

Simulation of Building Evacuation



**LUNDS
UNIVERSITET**

Lunds Tekniska Högskola

**LTH School of Engineering at Campus Helsingborg
Computer Engineering**

Bachelor thesis:
Anton Botvalde
Mats Lundkvist

© Copyright Anton Botvalde, Mats Lundkvist

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2012

Preface

This bachelor thesis has taken place during the spring of 2012 at LTH Campus Helsingborg. The work has been focused around simulation software development. The thesis is the final part of the computer engineering program at Lund Tekniska Högskola Campus Helsingborg.

It has been very rewarding to learn about simulation and to develop a simulation tool. We would like to thank our supervisor Christian Nyberg for all his support during this thesis, especially in the early stages of the thesis when he introduced us to the world of simulation.

“I would like to thank my family and friends for all their support during my years at LTH. I especially want to thank Fredrik Persson and his family for always helping me to prepare for the exams that I had. I also want to thank Mats Lilja for all the interesting conversations we have had during my time at Campus Helsingborg. I will miss them!”

- Anton Botvalde

“I would like to thank my family and friends for all their support during my years at LTH.”

- Mats Lundkvist

Helsingborg, May 2012

Abstract

This thesis describes the development of a discrete event building evacuation simulator using the Java programming language and the various results that were produced from the simulations that were done. The simulator allows people to evacuate a building which has been defined via a graphical interface. People are then split into two groups, those who use elevators and those who do not. The program provides various statistics based on the simulation that was done.

Due to safety concerns, during a building evacuation only stairs are used to evacuate a building. This often leads to huge queues forming up and could potentially be avoided by allowing elevators to take some of the load of the stairs. After the 9/11 attacks in 2001 there have been some debate about changing the evacuation procedures for tall buildings. There have been previous work done on simulation of tall buildings evacuation and big time related gains were shown by allowing people to use elevators. This thesis concentrates around evacuation of smaller buildings, up to ten stories high.

The results of the thesis show that faster evacuations are possible with elevators in smaller buildings, although further research is required to generalize the results.

Keywords: Discrete event simulation, building evacuation simulation, simulation in Java, elevator simulation, elevators

Sammanfattning

Det här examensarbetet beskriver utvecklingen av en diskret händelsestyrd byggnadsutrymningssimulator med hjälp av programmeringsspråket Java samt de olika resultat som togs fram med hjälp av simuleringarna. Simulatoren tillåter människor utrymma en byggnad som har blivit definierat med hjälp av ett grafiskt gränssnitt. Människor delas sedan upp i två grupper, de som använder hissar och de som använder trappor under en utrymning. Programmet tillhandahåller diverse statistik baserad på simuleringen som utförts.

På grund av säkerhetsskäl används enbart trappor under en byggnadsutrymning. Detta leder ofta till att stora köer formas och kan potentiellt undvikas genom att låta hissar användas för att minska köerna i trapporna. Efter attackerna 11/9 2001 har en viss debatt förekommit angående ändringar av utrymningsprotokollen för höga byggnader. Det finns tidigare arbeten på utrymningssimuleringar av höghus och stora tidsrelaterade fördelar har påvisats genom att tillåta folk använda hissar. Detta examensarbete fokuserar på utrymning av mindre byggnader, upp till tio våningar höga.

Resultatet av examensarbetet påvisar att det är möjligt att uppnå snabbare evakueringstider med mindre byggnader med hjälp av hissar, men mer arbete krävs för att generalisera resultatet.

Nyckelord: Diskret händelsestyrd simulering, byggnadsutrymnings simulering, simulering i Java, simulering av hissar, hissar

List of contents

1	Introduction	1
1.1	Background.....	1
1.2	Problem description	1
1.3	Goals.....	2
1.4	Limitations.....	3
2	Work plan.....	4
2.1	Pre-study phase	4
2.2	Implementation phase	4
2.3	Test and verification phase.....	5
2.4	Source criticism	5
3	Introduction to simulation	6
3.1	Overview.....	6
3.2	Dynamical systems.....	7
3.3	Approaches to simulation	7
3.3.1	Event scheduling	7
3.3.2	Process interaction.....	8
4	Building Evacuation Simulation Tool (BEST).....	9
4.1	Overview.....	9
4.2	The GUI.....	9
4.2.1	Tabs	10
4.2.2	Floor editor	11
4.3	The graph	13
4.3.1	From a building to a graph.....	13
4.4	The simulation	14
4.4.1	From a graph to a simulation	14
4.4.2	Signals and the signal list	17
4.4.3	ElevatorVertex and Elevator.....	18
4.4.4	Main simulation loop.....	21
4.4.5	Technical information	23
4.4.6	Technical limitations	23
5	Verification of BEST	24
5.1	Modifications.....	24
5.2	The verification tests	25
5.2.1	Theoretical calculations	25
5.2.2	Verification test 1	25
5.2.3	Verification test 2.....	26
5.2.4	Verification test 3.....	27
5.2.5	Verification test 4.....	27
5.2.6	Elevator verification	28

5.2.7 Conclusion of verifications.....	29
6 Test cases.....	30
6.1 The small building test.....	31
6.1.1 Results summary of the small building	33
6.2 The medium building test	33
6.2.1 Results summary of the medium building.....	35
6.3 The large building test	35
6.3.1 Results summary of the large building	37
6.4 The extended medium building test.....	37
6.4.1 Results summary of the extended building.....	39
7 Conclusions	40
8 Further work.....	41
9 Terminology	42
10 References.....	43

1 Introduction

1.1 Background

It is for most people common knowledge to not use an elevator during an evacuation of a building when there is a fire. There are valid reasons why it could potentially be a very bad decision to use the elevator. For instance, if one would take the elevator and the power is cut by the fire, there is an overwhelming risk that one becomes trapped in the elevator. There will however always exist a category of people that will have a valid reason to use an elevator instead of stairs.

Consider, for instance, a group of elderly or disabled people in a situation where a tall building has to be evacuated. It may be difficult for them to use the stairs, and in some cases it might even be impossible – they have to use the elevator.

After the 9/11 attacks in 2001 there have been some debate about whether or not to change the evacuation procedures of tall buildings [ref 1]. For this reason and among others it is interesting to study the impact of elevator usage during an evacuation.

The inspiration for this thesis ironically came from the exceptionally poor elevators at LTH Campus Helsingborg. For a long period of time they were not working properly, if at all. It became an annoying issue for us who used them at a daily basis. They inspired us to think about various thesis topics involving elevators one way or another, and we are very pleased to have chosen this particular one. It turned out that the subject was most interesting to study and work with.

1.2 Problem description

It is possible with mathematical models to study different systems and their behaviours. However some systems are so complex in their nature that it becomes very hard, if not impossible to study them without a dynamic model. With the help of simulation it is possible to get insight of these complex models.

To study the events that occur during an evacuation of a building discrete event simulation was used in this thesis. Previous work on the topic at hand have shown significant time related benefits when allowing people to use elevators as well as stairs when evacuating tall buildings [ref 2]. Unlike the previous work, the focus lies on studying smaller buildings, up to ten stories high. With the help of simulation, it is possible to get a better understanding of the potential gains of using the regular stairs alongside elevators during an evacuation. The purpose of this thesis is to study the differences in evacuation time when allowing people to use both stairs and elevators.

1.3 Goals

Ultimately the goal with this thesis was to study what kind of impact elevators would have during a building evacuation with different types of buildings. To do so, a simulation tool had to be developed. The tool would consist of two parts, one with a graphical interface which allows the user to define a building and a second part which actually ran the simulations.

The graphical tool had to:

- define a building with the help of a graphical interface
- change the layout of the building
- allow different testing parameters to be set
- save a building to a file
- load a building from a file

The simulation tool had to:

- run a simulation based on the building
- provide useful statistics of the simulation
- be verified to be correct

Because of our previous knowledge with the Java programming language it was decided that all software development would be done in Java in order to save time.

Since neither of us had any previous experience with simulation we had some goal of our own as well:

- learn about discrete event simulation and how it could be applied to our work

- investigate different ways to represent a building in a graphical interface and how it can be used for a simulation
- investigate different data structures to represent a building for a simulation
- investigate different ways to represent people in a building and how they navigate in the building

1.4 Limitations

Due to time constraints and complexity, some simplifications were made. Most of them are related to the behaviours of the people in the building. In an evacuation where there is a real fire people act differently than they normally do. There are likely people who panic and act irrational, everyone just want to get out as fast as possible to safety.

Such behaviours are not taken into account in this thesis. While such factors are important, the purpose of this thesis is to study the *impact of elevators* in an evacuation and not human behaviour. The simulations do not take such things into account; all people behave exactly the same. The goal for an individual in the simulation is to get out using a *predetermined fixed path*, thus a person *queues as long as necessary* in order to get out of the building.

As mentioned earlier, using an elevator in a building which is on fire could be dangerous due to power outages, fire blockage etc. The simulations in this thesis only focuses on evacuations to the extent that it acts as a fire drill, there is *no actual fire* being simulated. More technical limitations are discussed in chapter 4.

2 Work plan

In the beginning of the thesis a preliminary work plan was created. The plan contained several different phases. The phases were – a pre-study phase, an implementation phase, a test and verification phase and finally the final phase which would consist of final polish and report writing. Since the project work plan was not cast in stone the work plan was not followed strictly, but was rather followed with a more flexible approach. Sometimes jumps were made between phases in order to proceed more efficiently with the thesis.

A more in depth overview of the constructed simulation tool, called Building Evacuation Simulation Tool (BEST), is described in chapter 4.

2.1 Pre-study phase

The pre-study phase mostly consisted of information gathering. Several papers and books about elevator traffic and elevators in general were found and studied [ref 3, 4, 5]. Afterwards an introduction to simulation was provided by Christian Nyberg who is the supervisor for this thesis. Christian, who is a lecturer for the advanced simulation course at LTH, provided course material from the simulation course and a private introduction lecture about simulation.

Two home assignments about the development of simulation systems were done. These provided a good insight of how a simulation tool would have to be designed for this thesis.

2.2 Implementation phase

The implementation phase was divided into three stages (in order) – a GUI (Graphical User Interface) development stage, a simulation development stage and a last stage which combined both to a fully functional program.

In the beginning of the GUI stage a lot of decisions were made since the GUI would define the rest of the program. Several different types of graphical layouts were discussed before finally choosing one that suited the project needs.

The simulation development started after the base GUI was functional. In this stage functionality was created so that data from the GUI could be used to run a simulation.

In the final stage enhancements of both the GUI tool and the simulation part were made in order to improve the whole system.

2.3 Test and verification phase

In order to insure that the simulation tool provided correct results mathematical verifications were made. The different test cases (buildings) were simulated and their results studied.

2.4 Source criticism

The different information sources used in this thesis has either been from published articles or literature used in university computer science programs. Since the articles have been peer reviewed by others they have to be considered reliable. The books have been used in a wide range of university programs and are also considered reliable.

3 Introduction to simulation

3.1 Overview

There are several ways to study a system. With different mathematical models it is possible to study some systems. For instance, it is possible to study population growth with various mathematical models. Another way to study a system is to simply observe or experiment with it. Consider a cashier queue at a grocery store. It is possible to observe and measure the queue in order to understand the system.

There are many reasons why simulation is a good way to study a system [ref 6]. Some systems are so complex that it becomes very difficult or even impossible to study them without a dynamic model. In such cases it is possible with the use of simulation to study the system.

Consider the previous cashier example. This system requires a lot of time to test different types of scenarios. For example if one would want to study queue time differences between having one and two open cashiers it would require a lot of effort. Many measurements would have to be done before any conclusive results could be made. All these measurements require a significant time investment. With a computer such a simulation can be done many times in a very short period of time. Some systems may even be dangerous to experiment with or cost a lot of money in order to study them. These reasons among others are why simulation is an efficient tool to study and experiment with different systems.

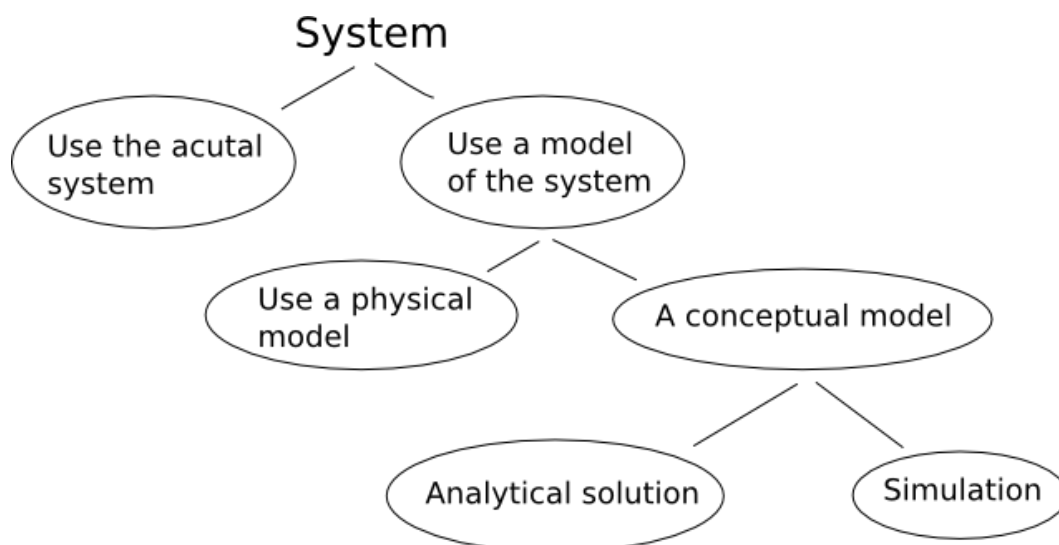


Figure 3.1: *Different ways to study a system*

3.2 Dynamical systems

There are essentially three kinds of systems:

- Continuous-time systems
- Discrete-time systems
- Hybrid systems

In a continuous system the state variable changes in a continuous way. The state change in a continuous-time system cannot be done abruptly. An example of a continuous system is the temperature in an oven. These systems are often described with differential equations.

