The Simulation of Evacuation

Connor Imrie

Matric No: S1233238

4th April 2014

1 Introduction

In this simulation, rooms with an exit centred on the left wall were created on-the-fly as required, described by the number of repeat experiments that were to be carried out. In each room a user-specified number of evacuees are based from a model evacuee defined in the problem statement, and were each given random positions initially, making sure to avoid collisions or intersections using a collision listener.

The initial model was set up as 100 evacuees in a 500x500unit room with each evacuee taking up a diameter of 15 units each. Rules for movement included moving towards the exit until a collision occurred, at which point the next time it moved it could randomly move to the sides or backwards to find another route, according to the probability set out by the Evacuee's patience and agility properties.

1.1 Aims

The simulation was created to investigate the relationship between population density, door width, and evacuee movement speed. This was done by fixing two of these potential variables while adjusting the other, to analyse how this factor varied the length that the evacuation took to complete.

2 Methodology and OO Design

2.1 Planning

The simulation code was written after extensive planning, using UML and generic notation and annotation on 'digital' paper. An emphasis was placed on building with scalability in mind rather than performance, in order to make the program maintainable and extensible in the future.

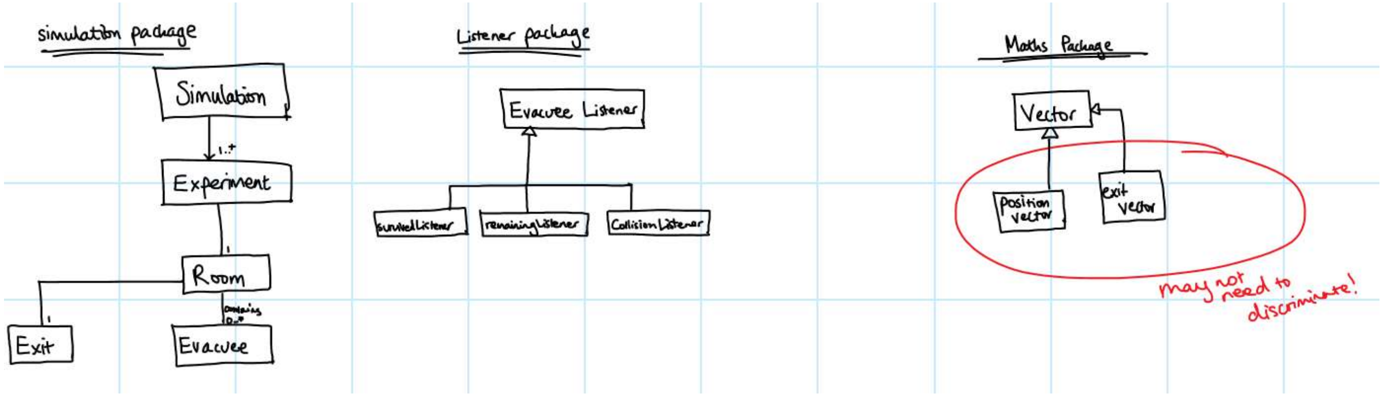


Figure 1 – Selection of UML created in the planning process

The notion to use packages to separate related Classes was implemented immediately and continued on in to the final code. Slight alterations to the names were made in order to prevent conflict with Java libraries already available.

2.2 Implementation and Methodology

Following extensive planning, the simulation was coded with maximum use of objects and classes. The classes were restricted to those that were essential, however sufficient that each class did not perform tasks that could be done by a specialized class elsewhere.

An example of this includes the CollisionListener class that I used to monitor the ArrayList of Evacuee objects in a Room. At each update of time or position, the CollisionListener is asked whether the change will cause a collision, and the CollisionListener performs the task and reports back while minimizing the Room Class interaction with collision detection logic.

This approach to Listeners (also used in 'SurvivedListener' to monitor escaped evacuees) was crucial in scaling the code later on, when adding more complex behaviour such as rotation of a vector or moving the maximum distance along the exit vector towards an evacuee it would otherwise collide with.

