MCMASTER UNIVERSITY

CAS 4ZP6

TEAM 9

CAPSTONE PROJECT 2013/2014

PORTER SIMULATION

Design Revision 0

Authors:
Vitaliy Kondratiev - 0945220
Nathan Johrendt - 0950519
Tyler Lyn - 0948978
Mark Gammie - 0964156

Supervisor:
Dr. Douglas Down

April 16, 2014

CONTENTS

1 Revision History

Revision #	Author	Date	Comment
1	Vitaliy Kondratiev, Nathan Johrendt, Tyler Lyn, Mark Gammie	January 11, 2014	Revision 0 Added to repository
2	Vitaliy Kondratiev, Mark Gammie	January 12, 2014	Added diagrams to Latex
3	Tyler Lyn	January 12, 2014	Added dispatch module
4	Vitaliy Kondratiev, Nathan Johrendt	January 13, 2014	Added GUI Component to document
5	Mark Gammie	January 13, 2014	Updated Dependency Diagram to contain algorithms and data structures. Updated dependency diagram with a legend, logging comment and fixed the porter dispatcher dependency
6	Tyler Lyn, Mark Gammie	January 13, 2014	Several Modules Updated
7	Nathan Johrendt	January 13, 2014	Added Appendix
8	Mark Gammie	January 14, 2014	Completed Design Overview

2 Executive Summary

2.1 Introduction

This document outlines the design decisions, style and methodology for the project of Porter Simulation to be completed for Hamilton Health Sciences. A modular design methodology has been chosen as our design principle. This document is based on IEEE Draft Standard for Software Design Descriptions (IEEE P1016/D5.0).

2.2 Purpose

The purpose of this document is to outline the design of each component and how they interface between each other. This document will aid the developers in the development process as well as any future maintenance required.

2.3 Design Overview

Our solution is a discrete event simulation that is implemented using the Python library SimPy. The simulation has three main sections named Import, Export and Core.

The Import section contains modules for accessing two different csv files that are used as input for the simulation. The first file contains a collection of data that has been exported from Hamilton Health Science's porter management system. The second file contains data that represents the work schedule of each porter that will be used in the simulation. Both of these files will be modified using Excel as the interface.

The Export section contains a module for reporting statistics that are generated by the simulation. The module will be used after the simulation has completed to write the generated data to a csv file that will be interpreted later by some Visual Basic scripts. These scripts will be used to generate predetermined graphs for the user to view.

The Core section is where the simulation actually occurs. The module Simulation Core contains the command line interface for the simulation and is responsible for initializing the Simulation State, Job List Builder and Dispatcher modules. The Job List Builder module will create a list of jobs using the data contained in the file filled with historical data. This list of jobs will be submitted to the Simulation State to be stored. Once the simulation is started, the Dispatcher module is responsible for receiving the jobs as they become available and assigning them intelligently to any eligible porters. The

porters that operate on the tasks follow a finite state machine that creates timeouts based on the time it would take for each state that the porter is transitioning into. There may be cases when all the porters are busy and the amount of assigned jobs piles up, or the porters are stuck waiting on jobs to become available. Once the simulation terminates, the export of the statistical data is finalized.

3 Implementation Material

3.1 Language of Implementation

Visual Basic in Excel will be used to interface GUI to Simulation Core module. Python Version 2.7.5 will be used for Simulation Core module and all other modules.

3.2 Supporting Technology and Frameworks

Simulation will be built on the SimPy 3.0.2 library
`https://simpy.readthedocs.org/en/latest/`. GUI will be built in Excel 2010.

