University of Glasgow | School of Computing Science

T.S.E.S: Tall Ship Evacuation Simulator

Michael Kilian– 1003819K
Tony Lau – 1102266L
Dan Tomosoiu –1102486T
Hector Grebbell – 1007414G
Peeranat Fupongsiripan – 2056647F

Level 3 Project — 19 March 2013

Abstract

This project is concerned with the development of an evacuation simulator for the Tall Ship at Riverside, Glasgow. Such a system may be used in place of mock evacuations. As well as saving the resources needed to run such evacuations, this gives the user control over environmental and population variables, allowing them to experiment with a large number of alternate scenarios at their own convenience.

Many such systems have been developed using varying approaches. The aims of this project reflect a desire to use the best of these approaches along with modern software frameworks to create a flexible and extensible system.

Most of the project's aims and requirements were satisfied. However, various difficulties were encountered in the late stages of the development life-cycle which ultimately led to a reduction of the system's scope.

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

1.1 Introduction

1.1.1 Definition of an Evacuation Simulator

An evacuation simulator can be defined as a system to determine evacuation times by predicting the egress of individuals in a building or similar structure [1]. They are used for the identification of weaknesses in the design of buildings which could detrimentally affect the egress of persons in an evacuation and to aid personnel in preparing for an evacuation [2].

1.1.2 Why Implement an Evacuation Simulator

The task of testing a building's evacuation procedure can be both difficult and expensive. One approach is to hire members of the public as stand-in "evacuees" and run a mock evacuation, such as those performed as part of Forward Defensive in preparation for the London 2012 Olympics [3]. In theory, this would allow an appropriate expert such as a consultant from a local fire department to assess the effectiveness of an existing plan. However, such tests on public buildings can be very expensive: the cost of hiring evacuees could be significant. Also certain aspects of evacuations, such as testing the possibility of a crush, can expose participants to real danger. Finally if an evacuee knows they are participating in an experiment their behaviour is inherently different than it would be in a real evacuation. This phenomenon is known as Evaluation Apprehension[4].
For these reasons large scale tests on a building are rarely performed. What a simulator provides is a means for an expert to extensively test the outcomes of evacuating a location at minimal cost. By configuring variables in the simulation the expert can examine the probable outcome of multiple evacuations and look for potential sources of danger in a building or evacuation procedure.

Of particular note is the ability to investigate the effects of human behaviour on the success of an evacuation. This ability is central to a successful system and it is desirable to model the behaviour of the population as accurately as is achievable [5].

1.1.3 Aims

The project's aims are driven by the desire to maximise the flexibility and extensibility of the system, and to make the system as accurate as is achievable within the time available. The four key aims can be summarised as follows.

A Multi-Agent Model of Individuals and Individual Behaviour

Previous projects have emphasised crowd behaviour: they have modelled the population from a top down perspective with little consideration for the perception and interaction which an individual experiences in an evacuation. One of the primary aims of this project is to represent an accurate model of individual behaviour. Previous work has shown that multi-agent systems can achieve this. By taking this approach "the full effects of diversity that exists among agents in their attributes and behaviours can be observed as it gives rises to the behaviour of the system as a whole" [6].

Use of Up-To-Date Technologies Wherever Possible

In even recently developed simulators now deprecated technologies such as Java3D are being used. This limits their future extensibility greatly and is not desirable. The underlying technologies used in development were chosen primarily based on their future potential and strength of community support. These are discussed extensively in the Research chapter.

Use of Navigation Meshes in Environment Modelling

A Navigation Mesh is a graph representation of an environment in terms of a set of convex polygons which describe the 'walk able' surface of an environment. This design aids agents in finding paths through large areas, whilst avoiding static obstacles in the environment. This technique has largely been pioneered in gaming, but is equally applicable in this setting. The benefits of a navigation mesh, or 'navmesh' is that it allows agents to move freely compared to other techniques such as representing the environment using a grid. A navmesh is typically combined with the powerful path finding algorithm A* to optimise agent movement [7].

1.1.4 Prerequisites

Where possible all technical concepts used within this paper will be clearly defined. However, to fully understand the remainder of this report, the reader should have at least some knowledge of the following concepts:

- Object-Oriented programming concepts (preferably in Java)
- Concurrent programming concepts
- Pathfinding simulation and collision detection/avoidance [8] [9]

- Herding, flocking behaviour and boids [10] [11]
- Human behavioural traits [12]

1.1.5 Previous Work

Fluid Based Systems

These systems model crowd movement as if the crowd were a fluid [13], using equations and principles taken from Physics. Exodus [14, 15] is an example of such a system. There are drawbacks in such a style of simulator. Crowds "have a choice in their direction, they have no conservation of momentum and can stop and start at will" [16]. These concepts are not accounted for in fluid models, and so this style of simulator is limited to estimating the movement of a crowd as a whole without considering individual interactions.

Matrix or Grid Based Systems

In a matrix or grid based system, the floor of the environment is represented by a series of adjacent nodes, often square or hexagonal in shape. Each cell can represent open areas, areas blocked by a static obstacle, exits, etc. This method is becoming less common, but two formerly well-known examples are Egress and Pedroute. It was suggested that the existing matrix-based models suffer from the difficulties of simulating crowd cross flow and concourses. Furthermore, the assumptions employed in these models are questionable when compared with field observations [16].

Emergent Agent Based Systems

The final class of simulator we discuss here is the emergent (agent-based) simulator. In this approach the system is composed of autonomous and interacting "agents". Agents interact within an environment using a defined set of simple relationships. The benefit of this approach is the introduction of *emergent* behaviour: "patterns, structures and behaviours emerge that were not explicitly programmed into the models, but arise as the result of agent interactions" [6].
The MASSEgress project developed at Stanford University is an ongoing effort to develop a framework for the development of such systems [17]. It has also led to the production of at least one prototype implementing this framework. This project has utilised a modified version of this framework for the integration of agent behaviour. This is discussed further in Sections 4.1 & 6.2.

1.2 Environment

For the purposes of designing and evaluating the system it was necessary to choose a readily available location to be used as the 3D model with which the system operates. By choosing such a location there is the potential to run mock evaluations and other tests in the future. The environment chosen is the Tall Ship at Riverside, Glasgow [18].

Figure 1.1: Glasgow Tall Ship

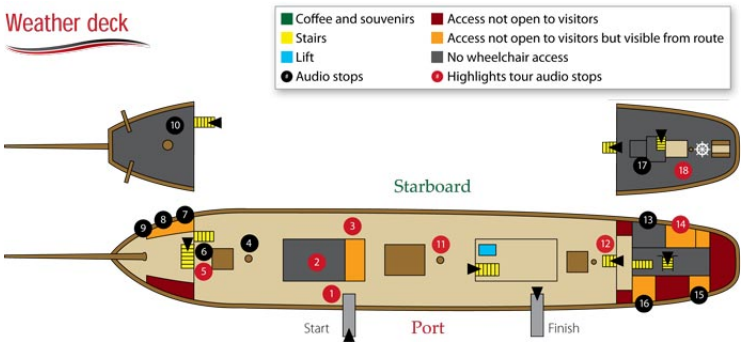


Figure 1.2: Weather Deck

The Tall Ship was chosen as the simulation environment because:

- It serves as both a tourist attraction and a function hall below decks.
- The ship is permanently docked and can be considered a static structure.
- Events can host up to 200 guests, excluding staff.
- It has a sufficiently complex structure in which to explore simulation techniques.
- No full scale evacuations have ever been held before - only staff have been used.
- These drills are infrequent.

It is important to note that only small scale staff evacuations have been conducted on the ship due to the impractical nature of carrying out an evacuation with actual visitors. Because of this restriction, an evacuation simulation of The Tall Ship is ideal to assess the safety of visitors on the ship in the event of an emergency evacuation.

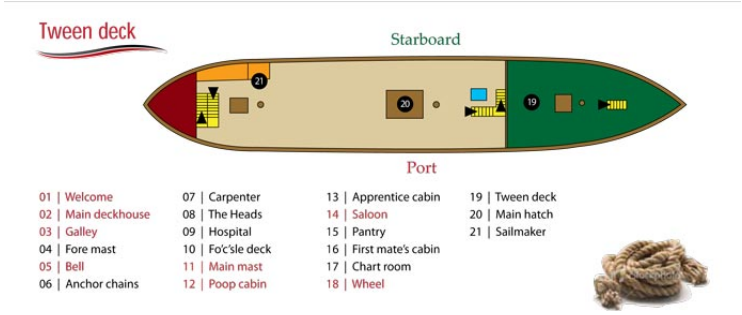


Figure 1.3: Tween Deck

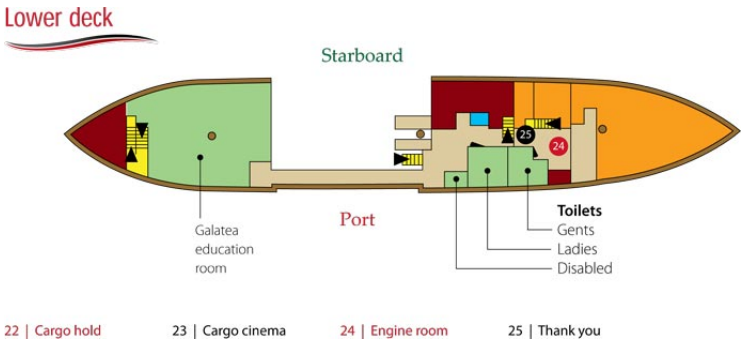


Figure 1.4: Lower Deck

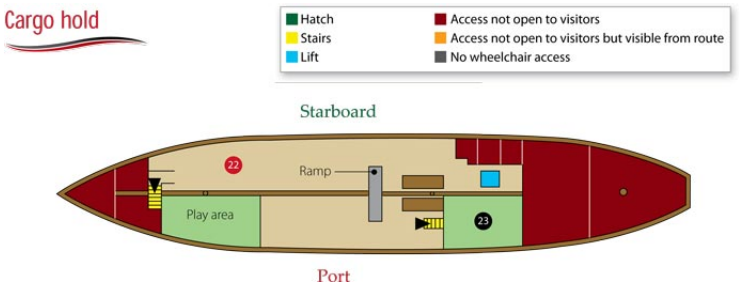


Figure 1.5: Cargo Hold

Chapter 2

Team Structure & Development Process

2.1 Team Structure

In order to develop a structured approach to task allocation, the software engineering tasks required were structured using the Administrative Programming Team [19]. This consists of the following roles:

- **Project Manager**
- **Librarian**
- **Configuration Manager**
- **Toolsmith**
- **Quality Assuror**

It should be emphasised that each person was not solely responsible for the tasks associated with their role; their responsibility is to coordinate these tasks within the team by proposing, implementing and maintaining effective procedures to achieve this.

In tandem with this, the team was further divided into two subteams for development:

- **Modelling and GUI Team:** responsible for development of any 3D models required including the final model of the ship. In the later stages of the project this team developed the graphical user interface.
- **Core Implementation Team:** responsible for all other development tasks, including implementation of the navigation mesh, population model, etc.

By separating the development of the user interface from the development of the underlying program logic, the team aimed to promote a Model-View-Controller design. [20, Ch 6.3.1].

2.2 Development Process

One of the challenges of designing an evacuation simulator is defining a level of accuracy which can be deemed acceptable with respect to the project's resources, and then translating this into an effective design which balances the use of up-to-date techniques with an implementation plan. Many of the techniques that this project aimed to implement are complex. These techniques are discussed further throughout the Research Chapter
Ultimately, it became clear that the most tangible way to make progress was to use a strategy of incremental prototyping [20, Ch 2.3.2]. This would allow the team to progressively 'scale up' ideas and to investigate the feasibility of implementing certain principles in a structured manner. However, incremental prototyping carries the following considerable risks which must be addressed:

- A tendency to produce low quality and difficult to maintain code.
- Difficulties in managing change.
- Tendency to sacrifice quality assurance and documentation because of poorly understood aims.

To mitigate these risks, several techniques taken from the field of agile development [20, Ch. 3] were employed as follows:

- **Division into Subteams:** The team was divided as outlined above so as to allow these subteams to work in parallel on orthogonal tasks. This reduced communication overhead and the difficulty of managing change to the system.
- **Constant Refactoring:** Before the completion of each prototype or upon fixing a defect, significant refactoring was undertaken to improve code quality.
- **Pair Programming:** This technique was particularly helpful when fixing defects related to navigation (see Section 8.2.1) due to the complexity of these defects.
- **Test First Development:** wherever the understanding of requirements was sufficient to allow it, test cases were developed for a feature before they were implemented.

Chapter 3

Requirements

Due to the scope of the project it would be impossible to commence without a clear vision of how the end product should function. This also aids greatly in breaking down the system into seperate components allowing team members to work independently without repetition of work. These requirements were decided on during the early stages of the project. The reasoning behind them comes from a number of sources, primarily -

- Research into previous attempts at evacuation simulation. The teams behind these will have each conducted their own research. Rather than repeat their efforts examination of their choices and rationalisation provides a valuable and dense insight into what shall be required.
- Interviews with a target user (The fire warden at our initial site).
- Requirements derived from research into evacuation and human behaviour.
- Discussions amongst the team.

The requirements have been split into several sections depending on their necessity, usefulness and difficulty in implementation.

3.1 Must Have

These are the bare minimum requirements for the system to be suitable for any kind of customer use. Without these the software would either be considered entirely non-working or faulty. Not managing to provide these would indicate complete failiure of the project.

- Representation of a 3D environment in a manner allowing the system knowledge of navigatable surfaces.
- Generation and representation of a population within the 3D environment.
- Ability to manipulate the population to allow movement towards a given location.

- Output to the user of time taken for the entire population to successfully move to a safe location.
- Basic behavioural and routing algorithms to generate paths of movement for the population.

3.2 Should Have

High priority requirements which should be satisfied where possible. Missing any of these would indicate the project as incomplete and failing to meet all of its objectives.

- Accurate behavioural models allowing realistic actions from individuals within the population.
- Interaction of members of the population to allow realistic crowd behaviour and collision control.
- Ability for the user to specify environmental and population variables (such as population size).
- Pseudo-random population generation including individual characteristics and behavioural patterns as well as initial position.
- Accurate Timescale for evacuation process
- Intuitive GUI allowing viewing of both environmental variables and a visual representation of the environment throughout the evacuation process.
- Ability for user to change perspective (camera location) within the visual representation.
- Ability to import alternative environmental models.
- Extensive Documentation and user help.

3.3 Could Have

Requirements considered desirable but not necessary. To be included if time and resources allow.

- Ability to pause and resume the simulation
- Representation of assurances and hazards within the environment such as exit signs and low ceilings.
- Variable run speed of simulation
- Ability to output information such as evauation time and path of each member of the population to file allowing further future analysis.
- Accurate modelling of advanced crowd interation, such as staff assisting the guest population.
- Unusual population traits such as disabled guests and the interaction and assistance required for their safe exit.

3.4 Would Like to Have

Features unlikely to be present in the initial release due to time and resource constraints.

- Accurate fire and smoke model.
- User control over fire start location.
- Environmental model support of different materials to allow further accuracy to the fire model.
- Advanced and highly realistic graphical representation of environment.

Chapter 4

Research

4.1 Research into Human Behaviour

4.1.1 Non-adaptive Behaviour

Adaptive behaviour is any behaviour "which contributes directly or indirectly to an individual's survival" [21]. Conversely non-adaptive behaviour is any behaviour which may be counter productive to an individual's survival [21]. In the context of an evacuation this refers to high risk actions which occur in a crowd such as stampede, pushing and shoving others, trampling others, etc.
The introduction of non-adaptive behaviour can be connected to the stress a person is feeling. More specifically, Law et al [22] propose that there are three factors which contribute to the emergence of adaptive or non-adaptive behaviour:

- **Panic**: when a person perceives danger they are more likely to make irrational decisions based on instinct [23].
- **Decision-making**: although panicked, a person is still capable of making rational decisions. This increases the likelihood of the individual making adaptive choices, such as correctly recognising an exit or refraining from shoving upon exiting.
- **Levels of urgency to exit**: individuals within a group will experience varying levels of arguing to leave the environment based on the level of danger they perceive themselves to be in. High urgency causes individuals to behave aggressively and prioritise self-preservation.

All of these theories provide insights into human behaviour but have yet to be unified into a comprehensive theory.

4.1.2 Bounded Rationality

Bounded Rationality can be expressed as the principle that in a given situation the rational decisions a person can make are bounded by the set of possible options available to them and the time in which

they can make a decision. This concept was initially proposed by Herbert Simon, who highlighted two interlocking components of bounded rationality [24]:

- **The limitations of the human mind**: the human mind does not have limitless processing power or memory and so must use approximate methods to handle most tasks. For computational purposes these methods can be expressed using simple heuristics.
- **Environmental structure**: Simon emphasised that the heuristics used must be adapted to the environment in which the decision is made.

An important example of this priciple is Simon's concept of satisficing: a "method for making a choice from a set of alternatives encountered sequentially when one does not know much about the possibilities ahead of time" [24]. In an evacuation a person will use this satisficing technique to make decisions such as where to move next based on what they can see.
It is important to note that, like other rational behaviours, this process can be disrupted by the introduction of stress factors, as previously discussed.

4.1.3 Conformity & Social Proof Theory

Conformity can be defined as a "change in behaviour or belief toward a group as a result of real or imagined group pressure". This can be further divided into *normative influence* and *informational influence*. Normative influence refers to changes in behaviour for the sake of winning the approval of other group members. Informational influence refers to conforming in an attempt to improve one's knowledge of reality and current situation. [25].
Informational influence, which is often called Social Proof Theory [26], has the effect that when facing a situation with uncertainty, a person may turn to the surrounding group for cues. Cialdini noted "we seem to assume that if a lot of people are doing the same thing, they must know something we don't" [23].
This principle is at the core of many of the behaviours observed in evacuations, particularly herding behaviour [23, 17].

