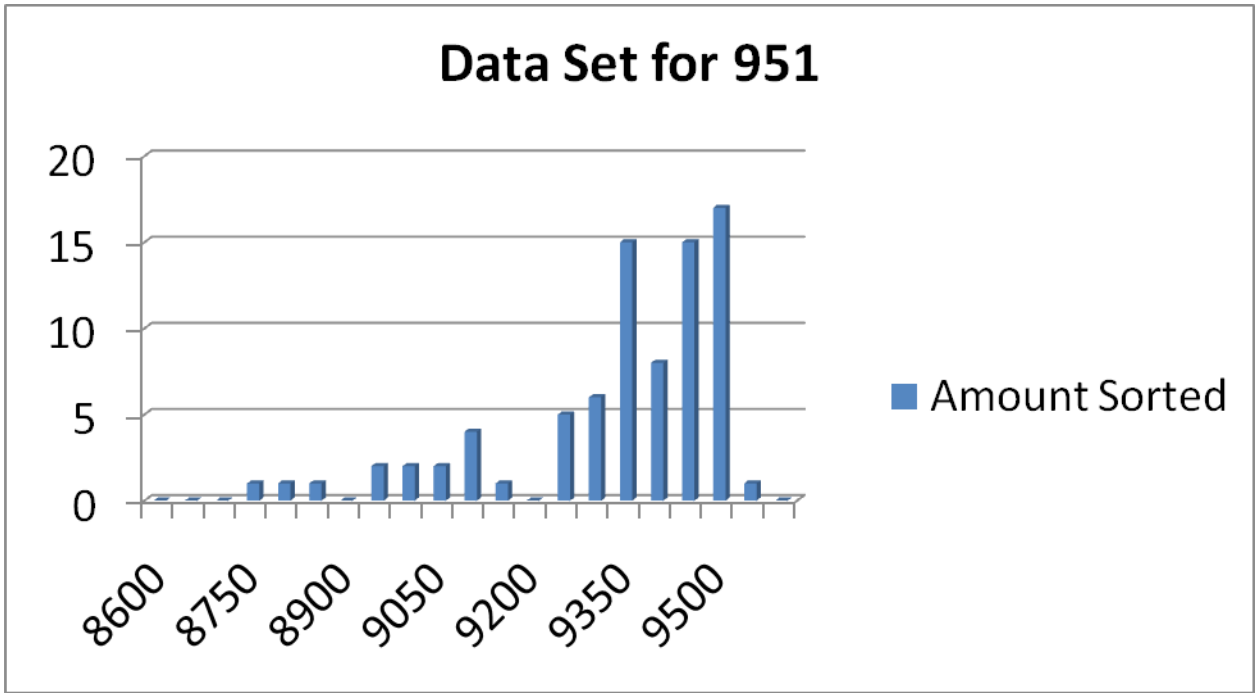

Figure 5- Frequency Graph of data set 951

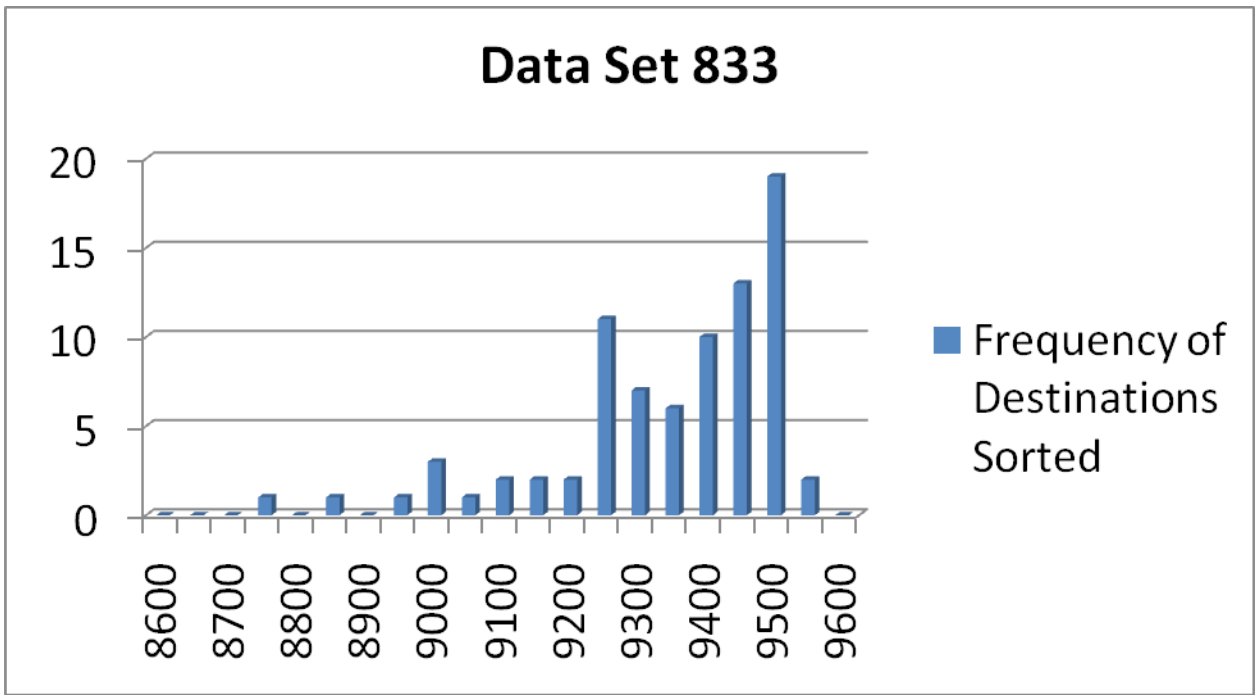

Figure 6- Frequency Graph of data set 833

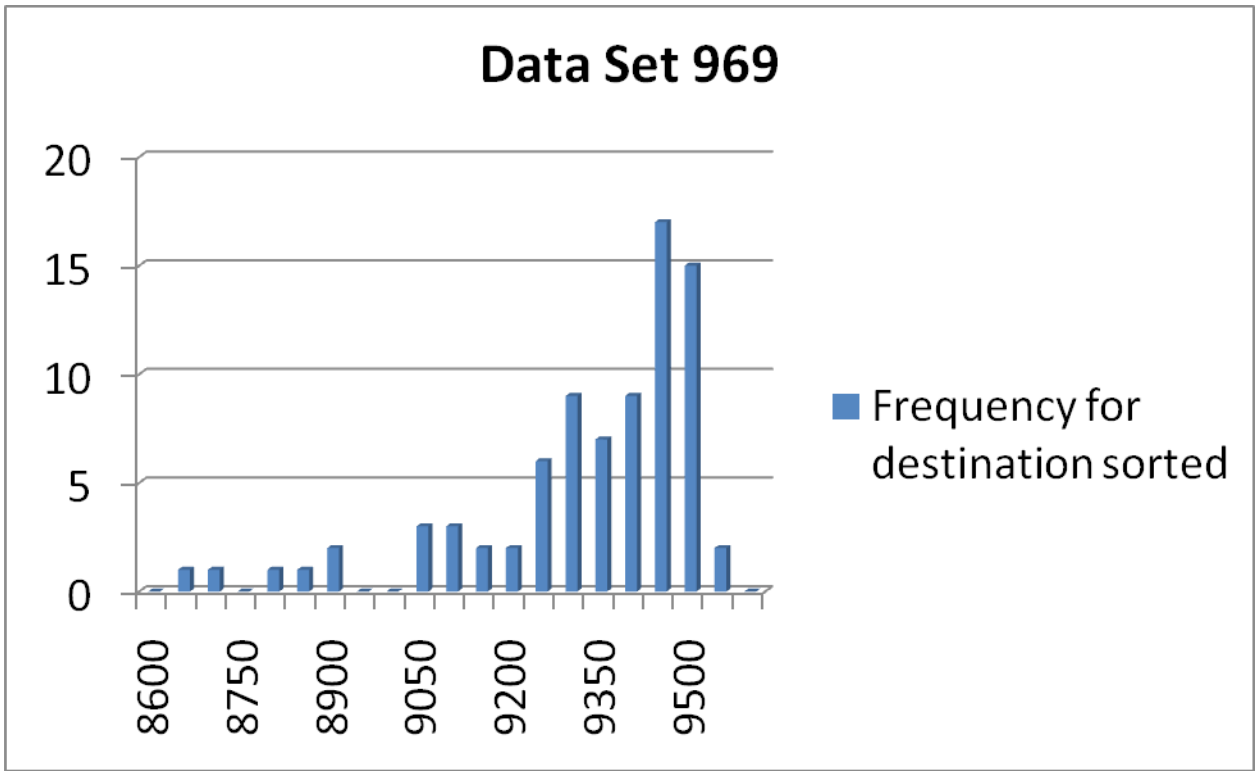

Figure 7- Frequency Graph of data set 969

Each set of data showed a fairly consistent pattern to sort the 4050 packages. A few destinations were sorted in a short time (8600 – 9200

Greenfoot tics). Most destinations took 9300-9500 Greenfoot tics, as seen by the peaks in these frequency charts. Generally the further away the destination, the more likely the boxes were to take more time to finish the sort to their destination. But if a lot of boxes were going to a single destination and were the last to enter the system this destination would end up with the longest sorting times.

