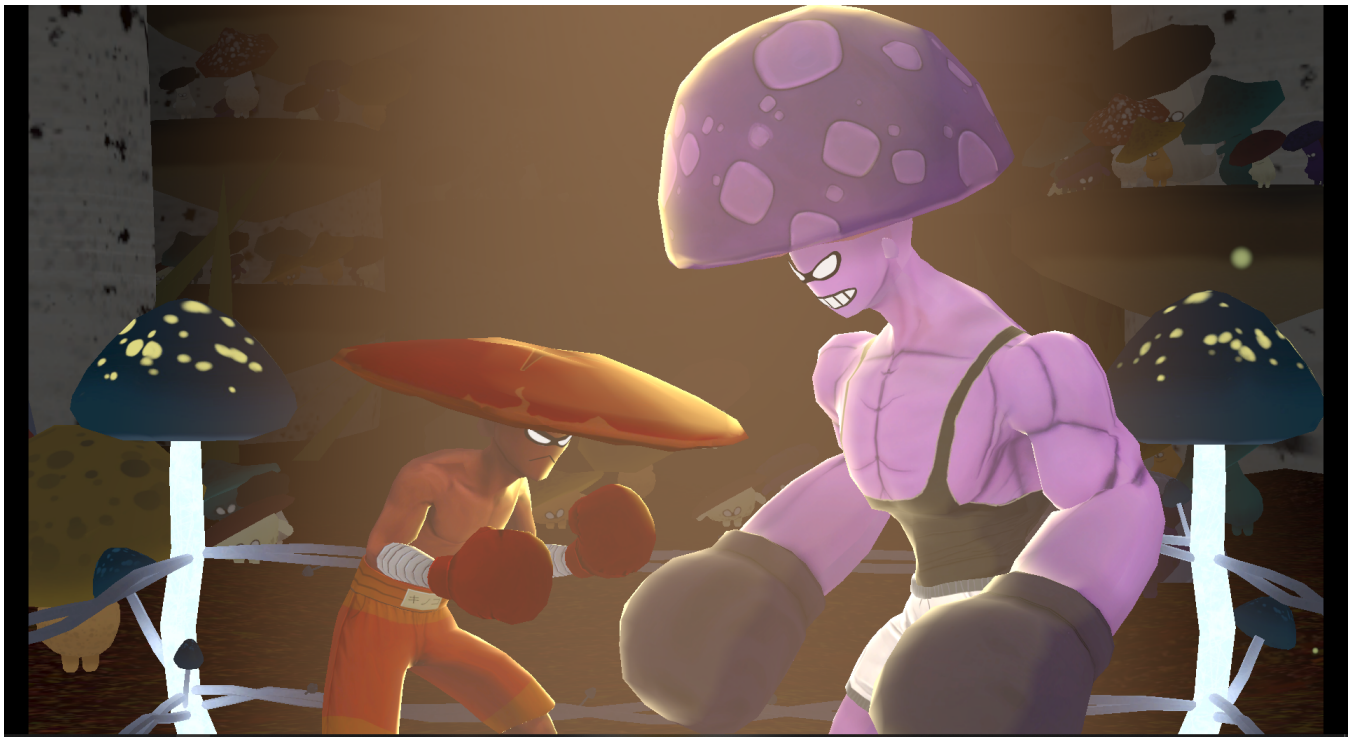

Portobello Beat - Individual Report

Chengyan Zhang, supported by Oleg Fryazinov*
National Centre for Computer Animation

1 Abstract

This specification covers my contribution in the Major Project - ”Portobello Beat”; a game made in Unreal Engine 4; detailing the methods to the challenges I’ve faced throughout this project.

