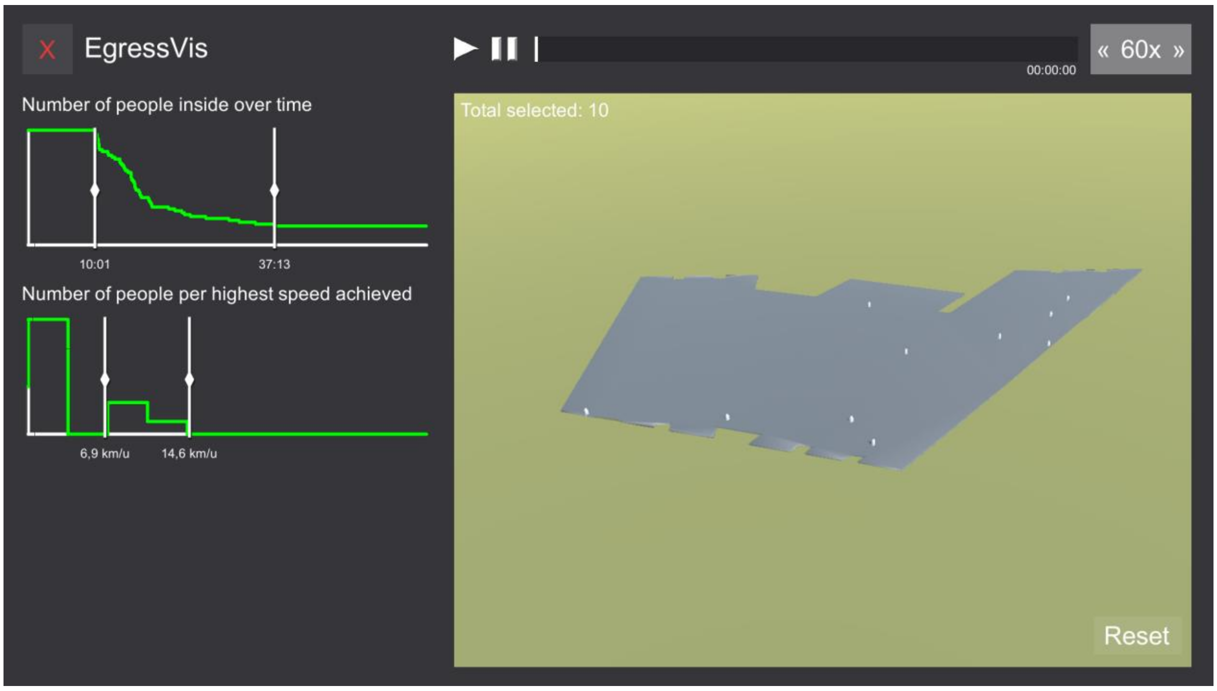
Modelling and Visualizing Emergency Egress

Sander Koning


X
EgressVis
Number of people inside over time
10:01
37:13
Number of people per highest speed achieved
6,9 km/u
14,6 km/u
00:00:00
« 60x »
Total selected: 10
Reset

BSc. Creative Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

University of Twente

Supervisor: dr. J. Zwiers

Critical Observer: dr.ir. P.W. De Vries

Abstract

As a graduation project for Creative Technology at the University of Twente, a novel visualization was made for the department Psychology of Conflict Risk and Safety to aid in research into emergency egress. This report describes the research done prior to, and the systematic approach to building the end product using iterative design. The product is evaluated with a set of assignments, and the well-researched SUS-test.

Table of contents

1. Introduction

1.1 Problem statement

During emergencies certain precautionary measures can increase the survival rate of those caught in the mayhem. Preventive measures, for example a sprinkler installation, could stop a disaster from ever happening. If such precautions may not be able to deter further escalation, evacuation is often undertaken to ensure the safety of those on the premises.

The success of an evacuation is dependent on the availability and visibility of exit routes. Often, a space is built with this in mind, and emergency escapes are built in, e.g. an emergency hatch in the headliner of a bus, or an external staircase on a building. In order to test the visibility of those routes, fire drills are a common way to do so.

It is often believed that people behave irrationally during an emergency, but those ideas are challenged [1]. Previous research is often limited to questionnaires after the evacuation, when a varied amount of time has passed. This reduces the impact of the results, as they can be altered by clouded memories, and purposefully untrue answering, e.g. when a respondent feels too proud to admit panic. Little research has been done using actual recorded data.

The research group at the department of Psychology of Conflict, Risk and Safety (PCRS) of the University of Twente is interested in testing the theories about the rational thinking during evacuations. Indoor positioning using Wi-Fi data we can deduct the paths people took during an emergency egress, and certain behavioural characteristics, for example walking speed. This could provide an insight in how efficient people behave when the alarm goes off.

1.2 The current research

The aim of this study is to support research of behaviour during emergency egress by means of data visualization. We will develop and describe a visualization of (generated or real) Wi-Fi data gathered during an evacuation, in such a way that it is possible to analyse the movement of evacuees on an individual level, on which in future research proper analyses can be performed. For this, the factors of the visualization that determine whether or not valuable information can be deducted should be discovered, and behavioural characteristics (e.g. chaotic movement of an individual and flocking behaviour) that are interesting to visualize should be summed up. The visualization has to be tested with the client.

The research question for this study is: ‘How can we develop a visualization on crowd behaviour on an individual level, supporting research of behavioural characteristics during emergency egress?’. A sub question is: ‘What behavioural characteristics of individuals vs. group behaviour are interesting to visualize?’

2. State of the art on modelling and visualization of emergency egress

2.1 Background research

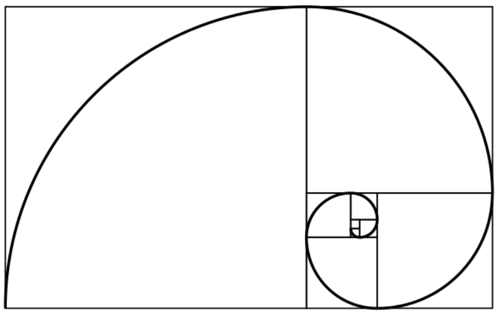


Figure 1: The Fibonacci spiral is a lot easier to understand than a seemingly random row of numbers like; 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 (actually pictured above) [2]

To obtain insights in data, such as indoor positioning data during emergency egress, we make use of visualizations in an efficient and effective way. It simplifies the complexity of data [3], and elicits interesting patterns and features in a short time. Visualization is a non-verbal method to communicate. It can be used to: show things that are incomprehensible in its usual form (e.g. too small, too large, too far); to make vast amounts of data easily understandable; clarify difficult or abstract concepts; and describe complex relations in a more understandable way [3]. With visualization, we make use of the unique capabilities of the human cognitive system [4], [5]. Moreover, it automatically uses the human system for pattern finding, the visual working memory, and the spatial working memory. It is suggested that using multiple subsystems aids in the learning process [5].

2.1.1 Effectiveness of modelling and visualizations

Visualization is a concept with a rather broad definition, as it can mean the research discipline, a technology, a specific technique (i.e. as part of a wider technology), or the visual result. We will consider visualization as a technology, which includes a collection of methods, tools, and techniques, developed to satisfy a need. Hence the standard measures apply: Visualization has to be effective and efficient. It must achieve the goal it was chosen for, and do so with minimal cost. This obviously implies that we have to take into account the context in which it is used [4].

Although exploiting human visual and cognitive systems benefits the processing of information, there are things to take into account. Our long-term memory is almost infinite, but our working memory is extremely limited, according to the Cognitive Load Theory [3]. The Gestalt psychology paradigm holds clues to reduce the load on our working memory, by means of a set of rules, or laws, that make sure that the visualization and the cognitive processing are in sync. [3]

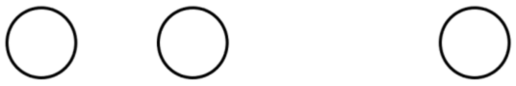


Figure 2: It appears there are two circles on the left, and one on the right.

For example, the Law of Proximity dictates that objects visually close to each other are perceived as a group. By making sure that they are in fact grouped, the cognitive system of the viewer is alleviated of going through the trouble of manually grouping objects. See Figure 2: It appears that there are two groups of circles, one on the right, two on the left. If these were supposed to be one group as a whole, they should have been at equal distance. The same holds for colour, shape, direction, among many more.

2.1.2 Methods of modelling and visualization of geometric data

A good start on developing a visualization is describing the type of data first. In our case that is location data. Where we are inside a building can be mapped to where we are on Earth, which in turn can be mapped to where we are in the solar system, and so forth. This data we call geometric, as it can be described by a system of coordinates.

For centuries, if not millennia, we have been trying to visualize these data and it is of no surprise there is a science discipline formed around it; cartography. Modern day cartography tries to find a good way to map 3D coordinates onto a 2D plane (e.g. Miller's projection, the Mercator projection) [6].

There are several methods to modelling geometric data (i.e., 2D modelling, 3D modelling and 2½D modelling). For instance there is 2D modelling, which is in essence just drawing a map. This may be useful in single-story buildings or situations, but for multi-story scenario's this may prove inadequate. If movement is perpendicular to the plane of view (e.g. in an elevator), it may appear as if there is little to no movement. Running up and down the stairs in panic would also skew the perception of the visualization.

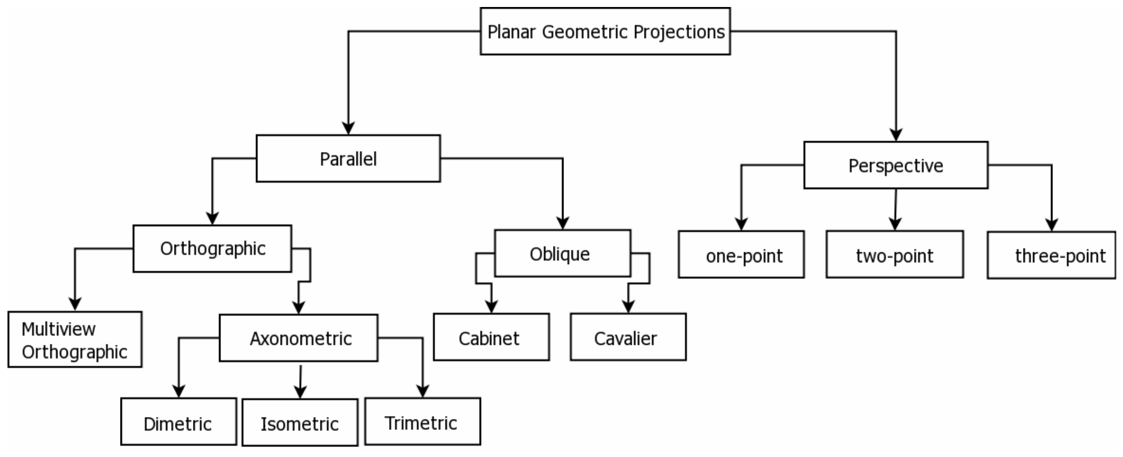


Figure 3: An overview of geometric projections, as presented by Erlingsson [7]

Another option is true 3D modelling. As seen in Figure 3 under *perspective*, there are variations to this, but they have one thing in common: vanishing point(s) [8]. Objects closer to the vanishing point appear smaller. Much alike perception from the human eye: objects further away appear smaller. Although this would solve the hiatuses left by the 2D projections, the information gained from the visualization will still be distorted.

Somewhere in the middle lies 2½D modelling. Perhaps best described as using 2D elements on a flat plane, to mimic a three-dimensional projection. By this definition, rendering 3D content on a flat screen can also be considered 2½D, but for this project. In this paper the term 2½D and parallel projection (more specifically axonometric projection) are strongly interrelated, to the point that at times they could be used interchangeably.

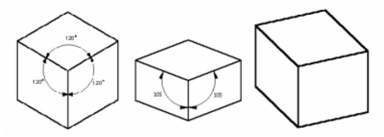


Figure 4: Axonometric projections of a cube, as presented by Erlingsson [7]

Parallel projections are projections that do not change the size of objects as they are positioned. This eliminates the problem of vanishing points, as it exists in perspective (3D) projections. A good example, and perhaps the most usable are the axonometric projections. Axonometric projections show, in general, 3 faces of an object, without converging lines, or lines that appear shorter depending on their position relative to the viewer.

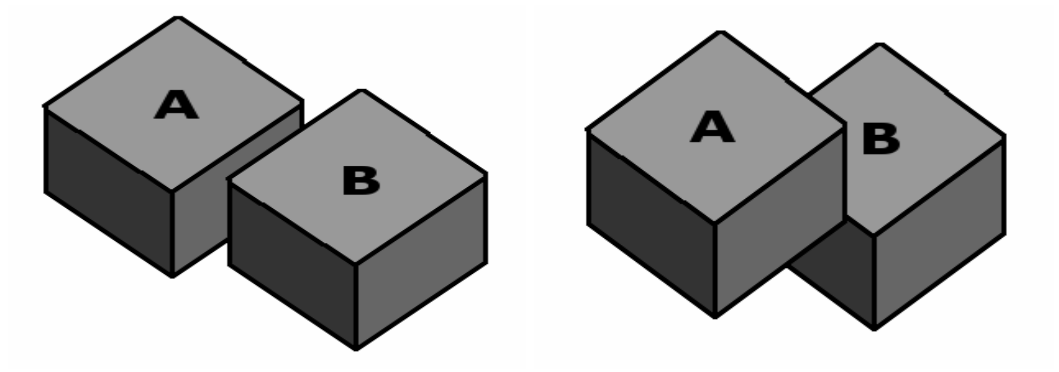


Figure 5: Which object is nearer is defined by which is 'on top'. In the top image ***B*** *appears closer, in the bottom image* ***A*** *is closer. As presented by Erlingsson [7]*

Inevitably, this means that objects of equal size, are projected with equal size, no matter their position. What object is nearest to the viewer is determined by something as simple as whichever is 'on top'. This is demonstrated in Figure 5.

2.1.3 Conclusion and problem analysis

We need to construct a visualization, so that information can effectively and efficiently be deducted. We must make maximal use of the cognitive abilities of the human brain, and reduce the clutter so that the cognitive load is minimal. This should lead to profound insights and will aid research into the topic. The Gestalt Laws hold clues as how to achieve this.

Furthermore, the visualization is best projected in a 2½D, or in fact axonometric projection. This should eliminate loss of information in depth, either by lack of differentiation in height differences, or by diminishing sizes as objects are positioned further away from the viewer.

Combining knowledge about Gestalt laws, and the Cognitive Load theory, with the insights we have gained in geometric projections, we should construct a view that is devoid of clutter, but does not make concessions on the trustworthiness of the information projection.

2.2 State of the art review of related works

2.2.1 Games

A prominent example of axonometric projections comes in the form of games. In the early days, when graphics processors were barely, or not at all capable of 3D processing, there was a demand for 3D graphics anyway. One technique often used to create the illusion of depth is parallax scrolling: a technique where foreground images move faster than images in the background. This technique is well exploited in (early) sidescrollers [9].

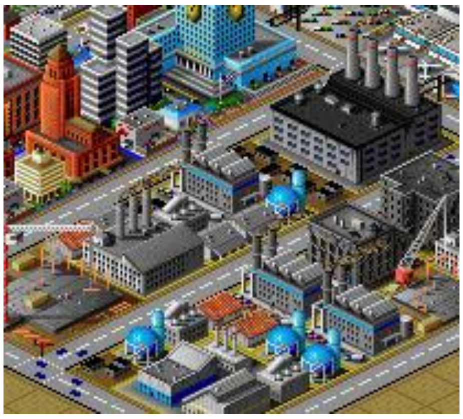


Figure 6: Screenshot of the playfield of SimCity2000, released in 1993

Another technique was found in the named thereafter tile-based games. Dividing a game field into tiles (rectangles) and drawing sprites (basically images, or parts of images) on those tiles. Notable games include SimCity2000, as can be seen in Figure 6. To the untrained eye it might be hard to differentiate single tiles, but a tile can be distinguished when looking at the lower corners of the screen. A tile is diamond shaped. Streets are a single tile wide, so looking at a crossing, the square (diamond) shape that is the actual crossing, i.e. part of both crossing roads, is the size of 1 tile. Looking at the buildings, we can distinguish at the floor level, that buildings are a one, two or three tiles wide, but never a fraction of 1.

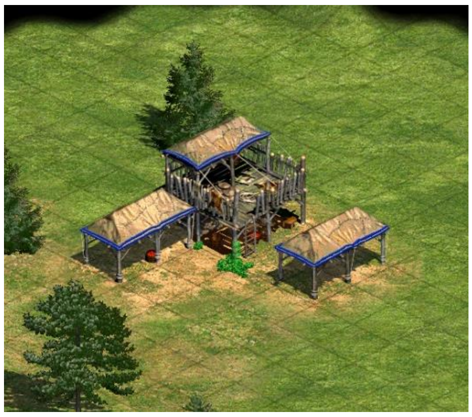


Figure 7: Even with updated high-definition graphics, Age of Empires II is still tile-based: a simple change of graphics shows a clear grid that shows where the tiles are situated.

2.2.2 Architectural design

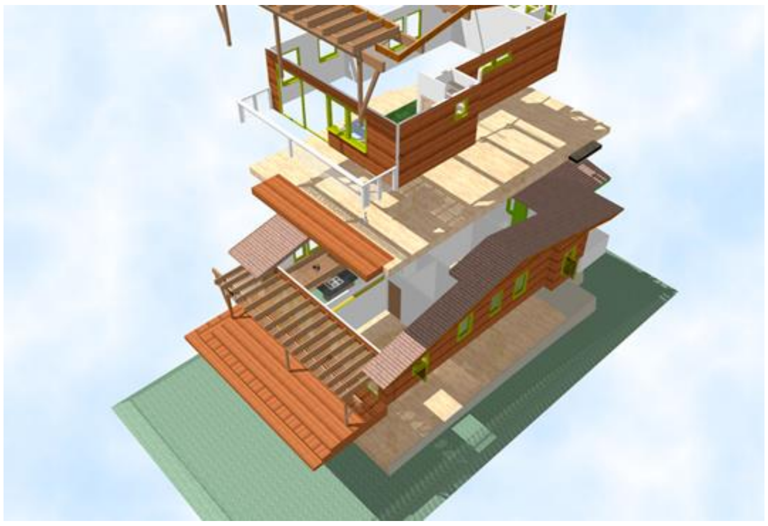


Figure 8: Although not in the strict sense axonometric, this building is rendered with a greatly reduced perspective effect. The floors are separated, forming an 'exploded' view. [10]

In architectural design, drawings are often made in (near) axonometric views. This is done not to differentiate details in the 'back' of the building, with those in the 'front' of the building. It provides a view of multiple faces, without the distortion that would have occurred if a perspective view was chosen. It helps the human visual system to, for example, compare walls that are on opposing ends of a room.

Although this technique traces back to ancient Greek vase paintings [11], computer graphics have certainly boosted the capabilities. If a building has been designed in a computer aided design programme, a CAD-programme for short, it is often really easy to produce axonometric views.

2.2.3 Indoor tracking applications

Certain businesses are using indoor positioning and tracking systems to keep track of their personnel and devices. For instance, on a large airport, it can save time and money to know where certain devices are. From luggage trolleys, to cleaning machines, knowing where they are means saving time on searching for them.

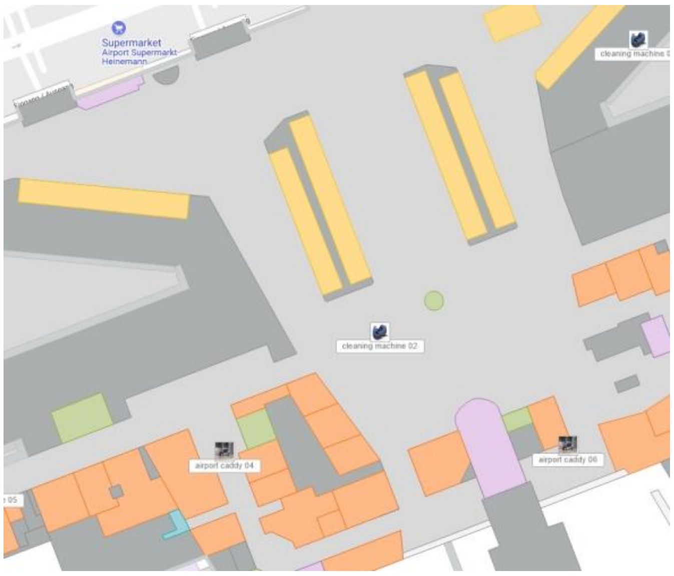


Figure 9: infSoft's dashboard for indoor tracking, using an airport as an example [12]

infSoft is a company that specializes in indoor tracking applications [12]. Although they sell different solutions based on a variety of sensors, they claim to have a system that uses only Wi-Fi data [13]. They claim that an accuracy of 5-15 meters is achievable using Wi-Fi alone. If this is enough for our application remains to be seen, but perhaps it is possible to improve on this. If their solution does no more than just overlay an image (e.g. a simple dot) over a map, and thus the illusion of someone travelling through walls is possible, there would be an obvious way to improve on this. However, their interface is related to what we want to achieve, because we want to display routes of individuals inside a building as well.

