

Final Report

|  |
| --- |
| School of Computing  Faculty of Engineering |

Automatically Designed Street Layout With Crowd Simulation

Edward A H Webb

Submitted in accordance with the requirements for the degree of  
BSc Computer Science

**2019/2020**

Type of Project: Exploratory Software - ESw

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)

© 2019/2020 The University of Leeds and Edward A H Webb

# Table of Contents

[Table of Contents ii](#_Toc28031159)

[Chapter 1 - Introduction 1](#_Toc28031160)

[1.1 Aims 1](#_Toc28031161)

[1.2 Objectives 1](#_Toc28031162)

[1.3 Deliverables 2](#_Toc28031163)

[1.4 Initial Plan 2](#_Toc28031164)

[1.5 Risk Mitigation 4](#_Toc28031165)

[Chapter 2 - Background Research 5](#_Toc28031166)

[2.1 Crowd Simulation 5](#_Toc28031167)

[2.1.2 Menge 5](#_Toc28031168)

[2.1.3 Crowd Simulation Paradigm 5](#_Toc28031169)

[2.1.4 Scoring the Layouts 6](#_Toc28031170)

[2.2 Automatically Changing the Layouts 6](#_Toc28031171)

[2.2.1 Generative Adversarial Networks 7](#_Toc28031172)

[Chapter 3 - Ethics 8](#_Toc28031173)

[List of References 9](#_Toc28031174)

# Chapter 1 - Introduction

The flow rate of people is an important factor when architects and others design a street layout, and serious thought needs to be put into the positions of things such as benches and display booths. Crowd simulation software provides insights as to how a group of people may move in certain environments and situations – such as in the event of emergency. Due to the importance of safety in design, crowd simulation is among the most prominent techniques for doing so [1].

## Aims

This project investigates how using crowd simulation in conjunction with an automatically changing environment could lead to improvements in the design process of large spaces.

This project aims to deliver a piece of software which will attempt to optimise the flow of people through a pedestrian shopping street. This will be done by automatically changing the positions of obstacles in the middle of the street to random locations, and then using crowd simulation software to measure the flow rate of people going about different activities in the street. A 2D model will be used, as this will maintain simplicity and because a 2D simulation is sufficient and more appropriate for this project’s flow-style simulation [1]MIGHT NEED TO CHANGE THIS REF.

## Objectives

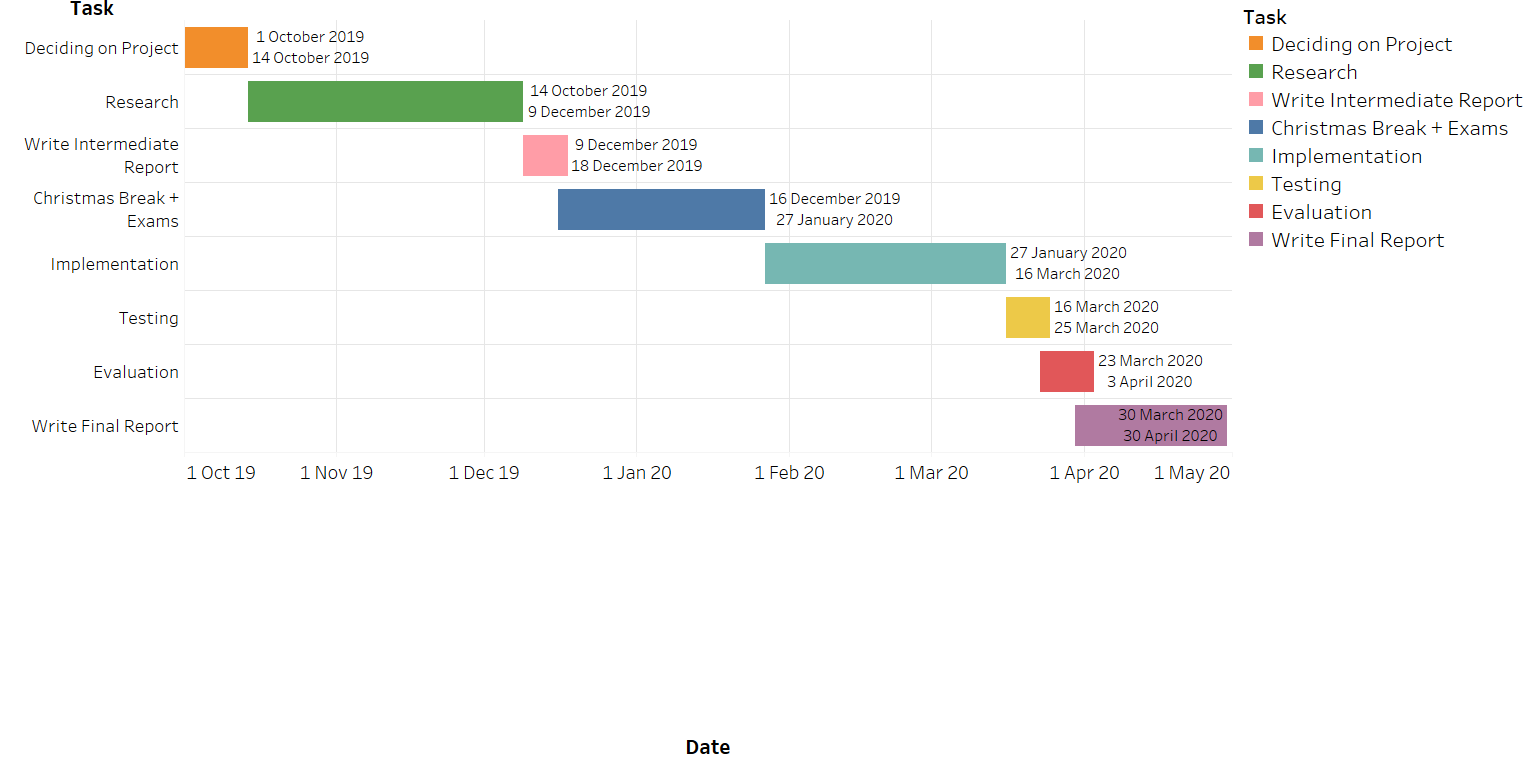
* Design and implement a shopping street layout using the Menge framework.
* Implement a way of automatically changing the obstacles in the street to random locations.
* Implement advanced behaviours you might observe on a shopping street for the agents to exhibit.
* Measure the flow rate through the street and make observations on the impact of the randomly placed obstacles.

