Department of Computing Science
Lilybank Gardens,
Glasgow, G12 8QQ.

University of Glasgow

PROJECT REPORT
LEVEL 4, JOHN URQUHART
FERGUSON, 2005/2006

P.U.R.G.E.S. (Public Underground Railway General Evacuation Simulator)

by

John Urquhart Ferguson

Class (CS4H)
Session 2005/2006

Abstract

The cost of running evacuation drills in public buildings is often very prohibitive for companies and organisations, both in time and money. This tends to mean that evacuation drills are run as few times as possible, making the testing of evacuation methods difficult. There are also some types of evacuation that cannot be drilled both safely and accurately (such as that of a large fire). It is also an important point that an evacuation drill cannot be run on a building until it is built. An evacuation simulator is a possible solution to these problems. The aim of this project is to produce a three dimensions visualised evacuation simulator of the Hillhead Subway station in Glasgow, UK. This project has part of its roots in research and part in implementation. The research side focussing on crowd patterns and psychology in order to best determine how people will react in an evacuation situation. The implementation side of the project involving the modelling of the Subway station in 3D and simulating an evacuation on this model. This will draw on the algorithms from the research component of the project to produce a realistic evacuation simulation.

Acknowledgements

A great thanks to Professor Chris W. Johnson for his invaluable help and guidance during this project. Thanks also to the students who were involved in the previous evacuation simulator projects at the University of Glasgow that this one builds upon. In particular, to Graham McKinlay and Jody Johnston for kindly discussing some of the aspects of their projects with me. Thank you, as well, to the developers of the Blender 3D modeler for making such a great product and for releasing it as Open Source. Johannes Raida was good enough to release his DXF loader for free use which saved me a great amount of time and for that I thank him. And thank you to Sun Microsystems for developing the Java Programming Language and for releasing the Java SDK and Java3D plug-in for free use.

A special thanks must also go to Strathclyde Passenger Transport for their invaluable support during this project. In particular, to Chris Thirlwall and Neil Gatenby for providing information about the layout of Hillhead Underground station. And a last special thanks to Liz Parkes, the Commercial and Production Manager at SPT. Her help with validation, as well as giving guidance and support, contributed greatly to the success of this project.

Contents

Chapter 1

Introduction

Simulators are a means of mimicking an environment for the purposes of research and education. This often includes the 'intelligent' prediction of outcomes in these environments based on variables in the simulation. Using these test conditions to approximate actual operational conditions means that we can use simulators to approximate actual operational outcomes. This ability often leads to simulators being used as a cheaper means of testing and learning about environments[1]. Naturally, the more accurate a simulator is, the more useful it is to researchers, but it can also be used as a rough guide to show where actual testing in the real environment is required. It can sometimes be more useful and cost effective to develop a simulation with only approximated accuracy in order to reduce the amount of real environment testing required, or to make the real environment testing more productive. It should also be noted that people do not behave the same way in drills as they would in real life. This is a well documented phenomenon called Evaluation Apprehension[2], which shows that people behave different when they know they are taking part in a test or experiment. In this sense, an accurate simulator might actually be more effective than a drill in showing what would happen in a real incident.

Of course the problem with simulators of a real world environment is that they can never take into account every possible variation that can occur in that environment. However, they do allow for a large scale testing of many possible variables with greatly reduced expense. Often, in real world environments, it is simply impractical to test many of the variables. Simulations allow us to bypass this problem. There are, however, other significant problems. For instance, simulators often require the development of specialised models to run the simulators on, which costs more time and money. Also, having a different model type from that of the architect means that using evacuation simulators to test buildings that have not yet been constructed becomes difficult without some kind of translator program.

However, there is a definite place for evacuation simulators. The aim of this project is to construct such a simulator for the Hillhead Subway station in Glasgow, UK. The simulation will show a pictorial depiction of the evacuation so that it is easier to see the results. The user of the software will be able to select the values of the variables to be tested in each simulation.

The reasons for choosing the Hillhead Subway station are important. Firstly, it is not a viable option for the operators of the station to run fire drills on the public. Due to the turnstile system used in the station for entry and exit, it would be hard to gauge who had and had not already paid to be in the subway when letting people back in (the tickets would already have been used). Therefore, the drills have to be run at night when the station is closed using hired people to pretend to evacuate. This is expensive due to the additional time that must be paid to employees, and the cost of hiring the evacuees. It often involves bringing in consultants from the Fire Department, which costs more money still. Consequently, this type of drill is not employed very often. Unfortunately, it is also the only drill currently open to the station operators. Due to the cost and relatively low frequency of these drills, it is important that the most is made of them. A simulator could help show which areas need to be fully tested in these drills.

Although this shows that evacuation simulators can be useful, there is not much precedent for them. Most safety based simulations are concerned with fire behaviour or structural integrity of buildings. The American agency, NIST (the National Institute of Standards and Technology), has in the past dealt with reports on Fires. These reports have used simulators to determine what happened in fires, but a simulator for crowd behaviour has never been produced by the agency. A particular case of interest was the fire at the Station Nightclub in Rhode Island[3]. The fire began from pyrotechnics used by a band on the stage. The walls and ceiling of the small stage were lined with polyurethane foam which is highly flammable. The fire ignited the foam and spread quickly along the ceiling over the dance floor. Within five minutes, fire could be seen outside, breaking through a portion of the roof. However, it was the crowding at the main entrance preventing egress from the nightclub that caused the greatest problem, and one hundred people lost their lives. Had an evacuation simulator been produced, it might have showed how badly laid out the club was for egress. The report showed that the bad layout of emergency exits was a major contributor to the number of deaths, but this is largely based on witness reports. No simulation tests were done to determine the reasons for the lack of egress. This shows that even though there has not been much precedent for evacuation simulators, this does not mean that there is no place for them.

The Department of Computer Science at the University of Glasgow has done quite a lot of work on evacuation simulators, and a Professor in that department, Chris W. Johnson, has written several papers on the subject[4]. The various simulators produced (of which Purges is one) have tried to concentrate on building an evacuation simulator for a specific environment. These have included University buildings, hospitals, ships and football stadiums. All of these projects have taken different approaches to solve the problems of building an evacuation simulator and some of these methods were studied when deciding on how best to approach the Purges simulator. The early simulators had very simple crowd movement and, although a 3D display was used, were essentially based on two dimensional planes. Each new simulator has become more advanced and added its own unique features. There are, in fact, several features in previous University of Glasgow evacuation simulators that were not implemented in Purges due to time constraints and a different focus for the project.

There is another large evacuation simulator project at the University of Greenwich called 'Exodus'[5]. Exodus is similar to the Glasgow simulators but has a greater emphasis on environmental hazards such as toxicity. Exodus has been used to simulate evacuations from buildings, ships and aircraft. A railway version of exodus is also currently being developed.

The work done on simulators in the last few years has shown that there is a genuine reason for continuing development in the area. Improvement in crowd behaviour will most likely be the focus, possibly introducing techniques from Artificial Intelligence and Agent based systems. Further integration with existing tools could also be useful, especially for architects testing the safety of new structures.

The hopeful outcome of the Purges project will be an evacuation simulator with path finding and other 'intelligence' features in a three dimensional display. Since this is a simulator aimed at the Subway system of Glasgow, the project will try to be generic enough to be able to work with multiple underground stations. It should also be able to block paths and exits and the people should react accordingly.

The most fundamental way to achieve the 'intelligent' behaviour is to use a path searching algorithm. This means that each person will attempt to determine the shortest route to his goal before embarking on his or her journey. This at least makes people look like they have a purpose and are not just moving at random, but the full behavioural model will be discussed elsewhere in the report.

The rest of the report will cover the following topics: Chapter 2 will discuss the theoretical background for the project; Chapter 3 will show the research that was carried out in the project; Chapter 4 will detail the full requirements and specifications for the project; Chapter 5 will cover all the details of the system design; Chapter 6 will discuss the implementation of the system; Chapter 7 will cover the changes made to the design of the system during the implementation phase; Chapter 8 will show the results of the evaluation and validation from Strathclyde Passenger Transport and Chapter 9 will give a conclusion to the whole project and show the various successes, and failures, of the project.

Chapter 2

Background

In order to understand how the simulator works, it is necessary to understand some background theory. In this section, the background theory will be explained in order to give a good footing for reading the rest of the report. In particular, the area of crowd behaviour will be discussed as this is the foundation for trying to build realistic evacuation simulators. Then, more specifically, path finding algorithms, such as 'A*', will be discussed as these are the basis for the implementation methods used in the Purges simulator.

2.1 Crowd Behaviour

In an evacuation, the behaviour of people is quite interesting: People do not behave the same way in a group as they would if they were by themselves. In one way, this makes the task of simulating an evacuation much simpler as group behaviour is more predictable than that of individuals. However, it is still very complex and is dependent on many variables. Before it can be modelled, we must first recognise the peculiarities of crowd behaviour.

Crowd behaviour is type of group behaviour[2]. Most human life contains a large amount of group interaction as we are social animals. We often put reliance on others for our feelings of security and comfort. But groups can have a significant effect on our behaviour. The influence of groups can be split into three main areas: Social Facilitation; Social Loafing and Deindividuation.

2.1.1 Social Facilitation

Mankind has an odd inability to judge its limitations. This is shown by the effects of social facilitation. A simple example of this effect would be someone jogging alone without any people around. This isolated jogger will run as hard as she can and may note how tired she feels. But if we change the scene such that there is someone sitting on some grass, reading and facing the jogger, the jogger will indeed find the strength to run harder than before: Even though the reader is not looking at the jogger. Making the simple change of having the reader facing away from the jogger will produce no notable increase in performance from the jogger. Even though in both cases the reader was paying no attention to the jogger, the jogger feels a subconscious need to prove herself in front of someone who might see. The effect can also be seen if the jogger is jogging with friends. The time taken to finish the specified distance will be reduced and the jogger will not notice how tired she is to the same extent[2].

The mere presence of other people has affects on our behaviour. However, it is not as simple as trying harder when there are other people around. Let us take another example of giving two people an extremely difficult task to do at the same, but at no point implying that it is a competition. The time taken for each of these people to complete the task will be longer than had they been by themselves. It has also been shown that people who have a natural talent, such as being good at snooker, will perform better when amongst their friends. However, a person with little talent in snooker will perform worse than they are capable of when their friends are watching.

What is actually happening in social facilitation is that people's relative abilities are simply exaggerated in groups. But what does this mean for our simulator? Well, it is shown that people in a tightly packed group are more likely to be affected by the presence of others than if the group is spread out[2]. This would mean that the group affects of social facilitation will have a greater affect when the subway being evacuated is more crowded. This could mean that people will be more or less confident than normal, producing leaders and followers. People may take orders from strangers purely because they themselves are usually less confident than the person in question. In effect, the decisions of the group could come solely from the leaders of small segments of the evacuees.

A secondary effect of social facilitation is that of evaluation apprehension[6]. When people know they are taking part on some sort of test or evaluation, their innate tendencies will be amplified. This is actually an even greater exaggeration than would occur in a real evacuation and thus is not a true representation of how people would react. It has been suggested that evaluation apprehension is one of the main reasons behind social facilitation. However, the effect in real life is not as strong as that of an actual experiment.

2.1.2 Social Loafing

Social facilitation is a very passive affect of group interaction. It occurs merely from the presence of other people. Social loafing, on the other hand, occurs during a group effort of some kind. It has already been shown that people behave differently in groups than when alone, but we have only looked at one aspect of that so far. The interesting thing about social facilitation is that is can make people both better and worse performers than normal depending on a person's innate tendencies. However, social loafing is an affect that only causes negative behaviour[2].

