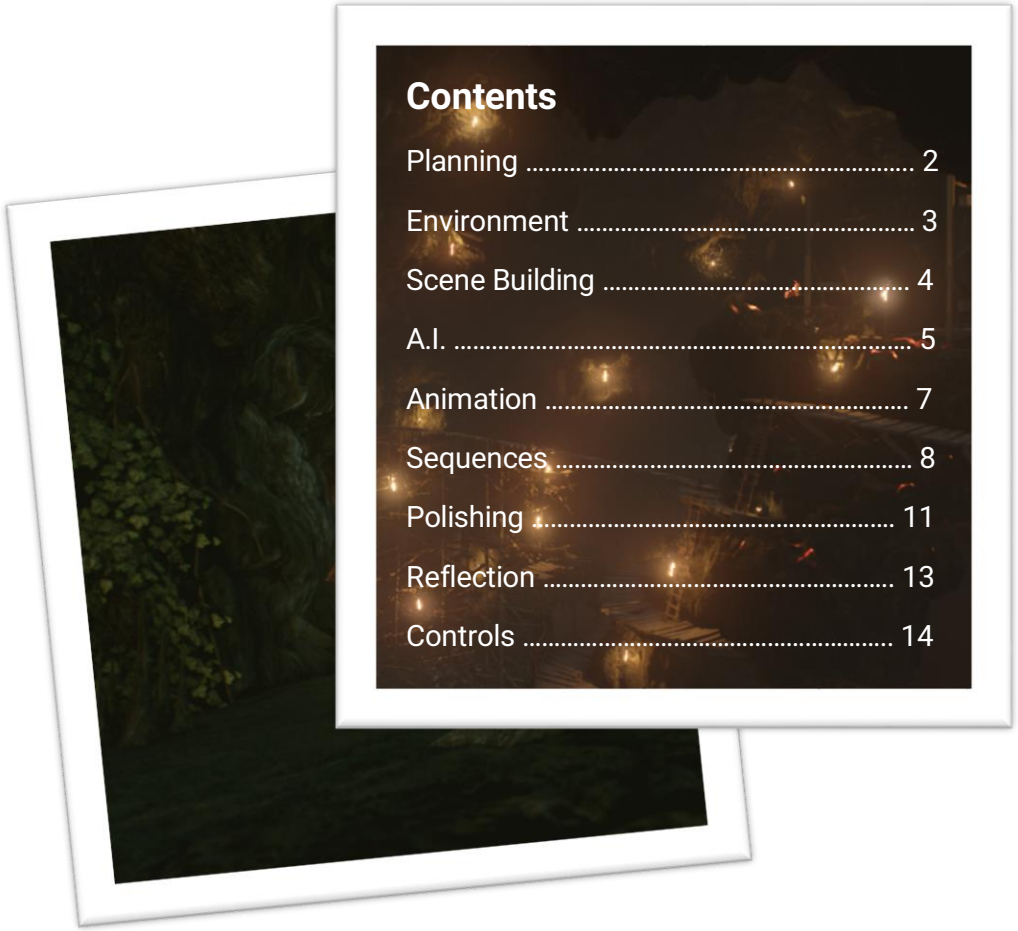
Games Programmer and Technical Artist

AARON JAMES TROTTER

UNDERWORLD



Contents

OVERVIEW

GOALS

To follow each role of the game development pipeline and investigate the importance of programmatic animation in modern animation.

To establish the technical restrictions of artificial intelligence and explore the future of animation.

PROJECT

To develop a game episode that focuses on the use of short cinematics to narrate the game and set the game's atmosphere.

Cinematics are to run in real-time and update in-game features to allow the player to proceed to the next stage.

To use crowd generation to produce an immersive and thrilling experience.

The game must ship in a playable state and run smoothly across different hardware and platforms.

PLANNING

I believe that a fantasy setting gives me the scope to take full control over the environment. I plan to follow typical lore where orcs are at war with humans. Orcs are nomadic and can survive harsh environments, moving when local resources are depleted. More permanent settlements are built deep underground.

I envision a natural gorge inside of a mountain with lava flowing through it. The inhabitants have mined a labyrinth of tunnels and using local resources, have built bridges, fences, huts and machinery. The surrounding landscape will be barren and rocky.

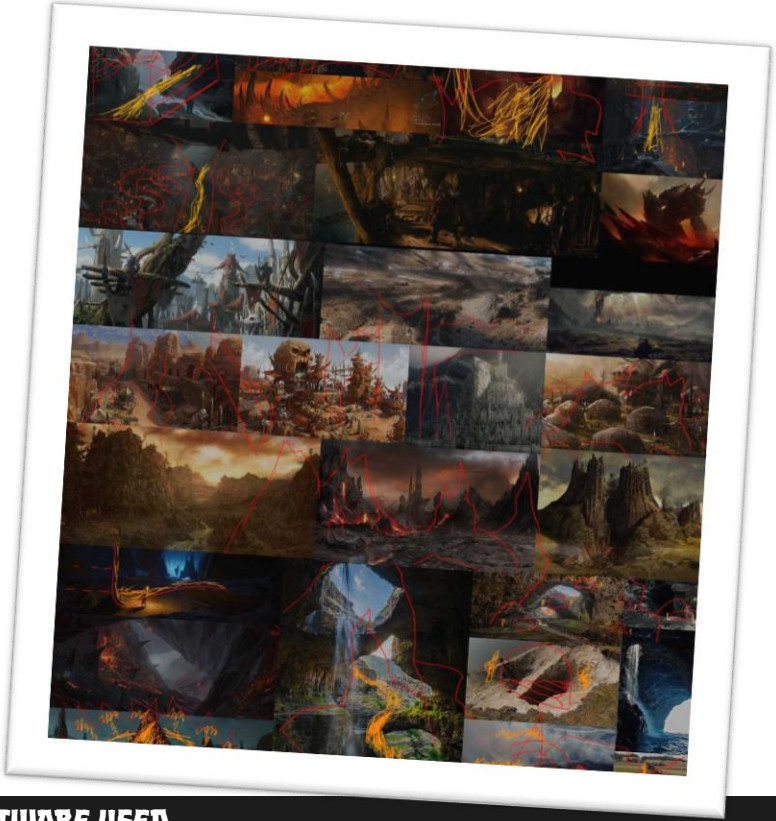
An underground scene brings challenges in terms of lighting because of the **limited natural light**. Scene lighting such as lava, torches, luminescence plants/insects are required to bring the scene to life, however, creates performance issues and the use of dynamic or baked lights must be considered.

DEVELOPMENT PIPELINE

Game engines provide much greater support for the insertion of algorithmic expressions to programmatically animate scenes than traditional software.