2.3 Class Descriptions and Functionality

2.3.1 <u>Simulation Package</u>

2.3.1.1 <u>Simulation</u>

The Simulation class acts as a main function to begin the simulation. It reads data from the user console, including the exit width and No. of Evacuees per room. It uses the Scanner library to read data from the user. For each of the 'sub' experiments (i.e. repetition of an experiment in order to gain an average & uncertainty in the average) a new Experiment object is created inside a loop counting over the number of 'sub' experiments. Inside the loop the class calls on a helper function called retrieveSubExpRunnable() and assigns the runnable task to a new (available) processor thread. This part of the simulation is multi-threaded in order to run experiments in tandem that only need to communicate at the end in order to report an average. Switching to multi-threaded processing showed significant gains in performance, especially relating to collision-heavy experiments.

The class uses helper functions to calculate the average evacuation time of these experiments, and the error on that average. It then writes it to a file, ready to be imported in to Excel or similar for graphing.

2.3.1.2 <u>Experiment</u>

The Experiment class handles much of the timer-based logic. Each Experiment object contains a room and updates the room (and thus evacuee positions) using the timers at particular time-steps.

Once the experiment has ended (checked using hasEnded() method, which returns true in the case that all evacuees have left safely) the Experiment object stops the timers.

The Experiment object is also responsible for gaining the latest clock time in the simulation in order to gauge the evacuation time. This is taken as the time the last evacuee left (which the evacuee monitors in its own class).

2.3.1.3 <u>Room</u>

The Room class has the purpose of managing all the evacuees designated for that room. This includes generating evacuees in random positions at the start of the simulation and also ensuring that the appropriate movement is made of the evacuees, for example generating random exit vectors so that the evacuees know where to head towards.

In order to aid collision detection the timeStep is split up in to intervals of size

$$(int)(speed\ x\ timeStep + 1)$$

This allows the detail of each time-step to be dynamically altered based on either the speed of the evacuee or the time-step given. As it is cast as an integer (int), 1 is added to the value to prevent it truncating to 0 (which would halt the simulation). This is an unlikely event but one that must be considered.

Significantly, the Room class distinguishes between the Math.random() and the Random classes. In particular Gaussian Random Floating Numbers are generated from the Random Class for use in calculating the exit vector at each interval, since even though Gaussian is weighted, it is generally more uniform than Math.random(), and used in its place to increase realism in the movements of the evacuees.

In addition to this the class also shuffles the Evacuees in their list in order to randomise which evacuee gets to move at any given time.

2.3.1.4 <u>Evacuee</u>

The main role of the Evacuee class is to represent an evacuee, with methods to set/update the position based on the time-step or condition given to the class. The position of the evacuee is taken as the centre of the circle representing the Evacuee. The drawing by mySwing package takes the differences in how java draws circle coordinates in to account (from the top left corner instead of the centre).The evacuee also keeps track of the internal time of the simulation, where the last evacuee to leave the simulation is taken as the clock for the experiment.

2.3.1.5 Exit
The Exit class is very simplistic and serves to represent an exit in size and position only.

2.3.1.6 BoolExpPair and BoolEvacPair
These classes are 'helper' classes and are used to facilitate returning more than one item. They are both comprised of a pair of different types of information – Namely a Boolean, and Experiment or Evacuee respectively.

2.3.2 mySwing Package

2.3.2.1 Frame
The Frame class serves to extend the JFrame class from the Swing Library, taking values such as size, visibility and title relating to the experiment it will display within it.

2.3.2.2 RoomPanel
This handles the painting of the GUI, drawing the position of the evacuees and the exit, and handling the disappearance of them as they exit the room.

2.3.3 myMath Package

2.3.3.1 Vector
The Vector class manages all manipulations related to vectors. It handles standard tasks such as rotation of a vector, addition, subtraction of a vector and multiplication by a scalar (useful in calculating the vector with the largest magnitude between two evacuees).