4 Process Diagram

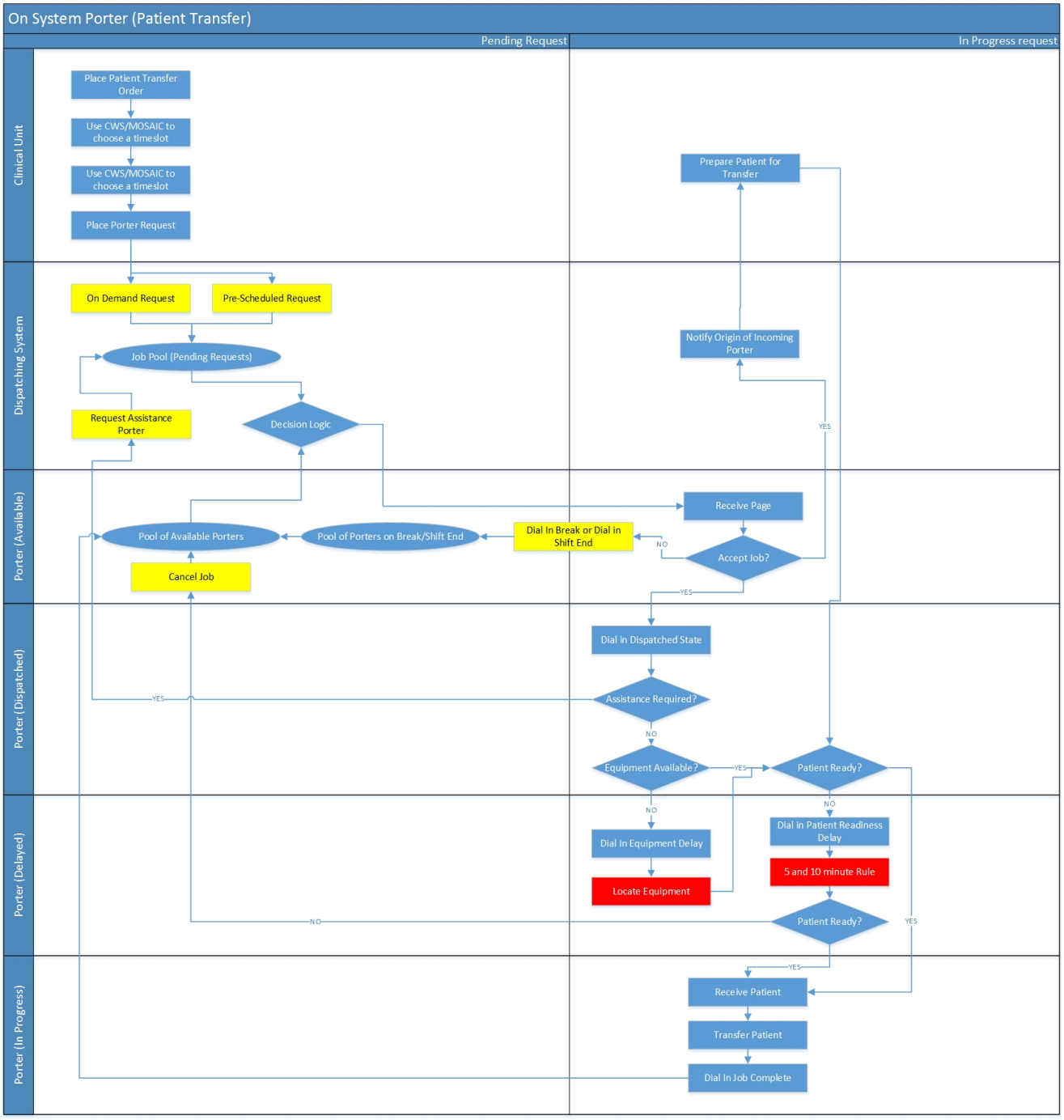

Figure 4.1: Process Diagram

5 Dependency Diagram

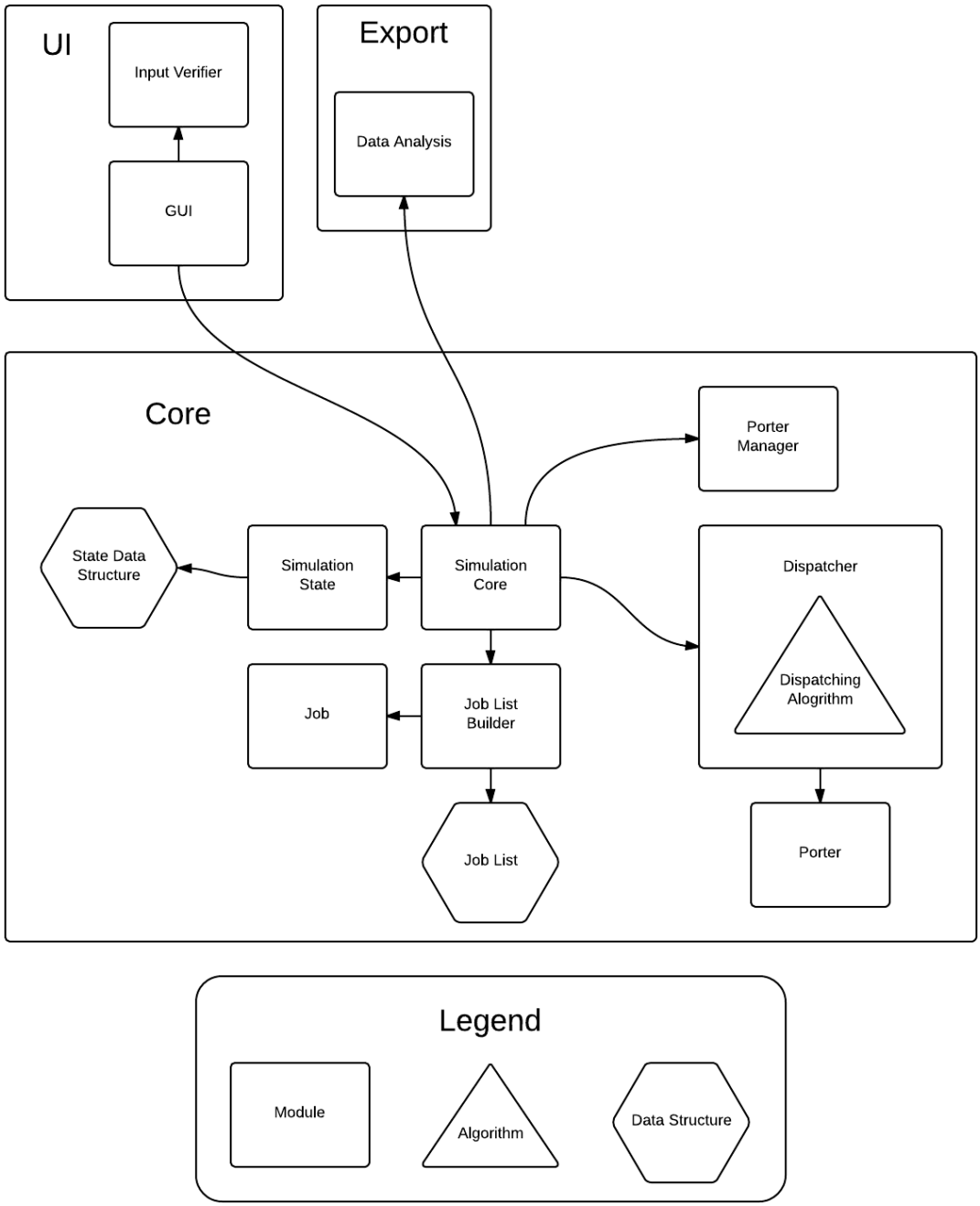

Figure 5.1: Dependency Diagram

6 Decomposition Description

6.1 Core - Simulation Core

Type: Module

Purpose: This module calls the required functions to create the Job List, initialize the simulation state and initialize the simulation processes.