In a discrete-time system a state can abruptly change to another. Such systems can be described by queues.

Hybrid systems are systems that contain both continuous-time and discrete-time subsystems.

Since the simulation in this thesis is described by using queues, the system is discrete.

3.3 Approaches to simulation

There are two approaches to model a system for simulation, event scheduling and process interaction.

3.3.1 Event scheduling

In order to do an event scheduling simulation the systems different states needs to be analysed and defined. After this the different events that triggers the states needs to be defined. This is done in order to create rules which define how an event is handled.

A simulation consists of different events being triggered at different times. In order to keep track of which events shall occur an event list is needed. The event list keeps the order sorted by the arrival time in ascending order. An event must contain a time and an event type. It is not unusual that an event also contain some kind of attribute (e.g. which state added the event).

12	21	30	Time
Ready	Down	Up	Event
B	A	C	Attribute

Figure 3.2: An event list containing three events, sorted by time

With the help of an event list it is now possible to perform a simulation. A simple simulation could look like the following:

1. Pop the first event in the event list
2. Update the simulation time to the event time
3. Change state depending on the event type
4. Execute current state and add new events if needed
5. If the list is empty, quit, otherwise repeat steps

3.3.2 Process interaction

The event scheduling method and process interaction have a lot of in common but differ on a few key points. Process interaction simulations consist of processes. A process has some internal states and communicates with each other by sending signals. When a process receives a signal it treats the signal and various state changes and further signal sending may occur. If a signal is created, the sender must assign an arrival time to the receiver.

A simulation consists of different processes treating signals. Just like with event scheduling, a list containing the signals is needed. This list is ordered by arrival time in ascending order.

With the help of the signal list it is now possible to perform a simulation. A simple simulation could look like the following:

1. Pop the first event in the signal list
2. Update the simulation time to the arrival time
3. Activate receiver process
4. Process executes current state and sends new signals if needed
5. If the list is empty, quit, otherwise repeat steps

4 Building Evacuation Simulation Tool (BEST)

4.1 Overview

The BEST system is a program written in the Java programming language with many classes working together. In this chapter a more in depth overview of the different components will be explained.

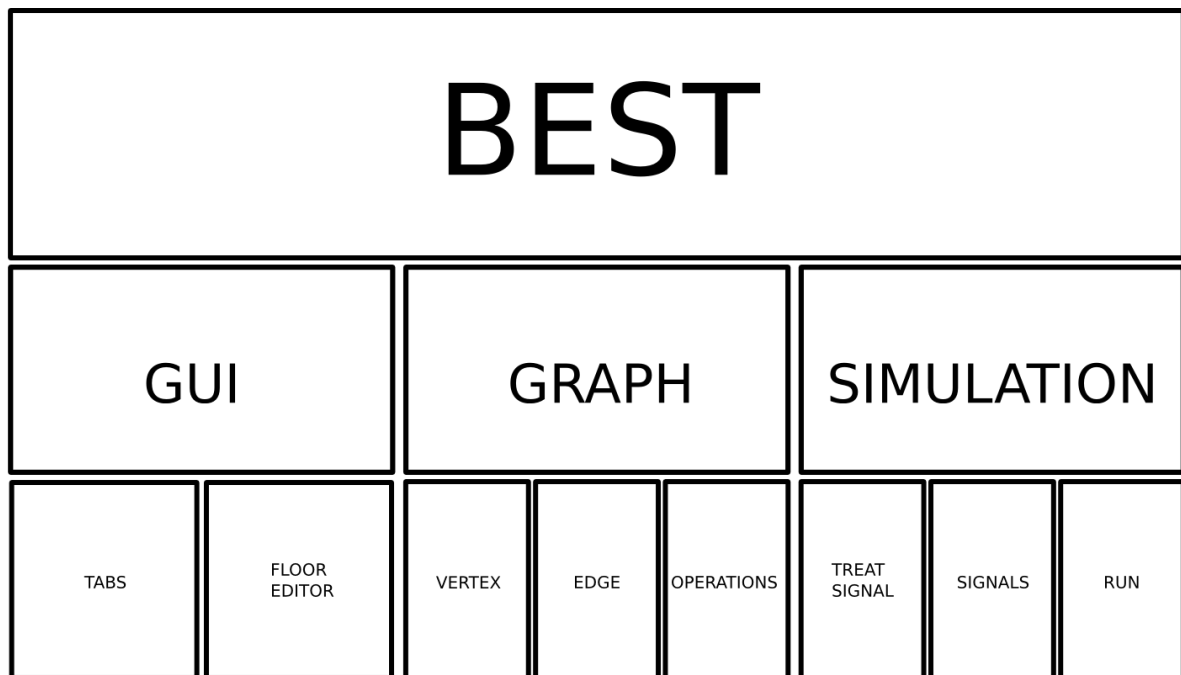


Figure 4.1: An overview of the major system components in BEST

The system is designed with three major parts that interact with each other to form the complete system. They are the GUI part, the graph and finally the actual simulation. The main workflow for the system is first to model a building with the GUI. While the building is actually being modelled, a graph is constructed containing the important data which is required in order to run a simulation.

4.2 The GUI

In the beginning of the implementation phase there were two main ideas for the GUI. One was a form of a cell based editor (which is sometimes used for 2D game development. The idea was that it could be possible with the help of a number of cells to define a room. The room could then be connected with

other nearby rooms by performing some calculations. This way it would be possible to construct a building.

The other idea was to use a canvas to draw different types of rooms (with shapes). It was then possible to connect rooms to each other to form a building.

An early realisation was that both these methods could work to visually represent a building. However they both suffered from the problem that they were difficult to model with a data structure. They worked well visually but the transition from a visual model to a functional model that could be used for simulation made them unwanted.

Instead of thinking of how to represent a building visually, the solution was to think of the functional representation of a building. With the help of a working functional model it would be easier to construct a visual one.

An idea was to model buildings with the help of a graph. It would be easy to construct a graph and to represent it visually in a way that it resembles a floor in a building.

4.2.1 Tabs

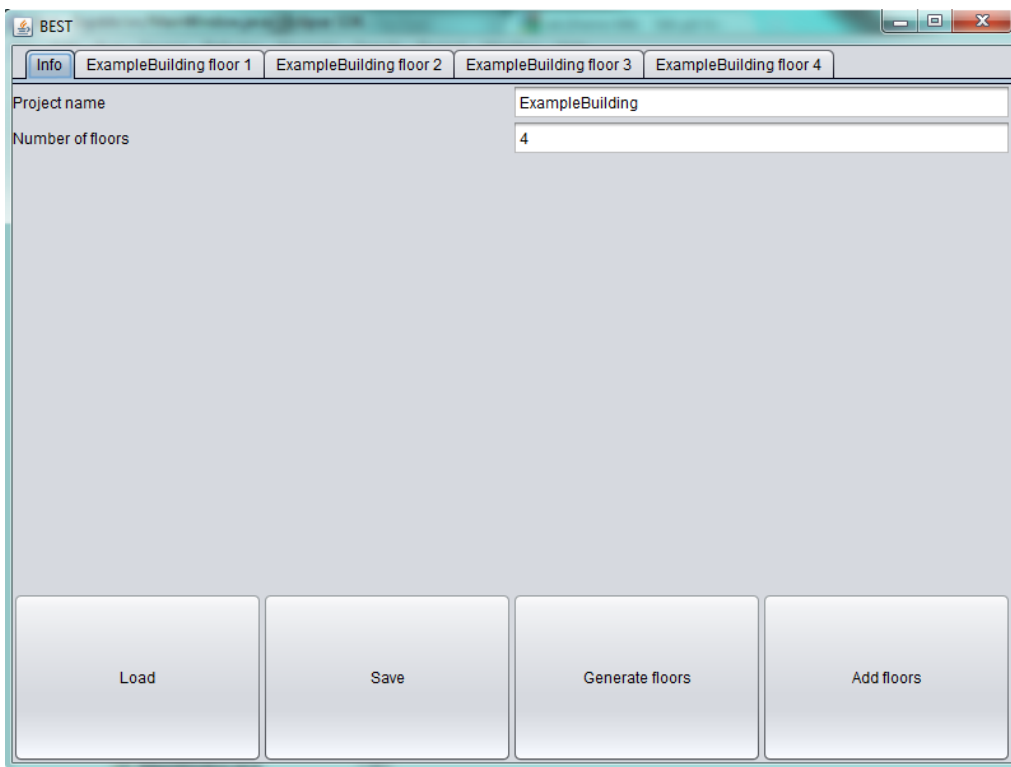


Figure 4.2: An overview of the tab layout used in BEST

The BEST system uses a tabbed layout to represent floors in a building. When the program is started only a single tab exists from the start, the information tab (info tab). In figure 4.2 the info tab is highlighted. With the help of this tab it is possible to create a new project, load an old project, save the current project and to extend the number of floors the project has. The generated tabs contain each a floor editor to model the building.

4.2.2 Floor editor

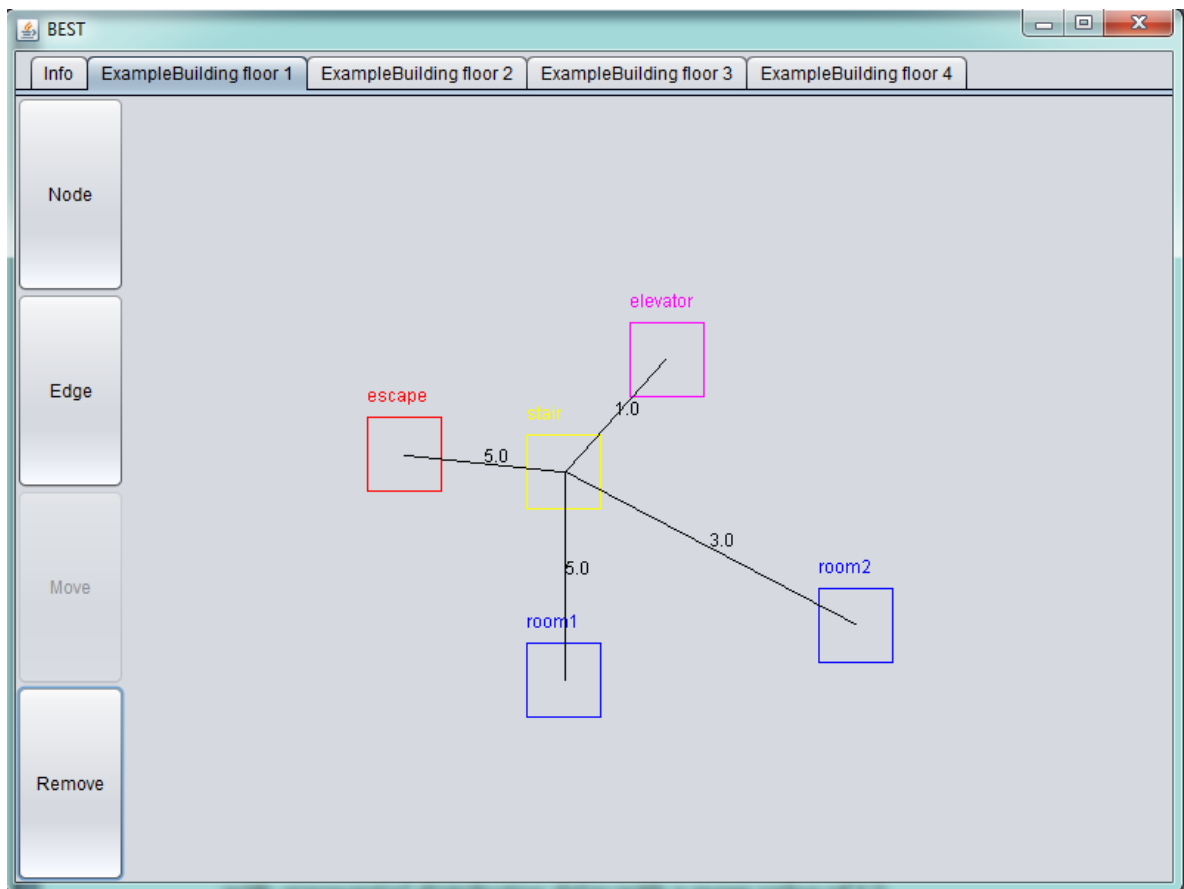


Figure 4.3: An overview of a floor

In Figure 4.3 a floor editor is displayed with five nodes. Each node represents some kind of area. There are a total of four different placable nodes in BEST. They are

- Room node (*RoomVertex*)
- Stair node (*StairVertex*)
- Elevator node (*ElevatorVertex*)
- Escape node (*EscapeVertex*)

By placing nodes on each floor and connecting them with edges it is possible to form a model of a building.

