touhou megaman fangame in unreal engine 5

University of Staffordshire

By Oskar Zywer

Final Year – Games Development Project

contents

introduction 4

Introduction 4

aims, objectives, and deliverables 5

Aims, Objectives, and Deliverables 5

research methodologies 6

Research Methodologies 6

documentation of production 7-21

Asset Gathering | Tracking and Documenting Project Progress 7
Unreal Engine Project Planning and Organization 8
Conducting Research into Mighty No. 9 9-11
Shovel Knight GDC Talk 12
The Renderware Engine 13
Unreal Plugins 14
Downside of Modern Solutions 15-17
Change of Plans 18
Blueprint Showcase 19-21

final artefact 21-29

Character Movement 21
Level Diagrams 22-27
Bossfight 28
Menu UI Showcase 29

contents

evaluative review.....30

Evaluative Review.....30

bibliography and references.....31-33

References.....31

Bibliography.....32-33

special thanks.....34

Research Methodologies.....34

introduction

Project Problem: Breaking away from modern video game trends through creating a 2D, Megaman X inspired Sidescroller faithful to mid 2000s aesthetics and trends in entertainment

Indie games are thriving, Triple A is collapsing, is it the tools and trends used today behind the products creation that are responsible, or simply the rise in gaming development costs?

We could debate this forever, but one thing that's obvious is that:
Games are an art form, and they are best when expression in its creation is unlimited.

In recent times a lot of titles, especially in the indie space, have been bringing back aesthetics and trends from 2000s media that were neglected overtime in favour of trends that proved to produce large profits. These titles have seen immense success in doing so.

Games were cheaper to make back in the 2000s, it was a time of discovery and experimental trends and aesthetics, but were they better to what we have now?

Comparatively, were they created more, or less efficiently in regard to development practices we have today?

The goal of this project is to prove that something old, can be made again using something new

From beginning to end, the Project's Development Cycle lasted 16 weeks

aims, objectives, and deliverables

aims

Taking a modern game engine, Unreal Engine 5, which majority of video games today are developed with, is completely free to use, has plenty of documentation available and efficient workflows established online, and seeing how it hold ups with creating a game that is meant to capture the feel of early 2000s titles. The entire games art direction will be centred around the Touhou Project franchise, it is a bullet hell game series from Japanese Doujin creator ZUN (Team Shanghai Alice) which has been around since the late 90s and has since become a staple of Japanese media as the series itself is nearly entirely fair use, with ZUN encouraging derivative works of the series [1], it is a great contributor to indie games as a whole around the world with many famous developers (like Toby Fox) starting out inspired by the Touhou Project. Due to the series falling under fair use, there are thousands of free assets online that I can use to ensure consistent art direction in my game. I will be taking the Megaman design formula, a formula that fell into obscurity in the late 2000s, with titles made around it today being hit or miss and trying to capture it properly while attempting to innovate with knowledge available today that was not around when the formula was at its peak in popularity. [1]

objectives

I will be doing a deep dive into Unreal Engine's blueprint techniques in order to find the most efficient way of developing this title. A lot of research will be conducted into fields which are relevant towards the Project Problem, those being relevant articles around games made around the Megaman formula, watching conference talks around games which are similar or relevant to the title I want to create, consuming various video avenues around the 2000s aesthetic in gaming and Unreal Engine tutorials where necessary, and finally, actually playing some games which have tried to capture the Megaman formula and the reception around them. As my game will be using the Touhou Project franchise for it's theming and narrative setting, a lot of asset gathering will be conducted online too.

deliverables

- 2D Touhou Project Sidescroller Fangame developed in Unreal Engine 5.x using the PaperZD plugin, compiled, and to be playable on PC platforms
- Consistent Artistic direction through Sprites and Music (Free Assets, Commissioned, Self-made)
- Gameplay loop to feel fun and challenging to play
- UI and UX to be easily navigable and look pleasing
- Several QOL options to be available (Game Settings, controller support etc.)

research methodologies

research avenues

I have selected very specific avenues of research that I believe will assist me during this project, as my project problem is very broad, allowing me to consume several different avenues that all link to the problem at large:

Articles about the release of Azure Striker Gunvolt and Mighty No. 9

GDC Talks, one about bringing a bullet hell game titled 'Mothergunship' to VR, and a Q&A with a lead dev from Yacht Club Games, David D'Angelo, the studio behind Shovel Knight

A Youtube video from channel 'Retrohistories' about the Renderware game engine

Several websites for asset and knowledge research:

- Itch.io (Assets)
- Moriya Shrine (Fansite for all things Touhou)
- Opengameart.org (Assets)
- X (Twitter) (Communities for research, both asset and Unreal tutorials)
- Reddit (Communities for research, both asset and Unreal tutorials)
- Youtube (Unreal tutorials)

And finally, playing Mighty No. 9 and Megamari, two games utilizing the Megaman formula

methodology

Mighty No. 9 is a game that famously flopped on release, it was meant to be the revival of the Megaman formula in a modern Unreal Engine developed title, by reading articles and playing it, I intend to find what went wrong, to not repeat the same mistakes with my title, that is also using the Megaman formula and developed in Unreal Engine. Megamari is a Touhou themed Megaman fangame, and Azure Striker Gunvolt is a game using the Megaman formula, all of which I am researching in some way to see what they did well, and what they did wrong.

GDC Talks about; Mothergunship is a rougelike, bullet hell game which had a sequel made for VR in Unreal Engine, it's development cycle was very interesting with the team having to work around the engine's performance issues in creative ways, at the same time I choose to watch a Q&A from Yacht Club Games developer David D'Angel, the studio behind Shovel Knight, a 2D platformer series from an indie team that nailed the action platformer genre

As I am focusing on 2000s aesthetics for my art direction, I need to make sure I have every avenue I possibly can where I can find art assets to ensure I nail the art for my game.

I also choose to watch a video about the Renderware game engine as majority of games developed in the 2000s used the engine, and in turn it defined a lot of the aesthetical trends in games in that era as a result.

My research avenues are very varied, and I have done this on purpose because by doing so I consume a bit of every type of media, actively engaging in existing products that are all relevant in some way to my project problem, this results in me being able to create backup plans and contingencies around my project, because the variation in research types here lets me see the best and worst behind the different solutions that already exist around my project problem. Therefore, the variety in the research avenues I have here let me evaluate the best approaches to solve the different problems that I am likely to encounter throughout the project, because I will get a glimpse into how these different products all tackled similar problems, and choose solutions and create contingencies around this knowledge, letting me get a clear idea of where my priorities should lie and where I can adapt my plans if things take an unexpected turn.

documentation of production

asset gathering

The first thing that I did when starting the project, was searching for assets, did Touhou themed Megaman-like sprites even exist? I myself am not an artist, so I had to make sure that I had every asset I needed before I even started planning the game loop itself.

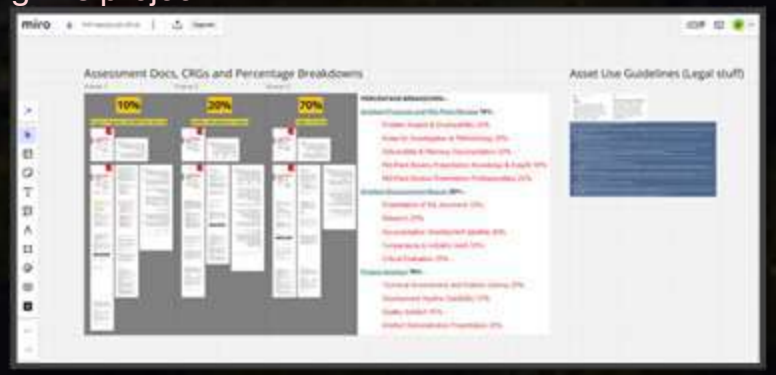
The Touhou Project is massive, and luckily, I did not have any issues finding Touhou assets that worked for this project, and I personally also have a massive library of assets that I have found over the years.