## Deliverables

* Demonstration of the software.
* Report on the project.

## Initial Plan

The project’s panned time allocation is shown on the following page as a Gantt chart. The different areas have deliberately been left relatively vague and open ended, as it is difficult to know what will take large chunks of time in the Implementation stage for example.



## 1.5 Risk Mitigation

A key aspect of risk mitigation is guarding against the loss of work due to lack of version control. To this end, Git will be used to keep an online repository of work safe. This means that in the event of computer failure or theft, the project will be backed-up online. Furthermore the use of Git will allow the use of commit and merge points to return to in the event of faults in code; and makes developing new features easier, as branches allow for testing and prototyping before being included in the codebase.

Another important risk for this project is the general aspect of venturing into the unknown in many ways, as previous lack of experience in crowd simulation and neural networks throw up steep learning curves. Therefore there is the danger of the project’s scope becoming too large, resulting in planned features having to be cut short or revised. Because of this, constant evaluation will be needed as well as advice from the project coordinator to keep the project in check and make sure it doesn’t get too complicated and result in failure and disappointment.

## Relevance to Degree

* Graphs/ graph theory in roadmap design
* Finite automata for the bfsm behaviour stuff

# Chapter 2 - Background Research

## 2.1 Crowd Simulation

When simulating a crowd there are two major ways in which to do so. One approach, under the name of macroscopic models, offers higher realism of general large scale crowd behaviour and models the crowd as more of a gas or liquid-like object [2]. The others, microscopic models, are concerned with higher detail in individual simulation of movement, but with the downside of not behaving properly in highly dense crowd situations [1][3].

As macroscopic models are more suited to evacuation and emergency simulations, this project will attempt to implement a simulation of this type.

### 2.1.2 Menge

Menge is a modular framework developed for crowd simulation developed at the University of North Carolina [4] and has been selected for use by the project as it allows for relatively simple manipulation of the four crowd simulation subproblems discussed below.

## 2.1.3 Crowd Simulation Paradigm

A common paradigm in crowd simulation is to break the problem down into four subproblems: goal selection, path computation, path adaptation, and spatial queries [3]. Each of these subproblems need to be addressed and together they make up a crowd simulator’s basic structure and operation. Many simulators follow this abstraction, and there are others which differ slightly – combining the path computation and path adaptation steps for example. Menge, the crowd simulation framework selected for this project, provides ease of use when changing and using the four different subproblems [5].

The first subproblem of goal selection deals with what each agent wants to do, as they each need a goal to strive for when thinking about modelling how they’d move to fulfil that goal. In the case of this project, each agent will want to fulfil various goals you’d expect from a shopping street.

Path computation is how the simulator works out the path that the agent should take in order to try and fulfil their goal. A complete path for the agent isn’t always needed, and path computation can simply serve to compute the agent’s preferred direction of travel at that moment of time. Path computation can be achieved by modelling the space as a graph and then searching it using algorithms such as A\*.

WRITE MORE ABOUT ROADMAPS AND NAV MESHES HERE.

Potential fields can also be used, wherein the space is subdivided into a grid and negative goals push the agent away from them, and positive goals pull the agent towards them using gradients of cost functions. This method doesn’t compute a complete path for the agent, instead providing a preferred direction of travel, which is then used as input for the next computation.

Path adaptation describes how agents deal with changes to their paths on the fly. For example, an agent walking straight towards their goal would be able to steer around another agent coming the opposite way using path adaptation, instead of crashing into them and stopping completely.

Spatial queries refer to an agent’s ability to recognise such things as: if stimuli are in their field of view, or if stimuli are in their proximity. Effective use of spatial queries would prevent an agent from starting to run from a fire it couldn’t logically see or otherwise detect.