When people pool their efforts together, we tend to think of it as being a better way to solve a task. Common expressions such as *many hands make light work* are often true, but this is not the whole story. A good example of this is a game of tug-of-war. In a laboratory test, subjects who believed they were competing as part of a group would not tug as hard as when they believed they were competing alone[2]. Although this is an interesting result, what is more surprising is that people do not realise there is any difference in their performance. It appears to be a completely subconscious behaviour.

So, while people see themselves as individuals in a group and feel they are not working together, their natural tendencies are amplified. But when people see themselves as individuals in a group and feel they are working together, they put in less effort but don't realise that this is what they are doing.

But once again, this result is not absolute. People will put in full effort while working in a group if they consider the task to be challenging, appealing or involving. In these situations, people might see their efforts as completely necessary or might see that they are more suited to a task than others and then work harder. The possibility of receiving a desired reward can also increase a person's performance.

Social loafing is also more likely to occur in people that are more individualistic as there is less group spirit. Women are statistically less prone to individualistic behaviour than men and so, in general, work better in groups than men do.

Ultimately, what this means is that many hands do not always make light work. In particular, large groups without individual accountability tend to produce these effects. Since this is exactly the kind of group that Purges will be simulating, this is a highly relevant area. It means that crowds might produce a *follow the leader* effect where people are less likely to take charge of their own well being and will follow the crowd instead.

2.1.3 Deindividuation

Another possible affect of crowds working towards a common goal is that of deindividuation. This refers to people losing their own individual identity in favour of the anonymity of a crowd member. The consequence of this is that people in a crowd will be willing to do things that they would not even consider if they were acting alone. As social facilitation exaggerates natural tendencies and social loafing diffuses responsibility, combining these two factors is what leads to deindividuation.

The resulting lack of inhibition can lead to startling results. An example of this was shown in 1967 at the University of Oklahoma[2]. Two hundred students had gathered to watch another student at the top of a tower on campus. The disturbed student was threatening to jump. Rather than attempt to stop the student, the crowd below began to chant, "Jump. Jump...". The student eventually jumped to his death. Putting yourself in the situation of one of those crowd members, it seems unthinkable that you or anyone else could do something so cruel as to coax someone to commit suicide. And yet, the incident did occur. Such dramatic character changes in people seem to be fuelled by the power of the group: Somehow producing a feeling of being caught up in something bigger than yourself.

It seems that in certain group situations, people will ignore normal restraint in favour of impulsive self-gratification and destructive social behaviour.

The reason for this change in attitude seems to stem from a person losing his or her identity when part of a large group. Actions are perceived as that of the group rather than that of themselves. There is a belief that if so many take part, then no retribution can occur: People feel there will be no individual accountability. The level of the effect seems proportional to the size of the mob as well. The larger the crowd, the less people feel like individuals.

The effect can also be seen with uniforms. For example, anthropological studies have shown that warrior cultures where the group identity was strong were also the ones most likely to torture the enemy survivors of their battles. However, the uniform effect can have the opposite effect. Since mankind associates the colour black with death and evil, dark uniforms produce a more vicious group. White uniforms on the other hand, usually associated with purity and kindness, produce altruistic groups. An example of each would be the black outfitted medieval executioner compared with the white nurse's uniform. Although uniforms are not directly related to the crowd behaviour of the Purges simulator, they do demonstrate deindividuation.

The real way that deindividuation is likely to affect a crowd of people evacuating a scene is that they will be less courteous than normal. The gentlemanly conduct of *women and children first* will probably disappear in favour of *every man for themselves.* People will not behave like their individualistic selves and will see only the group goal of getting out. Even though this may be interpreted as the individual action of getting yourself out, it is still not what a person would do if there were only a few people around.

A secondary effect of deindividuation is group polarisation. This refers to how risky or conservative people are with their decisions. Most people are fairly reserved with decisions concerning safety or personal security[2]. For instance, if there was a decision to be made on evacuating a building that involved jumping through a short distance of fire, a large section of the group might individually feel that it was a tough call. Assuming that everyone leaned a little more one way or the other, people would suddenly become much stronger in their opinions when the group began to discuss what to do. This would effectively produce two strong camps at polar ends of the decision. This is very much a group dynamic but it causes the original group to divide almost in a battle with one another.

In an evacuation this is more likely to affect the exits that people choose. People who are more or less the same distance from two exits will have an initial inclination and will commit to it more strongly when people around them to start to move towards a destination themselves.

2.2 'Groupthink'

A decision making group will often make clear decisions even though the whole group does not individually consider the decision to be the right one. This is the basis of groupthink[7]. It occurs in an established group of people who would rather sacrifice the opinions of any dissidents in the group in favour of group harmony. The concept of groupthink is not such a large consideration in the Purges simulator, but it is mentioned here for fullness. The only time this might occur in an evacuation situation would be if an existing group were all in the building at the same time. Although this would be common in a workplace evacuation, it is less likely to occur in a Subway station.

It should be noted as well that the minority is not without influence. A single individual who consistently presses towards a goal with an air of confidence will engender trust an inspire others to follow. Even if it doesn't change the group opinion, it will most likely cause them to re-evaluate their current goal[2].

2.3 Risk Perception

People are often inaccurate in their assessment of risks. This seems largely to do with a risk being within or outwith their control. For example, even though qualified experts say that nuclear power is safe, most people see it as dangerous. And while most experts agree that radon is a major hazard, most home owners are unconcerned about it. People feel that if a hazard is voluntarily accepted, then it is less of a hazard[8]. This is just an example to show that people make judgements without being entirely logical.

In fact, most people assess risks based on some kind of heuristic. This is generally not a very accurate way of making decisions but will often produce correct, or near correct, results. Examples of such heuristics include the availability heuristic and the asymmetry between gains and losses[2].

The availability heuristic is when people will think that possible effects that they can think of first are more likely to actually occur. This is an experience based heuristic such that events which occur more frequently in our experience are the ones we think of first. The very fact this is true means that, statistically speaking from our own experience, the event thought of first is probably going to be quite likely. In terms of an evacuation, this means that the potential hazards that are perceived first are also considered the ones most likely to occur.

The asymmetry between losses concerns another aspect of making decisions: That of gambling[8]. Essentially, when a person is gambling about a gain, people are less risky. People tend to pick a sure thing over a gamble to get something more. Conversely, however, people are more risky when gambling a loss. Let us say there are two possible gambling situations. In the first, if the gambler wins, then he gains something and loses nothing. However, if he loses, he will lose everything he has won so far. The second situation is that if the gambler wins, there will be a definite small loss, but it will only be small. Or, if he does not play, he might lose everything or he might win everything. Even though the second situation is the only one that has a guaranteed result, which is relatively favourable, it is the first choice that people are most likely to pick. In an evacuation situation, this is likely to produce the effect of being risky about evacuating, but very guarded about going back into any danger zones.

There are, of course, other ways in which people assess risks, but these ones are both pertinent to Purges and evacuations in general.

2.4 Agent Based Approaches

All the theories on crowd behaviour that have been discussed so far have looked at the crowd as an object to be read. This is a top-down view of groups. Trying to theorise what a crowd will do based on known behaviour patterns of similar crowds. But one could also argue for a bottom-up approach, where it is the behaviour of people that is modelled without any thought of a central controller in the simulator. Instead of simulating the people in a crowd as one unit, we could simulate the individual people but give them social interaction characteristics.

One way of approaching the problem from a bottom-up method would be using an agent based system[9]. Rather than say, “The crowd should behave like this, so update all the people to reflect that”, we can instead say for each individual person, “Take in the environment, see what others are doing and decided what to do based on that”.

An agent based method allows each person to control his own actions. But if every person has an isolated view of the environment, the crowd behaviour theories already discussed would be unlikely to show up. People make their decisions based on surroundings, and this is what agent based systems must also do if they are to be realistic.

An example of giving people social interactions is the joint field of influence technique[10]. In this system, every person has a certain amount of space surrounding them called their field of influence. This is dependent on factors such as a person’s field of vision and possibly their physical attributes (such as size). In addition, each person has a field of comfort surrounding them. If the field of influence of someone else overlaps with your field of comfort, you will be drawn towards them, forming a group. Since the group has more than one field of influence and field of comfort, its influence on others will be greater. Thus, greater groups will form. However, this does not deal with the problems of escaping the environment. Some person in the crowd will eventually have to take control of the crowd in some fashion by ignoring the influence fields and evacuating.

Moving the control of the crowd behaviour to the level of the people seems a valid way to approach the problem. However, agent based systems are generally much more complex to code and due to this complexity will be harder to ‘tweak’ to improve the realism of the crowd. It is for this reason that Purges uses a top-down solution.

Chapter 3

Research

The previous section of this report covered much of the background theory that was important in the creation of the Purges simulator. This section will attempt to show the research into applying these theories in practical applications. In particular, it will highlight how the theories could be applied to an evacuation simulator like Purges. It will not, however, say specifically how Purges was constructed as this will be covered in the design and implementation sections of this report.

This section will also talk about the specific research for the Purges simulator, rather than just general theory. This will involve safety information about the Glasgow Subway System and technical information about the Java3D system.

3.1 Path Finding Algorithms

The most basic thing that an evacuation simulator must do is evacuate people from an environment. We can simplify this even more by saying that we require to get some people from their current position to some goal. In order for a person to make his or her way towards a goal, he or she must find a path to their destination. Therefore, we need some method of finding paths.

Path finding algorithms can be split into two main categories. Those which find a best path and those which find a near best path. An example of each will be discussed below as they are both pertinent to the way that Purges works. It might seem odd to even consider using a near best path finding algorithm when there are methods of always finding a best path. However, there are reasons for this. Firstly, the near best algorithms tend to be faster by skimping on some part of the calculations that best path algorithms must deal with. Secondly, near best algorithms often use heuristics to get a solution. As shown in the background section of this report, humans also tend to use heuristics in their decision making, so this is a more accurate model of human behaviour. It should also be noted that people are not generally accurate enough to always make the correct decision. Thus, a near best algorithm is actually more accurate, in this instance, than an actual best algorithm.

The two algorithms that will be discussed are based on graph structures. A graph is a series of vertices and edges, with each edge connecting two vertices. A vertex represents a location that a person can be situated at and an edge represents a distance between two edges that can be travelled by a person. Although graphs can be directed (such that you can only travel along an edge in one specified direction), the graphs in Purges are undirected and a person can travel along edges in either direction. The edges in Purges are also weighted, such that it may be further to travel along one edge that another (even if the edges share a start and end vertex with each other). With this structure in place, we now need to find a path from a given start vertex to a given goal vertex. The distance of the path is measured as the total weight of all the edges on the path.

3.1.1 Dijkstra's Algorithm

Dijkstra's algorithm is a best path algorithm. It is guaranteed to find one of the best paths (there may be more than one path with the same distance) from a start vertex to a goal vertex. The basis for the algorithm is to assign a score for every vertex of the best path from the source vertex to that one.

Initially the value of the source vertex is set to zero and every other vertex is set to infinity. If we examine all the edges that the source vertex is connected to, we will also get all the vertices immediately reachable from the source. We assign the value of each vertex to be that of the edge weight leading to it, plus the value of the vertex the edge is from (in this instance, zero). However, in each instance, the current vertex value (initially infinite in this instance) is only changed if the new value is lower. We then apply these rules to each of the vertices reachable from the ones we have just examined, choosing the one with the lowest edge weight first. This is continued until all the vertices have been examined. At this stage, each vertex should have the best value possible. This value corresponds to the shortest path to that vertex from the source.

We now know how long the shortest path is, but we have yet to determine what that path is. To do this, we must backtrack from the destination to the source. To do this, take the destination value then make a note of all the values you get when you subtract all the edge weights of edges leading to the current node. At least one of these values will equal the value of the corresponding vertex for that edge. This vertex is the previous one on the shortest path (if there is more than one vertex that satisfies this criteria, it does not matter which one is picked. It must be on an equivalent shortest path). Backtracking in this manner and keeping a record of what vertices you have taken to get back to the source will give a shortest path in reverse order. Simply reverse this order and that is the shortest path.