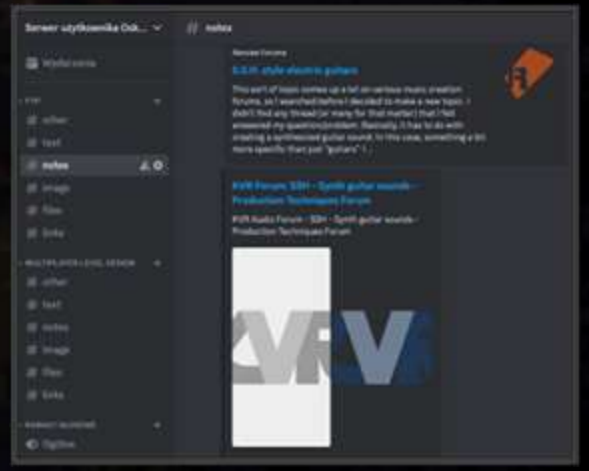
My primary source of asset gathering was through itch.io, having created a collection to ensure that I kept it all in one place, ready for me to find at anytime.

tracking and documenting project progress

Part of modern development workflows, is using programs that make planning and organization efficient, during my project I utilized the following:

- Miro board for quick access and easier viewability of Module related documents
- Trello Kanban Board for easily keeping track of Tasks
- Personal Discord Server that acts as my file/link database during this project
- Gantt chart tracking my progress throughout the module

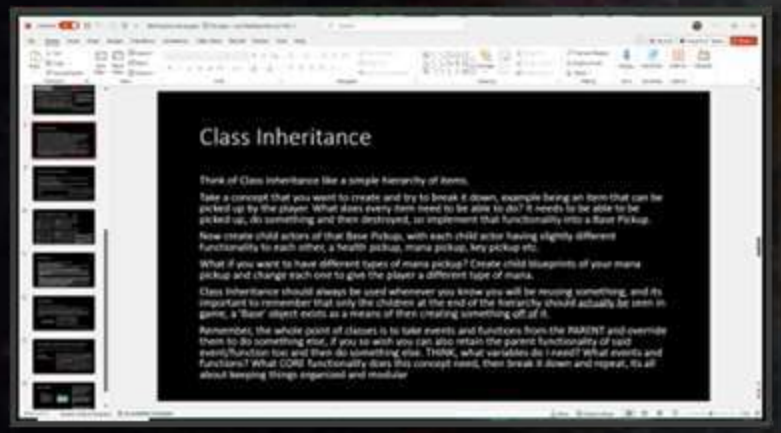


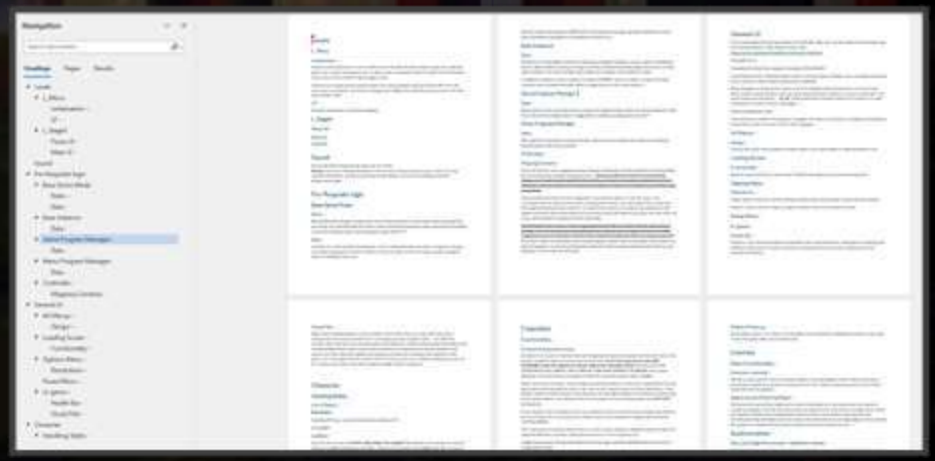


documentation of production

unreal engine project planning and experimentation

Before starting development of the Unreal project, I ensured that I did a full recap of the engines systems and made sure that I had knowledge fresh in my head, ready to develop, I did this through experimentation in engine and notetaking as I was doing so, this is because I had to make sure that I knew exactly what techniques to use when I would get round to creating specific things.





Once I had felt confident in my abilities, I started working on a document that would act as my full project breakdown, where I would divide my game into individual concepts, and then under each concept I would break it down into individual sections and what would need to be done to create what in regard to the blueprint code creation. This proved to be really efficient down the line, as by doing this I ensured that my game would end up relatively bug free due to already having planned out systems ahead of creating them, and whenever I would run into a bug, I would usually be able to track it down fairly easily due to me having a written explanation behind anything in front of my face whenever I needed it .

documentation of production

conducting research into mighty no 9

The Megaman gameplay formula is one that is famous for its difficulty, and one that over the years had not seen much innovation done around it nor games created with it, therefore when a new game is developed around the Megaman formula today, it is easy to look at it and think that because the formula has not been used often in recent years with attempts at it being mediocre, that the new game will likely not nail it well either, but is that due to people not knowing what makes a good application of the formula, or is it the finished product that is the problem?

An article published before the release of Megaman 11, which is the newest official Megaman title from Capcom, states that the series is extremely difficult by modern day standards, it's design philosophies were solidified in the 80s and have not evolved to accommodate the rise in gaming popularity in casual players, the formula had stagnated, and new titles in the series did not appeal to new players [2], this is something that Mighty No 9 tried to fix, and failed horribly trying to do so.

A month after Mighty No 9 released, an article released reviewing the game, it mentions that the gameplay is stale, the game introduces new gimmicks that are made with the intention of appealing to new players, but the gimmicks use case is not justified well in the level designs, and the mechanics that the game wants you to use in specific sections, are sometimes less efficient than others that the game isn't intending for you to use at a given moment. Mighty No 9 tried to capture the Megaman formula well, but through forced gimmicks and a bunch of other issues, it has failed to do so completely. [3]

documentation of production

conducting research into mighty no 9

I personally played the game and found too many issues with the design philosophies around it to write them all up here, but while there were a lot of issues, I also found myself having fun, it was not all doom and gloom, and that is because the sections in levels which did justify use of the new gimmicks made the gimmick fun to use.

Therefore, I thought about how I could potentially incorporate a gimmick into my own game, taking the Megaman formula and trying to do something new with it. The sprite sheet I used for the player character, features an animated broomstick projectile, the idea I had was that the player could hold a button while on the ground which will slowly pull them backwards, the longer they held the button, the further they will then throw a powerful projectile forward dealing big damage, think of it like a slingshot being charged. This mechanic would be justified through precise enemy placement which would encourage use of the mechanic over traditional movement, something which Mighty No 9 failed to do with its gimmicks.

Writing up my thoughts while playing Mighty No 9

Player holds button

Keeps holding...

Button released

documentation of production

conducting research into mighty no 9

In regards to all of the research that I conducted through this project, I found that the genre of platformer that is Megaman, is one that has a fanbase full of purists around it, people who do not particularly like it when a perfectly fine formula is taken and then changed heavily, so while I wanted to innovate on this formula, I also considered the outcomes that would likely occur if I tried to innovate on it too much, through the conclusions that I came to when consuming all of my research avenues, which I will talk about later.

