UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

**Aleksander Nikolajev**

# Delta Building Visualisation and Optimisation

**Bachelor's Thesis (9 ECTS)**

Supervisor: Raimond-Hendrik Tunnel, MSc

Tartu 2018

## Delta Building Visualisation and Optimisation

**Abstract:**

During this thesis, a visualisation of an academic building model in the Unity game engine was created. The thesis describes the optimisations and pathfinding solution for simulated people, as well as the design principles used to make the visualisation enjoyable for the viewer. The thesis concludes with testing the optimisation and pathfinding, along with verifying if the visualisation was pleasant to watch.

**Keywords:**

Visualisation, optimization, animations, pathfinding, colours, Unity, design

**CERCS:** P170 Computer Science, numerical analysis, systems, control

## Delta Õppehoone Visualiseerimine ja Optimeerimine

**Lühiskokkuvõte:**

Antud töös loodi õppehoone visualisatsioon Unity mängu mootori kasutades. Lõputöö kirjeldab visualisatsiooni optimeerimist ning simuleeritud inimeste raja leidmise lahendust. Lisaks oli arendatud visualisatsiooni diasin Delta õppehoone visualisatsiooni vaatajate jaoks selleks, et muuta visualisatsiooni kasutajasõbralikumaks ja ilusamaks. Töö käigus oli tehtud optimeerimise ja raja leidmise testimine. Lisaks oli katsetatud visualisatsiooni meeldivus kasutajatele.

**Võtmesõnad:**

Visualisatsioon, optimeerimine, animatsioonid, raja leidmine, värvid, Unity, disain

**CERCS:** P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

# Contents

# 1. Introduction

With big corporations constantly improving work atmosphere using methods such as encouraging to have fun during work [1], it is good to come up with different strategies to encourage said fun in studying environments. It is also important to motivate people in those work environments, because, as explained by Tohidi on the effects of motivation, "motivation powers people to perform better" [5]. As explain by Kim *et al* on the topic of effects of animations in interest, "due to their aesthetical appeal and interest, animations provide a motivation boost" [6].

The goal of this thesis is to create a 3D visualisation of the Delta building. The Delta building itself is an academic building of many institutes, one of them being Institute of Computer Science[1]. Also, one of the goals of the visualisation is to potentially provide insight on how the visualisation can improve work atmosphere (chapter 5) and positively impact the people viewing the said visualisation (chapter 3). The visualisation is created in Unity[2] – a game engine, which is often used to develop three-dimensional games and simulations. The visualisation itself simulates how students and educators behave during their time in an academic building (chapter 4). Furthermore, the visualisation is done in cooperation with Andrei Voitenko. The goal of his thesis is to provide simulated sensor data and schedule data to this part of the visualisation [11]. The sensor data will be used to showcase the student and educator behaviour. Lastly, the thesis will provide insight on what future developers or students of this visualisation can do in other to further improve said visualisation (chapter 6).

The model of the Delta building was provided by the architecture firm Arhitekt11 OÜ. The visualisation is planned to be displayed on either Video Wall Screens inside the Delta building. The screen is expected to be 6 meters wide and 3 meters tall and is expected to be placed on the first floor near the main entrance. This provides a good opportunity for any passing viewers to notice the visualisation. As explained by Jin in his research on role of animations in the consumer marker: "Animations makes a product become capable of eliciting a positive emotion and attitude" [2], the visualisation can be seen as "selling" the opportunities Computer Science Bachelor curriculum can provide to prospective students and motivate them to enrol.

[1] https://www.ut.ee/en/news/new-academic-building-narva-mnt-4-will-be-called-delta
[2] https://unity3d.com/

Visualisation features three-dimensional objects, which represent people in the building. They have the ability to determine their destination. They also imitate student and educator behaviour using abstract animations. For the purpose of this thesis, these three-dimensional objects will be called Actors. Actors will have their own colour and shape in order to distinguish students from educators (chapter 3). Other abbreviations or definitions, as well as additional files and tables, can be found in the appendix.

## 2. Requirements

Because an academic building and the people in it will be visualised, the visualisation is expected to support a high Actor. The visualisation should be pleasant for the viewer to watch, as the goal is to motivate prospective students to enrol. Furthermore, this project expects input from sensor data in order to guide the Student Actors to their destinations and schedule data to guide Educator Actors. For these reasons, both functional and non-functional requirements were defined in order to fulfil the above mentioned criteria. These requirements also served as main goals for the development.

### 2.1 Functional Requirements

Functional requirements focus on Actor destinations from the sensor data and the overall look of the visualisation. Five functional requirements were created to define the Actor behaviour (F1, F2) and animations (F3). They also provide information on the visualised areas (F4) and the way the building model should be furnished (F5).

**F1. Student Actor destination from sensor mock data**

Student Actor movement in the visualisation should be based on sensor mock data. Because the building is not built, data simulation will be used. Sensor mock data will be provided in the form of a room name and the amount of people going to that room by Andrei Voitenko [11].

**F2. Educator Actor destination from a schedule**

Educator Actor destination should be set once an educator is needed in any of the classrooms. This should be determined by a SIS[3] schedule - a Study Information System that manages a range of information for students in Info Technology (IT). The SIS schedule gives the Educator Actor the time it has to leave and the room it is going to via a Schedule Module created by Andrei Voitenko [11].

**F3. Actor animations based on their activity**

Actor animations should be based on the classroom the Actor is going to and what kind of table the Actor is going to be sitting at once it has reached its destination. Educator Actors will be animated as if they were education the students in the classroom. Student Actors will be animated as if they were studying.

[3] https://itservices.usc.edu/sis/about/

**F4. Visualised areas with high student presence**

The visualisation should only feature areas of the building where a high student presence is expected. These places are the first floor, second floor and the outside area. First and second floors will be visualised because they have lecture halls, labs, seminar rooms etc., while the outside area will be visualised because the Actors can be often seen going into the visualised building model. Third and fourth floors will not be visualised because the student presence is low due to these floors mostly having educator offices.

**F5. The furnishment of the visualisation**

Most of the rooms in the given Delta building model were empty and should be furnished with suitable furniture models. The amount of furniture models expected in each room was written in the building's architectural blueprint. The blueprint of the building was provided by Administrative Manager of the Institute of Computer Science, Piret Orav.

## 2.2 Non-Functional Requirements

Non-functional requirements focus on the performance of the visualisation, as well as defining the emotional factors[4] of the visualisation. Non-functional requirements explain the amount of expected Actor number (NF1), the performance of the visualisation (NF2, NF3), the planned views (NF4) and the planned viewer experience (NF5).

**NF1. The visualisation should support a maximum of 2010 Actors**

As stated by the Administrative Manager of the Institute of Computer Science, Piret Orav, the building is expected to support up to (but not limited to) 1600 students and 410 employees (e.g. educators). Thus the visualisation should support a maximum of 2010 concurrent Actors. Note that during testing, 2010 Actors will be scattered throughout the visualisation, meaning that the Actors might not all be seen in the visualisation in any given moment.

**NF2. The expected frame rate of the visualisation**

Pourazad *et al* state in their research on the effects of frame rate that "good frame rate (FPS) improves the Quality of Experience[5] - a measure of delight or annoyance of a customer's experience. Also, if a scene contains fast moving objects, it is best to run the scene at higher than 30 FPS" [4]. Thus the FPS of the visualisation should be above 30 FPS.

[4] https://en.wikipedia.org/wiki/Non-functional_requirement
[5] https://en.wikipedia.org/wiki/Quality_of_experience

In order to achieve this kind of performance, a modern PC with good hardware is required. Below you can see the hardware that was used when creating and testing the visualisation's performance for this thesis:

OS: Microsoft Windows 10 Enterprise x64 bit

Processor: Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz

GPU: NVIDIA GeForce GTX 980 Ti, 4 GB RAM

RAM: 8 GB

**NF3. Actors should find a path to their destination in 2 seconds or less**

With multiple possible spawn points (subchapter 4.2) and a 125 different rooms, finding paths for all those locations should be relatively fast so that actors do not stand idly in their current location when they are required to move.

**NF4. Views should be logically placed**

The visualisation should feature multiple different views[6] in the three-dimensional environment of the building as to cover most of the interesting areas of the Delta building model. This means the views should have a logical position to allow the visualised areas to be seen in detail. Three viewpoints for each floor should allow the viewer of the visualisation to get a better representation of the activities that are happening on the given floor. The vertex (see the glossary) count for each viewpoint in the visualisation should be below 3M[7], as recommended by the official Unity documentation on graphics optimisation.

**NF5. Design of the Visualisation**

The visualisation should be simple and understandable to the viewer. Colours, the animations and student behaviour in the visualisation should provoke a positive response to the viewers of the visualisation and potentially improve work atmosphere.

[6] https://en.wikipedia.org/wiki/Viewpoints
[7] https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html

# 3. Design

Design is a process of creating a solution for the functional and aesthetic demands of the viewer[8]. In the case of this thesis, design is necessary as it is expected that a viewer should enjoy watching the visualisation (NF5). The design of the visualisation is based on a minimalistic aesthetic - an aesthetic that reduces the visualisation to its necessary elements[9]. The most important elements of the visualisation are the Actors, their animations and the building model. In order to distinguish them from other objects in the visualisation, each object has its own defining colour.

The following chapters describe different design ideas and principles that were used in the making of the visualisation. Design principles such as game design and film/animation principles and data visualisation were analysed in order to satisfy requirements F3, F5 and NF5.

## 3.1 Colour Palette

As Rost states in her guide to data visualisation: "In data visualisation, it is important that colour in data highlights the important areas to the viewer" [8]. Furthermore, Zhang adds in her explanation of colour distinguishment that "these colours should allow the viewer to easily distinguish between different sets of data" [9]. In the case of this visualisation, we can assume that the important areas to be highlighted and distinguished are the Actors.

Regarding the design of colours, different colour palettes allow the creation of different moods. Some colour palettes follow patterns that feel more natural to the viewer [9]. In game design, colour palettes affect the mood of the players, giving them different feelings and atmosphere. For example, warm colours make you feel comfortable, while cool tones give a more futuristic feel, as explained by Knez and Niedenthral in their research on the effects of colours on play performance [10]. Thus the visualisation should not only highlight important objects, but also provide an academic and aspiring feel of a university.

Because the visualisation is of a real-life building, some colours are already placed[10]. These include staircases, doors, inside walls and other structural objects. When picking the other colours for the visualisation, these colours were taken into account.

[8] http://www.svid.se/en/What-is-design/Definition-of-design/
[9] http://itprotech.eu/index.php/en/incdtp-2/modele/minimalist
[10] http://ehitustrust.ee/2018/01/tartu-ulikooli-it-keskus-delta/

Colours for the different furniture, rooms and the Actors were determined by creating three distinct colour palettes. An online software Coolors[11] was used to create the colour palettes.

The following chapter describes the reasoning behind the Actor colours. The next chapter explains the remaining picked colours for the visualisation.

### 3.1.1 Actor Colour

According to Kurt and Osueke when explaining the effects of colours on college students: "Blue encourages intellectual activity and logical thought" [3]. Thus it seemed suitable to make the Student Actors blue, as they are associated with learning and knowledge gathering at schools or higher education institutions.

Kurt and Osueke also state: "Yellow is associated with comedy and optimism" [3]. Because of that, the educators were chosen to be yellow to give the visualisation a joyful feeling to the viewers. Both Actors can be seen in Figure 1.

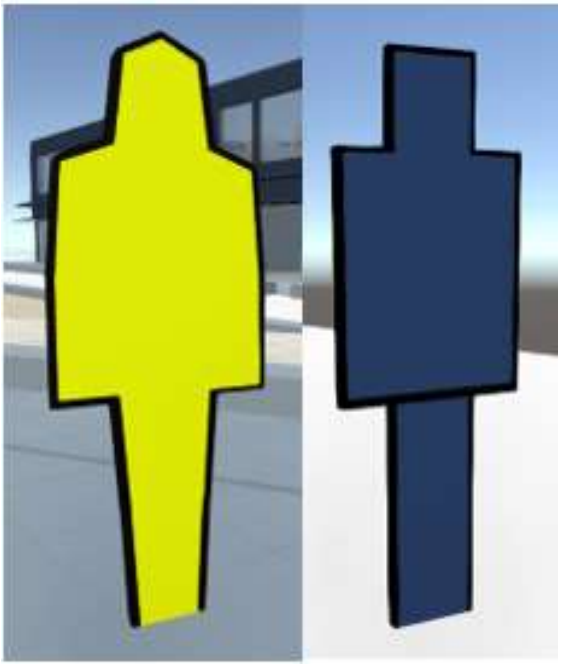

*Figure 1. Educator Actor on the left, Student Actor on the right.*

### 3.1.2 Final Colour Palettes

When generating a colour, the following factors were considered:

- Least favourite colours that of GRAPH 5 in Figure 2 were strongly avoided.
- Favourite colours that of GRAPH 6. in Figure 2 were picked more often.
- Shades of blue and yellow were avoided as they were already used for the Actors.
- Warm colours were supported because they increase arousal and stimulation [3].

