CONDUCTOR: Highway-Free Rhythm in VR Spatial Engagement

Engineering Project

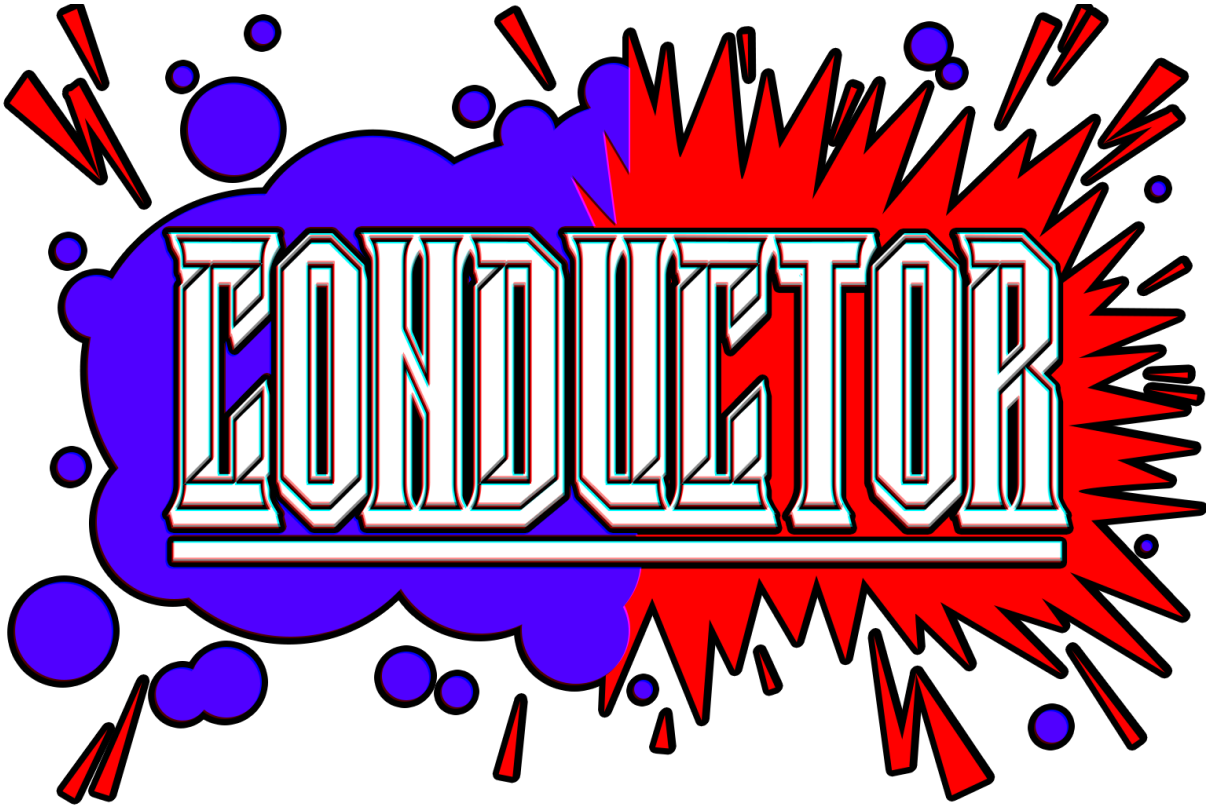

Table of Contents

Introduction **3**
- Problem Space 3
  - Note Highways 3
  - Rhythm Shooters 4
- Solution 5
  - Project Aims 5
  - Solving the Problem 5
  - Game Overview 6

Description and Justification **7**
- Design Requirements 7
- Development 7
  - Development Approach 7
  - Tools 8
  - Game Design 8
    - Enemy Mechanics 8
    - Advanced Combat 9
    - Level Design 10
    - Beat System 11
    - Player Attention 12
  - Technical Implementation 13
    - Beat System 13
    - Dynamic Soundtrack 14
    - Enemy AI 15
    - Interaction 16
    - Player Actions and UI 17
    - Settings 18
    - Pathfinding and Navigation 18
    - Standalone Support 19
    - Shaders and Visual Effects 19
  - Aesthetic Rationale 21
    - Visual 21
    - Audio 22

Evaluation **24**

Reflection **25**

Bibliography **26**

Appendices **30**
- Appendix A. Unreal Engine Terminology 31
- Appendix B. Asset Sources 32
- Appendix C. Design Document and Development Log 33

Appendix D. *Lazarus* Album MP3..... 34
Appendix E. Participant Information Sheet.....35
Appendix F. Participant Consent Form.....39
Appendix G. Research Ethics Checklist.....41

Introduction

Virtual reality (VR) rhythm games use note highways to facilitate rhythm gameplay but deny players the freedom to engage with a 3D environment. To solve this issue, I developed *Conductor*, a VR game that uses rhythm to augment environmental combat without using a note highway. The game takes design principles from rhythm shooters to promote engagement with the environment while performing actions on beat.

Problem Space

Note Highways

Fig 1. Note highway in *Guitar Hero*; Fig 2. Note highway in *Beat Saber*.

As a medium, VR offers users the unique ability to immersively engage in a virtual environment by looking and moving in any direction. Combat games like *SUPERHOT VR* capitalise on this by forcing players to watch their backs and engage with their surroundings, creating an immersive and engaging experience that makes the most of its platform. In the VR rhythm genre, however, games like *Beat Saber* and *Synth Riders* fail to use the full range of motion inherent to VR. These games are built on note highways: linear tracks where beats, indicating upcoming actions, move toward the player. Popularised by the *Guitar Hero* franchise, note highways facilitate accessible gameplay for rhythm games, keeping the player's focus on a linear track with no surprises. In VR, however, they deny the player their full range of rotation and movement, disregarding the core functionality of VR as a platform by restraining gameplay to a single linear track.

While highways persist across all VR rhythm games, some games have attempted to rework their highways to create a more dynamic experience. *Pistol Whip* and *Against* emulate VR shooters by moving the player along a highway as they eliminate enemies to the beat. While initially convincing, these games also fail to utilise the core functionality of the head-mounted display (HMD). Moving the player only functions to disguise the underlying note highway in each level. Enemies will always appear in the same place in the player's immediate view. *Beat Saber* includes a 360° mode that rotates the highway around the player. While this mode is a step away from linear gameplay, it still operates on a single, rotating highway. The player will never be required to look behind them or away from the immediate action, limiting their spatial engagement to the note highway. It only serves to require the player to

rotate left or right from time to time, and never challenges players to the same level of environmental engagement present in other VR games.

The most likely explanations for the persistence of note highways in VR rhythm games are difficulty and accessibility. Placing all gameplay elements in the player's view allows the player to focus on rhythmic gameplay without worrying about stray notes appearing out of view. However, this focus on a single lane of gameplay may not be suited for the platform. As a result of being designed for static 2D screens, note highways disregard the player's ability to rotate and move at will. While this is perfectly functional for 2D media like computer monitors, VR functions by granting the player free movement and rotation at all times. Note highways force players to act as static cameras observing a linear axis of gameplay, separating them from the immersive environmental engagement present in *SUPERHOT VR* and VR shooters. To promote the player's "physical presence for the environment", their body must be "at the centre of the experience" rather than standing at one end (Dongas and Grace, 2023).

Rhythm Shooters


Fig 3. *BPM: Bullets Per Minute*; Fig 4. *Gun Jam*.

The rhythm shooter genre uses rhythm elements to augment traditional shooter gameplay, operating in full 3D space. Games like *BPM: Bullets Per Minute* and *Metal: Hellsinger* involve the positioning and enemy-tracking in 3D space seen in most shooters but add an additional layer of complexity via rhythm elements: players must move, attack, and shoot on-beat. They use rhythmic gameplay as a feature that can be mixed with other forms of gameplay. Dan Da Rocha, developer of rhythm shooter *Gun Jam*, described his game as containing "a lot of the familiar themes… It's just that there's this new explicit rhythm layer to it that shakes up the play style" (as quoted in Henley, 2021).

Unlike the VR rhythm genre, rhythm shooters are designed for 3D environments. Enemies can attack from any angle, requiring players to pay attention to their environment and position themselves accordingly. Notably, this genre does not rely on note highways to assist players in rhythmic gameplay, proving the viability of rhythm elements as an augmentation of gameplay rather than a static form of gameplay. Rhythm shooters display the potential of rhythm game elements in 3D environments, providing an exemplary set of design principles from which a VR rhythm game could be developed.

Solution

From my problem space, I created a list of project aims and objectives, which I used to conceptualise and develop a VR rhythm combat game.

