THE LEGEND OF ZELDA™

The Moral Link

A third person adventure puzzle game. Achieving ethical and moral game play though a mental and visual experience.

Reflective Report with Portfolio

Submitted as a required component for the degree of MA Games Development.

School of Computing

Teesside University

Middlesbrough TS1 3BA

Date: 19th February 2020

Lewis Simon Preen

Supervisor: Ian Sturrock

Second Supervisor: Chris Bailey

Disclaimer

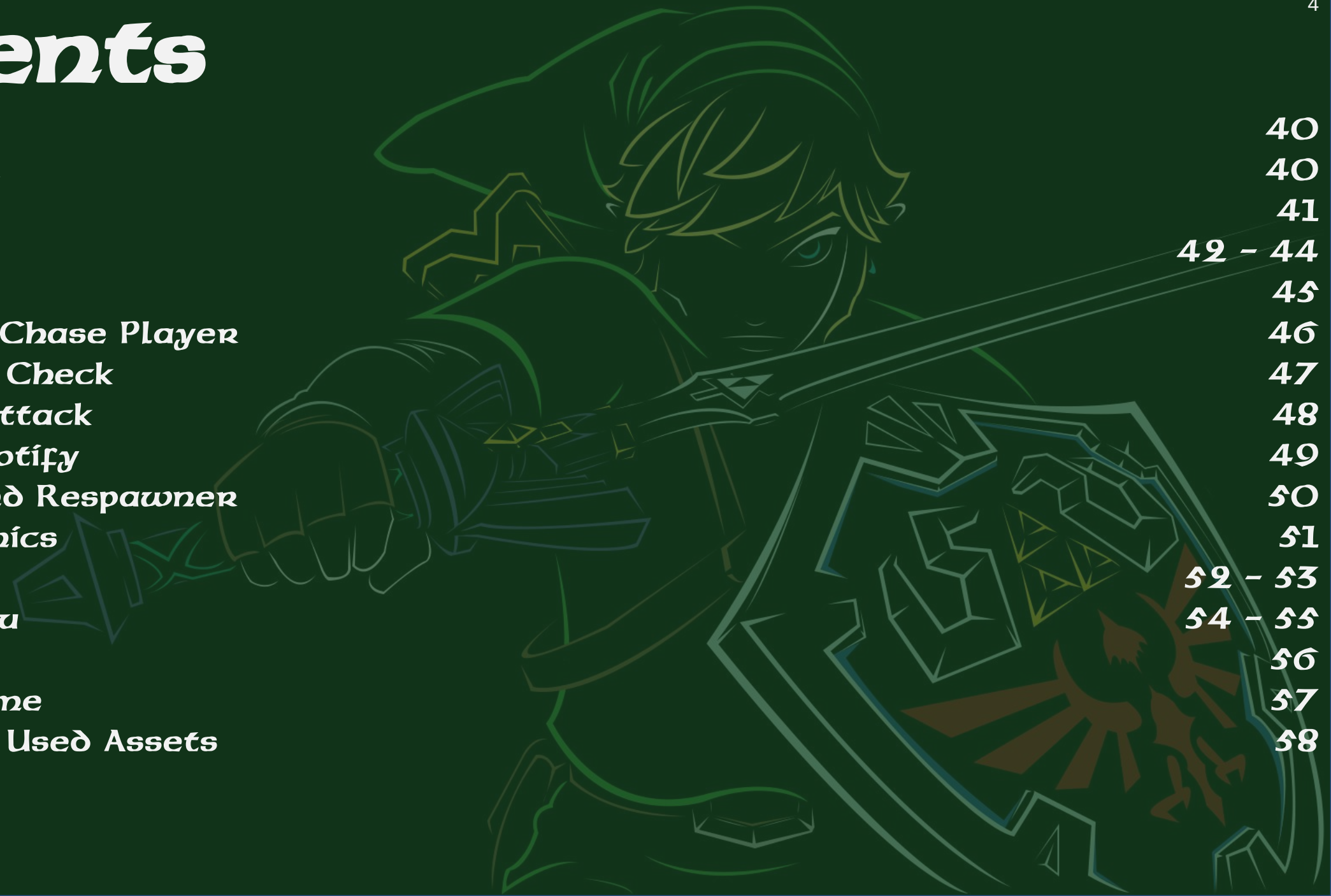
The following is a (non-profit) fan-based MA Degree project.

The Legend of Zelda franchise, characters and music are all licenced property of Nintendo Co., Ltd., Shigeru Miyamoto and Takashi Tezuka.

Any resemblance to real persons or other real-life entities is purely coincidental. All characters and other entities appearing in this work are fictitious. Any resemblance to real persons, dead or alive, or other real-life entities, past or present, is purely coincidental.

Contents

Contents



Project Management

I originally started using Trello to plan out my project as it was easily accessible, meaning I could see what I had to do and what I had accomplished. I decided to use Trello as it was free to use over other planning software such as Microsoft Project. I have also had experience using Trello when working on other projects. During the middle of the project I didn't need to boot up the internet everyday whilst working so I started using white boards hung above my monitors at home that I used to make notes and keep scheduling times on.

