UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Crowd Simulation in an Emergency Situation

Project management

Author: Daniel Roca Lopez

Director: Nuria Pelechano

INDEX

TABLE OF CONTENTS

LIST OF TABLES

Chapter 2.1 Ship

TABLE OF FIGURES

Chapter 3.2 Navigation Mesh

Chapter 3.3.2 Representation of Danger

Chapter 4.2 Graphics Pipeline

Chapter 4.3 Summary of Optimizations

Chapter 5.1 Mind Map

1. INTRODUCTION

1 Introduction

The evacuation plan for different buildings, ships and all kind of public spaces is hard to define, at least, if you want it to be the most efficient plan, thus the safest and the fastest. There are also laws (United States Code sections, 2016; United States Departament of Labor, 2017) related to these plans. That fact, in combination with the latest technologies related to virtual reality, that came to live, made us think about improving the creation and design of these plans, not with real tests which are hard to simulate and expensive, but using immersive technology, “Virtual Reality”.

There are already some experiments related to crowds (Pelechano & Badler, 2006) and evacuations but they are done in simple 3D, shown on computers which make it a bit hard to feel it real, thus to react as you would do in a real situation. The advantage of VR is that the visual and auditory senses are covered. In a near future, more haptic sensors will come to light, such as AxonVR (Rubin & Crockett, 2012) and others (Stone, 2001; van der Meijden & Schijven, 2009; Carlin, 1997; Burdea, 1996; Robertson, 2014). Therefore, the idea is to merge this topic (crowd simulation and evacuation plans) with virtual reality.

1.1 Project Formulation

The project idea is about simulating a realistic environment where the user (we understand user as the person who is going to run the application) will experience a situation of danger in virtual reality. That way, we can make him feel like the experience he is living is true thus we can get more realistic results when using this project as an experiment.

There are already some experiments trying to simulate those environments in a realistic fashion (Pelechano Gómez, 2006) but most of them are in simple 3d where you have to use your keyboard and/or other controllers to move around, while you see the simulation on a window.

In order to accomplish that simulation in VR, we have to study and simulate the behavior of random people to add some autonomous avatars into the simulation, which will increase the overall realism of the virtual scenario. On the other hand, such simulations are computationally expensive, not only because of

the scenario but the simulation of many rigged avatars, as well as their behavior. Therefore, we also have to study and find some optimization techniques to make it runnable on a head mounted display like Gear Vr, and to decide whether run it on this hardware or choose another alternative like the HTC Vive (HTC Corporation, 2017) or Oculus Rift (Oculus VR, LLC., 2017).

On this document we will be describing a complete project. The project will be considered minimally completed once we have a working environment with a few agents (could be 10 or 15), with an announcement of the dangerous situations and there is a goal (a safe place) where the user has to go in order to finish the application. This will represent the setup of an experiment, and we will run a pilot test to determine the degree of realism and the naturalness of the navigation and interaction with the environment and other agents. The result of this work could be then used for a longer experiment to evaluate human behavior in such a dangerous situation. From there, anything else is considered an improvement.

1.2 Project Structure

In **chapter 2** there are explanations about the project's design, and more specifically, how the models were created, and how we modeled the ship using Blender (Blender Foundation, 2017) and Unity (Unity Technologies, 2017), as well as the reasons for every choice in the modeling process. There are also explanations about how the avatars were created. **Chapter 3** is used to explain how the application was programmed, starting from crowd AI ("Artificial Intelligence"), up to application running pipeline. **Chapter 4** is mainly focused on explaining the applied optimizations on this project and possible improvements for a better performance. The proper way to use this project as an experiment is explained on **chapter 5**. **Chapter 6** is used to explain how this project was managed, such as developing expenses and sustainability related issues. It also includes the relation between the project and Computational branch in Computer Science. Finally, **chapter 7** is used to explain the summary and recommendations for further studies and own conclusions.

2. DESIGN

2 Design

The events on this application are set on a ship, which is sailing in the ocean, with people inside. At some point, there must be a danger situation and people should be running, searching the closest safe place to be. These events should happen in a realistic fashion, therefore, there should be different floors, including the deck of the boat, some rooms per floor and stairs to connect the different floors. To make it look a bit more like a maze, we added different rooms in the middle of each floor, and also different stairs to move through floors. This helps, not only because it adds complexity to the path that needs to be followed to get to a safe place, but also because there are several ways to get there, thus the programmer can “play” blocking one of the paths to make the user change its exit path. These things belong to a structural design but, in this application, there is another must have: “Crowd”. This is where the avatars come to life.

Avatars, on the other side, represent the crowd (from ship's crew to passengers). These avatars are there to make it feel more realistic, not only because of the realism in the fact that there should be more people on a ship other than you, but also because of the human factor and “embodiment” (Nurislamovich Latypov & Nurislamovich Latypov, 1999). Embodiment attribute is used to represent the feeling of thinking that the virtual body is your body. It can also be used to represent how real you feel the virtual environment. The avatars, in this application, will help you to feel more embodied or to feel the virtual environment more real, boosting the visual and auditory senses.

2.1 Ship

To begin, we will start showing two pictures, **Figure 1**, **Figure 2**, to indicate the names of the different parts of a ship.

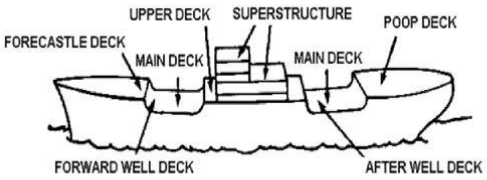

Figure 1: Deck names

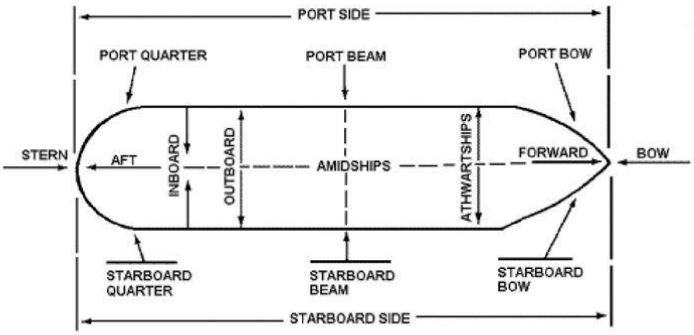

Figure 2: Locations and directions aboard ship

The figures are a close representation to the ship model, being used on this project. The process of modeling this ship is the following:

- Getting a simple model of a ship.
- Split the model into smaller parts to be able to control it easier, this is, the materials for every part of the model and, to be able to enable/disable different parts with an occlusion algorithm implemented specifically for the purpose of this project.
- Model all the missing parts of the ship, this is, the complete interior:
  - Rooms
  - Stairs
  - Floors
  - Decoration: Piano, water …
- Optimize model, this is, reduce the number of polygons.
- Add different materials for the different parts of the ship.

Everything but the last step, is going to be done with blender, which is a modeling software designed for that purpose. The model was downloaded from a 3D assets website (CadNav.com, 2017), but it has only the hull. Therefore, we had to design everything else and the materials, because it did not come with the correct materials.

A few commands from Blender that helped me to do all the modeling where the following:

- Alt + B + selecting square → Clipping border, used to select the part of the model you want to see, and hide the rest, so it helps you to model inside the geometry. (**Figure 3**).
- In edit mode (vertex selection instead of objects):
  - P + Selection → To separate different parts of the model.
  - X + Limited Dissolve → Used to decimate the mesh, which means to reduce the number of polygons in the geometry. (**Figure 4**). This is a very important feature toward reducing the complexity of the model and thus gaining performance during rendering time.

Commands representation

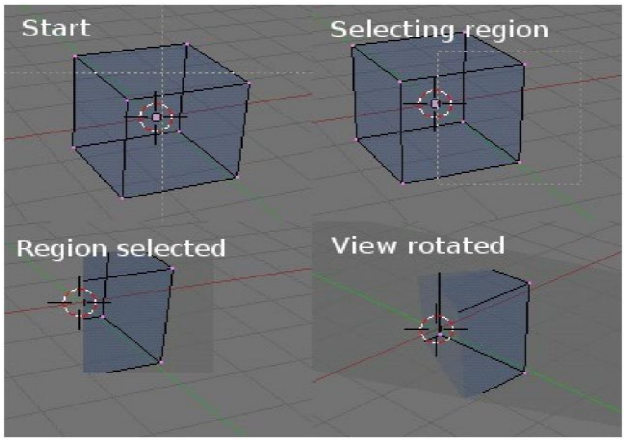

Figure 3: Clipping border

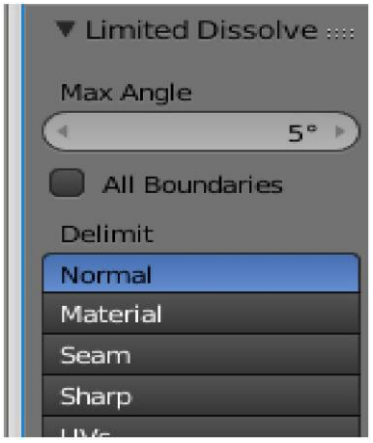

Figure 4: Limited dissolve

Thanks to all the previous shortcuts and some other tools that Blender offers to create new shapes and all kind of geometries, we could design and create all the floors, rooms, stairs and decoration, as well as separating the ship into smaller parts.

The way we separated the ship is by levels and sides. It is, the whole ship is being divided into 4 different levels of height, plus two sides per level. That way, we ended up having 8 main parts of the ship. That was not enough when it came to rendering performance, therefore we decided to divide hammocks, pipes, windows and other smaller components such as chairs or tables. All this process, in combination to mesh decimation, helped me to reduce the ship as we see in the **Table 1**.

Table 1: Comparison between original and optimized ships

	Original Ship	Optimized Ship
Vertices	650.000	353.000
Triangles	1.200.000	550.000
File Size	26.5Mb	15Mb

The rest of the ship optimizations will be explained in **chapter 4**, along with the other optimizations that were made in the application. The remaining tasks of the modeling process were done inside Unity. On Blender we created some default walls, doors, furniture and textures, and then, from there, we created a few prefabs[1] on Unity to speed up level design, creating similar rooms or, for instance, the ceiling lamps.

[1] Prefab: Set of geometry or unity Game Objects, which share the same attributes thus are saved as a predefined object.

Add different materials for the different parts of the ship

When all the model was already defined, created and decorated, I started creating materials for all the geometry. A material consist on a color or texture, with some extra attributes such as receive or cast shadows, use specular light and/or reflections and some other attributes. For instance, the material used on glass geometry has the attributes seen on **Figure 5**. There we can see that specular light and reflections are enabled but, the remarkable attribute here is the one that makes the glass transparent.

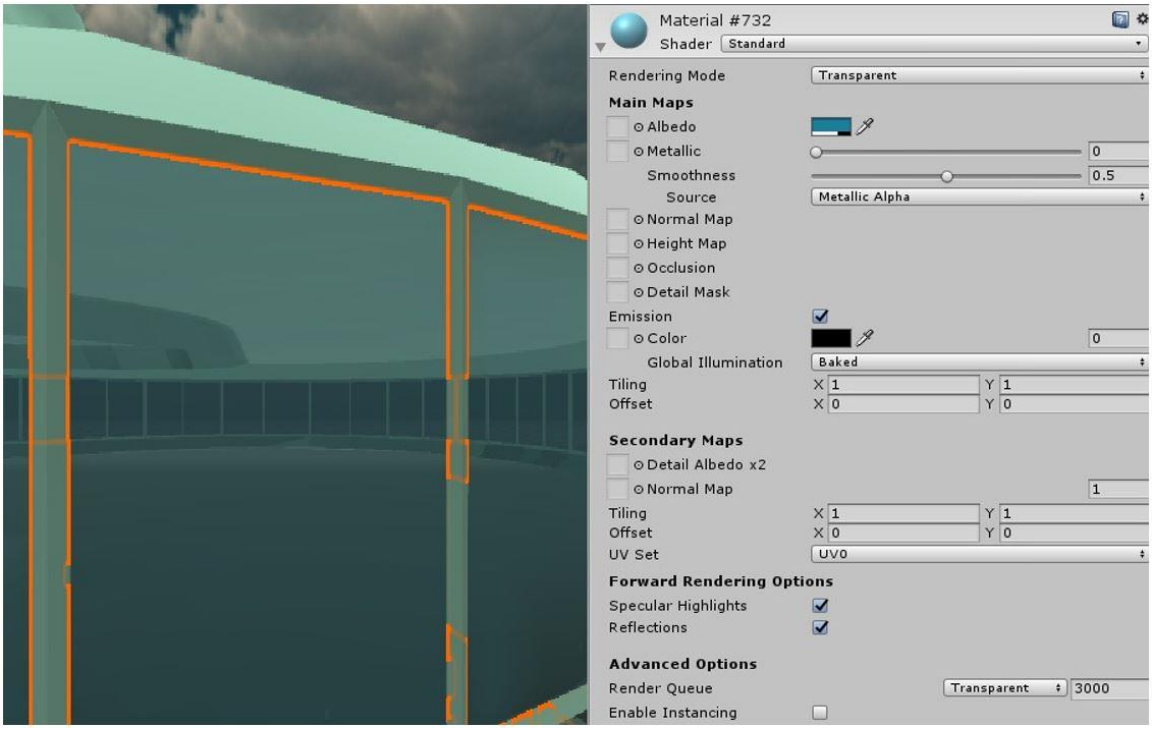

Figure 5: Example of window and material attributes

On **Figure 6** we can see the following settings: Rendering Mode: Transparent, and Albedo color. The color is selected carefully, with an alpha of 165 out of 255, that way, in combination with the transparent attribute, makes you capable to see through.