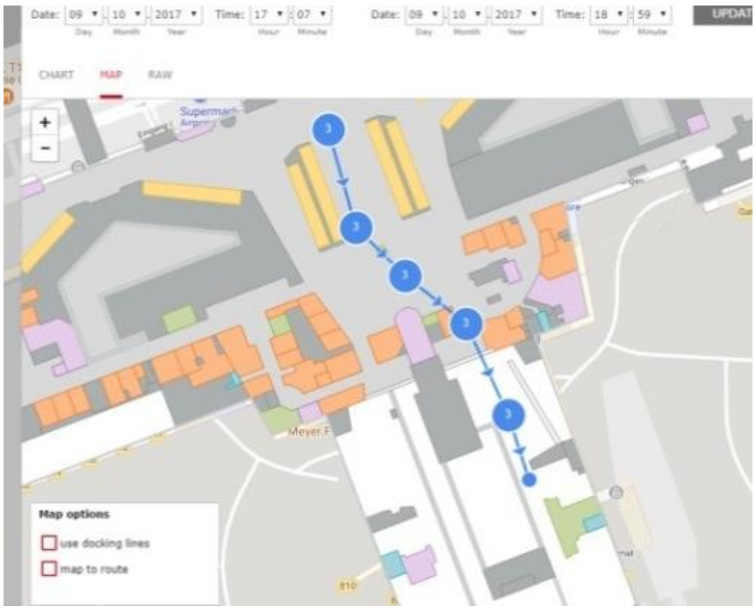


Figure 10: Displaying position over time in the infSoft dashboard

The dashboard seems to have integrated a feature that displays the position over time. The screenshot they provided seems to confirm earlier suspicions, namely that they don't filter routes to the point that the paths taken are actually possible. This is for their use cases not a problem I suspect, but it does leave room for filtering techniques, as to improve on their 5-15m accuracy.

As for the visualization: it is very basic. Nothing has been done to make the interpolated path look natural, as it is just a game of connect-the-dots, where the dots are recorded locations. However, it shows a clear arrow to display the direction in which the person or device has moved. This takes away the ambiguity what the start and endpoints are.

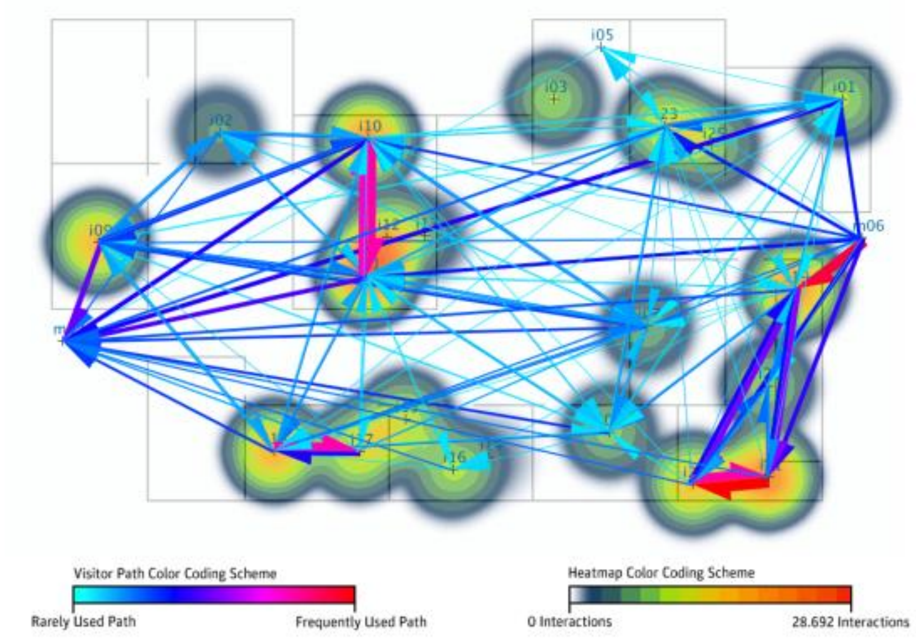


Figure 11: A visitor flow diagram of a museum exhibition [14]

During the exhibition 'Heart over Heels' by 'FRida & freD – Graz Children's Museum' data was collected using RFID tags on 17 interactive exhibits. A tag was used by visitors to save progress and data during their interactions. For a conference paper on using visitor-flow visualization to improve visitor experience in museums and exhibitions, this data was used to produce the map in Figure 11. The first thing we notice is the not-so-clear spaghetti of arrows on the screen. This amount of clutter is something we have to avoid in this project. The second thing we notice is a heat map, which unfortunately is not the most helpful in displaying paths people took.

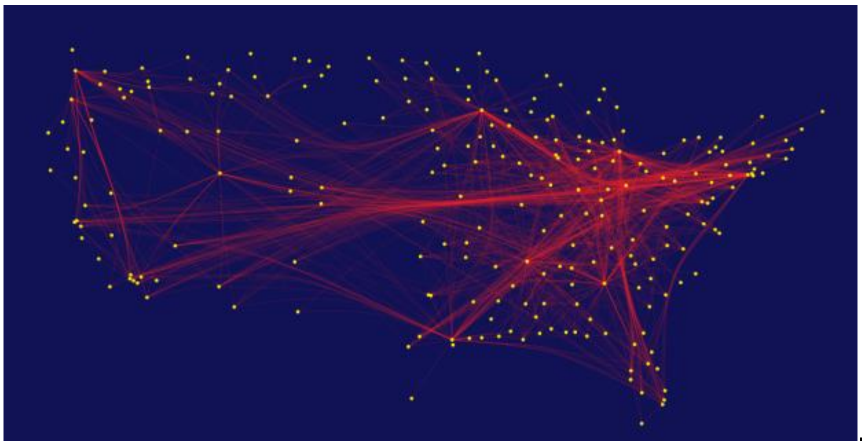
.

Figure 12: A d3.js visualization using edge bundling [14]

A visualization tool, called d3.js, is capable of visualizing a myriad of types of information. When it comes to paths, this is an interesting visualization that intentionally alters paths for a clearer visualization [15]. The technique is called ‘edge bundling’. It is primarily useful when visualizing a large amount of paths, as it ‘binds’ similar paths together, as can be seen in figure 12.

2.2.4 Modelling 3D from camera feeds

From a completely different data source (i.e. camera feeds), a novel approach comes from Roth et al. [16], who developed a surveillance system. Traditionally, security personnel had to keep an eye on multiple camera feeds, and depend on cognition for this person to track another person across screens. This requires prior knowledge about the positioning of cameras, and can be a large mental strain [16]. From the philosophy that information should be visualized in one environment, they offer an alternative. They developed a 3D model of the building, and projected camera feeds onto the surfaces (e.g. walls, floors) so that they would match.

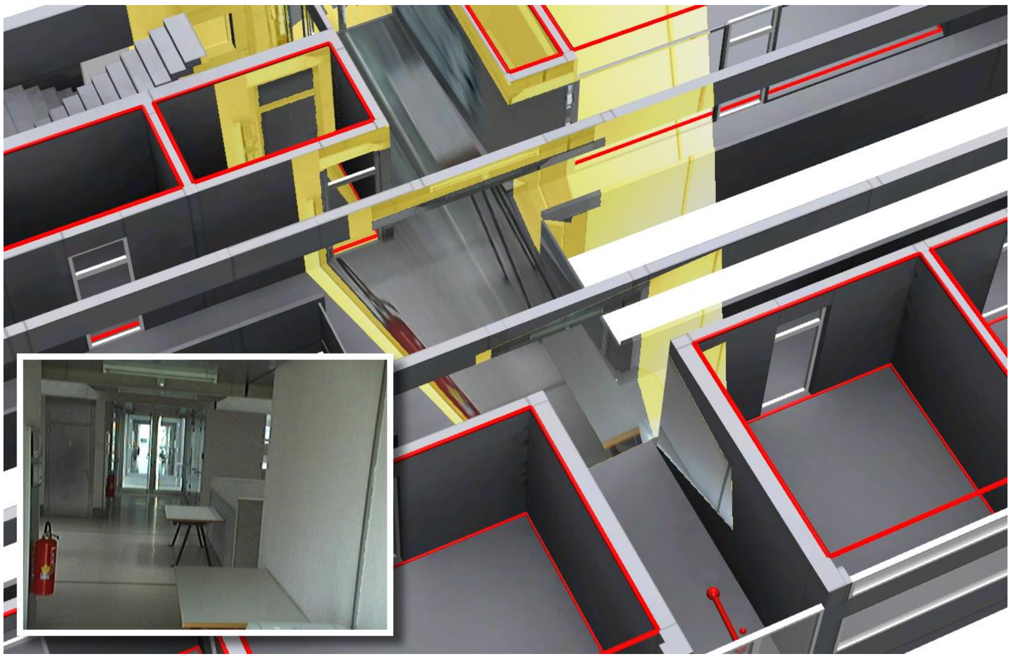


Figure 13: Yellow surfaces are projected on based on the camera feed (picture in picture)

Using this data, and complex algorithms, there are two scenarios. One where individuals can be tracked, and one where due to crowdedness only flows of people can be deducted. It would have been really useful if they described a way to visualize both scenarios in one, but the latter scenario does provide insights nonetheless.

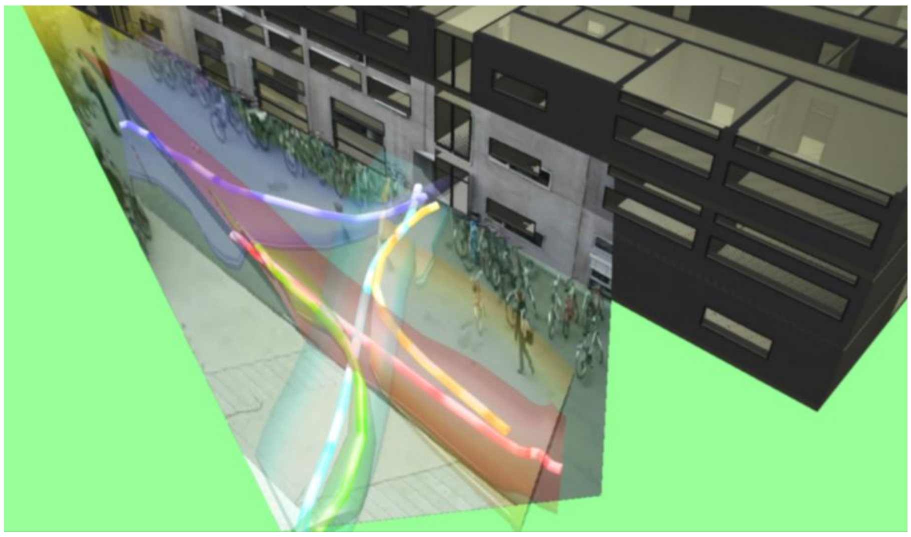


Figure 14: Crowd flow visualization

Their solution is to draw so-called flow-pipes. At hip level (~1m above ground) they paint a tube like flow diagram, where the 'spine' represents the mean, and the width represents the variance.

2.2.5 Crowd characteristics

Their end-product may not be interesting in terms of visual appeal of their end product, however what leads to it is much more so. Wirz et al. developed a visualization for large crowds, and focussed mainly on crowd characteristics [17]. Visualized as heat maps, they distinguish between 4 characteristics.

The first two don't require much imagination, as they are crowd density (i.e. how many people at a given spot), and crowd velocity, which is the mean velocity of a crowd, at a given spot. These could be rather trivial, but provide insights nonetheless.

Thirdly, they discuss crowd turbulence, the difference in directions at a certain spot. In other words, the rate at which the crowd appears to scatter, as opposed to flocking behaviour. Of the three perhaps the most interesting, as this rate could provide valuable insights in the measure of rational thinking of individuals.

Lastly, crowd pressure is visualized, which is the crowd density multiplied by the crowd velocity. Higher crowd pressures could give insights in bottlenecks in emergency egress of a certain building. Perhaps not directly related to this project, but it might be useful if this project is utilized in other contexts (e.g. building safety research based on fire drills).

2.2.6 Conclusion

To sum up our findings: when it comes to visualizing indoor geometric data, the obvious choice for the building seems to be a map, but that leaves out many architectural details that could be of importance, not to mention the problem of multi-story buildings. For this, we found that using axonometric, or 2½D rendering of a 3D model could preserve those valuable details.

As for visualizing the flow of the crowd, we may have to resort to using abstract diagrams. There is little research on visualizing individual behaviours in crowds, nor making deviant behaviour stand out of crowd visualizing, but there are techniques to reduce the cognitive load.

A notable example would be edge-binding, in case every single path is visualized as a separate line. This makes it possible to have a clean overview of paths, even at points where a lot of paths cross.

Those single paths could also be combined in flow-tubes. These are aggregated paths where a spine would stand for the mean path, and the width of the flow would stand for the variance.

Another way of visualizing crowds is in the form of heat maps. Rather than visualizing paths taken over a time span, heat maps provide insights of for example crowd density, crowd velocity, crowd turbulence (i.e. the rate of scattering), and crowd pressure (e.g. bottlenecks in escape routes).

The challenge for this project is to find a visualization that provides some, if not all of the above insights of the aforementioned techniques, in one information space.

3. Ideation & specification phase

3.1 Visual style

The research done in Chapter 2, revealed a predisposition towards a more visual, less abstract way of visualizing positional data. This opposed to a more abstract way, like diagrams. This was supported in the ideation phase, because of several reasons. Firstly, it is very intuitive. It’s extensively researched, and it’s the first thing that comes to mind. Secondly, where abstract visualizations are good for answering predefined questions (e.g. how many people were inside at an arbitrary time), a visual representation provokes noticing unusual behaviour and thinking of new questions.

At first, the idea was to model a building in a 3D modelling program (e.g. SketchUp), and import that into the program. This would allow for great details to visible, thus a more realistic representation it would become. In turn, this would allow for quick hypothesis generation for certain behaviours. For example, if a bottleneck would be visible in the data, it could very easily be linked to features of the building (e.g. a narrow doorway). However, the added value of creating a lifelike model versus the extra cost of this time consuming endeavour is minimal. It was decided that a more abstract representation would suffice, if not be the better alternative.

3.2 Data selection

Having defined the main visualization, the question arises what sets it apart from plotting dots on a map. The visualization has to be extended to be a meaningful attribution to existing tools. The aforementioned predisposition towards lesser abstract ways in favour of diagrams, doesn’t mean there can’t be any diagrams, as it can be helpful to answer predefined questions before further inspection. This leads us to believe that diagrams can be a helpful asset for interpreting the visualization, especially if they are synchronized with each other (i.e. changes in one, lead to updated output of the other).

From the research in chapter 2, we also learn about clutter, and how too much information can reduce the uptake of information. With this in mind, it would make sense to be able to highlight certain groups of people, or hide others. This requires a filter system, that could be in the form of a query-like selectors. A query can be thought of as a line of programming code, that selects data that meets certain criteria, from a larger data set. Take for example the structured query language MySQL, which is a language that is used to manipulate databases of the same name. For example, a query `SELECT name WHERE age=25` would retrieve all the names of people who are 25 years old from a database.

These query-like selectors are preferably visual, as the client has little affinity with, or interest in writing computer code. Writing a program that can read and execute code would make little sense, because there are a plethora of other solutions doing just that. We don't want to recreate a program like R, or SPSS. Adding to this is the idea that the project should require zero prior knowledge, and still be useable.

3.3 Visual query-like selector

With some time passed, the idea of combining the abstract output of a diagram, with the query-like data selectors took shape. The intuitiveness of selecting parts of a graph, and having the result be visualized in the 3D environment had been agreed upon. What had to be agreed upon, was what the selectors would select upon. This was partially left open for discussion, as at this stage it would mostly be wild speculation. However, two selectors were agreed upon.

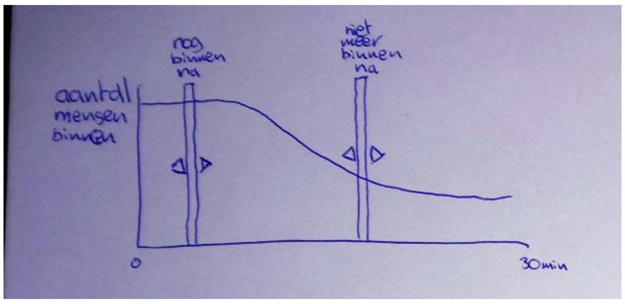


Figure 15: First selector

The first selector would select the people who are inside the building during a timespan. The graph would show the time passed on the horizontal axis, and the number of people inside of the building on the vertical axis. This would ideally be a line that drops quickly, and would show a measure of how quick the evacuation has happened. By moving two sliders, it is possible to select a range of people who have egressed from the building between a certain time span. In conjunction with the graph, it is possible to easily select the people who egress quickly, or those who lagged.

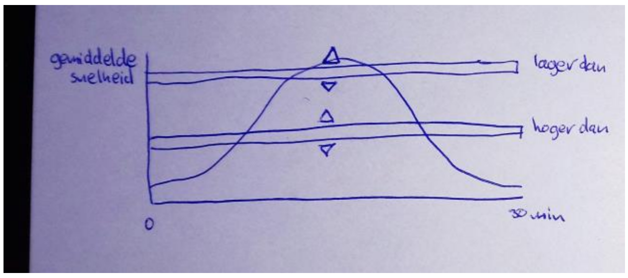


Figure 16: Second selector

The second selector that was agreed upon was the speed graph. Although in a later stage this had changed, initially it would show a graph with the time passed on the horizontal axis and the average highest achieved speed on the vertical axis. Two horizontal sliders would select the range of people who have achieved that speed. This selector would be able to distinguish between those who ran outside, and those who took their time to walk.

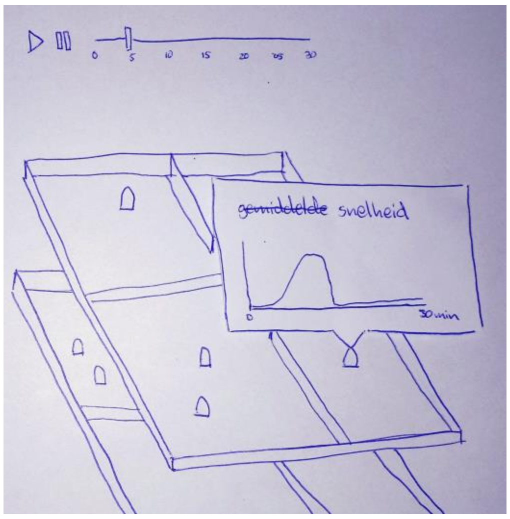


Figure 17: time-dependent abstract visualization of individuals

In chapter 2 there was lot of research into time-independent aggregated views like lines on the floor, or the so-called flow-pipes. Initially it was planned to corporate this in the final product, but the added value was rather low, especially compared to the cost and complexity involved in making it. Playing back the data will provide an insight in what routes were taken, and the number of people that have taken it. The aggregated view would have added little new insights, but would have required a completely new interpretation of the visualization, code-wise, and would have added complexity to the interface. Instead, an abstract, time-dependent visualization of individuals was chosen for.