[11] https://coolors.co/

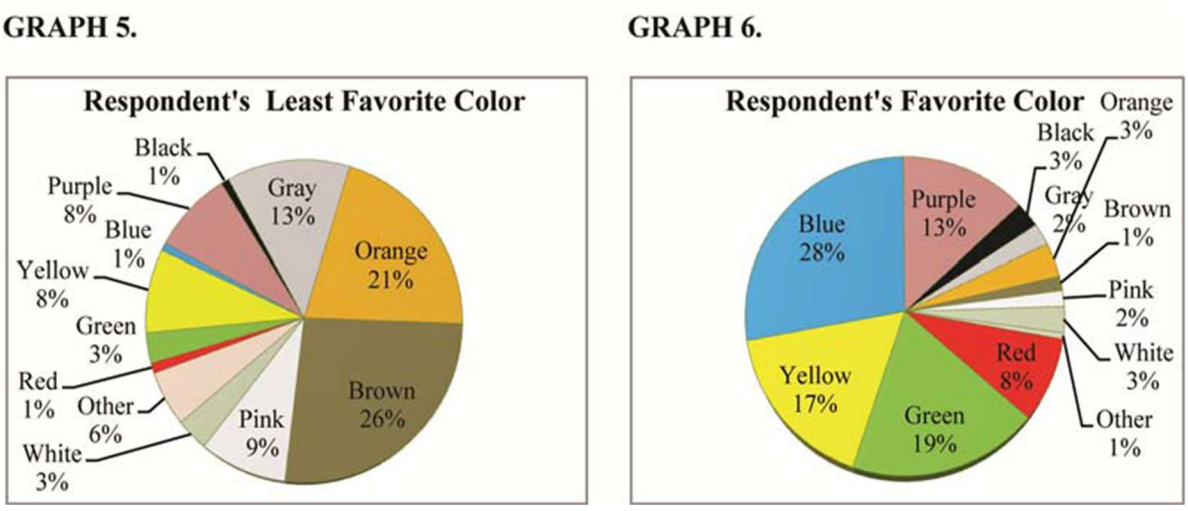


*Figure 2. Graph 5 and Graph 6 from research done by Kurt and Osueke [3:8]. Graphs describe the least and most favourite colour of university students.*

One of the colour palettes that was generated can be seen on Figure 3. Colours were chosen by the factors mentioned in chapter 3.1.2. Dark blue (#293241) was chosen to colour the outside walls of the Delta building based on the images already on the web and videos of the Delta building provided by Piret Orav. Blue in this case are the Student Actors; Educator Actor colour can be found on Figure 4. Red was used on furniture models in the visualisation. Lastly, green was used for areas outside to represent grass.

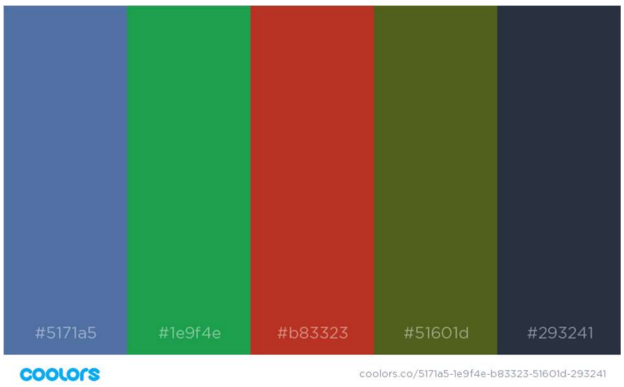


*Figure 3. The first colour palette*

The second colour palette can be seen on Figure 4. These colours helped add a less gloomy feel to the visualisation. Some objects in the visualisation had to be blue as well, but in order to not use the same colour as the Student Actors, a different blue colour was picked. The beige colour was used to colour the floors of the visualisation. A different shade of red was picked to distinguish between those objects that had the red colour of Figure 3. Yellow was used to represent Educator Actors.

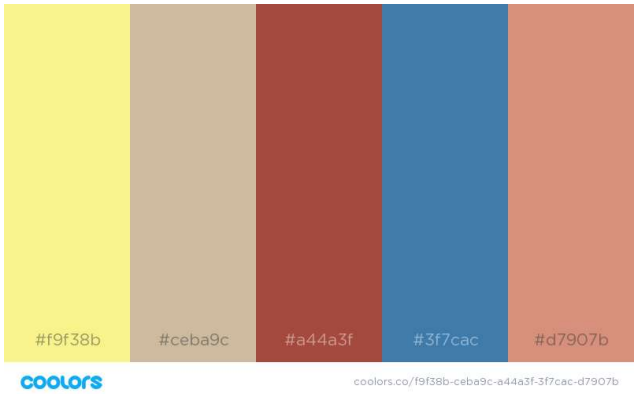


*Figure 4. The second colour palette*

The final colour palette, Figure 5, has darker colours that were used for wooden objects and roads around the building. The last colour in *Figure 5* was used to distinguish some wooden parts from others, while the first one was the primary colour for all wooden objects. White was used on objects such as paper, sinks, toilets and some walls. The second and third colours were for roads, keyboards and computer screens.

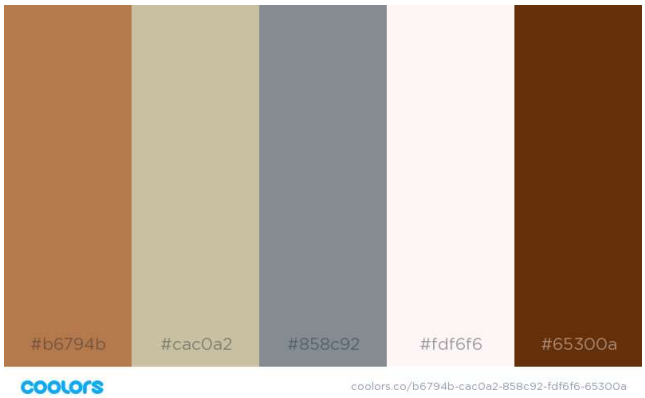


*Figure 5. The third colour palette*

## 3.2 Minimalistic Aesthetics

Because of the high actor count (NF1) and performance requirements (NF2), simplistic animations with little actions were chosen. The animations are played in a loop, which allows them to play for the whole duration of the activity.

The following conditions were used to determine what animation should be played for the Student Actors:

1. If the Student Actor would sit on a chair near a desk with a computer on it, the Student Actor would work on the computer.
2. If the Student Actor would sit on a chair near an empty desk, the Student Actor would either take out a laptop or start writing on a notebook.

3. If the Student Actor would sit on a chair near an empty desk in the first floor, the Student Actor would write on a notebook.
4. If the Student Actor would sit on a chair near and empty desk in the second floor, the Student Actor would use the laptop.
5. If the Student Actor would sit on a chair without a desk near it, the Student Actor would use a tablet.
6. If the Student Actor would sit on the sofa, the Student Actor would sit on the sofa relaxing.
7. If the Student Actor is in a seminar room, the Student Actor would play an animation as if he was discussing something important with the other Actors in the Room.
8. If the Student Actor is in the computer lab near an empty table, the Student Actor would build a computer.
9. If the Student Actor is in a hallway or standing somewhere, the Student Actor would talk to other Actors.
10. If the Student Actor is sitting on a chair in the lecture room, the Student Actor would move its body left to right. This is done because the lecture rooms are the most crowded rooms in the building, meaning that more detailed animations would significantly decrease performance.

Educator Actors appear in the visualisation the moment the Schedule Module sends the data that a subject is about to start. Their animation consists of arm movements as if giving a lecture or explaining something to the Student Actors.

## 3.3 The Speech Bubbles

Because the animations have to be simple with limited actions, it was difficult to express the Student Actor emotion and behaviour (NF5) though animations alone. Thus, speech bubbles (Figure 6) were added to the Student Actors. As Cohn says in his research on speech bubbles: “Speech bubbles are one of the more popular techniques to convey visual imagery and speech” [7]. They help to show that the Student Actors are thinking and learning, in addition to moving around the building.

The speech bubbles appear on the Student Actors that are engaged in an activity other than walking. For example, when they are sitting in a classroom, working at a computer screen or relaxing. During the activity there is an 8% chance every 15 seconds for the speech bubble to appear.

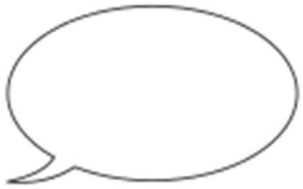

*Figure 6. Example of a speech bubble*[12] *by Valeria Capello.*

When a speech bubble appears, it will slowly increase in size for the 3 seconds. After the 3 seconds the speech bubble begins to fade and will become invisible in a second. When a speech bubble is active for a Student Actor, that Actor cannot spawn an addition speech bubble for 4 seconds.

## 3.4 Furniture Design and Placement

The furniture models for the visualisation were created with a free and open-source program named Blender. When creating the furniture models in Blender, a similar real-life counterpart of the desired furniture piece was measured first in order to create the said furniture piece with approximately the same dimensions. This allowed for precise placement of furniture in different rooms based on the building's architectural blueprint. For example, if a room 1004 was planned to fit 25 people according to the blueprint, 25 chairs had to be put in the room together with other furniture such as tables and teacher desks. During the furniture placement, there were some difficulties to fit the required amount in each room. The difficulty was discussed with the Administrative Manager of the Institute of Computer Science, Piret Orav. The issue was resolved by allowing the author to place as many furniture models as was possible when a classroom in the model did not have enough space to fit the planned amount.

The furniture models themselves have a simple and understandable design behind them, allowing good performance even in the case of hundreds of objects being visible on the screen. In this context, simple means that not a lot of extra details were put to the models, as the extra details will be barely visible in the visualisation (Figure 7). The intent was also to make sure the furniture models would be easily recognized by the viewers of the visualisation.

[12] https://en.wikipedia.org/wiki/Speech_balloon

## 3.5 The Views

Multiple views let the viewers to see the all different sides of the building. The views use an orthographic[13] projection - a projection that allows three-dimensional objects to represented in two dimensions and the lines of the projection are perpendicular to the drawing surface[14]. The orthographic projection was picked because it looked best when testing the visualisation on a Video Wall. Each view position and rotation can be seen in Appendix A. View parameters can be seen in Appendix B.

Furthermore, the views are panning from one direction to the other at a constant loop. This was done to allow all the views to see different areas of the building model much clearly. Figure 7 shows the starting position of all views.

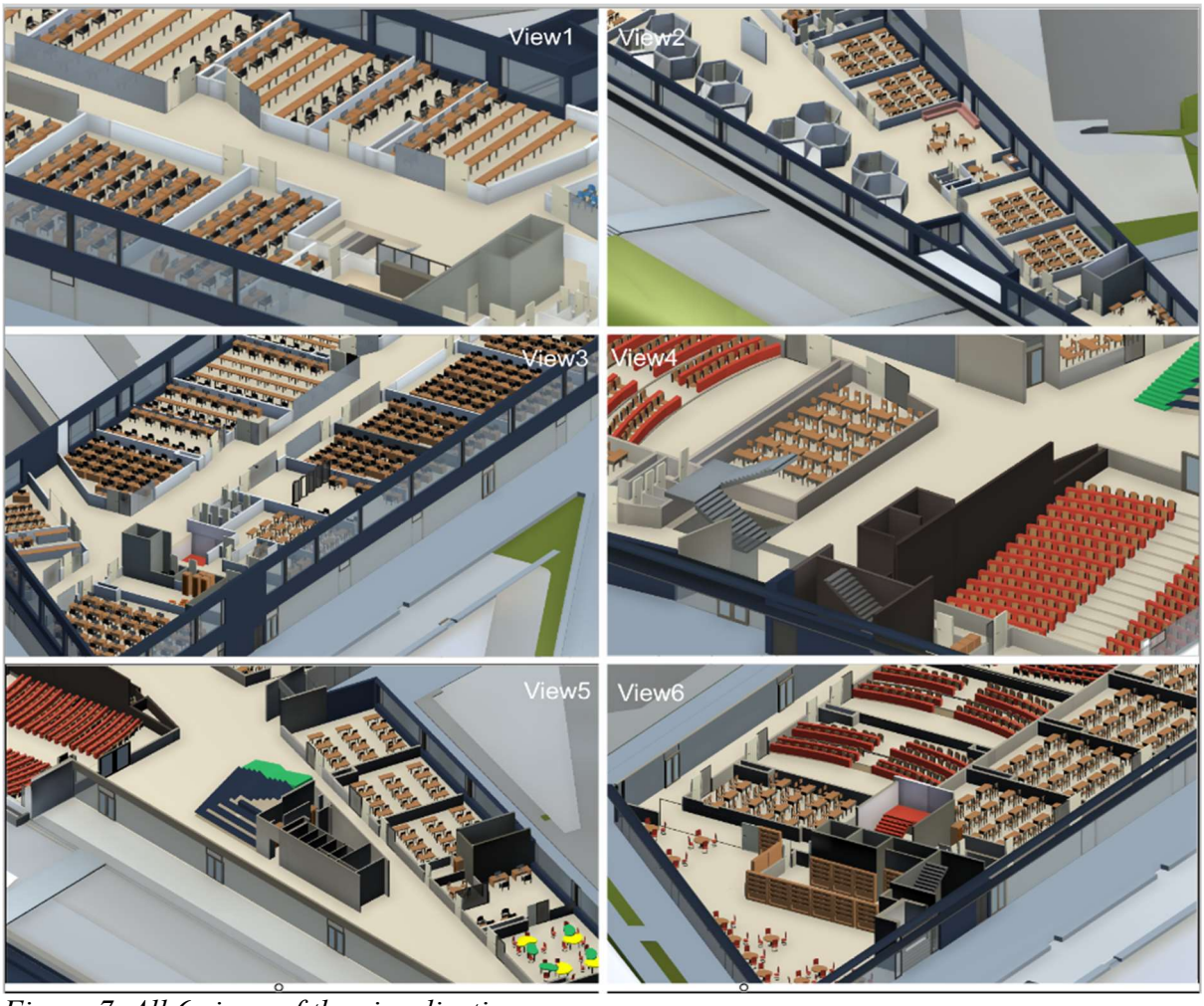


*Figure 7. All 6 views of the visualisation.*

[13] https://en.wikipedia.org/wiki/Orthographic_projection
[14] https://www.merriam-webster.com/dictionary/orthographic%20projection

## 3.6 Wall Design

One major issue when creating this visualisation was figuring out a way to show the Actors properly. Due to the rooms having walls (Figure 8), a way to show the Actors though these walls had to be found.