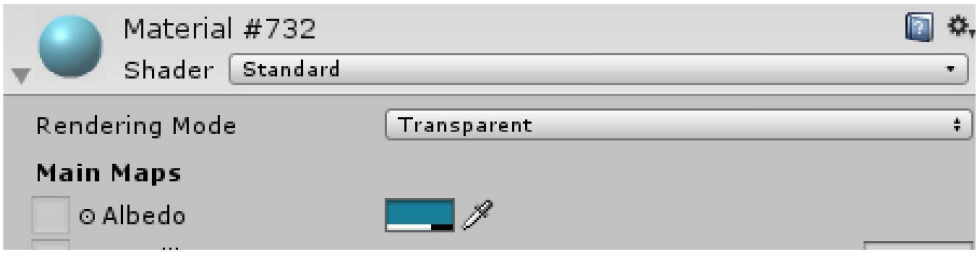

Figure 6: Glass material settings

Finally, the ship's design per floor is almost the same for every floor, with small differences such as rooms distribution. It could have been more complex to make the user get lost easily thus get a bigger variation on the experiment's results but, at the moment, this is a first approach.

On **Figure 7** and **Figure 8** we have a floor example, we can appreciate the walls and rooms of that floor. We can also notice that there are no roofs on the rooms. This is not true, there are roofs on the rooms but they have “Backface-culling[2]” activated. The second reason is that the upper floor is disabled to capture the photo, otherwise, it would be activated. Therefore, the user would be able to see the roof of each floor.

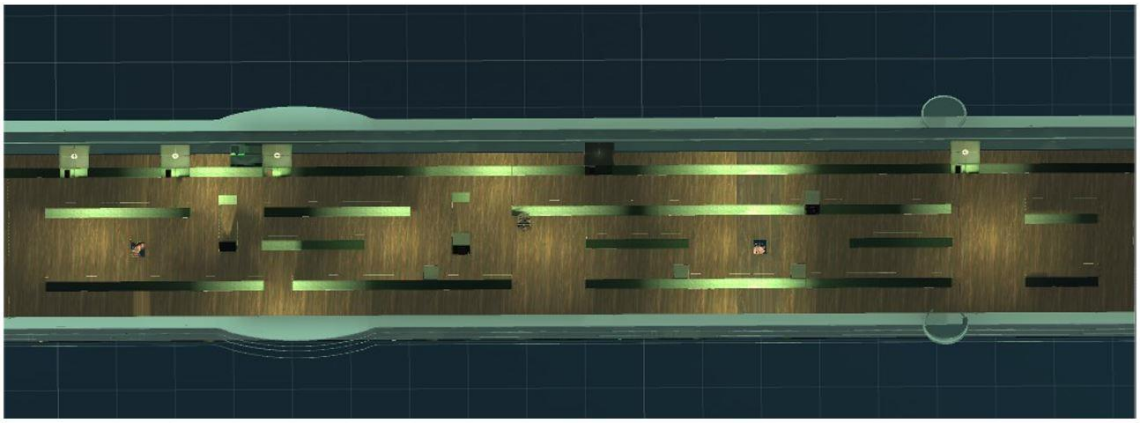

Figure 7: Floor view corresponding to one of the ship's floors

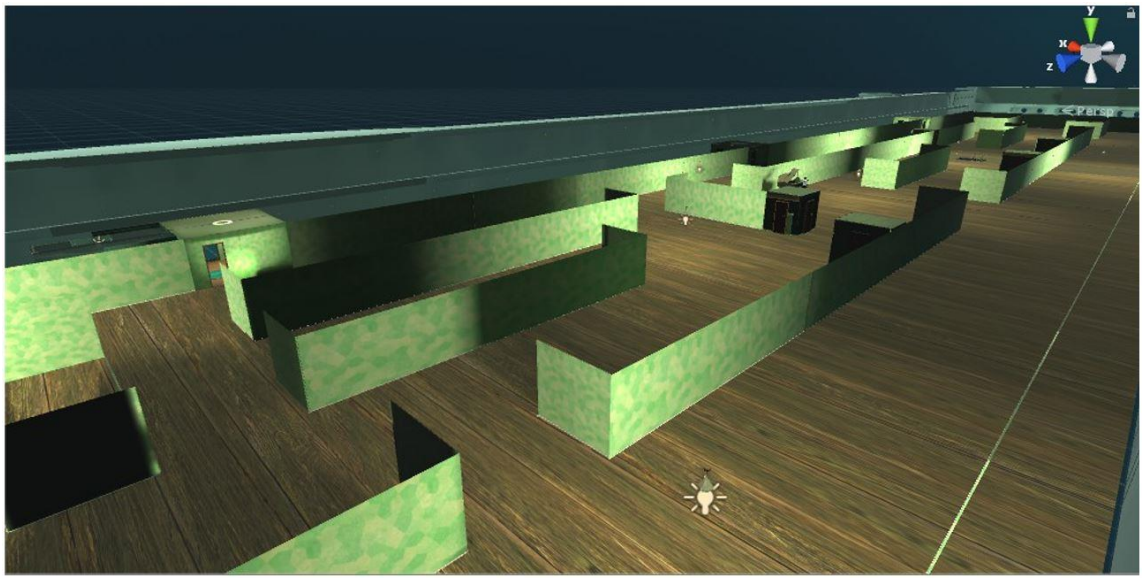

Figure 8: Elevation view corresponding to one floor

[2] Backface-culling: Its an algorithm used in graphics to determine whether a face should be drawn/seen or not.

Finally, we can see all the ship, from the outside, in **Figure 9**. For this first approach, this ship is not close to the beach at the beginning, and it does not have water on the outside because for the current experiment setup, the user would not be able to see through windows nor from the ship's deck. However we have included the sound of the ocean, to boost the realism of the environment. There are not any life saver boats either, because it would drastically increase the amount of work involved in this project and, considering this is just a first approach and an attempt to see how embodied do people feel inside the application, it could be used for a future version.

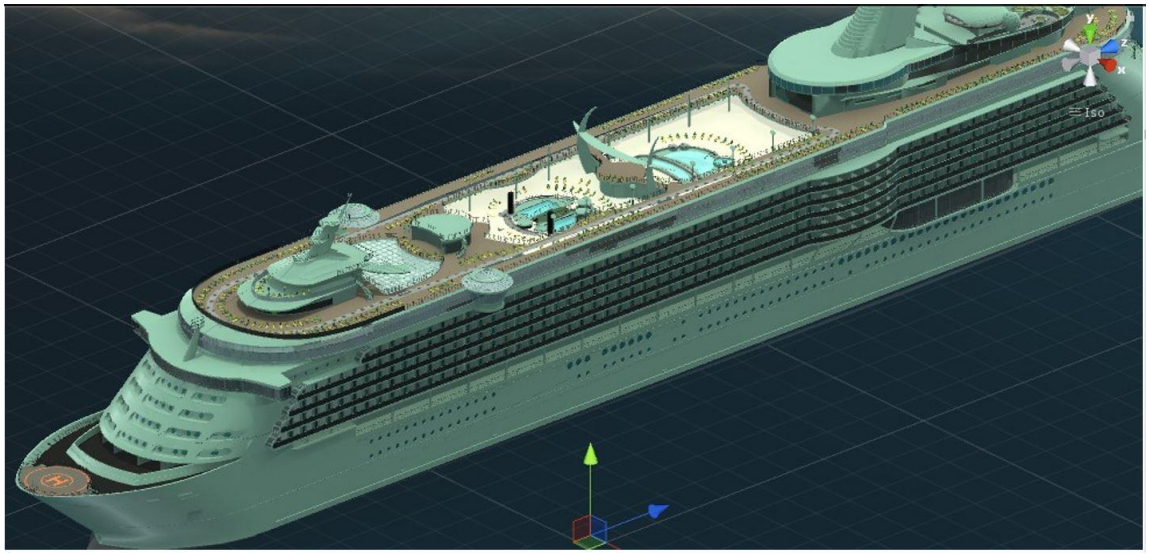

Figure 9: Complete ship from the outside

2.2 Avatars

2.2.1 Customizing Avatars

This step consists on creating avatars whose are going to represent the crowd. These avatars can be created in several ways but, the easiest, fastest and cheapest is through Adobe Fuse (Adobe Systems Incorporated, 2017) due to its very intuitive software to model avatars or choose between already existing body components. Last but not least, it is associated with Mixamo (Adobe Systems Incorporated, 2017), which is an on line software that gives you the rigging[3] for your character, automatically, depending on the character's UV's[4]. On **Figure 10** and **Figure 11** we can see the process of creating an avatar with Adobe Fuse.

[3] Rigging: process of attaching bones to each vertex of the character.

[4] UV: texture coordinates, U corresponds to X, V corresponds to Y.

On the other hand, these avatars are a bit heavy in term of polygons, but thanks to decimation techniques, we can reduce the output character with around 11.000 vertices, to another character, very similar, with around 3.500 vertices. Therefore, my avatars will be decimated to speed up the application. It is, not only because of the number of vertices to draw but because of the number of operations needed to perform. It will be explained with more detail on **chapter 4**.

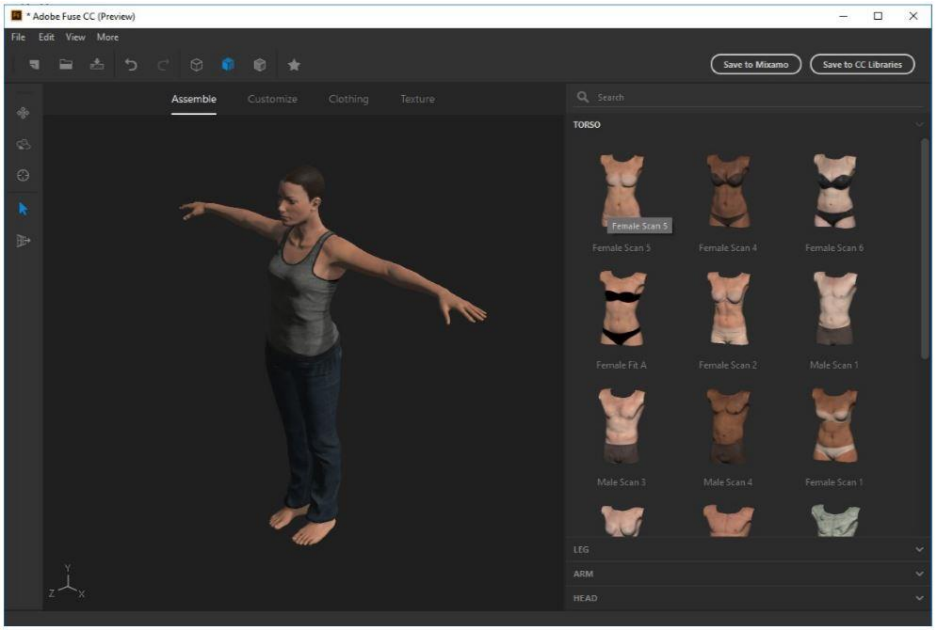

Figure 10: Customizing avatar's body with adobe fuse

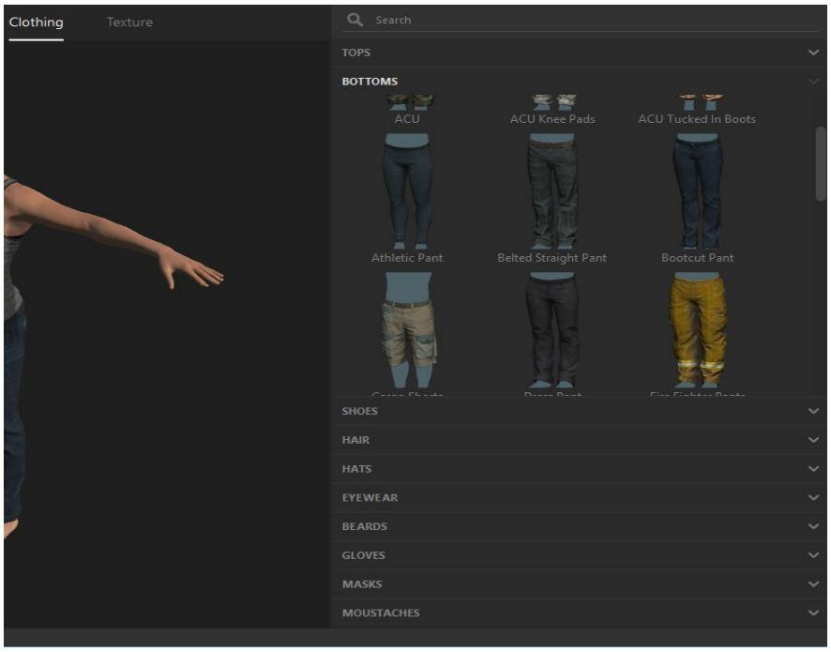

Figure 11: Customizing cloths for the avatar with adobe fuse

Previous techniques are used for common avatars but, we wanted to add a special avatar, an avatar for the user and we decided to create one, similar to Wentworth Miller, customized the same way he was, as an actor, on Prison break. This is because of the analogy, he tried to escape from prison, and the user is going to try to escape from the sinking ship. This avatar had to be customized in a different way, therefore, the steps to create it were the following:

Get his face in 3D

Thanks to existing free software Faceworkx (Looxis GmbH, 2017), giving two photos, after a bit of work, you can get the 3D face as seen in **Figure 12**. To get face photos, it was a bit tricky as we had to get screen shoots of different scenes, clean them with Gimp2 (The GIMP Team, 2017) as seen in **Figure 13**, and then, add them into the software.

