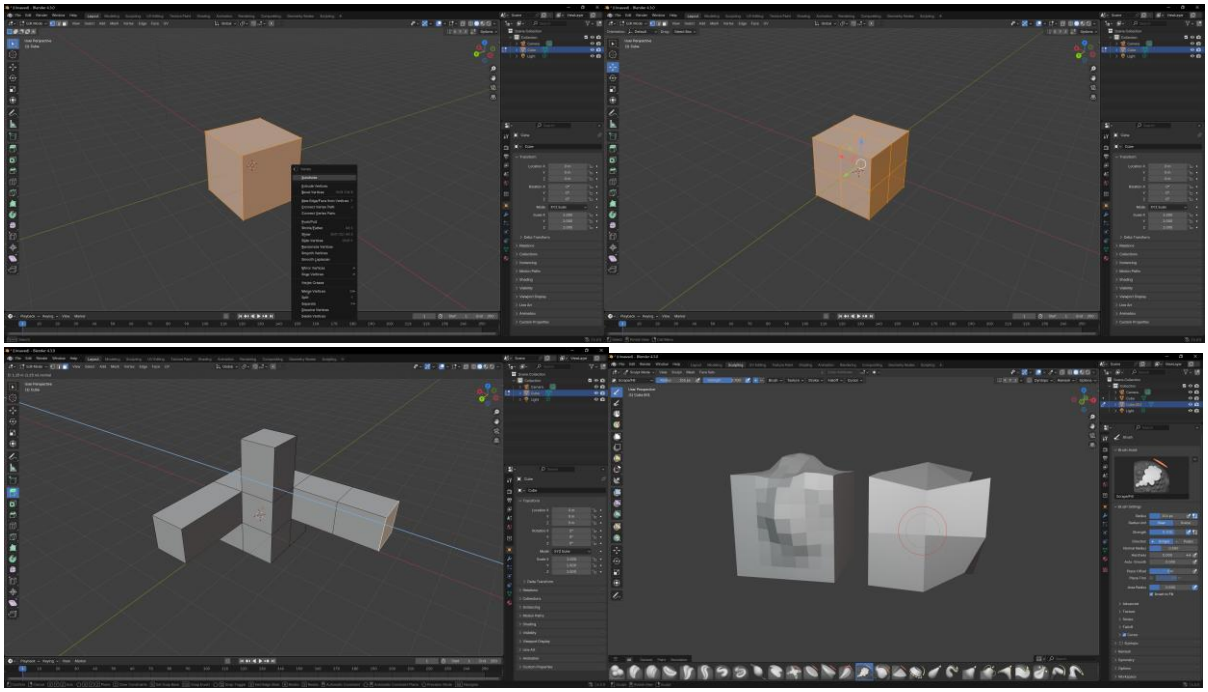
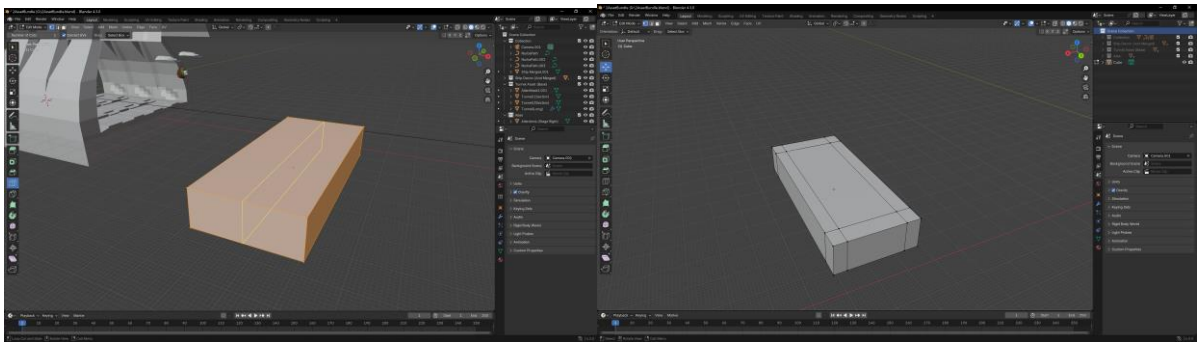
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820
Asset Design Documentation

I had never used either Unity or Blender before we began creating *Escape Velocity*. The closest experience I had was in CAD and designing art for clothing on Photoshop. The user interfaces (UIs) and software of Blender and Unity are not intuitive, but the possibilities within the programmes are endless once one starts to understand the potential for manipulation and generation within them. To create the assets, I primarily used Blender and had to periodically adjust elements within Unity to ensure compatibility. Almost all the assets I created were exported as .fdx files, which Unity can read.

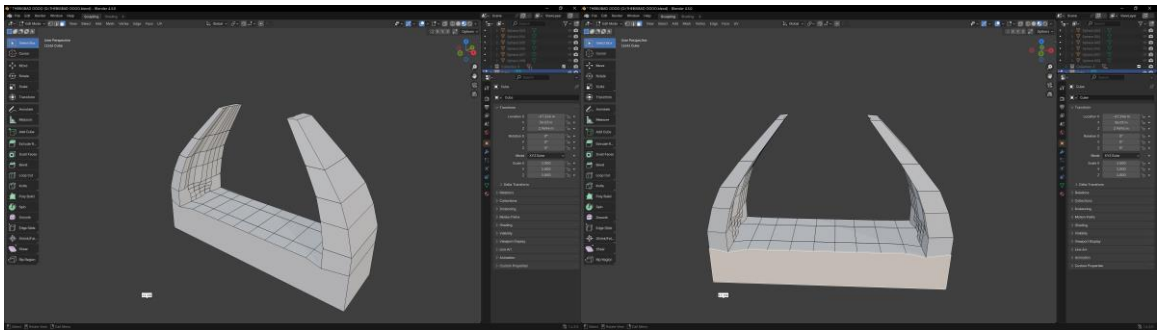


When Blender is first opened, there is only a single cube, which, at first, is quite daunting, especially when none of the interface makes sense, and no shortcuts are embedded in the mind. From this single cube, I began to experiment with many of the manipulation tools, the ones I found most interesting and useful at the time being *subdivide* and *extrude*. The former allows one to subdivide the faces of an object. If the object is a cube, this will split the face, enabling the user to manipulate the object in more abstract ways. Extrude, on the other hand, extends and inverses selected faces. With other inputs, I found later, the extrusions could be skewed and manipulated in different ways to create tunnels, etc. I also found use for the various workspace modes within Blender, such as sculpting, which allowed me to understand how specific shapes can be manipulated and how faces and vertices are integral to how a shape can be moved and altered. The alien design originated from a single ICO Sphere, where I increased the subdivision of the faces to observe how the sphere would behave with more faces to manipulate.

COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

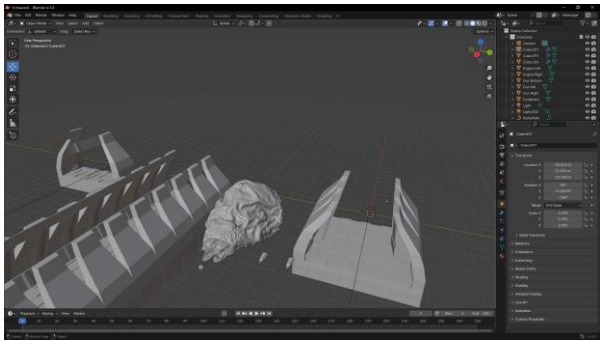
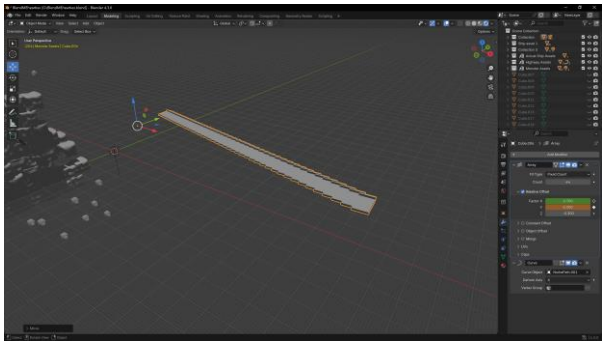


The image to the left shows a rectangle being *Loop Cut*. This adds an *Edge*, which is tied to *Faces*. It can be thought of as drawing faces without using subdivision, though both techniques have their specific uses. By using the loop cut and pressing shift (which allows the use of a metric measurement system when placing the cuts), one can place the edges specifically to create a *Tile*, which can then serve as a base asset to be further manipulated into variations. Behind the main object in the image, a more complex structure is shown, which involves more advanced techniques. The image to the right is the base asset of every tile used in the game. As stated previously, it is essentially a rectangle with loop cuts that have been placed, and the edges that were created and not used have been dissolved, so that the edges I wanted remained