3.4 Final proposal

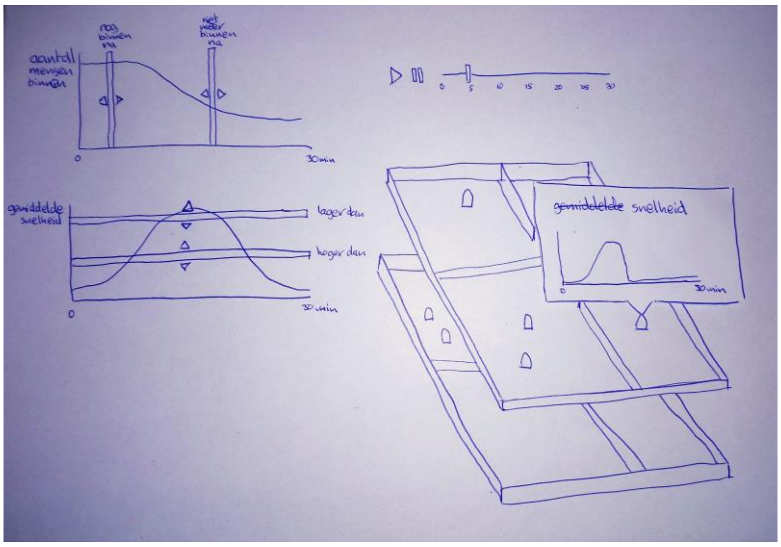


Figure 18: The complete sketch

The ideation was concluded with a final proposal, that the client would agree upon. The complete sketch, consisting of aforementioned elements is added as the leading design. The full document can be found in the appendix Functional Requirements. It was attempted to write the document, so that a programmer not involved could make the program. In this document, the project's purpose and usage are explained.

The purpose is explained in a short fashion: The product is to visualize data gathered during an emergency evacuation, i.e. plotted in a model of a building, so that research into it can provide non-ambiguous conclusions on questions like how long it took for people to exit the building, measured from the moment the alarm went off.

The usage describes the users of the product; researchers with average technical know-how, and describes what data should be fed to the visualization. Then the visualization is explained; the flat-hierarchy interface, the time-dependent abstract visualization of individuals, the several selectors, how they should function, and what they mean. Lastly it is noted that there is room for more selectors and graphs.

4. Realization

4.1 Iterative design

The first step in developing a novel product, is defining what to develop. Traditional ways of doing so involve a cascading ‘waterfall’ type of process, where each step is clearly distinct. When a step is taken, the results are final and taking a step back is impossible. At an early stage, stakeholders are identified, requirements are defined, and the planning is made. Then development starts and the product is made. Lastly the product is evaluated.

It is possible –if not very likely– that misjudging and mistakes happen early on, and looking back on previous decisions is crucial. Hence Creative Technology –not uniquely– insists on a more agile approach, where it is at any moment possible to reflect on previously made decisions in previous phases. An extensively deployed paradigm is that of so-called iterative design. In essence a more trail-and-error approach to previously aforementioned waterfall-approach.

Each iteration consists of a traditional waterfall process, albeit with a flow in a less fixed direction. The evaluation of a previous iteration, is food for the ideation of the next iteration. Iterations take a short time-span, as to allow for rapid changes in development choices.

This project employs the same strategy. Three iterations were planned, where a meeting was scheduled to test the current stage. The last stage (the final product) was tested in an evaluation setting, which is extensively described in the corresponding chapter.

4.1.2 First iteration

A first meeting was held with the client to test a preliminary iteration of the product. The program was non-functional, and really mostly to test the interface. A the selectors were visibly functional, but had no further effect on the visualization. The visualization itself was a mock-up building with just one room, and a single puppet inside. The puppet would not move, but the orbit, pan and zoom functionality was functional and tested. Although less intuitive than hoped for, the functionality was usable. There were little negative remarks on the interface, which seemed good, but as the functionality was mostly absent, more remarks were to be expected later.

4.1.3 Second iteration

A second meeting was held to test the second iteration. This time, the interface was mostly functional. The buildings were dynamically generated from a geojson file, and three puppets would move around in a smooth fashion, but with an erratic path. The selectors, although nowhere resembling the expected shapes, were functional, and selections on puppets could be made.

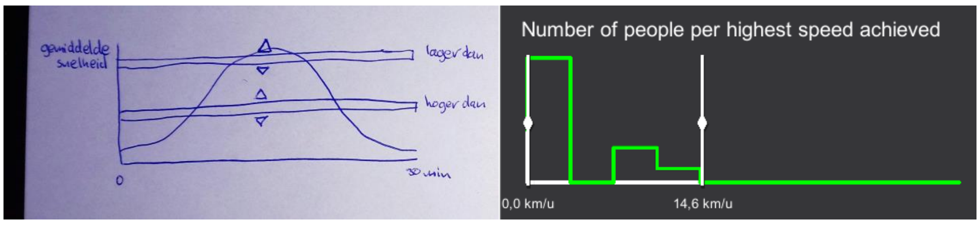


Figure 19: Left: original sketch, right: final implementation. Notice how the horizontal axis was altered to be time-independent, and the sliders are now vertical

The speed selector had been overhauled to make more sense between the first and second meeting. Rather than drawing time on the horizontal axis, and the average highest achieved speeds on the vertical axis, the Puppets were grouped in 10 speed scales. The speed scales are from 0 to 10 m/s with increments of 1m/s. The speed scales are on the horizontal axis, and the number of people on the vertical axis. This is visualized in a bar chart.

Although the world record is slightly over 10m/s [18], it is safe to assume that a person achieving more than 10m/s is an outlier or a faulty measurement. With the exception of indoor vehicles in large buildings, this should hold, but this exception is not expected in scenarios in which the product is to be deployed.

During testing, there were a few remarks to the usability. Firstly, the client had requested for an indicator with how many people were selected and visualized, as to answer questions about proportions of a group with a certain behaviour.

Secondly, although the orbit, pan and zoom functionality worked, it was still not as intuitive as hoped for. At one point, the camera was in a position that made the visualization so unclear that the client remarked that a reset-button to the original camera position. A rewrite of the script (see chapter on source code of OrbitPanZoom) should provide a better experience.

Thirdly, was considering playback speed. Running the visualization in real-time (one second in visualization equals one second in real life) provided a realistic view, and dragging the indicator along the playback bar allowed for skipping through time, there was demand for different time scales in the playback.

Lastly, when the playback had reached the end, and the client wanted to replay the visualization, it became apparent that many mediums go back to the beginning of a video when the play button is pressed after the video has finished. This behaviour was missing, and would have to be implemented.

4.2 Development environment

Where the first iteration did not clearly define a development environment or deploying environment, it was quickly decided that a regular PC should be the best solution. It is rather easy and cost effective to use a PC running windows, rather than one with a different operating system, or building a custom environment. Windows is the operating system of choice by the University of Twente.

As for the choice of development environment, there were several options as well. Two notable options are discussed. Firstly, there was a combination of D3.js and Three.js. Both are JavaScript libraries that allow for beautiful websites and other documents doing just what we want: visualize data. D3.js is a well-built library for creating interactive visualizations based on data. It comes with many examples to draw from. Three.js is a 3D visualization library that uses hardware acceleration (meaning it will run smooth when visualizations become complex, as ours will). It is likely possible to have both libraries run in one single document, that can be used on virtually any device, instantly.

There are some caveats with this approach. It has not been attempted before to get both libraries to ‘communicate’ with one another. This means that the libraries could run alongside each other, but changing parameters in one, would not be visible in the other. It is likely possible, still, but it could become a major time consuming ordeal. Adding to that, is the non-trivial way a D3.js visualization is programmed, and the fact that Three.js appears to require a substantial amount of low-level coding (low-level is considered more difficult and error-prone).

Another option is Unity3D. It is an extensive gaming engine that can be used freely in a non-commercial way. It has a vibrant, helpful commUnity3D, and is known for the ease of setup and use. Unity3D, being a game engine, has lots of features built in that could benefit the development of this project.

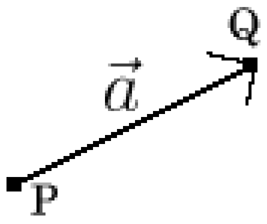


Figure 20: Vector a, between points P and Q

First is the extensiveness of implemented vector mathematics. Vectors can be described as *points* in space, or *directions* from a point. Locations of buildings, features, people and their trajectories can all be described with vectors. Shorthand functions like: Vector3.Distance(Vector3 a, Vector3 b); quickly provide an answer to most occurring questions, but also provide an easily readable codes in more complex structures.

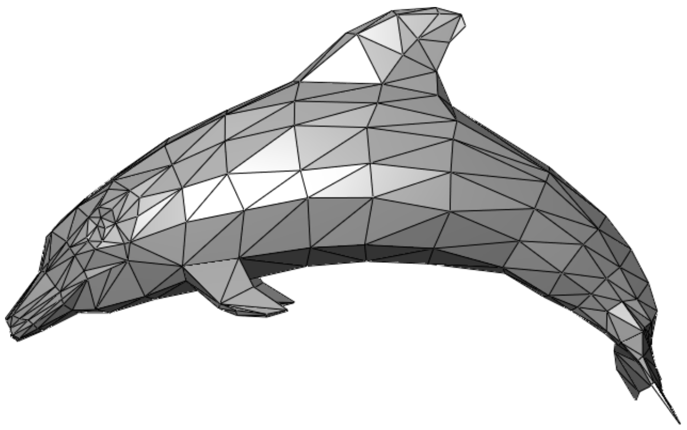


Figure 21: A polygon mesh of a dolphin [19]

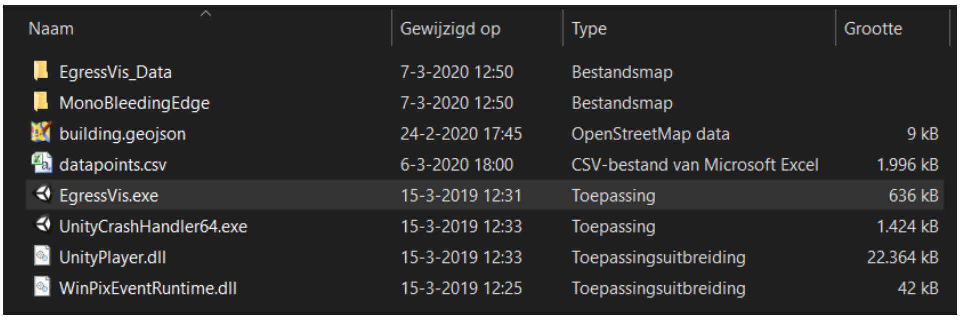
Second is the fast and easy implementation of polygon meshes and colliders. A polygon mesh is in essence a wireframe consisting of vectors and triangle shapes between those points. A collider is a piece of software that can take any point or mesh, and calculate whether or not a *collision* has occurred; that is, whether or not that point or mesh is somewhere inside another mesh. This is very useful for finding out if a position of a human is inside a building (or room) with a very unusual shape.

All in all, the extensive work on logic that would be useful for functionalities that have to be implemented, but also those that could be implemented in future iterations, makes that Unity3D is the environment of choice. Adding to that is the fact that exports could be made for practically every operating system, even as webpages like those created with D3.js and Three.js. (Note: although Unity3D allows for exporting to many different operating systems, development was done, and was only tested in a Windows environment.)

4.3 Build

4.3.1 Folder structure and data set

To allow further development, two versions are delivered; source and build. The source is a Unity3D project that can be altered and built upon. More on this version in later chapters. The other version is the deployed version that is readily executable. The latter version comes in the form of a folder with a Windows executable (.exe file). The folder structure is as follows:



Naam	Gewijzigd op	Type	Grootte
EgressVis_Data	7-3-2020 12:50	Bestandsmap	
MonoBleedingEdge	7-3-2020 12:50	Bestandsmap	
building.geojson	24-2-2020 17:45	OpenStreetMap data	9 kB
datapoints.csv	6-3-2020 18:00	CSV-bestand van Microsoft Excel	1.996 kB
EgressVis.exe	15-3-2019 12:31	Toepassing	636 kB
UnityCrashHandler64.exe	15-3-2019 12:33	Toepassing	1.424 kB
UnityPlayer.dll	15-3-2019 12:33	Toepassingsuitbreiding	22.364 kB
WinPixEventRuntime.dll	15-3-2019 12:25	Toepassingsuitbreiding	42 kB

Figure 22: Folder structure

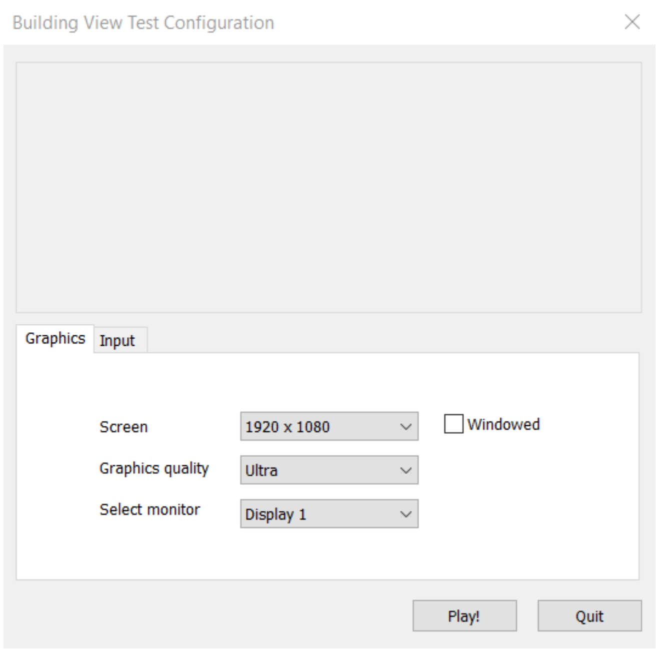
Three files are important to discuss. First is EgressVis.exe, which is the executable. Hence, a double click on this will launch the program.

Second is building.geojson. The extension gives it away, but this is a GeoJSON file. This is a type format that is widely used for describing geographic features, like buildings. It is well described by the IETF [20], however the implementation that the project uses is limited (more in a later chapter). It is rather easy to export GeoJSON files using programs like JOSM [21] or websites like geojson.io [22]. Note: the program reads all the geo-coordinates from this file to determine what part of the world is visualized. It needs to have at least one feature with at least one coordinate to do this, otherwise the data points are rendered well out of range, and unexpected behaviour may occur. Any arbitrary shape can be drawn using the aforementioned website [22]. See Appendix for an example of a drawn triangle on the Carrillonveld of the University of Twente.

Lastly, datapoints.csv is the file that holds all the data points. It is a comma-seperated-values file that is easy to export with programs like Microsoft Excel, SPSS, and R. One single line describes one single data point in this format: ID;TIME;LONGITUDE;LATITUDE, where ID is a unique number (no comma) identifier for a subject, TIME is the time in seconds (commas allowed) on which a data set was started (starting from 0), LONGITUDE and

LATITUDE are a single number describing the longitude (-90 to 90 measured from equator) and latitude (-180 to 180 measured from IERS Reference Meridian [23]) with theoretically infinite precision. This is the same coordinate system used by GPS, and is considered the standard.

4.3.2 Interface


Building View Test Configuration

Screen: 1920 x 1080

Windowed: ☐

Graphics quality: Ultra

Select monitor: Display 1

Figure 23: Opening screen

When the executable file is run, a screen pops up, where quality settings can be altered, in case the performance is bad (e.g. stuttering of screen between frames). Lowering settings allows for smoother interaction, however, the visual quality will degrade, which might hamper a good visual understanding. Keeping Windowed unchecked will launch the program in full-screen mode, which will provide the highest possible resolution (i.e. most distinguishable details). Checking it will launch the program in a window, so that multi-tasking becomes easier. In a setup where 2 or more monitors are connected to the PC, it is also possible to select the monitor where the program will be displayed on. Clicking Play! (Unity3D is a game engine) will launch the program.

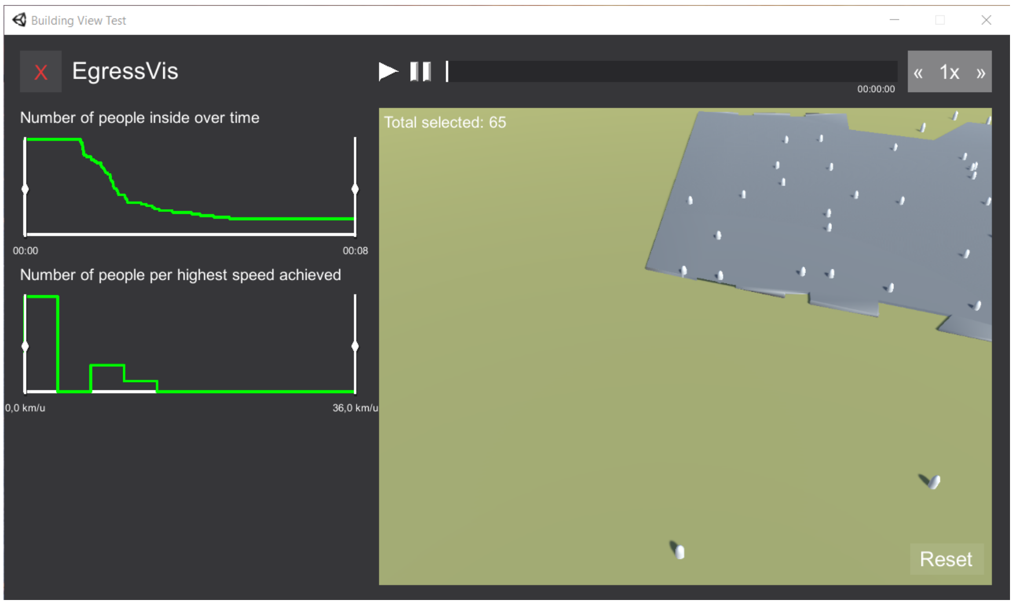


Figure 24: Program run in Windowed mode

When the program is launched, the files building.geojson and datapoints.csv will be loaded. If those files have changed, or a different file has to be loaded, the program has to be launched again. The former running instance does not have to be closed, so it is possible to compare data sets.

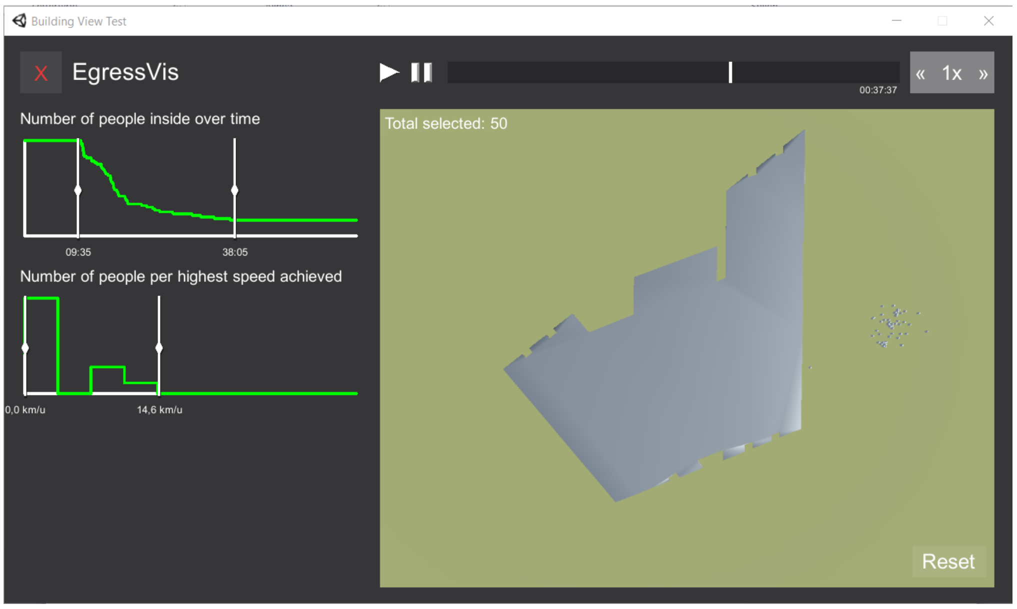


Figure 25: A typical selection of data

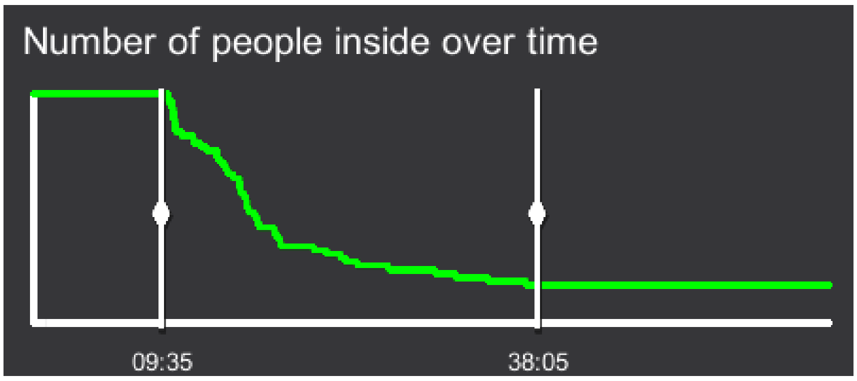


Figure 26: Only the people who evacuated are selected; the evacuation took roughly 30 minutes.