3.1.2 The A* Algorithm

The A* path finding algorithm is similar to Dijkstra's but with the inclusion of a heuristic. The inclusion of a heuristic makes a huge difference to the algorithm (even if the outcome essentially looks the same). Dijkstra's algorithm is informed in its calculations, which means that it knows that the various conditions used in the algorithm will lead to a shortest path: This is done in a logical way with no guess work or assumptions. However, A* uses a heuristic and is essentially based on some kind of assumption.

The method of A* is similar to Dijkstra's algorithm: The vertices are assigned the shortest path length to get to themselves from the source and then backtracking takes place to find the path. A* tries to optimise this solution by reducing the number of vertices that are checked. Instead of checking the vertex with the lowest path weight first, a heuristic is used to find the best vertex to check. The heuristic is generally based on the distance travelled so far to get to a vertex as well the estimated distance from that vertex to the goal. This is essentially making sure that vertices which lead away from the goal are not checked. However, in a largely sectioned off environment like a maze, or a building with lots of corridor twists, this may not always lead to the best path as that path may require you to walk further away to get back quicker.

The heuristic used is essentially independent from A* itself. It merely needs to satisfy the condition:

```
f(x) = g(x) + h(x)
```

Where f(x) is the value produced by the heuristic function, g(x) is the value of the path weight up to the vertex, x, and h(x) is the score for vertex x produced by the heuristic function. It is the h(x) that can be satisfied by any one of several heuristics. The heuristic is usually some sort of weighting system to determine which vertex is likely to produce the best result. In the case of path finding, this will usually be the estimated distance from the vertex x to the goal vertex.

The heuristic that Purges uses is called the Manhattan heuristic. This heuristic is based on Manhattan distance, sometimes called 'taxi cab geometry'[11]. This is a way of estimating distance between points by calculating how far would need to be travelled to get from vertex x to the goal if they were both vertices on a rectangle and you could only travel along the edges. The premise is to assume that you cannot travel diagonally, but only in horizontal and vertical lines (much like a rook on a chess board). It doesn't actually matter if you travel along the edge of the rectangle or in a mixture of horizontal and vertical lines between the vertices, it should always be the same distance travelled (assuming you do not travel outside the boundaries of the rectangle or go in a direction away from the direction of the goal).

We know from Euclidean geometry[12] that the shortest distance between two points is a straight line and, therefore, the Manhattan method does not provide the shortest distance.

This is still valid, however, as the relative distances found for all vertices to the goal vertex will be in the same order of ascension. The benefit of this is that it is quicker for a computer to calculate Manhattan distance than Euclidean distance and in the Purges simulator the results of both methods will still return the same vertex.

The A* algorithm will, in most cases provide the shortest path and is, therefore, generally better to use than Dijkstra's algorithm as it is quicker. However, there is a particular case when A* will not provide the shortest path, but will probably produce a near best solution. This is when the direct route (i.e., the straight line) between two vertices is blocked or cannot be travelled. In this instance, the person must temporarily travel away from the goal vertex in order to get there in the shortest distance. In A*, however, the person will most likely travel towards the goal before attempting to manoeuvre around any blocked areas. In Dijkstra's algorithm, the person would turn away from the direct route immediately, which will be the shorter path. However, as previously discussed, the speed improvement of the program, as well as the fact that people do not generally know the shortest route and use heuristic solutions themselves, makes this solution optimum for an evacuation simulator.

3.2 Monte Carlo Systems

Not all problems can be solved with deterministic algorithms. There are many situations where an element of unpredictability is required in order to solve a problem. In the particular case of the Purges simulator, there are many elements that require an element of randomness. For instance, placing people at their start vertices; deciding how aggressive people will be; or allowing people to make choices without a strong correct answer.

The randomness is not just about making people more realistic, it could also be used to trigger events in the world such as a fire breaking out or a roof collapsing. Although, in Purges, all environmental factors are controlled by the user, a technique called Monte Carlo[13] is used to make people behave in a more nondeterministic way.

The Monte Carlo method used by Purges is based around generating random numbers between 1 and 100 (inclusive). Any choice by a person is then divided into choices and each choice given a percentage of likelihood. For instance, say there are two choices, A and B. We want there to be an 60% likelihood that A will be chosen and thus a 40% chance that B will be chosen. We now generate our random number and if it is less than 60 the person chooses A, otherwise the person chooses B.

We can expand this solution to any number of choices as well. If we have three choices, A, B and C, of equal likelihood, we can assign A the values 1 to 33, B the values 34 to 66 and C the values 67 to 100. Whichever range our random number falls into is the choice made.

Certain of the randomisation features of Purges are not based on this percentage method, such as placing people at their starting vertices. This is because all that is required is returning a random vertex, it is not a human choice that is being made. The simulator simply produces a random dispersal of the people. In actual fact, this is not a particularly probabilistic way of achieving the population dispersal. In a Subway station, it is likely that most people will be on the platforms, but in Purges it is just as likely that people will be at any point in the station. A more advanced simulator should really address this problem, but in most cases a non-probabilistic dispersal produces a realistic enough result.

3.3 3D Environments

So far the report has only discussed the theoretical aspects of the report such as moving people in an environment that is very abstract. In order to better show the simulation taking place, it would be better to show a visual reproduction of the environment. In Purges, this takes the form of a 3D model with people moving on it. This model can be manipulated by the user so that it can be viewed from different angles.

A number of possible approaches could have been taken to produce these environments. Past simulators have used the actual architectural computer files and directly imported them for use with the program. Others have modelled the environments by hand and imported them that way.

The benefit of the first way is that it allows for simulations of environments to be done very easily. Not only for existing environments but for environments that are still in the design process. An architect can quickly test his or her design for egress optimisation. The down side is that there are many types of architectural computer files and many variations of these types as well. To truly make this aspect a feature, all file types would need to be supported. In the case of Purges, it would only need to be the file types used by the architects of the Glasgow Subway system. However, the subway system in Glasgow predates the use of computers in the area of architecture. Any computer files for the stations were constructed after the stations were built. Therefore, removing the cost of creating computer files is not as much of an issue.

The second solution, making a new model specifically for the simulator, has the benefit of being optimised for the simulation in terms of visuals, speed and features. It also allows for construction of a specific file format for the Purges program. The down side of this solution is the time and man power cost of creating these program specific models. It is this second solution that Purges uses. This was partly to ensure flexibility with the Purges file format, but also because access to files provided by Strathclyde Passenger Transport were not available until quite late in development of the system.

The first thing required for this solution was to construct a model of a subway station to be used by Purges, and the second thing was to manipulate this model in Java. The rest of this section will attempt to show how this was achieved and how the solution came about.

3.3.1 Blender

The first step in creating the 3D environment is to model it. Although this is technically possible without a modeler by determining all the points of the vertices on the model, it is a painstaking process and would take far too long. Using a program with proper modelling capabilities makes the process much easier.

It was already known at this point that the Java programming language would be used for the simulator so it was necessary to find a modeler that could output to a format that Java would understand. A little research showed that most 3D modelers (in particular, those used in architecture) support the Drawing Exchange Format (DXF). So in order to make Purges as flexible as possible, a way to import DXF files was required. Rather than waste time coding this myself, an existing free import library for Java was used instead[14].

Since the common DXF format was now supported, this allowed for a lot more choice in deciding which 3D modeler to use. Cost was a big factor in this decision, as no commercial 3D modeler was available. The choice was to use a demo of a commercial program (which often allows only a limited period of use or limited functionality) or to use a free program. Using a free program seemed like the only long term solution so a comparison between some free modelling tools was made. The down side of using free programs is that there are not very many and most are inferior to the commercial products. Another surprising problem was that although some programs supported many file formats, most of the free programs did not support DXF.

The first tool looked at was Anim8or[15]. This is a fairly simple package, and would have made the creation of complex environments difficult. Also it did not support the DXF file format. For these reasons it was not used.

The second tool that was looked at was DXStudio[16]. The focus of this product is not so much on modelling but on the creation of interactive 3D content. It was not capable of doing what Purges required and, as such, was not chosen.

The third tool tested was Wings 3D[17]. This is a simple modeler and was simply not as capable of doing what was required as other packages. Also, this modeler did not support the DXF file format.

The best tool found was Blender. This is a full featured 3D modelling package and was capable of everything that Purges required. It also featured the best interface and had a lot of documentation so was easier to learn that the other packages. This is an extremely good package and is as powerful as some professional level tools of which I have tried demos. It was far superior to all the other packages and since it is open source under the General Public License this ensures that it will always be usable for free to make the models for the Purges simulator. It is, quite simply, the best free package available for 3D modelling and has a large community of users offering help, support, examples and documentation. A screenshot of the Blender interface can be seen in figure 3.1.

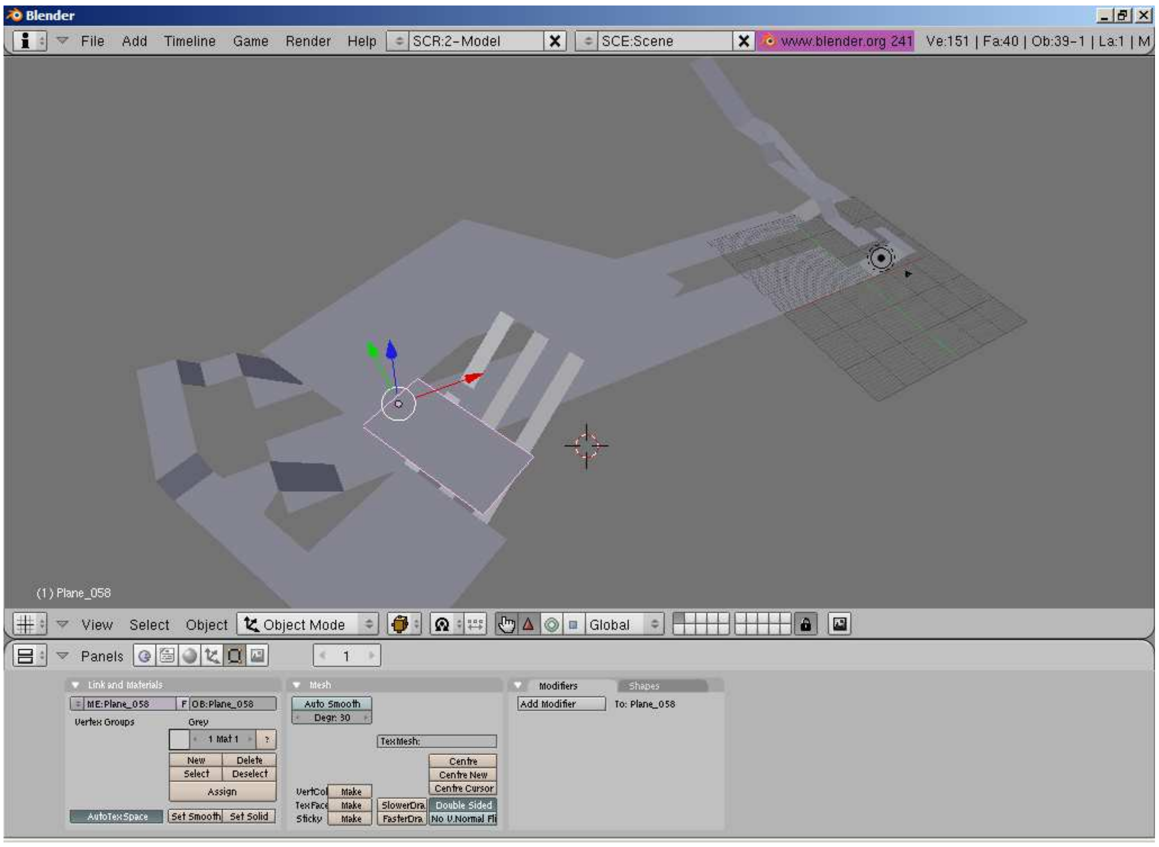

Blender
File Add Timeline Game Render Help SCR:2-Model SCE:Scene www.blender.org 241 Ve:151 | Fa:40 | Ob:39-1 | La:1 | M
(1) Plane_058
View Select Object Object Mode Global
Panels 1
Link and Materials
ME:Plane_058 F OB:Plane_058
Vertex Groups Grey
1 Mat 1
New Delete Select Deselect Assign
AutoTexSpace Set Smooth Set Solid
Mesh
Auto Smooth Degr: 30
TexMesh:
Centre Centre New Centre Cursor
VertCol Make TexFace Make Sticky Make
SlowerDraw FasterDraw Double Sided No V.Normal Flip
Modifiers Shapes
Add Modifier To: Plane_058

Figure 3.1: Blender Interface

Another tool was Povlab. This was a fairly full featured editor and was capable of doing everything required for the Purges program, but was not quite as feature rich or easy to use as Blender.