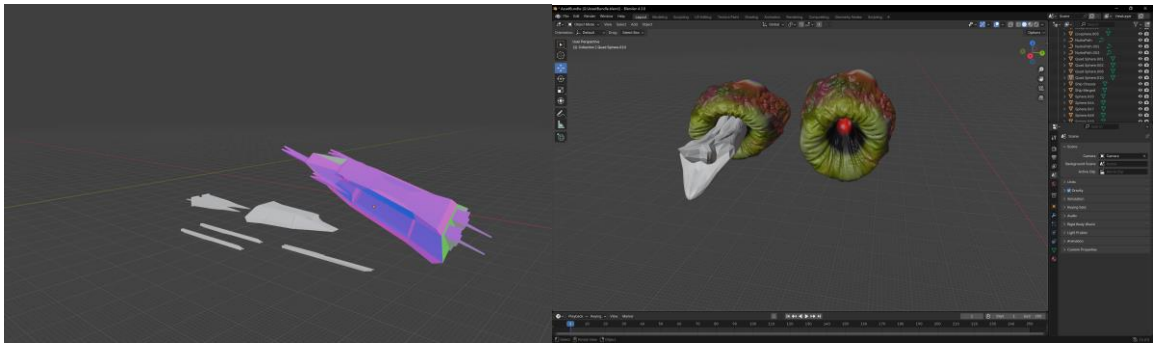
The images above depict more complex tiles designed to create a tunnel for the game, which would transition the player(s) between stages. The original plan for these tile variations and tunnels was to transform the asset and stretch them to shape, incorporating obstacles within it. However, I soon realised that there were better alternatives within Blender to achieve the desired effect, prompting me to create a more in-depth generation system for the game's level design. One issue I encountered, and have since resolved, was that while extruding the top faces for the tunnels and then skewing the new extrusions, I was ultimately making the tunnel thinner. This can be fixed by using a different method of transformation, but this effect remains useful, as it allows the extrusion of specific faces of the wall’s edge to create a new tunnel mesh, which plays off the lighting and surrounding world assets differently.

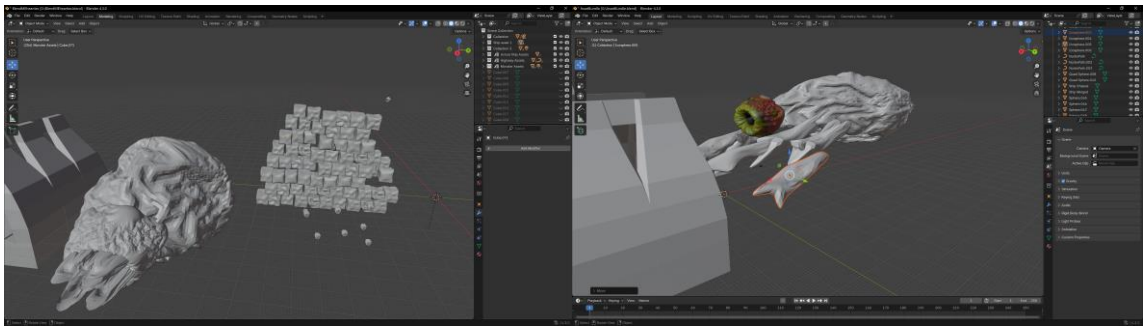
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

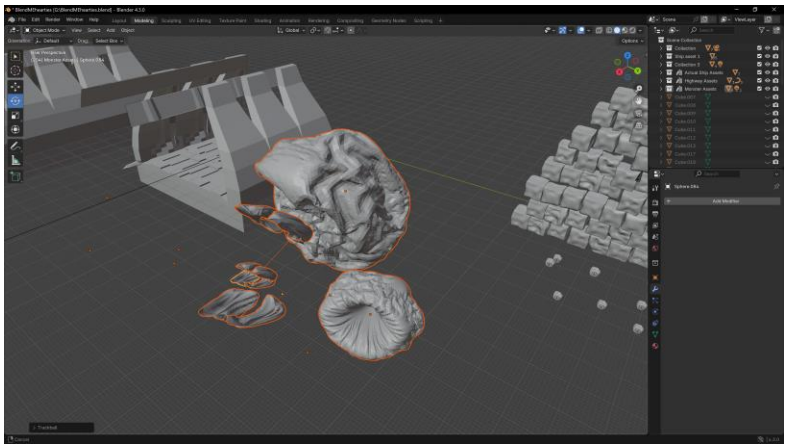



The first iteration came in the form of an *Array* modifier. These arrays can be applied to any object, and with a simple toggle of the count, the modifier duplicates the selected object and projects it with (or without) a relative offset. This is useful for the base tiles, such as the tunnel and road, as it is a streamlined process and is quite simple once understood. For reference, with an array applied to an object, it is possible to create a very atmospheric setting using only a single asset. The image below shows the same process applied to the tunnel asset. Applying the modifier is necessary for functionality, though it does not need to be applied to be visible; however, it may cause issues with meshes, etc. Thus, it is good practice to constantly back up data and use copies of assets when experimenting. These arrays can also be used as modifiers on existing objects, and can be set to paths of any length or axis (the effect works better the more “flexibility” the object has to connect to itself along the NurbsPath, *non-uniform rational B-spline*, which is achieved by having more vertices and faces to transform against itself). This allows for quick turnaround when designing a track for the game to traverse. Initially, the tunnel design was intended to remain static, with the monster itself firing projectiles and the player having to evade these attacks, while some hazards were physically placed down to fill the level. The issue arises when one wishes to add complexity to the track. The tunnel design works well for sections of *stagnation*, enabling the game to convey dialogue or explain mechanics to the player. However, when it came to adding hazards and obstacles, the process became quite time-consuming, and achieving a playable length for the track while maintaining complexity became more difficult. Moreover, the time spent would have been greater than finding an alternative solution to the issue of game design, which is where I spent most of my time researching, while also creating other assets and working on shaders and lighting systems.

COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

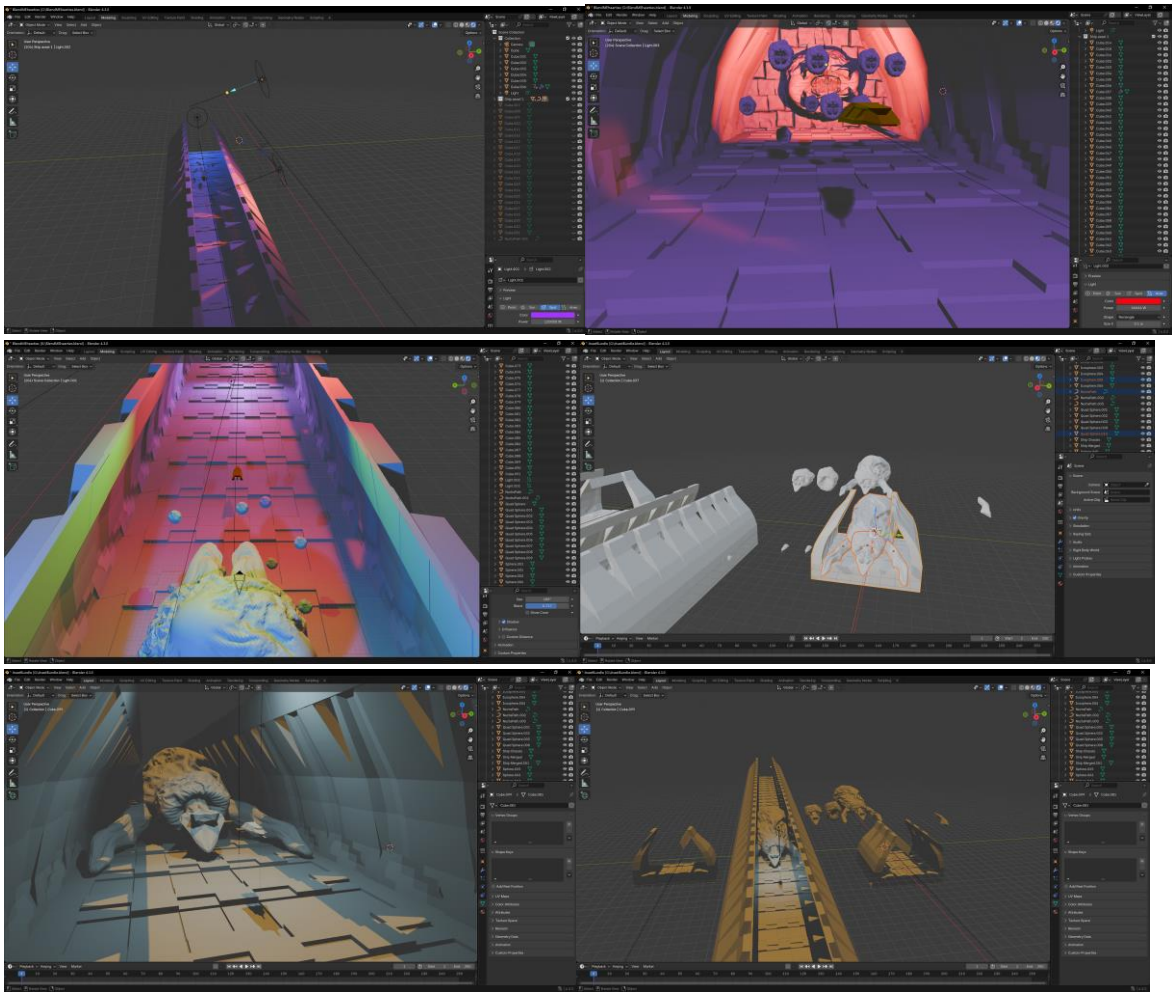






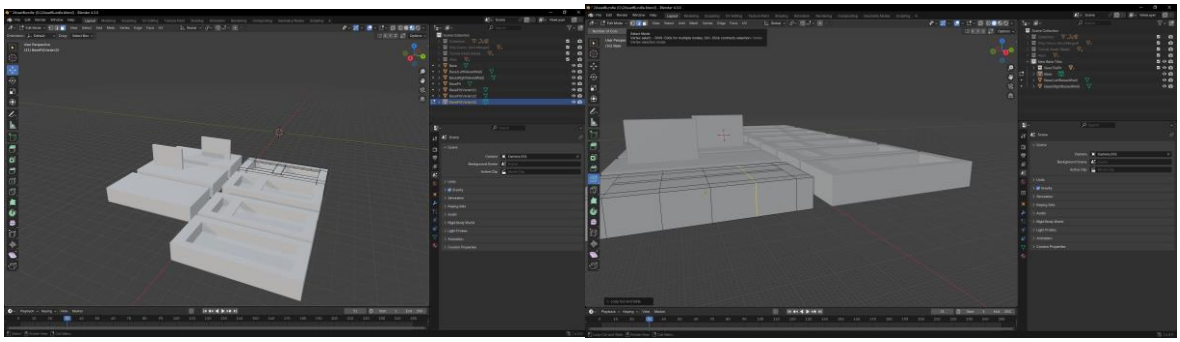
The next focus was on the player's control. *Escape Velocity* is a game where the player outruns a monster while piloting a craft, avoiding obstacles and hazards in the generated world that develops before them. The "monster" could ultimately be simulated as a moving wall; it could be a dot following the player on a set path, with a mesh placed over the top of it. One original idea I had, and partly developed, was for the "alien," which was more humanoid in shape and would be angled to "throw" collision objects from the background to the foreground. This idea was ultimately put aside, while I used the head of the first alien design to construct the near-final design of the alien that follows the player throughout the levels. One of the most frustrating parts of the entire project, and asset creation in general, is determining what is "usable," as many of the assets underwent total overhauls or downgrades from the original concept. The scope of the game had to be laser-focused, rather than attempting to include everything imaginable within *Escape Velocity*. When designing the alien, I wanted it to feel out of place, so I made everything except the monster angular and defined, lending a clean, structured feel, while the monster itself was amorphous, circular, and less rigid, like an organic mass in a sterilised environment.

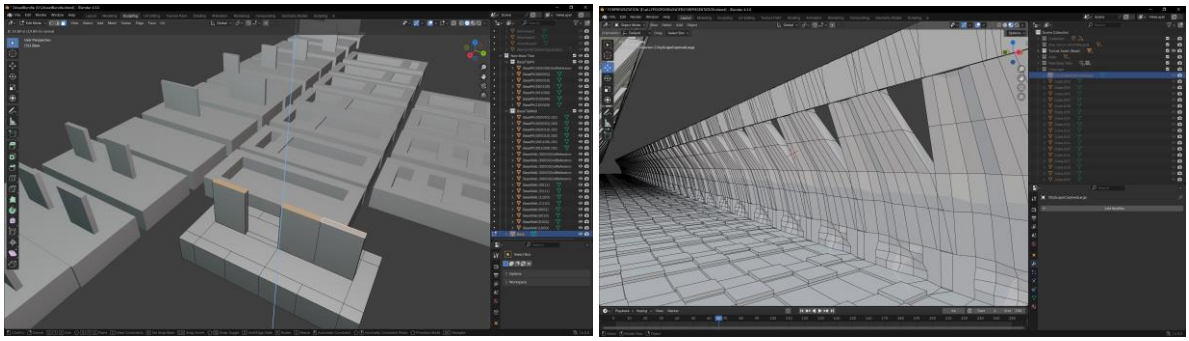
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820



Once acquainted with Blender's basic functions for creating unique assets and gaining a better understanding of modifiers, I began experimenting with the assets through Blender's lighting system, which can be mimicked (though not perfectly replicated) in Unity (with some effort). This was where the game began to take shape, with the colouring and balance between bright and dark elements designed to draw the player's attention to what is important. To get the lighting to work in Blender, it was simply a matter of adding it from the menu. Its default setting is "sun," which does exactly what it suggests: it provides a universal light source for the entire workspace. I adjusted its saturation and experimented with various settings until I eventually settled on the more vibrant side of the world's palettes, which further shaped the game's minimalist approach to the overall design of the playable space. As seen in the images above (although they are not fully textured, nor do they contain shaders, making it difficult to distinguish objects and the player from the intended viewpoint), the floor design in the game is indicative of the intended final version.

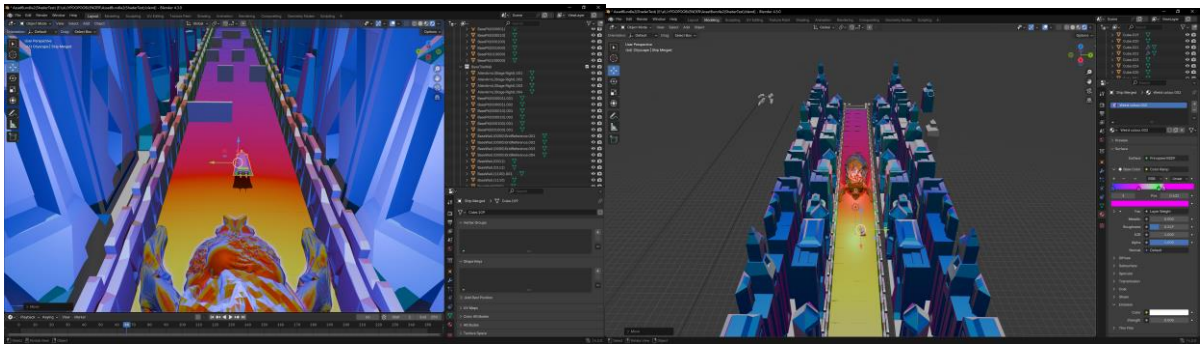
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

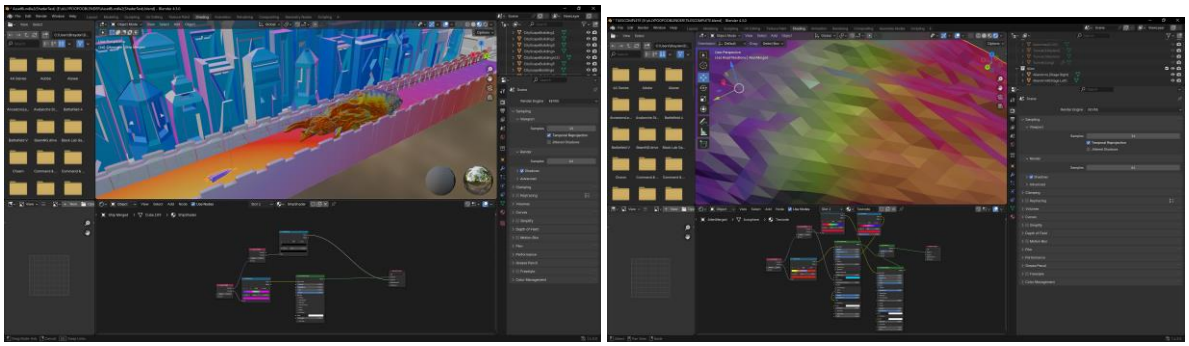




Building upon the lighting effects and considering the limitations of the array functionality, I redesigned the entire track system to be entirely modular. Initially, the concept was to lay individual tiles together as prefabricated units, joining their geometry to allow for infinite replication. The process started with a singular rectangle, to which I added the necessary edges and faces to create the required shapes. It was also during this phase that I learned that holding *Ctrl* while extruding allows for the use of a metric system, ensuring uniform measurements. The design of hazards on the tiles must remain rudimentary, as the game has a basic movement system with a somewhat locked point of view. The ideal would be to incorporate actual weighted physics with proper collision and hovering mechanics, so the pits would function as intended, rather than as "termination" zones outside of the bridges. During the creation of the new tile system, I also removed the walls, as it became evident that having the walls directly connected to the platform limited subtle adjustments and blocked the camera's viewpoint. Additionally, the extra rendering required for the faces and edges to be present in the game world was not worth applying directly. When comparing the two pictures at the bottom, the difference in geometry cleanliness and the number of faces present is substantial, and the resources required to render either system are drastically different, particularly as *Escape Velocity* progresses through its development.