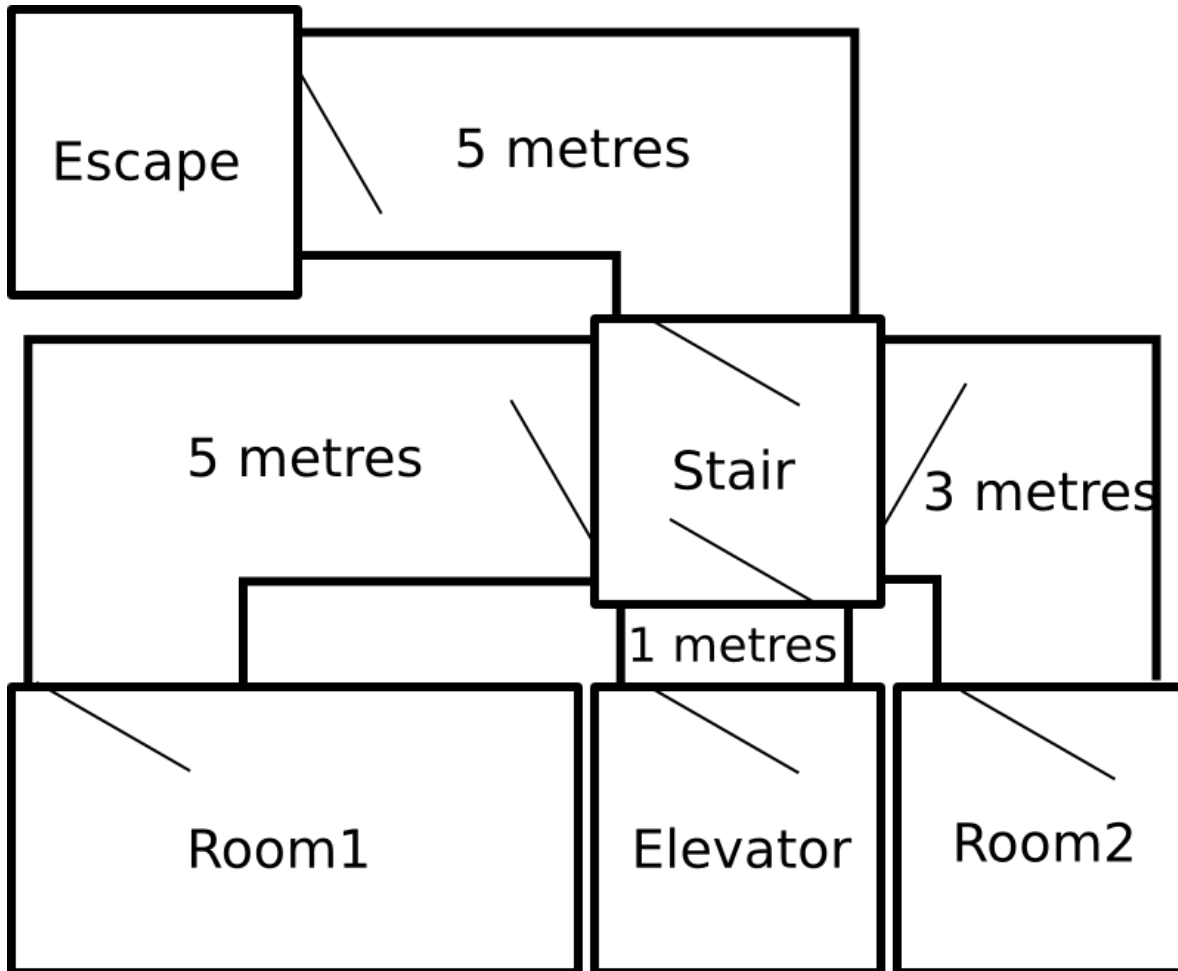


Figure 4.4: A conceptual representation of the floor in figure 4.3

The edges represent hallways in some way between doors. Floors are connected to each other with stairs and elevators.

A floor editor can perform five main operations:

- Place nodes
- Connect nodes to each other with edges
- Move nodes and edges
- Remove nodes and edges
- Set and change properties of nodes and edges

4.3 The graph

Once it became clear that a graph would be needed in order to construct a building it was realised that writing a custom made graph would not be necessary since there would likely exist graph libraries on the internet. Unfortunately the libraries that were found online were surprisingly difficult to use. Because all computer engineering students at LTH Campus Helsingborg take the course Algorithms and Data structures it was remembered that the course literature had a robust chapter on graph theory [ref 7]. This chapter also contained a fully functional graph class that was easy to use. Even though it was fully functional, several big adjustments had to be made in order to suit the BEST system.

4.3.1 From a building to a graph

Consider a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. Consider now a simplified building model which can be said to have a set of rooms that are connected to each other. Such a building can be represented by a graph, where the rooms represent vertices and the distance between them are the edges.

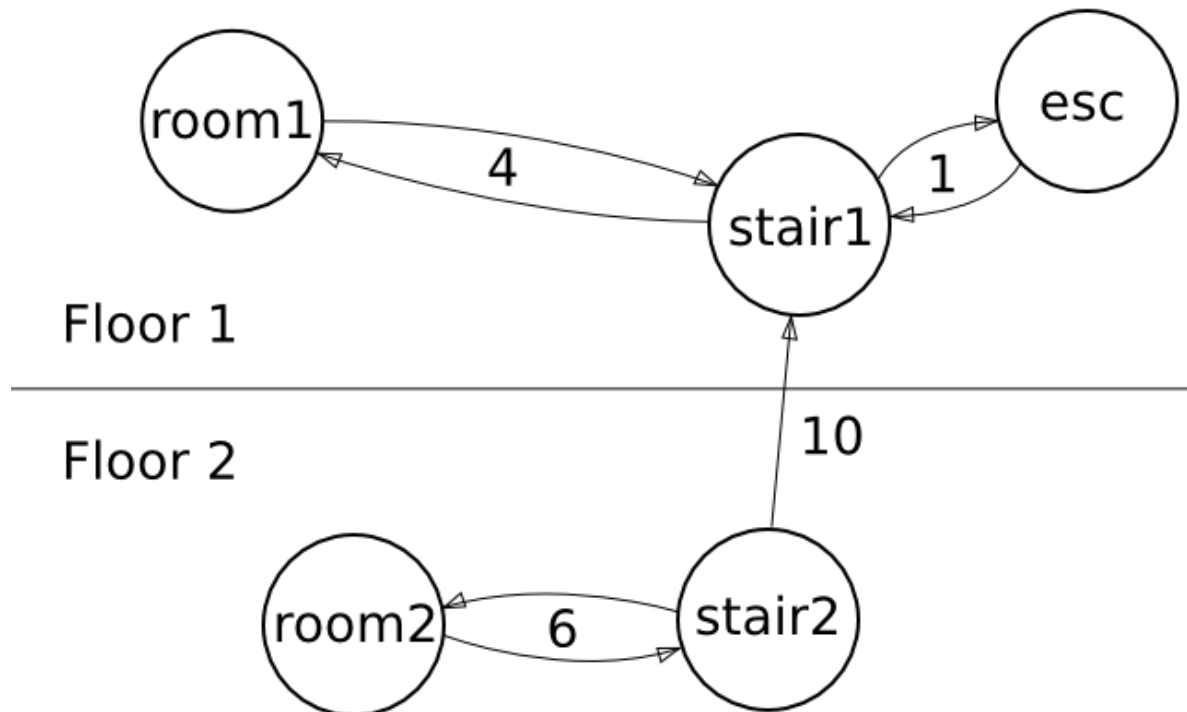


Figure 4.5: A building with two floors represented as a graph. Notice the single direction between stairs due to technical implementation in BEST

The graph in Figure 4.5 has five vertices,

$$V = \{esc, room1, room2, stair1, stair2\}$$

and seven edges,

$$E = \left\{ \begin{array}{l} (esc, stair1, 1), (room1, stair1, 4), (room2, stair2, 6), (stair1, esc, 1) \\ (stair1, room1, 4), (stair2, room2, 6), (stair2, stair1, 10) \end{array} \right\}$$

With a functional graph it is easy to traverse it and calculate paths for people to use. Consider for example a person standing in room2 in Figure 4.5. In order to calculate a path from room2 to the exit (esc) Dijkstra's shortest path algorithm can be used.

Since the BEST system would need four different placeable node types containing unique data for each type. Four sub classes were created with the original Vertex acting as the super class.

When a node is created in the floor editor it is added to the graph (this is possible since all placable nodes inherit from Vertex). The different nodes can then be connected to each other by adding edges with a cost, which represent the distance between the nodes. When an edge is added to the graph, it is added both ways. However between two stairs they are not added both ways. This was an early design decision for BEST since during an evacuation normally people only move down in the building.

4.4 The simulation

A building evacuation can be described with the help of queues. When people move in the building, different queues arise. A door in a room or the stairs of a floor is a typical place where large queues form up. The BEST system uses discrete event simulation for this reason. The simulation method was chosen to be process interaction since the simulation is based on many processes sending signals to each other.

4.4.1 From a graph to a simulation

In order to simulate a building, a valid graph is needed. A valid graph must contain at least one exit (EscapeVertex) and for each room (RoomVertex) there must be a path leading to an exit. Before the simulation can run all the data created by the user must be collected and processed. The goal for a

person in a building is to use its assigned path to reach the exit. A simulation will end when all the persons in the building has reached their exit. To fully understand how the simulation works a brief explanation of the different nodes is needed. The nodes are:

- Vertex
- RoomVertex
- StairVertex
- ElevatorVertex
- EscapeVertex

The Vertex is the super class for all other vertices. It contains attributes which all types of vertices have in common and information needed in order to build a proper graph. A Vertex has a priority queue containing persons. It also contains different measurement variables and *level of service* (LOS) variable. The LOS variable describes how many persons a queue can work with at the same time. For example, say that a Vertex has a LOS of 2. This means that the door in the room can only service a maximum of 2 people at a time. The measurement variables and priority queue will be discussed in further detail later in this chapter.

The RoomVertex is the only node in the graph where people can exist in the beginning of a simulation. A room has a certain width and height (area), a max number of people in it which is used in case a random number of persons should be generated. If a random number is not wanted it is possible to set a fixed number instead. A percentage can also be set to indicate how many from the specified room should use an elevator. If wanted, a room can be set not to generate any persons in the beginning of a simulation.

The StairVertex and ElevatorVertex are used to connect floors with each other. If a building is constructed with more than one floor stairs or elevators are needed in order to build a valid graph.

EscapeVertex is the final destination for each person. Like with all real life buildings a building must contain at least one exit.

Before the simulation start all the persons in the building need to know what path they should take. This is done by evaluating the layout of the current floor. Say for instance that a floor has two different stairs and the building has two different exits. Then the persons generated on the floor will be equally split among all the stairs and exits. The same applies for buildings with elevators as well. A person has a random walking speed between 0.5 and 1. When walking

between two stairs then speed is half the normal one.

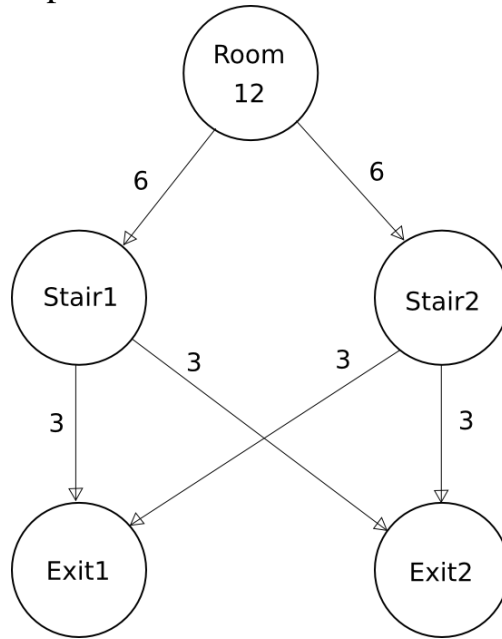


Figure 4.6: The person distribution for a room between two stairs and exits

A RoomVertex has a width and height. This allows persons to be placed at a random position in a room. To simplify calculations in BEST, each room has a door set at the upper left corner. Since it takes a certain time to reach the door in a room a person's path list will always begin with a path to the same room with a distance to the door.

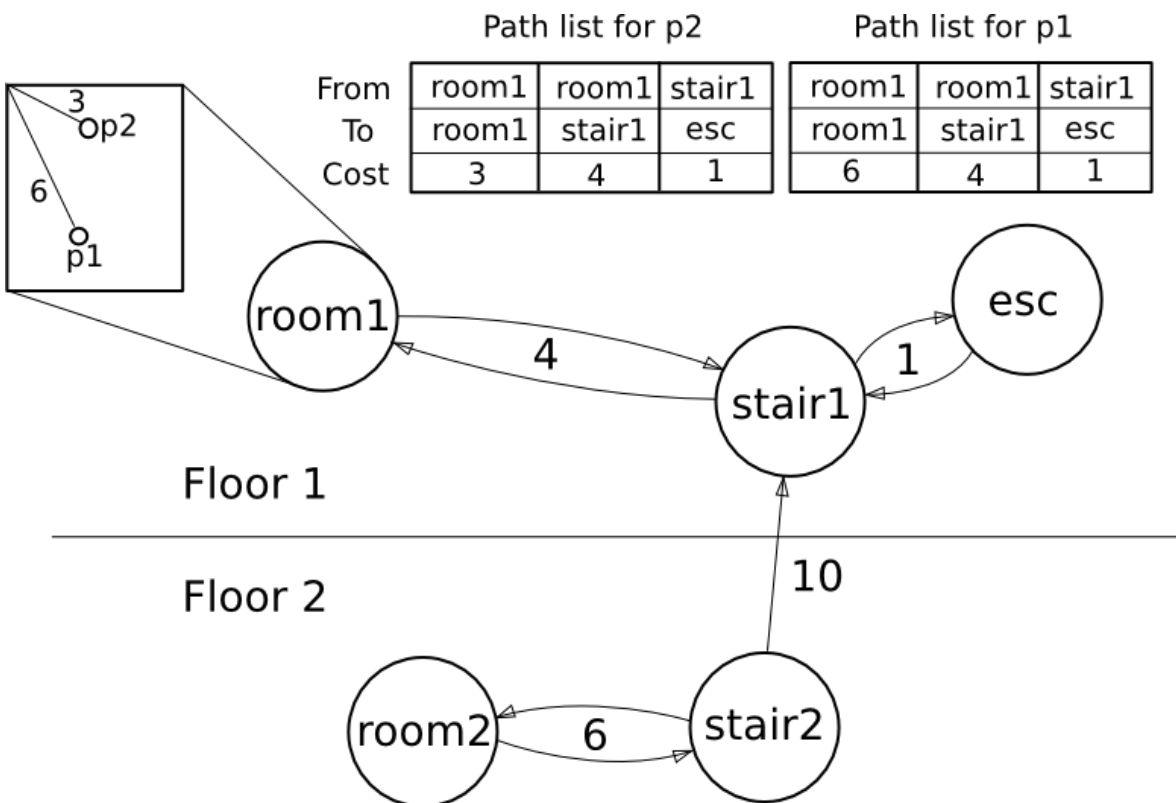


Figure 4.7: Room1 containing two persons. Their path list is displayed

When the persons in the rooms have been generated the simulation can begin.