One thing we could see from the results is that the distribution of sorting times was not a simple bell curve. It was skewed heavily toward the longer sorting times. Since the outbound flights could not leave for a destination until all packages are sorted for this destination, the skew toward the right (longer times) will have an impact on the overall efficiency of the system. If anything were to go wrong with the sort we would expect more of the destinations to fall into the longer sort times.

Conclusion

This year, we were able to create a mock version of the package facility, in Memphis, run by FedEx using the Greenfoot environment. We were also able to make the simulation sort boxes which had preprogrammed destinations stored in a database. The boxes were sorted throughout the facility, by the program, which then recorded the frequency for destination sorted, our primary result, in the database. The results clearly showed that there was a distribution of sorting times. The sort time distribution curve indicated that most of the destinations would be sorted at the longer times, and that the sort times were not evenly distributed.

Accomplishments

We prevailed over many obstacles throughout the course of our project's development. We made attempts to work in four programming languages or combinations of languages, Java, NetLogo, NetLogo with Java extensions, and finally Greenfoot with SQL, a database language that we didn't know about until recently. We also managed to replicate the Memphis Distribution Center owned by FedEx, allowing us to accurately test our program. We also learned a great deal about the Greenfoot environment and how a package makes its way from one part of the

country to its destination. We also learned how to use SQL, a database language.

Some of the most significant accomplishments were getting Greenfoot to run the complex conveyor system, getting the data (results) into the database after each simulation run. So one thing we did was integrate Greenfoot with HSQLDB which may have never been done before now.

Acknowledgments

Many domain experts advised us on the development of our project and we would like to recognize their valuable input and assistance. Paul Fisher aided us by explaining the functions of a distribution center and shared with us many other insights into the shipping business, which he worked in for many years. Jim Redman provided us with information during an interview on the issues that can occur in a FedEx facility. Elizabeth Cooper helped us with creating the simulation and reviewing the final report. Wyatt Dumas introduced us to the Greenfoot environment which we used to create our simulation. We would also like to thank Tom Laub for giving us feedback on the interim report and we also want to thank those who evaluated our project at the Northern New Mexico Community College.

Future Work

In future expansions of this project, that we may try next year, we will try to better replicate the sorting center environment, adding things such as conveyer belt breakdowns and other errors that could possibly occur in a sorting center. We may also try to replicate the results of Dr. Bowden's team that wrote the paper titled "Improving the Operation Intermodal Cargo Terminals Using Simulation and Optimization" which simulated the Hub in Memphis [8]. We might start by converting the source code they used for ProModel into Java code.

We also want to create another simulation that we can design that would theoretically be the most efficient yet feasible facility any shipment company could make. We will create it by eliminating what should be the

part of the facility most at risk. The “backbone” stretch of conveyer that ran the length of the entire facility. We will change the simulation of the real life facility into a more decentralized version that had a fork system. Each fork will have exits on both sides of either node in place in the fork system which created a much closer packed facility. It also got rid of the central backbone that would be a large collision risk in a real life facility. This design should allow more packages to be sorted throughout the facility while decreasing the amount of time that each box would be in travel.

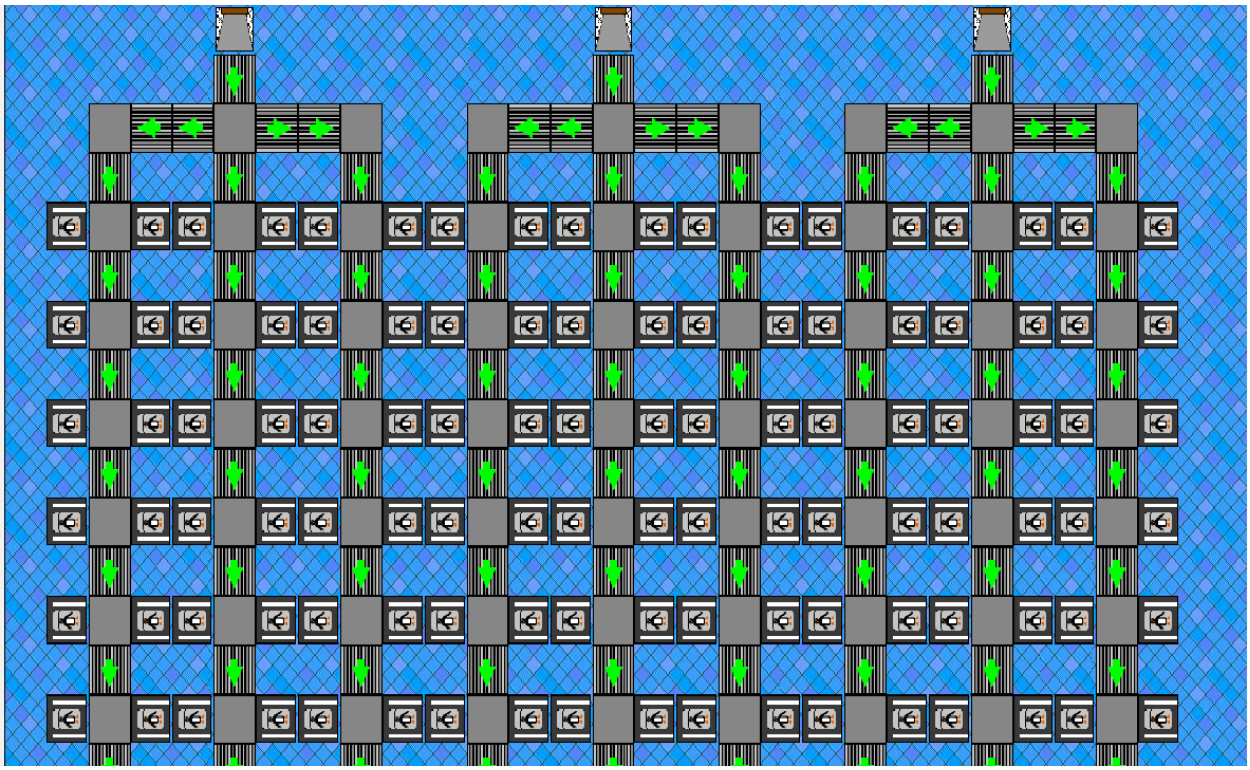

Figure 8. Part of the improved simulation showing the 3 inputs and the forks they divide into

We may also create another facility in Greenfoot that would be a, “perfect” scenario. The facility will consist of nine inputs each leading to nine outputs. The reason we called it the perfect loop is, in order to create a facility this simple, boxes would have to be half presorted going into the facility.