Blender is a mesh based modelling environment. This means it is based around the construction of polygons which are made up of vertices and edges. A cyclic collection of edges into a shape is a polygon and can be used to form a plane. The only problem with this system is creating curves. However, these can be approximated by using many planes. The more planes used, the more accurate the curve. This allows for the creation of most environments.

Even though blender is a very capable program and has a fairly intuitive interface, for someone who has not done much 3D modelling it still has a huge learning curve. Consequently, building the models used by Purges did not take very long once I had learned Blender, but learning Blender took longer than expected.

3.3.2 Java 3D API

A solution to displaying the 3D environments was required for the Purges simulator. There were a number of possibilities to examine. The first was to construct some sort of 3D engine.

This would have been very time consuming and was really out with the scope of the project. A second solution was to use an existing 3D engine. As with the 3D modeler, this would have to be a free to use engine. It was also seen as important that it worked on the major types of 3D acceleration hardware. The two main types are DirectX and OpenGL. These are actually libraries that programmers can use to tell the two types of hardware to carry out 3D manipulations. Most 3D Cards on computers now support both languages, but they are both independent. They are, however, the only real competitors left. Thus, supporting both of these means that most computer hardware will work.

Although there are many free 3D engines available, there are few that easily integrate with Java. There were a number of possibilities, such as Crystal Space and Cube, but most were more complicated than was required for the project. A notable point about the Cube engine is that it allows for real time editing of environments while the engine displays them. This is a fairly uncommon feature for a 3D engine and could potentially mean that parts of the environment could be built while the simulator was actually running. This would be a very quick way of seeing the effect of modifications on egress. Even more impressive is that Cube allows for multiple people to edit the environments in this fashion allowing for easy collaboration and testing of ideas. However, cube was fairly complex to use and would have been less easy to integrate with Java.

There is a relatively simple plug-in for Java called Java3D which has very good integration and works with both OpenGL and DirectX accelerated systems. This supplied all the functionality required by Purges and was relatively easy to install and use. The down side was that it currently does not come as part of the Java Runtime Environment and so most people do not have it installed. However, the installation procedure is very straightforward, so this was seen as a fair trade off for its relative ease of use.

Java3D uses a hierarchical content system to make 3D environments. This is a tree structure where the root of the tree is the universe and the branches and leaves represent groups and items, respectively, in the universe. An item is any single element in Java3D and items can be conglomerated into groups. These are called branch groups. A branch group can also be made up of any mixture of items and other branch groups, thus creating a hierarchical structure.

In order that the contents of the universe can be manipulated in the program, there is another special type of group called a transform group which specifies movements in Java3D during execution of the program. These behave in the same hierarchical way as branch groups except that all branch groups, transform groups and items which are contained in a transform group are affected by the particular transformation. For example, if we made a transform group directly after the root of the tree (i.e., the root of the universe) that had a rotation transformation associated with it, and everything else in the tree was contained in that transform group, then applying that transform would rotate everything in the world and also take into account any other connections and transforms in the tree.

Although it can take a little while getting used to this tree based creation of environments, it does allow for a great flexibility in creating environments in an object based way. For example, if we had decided to add the entire universe rotation late in the programming and were not using a tree based system, we would need to modify every item to apply the rotation, often involving complicated geometry rules to calculate the overall outcome of multiple transforms on an item.

There is only one quirk of the Java3D system that can prove to be annoying, which is that only branch groups can be added to the universe tree dynamically (after the program has started running). This does not prevent the programmer doing anything at run time that they could not do before the program is run, it just means that a clear plan of what needs to be done with the environment is made. For instance, since a transform group cannot be added during execution, all the transforms that are going to be applied must be thought out ahead of time. Each transform must have a transform group set up in the tree before any dynamic transforms take place.

Although Java3D does use hardware acceleration, the programmer does not use either OpenGL or DirectX commands in the source code. Java3D is essentially a wrapper for both these graphics libraries. The benefit of this is that a programmer only needs to write graphics code once and it will work on both types of accelerator hardware. The drawback is that adding a wrapper makes the program slower and this affects the speed of the 3D graphics. The program is likely to be slower in Java in any case since Java programs are run on a virtual machine and this is likely to have a greater impact than the wrapper. A more important drawback is that, in order to support both libraries, Java3D has to appeal to the lowest common denominator and does not use the advanced features that might be specific to a particular library. However, this is an understandable cost for the benefits of not coding the graphics code in two separate libraries.

Although Java3D seems complicated if the programmer has little graphics programming experience, it is a higher level system that using the two main industry graphics libraries (i.e., DirectX and OpenGL) directly. In this respect it was certainly the better choice and is also higher level than many of the other free to use graphics engines that were found.

3.4 Safety Research

Major evacuations of subway stations have taken place in the last 20 years including the Kings Cross fire in 1987 and the London Bombings in 2005. In each case, a quick egress of the passengers was essential to saving lives. Both incidents showed that blocking the regular paths of egress can have major impacts on the movement of the crowd. The safety staff must be able to take these possible changes into consideration ahead of time so that they are already prepared if those events occur. After the Kings Cross fire, the Fire Precautions (Sub-surface Railway Station) Regulations were written to ensure that evacuation and other safety procedures were in place for all subway stations in the United Kingdom.

Thus, the procedures for staff in the event of an evacuation are very clear and detailed.

Strathclyde Passenger Transport (SPT) were kind enough to allow me to see their procedures in the event of an evacuation in order to help me develop a more accurate simulator for the Hillhead subway station.

In the event of a fire, there could potentially be 800 people attempting egress from the Hillhead subway station. SPT have a detailed plan for the staff members in the event of such situations. However, they do note that the actual situation should dictate the actions of staff. For instance, if something happens that has not been thought out before the incident, then the staff must use their training to decide on the best course of action, even though specific details of what to do will not be known to them.

The first of the specified actions that the staff must undertake is to initiate the evacuation process. This is the responsibility of the Duty Station Manager. However, other suitably trained personnel can assume this responsibility if the Duty Station Manager is temporarily unavailable.

The next task for the staff is to organise the evacuation. An automated announcement system is used to inform the passengers to evacuate by their nearest exit. The Duty Station Manager must maintain responsibility for ensuring that all passengers and staff are safely evacuated, using any other SPT staff available at the time. Those staff should make their way into the station, as safety permits, and direct people to their nearest exit as well as helping people with poor mobility to leave the scene. The Duty Station Manager should also maintain telephone contact with the System Control for as long as possible. Once all passengers and staff have evacuated, those staff members assisting with the evacuation should also vacate the premises, closing the station entrance doors to discourage any members of the public from entering the station. However, one door must be kept open to allow safety services to enter (if they have not already arrived). If more than one staff member is available, as many exits as possible should have a member of staff located at them. These staff members should maintain that one door at each exit stays open in case more people are trying to evacuate and should ensure that no one else enters the station.

This shows that the staff have a lot to do during the evacuation process and that much of their activities involves helping the egress of others. Because of this, they will not behave the same way as the rest of the occupants of the station and this should be shown in the Purges simulator.

It is also important to note that due to time and monetary costs, evacuation drills cannot be run excessively. Certainly, the multitude of variables of an evacuation cannot all be tested. There are some aspects of an evacuation that can never be drilled accurately, such as a large fire or flood. This research demonstrates how useful a simulator could be as a more cost effective way of simulating these large scale disasters.

3.5 A General Solution

From what has been discussed so far in the report, a simplistic solution to creating an evacuation simulator can already be constructed. The basic steps are shown below.

1. Build a 3D model of the environment in Blender
2. Construct a corresponding logical model of the environment to be used to make the graph structure
3. Import the blender model into Java3D
4. Make the graph structure that represents the environment from the logical model
5. Populate the graph with people
6. Create people on the Java3D tree that correspond to the people in the graph
7. While there are still people in the graph, for each person calculate a path to the exit, and move them to the first vertex in that path. If people are at the exit, remove them from the graph and the 3D display.

This is a very simplistic view of a solution, but the basic design is there. The actual solution used in Purges will build upon this system. So far only the basic requirements of a simulator program have been discussed. Before constructing a full design, a full picture of what is required must be constructed. The next section of the report will discuss in full what is required of the Purges program.

Chapter 4

Requirements

It is important when designing any system to make sure that the programmer knows what they are building. The best way to do this is to be clear about what the system must achieve. This section will not only discuss the requirements of the simulator program, as the absolute minimum requirements would not add any interesting or innovative features: This section will also discuss features that would be good to achieve. Some of these made it into Purges and some did not.

The chapter is divided into three sections. The first section shows the absolute minimum requirements that must be achieved. The second section will show additional features that would make the simulator more than the basic necessities. Finally, the last section will show features that would be nice to include but are really not required. These are simply nice features that really do not affect how good the simulator will be.

These categorisations are subjective on my part as a meeting with Strathclyde Passenger Transport was not possible until development had begun on Purges. However, I will attempt to justify my reasoning in the design section of the report.

4.1 Must Have

1. Ability to show an evacuation of people from a subway station
2. Be able to change the number of people in the simulation
3. A visual representation of the evacuation
4. Ability to manipulate the visual representation
5. Passably realistic behaviour of people

4.2 Should Have

1. The ability to support multiple environments (for running the program on different subway stations)
2. Ability to block parts of the simulator (to represent hazards)
3. Change speed of simulation

4.3 Could Have

1. A timer to show time of evacuation
2. Pause/Resume simulation
3. Restart Simulation
4. Staff members should be able to influence (help) people's egress
5. People should move differently on stairs, going through turn styles, across slippery surfaces, through smoke filled areas .etc
6. Randomly allow trains to enter station and drop off more passengers.

Chapter 5

Design

A basic design for the Purges simulator was shown in the research chapter of the report. This chapter will attempt to build upon this initial design and be more specific with details. However, technical details will still be relegated to the implementation section.

The design is broken into three main sections. A logical view, a physical view and a controller. The logical view refers to all the aspects of Purges that the user does not directly see. These are the abstract features like the graph structures and algorithms. The physical view refers to what is actually seen by the user such as the interface, the display of the environment and people and the interactions with the user. These two 'worlds' have no real knowledge of each other. Instead, part of the program works as a controller. It coordinates all the people's movements in both worlds and keeps the worlds in synchronisation. Choosing to use a central controller was a significant point in the design. As discussed in the background section, a different approach was to use agent based systems, but this was seen as an overly complex solution. The basic design of the system can be seen in figure 5.1.

5.1 Logical View

The logical view is based on a graph structure, as has already been discussed. The graph does not need to be a complicated structure and is just a way of grouping vertices together. The only function the graph will need to perform is path finding. The vertices which make up the graph will be more complicated.

A vertex will need to contain a list of all other vertices that it can reach. Since every vertex will have this feature, paths of vertices can be formed. This is, in effect, the graph representation. In order to map the vertex to a point in the Java3D model that people can be positioned at, each vertex must also contain a coordinate to represent such a point.