Animation is tedious and traditionally done through keyframes and motion capture. More recently, minute animations such as facial features are audio-driven but, it is still too early to use algorithmic animation on a large scale. Algorithmic animation can be best seen through A.I. as characters can be pre-computed to interact with each other or the environment. Film and Games are becoming more and more intertwined as can be seen in Pixar's "Finding Dory" which made use of Unreal Engines VR support.

Rendering animated videos through specially designed software such as MentalRay can take hours, days or weeks depending how much data needs processed. Whereas game engines render in real-time. Game engines can achieve this by pre-calculating data and use of optimised meshes and lighting. Game engines are simply not designed to render extremely detailed scenes and are incapable of rendering complex effects.



SOFTWARE USED

- Unreal Engine 4
  - Smart Spline Generator (modded)
  - Sangloo's Modular Assets (re-textured)
  - Planet Venus Landscape (modded)
- 3dsMax
- Mudbox
- World Machine

PROCEDURAL TEXTURES

ENVIRONMENT

I have previously only worked with basic textures and because of time restrictions, I found "Planet Venus Landscape", an atmosphere and procedural texture for a Venus-like setting.
I learn from experience and learned a lot from stripping apart their code. I modified the code to work with visibility layers to allow me to cut holes in the landscape as well as tweaks so that material could be applied to meshes

Heightmaps

Heightmaps work by storing the height component for each vertex in a quad and are the fastest method for geometry lookups e.g. collision detection. They are relatively low on memory usage and their ability to allow for dynamic LOD make them the most efficient method of rendering terrain. However, they cannot have any overlapping geometry and as a result, holes cannot be created. Caves and overhanging cliffs must be created within the engine through meshes and transparent materials. Moreover, heightmaps are prone to artifacts and ultimately provide less control whilst texturing. Heightmaps can be rendered as a mesh but chunking is required so that LOD and other culling techniques can lower the detail or prevent the rendering of hidden chunks. Some off-the-shelf game engines such as UE4 come with automatic frustum and occlusion culling set by object bounds.

Voxel

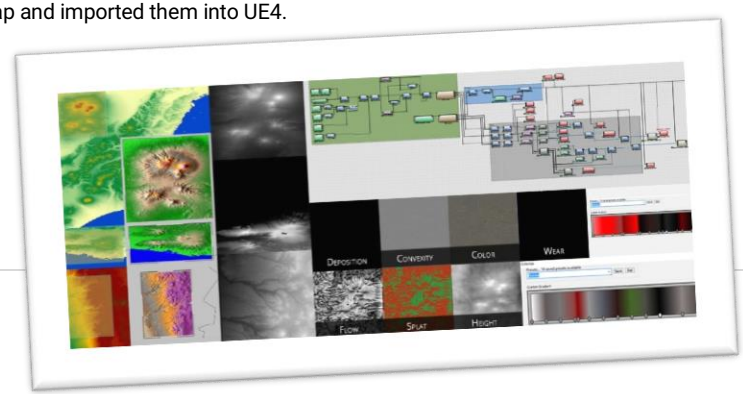
Voxel terrains store data for each point inside a 3D grid and is particularly common in older games but has made a huge come back in recent years. Voxels allow for continuous storage of hidden geometry popularising infinite open worlds like Minecraft and No Mans Sky. Voxels are easy to modify and their structure can be changed on the fly as can be seen in games such as Astroneer. The main advantage of using Voxels is that they can produce caves and overhanging geometry, however, these are much slower than its competitors to render. Furthermore, artifacts are common and are incredibly taxing on VRAM. Here's a delightful read on procedural world generation using voxels.

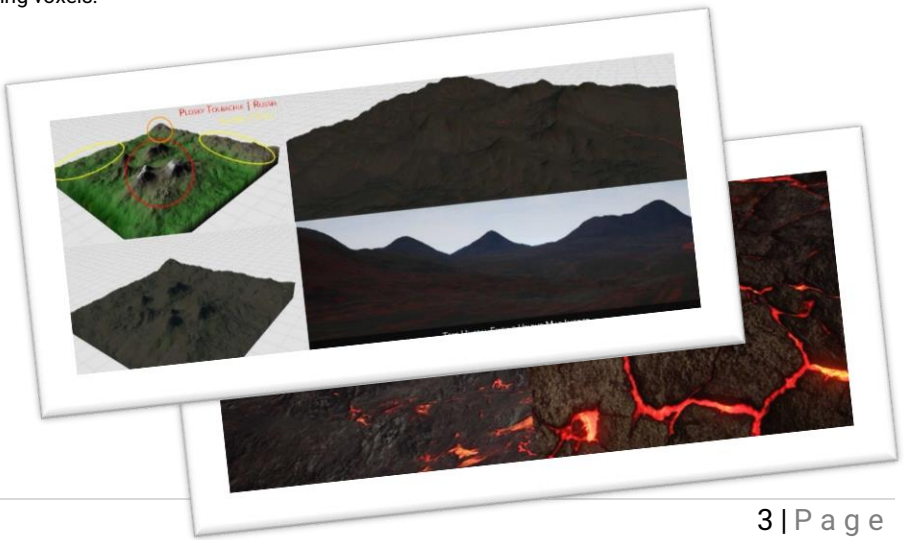
Meshes

Terrains can also be created through meshes which are very flexible and give full control over advanced terrain features and textures. Meshes do not require a geometry shader and as a result render extremely fast. As all co-ords are stored individually for each vertex, they are precise and have a low memory impact. They are commonly used to create interiors, caves and overhanging features (close quarter areas). However, it is difficult to place meshes through code and have a poor-dynamic LOD and collision detection.

WORLD MACHINE

Using Nasa's satellite imagery, I could export a height map into World Machine and manipulate the terrain through a variety of nodes. I merged multiple locations together and created a to scale, 3D representation of them. I exported the new height map from WM along with a colour map and a splat map and imported them into UE4.





SCENE BUILDING

I created most of my own assets (e.g. bridges, torches, scaffolding). However, others were taken from various sources online; Luos's Free Modular Cave Assets & Sangloo's Modular Assets. I created a variety of bridges and their LODs inside 3dsMax before finding that I could create LODs inside of UE4. In Max, I used soft selection to make planks look like there were warped from the heat or bent from continuous trampling. The bridges are low poly to allow for mass duplication.

I added the bridges along splines, setting the curve and the gravity for each. SSG does not yet support blueprints, so I had to modify the code to have supports across some walkways. I wrote blueprints to randomise the scale and rotation of frequently used assets such as torches to speed up development. I also wrote a script to snap objects to either the floor or the roof by sending a raycast every time the mesh was moved and jumping to where it hit. This made it much easier to position objects.