2 Introduction

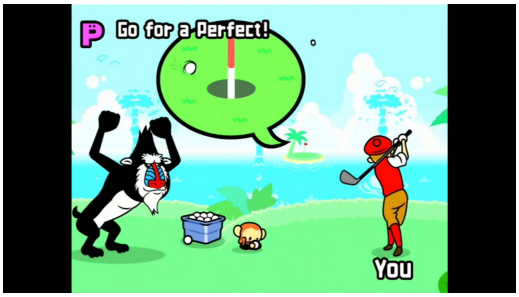

Figure 1: *Gameplay image of Rhythm Heaven, referring to [Wikipedia d].*

This project is a rhythm game at-core, with a theme of fighting (boxing), although, the aesthetics are family-friendly; comparable to franchises such as ”Super Mario Bros” and ”Legend of Zelda”. It has been heavily influenced by a casual rhythm game - ”Rhythm Heaven” which has a distinct feature where the visual input cues in gameplay are mostly signified by animations. Although the game may steer towards a rhythmic interaction, the inspiration from fighting games genres may also be introduced to a degree.

*e-mail:s4903744@bournemouth.ac.uk

3 Roles & Implications

For this game project, my main roles are to develop and maintain the main game logic using the Unreal tool-set; define an asset workflow for the Unreal project and the implementation of animation and audio assets into the game logic.

I am also partially responsible for creating particles effects for the game’s characters and environment under the direction of my project director.

The roles require extended reading of the Unreal Engine’s documentation to determine whether the current engine build (4.20.2) has the appropriate functionalities to achieve the effects of the project.

4 Related Work

4.1 Approaches to the project

Firstly, research on typical approaches to rhythm-based mechanics are required. After reading the posts on [Unreal Forum User - chubbspringle 2018], [User - fizzd] and [James Goldsmith], the articles inform me of the possible synchronisation issues with audio tracks playing at fluctuating frame rates as well as the importance of using ”BPM” for both synchronisation and designer reference. The articles mention the methods of synchronising possible latency between frame-drops, using the previous playback position as a ref-

erence point for playback comparison. This may become essential if Unreal Engine does not have a well-synchronised audio engine for run-time-event-tracking.

The most important game feature to be determined, is the feasibility of facilitating time-sensitive input events; ensuring events are safely executed and locked from degenerative edge cases, for instance, quick repetitions of input when only the first input is accepted with a possibility of a race condition when two somewhat-simultaneous inputs are accepted when the acceptance rule for the event-handler is mutually exclusive.

Although the Unreal Engine documentation for C++-to-Blueprint interaction in [Unreal Engine Support a] hasn't mentioned any potential situations for race conditions between the two entities; testing and observation is required during the implementation stage.

Searching through several posts related to developing rhythm-based games with Unreal Engine, such as [James Goldsmith], it appears that many rhythm based approaches has used a form of spectral analysis to automatically generate positions for input windows in a specific sound track.

However, after a discussion with my director, the approach of our game relies on hand-crafted timings for input windows as he wishes to have a strong control over the flow of the level. Therefore, an alternative method would require an user-interface for the designer to create the input timings of the level's soundtrack.

The sequencer tool in Unreal Engine seems to be the best tool for the problem. According to the documentations - [Unreal Engine Support b], the artist can use the sequencer as a timeline to trigger specific game events. Using this tool will only require the artist to place their desirable frames of player input which can also be synchronised with soundtracks, supposedly, on the same run-time thread (which prevents inconsistent delays on event triggers). This artefact is considered a "beat-map" of a level (please refer to [osu! development team]).

On the other hand, the sequencer tool requires the programmer to create an event system that will trigger the correct game logic whilst keeping the number of event-pin insertions in the sequencer U.I, to a minimum as there will potentially be a high number of inputs required for a game level. The thought of using JSON files to script input properties have been presented during the earlier stages of our development as this may allow the designer to structure various side-effects of the input commands for a more complex back-end calculation for our game states, this was inspired by the general boxing aesthetics, similar to various fighting games which include more interactive mechanics, such as a health bar, damage-taken and a gauge for special attack sequences.