4.1.4 Personal Space

According to Edward T. Hall [27] the space surrounding a person can be divided into four "reaction bubbles" which represent an acceptable radius around the person in which different categories of interaction. These are:

- **Public Space**: this space is used for speeches or other interactions with large audiences. This includes anything beyond roughly 2.4m
- **Social Space**: the distance reserved for interaction with strangers or newly formed groups. This ranges from around 1.2m to 2.4m
- **Personal Space**: this begins an arms length away from the individual's center and is reserved for interactions with friends or other close associates

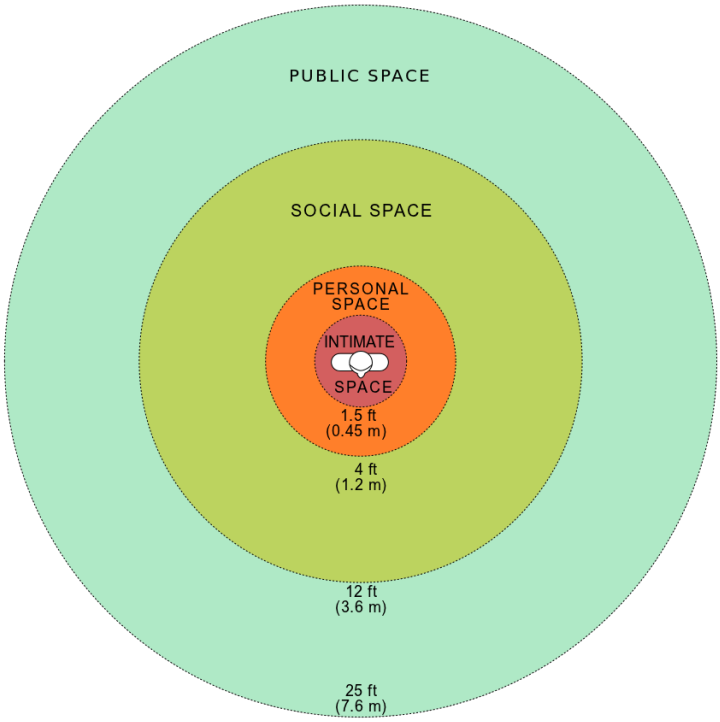


Figure 4.1: Edward T Hall's 'reaction bubbles'

- **Intimate Space**: The smallest of the spaces, this ranges from touching the person to around 50cm from their body. Interactions in this space are extremely personal and normally occur only with close members of family or friends

The last two of these spaces are of particular interest. Another definition of personal space is the area surrounding an individual "into which others may not intrude without causing discomfort" [25, pg. 424].
In a crowded environment the violation of one's personal space is likely to increase stress and anxiety (see page 17. Naturally this effect is even greater when the intimate space is violated. An individual's desire to re-establish this boundary can introduce non-adaptive behaviour.
It should be noted that the definition of these four spaces varies between individuals and between different cultures. Therefore the ranges discussed here are only an estimate [28]. Figure 4.1 shows these 'bubbles'.

4.1.5 Classes of Evacuation Behaviour

The MASSEgress framework [17, 23, 29] defines three classes of evacuation behaviour which will be considered in this project's behavioural model for agents. These are *competitive behaviour*, *queueing* and *herding*.

Competitive Behaviour

Competitive behaviour is observed when individuals attempt to force their own exit by competing with others. This includes pushing, moving at an unsafe speed towards exits, etc. Such behaviour generally reduces the efficiency of egress, especially at narrow doorways or other tight spaces[30], compared to exiting in a non-competitive (queuing) manner [31].
In general, this behaviour emerges when a person is highly stressed and perceives an urgent need to evacuate. However, an individual can also take up competitive behaviour as a result of social proof, if other individuals around them are acting competitively.

Queueing

This can be seen as the converse of competitive behaviour. It is characterised by a group organising themselves to facilitate efficient and safe passage through an exit.

Herding

Broadly speaking one can define herding as "the alignment of the thoughts or behaviours of individuals in a group (herd) through local interaction and without centralized coordination" [32]. Again social proof is a key contributor to the emergence of this behaviour. When an individual has a high degree of uncertainty they will follow the crowd.

Unlike other behaviours herding can be either adaptive or non-adaptive depending on the context in which it occurs. Suppose that an individual is following a crowd through an exit. It is possible that this exit is the correct way to proceed, so this choice has aided the person's egress. Equally, it may lead to a dead end.

4.1.6 Unifying These Concepts In A Behavioural Model

We have discussed some of the properties of an individual and groups (such as panic, urgency to exit, etc.) which affect the behaviours exhibited in the evacuation process.

For modelling purposes, these factors are unified into a single parameter which shall be referred to as 'evacuation stress'. This can be expressed as a percentage where 0 is equivalent to an individual being in regular day-to-day conditions, whereas 100 represents an individual in blind panic acting almost entirely on instinct. Of course, these two extremes are unlikely to occur in practice.

The reason for this extreme simplification is to limit the possible conditions which must be checked in decision making. The more attributes an agent possesses which may affect decision making there are, the more conditions must be checked at each decision making stage. Even a small number of variables can lead to an very large set of possible outcomes which all must be checked. This can

exponentially increase both the complexity of the behavioural model for the programmer and the computational complexity of making a decision.

4.1.7 The Perceive, Decide, Act Process

The decision making process for each agent can be expressed in three stages:

1. **Perceive**: the agent scans the area around themselves for visible goals or other information. This produces a set of goals in sight.
2. **Decide**: based on what they can perceive and their current level of evacuation stress, the agent chooses an action with which to proceed.
3. **Act**: the agent carries out this action.

An example of this procedure is as follows:

1. An agent scans the room they are in. They perceive two exits, one with only two people moving through it, the other with twenty people waiting at it.
2. The agent's current evacuation stress is high enough that they will engage in herding behaviour. They decide to move toward the most crowded of the two doors.
3. The agent plans their route to this destination and initiates the movement.

The implementation of this theory is discussed in the Implementation section 6.

4.2 Selection of 3-Dimensional (3D) Environment

The key decision before implementation could commence was the environment to work in. This was heavily interlinked with the choice of main programming language. Due to time constraints, building an entire 3D engine would be outside the scope of the project.

4.2.1 Programming Language Considerations

C++

Designed to be a superset of the C language, supporting the object-orientated paradigm. It is industry standard for almost all 3D graphics development [33], bridging the gap between lower level languages such as C and object orientated languages such as Java and C#. Some of the advantages of the language include its speed and power combined with the available 3D libraries and very high portability. The majority of available engines are also written in C++ offering a large selection.

However, as a team we had no experience at all with the C++ programming and at the time of selection only a minor knowledge of C. This alone would be a steep learning curve, but to provide the 3D environment required an understanding of either the OpenGL library or Direct3D (see Graphics Standards below) would have possibly also been required. There are also those who argue instead of trying to bridge gaps, low level components of games should be written in C. They argue that due to the information hiding afforded by C++ it is often easier to write efficient code in its lower level counterpart[33]. Memory management is also far less advanced than alternatives offering an easy leeway to memory leaks.

Java

Written to have as few implementation dependencies as possible, Java is incredibly portable[34]. Rather than to binary, Java compiles to what is known as byte-code which runs on the Java Virtual Machine, unrelated to the architecture underneath. The language as with C++ implements the object-orientated paradigm allowing abstraction and with standard extensions such as Java3D offers a less intimidating entrance into 3D.

With powerful free IDEs such as Eclipse and Netbeans writing in Java becomes quick and painless. Threading is built in and whilst large, the instruction set is easy to learn. Furthermore, Java is the language we as a team have the most experience with. This would allow implementation of the basic components to begin immediately. There is also the Java garbage collector. This automatically retrieves memory which is no longer reachable taking away risks of memory leakage.

The downside to Java is that the abstraction from architecture comes at a cost of speed. Since there is both virtualisation combined with a high level environment it is not as efficient when compared to other languages such as C[35].

Python / A high level scripting language

The majority of performance issues relating to 3D programming come from the underlying engine. Since we were anticipating usage of a pre-existing system, we could then build our system over the top of this using a high level language such as Python. Using libraries such as Boost.Python[36] the two languages can be bound, allowing claimed seamless interoperability. Performance bottlenecks due to Python could be overcome using C++. Less critical tasks could be written quickly and easily in Python.

Some of the team had some Python experience. This fact combined with the less intense learning path than pure C++ meant that this methodology was a strong consideration.

4.2.2 Graphics standards

Whilst this mostly came down to our choice of engine, it was a consideration to take during our decisions. There are two viable options, Direct3D and OpenGL.

Direct3D

Direct3D is a proprietary API (Application Programming Interface) designed by Microsoft Corporation. It was created to allow games creators more open access towards hardware giving much better performance. Whilst in previous years it suffered performance issues and multiple bugs it has improved drastically and is now considered by many to be the industry standard for Windows platforms[37].

Its power is also one of its key weaknesses. There is a steeper learning curve than OpenGL and it takes considerable work just to initialize. The other big weakness is portability. Support outside of Windows is extremely poor. Whilst Wine, a compatibility layer for Unix-based systems offer mostly functional ports, these are impeded due to dependencies on other Windows libraries.

OpenGL

OpenGL is an open standard API which was for a number of years little disputed as the industry standard. It is available on a large variety of platforms including Windows, Mac, and Linux based systems. It provides a strong range of functionality and was designed to be as future-proof as possible. There is a proven history of stability and to add to its core functionality there is the ability for extensions. Since its future is controlled by a board made up from a large diverse group of companies its strengths apply to a large number of applications.

Downsides are also many but revolve around two main issues. OpenGL was built 10 years ago and the future it was built to work for has arguably come and gone. Extensions go some way to remedy this but, many of these are vendor specific.

4.2.3 3D Engines

The majority of the 3D functionality we needed could be provided by an existing engine, either developed for simulation or game purposes. Concepts required are needed by a wide range of industries making current developments extensive and abundant.

Due to time constraints it would be difficult to create a bespoke system able to provide as detailed and efficient performance. As a result of this we decided to use one of these pre-existing solutions. This choice would be heavily interlinked with our preferences towards other tools.

Many of these are games engines. Games engines often are designed to simulate a real-world environment which offers exactly what is needed for this project. Due to the many options available, only a subset are mentioned below.

Unity

Unity is a cross-platform engine written in C/C++, however it also supports code written in C# and JavaScript. It has its own rendering capable of using Direct3D or OpenGL. There is strong support for 3D model importation from a large range of formats. It has its own scripting language as well

as supporting C# and Boo (which has a syntax inspired from Python). The basic license would provide all the features we needed and is free.

The emphasis is on providing incredibly powerful GUI design tools not games. The logic is aimed to be done entirely in scripting languages which would give performance issues when combined with the behavioural processing that would be required.

Whilst C# can be used, we have no experience with the language. C#, as with Java, offers performance shortfalls when compared to C/C++.

Panda3D

Panda3D is an open source framework for 3D rendering and development of programs written in Python and C++. It offers a reasonably powerful environment with a relatively shallow learning curve. There is a strong and active community support system but the documentation appears to be lacking compared to other alternatives.

It is cross platform among Windows, Apple Mac and Linux. It supports both OpenGL and Direct3D providing a relatively thin wrapper around the lower level APIs.

If the decision was made to take the Python and C++ route, this would be a strong option to consider.

jMonkeyEngine

jMonkeyEngine is designed partially as a games engine and partially as a replacement for the now deprecated Java3D. Written purely in Java all the advantages mentioned towards the language above would also apply here. All recent versions of OpenGL are also fully supported offering advanced graphics capabilities.

Fundamentally, the project is solely a collection of libraries making it a low-level tool which would give us the flexibility we would need considering the majority of our code would be related to the simulation as opposed to the graphics rendering.

If the decision was made to work in Java, then there would be no comparable competition.

CryENGINE 3

CryEngine is an advanced engine created by Crytek originally as a technology demo for Nvidia but the company soon saw its potential. This has led to massive success with a burst of successful high profile games based on the engine.

Programming for CryEngine is accomplished using C++. This gives an extremely powerful combination allowing incredible graphics with a high performance back-end.

The big downside is the lack of support for OpenGL. As a result there is little portability outside of Windows. It is also only free for non-commercial use, meaning if the project were to be taken beyond the initial research aims an incredibly expensive license would be required.

Game Blender

Blender is a free and comprehensive 3D production suite, one component of which is a games engine. Considering Blender was a strong contender for use in our modelling (see Modelling below) there would be no importation issues. The engine is a mostly independent component written in C++ including support for Python scripting. Whilst a relatively young project, it offers all the 3D functionality that would be required, however is lacking in the back-end code support which would be needed.

4.2.4 3D Modelling

To manipulate a 3D environment, such an environment must first be created. This involves modelling the chosen structure in a way that could be imported into the physics engine. Since many of the engines considered included modellers, the decision of what 3D modelling tool to use is linked to the decision of what graphics environment is chosen.

Blender

As mentioned above, Blender is a comprehensive 3D production suite. Its main usage is in creation of 3D models. A large number of the games engines we were considering either supported Blenders native .blend format, or one of the many alternative formats the suite could export as. There is extensive documentation combined with a strong and active support community.

The feature-set offered is comparable with some of the widespread industry tools. Released under a GNU General Public License the software is free to use.

Autodesk 3DS Max

3DS Max is an incredibly powerful suite and the one most used in the modelling industry. It is comprehensive and versatile but the features come at a cost. Licenses are extremely expensive and the system requirements are significant. Whilst the license cost can be avoided since free versions are available for students, one would need to gain access to hardware capable of handling the software.

Hexagon

Hexagon, unlike the other options discussed is purely a modelling program. Offering equal functionality in this area, advanced and detailed models can be built. The interface is intuitive and easy to learn. The software also has a very low retail price.

4.2.5 Choices

Whilst the Python and C++ routes were a strong consideration it was decided the benefits failed to overcome the time required to learn a completely new language. Although the documentation was not perfect, jMonkeyEngine appeared to offer all the features we required. It was decided that, provided the back-end code was written in a reasonably efficient manner, performance should not be an issue.

Since jMonkeyEngine offered an inbuilt importer, Blender was a complimentary modelling choice. The minor benefits offered by the proprietary solutions were far from counterbalancing the restrictions licenses would incur.

Chapter 5

Design

5.1 Population

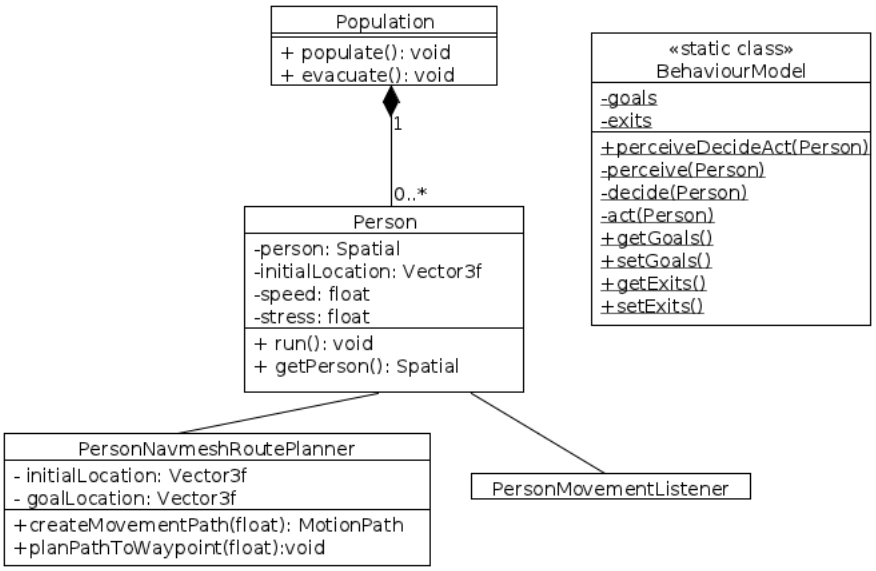


Figure 5.1: Population Package

Figure 5.1 shows the relationship between the various classes responsible for the realisation of autonomous agents and their behaviour within the simulator.
The Population class represents the set of agents. A Population instance is ran as a separate thread of execution which incrementally performs computations on the entire set of agents. It is also responsible for the generation of the set of agents using the information provided in a set of PersonCategories (not shown) and the initiation of agent evacuation.
BehaviourModel is a static class designed to implement the Perceive, Decide, Act process (see Section 4.1.7. For each step there is a corresponding private method which is called. The output from the perceive method, a list of goals visible to the agent, is passed to the decide method which chooses a new target location for the agent to plan a path towards, and this location is passed to the act method which initiates the agents movement towards its new goal.

The seperation of these three stages is important to maximise the extensibility of the behavioural model. As mentioned, the decide method processes a set of goals and, based on a set of rules and heuristics, chooses a new location for the agent. To change or extend the range of decisions an agent can make, the programmer can simply add new rules to the decide method. In fact it is possible that multiple decision methods could be added where each implements a different behavioural model. The only restriction is that the decide method must take a set of Goals (see below) as one of its inputs and must output a 3D point. By calling the perceiveDecideAct method, an agent is taken through all three stages of the process and need have no knowledge of the decision algorithms used to select this new goal.

An agent is primarily realised by the three classes: Person, PersonMovementListener and PersonNavmeshRoutePlanner. Each Person holds the attributes, including speed and stress (see Section 4.1.6) which represent the characteristics of an evacuee. It also coordinates the visual representation of an agent and the movement of the agent towards its goal.
In order to plan a route to a location, a Person must create a PersonNavmeshRoutePlanner instance. Its purpose is to act as if it were a 'ghost' agent: it rapidly traverses the navmesh to establish a route. This route is then returned for the agent to move over. A fresh instance of PersonNavmeshRoutePlanner should be used every time a new route must be calculated and can be disposed of once the route has been returned.

5.2 Goals

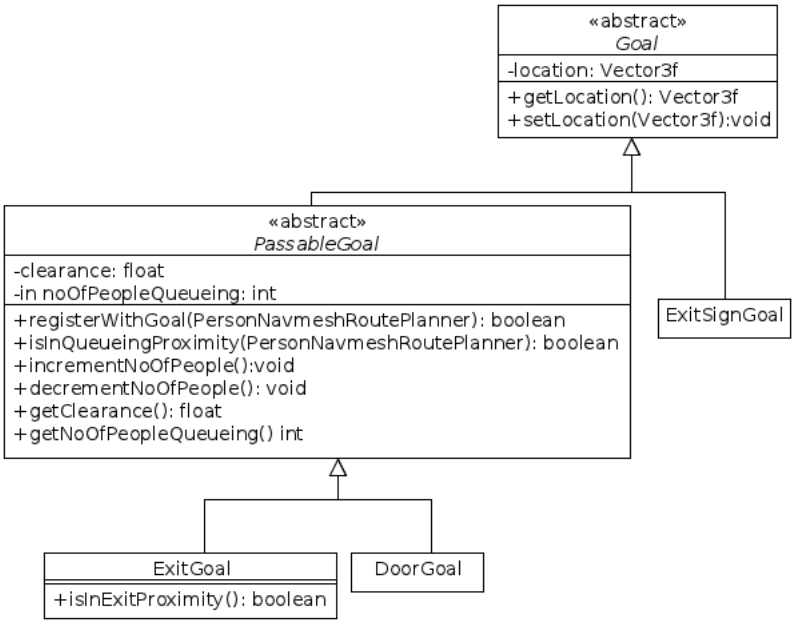


Figure 5.2: Hierarchy of goal package.

A goal can be defined as anything in the environment which can steer an agent's direction. Figure 5.2 shows the inheritance hierarchy for goals in this simulation. At the top level we define a generic Goal, which simply has a location and does not direct agents explicitly.