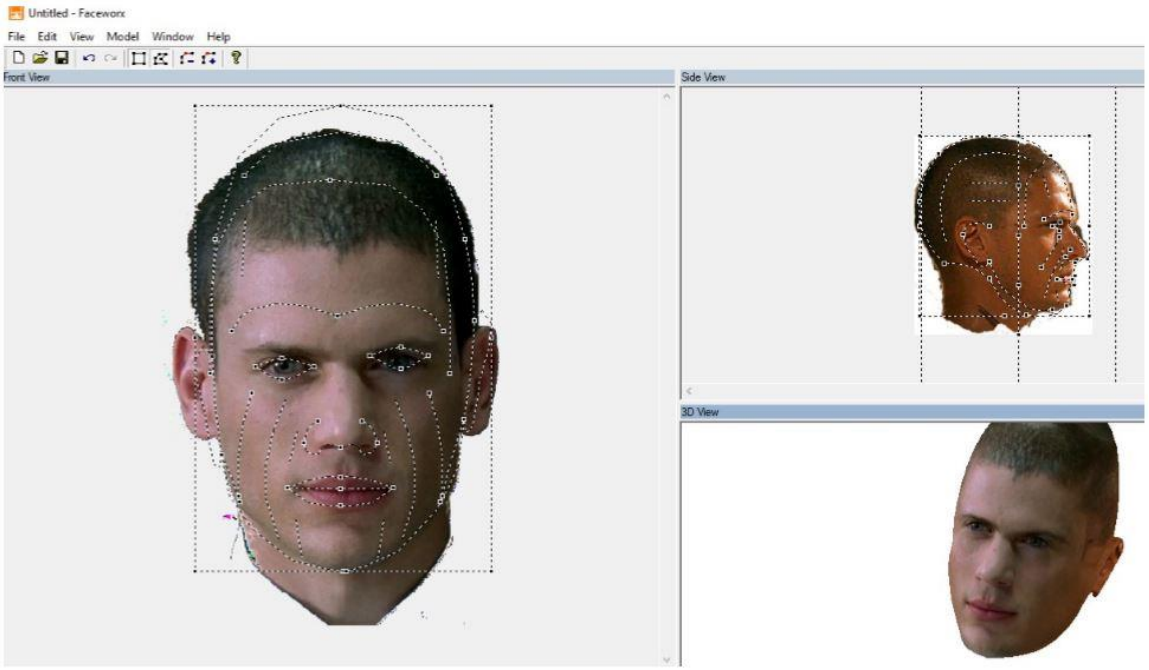

Figure 12: Faceworx software, creating Wentworth Miller's face

First step was to get his face, then, thanks to another private and confidential software, designed to do “non rigid registration”, in other words, it works modifying original mesh, keeping vertices number, to a given mesh. That way, we could get one of the previous avatars and modify his face to become this one, keeping its character rigging.

Finally, we have his body but, we wanted to make him look even more realistic and closer to the roll he plays on that TV Serie. Therefore, we had to add the tattoos the actor wear on the show, to this avatar. For that step, we used blender. We got a few photos of his tattoos and then, thanks to some techniques that we have learnt with Blender, we could unwrap the avatar's mesh into a plane, representing the character's UV mapping, and then, we painted this texture, representing the character's texture with the correct UV mapping. A small example can be seen on **Figure 14**, resulting into a finished rigged avatar.

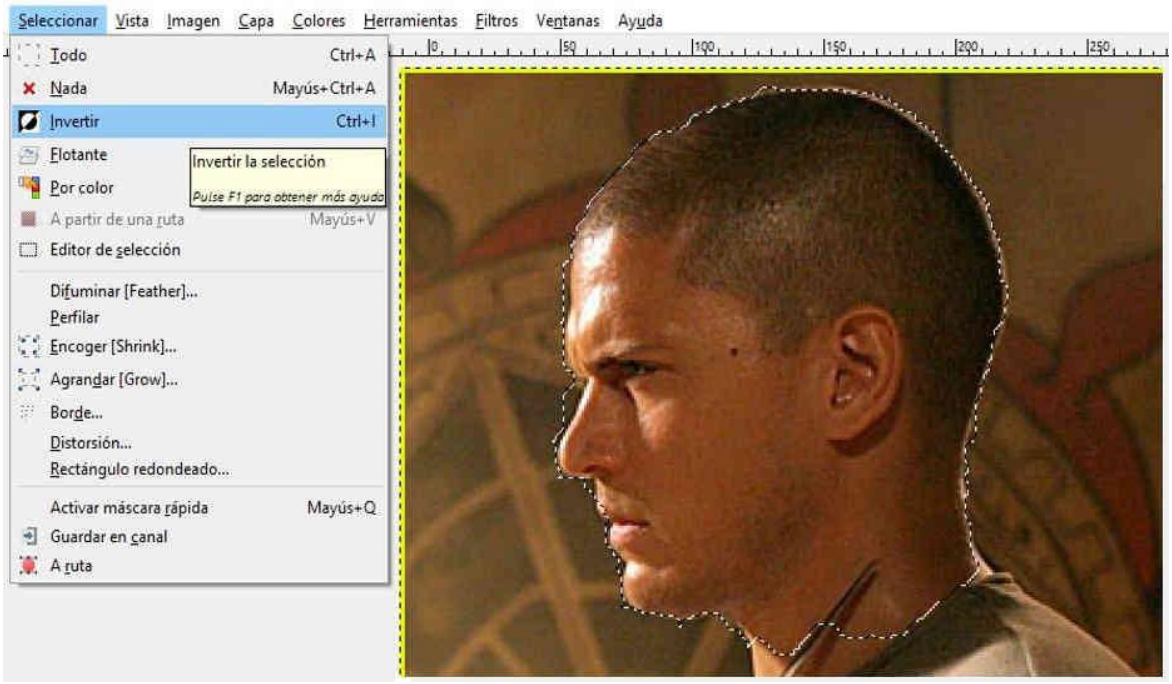

Figure 13: Cleaning photo with Gimp

The process of painting textures into the avatar consist on creating different UV maps, an UV map example is the right side of **Figure 14**. We created an UV map for every arm, torso and legs. Then, we selected the vertices of the model that we wanted to represent on this map, clicked on U → Unwrap from projection. That way, we could get different parts of the body into the texture. After that, we selected which texture to use to clone the color from, and started painting the body, while the UV map was being painted.

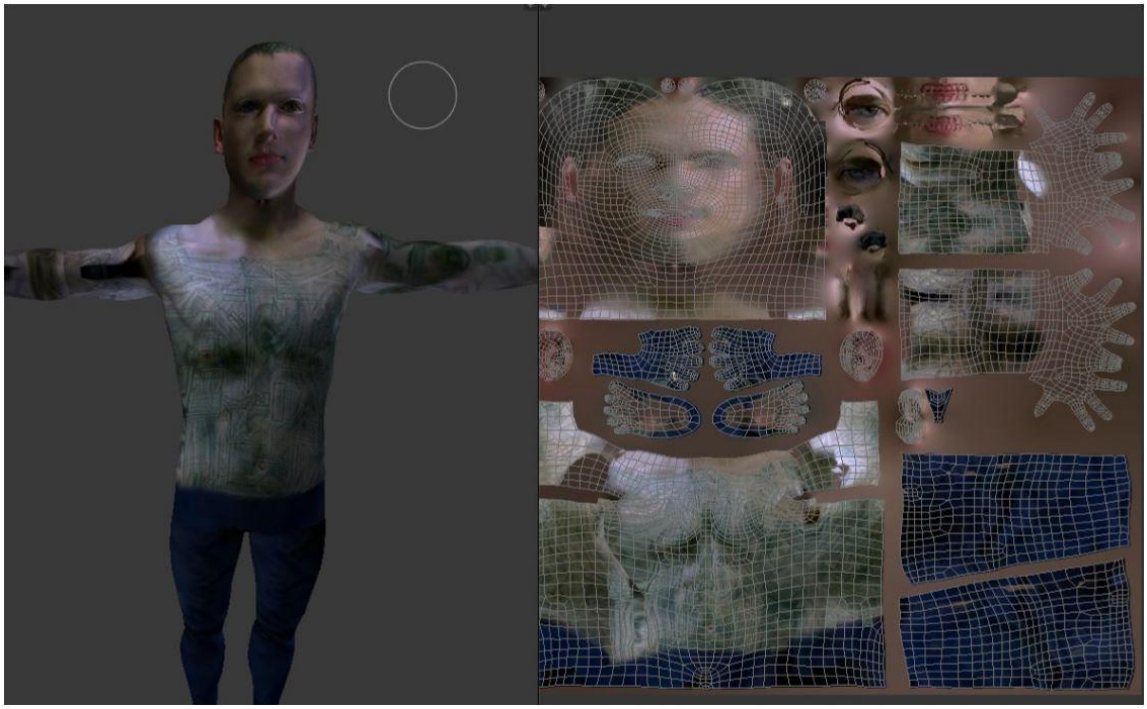

Figure 14: Painting avatar's texture with Blender

2.3 Additional Features

The ship itself is not enough to create a good realism effect, this is where this additional content or features come to light, to improve the feeling of belonging to this situation. We could distinguish between visual and sound effects. For the sound effects we have:

- Chattering sound in the background, as if the people were talking.
- Ocean sound, on both sides of the ship, with 3D sound effect enabled, to increase the feeling of being there.
- Music sound. There is a piano player, playing a piano song, which also has the 3D sound effect. It also helps the user to orientate himself around the ship, depending on how loud does he listen the music.

On the other hand, we have visual effects, such as:

- Pictures on the bathroom.
- Beds and other dorm furniture.

- Ceiling lamps all around the ship, not only to increase the light but to create the effect of different light spots, as we would see in real life.
- Piano with an avatar playing it (**Figure 15**).
- Furnished rooms, with decoration such as a bowl with apples.

Figure 15: Piano with an avatar playing a song

We have also played with textures, to create the effect of deep, for example, on the ground, made of wood, it looks like there are tiny horizontal holes but, actually, it is achieved with bump mapping[5]. This is used to avoid modifying the real mesh, because it would mean to increase the number of vertices and total polygons, decreasing the performance. The animation for the piano player was downloaded from Mixamo. Animations are usually created with tools where you select where every bone of the avatar should be on every frame, but you can also record any human doing the same movements, with any tracking system such as the one that comes with vive or perception neuron. That way, you record all the user's gestures and movements, resulting into a smoother and more realistic animation. Animations from Mixamo are produced using the second methodology.

[5] Bump mapping: is a technique used to simulate bumps and wrinkles on the surface of an object. This is achieved by perturbing the surface normals of the object and using the perturbed normal during lighting calculations

3. CONTEXT PROGRAMMING

3 Context Programming

The application itself, has a set of events, driving the user from a peace and exploration instance, to a danger situation. These events are going to happen for you and some other avatars, who represent the crowd. These avatars and you are going to be influenced by the environment and the perception of danger, but also by the walkable or explorable places, this is where Navigation Mesh comes into place. All these components, with the application logic, are going to work in harmony to create the application flow.

3.1 Crowd Artificial Intelligence

There are a few remarkable points regarding crowds AI (Reynolds, Red3D, 1997; 2001). First of all, path-finding algorithm. Unity uses his own path-finding algorithm, which is optimized enough for the uses we will give. There are lots of algorithms for path-finding, such as A* or IDA*, all of them are very efficient but, unity already implemented their own path-finding which is easily compatible with their navigation meshes, therefore, it is highly recommended to use their own path-finding. Crowds will use path-finding for one main purpose: find the way to get to the next step.