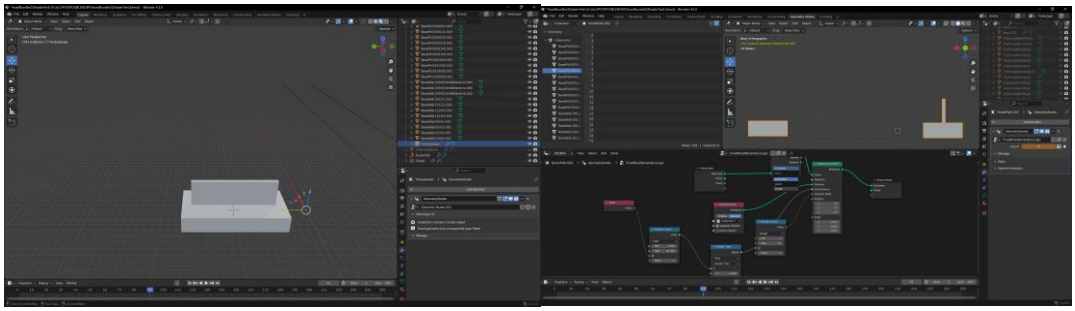
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

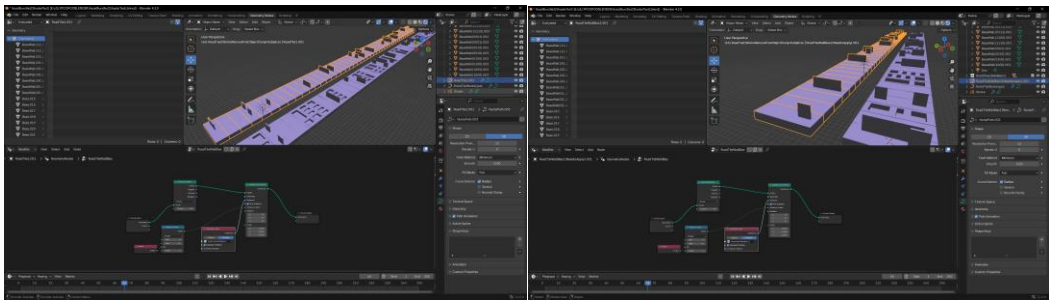




After completing the new tile system, I focused on creating shaders that would simulate the retro iridescence effect for the game's updated version, which would also be easily interchangeable and provide the player with manual control over the effect. The creation of the shader took considerable time to understand, even with online guides explaining the purpose of each node and their interactions. The initial nodes presented were always a Principled BSDF (Bidirectional Scattering Distribution Function) node and a Material Output node. The latter is self-explanatory, while the former allows for the simulation of materials and their interaction with light, including a base colour for the object. Much can be accomplished with just these two nodes. After further research, I discovered two nodes that helped me achieve the desired effect: *Colour Ramp* and *Layer Weight*. The Colour Ramp node allows for the setup of up to seven colours along a gradient, which is key to creating the iridescent effect, while the Layer Weight node is used to set values based on the surface angle and the camera's point of view. This results in either a *Fresnel* effect (highlighting the edges more than the face) or *Facing* (providing an output based on how much of the object faces the player's camera). By combining these nodes, I was able to create a vibrant, special-looking game while ensuring that hazards and the player were easily distinguishable. Another idea I explored was cycling this shader through a day-night cycle, allowing for a broader range of colours and specific atmospheric markers for different regions based on particular colour palettes. Upon further discovery, I found that by connecting the output node to the BSDF node and the volume output, the object would emit its colour across its radius.

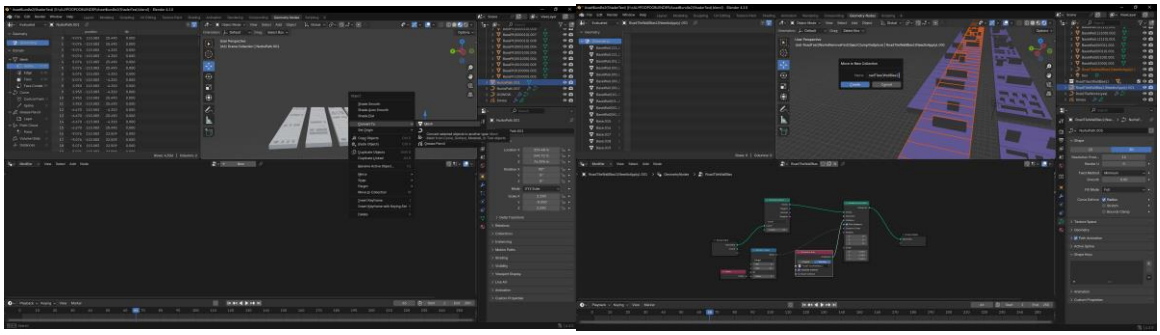
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

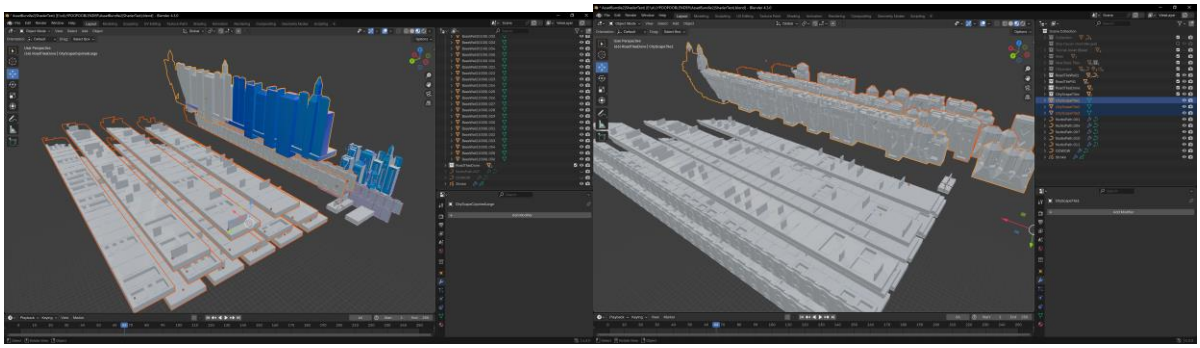


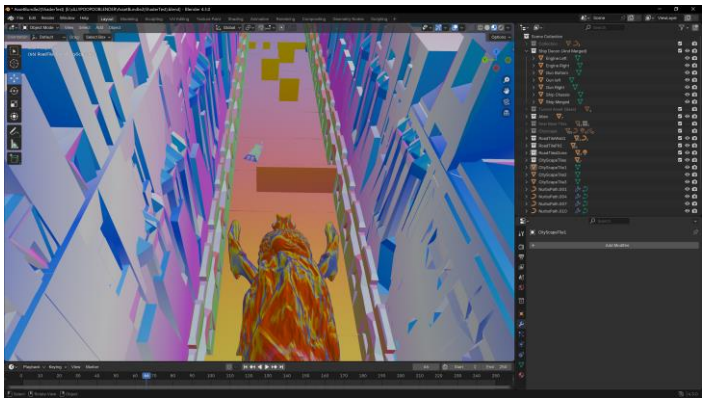


Having resolved the shader issue, the next task was to create a geometry node that would enable the generation of a randomised pathway (within the collection of objects for the set of tiles). The Geometry Node editor opens with only input and output nodes. The first node used was *Curve to Points*, which, when combined with a NurbsPath (with the count set to match the length and ensure the tiles slot together), acts like an array that follows the path and places objects at set intervals. The *Index* node, placed at the start of the chain, is used to create a unique index value on the curve itself, so that subsequent nodes can select individual “RoadTiles” mapped along the curve. Placing assets onto the curve is done using the *Collection Info* node, which identifies the collection chosen. The assets within this collection must be aligned on the same axis, which was initially trial and error. By checking the *Relative* option in the Collection Info node, the objects are placed on the path, rather than in their original positions. The *Separate Children* option ensures each asset is placed separately, rather than dumping the entire collection at one point. The *Random Value* node changes the value of each object along the path, and once everything is set, it fine-tunes the tiles. The final node, *Realize Instances*, transforms the geometry into a fully rendered object with a mesh, although issues arose when this mesh was later converted back into a curve. This geometry node setup allowed for minimal effort in handling complex designs, as it could be adapted to produce brick walls or cobblestone streets simply by changing the collection used and adjusting the placement of objects.

COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

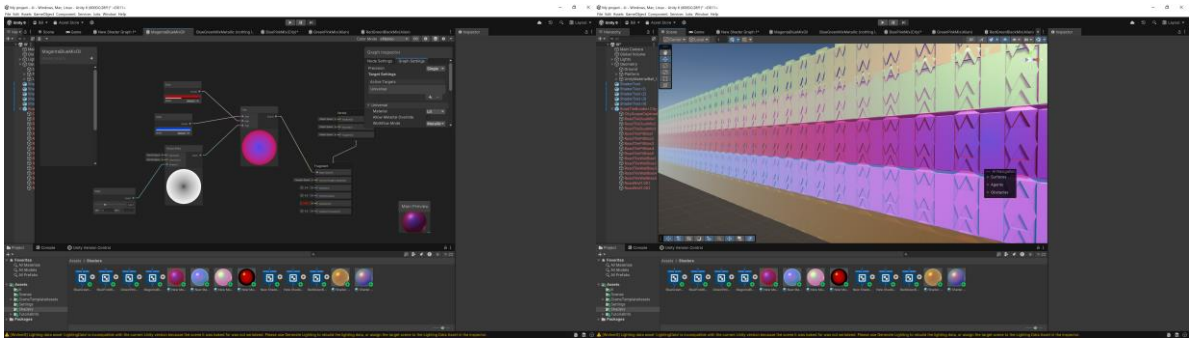




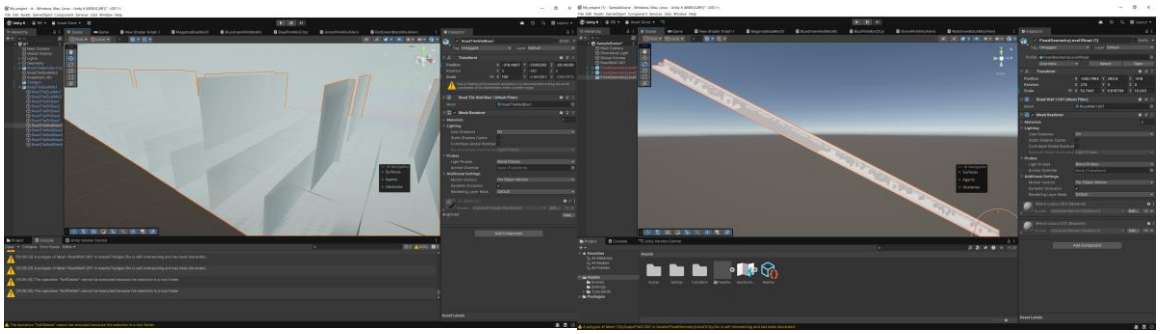


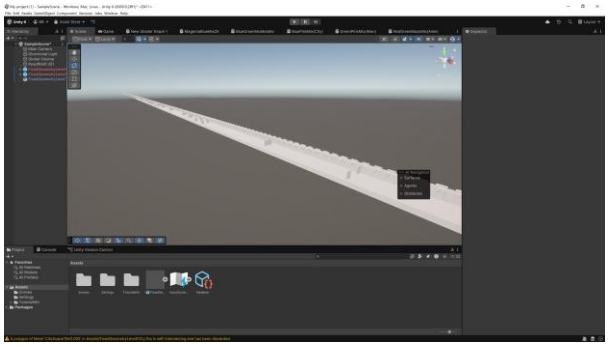
An issue arose when attempting to convert the curve back into a mesh for exporting the new randomised tile workflow to Unity. The solution was simple: apply the geometry, ensure a spare asset for the previous state, and then right-click on the object and convert it to a mesh. Another problem occurred when the shader, once applied, would not render correctly when the object was arrayed from a converted curve; the shader applied to the first object would not extend to subsequent objects. This issue was resolved by not applying the shader until after the final geometry was created. There were also challenges in determining the final camera angle, which was eventually resolved by incorporating a toggleable camera that shifts from an overhead to a left or right perspective, giving the game a 2.5D feel with added depth.

COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820



I explored Unity's Shader Graph system to create the shaders, as repeatedly recreating the shaders from Blender would be too time-consuming. One important node in Unity is the *Fresnel Effect* node, which controls how light interacts with the surface of a material. By attaching a slider to this node, the reflection angle could be altered. The *LERP* (Linear Interpolation) node allows blending between two base colours, a feature that was more time-consuming in Unity compared to Blender, where the same effect could be achieved with fewer nodes. The final outcome closely matched the Blender version, and this shader system was incorporated into the entire game world, with shaders dynamically adjusted depending on the region and time of day.



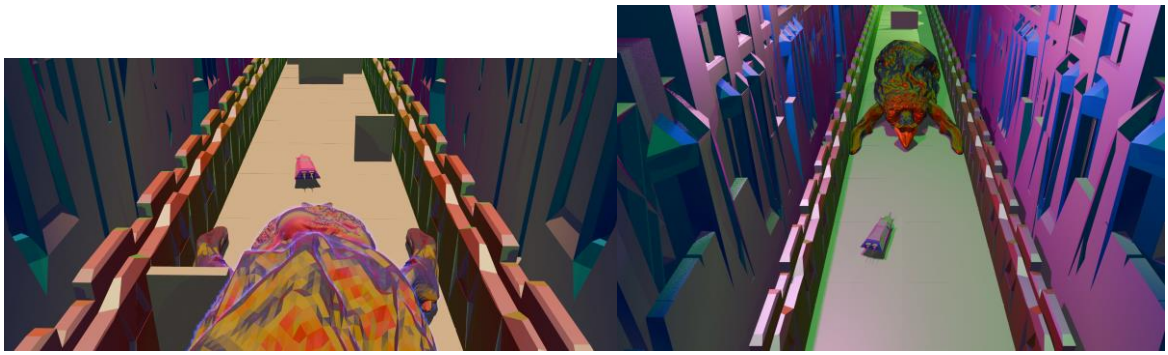


The tiles I had created and exported were not functioning as expected. Their mesh appeared hollow, and for a period, it seemed there was no solution. However, upon revisiting Blender, I deduced that the issue stemmed from the need to specifically select the object as a mesh and apply certain transformations through a process of elimination, adjusting each setting individually. Once this was done and the object was re-exported as an .fbx file, it began displaying the correct geometry in Unity. After adding a shader, another issue emerged: the object began to exhibit a pronounced jitter when

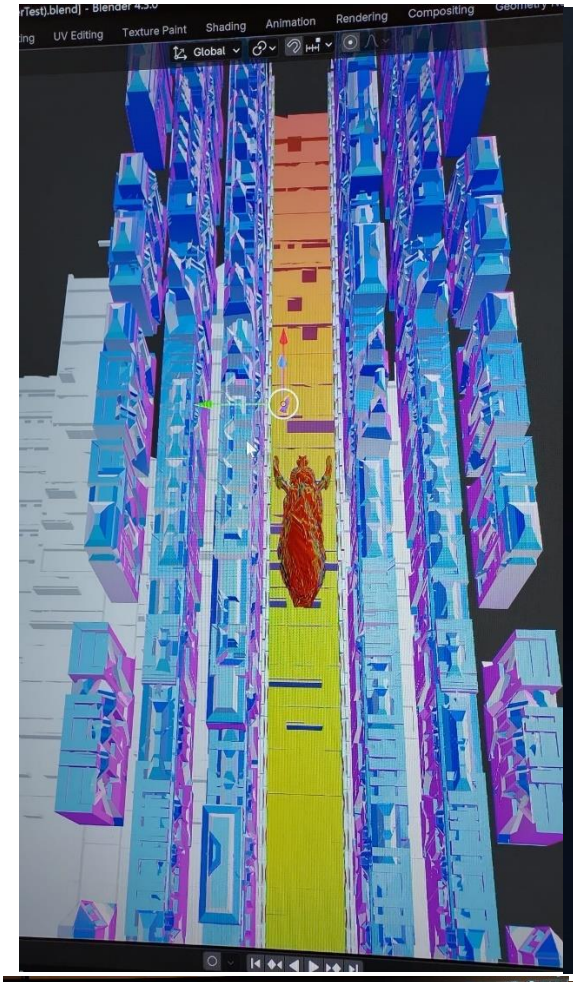
the camera rotated or when the object itself was in motion. This created an interesting visual effect, which, though unintended, contributed positively to the game's aesthetics.

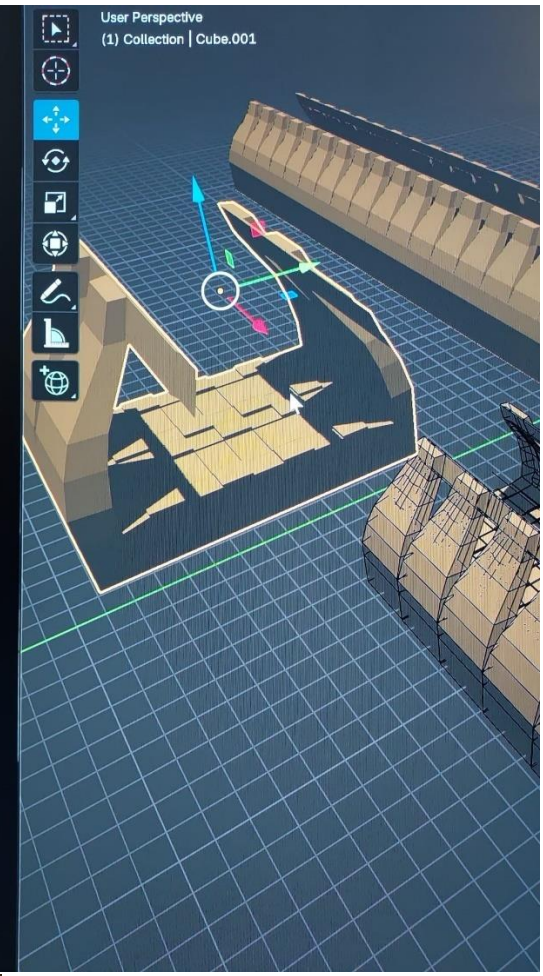
After identifying the problem with the mesh export to Unity, I returned to Blender and used the random tile samples I had previously created to design a long platform. I also included an arrayed section of wall that spanned the same length as the road. This structure would eventually form the complete "playable" area of the first level, excluding the entrance and exit tunnels. This process was repeated for all the assets, ensuring that each asset functioned properly within Unity.