For a rhythm-based game to operate smoothly, most games of the genre avoid implementing various animation sequences and resort to express visual cues as simple shapes to reinforce the rhythm into a player's instinct (for instance, "osu!", referring to [osu! development team]). For games such as "Rhythm Heaven", the use of 2D animation allows the animation to have a stylistic transition whilst keeping up with a fast tempo. Unfortunately, it is very easy to notice discrepancies between animations of a 3D model, especially when a model's actions are not consistently rhythmic; forcing our pose-blending time to be very short.

Following through the documentation, the general approach to Unreal Engine's animation relies on a combination of finite state machines and montages. The state machines allow a continuous animation system where each state can either be a regular animation sequence or a blend of many animation sequences depending on user parameters (with game logic or a model's skeleton joint reference). This may prove useful for animations which affects a model's skeleton indefinitely yet keeping the changes reversible. Montages also allow quicker execution of animations, possibly satisfying the need of an immediate input feedback and completion for a punching or dodging animation; under a high tempo section of an audio track.

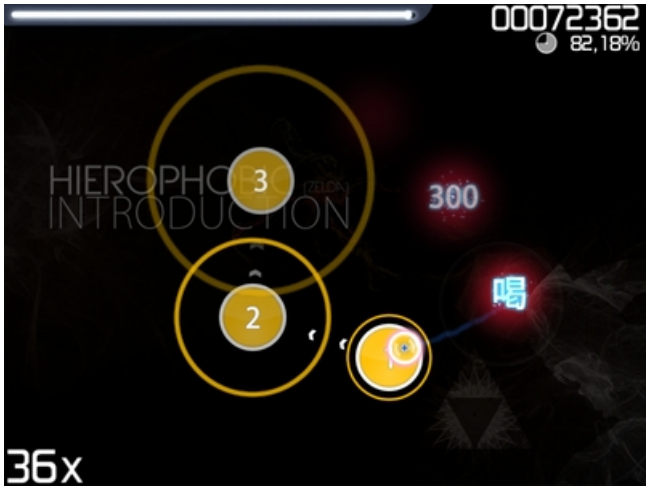

Figure 2: *Gameplay image of osu!. Taken from the [osu! development team].*

Figure 3: *Gameplay image of Blazblue.*

4.2 FX

Special-effect-wise, a game with fighting aesthetics and a retro theme may take the inspiration from classical fighting games such as Street Fighter, Guilty Gear and Tekken (refer to [Wikipedia e],[Wikipedia b],[Wikipedia a] respectively). Most effects from these games are centred around the impact of connected inputs and force-field-esque effects to represent blocked (failed) inputs. The impact generally includes an explosion of sprite-based flares with colours corresponding to a character's motif, whilst the degree of the effect's intensity also describes the power of an attack in forms of ragged beams and an increased radius of impact rings as well as the various versions of the accompanying impact sound. Although our game is targeted towards younger people, these sources help to picture the general composition of an impact effect in-combat.

4.3 Miscellaneous

During a discussion session with the team, it has been confirmed that a facial expression should also be able to transition when cer-

tain events trigger. This idea has encouraged to look into several ideas of material-based animations.

Firstly, we can consider the different expressions as sprite-based cycles, the material will use a panning state to create an animation loop of an expression; to change the expression, we can introduce different offsets to the UV space of the sprite texture to iterate through a different series of sprite sequences. However, we don't expect a detailed face to be seen by the player during most of the game-play. Therefore, the decision is to use a static material as opposed to a animated sequence, reducing the complexity of our character shaders by a fair margin.