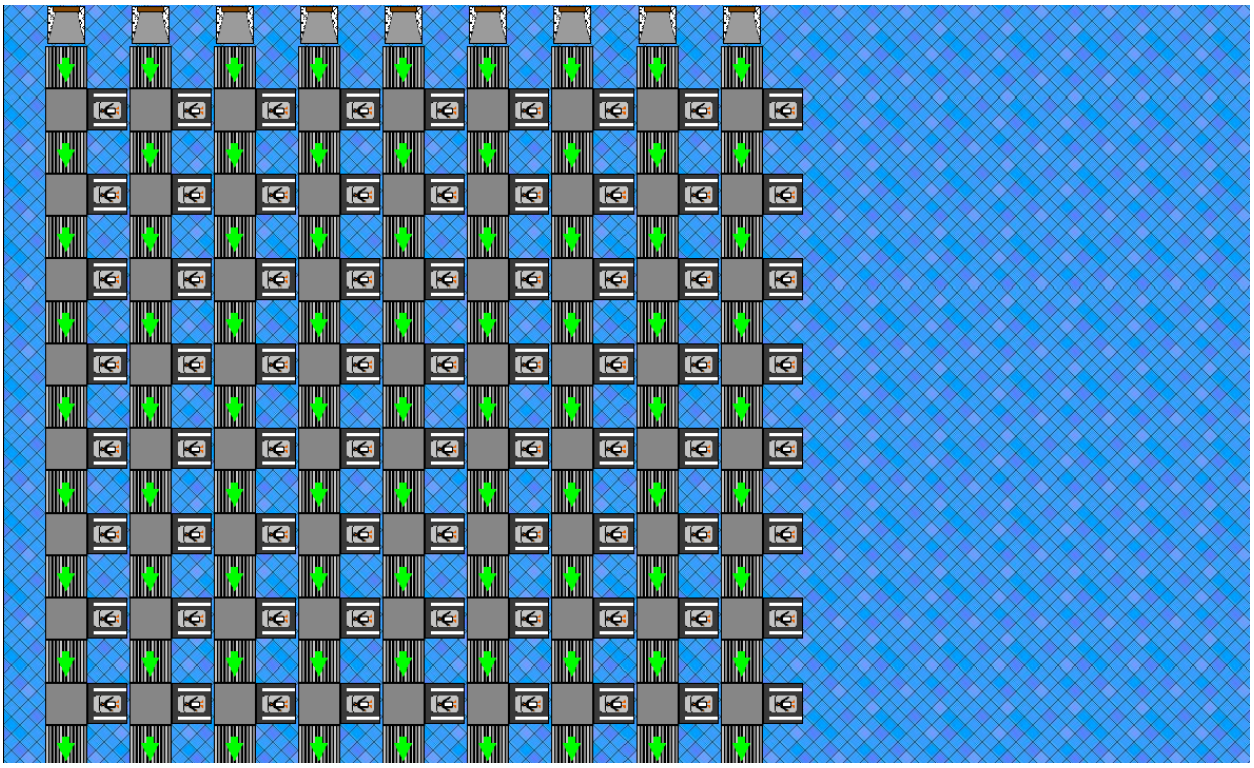

Figure 9. Part of the perfect loop simulation showing the 9inputs leading to the 9 outputs they are connected to

References

[1] "Jim Redman on How the Memphis FedEx Sorting Facility Is Arranged and the Problems That Occur." Personal interview.

[2] UPS and FedEx Air Hubs: Comparing Louisville and Memphis Cargo Hub Operations by Alex Cosmas and Bastien Martini, December 14th, 2007.

[3] Greenfoot (The Java Object World) http://www.greenfoot.org/

[4] IMPACTS OF THE DECEMBER 22, 2004 WINTER STORM ON FEDEX'S MEMPHIS OPERATION, Erik A. Proseus and Trevor K. Hansen, FedEx Express, Memphis, TN.

[5] Reading Rainbow, How FedEx Sorts Their Packages, *www.youtube.com/watch?v=Ug4GH3lPjaI* , 1998.

[6] FedEx Fact Sheet, “The FedEx Express Super Hub in Memphis, TN”, *http://news.van.**fedex**.com/.../**FedEx**%20Express%20Super%20Hub%20in%20Memphis.pdf*

[7] HSQLDB, http://hsqldb.org/, 100% Java Database

[8] R. Bowden, S. Gadiraju, G. Reginelli, and C.R. Cassady, *Improving the Operation of Overnight IntermodalCargo Terminals Using Simulation and Optimization*. Rep.FedEx. Web. 5 Apr. 2011

Appendix A. Source Code

GeneratePackageSets.java

This was used to create the many experimental randomly ordered packages for the simulation. These package lists were stored in the database table “Packages”.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Random;

//Utility to write all or some combinations of packages and
destinations to the database
public class GeneratePackageSets {
     static Connection conn = null;
     static Statement st = null;
     static int numberOfExits = 3;
     static int numberOfPackages = 4050;
     static boolean distributeEvenly = true;
     static int numberOfRuns = 1000;
     public static void main (String[] args) {
          String dburl =
"jdbc:hsqldb:hsql://localhost/PackageSorting";
          String user = "SA";
          String password = "";
          try
          {
               Class.forName("org.hsqldb.jdbcDriver");
               conn = DriverManager.getConnection (dburl, user,
password);
               System.out.println ("Database connection
established");
```

```
          }
          catch (Exception ex)
          {
               System.out.println("Exception: " +
ex.getMessage()+ ". Did you forget to start the database?");
          }
          //test database
          try {

               ResultSet rs = null;
               st = conn.createStatement();
               rs = st.executeQuery("SELECT COUNT(*)as rowcount
FROM packages");
               rs.next();
               System.out.println("Number of Package experiments
in database is : " + rs.getInt("rowcount"));
               st.close();
          }
          catch  (Exception ex) {
               System.out.println("SQLException: " +
ex.getMessage());
          }
          if (distributeEvenly) {
               for (int i=0;i<numberOfRuns;i++) {
                    String list =
generatePackageList(numberOfPackages,numberOfExits);
                    //check if this is not already in the
database
                    if
(checkDatabase(list,numberOfPackages,numberOfExits)){
                         //write new package list to the database
                         try {
                              st = conn.createStatement();
```

```
                              String sql = "INSERT into packages
(packages,destinations,total) values
('"+list+"',"+numberOfExits+","+numberOfPackages+")";
                              st.execute(sql);
                              st.close();
                         }
                         catch (Exception ex)
                         {
                              System.out.println("SQL Exception
trying to write new package list to database: " +
ex.getMessage());
                         }
                    }
               }

          }


     }

     /** generates a list of package destinations in some random
order.
      * This method assumes you want the same number of packages
going to each destination */
     public static String generatePackageList(int packages, int
destinations) {
          String packageList="";
          int numberForEachExit = packages/destinations;
          String packageDestinations[];
          Random r = new Random();
          packageList="1";
          for (int j=1;j<destinations;j++){
               packageList=packageList+","+(j+1);
          }
```

```
            for (int i=1;i<numberForEachExit;i++){
                 for (int j=0;j<destinations;j++){
                      packageList=packageList+","+(j+1);
                 }
            }
            //System.out.println(packageList);
            packageDestinations = packageList.split(",");
            for (int i=0;i<packages;i++){
                 //swap postions for two random values
                 int index1 = r.nextInt(packages);
                 int index2 = r.nextInt(packages);
                 String value1=packageDestinations[index1];

      packageDestinations[index1]=packageDestinations[index2];
                 packageDestinations[index2]=value1;
            }
            packageList=packageDestinations[0];
            for (int i=1;i<packages;i++) {

      packageList=packageList+","+packageDestinations[i];
            }
            System.out.println(packageList);
            return packageList;
      }

      /** this returns true if the package list is a new one (not
in the database) */
      public static boolean checkDatabase(String list,int
packages, int exits) {
            ResultSet rs = null;
            try {
                 st = conn.createStatement();
                 String sql = "select id,destinations from
packages where packages = '"+list+"'";
```

```
                rs = st.executeQuery(sql);
                //st.close();
                boolean found = rs.next();
                if (found) {
                     st.close();
                     return false;
                } else {
                     st.close();
                     return true;
                }

           } catch (SQLException e) {
                e.printStackTrace();
           }

           return false;
     }
}
```

Greenfoot Code

The skeleton code for each of these agents was generated by Greenfoot and edited to add custom code in the Greenfoot environment. Most of the code shown in this Appendix was written by us (the skeleton code for Greenfoot can be used as a guide but it does not do anything). Greenfoot compiles the changed code before each run and allows you to stop and start the simulation and set the time interval for the simulation clock so that it can be run faster or slower. Debugging in Greenfoot is not as convenient as in a real Java IDE such as Eclipse. Most of the time we have to put in println statements (which are commented out) to check on variable changes and program logic.

Facility.java

The facility is the simulation “world” or environment. This class contains all the conveyor, sensor, exits, and inputs setup for the simulation. Most of the database communication code is in this class. It also writes the results for the simulation run to the database. Interesting features of this code is the section where the agents are placed in their positions to form the conveyor system, and the SQL code that creates the database columns if they do not already exist (to avoid manually entering SQL for each of these 81 columns in the final simulation).