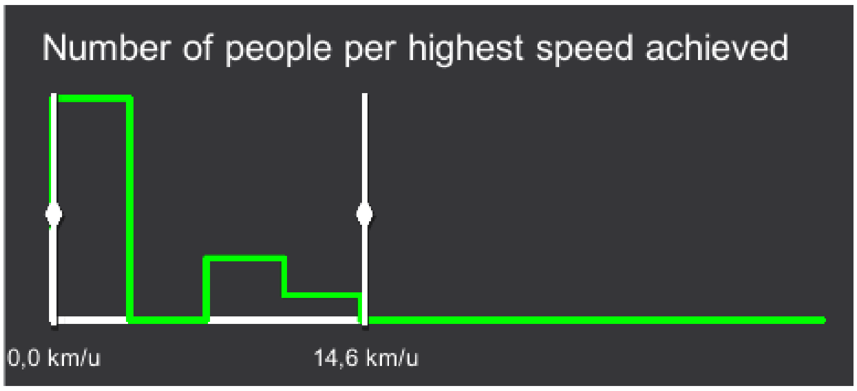


Figure 27: Only the people who ran no faster than 14,6 km/h are selected

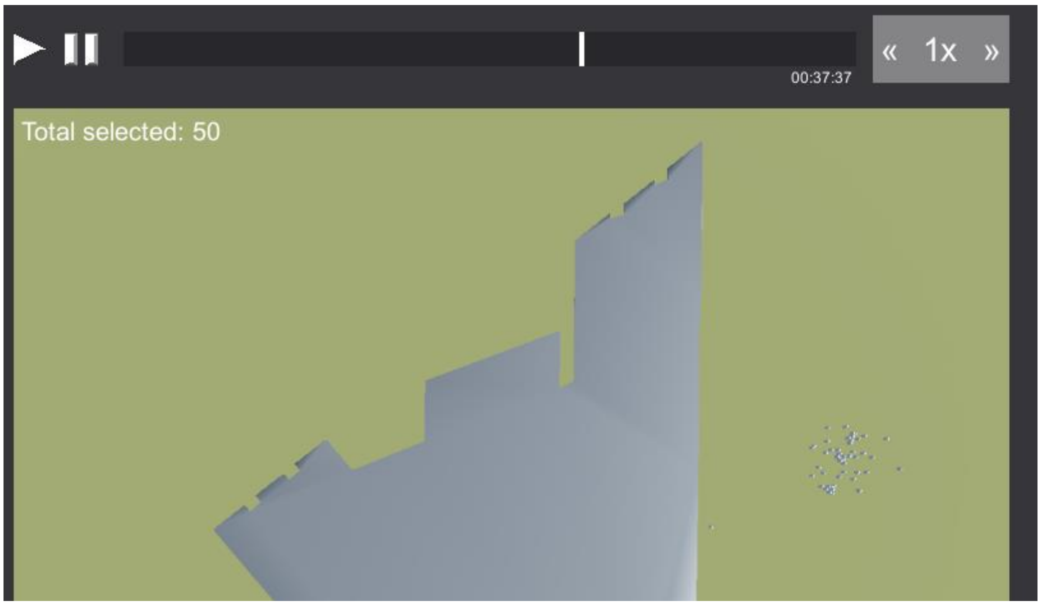


Figure 28: The moment the last person has left the building

The interface very closely resembles the design proposals in Chapter 3. The graphs on the left both feature two vertical sliders that can be used to perform a selection on the data set. See Figure 25 for a typical selection of data: the top-left selector has been set to filter out the people who were never inside the building, and the people who never left. The second selector is set to filter out people who ran faster than 14,6 km/h (no one in this case). Notice in Figure 28 that there were exactly 50 people selected. Also notice how in Figure 26 the timestamp 38:05 roughly describes the moment the last person left the building, and in Figure 28 this moment (37:37) is visualized.

4.4 Source

4.4.1 Unity3D environment

To allow for further development, which is highly encouraged, the source code is also available and documented below. The source comes in the form of a Unity3D project folder. In order to use it, Unity3D has to be downloaded [24]. The version used during development is version 2018.3.9f1, and it is highly recommended to use that version to further develop the project.

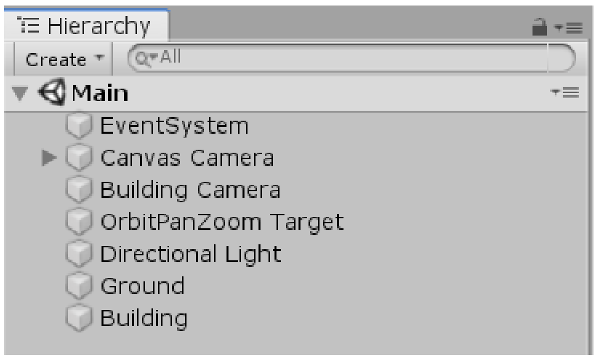


Figure 29: First level of object hierarchy in Unity3D

When the project is opened in Unity, in the Hierarchy tab, is a list of objects. EventSystem is the object where the scripts are loaded and configured. Canvas Camera is the camera where the interface is drawn in. When expanded, there is a list of interface elements, neatly organized and trivially named so that it should require little to no effort to deduct what they are. Building Camera is the camera that renders the 3D image of the data, which it renders onto a render texture called RenTex. OrbitPanZoom Target is an invisible object that the aforementioned camera always 'looks at'. Directional Light and Ground are objects that define a basic 3D environment. Building is the object where the geojson building is constructed into.

4.4.2 Source

Even though Unity3D has a very user-friendly visual drag'n'drop philosophy, the non-trivial use required a lot of coding to be done. For example, even though the editor has a functionality to change the camera view with the mouse, this functionality is not easily copied to the running program. For this, the class OrbitPanZoom was created. This class introduces orbit, pan and zoom functionality to the playing environment. This means that the camera will look at an object; the target, i.e. the target will always be in the centre of the screen. Using the different mouse buttons and by moving the mouse, it is possible to rotate around, zoom in and out of, and move the target. It was largely based on MouseOrbitZoom [25], but completely written from the ground up. The author did a great job combining several functionalities, but it was rewritten to accommodate a different behaviour for the target; to stay on one floor, and to allow the camera and target to be reset.

Because of the setup of the environment, there was need for another class. The main camera holds the interface, and a secondary camera generated the visual representation of the data. All the interactions (e.g. mouse clicks) happen on the interface, but the aforementioned orbit, pan and zoom functionality has to happen with the secondary camera. On top of that, a click on the interface in 2D space, has to be converted to a colliding ray in 3D space, to provide the click functionality for the puppets.

The visual representation of the building and the people is orchestrated by the Puppeteer. It is named after an old form of art; the art of telling a story by moving puppets. In this project, a Puppet is a prefab object with a tiny script, that holds the data of a single person. The puppeteer can move around, show or hide, and highlight those puppets, and is responsible for the logic between selectors, and the diversity of interface elements.

More abstract calculations on, and the parsing of data is done in DataHandler. This is the class that is responsible for interpolating locations between data points, calculating speeds at arbitrary times and collisions with buildings or rooms of Puppets. It makes use of LoadGeoJSON to turn geo-coordinates into coordinates in the environment. The logic for doing this is based on the Mercator projection [26] as used by the OpenStreetMap project.

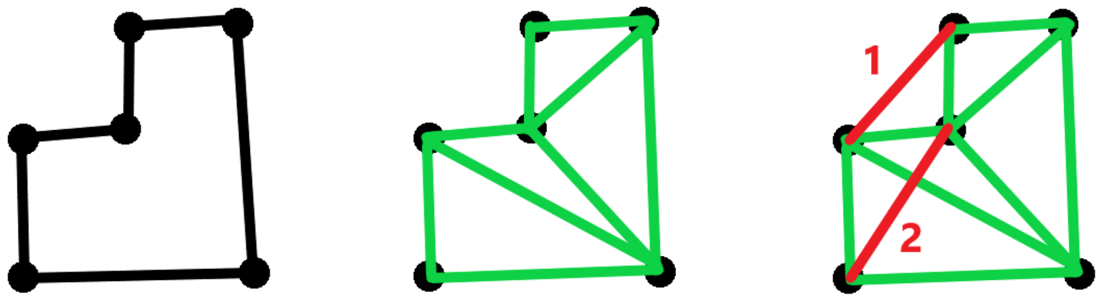


Figure 30: Triangulation from a flat shape (all black). One of the correct solutions (all green), and with two faulty connections. Red 1: creates a triangle outside the shape. Red 2: crosses another line.

LoadGeoJSON is itself responsible for reading and parsing a geojson file, and generating a fully functional 3D mesh, including colliders and renderers. As previously described, Unity3D uses triangle meshes for rendering objects. During development, it became clear that triangulating is quite the mathematical endeavour, and not just a matter of connecting dots. In order to generate a mesh, a plugin is used called Triangle.NET, which is said to use the Delaunay's algorithm [27], among the fastest of triangulation algorithms. Using a custom extrude algorithm, a 3D shape is then made.

Lastly, the Selector script handles the logic of the selectors. It takes a data set, and generates a graph as a texture. Over this graph, there are two sliders (which can be hidden, to show just a graph), whose positions are internally mapped to correspond to values based on the given data set. It was built so that in its entirety it can be duplicated, and it still behaves independently. It should therefore be really easy to add another selector (or just a graph) for the data in future iterations.

5. Evaluation

5.1 Setup

After the third iteration, an evaluation had to take place. This evaluation consisted of a meeting with a potential user who would –after signing an informed consent– go through a series of assignments, and afterwards rate the usability with a SUS-test [28].

The author of the SUS-test describes it as a quick-and-dirty tool to establish the usability of a system. It consist of 10 questions that are designed to test for overall satisfaction, without providing diagnostic feedback. In the Appendix the 10 questions can be found. The answers can be scored from 1 to 5 (completely disagree to completely agree), also known as a Likert scale. For each positively worded question, the score is minus 1, for the negatively worded questions 5 minus the score. Add these all, and then multiply by 2,5 to get the SUS score, which is between 0 and 100 (both inclusive). The score is not a grade, or a percentage, but has its own meaning.

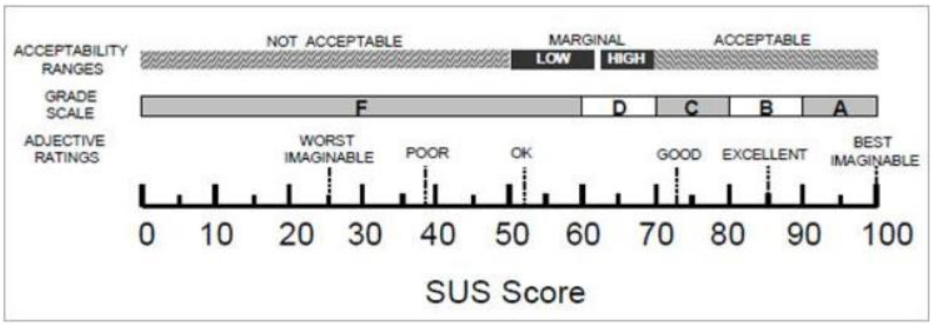


Figure 32: SUS score ranking [29]

5.2 Scenarios

Before testing, a mock-up data set was generated (see Puppeteer code) based on several types of behaviours. These behaviours are based on hypothesised behaviours during emergencies. The major distinction is based on how serious people take the evacuation, ranging from panic as soon as the alarm goes off, to finishing their jobs before walking outside. Those behaviours are grouped in three classes; Runners, Laggers and Normies. Runners take a sprint (speeds over 1 m/s) as soon as the alarm goes off, Laggers wait 10 minutes on average with a variance of 5 minutes, before they walk outside (speeds of maximum 1 m/s), and Normies don't wait, but don't run either.

Two classes are meant to represent outliers; neverInside and neverOutside. Although the visualization expects the data to be cleaned and made compatible before it is used, there is still the possibility that artefacts occur. For example, when devices are measured, it is possible that devices are left inside during an emergency escape, and therefore do not represent a human. It is also possible that devices are at some point in range of a building, but were never truly inside and are not interesting to visualize. Both can be filtered with the visualization.

What holds for all generated classes, is that they always wander around in a random fashion. This is supposed to represent both actual walking around, but also data that is not accurate. This is a common problem in indoor location tracking. In fact, it is among one of the reasons not to use real data, but generate it for this project. This wandering makes the visualization look more real, and thus would make the test more valid.

5.3 Questions and assignments

The generated data is supposed to represent real behaviours, but is deliberately defined to support cases that can be selected for with the visualization. That means that it is possible to use the selectors, to filter specifically for these cases.

The assignments are designed so that each interface element is used at least once. There where possible, the combining of certain elements was encouraged. Questions one and two, for example, both investigate the moment the alarm went off, but using the playback bar and the first selector respectively. However, elements were not tested more than once in a repetitive fashion. This is to keep the test short, which minimizes exhaustion. The questions were made to fit on one sheet of paper (A4 size), for exactly this reason as well. The language is specifically in the tester's native language, to reduce mental strain of interpreting the questions.

The SUS-test that followed is copied from the original, without translating to the native language of the tester. This is to avoid translation errors and misinterpretation by the translator. It was unchanged, as there is a lot of research about the validity of the test, which could be compromised if questions are changed. Of course the tester's proficiency in English was known to be adequate.

5.4 Outcome

5.4.1 Assignments

Without diving too deep into the answers that were given to the assignments, a summary of the answers of the assignments will be discussed. Although the answers did differ on occasions, they were mostly in line with the predicted answers. The first two assignment were correctly answered, but the third was slightly off. This assignment involves setting the first selector so, that the time span selected starts at 10 minutes, and ends roughly 27 minutes after. This is the time it took for the evacuation to take place, and filters out those who were never inside, and those who never left the building. The tester only slid the last slider, which appeared to work as well. Then, to answer the assignment, the tester counted the puppets (46), but failed to notice the fact that the number was on screen.

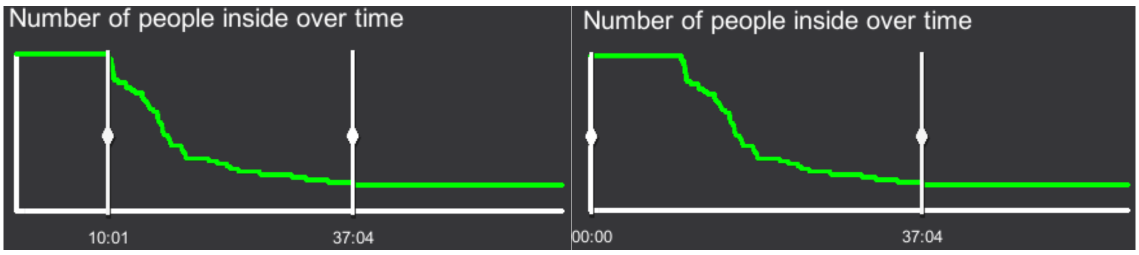


Figure 33: Left: expected, right: outcome

However, by not sliding the left slider, it included the time before the alarm as well. This led to the two follow up questions to be 10 minutes higher than predicted. This, however, cannot be said to be completely wrong, because with a little stretch, the question can be interpreted as to include the time before the alarm as well. See figure 33.

As for the questions 6, 7, 9 and 10, the answers did differ, but can be considered rounding errors, as the answers were read from the screen from the correct places, but still managed to be a few units off, due to imprecise positioning of sliders, for example.

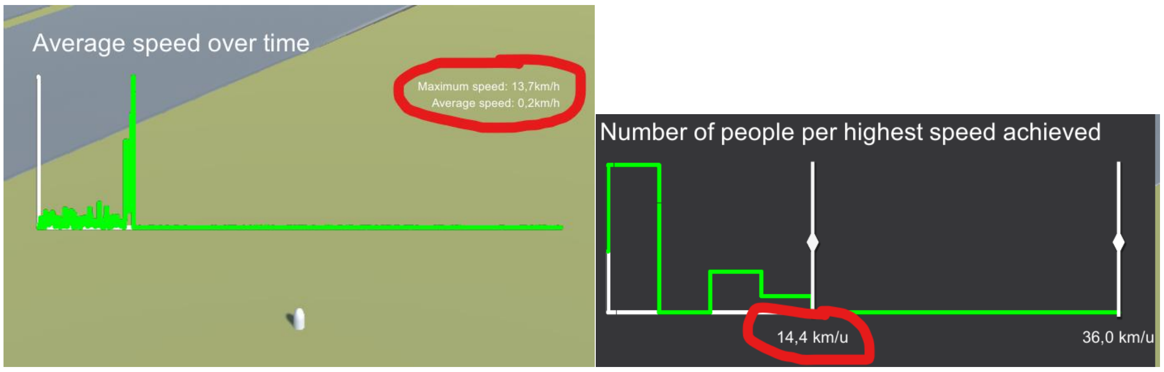


Figure 34: Left: expected information location, right: outcome

Assignment 8 was not far off, but the strategy applied was not the expected one. Finding out the maximum achieved speed was supposed to involve sliding the speed slider to only include the fastest (one, or several), and then clicking all the remaining puppets, or puppet, to see a summary of data of that puppet in a pop-up screen. However, the tester did not take the last step, and read the data from the graph. This is not the fastest speed, but the upper bound of the speed category the fastest person was in. From this we can conclude that apparently, it is not that clear that puppets are clickable as well.

5.4.2 SUS-test

Interestingly, the scores given to the SUS-test questions are never neutral, nor extreme. This means that all questions are answered with either a score of 2 or 4, leaving out 1, 3, and 5. Even more interestingly, the negatively worded questions were all scored 2, and the positively worded were all scored 4. This makes the score 10 * 3 * 2,5 = 75. As we can see in Figure 32; this makes the score Good, with a grade C.

6. Conclusion

6.1 Findings

Recall that the research question for this study is: ‘How can we develop a visualization on crowd behaviour on an individual level, supporting research of behavioural characteristics during emergency egress?’. In the review of previous research, we discuss relevant solutions of visualizations that have already been developed. Where we find interesting solutions on visualizing crowd behaviour in interesting perspectives, like the so-called flow-pipes, but also discuss simpler solutions like plotting dots on a map, an idea was devised that takes the best of both worlds. The visualization will allow multiple perspectives, but not aggregate paths, as to keep details on individual behaviours.

In the ideation phase we shift emphasis to ‘supporting research of behavioural characteristics of emergency egress’. How can this research be a useful addition to the research already done? The answer lies in distinguishing behavioural characteristics, and filtering on them. The idea of selecting partial data, as one could do in a database took shape. This in turn, became the selectors we see in the sketch made for the functional requirements.

The sub question: ‘What behavioural characteristics of individuals vs. group behaviour are interesting to visualize?’ was mostly answered implicitly during development. The selectors that were designed allowed for few characteristics to be distinguished on. In the evaluation phase it was chosen to visualize 5 classes of behaviours; where 2 are not useful by design. The other 3 separate those who run immediately when the alarm goes off, those who wait to finish what they’re doing before slowly walking outside, and those who evacuate immediately, albeit with walking speed.

We have discussed a way how to develop a visualization on crowd behaviour on an individual level, supporting research of behavioural characteristics during emergency egress and what behavioural characteristics of individuals vs. group behaviour are interesting to visualize. To evaluate is the end product is useful, a test on the usability was done, which resulted in a Good result. This means it was a success, but there are ways to improve.

The assignments done prior to the test gave valuable insights in possible improvements. To sum up the most significant findings; the number of selected puppets was not visible enough on screen, and it was not apparent that puppets are clickable as well.