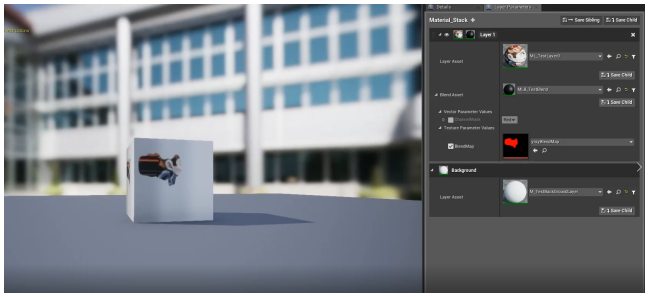

Figure 4: *A test material using the Material Layer feature, referring to [Unreal Support].*

Secondly, we need an approach to define an independent facial texture. I've originally researched into "Material Layers" as one of Unreal Engine's experimental features. The tool is somewhat straightforward to use under high-complexity models where opacity of different materials can be precisely blended using additional masks. For our project, it is not necessary to use this feature until additional expansions in the future are considered. For the time being, we are choosing to use a transparent plane to paint on top of our faces for the sake of simplicity.

5 Implementation

5.1 Initial Prototyping

Throughout the first months of the development, my main goal has been focused on defining a structured workflow using the Sequencer described in Section 4.

Ideas presented from Section 4 all take approaches in a pure-source-code style for the construction of audio cues, input frames and their timings; the reasoning being that the source code allows quick calculations to be performed for event-handling and lower probability to introduce latency and desynchronisation between interactive audio tracks.

For these reasons, my C++ source code will consist of: the numerical records of the game; methods to modify such records; methods to allow communication from C++ to Blueprints and handling of some minor non-character-specific audio effects. The blueprints will then direct the majority of the run-time behaviours of the game; using the given functionalities from the C++ side with event notifications coming from animations and sequencers.

Earlier in Section 4, I have mentioned that a JSON scripting can enable scripted sequences and quick lookup of various data for mechanics in a fighting game. The decision has been made that JSON is not needed due to a more focus on the musical aspects of the game. Additionally, the JSON format has a poor readability as an

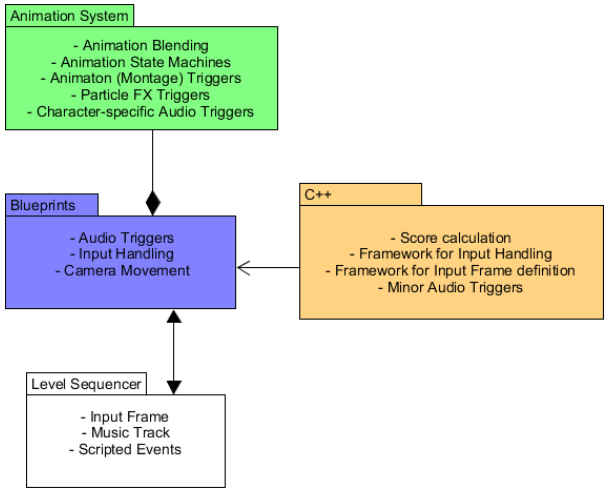

Figure 5: *General workflow structure of our project.*

abstract format for level design, which has made my partner feeling confused during a work session.

It has also been noticed that the sequencer can cause desynchronisation due to sudden frame-rate changes, however, it has only happened during test-play sessions with extremely low-capability hardware (Discussed later in Section 6.), playing from an incomplete part of the audio track, or "alt-tabbing" out of the game context. If the issue persists, precautions can be taken by a work-around, for instance, "automatically pausing the game when the current open context is not the game window".

5.2 Game mechanics & issues encountered

In the initial stages of the development, after a defined workflow, I have encountered a fringe-yet-problematic issue in the prototype; relating to the integrity of the input system as I have predicted in Section 4; where two or more commands can be activated simultaneously when two or more of the corresponding inputs are pressed. This causes the player's in-game model to switch animation states erratically and cause the enemy to also have an unexpected behaviour.

To solve this, I have to consider the possible areas for race conditions between event threads inside Unreal. It has eventually revealed to me that the main cause of the issue comes from the simultaneous call from the input-bound methods in a pure-C++ framework, and the Unreal Engine's Blueprint event system, has a somewhat ambiguous description of thread-safety measures as it hasn't been clearly noted in the documentation.