A PassableGoal is any goal through which an agent can move, such as a door or exit. These also have a clearance: a radius around its center which defines the maximum distance at which an agent can be considered to be queueing at that goal. This is vital for the computation of queueing behaviours. The number of people queueing at a PassableGoal is also stored and must be updated externally by agents as they approach or leave the goal. Exits are represented by ExitGoals which are equipped with methods for recognising when an agent is close enough to exit the simulation. ExitSignGoal is an example of a non-passable goal which could direct agents to another goal. However non-passable goals are out of the scope of this project. ExitSignGoal is shown here purely to demonstrate that the existing framework could be extended to include such goals in future work.

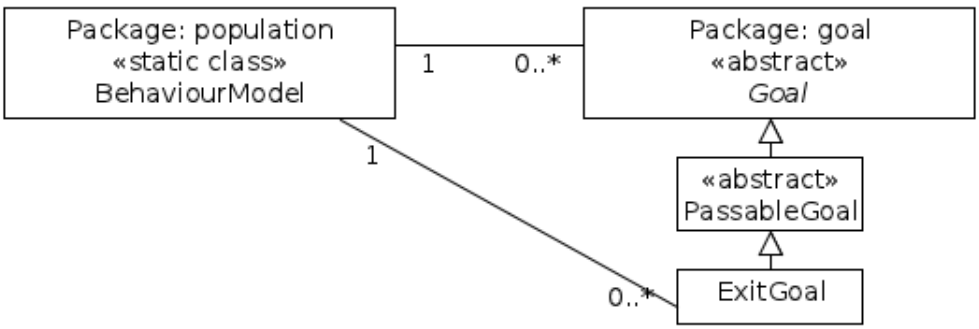


Figure 5.3: Relationship between BehaviourModel and Goals

As Figure 5.3 shows, the static BehaviourModel instance holds a collection of all Goals in the simulation. For the purposes of calculating agent's paths out of the environment, a collection of all the ExitGoals is also kept separately, since it is anticipated that these will be used most frequently in calculations.

5.3 Using the 3D Model

5.3.1 jMonkey Model Import

Once the 3D model of the evacuation environment has been completed using Blender, it needs to be converted to a logical data structure (in this case, a Navigation Mesh) that can be used by the project for path-finding purposes. The first step of creating this structure is to convert the Blender file into a jMonkeyEngine (jME) compatible file. This process can be done automatically with the help of the jMonkey Integrated Development Environment: the file created in Blender is converted to a binary encoding of the mesh, which can be interpreted by jME. Once created, this file is permanently stored as a project asset and can be used whenever the user wishes to simulate an evacuation on that specific model. The next step in creating the compatible data structure is to read the binary encoding of the environment and transform it into an initial jME mesh. This is a geometric mesh consisting of collections of three types of geometric primitives:

- **Points**: Holds a vertex representing a single point in space
- **Lines**: Two vertices representing a line segment
- **Triangles**: Three vertices representing a solid triangle primitive

In this particular case, the jME mesh will hold the environment's geometric information using a list of 3D triangles, that were mapped from the imported model's surfaces.

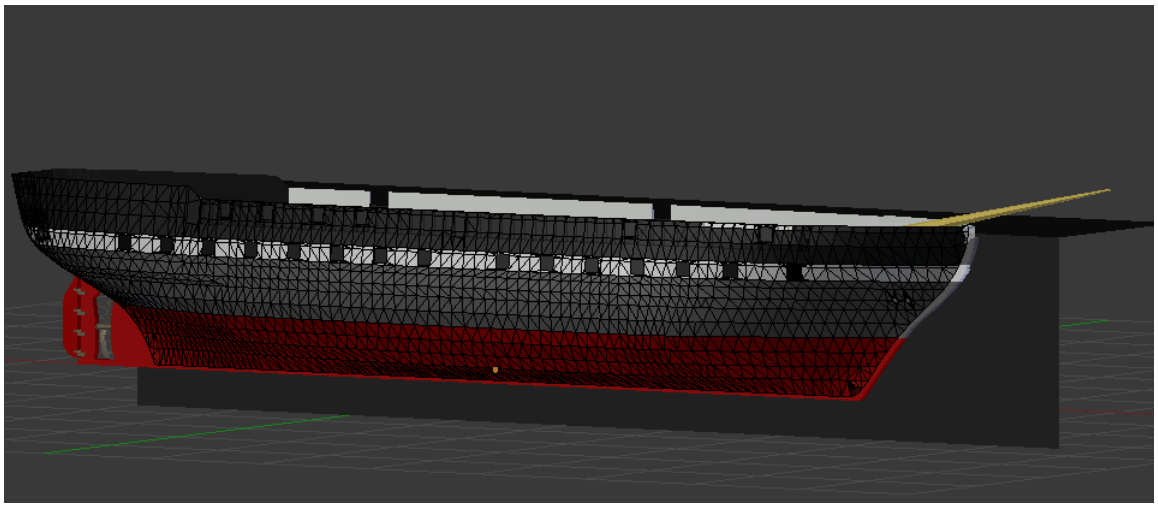


Figure 5.4: View of the Blender model

5.3.2 Choice of data structure

In order to simulate a person's movement in the environment, we need a way to store and use information about its walkable surfaces. A matrix-type structure, or a grid, is easy to visualise and implement. Surfaces are divided into tiny cells of the same size, each holding information about the type of surface it maps, and links to its neighbours. However this is costly in terms of memory consumption and processing power, as we will need a very large amount of cells for reasonably large environments.
A more efficient alternative are Navigation Meshes, as convex polygons are used to cover the walkable surface areas. This way, large open areas can be covered by a handful of polygons, greatly reducing the amount of memory needed. Since each individual polygon can still be considered a node, holding references to its neighbours, the whole structure can be used as a graph, which will later allow us to use efficient pathfinding algorithms, such as Dijkstra or A*.
Since 3D environment models contain information about more than just the walkable surfaces, a series of algorithms have to be used to extract and filter it into a navigation mesh.

5.3.3 Conversion to a Navigation Mesh

In order to allow agents to navigate over a mesh, a collection of the 3D surfaces (in this case, interconnected triangles) can be used. Any point interior to these triangles represents a valid location at which an agent can be positioned at any point in time. Furthermore, the agent can move from any

point inside the triangle's surface to any other point inside the same surface.
Each of these triangles can have references (links) to zero to three other neighbouring triangles, one for each edge of the triangle. If a link exists between two triangle (i.e. they share a common edge), this means that an agent can traverse from one triangle to the other or vice-versa. As a result, a collection of such triangles, or cells, is a convenient structure on which to use graph path-finding algorithms: every cell can be interpreted as a graph node, and any link to other cells as a vertex between two nodes. This whole collection of interconnected triangles (cells), which can be abstracted as a graph, is called a Navigation Mesh (navmesh).
The Recast[38] project is an open-source tool-set focused on the creation of navigation meshes from geometry meshes. The jMonkeyEngine's navmesh package, which is based on Recast, was used in order to convert the project's imported environment models into agent-navigable structures.
To create a navigation from an existing geometry mesh, a series of sequential processes is required. A summary of these processes and their parameters and results will be discussed next, based on the NMGen study [39].

1. **Voxelisation**: Create a solid height-field from the source geometry, made from a collection of voxels. A voxel (volumetric pixel) is a 3-dimensional, box-shaped unit used in representations of 3-dimensional images;

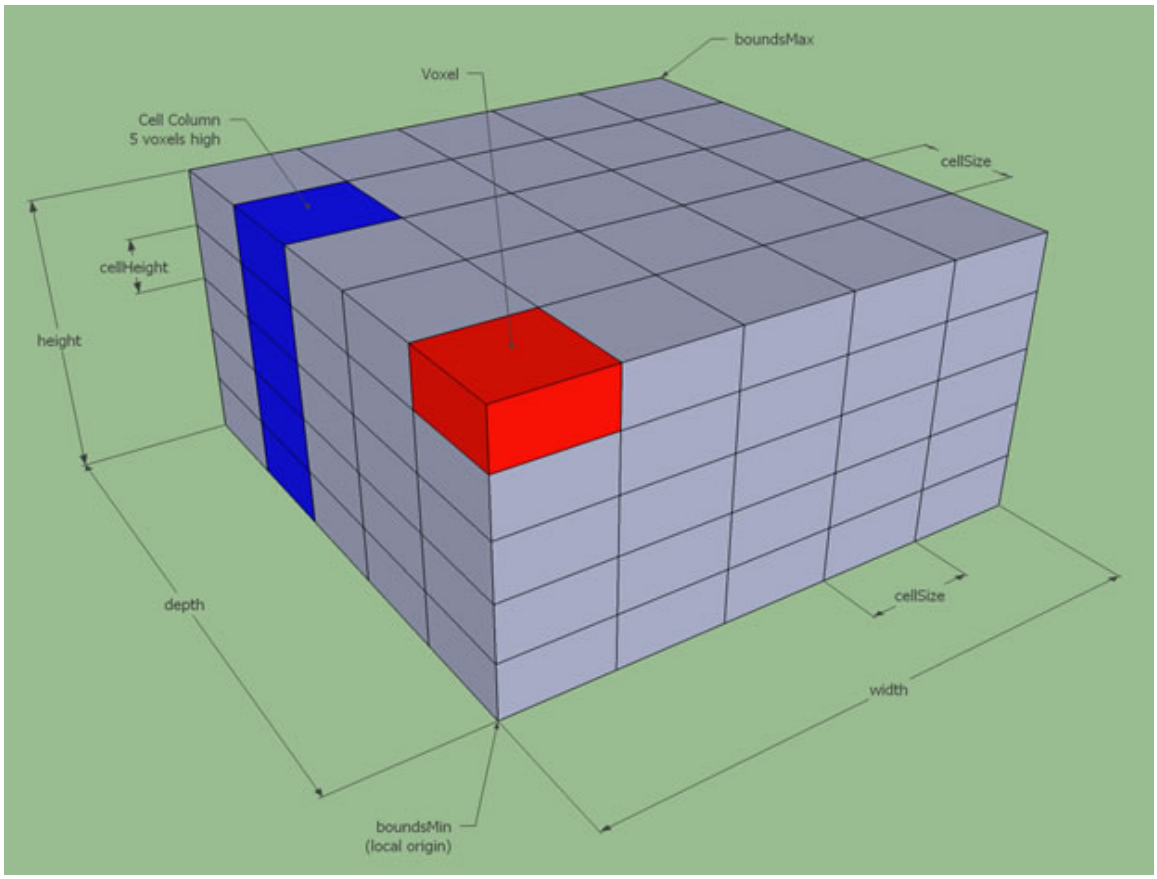
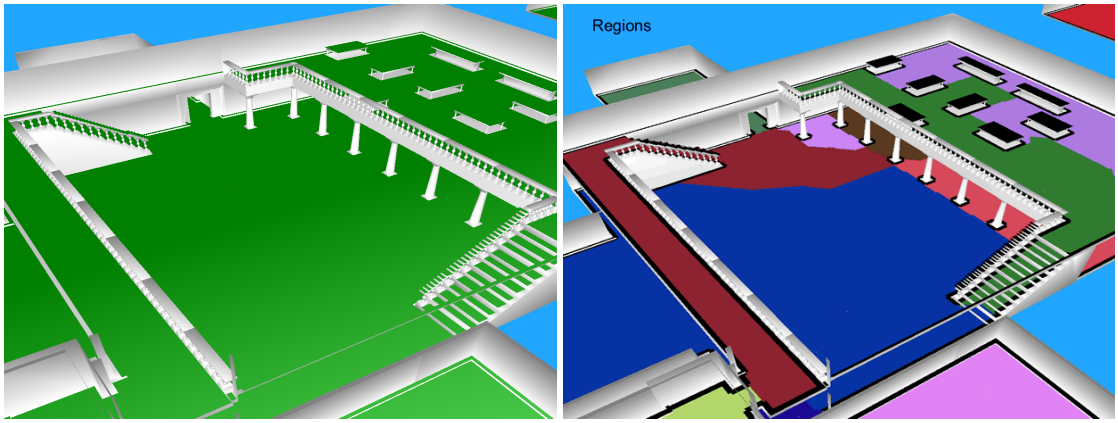


Figure 5.5: A voxel grid

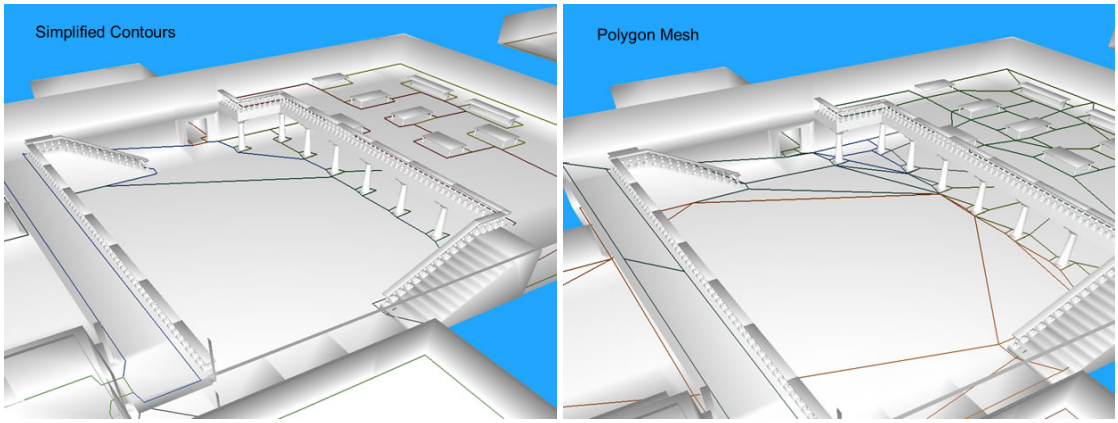
2. **Region Generation**: Detect the top surface area of the solid height-field and divide it up into regions of contiguous spans;



(a) Overview of the model (b) Model split into regions

Figure 5.6: Region generation

3. **Contour Generation**: Detect the contours of the regions and form them into simple polygons;
4. **Convex Polygon Generation**: Sub-divide the contours into convex polygons;



(a) Contour generation (b) Convex polygon division

Figure 5.7: Region generation

5. **Detailed Mesh Generation**: Triangulate the polygon mesh and add height detail.

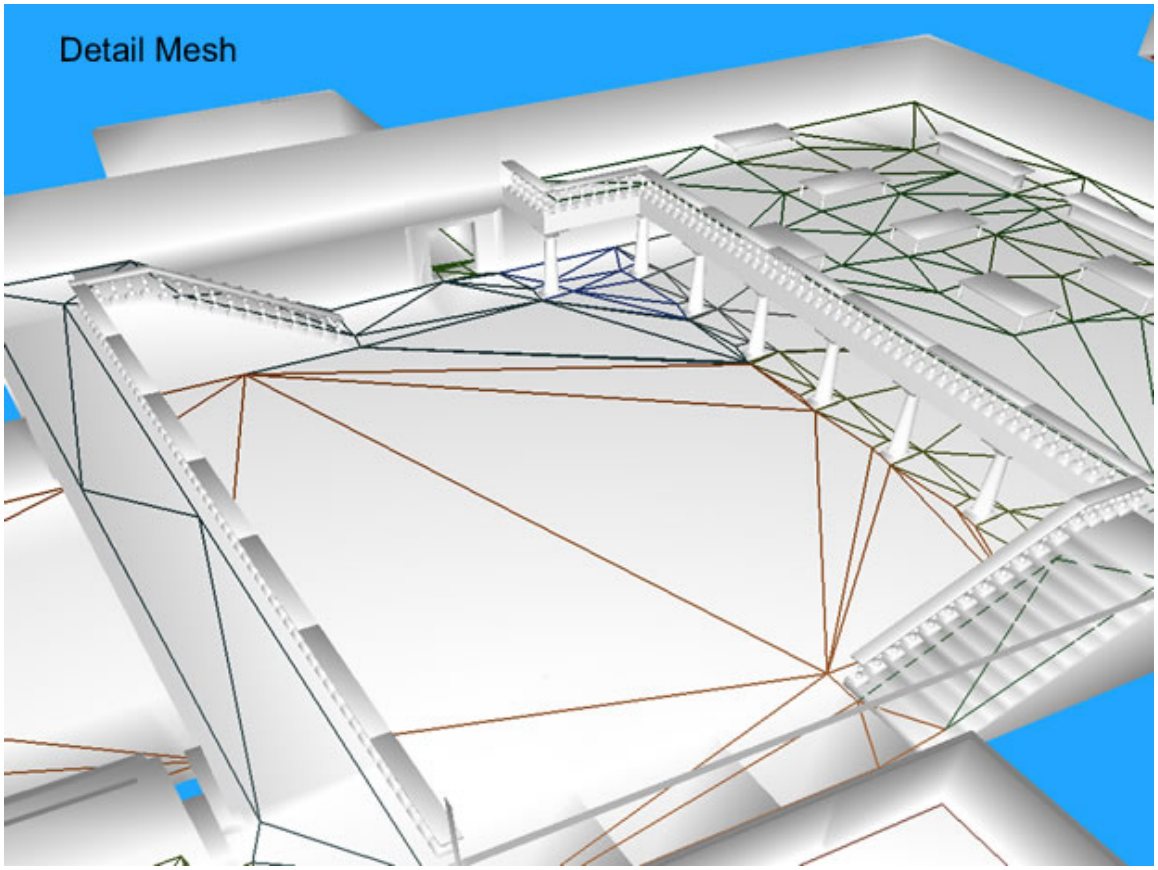


Figure 5.8: Detailed mesh

In order to obtain a well-formed mesh, several parameters are required to be passed to the generator (note: where parameters are dependent on variables like agent surface area or agent height, the corresponding values should be calculated in relation to the scale of the imported model):

- **Cell size**: Represents the width and depth of the voxel units which will fill the source geometry. This parameter will influence the accuracy of the generated navigation mesh in relation to the supplied geometry mesh. Lower values closely make the result closely match the source geometry but with a proportional increase in computation time and space costs. In order for the navmesh to be well-formed, this parameter needs to be at least several times smaller than the size of an agent's surface area.
- **Cell height**: Represents the height of the voxels used to create the solid height-field. Like the width and depth, the height of the voxel influences the level of accuracy of the navmesh. A well-formed navmesh requires this parameter to be at least several times smaller than the height of an agent's maximum step (step-height).