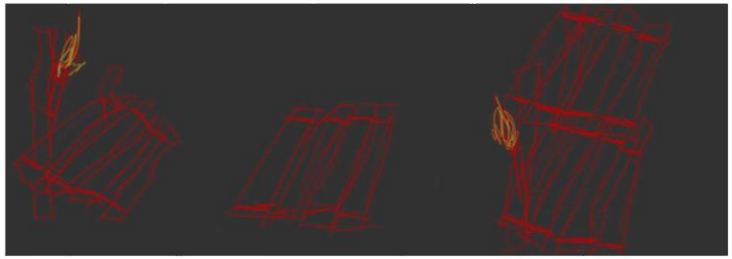
COLLISION & BLOCKING

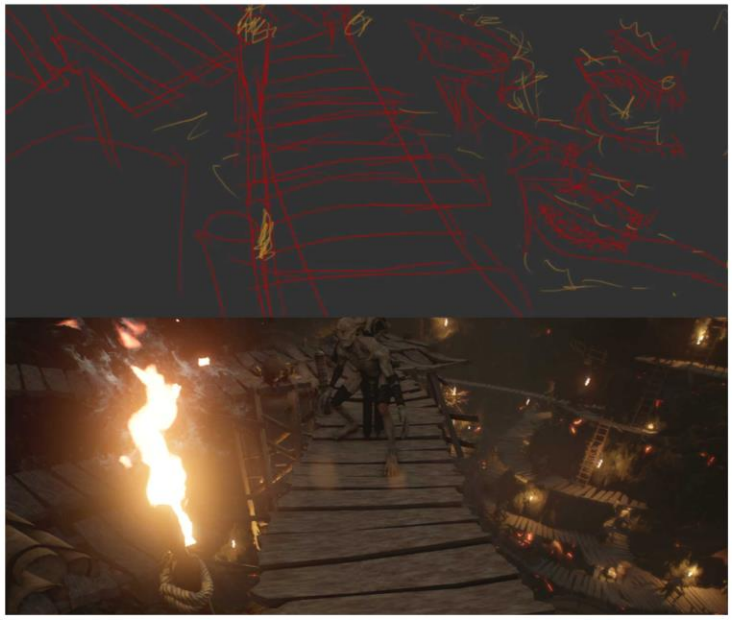
Collision layers can take the form of scalable spheres, boxes or capsules. In UE4 boxes can be converted to convex collision layers allowing the collision to take the form of the mesh itself considering accuracy other parameters. In UE4 it is possible to allow collisions to block all or ignore certain actors. Events can be created for actors that overlap or meet collisions.

Convex collisions are great for hollow objects or those with cut outs. However, the more vertices a collision has, the more processing power is required to render it. As a result, convex collisions should only be used characters or dynamic objects need to be near or inside of a hollow mesh.

Collision is great and ultimately doesn't require too much processing but, large blocking volumes can be added in place of multiple small collision boxes to further improve performance. I have used small blocking volumes to cover areas the AI is not supposed to go or areas that I have noticed the AI getting stuck.

I converted some bridges and their supports to a destructible mesh so that I could film a boulder smashing through them. However, UE4 currently does not support LOD of destructible made within the editor. An alternative would be to create the destructible chunks in other software and import them with LOD. Destructible objects are considered essential by UE4.

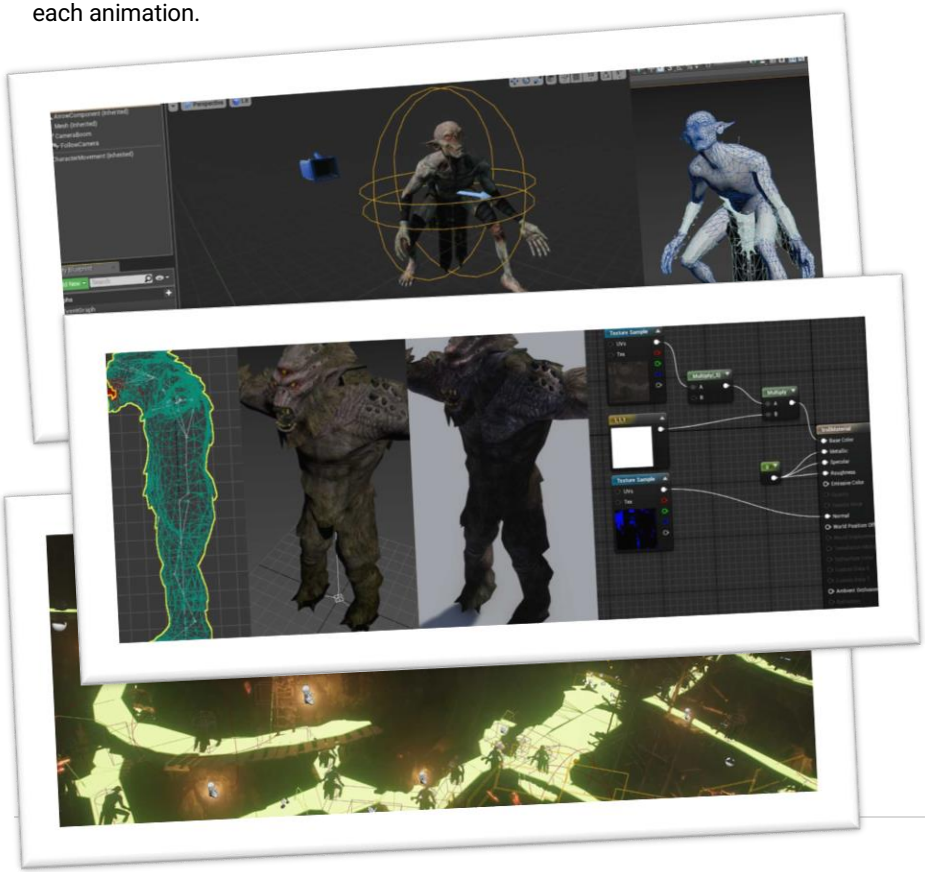




MONSTERS

Because of time restrictions and my focus on A.I. instead of character modelling. I searched the internet to find models for my scene but only found cartoon goblins or high poly assets. As my machinima is purely for educational purposes, I ported a 'falmer' and a 'troll' out of The Elder Scrolls: Skyrim and into UE4. I sent Bethesda an email asking if they had problem with this but I did not receive a reply.

I created a material from the monster's textures and added my own adjustments so that the material would display correctly with my scene lighting. I had to edit the monster's bones as they did not import correctly into MAX and thus had to update each animation.



A.I.

NAVIGATION DATA

Navigation data must be built on the persistent map for AI to roam. I have found it does not work if built on sub level when using level streaming.

Navigation Mesh
Unreal automatically generates a navigation mesh inside of NavMeshBoundsVolumes and to the specifications set up inside the level's navigation settings. It will be as close to objects as the bounds of the smallest AI character allows. NavMeshBoundsVolumes are square/rectangular and can build a nav mesh on unwanted items outside of the intended scope, this can be combated by adding a NavMeshModifer with a Null class and cutting away the unwanted areas.

I had a lot of trouble setting up my navigation mesh as it doesn't read the navigation data attached to meshes in blueprints. For example, I created a blueprint that generated supports for paths. However, only paths should affect navigation data and not the supports. Supports that had a hidden path still affected the navigation mesh so I replace the main problematic instances of the blueprint with static meshes.