The issue comes from my choice of handling input events for this project. The canonical workflow for input handling has been done with either exclusively-C++ or exclusive-blueprints, however, mine contains events from both blueprints and C++; this possibly causes the separate events to execute in separate threads where they cannot stop each other from accepting the very-next degenerate input as an event trigger.

The choice of having a blueprint and C++ implementation of events allows numerical calculation to be done on the C++ side (which is much quicker than blueprint) whereas the blueprint is suitable for visualising flow-control for the game as well as a visual control over the animation blueprints.

The solution for this matter is somewhat trivial once I have found the point at which the event threads diverge, I have added an ”input lock” as a mutual exclusion lock for my input events on the blueprint-side as my C++ side tends to accept the input before my blueprint system. Additionally, a timer is also implemented to control the timing of the locks' release. This way, my blueprint will not accept any inputs immediately proceeding the first occurrence for a set amount of time.

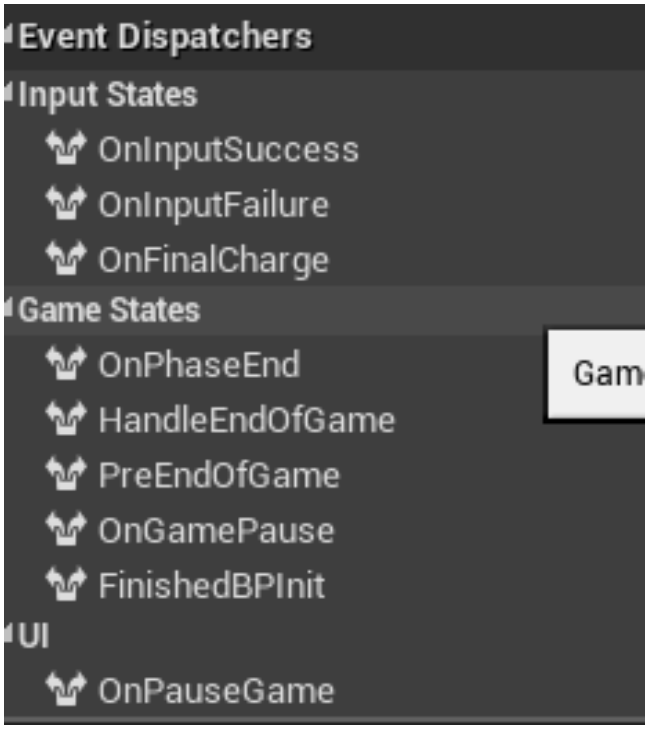

Figure 6: *A list of event-dispatchers used in the main blueprint to emit signals to their subscriber-objects.*

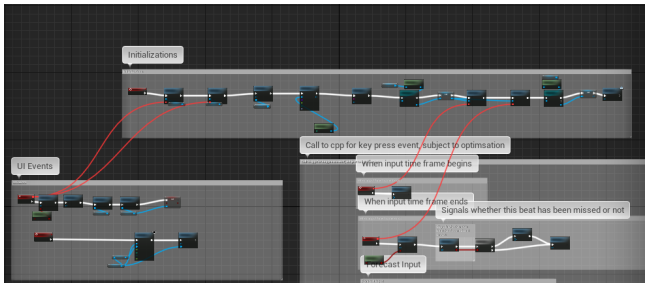

Figure 7: *A rough view of connections between foreign event-dispatchers and handler events in the main blueprint(Red connections to an event define the specific event called whenever the dispatcher calls.).*

The main mechanic of this game revolves around input-handling with a constraint on time-frames. This has boiled down to a complex system of event-handling, for events coming from different mediums; with different implications (whether it is input, a change in the game outcomes or U.I triggers). In this case, various event-dispatchers are created to communication with different assets in the world.

5.3 Sequencer Workflow

The goal of the sequencer is to give an artist the ability to event scripting for ”beat-maps” (refer to [osu! development team]), whilst being able to compare and sync with the music track in-question. The tool have achieve such task quite effectively.