Mighty No 9 tried to appeal to a new audience through gimmick creation, the game was extremely easy in some parts while really hard in others as a result, this just is not acceptable when you are making a game around a formula famously known for its difficulty while trying to reach a new audience, as neither audience can then be fully satisfied, casual players would encounter crazy difficulty spikes, while hardcore fans of the genre couldn't ignore the exploitation of mechanics to traverse things in unintended ways making it easier to play, or underutilization of the gimmicks in areas where its meant to be effective, there is one very good example of a boss fight which wants you to use a specific character ability to fight it, but the regular attack ends up being more efficient.

Mighty No 9 proved to me that the core aspects of a games design need to be nailed before trying something new in a formula that has been solidified for over 30 years, sometimes it's better to hold back than to try forcibly innovate on top of something that already works... But the problem is that it does not work! And that is because the formula has stagnated over so many years, very few games release today using it, and the core reason why is because it has failed to adapt to the rise in casual players.

Mighty No 9 gives players Power Up items before a boss room after the players first failed attempt at the fight

Depending on the boss fight, these Power Ups are actually less efficient than just using the regular shooting mechanic!!

documentation of production

shovel knight gdc talk

In the GDC talk from David D'Angelo at Yacht Club Games, talking about Shovel Knight, he mentioned that their approach to creation of the Shovel Knight games has not changed based on the influence that Shovel Knight had left on the 2D action platforming genre as a whole, the games do of course feature new mechanics and gameplay elements with every new entry, but because the genre is targeted at a niche audience already, they do not have to change their games too drastically between each release, as they know what their fans like. [4]

With this in mind, I thought about my initial goal of fixing the Megaman formula and doing it right and realized that my goals had to change slightly from forcibly innovating on it, to making the core extremely high quality, using a mix of modern and old game design techniques, and keeping the game loop faithful to old Megaman standards while innovating only where necessary. The Shovel Knight franchise of games are extremely polished, with a difficulty curve that scales upwards as the game progresses, meaning that casual players can play it with the opportunity of getting better through natural progression, and hardcore players can play it knowing that it gets more difficult as they play.

It was clear that the barrier to entry to games under the Megaman formula is too high, the formula requires innovation in the form of accommodating casual players, which games released under the formula in recent times had not done well. After conducting a lot of research, I came to the conclusion that I would be scrapping the idea of trying to implement any gimmicks and would instead remain faithful to the genre conventions during the early 2000s but would add multiple difficulty options under the stage selection.

The Touhou Project is infamous for its difficulty, the official bullet hell games offer four different difficulty options, ranging from easy to lunatic. With my game, I would implement a Normal and Lunatic difficulty option before starting a level, the core gameplay between the two will be identical, with the differences lying in things such as amount of invincibility frames after taking damage, enemy and player health amounts, and slight level design changes.

The difficulties themselves, in order to cater to the right audience, will have descriptions shown on screen telling players the differences between the two, ensuring through such a quality-of-life feature that players are given options that accommodate new players, while keeping the difficult nature of the gameplay formula intact.

Difficulty options in Touhou 6: Embodiment of Scarlet Devil

documentation of production

the renderware engine

Before Unreal and Unity, Renderware was the 3D engine that powered at least a quarter of all games released around the early 2000s.

Started out as just a 3D rendering software by none other than Canon (yes the camera company!!) It had tons of development done on it over the years and started out in the PC and Dreamcast markets, it didn't reach its breakthrough until the PS2, and by this point they had optimized the engine's libraries to work really well on basically all platforms at the time

Renderware was huge, but it was then bought out by EA and effectively killed off, the reason is not entirely clear, but at it's core it's because the devs behind Renderware did not have enough time to get the new version out which would power most 360 and PS3 titles, and Unreal Engine quickly took Renderwares place as the most widespread game engine, as it was showing promising results on new hardware in graphics and performance.

Renderware defined the era in terms of visuals, because of the engines quirks, games looked the way they did back then, and the shift to Unreal as the new engine with the highest market share and a new more powerful console generation, caused titles to release with a bigger focus on graphics fidelity, which was not bad at the time, but would have long lasting effects into todays game industry by the now evident extreme divide in costs and development processes of AAA vs Indie titles. [5]

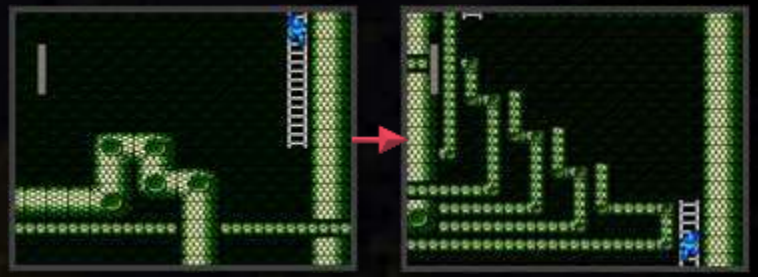
I am making a game that would've likely been made with at the time an engine similar to Renderware, or with it. Unreal Engine is an engine focused around 3D games, meanwhile Renderware is an engine that did a bit of everything, do I have the tools available to create a game with Unreal that can look and run like it would if made with a tool like Renderware?

Part of the aesthetics that came from that era of gaming were because of hardware and engine limitations at the time, I should not, or likely will not have any of those limitations present with the Unreal Engine, so how can I recreate the illusion of those limitations with this modern engine?

With my game, part of ensuring that I stay faithful to the aesthetics of the era, was by making sure that I captured the little quirks that were common in games of the era. One example is the UI for my game, which would of course be stylized, but in terms of animation it had to be snappy, a lot of UI back in the day was snappy and didn't rely much on animations or had very simple animation on screen.

Resident Evil 2 (PS1) uses a door opening animation between areas to hide the loading screen

The original Megaman games used a camera pan animation between areas, to give time to load the next area



documentation of production

unreal plugins

I will talk a bit more about pieces of research I did about development practices in the next slides, but before that I want to showcase the plugins that I utilized during development of my game.

Efficiency in modern day game development workflows comes in many forms, and one of them is in the form of engine plugins, back in the day the internet was nowhere near as widespread as it is today, and with the Unreal Engine, one of the main benefits that comes with using it is the number of user-created plugins that can be found online that perform various tasks. I briefly mentioned my UI needing to be snappy with short animations in the last slide, and while I could've already achieved such a solution with the stock engine, there is a plugin I utilized titled 'ScreenFade' which made it a lot easier for me to do animations on screen and tying it in with other logic at the same time. Another example is a Project Cleaner plugin, which does what it says on the tin, and cleans a project of unused files and maps out what is taking up most space where over the entire project. These are just a few examples of where I used plugins during my project, but the moral of the story is that game development has come far over the years, and certain things are much more widespread and efficient to do now than they were with older tools.

Screen fade transitions in my own game using one of the plugins in my project

Plugins present during my project development

documentation of production

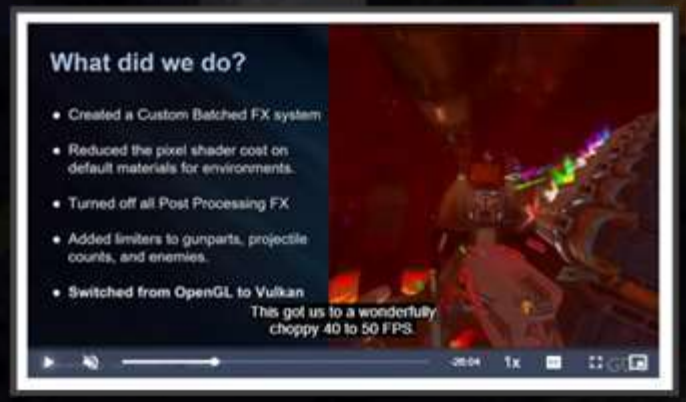
downside of modern solutions

Unreal Plugins are one example of modern game development workflow techniques that have proven to be efficient, but not every new tool is better than an old one when trying to create something old...