### 2.1.4 Scoring the Layouts

In order to improve the layouts, each one will need to be evaluated for its safety score. Each simulation will see a certain number of agents reach their goal in a certain amount of time, with an upper limit being imposed for each scenario. When evaluating the flow of a crowd, previous approaches often use a fluid dynamics approach and employ the flow-density relationship Q ( ρ) = ρV ( ρ), where Q represents the flow, ρ is the pedestrian density, and V is the average velocity [10]. However, approaches which use this equation are more concerned with the overall flow of agents through a space, and while useful, we are more concerned with generating a score for scenarios which can then be optimised. Burseth et al. [11] defined an equation we will modify slightly for calculating the score of a scenario s. This modified version is:

Where PA is the set of agents who made it to the goal before the time limit; and is the average amount of time the agents took. The modification includes the unsuccessful agents in the calculation of , by using the scenario’s upper time limit as their recorded time. In this way, scenarios in which additional agents survive are scored higher. This is important because safety is the key concern for each scenario and not just flow rate, which is what Burseth et al. [11] compute with their method.

## 2.2 Automatically Changing the Layouts

The third and final aspect for this project will be automatically changing the floor layouts. There are many different approaches to this which have been done in the past using techniques like Bayesian Networks [6] Generative Adversarial Networks [7]. Procedurally generating them is a possible approach but has not been chosen as we would like to see the computer start to learn or appear to learn how to make floorplans more safe. It is with this in mind, then, that we look to neural networks.

Menge’s architecture relies on a ‘Scene’ file to specify the locations of a simulation’s obstacles, inputted as sets of vertex coordinates [5]. It is these coordinates which will be changed automatically for each simulation and scoring attempt, as these obstacles’ coordinates will dictate the layout.

## 2.2.1 Generative Adversarial Networks

A Generative Adversarial Network (GAN) is a type of machine learning system in which two neural networks compete in a game. One of the networks, the generative network, generates new objects (such as images), which are then judged by the other network, the discriminative network. The discriminative network’s goal in the game is to identify which objects are from the real data set and which have been made by the generative network, while the generative network’s goal is to produce objects which fool the discriminative network, making it think the generated objects are in fact from the data set [8].

The discriminative network is trained on an existing data set, and the generative network is trained according to how well it manages to fool the discriminative one. Through the use of Backpropogation, the generative network is able to improve at generating new objects whilst the discriminator improves at catching the fake ones. In this way the generator is eventually able to manufacture objects which are extremely like those in the original data set. The power of GANs has been shown by the use of a particular GAN, StyleGAN2, which can produce incredibly realistic portraits of fake human faces which are indistinguishable from real ones [9].

Using a GAN for the floorplan generation is tempting due to their power and novel concept. A potential pitfall using it and any other kind of neural network however is the dataset it’d be trained on, which would need to be generated beforehand.

# Chapter 3 – Doing It

## 3.1 MengeUtils

An invaluable resource has been MengeUtils, a “Collection of utilities for working with Menge configuration and output files”(CITE MENGE UTILS) created by one of Menge’s co-creators. These include a “Scenario Authoring” tool and a “Crowd Analysis” tool. The Scenario Authoring tool allows for viewing and modifying the positions of goals, agents, obstacles, and roadmaps, which is extremely useful when testing any of those aspects. The Crowd Analysis tool is for performing anaylsis on the movements of agents after a simulation has been run, providing data and graphs for a crowd’s flow, density, population and more.

Both of these tools were used extensively when setting up the simulation and for analysing the results post simulation.

## 3.2 Prototype

Menge Utils

Running AnalysisTask.py through command line

Applying Score Equation to output

Automatically change the obstacles

## 3.3 Shopping Street

### 3.3.1 Layout

The first task for simulating the street was to design the simple base layout. It was designed to be 100m long, with shops either side of the 15m wide space in the middle. The decision was made to keep the layout and size of the shops and street constant, as the project is concerned with how changing the positions of objects in the middle of the street affects flow rather than the position of shop doors or the size of the shops, for example. The shop doors are however specified to be 0.70m wide, which is the standard in the UK, (CITE HERE) to make the simulation more realistic.

The randomly placed objects in the middle of the street have their bottom-left vertex constrained within the space represented by the blue box. This means that there will always be a minimum gap of 4m between the objects and the shop fronts, and a 5m gap between the objects and the end of the street. This is to ensure that shop doorways aren’t blocked by obstacles as well as to improve the realism of the simulation - as for a real street, people would be left space between the shop front and obstacles to move freely in. The basic layout can be seen in Fig. [FIGNUM1] below. The shops’ exterior walls is displayed in red and the area in which a random obstacle’s bottom-left vertex can be placed is shown in blue.



100m

15m

### 3.3.2 The Random Obstacles

Two types of objects very often seen in the middle of pedestrianised shopping street are benches and large units, used perhaps to hold a winnable car on display, or house a small food stall or small accessories stall. These are therefore the two types of objects which are generated and randomly placed in the middle of the street before every simulation.

They are generated in much the same way as the one in the prototype. First, the bottom-left vertex’s x and y coordinates are created using python’s built-in random library, constrained to be within the area shown by the blue box in Fig. [FIGNUM1]. Then, the other vertices’ coordinates are simply calculated from that anchor point, depending on what type of obstacle is being created. The large units are all 5m squares and the benches are 2m long and 0.5m wide to allow room for agents to sit at them. The benches are also randomly made to be either portrait or landscape in orientation. Using the shapely Python library [CITE HERE] each object’s placement is checked so that none of them intersect or touch.