However the problem for me to solve is to ”minimise the number of events needed for every beat requested.” If events are not designed to encompass the multi-layered interaction between input acceptance and the definition of a time-range for said acceptance.

Ultimately, I have reduced the process to around three events required to be placed per-wanted-beat: first comes the forewarning cue, second - the beginning of input acceptance checks and finally, the end of input acceptance. This process are similar between the two different types of input that the players are expected to play with. However, after reports from my designer partner, ”punching” is proactive and ”dodging” is reactive, henceforth, the two inputs events are segregated and categorised as different input acceptance events.

Figure 8: *The final version of the dodge cue workflow, only two events are required to be defined per-dodging-input.*

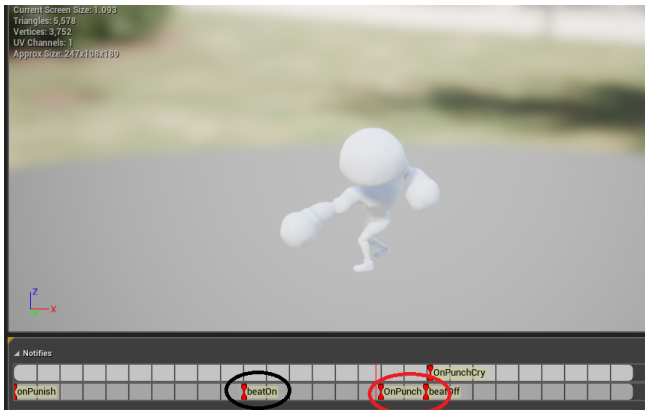

Figure 9: *A definition of input window on an enemy's attack, bound by its animation sequence. Black circle denotes the beginning of input window, red being the end point(s) of input window.*

To further reduce the repetition and improve integrity of this task, I have decided to embed the input acceptance events into animation assets. This implies that the definition of input windows is not prone to human inconsistencies as one definition will replicate the behaviour for all instances of the animation sequence; also simplifies the process to only the placement of the beginning trigger of input acceptance instead of the full definition.

The simplification does come with a minor drawback: as animations take time to play, the sequencer will not explicitly present the frame delay between input acceptances, this will require the designer to calculation an acceptable placement of a beat more carefully than the original, lengthier method.

This method may not be the most elegant approach pragmatically (Automatic spectrum analysis could generate a believable beat-map for a level. Abstract event scripting on JSON files can also be possibly quicker than using a manual design on a sequencer.), but with the insistence of artist-control, the use of a visual framework is a necessity.

5.4 FX

With references from series mentioned in Section 4, the general form of some conventional effects can be observed.

Figure 10: *A punching effect from Street Fighter V, taken from [Street Fighter].*

Figure 11: *A punching effect from BlazBlue.*

The impact from landing a punch consists of:

1. several rings of shock-waves expanding from the centre of the explosion.
2. An intense spark at the middle of the explosion, the shape is expressing the direction of force, very similar to a muzzle fire from gun-fires.
3. Particles emitting from the impact.
4. Minor hits are usually lacking the more dynamic effects listed above, leaving only the spark at the centre.

My approach to this effects consists of a series of fan-shaped emitters accompanied by bursting particles. The overall effect is less intensified compared to the references due to a less focused violence theme. The choice of rings' behaviours imitates the shock-wave effect as well as a spore-like dust cloud.

The effect for blocking an attack is composed of:

1. No over-exaggerated effects applied.
2. Circular shield-like shock-wave effect at-core.
3. The shields usually correspond to the game's (or a character's) theme.

Figure 12: *A punching impact effect from our character.*

Figure 13: *An uppercut impact effect from our character.*

Figure 14: *A blocking effect from Street Fighter V.*

Figure 15: *A blocking effect from BlazBlue.*

The previously developed effect has some similarities to this one, therefore, some of the emitters can be reusable by modifying the style of the the shock-wave. There are also assumed to be some minor bursting particles due to a spore release when fists clash with a character's surface.