On **Figure 16**, we can see an example of how to configure paths for the crowd.

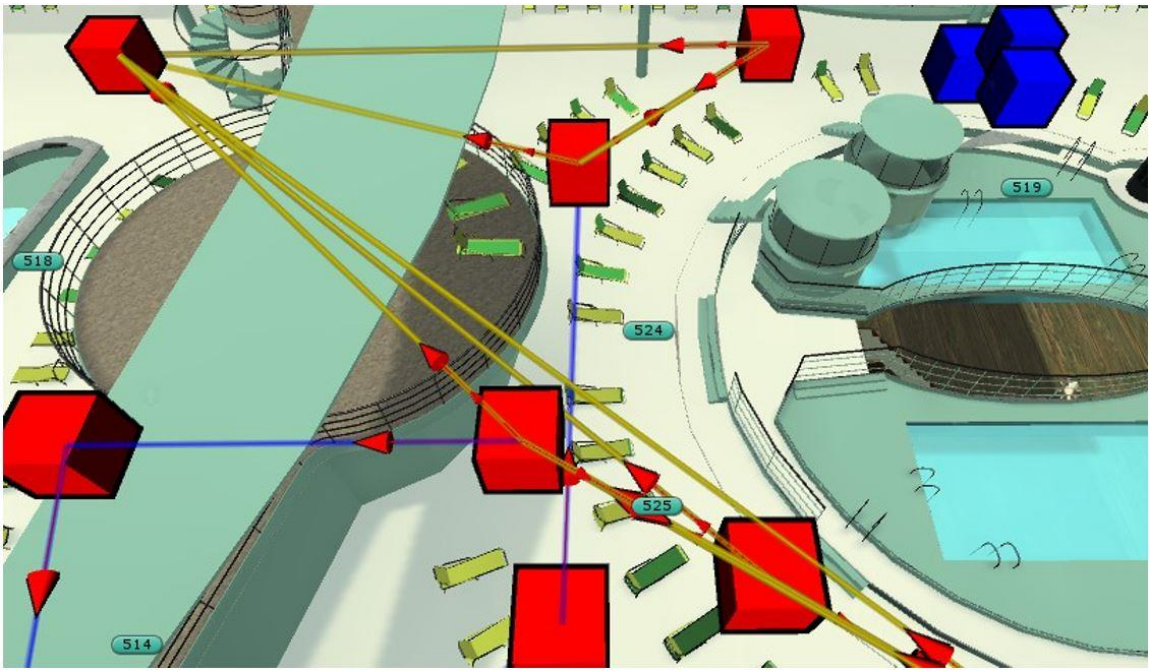

Figure 16: Example of steps to go, for crowds

Every red square means that an avatar could appear there at the beginning of the application. If it is connected, it also means that an avatar can walk on this direction. Mainly, it is a directed graph. Blue squares are used for special cases, such as giving different chances of going there, or appearing there as an avatar. We have implemented this system of placing squares, with an intuitive interface on Unity, to make it easier to create new paths and to decide where should the avatars appear when the application is running.

The avatars need to have, not only this path attached but a script to manage all its behaviors, such as deciding whether to run, walk or stay idle, as well as the corresponding speeds for walking and running. With the script on the **Figure 17**, the one on the left, you can even select the speed to use when the avatar is scared, or the gender (used for voices). The Character Controller on the right, and Nav Mesh Agent scripts, are used to specify the avatar's dimensions, to be compatible with the navigation mesh. The Obstacle Avoidance section inside “Nav Mesh Agent” script, is used in combination with the navigation mesh, to decide if the user fits into a specific hall or place. For instance, the character's height had to be reduced from 2.5 (initial value) to 1.7. Otherwise, the avatars were not able to go downstairs because the stairs have kind of looping shape, and the algorithm would think that the user would not fit on the stairs. We can see an example of the navigation mesh on the stairs, on the next section.

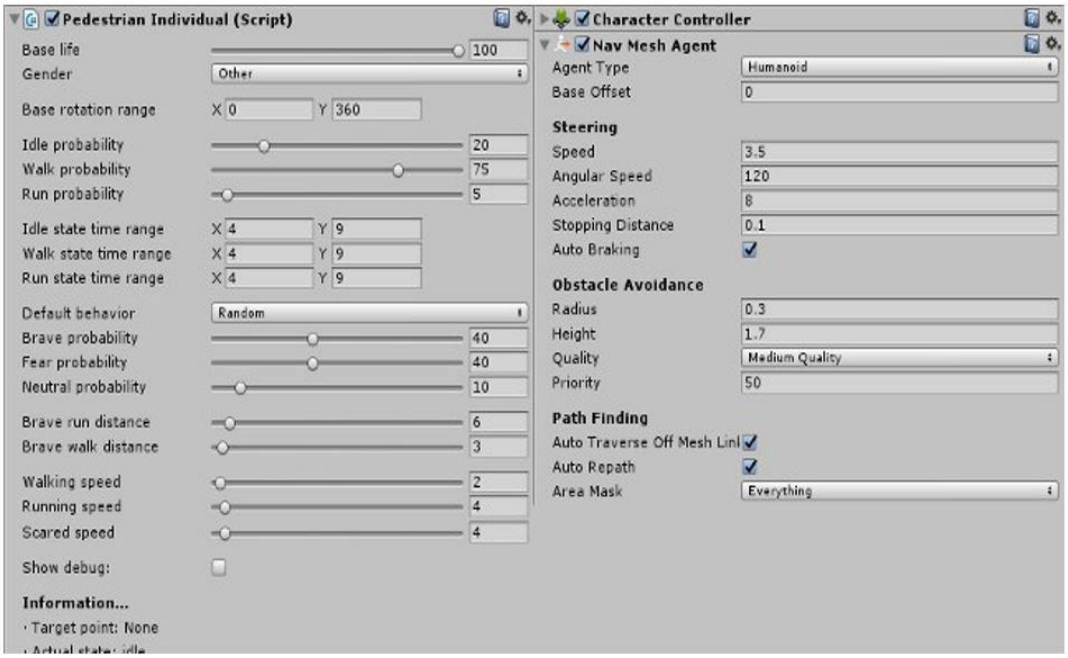

Figure 17: Scripts attached to every avatar

Safe Zones

The avatars had to have another behavior for the emergency situation, therefore, we have created what we call “Safe Zones”. A safe zone, basically, is a trigger that gets called when an avatar gets inside the working distance. If an avatar needs to find a safe zone, they will choose between one of the existing safe places. To make it more realistic, we placed safe zones at different spots inside the ship, even if they are not really safe, so the user can get confused about who to follow or not. Those safe zones can also be linked between them. For instance, there is a big safe zone where almost all other safe zones are linked to. On the **Figure 18** we can see the general safe zone, and **Figure 19** is an example of an interim safe zone, linked to the general one.

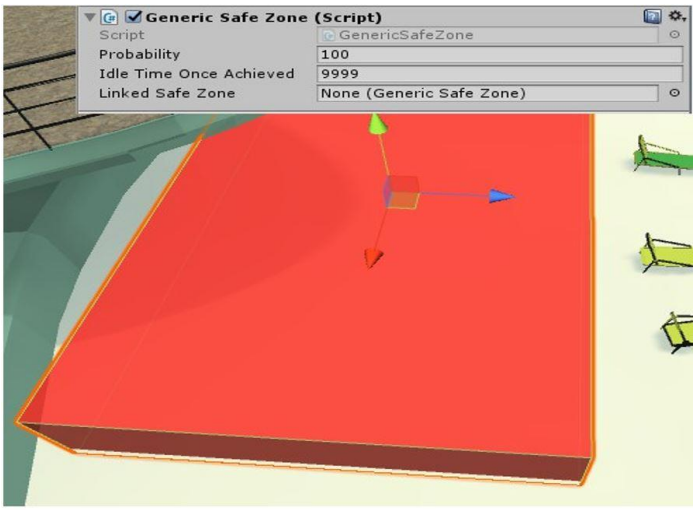

Figure 18: General safe zone, at main deck of the ship

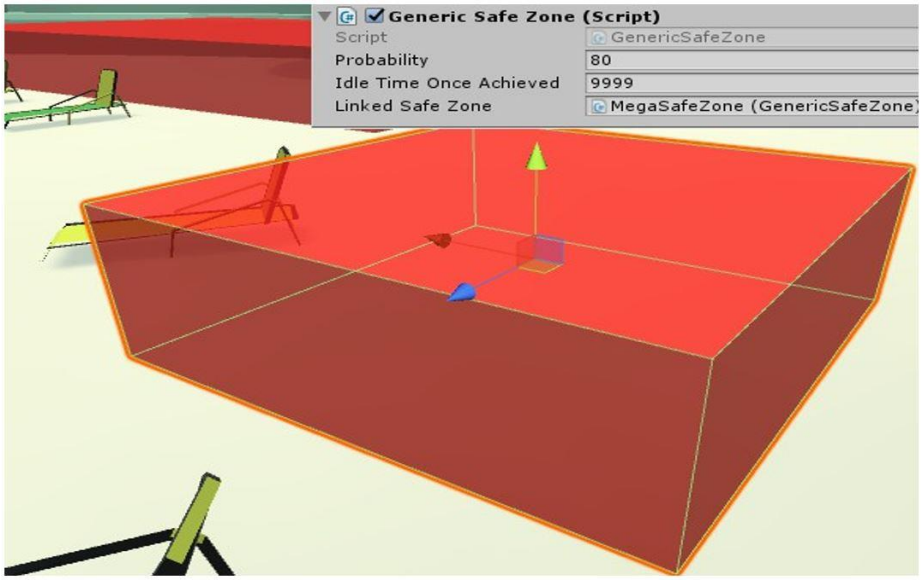

Figure 19: Simple safe zone, at ship's main deck, with the general safe zone behind

3.2 Navigation Mesh

Navigation Mesh is a data structure which describes the walkable surfaces of the game world and allows to find path from one walkable location to another in the game world. The data structure is built, or baked, automatically from your level geometry. There are some configurable settings before baking your world's navigation mesh. On **Figure 20** we can see some of them. A summary of the basic attributes would be the following:

- Agent radius: it sets to walkable any surface wider than this radius, as we expect the avatar/agent to have that radius.
- Agent height: similar to previous attribute but for the height. If the distance between two floors were lower than this height, that part would not be walkable or baked.
- Max Slope: used to determine how big the slope can be, to be considered into baking.
- Step height: it refers to the difference in height between two surfaces, you would let the user to step.

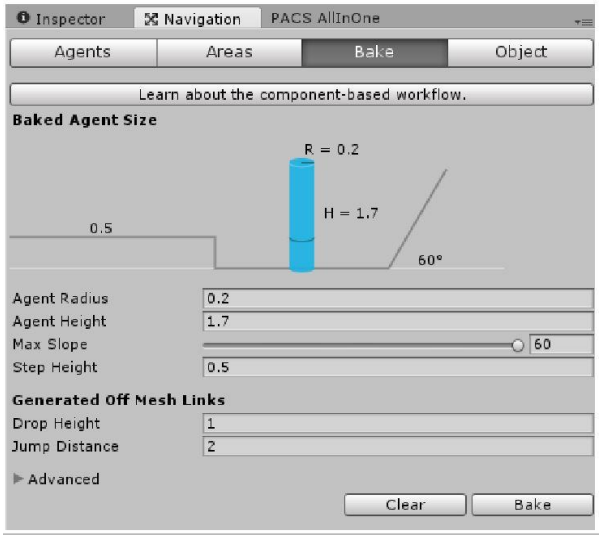

Figure 20: Settings to bake navigation mesh

Unity's default navigation mesh system also allows you to set costs to different surfaces. For instance, you could let an avatar to walk inside a swimming pool but, on the other hand, it would slow it down, so, you can set a higher cost to the

swimming pool surface, that way, when calculating path-finding, this path will be taken into account but also its cost, so, if there is a better way to get to the desired spot, it will use the alternative.

To get the stairs working with the navigation mesh, it was a bit tricky because if the stairs design (**Figure 21**). The height between the stairs on the corner and the roof was not high enough so, the workaround was to make the upper floor tinier in order to increase the distance between both surfaces. We can see the same stairs with navigation mesh painted on them on **Figure 22**. The blue shape belongs to the walkable surface, this is why it does not get too close to the railing, because the agents are supposed to be 0.2 units wide.