### 3.3.3 Navigation

From the prototype, it was discovered that agents could easily become stuck behind obstacles when trying to reach their goal. The agent’s at the time were using the ‘goal’ option for their Velocity Component which meant that they’d try to move in a straight line towards their goal, regardless of obstacles. Further research revealed that certain navigation structures can be used so that agents can better plan the route to their goal such as roadmaps, navigation meshes, and … (CITE HERE). After reading about the format in which Menge stored navigation meshes and the more involved method of generating them, it was decided that using a roadmap would be both easier and adequate.

After some initial testing of roadmaps, it became clear that for an agent to effectively use them to plan their routes to goals, the roadmap needed to be unobstructed by any obstacles – meaning there can’t be any intersections between obstacles and the roadmap graph.. This meant that using a static roadmap was infeasible due to the random positions of the obstacles. Therefore, the roadmap needed to be changed every time the obstacles were.

[ADD PIC OF AGENT TRYING TO GET THROUGH AN OBSTACLE]

#### 3.3.3.1 Updating The Roadmap

The method random.randint() only generates whole numbers for the obstacle’s positions and none of the obstacles can touch. This means that a square grid graph can be used for the roadmap, allowing agents to pass around the obstacles with ease so long as the parts of the roadmap which intersect with the obstacles were removed. After writing a simple python script to generate the text file describing the grid in the middle of the street, the roadmaps for each of the shops were drawn using the Scenario Author tool from MengeUtils[CITE?] and joined to the grid.

Using graph.py from MengeUtils[CITE?], the roadmap can be read from the text file and made into a Graph object. Then, the Python script modifyRoadmap.py was written to compare each of the roadmap vertices with the obstacles’ vertices, deleting the roadmap’s vertices and edges which caused it to intersect with any of the obstacles. This resultant roadmap graph, free of any intersections, is then written back to graph.txt, ready to be used in the next simulation.

In Figures 1-3 below, the process of modifying the roadmap to fit around the obstacles is visualised.