I have kept the overall appearance simplistic yet easy to recognise due to a high emissive value; in order to notify the player of a mis-play.

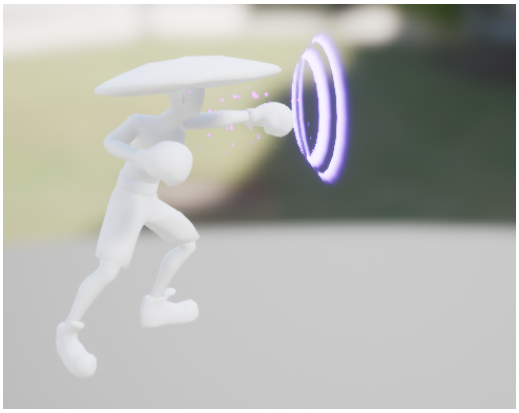

Figure 16: *A blocking effect from a missed hit on our enemy.*

5.5 Animation System

The animation system from Unreal Engine comprises montages and state machines (both managed by animation blueprints) - the former is used for one-off triggers from game logic and the latter defines a series of transitional behaviour and more options for animation blending. The main blueprint would then drive the transitions through montage triggers or by modifying state variables.

To utilise the strengths of the state machines for this projects, the "blend per-bone" and boolean-case blending are used quite often for persistent changes on the character models. For instance, our enemy - "Deathcap"; has an enraged mode with his spikes extruding from the gloves permanently, this can be achieved through using the skeleton to blend between a "normal" set of joint transforms and a "enraged" set at the same joint locations.

5.6 U.I System

Using the experience from the previous project from last year, the process of implementing U.I into the game has been mostly frictionless. For instance: the time spent on variable-tracking, transitions using opacity and audio decay has been greatly reduced.

6 Evaluation

Overall, the project has achieve its main concept and mechanics of being a rhythm-focused game, incorporating a fighting game aesthetic whilst keeping the graphical intensity somewhat friendly for a younger audience.

The general approach to the mechanics has worked well to a degree. The construction of a "beat-map" is still somewhat arduous and very required many duplication of patterns. Approaches of "creating a library of smaller, reusable patterns of beat-maps" has an unwanted latency between the boundary of audio track switching, forcing us to make a singular map at once.

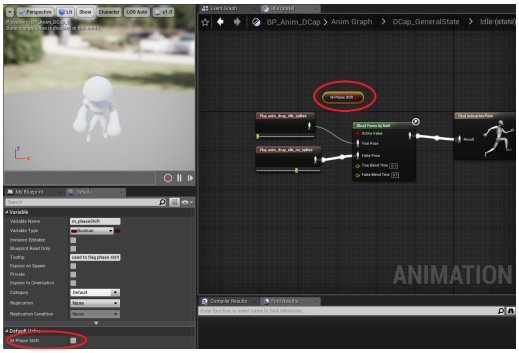

Figure 17: *Example of blending by a boolean, there are no spikes on the gloves when false.*

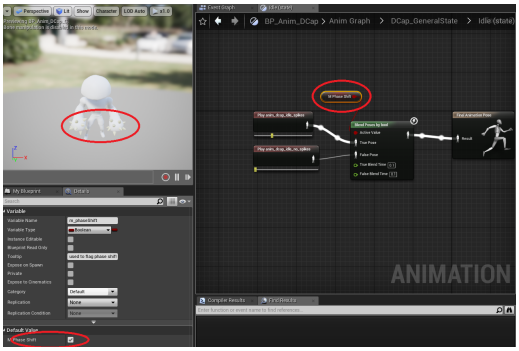

Figure 18: *Example of blending by a boolean, there are spikes on the gloves when true. This value can then be layered across the state machine to define compounding effects.*

Will the input handling issues resolved, the C++ and blueprint interaction has been very successful; although some code structures may prefer less dependency, most of the structures are quite organised.