The last thing that must be contained in a vertex is a flag to say whether or not it is blocked. This could either be from a person being located at the vertex or the vertex

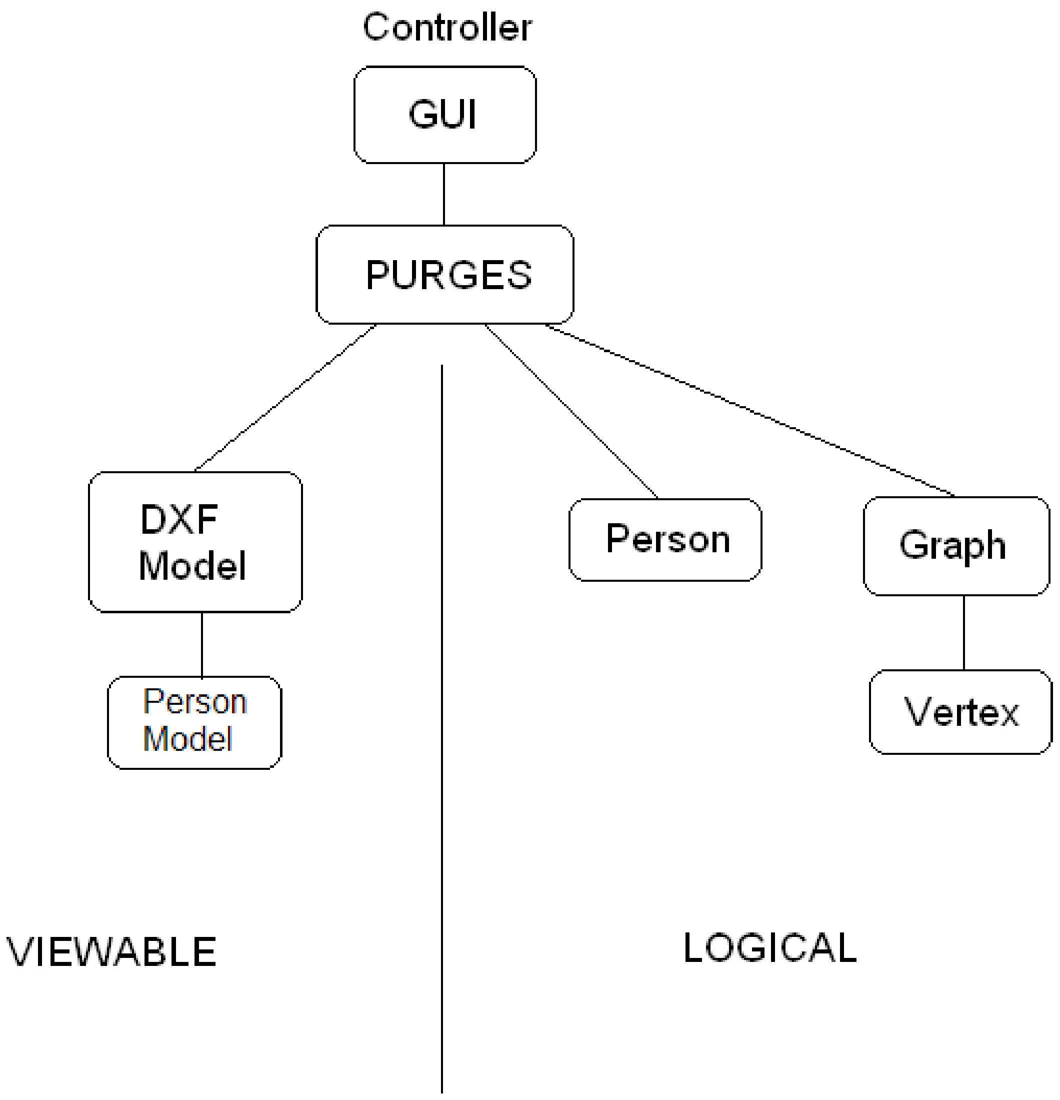


Figure 5.1: Basic System Design

being blocked to represent some kind of hazard. It could even represent a wall. It might seem pointless to catalogue a wall at all in the graph since a person can never stand in a wall. However, it could be possible that a wall be destroyed during an evacuation leading to a short cut between corridors which the people should be free to use. The scope of the simulator has no need to differentiate as to why a vertex is blocked. However, there are two special cases.

Certain vertices will need to be marked as possible goals that people can head for. But if the goal vertex is blocked because a person has moved onto it, the other people will believe the goal to be blocked and will attempt to find an alternative exit. This is obviously not very realistic, as a real person would know that the exit will soon become clear again. To combat this, there will need to be an override to the regular blocking flag which makes a vertex always appear unblocked. This way it can always be included in a path search.

Another special case will occur if the user wants to block a vertex. If a person is on the vertex when the user blocks it, nothing will actually change. The person will move out of the vertex, setting it to be free again and all the other people will then see the vertex as free. Thus, another override will be required to set vertices to be always blocked. Also, in the situation that a person is in a vertex as it is blocked, this should really be dealt with. Blocking vertices in the simulator is supposed to simulate some kind of impassable hazard. The scope of the project does not specify the type of hazard, so in order to keep as real a simulation as possible, such people should be removed from the simulator. In essence, this is simulating death. This may seem a little strong, but it is more realistic than the other options.

The first option is that we could allow the person to continue about their business, but if there was really a hazard, this seems unlikely. The second option is to not allow them to leave the vertex. This is simulating that the person has become incapacitated. Although this is also viable, it will look peculiar to the user that someone has just stopped moving. For the sake of clarity, people are killed if they are in a vertex when it is blocked. The ideal solution would be to use Monte Carlo techniques to randomise whether or not a person is incapacitated, dead or somehow escapes, but time constraints on the project did not allow for this to be completed.

It is important to note that marking a vertex as always blocked should take precedence over marking a vertex as always free. This is so that we can allow the user to block an exit which seems like a prudent thing for the user to be able to do.

The people in Purges should not be too complicated. Their behaviour will be dictated by the controller and A* so their actual representation will be relatively simple. A person will need to have a vertex associated with them to represent their current location in the graph. A person must be able to move as well and the move function will only need to change the current vertex of a person. Each person must also have a goal vertex associated with them to show where they are ultimately trying to get to. The current and goal vertices can then be used by A* in calculating the movement path for each person.

As discussed earlier in the report, Monte Carlo techniques should be used to demonstrate some of the random aspects of human behaviour. In order to give people unique personalities (or something close to it), each person should have a number associated with them between 1 and 100 (inclusive). This number can be used by the controller to make decisions for each person as well as deal with areas such as aggression. For instance, if two or more people wanted to move into the same vertex, the most aggressive would be the one to actually succeed. Again, this will be directed by the controller.

Since the simulator has to deal with multiple environments so that it can be used with different subway stations, some sort of file format will be necessary. The file format will need to contain information about both the 3D model and the logical model. The logical model will need to contain enough information to construct the graph. Some kind of loader will have to be written to deal with these Purges environment files. However, this should be a fairly simple parser. Details of which vertices start off blocked (collapsible walls for instance) must be included as well as indicating which vertices the user should be allowed to block. Rather than allow the user to block individual vertices, it would be more user friendly to allow the user to block groups of vertices (such as a stairwell or corridor). This will also need to come from the file format.

The file format will also need to make some indication of the maximum population that should be allowed in the environment. Obviously it would not be sensible to tell the simulator to put 1000 people in an environment that only supports 20. Having the simulator specify this is a convenient and economical solution. It saves time from the simulator having to determine the maximum size. It should also be noted that the actual environment could have a specific capacity limit for safety which might not equal the algorithmic population limit.

The last thing that should be specified in the file format is which of the vertices should be goal vertices. This cannot be achieved algorithmically so it is very important that this come from the file format.

When the simulation starts, the environment must be populated with people. In order to do this, some means of returning a random vertex from the graph will be necessary. By assigning everyone a random position to start off in, the capability to test for unexpected eventualities is maintained. However, in an environment such as a subway station, there is a greater likelihood that the greatest portion of the population will be contained on the platform area. As true as this may be, time constraints will not allow for it to be implemented. The random dispersal method is still valid as it approximates the variability in the environment and could potentially cover all possibilities.

It should also be noted that the Purges system will use a version of A* that does not require the trace back step. To avoid this step, each vertex will contain a current best path from the start vertex to itself and a number value representing the travel cost (i.e., distance) of that path. The A* algorithm will keep these elements up to date as it makes it way through the algorithm. The only difference in this version is that in addition to keeping track of the best travelling cost for each vertex, the best path is also tracked. This means that as soon as the goal vertex is found, the best path will already be stored in the vertex that leads to the goal. The first element on the path is the one that will be taken. Although this makes the A* algorithm more complicated to implement, it will increase the speed of the program.

5.2 Physical View

The physical representation of the world will be purely superficial, updated by the controller with information collected from the logical view. The physical view contains no concept of a graph or vertices, or boundaries of any kind. It only deals with coordinates. So, if the controller tells a person that his or her current location is somewhere 200 feet above the model flying around, this is where the person will go. It does not matter that no vertex exists for such a space or that it is impossible for a person to be flying around in this manner. The physical view has no algorithms to work out where people should be at any one time, it simply does what it is told.

This level of abstraction is quite important to the design as it allows for a lot of flexibility for future development with Purges. The logical view could be kept if a different method of displaying Purges was later used (such as switching to a different 3D engine). Conversely, improvements made to the logical view will generally not have an impact on the physical view as the only information passed is coordinates. Keeping this communication simple allows for a more extensible design.

The file format being used by Purges will, in addition to the logical view information, also need to store the physical view information. This will certainly involve embedding the DXF 3D model file.

One of the problems with having a simulator that can display a variety of environments is that each environment will have to be resized in order that it can be fully displayed to the user. The third party DXF loader that will be used in Purges will actually take care of this problem itself. However, the people created in purges will have to be scaled as well to match this size. The easiest solution to this problem is to have the file format supply a scaling percentile for the people. The controller can them resize people by the specified percentage of the original.

The second issue of this resizing is that the global coordinate system does not get resized. Although this may not initially seem like a problem, it means that the designer of an environment will have to commit to the 3D model before doing the vertex cataloguing of the logical model. If any change is made to the DXF model that affects what the DXF loader considers to be the environment's centre, the vertex coordinates will not be accurate anymore. Ideally, a system in which Purges could automatically generate the vertex information and the user could just specify which parts were goals, or user blockable, would be more suitable. However, time constraints mean that this will not be attempted.

Making the people should be a simple case of creating a model in Java3D that can easily be scaled. It is important that the people appear in the Java3D tree lower than the imported DXF model. This is because there will be a rotation feature which will allow the user to view the model from different angles. As long as the people are created lower in the Java3D tree than the DXF model, any transformations on the DXF model will also affect the people. Thus, the rotation feature will correctly position the people relative to the DXF model when the environment is rotated.

The third party loader for the DXF model will position the model such that it is viewed directly from above. This may not be suitable for optimal viewing. However, it is relatively easy to rotate the environment by applying a transform to it. As discussed above, this will also correctly transform the people as long as they are correctly positioned in the Java3D tree. The amount to rotate the environment by, which I shall call the camera offset, should also come from the Purges environment file. The rotation will come as three separate axes rotations. If the rotation feature on the Purges interface also works by applying three axes rotations, then the camera offset values must be added to these each time. Otherwise the user applied rotations will reset the camera offset.

Animating the people will largely be directed by the controller. However, the people will need to support some kind of positioning system outwith their current vertex. The current vertex only represents the vertex that they are occupying in the sense that no one else can move there. But to make the people look like they are moving, intermediate locations between the last occupied vertex and the new, or 'current', vertex must be made. By moving the people to these intermediate locations between vertices, it will create the animation effect and make the movement look more realistic on the display. Since this animation will be moving a person in a straight line between two points, Euclidean geometry should be used. However, this requires calculations that are relatively complex and will slow down the program. Instead, advantage can once again be taken of Taxicab geometry (as in the Manhattan heuristic used in the Purges A* algorithm). By incrementing or decrementing each of the x, y and z coordinates of the person, the animation effect can still be produced with a reasonable looking movement and a much lower processing cost.