Project Aims

- To develop a highway-free VR rhythm combat game that promotes engagement with 3D space
    - Develop a rhythm-based, highway-free movement system
    - Develop a combat system that engages the player with 3D space
    - Develop a game to test the efficacy of these systems
- To explore key design principles for developing highway-free VR rhythm games
    - Identify level and gameplay design principles for highway-free gameplay
    - Identify a workflow for developing dynamic soundtracks in non-linear rhythm environments

Solving the Problem

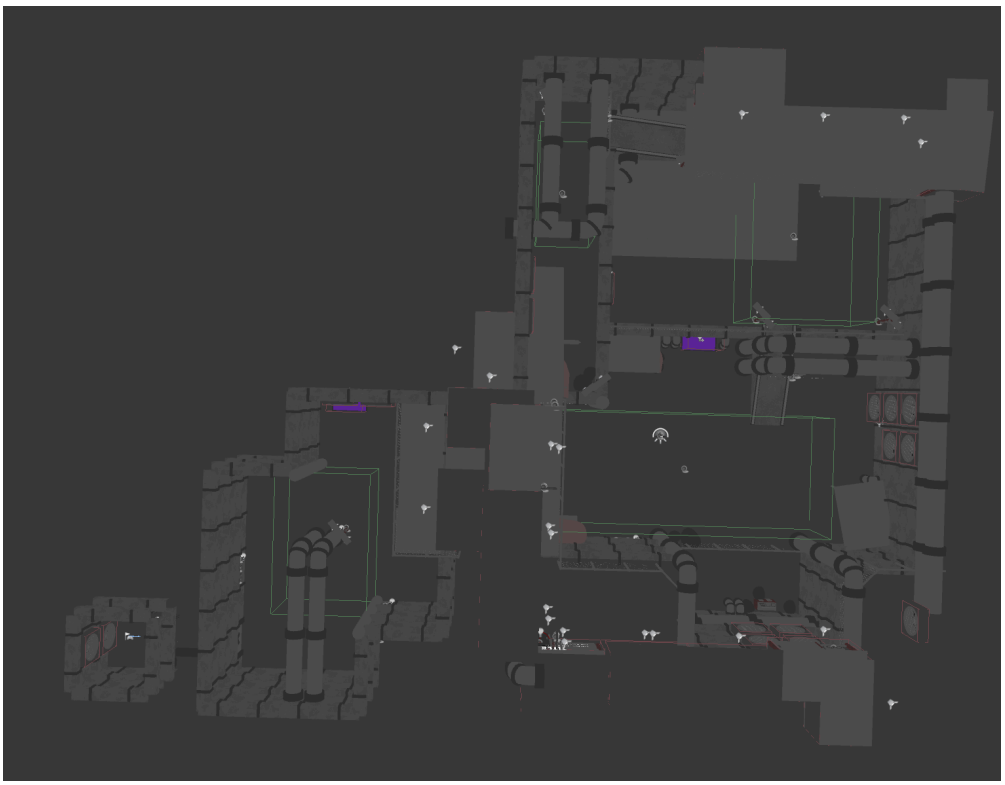

Fig 5. Top-down view of an explorable in-game environment

Following design principles from rhythm shooters, I developed a game fully built around 3D gameplay. Note highways are not used; gameplay is not restricted to a single lane. The game is built for standalone Quest 2, so players are not bound to a single lane of play by a wired connection. Full 360° environmental combat tests the player's ability to operate rhythmically while engaging with the environment by positioning themselves, manipulating interactables, and dodging enemy attacks to the rhythm.

Unlike rhythm shooters, the player must use the environment's resources to defeat enemies, further testing their ability to engage with the environment. Different enemy types must be defeated with different tactics. They test the player's movement by maintaining different distances from the player and strafing out of the player's immediate view. A dynamic soundtrack reacts to the game's events: gameplay does not follow music; music follows gameplay. Rhythm elements enhance 3D gameplay rather than dictating the flow of gameplay. Seven playtesters provided feedback on the game to determine its successes and failures.

Game Overview

Fig 6. *Conductor* logo.

Conductor is a cyberpunk-horror VR rhythm game where the player must electrocute waves of enemies to the beat by manipulating electricity and liquid. Players defeat enemies by drawing electricity or liquid from "element sources" in the environment and hitting enemies with the controlled element. The player, enemies, and projectiles all perform actions on the rhythm, creating staggered beats of gameplay that require quick thinking from the player. Different enemies and environments pose different gameplay challenges, requiring the player to work with their surroundings to dodge attacks and defeat enemies. Over time, the player will learn advanced combat techniques which can be used to clear different enemies and waves more efficiently. A dynamic soundtrack follows the gameplay, emphasising exciting moments and warning the player about enemies on the field. A high-contrast art style keeps players focused on enemies and interactables, making the game readable and easy to learn. The game includes a main menu, a tutorial, two levels, and configurable gameplay and accessibility settings.

Description and Justification

Design Requirements

Design decisions were made in accordance with the following Functional and Non-Functional Requirements (FRs/NFRs). These are referenced wherever they were used to inform design choices. Table 1 lists requirements as drawn from my research and their justification.

Req. Name	Requirement	Justification
FR1	Wireless VR game	Avoid physically restricting movement to a single lane with a wired connection
FR2	Environment-focused gameplay requires players to engage with their full 360° range of motion	Following rhythm shooters' example, test players' ability to engage with a 3D environment
FR3	Music/rhythm-enhanced gameplay	Use rhythm to augment rather than dictate the flow of gameplay
NFR1	Gameplay fully holds the player's attention without any external distractions	Without note highways, players will have more difficulty focusing on rhythmic gameplay
NFR2	Difficulty is suited to VR and accommodating to inexperienced players	Forgiving gameplay and controls will help players focus on the environment and enemies
NFR3	Cyberpunk/metal/industrial theme	Adhere to VR rhythm game and rhythm shooter aesthetics to best communicate the game's identity

Table 1. Design Requirements.

Development

Development Approach

Gameplay took priority during development (NFR1). Features like cutscenes, narrative, and high-quality graphics were cut to prioritise gameplay quality.

I split development into three stages: system development, prototyping, and content. System development took the longest, as the game's systems were designed to be adaptable, robust, and easily modifiable. This made the prototyping stage very fast, as it only required tweaking values and player abilities. This stage was most important to ensure quality gameplay, and I finalised all core gameplay features before proceeding (NFR1). The content stage then consisted of improving quality of life, fixing bugs, and creating levels, soundtracks, and new enemies and interactables.

Tools

I developed the project using Unreal Engine 5.3, targeting standalone Quest 2 (FR1). I initially used the *VR Expansion Plugin* to jumpstart the project but later restarted the project without it to improve performance and create all gameplay features from scratch (FR1). I created models and animations in Blender and textures in Photoshop, GIMP, and Aseprite. Music and sound effects were produced in Logic Pro using synths and samplers like *Massive*. I used GitHub for source control and kept an extensive history of builds to ensure backups and older versions were always available.

Game Design

I took great care during development to ensure every design decision was supported by my design requirements, resulting in a cohesive, focused game. My design process is heavily documented in Appendix C.

Enemy Mechanics

I designed and implemented five enemies: the Bishop, Butcher, Hammer, Squid, and Plutocrat. Each enemy fills a unique role in combat, creating dynamic wave-based encounters based on the environment and enemies in each wave (FR2). Enemies move toward or away from the player when outside their desired range and strafe around the player when at their desired range. This prompts the player to move frequently to bunch up enemies and watch various angles for strafing enemies. Mixing these two tactics will allow the player to manage enemy positioning while also taking moments to locate and interact with element sources.

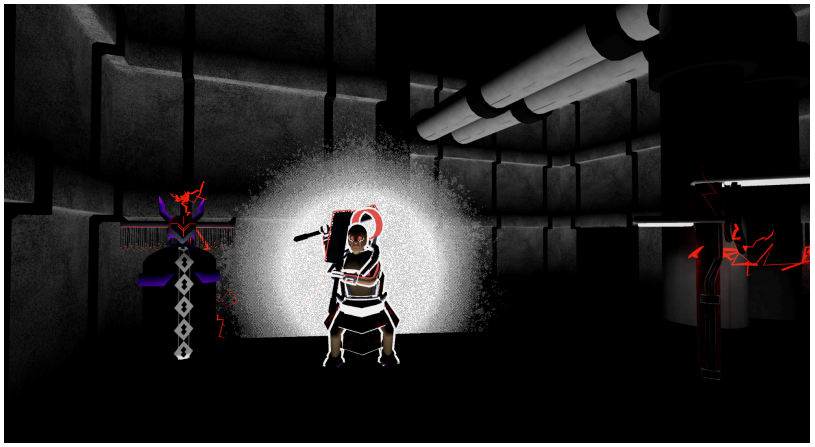

Fig 7. Bishop and Butcher.

The Bishop is a ranged robot enemy that can fire electric beams from a long range. When approached, it flees from the player. Unlike melee enemies' attacks, players can dodge Bishop beams simply by ducking/tilting their heads.

The Butcher is a mid-range human enemy that throws two knives in quick succession when attacking. It does not attack often when other enemies are around but can deal heavy damage if ignored. Butchers can be picked up and thrown to buy time (FR2).

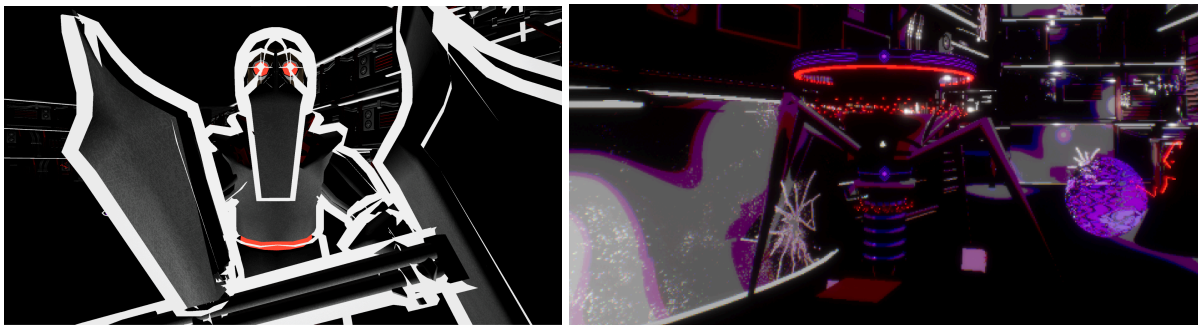

Fig 8. Hammer; Fig 9. Squid.

The Hammer is an aggressive human melee enemy that charges the player, only attacking when close. It takes movement and attack priority over all other enemies, creating focused, dramatic 1v1 duels between the player and the Hammer (NFR1). The Hammer moves every beat, requiring an immediate response.

The Squid is a large, threatening robot that can attack the player from any range for double the damage of a regular enemy. This enemy cannot be electrocuted with liquid alone, so players must find their own means of creating an electrocution to defeat it.

The Plutocrat is a defensive robot that blocks off areas with force fields, defending enemies near/behind it and preventing the player from reaching element sources. While not an aggressive enemy, players may need to prioritise defeating it depending on its position in the world (FR2). Its force field activates and deactivates every four beats, so players can predict when they will be able to pass the Plutocrat (FR3). The player can reach through the force field with controlled liquid to electrocute it.