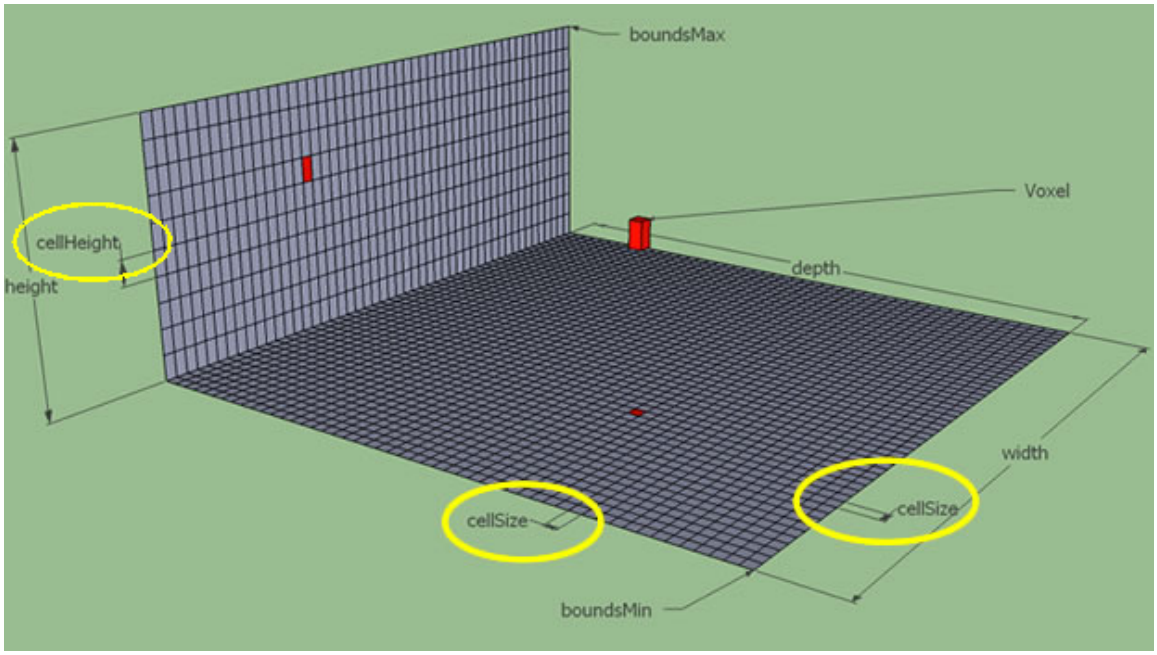


Figure 5.9: Cell size and height

- **Minimum traversable height**: Represents the minimum distance from the floor to the ceiling that will still allow an agent to pass through. Should be at least the size of maximum agent height. Should also be at least two times the height of a voxel (cell height).

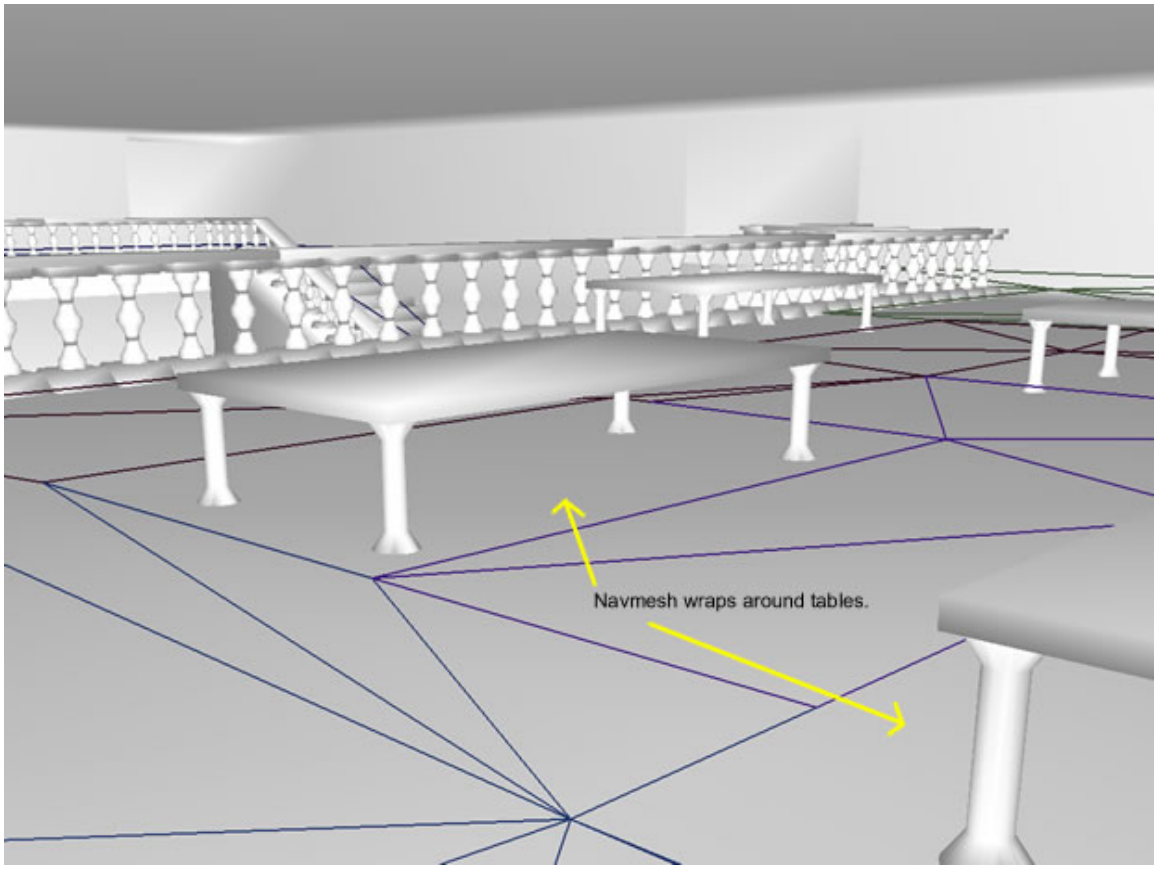


Figure 5.10: When minimum traversable height set correctly, the mesh does not flow under the tables

- **Maximum traversable step**: Represents the maximum height of a ledge that can be climbed by an agent. Should be at least twice the height of a voxel.

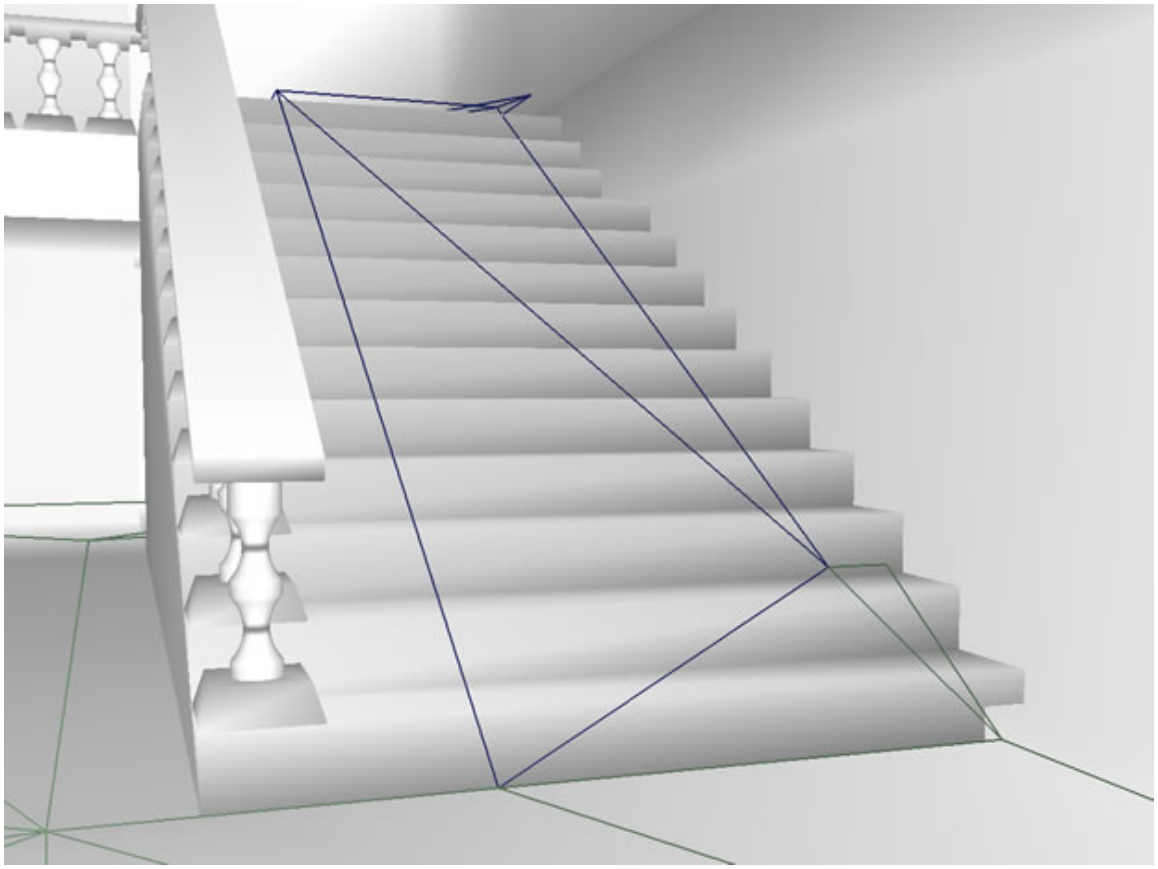


Figure 5.11: Setting a correct value for the maximum traversable step, elements such as stairs can be detected and made walkable

- **Maximum traversable slope**: Represents the maximum slope of a ramp that is still deemed traversable. Any ramps with a higher slope will be considered unwalkable and will be considered an obstacle by the navmesh.
- **Border size of traversable area**: Represents the distance from the walls to the actual walkable area. For a well-formed navmesh, this should be at least the same size as an agent's surface area.

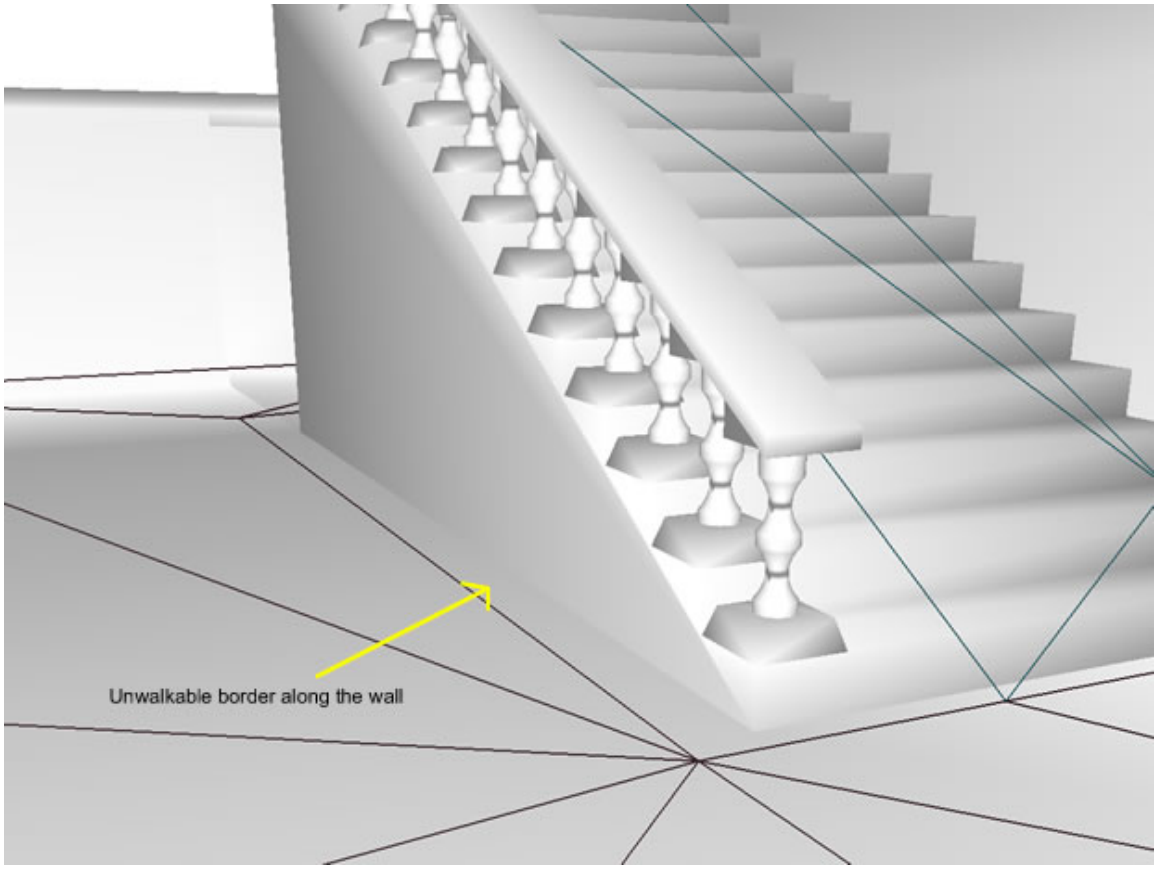
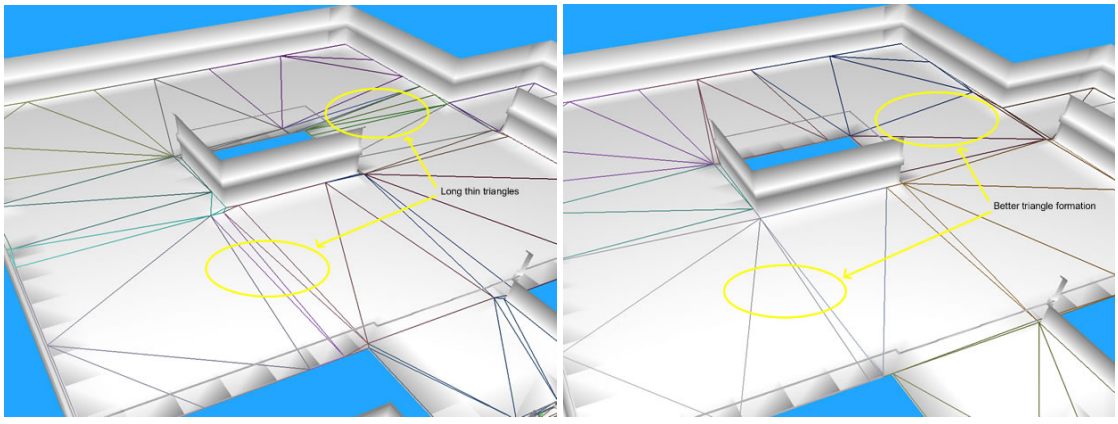


Figure 5.12: The walkable surface will be limited to at least the size of the border. Setting the border size to at least the size of an agent's surface area enables the agent to be positioned on any point of the mesh, without worrying of contact with the walls

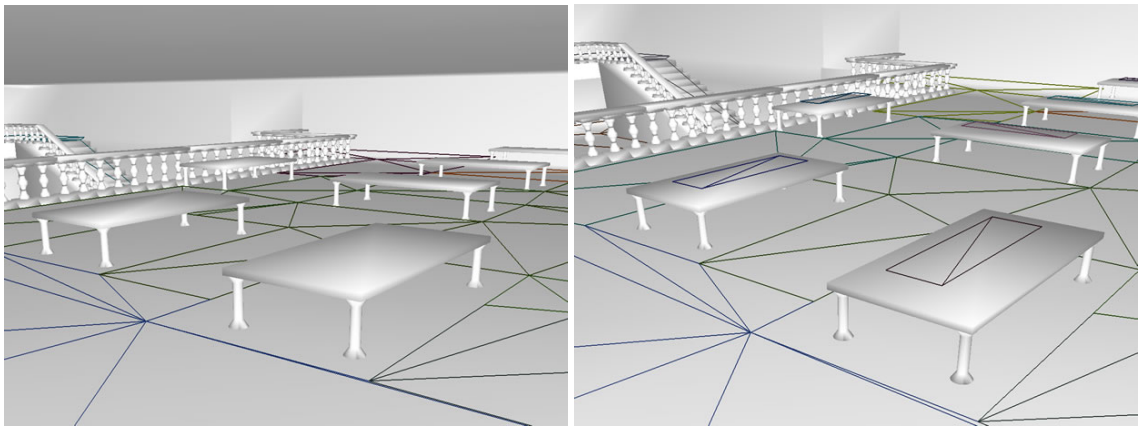
- **Smoothing threshold**: The amount of smoothing (larger region size, fewer thin triangles) to be performed when generating the distance field



(a) Smoothing disabled (b) Smoothing enabled

Figure 5.13: Smoothing threshold

- **Minimum size of unconnected region**: Represents the surface, in voxels, of regions that should not be added to the navmesh if they are unconnected to other regions (i.e. islands).

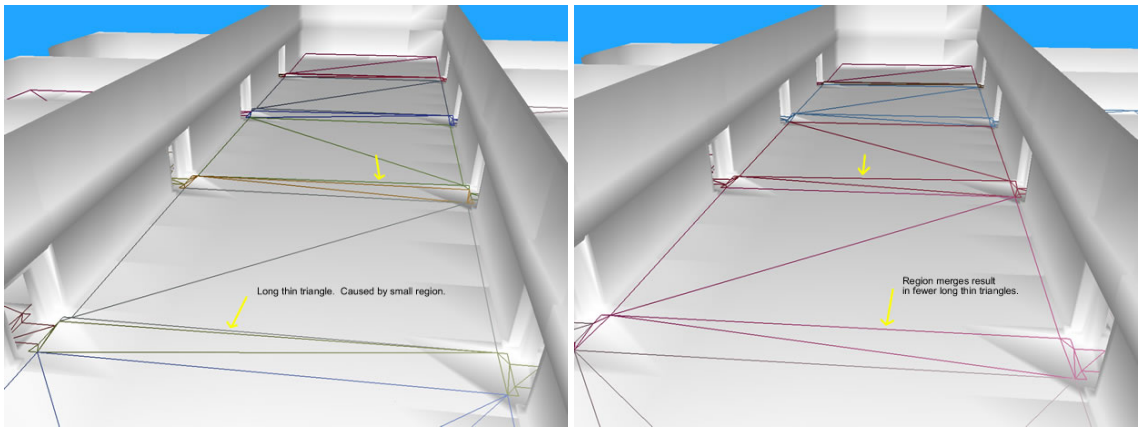


(a) A correct value for the minimum unconnected region size does not create a mesh surface for the table tops

(b) Setting the value of the minimum unconnected region size too low, the table tops are interpreted as walkable surfaces

Figure 5.14: Minimum unconnected region size

- **Merge region size**: Represents the minimum number of voxels a region should have in order to not try and merge it with other adjacent regions. This influences the number of long thin regions.



(a) With merge region size set too low, the surfaces can be covered by a large number of long thin triangles

(b) Setting the merge region size to a large enough value, some of the thin triangles are merged together with adjacent triangles (where possible)

Figure 5.15: Merge region size

- **Maximum edge length**: Represents the maximum length of a triangle's edge. If a triangle has a greater length, it will be split into several legal triangles, by adding more vertices on the border edges.