The first approach was to make the walls semi-transparent, allowing the contents of the room to be semi-visible. However, this approach was avoided due to reasons explained in subchapter 4.3.2.

The second approach was to remove the walls of each room, not including the outside walls altogether. However, that made it difficult to understand the bounds of each room.

The third approach, which is the current solution, was to lower the walls. This way, it is clear to the viewer the bounds of each room, as well as gives the views a good oversight of the Actors. Some walls have not been lowered, like the outer walls. By not lowering the outer walls, it is clear what the bounds of the building are. Lift walls have not been lowered either, as they are “Remove” points that despawn (see the glossary) Actors.

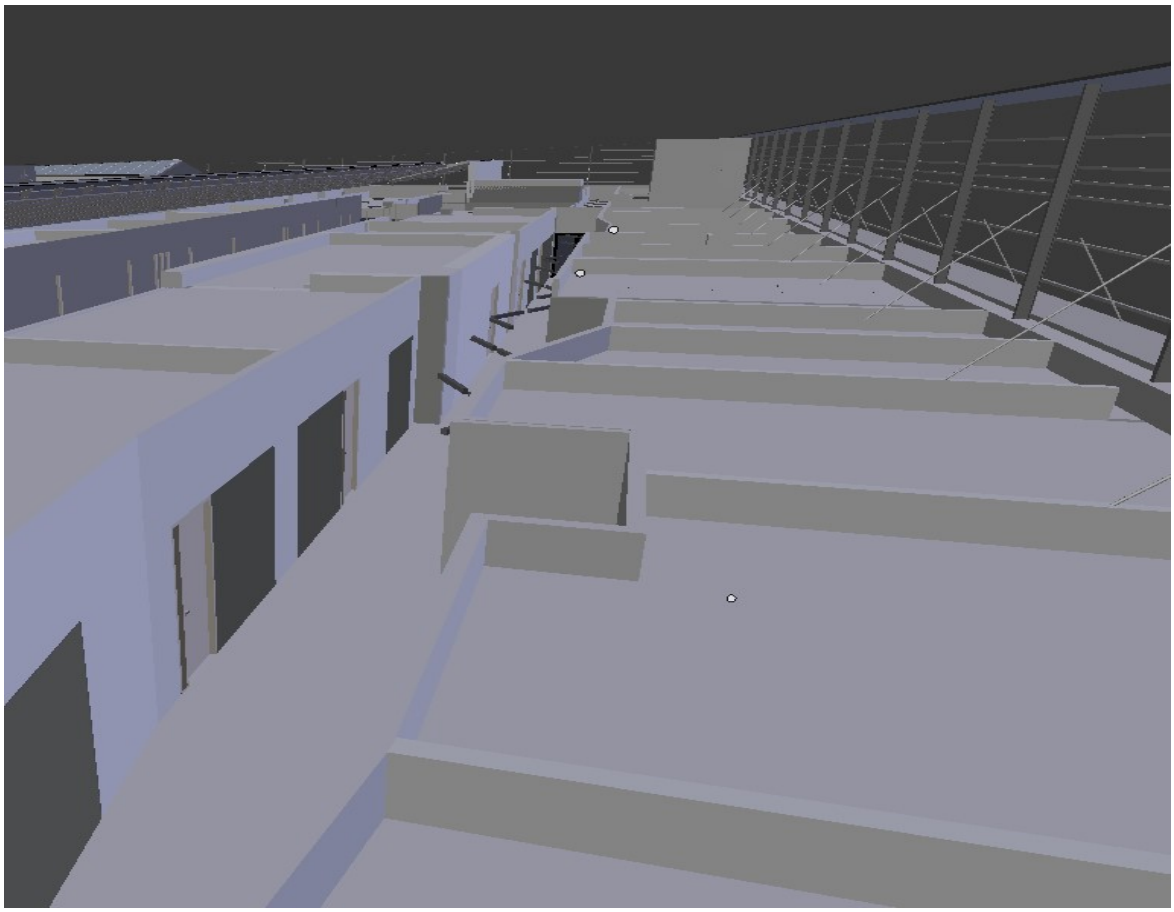

*Figure 8. One of the hallways of the building model with walls blocking the contents of rooms.*

## 3.7 Visualisation Design Optimisation

When designing the visualisation, it was important to make the visualisation pleasant to look at (NF5), but also perform well (NF2). When testing the performance of this visualisation without any optimization, the frames per second requirement has not been met. The following chapters describe optimization techniques that were used to meet the requirement NF2, but also provide information on the design of the visualisation and how it was made pleasant to look at.

### 3.7.2 Lighting

Research on lighting in digital worlds provided Knez and Niedenthal suggests that people feel significantly more pleasant in warm lighting colours [10]. This statements helps satisfy requirement NF5. To achieve this, Unity provides some tools to make the lighting not only pleasant for the viewers of the visualisation, but also be optimized in order to satisfy requirement NF2.

In order to better understand the tools Unity provides when dealing with lighting, first the idea behind lighting in computer graphics should be explained. Lighting in computer graphics consists of a range of techniques and mathematical equations which attempt to simulate real-life light optics, such as where the light bounces and how it reflects from objects. By default, Unity simulates real-life lighting by updating the way light should behave every frame – realtime lighting. However, light rays do not illuminate other objects during realtime lighting. This means that other techniques need to be used to create more realism to scenes[15]. In the case of this thesis, Figure 9 shows what happens when light rays illuminate other objects.

[15] https://unity3d.com/learn/tutorials/topics/graphics/choosing-lighting-technique?playlist=17102

*Figure 9. Lighting with only Realtime Lighting enabled.*

Observing Figure 9, realtime lighting causes walls to have an illumination with cold colours. This is evidenced by the fact that the walls are bluish white[16], which from the research done by Knez and Niedenthal suggests that it is in fact cold lighting [10:6]. In order to avoid cold lighting, Unity provides a method called Global Illumination[17] – a technique to calculate how light is bounced off of surfaces into other surfaces. Also, to satisfy NF2, Global Illumination can be baked, or in other words, pre-calculate the lighting bounces into textures to avoid having to calculate them in the visualisation every frame. Thus, Global Illumination was enabled in the visualisation. To see the effects of Global Illumination, see Figure 10. The beige floors now reflect unto the white walls, giving them warm lighting colours.

[16] https://en.wikipedia.org/wiki/Color_temperature
[17] https://docs.unity3d.com/Manual/GIIntro.html


*Figure 10 Lighting with Global Illumination.*

### 3.7.1 Shadows and Filters

Like realtime lighting, shadows are calculated every frame unless they are pre-calculated and saved to textures. However, based on Andrei Voitenko's thesis, the visualisation will change the direction the light is coming from based on time of day [11], meaning that baked shadows will be stationary when they should be changing their position and shape. However, having shadows enabled substantially decreased the performance as can be seen in Figure 14. Thus, shadows have been disabled altogether in the visualisation.

In order to offset the loss of shadows, a post-processing[18] filter called Ambient Occlusion (AO) has been put on all views of the visualisation. Ambient Occlusion is an effect that approximates how light should be shined on every surface. Figure 11 shows the comparison when Ambient Occlusion is enabled and when it is disabled. In order to better notice the difference, two more figures have been created. Figure 12 shows one of the corners in the view with Ambient Occlusion, while Figure 13 shows the same corner without Ambient Occlusion.

[18] https://docs.unity3d.com/Manual/PostProcessingOverview.html

*Figure 11. A view of the visualisation with and without Ambient Occlusion.*

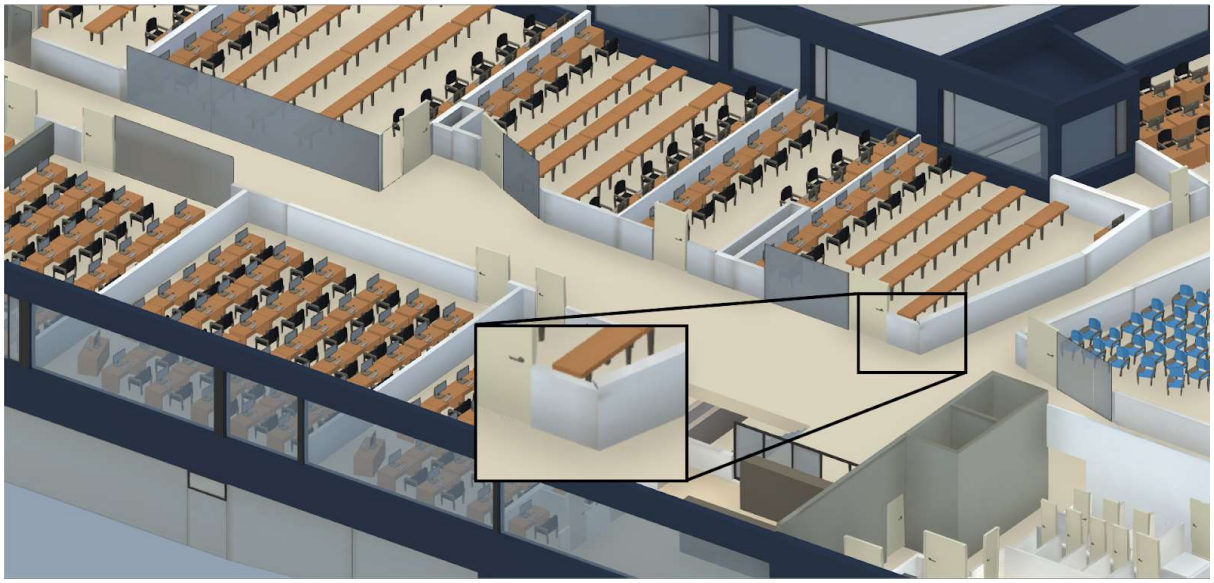
*Figure 12. Showing a corner of the wall with Ambient Occlusion.*

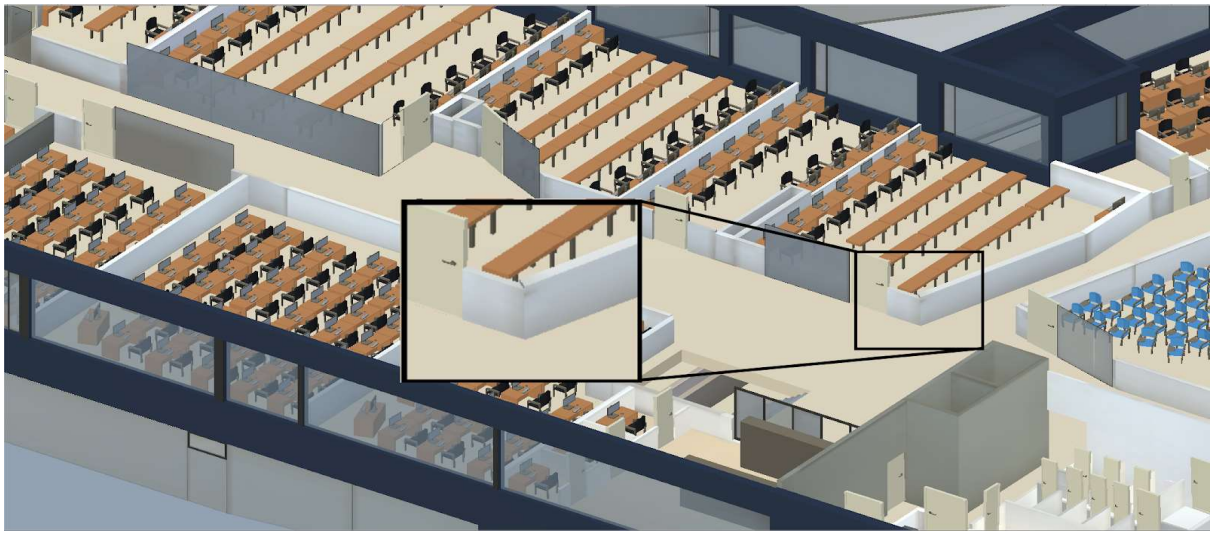
*Figure 13. Showing a corner of the wall without Ambient Occlusion.*