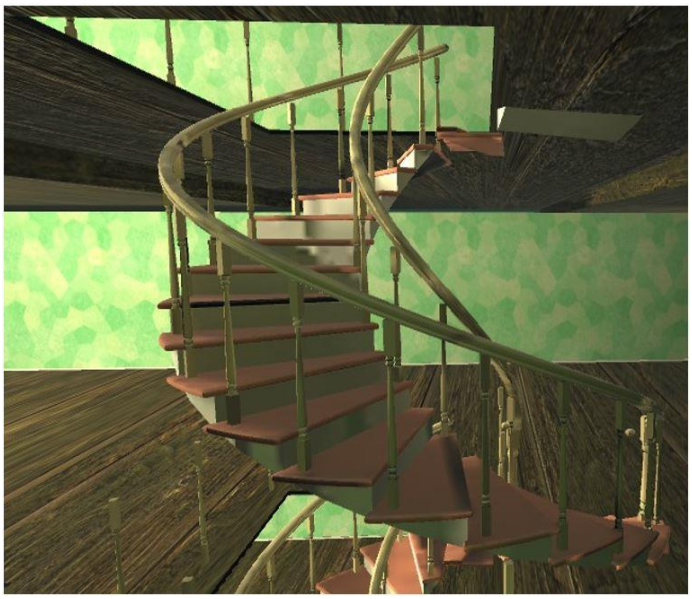

Figure 21: Stairs between floors

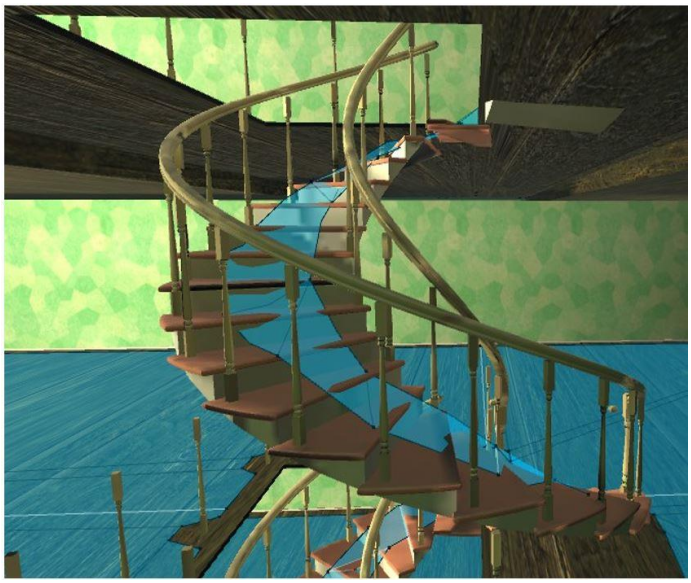

Figure 22: Stairs between floors with navigation mesh drawn

This navigation mesh is applied to the whole ship, floor by floor, even inside every room, so every avatar knows where to walk. We can see an example of the navigation mesh's step height parameter on **Figure 23**. The circles on the ground are the limit of the plane surface, and those lines moving to the top of the higher surface, are calculating the distance.

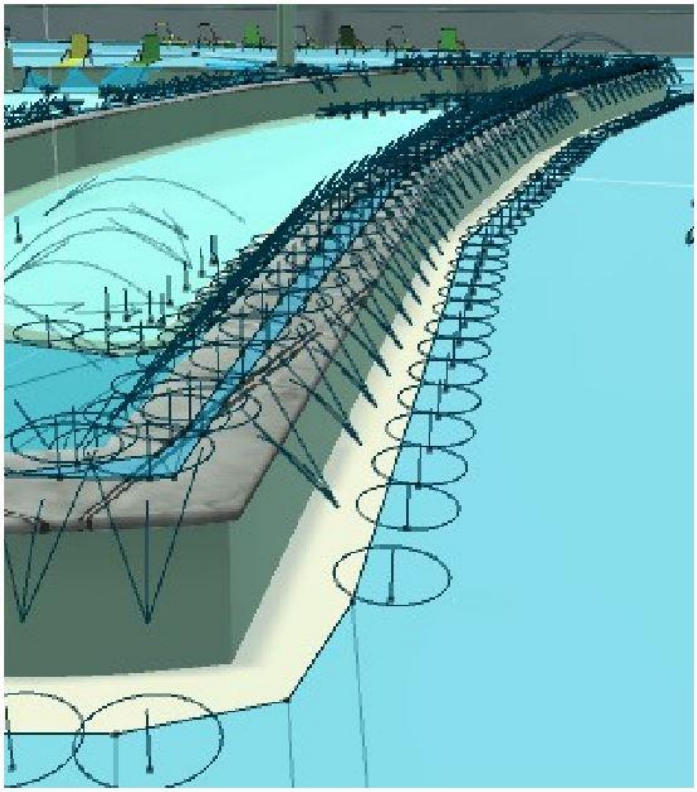

Figure 23: Example of navigation mesh's step height parameter

3.3 Application Logic

The expected flow of the application was to simulate the whole boarding process, until you get to your room, then, you would find a room mate, take a walk around the ship before the action begins, and then, the alarm. To make it simpler and to be able to focus on optimization processes, we avoided the first steps of the pipeline, leaving it to the last step. The application will begin at your stateroom. You will be inside a body that does not react to any of your movements, there is no tracking at all but, you will be able to walk pressing a button from your hand device, from the HTC Vive.

There are different avatars on the ship, on different floors, and some decoration, such as a pianist and some people listening to him. The ship is divided

by floors, therefore, we can just focus on the user's current floor to avoid unnecessary computation and also to increase the activity on your current floor, with details such as increasing the number of avatars on current floor or opening all the room's doors of that floor. In order to accomplish that, we needed to check whether the user changed floor or not and we had to track the user's movement to see if he went upstairs or downstairs at any moment. There was an easier way to trace the user's floor change, this is, detecting whether the user stepped into any stairs and stepped out on a different floor. That could be achieved adding triggers on every stairs, at the top and the bottom so, when the user gets into any of these triggers, we save the Y coordinate and then, when he gets into another of these triggers, we check whether the Y coordinate changed or not. This is done to make sure the user is not getting in and out of the same trigger and also to know the floor change, this is, if the user went up or down. That system also helped to boost the performance of the application and will be explained with more detail on next chapter.

3.3.1 Alarm and Beginning of Danger

At this point, we will simulate an alarm, to give feedback to the user, so he knows he is in danger. There could be some other methods such as moving another avatar to your room, and yelling you to run. There is a wide number of choices but we will stick to the alarm sound. The alarm starts beeping quite strong and then, it lowers its volume, so it does not annoy the user. The alarm does not have 3D sound effect, so you can hear it from all the ship, as if you had a lot of speakers all around the ship.

When the alarm starts, the panic begins on all the avatars, who try to find one safe zone, randomly, and they switch from idle or walk states to running scared states, trying to avoid collisions and other obstacles. At the same time than the alarm begins beeping, the floors from the doors of your current floor get opened. They get rotated 90 degrees to stay open and let any avatar who could be inside that room, to leave.

Avatars, when running, can collide with you. In that case, they will, sometimes, yell you one sentence, such as: “Follow me to the exit” or “The exit is this way”. This is done to analyze user's response depending on avatar's behavior.

3.3.2 Representations of Danger

The logic of the application regarding this point, is to activate, randomly, some danger spots on the ship, to alter the user and avatar's behavior. The danger spot is marked as a navigation obstacle, which makes the avatars not to be able to walk through that place, and it also has a collider, which blocks the user if he wants to go through it. These danger places or events would be also connected to the avatar's phrases or communication, so an avatar who has seen a danger place or spot, would suggest you not to go on that direction because of that. There are, on this prototype of application, two kinds of danger, one of them is fire, there could be fire in the middle of any stairs, but it will not get propagated all around. The other kind of danger we can see on the application is water, as seen on **Figure 24**.

Figure 24: Water filling the lowest floo

The ship is supposed to be sinking, therefore, randomly, the lowest floor could be filled with water and, consequently, no avatars will come from that floor, thanks to the programmed scripts.

4. OPTIMIZATIONS

4 Optimizations

4.1 Preamble

The application had to be run on any virtual reality system. We had different options to choose but, once we saw the size of the application, not only because of the total amount of meshes and polygons but also because of the big amount of computation needed to perform on runtime, we discarded Gear Vr system, leaving the choice between Oculus Rift and HTC Vive. Both run with a desktop computer doing all the computation thus we decided to use HTC Vive because it can run up to 90 FPS instead of Oculus's 60 FPS. From there, we could have taken the best current desktop computer to run the application and it would have worked but the purpose of this project, aside from running an experiment to measure the user's behavior regarding the environment, was to get it runnable on average computers.

From this preamble, we had been profiling the application and determining the most efficient ways of optimizing it. The way we optimized it requires to know a few concepts:

- **Draw call:** To draw a Game Object on the screen, the engine has to issue a draw call to the graphics API (such as OpenGL or Direct3D). Draw calls are often resource-intensive, with the graphics API doing significant work for every draw call, causing performance overhead on the CPU side.
- **Batches:** A batch is a group of draw calls to be drawn together. Batching objects to be drawn together, minimizes the state changes needed to draw each object inside the batch. This is turn leads to improved performance by reducing the CPU cost of rendering the objects. There are two kinds of batching.
  - Dynamic batching: For small enough Meshes, this transforms their vertices on the CPU, groups many similar vertices together, and draws them all in one go.
  - Static batching: Combines static (not moving) Game Objects into big Meshes, and renders them in a faster way.

- **SetPass Calls**: This is just a measure of how many times the scene needs to switch shader passes. To reduce SetPass calls, it is recommended to reuse materials or to find ways to allow Unity to batch your rendered objects.

4.2 Graphics Pipeline

This section is intended to summarize the reasons for some of the performed optimizations, because most of them rely on OpenGl's and Unity's rendering pipeline. **Figure 25** is a compressed representation of this pipeline. From there, we should remark some steps. Primitive Assembly step includes the process of "Face Culling[6]". Later on, close to the end of the pipeline, at Per-Sample Operations step, there are also a lot of tests that the incoming fragment from previous step has to pass. If it does not pass these tests, the fragment is discarded, thus it is not painted. Some of this tests, which have relation to some of the optimizations, are the following:

- **Scissor Test:** When enabled, the test fails if the fragment's pixel lies outside of a specified rectangle of the screen.
- **Depth Test:** When enabled, the test fails if there was another fragment on top of it with a value of depth showing it closer. There is a depth buffer to check it.

The rest of the steps are useful on the pipeline but they do not have any remarkable influence on our optimizations. They are shown, mainly to give an idea of how many processing is applied to every vertex thus the need to reduce the number of vertices, even if it is only temporally.

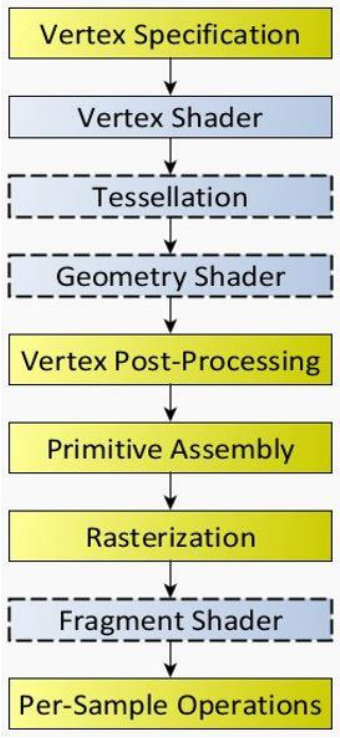

Figure 25: Diagram of the rendering pipeline. The blue boxes are programmable shader stages

[6] Face Culling: Triangle primitives can be culled (I.E.: discarded without rendering) based on the triangle's facing in window space. This allows you to avoid rendering triangles facing away from the viewer.

4.3 Summary of Optimizations

Crowd – Pedestrians

The crowd simulation implies a lot of computation. It has relationship with different components of the application:

- **AI:** The relation here is very tight because every avatar needs a small AI to decide where to go and what to do. To choose its next position and to get there, he needs to use the navigation mesh and Unity's native path-finding. Therefore, if the avatar is not going to be seen, we should not be using computational time on this avatar.
- **Rigging:** Every avatar is being animated every frame. That means, every vertex is changing its position depending on the avatar's bones. Every avatar has an average of 33 bones, except for the user's avatar who has 51 bones. To get every vertex position that belongs to every avatar, Unity has to multiply every bone's transform (this includes position, rotation and scale) per vertex. For performance reasons, every vertex is only affected by 4 bones, that is, 4 matrix multiplications to know every vertex positions. In summary, these are a lot of calculations for every frame.

Because of the previous reasons, we can disable all the avatars except those who are on the same floor than the user. That would be a bit weird because you would not be able to see any avatar coming from any lower or higher floor. Therefore, we decided to enable avatars in range of +1 and -1 floors. Thanks to the piece of the script seen on **Figure 26**, the attribute “Height Working Range” let's us parametrize that. When the application starts, the X (minimum value of Y coordinate) and Y (maximum value of Y coordinate) are set to the proper values of floors +1 and -1. Every time the user changes floor, those values are updated.