Navigation Link
NavLinkProxies can be added to specify where Pawns can jump or drop off ledges allowing them to temporarily leave the NavMesh to cross gaps in it. I have used this substantially ensuring A.I. can drop down anywhere with a reasonable height but not jump back up. However, NavLinkProxies are limited and A.I. will only jump at the specified point, so many may need added along a path.

PATH FINDING

I have scripted my classes in such a way that allows me to reuse them for each character type.

Go to Location and Patrols
There are a couple of methods for generating crowds but, I only require the A.I. to move in a set direction for different shots of the cinematic. To do this, I have placed collisions around the map and attached a collision actor to the characters that I want to move. Each character has a patrol and a wait flag. When patrolling the characters walk around their local radius and if they stray too far from their initial location, they will go back to it. When the wait flag is removed, they will run to a location near the assigned collision before patrolling the new location. In a film, each shot will be rendered separately and the characters will be told exactly where to go where as this cinematic is rendered in one take.Unreal has four main behaviour nodes; decorators, services and tasks.

Composite

Composite Nodes define the root of a branch and the base rules for how that branch is executed. They can have Decorators applied to them to modify entry into their branch or even cancel out mid execution. Also, they can have Services attached to them that will only be active if the children of the Composite are being executed. Composite nodes come in the form of selectors or sequences.

Services

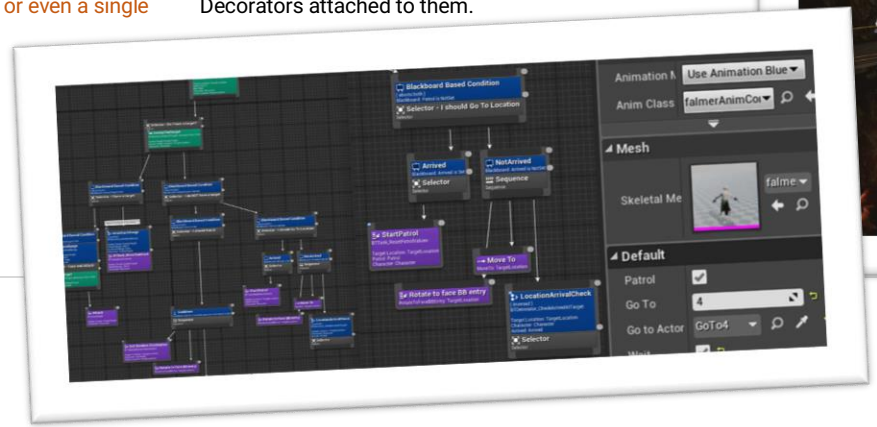
Services attach to Composite nodes, and will execute at their defined frequency if their branch is being executed. These are often used to make checks and to update the Blackboard. These take the place of traditional Parallel nodes in other Behaviour Tree systems

Decorators

Decorators (conditionals) are attached to either a Composite or a Task node and define whether a branch in the tree, or even a single node, can be executed.

Tasks

Tasks are nodes that "do" things, like move an AI, or adjust Blackboard values. They can have Decorators attached to them.



AVOIDANCE

The character movement component in Unreal comes with a prebuild avoidance system that changes the avoidance state. However, my navigation paths are very small and not much wider than an individual orc. This results in the A.I. blocking each other's path. I wrote a script to combat this by pushing nearby characters considering velocity and speed.

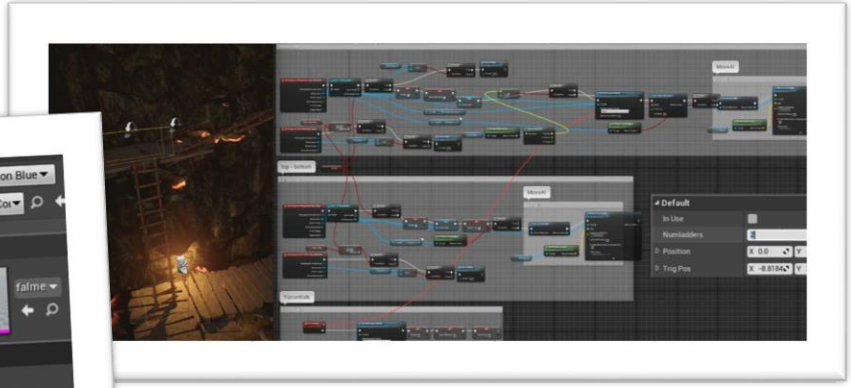
LADDERS

I use nav link proxies to allow the A.I. to jump between gaps in the navigation mesh but, I don't want my A.I. jumping to a higher platform because that would look unnatural. Instead, you must either play a climbing animation or have them walking in a forward motion whilst they move upwards. One problem I had, was that all my ladders are positioned at different angles which would require multiple animations that I wouldn't not have time to create. Therefore, I went with the latter option.

Ladders must be able to move multiple A.I. at the same time in the one direction and prevent other A.I. from attempting to go the opposite direction whilst in use. To do this, I created variables on each character telling it if it can climb ladders or not. However, this required duplicating my code for each character.

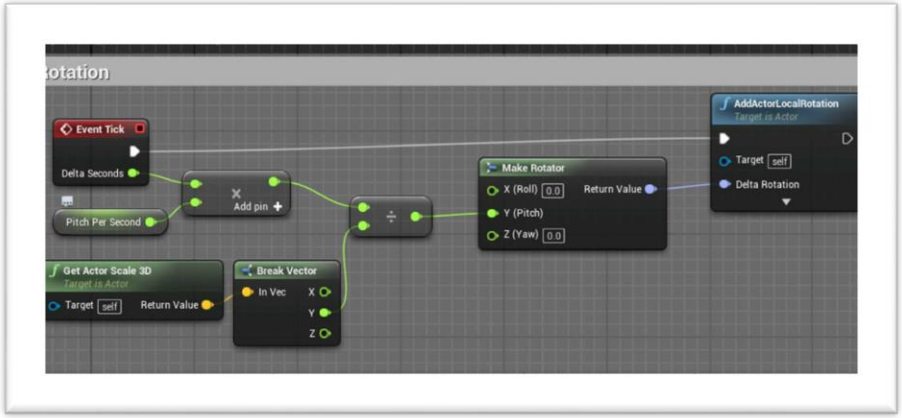
I put a trigger at the top and bottom of the ladder and when activated, it will move the character to the opposite end. I set the character's character movement mode to flying to move the characters upwards and escape gravity. I have a timer that will auto force the character to walk should they not arrive at the second trigger.

I wrote a script to add multiple ladders to same blueprint and their positions can be changed, updating the top and bottom trigger.