In the Shovel Knight GDC talk, the devs wanted to prove to people that games which look and play like old ones are still good, while most devs today focus on pushing graphical fidelity in games, the Shovel Knight team were using Google Sheets for all of their planning needs, and wrote the entire game in an in-house C++ engine, by modern standards this seems inefficient as there are many modern solutions readily available for creation of such simple games, and much more efficient programs for planning needs, but they ended up creating an experience that proved to be very polished and effective. One of the things mentioned in the talk, is that the devs purposefully did not write code for specific lighting effects in their engine, as they did not need such complex calculations, something which would've instead been present if they used an engine such as Unreal, which then would mean that they would've had to work around it.

...And that is unfortunately what the developers of Mothergunship VR encountered, something as simple as a concept of bullet hell in VR, was extremely taxing on performance for them due to how Unreal Engine did lighting calculations, the devs had to improvise and come up with very creative solutions around fixing these performance issues without affecting the artistic vision behind the game. [6]

Unreal Engine is a game engine primarily focused around 3D games, it is possible to make 2D games in Unreal Engine, and for a while there was an official plugin that assisted creation of 2D games in Unreal Engine titled Paper2D, that was actively getting worked on by the teams at Epic, until it got deprecated one day in favour of focus being put on other parts of the engine. A group of fans started developing their own plugin titled PaperZD that would continue what would've been development on Paper2D.



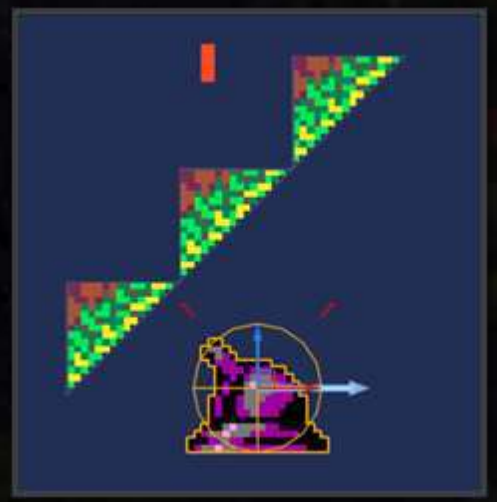
Snippet of the talk the Mothergunship VR devs gave at GDC, where they discuss the problems they ran into with the engine and what they had to do to fix them [6]

documentation of production

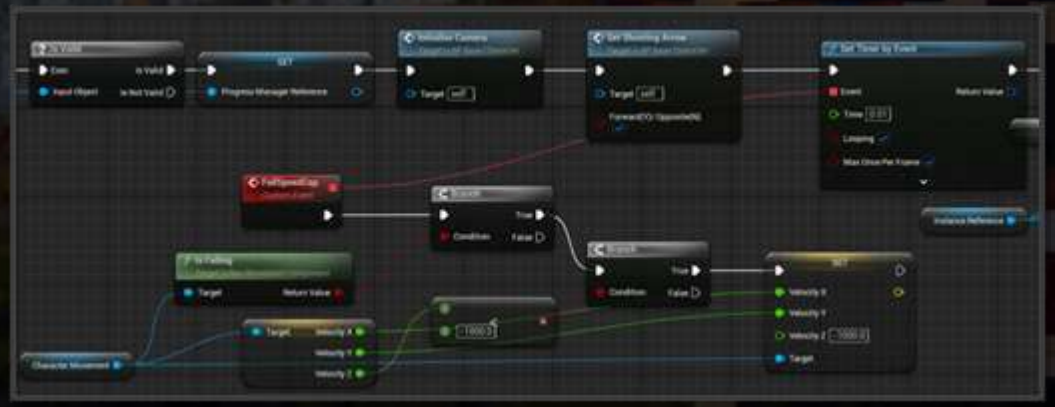
downside of modern solutions

What I soon encountered during development is that there were quirks about developing a 2D game in Unreal Engine that I had to unfortunately accept, ones which if I wanted to fix would require me re-writing most of the codebase that plugins like Paper2D and ZD are built on top of. One such example of an occurrence is in the PaperZD character class, which is built on top of the default Unreal player character class, which is of course built to work around 3D environments, and as a result the collision detection at the root of the character has to be a capsule component, which meant that the edges of the characters collision were curved.

This would cause unintended behaviours like instead of having the character hit their head on an angled ceiling, and either lose all air momentum or simply scale the ceiling during the duration of a jump before then falling back down, the character would have extra velocity added to its air trajectory because of the smoothly angled collision of the capsule adding such momentum. Of course if I had an idea of where I could 'fix' a quirk, then I would try doing so if it was reasonable and wouldn't take a hit on performance, one such example is the falling speed of the character, which the logic for it uses a variable called 'Gravity Scale' which also determines jump height for the character, so the higher I would want the character to jump, the faster they would fall, which because I was trying to stay faithful to the formula of Megaman simply did not yield good results, and I 'fixed' this by applying a Z axis velocity value cap that would apply whenever the characters downwards velocity would've exceeded a certain amount. Simply put, I had to improvise a lot around the plugin limitations.



Capsule components in engine



Falling speed cap solution for the player

documentation of production

downside of modern solutions

The Shovel Knight developers stuck with what they knew would work, being made in a custom C++ engine they had full control over aspects of their games. Learning about engines such as Renderware showed me how the push for graphics in video games overtime resulted in an engine that had solidified practices and solutions in place effectively getting wiped in place of a new one, which is not bad when the new solution is just as good or better, which in many ways the modern options around today are, but sometimes things end up inefficiently made as a result...

Unreal Engine is the leading engine in market share in the world at the current moment, it powers majority of games that come out today, but the push for graphical fidelity and experimental features in Unreal came at the cost of sacrificing other parts of its functionality, in this case making a 2D game in Unreal comes with unique problems and quirks which simply do not exist in other engines or even in older ones.

I found myself having to spend a lot more time developing solutions for things which on paper sounded simple, but in practice ended up proving to be complex in implementation in Unreal Engine, I have already talked about examples of issues I encountered with creating character movement, but there were bigger issues I encountered; UI elements in Unreal Engine are very efficient, and buttons for menu navigation came with a lot of nice logic blueprints, but the problem I found was that not all of this logic worked around keyboard or controller navigation as it was made to work around mouse input, which one of my solutions to my project problem was to implement controller support, as modern engines make this a lot easier to do compared to older ones, where controller support especially in PC titles was not ideal. This resulted in me having to create custom Button UI elements that replicated the functionality that was already there for mice, but not for controllers, doing so came with its own quirks, and so on.

The moral of the story, is that for every good feature that the engine offered, there were also several quirks and caveats that came with it, and I cover most of these in detail in later slides.

RenderWare

UNREAL ENGINE

documentation of production

change of plans

Around 9 weeks into the course of the 16 weeks that I had to develop this project, I had finished consuming and writing about my research avenues entirely and came to the conclusion that I had to change the scope and direction of the project slightly in terms of planning and documentation. After having come so far into the project using several platforms for documentation (Discord, Miro, Trello, etc.) I found that the Miro board I was using could do everything that I was doing in the other programs. The developers of Shovel Knight used Google Sheets for literally everything, and part of modern game development workflows is working out what works best for a project in regards to software planning solutions, so I eventually ended up migrating everything from my Discord and Trello board, onto one Miro board, and organized it like this:

I used a 'Skill Tree'-like layout, dividing the board into three core sections:
a breakdown of the module details, and sections for me to write about what I think I should do to achieve specific goals
Large boxes for notetaking, images, files and links, which let me categorize information properly
And lastly a monthly calendar where I would log my daily progress, which made it really easy to keep myself on track and know what I need to prioritize next

To put simply, halfway into the project based on all of my research verdicts and personal experiences with engine limitations around my project problem so far, I chose to improvise and put focus on making sure that the end product that I would produce would be quality over quantity, which in this case meant making sure that the core game design is solidified, that the UI was fully functional, and that the game was bug free. I ended up creating one complete level with a boss fight, with variation depending on the difficulty that the player chooses to play on.