```
import greenfoot.Greenfoot;
import greenfoot.World;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.List;
import java.util.Random;

/**
 * Write a description of class facility here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Facility extends World {
    int step = 0;
    // String[] destinations = {"1","2","3"};
    int listIndex = 0;
    int totalPackages = 4050;
    int numberOfPrimaryExits = 9;
    int numberOfSecondaryExits = 9;
    int experimentsInDatabase=0;
    int packageSet = 1;//randomly determined for the run
    // packageList is read from the database
    String packagesList = "";
    String[] packageDestination = null;
    Connection conn = null;
    Statement st = null;
    int finalStep;

    /**
```

```
     * Constructor for objects of class facility.
     *
     */
    public Facility() {



        // Create a new world with 600x400 cells with a cell
size of 1x1 pixels.
        //getBackground().scale(60,60);

        super(30, 30, 40, true);

        // connect to database so we have a source of packages
        // String dburl = "jdbc:hsqldb:data/PackageSorting";
        String dburl =
"jdbc:hsqldb:hsql://localhost/PackageSorting";
        String user = "SA";
        String password = "";
        String drivename = "org.hsqldb.jdbcDriver";
        try {
            Class.forName("org.hsqldb.jdbcDriver");
            conn = DriverManager.getConnection(dburl, user,
password);
            System.out.println("Database connection
established");
        } catch (Exception ex) {
            System.out.println("Exception: " + ex.getMessage()
                    + ". Did you forget to start the database?");
        }
        // test database
        try {

            ResultSet rs = null;

            st = conn.createStatement();
            rs = st.executeQuery("SELECT COUNT(*)as rowcount
FROM packages where (total="+totalPackages+" and
primarysort="+numberOfPrimaryExits+" and
secondarySort="+numberOfSecondaryExits+")");
            rs.next();
            experimentsInDatabase=rs.getInt("rowcount");
            System.out
                    .println("Number of Package experiments for
"+ totalPackages +" packages and " + numberOfPrimaryExits + "
primary exits in database is : "
                            + experimentsInDatabase);
```

```
            st.close();
        } catch (Exception ex) {
            System.out.println("SQLException: " +
ex.getMessage());

        }
        setUpFullLoop();
    }

    int timer = 3;

    public List checkNodeNeighbours() {
        List nodes = getObjects(Node.class);
        for (int i = 0; i < nodes.size(); i++) {
            Node node = (Node) nodes.get(i);
            node.checkNeighbours(this);

        }
        return nodes;
    }

    public void act() {

        if (getObjects(BluePackage.class).size()==0){
            endSimulation();
        }
        step++;
        /*if (timer == 3 && (listIndex < totalPackages)) {//
make sure we still
                                                        // have
some packages to
                                                        // sort
            // BluePackage pack = new BluePackage
            // (destinations[Greenfoot.getRandomNumber(3)]);
            BluePackage pack = new
BluePackage(packageDestination[listIndex]);
            // if this is the LAST package mark the box
            if (listIndex == totalPackages - 1) {
                pack.markAsLast(true);
            }
            pack.setStartStep(step);
            listIndex++;
            addObject(pack, 1, 1);
            timer = 0;
            checkNodeNeighbours();
        } else {
            timer++;
```

```
            step++;
        }*/
        // if we have sorted all the packages get the statistics
and quit

    }

    public BluePackage getNextPackage(){

        if ((listIndex < totalPackages)) {// make sure we still
                                                          // have
some packages to
                                                          // sort
            // BluePackage pack = new BluePackage
            // (destinations[Greenfoot.getRandomNumber(3)]);
            //System.out.println(packageDestination.length);
            BluePackage pack = new
BluePackage(packageDestination[listIndex]);
            // if this is the LAST package mark the box
            if (listIndex == totalPackages - 1) {
                pack.markAsLast(true);
            }
            pack.setStartStep(step);
            listIndex++;
           // addObject(pack, 1, 1);
            timer = 0;
            checkNodeNeighbours();
                    return pack;

        } else{

        return null;
    }
       }

    public void setUpBasicLoop() {

        placeConveyor(1, 1, 180);
        placeConveyor(1, 2, 180);
        placeConveyor(1, 3, 180);
        placeConveyor(3, 5, 180);
        placeConveyor(5, 5, 180);
        placeConveyor(7, 5, 180);
        Node node = new Node();
        addObject(node, 1, 4);
        node.putRoute("1", 90);
        node.putRoute("2", 90);
```

```
        node.putRoute("3", 90);
        placeConveyor(2, 4, 90);
        Node node2 = new Node();
        addObject(node2, 3, 4);
        node2.putRoute("1", 180);
        node2.putRoute("2", 90);
        node2.putRoute("3", 90);
        System.out.println(node2.preRoutes.get(1)
                + " place destination of node 2");
        placeConveyor(4, 4, 90);
        Node node3 = new Node();
        addObject(node3, 6, 4);
        node3.putRoute("1", 270);
        node3.putRoute("2", 180);
        node3.putRoute("3", 90);
        placeConveyor(6, 4, 90);
        Node node4 = new Node();
        addObject(node4, 7, 4);
        node4.putRoute("1", 270);
        node4.putRoute("2", 270);
        node4.putRoute("3", 180);
        placeExit(3, 6, 1);
        placeExit(5, 6, 2);
        placeExit(7, 6, 3);

        for (int i = 0; i < node.preRoutes.size(); i++) {
            // System.out.println(node.preRoutes.get(i)+" The
list of locations for node 1");
        }
        // get a set of packages from the database
        ResultSet rs = null;
        try {
            st = conn.createStatement();
            //we are going to get some random database entry for
this number of exits and packages
            String sql = "SELECT id,packages FROM Packages where
(primaryExits=" + numberOfPrimaryExits+ " and total=" +
totalPackages + " and secondaryExits="+numberOfSecondaryExits+")
ORDER BY RAND() limit 1";
            System.out.println(sql);
            rs = st.executeQuery(sql); // run the query
            rs.next();
            packageSet = rs.getInt("id");
            packagesList = rs.getString("Packages");
            packageDestination = packagesList.split(",");
            System.out.println("experiement setup # " +
packageSet + " package order: " + packagesList );
```

```
            st.close();
        } catch (Exception ex) {
            System.out.println("SQLException: " +
ex.getMessage());

        }

        // placeConveyor(5,4,90);

    }

    public void setUpBetterLoop() {


        placeConveyor(1, 2, 180);
        placeConveyor(1, 3, 180);
        placeConveyor(3, 5, 180);
        placeConveyor(6, 5, 180);
        placeConveyor(9, 5, 180);
        placeConveyor(3, 7, 180);
        placeConveyor(6, 7, 180);
        placeConveyor(9, 7, 180);
        placeConveyor(3, 9, 180);
        placeConveyor(6, 9, 180);
        placeConveyor(9, 9, 180);
        Node node = new Node();
        addObject(node, 1, 4);
        node.putRoute("1", 90);
        node.putRoute("2", 90);
        node.putRoute("3", 90);
        placeConveyor(2, 4, 90);
        Node node2 = new Node();
        addObject(node2, 3, 4);
        node2.putRoute("1", 180);
        node2.putRoute("2", 90);
        node2.putRoute("3", 90);
        System.out.println(node2.preRoutes.get(1)
                + " place destination of node 2");
        placeConveyor(4, 4, 90);
        placeConveyor(5, 4, 90);
        Node node3 = new Node();
        addObject(node3, 6, 4);
        node3.putRoute("1", 270);
        node3.putRoute("2", 180);
        node3.putRoute("3", 90);
        placeConveyor(7, 4, 90);
        placeConveyor(8, 4, 90);
```

```
Node node4 = new Node();
addObject(node4, 9, 4);
node4.putRoute("1", 270);
node4.putRoute("2", 270);
node4.putRoute("3", 180);
node = new Node();
node.setSortNumber(2);
addObject(node, 3, 6);
node.putRoute("1", 90);
node.putRoute("2", 180);
node.putRoute("3", 180);
node = new Node();
node.setSortNumber(2);
addObject(node, 3, 8);
node.putRoute("1", 90);
node.putRoute("2", 90);
node.putRoute("3", 180);
node = new Node();
node.setSortNumber(2);
addObject(node, 3, 10);
node.putRoute("1", 90);
node.putRoute("2", 90);
node.putRoute("3", 90);
node = new Node();
node.setSortNumber(2);
addObject(node, 6, 6);
node.putRoute("1", 90);
node.putRoute("2", 180);
node.putRoute("3", 180);
node = new Node();
node.setSortNumber(2);
addObject(node, 6, 8);
node.putRoute("1", 90);
node.putRoute("2", 90);
node.putRoute("3", 180);
node = new Node();
node.setSortNumber(2);
addObject(node, 6, 10);
node.putRoute("1", 90);
node.putRoute("2", 90);
node.putRoute("3", 90);
node = new Node();
node.setSortNumber(2);
addObject(node, 9, 6);
node.putRoute("1", 90);
node.putRoute("2", 180);
node.putRoute("3", 180);
```