4.4.2 Signals and the signal list

A signal contains a type, a destination, an arrival time and a person.

The BEST system uses six types of different process signals. They are:

1. Arrival
2. Ready
3. Measure
4. Pickup
5. Down
6. Release

Signals 4-6 are elevator specific and will be treated in the next section.

```
public void TreatSignal(Signal x) {
    switch (x.signalType) {
        case ARRIVAL:
            numberInQueue++;
            if (numberInQueue > los) {
                person.add(new PersonContainer(x.person, x.arrivalTime));
                x.person.updateTimes(this);
            }
            else {
                x.person.updateTimes(this);
                SignalList.SendSignal(READY, this, time + 1, x.person);
            }
            break;
        case READY:
            numberInQueue--;
            Vertex current = x.person.currentDestination();
            Vertex dest = x.person.nextDestination(true);

            if (dest != null) {
                if (dest instanceof StairVertex && current instanceof StairVertex) {
                    SignalList.SendSignal(ARRIVAL, dest,
                        time + (x.person.getDestinationCost() / x.person.getStairSpeed()), x.person);
                }
                else {
                    SignalList.SendSignal(ARRIVAL, dest,
                        time + (x.person.getDestinationCost() / x.person.getNormalSpeed()), x.person);
                }
            }
            else if (this instanceof EscapeVertex) {
                x.person.updateTimes(this);
                count++;
            }
            if (!person.isEmpty()) {
                SignalList.SendSignal(READY, this, time + 1.0, person.poll().p);
            }
            break;
        case MEASURE:
            noMeasurements++;
            accumulated = accumulated + numberInQueue;
            SignalList.SendSignal(MEASURE, this, time + 2, null);
            break;
    }
}
```

Figure 4.8: *The TreatSignal method in Vertex*

When signal is sent to a Vertex the TreatSignal method is invoked. There are three states that can occur.

If the treated signal is of type arrival then the number of people in the queue is increased. If the number of people in the queue is greater than the level of service then the person is added to queuing list. Otherwise it sends a new signal to the same process in ordered to be served (in other words the person is ready to walk through the door).

When a ready signal arrives the number of persons in the queue is decreased. A check is made to establish whether the person has reached the final destination. If not, then the person is sent to its next destination. Before leaving the vertex, the person next in the queue is alerted that the queue is now ready to serve.

The measure signal is an independent signal that measures the activity in the queue.

The signal list is globally available for all processes in the system. This allows vertices to send signals between each other. Just like described in the simulation chapter, the signal list is sorted by arrival time.

4.4.3 ElevatorVertex and Elevator

There are two types of elevator modes in BEST, a normal mode and an emergency mode. The elevator has a priority queue with *destinations* (a destination is a call from a floor with a certain time). Depending on which mode is selected different priorities are made. In normal mode the queue is sorted by time in ascending order while in the emergency mode the sorting is made by floors in descending order. When a simulation begins all elevators are located on the bottom floor. When a call to an elevator in normal mode is received the following happens:

1. Add the call to the destination list
2. If the elevator is not busy pop the first element in the queue
3. Pickup at the popped destination
4. If there are destinations below the current one, go to those floors
5. Release the persons at the bottom floor

In normal mode, even if the elevator is at max capacity it will make stops at lower floors. In emergency mode no such stops will occur in order to save time.

The ElevatorVertex uses a modified method call of TreatSignal in order to function properly. If the ElevatorVertex is on the bottom floor, the TreatSignal call works the same way as in any other Vertex; otherwise a queue to the elevator is formed. Many ElevatorVertex objects share a special elevator class with each other in order to form a working elevator. This elevator class uses a heavily modified TreatSignal method.

```

public void TreatSignal(Signal x) {
    switch(x.signalType) {
        case ARRIVAL:
            dests.add(new DestinationContainer(x.person.getFloor(), x.arrivalTime,
emergency));
            if(dests.size() == 1 && theOne == null) {
                SignalList.SendSignal(READY, this, time, x.person);
            }
            break;
        case READY:
            theOne = dests.poll();
            SignalList.SendSignal(PICKUP, this, time +
((theOne.getDestination()*cost)/speed), x.person);
            break;
        case PICKUP:
            ElevatorVertex current = null;
            for(ElevatorVertex e : ev) {
                if(e.getFloor() == theOne.getDestination()) {
                    current = e;
                    break;
                }
            }
            int count = 0;
            PersonContainer p;
            do {
                p = current.pop();
                if(p != null) {
                    count++;
                    persons.add(p.p);
                }
            } while(count < capacity && p != null);

            SignalList.SendSignal(READY, current, time, x.person);
            next = new DestinationContainer(-10, -10, emergency);
            for(DestinationContainer d : dests) {
                if(d.getDestination() < theOne.getDestination()) {
                    next = d;
                    break;
                }
            }
            if(next.getDestination() == -10) {
                SignalList.SendSignal(RELEASE, this, time + (count/current.getLos()) +
((theOne.getDestination()*cost)/speed), x.person);
            }
            else {
                if (emergency) {
                    if (persons.size() >= capacity)
                        SignalList.SendSignal(RELEASE, this, time +
(count/current.getLos()) + ((theOne.getDestination()*cost)/speed), x.person);
                    else
                        SignalList.SendSignal(DOWN, this, time +
(count/current.getLos()) + (((theOne.getDestination()-next.getDestination())*cost)/speed),
x.person);
                }
                else
                    SignalList.SendSignal(DOWN, this, time + (count/current.getLos()) +
(((theOne.getDestination()-next.getDestination())*cost)/speed), x.person);
            }
            break;
        case DOWN:
            counter++;
            if(persons.size() >= capacity) {
                last = new DestinationContainer(next.getDestination(), next.getTime(),
emergency);
            }
    }
}

```

```

        int save = next.getDestination();
        next.setDestination(-10);

        for(DestinationContainer d : dests) {
            if(d.getDestination() < last.getDestination()) {
                next = d;
                break;
            }
        }
        if(next.getDestination() == -10) {
            next.setDestination(save);
            SignalList.SendSignal(RELEASE, this, time + 3 +
((last.getDestination()*cost)/speed), x.person);
        }
        else {
            SignalList.SendSignal(DOWN, this, time + 3 +
(((last.getDestination()-next.getDestination()*cost)/speed), x.person);
        }
    }
    else {
        dests.remove(next);
        last = new DestinationContainer(next.getDestination(), next.getTime(),
emergency);
        next.setDestination(-10);
        ElevatorVertex curr = null;
        for(ElevatorVertex e : ev) {
            if(e.getFloor() == last.getDestination()) {
                curr = e;
                break;
            }
        }
        int counter = 0;
        PersonContainer pc;
        do {
            pc = curr.pop();

            if(pc != null) {
                counter++;
                persons.add(pc.p);
            }
        } while(counter < capacity && pc != null);
        SignalList.SendSignal(READY, curr, time, x.person);
        for(DestinationContainer d : dests) {
            if(d.getDestination() < last.getDestination()) {
                next = d;
                break;
            }
        }
        if(next.getDestination() == -10) {
            SignalList.SendSignal(RELEASE, this, time + (counter/curr.getLos()) +
((last.getDestination()*cost)/speed), x.person);
        }
        else {
            SignalList.SendSignal(DOWN, this, time + (counter/curr.getLos()) +
(((last.getDestination()-next.getDestination()*cost)/speed), x.person);
        }
    }
    break;
case RELEASE:
    for(Person per : persons) {
        SignalList.SendSignal(ARRIVAL, first, time +
(persons.size()/((ElevatorVertex) first).getLos()), per);
    }
    persons.clear();
    if(dests.size() > 0)
        SignalList.SendSignal(READY, this, time +
(persons.size()/((ElevatorVertex) first).getLos()), x.person);
    theOne = null;
    next = null;
    last = null;
    break;
case MEASURE:
    break;
}
}

```

Figure 4.9: *The TreatSignal method in Elevator*

When a signal is sent from an ElevatorVertex to an Elevator there are five states that can occur (technically six with measure, although it is not used). Unlike other signals, the signal to an elevator contains a destination instead of a person because an elevator is a moving queue and people cannot be served unless the elevator is at the same floor as them.

The arrival state works the same way as in Vertex. When an elevator call with a destination arrives it is added to the queue and treated if the elevator is not busy.

The ready signal pops the queue and serves the popped element. This means that the elevator just released people and is ready for a new destination.

In the pickup state the elevator has arrived to its destination. The elevator gets filled with people until the max capacity is reached or the people queue becomes empty. If emergency mode is set the elevator will only stop and pickup people at lower levels if the max capacity is not reached. In normal mode the elevator stops at each lower floor where a call has been made even if max capacity has been reached.

The down signal is essentially the same as the pickup state. If the elevator has reached max capacity only a door opening will occur.

When a release signal is treated all the people in elevator will be sent to the bottom ElevatorVertex's queue. From there on, people will move normally again. If the destination list in the elevator is not empty, the whole process will be repeated.

4.4.4 Main simulation loop

```
ArrayList<Vertex> all = this.getAllVertices();
for(Vertex v : all) {
    if(v instanceof RoomVertex) {
        ((RoomVertex) v).init();
    }
    else {
        SignalList.SendSignal(Global.MEASURE, v, Global.time, null);
    }
}
for (Person p : persons) {
    SignalList.SendSignal(Global.ARRIVAL, p.getStart(), Global.time +
        (p.getDestinationCost() / p.getNormalSpeed()), p);
}
while(Global.count < persons.size()) {
    actSignal = SignalList.FetchSignal();
    Global.time = actSignal.arrivalTime;
    actSignal.destination.TreatSignal(actSignal);
}
```

Figure 4.10: *The main simulation loop*

When a valid graph has been constructed the simulation can start. The simulation can be described as the following:

1. Initialize all rooms and begin to measure each vertex
2. Send the first arrival in every person's path
3. Pop the first signal in the signal list
4. Set current time to signal arrival time
5. Invoke TreatSignal for the next destination
6. If there are people left in the building, repeat from step 3

When the simulation is complete BEST presents a list of relevant simulation data.

```

===== 1 =====
Room      Time      ActTime  QTime
room1     4,57      4,57     0
exit      5,71      6,71     0

Total time:      12,28
Total qtime:     0
Avg time per room: 6,14
Avg qtime per room: 0
=====

===== 0 =====
Room      Time      ActTime  QTime
room1     4,99      4,99     0
exit      8,32      9,9      0,58

Total time:      15,89
Total qtime:     0,58
Avg time per room: 7,95
Avg qtime per room: 0,29
=====

Number of persons: 2
Avg persons per floor: 2
Total simulation time: 15,89
Total queue time: 0,58
Avg time per person: 14,09
Avg qtime per person: 0,29

```

Figure 4.11: *The output of a simulation with two persons in one room and exit*

BEST can also produce optional data such as the entire signal list and individual signal list for each person. Such data can be interesting to study when a more in depth overview of an evacuation is needed.

4.4.5 Technical information

In order for the queues to work properly they have a fixed service time. For instance, in a room this time represents the time it takes to pass through the door. For RoomVertex, StairVertex and EscapeVertex this time is set to 1. For elevators the service time is based on the number of persons entering the elevator and the elevators LOS plus a constant value for opening and closing the doors. This constant is set to 3. For people that get out of the elevator there is only a door opening and thus the constant is half of the original value. For the tests that were made in Chapter 6 cubic rooms were used. Because each person have a randomly selected walking speed it is possible for persons to walk pass each other in queues.

Before a simulation begins, each room distributes travel paths to all people and each room has a percentage set which sets the total amount of people that goes to the elevator. For instance, if a room has 10 people and 5% is set to go to the elevator then 0 people will go to the elevator (because of rounding). If instead 19% would, 1 person would go. BEST is written without threads.

4.4.6 Technical limitations

There are some technical limitations in BEST that are important to understand.

Each person is given a fixed travel path. This path is never changed during a simulation. This means that people will queue as long as necessary in order to take their path even if there might be a better nearby path with less queue time. This does not affect the outcome of the main analysis since all people behave the same way, even if they use a stair or an elevator.

People can only move down in a building. When a connection between two stairs is made in the graph, it is only made one way, from the high stair to the low one. The same is somewhat conceptually true for an elevator. A person can never enter the elevator on the bottom floor and exit at another.

An elevator always is set always to the first floor in the beginning of the simulation and exits may only be placed at the first floor.

Because rooms are connected to each other with the graph, the distance between them is always the same. This means that all persons that walk the same path will have the exact same walking distance. In a real life example, if two people walk along side each other in a curved hallway, their walking distance will not be the same.

In BEST a building can have several elevators side by side on each floor. They will however not work together; they all operate independently of each other.

5 Verification of BEST

Before studying the simulation data provided by BEST it is very important to verify the correctness of it. Even though the data might appear to be correct it is still important to actually verify it. In fact all simulations programs must be verified to be correct and BEST is no exception. It is possible with different mathematical approaches to verify the correctness of the simulation. Since the BEST system uses queues for the simulations regular queuing theory can be applied to validate the accuracy of the system.

5.1 Modifications

Since the simulations done by BEST are based on a finite amount of people in a building some small changes had to be done in order to be able to verify the correctness of the system. If a lot of changes are done to a system in order to verify it then only the modified system will be verified and not the actual system.