Story

The game is set in the land of Hyrule, Link has been away for a couple of years, during which time an evil entity has awoken and fractured the Triforce into several pieces. This caused the three founding Hylian Goddesses Din, Nayru and Farore to split and create a new Deity for each fractured piece, in which many of the various civilizations began worshipping these new Deities in their own way causing hatred towards one another. When Link arrives home, he finds a hostile new Hyrule in which he is forced to accomplish tasks in order to leave or enter specific villages and towns. Playing on his own moral compass, Link must collect the fractured pieces of the Triforce whilst keeping all the civilizations in good standing before bringing peace back to Hyrule.

Art, Animation, Sound & Visual Effects

As I am predominantly a designer for the game's animation and art, I have chosen to take advantage of a lot of the free assets available from the Epic Games UE4 Marketplace. I also have access to models and sprites via the free source websites www.models-resource.com and www.spriters-resource.com. I will also source assets and models from previous single and group projects if I feel they fit in with the art scheme of the project.

For the sound I will try and stick to Zelda sounds using www.zeldauniverse.net which is a non-profit Zelda website and as this game is non-profit project, I am not in breach of any copywrite laws, if I cite the original artist/author.

Development

For this project I decided to use Unreal Engine over other video game engines as I have more experience using this software. I started the project using version 4.19 but during the project I updated to 4.21 as some of the mechanics I had started to develop were outdated in 4.19 and there were faster and easier construction methods in the later versions of the engine. This also allowed me to use more assets from the Unreal store as they only worked with specific engines.

Third Party Evaluation

As this project was for a MA degree, I needed to have my project tested by third party play testers. I used two stages when developing my game, the first was internally creating and playing the game for its technical features ensuring everything was to scale. Play testing the games movement mechanics and ensuring that the quest settings worked and synchronized with the character journal system so I could hand it over for beta testing.

Beta testing was stage two which was done by third party play testers, who were play testing for game bugs and feedback. After receiving feedback, from the results from the Beta test, I begun implementing improvements to the project. I added combat system, an enemy Behaviour tree and new quests that only give you positive moral such as: delivery and hunting quests, where the player must take an object to a player and hunt a bunch of monsters respectively. The Journal function was updated to add a pause function and transparency so the player could be saved from enemy attack and could see anything around them.

Development Diary
Pre-Production

During the first three weeks of the project I spent researching various games that utilised some form of moral choice and began reading publications that helped with project production in this area. There was a lot of useful information online, however I found a lot of the papers I was reading to be a false analogy of moral choices with ethical gameplay and realistically they were just papers on choices in video games that had slight moral choices.

I then spent a couple of weeks learning Unreal Engine scripting using Custom Functions, Enumerations, Macros and Structures, in order to improve my Unreal skillset. I had previously used Custom functions and Enumerations in the past but only the dogmatic basics and simple functions. Most of my pre-production and basic mechanics were completed by July 2019, however I suffered a mental health set back and ending up being signed off for 15 weeks. When I got back to work around mid-November, I wasn't happy with my previous character controls, basic quest system and the inventory system I created, as I felt it wasn't up to a MA Degree standard. So, I began creating a whole new project from scratch based on my original research with a fresh outlook on the project.