6.2 Future research

As there was a limited time and budget, choices were made that were less than ideal. Among the most profound is the limited research that was done on the behavioural characteristics that lead to the choice of the selectors. This raises the question if they hold up when used on a real data set.

Also, there is a lot to be said on the small test group (N=1), and the quick-and-dirty method of testing the usability; the SUS-test. A more in-depth evaluation on objective effectiveness and efficiency on a much larger group would have yielded a more significant result, and more details on what could be improved.

6.3 Implementation recommendation

For future iterations, a few tips can be given. The most profound is the lack of implementation of multiple floors in a building. This was initially supposed to be built in, but other priorities led to the demise of the implementation. Several options have been reviewed nonetheless, and can be sought in a button not much different from the playback speed selector that loads a new building in the place of the current one. Another option is to load the whole building as one object, and use a selection mechanism to switch viewing floors.

As the parser for GeoJSON data had to be built from the ground up, that is a lot that wasn’t implemented. In detail: Only so-called polygon features are read, and not even in detail (e.g. the names are not displayed). Several other elements would be useful to implement as well, such as point features. They are often used to mark exits, which would be useful in determining the effectiveness of a certain fire exit, for example.

Although it is functional already, the product requires files to be named exactly according to a hardcoded scheme. There is also virtually no feedback on reading errors. All in all, this is a rather error prone system that could be improved drastically by implementing dynamically loading of files and using proper error messages. Adding to that is the fact that right now, a building.geojson file is required, with at least one polygon feature, in order to zoom in on the right spot on Earth. Otherwise the default of Null Island [30] is zoomed in on, rendering basically all data points from datapoints.csv imaginable out of the scope of the visualization. The implementation for the default location could be improved on this point.

Lastly, there is no help function or tutorial for the product. How to use it is based on intuition, but this does mean that if help is needed, there is none. There is a plethora of obvious solutions to this, that have all been extensively tested. For one, the orbit-pan-zoom functionality is nowhere described, and is based on the same functionality in 3D programs (like Unity3D itself). No prior knowledge of such programs can make it difficult to find out how it works. From the assignments in the evaluation we learn that certain elements are missed when there is no prior information on the interface given.

References

[1] J. D. Sime, ‘Crowd psychology and engineering’, *Saf. Sci.*, vol. 21, no. 1, pp. 1–14, Nov. 1995, doi: 10.1016/0925-7535(96)81011-3.

[2] Dicklyon, ‘Fibonacci spiral’. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=3730979.

[3] J. M. Van den Broek, A. W. M. Koetsenruijter, J. C. De Jong, and L. Smit, *Visual language: Perspectives for both makers and users*. Eleven International Publishing, 2012.

[4] J. J. Van Wijk, ‘The value of visualization’, in *VIS 05. IEEE Visualization, 2005.*, Oct. 2005, pp. 79–86, doi: 10.1109/VISUAL.2005.1532781.

[5] S.-O. Tergan and T. Keller, *Knowledge and information visualization: Searching for synergies*, vol. 3426. Springer, 2005.

[6] W. Tobler, ‘Alternatives to Miller’s Projection’, *Cartogr. Geogr. Inf. Syst.*, vol. 24, no. 2, pp. 110–112, Jan. 1997, doi: 10.1559/152304097782439358.

[7] E. Erlingsson, *Axonometric Rendering Algorithms for Mobile Devices using J2ME*. Datavetenskap och kommunikation, Kungliga Tekniska högskolan, 2008.

[8] S. T. Barnard, ‘Interpreting perspective images’, *Artif. Intell.*, vol. 21, no. 4, pp. 435–462, Nov. 1983, doi: 10.1016/S0004-3702(83)80021-6.

[9] B. I. Purcaru, *Games vs. Hardware. The History of PC video games: The 80’s*. Purcaru Ion Bogdan, 2014.

[10] J. Levine, *Exploded Axonometric*. 2008.

[11] M. Carpo, F. Lemerle, and F. Lemerle, *Perspective, Projections and Design : Technologies of Architectural Representation*. Routledge, 2013.

[12] ‘infsoft Tracking - Real-Time Visualization of Devices’. https://www.infsoft.com/software/processing-output/indoor-tracking (accessed Oct. 17, 2019).

[13] ‘Indoor Positioning and Indoor Navigation with Wi-Fi’. https://www.infsoft.com/technology/sensors/wifi (accessed Oct. 17, 2019).

[14] ‘Using visitor-flow visualization to improve visitor experience in museums and exhibitions | MW2015: Museums and the Web 2015’. https://mw2015.museumsandtheweb.com/paper/enhancing-visitor-experience-and-fostering-museum-popularity-through-deep-insights-in-the-placement-of-exhibits-by-new-techniques-in-visitor-flow-visualization-in-space-and-time/ (accessed Oct. 17, 2019).

[15] ‘Example of d3-ForceEdgeBundling on US airline routes graph.’ http://bl.ocks.org/upphiminn/6515478 (accessed Oct. 17, 2019).

[16] P. M. Roth *et al.*, ‘Next-generation 3D visualization for visual surveillance’, in *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, Aug. 2011, pp. 343–348, doi: 10.1109/AVSS.2011.6027348.

[17] M. Wirz, T. Franke, D. Roggen, E. Mitleton-Kelly, P. Lukowicz, and G. Tröster, ‘Inferring Crowd Conditions from Pedestrians’ Location Traces for Real-Time Crowd Monitoring during City-Scale Mass Gatherings’, in *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Jun. 2012, pp. 367–372, doi: 10.1109/WETICE.2012.26.

[18] ‘Olympic Legends: Usain Bolt - Fastest man on the planet’, *Guinness World Records*, Aug. 05, 2016. https://www.guinnessworldrecords.com/news/olympics-news/2016/8/olympic-legends-usain-bolt-fastest-man-on-the-planet-438787 (accessed Mar. 27, 2020).

[19] Chrschn, ‘The polygon mesh of dolphin.’ [Online]. Available: http://en.wikipedia.org/wiki/File:Dolphin_triangle_mesh.png.

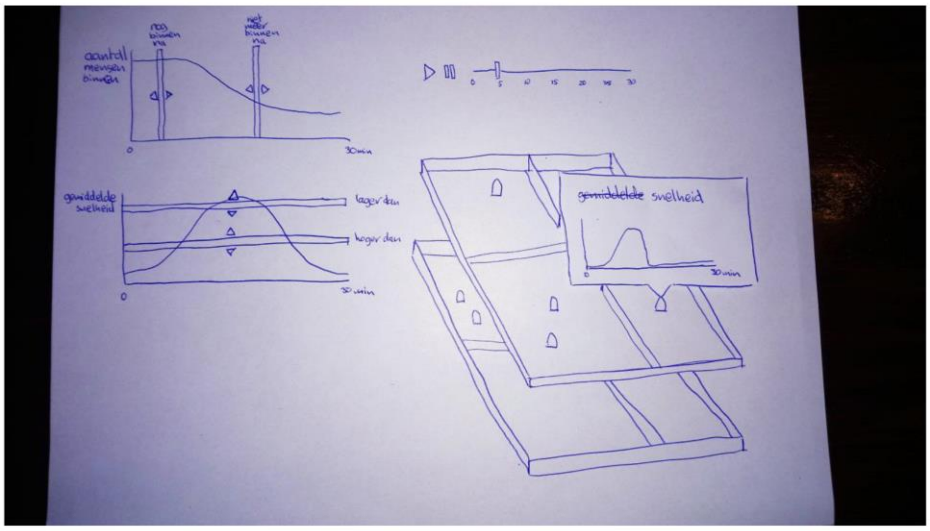
[20] S. Gillies, H. Butler, M. Daly, A. Doyle, and T. Schaub, ‘The GeoJSON Format’. https://tools.ietf.org/html/rfc7946 (accessed Mar. 20, 2020).

[21] ‘JOSM’. https://josm.openstreetmap.de/ (accessed Mar. 20, 2020).
[22] MapBox, ‘geojson.io’, *geojson.io*. http://geojson.io/#map=2/20.0/0.0 (accessed Mar. 20, 2020).
[23] S. Malys, J. H. Seago, N. K. Pavlis, P. K. Seidelmann, and G. H. Kaplan, ‘Why the Greenwich meridian moved’, *J. Geod.*, vol. 89, no. 12, pp. 1263–1272, Dec. 2015, doi: 10.1007/s00190-015-0844-y.
[24] Unity Technologies, ‘Unity Real-Time Development Platform | 3D, 2D VR & AR Visualizations’. https://unity.com/ (accessed Mar. 23, 2020).
[25] ‘MouseOrbitZoom - Unify Community Wiki’. https://wiki.unity3d.com/index.php?title=MouseOrbitZoom (accessed Mar. 26, 2020).
[26] ‘Mercator - OpenStreetMap Wiki’. https://wiki.openstreetmap.org/wiki/Mercator (accessed Mar. 26, 2020).
[27] B. Delaunay, ‘Sur la sphère vide. A la mémoire de Georges Voronoï”, *Bull. Académie Sci. URSS Cl. Sci. Mathématiques Na*, no. 6, pp. 793–800, 1934.
[28] J. Brooke, ‘System usability scale (sus): A quick-and-dirty method of system evaluation user information. Digital equipment co ltd’, *Read. UK*, pp. 1–7, 1986.
[29] J. Brooke, ‘SUS: a retrospective’, *J. Usability Stud.*, vol. 8, no. 2, pp. 29–40, 2013.
[30] T. St. Onge, ‘The Geographical Oddity of Null Island | Worlds Revealed: Geography & Maps at The Library Of Congress’, Apr. 22, 2016. //blogs.loc.gov/maps/2016/04/the-geographical-oddity-of-null-island/ (accessed Mar. 31, 2020).

Appendix

Functional specifications

Functional specifications



1. Purpose

A visualization has to be made for research into emergency egress. In essence, a program that plots location data of human beings (further called 'subjects') on a map of a building, over a certain timespan. Assumed is that researchers have collected location data on subjects in a certain building (or several), and wish to visualise this, with the use of clever 'selectors' to limit the information on screen. Aggregated views should visualize short-hand data, like walking speed and whether or not a subject is inside the building or not. This should lead to non-ambiguous conclusions, for example to answer the question how long it took for every subject to exit the building, measured from the moment the alarm went off.

2. Usage

Researchers with average technological know-how should be able to upload location data of building(s), more specifically with details like (but not necessarily limited to) rooms, hallways, doorways, staircases and exits. Then, cleaned and prepared data of tracked subjects and their locations at certain timestamps should be uploaded as well.

After uploading the data into the visualization, the building(s) and the subjects have to be displayed in a 3D environment, to accommodate a birds-eye-view of the situation. Important is the possibility to zoom, pan and change perspective to select and magnify certain parts of the building/situation. This can be considered a selector for data.

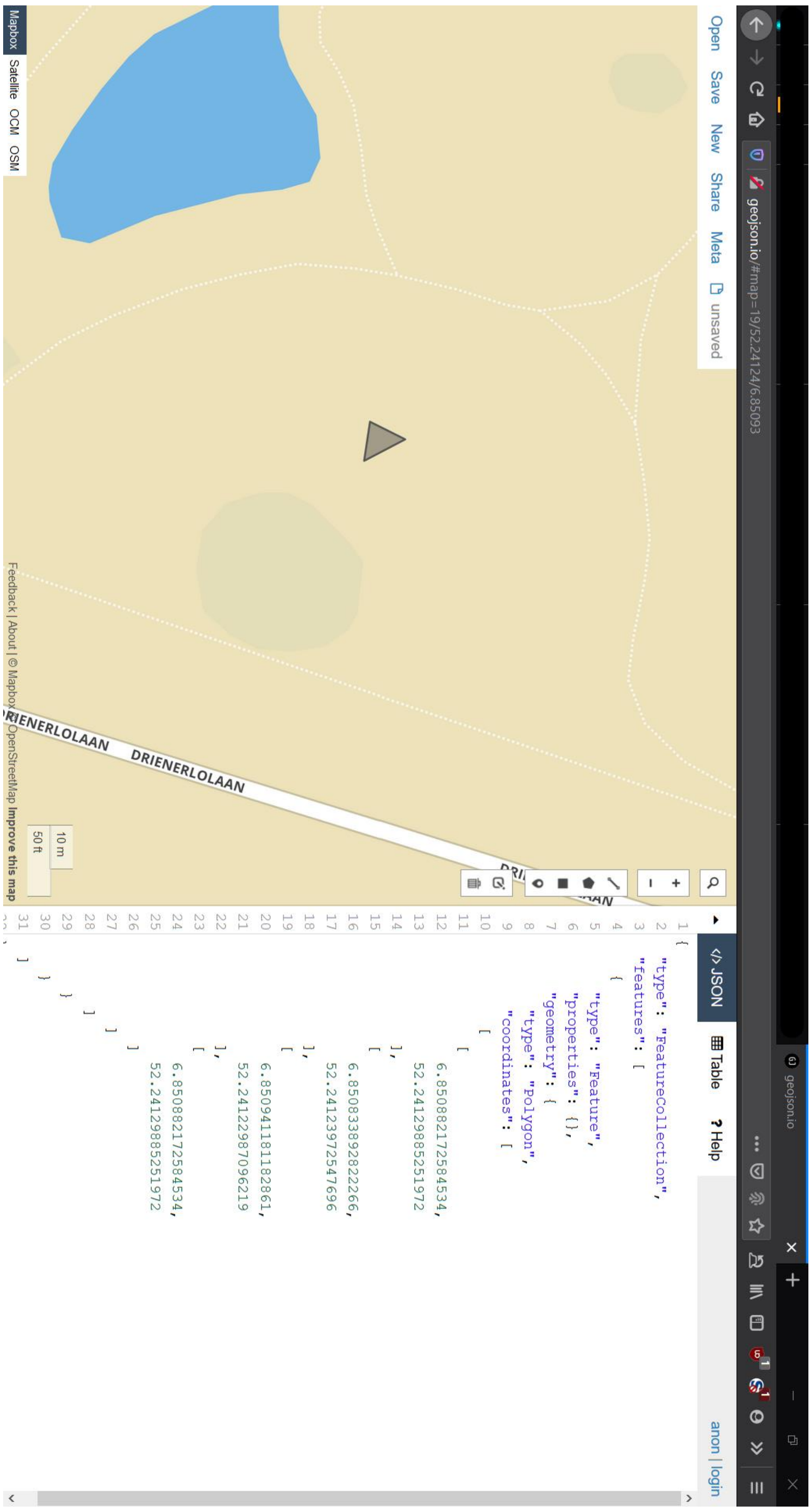
Near the top of this view, or at least there where it is clear what its purpose is, there should be a 'play bar' similar to that of video players. This is so that in the 3D environment, the location of subjects changes in time, as if a video is playing. It should be possible to select any certain point in time, and the location of the subjects should change accordingly. To reduce ambiguity, the locations of the subjects have to be visualized smoothly, and have to be interpolated.

In the same screen (that is, has to be visible without interaction), a panel should hold several plots and selectors. One such selector is meant to select subjects that were inside or outside the building at a certain time. The purpose of this to highlight subjects in the 3D environment who were successfully evacuated within or outside of a certain timespan. For this, a graph should be drawn, with on the x-axis the time, and on the y-axis the amount of subjects inside the building. In theory, the line should from top-left, to bottom right (although this is far from guaranteed). Two vertical bars should correspond to 2 points in time. What is selected is best described as; the subjects who hadn't left the building before *the left-most vertical bar*, but have left the building after *the right-most vertical bar*. Only the subjects selected, should be displayed in the 3D environment.

A similar selector should be made for walking speed. Again, a graph should be drawn, which in this case plots the walking speed on the x-axis, and on the y-axis the number of subjects who had that walking speed as their highest achieved. This should form a Bell curve in theory (although far from guaranteed again). Two horizontal bars should form a selector for the highest achieved walking speed, which is best described as; the subjects who had a peak walking speed below *the upper bar*, but above *the lower bar*. Again, only the subjects selected, should be displayed in the 3D environment.

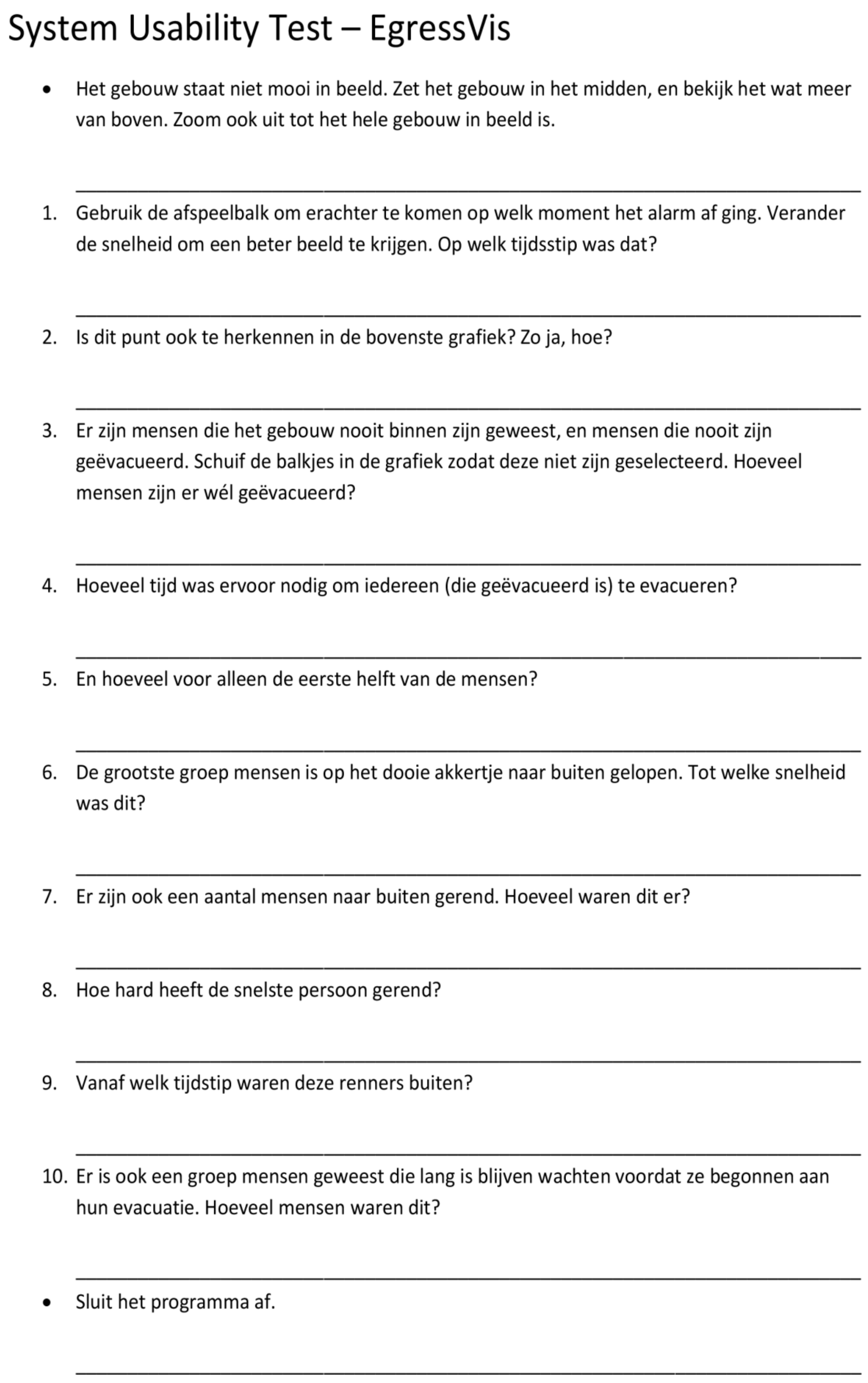
Below that, there is room for more selectors (or simple graphs) to visualize more aggregated data. Aggregated data should also be displayed, when clicking on a subject in the 3D environment. What this data is and how it is displayed, is left open for discussion during development.