Migrating everything that I had previously organized across different platforms, all into one place that proved efficient for my Project Development

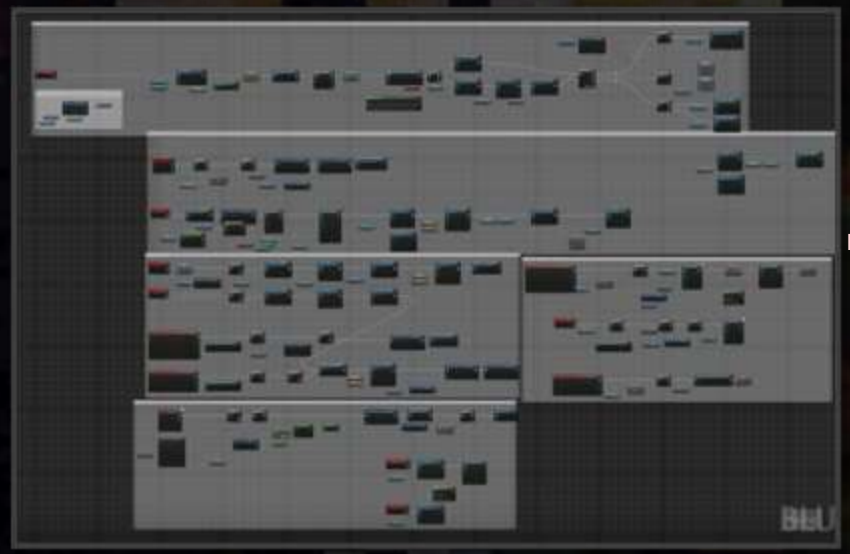
documentation of production

blueprint showcase

The following slides showcase snippets of core functionality that make up my game.

enemy system

The game features many enemies, I created a base enemy class with a lot of instance editable functionality, things such as health and attack values, sprites, sounds etc. were all customizable per instance of enemy in the level. I made the enemies function in a way where they can be ‘turned on and off’ by other actors, in the sense that when a player enters a new area, the enemies in the last area are no longer active, and the new enemies are turned on. Enemies are made to work around the core loop of the level too in several ways, examples being where enemy deaths are kept track of throughout the level and respawned as needed based on checkpoint locations and when the player loses a life.



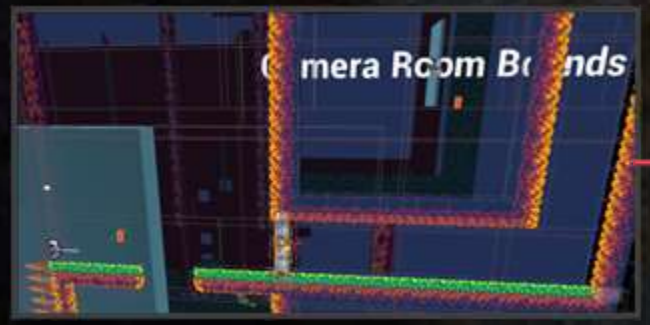
Screenshot of Base Enemy blueprint, every single child blueprint enemy had the functionality present here

Example of an enemy placed in my level with all of the customizable instance editable parameters on the side of the screen

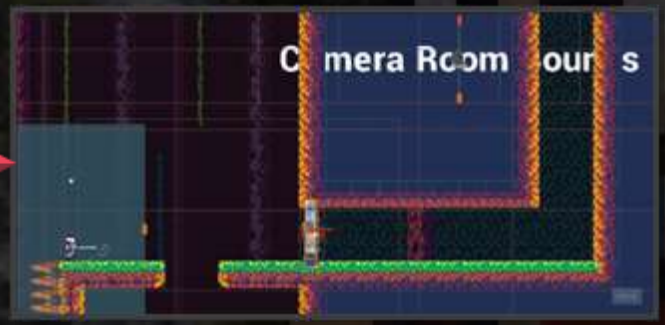
documentation of production

camera system

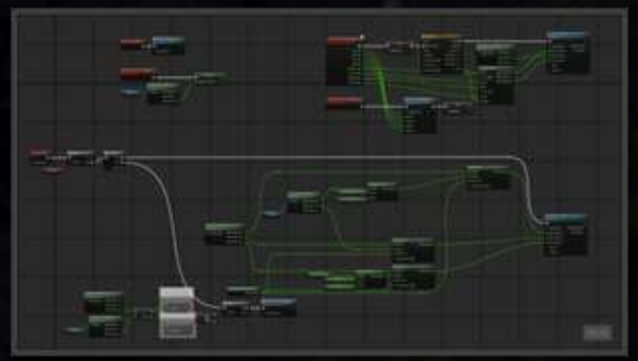
The camera system in my game is pretty complex and works directly with a lot of other actors during a level, particularly turning pools of enemies on and off as the player walks between two areas. There are a few actors that make up the system, the camera itself, bounding boxes, and transfer boxes, the bounding boxes are placed in the level and scaled to a size that determines how far the camera should be able to go before it stops in place, locking the camera into the bounds of that box, this was my technique behind separating the different areas to each other, these bounded values are passed into the camera, and then the right calculations are done to prevent the camera from going beyond those values in 3D world space. The transfer boxes are responsible for the slowed down camera pan effect between two areas, it has two cubes that are put into world space, and then the location values of those two cubes are taken and then a lerp math calculation is done between them, when the player moves through a box, it briefly detaches the camera from a bounder box, runs the location calculation between the two boxes, and once finished, attaches the camera back to the new bound box, creating the effect of transferring between two areas.

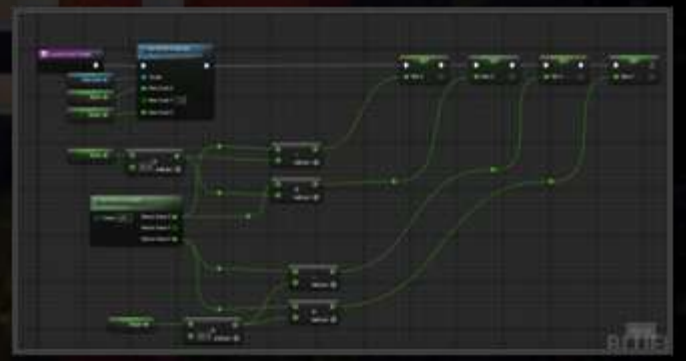


Placement of Camera Transfer Location boxes in 3D space



And how they perfectly align within the bounded camera boxes between areas, creating a seamless panning effect similar to the one in old Megaman titles