In order to verify the simulations the following was changed:

- Instead of a room generating a fixed amount of persons, a *generator* was added to each room. The generator sent a new person to the room with *exponential distribution delay* with a mean value of $1/\lambda$.
- The generator sent a person directly to the door.
- A person no longer had a random speed; instead a constant value of 1 was given.
- The serving time in every queue was set to an exponential distributed delay with a mean value of 1.
- The distance between the queues for each person was exponential distributed with a mean of the given cost between the queues.
- Instead of showing individual results of each person, the simulation now showed the average number of persons in every queue.

With the help of queuing theory these changes allowed verification of the system. The exponential distribution function has the nice property of being memoryless which simplifies the verification calculations.

These changes did not affect how the queues worked in general and thus did not impact the system in any significant way. This implies that if the modified system is shown to be correct, then the original system is also correct.

5.2 The verification tests

To verify the BEST system four different verification tests were made. For each test four different values on λ were used. The different tests were then simulated and the results compared with theoretical ones. Instead of ending the simulation after a certain amount of time, the tests ended when a specific amount of people had exited the building.

5.2.1 Theoretical calculations

In order to verify the system some theoretical values needed to be calculated and then compared with results from the simulations. The following formulas were used:

$$N = \frac{\rho}{1-\rho}, \quad \rho = \frac{\lambda}{\mu}$$

where N is the average number of people in a queue, ρ is the average time a is queue busy, λ is the arrival intensity and μ the serving for a queue [ref 8].

5.2.2 Verification test 1

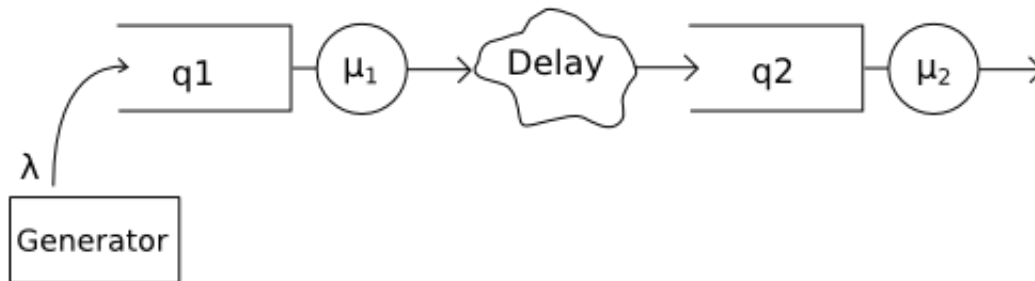


Figure 5.1: A queuing network with one room and one exit

$$\mu_1 = \mu_2 = 1$$

Each test ran until 100 000 persons had exited the building. Each test was repeated 250 times and the average values were calculated.

λ	μ	ρ	Theoretical N	Simulated N q1	Simulated N q2
0.2	1	0.2	0.25	0.25011	0.25001
0.5	1	0.5	1	0.99985	1.00033
0.8	1	0.8	4	3.97880	3.98411
0.9	1	0.9	9	8.96887	9.05950

5.2.3 Verification test 2

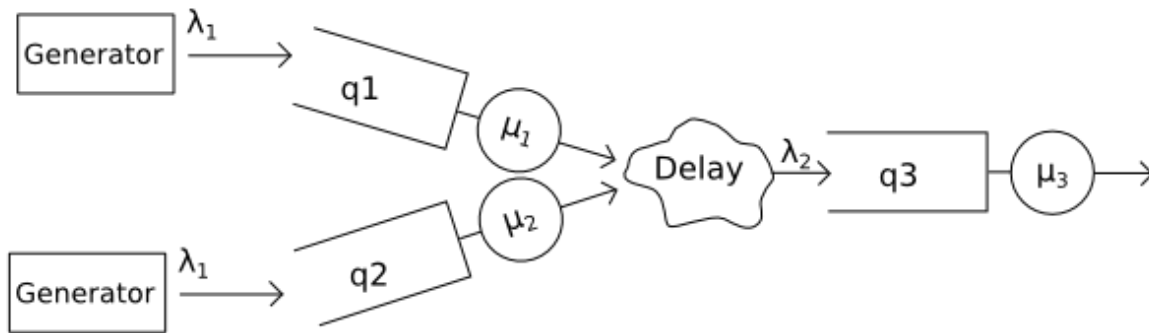


Figure 5.2: A queuing network with two rooms and one exit

$$\mu_1 = \mu_2 = \mu_3 = 1$$

Each test ran until 100 000 persons had exited the building. Each room generated 50 000 persons each. Each test was repeated 250 times and the average values were calculated.

λ_1	μ	ρ	Theoretical N	Simulated N q1	Simulated N q2
0.1	1	0.1	0.11111	0.11074	0.11051
0.25	1	0.25	0.33333	0.33267	0.33260
0.4	1	0.4	0.66667	0.66631	0.66418
0.45	1	0.45	0.81818	0.81440	0.81584

Because there are two queues sending people to q3, q3's λ will be doubled.

λ_2	μ	ρ	Theoretical N	Simulated N q3
0.2	1	0.2	0.25	0.24947
0.5	1	0.5	1	0.99360
0.8	1	0.8	4	3.99213
0.9	1	0.9	9	8.97835

5.2.4 Verification test 3

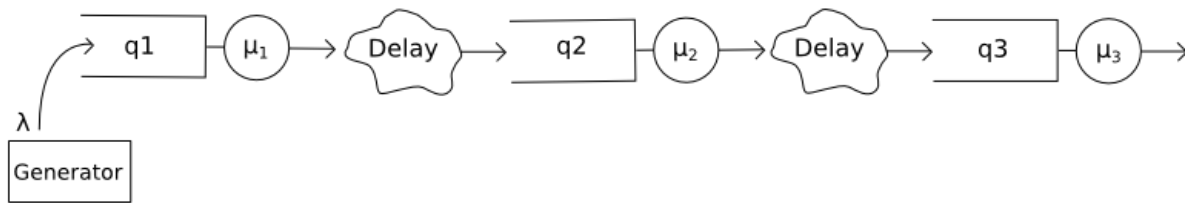


Figure 5.3: A queuing network with two rooms and one exit

$$\mu_1 = \mu_2 = \mu_3 = 1$$

Each test ran until 100 000 persons had exited the building. Since q1 is the only room with a generator it generates all persons. Each test was repeated 250 times and the average values were calculated.

λ	μ	ρ	Theoretical N	Simulated N q1	Simulated N q2	Simulated N q3
0.2	1	0.2	0.25	0.25003	0.24976	0.24999
0.5	1	0.5	1	1.00645	1.00001	0.99950
0.8	1	0.8	4	3.99011	3.99050	4.00531
0.9	1	0.9	9	8.95008	8.98131	8.84814

5.2.5 Verification test 4

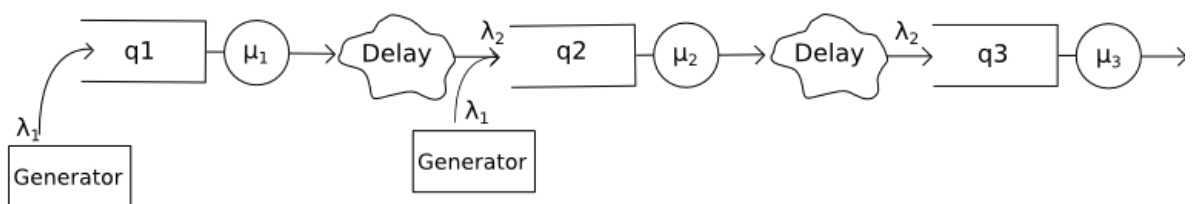


Figure 5.4: A queuing network with two rooms and one exit

$$\mu_1 = \mu_2 = \mu_3 = 1$$

Each test ran until 100 000 persons had exited the building. Each room generated 50 000 persons each. Each test was repeated 250 times and the average values were calculated.

λ_1	μ	ρ	Theoretical N	Simulated N q1
0.1	1	0.1	0.11111	0.11097
0.25	1	0.25	0.33333	0.33246
0.4	1	0.4	0.66667	0.66381
0.45	1	0.45	0.81818	0.81535

Because q2 gets persons from both q1 and a generator, q2's λ will be doubled. Since q3 is affected by q2, q3's λ will be doubled as well.

λ_2	μ	ρ	Theoretical N	Simulated N q2	Simulated N q3
0.2	1	0.2	0.25	0.24923	0.24954
0.5	1	0.5	1	0.99578	0.99514
0.8	1	0.8	4	3.96403	3.99758
0.9	1	0.9	9	8.89662	8.90404

5.2.6 Elevator verification

The previous verification tests were made with no elevators. In order to verify elevators in BEST a different approach was used since the elevators do not behave in the same manner as a normal queue (an elevator is a “dynamic queue”).

To verify elevators Little's Law was used. Little's Law states that the average number of people in a queue system is equal to arrivals per second times the average time spent in the system.

$$N = \lambda \cdot T \quad (\text{Little's Law})$$

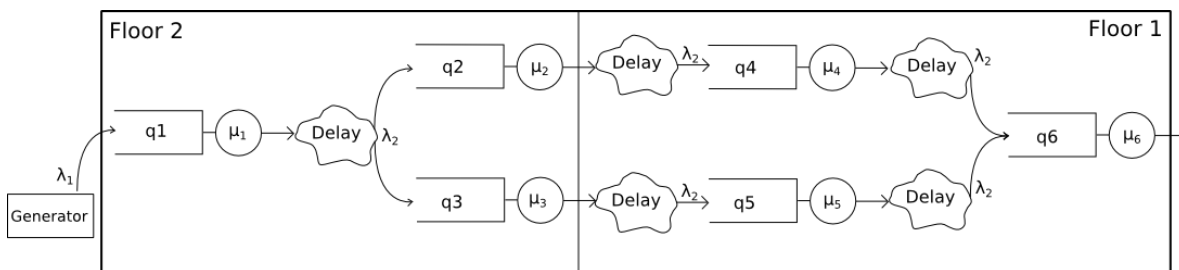


Figure 5.5: A queuing network with one room, one elevator, stair and exit

$$\lambda = \lambda_1 = 0.2$$

$$\mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5 = \mu_6 = 1$$

Each test ran until 100 000 persons had exited the building. Since q1 is the only room with a generator it generates all persons. Every other person was sent to q2. Each test was repeated 250 times.

During the test the time (T) and the average number of people in the building (N) was measured.

$$T = 34.62101$$

$$N = 6.92338$$

According to Little's Law the average number of persons in the house would be

$$N = \lambda \cdot T \rightarrow N = 6.92420$$

5.2.7 Conclusion of verifications

From all the different verification tests no major deviation between the theoretical results and the simulated results were found. Since the modified system has shown to be correct it implies that the original one is correct as well.

6 Test cases

To study the differences between evacuation times four buildings were modelled; a small, medium, large and an extended medium building. Each building was simulated several times, with an increasing percentage of people choosing an elevator to exit the building. In this way it was possible to plot a full scale graph from 0% usage to 100%. All floors in a building are identical, with the exception of the first floor which has a number of exits.

Building size	Small	Medium	Large	Extended Medium
Number of floors	3	5	10	10
Number of stairs	1	1	2	1
Number of elevators	1	2	4	2
Number of exits	1	2	4	2
Persons per floor	20	200	400	200
Rooms per floor	3	10	20	10
Elevator capacity	5	10	15	10
Elevator speed	1	1.25	1.5	1.25
Height between floors	3	3	4	3
Stair length	6	6	8	6
LOS elevator	2	2	2	2
LOS stair	2	2	2	2
LOS rooms	1	1	2	1
LOS exit	2	2	3	2
Number of 5x5 rooms	6	20	0	40
Number of 7x7 rooms	0	15	0	30
Number of 9x9 rooms	0	0	80	0
Number of 10x10 rooms	3	15	80	30
Number of 12x12 rooms	0	0	40	0
Number of simulations per %	25 000	25 000	1000	10000

Figure 6.1: *Building description of the test cases*

Room type	Number of persons in room
5x5	5
7x7	10
9x9	15
10x10	20
12x12	30

Figure 6.2: Room capacity overview in the test cases

Due to the complexity of the large and the extended medium building the simulations per percent had to be lowered.