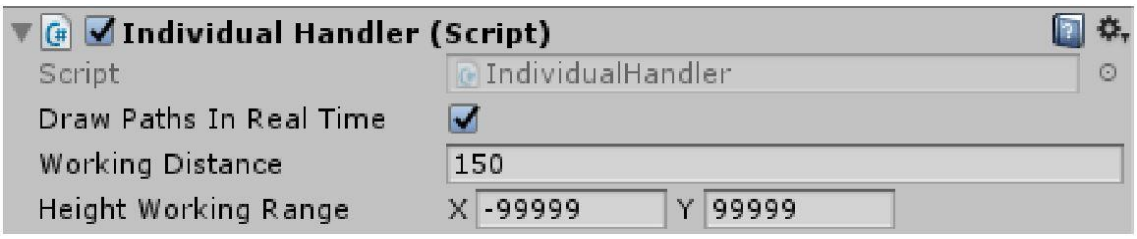

Figure 26: Piece of the script to handle avatars

Regarding crowd optimization, there is another optimization done. This one is simpler. We can also configure how many avatars update per frame. This is, if we have, for instance, 50 avatars and we update 10 per frame, avatars will calculate a path and keep moving or whatever they were doing for the next 4 frames. Then, we will calculate again the path to see whether there is an obstacle in the path or some other thing that changes avatar's behavior.

Lightning Performance

In a standard forward rendering pipeline, the lighting calculations have to be performed on every vertex and on every fragment in the visible scene, for every light in the scene.

If you have a scene with 100 geometries, and each geometry has 1,000 vertices, then you might have around 100.000 polygons (a very rough estimate). Video cards can handle this pretty easily. But when those polygons get sent to the fragment shader, that's where the expensive lighting calculations happen and the real slowdown can occur.

The expensive lighting calculations have to execute for each visible fragment of every polygon on the screen, regardless if it overlaps or is hidden by another polygon's fragments. If your screen has a resolution of 1024x768 (which is, by all means, not very high-res) you have nearly 800.000 pixels that need to be rendered. You could easily reach a million fragment operations every frame. Also, many of the fragments will never make it to the screen because they were removed with depth testing, and thus the lighting calculation was wasted on them.

If you have a million of those fragments and suddenly you have to render that scene again for each light, you have jumped to [number of lights] x 1.000.000 fragment operations per frame! Imagine if you had a town full of street lights where each one is a point-light source...

The formula for estimating this forward rendering complexity can be written, in big O notation, as O (number_geometry_fragments * number_lights). You can see here that the complexity is directly related to the number of geometries and number of lights.

Previous calculations work for forward rendering. There is another approach which could solve this performance problem, deferred rendering. The complexity of

deferred rendering, in big O notation, is: O (screen_resolution * number_lights). The way it work is, every geometry is rendered, but without light shading, to several screen space buffers using multiple render targets. In particular, the depth, the normals, and the color are all written to separate buffers (images). These buffers are then combined to provide enough information for each light to light the pixels. You can see a comparison on **Figure 27**.

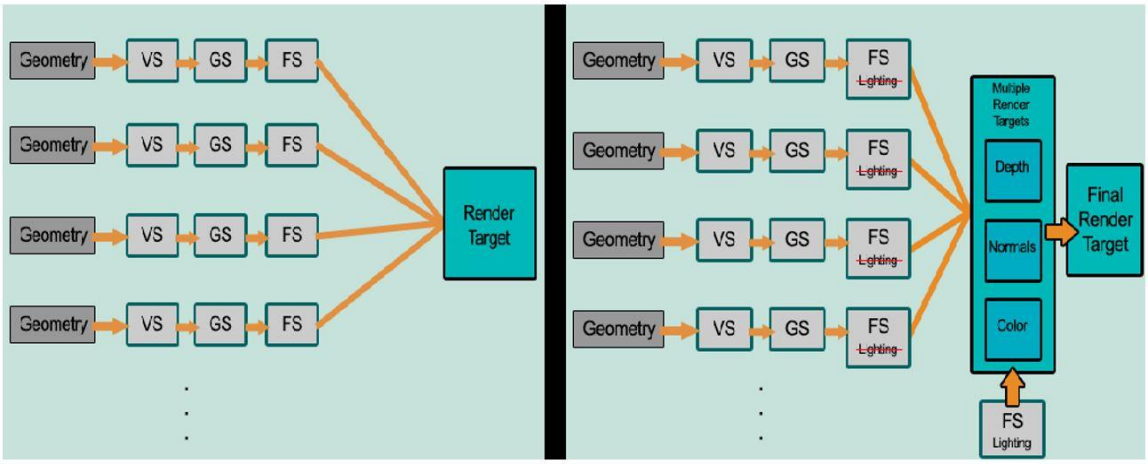

Figure 27: Forward rendering on the left and deferred rendering on the right

Deferred rendering is not always usable, though, because it requires a powerful graphics card with enough memory to store, on buffers, all the geometry. Therefore, in this application it is not convenient. There is a third approach which is to bake the light. To get the lightning effects we can create light maps for all the geometry and then, use them on runtime. The advantage of using light maps is that the computational cost is negligible, it takes the color from the light map as it would do from a texture. The problem comes when you have to mix dynamic objects with static objects. Light maps work only with static objects but they do not work casting shadows between dynamic and static objects because they can not predict which will be the lightning at that point. There is a workaround which is to have mixed lights. We have light maps for static objects but one real time light that cast shadows. It is not the most realistic approach but gives a big boost on performance as we can see on **Figure 28** and **Figure 29**. The first one shows the number of batches and SetPass calls of the real time lightning with forward rendering, and the last one shows the same stats with forward rendering with one real time light but all the other lights, baked.

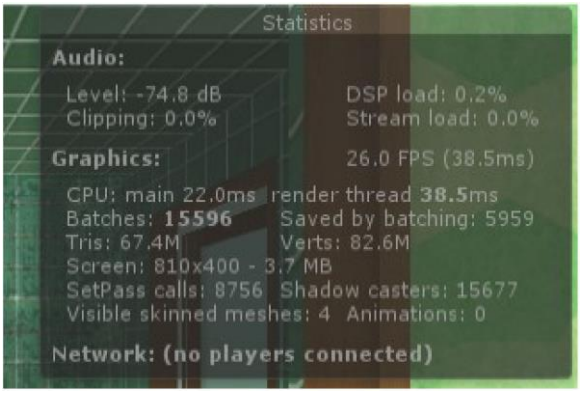

Figure 28: Performance before baking lights

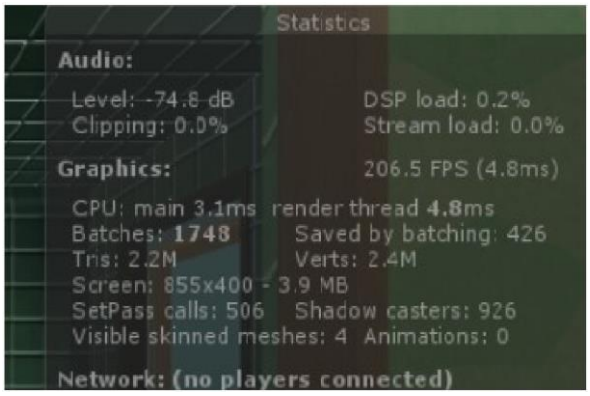

Figure 29: Performance with baked lights

Occlusion Algorithm

The decision of making an occlusion algorithm became when we analyzed the environment structure. This is a floor based design, with large floors but also a lot of levels. From every level, you can only see the immediate upper and lower floors. Taking this information and graphics pipeline into account, we decided to disable ship parts depending on user's floor. This is, if user is at 4th floor, the only active floors will be 5th and 3rd, reducing this way, the number of vertices to process in the pipeline. We have a comparison of the number of batches and SetPass calls between occlusion script activated and disabled, on **Figure 29** and **Figure 30**. We could improve even more the algorithm, disabling parts of the current floor depending on the user's position, but it would increase the complexity of the algorithm too much for this project and, the already existing optimization in combination with the other ones, will be enough to run the application.

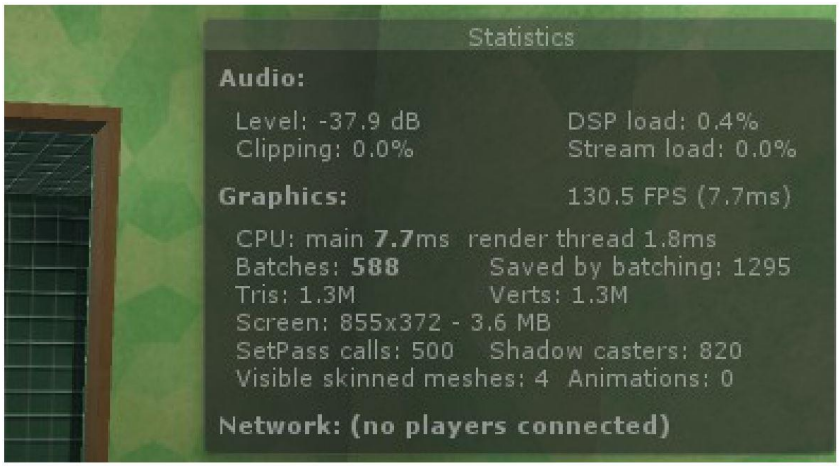

Figure 30: Performance on runtime, with baked lights and own occlusion gorithm

Hammocks on Navigation Mesh

This optimization is quite simple but helped to reduce the complexity of the navigation mesh. As we can see on **Figure 31**, there are much less black lines than on **Figure 32**.

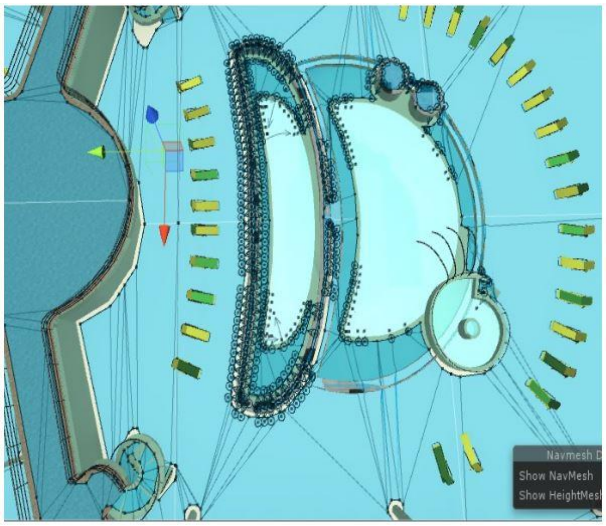

Figure 31: Navigation mesh without hammocks included

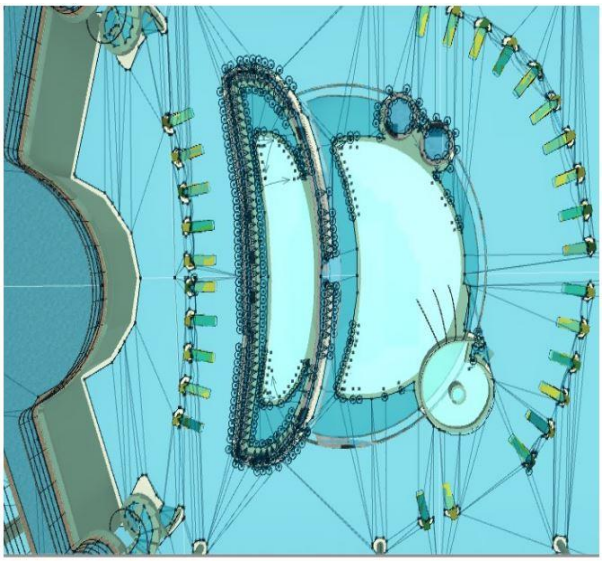

Figure 32: Navigation mesh with hammocks included

Black lines represent the triangles on the navigation mesh, used to predict the walkable surfaces. The thing is, there are a lot of hammocks on this ship and they are walkable, as the ground is, and they are always on top of floor surface, therefore, they should be always walkable. Removing them before performing the bake and enabling them right after, makes the baking algorithm not to take them into account, leading to a reduction in complexity.

Backface-Culling

Backface-culling algorithm is used to avoid painting hidden faces. By default, the standard shader applied to the materials in Unity, have it activated but, it is not like the backface-culling from graphics pipeline, which processes the fragment and then decides whether to paint it or not, the backface-culling implemented on Unity's default shader works independently of the user's camera. This one only draws the mesh if you are seeing it in the normal's direction. The problem comes when you have to place walls for the rooms, because you need to see those walls from both sides, but the shader just draws the face in one direction. To avoid this optimization on some cases, we had to rewrite the shader and apply it in some cases. The only needed line was this one: “Backface Off”. We could see an example on **Figure 33**.