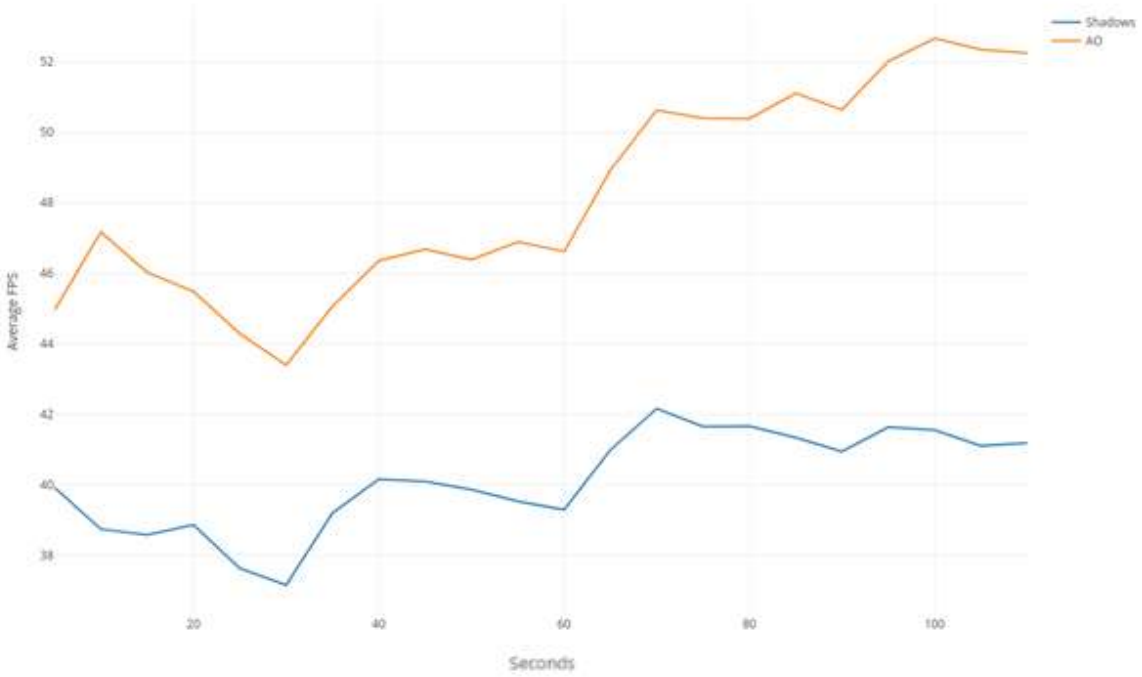


*Figure 14. Average FPS comparison with shadows enabled and with AO enabled.*

# 4. Implementation

During the development of this visualisation, it was important to create good a code structure in order to allow fast and easy changes in the case of bugs and/or wrong outputs (subchapter 4.1). Also, good code structure allows future students and developers who might work on this visualisation to easily read the code and create their own features (chapter 6).

In the code, an algorithm was implemented for logical Actor distribution. The algorithm uses a simulated count of people in a room in order to distribute Actors to their destinations (subchapter 4.2).

With the code architecture and algorithm created and defined, the visualisation was optimized in order to satisfy the non-functional requirements NF1, NF2, NF3 and NF5. The optimization first focused on removing or replacing some objects that were already present in the initial Delta building model. Second, materials (glossary) of created objects were optimized. Lastly, the physics simulation in Unity were disabled as they greatly reduced the performance (subchapter 4.4).

## 4.1 Architecture

From the functional and non-functional requirements, 5 goals have been defined that the code should accomplish:

1. Spawn Student Actors based on the simulated count of people in the rooms.
2. Spawn Educator Actors based on the Schedule Module.
3. Give Actors destinations based on the received simulated data.
4. Actors need to be seated in order to determine what animation needs to be played.
5. Actor should be distributed correctly to rooms using an algorithm (subchapter 4.2).

In order to fulfil these goals, classes have been created to fulfil individual goals. Figure 15 shows a picture of the system architecture - a diagram that shows how the classes interact with each other. Rectangle shapes in the picture depict the classes, while diamond shapes depict the data. The following 8 subchapters give a more detailed overview of what each class does.

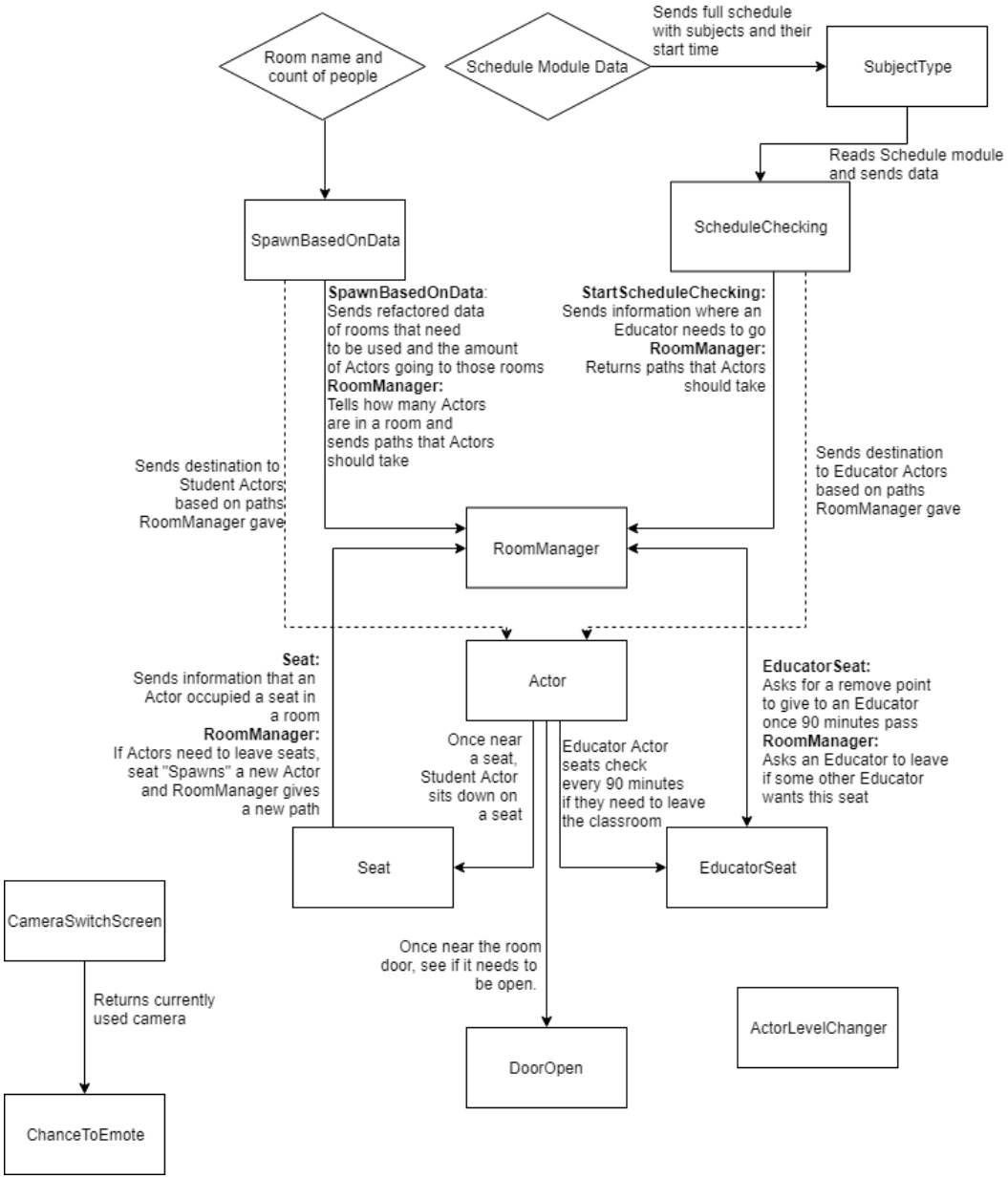


*Figure 15. System architecture used in the project*

### 4.1.1 SpawnBasedOnData

SpawnBasedOnData first starts when the sensor mock data sends information via the method Refactor(int amount, string nameOfRoom). In order to better understand how data is being sent to this method, read *Delta Õppehoone Keskkonna Visualiseerimine* by Andrei Voitenko [11]. We use the argument amount to see how many Actors are going to a room and nameOfRoom to see which room they are going to. The method is called Refactor because it changes the string class to a *GameObject*[19] class – the base entity in all Unity scenes which allows determining the room coordinates.

[19] https://docs.unity3d.com/ScriptReference/GameObject.html

After that, the data is stored. This process can be done multiple times, but after 60 seconds Actors will be distributed across all rooms. The way Actors are distributed can be seen in subchapter 4.2.

### 4.1.2 SubjectType

SubjectType reads all the data from a schedule file. A mock-up of the schedule file can be found in Delta Õppehoone Keskkonna Visualiseerimine by Andrei Voitenko [11].

The class then checks the current day that is set in the computer that is running the visualisation and returns only the necessary subjects to SpawnBasedOnData. We find the current day using the DateTime structure of .NET Framework[20] - an assortment of code and structures the programmers can call without having to write them themselves.

For example, if the schedule has subjects for all five work days, only the current day subjects are required to give Educator Actors their correct destinations. *We assume that the computer will have the correct date, however the visualisation will work even if the date is wrong in the computer, even if wrongly.*

### 4.1.3 ScheduleChecking

Once the proper subjects have been given, the class ScheduleChecking checks if a subject is going to start with a 60 second delay. It does so using a *Coroutine*[21] named CheckEveryMinute() – a function that can suspend its execution for a given amount of time. It can also be used to call the function multiple times, but with a set delay[22].

The moment a subject is going to start, an Educator Actor is spawned and is sent to the room where the subject is happening. In the case that some Educator Actor is already present in that room, the old Educator Actor will leave the room and a new one will take its place.

A delay of 60 seconds is used for the following reasons:

1. It is assumed that subjects start by the minute and the current seconds do not matter. For example, if a subject starts at 16:15, it does not matter that the current time with seconds is 16:15:08.
2. It is assumed that only one educator will educate students in a single room.
3. Checking in less than a minute will result in unnecessary statements because of reason 2. For example, assume that some subject starts at 14:15:00. Also assume that

[20] https://lifehacker.com/5791578/what-is-the-net-framework-and-why-do-i-need-it
[21] https://docs.unity3d.com/ScriptReference/Coroutine.html
[22] https://docs.unity3d.com/ScriptReference/WaitForSeconds.html

CheckEveryMinute() checks if a subject is going to start every 30 seconds and starts checking at 14:14:30. The first time CheckEveryMinute() will send an Educator Actor to the room is at 14:15:00. The second time it will send a different Educator Actor is at 14:15:30. This is faulty behaviour, as it is assumed that only one educator is required to educate the students. This can be resolved by checking if an Educator Actor is already going to the given room or by setting the delay to 60 seconds. The second solution does not require an extra statement; thus it is the current solution.

4. Checking less often than once a minute will result in some minutes not being checked. For example, assume two subjects. The first starts at 14:16 and the second one starts at 14:17. Also, assume that the delay is 90 seconds and CheckEveryMinute() starts at 14:15:00. The first Educator Actor for the first subject will spawn at 14:16:30. however the next check will be at 14:18, meaning that the method missed a subject that starts at 14:17.

### 4.1.4 Actor

Actor class is used to set destinations for the Actors. It does so once ScheduleChecking or SpawnBasedOnData gives a destination via a method called SetDestination(SeatPath seatPath). The SeatPath argument is a class that holds the room the Actor is going to go to, the seat it will sit on and the pre-calculated path it will be using (subchapter 4.1.5).

In SetDestination(SeatPath seatpath), Actor will set its destination using a method called SetDestination() found in *NavMeshAgent*[23] – a component that allows objects in Unity to navigate around the 3D space of the visualisation and perform tasks such as reaching their set destination. However, an object with a *NavMeshAgent* component cannot walk on any object, but only set objects with the *NavMesh*[24] property. For the purpose of simplicity, we can assume that the *NavMesh* property is a setting that can be enabled on any object.

Once the destination has been set, the Actor will move to its target room and target seat. If a room has a door, it will open the room door once being near it. Note that it will not open the door if it is already open. Once near its seat, the Actor will sit on it (subchapter 4.1.5).

### 4.1.5 RoomManager

RoomManager records seat occupation of every room in the visualisation and the amount of Actors going to a given room. RoomManager also stores all the positions of every room, Spawn

[23] https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html
[24] https://docs.unity3d.com/530/Documentation/ScriptReference/NavMesh.html

point and Remove point. It updates these values when certain methods in StartScheduleChecking, SpawnBasedOnData, Seat or EducatorSeat require a position of a room or seat from RoomManager.

Because the RoomManager stores the positions of rooms and seats, it also pre-calculates paths that Agents will use to reach their destination. It does so using the *NavMesh* class, which has a method called CalculatePath()[25]. For the purpose of simplicity, we can assume that the arguments of CalculatePath() are a starting position and the target position. Paths are pre-calculated in the following way:

1. From each room to every other room.
2. From every room to every "Remove" point.
3. From every "Spawn" point to every other room.
4. From every "Spawn" point to every "Remove point.

Pre-calculated paths allow Actors to find a path to their destination faster, as required by NF3. Subchapter 5.1 shows the difference when not using pre-calculated paths and having Actors to find their path without any pre-calculation.

### 4.1.6 Seat and EducatorSeat

The Seat class is responsible for animating the sitting animation once an Actor is near its seat. Once an Actor is near the seat, that Actor is removed from the scene and is replaced by enabling a similar looking object, *StudentActorAnimation,* in the scene. Finding precise coordinates for seating the original Actor proved to be difficult, as its position and rotation are often unpredictable. For this reason, it was easier to delete the original Actor and replace it with a similar looking object.