It is important that people are not processed (i.e., the controller does not try to move them again) until they have reached their 'current' vertex through animation. Otherwise the animation will not be effective. The draw back to this is that the user will be seeing everything after the event. When a person moves to a new vertex, the location of the person in the logical view will immediately be this new vertex. But the user will see the person moving to this new vertex in intermediate steps. This obviously does not correctly show the logical state of the person but provides a more realistic output for the user. This can, however, produce side effects that may seem incorrect to the user. If a user blocks an area while a person is moving into it, but the person has not yet finished the animation step to that position, it will appear that the vertex is not blocked for that person. The person will continue to move into the blocked area and will then be removed from the environment (simulating death as previously described in this report). Although this seems counter intuitive, it is a relatively small side effect and provides a relatively easy means of applying animation. Had more time been available, this could have been fixed by allowing people to 'back track' to their last position if completing the animation put them in a hazardous environment.

The animation speed will depend on two factors: The amount that a person moves during each animation step and the amount of time between animation steps. The amount that a person moves during an animation step could be considered a person's speed. As such, this is just an attribute of each person and can be adjusted for each individual for more realism. This will most likely not be done due to time constraints. However, adding in different speeds for people using Monte Carlo methods shouldn't be all that hard.

The time between animation steps could be considered the frame rate of the animation. This will be the amount of time the animation thread is put to sleep between animation steps. Since the Purges program will take less or more time to work out path lengths depending on the size of the environment, the frame rate will need to come from the Purges environment as well.

There is one other aesthetic feature that will not be achieved due to time constraints, but it is important to mention should any future work be done with the simulator. During execution of the program, Purges has the ability to represent walls collapsing to provide alternate routes in the environment and also allows for parts of the environment to be blocked so that people cannot pass them. However, in both cases, there is no visual feedback for the user that this has taken place. This is a fairly advanced feature to include and would involve a much more complicated system for representing the environment. I could postulate that this would involve a 3D model with all the changeable features missing which are then added or removed in Java3D. But these removable pieces will need to be integrated into the Purges file format as well.

5.3 Controller

The controller is responsible for directing all tasks in the program and for coordinating the logical and physical views in Purges. The simplest way to show the functionality of the controller is to chronologically go through the tasks that it will perform.

The first thing that must be done is to create the graphical user interface for the program. The user will not be able to do much until opening a Purges environment file. Once the user does this, the controller will invoke the parser for the file format which will set up the logical and physical views, as well as any other variables already discussed.

The simulation should not start until the user wants it to. This will be done using some signal on the interface. Once the signal has been set, the controller will populate the environment with people according to the amount specified by the user. Each person is assigned a Monte Carlo value to represent their aggression. The simplest way of ensuring that the most aggressive person always gets to move into a vertex before any less aggressive person is to order the list of people by their aggression value and always process the list in order. This method would not work for an agent based system, but it is very effective in this central controller system.

The people will also be assigned goals to try and escape to. These goals are based on distance from the person. So, if a person knows that there is an exit near them, they will head to that one instead of travelling to one that is further way. However, people are not that accurate with decisions like this. So, instead of always making the correct decision, there is a crossover amount for the distance which means that two goals that are approximately the same distance will be seen as equally advantageous. In this case a random choice between the ‘equally advantageous’ vertices is chosen.

Once populated, the people will immediately begin their egress from the scene. When people have managed to evacuate the scene, the controller ensures that they are removed from both the logical and physical views of the environment. So the controller will just be in a loop trying to get people out of the environment until there are no people left. Each time the loop is run through, the controller will check to see if the animation process between vertices has been completed for each person. If a person has not reached their ‘current’ logical vertex in the physical view, the controller will animate the person another step towards the vertex. However, if the person has finished animating towards the vertex, the controller will assign the person a new vertex to move to by using the A* path finding algorithm. The controller will also need to check if a person has ended up moving into a blocked vertex (this may happen if the user blocks an area will a person is animating towards it). In this case, the controller will delete the person to simulate death.

In the step of finding a new vertex for a person, it might be that no path exists to the goal vertex. This can also happen if the user has blocked a goal vertex. In this case, the person is assigned a new goal. However, it is not a thought out goal, but a random one. This is intended to show that the person is now in a state of panic and is now heading towards which ever exit occurs first in their mind. It is also possible that the person cannot move to any goal as all paths to goal vertices are blocked. However, before being assigned a new goal, the person should still be assigned an immediate vertex to travel to. This should be handled by the A* algorithm. If no path to the goal is found, the longest moveable path in the direction of the goal should be returned. This will create the impression of moving towards the goal, even though a connected path does not exist. This is to represent the thinking time to determine that the person should change where they are heading. The added benefit to this is that when no goal can be reached, the people will still move around in a way that does not appear erratic. They will still seem to be searching for a way out and not simply giving up. If a path to a goal then becomes available again, the people should eventually find this as they will be assigned new goals continually until they get one that they can reach.

Once everyone has escaped, the program will wait for more input from the user. Whether that be to run the simulation again, or to open a new environment. However, if the entire population cannot escape, the simulation will continue running. It should be noted that the interface should always remain active to events such as opening a new environment. The user should not be locked into a specific environment until the simulation is done.

Chapter 6

Implementation

Having discussed all the theory and design for the system, this section will show the technical implementation aspects of the designs. This is, in essence, how the system was actually built from the designs. This section will also discuss any problems with the design that were not seen until implementation, as well as any other problems in building the system.

6.1 Language and Other Tools

As discussed in the research section of the report, the tools used were the Java programming language, the Java3D plug-in and the Blender 3D modeler.

Java was chosen due to my own familiarity with the language as well as its object oriented capabilities. It was felt that a graph based structure would be easier to program in an object oriented language. Java is often chosen as a programming language due to its portability. However, this was not really such an issue with this project due to the use of the Java3D plug-in.

The Java3D plug-in was chosen due to its easy integration with Java, as well as its relatively simple method of use. It allowed for the program to take advantage of both DirectX and OpenGL based systems. However, the Java3D plug-in is currently only available for Windows, Linux and Solaris systems. This means that the Java Runtime Environment is available for more systems than the Java3D plug-in. This reduces some of the portability of using Java, but still allows the program to work on most systems.

Blender, an open source 3D modelling package, was chosen for its features and cost. It is a very capable modelling package and costs nothing to use. It also works on all the systems that Java3D and Java work on. This means that it could be easily used, along with the other tools, to make more environments for Purges at a later stage.

These tools all work well together, have good features for making the program and work on the same systems, allowing for a good development platform.

6.2 Classes and Algorithms

The logical view of the system is largely based around three classes. The Graph class, the Vertex class and the Person class. The Graph class is essentially a collection of vertices. In fact, it is just an array of them. Each vertex has a list of all the other vertices it can reach, and it is this property that allows the graph structure to exist. Other than allowing the addition of vertices and edges to the graph, the only notable feature of the Graph class is the A* algorithm, which has been discussed earlier in the report. A depth first traversal function is also included so that the contents of the graph can be displayed for debugging purposes.

Although the A* algorithm is the only path finding system used by Purges, another could also have been included in the Graph class. This could then have allowed the user to choose which method to use in Purges. It should be noted, however, that the A* function is closely tied to the representation of the Vertex class. However, this should still allow for the addition of a breadth-first path finding algorithm such as Dijkstra's algorithm. Only breadth-first algorithms make sense as depth-first would not necessarily return a shortest, or near shortest, path. The A* algorithm used in Purges avoids the back tracking step due to a closely tied representation of the Vertex class, which could also be used to avoid the back tracking step in Dijkstra's algorithm as well.

The Vertex class is exactly as the design described it should be. However, after implementing this, it was felt that the close tie to the A* algorithm reduces the flexibility of this class. Although this has no direct effect on Purges, it will most likely need to be fixed if more serious work was to be done on the program. The values stored in the vertices for the A* algorithm could probably be shifted to the A* function itself. When initially implementing the A* function, storing the path information about the vertices in the vertices themselves seemed like an easy way of implementing the system. This is indeed true but is not good object oriented design.

The last aspect of the logical view is the Person class. The logical view aspects of this class are simply to associate each person with a current vertex and a goal vertex.

In order to construct the logical and physical views, a parser is required to process the Purges file format. The GraphMaker class is used to do this. The Purges file format is simply an XML file. There are two libraries in Java which can be used to deal with XML files: The SAX and DOM parsers. The SAX parser simply recognises tags and sets flags to deal with the varies tag elements. But the overall file structure is not dealt with. The DOM parser, however, converts the XML file into a hierarchical tree which can be manipulated by the user. Generally, when the program needs to modify an XML such that a new one is produced (or the old one is to be overwritten), a DOM parser is more suitable. However, if the XML is just being read and processed, a SAX parser is more suitable due to the linear nature of its processing system.

Consequently, the GraphMaker class is extended from the SAX parser supplied with

Java. This allowed for a very simple means of implementing a file format for Purges. All the information that needs to be put in the file is simply embedded in XML tags. This also makes the file format 'future proof' as adding new tag types to the file format doesn't effect the SAX parsers ability to read the old information that is required for this version of Purges. A particularly notable point is that the DXF file format, used for the 3D models of the environments is also text based. This means that it is easy to embed in the Purges file format within a DXF tag.

The physical view of the system is primarily dealt with by two classes: The Person class and the GUI class. The Person class deals with the animation aspects of each person by creating intermediate locations between vertices as well as dealing with the speed of each person in these animation steps. However, the actual display of the animation is still controlled by a different class: The GUI class.

The GUI class is the controller that was discussed in the design section of the report. It deals with the graphical user interface for Purges and also contains the crowd movement engine. This will be discussed in greater depth in the rest of this section. A full class diagram for Purges can be found in the appendices to this report.

6.3 Crowd Engine

The crowd engine contained in the GUI class directs everything that is displayed to the user after the 'start simulation' button is pressed. This includes the visual representation of the simulation as well as the calculations for the egress of the crowd. This section will discuss the technical aspects of note in the implementation of the GUI class.

6.3.1 Movement

The movement in Purges is largely governed by the A* algorithm. But in turn, the A* algorithm is governed by the state of the vertices in the graph. A* does not see if a person is on a vertex or not, it merely sees if a vertex is blocked. However, this blocked state will always prevent people from occupying the same vertex. But it may not prevent people from travelling 'through' each other. For instance, let us assume that there are four vertices all side by side forming a square and two people occupy two adjacent vertices. If the two people wish to diagonally cross each other to reach the vertices diagonally opposite their current position, then Purges will see no problem in this and allow the movement. However, this will lead to the people apparently moving through each other.

Although this movement seems unrealistic, it is still valid because the total number of Vertices involved in the movement compared to the total number of people in the movement is a feasible use of space. Even though the movement would doubtless not happen in this exact manner (obviously people do not move through each other in real life), the start and end positions of the people in the specified amount of time is still valid. This is certainly an aesthetic problem that future versions of the program should address, but it does not affect the validity of the people's movement. Essentially, all movement in Purges is a representation of space used by people, not the actual people themselves.

Another problem with the movement system is the animation and the limitation of processing power. Essentially, the speed of the computer being used to run Purges will affect the speed of the animation. The speed of the animation is governed by two factors: Frame rate and person speed. The person speed is a variable set in the Person class. It was designed this way so that the speed of individual people could be easily changed to represent the different mobility capabilities of people in the real world. However, due to time constraints, there is no variance in people's speed and everyone moves at the same specified velocity. This is not a true representation of speed, but merely the distance that a person will travel during each animation step. The second variable in animation speed is the frame rate.