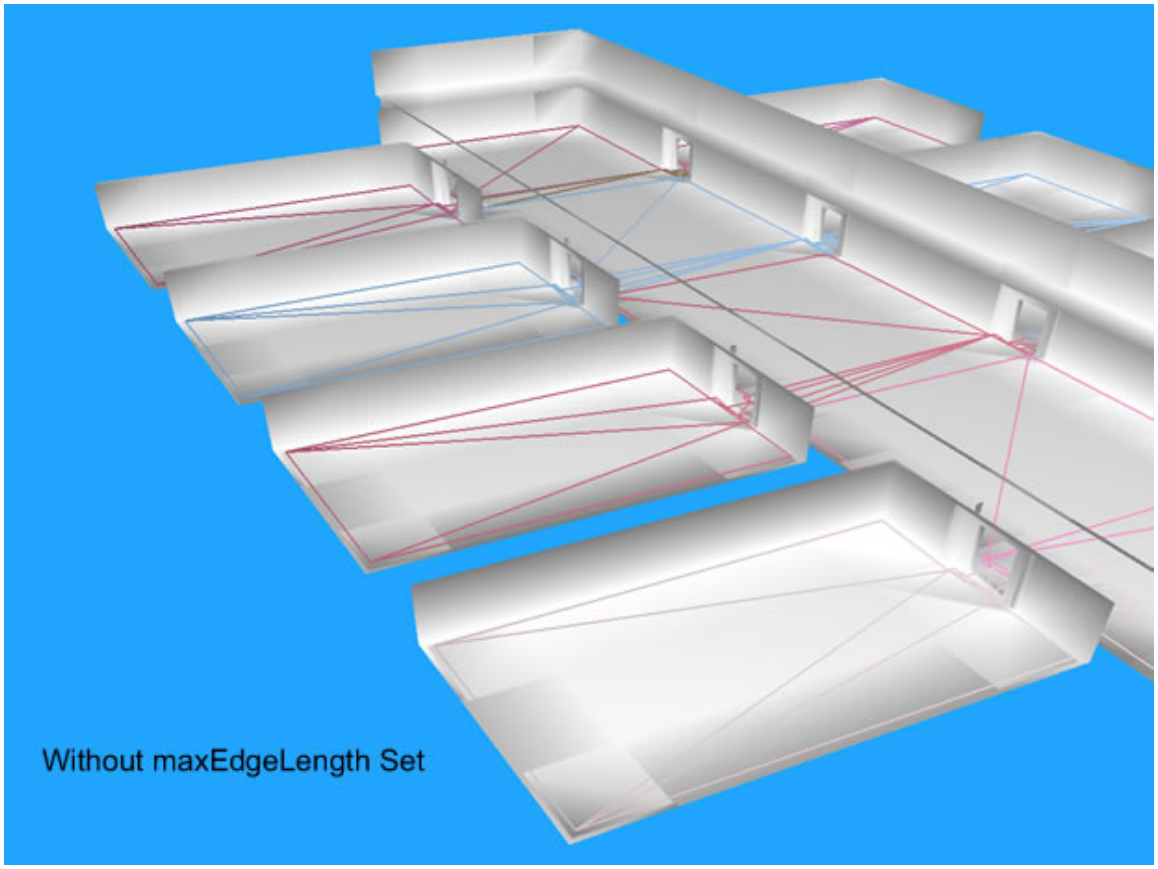


Figure 5.16: With the maximum edge length setting disabled, very long triangles can be generated

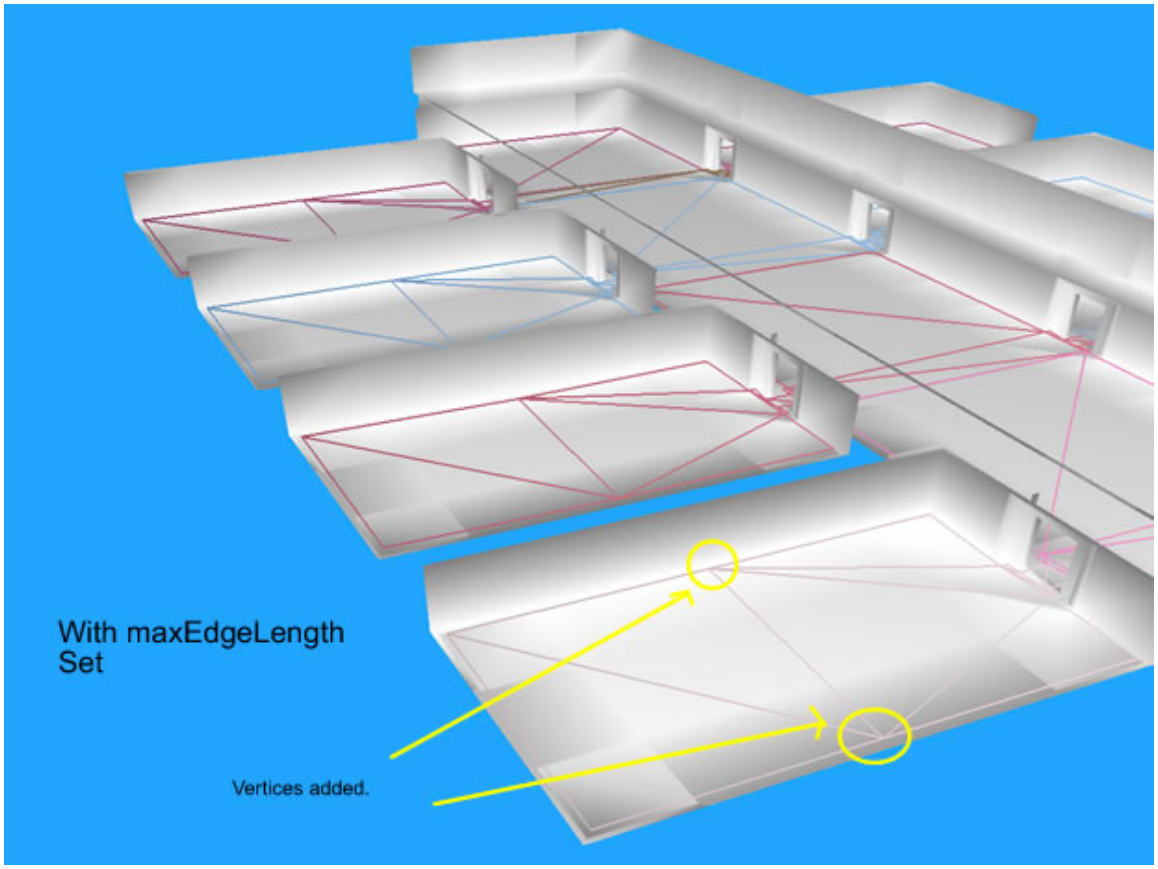


Figure 5.17: With the maximum edge length setting enabled, more vertices are added along the edge of the initial long triangles

- **Edge maximum deviation**: Influences the accuracy of the edges following the source geometry. The lower the value, the higher the accuracy, but at increased triangle count and processing cost.

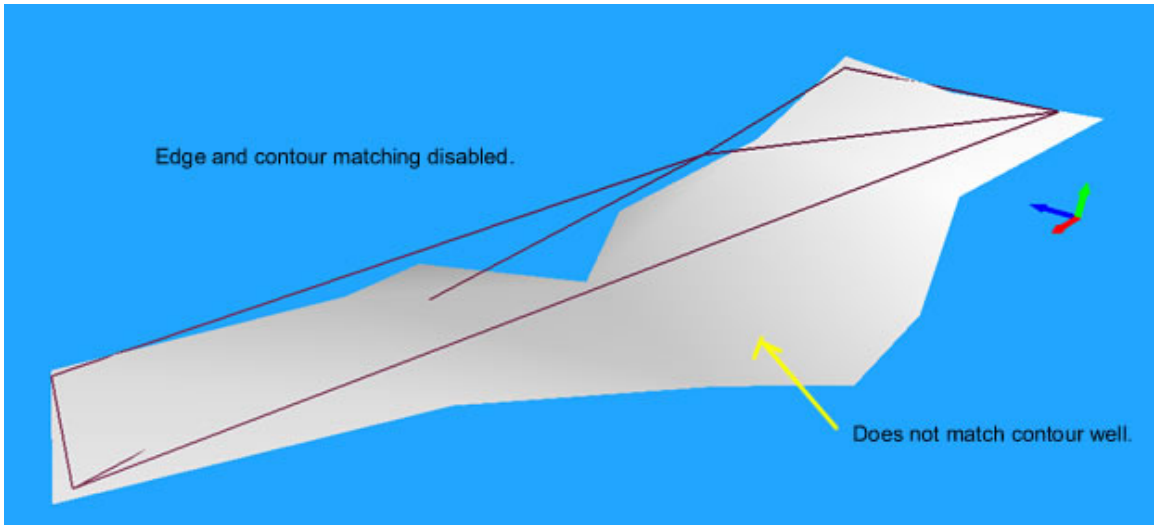


Figure 5.18: Edge maximum deviation disabled

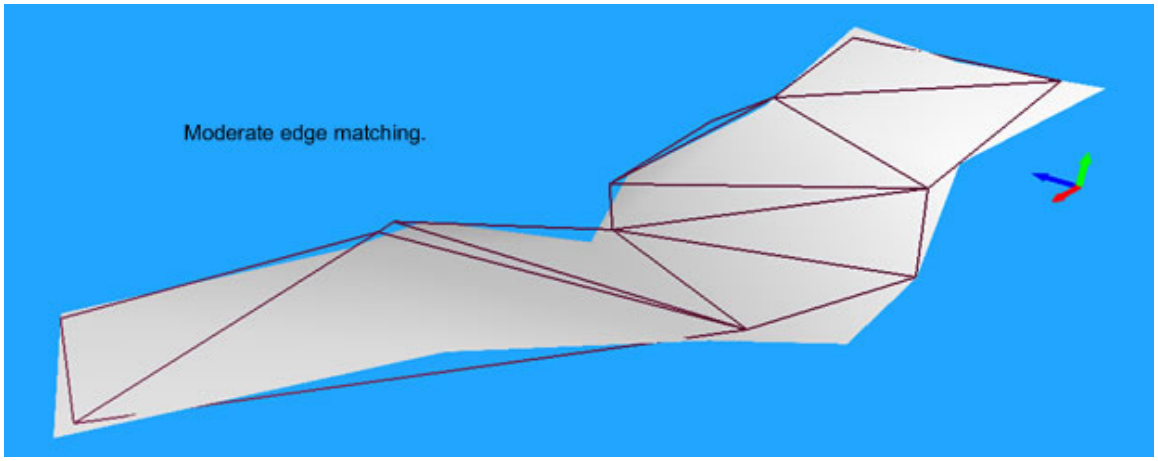


Figure 5.19: Moderate edge matching

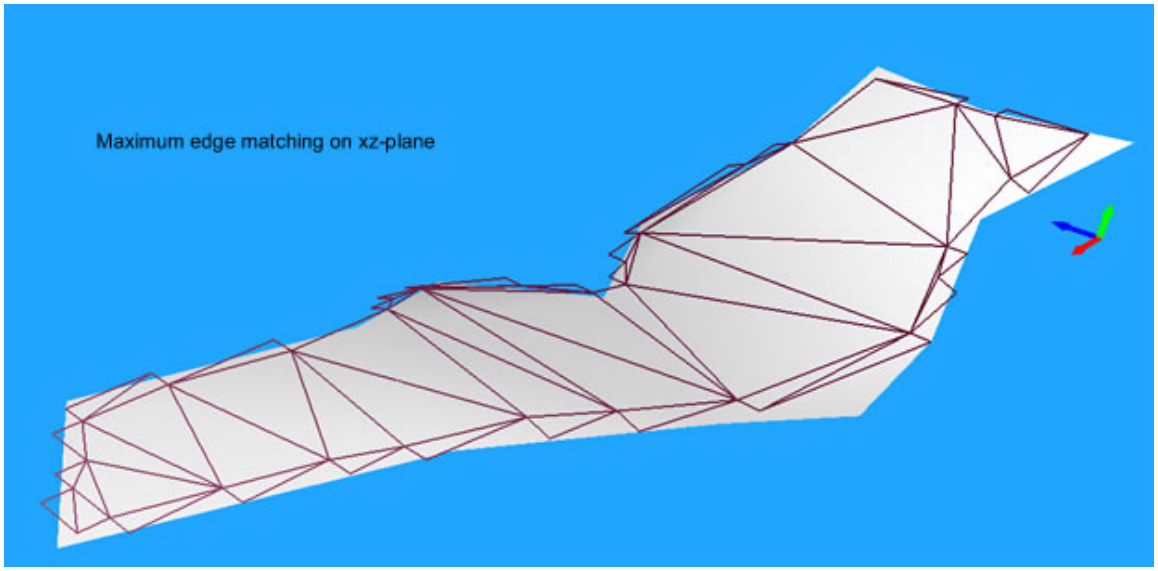


Figure 5.20: Higher fidelity edge matching

- **Max vertex per polygon**: If set to higher than 3, increases the cost of the computation but can result in better formed regions. However, for the current project, a value of 3 should be used, since the algorithms designed for path-finding only support 3-sided cells.

The problem that arises with such a high number of variables is the fact that end-users will not be able to import new models from different evacuation environments without a significant overhead. If wrong parameters are used for the mesh generation, the set of algorithms used might generate badly formed meshes, which do not map the source environment closely enough or even fail all-together. Still, with proper scaling of the model, either in Blender stage or after importing the geometrical mesh, importing models of different magnitudes can be achieved.

Since the space available for this report is limited, navigation mesh generation was presented only succinctly. For more in-depth information about each of the steps taken and how the previously

discussed parameters influence the resulted mesh, the reader can refer to the NMGen Project documentation [39].

5.4 Navigation

5.4.1 Logical structure

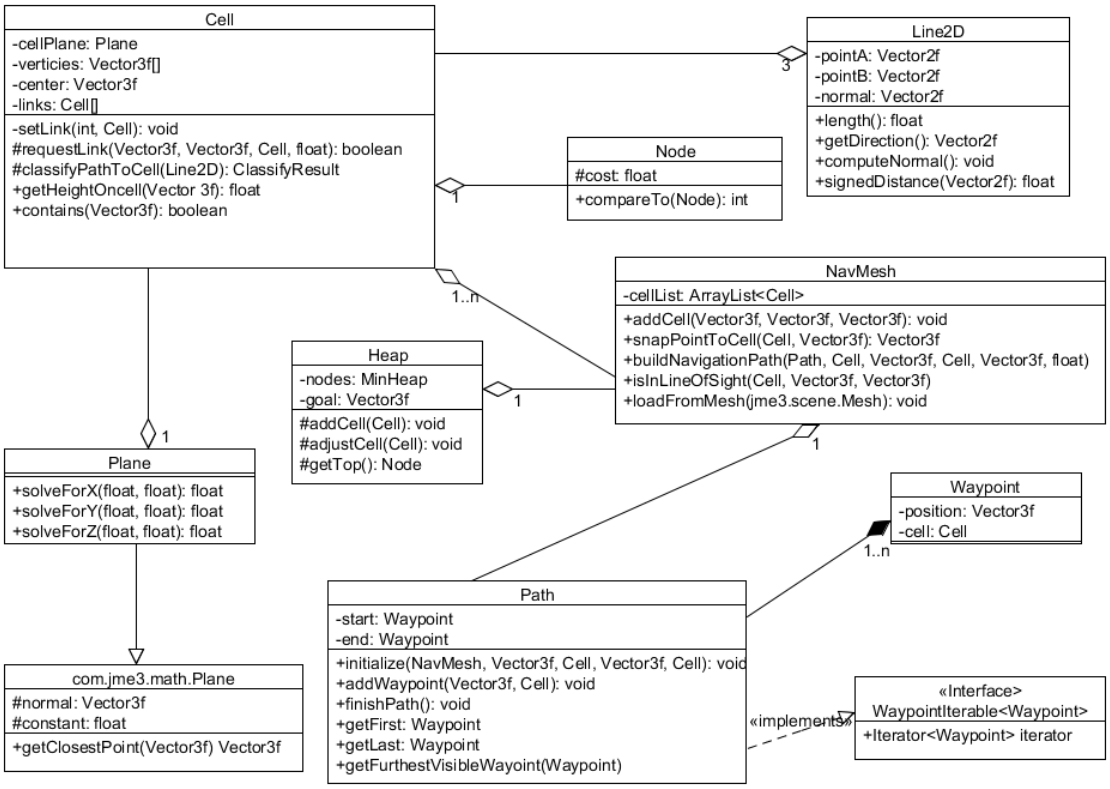


Figure 5.21: Navmesh Package

Figure 5.21 shows the relationship between the various classes responsible for path generation. The NavMesh class contains a list of Cells, each being a triangle representing a piece of the surface mapped by the mesh. Each cell has references to its three vertices (of Vector3f type), its three edges (of Line2D type) and its corresponding neighbours, or links (of type Cell) - three or less, depending on the number of neighbours. Each cell also has a corresponding plane reference, used to help with computing 3D calculations. The NavMesh initialisation is made by calling the loadFromMesh method, with the mesh resulted from the previous generation step as a parameter. As a result, the cell list will be populated with the relevant data.
Upon creation, each agent object representing a person will be assigned a new Path. By initialising the path with the relevant NavMesh reference, each agent can then independently use the reference to call the NavMesh's buildNavigationPath method. While calling this method, the Path in which to write the result is passed as a parameter, along with the start-point vector and end-point vector, the respective cells in which each vector reside, and a floating point number representing the size of

the increments in which to place waypoints. More details about the way the path is calculated will be provided in the next section 5.4.2.
Once the buildNavigationPath has returned control to its caller, the Path passed as a parameter will contain a list of Waypoints - a tuple of intermediary vectors, in increments of the floating point number provided, along with the cells in which each vector reside. Since Path implements the Iterable interface, the agent owning it can simply iterate over the waypoints contained and move along them sequentially.

5.4.2 Pathfinding

The buildNavigationPath method from the NavMesh class is in charge of the actual pathfinding routine. Since the navigable surface of the environment is mapped into cells which link to their neighbouring cells, this data structure can be regarded as a non-oriented graph. Therefore, graph pathfinding algorithms like Dijkstra[40] or A*[41] can be used. While the number of agents required for the initial environment is small (200), A* would be a more scalable option, being faster than Dijkstra thanks to heuristics which attempt to preempt the optimal path.
The pathfinding method used in buildNavigationPath is using A*: a priority queue structure (Min-Heap) is used to store intermediary paths from the source to destination, sorted by their lowest expected cost. The heuristic function used in the A* implementation is the euclidean distance from the center of the current cell to the goal. This distance is added to the total cost of the route up to the current location, and based on these data for each of a current cell's neighbours, decisions are made to consider the cells for further paths or ignore them.
Once the calling agent receives control back from the method, the Path passed as a parameter will hold a valid route from the agent's position to its destination, and can be used for scheduling movement.