MACHINERY

The scene still looked a bit lifeless and I filled it out with some animated mechanical props. For example, I created rotating wheels by adding a y-axis rotation to a mesh on every game tick. Sounds and particles helped the machinery look more believable. Others such as a crane rotate on their z-axis and a variable velocity.



BLENDSPACES

Blendspaces are used to seamlessly blend animations into each other over a variable value. In my case, I used a blend space to blend the Idle/Walk/Run animations regarding the character's speed.

A blueprint is used to combine all the character's assets. Epic allows you to use their Character Class as a base which really speeds up development. Through it, you can pick your AI Controller Class and set up variables regarding the characters' movement. That's half the work, so thanks EPIC! In the blueprint, you need to add a mesh and pick the animation controller as well as a collision component. Capsule's work best for humanoids

For Navigation Mesh to update correctly, the character must be added as an 'Agent' to the navigation system in project settings. The agent radius and height refers to the radius and height of the collision component added to the character. Unreal works by checking for characters that share the closest values to the added agents.

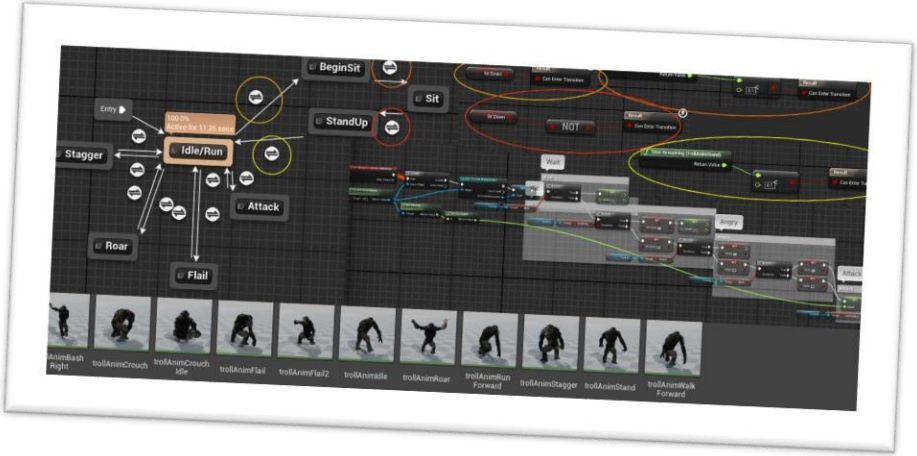
ANIMATION

HUMANOIDS

As previously mentioned, I borrowed the falmer and troll assets from Skyrim. I also borrowed their animation sets but had to set them up in Unreal which is a good bit of work.

I converted each animation separately to an fbx before importing them into Unreal. I then created an animation controller consisting of a state machine.

State machines break up a skeletal mesh's animations into a series of states governed by Transitional Rules that control how states blend into each other. State machines are largely used for the character's main processes such as idle, run, jump, attack.



SEQUENCES

Level sequences have replaced the legacy matinee feature and is Unreal's solutions to cinematics and complex scripted events. They bring together cameras, audio, particles, events and all meshes in the scene through a timeline.

Unlike other animation software, Unreal's timeline can access and edit game variables which can be scripted to fire off other events and sequences.

The main cinematic was created by a master sequence containing sub-level sequences containing different camera-cuts. However, Unreal is incapable of any fancy transitions and comes with a basic fade in and out of black. Therefore, it is quite difficult to prevent hard cuts without post editing.

EVENTS

Game objects such as audio or meshes can be added directly onto a sequence and have their values changed. This works great for playing audio at a certain time but, these changes will not impact the actual game as they revert to their default values once the sequence finishes.

On some occasions, I require objects to retain their new values. This can be done through events. Events can be added to the timeline which then run a custom event on the level blueprint. For example, when the troll bursts through the doors in the final cut, the doors need to stay open in the actual game.

I have programmed a skip feature to allow the player to skip cutscenes. This has led to problems where the player skips before events are fired. I have temporarily solved this by not allowing the user to skip the sequences that impact gameplay.

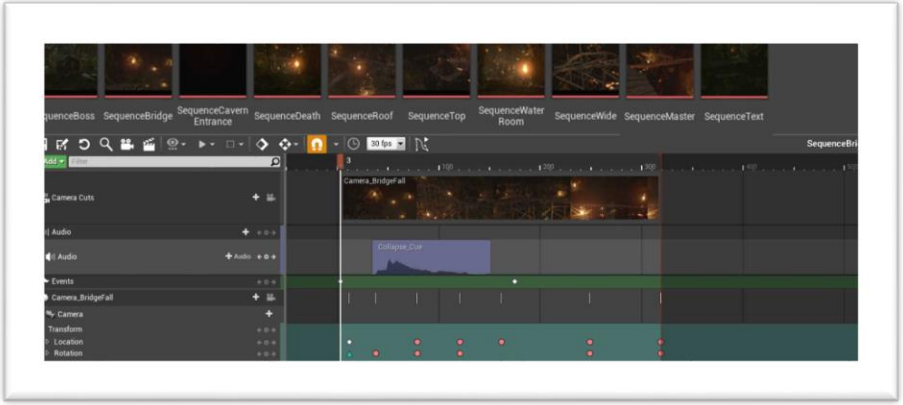
RENDERS

Level sequences can be rendered as a series of images or an AVI. This allows me to beef up the draw distance, shadows and textures and have the engine render each frame in its own time. This quality would be impossible to render in real time on consumer desktops.

As a result, audio cannot be recorded with the render and a separate real-time render must be produced in-order to capture the audio. I did not take much notice of this until I began production when I remembered I am using A.I. and no two scenes will ever be the same. Dang.

As detailed later, my A.I. make sounds and those recorded from a real-time render will not necessarily line up with the location of the A.I. in the fully rendered sequence. So, I have big problem.

A solution to this would be to always know where the AI should be but this requires rendering each camera shot separately and manually scripting each character. But this defeats the purpose of my original goal to create a machinima using crowd generation to populate an immersive environment. Therefore, I will have to work at optimising the game to run as smoothly as possible in real-time at a reasonable quality.



PARTICLES

I had previously only worked with particles in 3dsMax and had to learn an entirely new process to create particles inside of UE4. I followed a tutorial by DigitalTutors (DT) to give me a head start before making some myself.

Particles Types
Sprites are much easier to process and are used predominantly when multiple particles are required to emit at the same time or very quickly. As a result, they are commonly used for smoke, sparks and flames. Sprites are 2D and therefore, can rotate to face the camera, preventing the particle from disappearing. Engines like UE4 can process sprites through the GPU which allows the output of significantly more particles. However, GPU sprites are generally restricted in terms of functionality such as light emission.

Meshes are 3D and therefore, are heavier to process. They are usually used for larger particles such as falling rocks or leaves. They work together with sprites and often benefit from ribbon trails.

Beam emitters create an effect between two points and are usually used to simulate electricity effects or bullet trails.

Ribbon trails are usually created from sprites are very useful when creating projectile effects.