Advanced Combat

While players can defeat any enemy with electricity, liquid, or both, there are many more efficient, creative, and expressive ways to defeat enemies. This encourages players to master combat and use quick problem-solving to clear waves (NFR1).

Player movement is key to efficiently defeating enemies. The player can dodge ranged attacks by simply ducking or tilting their head without repositioning; this is especially useful when evading enemy attacks from behind or approaching an enemy (FR2). Players can “stab” forward with their controlled element to defeat enemies just out of reach or corner them to prevent them from evading. After moving to the controlling hand’s target position, controlled elements will continue to follow the hand for a split second. This allows skilled players to perform swift, sweeping motions to defeat multiple enemies in a single beat and promotes rhythmic movement (FR3).

Bishops can be beneficial to the player in many ways. Their beams will kill any enemy they hit, meaning players can hide behind other closer-range enemies to thin the crowd (FR2). Bishop beams also electrify chainable interactables they hit, meaning the player can use Bishops to create sources of electricity on objects like street lamps. Bishops themselves are a moving electricity source the player can use to electrocute nearby enemies (FR2).

As humans are mostly water, the player can grab any human enemy to prevent it from attacking or to use it as a meat shield to protect themselves from ranged attacks. By flinging their hand and releasing, the player can throw the enemy, buying time and reordering enemies. This has different uses with different human enemies. The Butcher throws two knives in quick succession, so the player can throw it to the back of a crowd of enemies to make its knives hit and kill nearer enemies (FR2). The Hammer will not initially attack the player until within close range, meaning the player can throw it to buy time and prevent it from attacking (FR2). However, this can be a risky manoeuvre: if the Hammer gets too close to the player, it can leap attack right back to the player after being thrown.

Liquid has more uses than directly defeating robots. Any wettable interactable touched by liquid will become wet, priming it for electrocution if touched by electricity in the future. This is especially useful against the Squid, as touching it with liquid before electricity will electrocute it. Human enemies can also be picked up by controlled liquid. This allows the player to move multiple human enemies, prevent them from attacking, and electrocute them simultaneously (FR2). Using liquid this way is especially efficient when a robot enemy is on the field: the electrocution caused by touching them with the liquid will kill all humans inside. Grabbing an enemy with liquid mid-attack will not cancel its attack, so players must either wait to grab the enemy or dodge the attack by carefully moving the liquid.

Through combat, the player will learn that electrocutions are area-of-effect attacks that kill any enemy inside; by using the correct element against different enemy types, players simply create electrocutions, and electrocutions are what then defeat enemies. With this in mind, players can aim to create larger, more effective electrocutions, defeating multiple enemies in one action. Electrocutions performed on large interactables like the Squid, Plutocrat, and fountains can defeat all nearby enemies. This encourages players to bunch enemies together by luring them to a target area (FR2). The Plutocrat can also be used to the player's benefit by blocking ranged attacks and segmenting waves into smaller, more manageable sets of enemies. Enemies inside its force field will usually stay inside, allowing the player to electrocute nearby enemies by defeating a Plutocrat inside a force field. Clearing multiple enemies with a single electrocution is especially useful when element sources are difficult to access (NFR2).

Level Design

Levels are designed around arenas where combat takes place. These range in size from open courtyards to tight alleyways, creating different challenges based on the enemies present in each area. Electricity and liquid sources can be found on the ground, walls, or ceiling to vary gameplay (FR2). Difficulty can come from enemy mechanics alone or the challenge of reaching a source while avoiding attacks.

The game begins with a short tutorial explaining the basic mechanics and an introduction to combat with a robot and human enemy. This fairly forgiving area allows players to retry without losing much progress as they learn to dodge attacks and defeat different enemy types.

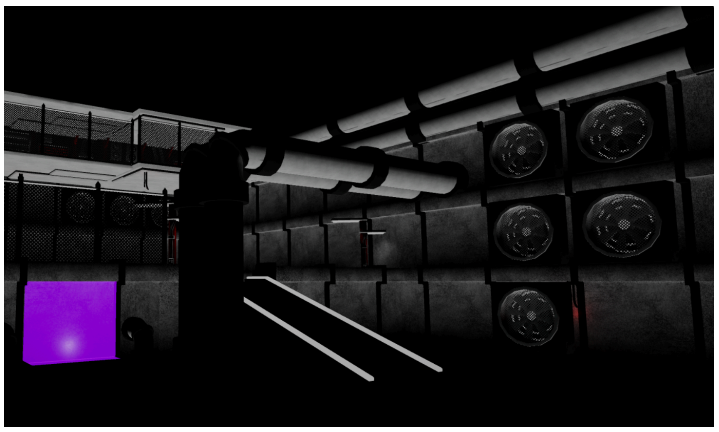

Fig 10. Industrial.

Industrial, the first level, is an abandoned industrial area turned slum that introduces combat to the player. The first few waves take place in a small room with a pillar in the centre, teaching the player to use the environment to block incoming ranged attacks and buy themselves some time to find electricity and liquid in the surrounding area (FR2). The player moves deeper into the industrial area, fighting enemies in a series of open courtyard arenas. These get progressively smaller, making the final fights against Hammers and Butchers especially claustrophobic.

Fig 11. Inner City.

Inner City, the more difficult second level, takes place in an open city square surrounded by cyberpunk apartments. Most of this level is in the same arena, with progressively difficult waves spawning over time. Inner City introduces the player to the final two enemies: the Squid and Plutocrat. These larger enemies are more suited to this level's open space for their jump attack and large force field (FR2). After clearing all waves in the main square, the player continues around the corner to a short, tight alleyway to defeat a handful of human enemies before completing the level.

Beat System

Gameplay only progresses on beats, segmenting actions into timed rounds. Inspired by *Crypt of the Necrodancer*, players can quickly assess what each enemy has done each beat and respond accordingly on the following beat (FR3). To make this more manageable for the player, enemies telegraph their attacks a beat or two before

attacking. The player can queue actions like moving or grabbing to occur on the following beat (NFR2).

Each level's beats per minute (BPM) is a major difficulty factor, increasing gameplay speed and the reaction time required from the player (FR3). Following the success of *BPM: Bullets Per Minute*'s developers, I used 88 BPM as a reference point and slightly modified it per level, keeping the increased difficulty of VR gameplay in mind (Awe Interactive, 2022).

Player Attention

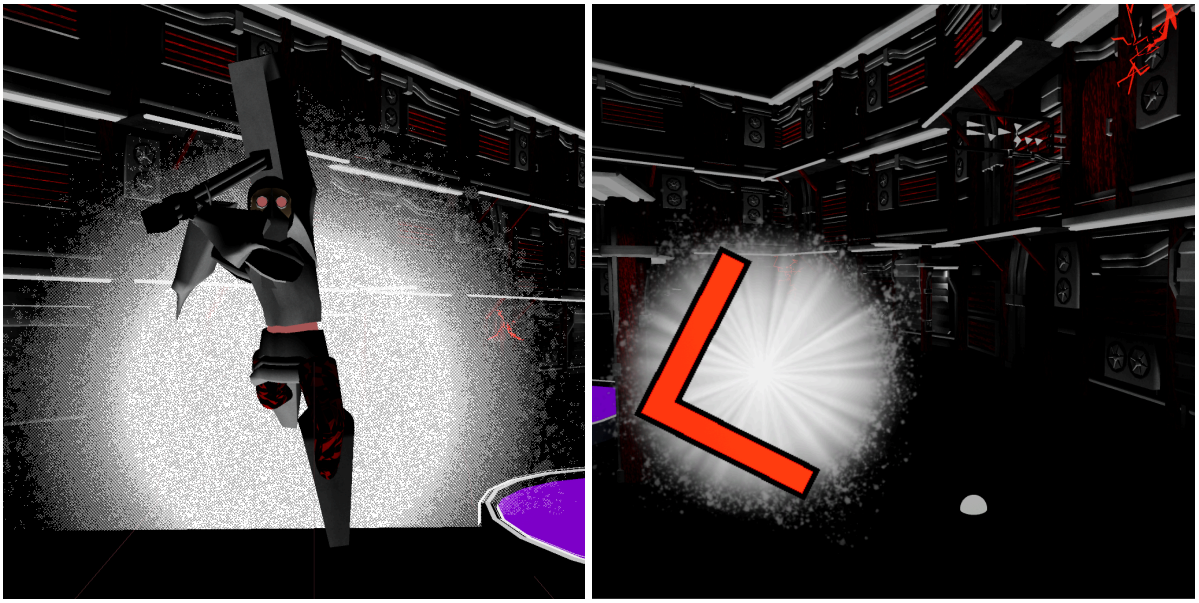

Fig 12. Attack warnings.

Visual, audio, and haptic feedback give players information about in-game events (NFR1). When the player encounters a new type of enemy for the first time, it will display its type and how to defeat it above its head. The player can also make these hints visible for every enemy or permanently hidden via the settings menu. The environment's LED lights and controller haptics thump to the rhythm to help the player stay on beat; the controller haptic beat thump was adapted from *Pistol Whip*'s haptics. Incoming enemy attacks are indicated by a bright white flash, a dramatic pose, and screeching synths. Enemies also indicate their attacks a beat before this, but more subtly; this gives additional warning to players paying close attention. UI indicators point to off-screen enemies and warn the player of incoming attacks with the same flash effect enemies produce before attacking. To reduce visual clutter, these indicators can be set to fully disabled or only shown for incoming attacks via the settings menu (NFR1). When the player is damaged, their remaining health is shown on-screen with accompanying visual and sound effects to ensure the player knows they were hit.

Moving enemies and projectiles smoothly slide to new positions and leave a particle trail behind them to help the player understand where they moved to/from (NFR1). This is crucial due to the beat-based gameplay; without a clear movement path, enemies would appear to teleport to their new positions, confusing the player and disconnecting them from the environment.