Chapter 6

Implementation

6.1 Route Planning

Planning an agent's route from its current point to a given point is acheived by a collaboration between a Person and PersonNavmeshRoutePlanner instance. Routes are stored using the jMonkeyEngine class MotionPath. This contains a set of waypoints and methods to animate the agent's movement between these waypoints. The maximum distance between each waypoint is roughly constant and can be set as a parameter to the following procedure (it is not constant when the agent must turn at a point closer to its current position that the maximum distance between points). The distance used is an important decision in the implementation. If it is too large the accuracy of the animation tends to degrade. If it is too low then a large number of waypoints will be stored needlessly. After extensive experimentation 0.5 was found to be an acceptable value for the maximum distance between waypoints. In practice this provided a balance between quality and efficiency.
A route is calculated in two stages:

1. A path is calculated on the navigation mesh using a modified A* algorithm to traverse the mesh like a graph. This returns a small set of points. These points illustrate the lines of motion an agent must take to reach their goal. This is performed in the constructor for a PersonNavmeshRoutePlanner.
2. Using these points as guidance, the path is 'fleshed out' by moving along the path and placing MotionPath waypoints no further apart than the defined maximum distance. This terminates with a waypoint being placed on the goal location the agent must reach.

Note that a fresh PersonNavmeshRoutePlanner instance must be instantiated to calculate a route. The relationship between the Person and PersonNavmeshRoutePlanner instances is expressed in **??**.

6.2 Implementing the Perceive, Decide, Act Process

Here we discuss the techniques and algorithms used to realise the Perceive, Decide, Act process previously discussed in Section 4.1.7.

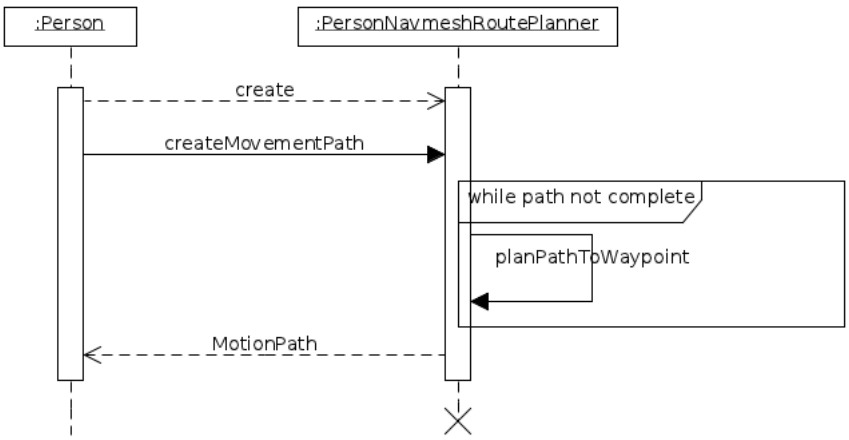


Figure 6.1: Generation of an agent route as a MotionPath

6.2.1 Perception

Agent perception is achieved using the following simple algorithm:

input : The set of goals in the environment: *goals*
output: A set of goals visible to the user at the given instant in time: *visibleGoals*

for *each goal g in goals* **do**
    if *g is in line of sight of agent* **then**
        add g to visibleGoals;
    end
end

Algorithm 1: Agent Perception Algorithm

6.2.2 Decision

In the Research Chapter [Reference] the three classes of human behaviour considered in the scope of this project were defined. In practice, only herding behaviour implemented as part of the Decide step; queueing and competetive behaviour must be handled asynchronously using collision avoidance techniques. Unfortunately collison avoidance was not implemented in the final system due to various problems discussed in Section 8.2.1. Future solutions to implement collision avoidance are discussed in Section 8.2.2. To decide on a new action, an agent must select one of the visible goals that were found in the Perceive step (if any). Otherwise it must continue on its current path. The decision making process is realised using an extendable algorithm, which sequentially considers sets of different classes of goal according to their priority. Exits have the highest priority.

The final algorithm only makes decisions regarding exits, due to various issues which emerged during development (see Section 8.2.1). However it is easy to extend this process to include other goal types by adding further conditionals following the pattern laid out below.

```
input  : Person person, ExitGoal[ ] exits /*add further exit types here as arrays */
output: Target Goal for agent to move toward
if no of exits > 0 then
    ExitGoal currentExit = exits[0];
    if person is stressed then
        for each exit e in exits do
            if number of people queuing at e > no. of people queueing at targetExit then
                targetExit = e;
            end
        end
        return targetExit;
    else
        Vector3f position = person.location;
        for each exit e in exits do
            if distance to e < distance to targetExit then
                targetExit = e;
            end
        end
        return targetExit;
    end
else
    return null;
end
```

6.2.3 Act

This step's representation in the BehaviouralModel class is trivial since the Decide step already returns a target goal. It is left in as a place for performing any calculations which should be performed before returning the target goal to the agent.
Upon receiving a new target goal, an agent should perform the following:

- Use a PersonNavmeshRoutePlanner to calculate a route to this goal
- Set the returned MotionPath as the current MotionPath for the agent
- Begin moving down this path

6.3 GUI Design and Implementation

The target userbase for the evacuator does not guarantee a good degree of computer-literacy. Hence a clear and easy to use GUI was required.

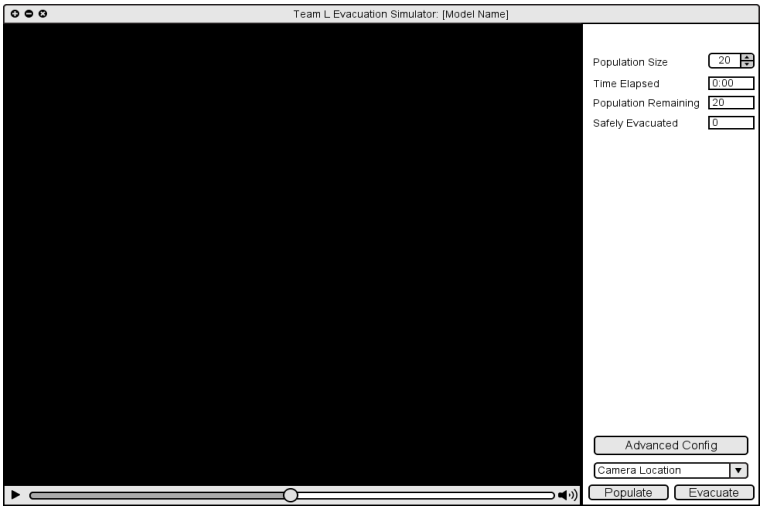


Figure 6.2: First attempt at wireframing GUI

6.3.1 Initial Design

Based from initial system requirements a wireframe for the GUI could be generated. To keep the user experience simple it was decided to keep all the basic functionality within the main window. This allows unfamiliar users to gain from the software without being overwhelmed by endless options. For advanced configuration there was to be a detailed configuration panel allowing advanced users to set the majority of variables as they saw fit.

The wireframe below was drawn up to support these concepts. By sticking to conventions it gives a familiar feel to most who have used a typical windows based system within the last decade.

6.3.2 GUI Implementation and Revision

It was decided since the project was to be built in Java and several members had knowledge of the library to use Swing for implementation of the GUI. The toolkit was more than capable of our needs and any advantages of alternatives seemed unlikely to counterbalance the time required to learn a new environment. SWT was briefly considered since the interface created generally allows a closer to native feel and higher performance [42]. This however would come at a cost of portability as well as the time consideration mentioned above.

The initial GUI implementation (6.2) was a quick and crude attempt towards the initial wireframe. Its purpose was more towards checking how well the JMonkey canvas could be displayed within the Swing layout than producing a suitable end product.

Whilst not particurly attractive a suitable platform for basic testing was then available.

At this point it became apparent that the back end would not easily allow for the evacuation to be “scrolled through” in the way we had initially anticipated. The play bar at the bottom of the screen was removed from designs. this led to the control buttons being moved to the panel on the

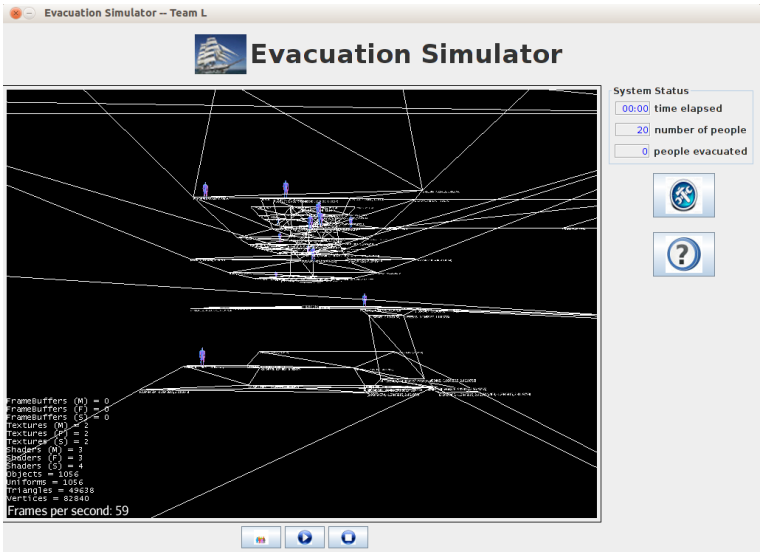


Figure 6.3: First refinement of GUI

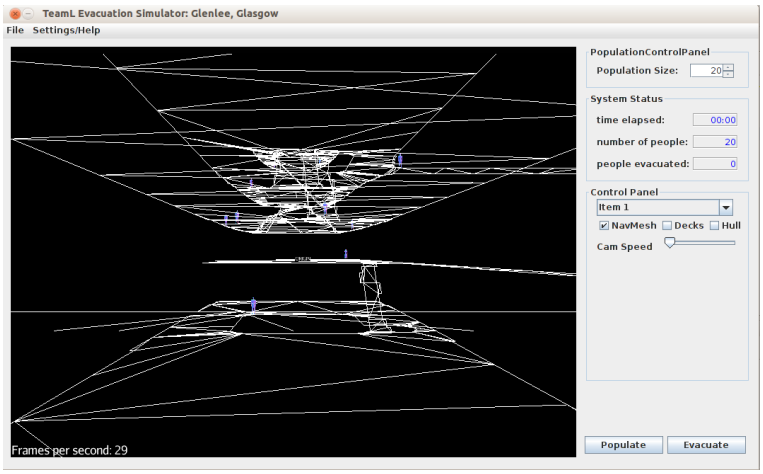


Figure 6.4: Second refinement to GUI

right. After speaking to some less computer-literate users it was further decided to use a traditional menu-bar. Some also found the navmesh didn't give an immediate understanding of what the model represented so toggles were added to allow a physical representation of the ship to be shown.

These changes manifested themselves as can be seen in 6.4. Due to more time being spent on this iteration it gives a much more complete feel and is considerably closed to the initial wireframe.

Further talks with users suggested that for those not used to computer-games the keyboard based camera controls were not immediately understandable. To fix this a control panel was added. Due to the way functionality was implemented a route button was added for testing purposes. It was then decided this would be of use to users so was left in. Finally, to allow a more native feel the theming was set to inherit from the system the code was run from. A new wireframe (6.5) was drawn up to show this.

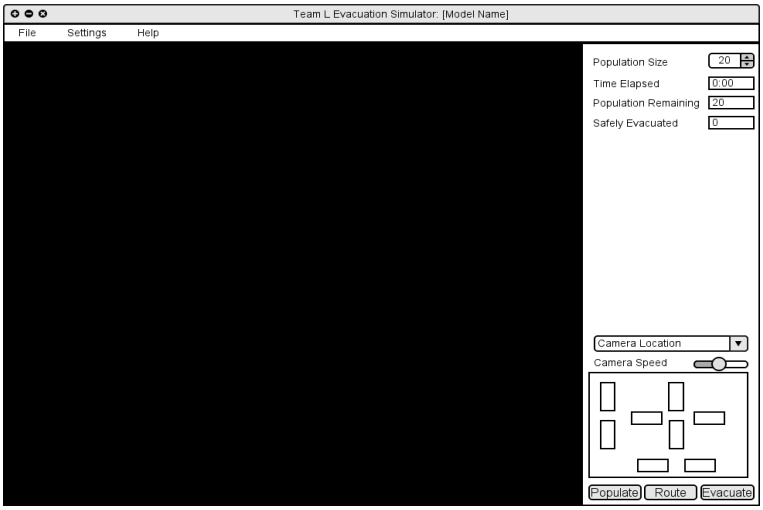


Figure 6.5: Third refinement to GUI

6.3.3 Final GUI

The wireframe in 6.6 was implmented to give our final GUI. It gives a polished finish and meets original aims well and takes into consideration user feedback.

Initial work was done using the Netbeans GUI builder. This allowed the layout we required to be built quickly, but not the functionality. To achieve this some work was needed on the raw swing code.

The final interface is shown in 6.7, both in Linux and Windows.

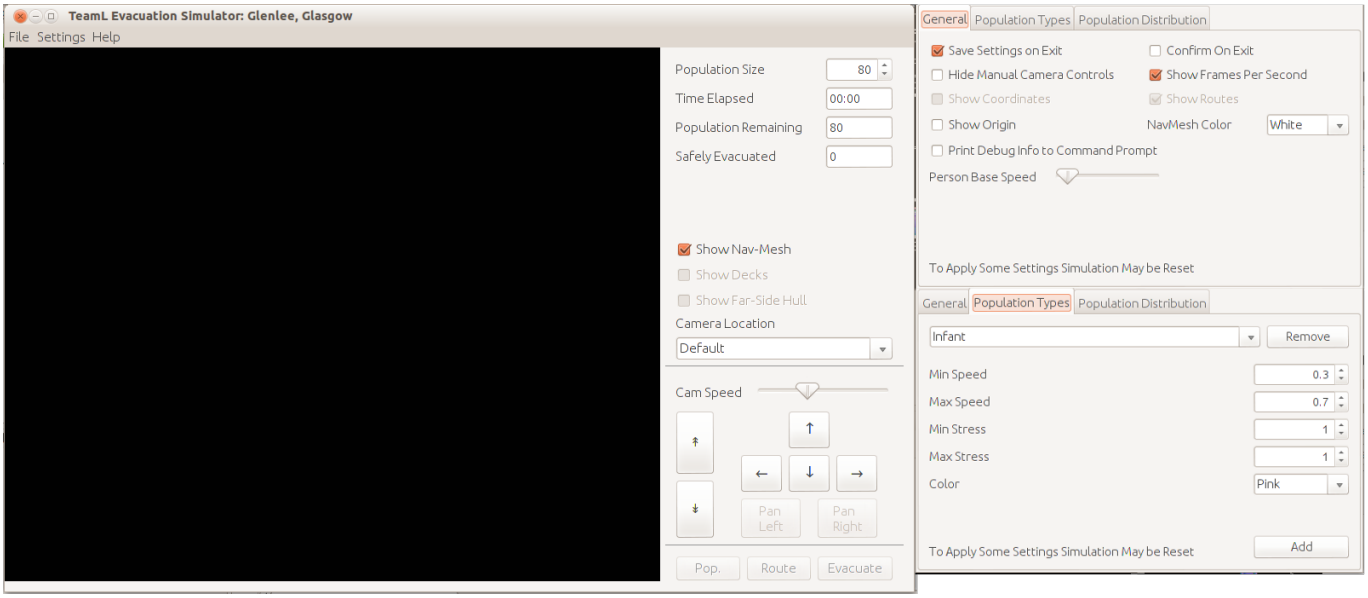


Figure 6.6: Final GUI wireframe

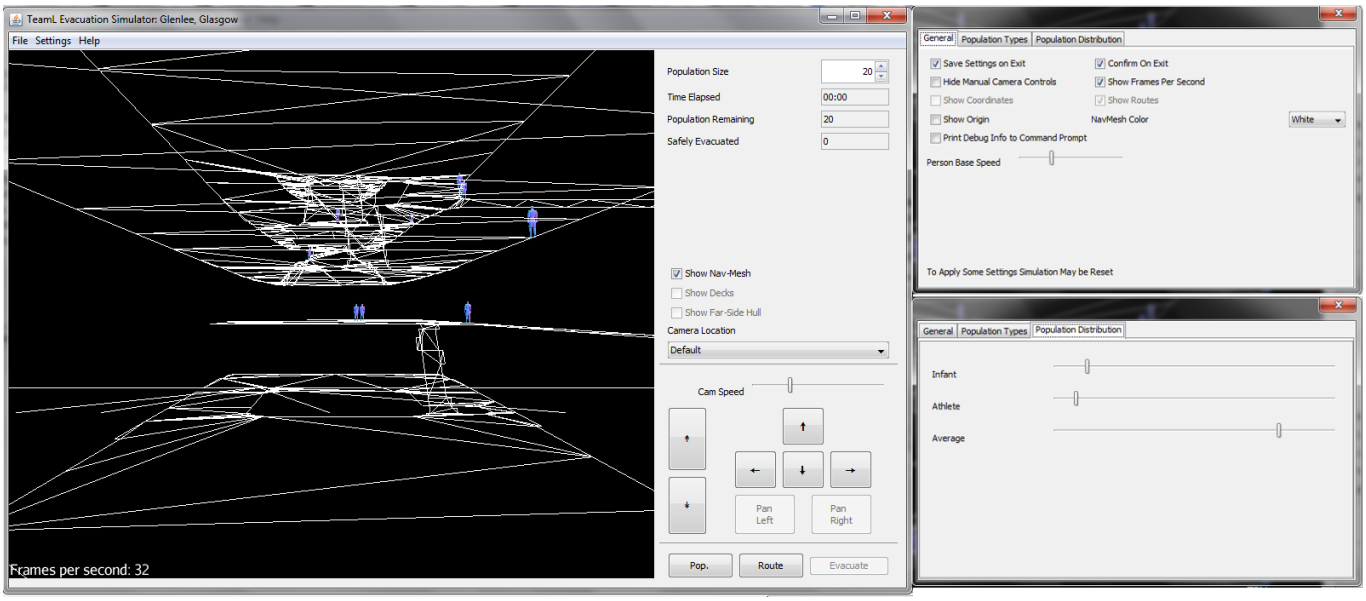


Figure 6.7: Final GUI in Swing

Chapter 7

Evaluation

7.1 Evaluating the Behavioural Model

When attempting to evaluate the effectiveness of a behavioural model implementation, in general all approaches involve a real-world enactment in the form of a mock evaluation. Alternatively, it is possible to replicate a previous incident where a real evacuation occured and try to recreate this in the system. Both of these approaches have severe drawbacks.