2.3.4 Listener Package

2.3.4.1 EvacueeListener
This is an ***abstract*** class developed for extension by the CollisionListener and ExitListener classes.

2.3.4.2 CollisionListener
CollisionListener handles all collision through two methods contained within. This is through checkFutureCollision(Vector trialPosition, Evacuee evacuee) which returns a BoolEvacPair, and also checkWallCollision() with the same parameters.

In the former, an arbitrary vector representing where an evacuee wants to move to is given alongside that evacuee. The CollisionListener then checks to ensure that the circles do not touch or intersect with any other evacuees in the room.

In the latter, the same principle is applied but to the boundaries of the room instead.

2.4 ASSUMPTIONS

2.4.1 The Exit remains on the left wall
This assumption is brought upon by the lack of detail in the problem statement concerning it. The exit can be moved along the wall and changed in size, however it is assumed that any detail of an evacuation will be seen equally well from the left wall as much as the other walls would represent.

2.4.2 <u>The specified requirement to allow the creation of a new room layout without recompiling</u>

The assumption regarding this is that the program must ask the user if they want to create a new room once the simulation is over, allowing more measurements to be taken without stopping the program at all.

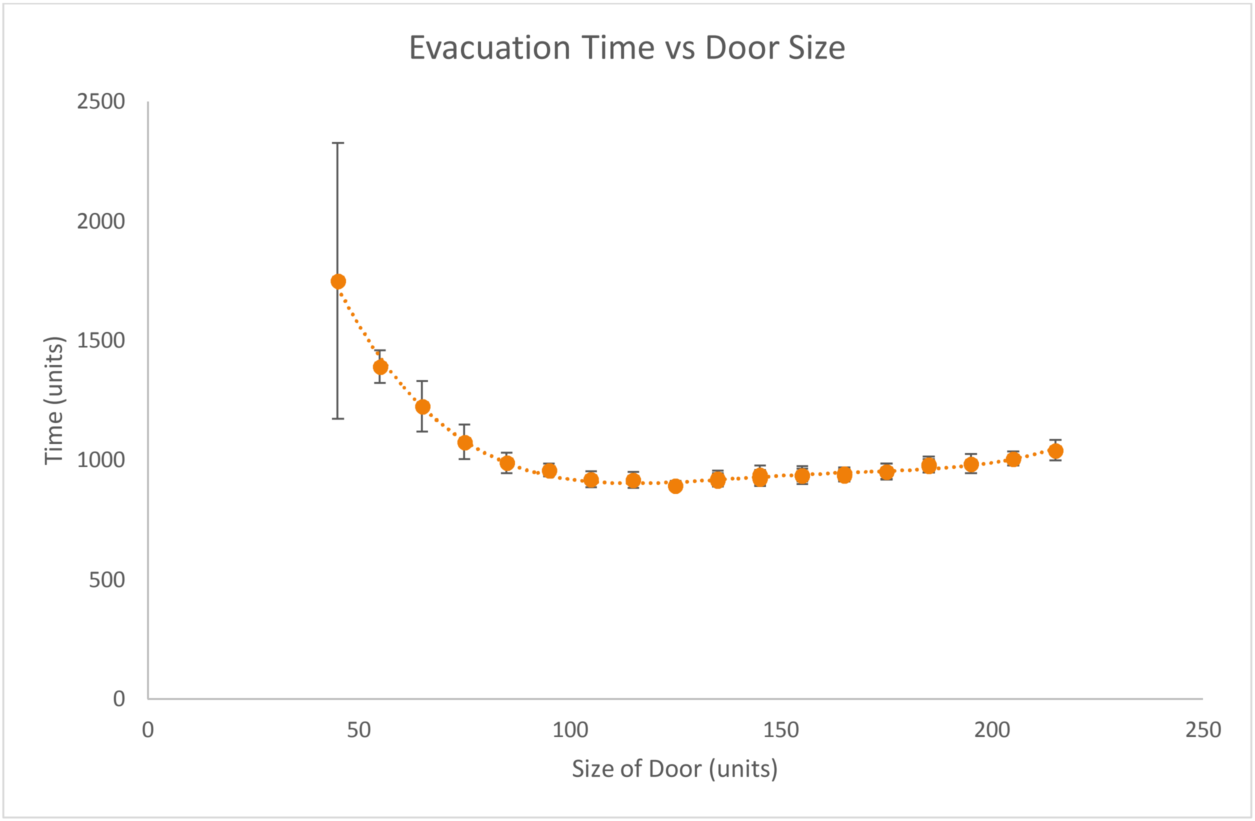
3 Results

3.1 Question 1

(a) How does the size of the emergency exit affect the length of the evacuation? Plot a graph of door size versus evacuation duration.

The size of the exit for constant population density (same number of evacuees for each experiment trial) and non-varying speed resulted in an exponential decay curve as seen below.

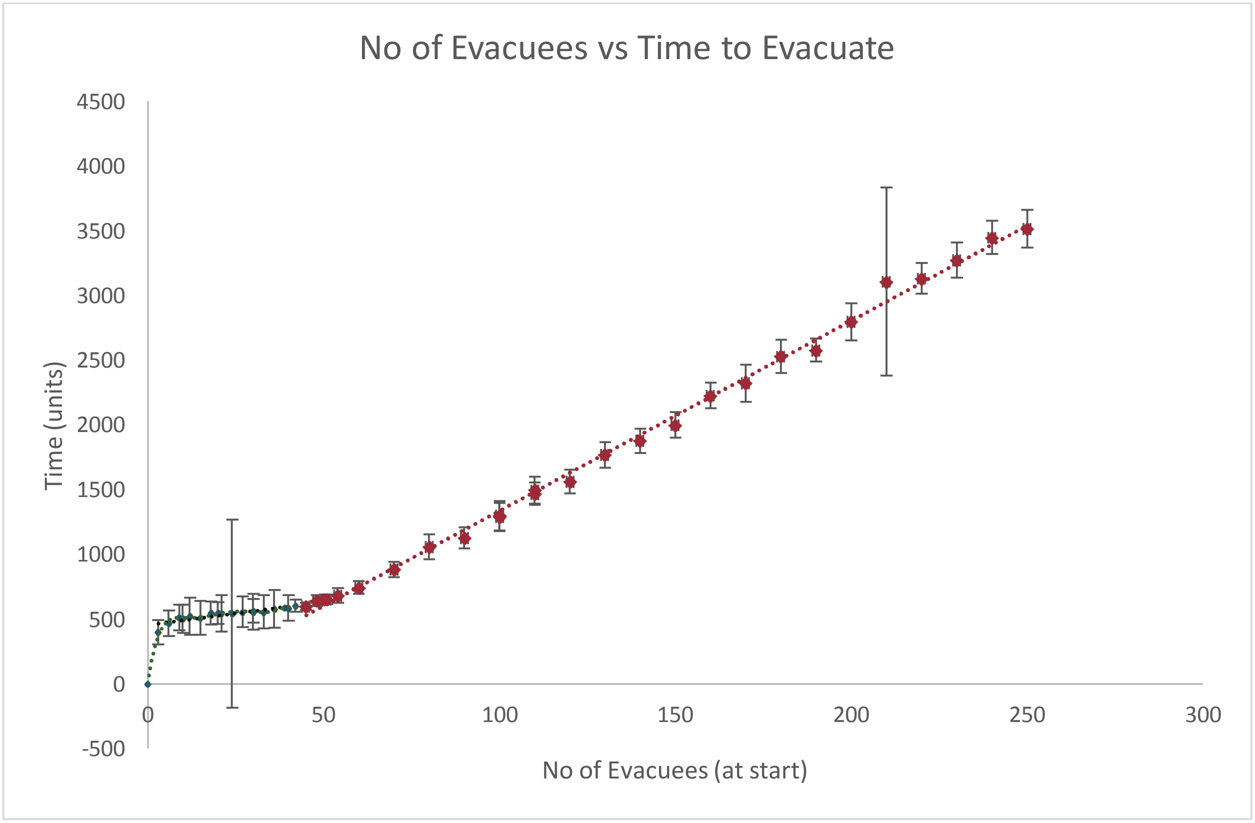
This is to be expected since for small doors, jamming occurs where evacuees simply cannot move since most of them are surrounded. A small door is a chokepoint that causes severe 'traffic congestion' for the escaping evacuees.