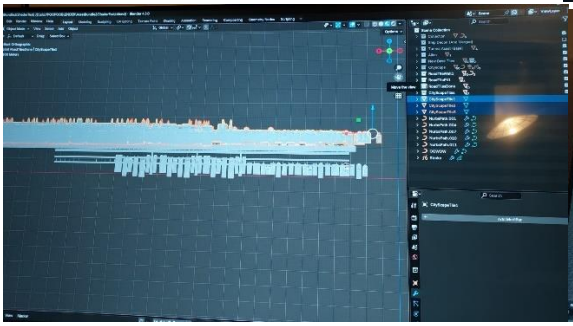
Concept art and level design within Blender –

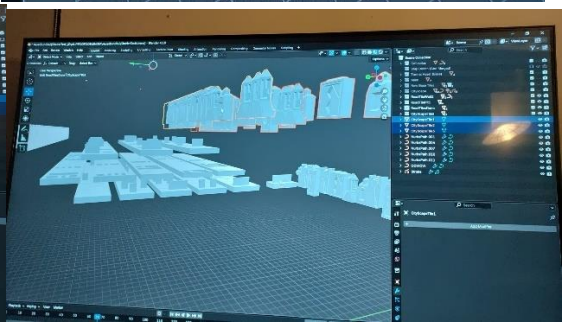


COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820
Assortment of Development Photos taken on the phone –

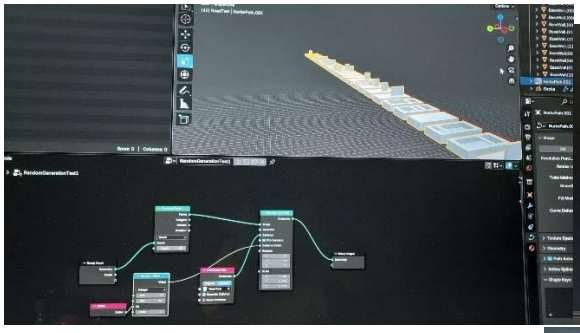


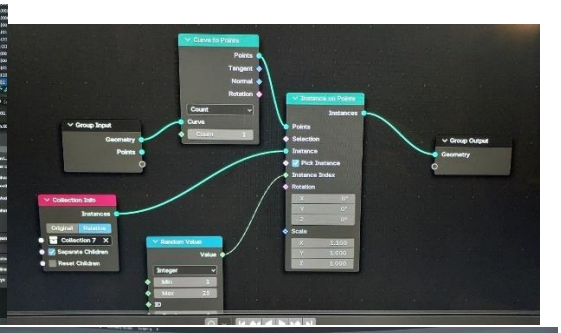


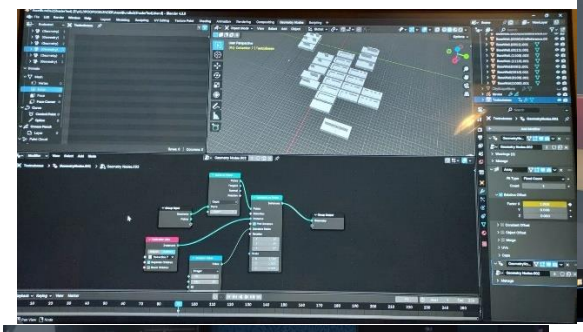


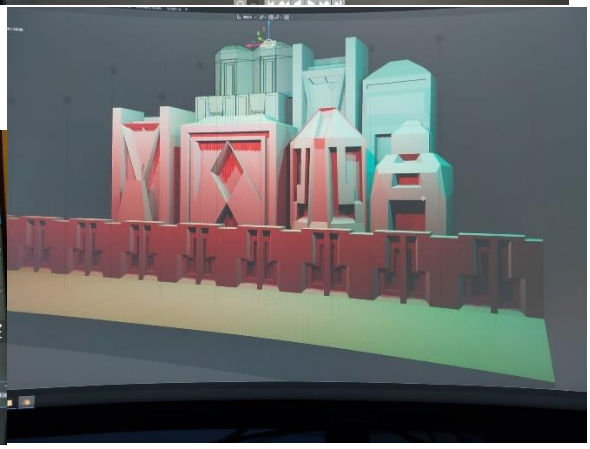


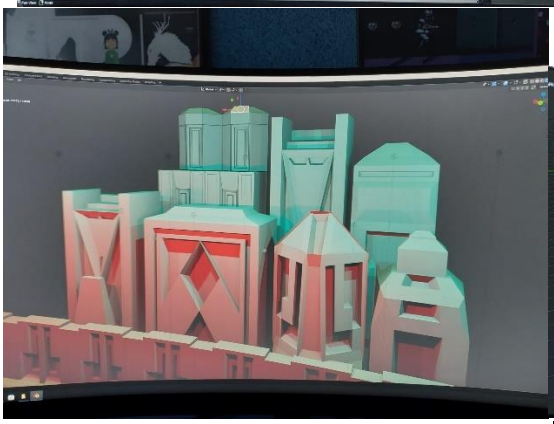
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