Anim-trails are particles that can follow the bones of an animation mesh and can be used to produce sword slashes or magic casts.

Triggering Particles
I created a blueprint to simulate a gust of wind stoking fires. I created my own flames and embers using sprites from Epic's elemental demo and attached them to broken pot meshes. I added two trigger volumes, one inside the other. When a moveable actor enters the outer trigger, small embers will be emitted once and fly out of the fire in a circular motion. The embers will continue to produce whilst the actor is within the inner trigger and will dissipate when the actor leaves it.

Socket Emission
Whilst researching particles I decided to go back and revamp one of my earlier pieces which would benefit from my newly learned skills. I attached a fireball particle and had it follow the hands of the character.

LIGHTING

I have scattered torches and pot fires across the map to illuminate dark corridors and bring to life the scene. I used an emissive, slow flashing light function material that is attached to the light source to mimic torch/fire flickers.

Light Types
UE4 has four light types; Directional, Point, Spot, and Sky. I have taken the following descriptions from the UE4 knowledge base as I can't word it any better.

Directional lights are primarily used as your primary outdoor light or any light that needs to appear as if it is casting light from extreme or near infinite distances.

Point lights are your classic "light bulb" like light, emitting light in all directions from a single point.

Spot lights emit light from a single point, but have their light limited by a set of cones.

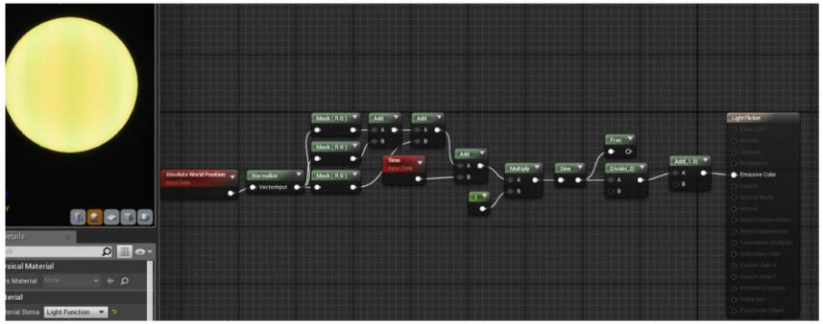
Sky lights capture the background of your scene and apply it as lighting to your levels meshes.

Light Mobility
Lights in UE4 come with three mobility settings; static, Stationary, and Movable. Each dramatically changes the way the light works and their impact on performance.

Static lights have no overhead during a running game and their colour, brightness and shadows baked into meshes. They have an extremely negligible impact on performance as their data is pre-calculated.

Movable lights are the opposite and are fully dynamic. Therefore, they can change all its properties during runtime and make the perfect light source for items carried on the player or moving objects. However, they require the most processing power to render and are only used if necessary.

Stationary lights however, are a happy medium between static and moveable. They can change their colour and brightness at runtime but cannot move, rotate or change influence size. They can cast dynamic shadows and are perfect for mounted light sources such as torches and candles.



CAMERAS

Over a timeline, I have cameras pan across my environment, with some targeting and following certain actors. I have used auto-key when animating cameras. Auto-key draws a path between two locations over the timeline which the cameras follow. The path can have edited through the curve-editor. Unreal's auto-curve works well but, often uses acceleration which is not necessarily the best when switching between cameras. As a result, I have modified the curvature of most paths.

As mentioned previously, it is difficult to make soft camera cuts without transitions and therefore, I must think outside the box. Some possible methods of reducing hard cuts is to have a constant acceleration of the camera between shots or have the player's/viewer's focus on an actor that is present in both scenes. I have used linear curves at the beginning and ending of some shots to make for a smooth transition. I have also centred actors in some shots as well as over populate the scene at the ending and beginning of others to distract the user from the camera cuts.

POST PROCESSING

Torches have a very limited range, and do not illuminate the cavern enough. Using multiple layers of post processing, I was able to give different areas a different hue and brightness. For example, the water room has a natural blue/green hue whereas the main chamber is red and brown. Post Processing allowed me to alter the contrast and add bloom and lens flares. Together these helped to brighten the whole scene as well as make it look a little more believable. I added a dirt mask to the post processing which gives a nice effect when illuminated through lens flares.

In Unreal, Cameras can also have post processing. I have used this only to add the world's post processing. However, as cameras pan through meshes I experienced issues with the depth of field and auto exposure where the scene will darken and then slowly brighten. Some scenes required the camera to pass through multiple meshes and therefore these features proved to destroy immersion rather than add to it. I combatted by temporarily disabling the depth of field and auto exposure when overlapping other actors.

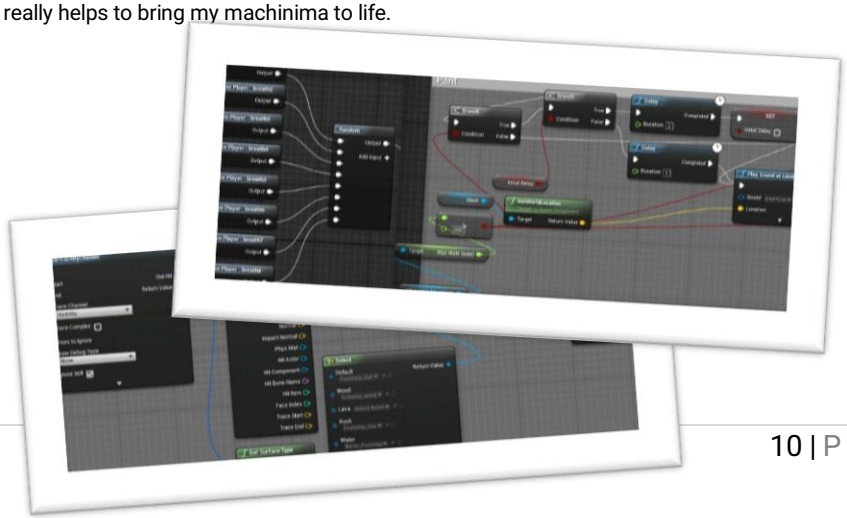
AUDIO

Audio is extremely important in creating immersion but, I am in not a sound designer and had to outsource the audio. Unreal provided some assets such as fire sound effects but, others such as breaths, footsteps and scores were found online.

Footsteps
I added surface dependent sounds to each characters' walking animations. I did this by adding Notify events to the animation at the time where the character's feet hit the ground. In the character blueprint, every time the notify is activated it needs to send a line trace to hit the surface and return the surface type. The surface type is set up in Project Settings -> Physics -> Physical Surface. A physical material needs to be attached to the landscape/mesh that the character is walking on top of. The physical material contains the surface type. When the line trace returns, we do a switch on the surface types and play the selected sound/particle at the character's location.