Graph 1- Evacuation Time vs Door Size showing exponential decay curve (data points are averaged result over 20 simulations)

(b) How does the number of evacuees affect the evacuation duration?

The number of evacuees has a much less significant effect on the evacuation duration. For small numbers of evacuees a plateau-like flat relationship is seen. Since I noticed a slightly different gradient, measurements were taken at many different small values for 'No of Evacuees' and also repeated over 25 times to ensure a good average. As the number of evacuees exceeds approximately 45 in the 500x500 unit room, the relation is much more significant and a separate, steeper gradient is seen. It must be noted that as the No of Evacuees tends towards 0, the evacuation time theoretically tends to 0. This, however, is limited by the plateau and is likely to be limited to this plateau line due to the discrete and finite speed that the evacuees can move at. Some results were taken again and are visible on the graph, in order to ensure the best fit line was found.

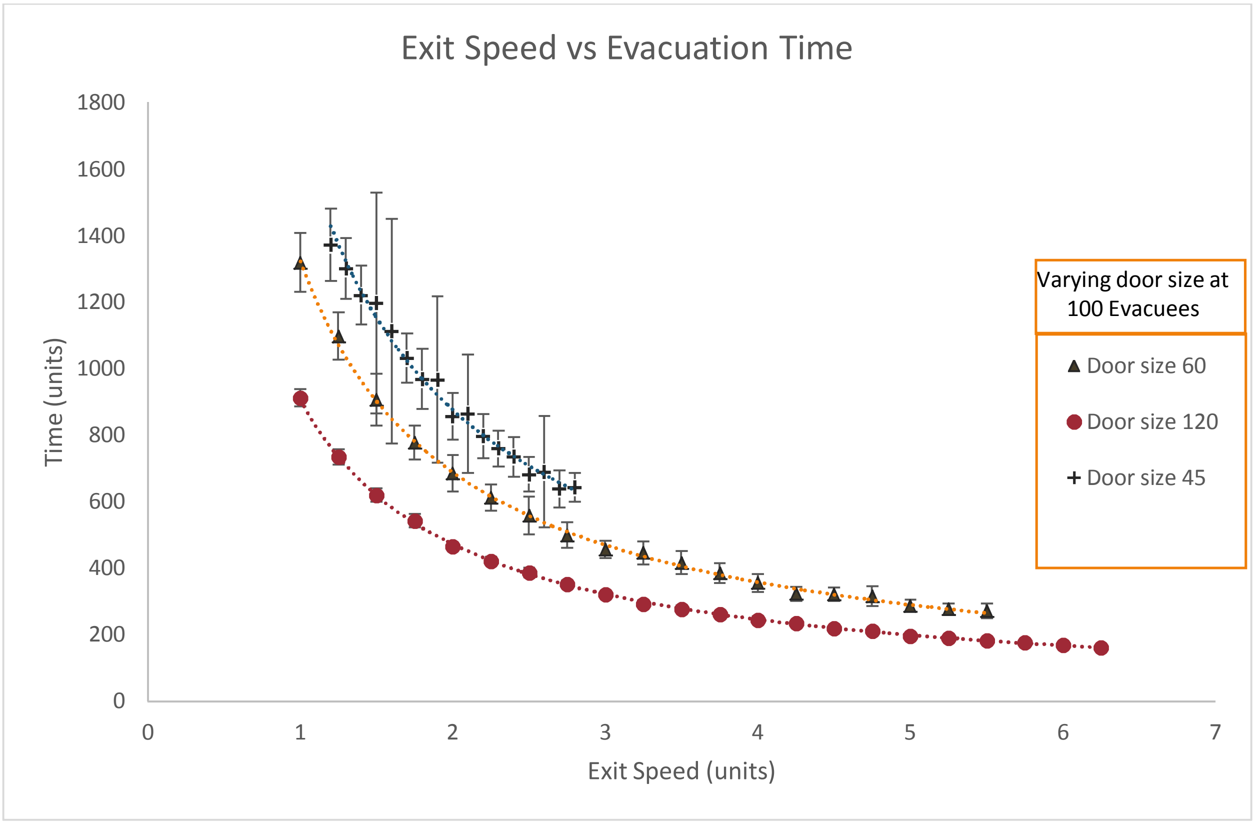


Graph 2 - No. Of Evacuees vs Time - piece-wise plot when door size = 60 units and exit speed = 1.0 units (data points are average results over ~25 simulations)

3.2 Question 2

(a) Should we walk out or run?

The speed of evacuation in this simulation was typically shortened by increasing the exit speed. This means that running is the preferred way to exit the room. When high population density (i.e. high number of evacuees) were seen in the room, it was often regarded as better to walk. Unfortunately trying to collect data for these events was difficult in that computation time was so large it was not possible to wait for the computer to calculate it.



Graph 3 - Evacuation time vs. Exit speed – Clear exponential decay for time decreasing as exit speed increases. Graph omits data for high density populations (only includes 100 Evacuee data sets). Data points are averaged results from 25 simulations each, and 15 averaged simulations for 'Door size 45'

4 Discussion

4.1 Question 1 – Evacuation Time vs. Door Size

The result that the evacuation time drastically drops upon small increases in door size is to be expected from day-to-day experience. What is difficult to explain though is the seemingly unnatural behaviour as the door size increases to a very large value. It is likely the slight increase in time is caused by the (unfortunate) lack of true randomness from using a random number generator, and that each point on the exit was not able to be modelled to be equally