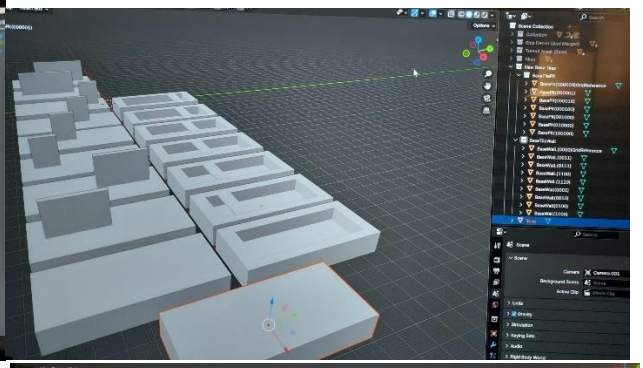


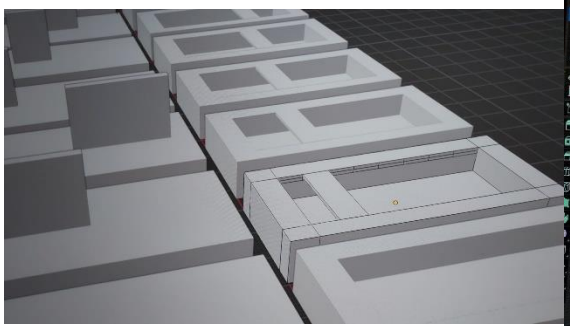




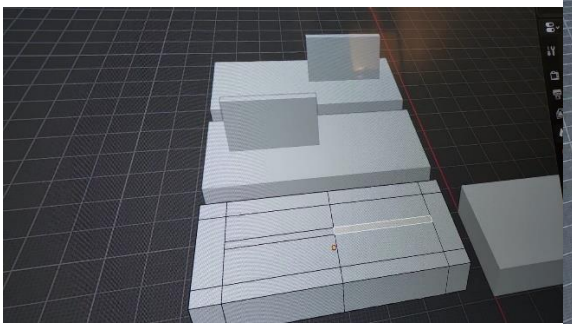
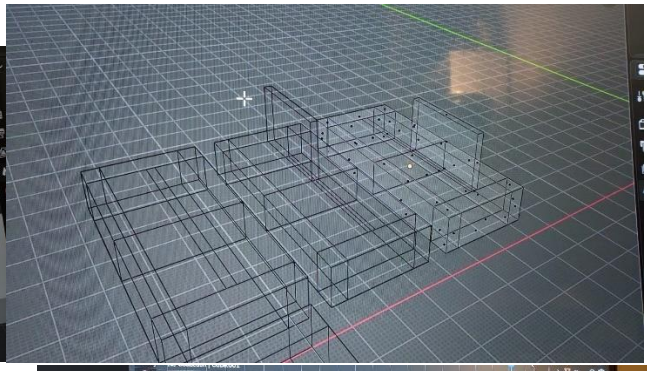
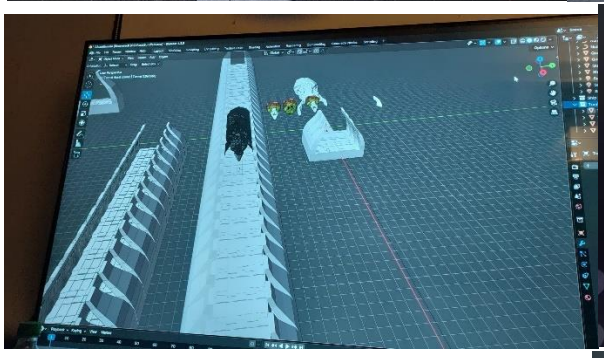
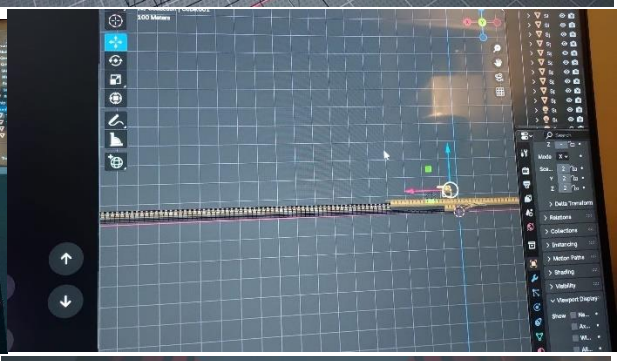
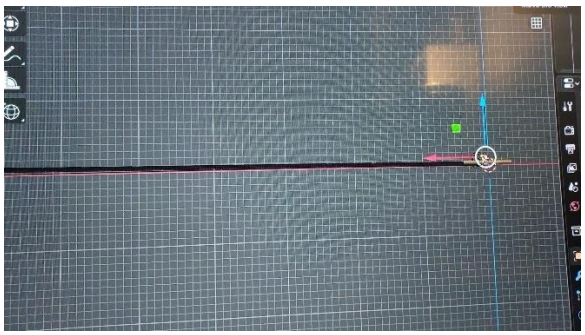
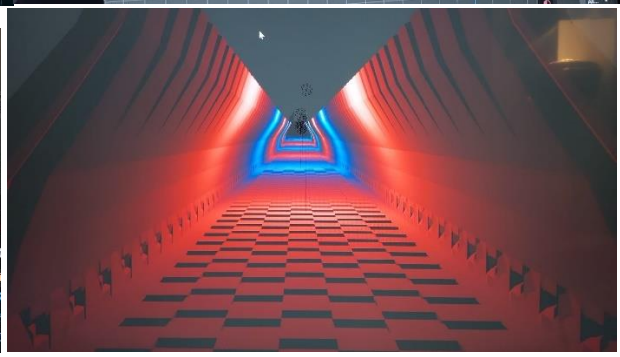




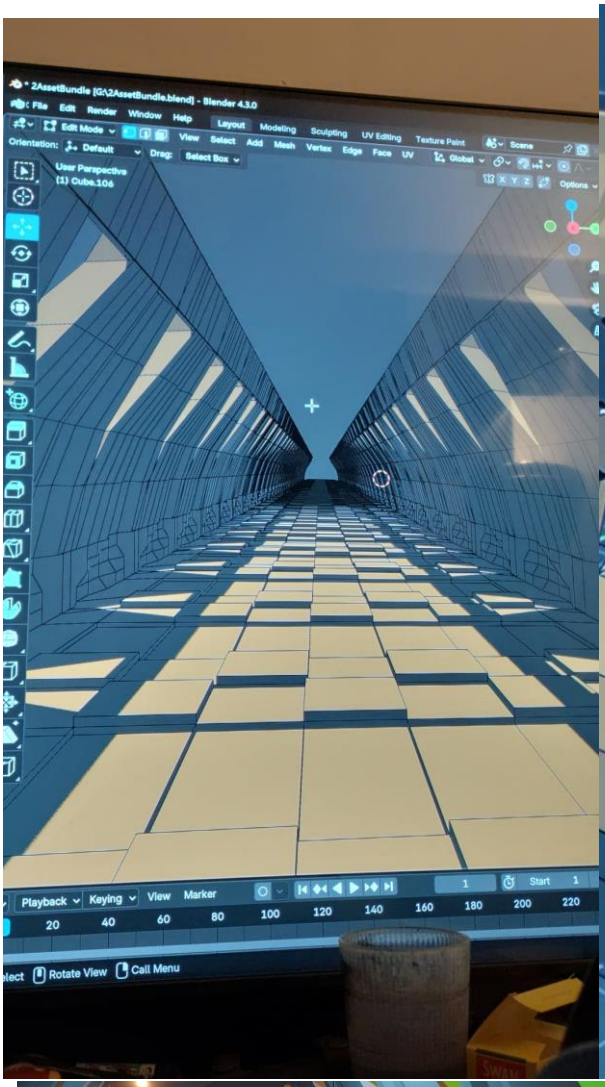


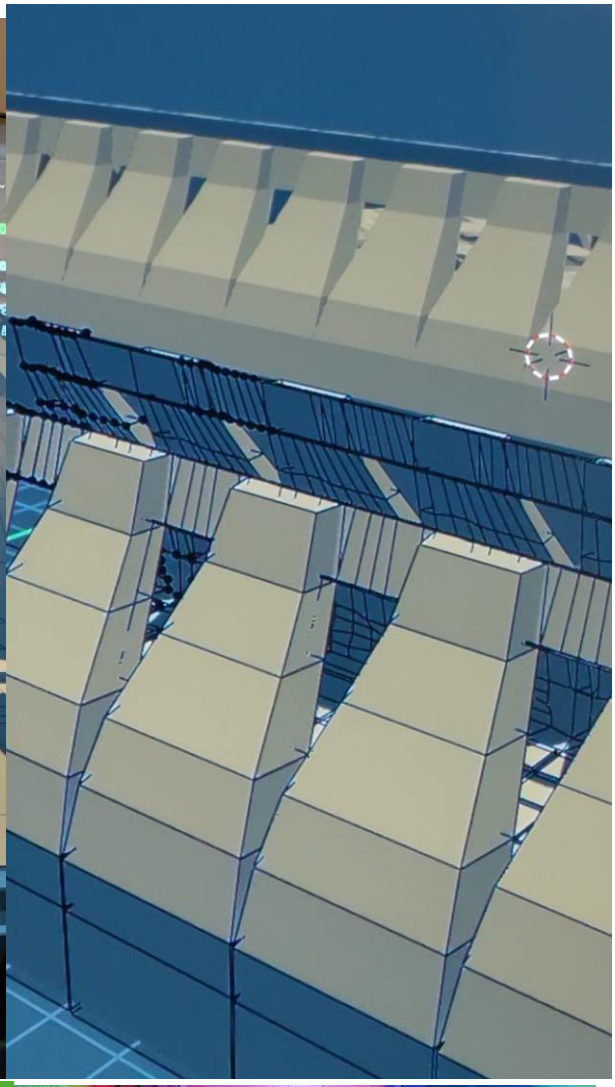


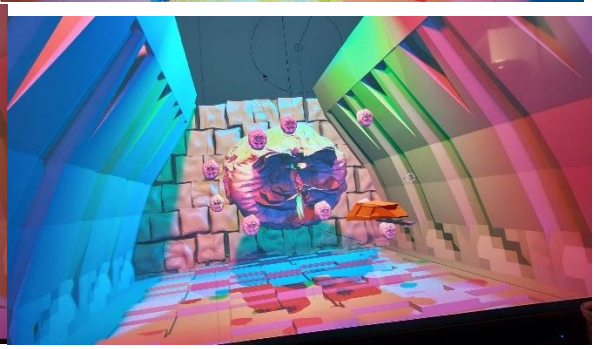
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

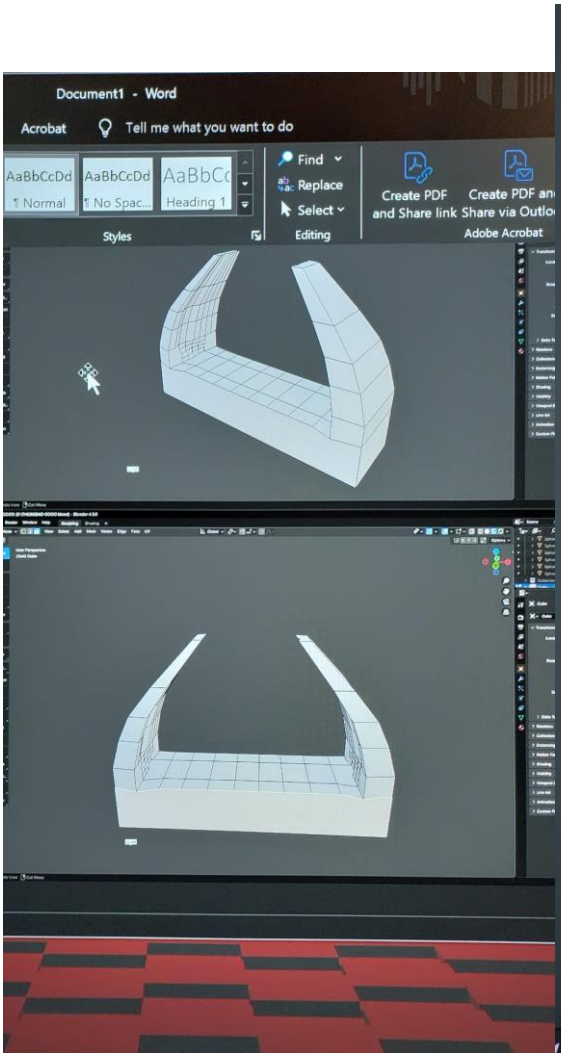
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820