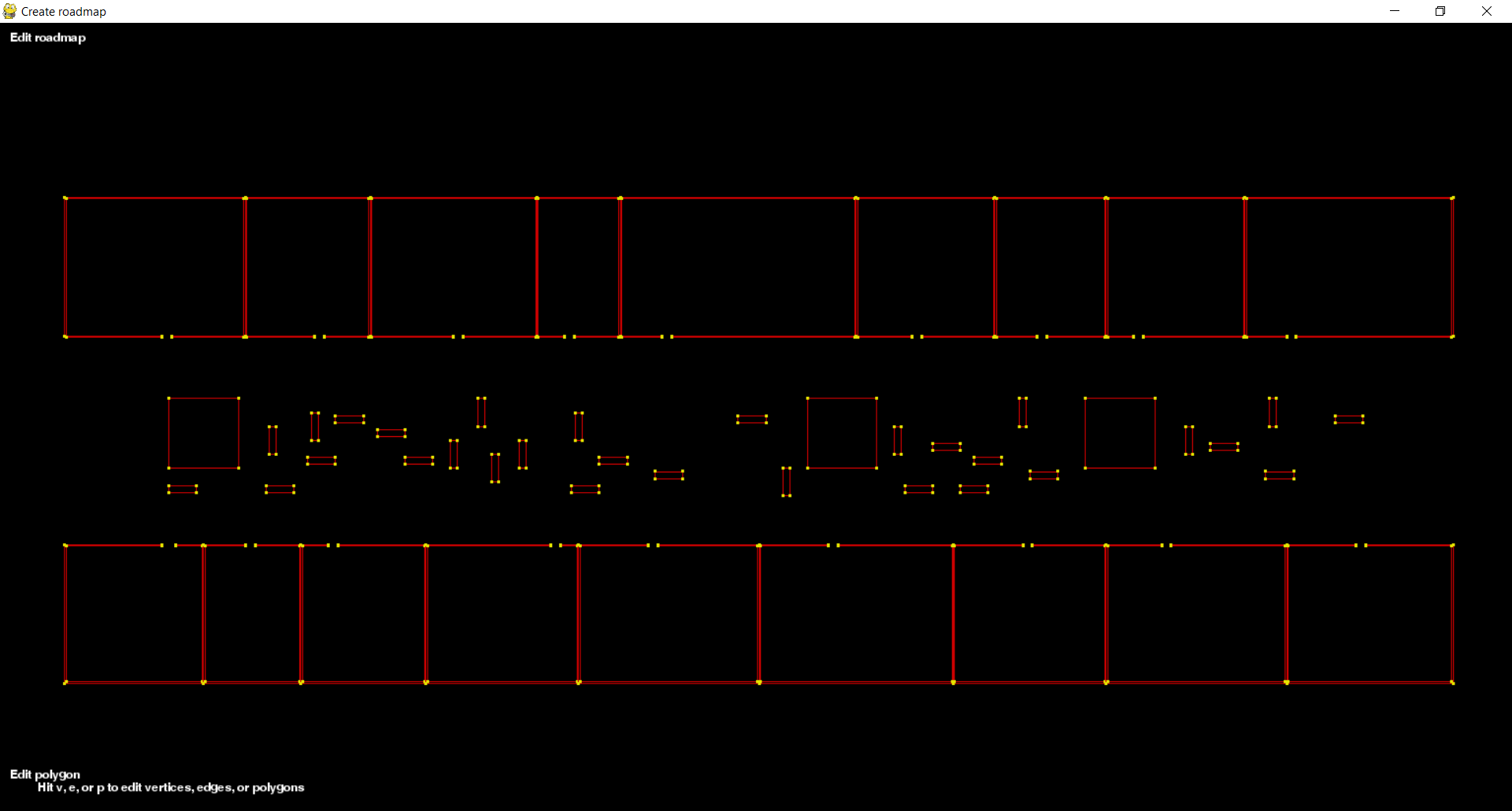


Figure 1: Placement of random obstacles

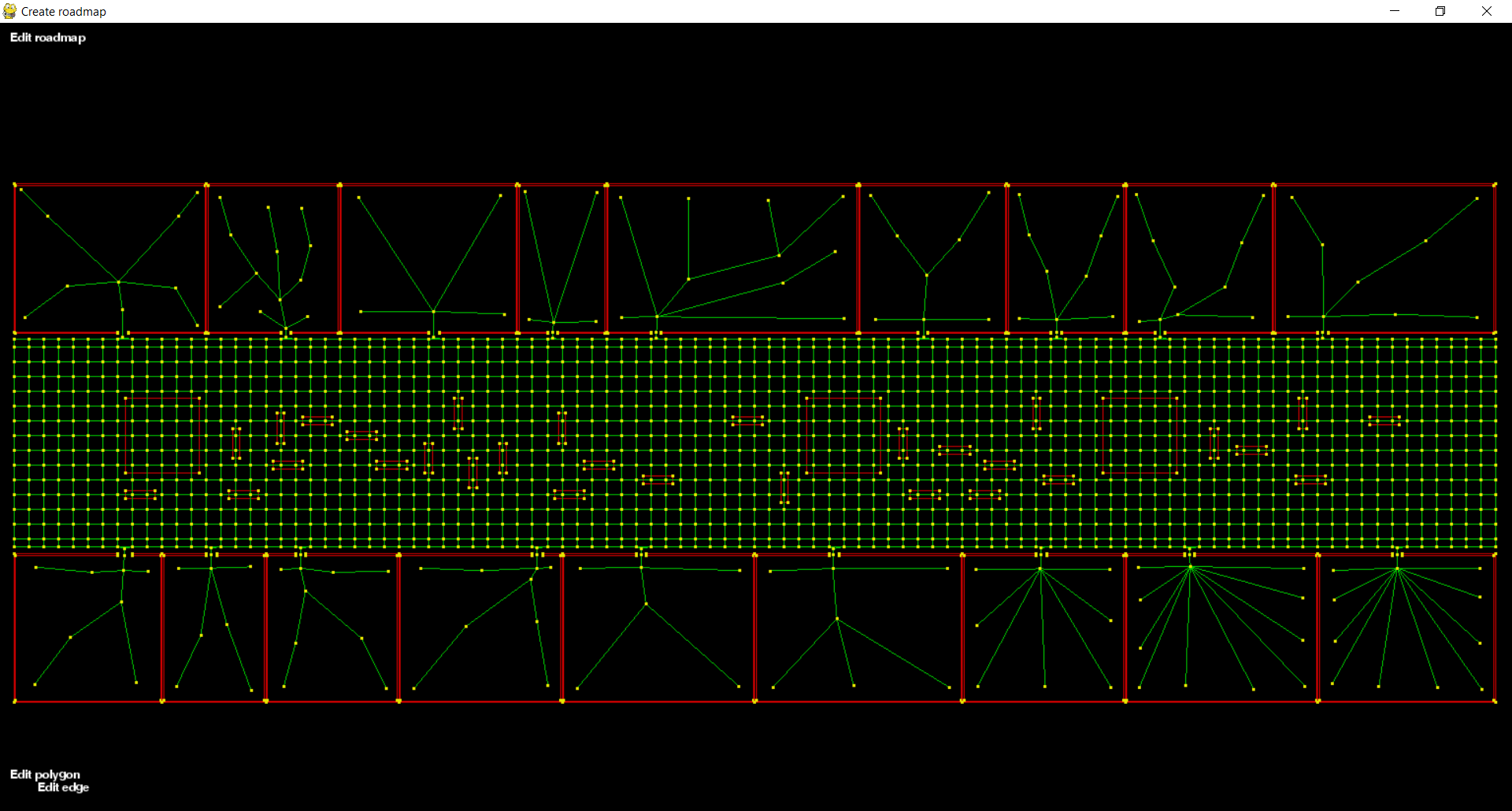


Figure 2: The full roadmap shown with green edges and yellow vertices, intersecting the obstacles