Once an Actor was determined to leave its seat because of sensor data (subchapter 4.2), the Seat class will spawn a new Actor near the seat and send data to RoomManager that a new Actor has spawned and is going to a certain destination given by sensor data.

The EducatorSeat class behaves in almost the exact same way as the Seat class. However, EducatorSeat checks every 90 minutes if it is time for an Educator Actor to leave his seat and be despawned. Currently it is assumed that each subject will have a duration of 90 minutes. This is necessary because the current mock-up schedule has the starting time of each subject, not the ending time. This means that if an EducatorActor does not get a new destination from the ScheduleChecking class, it will stay in its room until the program closes.

[25] https://docs.unity3d.com/530/Documentation/ScriptReference/NavMesh.CalculatePath.html

### 4.1.7 DoorOpen

The DoorOpen class is used to open and close the door when an Actor comes close. If the door is closed and an Actor comes close, the door will be opened. If the door is open, but the expected amount of Actors is less than 3, then the door will be closed. The reason for the number to be 3 or less is because it was observed that some Actors would find a way to their seat without going through their given door. This solution however has not fixed the issue with doors not being closed all the time and the author suggests future students and developers of the visualisation to find a way to extend the proximity detection to all doors, not just one.

### 4.1.8 CameraSwichScreen

This class has a method called CameraSwitch() that switches between the different views every five seconds. Five seconds was chosen as it gives enough time for the viewer to see what is happening in the current view. Views can be manually switched by first pressing space bar and then the 1 2 3 4 5 6 keys. Pressing space bar again will enable the automatic switching again.

### 4.1.9 ChanceToEmote

This class is separate from most as it is enabled once *StudentActorAnimation* is enabled in the scene. The class itself consists of methods that spawn speech bubbles (subchapter 3.3) and play animations, such as typing or notebook writing (subchapter 3.2). Speech bubbles look at the enabled view that is returned from the method getActiveCamera() in CameraSwitchScreen class.

### 4.1.10 ActorLevelChanger

This class allows the Actors to switch layers when they reach a certain elevation in the visualisation. All cameras can only see objects that have a set layer. For example, a camera that can only see objects with the layer *FirstFloor* will not see objects with the layer *SecondFloor*. This is important because if views have to render all Actors on both floors, the performance will slightly decrease, as seen in *Figure 16*.

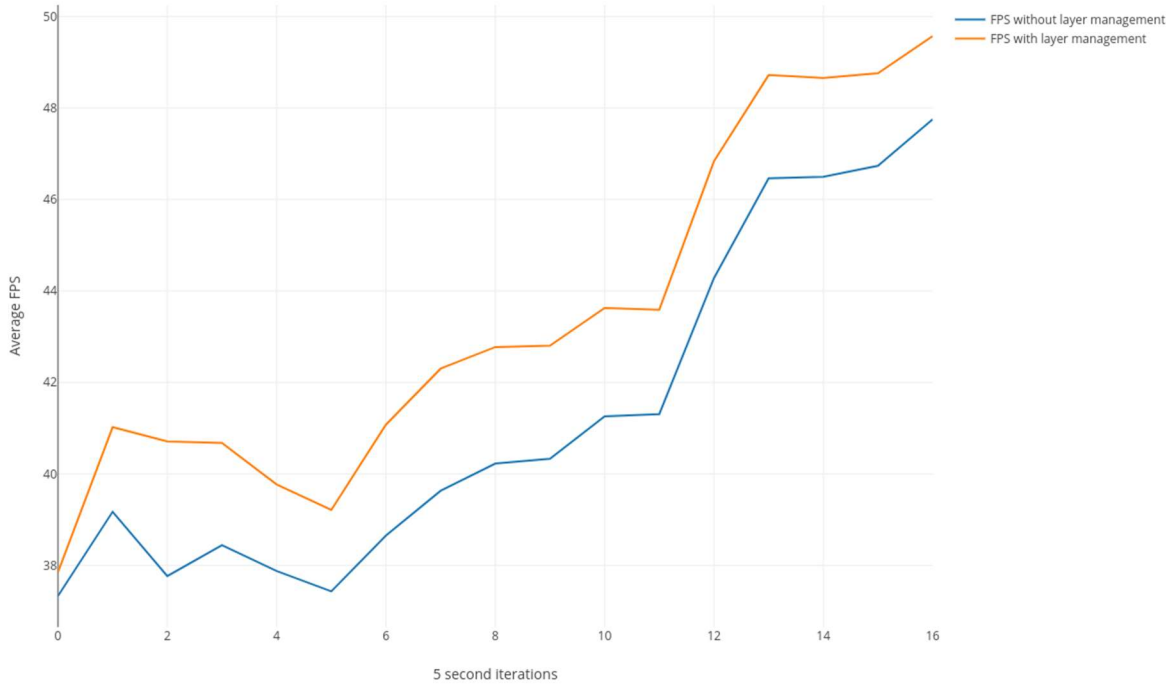


*Figure 16. Performance change when Actors have separate layers.*

## 4.2 Pathfinding

The Actors in the visualisation are required to have the ability to navigate around the model of the building, change their destination in any given time and find their path in the required amount of time (NF3).

The next 2 subchapters provide information on how the implemented algorithm distributes Actors and how the *NavMeshAgent* component is used to improve pathfinding and navigation. The subchapters will also explain what optimization techniques were used in order to satisfy the performance requirement (NF3). After those 2 subchapters, the pathfinding optimization methods will be explained to help satisfy requirements NF1.

### 4.2.1 The Preparation Stage

A class by the name of SpawnBasedOnData determines what rooms require an increase or decrease of Actors. The method Refactor() in class SpawnBasedOnData is called when sensor data is sending new data to the visualisation. This data includes important values such as the room number and the amount of Actors that are going to the said room. For the purpose of simplicity, let the room be called *Room* and the amount of Actors going to the room be called *ActorCount*.

With the data received, SpawnBasedOnData first checks the amount of Actors already present in the given room. After that calculation, it determines whether the new *ActorCount* is bigger than the old value or higher.

- If the value is higher, that means the room will need an addition of actors. This will be recorded in a dictionary[26] named *toAdd*, where the key is the room number and the value is a positive amount of Actors needed to be added to satisfy the data.
  For example, assume room 101 exists and there are 10 Actors in the said room. Sensor mock data sends information that room 101 now has 15 Actors. SpawnBasedOnData will now assume that room 101 requires an addition of +5 actors.
- If the value is lower, that means the room will need a decrease of actors. This will be recorded in a dictionary named *toRemove*, where the key is the room number and the value is a negative amount of Actors needed to be removed to satisfy the data.
  For example, assume room 102 exists and there are 10 Actors in the said room. Sensor mock data sends information that room 102 now has 5 Actors. SpawnBasedOnData will now assume that room 102 requires a decrease of -5 actors.

Note that the preparation stage can be done using a single dictionary. However, for the purpose of simplicity and better understanding, two dictionaries have been chosen instead.

### 4.2.2 The Actor Distribution

With the Dictionaries prepared, SpawnBasedOnData will then distribute the Actors. This is done in the following way:

- Determine how many Actors from the *toRemove* dictionary can go to the *toAdd* rooms. Using the above example, Actors from 102 that have left can go to room 101.
- In the case that all the Actors from the *toRemove* dictionary have been used, but *toAdd* rooms still require Actors, spawn new Actors outside the building and set their destination to the rooms from *toAdd* until the data is satisfied. For example, assume room 101 needs 15 actors and room 102 has a decrease of 5 actors. The 5 Actors from room 102 will go to room 101 and the other 10 Actors needed to satisfy the data will be spawned outside the building with their destination being 101.
- In the case that all the rooms in the *toAdd* dictionary are filled to satisfy the data, but toRemove still has Actors that need to leave the room, make those Actors go to "Remove" points that will despawn the Actors. For example, assume room 101 needs 5 actors and room 102 has a decrease of 13 actors. The 5 Actors from room

[26] https://unity3d.com/learn/tutorials/modules/intermediate/scripting/lists-and-dictionaries

102 will go to room 101 and the other 8 will go to special “Remove” points to get despawned.

- In the case that all the rooms in *toAdd* are filled to satisfy the data and *toRemove* has no Actors that need to leave, do nothing as the distribution was even.

### 4.2.3 Spawn Points

Actors spawn on special areas called spawn points. There are in total 10 spawn points. They are placed around the building model so that none of the 6 views can see Actors being spawned. Reason for having more than one spawn point is to allow Actors to move in different paths from other Actors. Also, multiple spawn points better distribute the initial Actor spawn and allow Actors to take different paths. *Figure 17* shows the location of all spawn points.

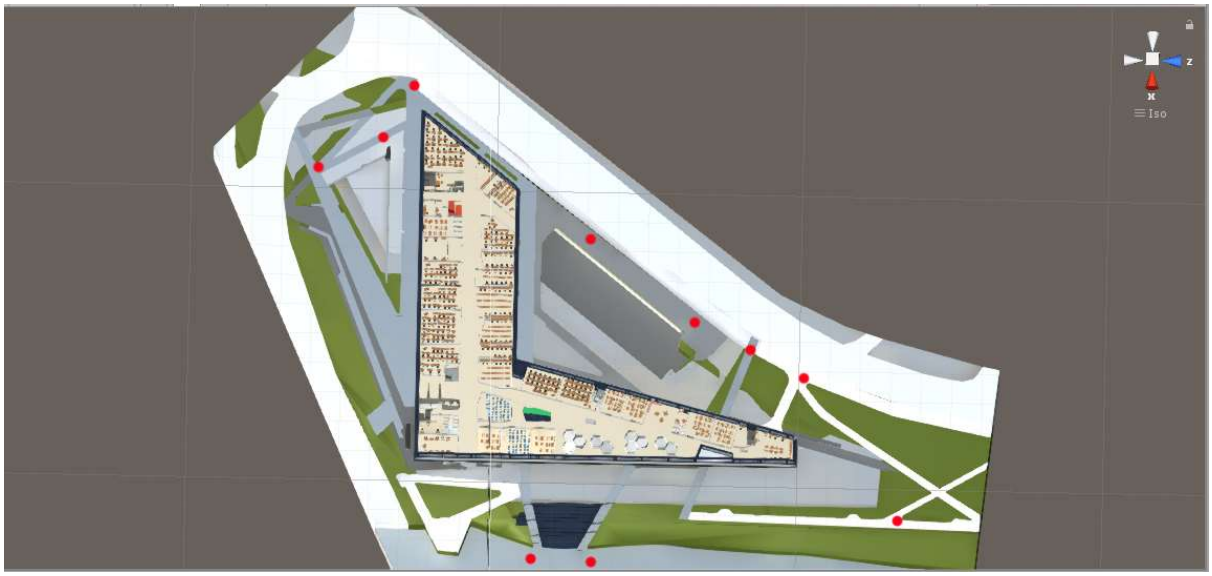

*Figure 17. A top down view of all spawn points. Spawn points are represented as red dots.*

### 4.2.4 Pathfinding Optimizations

The *NavMeshAgent* provides many different solutions to solve the problem of having a high Actor count. The following settings and methods of the Unity *NavMeshAgent* component have been chosen for better pathfinding in order to satisfy NF3:

1. The *ObstacleAvoidance*[27] radius parameter is changed to 0.1. By changing the *Obstacle Avoidance* radius, the Actors are less likely to create obstacles for each other.
2. Change the Quality in the *Obstacle Avoidance* parameter to Low Quality. This allows the Actors to not be as precise when avoiding obstacles. *Note that changing*

[27] https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent-obstacleAvoidanceType.html

*it to None will improve pathfinding and performance, but it was observed that the Actors would ignore each other and make their walking feel unnatural.*

3. Turn off *Auto Repath* in the *Path Finding* parameter. In the case if a path has been a found, but Actors have blocked the way, an Actor with *Auto Repath* enabled will try to find a different path. This will result in unnecessary calculations and increase the time it takes for Actors to find their destination.
4. Using *OffMeshLinks*[28] - shortcuts that Actors can use to either create a faster path to their destination or create a "bridge" from one point to another. This is important because the Actors can only walk on predefined areas and in the case that an area is not defined, they will avoid it. While testing with the above mentioned settings changed to worsen performance, *OffMeshLinks* were still placed in order for the Actors to have a guaranteed path to their destination.

Even though NF3 can still be satisfied without these settings, it was observed that without them Actors would get stuck in hallways and corridors trying to avoid each other. Thus the above mentioned methods and settings were enabled.

Furthermore, as already explained in subchapter 4.1.5, paths to certain destinations are pre-calculated so that Actors do not have to calculate most of their paths themselves. Without pre-calculated paths, the time it took Actors to find a path was substantially higher, as seen in Figure 18.