Breath/Grunt
I have added breath sound effects to the player and grunts to the monsters. When developing the player, I wanted breaths to be subtle and have them programmed to only play after the player has been running consecutively for three seconds. Whereas the grunts are set much like footsteps and activated in the character's animations. I have used a wide variety of breaths and grunts splitting each individual sound into a WAV file and used a music CUE to combine similar sounds. I have set the sounds to select a sound at random. This makes the sounds much more believable as loops cannot be recognised. The different monster types use the same grunt cue but with a different pitch. The troll has a much deeper grunt. This helps to reduce the project size as I am re-using as much as possible.

Scores
I have purchased the rights to use music scores from Devesh Sodha. His music is great and really helps to bring my machinima to life.



USER INTERFACE

User interfaces can make and break games. MMOs largely require complicated interfaces as the user is often given full control of their play through. Linear shooters generally are simpler and only the important and relevant information is displayed i.e. health and ammo.

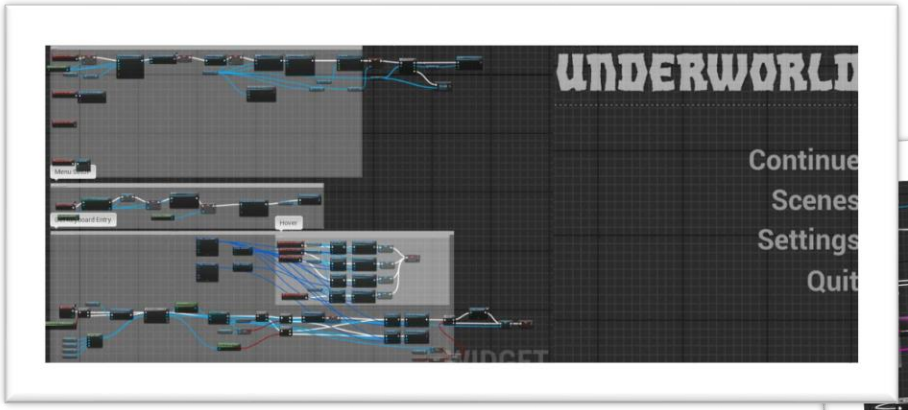
I have not found a genre for Underworld. It plays like a walking simulator yet has many interactive elements. Anyhow, a user interface would take away from the beauty of the environment but, is necessary for storytelling.

The main menu is simple and is scripted to work with a controller and mouse and keyboard. The background has been programmed to change depending on their player's progress. So far there only is one level so I have two cameras that swap every 20 seconds to show this.

I have added a scene selection that is currently programmed to play the cinematic that commences the scene. I have also added a settings menu that allows the player to reduce shadows, textures and post processing effects to increase their FPS.

In-game the user is presented with a crosshair that disappears during cinematics and can be toggled on and off.

Some cutscenes and game events trigger text that provides a background to the player's story and guides the player through the level. The text is programmed to show letter by letter at the speed of a narrator to give the sense that they are being read a story. Altogether, I believe the HUD is not intrusive and allows the player to play and explore at their leisure.



POLISHING

BUGS

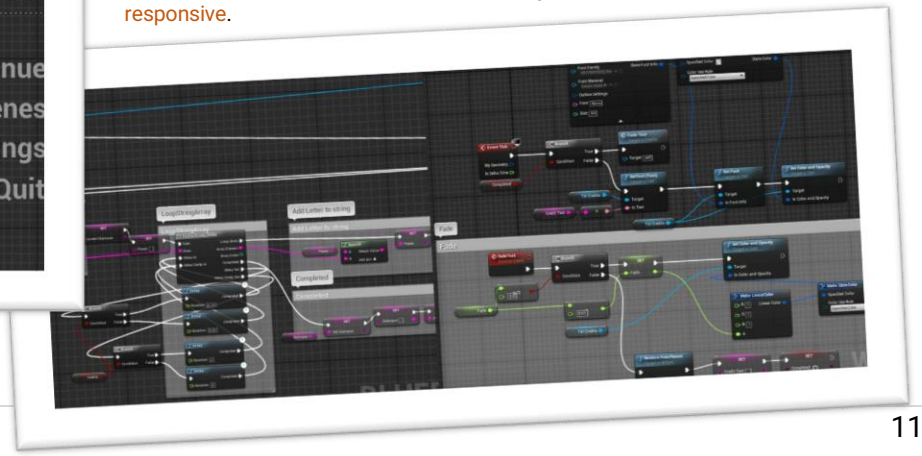
As I add new features, I am always trying to maintain stability. In the game's current state, there are a few issues regarding A.I., ladders and frames per second.

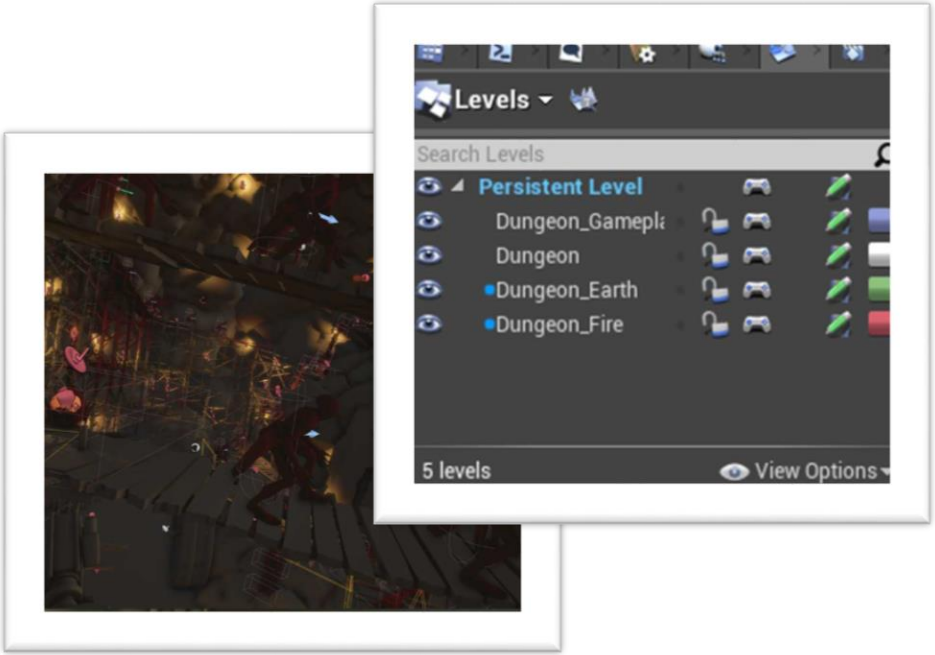
A.I.
My additional avoidance script is not perfect and on some cases, the characters struggle to get past each other,

Ladders
I have measure to reduce issues by only allowing characters that want to use the ladder in same direction, at the same time. However, some characters block the exit of ladder and prevent current climbers from finishing their animation.

FPS
When I started programming, I wrote a lot features around the game tick. Scenes with less content render faster and so have a higher FPS and compute game loops faster.