```
        node = new Node();
        node.setSortNumber(2);
        addObject(node, 9, 8);
        node.putRoute("1", 90);
        node.putRoute("2", 90);
        node.putRoute("3", 180);
        node = new Node();
        node.setSortNumber(2);
        addObject(node, 9, 10);
        node.putRoute("1", 90);
        node.putRoute("2", 90);
        node.putRoute("3", 90);
        placeExit(4, 6, 11);
        placeExit(4, 8, 12);
        placeExit(4, 10, 13);
        placeExit(7, 6, 21);
        placeExit(7, 8, 22);
        placeExit(7, 10, 23);
        placeExit(10, 6, 31);
        placeExit(10, 8, 32);
        placeExit(10, 10, 33);
        placeInput(1, 1, 180);
        for (int i = 0; i < node.preRoutes.size(); i++) {
            // System.out.println(node.preRoutes.get(i)+" The
list of locations for node 1");
        }
        // get a set of packages from the database
        ResultSet rs = null;
        try {
            st = conn.createStatement();
            //we are going to get some random database entry for
this number of exits and packages
            String sql = "SELECT id,packages FROM Packages where
(primarysort=" + numberOfPrimaryExits+ " and total=" +
totalPackages + " and secondarysort="+numberOfSecondaryExits+")
ORDER BY RAND() limit 1";
            System.out.println(sql);
            rs = st.executeQuery(sql); // run the query
            rs.next();
            packageSet = rs.getInt("id");
            packagesList = rs.getString("Packages");
            packageDestination = packagesList.split(",");
            System.out.println("experiement setup # " +
packageSet + " package order: " + packagesList );
            st.close();
        } catch (Exception ex) {
```

```
            System.out.println("SQLException: " +
ex.getMessage());

        }

        // placeConveyor(5,4,90);

    }
        public void setUpFullLoop() {



        placeConveyor(18, 1, 270);

        placeConveyor(17, 1, 270);
        placeConveyor(16, 1, 270);
        placeConveyor(14, 1, 270);
        placeConveyor(13, 1, 270);
        placeConveyor(12, 1, 270);
        placeConveyor(11, 1, 270);
        placeConveyor(10, 1, 270);
        placeConveyor(9, 1, 270);
        placeConveyor(8, 1, 270);
        placeConveyor(7, 1, 270);
        placeConveyor(6, 1, 270);
        placeConveyor(5, 1, 270);
        placeConveyor(4, 1, 270);
        placeConveyor(3, 1, 270);
        placeConveyor(2, 1, 270);
        Node node = new Node();
        addObject(node, 1, 1);
        node.putRoute("1", 180);
        node.putRoute("2", 180);
        node.putRoute("3", 180);
        node.putRoute("4", 180);
        node.putRoute("5", 180);
        node.putRoute("6", 180);
        node.putRoute("7", 180);
        node.putRoute("8", 180);
        node.putRoute("9", 180);
        placeConveyor(1, 2, 180);
        placeInput(19, 3, 270);
        node = new Node();
        addObject(node,1, 3);
        node.putRoute("1", 180);
        node.putRoute("2", 180);
        node.putRoute("3", 180);
```

```
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(18, 3, 270);
placeConveyor(17, 3, 270);
//placeConveyor(16, 3, 270);
placeConveyor(15, 3, 270);
placeConveyor(14, 3, 270);
placeConveyor(13, 3, 270);
placeConveyor(12, 3, 270);
placeConveyor(11, 3, 270);
placeConveyor(10, 3, 270);
placeConveyor(9, 3, 270);
placeConveyor(8, 3, 270);
placeConveyor(7, 3, 270);
placeConveyor(6, 3, 270);
placeConveyor(5, 3, 270);
placeConveyor(4, 3, 270);
placeConveyor(3, 3, 270);
placeConveyor(2, 3, 270);
placeConveyor(1, 4, 180);
placeInput(19, 5, 270);
node = new Node();
addObject(node,1, 5);
node.putRoute("1", 180);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(18, 5, 270);
placeConveyor(17, 5, 270);
placeConveyor(16, 5, 270);
placeConveyor(15, 5, 270);
placeConveyor(14, 5, 270);
placeConveyor(13, 5, 270);
placeConveyor(12, 5, 270);
placeConveyor(11, 5, 270);
placeConveyor(10, 5, 270);
placeConveyor(9, 5, 270);
placeConveyor(8, 5, 270);
```

```
placeConveyor(7, 5, 270);
placeConveyor(6, 5, 270);
placeConveyor(5, 5, 270);
placeConveyor(4, 5, 270);
placeConveyor(3, 5, 270);
placeConveyor(2, 5, 270);
placeConveyor(1, 6, 180);
placeInput(19, 7, 270);
node = new Node();
addObject(node,1, 7);
node.putRoute("1", 180);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(18, 7, 270);
placeConveyor(17, 7, 270);
placeConveyor(16, 7, 270);
placeConveyor(15, 7, 270);
placeConveyor(14, 7, 270);
placeConveyor(13, 7, 270);
placeConveyor(12, 7, 270);
placeConveyor(11, 7, 270);
placeConveyor(10, 7, 270);
placeConveyor(9, 7, 270);
placeConveyor(8, 7, 270);
placeConveyor(7, 7, 270);
placeConveyor(6, 7, 270);
placeConveyor(5, 7, 270);
placeConveyor(4, 7, 270);
placeConveyor(3, 7, 270);
placeConveyor(2, 7, 270);
placeConveyor(1, 8, 180);
node= new Node();
addObject(node, 1, 9);
node.putRoute("1", 90);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
```

```
node.putRoute("9", 180);
placeConveyor(2, 9, 90);
node= new Node();
addObject(node, 3, 9);
node.setSortNumber(2);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 10, 11);
placeConveyor(4, 9, 90);
node= new Node();
addObject(node, 5, 9);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 10, 12);
placeConveyor(6, 9, 90);
node= new Node();
addObject(node, 7, 9);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 10, 13);
placeConveyor(8, 9, 90);
node= new Node();
addObject(node, 9, 9);
node.setSortNumber(2);
```

```
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(9, 10, 14);
placeConveyor(10, 9, 90);
node= new Node();
addObject(node, 11, 9);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 10, 15);
placeConveyor(12, 9, 90);
node= new Node();
addObject(node, 13, 9);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13, 10, 16);
placeConveyor(14, 9, 90);
node= new Node();
addObject(node, 15, 9);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
```

```
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(15, 10, 17);
placeConveyor(16, 9, 90);
node= new Node();
addObject(node, 17, 9);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeExit(17, 10, 18);
placeConveyor(18, 9, 90);
node= new Node();
addObject(node, 19, 9);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 270);
node.putRoute("9", 180);
placeExit(19, 10, 19);
placeConveyor(1, 10, 180);
node= new Node();
addObject(node, 1, 11);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(2, 11, 90);
node= new Node();
```

```
addObject(node, 3, 11);
node.setSortNumber(2);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 12, 21);
placeConveyor(4, 11, 90);
node= new Node();
addObject(node, 5, 11);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 12, 22);
placeConveyor(6, 11, 90);
node= new Node();
addObject(node, 7, 11);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 12, 23);
placeConveyor(8, 11, 90);
node= new Node();
addObject(node, 9, 11);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
```

```
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(9, 12, 24);
placeConveyor(10, 11, 90);
node= new Node();
addObject(node, 11, 11);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 12, 25);
placeConveyor(12, 11, 90);
node= new Node();
addObject(node, 13, 11);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13,12,26);
placeConveyor(14, 11, 90);
node= new Node();
addObject(node, 15, 11);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
```

```
node.putRoute("9", 90);
placeExit(15, 12, 27);
placeConveyor(16, 11, 90);
node= new Node();
addObject(node, 17, 11);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeExit(17, 12, 28);
placeConveyor(18, 11, 90);
node= new Node();
addObject(node, 19, 11);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 270);
node.putRoute("9", 180);
placeExit(19, 12, 29);
placeConveyor(1, 12, 180);
node= new Node();
addObject(node, 1, 13);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 90);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(2, 13, 90);
node= new Node();
addObject(node, 3, 13);
node.setSortNumber(2);
node.putRoute("1", 180);
```

```
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 14, 31);
placeConveyor(4, 13, 90);
node= new Node();
addObject(node, 5, 13);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 14, 32);
placeConveyor(6, 13, 90);
node= new Node();
addObject(node, 7, 13);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 14, 33);
placeConveyor(8, 13, 90);
node= new Node();
addObject(node, 9, 13);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
```