Technical Implementation

As a heavily technical project, *Conductor* required robust systems for rhythm and music, enemy AI, actor interactions, and player controls and configuration. My development and iteration process is heavily documented in Appendix C.

Beat System

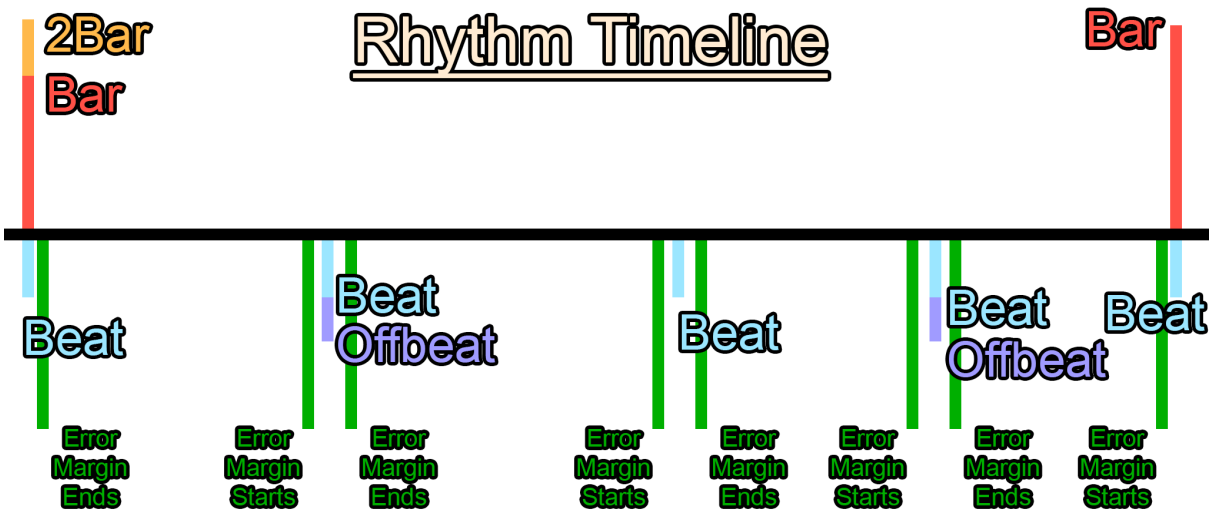

Fig 13. Rhythm timeline.

I used Unreal Engine's Quartz, a system for accurately tracking BPM outside of regular Tick functions, for rhythm-synced events. BP_BeatManager calls event dispatchers at regular rhythmic intervals like Beats, Offbeats, Bars, and 2Bars (the start of each two-bar loop). Actors can then subscribe to these rhythm bindings to receive events at rhythmic intervals. Subscribing to rhythm bindings via BP_BeatManager is handled with a global function library so any actor can easily implement rhythm events. Events like OnBeat usually replace Tick since updates are often unnecessary between beats; treating each beat as a global Tick is also beneficial for performance (FR1).

Using Quartz's BPM-synced events with fire-and-forget audio tracks (covered below) introduces some limitations. Pausing the game safely is difficult: the player may lose their rhythm, and the BPM will become unsynced with longer-looping tracks. It is also unsafe to predict upcoming beats without using one of Quartz's subscribable rhythms, meaning events leading into the next beat cannot stay synced with the BPM clock. BPM-synced events can occur anytime, meaning references to enemies may become invalid if they are killed midway through executing a function; I solved this and similar issues by catching invalid reference cases wherever necessary.

Each beat and offbeat, BP_BeatManager sends updates to MPC_Colors, a material parameter collection that allows materials to change colour and intensity in sync with the beat. Most of these effects were removed later in development to remove environmental distractions (NFR1) but are still visible in LED materials and almost any material with colour. Using colour values from MPC_Colors also allows for easily configurable colour schemes, though this is not made available to the player (NFR1).

I began work on a rhythm system to allow different levels or boss encounters to work on different rhythms, forcing the player to learn new rhythms and adding unique gameplay challenges (FR3). While partially implemented, I scrapped this system to ensure players were not caught up in complex rhythms and instead focused on the environment/positioning side of gameplay (FR2, NFR2).

Dynamic Soundtrack

The soundtrack adapts to in-game events such as enemy types, controlled interactables, and combat intensity (FR3). This is handled with multiple tracks with configurable looping and playback conditions.

DT_MusicLevels contains a list of SMusics containing information per level such as the BPM, rhythm, and SMusicTracks. SMusicTracks are individual tracks with playback conditions. Each level selects its music information by name from DT_MusicLevels and adds tracks stored in UNIVERSAL, a SMusic including non-BPM-synced tracks like single-hit drums and the enemy incoming attack track. Tracks in UNIVERSAL are shared across all levels. Despite using different tracks per level, enemy tracks always use the same instrument to build enemy identity while still providing variety across levels (FR3).

Track playback is handled by BPC_TrackManager, a component of BP_BeatManager. When the Quartz clock begins or when explicitly updated, BPC_TrackManager iterates through all tracks the level can play, attempting to play any track with playback conditions met that is not already playing. When a track is played, a BP_Track actor is spawned that handles properly looping and deleting itself if its condition is no longer met. BPC_TrackManager maps references to all BP_Tracks to their IDs, allowing for quick checks to verify tracks are not already playing. While track conditions could be checked every beat, this system is entirely event-driven, prioritising performance by bypassing unnecessary playback attempts across the full list of tracks (FR1). Actors can attempt to play tracks by their playback condition via the global function library, meaning integrating new playback conditions with the event-based system is straightforward.

Enemy AI

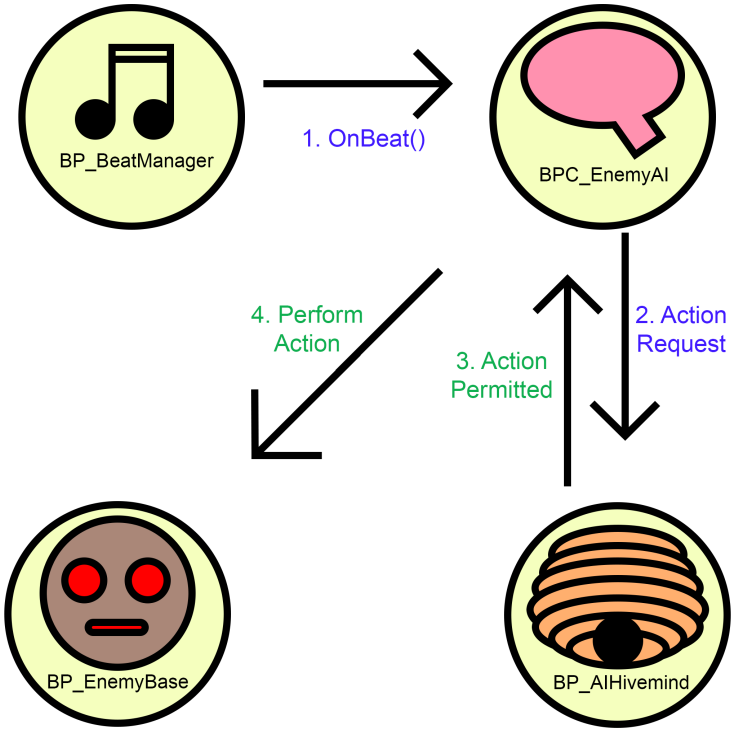

Fig 14. Enemy action request pipeline.

Enemy AI is structured in multiple parts: BP_EnemyBase, the physical enemy; BPC_EnemyAI, the enemy's "brain"; and BP_AIHivemind, the central hivemind managing all enemies' AI. This structure abstracts enemy intentions from their practical actions and allows for easy customisation on a per-enemy basis. Enemy behaviour is dictated by weighted actions which, when paired with other characteristics such as attack cooldown time, produce enemies that act with varying levels of aggression, avoidance, and importance in group settings (depending on the enemies they are paired with). For example, the Squid's attack is weighted very heavily, so it will typically get priority to attack. However, its movement weights are low, meaning other enemies will usually get to move instead.

Weighted actions on BPC_EnemyAI include attacking, strafing when at a desired range from the player, moving to/from the player when outside that range, and special actions. Each beat, each BPC_EnemyAI determines the highest-weighted valid action it can perform. It then sends a request to BP_AIHivemind containing the desired action and its weight.

BP_AIHivemind sorts action requests by their weight and permits actions from the highest weight downward per action type. It stops approving requests once the maximum concurrent number of an action type is reached. For example, if three enemies want to move while inside their target range from the player and the maximum number of requests for that action type is two, BP_AIHivemind will sort them by weight from highest to lowest, then approve requests until two requests are permitted. The third enemy will not be allowed to move. The hivemind prevents the player from being overwhelmed by too many concurrent actions (NFR2).

Once BPC_EnemyAI's request is approved, it fires an event on BP_EnemyBase corresponding to the action type. BP_EnemyBase then decides how to perform the attack. This structure allows enemies to treat different actions differently – namely, attacks. Enemies create attack patterns from 4 attack stages: Tell, when the enemy begins the attack; Preattack, when the player is warned that the enemy is about to attack; Attack, when the enemy performs the damage-dealing action; and Recovery, when the enemy rests a moment after attacking. For example, Butchers attack twice in one attack with the attack pattern Tell, Preattack, Attack, Preattack, Attack, Recovery. Events are also fired on BP_EnemyBase for each stage to allow for further customisation; the Butcher uses this to grab two new knives during the Recovery stage, while the Squid leaps into the air during the Preattack stage. This structure makes adding new enemies and attacks quick and straightforward.

BP_EnemyWaveSpawner spawns waves of enemies when the player enters its hitbox, spawning consecutive waves after all enemies are cleared. This uses SWave structs containing an enemy count and an EEnemyType, an enum containing a list of all enemy types, to facilitate wave spawning in levels. It also toggles actors, such as doors and GO! arrows, on spawning the first wave and once all waves are cleared through a level-specific referencing system.