The frame rate is simply how long the animation thread will go to sleep between animation steps. So, after everyone has been moved by one animation step, the animation loop waits for the specified amount of time before moving everyone in another animation step. The problem with this is that the time taken to process the animation step for everyone in the population is dependent on the size of the population, the size of the environment and the processing power of the computer hardware. The larger the number of people or the number of vertices, the slower the animation will become. Also, the slower the computer, the slower the animation will become. This means that the animation in Purges for a specific environment will run at different speeds on different computers. This is an extremely important point.

If the user is testing the speed of egress using the Purges timer, it is only a relative measure against other environments run on the same computer using Purges. The results cannot be compared against other environments tested on other computers. This raises another important question: Will a simpler environment run faster than a complicated environment on the same computer? The natural assumption would be "yes", but Purges does provide a way to equalise the environments. The Purges environment files have a tag called "framerate". The tag attribute "default" is set to equal the frame rate for that environment. In this way, the various Purges files can be made to show the same animation speed. This also allows for the Purges environment files to be run faster or slower on different computers. Although not an ideal solution, it does work. The only remaining problem is that there can be a slight increase in animation speed during a simulation. This is because, as people escape from the environment, there is less work for the computer to do. However, the size of the environment has a much greater impact on the animation speed than the size of the population, so the effect is minimal.

This change in speed is an unfortunate side effect of the animation process. However, the animation provides a much better representation of the egress, and showing an evacuation in an easy to understand manner is one of the main reasons for creating the program. The

Purges timer is not affected by the animation effect and so is not accurately connected to the simulation. It is merely useful as a relative timer between simulations on the same computer system. The last secondary effect is that the speed up options on the interface (the ability to run the simulation at double, four times and eight times normal speed) will not always be accurate due to computer hardware limitations. The pertinent point is that all these problems are solved by a fast enough computer system. Thus, perhaps the minimum system requirements for the program should be set accordingly.

6.3.2 Coordination

The coordination between the logical and physical views of the world is maintained by two factors. An accurate mapping of the 3D model file by creating the correct coordinates for the vertices means that the people will be moved to the correct position. The second factor is keeping the current vertex of each person up to date.

The first factor is maintained by a correctly made Purges environment file. The second factor is maintained in the Person class. The GUI class calls each person's move function in order to change the location of a person. For each logical person, there is a 3D model associated with it. In the current version of Purges, this is a simple box. The connection between each person and its 3D counterpart is maintained by means of a hash map (this maps a string to an object). If a person is moved or animated, then the 3D model for that person is updated by the GUI class.

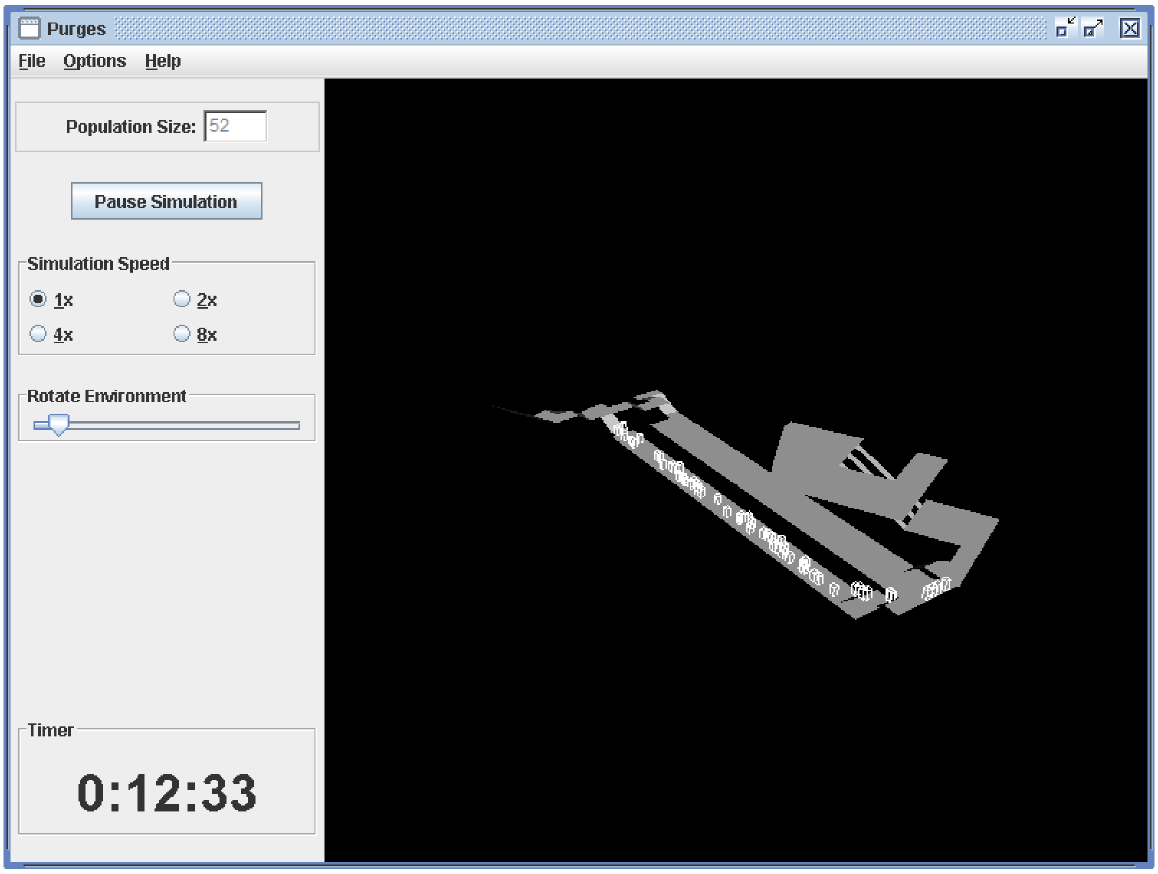
As long as these factors are correctly maintained, the coordination between the logical and physical views should always be accurate.

6.3.3 Person Removal

When a person is considered incapacitated or has finished his or her egress, that person is removed from the logical and physical views. Removing a person from the logical view is simply a matter of removing them from the population list of people to process. However, a person must also be removed from the physical view. This is also a fairly simple task: The 3D model for the person being removed is detached from the Java3D tree. As long as it is not in the tree, it is not a child of the root of the tree. Thus, it cannot be displayed. The Java garbage collector will detect that the 3D person model is no longer being used and will delete it from memory. In much the same way, if nothing is referring to the person in the logical view, it will also be deleted from memory.

6.4 Graphical User Interfaces

The Purges graphical user interface is built using Java Swing. This allows for a consistent interface across all platforms. The interface contains many elements which will be given a brief overview here. A screenshot of the Purges interface can be seen in figure 6.1.



Population Size: 52

Simulation Speed: [✓ 1x] 2x 4x 8x

Timer: 0:12:33

Figure 6.1: Purges Interface

There are a series of menus at the top of the Purges window. These provide easy access to open a Purges environment file and to open the block areas window. It should be noted that the block areas window was originally intended to be integrated into the main window. However, it is difficult in Java Swing to change the layout of windows during runtime (which is required for the block areas function as it is different for each environment). As such, the block area function is displayed in a new window.

Most of the interface is not usable until the user opens a Purges environment file. However, once this is done, there are many more options available. The options menu may or may not have the block areas option enabled. This will only become enabled if the Purges environment file has information that areas of the current environment can be blocked.

The rest of the interface is straightforward enough and had no significant implementation problems or points of interest.

6.5 Environments

The Purges environment files are XML based and include both the physical and logical view representations of the environment. The physical view is a DXF file embedded in one of the XML tags and the logical view is a series of tags as described in the design section of the report.

The ordering of some of the tags is important. In particular, it is assumed that the files are entirely correct, such that there are no duplicate entries or omissions. It is also important that all vertices are declared in the file before any of the edges. The implementation of Purges requires that the vertices that edges connect must exist before the edges themselves are made. This is largely due to the fact that edges are not a separate object in the implementation. It was seen at the time that making them objects was an over complication of the system, but this is the only current drawback to that choice.

The main problem with the environments is that their construction is long and tedious. In fact, the three environments supplied with the current version of Purges are more to demonstrate the flexibility of the simulator than to be run as actual simulations. Two of the environments are merely demonstration files, the third being an unfinished Purges file of the Hillhead subway station. Although in all cases, the DXF model file was successfully completed for each environment, cataloguing these models to produce a logical representation took an excessive amount of time.

This can in fact be attributed to time constraints in the project. The preferred way to produce the logical models would be to create an automated process that would determine where all the vertices were for the DXF model. However, this was a large undertaking and was unable to be achieved in the time set out for the project. However, the current unfinished Hillhead environment file contains approximately 300 vertices, each of which were coded by hand largely using trial and error methods. From this, it can be approximated that the entire model contains around 2000 vertices. However, it took around four days to catalogue the 300 that were done. This would mean that cataloguing all the vertices would take just under a month. This would be completely unacceptable should Purges be used to make environment files for all the subway stations in the Glasgow Subway system. There was not enough research done in this area for me to suggest a solution to this problem, but some form of automation must be attained for the creation of Purges files to make this a valid product.

Chapter 7

Prototyping and Improving

The initial version of Purges was the one that fully corresponded to the design that was set out. This was classified as the prototype version. However, there were changes made to the system after this initial stage was reached which led to the current version which, although still considered a prototype with many possible improvements, is now considered stable enough for production use.

The major changes made to the system were largely superficial and often involved making the GUI more robust. For instance, in the original prototype, the rotation slider would not reset when a new Purges environment was loaded. This produced a 'jump' effect when the rotation slider was first used for the new environment. This was easily corrected by manually resetting the slider as part of the file loader function. Similar effects like this, which were unseen in design, were found during rigorous testing of the program. They are too numerous to mention here, but the more major changes are discussed below.

A feature that was added was to have the population input text box show the current (i.e., changing) population once a simulation was started. Since the user is not allowed to change the population once the simulation has started, this seemed a sensible use of the component and a relatively easy improvement to make, which was appreciated by the end users at Strathclyde Passenger Transport.

A loading screen was added to the program to show that it was doing something when Purges environment files were being loaded. The program takes long enough to process these files to possibly lead a user to feel that the program is doing nothing. The user may then try to open the same file again, thus increasing the waiting time further. The addition of the loading screen lets users know that the program is carrying out their task.

There were no other significant departures from the original design in the final version of Purges in this project.

Chapter 8

Evaluation and Validation

The Purges simulator is a difficult program to evaluate. Although user testing could be carried out, it is essentially only aimed at a few people. Therefore, it is only the opinion of these end users that can determine if the program is valid or not. In fact, there is only one person that the program is aimed at: the Commercial and Production manager at Strathclyde Passenger Transport. Although the effects of using the program could affect many people, it is largely only this person, Liz Parkes, that would be using the program.

Liz Parkes kindly agreed to undergo an evaluation of the Purges simulator and provide feedback on the system, as well as validating whether or not she could use it. Her initial reactions to the program were extremely positive saying that she could immediately see a use for it in the safety training of SPT employees. She commented that the user interface was very straightforward, although initially she did not realise that a file should be opened before the simulator would be useful. However, this one point aside, she was fully capable of using all the program's features with little or no guidance.

She also commented on the features of the interface such as the ability to change the simulation speed and the timer, saying that she felt that the first was a good time saver and that the second would be a useful means of comparing the variables in the simulation such as population size and blocking environment areas. As these were the intended purposes of these components, this was very good feedback.

She also felt that the use of a file format made the program much more usable and that it was a very good feature to include as it would allow the same system to be used across all the stations in the Glasgow Subway, as well as other railway stations under the control of SPT. This demonstrates the flexibility of the system, as even someone seeing it for the first time realised that it was capable of more than just the evacuation of the presented subway station.