6.1 The small building test

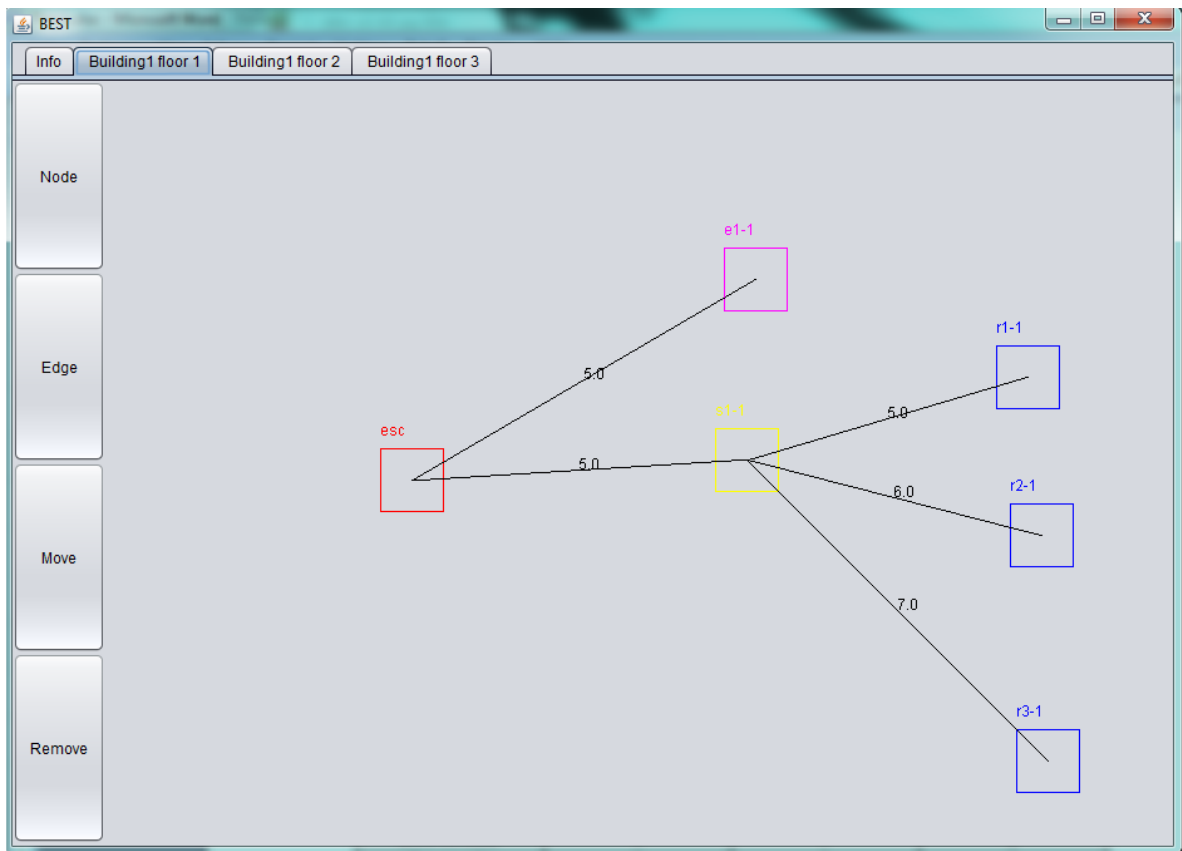


Figure 6.3: Building layout of the small building (first floor is shown)

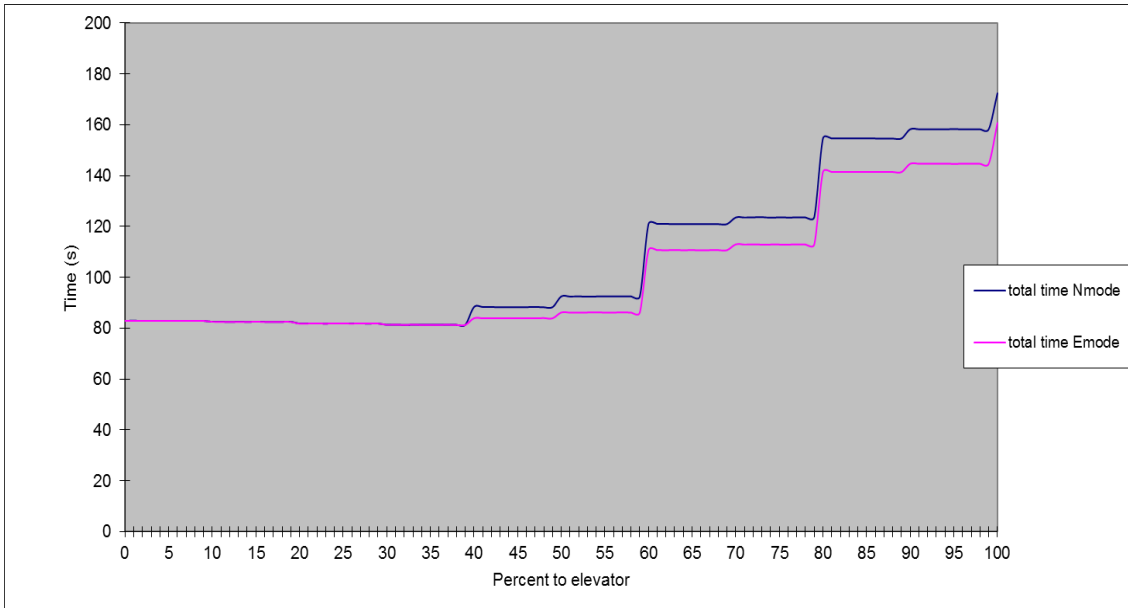


Figure 6.4: Total evacuation time for the small building

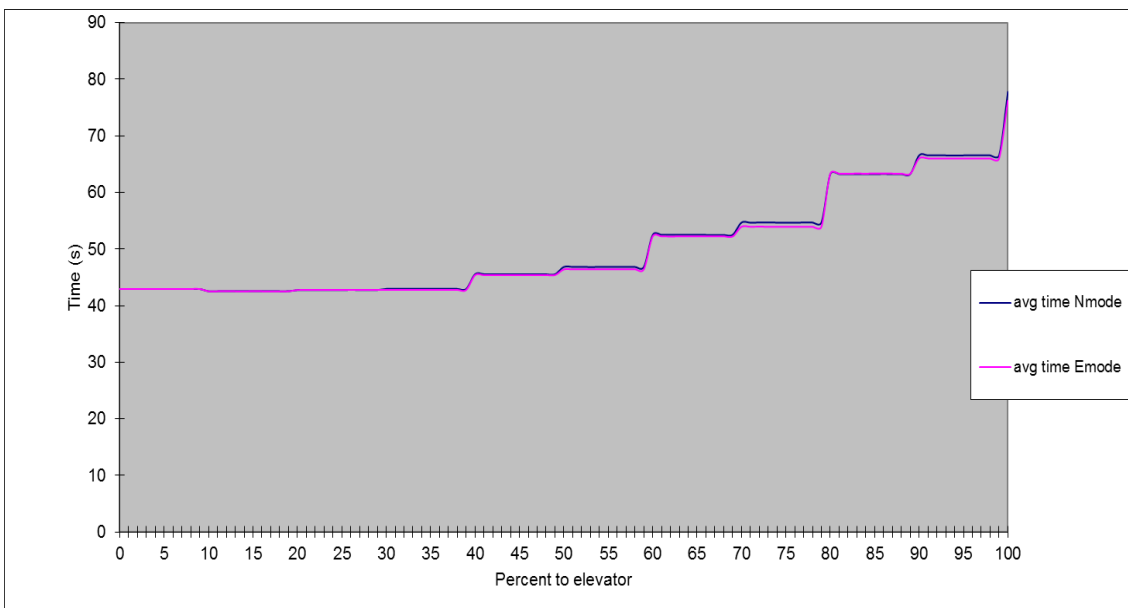


Figure 6.5: Average evacuation time per person in the small building

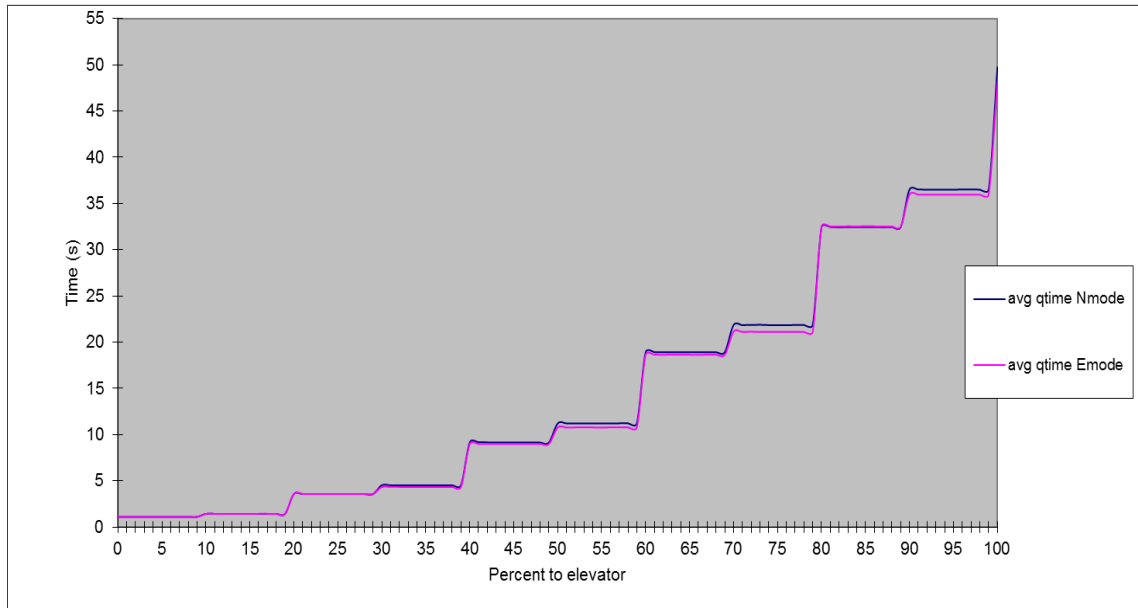


Figure 6.6: Average queue time per person in the small building

6.1.1 Results summary of the small building

Figure 6.4 show that there are some very small time gains around when 20% - 40% use the elevator. When the percentage of people that use the elevator increases, large queues form up in front of the elevator. Since the elevator has a small capacity people have to wait a long time in order to get out.

6.2 The medium building test

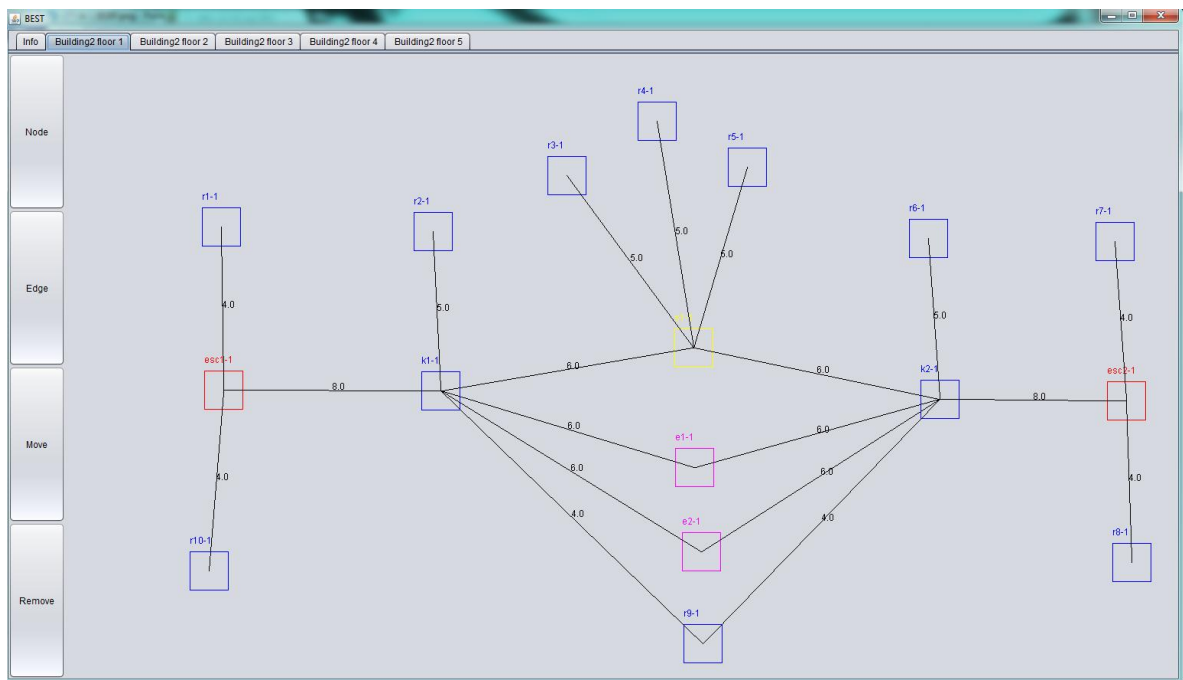


Figure 6.7: Building layout of the medium building (first floor is shown)

