# CS32310 Assignment

“The Hotel”

## Introduction

For this assignment, I decided I wanted to branch out from what I knew so many people would be doing. That’s why my submission is actually a game; albeit a horror game. I have designed the game in such a way that immerses the player into the scene, ensuring that tension rises as one plays on. This game actually has three different stages. Unfortunately I was not able to complete all stages by the deadline, however I am going to continue the development of the game as I have found Three.js inspirational for the production of advanced graphics. Within this report, I will elaborate on the procedures and techniques used in order to achieve the end result. As the report is going to have ‘spoilers’ in the way of what happens during the game, I would appreciate if the reader would play the game first, to ensure maximum immersion and enjoyment.

## The Scene

### The Story

There are three stages to the story. In the first stage, the player wakes up in a lift that has just opened on a floor of what looks to be a hotel. As they investigate the scene, they begin to notice something is not quite right, but did they notice too late? After stage one has finished, the player wakes up again at the start of the corridor, where everything has changed subtly. As they investigate more and more they begin to discover the story of what had happened in this hotel. As they wake up a final time for stage three, they are able to see what actually happened here, and who they are.
**Please note**: As mentioned before, unfortunately I was not able to complete stages two and three before the deadline. I will however continue after submission.

### Defining the scene

The scene is created by defining a WebGLRenderer; a Javascript API used to simulate and produce computer graphics within the browser. Once this has been created, Three.js interacts with this in attempt to make editing it more manageable. After the renderer has been defined, parameters are added in order to create a shadow map which will permit lights to cast shadows on objects, and designs the shadows that are created. The graphics for the scene are then defined (more on this below), before the ‘animate’ function is called.

The ‘animate’ function is called every frame within the WebGLRenderer. After the camera has been defined, the camera and every other dynamic change that occurs during the render loop will be added here. At the end of every iteration it will call the render function again, pushing the camera and renderer variable defined before (as a WebGLRenderer) in as parameters. There are many other functions and activities occurring within this animate function for the game, however for defining the scene these are the most crucial activities performed.

## Graphics in the scene

### Textures

Every single model, mesh and texture in the scene has been created by me, either in javascript or in 3DS Max (more on this later). The only one I did not create was a royalty-free texture for my floor by Cesar Vonc (http://www.textures.com/download/floorsregular0190/14264). Room for expansion here is to get the normalised texture working too, to give this mesh some depth.

### Extremities

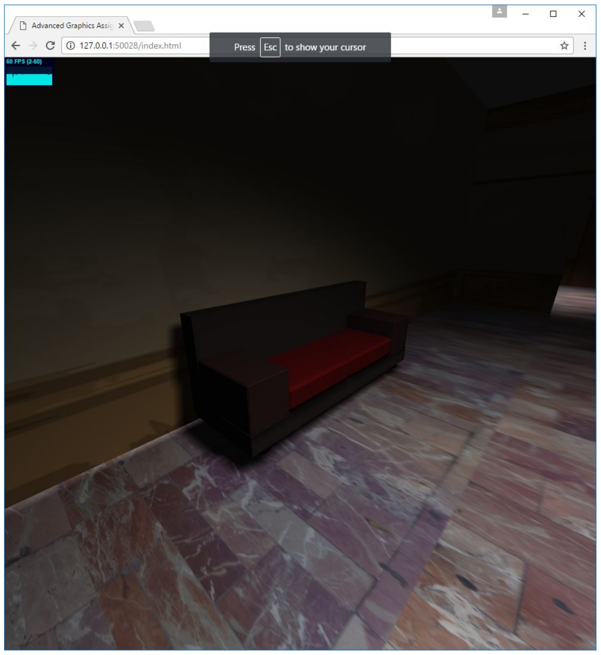
The scene is encapsulated by simple Three.js meshes acting as walls, roof and floor. These are very simple to create and demand very little from the renderer mentioned above. Ideally, these would only be used when creating anything within the scene, however there are two problems. The first is that it would take a long time to create intricate designs through Three.js only, as each curve and bend has to be added individually through Javascript, positioning it correctly each time. The second issue is that this can take a lot of code to create - which is one of the fallbacks of using Three.js to create this virtual scene. I will elaborate more on this later.

### Models

In order to create said intricate designs, I used 3DS Max to model .obj files, to which I then imported into Three.js. This was achieved through the ‘OBJLoader’ (https://threejs.org/examples/webgl_loader_obj_mtl.html) that Three.js provides, which takes the OBJ file along with the .mtl (material) file that is exported with it, and loads them into the scene as an Object3D. Once the object has been loaded in, the OBJLoader loops through all of the “child” meshes of the object, enabling the ability to perform functions on these meshes. One of these functions is to handle the shadows of the object - allow this object to both cast and receive shadows. Below is an example of the sofa object I create when it is in 3DS Max:

The textures on the models can be edited within 3DS max, with this edition being reflected within the scene (such as making a texture glossy etc).