Interaction

Non-destructive interactions between the player, enemies, and environment are handled by the BPI_Interactive interface, which passes events to BP_InteractiveBase and BPC_Interactive. Interactions typically fire from collision events, where an interactable object tells the hit object what kind of interaction it is performing. BPC_Interactive reacts to interactions by adjusting its properties accordingly. Interactive properties determine what interactions the object can make/receive and how it responds to interactions. Properties are listed below.

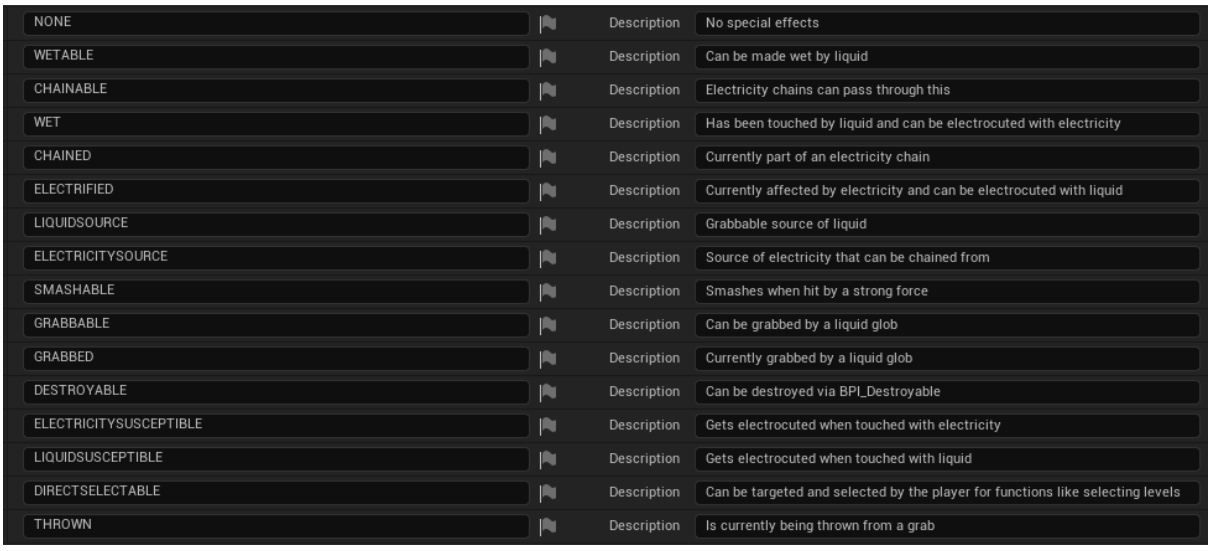

Fig 15. Interactive properties.

Destructive, directly damaging interactions are handled by the BPI_Destroyable interface. Damage dealt this way can be directed at the player, enemies, and/or environment. This allows interactions like enemy attacks damaging other enemies and player electrocutions destroying environmental obstacles. This is also used to

prevent the player from accidentally electrocuting themselves, as electrocutions will only damage enemies and the environment. Destroyables decide how to respond to taking damage or being destroyed. Enemies create particles and despawn when destroyed. The player instead sees a game over screen.

BP_InteractiveBase is a base class for all enemies, selectables, and element sources. It stores references to all mesh components, enabling multi-mesh material effects like outlining and wetness. It also features a configurable “Interaction Transform” offset from its root pivot, which defines the point at which the player can interact with it. Enemies automatically set this to the centre of their capsule collision, ensuring correct electricity chain positioning and centred movement when grabbed directly by the player or indirectly by player-controlled liquid.

Player Actions and UI

The player can snap turn with the left joystick, dash using the right joystick, and control electricity or liquid using the triggers and grips. While actions can be queued before the beat, players can also perform them on-beat if within the Error Margin, which lasts 1/32nd beat before and after each beat. The Error Margin uses Quartz rhythm bindings to be as accurate as possible, meaning no predictions from outside the Quartz system are required. The end of the Error Margin is also used to deal damage from enemies, allowing the player to dodge certain attacks on the same beat if done quickly enough (FR3).

Controlling electricity and liquid is managed with BP_AimTargeter, an actor attached to each hand. This actor uses a large capsule collider to track all overlapped interactables and filters the targetable interactables by the hand’s current targeting mode. A “default” targeting mode determines what the hand can target/control by default, while a separate “current” mode controls what is currently targetable; the current mode is affected by whether the hand is currently controlling an object and, therefore, unable to control another object. This was designed to allow quick modifications to player control abilities later in development, which was invaluable when collecting playtest feedback (NFR1).

When choosing which interactable to target, BP_AimTargeter finds the average position of all overlapped interactables and targets the closest interactable to that point. This helps players lock on to the interactable they are most likely trying to control (NFR2). The targeted interactable displays a beat-synced highlight effect to indicate it is targeted. This was created by duplicating the actor’s meshes, syncing their transforms and animations to the originals, and applying a material with inverted normals and a world position offset. I chose this solution to avoid using a post-processing layer, which would hurt standalone performance (FR1). When controlling an interactable, the player will pick it up if it is grabbable, draw electricity or liquid from it if it is a source, or select it if selectable (e.g. Exit doors, settings). Each beat, controlled interactables lerp to the position of their corresponding hand’s cursor, triggering interactions along the way. This allows players to perform sweeping motions to chain or wet multiple interactables along the controlled interactable’s path (FR2). Controlled interactables also update their target position until the end of the beat’s error margin, allowing players to quickly correct their path or perform wildly sweeping manoeuvres if quick (FR3, NFR2).

UI is handled with two separate heads-up displays (HUDs): one for “area cleared” messages, health and damage, and debug info during development and one for tracking enemy positions with enemy indicators. Enemy indicators are generated per enemy and rotate to point toward the enemy, allowing the player to track off-screen enemies (NFR1). As world-to-screen-space calculations are unavailable for VR due to its use of two cameras, I implemented my own solution. On Tick, each enemy indicator finds the world-space directional vector from the player to the enemy and the player camera’s forward vector, using arccos to find the final camera-relative angle and checking the cross product of the two vectors to determine the angle’s sign. This angle is then used to rotate the enemy indicator on-screen. Enemy indicators begin fading out once their angle’s absolute value is below a certain threshold and completely disappear below a lower threshold. These values were tweaked to match the Quest 2’s field of view, resulting in enemy indicators that fade away as the player turns toward enemies to prevent visual clutter (NFR1). I created a red, flashing animation using Unreal Engine’s Widget animation system to clearly indicate when enemies are about to attack (NFR1).

Settings

A settings menu accessible from the main menu allows players to tweak various accessibility and gameplay settings (NFR1, NFR2). These include configuring enemy indicators, tunnel vision, hints for defeating different enemy types, invincibility, how elements are consumed when creating electrocutions in-game, and post-processing (unavailable on Quest 2 for performance reasons (FR1)). To prevent the player from being overwhelmed with options, I only added settings I believed to be player-dependant (NFR1). There is also an option to restore default settings.

I implemented the settings system with JSON-like string and int struct pairs. Each setting consists of a string ID and an int Value, allowing for any number of options under a single setting. Values correspond to an array of options stored by each setting interactable in the settings menu and use 0 as a default value; the “restore defaults” interactable deletes all settings, which are then immediately set to 0 if nonexistent on the player’s save when an actor first attempts to access them. I used a global function library to work with the save system, allowing any actor to easily retrieve a setting’s value or save a new one. Settings are saved in a save file, persisting across levels and game restarts. Setting a new setting value immediately saves it and applies changes, so players can quickly preview settings like tunnel vision or post-processing without leaving the settings menu.

Pathfinding and Navigation

Beat-based gameplay removes the need for real-time gravity and physics, so I added my own methods of collision-based movement instead for performance (FR1). When a traveller (the player or an enemy) intends to move to a new position, it first raycasts from its current position translated upward by its steppable height toward the intended movement direction. If the raycast collides, it backtracks by the radius of the traveller to ensure the traveller would not teleport into a wall. Otherwise, it travels the traveller’s intended distance. That position is then grounded by raycasting downward and finding the highest floor below the position within reason; there is a maximum limit on this for performance (FR1). Finally, the traveller smoothly lerps to the location, indicating its path of movement to the player. Projectiles use this

method without grounding or checking step height, and the Squid uses a similar function with a height check to leap into the air without clipping through ceilings.

Enemies and the player verify navigable points using a navmesh. When enemies attempt to move to a position but find they would not typically be able to navigate there (such as if the raycast travelled past an unnavigable wall), the enemy instead attempts to move to a random navigable position to free itself from unnavigable areas. If that also fails, the enemy opts not to move and instead waits until its next permitted move action to try again; this is to prevent excessive pathfinding checks affecting performance (FR1). Unfortunately, Unreal Engine's navmeshes generate both outside and inside of colliders, meaning these cases must be handled despite ideally never occurring in-game. Otherwise, an enemy would have no issue with teleporting inside a collider if its movement raycast allowed it.

Standalone Support

To ensure players were not limited to a single lane of play by a wired connection, standalone Quest support was necessary (FR1). This was the most difficult technical undertaking of the project, requiring me to sacrifice post-processing, depth field and refraction shader effects on liquid, real-time lighting, and detailed environments. However, the game's performance has greatly benefitted from removing these features, allowing the headset to livestream footage so I could monitor playtests and better understand players. I took additional steps, such as disabling additional render passes, creating a shared master material, and limiting pathfinding and AI checks, to improve performance (FR1).

I suited my workflow for standalone Quest by using Android Vulkan to accurately preview lighting in-editor and implementing generic first-person movement that can be controlled by mouse and keyboard when not connected to an HMD. I frequently made test builds to ensure parity between the editor and APK (FR1).