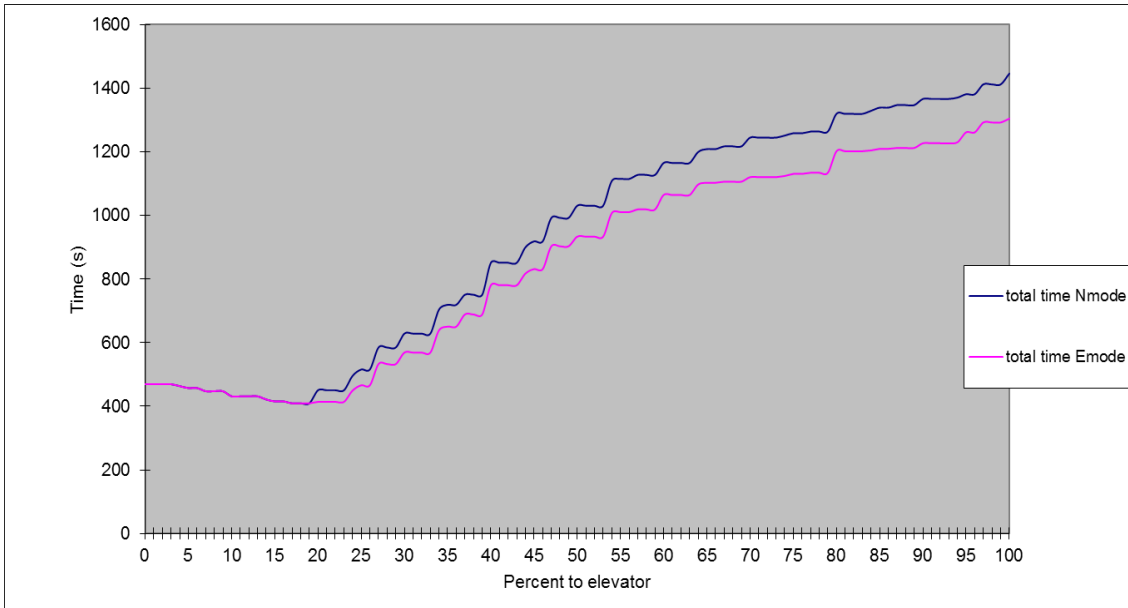


Figure 6.8: Total evacuation time for the medium building

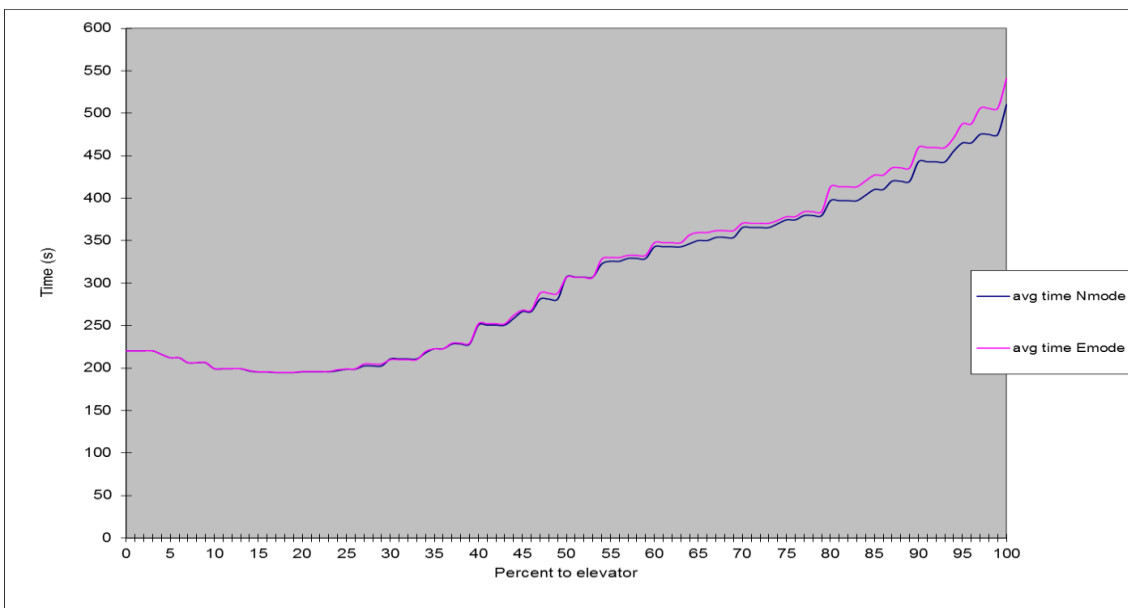


Figure 6.9: Average evacuation time per person in the medium building

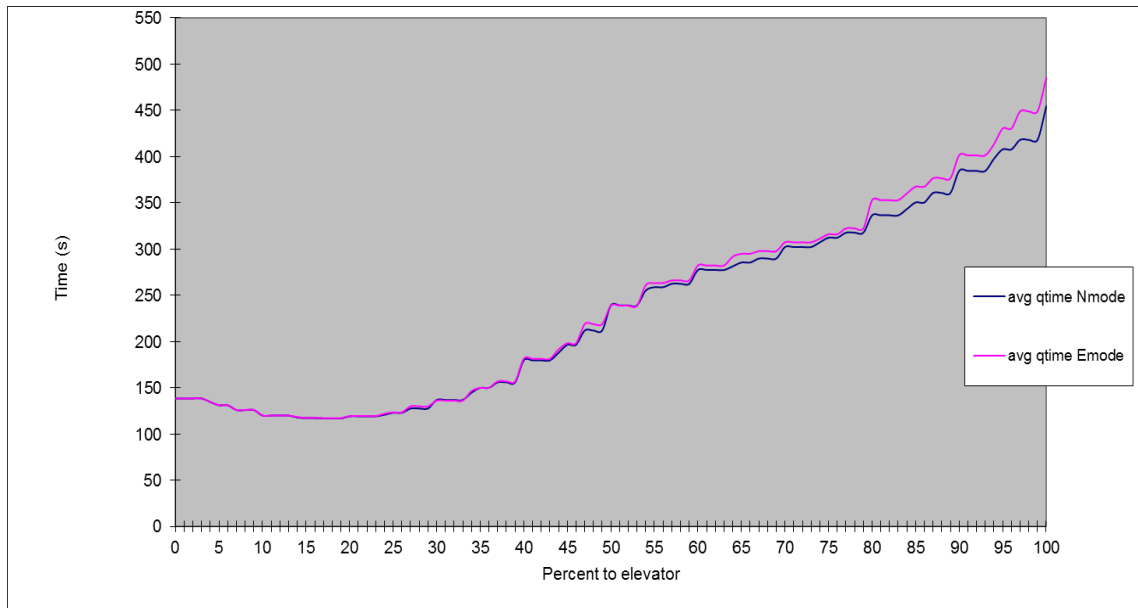


Figure 6.10: Average queue time per person in the medium building

6.2.1 Results summary of the medium building

Figure 6.8 show that there are significant time gains when 10% - 25% use the elevator, up to 60 seconds faster. Around these percentages the elevator capacity works well with the amount of arriving. When the percentage of people that use the elevator increases, large queues form up in front of the elevator and increase the total evacuation time.

6.3 The large building test

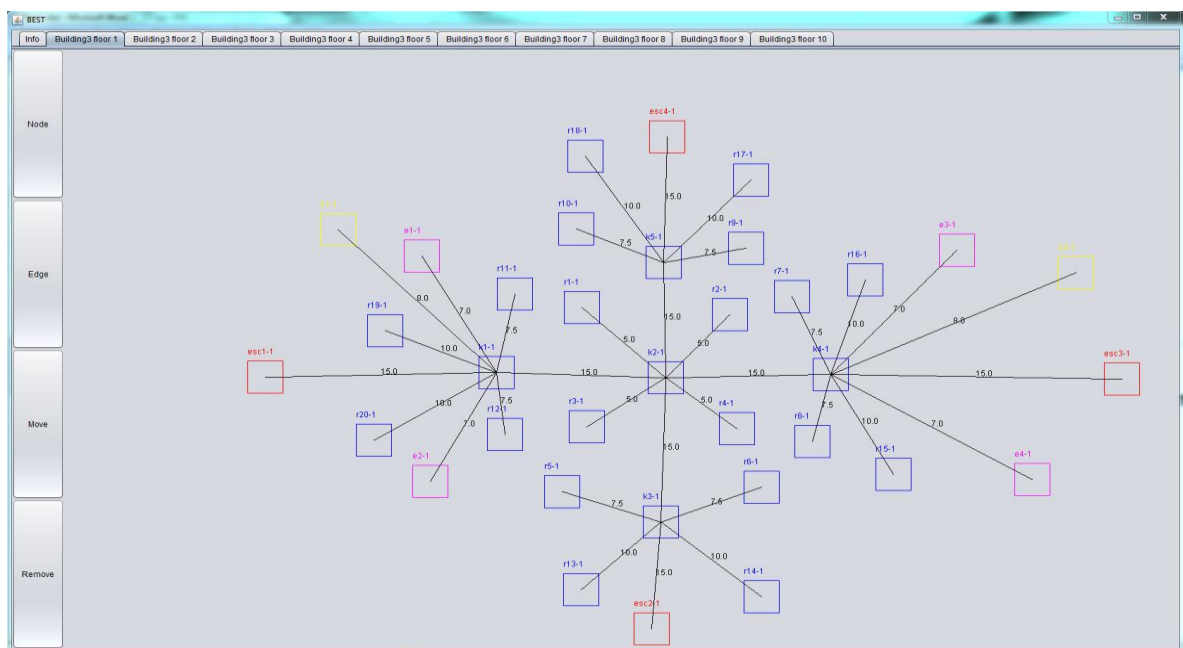


Figure 6.11: Building layout of the large building (first floor is shown)

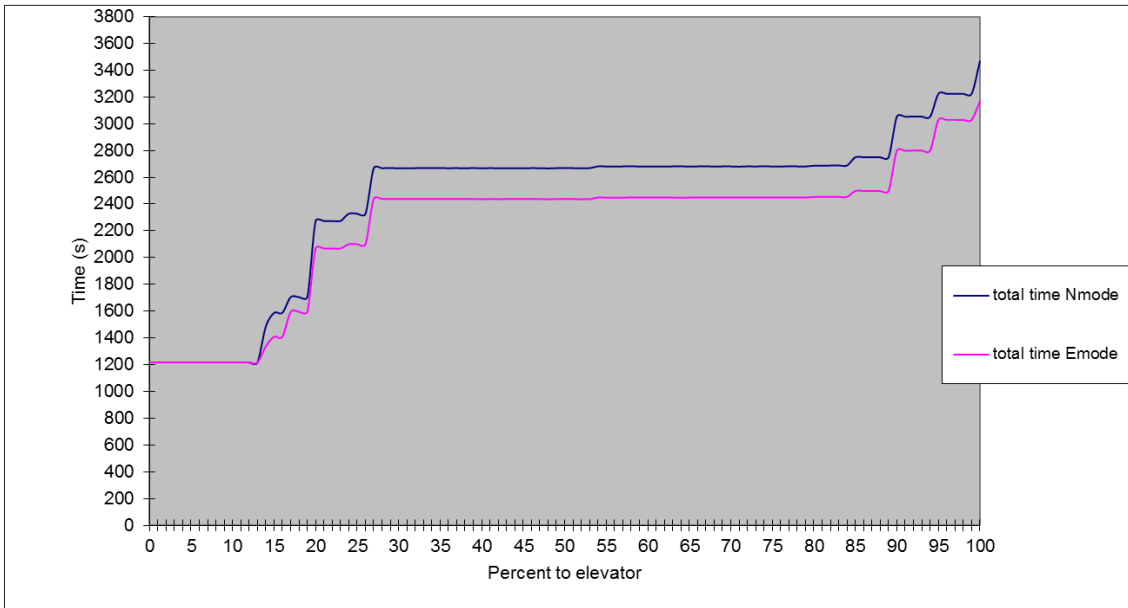


Figure 6.12: Total evacuation time for the large building

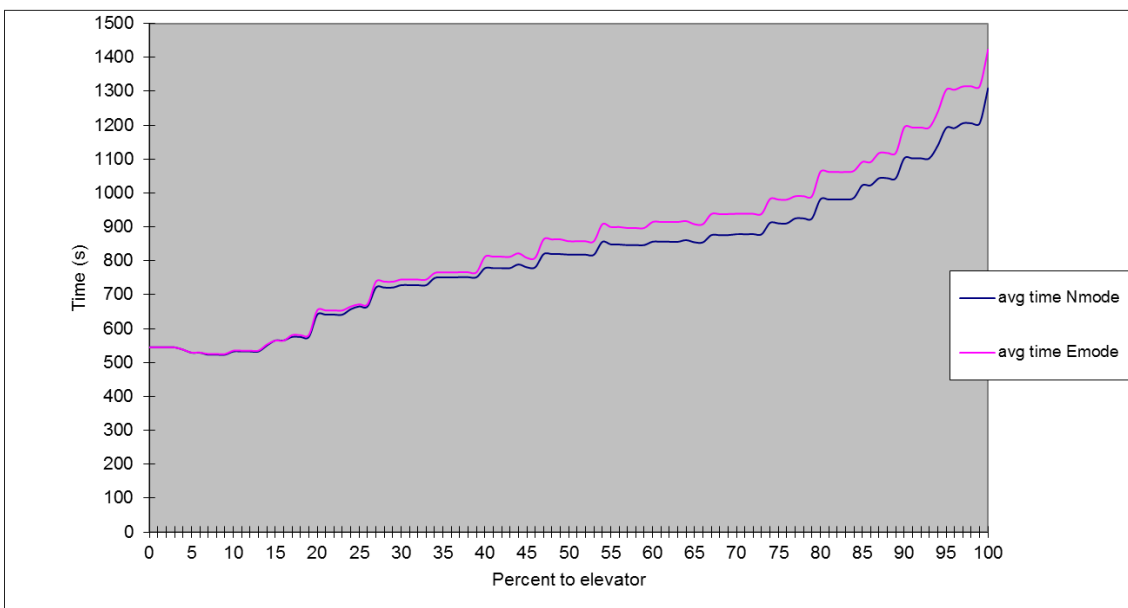


Figure 6.13: Average evacuation time per person in the large building

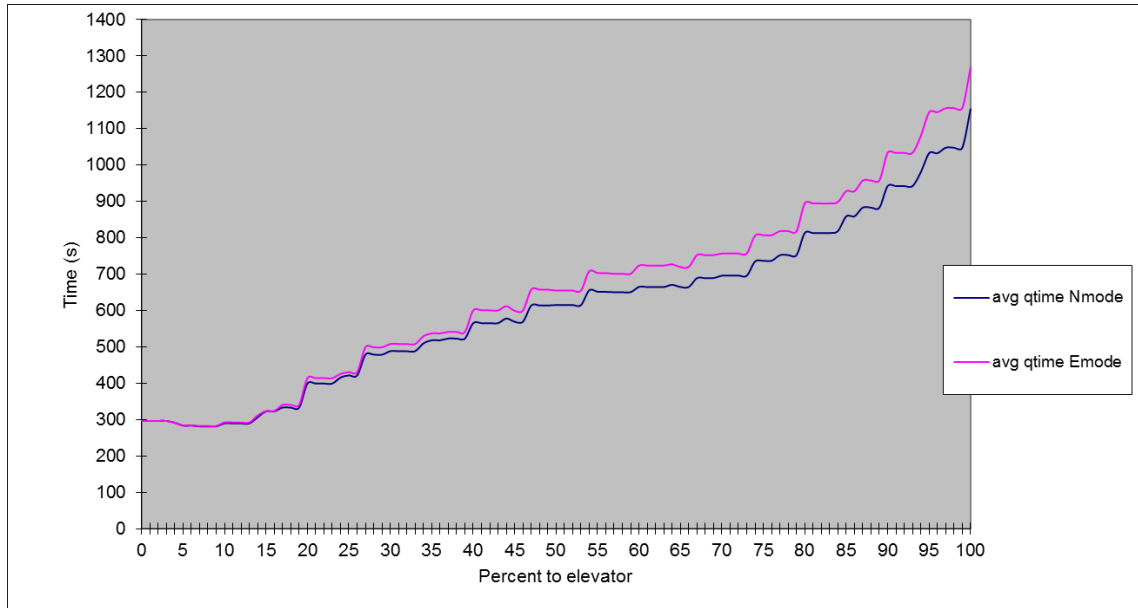


Figure 6.14: Average queue time per person in the large building

6.3.1 Results summary of the large building

The simulations produced some interesting results. The total evacuation time shown in Figure 6.12 did not show any time gains when allowing people to use the elevators. However the average time a person spends in the building, shown in Figure 6.13, is lowered when around 5-10% uses the elevator.