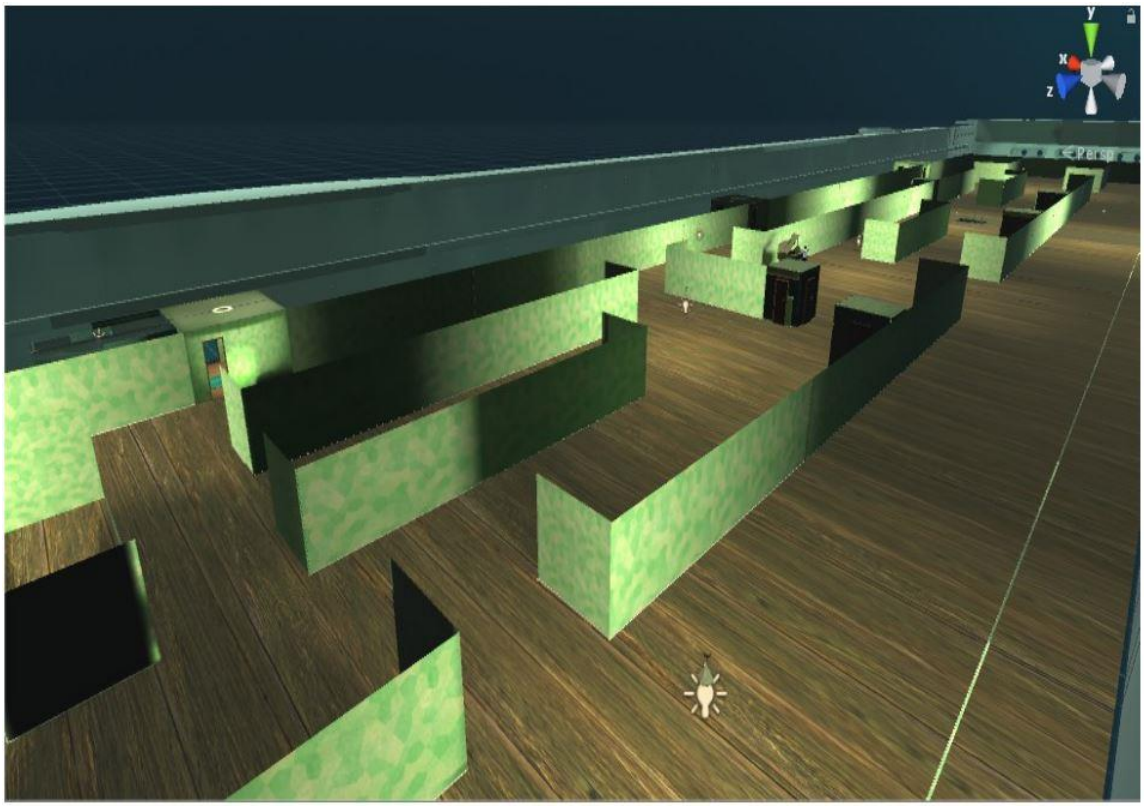

Figure 33: Example of backface-culling

We are only seeing the walls whose normal is pointing to the camera but we do not see the other walls.

Visual Optimizations

At this point, we have to think about how big a ship can be. That means, a lot of floors and staterooms, as well as bathrooms, furniture, pictures and all kind of common decoration. Having this statement into consideration, we have to count how many polygons a room could have, including all kind of furniture and decoration. For example, the main room of the application, the one with bedroom and bathroom, has around 4 rectangular walls per room, 4 rectangular pieces for the roof per room as well as doors, door joints and window. Two out of those 4 walls per room, have even more triangles and vertices than the rest because they need to leave a hole for the door like the one of **Figure 34**.

Figure 34: Wall with door hole.

Each one of those rectangles has an average of 50 triangles, taking into account the special squares. Then, we have to include door's triangles and all the furniture such as lamps, apples, night tables and bed. Then, we have the shower, water and all the typical furniture from bathrooms such as closets. Adding all these vertices, it gives a sum of more than 3.500 vertices. This problem made us think about which is the goal of the application's user when we decided to reduce the number of details on most of the rooms.

The goal is to get to a safe zone, therefore, users will not spend too much time inside any room as there will not be any way out. Therefore, the optimization consists on removing all the bathrooms from all the rooms except for the main room. To simulate a bathroom on all other rooms, there is a closed door inside each room. That way, it looks like there could be the bathroom but there is nothing, actually. A similar optimization was performed in a bigger scale. This is, there are some furnished rooms and some others which are empty.

The way we proceeded is the following, the empty rooms have double door, this is, you get into a very small room which has another door inside that would lead you to the real room but this second door is always closed. That is useful to let some avatars leave this fake room and let the user think there are a lot of rooms even though there are just a few real rooms. We left the ship with about 100 furnished rooms and 150 empty ones.

On furnished rooms without bathroom we are saving around 1.700 vertices and we are saving around 3.000 vertices on empty ones, giving it a save of around 620.000 vertices. Into visual optimizations we could include those relative to textures such as bump mapping, also called normal maps.

That consists on adding a normal texture to any material and then, adding an additional texture called normal map, which is usually drawn in black and white colors. The darker or closer to black the color is, the deeper the surface is supposed to be. That way, you can avoid creating circular (or any other shape which is not straight) thus saving a lot of vertices getting the same visual effect.

5. EXPERIMENTING

5 Experimenting

The idea of the project was to run an experiment with a limited number of users, to analyze different behaviors depending on application's setup. This experiment will be run right after the project ends, thanks to the developing made during the project's time. To run this experiment we will require at least 30 participants, because we will have three different setups, resulting into 10 different users per setup. This is a small amount but will be useful to see and analyze different behaviors depending on the application's settings. In order to compare the results between the three setups, we had to take different marks whose we can measure. Those marks are: Total distance, number of loops and total timing. First two marks depend on a concept called mind map.

5.1 Mind Map

A mind map (mindmapping.com, 2017; Inspiration Software, Inc., 2017) is a diagram used to visually organize information. A mind map is hierarchical and shows relationships among pieces of the whole. It is often created around a single concept, drawn as an image in the center of a blank page, to which associated representations of ideas such as images, words and parts of words are added. Major ideas are connected directly to the central concept, and other ideas branch out from those.

The easiest way to determine the user's mind map was to trace the user's movement while the application is running. The way to achieve that, was to add triggers on every corner of the ship, on every floor and also at the beginning and ending of every stairs, as well as at the middle of long halls. That way, every time the user stepped into any of those triggers, we keep add this number to the steps array, saving this way the user's path. **Figure 35** shows an example of the triggers location on floor 2

.

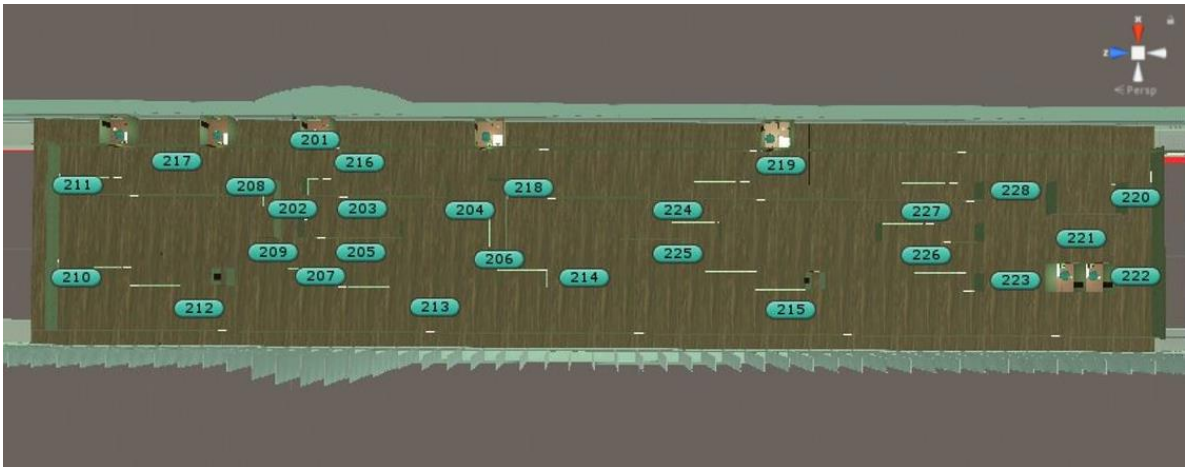

Figure 35: Mind points representation of floor 2

Thanks to this implementation of mind map, we can create a directed graph at the end of the play, knowing the total traveled distance by the user, as well as the amount of loops the user has done. With those two parameters plus the total timing, we can evaluate different users with different application's setup.

5.2 Experiment Setups

The application itself was designed and chosen to run an experiment. This experiment consist on trying different settings when evacuating the ship, to compare different timings and also to compare the resulted mind map between different executions with different settings. We have chosen three different setups for the experiment. All the experiments will begin with the user at his stateroom and the alarm beeping. The difference between the different setups are the following:

1. There will be no interaction between the user and the other avatars. You will see them walking and running when they perceive danger, but there will not be any kind of audible or visible communication between the crowd and the user.
2. On this setup, the user will be walking around the ship, as he would do on first case, but avatars will be randomly yelling something like: “Come, this is the way out” or “The exit is here!” to the user if they are close to each other. With this setup, we want to experience if the user gets influenced by the crowd or not and we want to see if the user follows those users who tell him where is the exit.

3. The third case scenario will be similar to the second one except for one thing: avatars who will be yelling things to the user when they get close, will be only those with staff's cloths, which means they belong to the ship's staff thus they are more trustable. This case was designed to analyze whether the user follows those instructions with higher chances if they come from ship's staff rather than normal people. This setup will require us to create another kind of avatar who will be wearing shining cloths, with any watermark showing that they belong to the ship's staff. To change the avatar's cloth we will use Adobe Fuse again to edit it and, to change cloth's color and to add those ship's brand watermark, we will edit the texture manually.

6. PROJECT MANAGEMENT

6 Project Management

6.1 Social Impact

The project itself does not have a direct social impact but, if we consider the objective of this project, we could say that this project can be useful for future experiments and tests to improve evacuation plans, to increase the chances of surviving on an accident like this one, which already happened in real life, where Titanic is the most known example. With the base of this project, it would be very easy to modify it or adapt it to another environment or situation, or even change the parameters like the map's locations (if any), the blocked rooms or floors, or the ship's staff behavior, to find out better evacuation plans and strategies.

About sustainability, this project is sustainable because it will not require any real environment like a big ship. All the problems like breaking the ship, will only happen virtually, therefore, there will not be any real impact on that side. It only requires hardware to run the experience and the software which is being developed by me. We could take into consideration, the effect that my hardware produces on the environment when recycling or not those items, but this discussion would be large enough for another thesis. Therefore, we only took into account the predictable facts.

6.2 Timing Planning

The estimated project duration is approximately 6 months. The project started on January 7th, 2017 and the deadline is on June, between 16th and 21st, depending on presentation date.

6.2.1 Project Iterations Planning

The main objective of this phase is to predict, as accurate as possible, the timings, so we can predict how will be looking the finished project and how many things we will be able to include on it.

The timings are hard to predict because of all the possible setbacks, therefore, the planning will consist on several iterations, this is, we will detail the basic phases

and, from there, we will keep improving all the project. The iterations, sorted by relevance, are the following:

1. **Initial setup:** This phase consists on getting a 3D model describing the environment. In this case, a 3D ship model. We also need to design all the hallways of each floor of the ship, which rooms will be visible or not (because of resources limitation, to get a proper performance).
2. **Setting up all the avatars:** The project itself consists on several avatars to generate the crowd simulation. On this phase we should get low polygon avatars (3D models representing real people, made with few triangles to increase performance). We should get or create some animations to simulate the movement of those avatars.
3. **Basic crowd simulation:** On this phase we should start creating a basic algorithm to give each avatar a basic artificial intelligence to avoid colliding with other avatars, walls and so on (also called Path Finding algorithm). We will also create an algorithm that enables or disables different floors and other visual elements of the ship, depending on your location, to improve the performance, reducing the amount of geometry the hardware has to process.
4. **Movement:** At this point we will implement the user's movement. You will be able to move using a small joystick and looking around. We will implement the “mind map” representation to store the user's mental map when doing the simulation.
5. **Adding floors:** Once all the previous steps are complete, we will add at least two more floors (from the start one to the surface), and adapt the previous algorithms for this improvement, so avatars will be able to go upstairs and downstairs as well as the user.
6. **Background history:** This step is about adding sense to the application. This is, from the beginning, you will be explained how to move (kind of tutorial inside the application), then you will be able to move around with another avatar which will be supposed to be your friend. Then, at some point, an alarm will start beeping and the

simulation will begin. We will check real evacuation process to make it more realistic.

7. **Finishing all details:** This is a long step but it's only important and related to how beautiful the application will get. This is, adding all the remaining floors, adding some good looking textures to the walls, roof and floor (images to simulate the wood of the ground or the walls of the ship).

6.2.2 Estimated Time per Iteration

Planning, writing and checking correctness	60h
Iteration 1	60h
Iteration 2	35h
Iteration 3	90h
Iteration 4	45h
Iteration 5	25h
Iteration 6	45h
Iteration 7	100h
Running some tests, debugging and getting everything ready	70h

Total: 530 Hours.

6.2.3 Possible Obstacles and Workarounds

We might find problems when building the application. Those problems might be related to different points. One of the main potential problems is the efficiency problem. The simulation we want to build relies on a lot of resources and Samsung S7 Edge (Samsung Electronics, 2016) in combination of Gear VR might not be enough to run such experiment. The possible solutions in this scenario would consist on reducing the complexity of the application to make it lighter. This reduction could be, for example, reduce the total polygons of the scene or the number of simulated avatars. Other possible solution to avoid these reductions could be switching to another platform, like Oculus Rift or HTC Vive.