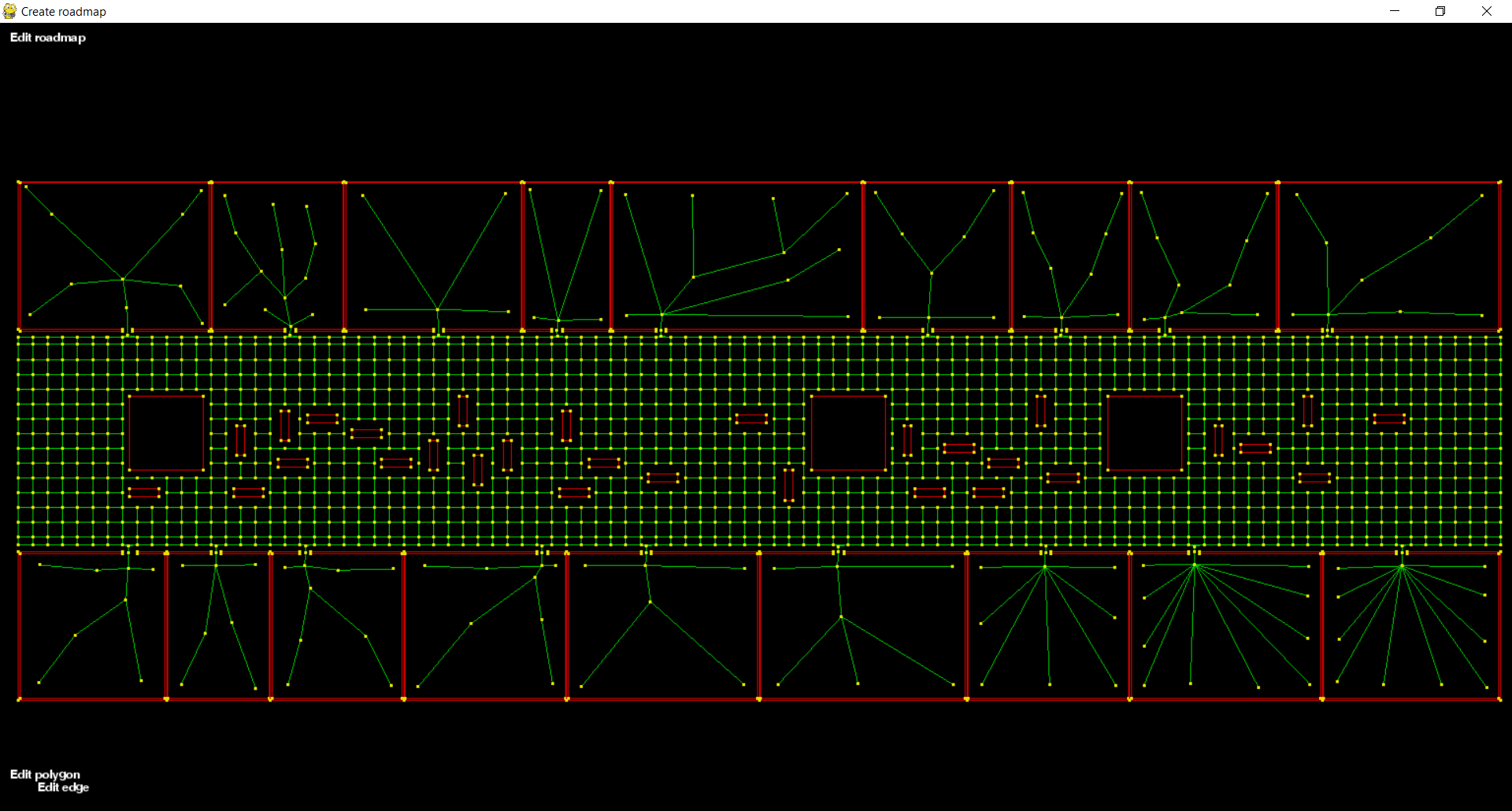


Figure 3: The roadmap after being modified

### 3.3.4 Behaviour

Paramount for useful results is the careful design of agent behaviour to result in more realistic simulations. To this end, the agents in the shopping street simulation’s behaviour was designed so they do a variety of different tasks, including: walking to the end of the street, window shopping, spending time in a shop, sitting at a bench, and looking at/visiting one of the large units. This behaviour, like the scene specification, is written in an XML file and is modelled as a Behavioural Finite State Machine (BFSM) (CITE MENGE HERE). In this way, each agent is in one state at a time, each having instructions on how and when to transition to the next state, such as an agent transitioning, after a timer, from the state “AtBench” to one of the other behaviours randomly - with each having a different probability.

The BFSM can be visualised using Figure[FIGNUM] below. The blue boxes represent states, the grey circles are transition targets, the black arrows are transition conditions, and the white diamonds are simply where arrows merge/split to keep the diagram less cluttered. Where a state also involves having the agent move or, in the case of reaching the end of the road, be teleported, it’s written below the state name.

When the simulation starts, the agents are randomly assigned to one of the following: walking to the end of street, either the right or left side; beginning to window shop; going to sit at a bench; or going to visit one of the units. Through the use of timers of varying lengths agents will appear to sit at benches, visit units, look through shop windows, and spend time in shops for differing amounts of time before transitioning to their next state. Certain details have been left out of the diagram to preserve simplicity: agents who are teleported to the right side of the street won’t try and walk right again (and vice versa for those walking left); agents who have just sat at a bench won’t immediately do so again; and agents who have just spent time in a shop are more likely to sit at a bench or go to a unit. These changes in probabilities serve as an attempt to make the simulation more realistic, thus acquiring better results.

Having the agents be automatically teleported to the other side of the street is an incredibly convenient way of maintaining a sense of continuous flow to the simulation. The drawback however, is that the population of the scenario remains constant throughout each simulation, which isn’t realistic.