[28] https://docs.unity3d.com/Manual/class-OffMeshLink.html

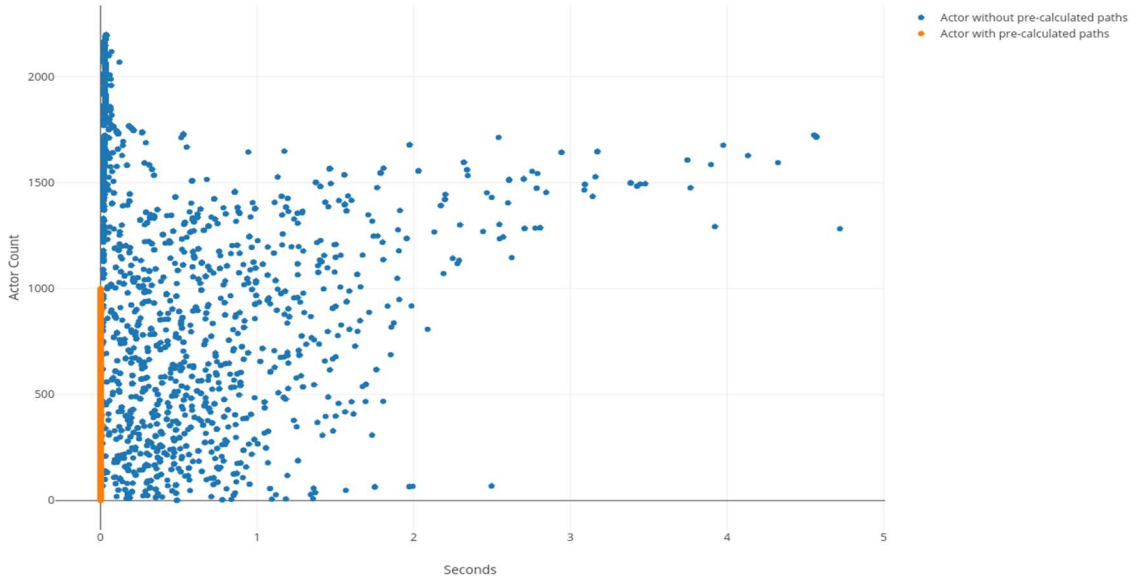


*Figure 18. Difference in path finding time with and without pre-calculated paths. Orange values are less than 0.1 seconds.*

## 4.3 Model Optimizations

Other than keeping the furniture models simple and understandable (subchapter 3.4), it was important that they would also be optimized. This means that the vertex count of the created object should be low. This is explained further in subchapter 4.3.1.

Each furniture model needed to be coloured in order for the visualisation to look pleasant (NF5). This was done in Unity using materials. subchapter 4.3.2 explains the material optimizations that were made in the visualisation.

Some objects have not been optimized, but deleted instead in order for the visualisation to perform well. These objects can be something like lamps that will not be visible in the visualisation (Figure 19). More about deleted objects can be found in chapter 4.3.3.

### 4.3.1 Furniture Models

When creating the objects, a non-functional requirement NF4 helped set the goals for optimized objects. Because each view renders a lot of objects, is important that each created object has a small amount of vertices, or else NF4 cannot be satisfied.

For the purpose of this thesis, a non-functional recommendation has been created by the author, which recommends each created object to be 400 vertices or less. By creating this side recommendation, part of NF4 has also been satisfied (chapter 5.3).

The recommendation also extends to doors that were initially placed in the Delta building model. The number of doors around the visualisation is high (over 100), meaning that if they are not optimized, they will take up a lot more resources than they should. For this purpose, an optimized version of the door has been created, which reduced the vertex count of the door by approximately 60%. This method was done for every already placed model that was decided not be deleted (chapter 4.3.3).

### 4.3.2 Material Optimizations

Materials play a big role in furniture models, because without them, the furniture models would not be rendered. Also, the default material for every object is white, meaning that it would be hard for the viewer to distinguish each furniture model. Thus the different furniture models required different materials.

In the case of this thesis, unoptimized material objects will drastically ruin the performance because of the high furniture model count. In order to avoid that, multiple material optimization techniques have been used to make sure the visualisation satisfies requirement NF2.

A lot of objects in the visualisation share the same mesh (see the glossary), as most objects are copies of their original furniture model. As not to render these meshes individually, a technique can be used for rendering multiple copies of the same mesh in a scene at once. This technique is called GPU instancing[29] - a technique that can be used on all materials created in Unity.

Other optimization techniques include limiting transparent materials in the visualisation. A semi-transparent material is a see-through material, commonly placed on objects such as windows. Semi-transparent materials are one of the main reasons pixels are being drawn multiple times. [30]

### 4.3.3 Object Model Removal

The building model had a lot of objects that would obstruct the views of the visualisation, thus it was important to remove only the objects that would hinder the view of the visualisation. Figure 19 shows the second floor of the Delta building model without its ceiling. After careful observation, it was decided that any object that was a part of the ceiling should be removed from the Delta building model, as they would obstruct the view to the

[29] https://docs.unity3d.com/Manual/GPUInstancing.html
[30] https://unity3d.com/learn/tutorials/temas/performance-optimization/optimizing-graphics-rendering-unity-games

Actors. These objects included lamps, ceiling supports or other objects that were present on the ceiling of any floor. *Another solution was to make these objects semi-transparent, but as already explained in subchapter 4.3.2, this method was avoided.*

In the same Figure 19, we can notice support beams present in the initial Delta building model. These were removed as well due to their amount (2240). Another reason for their removal to give the visualisation a cleaner feel, allowing the Actors to walk freely without avoiding these supports. Figure 20 shows the second floor with those objects removed.

Other removed objects include objects of a very high quantity; such as bars outside the building model that can be seen in Figure 21. An argument can be made to not remove them, but instead combine the meshes into one big mesh, with a technique that Unity provides called *CombineMeshes*[31]. However, because bars have different materials, combining them would not increase performance[32]. Also, removing these bars allows a better view of the second floor, making the visualisation that much cleaner.

Last objects that were removed were the window frames around the glass windows that can be seen in Figure 21. They were removed in order to allow windows to have a much clearer view of the Actors around the visualisation.

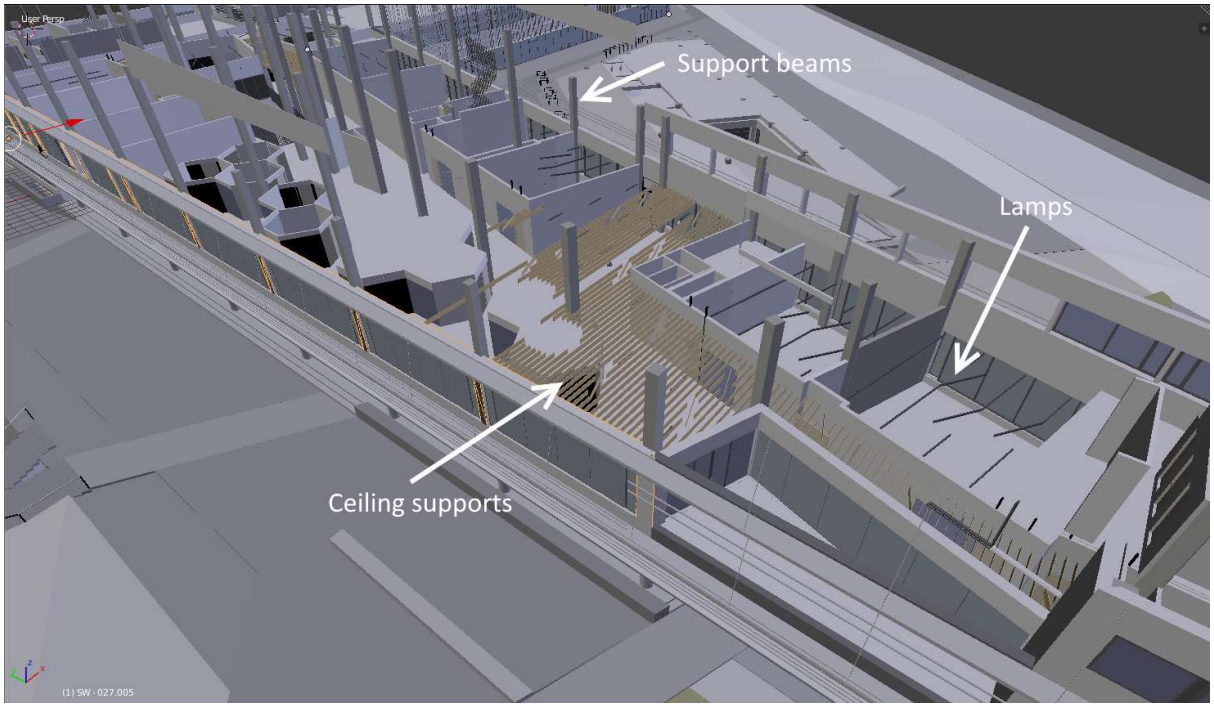


*Figure 19. A screenshot taken in Blender with visible support beams, lamps and ceiling supports.*

[31] https://docs.unity3d.com/540/Documentation/ScriptReference/Mesh.CombineMeshes.html
[32] https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html

*Figure 20. A screenshot taken in Unity of the second story with the objects removed.*

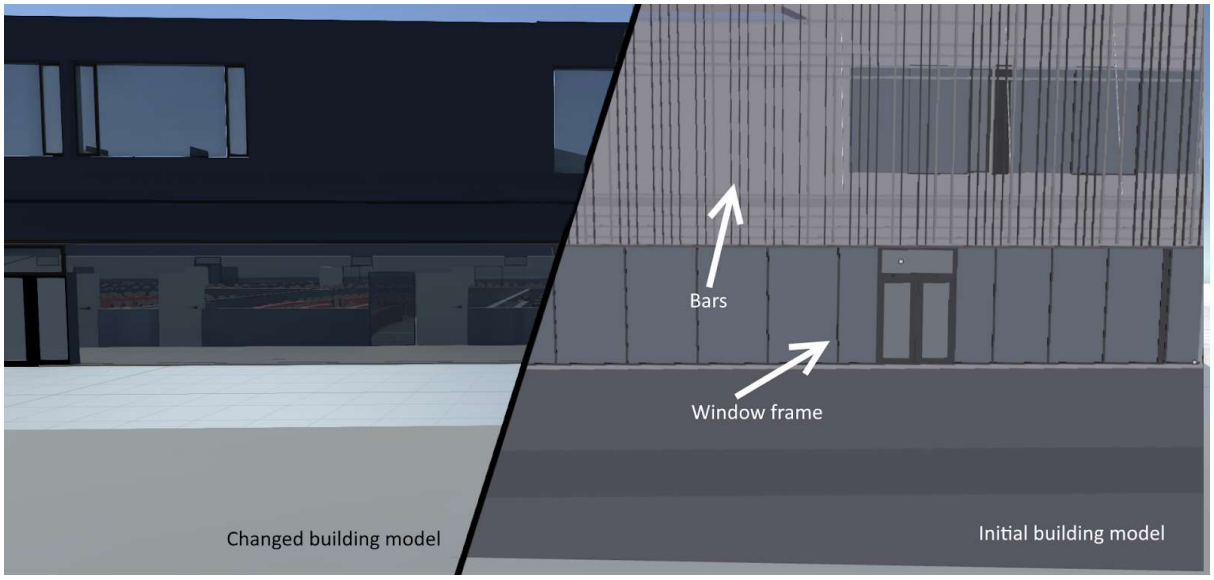


*Figure 21. A screenshot comparing the building model with and without bars and window frames.*

## 4.4 Game Engine Physics

Physics in games or simulations are simulated done by introducing the laws of psychics into the said simulation or game. This allows objects, like falling rocks, to be visualised in a realistic matter by adding physics simulations to it. In Unity, physics are enabled on an object once a *RigidBody* component is placed on it. With the *RigidBody* component, the object will respond to gravity created by the game engine.

However, during development of the visualisation, it was observed that when enabling physics in the visualisation, the performance would drastically decrease and make NF2 impossible to satisfy. The technique to test performance with physics was different than the

techniques done in chapter 5. This technique involves putting the *RigidBody* component on all Actors. Furthermore, it does not show frames per second of the visualisation, but instead show the amount of milliseconds it took to reach the next frame. This is done because when only calculating the FPS, the test would only show from 0-2 FPS, which is not descriptive enough. The following Figure 22 shows the difference between enabled and disabled physics.

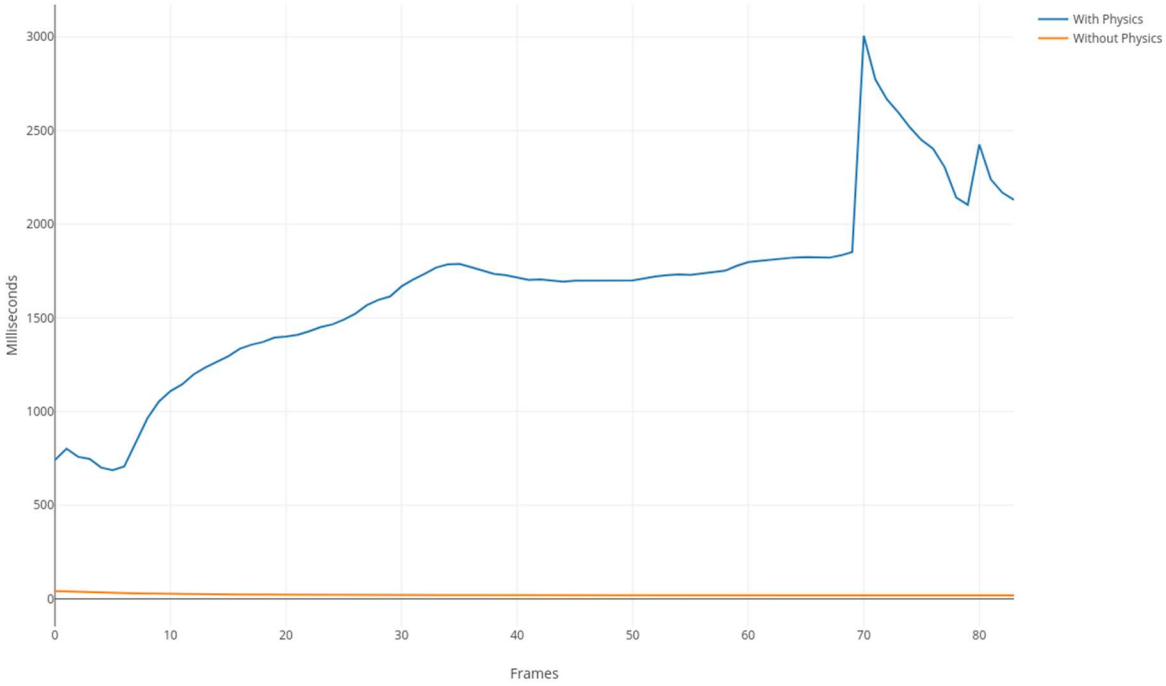


*Figure 22. Milliseconds per frame with and without physics.*

# 5. Testing

The following chapters describe the verification of the non-functional requirements (subchapter 2.2) and the methods used for that. It will also explain some of the results gathered from the tests. To see what hardware was used for performance testing, see NF2.