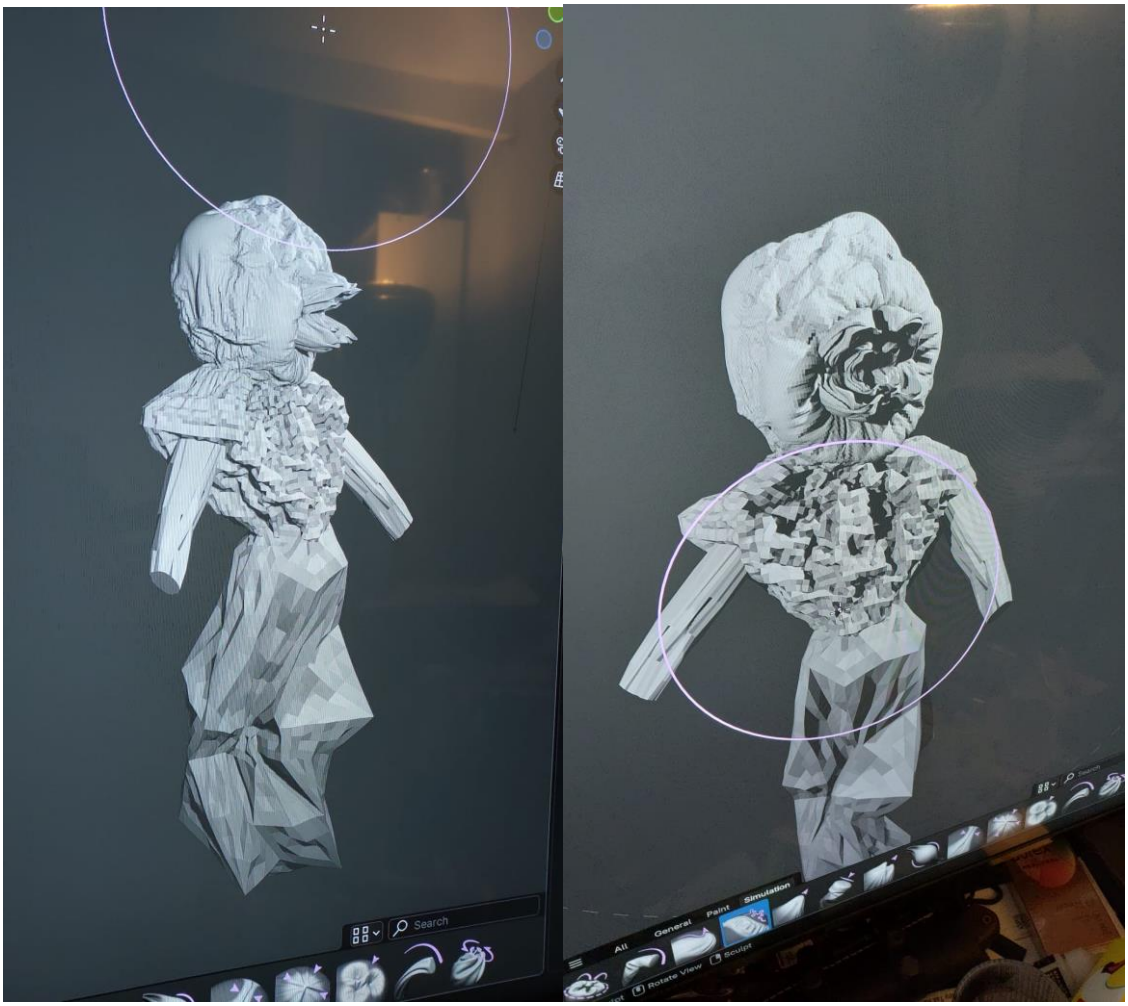




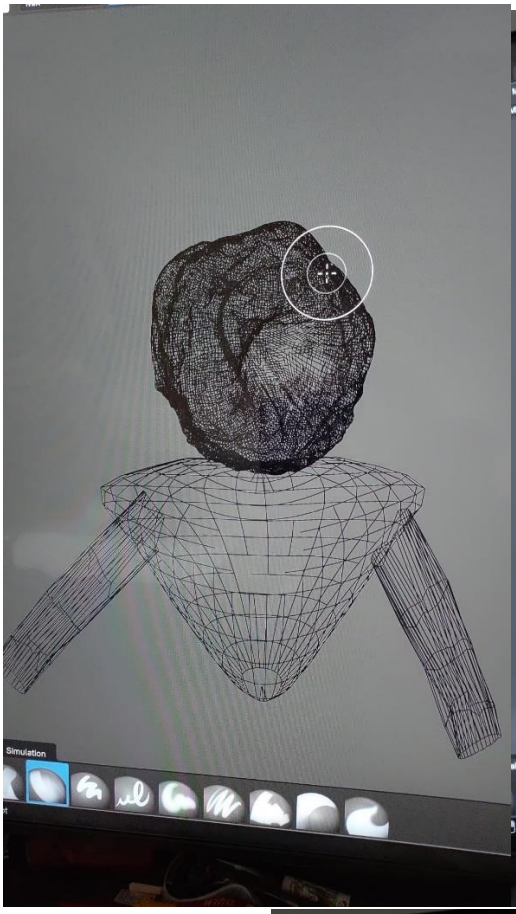
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820

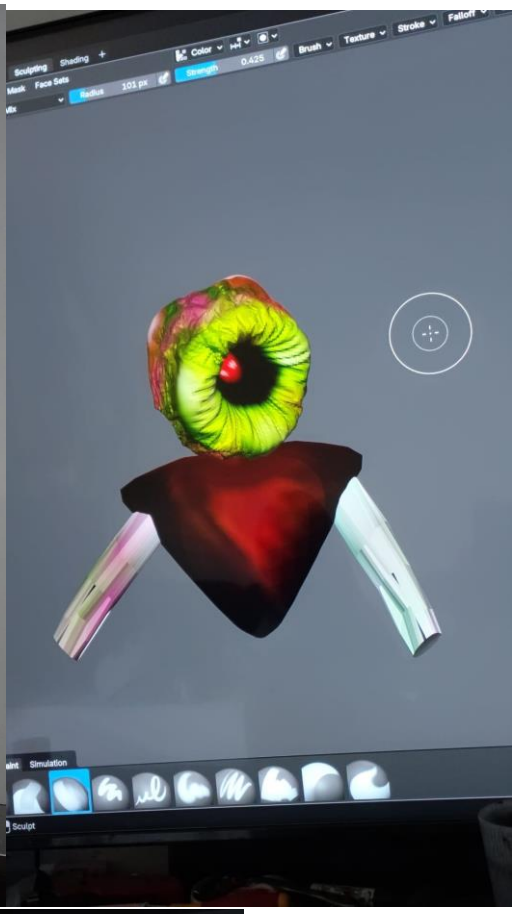


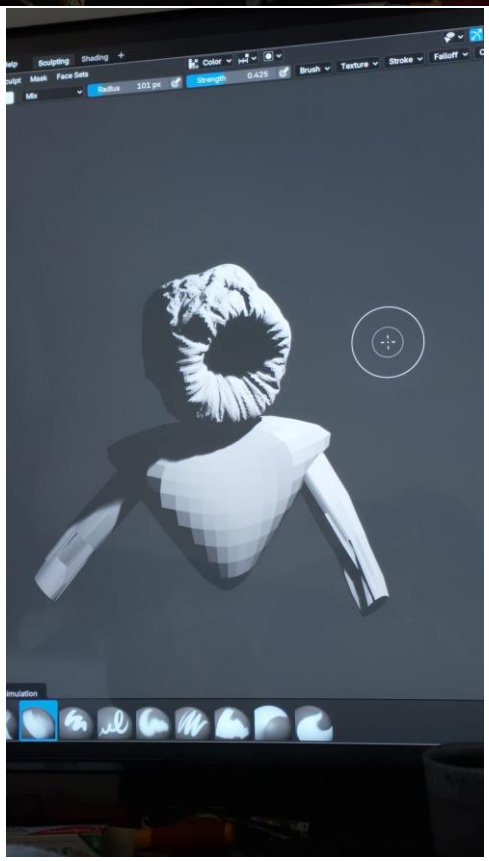
COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820



COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820







COM4008CW1
Brayden Smith + Luke Brighton
22326622 22331820