As a result, events that are based on ticks happen more often in areas where there is a higher FPS and less often in areas with a lower FPS. This a huge problem for me as the character movement is tick-based and so moves slowly in the main cavern. Events are also less responsive.





LEVEL STREAMING

To improve the game's FPS, I learned and implemented level streaming. Level streaming is a method of layering assets that can be quickly swapped in and out.

By using level streaming, I rewrote the game's story using the newly learned features; The player would wonder through a lush cave and have flashes of a time where orcs rule the underworld. So not only would level streaming add to the gameplay, it would also serve as a massive performance boost.

In Unreal, a persistent level is used to group all sub-levels. Sub-levels can either be always loaded or loaded in and out through blueprints and their lighting data either be baked across all levels or be level dependent.

I moved all rocks and props to an always loaded level, moved all foliage and natural lights to a blueprint loaded level and all the AI, fire effects and torches to another blueprint loaded level. All gameplay elements such as triggers were moved to an always loaded level.

Now as the player progresses through the game, Unreal automatically and seamlessly loads in and out levels depending if they are in view or not. It's cool. I also manually toggle levels when in sequences and it has had a noticeable improvement on the game's FPS.

However, every time the A.I. is loaded their behaviour trees must be re-computed causing the A.I. to temporarily break. Until I find a solution, I had to move them to the persistent level.

PROJECT

I have created a short game that is playable with a variety of input devices and across multiple resolutions. Game settings can be changed to modify the graphically quality. I have created multiple cinematics that are triggered within the game or can be played from the scene selector.

I have made use of crowd generation to spawn orcs on a large scale and developed artificial intelligence to allow them to interact with their environment.

I have put a lot of effort into small gameplay features that you may not notice on first play through but together help to create a robust and believable game.

CLOSING

I encountered many problems mostly brought about through my lack of experience with Unreal. I had previously written in C# but, Blueprints was an entirely new language and I had to learn and recognise new terminology.

The huge learning curve meant that I had to take time to learn new features and processes. As a result, I now have excellent experience with Unreal and 3D software.

I pushed all the skills that I have learned at university and industrial placement and have learned new skills and techniques. I have noticed a tremendous improvement in my work and I would love to be a part of team where I can continue learning and pushing my skills further.

When taking on the role of an environmental artist, I invested a HUGE amount of time learning World Machine. I made some cool environments but, I did not end up using my new skills in the final project.

Tackling issues in FPS has been a lot of fun as I have been able to use my knowledge of 3D and programming to understand where and why the performance hits occurred and what can be done to combat it.

I believe I have met all the initial requirements, however, if I was able to take on this project again, I would be able to better streamline my game development pipeline with my newly learned skills.

GOALS – DEVELOPMENT PIPELINE

I have taken on many roles of the game development pipeline throughout the creation of this short game and has helped to reshape the direction of work that I would like to enter. I am a fully capable artist and animator but enjoy taking on the challenges faced by developers. I believe that I would be the perfect technical artist.

I began the project with little experience in each role and worked extensively with Unreal each day to achieve the high targets. I learned the pipeline process for each role by tinkering with Unreal tech demos and free community projects.

The modern animation pipeline is changing as technology advances. The presence of artificial intelligence is becoming increasingly common in film and games. A.I. and programmatic animation has been able to severely reduce development time as it can auto compute time demanding process such as keyframing.

GOALS – TECHNICAL RESTRICTIONS

Technological advancement has led to improving home computers allowing developers to push better graphics and more resource intensive scripts. However, FPS is still a major issue in Game and Film development and throughout development I had to ensure that assets and scripts were refined and optimised.

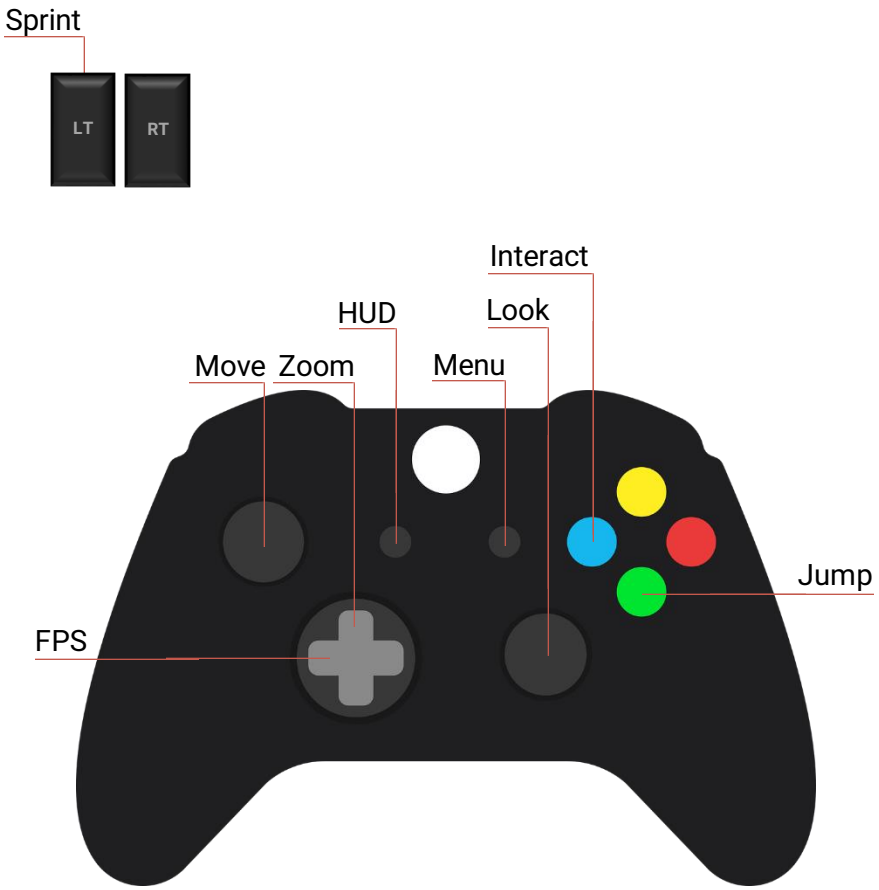
In recent years, facial patterns and speech databases have been used to construct procedurally generated conversations and animations in games. In film, the past few years have seen CG actors commonly replacing distance actors and stunt men. I believe that as technology improves, games and film will become more intertwined as animation becomes more dependent on artificial intelligence.

CONTROLS

MOUSE & KEYBOARD

Look	**MOUSE**
Move	**W** **A** **S** **D**
Jump	**SPACE**
Interact	**E**
Zoom (hold)	**RIGHT CLICK**
Sprint (hold)	**SHIFT**
Show/Hide HUD	**H**
Show/Hide FPS	**F**
Menu	**ESC**

CONTROLLER




aarontrotter.co.uk