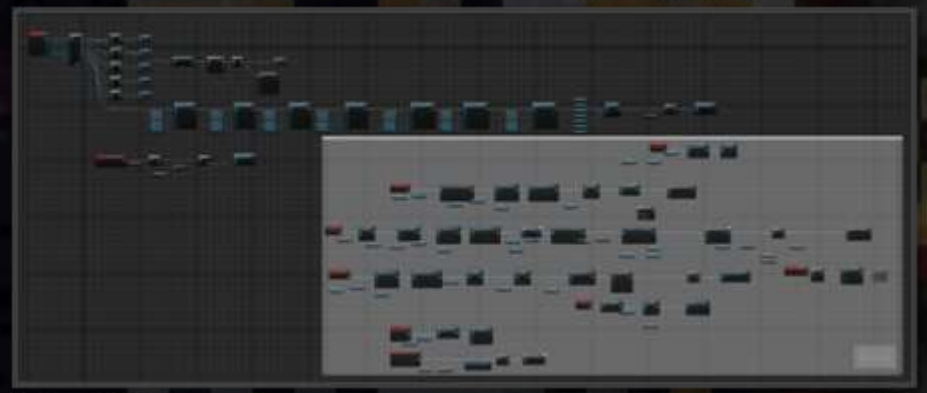
Screenshots of a few blueprints that make up this system, you can see a lot of green lines everywhere, and those are float values that make up the math behind making sure this calculation is done right. Something so simple as a camera panning effect like this, requires a lot of work in a 3D engine, something which would've likely been much simpler to do in a 2D based engine

documentation of production

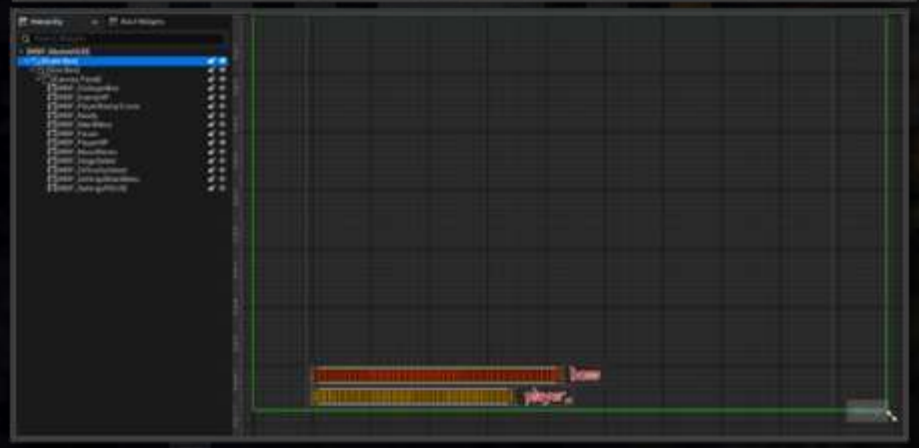
user interface

I put a lot of effort into making sure I nailed the UI for the game, the game has several menus, some of which are accessible during gameplay such as the pause and options menu. It is fully functional with keyboard and controller inputs and can also be navigated using a mouse pointer if one wishes to do so. I made the logic behind the user interface extremely efficient through making sure that all UI elements are linked together under one master HUD element, when the game starts the master element fetches all the necessary actors and blueprint logic that perform major and pre-requisite functionality during the games runtime, and can then pass references to those elements to other UI elements as they are needed, this means that the UI can easily manipulate game logic whenever it needs to without any issue. An example of this, is with the music and sound effect volume sliders in the settings menu that work globally across the whole game.

Additionally, I used resources online to workaround button widget logic restrictions that came with using a controller or keyboard to navigate menus [8], and to lock the games aspect ratio to 4:3, with support for Fullscreen, Windowed Fullscreen and Windowed view modes. [7]



Master HUD UI Widget Blueprint, notice how the initialisation chain at the start provides all of the UI elements in the game with the necessary references it needs to run logic at any time



Master HUD UI Widget Hierarchy

One of the more significant showcases of the UI logic working around major blueprint actors, outside of audio balancing in options, is with a cutscene system that I developed.

final artefact

character moveset

The character has a fixed running speed and can use a dash that permanently increases their speed by 1.5x for a max of 0.3 seconds. If they jump after a dash, then they jump with that dash speed! The players jump has a max height, however before that max height is reached, the player can let go of the button to control their jump height

When sliding down a fall, the characters falling speed is a lot lower than if they were naturally falling, they can then shoot off of the wall into the opposite direction, wall jump off of the wall, or keep jumping up the wall, as there is no limit to wall jumps!

The player can shoot indefinitely, with a tiny buffer between shots, the longer the player holds down the shoot button, the more powerful the shot output, with a total of three different strength outputs

final artefact

level diagrams

On entry to this area, if the player is walking, then the ghost will shoot them before they reach the jump upwards, something they will avoid if they dash instead, this way the dash mechanic is being slowly introduced to the player

Start of level, introduces ground fairy enemy to player, and the flying ghost above a pit, which is not shooting, so the player can safely make a mistake of falling down this pit and not get punished for it

Wide open area with ghost enemies that are flying in multiple angles, along with walking fairies that are standing in place and shooting from a distance, the open area should teach the player to use it to their advantage through avoiding projectiles by jumping

Spike hazards introduced, along with a lot more variation in platforming, in the lunatic difficulty, there is a larger density of spikes

Walking fairy enemy introduced, and the first wall jump is introduced, paired with a small rest area right after transferring into the next area

enemies

- **Ground Fairy**: Can move from fixed left and right positions on a map
- **Walking Fairy**: Same as previous, but also has the ability to shoot bullets
- **Flying Ghost**: Can fly in many trajectories on screen, and can also shoot bullets
- **Floating Gargoyle**: Floats in place or between two points, and can shoot downwards
- **Turret**: Shoots projectiles from a fixed location

In the Lunatic difficulty, small level layout adjustments are made, the player has half as much health, and enemies have twice as much health

final artefact

level diagrams

Death pitfalls are introduced to the player, the first pitfall is filled with spikes before the next jump being a full death pitfall, the ghost here also doesn't shoot, in the Lunatic difficulty this is changed, the spike pit if a full death pitfall and the ghost also shoots

A large spike jump is placed here, one which can barely be made with a full running jump, and which can be easily jumped over by dashing, again another technique to get the player using the dash, it was first introduced through a light trap in the previous area, and in the coming areas it becomes more evident through object placement like this on where it should be used. In Lunatic, there are more spikes here, meaning an even more precise jump is required

The player can make their way up this boxed area before getting greeted by a ghost enemy at the top, one which can be reached and killed before getting up there, if the player jumps on the floating platform which can only be reached through dashing and then shooting the enemy down. As mentioned previously, techniques like this are used to reinforce the dash mechanic. In Lunatic, the flying enemy is not reachable from the floating platform

final artefact

level diagrams

Second checkpoint, with a spike pit that is only passable by dash jumping

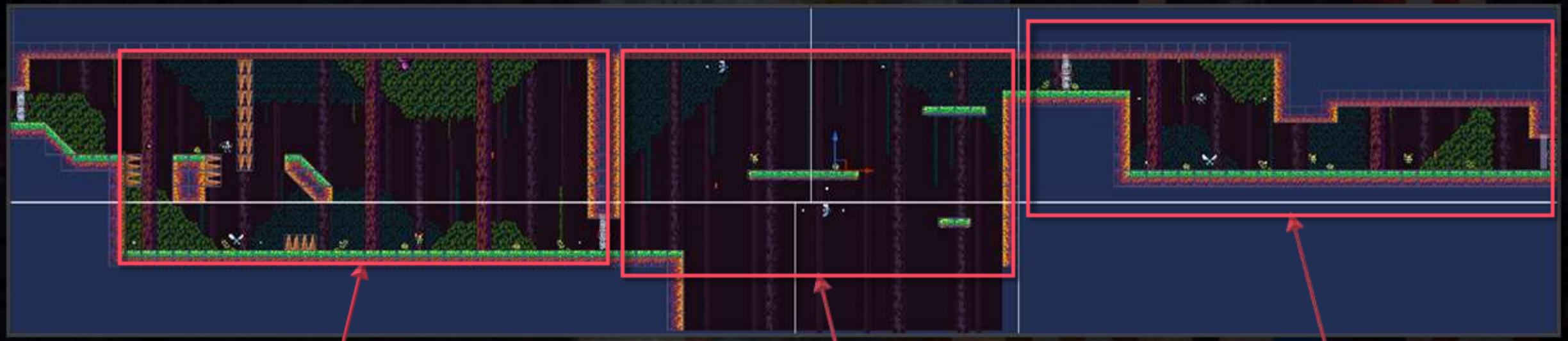
An optional area which has floating enemies lined up in a row, with every other one shooting faster than the other, testing the players knowledge on the dash and their reaction speed, if they get past, they are rewarded with a life and health pickup, before then exiting back out through a one way exit