At first glance a mock evaluation may seem a tangible approach. However it has the following major drawbacks:

1. **Cost:** one of the key reasons for building an evacuation simulator is to avoid the need for mock evacuations, as the cost of shutting down the building and/or hiring mock evacuees can be hindering. Obviously this creates a paradox: the system is built to remove the need for mock evacuations but mock evacuations are needed to evaluate it.
2. **Lack of Urgency:** many evacuation behaviours are driven by the individual's urgency to exit. In a mock evacuation, there is obviously no real urgency for participants to exit as there would be in a real evacuation. This is difficult to account for in results, since the extent to which urgency changes behaviour may vary so greatly between individuals.For this reason it is unlikely to observe any competitive or otherwise aggressive behaviour, again since there is no sense of urgency to drive such actions.
3. **Danger:** even if we assume that we can replicate a sense of urgency, this may expose participants to unacceptable danger. For example we may expect in a real evacuation that an individual may be trampled, but naturally this is not something we wish to have happen in an experiment. The risk of such incidents occuring outweighs the benefits.

It had been hoped that a mock evacuation carried out by the staff of the Tall Ship could be observed for some comparison data, but this could not be arranged. In conclusion no evaluation of the behavioural model could be carried out, and it is therefore impossible to say to what extent it is an accurate model.

7.2 User Interface Evaluation Methods

The simulator will be evaluated through a series of user tests with different participants. The goals of the user interface evaluation are to assess the effect of the interface on the user and to identify specific problems which should be rectified. The simulator will be first tested with the fire warden of The Tall Ship who is the primary user of the system. Secondly, feedback on usability of the system will be gathered from usability testing with the subjects being students at the University of Glasgow.

Two evaluation methods, namely Heuristic Evaluation and Usability Experiments, will be used to determine whether the requirements specified in the Requirements chapter have been met and also to determine the overall usability of the system.

7.2.1 Heuristic Evaluation

Heuristic evaluation is a usability inspection method pioneered by Jakob Nielsen and Rolf Molich which helps to identify problems with a user interface by judging the interface's compliance to recognized usability principles – heuristics[43].

The heuristics made use of in this part of the evaluation are Nielsen's Heuristics, developed by Jakob Nielsen and Rolf Molich in 1990[44]. Nielsen refined the original set of heuristics in 1994[44]. Below is list of the heuristics and a description of each one:
1. Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
2. Match between system and the real world: The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
3. User control and freedom: Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
4. Consistency and standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
5. Error prevention: Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
6. Recognition rather than recall: Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
7. Flexibility and efficiency of use: Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
8. Aesthetic and minimalist design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
9. Help users recognize, diagnose, and recover from errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

10. Help and documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

The user interface of the simulator will be examined and evaluated using the descriptions of each of Nielsen's 10 Heuristics above. Heuristic Evaluation is a relatively quick and inexpensive way to evaluate a user interface. Deviation from recognized usability principles, identified from the evaluation, can provide great insight into how the user interface could be further refined to enhance the usability of the system.

7.2.2 Usability Experiments

Usability experiments can be used in addition to Heuristic Evaluation to gain further feedback on the system. These are particularly useful because they allow the experimenter to gain insight to the reactions of the users of the system first-hand.

Two different sets of users will be used for evaluation: the Tall Ship's fire warden and a group of students. The fire warden will be able to tell us whether the system meets the requirements and also on how usable the system is. Since it is unlikely that the students work in the domain of fire safety, the set of students will be primarily used to test the ease in which the system can be used. Any suggestions of improvements to the system by either group will also be recorded.

Experimental Design

The experiment must be designed carefully in order to provide results that are both reliable and generalisable. Two types of experimental design can be used: within-subjects design and between-subjects design.

In a between-subjects (or randomized) design, each participant is given a different condition, of which there are at least two. A control condition, where the independent variables are not changed, is needed to ensure the measured differences in the other conditions are true. Since each subject only performs under one condition, the likelihood of any learning effect from performing two similar conditions one after the other is mitigated. However, a between-subjects design requires a large number of participants if one is going to extract meaningful information.

In a within-subjects design, each subject is given the same conditions to perform. The effect of learning is more prominent in this method, which is a disadvantage but it has an advantage compared to between-subjects design because less subjects and time are required.

Considering the advantages and disadvantages of both methods, a within-subjects design will be adopted for the user interface evaluation of the evacuation simulator. Since limited resources are available in terms of time and users, the less costly within-subjects design is more appropriate. Learning effects can be lessened by changing the order in which the conditions are carried out by the participants. This allows a comparison of participants who carried out a condition first and participants who carried out the condition after another one, and therefore subject to learning. The

results from the within-subjects evaluation will be analysed to determine whether effects of learning have adversely affected the results of the evaluation.

For both sets of users, they will carry out a set of fixed tasks and the time taken to perform these tasks as well as any mistakes they make will be recorded. This data will be used to form the evaluation results and will allow the identification of flaws in the usability of the system and areas for improvement.

Think-aloud Protocol

In addition to the usability experiment discussed above, the think-aloud protocol will also be used to gather information from users of the system. This method was introduced in the usability field by Clayton Lewis and is discussed in Task-Centered User Interface Design: A Practical Introduction by C. Lewis and J. Rieman[45]. Think-aloud protocols involve participants thinking aloud as they perform a set of pre-specified tasks. The idea is to have the users of the system saying out loud exactly what they are doing and how they are feeling. This allows the experimenter to gain a first-hand sight of a user using the product and provides insightful knowledge into how the end user would go about performing tasks.

The information gathered from the experiment will be analysed and the difficulties the user had will be discussed and rectified by changing the user interface. Any major changes to the user interface will have to be evaluated again to ensure the changes actually improve the usability of the system as a whole.

7.2.3 NASA TLX: Task Load Index

NASA Task Load Index (TLX) will be used to measure the workload of the participants relating to the tasks that they are asked to perform. It is a subjective workload assessment tool which is used to derive an overall workload score based on a weighted average of the six subscales, namely Mental Demands, Physical Demands, Temporal Demands, Own Performance, Effort and Frustration[46].

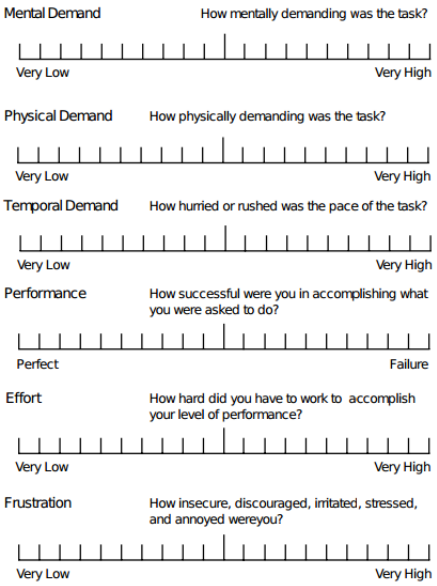


Figure 7.1: NASA TLX Rating Scales

After the participants have given a rating on each of the six subscales, they are asked to carry out a pairwise comparison between the subscales to identify their relative importance, in the eyes of the user.

This particular experiment involves asking the user to carry out pairwise comparisons, however, some research has suggested that the exclusion of the pairwise comparisons between subscales can increase experimental validity[47].

7.3 User Interface Evaluation Results

The results from the user interface evaluation using heuristic evaluation, think-aloud and NASA Task Load Index (TLX) are discussed below. The recommendations in this section were used to enhance the Graphical User Interface which is discussed in section 6.3.

7.3.1 Heuristic Evaluation

Visibility of system status

The system gives users little information about what is going on. For example, on pressing the populate or route buttons, no information is displayed on the screen informing the user that something is happening in the background. As a result, the user could be left wondering whether the button was correctly clicked.

Inclusion of messages on the screen stating what is happening at a particular moment in time would help the users to identify the status of the system. For example, inclusion of a message saying that the system is loading instead of a black screen will inform the user that something is indeed

happening. Messages when the user clicks a button confirming that the action is happening and a message that displays on the screen when the evacuation simulation has finished will also increase the users awareness of the system status which in turn makes the system more user friendly.

Match between system and real world

The arrow keys (up, down, left and right) for camera movement are positioned in a logical order which would be familiar to the majority of users. This allows an easy mapping from real world natural conventions to the system which makes the system easier to use. Non-natural placement of these buttons on the screen would cause confusion in users and lead to frustration.

The system does use the term 'navmesh' in the checkbox labelled 'show navmesh'. The user is likely to be unfamiliar with this technical term and would thus have to consult the documentation to learn properly what it does. The system should use words and phrases familiar to the user so it would seem appropriate to replace the word 'navmesh' with something along the lines of 'ship outline' or 'wireframe'.

User control and freedom

The user has the ability to control the camera manually using the on-screen buttons. The allows the user to view the ship in any way they want. There is also functionality to increase or decrease the speed of the camera and also to select preset camera locations from a drop-down menu which increases the control and freedom the user has. The number of preset camera locations is currently two. This should be expanded to give the user more options.

The user can change the population size from the main screen and a more advanced settings menu allows the user to create categories of people according to various factors. This gives advanced users more freedom in the interactions they make with the system.

Consistency and standards

In general, the buttons are labelled well and the user does have to wonder what they do. However, some problems exist in the camera settings part of the main screen. There are two up arrows and two down arrows and their functionality is not made entirely clear to the user. One of the up arrows is to pan up and the other up arrow is to rotate upwards. This should be made clear to the user by either increasing the size of the button and including a meaningful button name, or include the information as text above the buttons on the camera controls panel itself.

Error prevention

Some steps have been taken to prevent the users from executing an action which would lead to an error in the system. The evacuate button is grayed out and is only allowed to be clicked by the user when the ship has been successfully populated. By stopping the user from performing this action

until it is appropriate, allows the prevention of an illegal system state – namely evacuating an empty ship.

Error prevention related to the other buttons was overlooked and should be corrected. Other buttons on the main panel of the user interface should also be grayed out when it would not be appropriate to click.

Where there exist fields which can be altered by the user (for example, the population size field) a maximum and minimum value has been defined to prevent the user from entering too high or too low a number. This eliminates the possibility that the system will enter a state which it cannot handle as a result of user input which protects the system from mistakes made by the user.

Recognition rather than recall

The main buttons on the user interface are made to be as self explanatory as possible, however one shortfall is the design of the camera location panel. As discussed in the consistency heuristic, it should be made more clear what these buttons actually do. This would remove the need of the user to consult documentation for help and would thus reduce the extent of recall from one dialog to the next.

Flexibility and efficiency of use

The system has the ability to cater for more experienced user as mentioned in the user control and freedom heuristic. There exists the ability for the more experienced user to change a variety of properties of the camera such as speed and location. There is also the ability to use the keyboard to navigate around the ship which would allow more experienced users to increase their speed of interaction with the system.

There is, however, no method of allowing the user to tailor frequent actions through the use of accelerators or custom keyboard shortcuts, for example. The addition of such features would increase development time and since the experienced users group is a minority, it would not be in the best interest to develop this feature at this time.

Aesthetic and minimalist design

The system has a main screen which includes the most often used actions, and a settings screen which includes extra functionality. This separation allows a less cluttered minimalist view in the main screen which is easier to comprehend for the user. The system had been designed to display elements positioned in a natural way, allowing the user to focus on using the system and not needing to familiarise themselves with the interface for too long.

Help users recognize, diagnose and recover from errors

As mentioned in the prevention of errors heuristic above, some steps have been taken to prevent the user from making errors. However, since not all sections of the interface are removed from use when they are not supposed to be used, it is possible for a user to crash the system by pressing the buttons repeatedly. There are no friendly error messages to tell the user that something has went wrong – they are presented with a black screen. This leaves the user wondering whether the system is busy in the background carrying out some task, or has crashed.

To resolve this problem, better messages should be displayed on the screen to the user to increase visibility of system status. This would eliminate any confusion from the user when using the system. A reset button should also be implemented as a last resort for the user to click if the system crashes for an undocumented reason. This will ensure robustness in the system and increase the user-friendliness as a whole.

Help and documentation

No help or documentation is provided to the user. While the majority of the system is, on the whole, quite intuitive to use, some parts such as the camera controls panel and the advanced settings dialog box would benefit from documentation.

Brief documentation on the basic parts of the interface and what each button does as well as more detailed documentation on the more complex parts of the system should be created to assist the user in using the system and making decisions.

7.3.2 Think Aloud

The Think Aloud evaluation of the user interface was carried out with eight participants. The participants were asked to complete the six tasks listed below and were encouraged to think aloud during the evaluation. The participants were observed and notes were taken to allow a discussion of improvements which could be made to the user interface after the evaluation.

Tasks

1. Set the population size to 50. Populate the ship.
2. Hide the ship frame from view, then show it again.
3. Generate the exit routes for the population.
4. Change the camera angle so you have a bird's eye view of the ship.
5. Evacuate the ship. While the evacuation is taking place, change the camera view to face the exits of the ship.
6. Read out loud the time the simulation took and the number of people evacuated.

A summary of the findings of the Think Aloud evaluation is discussed below. Detailed notes on a per-participant level are included in Appendix C.

- The visibility of system status was a clear shortfall as indicated by the participants. Particularly in tasks 1, 3 and 5 where buttons which required the user to wait after being pressed, the participant was left guessing whether anything was happening with the system or whether they had done something wrong.
- The meaning of the camera location buttons was also not as clear as they should have been. Participants had to investigate what the buttons did and clear labelling would have solved this issue.
- The participants had no difficulty identifying key information such as number of people evacuated and the time taken for the evacuation which confirms that the information is well visible and the labelling is unambiguous.
- Identification of the completion of the evacuation was most often done by looking at whether anything more was happening in the graphical representation rather than looking at the 'remaining people' metric at the right hand side of the interface. It was noted that a pop-up message when the simulation is complete would make this more clear to the user.
- The terminology used in the user interface must be standardised to remove jargon words such as navmesh which would be unclear to the user. Task 2 highlighted that while some users guessed that ship frame was equivalent to the navmesh in this particular situation, 5 of the 8 participants were confused by the technical language.

7.3.3 NASA TLX: Task Load Index

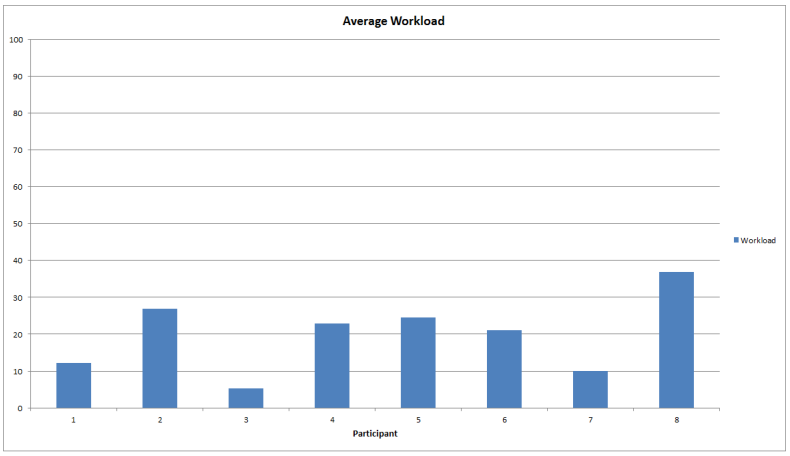


Figure 7.2: Average workload across participants

The above image shows the average workload of the eight participants in the experiment, which is calculated based on the ratings and pairwise comparisons given by the participant. It can be seen that the overall workload of each participant is reasonably low and so it can be said that using the system is not demanding.

Chapter 8

Conclusion

8.1 Requirements Acheived

The following is a summary of the requirements we feel have been achieved. Testing the initial requirements proved challenging because most of our client's requests were non-functional requirements or complex to test (e.g. the requirement of having a usable GUI). Unfortunately an acceptance test demonstration could not be carried out with the client at the time of writing.

1. All of the system's must have requirements
2. Ability to set different categories of agent
3. Ability for user to specify environmental and population variables
4. Pseudo-random population generation including randomising initial parameters as well as positions
5. Accurate timescale
6. Allow user to change the perspective (camera location and visual representation)

The extent to which we have achieved an effective GUI is discussed separately in the User Evaluation Section 7.3.

8.2 Major Problems Encountered & Future Work

Whilst in many ways the project was a success, due to several problems we failed to meet the full set of expected requirements (must have and should have). This section summarises the major issues that occurred and why. Further to this, an explanation of how the work should be continued based from experience so far is provided.

8.2.1 Problems Encountered

jMonkeyEngine

We had several issues with the engine with most relating to an underestimation of its complexity. This was partially down to poor documentation but also due to inadequate research into the system before commencing implementation.

jMonkeyEngine has extensive tutorials, however these are aimed at games development. The material soon got to a point where most of it was useless for our needs. This led us to abandon them quite early on.

Unfortunately this left several members of the team with a poor understanding of the system.

The documentation of advanced features also proved poorly organised and difficult to navigate.

NavMesh Generation

The NavMesh package used was based on an extremely powerful implementation in C++, Recast[38]. Unlike the original, instead of being implemented for speed it was created for educational purposes. It was also not well documented but claimed to be fully functional.

Throughout the project we encountered several problems with the generation of meshes. The largest of these was an inability to work with overlapping navigable surfaces (i.e. multiple floors in a building). At the stage where these defects became apparent, we were already heavily commited to this package, and a lack of suitable alternatives could be found. Overcoming this required large amounts of modification at a heavy cost in time.

Modelling Issues

An issue we had failed to consider when choosing our initial location was the complexity of the structure. The curved floors proved a serious challenge considering the team's lack of experience with advanced modelling. This was somewhat negated due to the issues with the NavMesh. The models generated became far more complex than we could convert and so are unused in the end product.

The other key issue with the modelling relates to the ship plans provided. Upon completion of each of the deck plans, these needed to be merged together into one model. It was then we realised the plans did not align. This required heavy modification to fit the stairways into place.

Population Architecture

At the moment, although coded, almost none of the agent behaviour discussed previously is in usage. This was the result of a design problem which restricted the capability to correctly tie the behavioural algorithms to the agents.

As discussed in the Agent Implementation Section **??** & Design Section 5.1 , an agent's route is calculated in a two stage process, at the end of which a MotionPath is produced. This design emerged as the result of difficulties in reaching the aims of Prototype 2.
To give an agent the required individual behaviour, the system must be able to:

1. Stop an agent. From the position at which they were stopped, calculate a new route to a chosen target. Restart the agent to continue down this route.
2. Access an agent's position at any time

Both of the above conditions need to be performed from a synchronised context. Unfortunately neither of these are possible in the current Population package architecture. The implementation of MotionPath heavily restricts access to an agent's position. While it is possible to stop an agent moving, no method has been found for triggering this in a meaningful sense that does not result in a system crash or thread deadlocking.
This design occurred because the routing procedures and behavioural algorithms were developed in parallel. Since they were built up in incremental prototypes (see Appendix A), the challenges associated with merging these two aspects were not considered until late in the project. This could have been addressed with a partial reimplementation of the Population package, but the extensive problems with the navigation mesh found during stage 3 (see Appendix A) meant that the individual behaviour features were ultimately dropped.
Nevertheless, we present an overview of the correct solution in Section 8.2.2. We are confident that should the proposed reimplementation be carried out that it would be relatively simple to tie the behavioural methods to agents effectively.