6.4 The extended medium building test

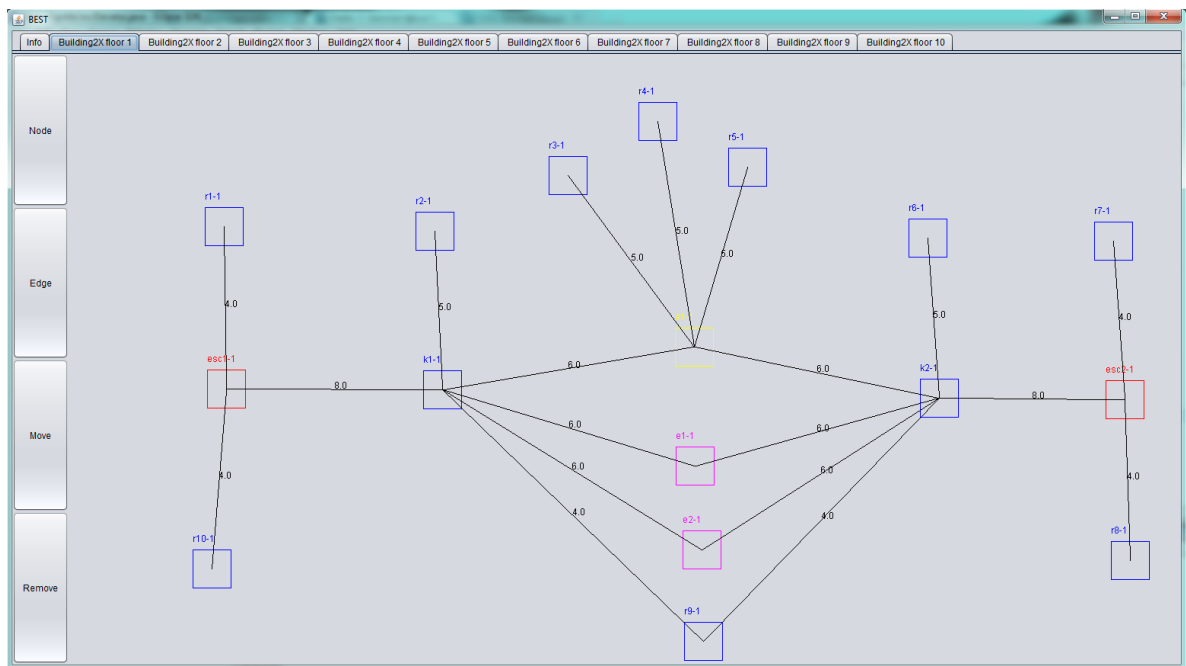


Figure 6.15: Building layout of the extended building (first floor is shown)

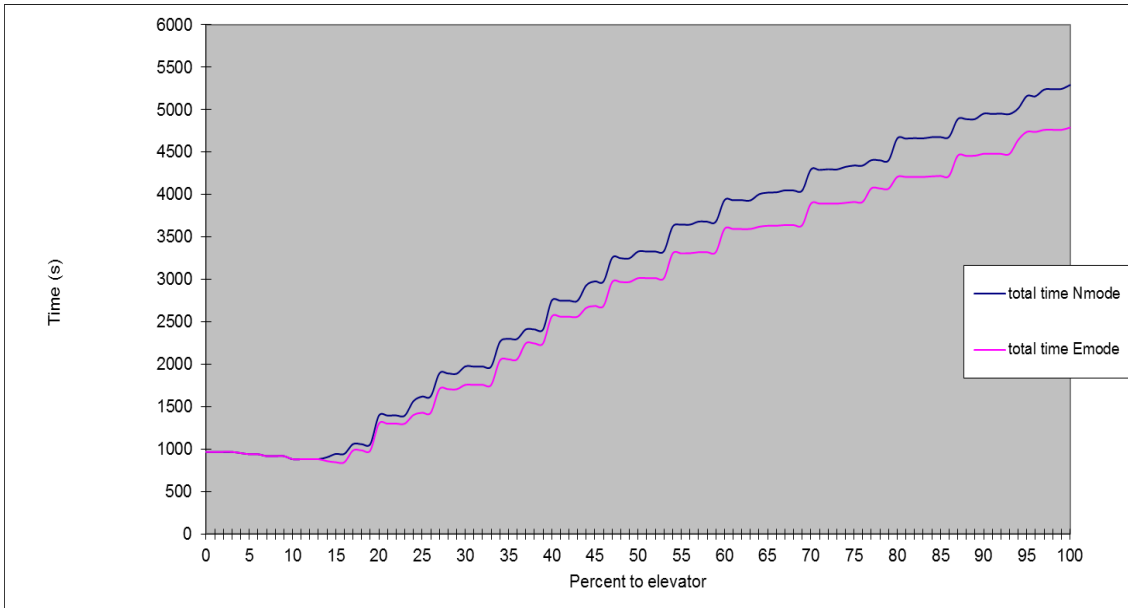


Figure 6.16: Total evacuation time for the extended building

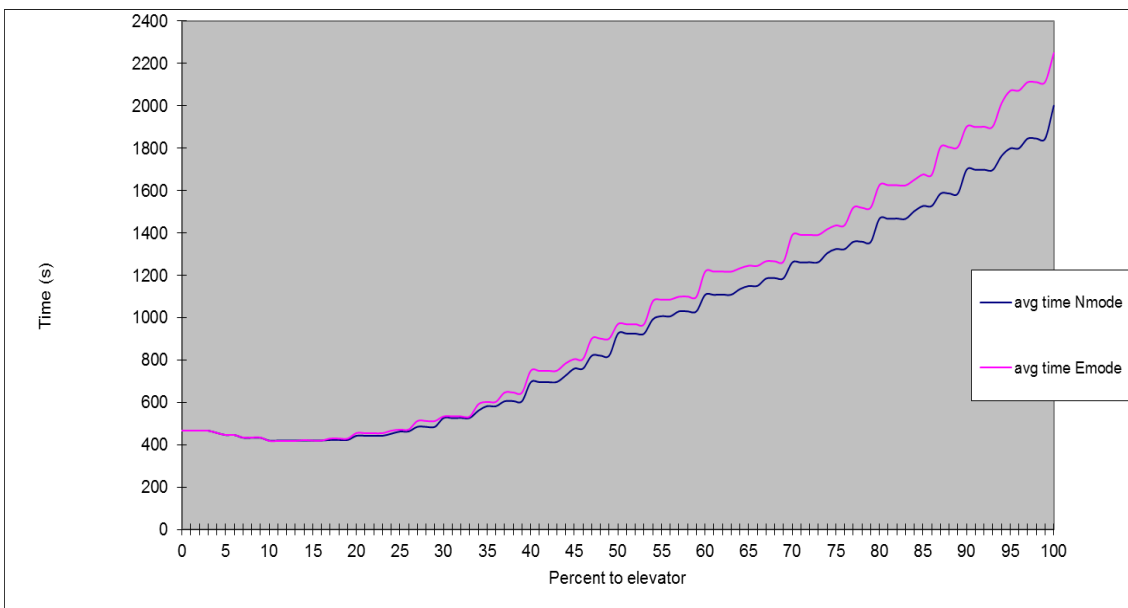


Figure 6.17: Average evacuation time per person in the extended building

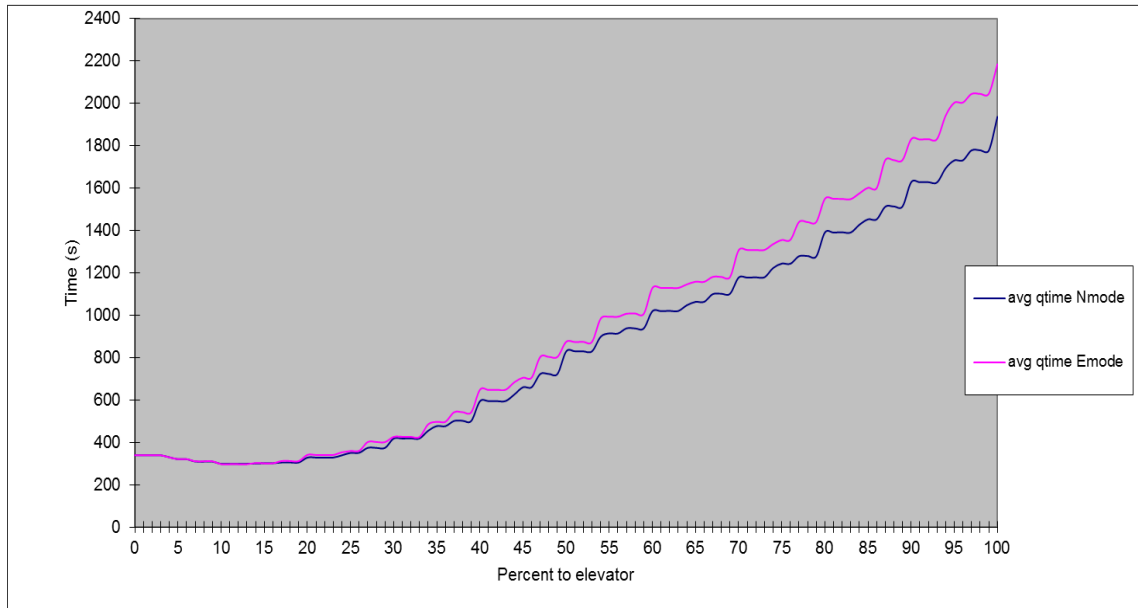


Figure 6.18: Average queue time per person in the extended building

6.4.1 Results summary of the extended building

The simulations of the extended medium building produced results similar to the medium building. Like in the medium building, Figure 6.16 show that there are significant times gains when 5% - 15% use the elevator, up to 120 seconds faster.

7 Conclusions

The tests performed in chapter 6 suggest that the architecture of a building and elevator properties play a big part on the evacuation times. The sudden jumps in the graphs are due to the fact that BEST rounds down the amount of people going to the elevators (this is described in detail in Chapter 4.4.5). Three of the buildings showed improvements in evacuation times whereas only two of them were significant enough to be of importance.

The emergency elevator mode always performed better than the regular elevator in respect to the total evacuation time. However in the emergency mode the average queue time and average time spent in a building was higher than in regular mode. This is because an elevator in emergency mode does not pickup people after when they called for the elevator but rather which floor the call was made from. This means that people on lower floors will always have to queue a long time since the elevator picks people up on the higher floors first.

The results from the large building suggest that there is not a clear connection between the average time spent in a building and the total evacuation time. It is hard to say exactly why the total time evacuation graph looks like it does, but one theory is that the layout of the building combined with the elevator properties are not suited well for evacuation with elevators.

From the data produced in the tests it is not possible to draw any solid conclusions about the benefits of elevator usage in smaller buildings. In all but one of the test cases elevators improve evacuation times when a low percentage of people use the elevators. As the percentage grows it has a negative impact on the evacuation. As mentioned, the data suggest that the architecture of a building and elevator properties play a major part in the evacuation times.

8 Further work

Many of the technical limitations in BEST were a result of time constraints. A future project would be to allow dynamic paths and decision making for people inside building. This would be interesting to simulate since people normally do not stay in a queue forever if there is a better alternative path. In BEST people stay in a queue as long as needed. This means that if a person on the top floor in a large building decides to use an elevator during an evacuation, the person is willing to wait a very long time for it to arrive. In a real life situation the person would probably look for another route out of the building after such a long period of waiting.

The gathered test results suggest that the design of a building has a great impact on the evacuation. The buildings in the test cases were made up and not based on any real data. It would be interesting to run simulations on real building data since the design of a building have such a great impact on the evacuations.

In future simulations elevators should be able to collaborate with each other in order to achieve optimal performance.

9 Terminology

BEST – Building Evacuation Simulation Tool, the tool created in this thesis

Discrete event simulation – A simulation of events ordered by time

Edge – The connection between two edges

Graph – A collection of a set of vertices and edges

GUI – Graphical User Interface

Level of service – The amount of people that can be served in a queue

Simulation – A way to study a system using a computer

Vertex – The central building unit of a graph

10 References

- [1] J. W. Fortune (2002). New Thoughts on the Use of Elevators for Emergency Evacuations of High-Rise Buildings, Elevator World, November 2002, pages 110-113.

- [2] Henri Hakonen (2003). Simulation of Building Traffic and Evacuation by Elevators.

- [3] M. Schreckenberg, S. D Sharma (2005) Pedestrian and Evacuation Dynamics

- [4] G.C Barney (2003) Elevator Traffic Handbook : Theory and Practice

- [5] Hatem El Shafie (2004) Simulation of pedestrian circulation in dining halls

- [6] Christopher A . Chung (2003). Simulation Modeling Handbook A Practical Approach. Chapter 1, Chapter 4.

- [7] Mark A. Weiss (2006). Data Structures and Problem Solving Using Java, Third Edition. Chapter 14.

- [8] Ulf Körner (2003). Kōteori. Chapter 2.