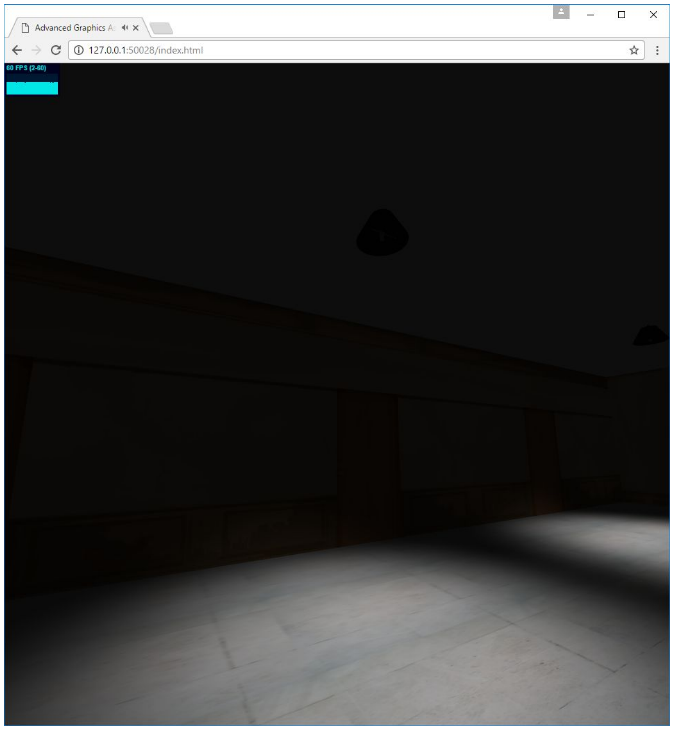
Here is the same sofa within the scene:



As you can see, the object is casting a shadow and the textures are modified by the light intensity in the area. The object also has ‘chamfered’ edges, which is easily done in 3DS Max; not so much in javascript.

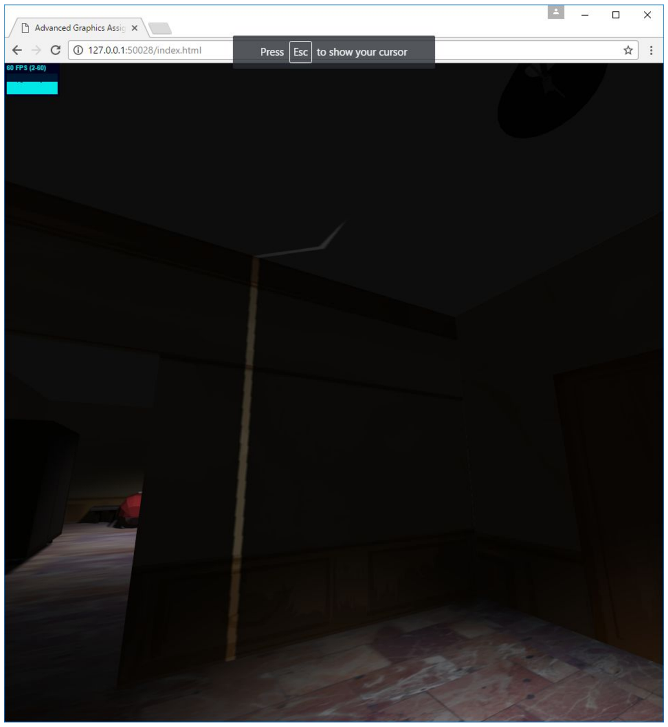
## Lights

Lights have been added with a small radius in order to create dark spots within the scene, in order to cause fear for that which can not be seen by the user. One huge fallback of Three.js that I have found, is that creating a light bulb that simulates a real life copy is almost impossible. The issue is that spotlights within the renderer do not display their source; they only cast the light in a given direction. This means that the closest one can get to simulating a real light, is to create the spotlight casting the light as normal, then create a pointlight where the bulb should be. The result is that the bulb does not dazzle the eye like a real life copy would. Below shows why this is an issue:



As you can see, the light is coming from the ceiling light model, however no source is showing.

Not only this, but it seems that after I added over 10 pointlights, this resulted in being too demanding for the average browser, creating multiple errors for lighting and crashing the browser.
In order to achieve advanced lighting, I have added a flicker near the final double doors. This flicker adds more to the immersive side of the game and, while it was originally demanding on memory resources, a simple interval on the flicker function ensured that the light was not turning on and off every frame. Instead of turning it on and off, which was very labouring for the browser, the light moves up above the ceiling and back down, imitating a flicker effect.
In the final room, there is a fallen over lamp that was modelled in 3DS Max. This has a spotlight tied to it, facing in the direction of the double doors, one of which is left ajar. The light comes through this crack, in a hope to cause interest for what is in the room.