Floating enemies are lined up in a sequence shooting downwards, covering the entire area with bullets, the player is forced to utilize wall jumping on one side, which they were made to do in the rest area, to jumping up small one block platforms and time their attacks at the enemies. In Lunatic, the player is forced to also use full charge attack here to destroy these enemies without getting damaged on every one block landing, justifying the mechanics use in the harder difficulty. The entrance to the optional area also becomes harder to reach, as in the Normal difficulty there is a spike missing above the entrance to make it easier, while in Lunatic the opposite is true

The first checkpoint of the level is here in a rest area, which has the player wall jumping all the way up to reach

final artefact

level diagrams



This section is a mix of all challenge concepts thrown together at once, with the only instance of the turret enemy being used from the ceiling, so the player knows that they should also be looking up instead of only looking at what is in front of them

This section tests the players platforming skills through floating platforms and a death pitfall beneath the whole area. In the lunatic difficulty, the first platform here has a ground fairy obstructing it, and the flying ghost at the top shoots bullets faster, forcing the player to take the one block gap in between the first platform and the ground, to land on the smallest platform lowest down, to then avoid getting hit.

This section starts with a walking fairy immediately shooting at the player, and tests the players reaction speed and also controlling jump height, as if the player does a full jump here, they hit their head on the wall, and come down quicker

final artefact

level diagrams

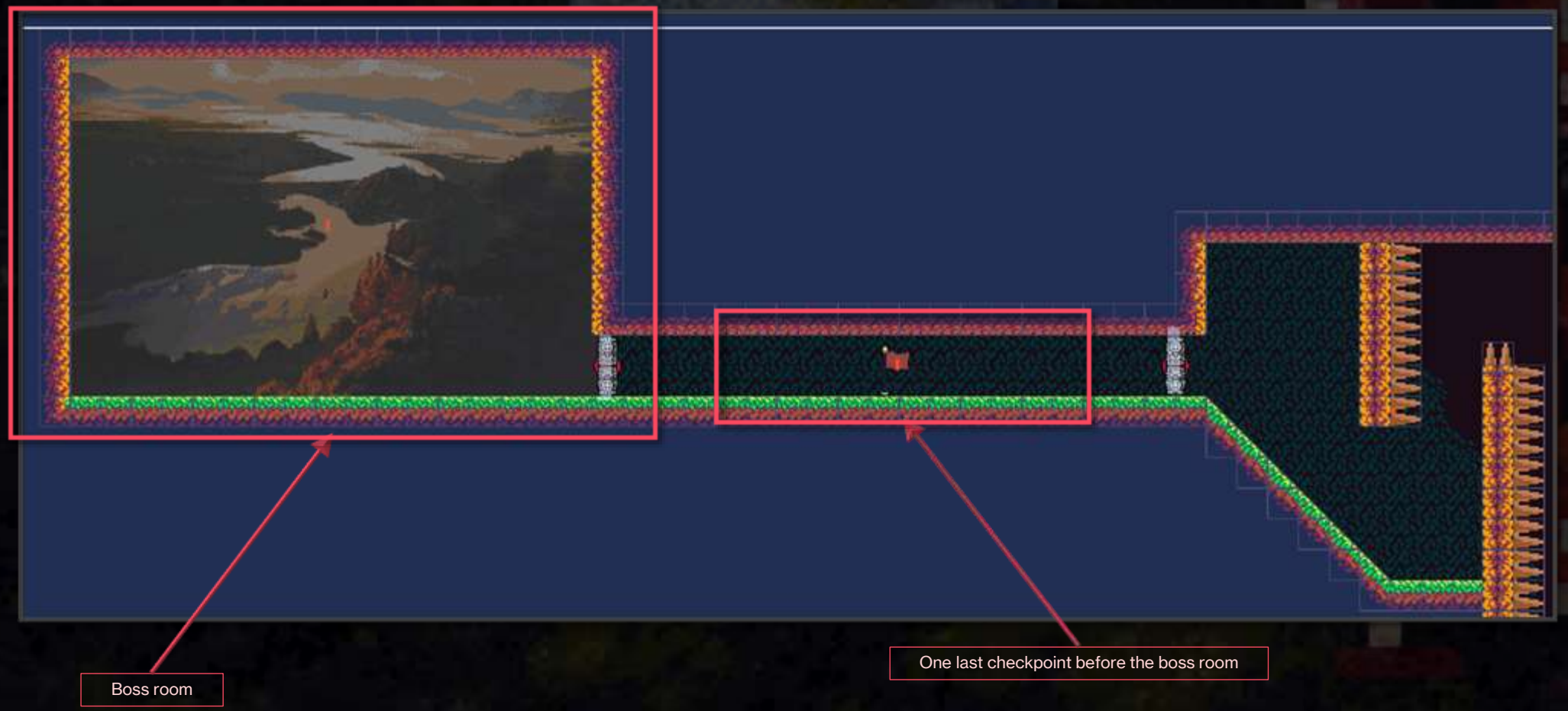
The last platforming challenge, the section here features two walls fully covered in spikes, which forces the player to dash jump and hit the opposite direction mid air to halt their movement, which will then let them fall safely between the two walls down, this is a test of the players knowledge of the dash paired with the jump so far. In lunatic, the pillars here are replaced with floating platforms, and they are also shorter in width

This section features the floating enemies again, all shooting at a set interval, it uses the concept from the optional area, but it is easier here, if the player visited the optional area before then they will be familiar with this and should be able to conquer it better than a player who did not go there

Rest area, it is important to break the level up into a few rest areas overtime to let players take a breather, and as a result they could also change some game settings if they wish, in the case of my game, the audio balancing between sound effects and music

final artefact

level diagrams



final artefact

bossfight

The fight starts with the boss running at the player, and shooting two bullets from a distance

After shooting the player, the boss will proceed to jump up from their position, and land at the players last standing position, then following up with either 1, 2 or 3 sword slashes sequentially, the number of slashes is random, this jump attack is repeated another two times. The random slash amount element, paired with the boss jumping at the players location, makes this an attack that requires very good use of the dash and reaction speeds to avoid

rain of bullet hell

At the end of the sequence, the boss flies up into either the left or right corner of the screen, and starts raining bullets down at the player, there is a shower of small bullets that come down in quick intervals, with four massive red bullets that come down at slower intervals and deal more damage. The position of all of these bullets have a small random deviation applied to them, so while the general pattern may be the same, the trajectory of the bullets is never the same

final artefact

menu showcase

evaluative review

evaluation on project outcome and future developments

I believe that I have created an experience here which perfectly captures the Megaman formula, as with every project things change along the way, as the purpose of a project outside of reaching the end goal, is to improvise around the challenges that one faces during its creation.

I have created a framework here that is actually very expandable, and if I could work more on this project I would do the following; As a result of my research it was evident that in order to innovate on the Megaman formula, one has to do so with caution and should do so with a lot of thought and effort put into innovating on it, be that through gimmick creation, new mechanics, gameplay options etc. I focused my level design around the mechanics that I could create within the constrains of the sprites I was using for my characters, making the level centred around using the dash mechanic that the player has, and a unique boss fight along with it that is made around the constrains of what the sprite sheet offers, however everything else that I used to make up my level is very expandable, my enemies and level layouts have the ability to change around the difficulty that the player chooses to play through the level, meaning that in future, I could totally implement new characters that the player could play, and then could make levels replayable by changing the layout of object and enemy placement based on the character that the player has chosen to play, because the systems I created around my enemies are very customizable. I would do this because this would be innovating on the gameplay formula of Megaman greatly as most games that come out today in the formula feature only one main playable character with a core moveset, and introducing new characters, which can all play the same level and get a slightly different experience, would be a twist and well justified aspect of the gameplay that would innovate on the formula greatly.