```
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(9, 14, 34);
placeConveyor(10, 13, 90);
node= new Node();
addObject(node, 11, 13);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 14, 35);
placeConveyor(12, 13, 90);
node= new Node();
addObject(node, 13, 13);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13, 14, 36);
placeConveyor(14, 13, 90);
node= new Node();
addObject(node, 15, 13);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(15, 14, 37);
placeConveyor(16, 13, 90);
```

```
node= new Node();
addObject(node, 17, 13);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeExit(17, 14, 38);
placeConveyor(18, 13, 90);
node= new Node();
addObject(node, 19, 13);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 270);
node.putRoute("9", 180);
placeExit(19, 14, 39);
placeConveyor(1, 14, 180);
node= new Node();
addObject(node, 1, 15);
node.putRoute("1", 180);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(2, 15, 90);
node= new Node();
addObject(node, 3, 15);
node.setSortNumber(2);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
```

```
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 16, 41);
placeConveyor(4, 15, 90);
node= new Node();
addObject(node, 5, 15);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 16, 42);
placeConveyor(6, 15, 90);
node= new Node();
addObject(node, 7, 15);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 16, 43);
placeConveyor(8, 15, 90);
node= new Node();
addObject(node, 9, 15);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
```

```
placeExit(9, 16, 44);
placeConveyor(10, 15, 90);
node= new Node();
addObject(node, 11, 15);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 16, 45);
placeConveyor(12, 15, 90);
node= new Node();
addObject(node, 13, 15);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13, 16, 46);
placeConveyor(14, 15, 90);
node= new Node();
addObject(node, 15, 15);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(15, 16, 47);
placeConveyor(16, 15, 90);
node= new Node();
addObject(node, 17, 15);
node.setSortNumber(2);
```

```
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeExit(17, 16, 48);
placeConveyor(18, 15, 90);
node= new Node();
addObject(node, 19, 15);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 270);
node.putRoute("9", 180);
placeExit(19, 16, 49);
placeConveyor(1, 16, 180);
node= new Node();
addObject(node, 1, 17);
node.putRoute("1", 180);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(2, 17, 90);
node= new Node();
addObject(node, 3, 17);
node.setSortNumber(2);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
```

```
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 18, 51);
placeConveyor(4, 17, 90);
node= new Node();
addObject(node, 5, 17);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 18, 52);
placeConveyor(6, 17, 90);
node= new Node();
addObject(node, 7, 17);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 18, 53);
placeConveyor(8, 17, 90);
node= new Node();
addObject(node, 9, 17);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(9, 18, 54);
placeConveyor(10, 17, 90);
node= new Node();
```

```
addObject(node, 11, 17);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 18, 55);
placeConveyor(12, 17, 90);
node= new Node();
addObject(node, 13, 17);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13, 18, 56);
placeConveyor(14, 17, 90);
node= new Node();
addObject(node, 15, 17);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(15, 18, 57);
placeConveyor(16, 17, 90);
node= new Node();
addObject(node, 17, 17);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
```

```
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeExit(17, 18, 58);
placeConveyor(18, 17, 90);
node= new Node();
addObject(node, 19, 17);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 270);
node.putRoute("9", 180);
placeExit(19, 18, 59);
placeConveyor(1, 18, 180);
node= new Node();
addObject(node, 1, 19);
node.putRoute("1", 180);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(2, 19, 90);
node= new Node();
addObject(node, 3, 19);
node.setSortNumber(2);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 20, 61);
```

```
placeConveyor(4, 19, 90);
node= new Node();
addObject(node, 5, 19);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 20, 62);
placeConveyor(6, 19, 90);
node= new Node();
addObject(node, 7, 19);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 20, 63);
placeConveyor(8, 19, 90);
node= new Node();
addObject(node, 9, 19);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(9, 20, 64);
placeConveyor(10, 19, 90);
node= new Node();
addObject(node, 11, 19);
node.setSortNumber(2);
node.putRoute("1", 270);
```

```
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 20, 65);
placeConveyor(12, 19, 90);
node= new Node();
addObject(node, 13, 19);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13, 20, 66);
placeConveyor(14, 19, 90);
node= new Node();
addObject(node, 15, 19);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(15, 20, 67);
placeConveyor(16, 19, 90);
node= new Node();
addObject(node, 17, 19);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
```

```
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeExit(17, 20, 68);
placeConveyor(18, 19, 90);
node= new Node();
addObject(node, 19, 19);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 270);
node.putRoute("9", 180);
placeExit(19, 20, 69);
placeConveyor(1, 20, 180);
node= new Node();
addObject(node, 1, 21);
node.putRoute("1", 180);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 180);
node.putRoute("9", 180);
placeConveyor(2, 21, 90);
node= new Node();
addObject(node, 3, 21);
node.setSortNumber(2);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 22, 71);
placeConveyor(4, 21, 90);
node= new Node();
addObject(node, 5, 21);
```

```
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 22, 72);
placeConveyor(6, 21, 90);
node= new Node();
addObject(node, 7, 21);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 22, 73);
placeConveyor(8, 21, 90);
node= new Node();
addObject(node, 9, 21);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(9, 22, 74);
placeConveyor(10, 21, 90);
node= new Node();
addObject(node, 11, 21);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
```

```
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 22, 75);
placeConveyor(12, 21, 90);
node= new Node();
addObject(node, 13, 21);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13, 22, 76);
placeConveyor(14, 21, 90);
node= new Node();
addObject(node, 15, 21);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(15, 22, 77);
placeConveyor(16, 21, 90);
node= new Node();
addObject(node, 17, 21);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
```

```
placeExit(17, 22, 78);
placeConveyor(18, 21, 90);
node= new Node();
addObject(node, 19, 21);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 270);
node.putRoute("9", 180);
placeExit(19, 22, 79);
placeConveyor(1, 22, 180);
node= new Node();
addObject(node, 1, 23);
node.putRoute("1", 180);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 180);
placeConveyor(2, 23, 90);
node= new Node();
addObject(node, 3, 23);
node.setSortNumber(2);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 24, 81);
placeConveyor(4, 23, 90);
node= new Node();
addObject(node, 5, 23);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
```

```
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 24, 82);
placeConveyor(6, 23, 90);
node= new Node();
addObject(node, 7, 23);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 24, 83);
placeConveyor(8, 23, 90);
node= new Node();
addObject(node, 9, 23);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(9, 24, 84);
placeConveyor(10, 23, 90);
node= new Node();
addObject(node, 11, 23);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
```

```
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 24, 85);
placeConveyor(12, 23, 90);
node= new Node();
addObject(node, 13, 23);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13, 24, 86);
placeConveyor(14, 23, 90);
node= new Node();
addObject(node, 15, 23);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(15, 24, 87);
placeConveyor(16, 23, 90);
node= new Node();
addObject(node, 17, 23);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeExit(17, 24, 88);
placeConveyor(18, 23, 90);
node= new Node();
```

```
addObject(node, 19, 23);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 270);
node.putRoute("9", 180);
placeExit(19, 24, 89);
placeConveyor(1, 24, 180);
node= new Node();
addObject(node, 1, 25);
node.putRoute("1", 180);
node.putRoute("2", 180);
node.putRoute("3", 180);
node.putRoute("4", 180);
node.putRoute("5", 180);
node.putRoute("6", 180);
node.putRoute("7", 180);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeConveyor(2, 25, 90);
node= new Node();
addObject(node, 3, 25);
node.setSortNumber(2);
node.putRoute("1", 180);
node.putRoute("2", 90);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(3, 26, 91);
placeConveyor(4, 25, 90);
node= new Node();
addObject(node, 5, 25);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 180);
node.putRoute("3", 90);
node.putRoute("4", 90);
node.putRoute("5", 90);
```

```
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(5, 26, 92);
placeConveyor(6, 25, 90);
node= new Node();
addObject(node, 7, 25);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 180);
node.putRoute("4", 90);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(7, 26, 93);
placeConveyor(8, 25, 90);
node= new Node();
addObject(node, 9, 25);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 180);
node.putRoute("5", 90);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(9, 26, 94);
placeConveyor(10, 25, 90);
node= new Node();
addObject(node, 11, 25);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 180);
node.putRoute("6", 90);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(11, 26, 95);
```