Ultimately, the manager said that this was a program she could, and indeed would, like to use. As such, SPT will be given a copy of the program for their own use. This also means that the project is a success in terms of its productive validity.

As for my own feelings about the program, I feel that much of what was set out to be achieved has indeed been accomplished. However, my greatest concern is that the research done on group behaviour was not all implemented in the crowd engine. The people certainly behave in a manner that is realistic enough to be of use, and all their behaviour can be explained using psychological research or logical basis. As such, there is nothing, strictly speaking, wrong with their movement. However, the insight of the research obtained by studying 'crowd wisdom' was never implemented in the program. This means that people do behave on their own rather than as part of a crowd. The research clearly showed that this is not the normal behaviour of people in this situation. This means that the simulator is still valid enough to be useful, but could certainly be more accurate.

A secondary issue was that the safety information provided by SPT showed that the staff would help passengers in the event of an evacuation. As seen from the simulator, there are no staff and certainly no one helping anyone else. Work in this area has been done in previous simulators, such as the G-HES[18] system made at the University of Glasgow, but there was not enough time to implement this in Purges.

The essential result is that the simulator is a success, but there is still a lot of room for improvement.

Chapter 9

Conclusion

The focus of the project was to create a usable evacuation simulator for the Hillhead subway station in the Glasgow Underground. In this respect, the project was a complete success. The system was accepted by Strathclyde Passenger Transport as a product of use and one that they would indeed like to use. All the essential features were included and even some of the additional features. However, there are still many aspects in which the simulator could be improved and many features that could be added.

The largest problem with the usability of the system is the creation of the Purges environment files. As shown in the report, cataloguing the 3D models with the logical information of the vertices takes too long to do by hand. It also locks the designer too much into the 3D model they create as changing the model changes the vertex locations as well. The only viable solution is to create a separate program that, at least partially, automates this process. This is not a small task but I feel it is necessary to allow Purges to be used completely.

The largest problem with the crowd engine is that it does not really simulate crowds: It simulates lots of individuals in the same environment. Almost none of the research done on crowd behaviour was implemented in Purges. This was largely due to time constraints, but this is more important that any aesthetic problems that the program might have. The most fundamentally important aspect of the simulator is that it is an accurate representation of people. At the moment it is passable enough to be useful, but it is by no means accurate.

I also feel that more work should have been done on the environments themselves. The only environment factor seen by Purges is whether or not an area is blocked. The real world is a lot more complicated than this. There are environmental factors such as the surface of the floor. Is it wet? Is it slippery with dust? And the program should also include information about the visibility of the environment. Is there smoke? If there is smoke, does this impair a person's ability to navigate? This still has not mentioned the hazards of the environment and how they will affect people. If there is a fire, does the heat produced impair people's capabilities? And we have still not mentioned permanent factors such as stairs or turn styles and that people should move slower in these areas. We have still not included features such as doors, escalators, elevators and other environmental factors that the evacuees could use to their advantage.

Naturally all these features would make Purges much more complicated than it is now. The design used in this version was intended to be simple enough to allow for easy expansion, but I am not sure that the design is solid enough to be able to incorporate all these features. Especially the aspect of choosing a central controller over an Agent based system. The central controller is much easier to implement and change on simpler systems, but in a larger system, the controller will become too complex. Probably much more so than other areas of the program. Even the current version of Purges has a controller that is larger than most of the other classes added together. An Agent based approach would help disperse the duties of the program in a much more object oriented fashion which will provide for easier expansion and growth in a more complicated system.

An agent based approach would also allow for more individualism in the people. At the moment, the only thing that really differentiates people is their aggression level. Everyone has the same physical capabilities and the same intelligence. Apart from different levels of aggression, they also have essentially the same personality. However, in real life, people are very different. All the aspects of crowd behaviour dealt with the changes in people's personalities when part of a group. However, if they have no real personality to begin with, this is no more interesting than programming a long, complicated list of rules for people's behaviour. This would essentially list to a more or less predictable behaviour for people. But by increasing the number of variables about people, the simulation will become more random within the boundaries or realistic behaviour, and the simulator will become more viable.

Obviously, improving the aesthetic capabilities is important as well. The better the program is at displaying the simulations to the user in an easy to understand way, the more useful it will be.

This report has shown that the project can be seen as a success. But it is merely a stepping stone to a better system. The area of evacuation simulators is still developing and is a fairly young discipline. The greatest achievement of the Purges system was introducing a simulator than could work on different environments, but even this was only done to a simple level. Had more time been available, I have no doubt that Purges would be a better program, but to implement some of the advanced features I have mentioned, a complete redesign would probably be necessary.

9.1 Future Work

From my experiences on building the Purges program, I feel that future work on evacuation simulators will stem from a more complex person model and a more complex environment model. I would imagine that the person model will include techniques from artificial intelligence. The creation of artificial agents that can learn from their environment during the simulation will be a more realistic representation of people. Should the creation of such entities be accurately created, this could lead to an easier method of modelling people. System that can accurately learn how to behave correctly will probably take less time to make than an explicitly programmed person.

The creation of a more complex environment is less likely to be done using artificial intelligence techniques. In fact, it will almost certainly not be. However, it is likely that the environment model will need to be modelled suitably enough for the artificially intelligent people to understand it.

I see these areas as the most significant areas of future work on simulators. As for building a general purpose evacuation simulator for any environment, I'm not sure that the more complicated systems will make the creation of such programs easier or harder. An artificially intelligent person would be more flexible with the environments that it could evacuate from, but the environments themselves would ultimately need to be created specifically for each environment. However, a unified modelling system for the creation of evacuation simulators seems like a good direction to go in. A program that allows the user to configure an environment using pre-built general purpose components, such as elevators or air vents, could end up being an easy way of creating simulators that would essentially be specific, but built from a general purpose evacuation simulator creator.

Bibliography

[1] Wikipedia (2006) *Simulation*, [Online],
Available: http://en.wikipedia.org/wiki/Simulation [10 Apr 2006].

[2] D. G. Myers. *Social Psychology, 7th edn.*. McGraw-Hill, New York, NY, 2002.

[3] NIST Investigation of The Station Nightclub Fire (2005) *Key Findings and Recommendations For Improvement*, [Online],
Available: http://www.nist.gov/public_affairs/factsheet/mar_3_rifindings.htm [10 Apr 2006].

[4] University of Glasgow, Department of Computing Science (2006) *Chris Johnson's Publications*, [Online],
Available: http://www.dcs.gla.ac.uk/ johnson/papers.html [10 Apr 2006].

[5] University of Greenwich, Fire Safety Engineering Group (2006) *Exodus*, [Online],
Available: http://fseg.gre.ac.uk/exodus [10 Apr 2006].

[6] Wikipedia (2005) *Evaluation Apprehension*, [Online],
Available: http://en.wikipedia.org/wiki/Evaluation_Apprehension [10 Apr 2006].

[7] Wikipedia (2006) *Groupthink*, [Online],
Available: http://en.wikipedia.org/wiki/Groupthink [10 Apr 2006].

[8] Wikipedia (2006) *Risk Perception*, [Online],
Available: http://en.wikipedia.org/wiki/Risk_perception [10 Apr 2006].

[9] B. Zarboutis et al. *Design of Formative Evacuation Plans using Agent-Based Simulation.* National Technical University of Athens, Athens, Greece, 2005.

[10] B. Zarboutis et al. *Investigating Crowd Behaviour during Emergency Evacuations Using Agent-Based Modelling.* National Technical University of Athens, Athens, Greece, 2005.

[11] Wikipedia (2006) *Taxicab Geometry*, [Online],
Available: http://en.wikipedia.org/wiki/Taxicab_geometry [10 Apr 2006].

[12] Wikipedia (2006) *Euclidean Geometry*, [Online],
Available: http://en.wikipedia.org/wiki/Euclidean_geometry [10 Apr 2006].

[13] Wikipedia (2006) *Monte Carlo method*, [Online],
Available: http://en.wikipedia.org/wiki/Monte_Carlo_method [10 Apr 2006].

[14] J3D.ORG (2004) *CAD Viewer DXF Loader*, [Online],
Available: http://java3d.j3d.org/utilities/loaders/dxf/raida.html [10 Apr 2006].

[15] anim8or (2006) *Free 3D Animation Package*, [Online],
Available: http://www.anim8or.com [10 Apr 2006].

[16] DXStudio (2006) *3D Editor and Real-time 3D Engine*, [Online],
Available: http://www.dxstudio.com [10 Apr 2006].

[17] Wings 3D (2006) *3D Subdivision Modeler*, [Online],
Available: http://www.wings3d.com [10 Apr 2006].

[18] C. W. Johnson. *Using Computer Simulations to Support A Risk-Based Approach For Hospital Evacuation.* Glasgow Accident Analysis Group, Glasgow, UK, 2005.

Appendices

Appendix A: Purges Class Diagram

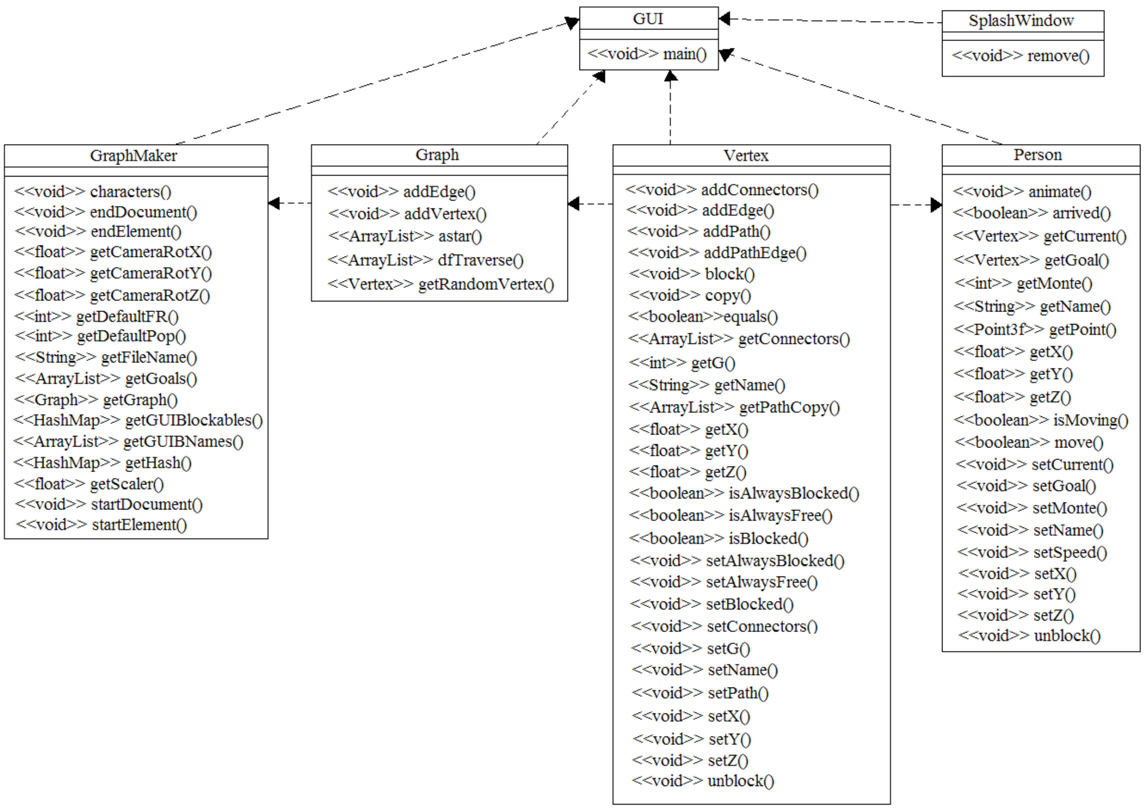


Figure 9.1: Class Diagram