GeoJSON.io arbitrary polygon on the Carillonveld of the University of Twente



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              6.850882172584534,
              52.24129885251972
            ],
            [
              6.850833892822266,
              52.24123972547696
            ],
            [
              6.850941181182861,
              52.24122987096219
            ],
            [
              6.850882172584534,
              52.24129885251972
            ]
          ]
        ]
      }
    }
  ]
```

Scenarios



System Usability Test – EgressVis

- Het gebouw staat niet mooi in beeld. Zet het gebouw in het midden, en bekijk het wat meer van boven. Zoom ook uit tot het hele gebouw in beeld is.

1. Gebruik de afspeelbalk om erachter te komen op welk moment het alarm af ging. Verander de snelheid om een beter beeld te krijgen. Op welk tijdsstip was dat?
2. Is dit punt ook te herkennen in de bovenste grafiek? Zo ja, hoe?
3. Er zijn mensen die het gebouw nooit binnen zijn geweest, en mensen die nooit zijn geëvacueerd. Schuif de balkjes in de grafiek zodat deze niet zijn geselecteerd. Hoeveel mensen zijn er wél geëvacueerd?
4. Hoeveel tijd was ervoor nodig om iedereen (die geëvacueerd is) te evacueren?
5. En hoeveel voor alleen de eerste helft van de mensen?
6. De grootste groep mensen is op het dooie akkertje naar buiten gelopen. Tot welke snelheid was dit?
7. Er zijn ook een aantal mensen naar buiten gerend. Hoeveel waren dit er?
8. Hoe hard heeft de snelste persoon gerend?
9. Vanaf welk tijdstip waren deze renners buiten?
10. Er is ook een groep mensen geweest die lang is blijven wachten voordat ze begonnen aan hun evacuatie. Hoeveel mensen waren dit?

- Sluit het programma af.

SUS Test

System Usability Test – EgressVis

1. I think that I would like to use this system frequently.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

2. I found the system unnecessarily complex.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

3. I thought the system was easy to use.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

4. I think that I would need the support of a technical person to be able to use this system.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

5. I found the various functions in this system were well integrated.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

6. I thought there was too much inconsistency in this system.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

7. I would imagine that most people would learn to use this system very quickly.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

8. I found the system very cumbersome to use.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

9. I felt very confident using the system.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

10. I needed to learn a lot of things before I could get going with this system.

Completely disagree	Disagree	Neutral	Agree	Completely agree
○	○	○	○	○

Selector.cs

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Selector : MonoBehaviour
{
    public Slider lower;
    public Slider higher;

    public Text labelLower;
    public Text labelHigher;

    [Range(0, 100)]
    public int minDistance = 10;
    private float distance;

    private int height;
    private int width;
    public int xMargin = 18;
    public int yMargin = 18;

    public RectInt area;
    public RawImage graph;
    public enum Orientation { HORIZONTAL, VERTICAL }
    public Orientation orientation;
    public enum DataType { INTEGER, FLOAT, TIME, METERSPERSECOND, KILOMETERSPERHOUR
}
    public DataType horizontalDataType;
    public DataType verticalDataType;
    public enum GraphType { LINE, STEPS, SMOOTHLINE }
    public GraphType graphType;

    public int lineThickness = 10;

    public DataSet data;

    private bool changed = false;

    public class DataSet
    {
        private SortedDictionary<int, int> data;

        public int yMin { get; private set; }
        public int yMax { get; private set; }
        public int xMin { get; private set; }
        public int xMax { get; private set; }

        public DataSet()
        {
            Reset();
        }

        public void SetPair(int x, int y)
        {

            if (data.TryGetValue(x, out int val))
            {
                data[x] = y;
                ReCalculate();
            }
            else
            {
                data.Add(x, y);
                xMin = Mathf.Min(xMin, x);
                xMax = Mathf.Max(xMax, x);
                yMin = Mathf.Min(yMin, y);
                yMax = Mathf.Max(yMax, y);
            }
        }

        public bool RemovePair(int x)
        {
            bool found = data.Remove(x);

            if (found)
                ReCalculate();

            return found;
        }

        private void ReCalculate()
        {
            IEnumerator<KeyValuePair<int, int>> e = data.GetEnumerator();
            if (e.MoveNext())
            {
                xMin = xMax = e.Current.Key;
                yMin = yMax = e.Current.Value;

                while (e.MoveNext())
                {
                    xMin = Mathf.Min(xMin, e.Current.Key);
                    xMax = Mathf.Max(xMax, e.Current.Key);
                    yMin = Mathf.Min(yMin, e.Current.Value);
                    yMax = Mathf.Max(yMax, e.Current.Value);
                }
            }
        }

        public void Reset()
        {
            xMin = xMax = yMin = yMax = 0;
            data = new SortedDictionary<int, int>();
        }

        public int Count
        {
            get { return data.Keys.Count; }
        }

        public SortedDictionary<int, int>.KeyCollection Keys
        {
            get { return data.Keys; }
        }

        public SortedDictionary<int, int>.ValueCollection Values
        {
            get { return data.Values; }
        }

        public bool TryGetValue(int key, out int value)
        {
            return data.TryGetValue(key, out value);
        }

        override
        public string ToString()
        {
            string temp = "[DataSet] x=(" + xMin + ", " + xMax + "), y=(" + yMin + ", " + yMax + ")";

            foreach (int x in data.Keys)
                if(data.TryGetValue(x, out int y))
                    temp += "\n\t" + x + "=" + y;

            return temp;
        }
    }

    public void Awake()
    {
        distance = (higher.maxValue - lower.minValue) * (((float) minDistance) / 100);
        lower.onValueChanged.AddListener(delegate { LowerChanged(); });
        higher.onValueChanged.AddListener(delegate { HigherChanged(); });

        RectTransform rect = GetComponent<RectTransform>();
        // Best type of anti-aliasing: render at quadrupel resolution, then assume built-in OpenGL scaling
        height = (int) rect.rect.height * 2;
        width = (int) rect.rect.width * 2;
        area = new RectInt(new Vector2Int(xMargin, yMargin), new Vector2Int(width - 2 * xMargin, height - 2 * yMargin));

        graph = GetComponent<RawImage>();
        DataSet d = new DataSet();
        RenderGraph(d);
    }

    public void SetPosition(Vector3 pos)
    {
        GetComponent<RectTransform>().position = pos;
    }

    public void Show()
    {
        gameObject.SetActive(true);
    }

    public void Hide()
    {
        gameObject.SetActive(false);
    }

    public bool HasChanged()
    {
        if(changed)
        {
            changed = false;
            return true;
        }

        return false;
    }

    public bool IsFullRange()
    {
        if (orientation == Orientation.HORIZONTAL)
            return lower.value == 0 && higher.value == 1;
        else
            return lower.value == 1 && higher.value == 0;
    }

    public float GetLowerBound()
    {
        if (orientation == Orientation.HORIZONTAL)
            return Mathf.Lerp(data.xMin, data.xMax, lower.value);
        else
            return Mathf.Lerp(data.yMin, data.yMax, lower.value);
    }

    public float GetHigherBound()
    {
        if (orientation == Orientation.HORIZONTAL)
            return Mathf.Lerp(data.xMin, data.xMax, higher.value);
        else
            return Mathf.Lerp(data.yMin, data.yMax, higher.value);
    }

    private void LowerChanged()
    {
        changed = true;

        float low = Mathf.Clamp(lower.value, lower.minValue, higher.value - distance);
        if (lower.value != low)
            lower.value = low;

        if (orientation == Orientation.HORIZONTAL)
            labelLower.text = BeautifyString(GetLowerBound(), horizontalDataType);
        else
            labelLower.text = BeautifyString(GetLowerBound(), verticalDataType);
    }

    private void HigherChanged()
    {
        changed = true;

        float high = Mathf.Clamp(higher.value, lower.value + distance, higher.maxValue);
        if (higher.value != high)
            higher.value = high;

        if (orientation == Orientation.HORIZONTAL)
            labelHigher.text = BeautifyString(GetHigherBound(), horizontalDataType);

        else
            labelHigher.text = BeautifyString(GetHigherBound(), verticalDataType);
    }

    public void RenderGraph(DataSet d)
    {
        data = d;

        Texture2D tex = EmptyTexture2D();

        // Render axes
        DrawLine(ref tex, area.min, new Vector2(area.xMin, area.yMax), Color.white, lineThickness);
        DrawLine(ref tex, area.min, new Vector2(area.xMax, area.yMin), Color.white, lineThickness);

        Vector2[] points = new Vector2[data.Count];
        int i = 0;

        foreach (int x in data.Keys)
        {
            if (data.TryGetValue(x, out int y))
                points[i] = new Vector2(Scale(data.xMin, data.xMax, area.xMin, area.xMax, x),
                                        Scale(data.yMin, data.yMax, area.yMin, area.yMax, y));
            i++;
```

```csharp
        }

        DrawLines(ref tex, points, Color.green);

        tex.Apply();

        graph.texture = tex;

        LowerChanged();
        HigherChanged();
    }

    private Texture2D EmptyTexture2D()
    {
        Texture2D tex = new Texture2D(width, height, TextureFormat.RGBA32, false);

        for (int x = 0; x < tex.width; x++)
            for (int y = 0; y < tex.height; y++)
                tex.SetPixel(x, y, new Color(0, 0, 0, 0));
        tex.Apply();

        return tex;
    }

    public void DrawLines(ref Texture2D tex, Vector2[] points, Color color)
    {
        if (points.Length == 0) return;

        if (graphType == GraphType.SMOOTHLINE)
            points = SmoothLines(points, 1);

        Vector2 cPos = points[0];
        Vector2 pPos;

        for (int i = 0; i < points.Length; i++)
        {
            pPos = cPos;
            cPos = points[i];

            switch (graphType)
            {
                case GraphType.SMOOTHLINE:
                case GraphType.LINE:
                    DrawLine(ref tex, pPos, cPos, color, lineThickness);
                    break;

                case GraphType.STEPS:
                    Vector2 corner = new Vector2(pPos.x, cPos.y);
                    DrawLine(ref tex, pPos, corner, color, lineThickness);
                    DrawLine(ref tex, corner, cPos, color, lineThickness);
                    break;
            }
        }
    }

    public void DrawLine(ref Texture2D tex, Vector2 p1, Vector2 p2, Color col, int thickness)
    {
        Vector2 t = p1;
        Vector2 perp = Vector2.Perpendicular((p2 - p1).normalized) * ((float)thickness / 2);
        float frac = 1 / Mathf.Sqrt(Mathf.Pow(p2.x - p1.x, 2) + Mathf.Pow(p2.y - p1.y, 2));
        float ctr = 0;

        while ((int)t.x != (int)p2.x || (int)t.y != (int)p2.y)
        {
            t = Vector2.Lerp(p1, p2, ctr);
            ctr += frac;

            // Draw perpendicular line if thickness exceeds 1
            if(thickness == 1)
                SetPixelClamped(ref tex, (int)t.x, (int)t.y, col);
            else
                DrawLine(ref tex, t - perp, t + perp, col, 1);

        }

        // Render circle on both ends (calculate circle once)
        int radius = thickness/2;
        int radiusSquared = radius * radius * 5 / 4;
        for (int y = -radius; y <= radius; y++)
            for (int x = -radius; x <= radius; x++)
                if (x * x + y * y <= radiusSquared)
                {
                    SetPixelClamped(ref tex, (int)p1.x + x, (int)p1.y + y, col);
                    SetPixelClamped(ref tex, (int)p2.x + x, (int)p2.y + y, col);
                }
    }

    private void SetPixelClamped(ref Texture2D tex, int x, int y, Color col)
    {
        if (!(x < 0 || x > tex.width - 1 || y < 0 || y > tex.height - 1))
            tex.SetPixel(x, y, col);
    }

    private string BeautifyString(float data, DataType type)
    {
        switch (type)
        {
            case DataType.INTEGER:
                return ((int) data).ToString();

            case DataType.METERSPERSECOND:
                return data.ToString("0.0") + " m /s";

            case DataType.KILOMETERSPERHOUR:
                return (data * 3.6).ToString("0.0") + " km/u";

            case DataType.TIME:
                return System.TimeSpan.FromSeconds(data).ToString(@"mm\:ss");

            default:
                return data.ToString();
        }
    }

    private Vector2[] SmoothLines(Vector2[] vertices, float size)
    {
        if (vertices.Length == 0)
            return vertices;

        AnimationCurve curve = new AnimationCurve();


        Keyframe[] keyframes = new Keyframe[vertices.Length];

        float minTime = vertices[0].x;
        float maxTime = minTime;

        for (int i = 0; i < vertices.Length; i++)
        {
            keyframes[i] = new Keyframe(vertices[i].x, vertices[i].y);

            if (vertices[i].x < minTime)
                minTime = vertices[i].x;
            else if (vertices[i].x > maxTime)
                maxTime = vertices[i].x;
        }

        curve.keys = keyframes;

        List<Vector2> vectors = new List<Vector2>();

        for (float time = minTime; time < maxTime; time += size)
            vectors.Add(new Vector2(time, curve.Evaluate(time)));

        return vectors.ToArray();
    }

    private float Scale(float OldMin, float OldMax, float NewMin, float NewMax, float OldValue)
    {
        float OldRange = (OldMax - OldMin);
        float NewRange = (NewMax - NewMin);
        float NewValue = OldRange != 0 ? (((OldValue - OldMin) * NewRange) / OldRange) + NewMin : NewMin;

        return (NewValue);
    }

}
```

OrbitPanZoom.cs

```
// original: http://www.unifycommunity.com/wiki/index.php?title=MouseOrbitZoom
using UnityEngine;

public class OrbitPanZoom : MonoBehaviour
{
    public bool enableControls = true;

    [HideInInspector]
    public float Y;
    [HideInInspector]
    public float X;
    [HideInInspector]
    public float distance;
    [HideInInspector]

    public Transform target;

    public float MinimumRotation = 0;
    public float MaximumRotation = 90;
    public float MinimumDistance = 10;
    public float MaximumDistance = 100;

    private Transform desiredLocation;
    private Quaternion desiredRotation;
    private float desiredDistance;

    private Vector3 resetLocation;
    private float resetX;
    private float resetY;
    private float resetDistance;

    public void Init()
    {
        //If there is no target, create a temporary target at 'distance' from the cameras current viewpoint
        if (!target)
        {
            GameObject go = new GameObject("Cam Target");
            go.transform.position = transform.position + (transform.forward * distance);
            target = go.transform;
        }

        X = transform.rotation.eulerAngles.x;
        resetY = Y = transform.rotation.eulerAngles.y;

        resetX = X = ClampAngle(X, MinimumRotation, MaximumRotation);
        desiredRotation = Quaternion.Euler(X, Y, 0);

        resetDistance = desiredDistance = Vector3.Distance(transform.position, target.position);

        desiredLocation = (new GameObject("DesiredLocation")).transform;
        resetLocation = desiredLocation.position = target.position;
    }

    public void ResetView()
    {
        X = resetX;
        Y = resetY;

        desiredRotation = Quaternion.Euler(X, Y, 0);
        desiredDistance = resetDistance;
        desiredLocation.position = resetLocation;
    }

    public void LateUpdate()
    {
        if (enableControls)
        {
            // Left mouse button? ORBIT!
            if (Input.GetMouseButton(0))
            {
                Y += Input.GetAxis("Mouse X") * 4;
                X -= Input.GetAxis("Mouse Y") * 4;

                X = ClampAngle(X, MinimumRotation, MaximumRotation);
                desiredRotation = Quaternion.Euler(X, Y, 0);
            }
            // Right mouse button? PAN!
            else if (Input.GetMouseButton(1))
            {
                desiredLocation.Translate(Vector3.right * -Input.GetAxis("Mouse X"));
                desiredLocation.Translate(Vector3.forward * -Input.GetAxis("Mouse Y"));
            }
            // Middle mouse button? ZOOM!
            else if (Input.GetMouseButton(2))
            {
                desiredDistance -= Input.GetAxis("Mouse Y") * 0.0625f * Mathf.Abs(desiredDistance);
            }
            // Scrolling? ZOOM!
            else
            {
                desiredDistance -= Input.GetAxis("Mouse ScrollWheel") * 0.25f * Mathf.Abs(desiredDistance);
            }

            desiredDistance = Mathf.Clamp(desiredDistance, MinimumDistance, MaximumDistance);
        }

        // Copy the y axis' rotation from this.rotation to the desired location transform
        Vector3 e = desiredLocation.rotation.eulerAngles;
        e.y = transform.rotation.eulerAngles.y;
        desiredLocation.rotation = Quaternion.Euler(e);

        // Smooth target to desired location
        target.position = Vector3.Lerp(target.position, desiredLocation.position, Time.deltaTime * 8f);

        // Smooth rotation to desired rotation
        transform.rotation = Quaternion.Lerp(transform.rotation, desiredRotation, Time.deltaTime * 8f);

        // For smoothing of the zoom, lerp distance
        transform.position = target.position - (transform.rotation * Vector3.forward * Mathf.Lerp(distance, desiredDistance, Time.deltaTime * 8f));

        // Set working distance (when moving target, we want to keep fixed zoom level)
        distance = Vector3.Distance(transform.position, target.position);
    }


    void Start() { Init(); }
    void OnEnable() { Init(); }

    private static float ClampAngle(float angle, float min, float max)
    {
        return Mathf.Clamp(angle % 360, min % 360, max % 360);
    }
}
```