```
placeConveyor(12, 25, 90);
node= new Node();
addObject(node, 13, 25);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 180);
node.putRoute("7", 90);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(13, 26, 96);
placeConveyor(14, 25, 90);
node= new Node();
addObject(node, 15, 25);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 180);
node.putRoute("8", 90);
node.putRoute("9", 90);
placeExit(15, 26, 97);
placeConveyor(16, 25, 90);
node= new Node();
addObject(node, 17, 25);
node.setSortNumber(2);
node.putRoute("1", 270);
node.putRoute("2", 270);
node.putRoute("3", 270);
node.putRoute("4", 270);
node.putRoute("5", 270);
node.putRoute("6", 270);
node.putRoute("7", 270);
node.putRoute("8", 180);
node.putRoute("9", 90);
placeExit(17, 26, 98);
placeConveyor(18, 25, 90);
node= new Node();
addObject(node, 19, 25);
node.setSortNumber(2);
node.putRoute("1", 270);
```

```
        node.putRoute("2", 270);
        node.putRoute("3", 270);
        node.putRoute("4", 270);
        node.putRoute("5", 270);
        node.putRoute("6", 270);
        node.putRoute("7", 270);
        node.putRoute("8", 270);
        node.putRoute("9", 180);
        placeExit(19, 26, 99);



        for (int i = 0; i < node.preRoutes.size(); i++) {
            // System.out.println(node.preRoutes.get(i)+" The
list of locations for node 1");
        }
        // get a set of packages from the database
        ResultSet rs = null;
        try {
            st = conn.createStatement();
            //we are going to get some random database entry for
this number of exits and packages
            String sql = "SELECT id,packages FROM Packages where
(primarysort=" + numberOfPrimaryExits+ " and total=" +
totalPackages + " and secondarysort="+numberOfSecondaryExits+")
ORDER BY RAND() limit 1";
            System.out.println(sql);
            rs = st.executeQuery(sql); // run the query
            rs.next();
            packageSet = rs.getInt("id");
            packagesList = rs.getString("Packages");
            packageDestination = packagesList.split(",");
            System.out.println("experiement setup # " +
packageSet + " package order: " + packagesList );
            st.close();
        } catch (Exception ex) {
            System.out.println("SQLException: " +
ex.getMessage());

        }

        placeSensor(15, 1, 270, 19, 3);
        placeSensor(16, 3, 270, 19, 5);

        Input convey = new Input();
        convey.setRotation(270);
```

```
        addObject(convey, 19, 1);
        convey.fire();
        convey.auto = true;




    }



    public void placeConveyor(int x, int y, int rotation) {
        Conveyor convey = new Conveyor();
        convey.setRotation(rotation);
        addObject(convey, x, y);
    }

    public void placeSensor(int x, int y, int rotation, int xs,
int ys) {

        SensorSection convey = new
SensorSection((Input)getObjectsAt(xs,ys,Input.class).get(0));
        convey.setRotation(rotation);
        addObject(convey, x, y);
    }

    public void placeNode(int x, int y) {
        Node node = new Node();

        addObject(node, x, y);
    }

    public void placeInput(int x, int y, int rotation){
         Input convey = new Input();
        convey.setRotation(rotation);
        addObject(convey, x, y);

    }

    public void placeExit(int x, int y, int exitNumber) {
        Exit exit = new Exit();
        exit.setExitID(exitNumber);
        addObject(exit, x, y);
    }

    public int getStep() {
        return step;
```

```
    }

    public void endSimulation() {
       // Greenfoot.delay (70);

        finalStep = step;
        Greenfoot.playSound("floop.wav");
        String results = "";
        // iterate through the exits to get total sort times
        List<Exit> exitList = getObjects(Exit.class);
        String exits = "";
        String values = "";
        for (int i = 0; i < exitList.size(); i++) {
            Exit exit = (Exit)exitList.get(i);
            int maxTime = exit.totalSortTime;
            System.out.println("Total Sort time for Exit " +
exit.exitID
                    + " is:" + maxTime);
            results = results + maxTime + "(" + exit.exitID +
")";

            exits = exits +",\""+ exit.exitID +"\"";
            values = values +","+ maxTime+"";

            String insertStatement = "ALTER TABLE table4050
ADD \""+ exit.exitID +"\" INTEGER";


                System.out.println(insertStatement);

            try {
            st = conn.createStatement();
            st.execute(insertStatement);
            st.close();
        }

        catch (Exception ex) {
            System.out.println("SQLException for inserting
results: "
                    + ex.getMessage());

        }

        }

         System.out.println("There are "+exitList.size()+" exits
- "+exits+"");
```

```
        // Put results into database


                java.util.Date now = new java.util.Date();
        String dateString = new
java.sql.Timestamp(now.getTime()).toString();

            String insertStatement = "INSERT into table4050
(rundate,id"+exits+") values ('"
                + dateString + "'," + packageSet +""+ values +
")";

                System.out.println(insertStatement);
         Greenfoot.stop();
        try {
            st = conn.createStatement();
            st.execute(insertStatement);
            st.close();
        }

        catch (Exception ex) {
            System.out.println("SQLException for inserting
results: "
                    + ex.getMessage());

        }
    }
}
```

BlueBox.java

The Blue Box was our main actor (agent) in the facility. It is a subclass of Package

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot
and MouseInfo)
import java.awt.Color;
/**
 * Write a description of class BluePackage here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BluePackage extends Package
{
```

```
public BluePackage (String destination){
        GreenfootImage gi = getImage();
         getImage().scale(40,40);
        gi.setColor(Color.blue);
        gi.drawString(destination,15,10);
        this.destination = destination;
       // setLocation(getX());
    }
    /**
     * Act - do whatever the BluePackage wants to do. This
method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        moved = false;
    }
}
```

Package.java

The Package is the superclass of BlueBox. It has methods common to all Boxes.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot
and MouseInfo)

/**
 * Write a description of class Package here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Package  extends Actor
{

    Boolean moved = false;
    boolean lastPackage = false;
    int startStep=0;
    int finalStep=0;
    String destination = "";
    int totalTime=0;

    /**
     * Act - do whatever the Package wants to do. This method is
called whenever
```

```
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
      totalTime++;
      moved = false;

    }
    //this is the simulation step at which this package begins
to be sorted
     public void setStartStep(int startStep) {
       this.startStep = startStep;
    }
    //this is the simulation step at which this package exits
the conveyor system
       public void setFinalStep(int finalStep) {
       this.finalStep = finalStep;
    }
    //allows us to "mark" this as the last package through the
system
     public void markAsLast(boolean last) {
         lastPackage=true;
        }
}
```

Conveyor.java

The Conveyor is the agent that moves the boxes along in some direction toward nodes and exits.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot
and MouseInfo)
import java.util.*;


/**
 * Write a description of class Conveyor here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Conveyor  extends Actor
{
    List intersects;
    public Conveyor(){
```

```
          getImage().scale(40,40);
    }

       public Node getConnection(){

        for (int i = 0; i<10000; i++){
        if
( getOneObjectAtOffset((int)Math.sin(Math.toRadians(this.getRota
tion()))*i,
        (int)Math.cos(Math.toRadians(this.getRotation()))*i,null
).getClass().equals("Node")){
            List returnt = null;
            returnt.add( (Node)
getOneObjectAtOffset((int)Math.sin(Math.toRadians(this.getRotati
on()))*i,
        (int)Math.cos(Math.toRadians(this.getRotation()))*i,Node
.class));
       // System.out.println("Found node");
        returnt.add(i);
        Node node =  (Node)returnt.get(0);
        return node;
        }
        }
        System.out.println("Found no node");
        return null;
    }

    /**
     * Act - do whatever the Conveyor wants to do. This method
is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
         intersects = getObjectsAtOffset(0,0,BluePackage.class);
       // System.out.println(getRotation());

         if (intersects.size()<=1){



         for (int i = 0; i<intersects.size(); i ++){
             if
(intersects.get(i).getClass().getSuperclass().getName().equals("
Package")){
                 Package pack = ((Package) intersects.get(i));
                 if (!pack.moved){
```

```
                // System.out.println(pack.getX());
                (pack).setLocation((int)( pack.getX() +
Math.sin(Math.toRadians(this.getRotation()))),
                pack.getY()-
(int)Math.cos(Math.toRadians(this.getRotation())));
               // System.out.println("rpation = " +(int)
(Math.sin(Math.toRadians(90))));
                pack. moved = true;
            }


                }
             //System.out.println(intersects.get(i).getClass().g
etName());
            }
        }

    }

    public void reverse(){

        Conveyor offset = (Conveyor)
getOneObjectAtOffset((int)Math.sin(Math.toRadians(this.getRotati
on())),
        (int)Math.cos(Math.toRadians(this.getRotation())),getCla
ss());
        setRotation(getRotation()+180);
        offset.reverse();


    }
}
```

Input.java

The Input is the actor that takes the packages and sends them down the conveyors in pre-set distance/time intervals.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot
and MouseInfo)