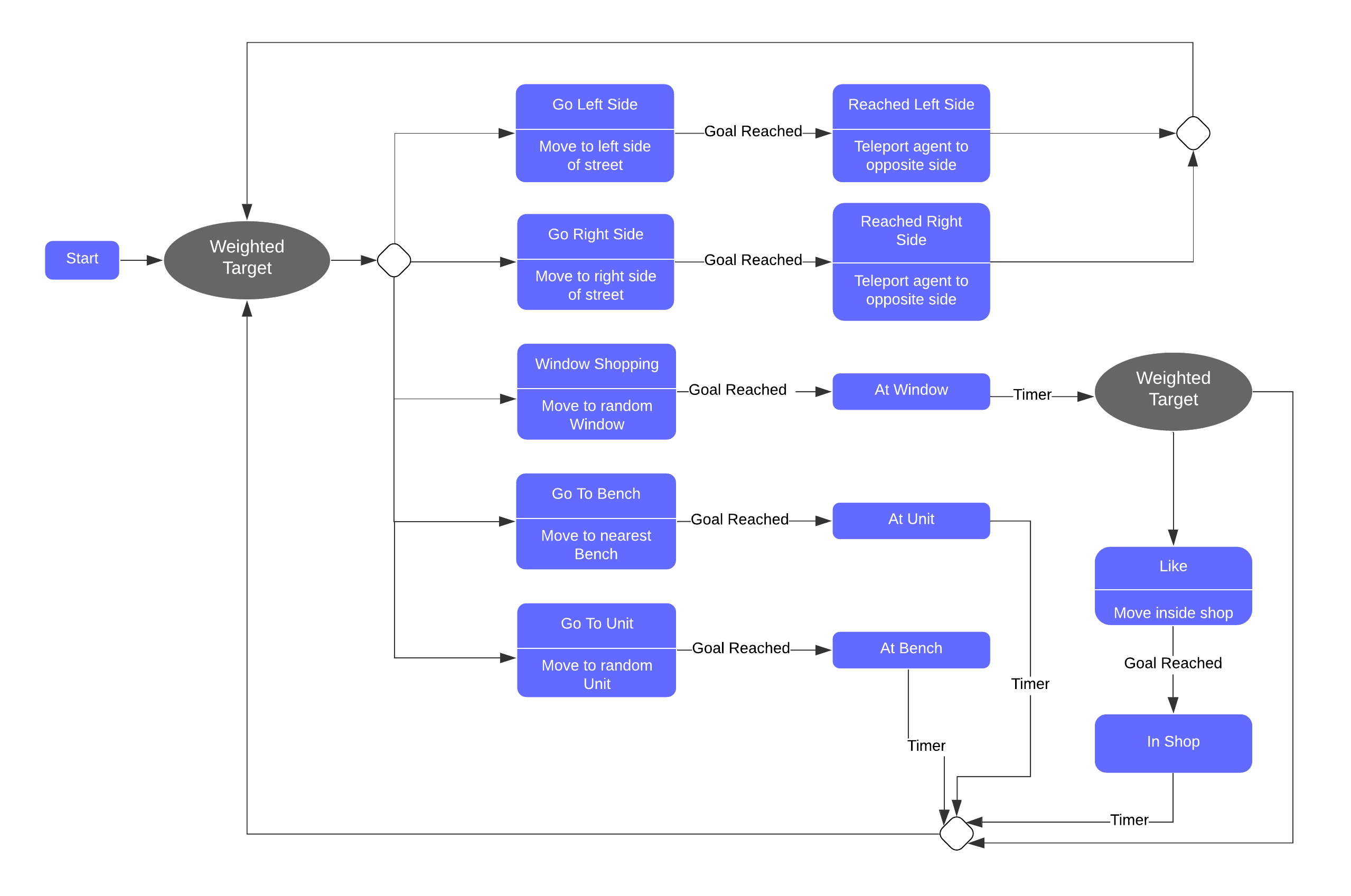


Figure 4: Diagram showing the BFSM used for the shopping street scenario [at unit and at bench wrong way round]

### 3.3.5 The Agents

To populate the simulations, different agent profiles were created to try and accurately represent a typical group of people found in a city centre. This meant that the simulation contained agents who had different preferred walking speeds, masses, and radii shown in table[TABLENUM] below.

Table 1: The different agent profiles and their characteristics

|  |  |  |  |
| --- | --- | --- | --- |
| Agent Profile Name | Mean Preferred Speed (m/s) | Mean Radius (m) | Mass (kg) |
| Adult man | 1.4 | 0.23 | 80 |
| Adult woman | 1.4 | 0.20 | 70 |
| Elderly man | 0.95 | 0.225 | 75 |
| Elderly woman | 0.95 | 0.195 | 65 |
| Teenager | 1.35 | 0.19 | 55 |

Each of the agent’s preferred speed and radius changes with each simulation. The values for each agent are calculated using a normal distribution with the values shown above. The standard deviation for these is constant for all agent profiles: 0.03 for the preferred speed trait and 0.01 for the radius trait. (CITE SOURCES).

### 3.3.5 Putting it all together

Talk about the main.py file and how it does everything in order.