I learnt a lot and greatly developed my blueprint knowledge during the course of this project, being able to take concepts that I had implemented in past Unreal projects and expand on them greatly, making more efficient blueprint solutions than I have ever before, but the one thing that is evident after this whole project, is that unreal is an engine that is catered towards 3D experiences, it is evident that the modern features of developing in Unreal came with great benefits, but at what cost? The entire engines 2D blueprint functionality is now upkept by a fanmade plugin, of course at the end of the day it is down to the developers and their code optimization to make a game run well, Unreal Engine runs under C++ after all, so anything is possible, but at its core Unreal is an engine made for 3D titles, so one has to wonder if engines today truly have advanced in all aspects of development pipelines, or if they fall behind in specific tasks. I learned the hard way throughout the course of this project why developers choose to stick to specific engine versions and frameworks, some things are simply the best the way they are, and are too risky to change, and this is something that I expected throughout my project and improvised on properly, I initially started out intending to follow modern industry development pipelines wherever possible, through ensuring that I used multiple pieces of modern software for documentation and planning, through installing and researching several plugins for Unreal Engine that were made to make workflows more efficient, and through development I naturally improvised around the tools that I found most efficient to get this project complete to a quality standard.

Indie devs today who develop titles similar to the one that I have created here will therefore stick to development pipelines that by industry standards today are outdated, but will do so with good reason, it is clear that engines that hold such a monopoly over the games industry such as Unreal Engine favour specific aspects of game creation a lot more over others, neglecting specific things that were taken for granted back in the day, we see this all too often today with games releasing under Unreal Engine with terrible optimization and performance, and if the right steps are not taken in the future then it will only get worse, especially with tools such as generative AI, which I believe will be a contributor the downfall in quality of artistic expression in game products if not controlled properly. At the end of the day, I have answered my initial project problem, and proven that with the right tools, a product catered towards aesthetics and trends in video games in the 2000s is totally feasible with modern tools.

bibliography and references

references

[1] Official Written Source of Usage of Touhou Project for Derivative Works
https://touhou-project.news/guideline/

[2] Revisiting the legacy of 'Mega Man'
https://libstaff.primo.exlibrisgroup.com/permalink/44STA_INST/25n4at/cdi_proquest_wirefeeds_1992719928

[3] Mighty No. 9 - Is it Really Mighty? University Wire, 2016
https://libstaff.primo.exlibrisgroup.com/permalink/44STA_INST/7gb1bm/cdi_proquest_wirefeeds_1801920230

[4] GDC Talk - Ask Me Anything: Q&A with Yacht Club Games' David D'Angelo
https://gdcvault.com/play/1026935/Ask-Me-Anything-Q-A

[5] Video - RenderWare: The Engine that Powered an Era | Retrohistories
https://youtu.be/FKpDFIWK1DI?feature=shared

[6] GDC Talk - Future Realities Summit: A Match Made in Bullet Hell: Bringing 'Mothergunship' to VR
https://gdcvault.com/play/1029090/Future-Realities-Summit-A-Match

[7] Video - UE4 UMG Locking aspect ratio | Narovana
https://youtu.be/Ljek3wkaMkU?si=2KCYFiJpNhmO1GxS

[8] Video - UE4 - Easy Gamepad and Keyboard navigation in UMG menus | Leen Academy
https://youtu.be/25mUUEdXMsU?si=A65NpGR3nZzSEX0O

bibliography and references

bibliography of asset links

https://beepyeah.itch.io/8-bit-sfx-pack (sound effects)

https://outspacer.itch.io/platformer-sfx (sound)

https://aamatniekss.itch.io/deep-forest-pixel-tileset (Level tile textures)

https://shoehead.itch.io/lifter-8bit-cyber-platformer-pack (Enemy sprites)

https://shackhal.itch.io/multi-platformer-tileset (Level tile textures)

https://bdragon1727.itch.io/basic-pixel-health-bar-and-scroll-bar (Misc textures)

https://nyknck.itch.io/fx062 (Misc textures)

https://bdragon1727.itch.io/custom-border-and-panels-menu-all-part (UI Element textures)

https://bdragon1727.itch.io/pixel-buttons-pack-all (UI Element textures)

https://masayume.itch.io/pixel-art-fantasy-lands-1 (Level backgrounds, including for this document)

https://iknowkingrabbit.itch.io/heroic-creature-pack (Enemy sprites)

https://www.dafont.com/korean-calligraphy.font (UI Font)

https://frol-game-music.itch.io/touhoulike-touhou10-bgm-5 (Touhou-like BGM)

https://ryann1908.itch.io/mkii-shot-sheet (Projectile sprites)

https://code-farmer.itch.io/touhou-pixe-characters (Touhou pixel character sprites used in cutscenes)

https://www.deviantart.com/hansungkee/art/Rockman-Style-Reimu-and-Marisa-Touhou-Sprite-Sheet-807373356 (Touhou Marisa and Reimu Megaman themed spritesheets)

https://www.deviantart.com/aburtos/art/Enhanced-Rockman-Style-Reimu-and-Marisa-Sprites-809335170 (Slightly modified version of the above spritesheet that I ended up using in the end)

bibliography and references

bibliography of plugins used

https://www.fab.com/listings/680e2270-e28f-4ca8-af7d-a2a93fd873b6 Rename Tool

https://www.fab.com/listings/499efd25-257d-4a8f-b66a-b522de0abb46 Safe Delete

https://www.fab.com/listings/fdb7e77d-be37-4feb-a6c9-60e317c10adf Auto Size Comments

https://www.fab.com/listings/dd380608-c8fd-4036-b4e9-f3931ff8df45 ImageResizUEr

https://www.fab.com/listings/56e2cd87-1f6f-425a-a8f5-8f513e3c3316 Better Debug

https://www.fab.com/listings/ec8e92ff-074b-4246-8fd1-e7f2c8510d76 File System Library

https://www.fab.com/listings/2668be14-6731-446d-a99d-b2b59ce6f32e Log Viewer

https://www.fab.com/listings/d9892568-d368-4332-bc42-2421d0151949 ProjectCleaner

https://www.fab.com/listings/dbf64b8c-4d92-400f-ab8f-a60f5f7c4d83 Extra Win Function

https://www.fab.com/listings/6664e3b5-e376-47aa-a0dd-f7bbbd5b93c0 PaperZD

https://www.fab.com/listings/d3597f4c-9984-471a-b606-b096eb6d737e Screen Fade

https://www.fab.com/listings/ae2e2112-46dd-43c4-93c8-81a690338571 Time Clock

https://www.fab.com/listings/19731af8-db81-46ad-9681-be07e454c9e1 Monitor Utilities

https://www.fab.com/listings/d3dd43fa-99a9-4ce8-9242-78023113bec5 ReadlocalTXT

https://www.fab.com/listings/5f5b357f-c6c0-4466-b616-25db02071c8d Plugin Builder

https://www.fab.com/listings/30b4914d-b2ae-4a8b-8d63-8252e871492e Crystal Nodes

https://www.fab.com/listings/29a452f8-5851-49dd-80af-cb2ee4bc9bcb Blueprint Screenshot Tool

special thanks

ZUN
(team shanghai alice)
creator of the touhou project

https://en.touhouwiki.net/images/5/5b/ZUNPlayDoujin.jpg

https://media.vgm.io/artists/10/1/1-1379673508.png

And to all of the users who created assets that I ended up using in the game, especially ‘hansungkee’ on DeviantArt, without their Touhou-themed spritesheets, this project would’ve looked very different :)

fin