Function: This module takes in configuration data from the GUI module and uses it to initiate the required modules Statistics Import and Porter Manager, passes their data so that the Simulation State, Job List Builder and Dispatcher can be initialized.

Interface:
The interface to this module is a dictionary containing the information for the configuration of the simulation.

Process Steps: This module first This module first uses the Simulation Settings, Hospital Layout Graph and Statistical data to read in the external data. This data is used to configure other core modules in the simulation.

Once the external data is imported and the core modules are configured, the Simulation Core will begin the simulation and manage the core modules.

Data: None

Error Handling: Catch all on the simulation loop to report pertinent errors and to prevent unexpected termination.

Requirement Reference: 11.5.1

6.2 Core - Simulation State

Type: Module

Purpose: This module's purpose is to be an interface between the simulation and its simulation state data structure.

Function: This module will contain functions that will be used by the simulation to perform queries on the simulation state data structure.

Interface:
initSimulateState():

- Instantiating the simulation state with null values

getSimulationTime():

- Returns the current simulation time

getPorterList():

- Returns a list of the porter objects

getTaskList():

- Returns the task list

Process Steps: This module handles the simulation state data structure and allows the rest of the modules to interact with it. If modules need information about the simulation as a whole, they will communicate with the Simulation State module in order to satisfy their needs.

Data: Simulation state data structure

Error Handling: Catch any null values in the simulation state data structure

Requirement Reference: None

6.3 CORE - JOB

Type: Module

Purpose: Provide information about a job to the porter.

Function: The task is declared with an origin, destination, required equipment and a priority. This information is necessary to allowing a porter to complete a job.

Interface:
getOriginLocation():

- Returns the origin of a job

getDestinationLocation():

- Returns the destination of a job

getEquipment():

- Returns the equipment required for a job

getPriority():

- Returns the priority of a job

getTime():

- Returns the time that the task will be requested

updatePriority():

- A process that will set the job to a higher priority after a specified amount of time has passed.

Process Steps: Tasks will be initialized once the Simulation Core shares the import modules with the Event List Builder. An initialized task will be able to provide the Dispatcher all the required information to organize a list of tasks. Also, once a task is given to a porter, the porter will then be able to start and complete a transport job.

Data: Transport Job Information

Error Handling: Catch any null values returned by the task's properties

Requirement Reference: None

6.4 CORE - TASK LIST BUILDER

Type: Module

Purpose: Produces the list of tasks for the Simulation Core and Dispatcher modules to process

Function: This module populates a series of tasks following either a distribution or statistical data.

Interface:
addTask(origin, destination, equipment, priority, time):

- Input the origin, destination, equipment and priority of a job. Also include the time the task will be added to the Dispatcher

setTaskBuilderStats(data)

- Input the statistical or distribution data as it is required for creating meaningful data

Process Steps: The Task List Builder will first receive the data from the Simulation Core. This data will allow the Task List Builder to create jobs based on user specified parameters. Secondly, the Task List Builder will generate tasks and append them to the BinTree data structure, to allow for ordering, inserting and retrieval of tasks. Once a task reaches its "time" it will be added to the Dispatcher where it can be given to an available porter.

Data: Task List BinTree

Error Handling: Catch any null tasks

Requirement Reference: 8.1.4

6.5 CORE - PORTER

Type: Module

Purpose: To complete jobs provided by the dispatcher

Function: Completes the transport jobs assigned by the dispatcher. Unless a job is cancelled the porter will traverse through four states ('pending', 'dispatched', 'inprogress', 'complete')

Interface:
setStatePending(state):

- Input the pending state
- Sets the porter's state to pending and waits to be assigned a job

setStateDispatched(state):

- Input the dispatched state
- Sets the porter's state to dispatched and calculates the time between the porter's location and the job's origin

setStateInprogress(state):

- Input the inprogress state
- Sets the porter's state to inprogress and calculates the time between the job's origin and destination

setStateComplete(state):

- Input the complete state
- Sets the porter's state to complete, records the completion time and sets the porter back to the pending state

getAutoLocation():

- Returns the estimated location of a pending porter
- Estimates the current location of a porter based on how many minutes they have been in
- Output the estimated location of a pending porter
- Estimates the current location of a porter based on how many minutes they have been in the pending state.