The layout – justify proportions, relate it to real world setting (Victoria, Leeds),

Agents – goal selection, justify radius (width of human body)

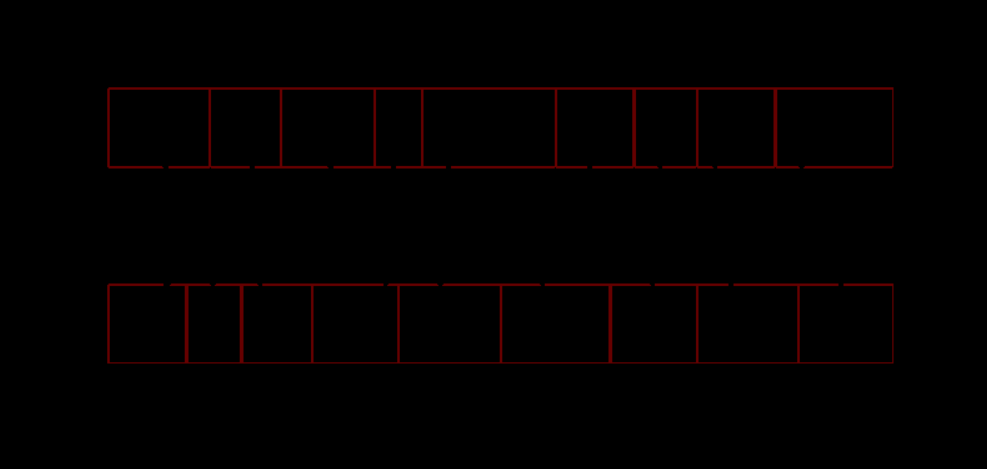
Average people dimensions: <https://www.firstinarchitecture.co.uk/average-male-and-female-dimensions/>

Average people weight: <https://www.onaverage.co.uk/>

Calculating acceleration <https://www.omnicalculator.com/physics/acceleration>

TODO: make a table with all these values.

Measures – flow lines



# Chapter 4 – Experimentation

## 4.1 Flow

# Chapter 8 – Self Evaluation

## 8.1 Could have done differently

* Have the different groups of agents more be likely to do different things. E.g. old people sit down more
* Use more types of obstacles – bins?.
* Introduce randomness to the teleportation- have agents off to the side who randomly telport in/out to simulate rising/falling population of the street. Or don’t have the teleports just go to other side – make that random too.
* Different placements of benches for example: like benches should probs be able to touch along their back sides etc.
* Whole thing is in python 2.7

# Chapter 9 - Ethics

As this project will involve software to evaluate the safety of different building layouts, ethical issues are non-trivial. It’s in this project’s hopes it could help develop a more sophisticated program that could be used by an architect to make a building as safe as possible. However, if a building was constructed based on a design supplied by the program and a fire caused human harm within it, serious ethical concerns could be raised. Who would be responsible should the design be found weak to fires? Some parties may claim that it was the builders, the architect, or potentially the program’s author – as they may claim it could have provided a false sense of security to the architect using it.

As such, if something like this project were ever to be used by designers, legal measures may have to be put in place to make sure that no writers of the code would be liable if something like the previously described event occurred. The author of the program would likely not be accredited by the Royal Institute of British Architects (RIBA) and so shouldn’t be held accountable – the onus is on the architect, engineers, and construction workers to ensure a building’s safety.

# List of References

[1] Jin, X., Xu, M., Jiang, H. and Deng, Z., 2016. Crowd Simulation and Its Applications: Recent Advances. [Online] Available from: <https://link.springer.com/article/10.1007/s11390-014-1469-y>

[2] Henderson L. The statistics of crowd fluids. Nature, 1971, 229(5284): 381-383.

[3] Funge, J., Tu, X., And Terzopoulos, D. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In Proc. of SIGGRAPH, 29–38.

[4] Curtis, S., Best, A., and Manocha, D., 2013. Menge. [Online] Available from: <http://gamma.cs.unc.edu/Menge/>

[5] Curtis, S., Best, A., and Manocha, D., 2015. Menge: A Modular Framework for Simulating Crowd Movement. [Online] Available from: http://gamma.cs.unc.edu/Menge/files/MengeTechReport.pdf

[6] Merrell, P., Schkufza, E., & Koltun, V. 2010. Computer-generated residential building layouts. *ACM Transactions on Graphics (TOG)*, *29*(6), 1–12. <https://doi.org/10.1145/1882261.1866203>

[7] Chaillou, S. 2019. AI + Architecture | Towards a New Approach. [Online] Available from: <https://towardsdatascience.com/ai-architecture-f9d78c6958e0>

[8] Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014) Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.

[9] <https://thispersondoesnotexist.com/>

[10] Helbing D, Johansson A, Al‐Abideen HZ. Dynamics of crowd disasters: an empirical study. Physical Review E (Statistical, Nonlinear, and Soft Matter Physics) 2007

[11] Berseth, G., Usman, M., Haworth, B., Kapadia, M., & Faloutsos, P. (2015). Environment optimization for crowd evacuation. Computer Animation and Virtual Worlds, 26(3-4), 377–386

Ulicny, B. and Thalmann, D. (2002), Towards Interactive Real‐Time Crowd Behavior Simulation. Computer Graphics Forum, 21: 767-775. doi:10.1111/1467-8659.00634 [SOURCE FOR BFSM]

# To look at

## To read (might be useful)

[23] Hutton, A.: London bridge station, the role of ped modelling: Pedestrian modelling 33 and design development. In: 6th International Conference on Pedestrian and Evac34 uation Dynamics. Zurich, Switzerland (2012)