### Ghost

The ghost was fairly simple to implement. It spawns when the cupboard door opens, and its movement is a check statement on the difference between X and Z axis of the ghost and player. Until the difference is less than two, it will continue to chase the player. The ghost was modelled in 3DS Max and imported with the OBJLoader.

## Techniques used

### Controls

In order to look around, move around and control mouse focus when in the game, I have used PointerLockControls; a javascript library provided by the creators of Three.js. I used and adapted the example code created by them (https://threejs.org/examples/misc_controls_pointerlock.html) in order to have smooth movement in my game. I implemented the sprint myself, and removed the ability to jump. Sprint was included to make the user feel like they have a chance of getting away, increasing immersion.

## Collision

My collision algorithm is where I hope to earn flair points. It is an adaptation of “Stemkoski’s” algorithm (https://stemkoski.github.io/Three.js/Collision-Detection.html), which uses ‘raycaster’ on each face of a cube to detect when the vertices have intersected any other vertices in the scene. ‘Raycaster’ is a Three.js function which protrudes a small plane to a specified distance in a specified direction.
Whenever it meets, it is possible to  Figuring out the vertices of the camera however was rather difficult, so in order to get around this, I tied a what I call a “character cube” to the camera. During the scene set up, whenever a mesh is defined, I add it to an array of objects that can be collided with. Within the animate function, raycaster is tied to every side of the character cube, and the function checks whether the vertices are overlapping by accessing the raycaster intersections.
In order to adapt it more to my game, I created a constant Three.Vector2 which stores the last position of the character that was not colliding. If the collision is detected, the position is restored, which results in a small glitch-effect, however it still works well enough to be acceptable as an algorithm.

## Interaction

Originally, I planned to use the same method for interaction as I have for collision - I planned to have a constant “Collision cube” placed in front of the player, and when this collided with objects, that object was editable. However, it proved extremely difficult to put this cube constantly in the middle of the screen even after rotation of the mouse. So instead I created a raycaster for interaction - whenever in range, an object is editable and therefore it’s movement function can be called. This is what was used for the doors.
Once an object is editable (if the player has pressed ‘E’), the function takes action depending on the object name. For example if the object name is “locked”, this triggers the “itsLocked()” function, which will played the locked sound.

## Sounds

All sounds in the game have been created by me, except the footsteps and door open/close sounds. These were royalty-free sounds acquired here:
https://www.audioblocks.com/royalty-free-audio/footsteps-sound-effects
https://www.audioblocks.com/royalty-free-audio/doors-sound-effects
The sounds are loaded and played through the THREE.AudioLoader, part of the Three.js library. The sounds are in .ogg format. Once loaded, they are applied to a positional audio loader, which allows the sounds to come from a specific place in a scene. This means that when a door is triggered, part of the door open/close function can be a trigger of the sound, which is

played after the sound positional loader is applied to that mesh. The footsteps are a little different however - they are triggered by a check in the animate function which checks whether a ‘can play’ boolean is true (in order to prevent thousands of plays per second), which then chooses a random footstep sound out of 6 and plays it. If the player is running, the ‘can play’ boolean is switched from true to false by about double time in comparison to when the character is walking.

### Minimizing Code

As mentioned before, a fallback of using Three.js to create the scene within the browser is that there is a lot of javascript that needs to be written to simulate a real scene; whether models are loaded or coded in javascript this is the case. This is not the fault of Three.js - it cannot be avoided as javascript is just what is used to create them; if anything Three.js makes it a lot easier to simulate this scene.
In order to avoid unnecessary amounts of code, for example by loading the same object more than once, I have implemented for loops wherever I need to load more than one of the same object. All objects, lights and most of the walls are loaded through these loops. The code can even be refactored more by creating a create function that takes in the parameters for creation to be done every iteration, however this is something I was not able to get round to.

## Reflection

### Regrets

While I am completely satisfied with my submission, I strongly regret not having the time to complete the final two stages; The final two stages of the game is really where the player gets to be more immersed, finding out what happened in the hotel.
I also regret not being able to make the lights more realistic, with realistic glare and light bulbs, however I think that this is simply not something Three.js have been able to work on. I’d expect that this is a feature to come in the future.

### Where I am ahead

I do believe my flair here is within the top tier of those within the module, as I made a story-based game with dynamic changes and interactions, with intricate lighting and custom adaptations of common algorithms to make my game more independent and outstanding. I show creativity in many places during this assignment, including my character simulation, my method of playing random footsteps (the velocity of which is changed depending on whether the shift key is pressed), and the creation of custom models.