Process Steps: The module listens for state changes provided by the dispatcher and updates its' internal components as necessary.

Data: Stores internal data relating to its' current state.

Error Handling: None

Requirement Reference: None

6.6 CORE - DISPATCHER

Type: Module

Purpose: To organize pending jobs based on a weighted-value and assign them to porters

Function: This module orders pending jobs based off of a Dispatch Value which is computed using several parameters (Proximity Match Value, Weighted Job Priority and Appointment Factor). The pending job with the greatest Dispatch Value will be assigned to the closest available porter. Once the job is assigned to the porter the job will be considered as a dispatched job.

Interface:
assignJob(Task):

- Assigns the job with the greatest Dispatch Value to the closest available porter.

getProxmityMatchValue(TaskOrigin):

- Input the origin of a pending job
- Output a value based on how close an available porter is to a job's origin

getWeightedJobPriority(TaskOrigin, TaskDestination):

- Input the origin and destination of a pending job
- Output a value based on the priority of the pending job

getAppointmentFactor(Task):

- Input a pending job
- Update the value for a job depending on if it was pre-scheduled or on-demand.

getDispatchValue(Task):

- Input a pending job
- Compute the DispatchValue for a job: (ProxmityMatchValue + WeightedJobPriority * AppointmentFactor)

Process Steps: All pending jobs are assessed and given a dispatch value (DV) based on the weighting and values of specified dispatch parameters.

These weights and values are determined using either the location of an available porter or the priority of a pending job.

All of the pending jobs are then ordered from greatest dispatch value to the least. When there is an available porter the pending job with the greatest dispatch value is given to the closest porter.

Data:

- Pending jobs

Error Handling: None

Requirement Reference: None

6.7 Export - Statistical Data Export

Type: Module

Purpose: Format and output the raw data from the simulation.

Function: This module will gather all of the raw data from the simulation and format it so that it can be exported into a readable excel file.

Interface: addExportData(data)

- Input raw data

publishExportData()

- This function will format the simulation's raw data, create an excel file and populate the file with the formatted data

Process Steps: All data that is to be exported must be added to this module. Once the simulation's raw data is gathered the module will publish the data. The publishing will generate an excel file and outline the results from the simulation.

Data: None

Error Handling: Produce warning message if data is missing during the publishing.

Requirement Reference: 8.1.5

Critical Revision 0 Component: True

To Be Completed By: 28/01/2014

To Be Tested By: 30/01/2014

6.8 GUI - Basic Settings - General

Type: User Interface

Purpose: Allows the user to change the basic-general settings of the simulation

Function:

(i) **Simulation Duration:** Number of days the software will simulate events.

(ii) **Porter Wait Times:** Maximum number of minutes a porter waits for a delay (patient, equipment, nurse) before cancelling the job

(iii) **Job Flow:** The rate of event generation during the course of the simulation.

(iv) **Start Day:** Define the first day the simulation generates events from.

Process Steps: Not Available

Data: Not Available

Error Handling: The interface prevents the user from violating constraints

Simulation Duration: Value 0 to 7

Porter Wait Times: Value 0 to 60

Job Flow: Drop down defined by interface Low, Normal, High

Start Day: Drop down defined by interface Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

Requirement Reference: 10.1.1, 10.1.2, 10.1.3, 10.1.4

6.9 GUI - Basic Settings - File Locations

Type: User Interface

Purpose: Allows the user to choose the directory paths for required import/export files

Function:

(i) **Statistical Data Source:** Determines which source file will be used to provide the simulation with statistical distribution information