Shaders and Visual Effects

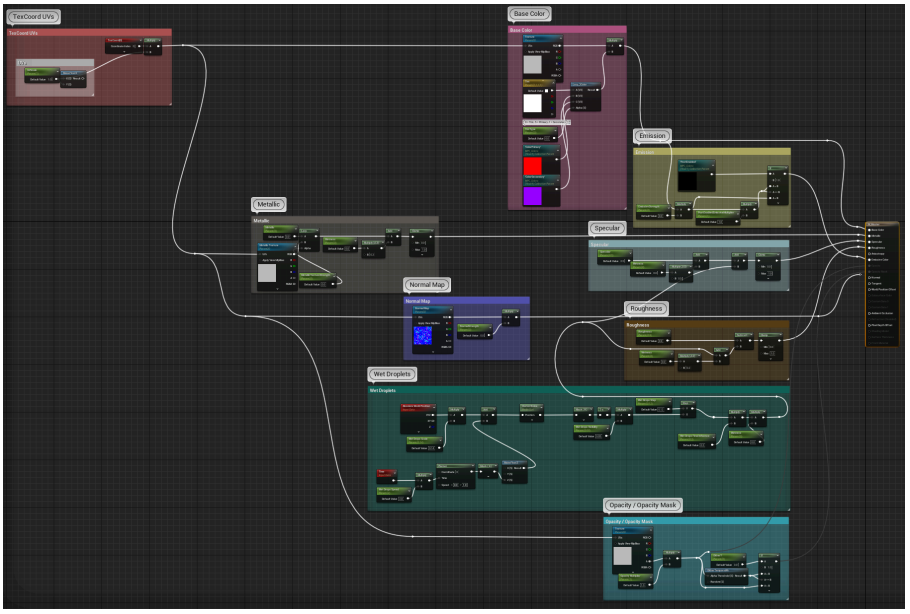

Fig 16. M_Master.

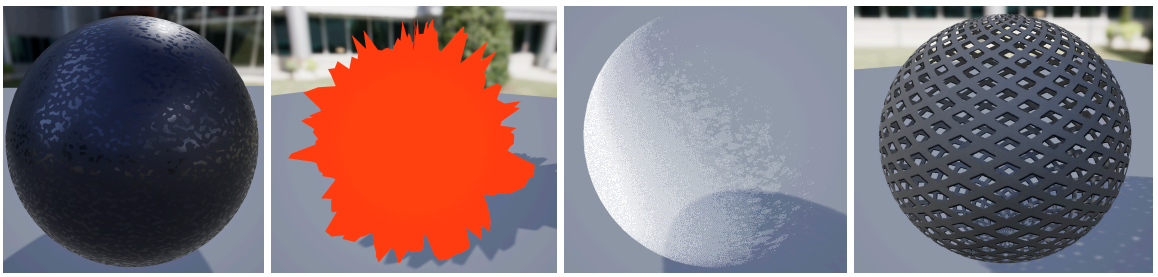

Fig 17. Materials.

Almost all materials in the game are instances of M_Master, a shared master material. This allows materials to use effects like beat-synced emission, colours from the active colour scheme, and easily-configurable metallic maps. M_Master handles wetness across any child material instance via a simple Wetness parameter, using world-space 3D noise to create shiny “droplets” that slide down the sides of meshes. It facilitates material creation, allows for shared properties across similar material types, and improves performance (FR1).

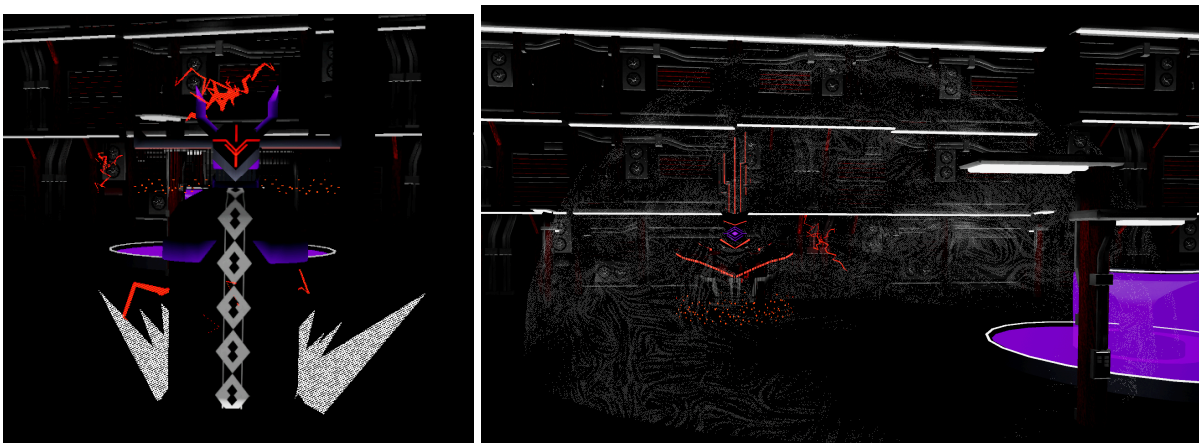

Fig 18. Spawn effects; Fig 19. Plutocrat force field (more easily visible in VR).

Effects like enemy spawn effects, attack warnings, and the Plutocrat’s force field use dithering instead of alpha blending to improve performance (FR1). These effects also use rotating meshes instead of panning their texture to save shader instructions on M_Master (FR1). The enemy spawn effect was made using rotating triangles with inward-facing normals, creating an eye-catching flash behind and around the enemy while keeping the enemy visible (NFR1).

Liquid and electricity use World Position Offset effects to manipulate vertices, creating distinctive wavey and spiky effects. Electricity particles create a single ribbon particle that chains between two points, offsetting its vertices following a curve that increases “spikiness” toward the centre of the chain. When controlling electricity or liquid, the player’s hands emit a glow with a colour matching the controlled element; this is accomplished by creating a dynamic material instance and setting its tint to match the primary or secondary colour from the active colour scheme.

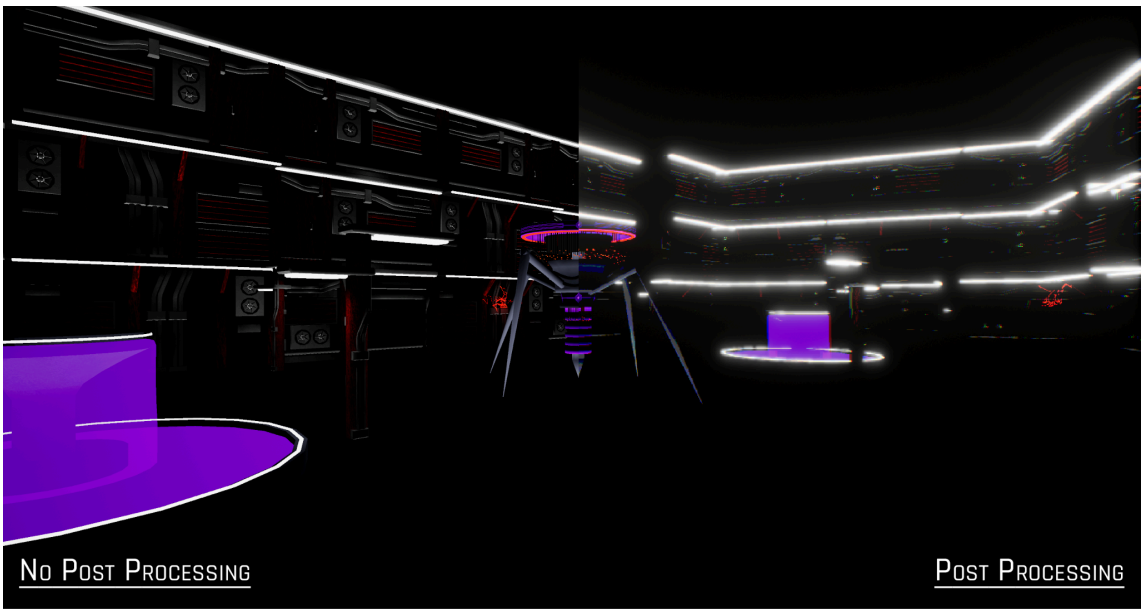

Fig 20. Post-processing comparison.

While post-processing is unavailable on standalone Quest, it is toggleable in the settings menu on the Windows build. The post-processing enforces the game's noir cyberpunk aesthetic but detracts from visibility, so it is disabled by default but available to players who wish to use it (NFR1, NFR3).

Aesthetic Rationale

Visual

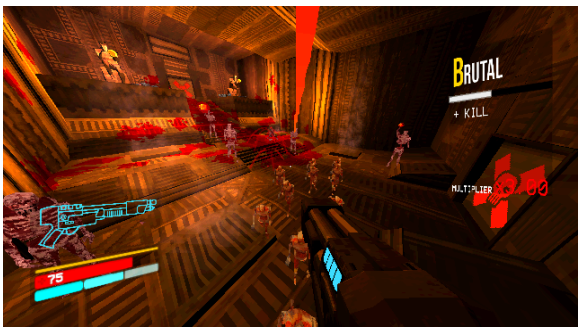

Fig 21. *Inscryption* shading effects; Fig 22. *ULTRAKILL Prelude*.

Inscryption's high-contrast visuals were a heavy inspiration for *Conductor*; its posterisation and low lighting act to hide unnecessary details from the player. I implemented this to accentuate enemies and interactables in the environment (NFR1). I desaturated the environment to bring more attention to colourful interactables (NFR1). Electricity is represented by sharp red particles, while liquid is represented by smooth, transparent purple (NFR1). I chose red as a colour commonly used in the horror/hell aesthetic of rhythm shooters like *BPM: Bullets Per Minute* and *Metal: Hellsinger*, while purple represents corruption or evil in games, a prevalent theme in the game's previous narrative (NFR3). This colour scheme is adjacent to *Beat Saber*'s classic red-blue and the traditional blue-pink of cyberpunk, creating associations with similar media while maintaining a unique colour palette (NFR3).