8.2.2 Future Work

Below is listed, in order of priority, the next steps that shall be taken if the project is continued

Implementation of a thread-pool and rewriting of the Person Class

The Person Class was developed early on and as such was built before we had a clear understanding of the technologies being used. As such it was built upon a bad methodology upon which the class relies too heavily to allow easy repair. Hence it seems more appropriate to rewrite it from scratch.

Further to this, currently each person is represented by an individual thread. This requires considerable overhead and is one of the key performance bottlenecks in the existing system. By converting a person to an abstract data structure and having a threadpool of workers, major decreases in waste could be overcome. In each person a reference to the relevant behavioural class would be stored. These would be static since only one for each category would be required and would inherit methods from a generic superclass. Variables in the classes would control basic and user created categories, whilst the possibility to override methods would allow hard coded extreme changes, such as to add in a disabled visitor or a staff member.

Some kind of data structure to store each persons location would be needed. This would have to be made thread-safe to allow usage by the pool. Ideally, this would store people in an order

linked to their position making locating others within a specific radius of a point easy (needed for collision control mentioned below). The worker threads would iterate over each person on each cycle of the update loop. The behavioural methods would be fed a list of other nearby people and environmental factors (such as nearby exits). It would generate a new location in a single unit of movement based upon this data, the category class's variables such as speed and the current average frames per second. The person in question would be moved by this distance and the worker would move on to the next person. We would either need to lock other people within close vicinity whilst this calculation was performed, or check before moving the person nobody had moved into their path. To decide which action to take here, further research would need to be performed both into the performance of each method and whether temporarily locking movement would restrict the accuracy of the behavioural model.

Use of D* algorithm

Currently A* is used to generate routes. By using D* instead these would support dynamic updating due to interactions with other members of the crowd or hazards.

Use of ORCA for collision avoidance

Due to the large number of people our system is aimed to support, collision control and avoidance needs to be properly implemented. Currently individuals are perfectly able to pass through one another. ORCA (Optimal Reciprocal Collision Avoidance) is an advanced set of algorithms designed to combat this.

Implementation of an advanced behavioural Model

Due to the issues relating to understanding of the underlying engine, work here had to be temporarily suspended. As a result of this, the current model is lacking. Once the person class has been re-created in a way able to support this properly it could be continued allowing the functionality originally envisioned at the start of the project.

Improvement of model importation and removal of maritime phrases

The project was always intended to be flexible towards a number of environments. Currently the importer is poor and allows no modification of the NavMesh generation variables. There are also several words used within the GUI hinting towards a maritime use (such as Show Far-Side Hull on the main side panel). These counteract our aims by suggesting the software to be bespoke.

Addition of hazards and assurances into the model

In a real situation there are multiple stimuli affecting human behaviour. Exit signs entice those trying to escape whilst the majority of people adhere to no-entry indications.

Hence additional environmental features should be added to assist with the behavioural model.

8.3 Summary

In conclusion, the majority of the project's aims were achieved successfully. However, there is the potential for significant future work to vastly improve the functionality the system provides. It is hoped that the project will provide an extensible and flexible foundation for future development efforts.

8.4 Contributions

- **Michael Kilian:** Quality Assurer. Member of Implementation Team. Responsible for research into individual behaviour and corresponding design and implementation. Tester/Debugger. Wrote sections on behaviour research, Population package design, behavioural algorithms, process model and prototyping and team structure.
- **Tony Lau:** Toolsmith. Member of Implementation Team. Provided coding support where needed. Coordinated evaluation and liaison between Core Implementation and Modelling teams. Wrote evaluation section with contributions to introduction.
- **Dan Tomosoiu:** Configuration Manager. Member of Implementation Team. Responsible for research into agent navigation and corresponding design and implementation. Tester/Debugger. Wrote sections on navigation structures design and implementation.
- **Hector Grebbell:** Project Manager. Member of Modelling Team. Chief 3D modeller. Contributed to GUI design and implementation. Provided occasional support in debugging and testing. Wrote sections on GUI implementation and technologies chosen.
- **Peeranat Fupongsiripan:** Librarian. Member of Modelling Team. Support 3D modeller. Contributed to GUI design and implementation. Wrote introduction section and requirements.

Appendix A

Prototypes and Problems Encountered

The prototyping process consisted of five deliveries in total, including the final system. Here we outline the aims and objectives of the first four of these, which problems/defects were discovered at each stage.

A.1 Prototype 1

A.1.1 Aims

- Generate a navmesh over a simple, one floor model of a building.
- Initialise and generate evacuation route for a single agent (note that they do not need to exit, animation will be implemented later. Nor should the agents have any behaviour at this stage.

A.1.2 Problems/Design Changes

The navmesh package was generating non-optimal routes between cells. This was easily fixed (see implementation).

A.2 Prototype 2

A.2.1 Aims

- Extend initialisation and path calculation to multiple agents.
- Introduce animation for agent movement to exits.

A.2.2 Problems/Design Changes

In implementing agent animation, it was found that the initial design for an agent would not allow for their animation within the constraints imposed by JMonkeyEngine's graphics update procedure. The design was changed to use the JMonkeyEngine class MotionPath. Eventually the logical and visual representations of an agent were seperated entirely, giving the Person and PersonNavmeshRoutePlanner classes.

A.3 Prototype 3

A.3.1 Aims

- Introduce goals and foundations for agent behaviour.
- Move to using the final, multi-storey model of the Tall Ship.
- Basic GUI, consisting of a simple settings menu and camera control buttons.

A.3.2 Problem/Design Changes

The move to the multi-floored model revealed an extensive number of defects in the navmesh package which were not apparent in the single-floor model (discussed further in implementation). Virtually all effort was expended in correcting these defects, leaving the first aim largely unrealised.

A.4 Prototype 4

A.4.1 Aims

- Implementation of herding algorithm.
- Refined GUI.
- Advanced settings made available.

A.4.2 Problems/Design

This phase saw the most significant period of refactoring, radically overhauling the system's package structure.

Appendix B

Think Aloud Evaluation Participant Notes

Tasks

1. Set the population size to 50. Populate the ship.
2. Hide the ship frame from view, then show it again.
3. Generate the exit routes for the population
4. Change the camera angle so you have a bird's eye view of the ship.
5. Evacuate the ship. While the evacuation is taking place, change the camera view to face the exits of the ship.
6. Read out loud the time the simulation took and the number of people evacuated.

Participant 1 (ID: 146)

1. Participant successfully changed the population size to 50 but was unclear about the status of the system after clicking the populate button. Participant questioned whether anything was happening.
2. Did not understand the task fully, particularly what was meant by ship frame. The participant tried to hide the ship from view by changing the location of the camera so that the ship was not in view but this was not what was wanted.
3. Participant was eventually able to accomplish the task although at one point was confused and thought camera location had an effect of route generation. This did not affect the outcome of the task, though.
4. Controls used correctly but the camera angle was slightly short of the bird's eye angle asked for.

5. Participant knew to use the evacuate button. When changing the camera view, the participant made use of the camera location controls and manually panned the camera to find the exits instead of using the preset camera location drop-down menu. Judging the completion of the simulation was done by noticing the absence of people from the ship and not by the remaining people display panel.

6. Completed successfully.

Participant 2 (ID: 102)

1. Accomplished successfully, although expressed views of uncertainty of system status.

2. No problems completing task.

3. Successfully generated routes although was unsure again about system status. The participant said that they did not know whether the button had worked or not.

4. Participant went straight to the camera location drop-down but realised another method was needed because 'bird's eye' was not there. Tried to use the manual controls and the mouse on the simulation environment but couldnt finish the task.

5. Clicked the correct button for evacuate but when selecting the exits item from the drop-down menu, did not immediately click the move button to register the action. The participant realised the mistake and clicked the move button to finish the task.

6. Completed successfully.

Participant 3 (ID: 93)

1. Accomplished without hesitation.

2. Looked at the interface for longer to identify the correct option. At first the participant clicked the drop-down camera locations but realised this was a mistake. The participant was confused as to what the task was asking and assumed navmesh was the ship frame.

3. Moved mouse from the top to the bottom of the display to identify the button to press. The participant correctly identified and pressed the button although was unsure about the progress of the system. The participant assumed something was happening in the background because a lot of text was being displayed in the console.

4. The camera locations drop-down was the first thing the participant looked at. The participant could not find the correct option so looked for an alternative. The cam speed slider was tried a few times. The participant expected it to do something but nothing was happening on screen. The participant then pressed buttons haphazardly and admitted that they could not finish the task.

5. Successfully evacuated the ship, moved the camera to the exits using the drop-down menu and identified when the evacuation ended.

6. Completed successfully.

Participant 4 (ID: 24)

1. Population size changed successfully but there was a slight hesitation finding the populate button.
2. Completed successfully although the participant had some confusion over whether the frame asked for in the question and the navmesh were the same thing.
3. Clicked the route button but was unsure whether the routes were actually being calculated.
4. Could not find 'bird's eye' view in the drop-down menu and was confused over the semantics of the camera control buttons. The participant failed to complete the task.
5. The participant completed the task successfully. The completion of the evacuation was immediately noticed.
6. Completed successfully.

Participant 5 (ID: 46)

1. Participant correctly changed the population size but thought that on entering the population, the system would populate itself. The participant clicked the populate button but was unclear when the system was finished completing the task.
2. At first the participant did not bring the ship frame back into view but was able to hide it with no problems.
3. Correctly identified the button to press but was unclear about the system state.
4. Participant was able to use the mouse and the buttons on screen to manipulate the camera angle. It was noted that the participant went to the on screen buttons first and then used the mouse on the simulation canvas to finish the task.
5. The participant panned to the exits using the manual buttons and not the preset camera location drop-down menu as anticipated. The participant also announced the end of the evacuation prematurely by not looking at the remaining people metric at the right hand side.
6. Completed successfully.

Participant 6 (ID: 81)

1. Completed successfully but was unsure of system state.
2. The participant had a slight hesitation to find the correct button to press but accomplished the task nevertheless.
3. The route button was found clearly by the participant although it was remarked that nothing was happening on screen when there was really something happening in the background.
4. The participant went straight to control the ship with the mouse directly interacting with the canvas. This was supplemented by use of the camera control buttons provided on the interface and the task was completed successfully.

5. The participant did not notice the drop-down menu item to set the view to the exits view already defined. Instead, the manual controls were used to pan to the exit location. The remaining participants metric in the sidebar was not used to determine whether the evacuation was finished. Instead, the participant preferred to see whether there were any people still on the ship by looking at the simulation graphics.

6. Completed successfully.

Participant 7 (ID: 334)

1. Participant thought it was quite obvious how to accomplish the task and completed it without any trouble. The system status was not clear to the participant.

2. The terminology of ship frame versus navmesh was questioned and assumed by the participant to mean the same thing. The task was completed after this initial hesitation.

3. Participant remarked that the button was called route and though that it should be called routes to indicate more than one route. The participant was not sure whether anything was happening and was confused about the overlapping lines which appeared on the screen after this task was accomplished.

4. Participant went straight to the camera locations drop-down to find a preset. After noticing that there was no preset, the participant proceeded to instinctively use the mouse and a combination of the on screen buttons to accomplish the task.

5. Successfully identified and clicked the correct button and then proceeded to investigate the scene using the mouse and the scroll on the mouse. The participant raised some questions such as how many exits there are and how the user is supposed to know where a particular exit is.

6. Completed successfully.

Participant 8 (ID: 336)

1. The participant highlighted the text box and changed the population size without any difficulty.

2. The terminology of frame confused the candidate and wasnt sure whether she had done the correct thing by hiding and showing again the navmesh.

3. The participant was not sure whether the route button was the correct button to press. It was assumed by the participant to be correct after noticing that more things appeared on the screen.

4. The camera location drop-down was selected first and the participant selected exits to see what it did. This was the wrong thing to do and realising the error, the participant proceeded to press all the camera location buttons to try to complete the task. There was not much structure to the way the buttons were pressed. On discovery of the fact that the mouse could be used on the simulation canvas, the task was completed successfully.

5. The participant was not sure they had to click the move button to register the camera move event but eventually realised this had to be done.

6. Completed successfully.

Appendix C

Glossary

- **IDE:** Integrated Development Environment
- **jME:** jMonkeyEngine – open source game engine in which the simulation was developed.
- **GUI:** Graphical User Interface
- **Navmesh:** Navigation Mesh – structure used for mapping environment in terms of the area which agents can navigate.

Bibliography

[1] M. Siikonen H. Hakonen, T. Susi. *Evacuation Simulation of Tall Buildings*. 2003.

[2] B. Aguirre G. Santos. *A Critical Review of Emergency Evacuation Simulation Models*. 2004.

[3] Eddie Wrenn. http://www.dailymail.co.uk/news/article-2104822/London-2012-Olympics-tube-terror-attack-drill-staged-emergency-services.html, 2012.

[4] Robert Rosenthal and Ralph L. Rosnow. *Artifacts in Behavioral Research*. Oxford University Press, 2009.

[5] Dr John Drury and Dr Chris Cocking. The Mass Psychology of Disastersand Emergency Evacuations, March 2007.

[6] CM Macal and MJ North. *Tutorial on agent-based modelling and simulation*. Palgrave Journals, 2010.

[7] Xiao Cui and Hao Shi. *A*-based Pathfinding in Modern Computer Games*. School of Engineering and Science, Victoria University, Melbourne, Australia, 2011.

[8] Mark A. DeLoura. *Game Programming: Gems 2*.

[9] Christer Ericson. *Real-Time Collision Detection*. 2005.

[10] C. Atkinson. R Hewitt. *Parallelism and Synchronization in Actor Systems*. 1977.

[11] Stephanie Pace Marshall. *Chaos, Complexity, and Flocking Behavior: Metaphors for Learning*. 1996.

[12] Susan B. Van Hemel Greg L. Zacharias, Jean MacMillan. *Behavioral Modeling and Simulation*. 2008.

[13] http://en.wikipedia.org/wiki/Fluid_mechanics.

[14] E.R. Galea. *The numerical simulation of aircraft evacuation and it's application to aircraft design and certification*. F.S.E.G., University of Greenwich, 1999.

[15] E.R. Galea. *Use of mathematical modelling in fire safety engineering*. F.S.E.G., University of Greenwich.

[16] G. Keith Still. *Crowd Dynamics*. University of Warwick, 2000.

[17] Xiaoshan Pan, Charles S. Han, Ken Dauber, and Kincho H. Law. *A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations*. PhD thesis, Stanford University, 2006.

[18] http://thetallship.com.

[19] Stephanie Ludi. *Student Survival Guide to Managing Group Projects 2.5*. http://www.se.rit.edu/~sal/SEmanual/TableOfContents.html, 2006.

[20] Ian Sommerville. *Software Engineering, 9 ed.* Pearson, 2011.

[21] http://en.wikipedia.org/wiki/Adaptive_behavior_(ecology).

[22] Kincho H. Law, Ken Dauber, and Xiaoshan Pan. *Computational Modeling of Nonadaptive Crowd Behaviors for Egress Analysis*. PhD thesis, Stanford University, 2006.

[23] Xiaoshan Pan. *COMPUTATIONAL MODELING OF HUMAN AND SOCIAL BEHAVIORS FOR EMERGENCY EGRESS ANALYSIS*. PhD thesis, Stanford University, 2006.

[24] Gerd Gigerenzer and Peter M. Todd. Simple heuristics that make us smart. .

[25] Theodore Millon & Melvin J. Lerner Irving B. Weiner, editor. *Handbook of Psychology, Volume 5: Personality and Social Psychology*. John Wiley & Sons, 2003.

[26] http://en.wikipedia.org/wiki/Social_proof.

[27] Edward T. Hall. *The Hidden Dimension*. Bantam Doubleday Dell Publishing Group, 1966.

[28] http://en.wikipedia.org/wiki/Proxemics.

[29] Xiaoshan Pan, Chuck Han, Kincho H. Law, and Jean-Claude Latombe. *A computational framework to simulate human and social behaviours for egress analysis*. PhD thesis, Stanford University, 2006.

[30] Tobias Kretz, Anna Grunebohm, and Michael Schreckenberg. *Experimental study of pedestrian flow through a bottleneck*. PhD thesis, University of Dulsburg-Essen, 2008.

[31] Ansgar Kirchnera, Hubert Klupfel, Katsuhiro Nishinari, Andreas Schadschneider, and Michael Schreckenberg. *Simulation of competitive egress behavior: comparison with aircraft evacuation data*. PhD thesis, Various Institutes, 2003.

[32] Nick Chater Ramsey M. Raafat and Chris Frith. Herding in humans. *Trends in Cognitive Science. Vol. 13 No. 10*, 2009.

[33] Kyle Wilson. Why c++. *Game Architect*, 2006.

[34] Oracle. http://www.java.com/en/about/.

[35] Dejan Jelovic. Why java will always be slower than c++. www.jelovic.com/articles/why_java_is_slow.htm.

[36] Dave Abrahams. Boost python documentation. http://www.boost.org/doc/libs/1_53_0/libs/python/doc/index.html.

[37] Promit Roy. Direct3d vs. opengl: Which api to use when, where, and why. *Graphics Programming and Theory; GamerDev.net*, 2002.

[38] https://code.google.com/p/recastnavigation/.

[39] http://www.critterai.org/nmgen_study/.

[40] E. W Dijkstra. A note on two problems in connexion with graphs. 1959.

[41] Bertram Raphael Peter E. Hart, Nils J. Nilsson. A formal basis for the heuristic determination of minimum cost paths. 1968.

[42] http://www.ibm.com/developerworks/grid/library/os-swingswt/.

[43] Heuristic Evaluation. http://www.nngroup.com/topic/heuristic-evaluation/.

[44] Nielsen's 10 Heuristics. http://www.nngroup.com/articles/ten-usability-heuristics/, 1995.

[45] C. Lewis and J. Rieman. *Task-Centered User Interface Design: A Practical Introduction*. 2006.

[46] NASA Task Load Index. http://humansystems.arc.nasa.gov/groups/TLX.

[47] Sandra G. Hart. NASA Task Load Index (NASA-TLX); 20 Years Later. http://www.nngroup.com/articles/ten-usability-heuristics/, 2006.