LoadGeoJSON.cs

```
using UnityEngine;
using System;
using System.Collections.Generic;
using TriangleNet.Geometry;
using TriangleNet.Meshing;
using TriangleNet.Topology;

public class LoadGeoJSON : MonoBehaviour
{
    public Material mat;

    [HideInInspector]
    public GameObject parent;

    [HideInInspector]
    public static GeoCoordinate center = new GeoCoordinate(0, 0);

    public void LoadFeatures(string text)
    {
        if(parent == null)
            parent = new GameObject("FeatureCollection");

        if (mat == null)
            mat = new Material(Shader.Find("Standard"));

        GeoJSON.FeatureCollection collection = GeoJSON.GeoJSONObject.Deserialize(text);

        Dictionary<Transform, GeoCoordinate> centers = new Dictionary<Transform, GeoCoordinate>();

        foreach (GeoJSON.FeatureObject feature in collection.features)
        {
            GeoJSON.GeometryObject geometry = feature.geometry;

            if (geometry.type == "Polygon")
            {
                List<GeoCoordinate> vertices = new List<GeoCoordinate>();

                foreach (GeoJSON.PositionObject position in geometry.AllPositions())
                {
                    vertices.Add(new GeoCoordinate(position.longitude, position.latitude));
                }

                try
                {
                    GeoCoordinate center = ComputeMinimumValue(vertices);

                    GameObject polyObject = GenerateGameObjectFromVector2List(ToPixel(vertices, center), mat);
                    polyObject.transform.parent = parent.transform;

                    if (feature.properties.TryGetValue("name", out string name))
                        polyObject.name = name;

                    centers.Add(polyObject.transform, center);
                }
                catch (NullReferenceException e)
                {
                    Debug.Log(e);
                }
            }
        }

        center = ComputeMinimumValue(centers.Values);

        foreach (KeyValuePair<Transform, GeoCoordinate> pair in centers)
            pair.Key.position = ToPixel(pair.Value - center);
    }

    public static Vector2 ToPixel(GeoCoordinate coordinate)
    {
        return new Vector2((float) MercatorProjection.LonToX(coordinate.longitude), (float) MercatorProjection.LatToY(coordinate.latitude));
    }

    public static List<Vector2> ToPixel(IEnumerable<GeoCoordinate> coordinates, GeoCoordinate center)
    {
        List<Vector2> vertices = new List<Vector2>();

        foreach (GeoCoordinate coordinate in coordinates)
        {
            Vector2 v = ToPixel(coordinate - center);
            if(!vertices.Contains(v))
             vertices.Add(v);
        }

        return vertices;
    }

    public static GeoCoordinate ComputeMinimumValue(IEnumerable<GeoCoordinate> coords)
    {
        IEnumerator<GeoCoordinate> e = coords.GetEnumerator();

        if (!e.MoveNext())
            return new GeoCoordinate(0, 0);

        double minx = e.Current.longitude, miny = e.Current.latitude;

        while (e.MoveNext())
        {
            minx = Math.Min(minx, e.Current.longitude);
            miny = Math.Min(miny, e.Current.latitude);
        }

        return new GeoCoordinate(minx, miny);
    }

    public static Vector2 ComputeMinimumValue(IEnumerable<Vector2> coords)
    {
        IEnumerator<Vector2> e = coords.GetEnumerator();

        if (!e.MoveNext())
            return Vector2.zero;

        float minx = e.Current.x, miny = e.Current.y;

        while (e.MoveNext())
        {
            minx = Mathf.Min(minx, e.Current.x);
            miny = Mathf.Min(miny, e.Current.y);
        }

        return new Vector2(minx, miny);
    }

    public static GameObject GenerateGameObjectFromVector2List(List<Vector2> contour, Material mat)
    {
        // Make sure all vertices are positive
        Vector2 topleft = ComputeMinimumValue(contour);
        for (int c = 0; c < contour.Count; c++)
            contour[c] -= topleft;

        ConstraintOptions options = new ConstraintOptions() { ConformingDelaunay = false };
        QualityOptions quality = new QualityOptions() { MinimumAngle = 0 }; // MUST be zero to preserve vertices
        var p = new Polygon();

        List<Vertex> vs = new List<Vertex>();
        foreach (Vector2 v in contour)
        {
            Vertex vertex = new Vertex(v.x, v.y, 1);
            vs.Add(vertex);
        }

        p.Add(new Contour(vs, 1));
        IMesh m = p.Triangulate(options, quality);

        List<Vector3> vectorList = new List<Vector3>();
        List<int> triangleList = new List<int>();
        bool clockwise = true;

        // Add all vertices
        foreach (Vertex v in m.Vertices)
        {
            vectorList.Add(new Vector3((float)v.X, (float)v.Y, 0));
            vectorList.Add(new Vector3((float)v.X, (float)v.Y, 1));
        }

        // Find out if triangles clockwise or not
        IEnumerator<Triangle> e = m.Triangles.GetEnumerator();
        if (e.MoveNext())
        {
            Triangle t = e.Current;

            int[] index = new int[3];
            index[0] = vectorList.FindIndex(v =>
                v.x == (float)t.GetVertex(0).X &&
                v.y == (float)t.GetVertex(0).Y &&
                v.z == 0);
            index[1] = vectorList.FindIndex(v =>
                v.x == (float)t.GetVertex(1).X &&
                v.y == (float)t.GetVertex(1).Y &&
                v.z == 0);
            index[2] = vectorList.FindIndex(v =>
                v.x == (float)t.GetVertex(2).X &&
                v.y == (float)t.GetVertex(2).Y &&
                v.z == 0);

            if (!(index[0] < 0 || index[1] < 0 || index[2] < 0))
            {
                clockwise = (new Plane(vectorList[index[0]], vectorList[index[1]], vectorList[index[2]]).GetSide(Vector3.forward));
            }
        }


        // Add triangles and mirror triangles
        foreach (Triangle t in m.Triangles)
        {
            int[] index = new int[3];
            index[0] = vectorList.FindIndex(v =>
                v.x == (float)t.GetVertex(0).X &&
                v.y == (float)t.GetVertex(0).Y &&
                v.z == 0);
            index[1] = vectorList.FindIndex(v =>
                v.x == (float)t.GetVertex(1).X &&
                v.y == (float)t.GetVertex(1).Y &&
                v.z == 0);
            index[2] = vectorList.FindIndex(v =>
                v.x == (float)t.GetVertex(2).X &&
                v.y == (float)t.GetVertex(2).Y &&
                v.z == 0);

            if(!(index[0] < 0 || index[1] < 0 || index[2] < 0))
            {
                if (clockwise)
                {
                    triangleList.Add(index[2]);
                    triangleList.Add(index[1]);
                    triangleList.Add(index[0]);

                    triangleList.Add(index[0] + 1);
                    triangleList.Add(index[1] + 1);
                    triangleList.Add(index[2] + 1);
                }
                else
                {
                    triangleList.Add(index[0]);
                    triangleList.Add(index[1]);
                    triangleList.Add(index[2]);

                    triangleList.Add(index[2] + 1);
                    triangleList.Add(index[1] + 1);
                    triangleList.Add(index[0] + 1);
                }
            }
        }

        // Add all the side panels

        for (int t = 0; t < (vectorList.Count / 2 - 1); t++)
        {
            if (clockwise)
            {
                triangleList.Add((2 * t) % vectorList.Count);
                triangleList.Add((2 * t + 1) % vectorList.Count);
                triangleList.Add((2 * t + 3) % vectorList.Count);

                triangleList.Add((2 * t) % vectorList.Count);
                triangleList.Add((2 * t + 3) % vectorList.Count);
                triangleList.Add((2 * t + 2) % vectorList.Count);
            }
            else
            {
                triangleList.Add((2 * t) % vectorList.Count);
                triangleList.Add((2 * t + 3) % vectorList.Count);
                triangleList.Add((2 * t + 1) % vectorList.Count);

                triangleList.Add((2 * t) % vectorList.Count);
                triangleList.Add((2 * t + 2) % vectorList.Count);
                triangleList.Add((2 * t + 3) % vectorList.Count);
            }
        }

        Mesh mesh = new Mesh
        {
            vertices = vectorList.ToArray(),
            triangles = triangleList.ToArray()
        };

        mesh.RecalculateNormals();

        GameObject obj = new GameObject("Polygon", typeof(MeshFilter), typeof(MeshRenderer), typeof(MeshCollider));
        obj.GetComponent<MeshFilter>().mesh = mesh;
        obj.GetComponent<MeshRenderer>().material = mat;
        obj.GetComponent<MeshCollider>().sharedMesh = mesh;
        obj.GetComponent<MeshCollider>().convex = true;
        obj.GetComponent<MeshCollider>().isTrigger = true;

        return obj;
    }

    public class GeoCoordinate
    {
        public double longitude;
        public double latitude;

        public GeoCoordinate(double lon, double lat)
        {
            longitude = lon;
            latitude = lat;
        }

        public static GeoCoordinate operator -(GeoCoordinate a, GeoCoordinate b)
        {
            return new GeoCoordinate(a.longitude - b.longitude, a.latitude - b.latitude);
        }

        public static GeoCoordinate operator +(GeoCoordinate a, GeoCoordinate b)
        {
            return new GeoCoordinate(a.longitude + b.longitude, a.latitude + b.latitude);
        }
    }

    /*
     * Mercator projection of Lattitude and Longitude Coordinates
     * Altered to use float values
     * source: https://wiki.openstreetmap.org/wiki/Mercator
     */

    public static class MercatorProjection
    {
        private static readonly double R_MAJOR = 6378137.0f;
        private static readonly double R_MINOR = 6356752.3142f;
        private static readonly double RATIO = R_MINOR / R_MAJOR;
        private static readonly double ECCENT = Math.Sqrt(1.0f - (RATIO * RATIO));
        private static readonly double COM = 0.5f * ECCENT;

        private static readonly double DEG2RAD = Math.PI / 180.0f;
        private static readonly double RAD2Deg = 180.0f / Math.PI;
        private static readonly double PI_2 = Math.PI / 2.0f;

        public static double[] ToPixel(double lon, double lat)
        {
            return new double[] { LonToX(lon), LatToY(lat) };
        }

        public static double[] ToGeoCoord(double x, double y)
        {
            return new double[] { XToLon(x), YToLat(y) };
        }

        public static double LonToX(double lon)
        {
            return R_MAJOR * DegToRad(lon);
        }

        public static double LatToY(double lat)
        {
            lat = Math.Min(89.5, Math.Max(lat, -89.5));
            double phi = DegToRad(lat);
            double sinphi = Math.Sin(phi);
            double con = ECCENT * sinphi;
            con = Math.Pow(((1.0 - con) / (1.0 + con)), COM);
            double ts = Math.Tan(0.5 * ((Math.PI * 0.5) - phi)) / con;
            return 0f - R_MAJOR * Math.Log(ts);
        }

        public static double XToLon(double x)
        {
            return RadToDeg(x) / R_MAJOR;
        }

        public static double YToLat(double y)
        {
            double ts = Math.Exp(-y / R_MAJOR);
            double phi = PI_2 - 2 * Math.Atan(ts);
            double dphi = 1.0;
            int i = 0;
            while ((Math.Abs(dphi) > 0.000000001) && (i < 15))
            {
                double con = ECCENT * Math.Sin(phi);
                dphi = PI_2 - 2 * Math.Atan(ts * Math.Pow((1.0 - con) / (1.0 + con), COM)) - phi;
                phi += dphi;
                i++;
            }
            return RadToDeg(phi);
        }

        private static double RadToDeg(double rad)
        {
            return rad * RAD2Deg;
        }

        private static double DegToRad(double deg)
        {
            return deg * DEG2RAD;
        }
    }
}
```

DataHandler.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;

public class DataHandler : MonoBehaviour
{
    public float TimeMin { get; private set; }
    public float TimeMax { get; private set; }
    public float TimeLength { get; private set; }

    public string building = "building.geojson";
    public string datapoints = "datapoints.csv";
    public GameObject parent;

    public Dictionary<int, List<DataPoint>> Points { get; private set; }

    private Dictionary<int, Dictionary<float, float>> speeds;
    private Dictionary<int, float> averages;
    private Dictionary<int, float> maximums;
    private float average;

    public Dictionary<float, float> GetSpeeds(int id)
    {
        if (speeds.TryGetValue(id, out Dictionary<float, float> s))
            return s;

        throw new Exception("No data points found for given id");

    }

    public float GetMaximumAchievedSpeed(int id)
    {
        if (maximums.TryGetValue(id, out float m))
            return m;

        throw new Exception("No data points found for given id");
    }

    public float GetAverageSpeed(int id)
    {
        if (averages.TryGetValue(id, out float a))
            return a;

        throw new Exception("No data points found for given id");
    }

    public float GetAverageSpeed()
    {
        return average;
    }

    public void CalculateSpeeds()
    {
        Dictionary<float, float> speedings;
        float avg = 0, max = 0, speed = 0, total = 0;

        foreach(KeyValuePair<int, List<DataPoint>> pair in Points)
        {
            int id = pair.Key;
            List<DataPoint> points = pair.Value;

            avg = 0; max = 0; speed = 0;
            speedings = new Dictionary<float, float>();

            total = points[points.Count - 1].time - points[0].time;

            speedings.Add(points[0].time, 0);

            for (int i = 1; i < points.Count; i++)
            {
                speed = Vector3.Distance(points[i].position, points[i - 1].position) / (points[i].time - points[i - 1].time);

                max = Mathf.Max(speed, max);
                avg += speed * ((points[i].time - points[i - 1].time) / total);

                speedings.Add(points[i].time, speed);
            }

            speeds.Add(id, speedings);
            averages.Add(id, avg);
            maximums.Add(id, max);

            average += avg / (float)Points.Count;
        }

    }

    public Vector3 GetInterpolatedLocationAtTime(int id, float time)
    {
        if (Points.TryGetValue(id, out List<DataPoint> points))
        {
            if (points.Count == 0)
                throw new Exception("No data points found for given id");

            for(int i = 0; i < points.Count; i++)
            {
                if (points[i].time > time)
                    if (i == 0)
                        return points[i].position;
                    else
                        return Vector3.Lerp(points[i-1].position, points[i].position, (time - points[i-1].time) / (points[i].time - points[i-1].time));
            }

            return points[points.Count - 1].position;


            /*
             * This code is for unsorted lists
             *

            float nearestLowTime = TimeMin;
            float nearestHighTime = TimeMax;

            Vector3? nearestLowPosition = null;
            Vector3? nearestHighPosition = null;

            foreach (DataPoint point in dataPointsOfId)
            {
                if (Mathf.Approximately(time, point.time))
                    return point.position;
                else if (point.time < time && point.time >= nearestLowTime) {
                    nearestLowTime = point.time;
                    nearestLowPosition = point.position;
                }
                else if (point.time > time && point.time <= nearestHighTime) {
                    nearestHighTime = point.time;
                    nearestHighPosition = point.position;
                }
            }

            if (nearestHighPosition == null)
                return nearestLowPosition.Value;
            else if (nearestLowPosition == null)
                return nearestHighPosition.Value;
            else
                return Vector3.Lerp(nearestLowPosition.Value, nearestHighPosition.Value, (time - nearestLowTime) / (nearestHighTime - nearestLowTime));
            */
        }
        else
        {
            throw new KeyNotFoundException();
        }
    }

    public void LoadData()
    {
        // Building file
        LoadGeoJSON geoJSON = GetComponent<LoadGeoJSON>();
        geoJSON.parent = parent;

        try {
            System.IO.StreamReader GeoJSON = new System.IO.StreamReader(building);
            geoJSON.LoadFeatures(GeoJSON.ReadToEnd());
            GeoJSON.Close();
        }
        catch (System.IO.FileNotFoundException e)
        {
            Debug.LogError(e);
        }

        parent.transform.SetPositionAndRotation(new Vector3(0, 0.5f, 0), Quaternion.Euler(90, 0, 0));


        // Datapoints file
        TimeLength = TimeMax = TimeMin = 0;
        Points = new Dictionary<int, List<DataPoint>>();

        speeds = new Dictionary<int, Dictionary<float, float>>();
        averages = new Dictionary<int, float>();
        maximums = new Dictionary<int, float>();

        try
        {
            System.IO.StreamReader CSV = new System.IO.StreamReader(datapoints);
            string line;
            string[] chunks;

            while (CSV.Peek() > -1)
            {
                line = CSV.ReadLine();
                chunks = line.Split(';');
                if (chunks.Length == 4
                    && int.TryParse(chunks[0], out int id)
                    && float.TryParse(chunks[1], out float time)
                    && double.TryParse(chunks[2], out double longtitude)
                    && double.TryParse(chunks[3], out double latitude))
                {
                    Vector2 v = LoadGeoJSON.ToPixel(
                                        new LoadGeoJSON.GeoCoordinate(longtitude, latitude) - LoadGeoJSON.center);

                    AddDataPoint(id, time, new Vector3(v.x, 0, v.y));
                }

            }

            CSV.Close();
        }
        catch (System.IO.FileNotFoundException e)
        {
            Debug.LogError(e);
        }

        foreach (List<DataPoint> l in Points.Values)
            SortDataPointListByTime(l);
    }

    public void AddDataPoint(int id, float time, Vector3 position)
    {
        DataPoint dp = new DataPoint(id, time, position);

        if (Points.TryGetValue(id, out List<DataPoint> list))
            list.Add(dp);
        else
            Points.Add(id, new List<DataPoint> { dp });

        TimeMin = TimeMin < time ? TimeMin : time;
        TimeMax = TimeMax > time ? TimeMax : time;
        TimeLength = TimeMax - TimeMin;
    }

    private void SortDataPointListByTime(List<DataPoint> l)
    {
        l.Sort((a, b) => {
            if (a.time < b.time)
                return -1;
            if (a.time > b.time)
                return 1;
            return 0;
        });
    }

    public class DataPoint
    {
        public int     id       { get; private set; }
        public float   time     { get; private set; }
        public Vector3 position { get; private set; }

        public DataPoint(int id, float time, Vector3 position)
        {
            this.id = id;
            this.time = time;
            this.position = position;
        }

        public DataPoint(int id, float time, float x, float y, float z)
        {
            this.id = id;
            this.time = time;
            this.position = new Vector3(x, y, z);
        }

        override public string ToString()
        {
            return id + " was at position " + position + " at " + System.TimeSpan.FromSeconds(time).ToString(@"hh\:mm\:ss\:ff");
        }
    }

    public void WriteCSV()
    {
        System.IO.StreamWriter writer = new System.IO.StreamWriter(datapoints);

        foreach (List<DataPoint> dpList in Points.Values)
            foreach (DataPoint dp in dpList)
            {
                writer.WriteLine(dp.id
                + ";" + dp.time
                + ";" + (LoadGeoJSON.MercatorProjection.XToLon(dp.position.x) + LoadGeoJSON.center.longitude)
                + ";" + (LoadGeoJSON.MercatorProjection.YToLat(dp.position.z) + LoadGeoJSON.center.latitude)
                );
            }

        writer.Flush();
        writer.Close();
    }

}
```

Puppet.cs

```csharp
using System.Collections.Generic;
using UnityEngine;

public class Puppet : MonoBehaviour
{
    public int Id { get; set; }

    public float MaxSpeed { get; set; }
    public float AvgSpeed { get; set; }

    public Dictionary<float, float> Speeds { get; set; }
    public Dictionary<int, bool> IsInside { get; set; }

    public void Awake()
    {
        IsInside = new Dictionary<int, bool>();
    }

    public void SetPosition(Vector3 position)
    {
        transform.position = position;
    }

    public bool HasMaxBetween(float l, float u)
    {
        return (l <= MaxSpeed && u >= MaxSpeed);
    }

    public bool IsInsideBetween(float l, float u)
    {
        for (int i = (int)l; i < u; i++)
            if (!IsInside.TryGetValue(i, out bool inside) || !inside) // Lazy evalua
tion ;)
                return false;

        return true;
    }

    public bool IsOutsideBetween(float l, float u)
    {
        for (int i = (int)l; i < u; i++)
            if (!IsInside.TryGetValue(i, out bool inside) || inside) // Lazy evaluat
ion ;)
                return false;