Early shooters like *DOOM* and the games they inspired, like *ULTRAKILL* and *Dusk*, provided a great reference point for *Conductor*'s low-fidelity models; these needed to be cheap to run on standalone Quest (FR1). The game's Windows-exclusive beat-synced post-processing effects like bloom and chromatic aberration were dropped for performance (FR1).

As the centre of the player's attention, enemies epitomise the game's horror-cyberpunk theme. Melee enemies like the Squid were inspired by sci-fi media such as *War of the Worlds* and *Titanfall 2* while ranged enemies like the Butcher drew inspiration from the religious, gothic aesthetic of *Dark Souls*, *Blasphemous*, and *Cult of the Lamb* (NFR3). The horror theme is cohesive with *Conductor*'s industrial electronic soundtrack and dark environments while keeping in line with the hellish environments of rhythm shooters like *Metal: Hellsinger* and early shooters like *DOOM* (NFR1, NFR3).

Audio

The industrial electronic soundtrack was inspired by industrial, hardcore, and punk acts such as Death Grips, Machine Girl, and The Garden (NFR3). While creating music limited to a four-on-the-floor rhythm was creatively limiting, it was necessary to ensure the player could handle rhythmic gameplay and aiming/positioning in 3D space (FR2, NFR1, NFR2). I heavily built on the audio direction via *Lazarus*, an industrial/drum and bass album I produced alongside development (available in Appendix D). This was especially useful for compiling production principles and synth patches created to match the aesthetic (NFR3). Having a collection of pre-made songs and concepts to work from sped up soundtrack production and ensured I would avoid aesthetic conflicts later in development.

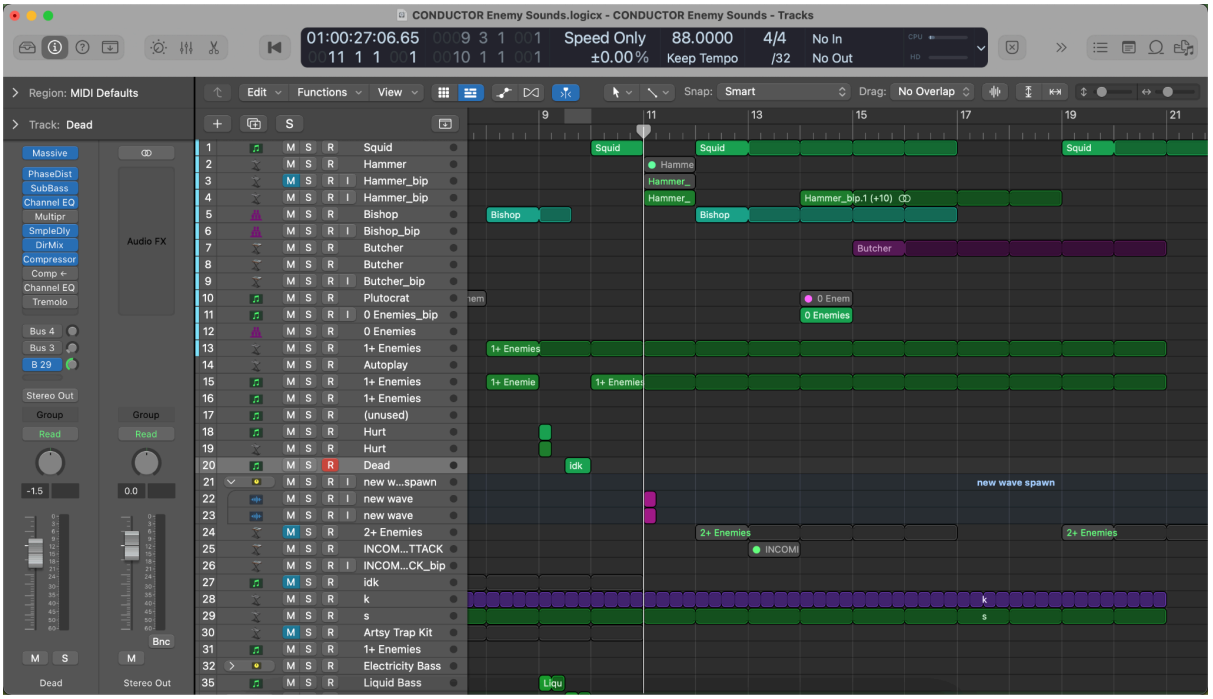

Fig 23. Persistent soundtrack session.

Each individual track was mixed in the same Logic session to ensure overlapping in-game conditions would not lead to a messy mix. For example, many subtle or

lower-importance tracks are sidechained to the kick, meaning they dip in volume on each beat. This gives more weight to the beat and helps the player stay on rhythm (FR3, NFR1). The incoming enemy attack sound is a shrill, shocking lead designed to grab the player's attention. As arguably one of the most important sounds in the mix for gameplay purposes, this track was mixed louder and brighter than other high-pitched leads to ensure it cut through the mix and indicated danger to the player (NFR1).

Less noticeable tracks, such as pads that play as more enemies spawn, are mixed to the edges of the stereo mix to indicate a fuller space without stealing the spotlight from each enemy's trademark track (NFR1). Each enemy uses a unique instrument for its track that builds on its identity and informs the player what enemies are in play (FR3). The Squid plays a fast-paced percussive clanging beat, drawing on sounds in thriller movies and aiming for a metallic spider sound. The Bishop and Butcher use synthetic choirs to draw on their cyberpunk/gothic appearances, with the Butcher's track being more percussive to indicate higher danger (FR3).

Fig 24. Electrocution 808 bass in *Massive*.

While a bass plays when one or more enemies are on the board, there are no sub frequencies until the player controls electricity or liquid or creates an electrocution. This leaves room in the mix for dramatic moments where an 808 bass finally hits as the player takes control of the field and defeats enemies (FR3). Different 808s play based on the controlled interactable: electricity plays a buzzing distorted bass mixed with amp noise, while liquid uses phasers and lowpass filters to create a watery sound (FR3). The 808 for electrocutions is built from a mix of these for a fast-paced, buzzing, warbling sound. The electrocution 808 is also mixed louder than the others to emphasise defeating an enemy (FR3, NFR1).

Evaluation

Seven participants of various levels of VR experience played *Conductor* for 10-15 minutes on a Quest 2 headset, progressing through increasingly difficult levels. After playtesting, participants answered a questionnaire about their experience. Their data was stored anonymously. Results are shown in Table 2.

Topic	**Result**
Enjoyability	82.8% avg
Difficulty	74% avg
Prior experience with VR	48.4% avg (mixed results)
Prior experience with VR rhythm games	54.2% avg (mixed results)
Prior experience with rhythm shooters	25.6% avg
Intuitiveness of rhythmic play	62.8% avg (mixed results)
Helpfulness of gameplay beat pauses	60% avg (mixed results)
Combination of rhythmic actions and environmental combat	42.9% majority said the combination was not particularly good or bad
Engagement with environment	85.6% avg
Focused on gameplay	74.2% avg
Enjoyed moving/positioning in 3D space	57.1%
Enjoyed tracking enemy positions	71.4%
Would prefer enemies to stay in view	42.9%
Could not keep track of environment	0%

Table 2. Summary of participant data. “Mixed results” indicates topics with a wide range of responses.

Despite having a wide range of experience with VR, participants enjoyed the game. They also found it difficult, though this may be tied to the mixed number of participants with prior VR rhythm game or rhythm shooter experience (NFR2). Individual playtests lasted 10-15 minutes, so it is also possible that players did not have time to properly acclimate to VR and the game’s controls. Rhythm elements received a mixed reception, with some players enjoying playing on beat while others found it too overwhelming in combination with 3D combat (FR3). Most players did not explicitly enjoy or dislike the combination of rhythm and environmental combat, though there was a positive response to the game in general. Environmental combat, positioning, and tracking enemies were all well received (FR2). Almost half of playtesters would have preferred if enemies remained in their immediate view,

though most players enjoyed tracking their positions in the environment. The gameplay kept most players' focus, though some players were distracted by environmental obstacles and learning to use VR and the game's controls (NFR1). In optional participant form comments, the game's aesthetic and soundtrack were very well received (NFR3). Playtesting also exposed bugs and gameplay issues, which I fixed.

Reflection

The project met its aims. I successfully developed a VR game for standalone Quest (FR1) that used rhythm game elements to augment (FR3) rhythm-shooter-inspired environmental combat (FR2). The gameplay fully captures players' attention through its colour contrast, dramatic effects, visual/audio/haptic feedback, and genre-consistent theming (NFR1, NFR3). While difficult, players enjoyed the game, proving the viability of highway-free VR rhythm games (NFR2).

There are still some bugs to be fixed with enemy collision; enemies can get stuck in walls or briefly spawn inside colliders before correcting their positions. There is some audio latency on Quest 2, which I could not fix. I could also improve the audio mix for clarity, especially between enemy tracks. Inner City features chopped-and-screwed versions of enemys' original motifs, resulting in overlapping rhythms that act against straightforward rhythmic gameplay. As playtesters were not widely impressed with the game's rhythm elements, I could implement a reward system for performing actions on beat: a score or other bonus for rhythm actions or a penalty for off-beat actions would motivate players to fully engage with the rhythm (FR3). I could also improve player attention with more visual/audio/haptic effects and enemy sound effects spatialised or rotated to indicate their source in the world to the player (NFR1).

Outside of bug and audio fixes, the next step for the game would be longer, more structured levels with new enemy types. While I opted for a wave-based system during development, I would like to open up more options for environmental engagement through linear levels, one-time-use interactables like destructible aquariums, and events like boss fights (FR2). For experienced players, boss fights with characterised rhythms other than four-on-the-floor would create a unique challenge. I had initially planned to include cutscenes and an overarching narrative to the game; while external to the actual gameplay, this would build immersion and give the player more purpose when progressing through levels.