(ii) **Schedule Data Source:** Determines which source file will be used to provide scheduling information to be used during simulation

(iii) **Output Location:** Determines the Location to store output data

Process Steps: Not Available

Data: Not Available

Error Handling: The interface prompts the user with an error message upon entry of erroneous data

Statistical Data Source: File exists. File extension is ".csv"

Schedule Data Source: File Exists. File extension is ".csv"

Output Location: Directory exists

Requirement Reference: 10.1.1, 10.1.2, 10.1.3, 10.1.4

6.10 GUI - Basic Settings - Buttons/Output Console

Type: User Interface

Purpose: Allows the user to activate simulation or reset all data

Function:

(i) **Simulate Button:** Runs all the integrity checks and initializes the simulation run

(ii) **Reset All Button:** Resets all data to initial values

(iii) **Output Console:** Prints any logging statements

Process Steps: Not Available

Data: Not Available

Error Handling: Interface prevents user from entering erroneous data

Output Console: Console is read only

Requirement Reference: 10.1.1, 10.1.2, 10.1.3, 10.1.4

6.11 GUI - Advanced Settings - Dispatcher/Random Seed

Type: User Interface

Purpose: Allows the user to tool around with dispatcher values

Function:

(i) **Appointment Factor:**

(ii) **Automatic Job Priority Values:**

(iii) **Weighted Job List:**

(iv) **Random Seed:** allows the user to enter a specific random seed instead of true random

Process Steps: Not Available

Data: Not Available

Error Handling: Interface prevents user from entering erroneous data

Appointment Factor: Value 0.0 to 3.0

Automatic Job Priority Values: Values 0.0 to 99.99

Weighted Job List: Values 0.0 to 99.99

Random Seed: Value 0 to 999999999

Requirement Reference: 10.1.1, 10.1.2, 10.1.3, 10.1.4

6.12 Schedule Verifier

Type: Module

Purpose: Confirms the integrity of the schedule

Function:

(i) **scheduleChecker(file):** checks integrity of the schedule as a whole

(ii) **verifyShiftID(element):** checks integrity of each shift id

(iii) **verifyStartTime(element):** checks integrity of each start time

(iv) **verifyEndTime(element):** checks integrity of each end time

(v) **verifyPorterID(element):** checks integrity of each porter id

(vi) **verifyDays(element):** checks integrity of each day entry

Process Steps: scheduleChecker opens the schedule file and verifies each element of the file with a subroutine

Data: schedule data file

Error Handling: Module gathers and returns all the errors

scheduleChecker(file): returns the returns from all the subroutines (returns the cell and place of error)

verifyShiftID(element): string of character [A-Za-z0-9]

verifyStartTime(element): is date

verifyEndTime(element): is date

verifyPorterID(element): string of character [A-Za-z0-9]

verifyDays(element): value 0 - 6

Requirement Reference: ??

6.13 Schedule Parser

Type: Module

Purpose: Converts the schedule to a more convenient format

Function:

(i) **scheduleParser(file):** checks integrity of the schedule as a whole

Process Steps: scheduleParser opens the schedule file and converts it to a python dictionary format

Data: schedule data file

Error Handling: No error if Schedule Verifier has been executed

Requirement Reference: ??

7 Anticipated Changes

None

8 Appendicies

8.1 Figure Appendix

(a) Figure 4.1: Process Diagram.jpg - the process flow diagram details the process modeled by the simulation.

(b) Figure 5.1: Dependency Diagram.png - layout of the module, algorithm and data structure dependencies.

(c) Figure 6.1: Basic GUI.png - prototype design of the GUI associated for use with the simulation.

8.2 Algorithm Appendix

(a) Dispatching Algorithm - main dispatcher algorithm used to assign jobs to porters based on a number of factors. Functionality outlined by section 6.6 of this document.

(b) Dijkstra's Algorithm - will be used to compute the shortest path between two hospital location.