However, this project has not fully considered a support on low-end machines. This has been noticed as I tried to run the packaged product in a very low-end laptop. The results are not satisfactory as the poor performance resulted in the game being unplayable, albeit the hardware has possibly not had the latest hardware drivers with an approximately eight-year-old laptop GPU from ATI/AMD (Which has not had the most active support from Unreal.).

There are also some effects and features that are marked as stretch goals, and are not developed further due to time constraint.

Figure 19: *Image of faint after-image in "Punch-Out!!", referring to [Wikipedia c].*

Figure 20: *Spore-release images from real-life*

The approach to after-mages has been researched, however, it has not been finalised. The desired effect is not an explicit copy of the current model, we seek a faint, partial "ghosting" effect that may also happen before a specific movement. The original thought is to use a post-processing material to capture motion blurs and discretise the following results. However, further research and trial is not complete at this point.

In addition, fungus-like effects and falling leaves are also some of the relevant aesthetics for this game. It is thought that our characters would excrete spore whenever they are damaged or enraged. We have referenced from real-life to capture the motion of the systems, but we also have to consider a somewhat simplified style to fit with our general theme. The implementations for stated effects has not finished at this point in time.

Mechanically, the project also lacks in additional visual cues for the player. The game may become somewhat confusing without additional aid provided to accurately determine the timing of a specific beat. We have considered implementing a mechanic similar to [osu! development team] (illustrated in 4, Figure 2) where simple, animated shapes determined a expected-time to press a specific command.

7 Conclusion

In conclusion, this project has been an overall success in producing a game with its mechanics intact with its themes. However, there are several unfinished considerations that may improve the quality of game-play and graphics.

Programming and problem-solving aside, the FX role has been a learning experience for me, as it requires me to critically identify useful and unnecessary elements within a range of diverse references and previous works whilst justifying our own stylistic decisions.

The challenges from a programmer's point-of-view has been mostly correlated to my understanding of Unreal Engine's defined behaviours; the methods of which prevents undefined behaviours and utilising the advantages of specific frameworks; as well as providing and improving upon a comprehensive set of tools for my designer to accomplish tasks efficiently.

References

JAMES GOLDSMITH. Making music/rhythm games in ue4. https://skillsmatter.com/skillscasts/12093-making-music-rhythm-games-in-ue4.

OSU! DEVELOPMENT TEAM. osu! introduction. https://osu.ppy.sh/help/wiki/FAQ#osu!-introduction.

STREET FIGHTER. Street fighter v: Arcade edition - g gameplay trailer. https://www.youtube.com/watch?v=6pYVvWuV0Bc.

UNREAL ENGINE SUPPORT. C++ and blueprints. https://docs.unrealengine.com/en-us/Gameplay/ClassCreation/CodeAndBlueprints.

UNREAL ENGINE SUPPORT. Sequencer editor. https://docs.unrealengine.com/en-us/Engine/Sequencer.

UNREAL FORUM USER - CHUBBSPRINGLE, 2018. Rhythm-based combat help. https://forums.unrealengine.com/development-discussion/blueprint-visual-scripting/1491340-rhythm-based-combat-help.

UNREAL SUPPORT. Material layers. https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/MaterialLayers/index.html.

USER - FIZZD. Here's a quick and dirty guide i just wrote: How to make a rhythm game. https://www.reddit.com/r/gamedev/comments/2fxvk4/heres_a_quick_and_dirty_guide_i_just_wrote_how_to/.

WIKIPEDIA. Blazblue. https://en.wikipedia.org/wiki/BlazBlue.

WIKIPEDIA. Guilty gear. https://en.wikipedia.org/wiki/Guilty_Gear.

WIKIPEDIA. Punch-out!! https://en.wikipedia.org/wiki/Punch-Out!!.

WIKIPEDIA. Rhythm heaven. https://en.wikipedia.org/wiki/Rhythm_Heaven.

WIKIPEDIA. Street fighter. https://en.wikipedia.org/wiki/Street_Fighter.