## 5.1 The Actor Count and Performance (NF1, NF2)

In order to verify both non-functional requirements, a basic frame per second counter is required to determine whether the visualisation is running at 30 frames per second or more. Because this visualisation consists of multiple views (NF4), each view needed to be checked whether it can handle 2010 Actors (NF1) at 30 FPS (NF2).

Unity provides multiple features to calculate FPS, such as the Statistics[33] screen or the Profiler[34] window, however these features were found to be inaccurate[35]. Instead, a script was used to determine FPS. The script for calculating FPS can be found on the Unity Wiki[36], created by Dave Hampson.

Once 2010 Actors appeared in the visualisation, every 5 seconds an average FPS count was recorded. If the FPS count was at any point smaller than 31, the test would throw an error and the test would stop. This method was determined by the author to be much clearer than having to record FPS every frame. Figure 23 shows the average FPS of every. Also to note, the test did not throw an error, meaning the FPS was never below 31. Thus, NF1 was satisfied because the visualisation supports 2010 Actors and NF2 because the FPS was above 30.

[33] https://docs.unity3d.com/Manual/RenderingStatistics.html
[34] https://docs.unity3d.com/Manual/ProfilerWindow.html
[35] https://answers.unity.com/questions/33369/profiler-fps-vs-stats-fps-vs-timedeltatime.html
[36] http://wiki.unity3d.com/index.php?title=FramesPerSecond

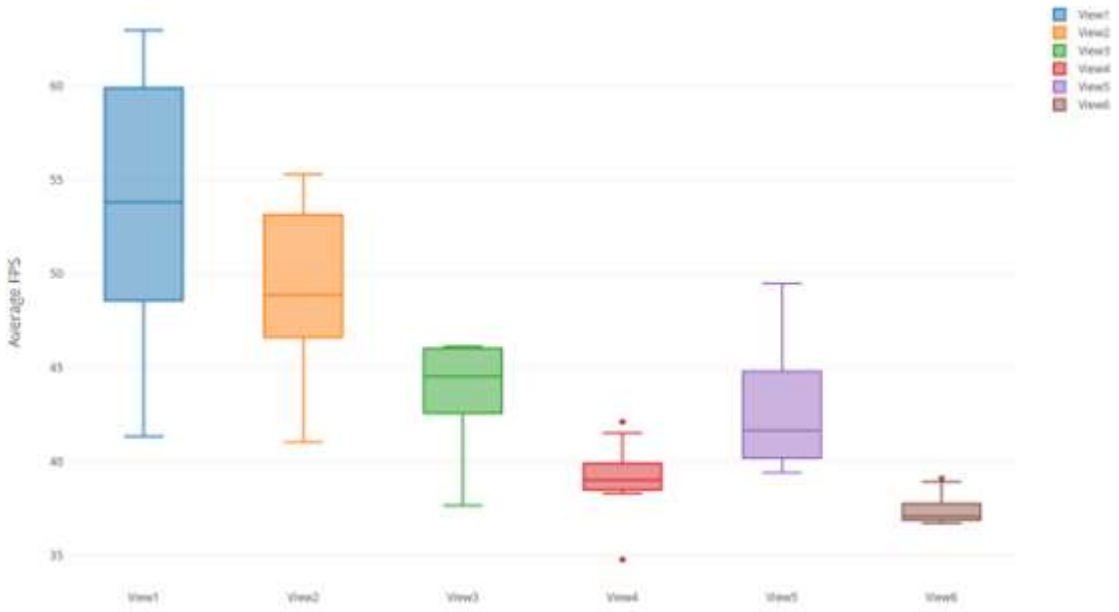


*Figure 23. The average FPS measured every 5 seconds for every view.*

It was observed that some views render more objects and Actors than the other views, hence the big difference in FPS. Views 4,5 and 6 all visualise the first floor, where a large influx of Actors first appears, as seen in Figure 24. Furthermore, views 4 and 6 visualise the lecture halls, which have the highest amount of Actors than any room.

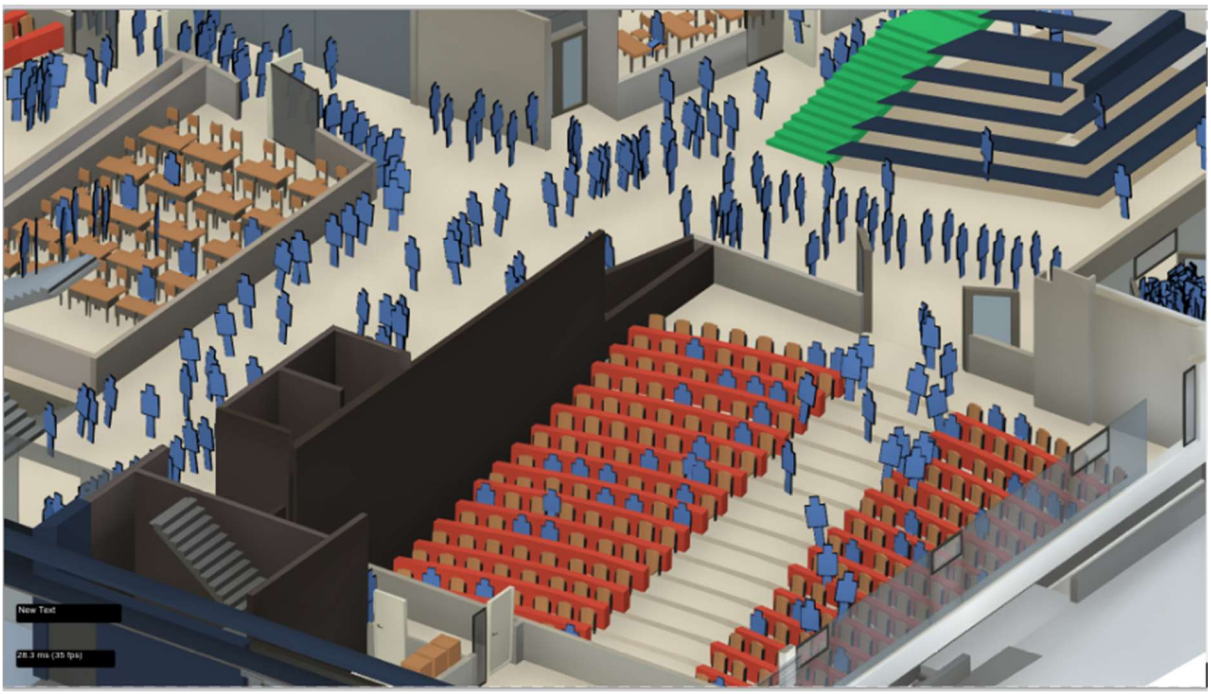


*Figure 24. Showing a large influx of Actors on the first floor.*

## 5.2 Actor Path Finding Time (NF2)

In order to determine if the Actors find their destination in 2 seconds or less, first the time an Actor was given a new destination was recorded. This was done in Unity using the class *Time* provided by the Unity game engine. The class *Time* has a property called *realtimeSinceStartup*, which calculates the amount of seconds passed since the visualisation started.

After that, the amount of time it took the said Actor to find a path to his new destination was recorded. This can be accomplished using the *NavMeshAgent component*, as they have a property called *hasPath* – a property that shows if an Actor has a found a path to his new destination. Every 10 milliseconds it is checked whether the Actor has found a path to his new destination. Once a path has been found, the parameter *realtimeSinceStartup* was called again and the difference between the two values was found and recorded in a separate file. This was done for all the 2010 Actors. The 2010 Actors appeared in the visualisation at the same time.

Figure 25 shows the time it took an Actor to calculate a path from his starting destination to his desired destination. However, it was observed that one took a significantly longer time to find his path (0.012 seconds). When trying to find a reason for this behaviour, the author did not find anything wrong with the starting destination, the path the Actor took or the desired destination. Thus, the test was done 2 more times and can be seen in Figure 26.

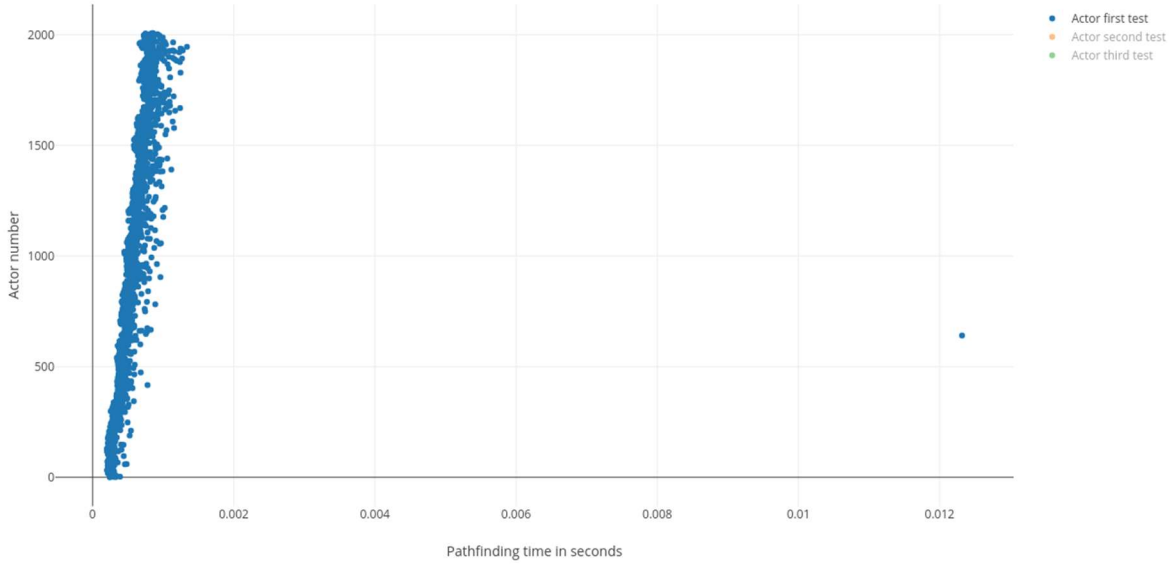


*Figure 25. A scatter plot showing the amount of seconds it took every Actor to find a path to his new destination.*

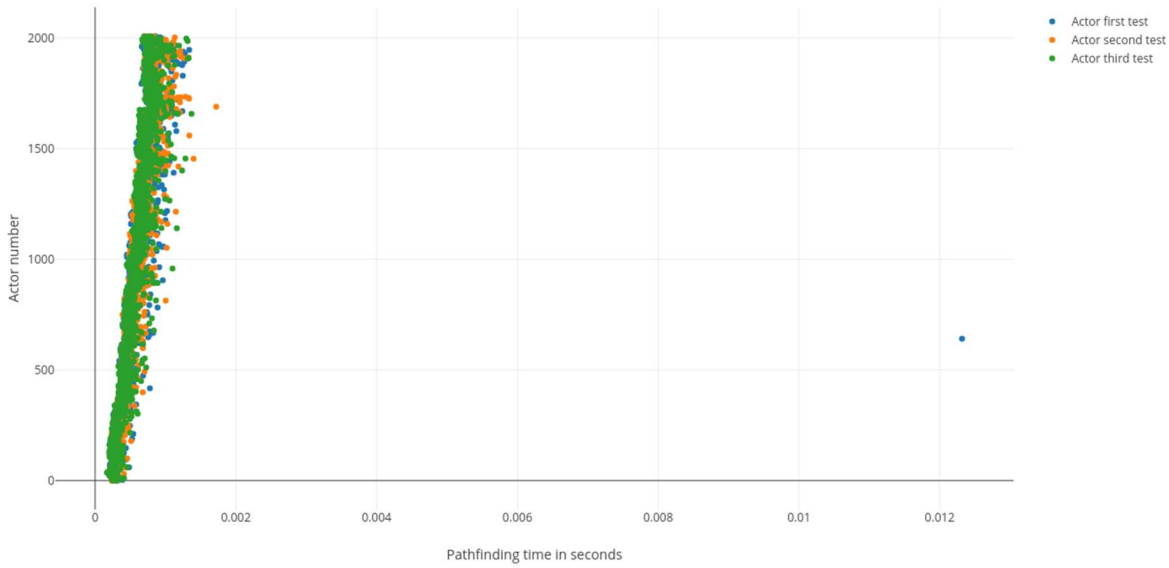


*Figure 26. A scatter plot of all 3 tests and the time it took Actors to find a path to their desired destination.*

As seen by Figure 26, this behaviour was not recreated. From the following tests, the author assumed that the difference in time (0.012 seconds) is not significant enough to mark it as an issue for the visualisation that needed attention.