likely for the evacuees to head towards. Better random number generators could have been used (like the SecureRandom() type in the Random() Class), however much for the reason these are used in cryptography, the randomness requires a large amount of computation power for increases in randomness that may not be seen in the simulation. The computation time could not have been allowed to increase further since it could often take an hour or two to get data points as it stands.

4.2 QUESTION 1 – EVACUATION TIME VS. NO OF EVACUEES

It can be seen that the evacuation time increases linearly with number of evacuees. What is interesting is that there seems to be a plateau for No. Of Evacuees < 45 in the simulation. This suggests that there is sufficient space between the evacuees that ***collision is unlikely*** and ***the time is only dependent on their ideal exit speed***. Note that up until No. Of Evacuees approx. 45 (a critical point), we see that the evacuees take around 500 units to get to the exit. This makes sense as it's likely one of the evacuees is near the other side of the room, and will take 500 time steps at speed 1.0 units per time step to reach the exit last (which is the final clock).

4.3 QUESTION 2 – SHOULD WE WALK OUT OR RUN?

From the graph and data it is obvious that in the conditions checked, running is the best policy. This is in relatively low-density crowds with larger doors than normal, so does not necessarily contradict popular belief that walking is better. As the door size got larger, exit speed was the dominating factor in reducing the evacuation time, so again for large doors and low population density, running is best.

5 CONCLUSIONS

Overall the simulation and experiments were a success. Many key characteristics of an evacuation or mass number of people moving through an exit were seen. This included the fact that the door size had the most significant impact on the evacuation time, and the time decayed exponentially as door size increased.

When analysing the impact that the number of evacuees in the room had on the time, it was found that the number of evacuees could be increased without much consequence up to a critical point, at which the time to complete evacuation suffered with a positive linear gradient as the number of evacuees increased.

In contradiction to popular belief and evacuation drills, the simulation indicated that it is indeed better to run rather than walk to the exit, although in high density situations it was observed to be beneficial to move slowly, although simulation computation time prevented data collection.