/**
 * Write a description of class input here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
```

```
public class Input  extends Actor
{
    /**
     * Act - do whatever the input wants to do. This method is
called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */

    boolean auto = false;
    int timer = 0;

   public void addedToWorld(World world){
        getImage().scale(40,40);
    }
    public void act()
    {
        if (auto){

        if (timer == 6){
            BluePackage pack = ((Facility)
getWorld()).getNextPackage();
            if (pack ==(null)){
                return;
            }
            getWorld().addObject(pack, getX(), getY());
            //System.out.println("Added object at " + getX()
+"-"+getY());
            //System.out.println("Object at " + pack.getX()
+"-"+pack.getY());
            if (!pack.moved){

                (pack).setLocation((int)( pack.getX() +
Math.sin(Math.toRadians(this.getRotation()))),
                pack.getY()-
(int)Math.cos(Math.toRadians(this.getRotation())));
                //System.out.println("rpation = " +(int)
(Math.sin(Math.toRadians(90))));
                pack. moved = true;

        }




            timer = 0;
```

```
        }
        else{ timer++;}

    }
}

    public void fire(){
         BluePackage pack = ((Facility)
getWorld()).getNextPackage();
            if (pack ==(null)){
                return;
            }
            getWorld().addObject(pack, getX(), getY());
            //System.out.println("Added object at " + getX()
+"-"+getY());
            //System.out.println("Object at " + pack.getX()
+"-"+pack.getY());
            if (!pack.moved){

                (pack).setLocation((int)( pack.getX() +
Math.sin(Math.toRadians(this.getRotation()))),
                pack.getY()-
(int)Math.cos(Math.toRadians(this.getRotation())));
                //System.out.println("rpation = " +(int)
(Math.sin(Math.toRadians(90))));
                pack. moved = true;

        }


    }
}
```

Exit.java

The Exit is the agent that removes the boxes from the simulation, and records the time for the sort to finish.

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot
and MouseInfo)
import java.util.*;

/**
 * Write a description of class Exit here.
 *
 * @author (your name)
```

```
 * @version (a version number or a date)
 */
public class Exit  extends Actor
{
    int totalSortTime=0;
    //World world;
    int exitID=0;
    int packsSorted = 0;
    //GreenfootSound blip = new GreenfootSound("blip.wav");

    //set the exitID value
    public void setExitID(int id) {
        exitID = id;
    }

    public void addedToWorld(World world){
        //this.world = world;
         getImage().scale(40,40);
         totalSortTime = 1;

    }

    /**
     * Act - do whatever the Exit wants to do. This method is
called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */


    public void act()
    {
         List intersects = getIntersectingObjects(Package.class);
       // System.out.println(getRotation());

         for (int i = 0; i<intersects.size(); i ++){
              packsSorted++;
             Package pack = (Package) intersects.get(i);
             //package has arrived at the exit so get sort time
from the package
             int step = ((Facility)getWorld()).getStep();
             pack.setFinalStep(step);
            int totalPackagesSortable =
((Facility)getWorld()).totalPackages/
((Facility)getWorld()).numberOfPrimaryExits*((Facility)getWorld(
)).numberOfSecondaryExits/3;

            //if (packsSorted >= totalPackagesSortable){
```

```
                totalSortTime = step;

            // }
              //totalSortTime=step;

              //check if this is the very last package to sort
              if (pack.lastPackage) {

                   getWorld().removeObject(pack);
                   //((Facility)getWorld()).endSimulation();
                 }
              else {
                getWorld().removeObject(pack);
                //blip.play();
             }
             }
    }
}
```

Node.java

The Node is the agent that senses the boxes destination and makes decisions on where the box should go next (stay on the conveyor or transfer to another one).

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot
and MouseInfo)
import java.util.*;

/**
 * Write a description of class Node here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Node  extends Actor
{

    List neighbours;
    List intersects;
    Hashtable preRoutes = new Hashtable();

    int sortNumber = 1;
```

```
    public void setSortNumber(int number){
        sortNumber = number;
    }

    public List addLists(List l1,List l2){
        for (int i = 0; i < l2.size(); i ++){
            l1.add(l2.get(i));
        }
        return l1;
    }

    public void addedToWorld(World world){
        getImage().scale(40,40);
        neighbours = new ArrayList();
        //preRoutes = new Hashtable();

       // System.out.println(this.getClass().getName()+" has
been added to the world!");

        //for (int i = 0; i < world.getWidth(); i++){

           if (!
getNeighbours(world.getWidth(),false,Node.class).isEmpty()){
                neighbours =
getNeighbours(world.getWidth(),false,Node.class);
            }
       // }


        /*
         neighbours = getNeighbours(1,false,new
Conveyor().getClass());
        for (int i = 0; i < neighbours.size(); i++)
        {
            Conveyor convey = (Conveyor) neighbours.get(i);
            Node nextNode = convey.getConnection();

            neighbours.remove(i);
            neighbours.add(nextNode);
        */
    }

    public void checkNeighbours(World world){
        if (!
getNeighbours(world.getWidth(),false,Node.class).isEmpty()){
                neighbours =
getNeighbours(world.getWidth(),false,Node.class);
```

```
            }

    }

     public void putRoute(Object key, Object value){
        preRoutes.put(key,value);
       // System.out.println(preRoutes.get(key)+" was added to
preRoutes under key "+key);
        if (preRoutes.isEmpty()){
        //System.out.println("something is messed up");
       }else{
           //System.out.println(preRoutes.keys()+ " keys");

           for (int i = 0; i < preRoutes.size(); i ++){
           //System.out.println(preRoutes.get(i+1)+"!!!!!!");
        }
        }

    }

    public int getDistance(Actor act) {
        int x = act.getX()-getX();
        int y = act.getY()-getY();
        double distance = Math.sqrt((x*x)+(y*y));
        return (int)(distance+0.5);
    }







    /**
     * Act - do whatever the Node wants to do. This method is
called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {

         intersects = getObjectsAtOffset(0,0,null);
       // System.out.println(getRotation());

         for (int i = 0; i<intersects.size(); i ++){
             if
(intersects.get(i).getClass().getSuperclass().getName().equals("
Package")){
```

```
                Package pack = ((Package) intersects.get(i));
                if (!pack.moved){
                    String destination = pack.destination;
                    //System.out.println(destination+"
destination of package here " + preRoutes.size());
                    for (int c = 0; c < preRoutes.size(); c ++){
                       //
System.out.println(preRoutes.get(c+1)+" key " + (c+1));
                        if
(destination.toString().equals(((c+1)+""))){
                           // System.out.println("happiness");
                           }
                    }




                    if
(preRoutes.containsKey((destination.substring(sortNumber-
1,sortNumber)))){
                       // System.out.println("contains
destination!");
                        Integer n =
(Integer)preRoutes.get((destination.substring(sortNumber-
1,sortNumber)));



                          pack.setRotation(n);

                         // if(!
getOneObjectAtOffset((int)Math.sin(Math.toRadians(pack.getRotati
on())),
                        //
(int)Math.cos(Math.toRadians(pack.getRotation())),Conveyor.class
).equals(null)){
                              // System.out.println("moving
package");
                                Conveyor convey = (Conveyor)
getOneObjectAtOffset((int)Math.sin(Math.toRadians(
                                    pack.getRotation())),
                                       (int)Math.cos(Math.toRad
ians(pack.getRotation()))*-1,Conveyor.class);
                                    (pack).setLocation((int)
( pack.getX() + Math.sin(Math.toRadians(pack.getRotation()))),
                                     pack.getY()-
(int)Math.cos(Math.toRadians(pack.getRotation())));
```

```
                        // }
                    }
                  }
                  //System.out.println(pack.getX());
                // System.out.println("moved package");
                 pack. moved = true;
            }


                 }

    }
}
```

Appendix B. Database Schema

Table Packages

CREATE TABLE PACKAGES(ID INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 0) NOT NULL PRIMARY KEY,PACKAGES LONGVARCHAR,PRIMARYSORT INTEGER,TOTAL BIGINT,SECONDARYSORT INTEGER)

Table Results (simple example, 3 regions, 3 local regions for 9 total destinations)

CREATE TABLE RESULTS(RUNDATE TIMESTAMP,"ID" INTEGER,"11" INTEGER,"12" INTEGER,"13" INTEGER,"23" INTEGER,"33" INTEGER,"22" INTEGER,"21" INTEGER,"31" INTEGER,"32" INTEGER, TOTAL BIGINT, INPUTS INTEGER, EXITS INTEGER)