## 5.3 Viewpoints (NF4)

In order to verify that all views have a vertex count below 3 million, the Statistic[37] screen in the Game View[38] was used. The Game View is a view in Unity that shows how all the views render the visualisation. Because the views pan from one location to the next, it proved difficult to find the best technique to prove that NF4 has been satisfied. The author decided that Figure 26 will show the maximum and minimum vertex count values once 2010 Actors spawn and are distributed in the visualisation.

[37] https://docs.unity3d.com/Manual/RenderingStatistics.html
[38] https://docs.unity3d.com/Manual/GameView.html

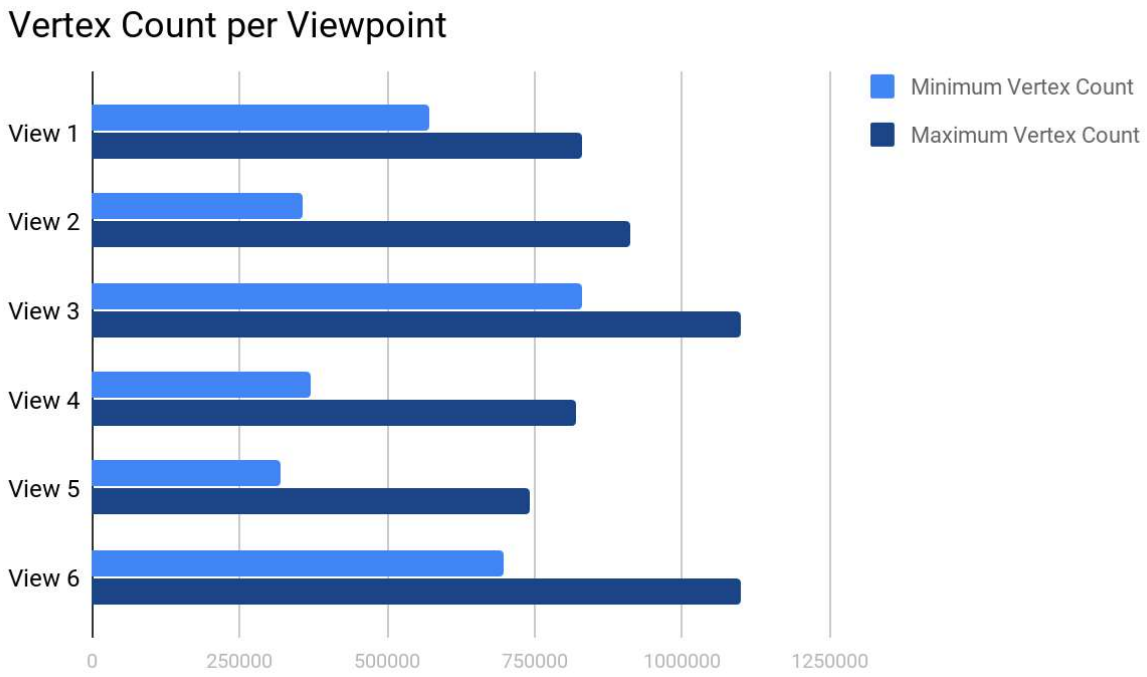


*Figure 27. Vertex count of each viewpoint with and without actors.*

Figure 27 shows that the vertex count for each view has not exceeded 3 million, thus satisfying the requirement.

## 5.4 Design of the Visualisation (NF5)

In order to potentially provide insight on how the visualisation can improve work atmosphere, a group of 5 people were questioned to give their feedback on the visualisation. Three of those people were prospective students from the University of Tartu. The rest of the group wanted to see what can be accomplished with the knowledge gathered from the Computer Science curriculum.

The visualisation was shown on a Video Wall in the University of Tartu Library, as can be seen in Figure 28, with a description that the goal of the visualisation is to simulate student and educator behaviour using real-time sensor and schedule data. The group was given a questionnaire with 3 questions. One of those questions was to grade the visualisation on a 4-point scale. The other two questions were given to allow the group to write their thoughts on what the visualisation did good and what the visualisation could do better.

### 5.4.1 Results

Five out five people have given the visualisation a maximum grade, potentially proving that the visualisation could improve work atmosphere. When asked about the good aspects of the visualisation, 3 out of 5 people said that the visualisation is easy to follow and is

understandable. The rest of the group said that they enjoyed it, but did not give a detailed answer.

The following things have been suggested by the group that can be improved in the visualisation:

1. Improve FPS to 60.
2. Improve the Student Actor walking animation, as well as add textures to objects.
3. Make it easier to distinguish when the visualisation is showing the first floor and when it is showing the second.

### 5.4.2 Conclusion

The author decided that not enough information has been given to make a verdict on the topic of this visualisation improving work atmosphere. However, the visualisation was enjoyable and understandable to the viewers, which was part of the non-functional requirement.

*Figure 28. Visualisation that was shown to the viewers.*

# 6. Future Development

The Delta building will be finished in 2020, providing enough time for future students and developers to continue working and improving this visualisation. The following is a list of author suggested features that could potentially be interesting to implement for future developers:

1. Randomize the Actor appearance. Currently Student Actors and Educator Actors are always the same colour. Adding textures and objects (such as clothing) to Actors would distinguish each individual Actor. This would give Actors more personality, potentially improving viewer enjoyment.
2. Create more animations that would improve the look of the visualisation.
3. Create textures to make the building and the objects potentially prettier.
4. Have people participate on the study of work atmosphere and see if they think the visualisation can improve the said atmosphere.
5. Improve the Actor behaviour in a way to make the visualisation more natural. Currently the visualisation only relies on sensor data to spawn Actors, however different Actors, such as janitors or other workers, can be spawned anyway.
6. Further improve the performance of the visualisation. Different methods, such as grouping Actors together, can potentially improve the visualisation.
7. Add additional speech bubbles to Actors. This would give Actors more personality, potentially improving viewer enjoyment.

Furthermore, as suggested by a viewer in subchapter 5.4.1, increasing the performance of the visualisation to 60 FPS is a mayor goal for future developers. When observing potential areas for optimisation (rendering (see the glossary), animations, pathfinding), animations and pathfinding was found to take the most resources. Thus, focusing on those areas for further optimisation is recommended.

# 7. Conclusion

The goal of this thesis was to create a 3D visualisation of the Delta building – an academic building of many institute, one of which is the Institute of Computer Science. The visualisation was done in the Unity game engine. The visualisation itself simulates student and educator behaviour using sensor and schedule data.

One of the goals of this thesis was to create a visualisation that allows a high Actor count (2010) with FPS above 30. This FPS goal was satisfied using multiple views, object and material optimisation and removal of physics simulations. Furthermore, the high Actor count was possible because of pre-calculation of paths and pathfinding optimisations using tools found in the Unity game engine.

The chosen colour pallets were designed to pleasant and academic, in order to accomplish the second goal of this visualisation – viewer enjoyment. This was evidenced by 5 testers of the visualisation, who rated said visualisation with top scores and gave positive feedback.

The architecture was created to be logical and readable to future student developers of this project. It is recommended that the said developers would focus on improving the visualisation for the viewers, either by improving the building model or improving Actor behaviour. However, in the case of wanting to improve performance – it is suggested to focus on improving animations and pathfinding even further.

During the development on this visualisation, using the Unity game engine provided an opportunity to ignore implementing a different path finding algorithm, as Unity already has its own implementation – *NavMeshAgents*. This allowed the author to focus more on the visualisation and viewer enjoyment. However, implementing a different pathfinding algorithm could potentially improve performance, which was not tested during the development of this visualisation due to time constraints.

First, I would like to thank the Administrative Manager of the Institute of Computer Science, Piret Orav, for providing information about the Delta building and giving the Delta building model. Second, I would like to thank the people of University of Tartu Library that provided me with the opportunity to test the visualisation on Video Walls. Lastly, I need to thank my supervisor Raimond-Hendrik Tunnel, who took the time to answer multiple questions that appeared during development and giving me feedback on how to properly write my thesis.

# 8. References

[1] M. Kroth, P. Boverie, J. Zondlo. What Managers Do to Create Healthy Work Environments. *Journal of Adult Education,* 2007, No. 2, pp 3-4.

[2] C. Jin. The role of animation in the consumer attitude formation: Exploring its implications in the tripartite attitudinal model. *Journal of Targeting, Measurement and Analysis for Marketing*, 2011, No. 19, pp 102-103

[3] S. Kurt, K. K. Osueke. The Effects of Color on the Moods of College Students. *SAGE Open,* 2014, No. 1-12, pp 2-8

[4] A. Banitalebi-Dehkordi, M. T. Pourazad, P. Nasiopoulos. The Effect of Frame Rate on 3D Video Quality and Bitrate. *3D Research Center.* Berlin Heidelberg: Kwangwoon University and Springer-Verlag, 2014, pp 7-8.

[5] H. Tohidi, M. M. Jabbari. The effects of motivation in education. *Procedia – Social and Behavioural Sciences,* No. 31*,* 2012, pp. 1-2.

[6] S.Kim, M.Yoon, S.M.Whang, B.Tversky, J.B.Morrison. The effect of animation on comprehension and interest. *Journal of Computer Assisted Learning,* 2007, No. 23, pp. 1-2.

[7] N.Cohn. Beyond speech balloons and thought bubbles: The integration of text and image. *DE GRUYTER MOUTON*, San Diego: University of California, 2013, pp 35-36.

[8] L. Charlotte Rost. Your Friendly Guide to Colors in Data Visualisation: 2016. https://lisacharlotterost.github.io/2016/04/22/Colors-for-DataVis/ (28.04.2018)

[9] S.Zhang. Finding the Right Color Palettes for Data Visualizations: 2015. https://www.invisionapp.com/blog/finding-the-right-color-palettes-for-data-visualizations/ (28.04.2018)

[10] I. Knez, S. Niedenthal, Lighting in Digital Game Worlds: Effects on Affect and Play Performance. *WP9 workshop*, Sweden: University of Gälve, Malmö University, 2006, pp 1-6.

[11] A. Voitenko. Delta õppehoone keskkonna visualiseerimine. Tartu Ülikool, bakalaurusetöö 2018.

## 9. Glossary

| | |
|---|---|
| Mesh | A collection of vertices, polygons and edges that defines a shape of a 3D object. |
| Vertex | A data structure that stores attributes such as colour and coordinates. |
| Face | Faces consist of triangles or convex polygons. |
| Polygon | Geometric figure with straight sides and angles. |
| Material | An enhancement of texture mapping that simulate real-life materials. |
| Despawn | Remove an object from the game environment. |
| Spawn | Make an object originate at a fixed point in a game environment. |
| Bake | Pre-compute something into a more permanent form.[39] |
| Render | Automatic process of generating an image from a 2D or 3D model.[40] |

[39] https://cgcookie.com/articles/big-idea-baking
[40] https://www.techopedia.com/definition/9163/rendering

## Appendix

The appendix includes tables that were deemed too big to include in the thesis, but can provide useful insight to future developers of this thesis.

There is also an archive accompanying this thesis that includes the following:

1. Images used for the visualisation that were not created by the author of this thesis (credit given in the README.txt file).
2. Textures used for the visualisation that were not created by the author of this thesis (credit given in the README.txt file).
3. README.txt file that contains the source of images and textures, the git repository used for the development of this visualisation, as well as recommendations on what Unity version is best to test the visualisation.
4. A project of the visualisation.

*Table 1. Views and their rotation and position.*

| **View name (See Figure 7)** | **x;y;z positions** | **x;y;z rotation** |
|---|---|---|
| View1 | -8; 18; -7 | 29; -35; 0 |
| View2 | 19; 15; 119 | 36; 223; 0 |
| View3 | -106; 16; -3 | 26; 57; 0 |
| View4 | 4; 12; -2 | 29; -35; 0 |
| View5 | 11; 13; 81 | 36; 223; 0 |
| View6 | -119; 13; 0 | 26; 57; 0 |

*Table 2. Views and their orthographic parameters.*

| **View name (See Figure 7)** | **Size** | **Clipping Plane Near** | **Clipping Plane Far** |
|---|---|---|---|
| View1 | 10 | 16 | 117 |
| View2 | 16 | 5 | 109 |

| | | | |
|---|---|---|---|
| View3 | 15 | 10 | 120 |
| View4 | 9 | 18 | 75 |
| View5 | 14 | 3 | 115 |
| View6 | 12 | 15 | 88 |

## License

**Non-exclusive license to reproduce thesis and make thesis public**

I, Aleksander Nikolajev (13.07.1996),

1. herewith grant the University of Tartu a free permit (non-exclusive license) to

    1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

    1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright, of my thesis

Delta Building Visualisation, supervised by Raimond-Hendrik Tunnel,

2. I am aware of the fact that the author retains these rights.

3. This is to certify that granting the non-exclusive license does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 09.05.2018