        return true;
    }

    public void Kill()
    {
        Destroy(gameObject);
    }
}
```

Puppeteer

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Puppeteer : MonoBehaviour
{
    public GameObject puppet;
    private Dictionary<int, Puppet> puppets;
    private float average;

    public DataHandler dataHandler;
    public ProjectionInteraction projectionInteraction;
    public OrbitPanZoom orbitPanZoom;

    public Button exit;

    public Button play;
    private Sprite normalSprite;
    private Sprite playingSprite;

    public Button pause;

    public Slider slider;
    public Text textTime;

    public Button speedUp;
    public Button speedDown;
    public Text textSpeed;

    private int speed;
    private float[] speeds = { 0.5f, 1, 2, 5, 10, 30, 60, 120};

    public Selector PeopleInside;
    public Selector AverageSpeed;

    public Selector SpeedGraph;
    public Text MaxSpeed;
    public Text AvgSpeed;

    public Text totalSelected;

    private float timer;
    private bool playing;
    private bool ignore;

    private void Awake()
    {
        play.onClick.AddListener(delegate { Play(); });
        pause.onClick.AddListener(delegate { Pause(); });
        slider.onValueChanged.AddListener(delegate { Slide(); });
        speedUp.onClick.AddListener(delegate { ClickUp(); });
        speedDown.onClick.AddListener(delegate { ClickDown(); });
        exit.onClick.AddListener(delegate { Exit(); });

        normalSprite = play.image.sprite;
        playingSprite = play.spriteState.pressedSprite;

        Invoke("LoadData", 0.05f);
    }

    private void LoadData()
    {
        dataHandler.LoadData();
        Invoke("Init", 0.1f);
    }

    private void FixedUpdate()
    {
        if (playing)
        {
            timer += Time.deltaTime * speeds[speed];
            if(timer >= dataHandler.TimeLength)
            {
                timer = dataHandler.TimeLength;
                Pause();
            }

            ignore = true;
            slider.value = timer / dataHandler.TimeLength;
        }

        if (PeopleInside.HasChanged() || AverageSpeed.HasChanged())
            UpdatePuppets();

        orbitPanZoom.enableControls = projectionInteraction.IsHovering;

        if (Input.GetMouseButtonUp(0) || Input.GetMouseButtonUp(1) || Input.GetMouseButtonUp(2) || Input.mouseScrollDelta.sqrMagnitude != 0) //sqrMagnitude is way faster than magnitude
            UnclickedPuppet();

        if(projectionInteraction.GetClicked(out RaycastHit hit, out Vector3 pos))
        {
            Puppet p = hit.transform.gameObject.GetComponent<Puppet>();
            if (p)
                ClickedPuppet(p, pos);
        }

    }

    public void Init()
    {
        // Generate fake data
        //TestPuppet();
        //dataHandler.WriteCSV();

        dataHandler.CalculateSpeeds();
        average = dataHandler.GetAverageSpeed();

        ResetPuppets();
        GeneratePuppets();

        MeasurePuppets();
        CollidePuppets();

        Slide();
        ClickUp();

        UnclickedPuppet();
    }

    private void Play()
    {
        if (timer >= dataHandler.TimeLength)
            timer = 0;

        playing = true;
        play.image.sprite = playingSprite;
    }

    private void Pause()
    {
        playing = false;
        play.image.sprite = normalSprite;
    }

    private void Slide()
    {
        if (ignore)
            ignore = false;
        else if (dataHandler.TimeLength > slider.value)
            timer = slider.value * dataHandler.TimeLength;

        textTime.text = System.TimeSpan.FromSeconds(timer).ToString(@"hh\:mm\:ss");

        PositionPuppets();
    }

    private void ClickUp()
    {
        speed++;
        speed %= speeds.Length;

        textSpeed.text = speeds[speed].ToString() + 'x';
    }

    private void ClickDown()
    {
        if (speed == 0)
            speed = speeds.Length;

        speed--;
        speed %= speeds.Length;

        textSpeed.text = speeds[speed].ToString() + 'x';
    }

    public void Exit()
    {
        Application.Quit();
    }

    private void ClickedPuppet(Puppet p, Vector3 pos)
    {
        MaxSpeed.text = "Maximum speed: " + (p.MaxSpeed * 3.6f).ToString("0.0") + "km/h";
        AvgSpeed.text = "Average speed: " + (p.AvgSpeed * 3.6f).ToString("0.0") + "km/h";

        Dictionary<float, float> speeds = p.Speeds;

        Selector.DataSet data = new Selector.DataSet();

        data.SetPair(Mathf.RoundToInt(dataHandler.TimeMin), 0);

        foreach (KeyValuePair<float, float> pair in speeds)
            data.SetPair(Mathf.RoundToInt(pair.Key), Mathf.RoundToInt(pair.Value * 100));

        data.SetPair(data.xMax + 1, 0);
        data.SetPair(Mathf.RoundToInt(dataHandler.TimeMax) + 1, 0);

        SpeedGraph.RenderGraph(data);

        Texture2D rentex = SpeedGraph.graph.texture as Texture2D;
        int average = Mathf.RoundToInt(
            Mathf.Clamp(
                    Scale(data.yMin, data.yMax,
                          SpeedGraph.area.yMin, SpeedGraph.area.yMax,
                          dataHandler.GetAverageSpeed()),
                SpeedGraph.area.yMin, SpeedGraph.area.yMax
            ) * 100
        );

        SpeedGraph.DrawLine(ref rentex,
            new Vector2(SpeedGraph.area.xMin, average),
            new Vector2(SpeedGraph.area.xMax, average),
            Color.magenta, 4);

        rentex.Apply();
        SpeedGraph.graph.texture = rentex;

        pos.y += 100;
        SpeedGraph.SetPosition(pos);
        SpeedGraph.Show();
    }

    private void UnclickedPuppet()
    {
        SpeedGraph.Hide();
    }

    private void PositionPuppets()
    {
        foreach (Puppet p in puppets.Values)
        {
            try
            {
                p.SetPosition(dataHandler.GetInterpolatedLocationAtTime(p.Id, timer)
);
            }
            catch (KeyNotFoundException e)
            {
                Debug.LogException(e);
            }
        }
    }

    private void GeneratePuppets()
    {
        GameObject o;
        Puppet p;
        foreach (int id in dataHandler.Points.Keys)
        {
            o = Instantiate(puppet);
            p = o.GetComponent<Puppet>();

            p.Id = id;
            puppets.Add(id, p);
        }
    }

    private void ResetPuppets()
    {
        if (puppets != null)
            foreach (Puppet puppet in puppets.Values)
                puppet.Kill();

        puppets = new Dictionary<int, Puppet>();
    }

    private void UpdatePuppets()
    {
        if (puppets == null) return;

        int total = 0;

        foreach (Puppet p in puppets.Values)
        {
            /*
             * Basically: active if within bounds, or both selectors are set to full range
             * In other words:
             *

            bool maxinrange = AverageSpeed.IsFullRange() || p.HasMaxBetween(AverageSpeed.GetLowerBound(), AverageSpeed.GetHigherBound());

            bool insidebetween = p.IsInsideBetween(dataHandler.TimeMin, PeopleInside.GetLowerBound());
            bool ousidebetween = p.IsOutsideBetween(PeopleInside.GetHigherBound(), dataHandler.TimeMax);
            bool insideinrange = PeopleInside.IsFullRange() || (insidebetween && ousidebetween);

            bool active = maxinrange && insideinrange;
            */

            bool active =
                (p.HasMaxBetween(AverageSpeed.GetLowerBound(), AverageSpeed.GetHigherBound())
                    || AverageSpeed.IsFullRange())
                &&
                (p.IsInsideBetween(dataHandler.TimeMin, PeopleInside.GetLowerBound())
                    && p.IsOutsideBetween(PeopleInside.GetHigherBound(), dataHandler.TimeMax)
                    || PeopleInside.IsFullRange()
                );

            p.gameObject.SetActive(active);

            if (active)
                total++;
        }

        totalSelected.text = "Total selected: " + total;
    }

    private void MeasurePuppets()
    {
        /* 10 speed classifications
         * 0-3.6 km/h, 3.6-7.2 km/h, 7.2-10.8 km/h, 10.8-14.4 km/h, 14.4-18 km/h
         * 18-21.6 km/h, 21.6-25.2 km/h, 25.2-28.8 km/h, 28.8-32.4 km/h, >32.4 km/h (Usain Bolts, or just bugs)
         */
        int[] speeds = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        int speed;
        foreach (Puppet p in puppets.Values)
        {
            p.Speeds = dataHandler.GetSpeeds(p.Id);
            p.MaxSpeed = dataHandler.GetMaximumAchievedSpeed(p.Id);
            p.AvgSpeed = dataHandler.GetAverageSpeed(p.Id);

            speed = Mathf.RoundToInt(p.MaxSpeed);
            speeds[speed < 10 ? speed : 10]++;
        }
```

```csharp

        Selector.DataSet data = new Selector.DataSet();
        for (int x = 0; x < speeds.Length; x++)
            data.SetPair(x, speeds[x]);

        AverageSpeed.RenderGraph(data);
    }

    private void CollidePuppets()
    {
        // Get all the MeshColliders from all the children of the parent GameObject from DataHandler
        MeshCollider[] colliders = dataHandler.parent.GetComponentsInChildren<MeshCollider>();

        Selector.DataSet dataCollisions = new Selector.DataSet();
        for (int t = (int) dataHandler.TimeMin; t <= (int) dataHandler.TimeMax; t++)
        {
            int collisions = 0;

            foreach (Puppet p in puppets.Values)
            {
                bool collided = false;
                for (int m = 0; m < colliders.Length && !collided; m++) // Stop when any collision occurres, speed saving when multiple meshes
                    collided = IsPuppetCollidingAtTime(p, colliders[m], t);

                if (collided)
                    collisions++;

                p.IsInside.Add(t, collided);
            }

            dataCollisions.SetPair(t, collisions);
        }

        PeopleInside.RenderGraph(dataCollisions);
    }

    private bool IsPuppetCollidingAtTime(Puppet puppet, Collider collider, float time)
    {
        Vector3 position = dataHandler.GetInterpolatedLocationAtTime(puppet.Id, time);
        return (position == collider.ClosestPoint(position));
    }

    private float Scale(float OldMin, float OldMax, float NewMin, float NewMax, float OldValue)
    {
        float OldRange = (OldMax - OldMin);
        float NewRange = (NewMax - NewMin);
        float NewValue = OldRange != 0 ? (((OldValue - OldMin) * NewRange) / OldRange) + NewMin : NewMin;

        return (NewValue);
    }

    private void TestPuppet()
    {
        int neverInside = 5;
        int neverOutside = 10;
        int Runners = 10;
        int Laggers = 10;
        int Normies = 30;

        float timeStart = 0;
        float timeOfAlarm = 10 * 60;
        float timeEnd = 60 * 60;
        float timeInterval = 1;
        float timeVariance = 5;

        Vector3 exit = new Vector3(67.33f, 0f, 10.15f);
        Vector3 congregate = new Vector3(103f, 0f, 4.7f);

        List<Vector3> startingPoints = new List<Vector3>();

        Transform g = dataHandler.parent.transform.Find("Ravelijn");
        if (!g)
            return;

        Renderer r = g.GetComponentInChildren<Renderer>();
        Collider c = g.GetComponentInChildren<Collider>();
        if (!(r && c))
            return;

        Vector3 center = r.bounds.center;
        float radius = r.bounds.extents.magnitude;

        int id = 0;
        float time = 0;
        float deltaTime = 0;
        Vector3 deltaPosition = Vector3.zero;

        while (neverInside > 0
            || neverOutside > 0
            || Runners > 0
            || Laggers > 0
            || Normies > 0)
        {
            Vector3 random = RandomOnFloorCircle(center, radius);

            if (random == c.ClosestPoint(random))
            {
                if (neverOutside > 0)
                {
                    time = Mathf.Abs(NextGaussian(timeStart, timeVariance));
                    float end = timeEnd - NextGaussian() * timeVariance;

                    // Just hang around same spot
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance,
1, 15);

                        do
                        {
                            deltaPosition = FlatVector3(Random.insideUnitCircle.norm
alized
                                * NextGaussianClamped(0, 0.005f, 0, 2)) * deltaTime;

                        } while (c.ClosestPoint(random + deltaPosition) != random +
deltaPosition);

                        random += deltaPosition;
                        time += deltaTime;

                    } while (time < end);

                    neverOutside--;
                    id++;
                }
                else if (Runners > 0)
                {
                    time = Mathf.Abs(NextGaussian(timeStart, timeVariance));

                    // Stick around untill alarm
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

                        do
                        {
                            deltaPosition = FlatVector3(Random.insideUnitCircle.normalized
                                * NextGaussianClamped(0, 0.2f, 0, 2)) * deltaTime;
                        } while (c.ClosestPoint(random + deltaPosition) != random + deltaPosition);

                        random += deltaPosition;
                        time += deltaTime;

                    } while (time <= timeOfAlarm);

                    // Move towards exit
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

                        do
                        {
                            deltaPosition = (exit - random).normalized
                                * NextGaussianClamped(2, 1, 0, 7) * deltaTime;

                            deltaPosition = Vector3.RotateTowards(deltaPosition,
                                FlatVector3(Random.insideUnitCircle.normalized),
                                Mathf.Deg2Rad * 10f, 0f);

                            deltaPosition.y = 0;
                        } while (c.ClosestPoint(random + deltaPosition) != random + deltaPosition);

                        random += deltaPosition;
                        time += deltaTime;

                        // Make sure the end time always reflects everybody getting out (otherwise it's a never getting outside)
                        timeEnd = Mathf.Max(time, timeEnd);

                    } while (!IsInsideFloorCircle(exit, 5, random));

                    // Move towards congregation spot
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

                        deltaPosition = (congregate - random).normalized
                            * NextGaussianClamped(2, 1, 0, 7) * deltaTime;

                        deltaPosition = Vector3.RotateTowards(deltaPosition,
                            FlatVector3(Random.insideUnitCircle.normalized),
                            Mathf.Deg2Rad * 10f, 0f);

                        deltaPosition.y = 0;

                        random += deltaPosition;
                        time += deltaTime;

                    } while (!IsInsideFloorCircle(congregate, 5, random) && time < timeEnd);

                    // Stay on congregation spot
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

                        do
                        {
                            deltaPosition = FlatVector3(Random.insideUnitCircle.normalized
                                * NextGaussianClamped(0, 0.01f, 0, 1)) * deltaTime;
                        } while (!IsInsideFloorCircle(congregate, 10, random + deltaPosition));

                        random += deltaPosition;
                        time += deltaTime;

                    } while (time < timeEnd);

                    Runners--;
                    id++;
                }
                else if (Laggers > 0)
                {
                    time = Mathf.Abs(NextGaussian(timeStart, timeVariance));
                    float commenceEgress = NextGaussianClamped(timeOfAlarm + 10 * 60, 5 * 60, timeOfAlarm, timeOfAlarm + 20 * 60);

                    // Stick around untill alarm
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

                        do
                        {
                            deltaPosition = FlatVector3(Random.insideUnitCircle.normalized
                                * NextGaussianClamped(0, 0.1f, 0, 1)) * deltaTime;
                        } while (c.ClosestPoint(random + deltaPosition) != random + deltaPosition);

                        random += deltaPosition;
                        time += deltaTime;

                    } while (time <= commenceEgress);

                    // Move towards exit
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
```

```
            deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

            do
            {
              deltaPosition = (exit - random).normalized
                * NextGaussianClamped(0, 0.2f, 0, 2) * deltaTime;

              deltaPosition = Vector3.RotateTowards(deltaPosition,
                FlatVector3(Random.insideUnitCircle.normalized),
                Mathf.Deg2Rad * 30f, 0f);

              deltaPosition.y = 0;
            } while (c.ClosestPoint(random + deltaPosition) != random + deltaPosition);

            random += deltaPosition;
            time += deltaTime;

            // Make sure the end time always reflects everybody getting out (otherwise it's a never getting outside)
            timeEnd = Mathf.Max(time, timeEnd);

          } while (!IsInsideFloorCircle(exit, 5, random));

          // Move towards congregation spot
          do
          {
            dataHandler.AddDataPoint(id, time, random);
            deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

            deltaPosition = (congregate - random).normalized
              * NextGaussianClamped(0, 0.2f, 0, 2) * deltaTime;

            deltaPosition = Vector3.RotateTowards(deltaPosition, FlatVector3(Random.insideUnitCircle.normalized), Mathf.Deg2Rad * 30f, 0f);

            deltaPosition.y = 0;

            random += deltaPosition;
            time += deltaTime;

          } while (!IsInsideFloorCircle(congregate, 5, random) && time < timeEnd);

          // Stay on congregation spot
          do
          {
            dataHandler.AddDataPoint(id, time, random);
            deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

            do
            {
              deltaPosition = FlatVector3(Random.insideUnitCircle.normalized
                * NextGaussianClamped(0, 0.1f, 0, 2)) * deltaTime;
            } while (!IsInsideFloorCircle(congregate, 5, random + deltaPosition));

            random += deltaPosition;
            time += deltaTime;

          } while (time < timeEnd);

```

```
                    Laggers--;
                    id++;
                }
                else if (Normies > 0)
                {
                    time = Mathf.Abs(NextGaussian(timeStart, timeVariance));

                    // Stick around untill alarm
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

                        do
                        {
                            deltaPosition = FlatVector3(Random.insideUnitCircle.normalized
                                * NextGaussianClamped(0, 0.2f, 0, 2)) * deltaTime;
                        } while (c.ClosestPoint(random + deltaPosition) != random + deltaPosition);

                        random += deltaPosition;
                        time += deltaTime;

                    } while (time <= timeOfAlarm);

                    // Move towards exit
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

                        do
                        {
                            deltaPosition = (exit - random).normalized
                                * NextGaussianClamped(0, 0.2f, 0, 2) * deltaTime;

                            deltaPosition = Vector3.RotateTowards(deltaPosition,
                                FlatVector3(Random.insideUnitCircle.normalized),
                                Mathf.Deg2Rad * 30f, 0f);

                            deltaPosition.y = 0;
                        } while (c.ClosestPoint(random + deltaPosition) != random + deltaPosition);

                        random += deltaPosition;
                        time += deltaTime;

                        // Make sure the end time always reflects everybody getting out (otherwise it's a never getting outside)
                        timeEnd = Mathf.Max(time, timeEnd);

                    } while (!IsInsideFloorCircle(exit, 5, random));

                    // Move towards congregation spot
                    do
                    {
                        dataHandler.AddDataPoint(id, time, random);
                        deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

                        deltaPosition = (congregate - random).normalized
                            * NextGaussianClamped(0, 0.2f, 0, 2) * deltaTime;

```

```
            deltaPosition = Vector3.RotateTowards(deltaPosition, FlatVector3(Random.insideUnitCircle.normalized), Mathf.Deg2Rad * 30f, 0f);

            deltaPosition.y = 0;

            random += deltaPosition;
            time += deltaTime;

          } while (!IsInsideFloorCircle(congregate, 5, random) && time < timeEnd);

          // Stay on congregation spot
          do
          {
            dataHandler.AddDataPoint(id, time, random);
            deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

            do
            {
              deltaPosition = FlatVector3(Random.insideUnitCircle.normalized
                * NextGaussianClamped(0, 0.1f, 0, 2)) * deltaTime;
            } while (!IsInsideFloorCircle(congregate, 10, random + deltaPosition));

            random += deltaPosition;
            time += deltaTime;

          } while (time < timeEnd);

          Normies--;
          id++;
        }
      }
      else
      {
        if(neverInside > 0)
        {
          time = Random.Range(timeStart, timeEnd);

          do
          {
            dataHandler.AddDataPoint(id, time, random);
            deltaTime = NextGaussianClamped(timeInterval, timeVariance, 1, 15);

            do
            {
              deltaPosition = FlatVector3(
                Random.insideUnitCircle.normalized *
                Random.Range(1 * deltaTime, 5 * deltaTime));
            } while (c.ClosestPoint(random + deltaPosition) == random + deltaPosition);

            random += deltaPosition;
            time += deltaTime;

          } while (Random.value > 0.5f && time < timeEnd);

          neverInside--;
          id++;
        }
      }
    }
  }
```

```

    public static Vector3 FlatVector3(Vector2 v)
    {
        return new Vector3(v.x, 0, v.y);
    }

    private static bool IsInsideFloorCircle(Vector3 center, float radius, Vector3 point)
    {
        return ((new Vector2(center.x, center.z)) - (new Vector2(point.x, point.z))).sqrMagnitude <= radius * radius;
    }

    private static Vector3 RandomOnFloorCircle(Vector3 center, float radius)
    {
        Vector2 random = Random.insideUnitCircle * radius;
        return new Vector3(random.x + center.x, center.y, random.y + center.y);
    }

    private static float NextGaussianClamped(float mean, float variance, float min, float max)
    {
        float val;

        do
        {
            val = NextGaussian(mean, variance);
        } while (val < min || val > max);

        return val;
    }

    private static float NextGaussian(float mean, float variance)
    {
        return NextGaussian() * variance + mean;
    }

    private static float NextGaussian()
    {
        // Non-zero point inside unit circle
        Vector2 random = Random.insideUnitCircle;
        while (random.Equals(Vector2.zero))
        {
            random = Random.insideUnitCircle;
        }

        float fac = Mathf.Sqrt(-2.0f * Mathf.Log(random.sqrMagnitude) / random.sqrMagnitude);

        return random.y * fac;
    }
}
```

ProjectionInteraction.cs

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class ProjectionInteraction : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler, IPointerClickHandler
{
    public Camera cam;

    [HideInInspector]
    public bool IsHovering;

    private Vector3 campos;
    private RaycastHit? clicked;
    private Vector3 clickpos;

    public void Start()
    {
        campos = Vector3.zero;
        IsHovering = false;
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        IsHovering = true;
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        IsHovering = false;
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        RectTransform rt = GetComponent<RectTransform>();

        Vector3[] corners = new Vector3[4];
        rt.GetWorldCorners(corners);

        clickpos = eventData.position;

        campos = new Vector3(clickpos.x - corners[0].x, clickpos.y - corners[0].y);

        campos = scale(new Vector3(rt.rect.width, rt.rect.height),
                      new Vector3(cam.pixelWidth, cam.pixelHeight),
                      campos);

        Ray ray = cam.ScreenPointToRay(campos);

        if (Physics.Raycast(ray, out RaycastHit hit, 500f))
        {
            clicked = hit;
        }
    }

    public bool GetClicked(out RaycastHit hit, out Vector3 pos)
    {
        bool and = clicked.HasValue;
        hit = clicked.GetValueOrDefault();
        pos = clickpos;
        clicked = null;
        clickpos = Vector3.zero;

        return and;
    }

    private float scale(float OldMin, float OldMax, float NewMin, float NewMax, float OldValue)
    {
        float OldRange = (OldMax - OldMin);
        float NewRange = (NewMax - NewMin);
        float NewValue = OldRange != 0 ? (((OldValue - OldMin) * NewRange) / OldRange) + NewMin : NewMin;

        return (NewValue);
    }

    private Vector2 scale(Vector2 from, Vector2 to, Vector2 old)
    {
        return new Vector3(scale(0, from.x, 0, to.x, old.x),
            scale(0, from.y, 0, to.y, old.y));
    }

    private Vector3 scale(Vector3 from, Vector3 to, Vector3 old)
    {
        return new Vector3(scale(0, from.x, 0, to.x, old.x),
            scale(0, from.y, 0, to.y, old.y),
            scale(0, from.z, 0, to.z, old.z));
    }
}
```