Overall, I am very proud of *Conductor*. Though somewhat lacking in content, the backend systems and gameplay loop are robust, and the game could very easily be expanded with new enemies, levels, and soundtracks. All systems and workflows are in place for a straightforward continuation of development. Despite sacrificing visual effects and complex rhythms during development, I am very pleased with the game's aesthetic and soundtrack, especially in the case of enemy design and element 808s. I have set a proper difficulty level for the game, requiring attention and engagement from the player while allowing for advanced, expressive combat through the interaction system. I have successfully proved the viability of note highway-free VR rhythm games and hope to see more games follow suit as VR media develops in the coming years.

Bibliography

2K. (2007). *Bioshock.* [Video game].

2K. (2019). *Borderlands 3.* [Video game].

Adams, R. N. (2021). *Beat Saber PS4 Multiplayer is Finally Here* [Image]. Available at: https://techraptor.net/gaming/news/beat-saber-ps4-multiplayer-is-finally-here [Accessed 27 May 2024].

Akupara Games. (2017). *Rain World.* [Video game].

Angry Blackmen. (2024). *The Legend of ABM*. [Musical album].

Awe Interactive. (2020). *BPM: Bullets Per Minute.* [Video game].

Awe Interactive. (2022). *Syncing Music and Gameplay in BPM: Bullets Per Minute*. Audiokinetic. [Online]. Available at: https://blog.audiokinetic.com/en/syncing-music-and-gameplay-in-bpm/ [Accessed 17 June 2023].

Bandai Namco Entertainment. (2011). *Dark Souls.* [Video game].

Beat Games. (2019). *Beat Saber.* [Video game].

Brace Yourself Games. (2015). *Crypt of the NecroDancer.* [Video game].

Capcom. (2018). *Monster Hunter: World.* [Video game].

CD Projekt Red. (2020). *Cyberpunk 2077.* [Video game].

Cloudhead Games. (2019). *Pistol Whip.* [Video game].

Collins, K. (2009). *An Introduction to Procedural Music in Video Games*. Contemporary Music Review. 28(1), 5-15 [Online]. Available at: https://doi.org/10.1080/07494460802663983 [Accessed 31 May 2023].

Costello, B. M. (2020). *Moving in rhythms: what dancers and drummers can teach us about designing digital interactions*. Digital Creativity. 31(3), 181-191 [Online]. Available at: https://doi.org/10.1080/14626268.2020.1782944 [Accessed 31 May 2023].

Cuddle Monster Games. (n.d.). *Ocean Mirror.* [Video game]. Available at: https://store.steampowered.com/app/2725640/Ocean_Mirror/ [Accessed 22 May 2024].

Danny Brown. (2016). *Atrocity Exhibition*. [Musical album].

Death Grips. (2016). *Bottomless Pit*. [Musical album].

Death Grips. (2012). *The Money Store*. [Musical album].

Death Grips. (2018). *Year of the Snitch*. [Musical album].

Devolver Digital. (2022). *Cult of the Lamb.* [Video game].

Devolver Digital. (2021). *Inscryption.* [Video game].

Devolver Digital. (2019). *Katana ZERO.* [Video game].

Digital Extremes. (2013). *Warframe.* [Video game].

Dongas, R. and Grace, K. (2023). *Designing to Leverage Presence in VR Rhythm Games*. Multimodal Technologies and Interaction, [Online] 7(2), 18. Available at: https://doi.org/10.3390/mti7020018 [Accessed 15 November 2023].

Drool. (2016). *Thumper.* [Video game].

Electronic Arts. (2016). *Titanfall 2.* [Video game].

Elliot George Mann. (2021). *Saliva*. [Musical album].

Funcom. (2022). *Metal: Hellsinger.* [Video game].

Giant Bomb. (2023). *Note Highway*. Giant Bomb. [Online]. Available at: https://www.giantbomb.com/note-highway/3015-7102/ [Accessed 10 November 2023].

Hakita. (2020). *ULTRAKILL Prelude*. itch.io. [Image]. Available at: https://hakita.itch.io/ultrakill-prelude [Accessed 27 May 2024].

Harmonix Music Systems, Inc. (2019). *AUDICA.* [Video game].

Heart Machine. (2016). *Hyper Light Drifter.* [Video game].

Henley, S. (2021). *Gun Jam's Developer On The Rise Of The Rhythm Shooter. Interview with D. Da Rocha*. TheGamer. [Online]. Available at: https://www.thegamer.com/gun-jam-rhythm-shooter-interview/ [Accessed 14 November 2023].

id Software. (1993). *DOOM.* [Video game].

id Software. (1996). *Quake.* [Video game].

Joy Way. (2021). *Against.* [Video game].

JPEGMAFIA. (2016). *Black Ben Carson*. [Musical album].

JPEGMAFIA. (2018). *Veteran*. [Musical album].

Jaw Drop Games. (2023). *Gun Jam.* [Video game].

Kluge Interactive. (2018). *Synth Riders*. [Video game].

Lorente Martínez, P. (2020). *DOWNBEAT : Development of an adaptive combat system based on the use of the downbeats and upbeats of a musical piece*. Jaume I University. [Online]. Available at: http://hdl.handle.net/10234/191327 [Accessed 31 May 2023].

Machine Girl. (2017). *...Because I'm Young Arrogant and Hate Everything You Stand For*. [Musical album].

Machine Girl. (2018). *The Ugly Art*. [Musical album].

Machine Girl. (2020). *U-Void Synthesizer*. [Musical album].

McMillen, E. and Himsl, F. (2011). *The Binding of Isaac.* [Video game].

Mullins, D. (2022). *Independent Games Summit: Sacrifices Were Made: The 'Inscryption' Post-Mortem*. Game Developers Conference 2022. [Presentation]. Available at: https://gdcvault.com/play/1027609/Independent-Games-Summit-Sacrifices-Were [Accessed 10 June 2023].

Native Instruments. (2006). *Massive.* [Synthesiser plugin].

New Blood Interactive. (2018). *Dusk.* [Video game].

New Blood Interactive. (2020). *ULTRAKILL.* [Video game].

Raw Fury. (2023). *GUN JAM*. Steam. [Image]. Available at: https://store.steampowered.com/app/1308360/GUN_JAM/ [Accessed 27 May 2024].

RedOctane. (2005). *Guitar Hero.* [Video game].

Samurai Punk. (2023). *KILLBUG.* [Video game].

Sony Interactive Entertainment. (2017). *Horizon Zero Dawn.* [Video game].

Statzer, J. (2023). *VR Expansion Plugin.* [Unreal Engine plugin].

SUPERHOT Team. (2016). *SUPERHOT VR.* [Video game].

Team17. (2019). *Blasphemous.* [Video game].

The Garden. (2022). *HORSESHIT ON ROUTE 66*. [Musical album].

Vince Staples. (2017). *Big Fish Theory*. [Musical album].

Void Heat. (n.d.). *Void Heat*. [Video game]. Available at: https://www.youtube.com/@VoidHeat [Accessed 22 May 2024].

Wells, H. G. (1898). *War of the Worlds*. William Heinemann.

Wen, A. (2020). *'BPM: Bullets Per Minute' review: an inventive rhythm FPS hybrid that's also a hellish roguelike* [Image]. Available at: https://www.nme.com/reviews/bpm-bullets-per-minute-review-an-inventive-rhythm-fps-hybrid-thats-also-a-hellish-roguelike-2750616 [Accessed 27 May 2024].

Appendices

Appendix A. Unreal Engine Terminology

This list of terms will be useful for readers unfamiliar with Unreal Engine.

Term	Meaning
Actor	Object in the game world. Equivalent to Unity's "GameObject"
Blueprint, "BP" prefix	Predefined object with scripts and/or components. Equivalent to Unity's "Prefab"
Component, "BPC" prefix	Reusable component that attaches to Blueprints. Equivalent to Unity's "Component"
Data Table, "DT" prefix	Spreadsheet of Struct data
Quartz	BPM syncing system
Tick	Function called every frame. Equivalent to Unity's "Update()"
Widget	UI element, often consisting of other Widgets

Appendix B. Asset Sources

These assets and tutorials were used during development.

Description	Author	Available At:	Date Accessed
Posterisation shader tutorial	Leif Peterson	https://youtu.be/hiHOosepHXM	10/06/2023
Pixelation shader tutorial	CG Hideout	https://www.youtube.com/watch?v=n2Fw7rm8KFg	10/06/2023
Unreal Engine Quartz (audio sync/beat system) tutorial	Michael Gary Dean	https://youtu.be/7Qq6frAwkYM	17/06/2023
Electricity chain particles tutorial	Rimaye [Assets and Tutorials - NIAGARA]	https://youtu.be/SajyFZ35OBI?si=6Cg2jwUpPj52FtWV	27/01/2024
Map/struct array sorting (for the hivemind sorting weighted enemy actions)	trutty	https://forums.unrealengine.com/t/sorting-float-array-numerically/307771/7	28/01/2024
Unreal Engine VR optimisation tutorial	GDXR	https://www.youtube.com/watch?v=w3bgTzZuT48	11/03/2024
Outline material tutorial	UnrealCG	https://www.youtube.com/watch?v=BWoVkQjYncY	03/05/2024
“Damages3D” logo/title font **Free for personal use*	Vladimir Nikolic	https://www.dafont.com/damages.font?fpp=50&text=CONDUCTOR	04/05/2024
Function to fix Unreal Engine’s Sweep collision parameter	anonymous_user_5a95298e	https://forums.unrealengine.com/t/sweep-only-works-with-root-component/319976/13	27/05/2024

Appendix C. Design Document and Development Log

Available at:
https://drive.google.com/file/d/1xhBykGSL57EKyWIKFzqkQOzLRMoiyA3l/view?usp=sharing

Appendix D. *Lazarus* Album MP3

Available at:
https://drive.google.com/file/d/1nCNk3qzBIHiA-o_dEv0zuG7laiGZt3Bj/view?usp=sharing

Appendix E. through G. OMITTED.