There might appear bugs too but thanks to the current debugging tools, there should not be any bug that slows me down for too long. If it happened, we would try to find a workaround. Of course, there will be some problems that will make me lose more time than expected, if that problem gives me a very big delay, thanks to the iterations way of working, we could take some hours from the last iteration and use them in previous iterations for fixing problems. The way we have ordered the priorities, it means that the project would not be that perfect but it would be complete.

6.3 Development Tools

This project needs several tools. The selected tools are not the only available to accomplish what we wanted or needed but, after evaluating different softwares, the following are my choices:

- **Unity3D:** This is the main tool. Most part of the project is going to be developed with this platform. We had different game engines such as Unreal Engine or CryEngine (Epic Games, Inc, 2017; Crytek GmbH, 2017), to compare and, actually, unity is not considered the most efficient but, this is the one we feel most comfortable with and, thanks to the optimizations that we will be applying, it should be efficient enough.
- **Blender:** This is the modeling software. It is used to model the ship, as well as dividing the ship parts into smaller pieces and creating small structures or items to decorate the ship, for instance, the stairs that connect the different floors. This modeling software is probably not the best one but, compared to other software such as Maya or 3dMax, which are not free (although they have free version for students), this one is free and Open Software, which makes it more attractive.
- **Visual Studio 2015:** This tool helps me writing and compiling scripts for the application. It is the most recommended code editor to work with unity, because of the compiler and the debugger (Microsoft Corporation, 2017).
- **Adobe Fuse and Mixamo:** These tools are going to be very useful to get rigged avatars and animations. They are free, yet, and very easy to use.

- **Github:** This tool will help me to keep the project safe and always accessible (Github, 2017).

6.4 Budget Monitoring

6.4.1 Hardware and Software Budget

The required or used hardware in order to develop this project is the following: Hardware:

- Samsung Galaxy S7 Edge – 590€
- Gear VR – 40€
- Oculus Rift / HTC Vive – 700€ for the first, 900€ for the second
- Desktop Computer with the following components (plus the basics, mother board, monitor...) – 1650€
  - Processor: Intel i7-6700
  - Graphics card: Gigabyte GTX 1070 with 8GB dedicated
  - RAM: 2x 8GB 2133Mhz, DDR4
  - SSD Disc: Kingston UV400

Software:

- Visual Studio 2015 – Free (student version)
- Unity3D – Free (if less than 200.000$ of income per year)
- Android SDK – Free
- Oculus SDK – Free
- Adobe Fuse – Free
- Blender – Free
- Github account – Free
- Open Apache Office – Free (The Apache Software Foundation, 2017)

6.4.2 Human Resources Budget

For the human resources economic management, we will evaluate the amount of money paid, in average, to a computer science student. We will also take into account the expected project length, which is 530h. The average payment for an employee who is about to finish his degree, could be 18€/h[7] (12€ of income for the employee plus 6 of cost to the enterprise for Social Security fees), therefore, we have this calculation: 530 * 18 = 9540€.

6.5 Relation to Computation Branch

6.5.1 Relation to Computer Science

The project is based on graphics and efficiency. We had to evaluate different hardware systems to decide which one could fit better for our purpose. We also had to evaluate the complexity of adding different components such as path-finding, 3D sounds, and some other features, to decide whether it was viable or not to add them. We also had to choose between different game engines such as Unity3D or Unreal Engine, depending on their features and efficiency. There is a huge part of the project that relies on AI, from avatar's behavior to algorithms to improve performance such as the occlusion algorithm. It also has an important dependence of graphics and 3D modeling, as the project itself is visual, more specifically, Virtual Reality. We also had to analyze the objective hardware to use it when implementing some of the application's features, such as deciding whether to use forward or deferred rendering, as it depends on the graphics card. Overall, this project was possible thanks to the knowledge acquired on computer science degree.

[7] Source: http://www.pagepersonnel.es/sites/pagepersonnel.es/files/er_tecnologia16.pdf

6.5.2 Related Competences

- **CCO1.1:** “Avaluar la complexitat computacional d'un problema, conèixer estratègies algorísmiques que puguin dur a la seva resolució, i recomanar, desenvolupar i implementar la que garanteixi el millor rendiment d'acord amb els requisits establerts [Bastant]”.
- **CCO1.2:** “Demostrar coneixement dels fonaments teòrics dels llenguatges de programació i les tècniques de processament lèxic, sintàctic i semàntic associades, i saber aplicar-les per a la creació, el disseny i el processament de llenguatges [Bastant]”.
- **CCO1.3:** “Definir, avaluar i seleccionar plataformes de desenvolupament i producció hardware i software per al desenvolupament d'aplicacions i serveis informàtics de diversa complexitat [Una mica]”.
- **CCO2.1:** “Demostrar coneixement dels fonaments, dels paradigmes i de les tècniques pròpies dels sistemes intel·ligents, i analitzar, dissenyar i construir sistemes, serveis i aplicacions informàtiques que utilitzin aquestes tècniques en qualsevol àmbit d'aplicació [Bastant]”.
- **CCO2.2:** “Capacitat per a adquirir, obtenir, formalitzar i representar el coneixement humà d'una forma computable per a la resolució de problemes mitjançant un sistema informàtic en qualsevol àmbit d'aplicació, particularment en els que estan relacionats amb aspectes de computació, percepció i actuació en ambients o entorns intel·ligents [En profunditat]”.
- **CCO2.6:** “Dissenyar i implementar aplicacions gràfiques, de realitat virtual, de realitat augmentada i videojocs [En profunditat]”.
- **CCO3.1:** “Implementar codi crític seguint criteris de temps d'execució, eficiència i seguretat [En profunditat]”.
- **CCO3.2:** “Programar considerant l'arquitectura hardware, tant en asemblador com en alt nivell [Bastant]”.

7. SUMMARY AND RECOMENDATIONS FOR STUDIES

7 Summary and Recommendations for Further Studies

The project can be considered complete after all the work done on it. To summarize it, we could say that the experiment is ready to run once we find the users and it was efficient enough to run it at a decent frame rate on HTC Vive. It required some optimizations to make it runnable as it was running very slow without any of the optimizations.

This project helped a lot to study Unity3D programming system and 3D in general. We found some difficulties optimizing the lights because Unity3D released a new system to bake lights on their latest version. This is called progressive baking. Initially, the baking of the lights was very slow even on a desktop computer with latest available hardware but it was worth it as we found out that lightning was the most noticeable optimization, this is, the one that increased the most the frames per second, as well as the one that lowered the most the number of draw calls.

We had some other issues implementing scripts and programming all the triggers, but we found easy ways to fix the problems or to find a workaround, making it possible to finish the project on time.

On the other hand, we could have implemented some other impressive features such as to simulate boarding before the alarm begins or better communication between avatars and the user.

Some other realistic features that will take the user closer to the reality, would be to simulate some furniture floating, and water raising while the time passes.

7.1 Future Work

Future work would be related to improve the realism of the application, increase the interaction with the avatars and also to create the embodiment effect with body tracking. There are already some labs doing research on this field, to make the user feel the virtual reality more real, but that topic was out of our scope as it would require much more time to study human reactions with different kind of embodiments.

Another point to improve would be to create a story before the danger begins. That way, the user would be capable of getting used to the environment before he has to find the way out. Finally, to make more profitable experiments, the number of different setups for the experiment should grow, adding different parameters and different marks to compare each setup and execution.

This project is useful to study questions like “Do you belief in other people?”. But, it can be also the beginning of any other project to study several questions such as:

- At what point do you perceive the danger? When Do you realize there’s an emergency situation?
- How long does a human to realize he has to leave the place because there is a real danger?
- What is the average first reaction when the alarm sounds?
- Do you react according to your mental map? Are you influenced by noises?

The overall is that the project was successful even though we had not run the experiment yet, because we will be able to study and analyze people's behavior once we run it.

7.2 Personal and Academic Endings

This project helped me to work on a current topic. That way, I could see from own point of view the gratification of creating a personal and unique project, where the obtained results will become part of scientific knowledge in our society.

On the other hand, this project will be useful to study different evacuation plans, making it possible to improve the current system thus to make it safer.

This project also helped me to find out how it feels to work on a research project and, even if it is not close enough to a PhD, that helped me to get a small idea of it.

Overall, I can say that my assessment of the project was positive and it was satisfying to work on that.

8. REFERENCES

8 References

Adobe Systems Incorporated. (2017). *Mixamo*. Obtained at https://www.mixamo.com/

Adobe Systems Incorporated. (2017). *Mixamo*. Obtained at https://www.mixamo.com/fuse

Blender Foundation. (2017). *Blender*. Obtained at https://www.blender.org/foundation/

Burdea, G. (1996). Haptic Feedback for Virtual Reality. *The State University of New Jersey*, 1-11.

CadNav.com. (2017). *3DCadNav*. Obtained at http://www.cadnav.com/3d-models/model-26583.html

Carlin, A. (February of 1997). Virtual reality and tactile augmentation in the treatment of spider phobia: a case report. *Behaviour Research and Therapy, 35*(2), 153-158.

Crytek GmbH. (2017). *CryEngine*. Obtained at https://www.cryengine.com/

Epic Games, Inc. (2017). *Unreal Engine*. Obtained at https://www.unrealengine.com/what-is-unreal-engine-4

Giga-Byte Technology Co., Ltd. (2017). *Gigabyte*. Obtained at https://www.gigabyte.com/Graphics-Card/GV-N1070G1-GAMING-8GD#kf

Github. (2017). *Github*. Obtained at https://github.com/

HTC Corporation. (2017). *HTC*. Obtained at https://www.vive.com/eu/product/

Inspiration Software, Inc. (2017). *Inspiration Software*. Obtained at http://www.inspiration.com/visual-learning/mind-mapping

Intel Corporation. (2017). *Intel*. Obtained at https://www.intel.es/content/www/es/es/products/processors/core/i7-processors.html

Kingston Technology Europe Co LLP. (2017). Obtained at http://www.kingston.com/es/ssd/consumer/suv400s3

LOOXIS GmbH. (2017). *Looxis Faceworx*. Obtained at http://www.looxis.de/de/looxis-faceworx-tool

Microsoft Corporation. (2017). *Visual Studio*. Obtained at https://www.visualstudio.com/es/

Mindmapping.com. (2017). *Mindmapping*. Obtained at http://www.mindmapping.com/mind-map.php

Nurislamovich Latypov, N., & Nurislamovich Latypov, N. (december of 1999). Obtained at https://www.google.com/patents/US6005548

Oculus VR, LLC. (2017). *Oculus Rift*. Obtained at https://www.oculus.com/rift/

Pelechano Gómez, N. (2006). *Modeling realistic high density autonomous agent crowd movement: social forces, communication, roles and psychological influences.* Pennsylvania: University of Pensnsylvania.

Pelechano, N., & Badler, N. (January of 2006). *Modeling Crowd and Trained Leader Behavior.* Obtained at http://repository.upenn.edu/cgi/viewcontent.cgi?article=1288&context=cis_papers

Reynolds, C. (1997). *Red3D*. Obtained at http://www.red3d.com/cwr/steer/

Reynolds, C. (2001). *Red3D*. Obtained at http://www.red3d.com/cwr/boids/

Robertson, A. (June of 2014). *The Verge.* Obtained at https://www.theverge.com/2014/6/13/5805628/at-e3-virtual-reality-goes-beyond-goggles

Rubin, J., & Crockett, R. (2012). *AXONVR*. Obtained at http://axonvr.com/

Samsung Electronics. (2016). *Samsung*. Obtained at http://www.samsung.com/es/smartphones/galaxy-s7-g930f/SM-G930FZKAPHE/

Stone, R. (2001). *Haptic feedback: a brief history from telepresence to virtual reality.* (S. Brewster, & R. Murray-Smith, Edits.) Berlin.

The Apache Software Foundation. (2017). *Apache Open Office*. Obtained at https://www.openoffice.org/

The GIMP Team. (2017). Obtained at https://www.gimp.org/

United States Code sections. (2016). Emergency Evacuation Plan. En *Navigation and Navigable Waters* (págs. 140-146). United States: DEPARTMENT OF HOMELAND SECURITY, Coast Guard. Obtained at https://www.law.cornell.edu/cfr/text/33/146.140

United States Departament of Labor. (2017). *Occupational Safety and Health Administration.* Obtained at https://www.osha.gov/SLTC/etools/evacuation/evac.html

Unity Technologies. (2017). *Unity 3D*. Obtained at https://unity3d.com/es

van der Meijden, O., & Schijven, M. (2009). The value of haptic feedback in conventional and robot-assisted minimal invasive surgery and virtual reality training: a current review. *Surgical Endoscopy*, 23: 1180.