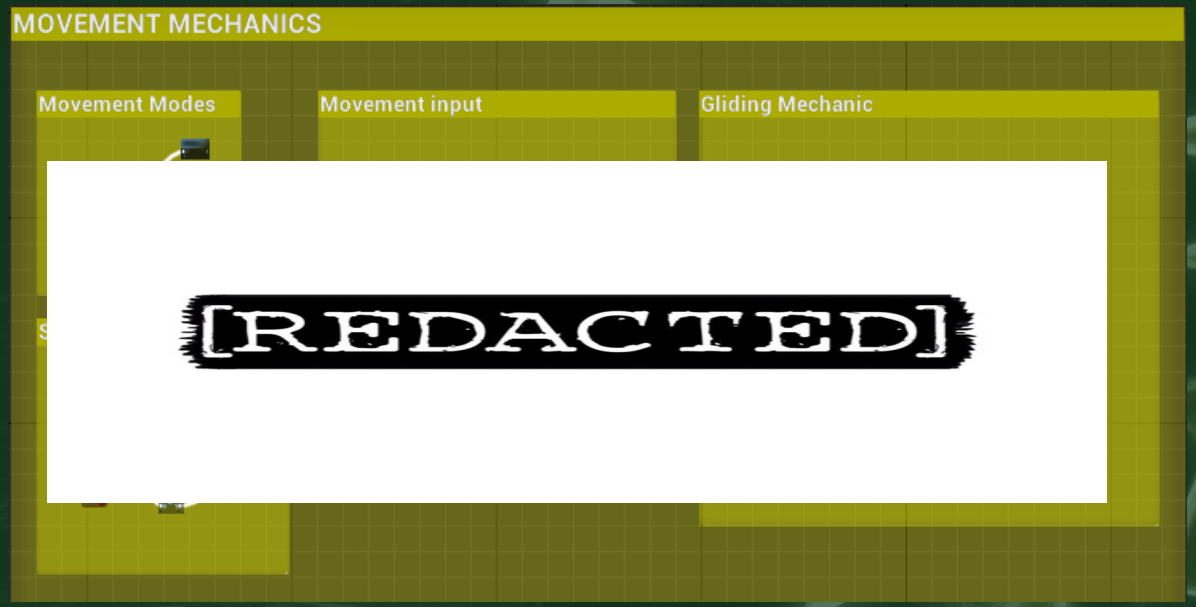
Player Character

Because I was basing my game on The Legend of Zelda franchise I begun researching and playing the latest instalment, The Breath of the Wild. I wanted to incorporate the player Hud and movement that this game has. This meant that I would needs to create a movement function, that makes the player run, sprint and walk. I also wanted to add other mechanics such as climbing, gliding and swimming. However, due to time restraints I was only able to incorporate the gliding mechanic form the latter three. I also wanted to create a health bar that worked like Zelda's heart system rather than a progression bar, similarly I wanted to also create a stamina wheel so when the player was exhausted there would be visual feedback for the player and also set the character to their minimum walk speed.

The final player character function I wanted to add from The Breath of the Wild was a Mini-Map that would help the player navigate around the games map, highlighting NPC's and objectives. I really wanted to try and add the sound motion detector and day and night cycle detector to the game, but time restraints meant I couldn't.

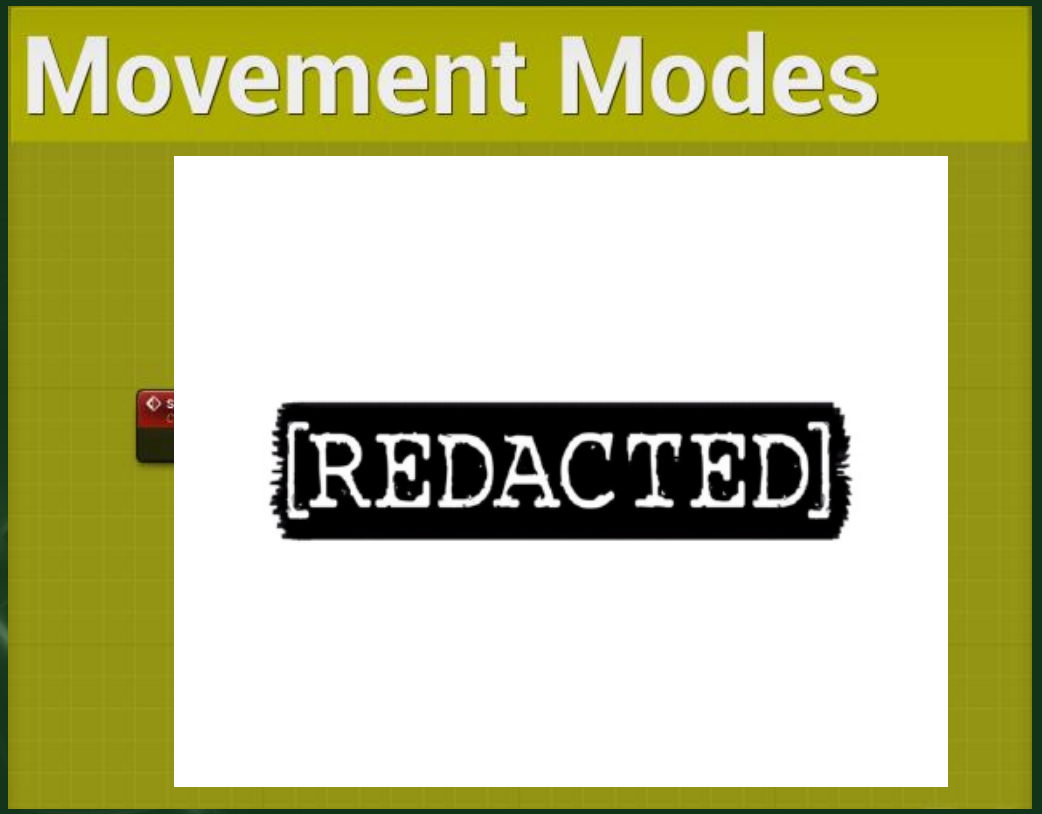
Player Movement

As mentioned earlier I wanted Link to have four movements Gliding, Run, Sprint and Walk, to achieve this I created custom functions and macros that all linked together and changed via an Enumeration. I used various Booleans with true and false statements to check if the player could do a specific movement.



Finalized Movement Mechanics.

Above are the custom functions that make allow the movement mode to transition between movement modes and the various functions.



Movement Modes that travel through an enumeration.

Sprint Mechanic.

The sprint mechanic checks that the sprint key is pushed and also if the player is gliding then activates the function. When the player is exhausted the Stamina regeneration function is launched.

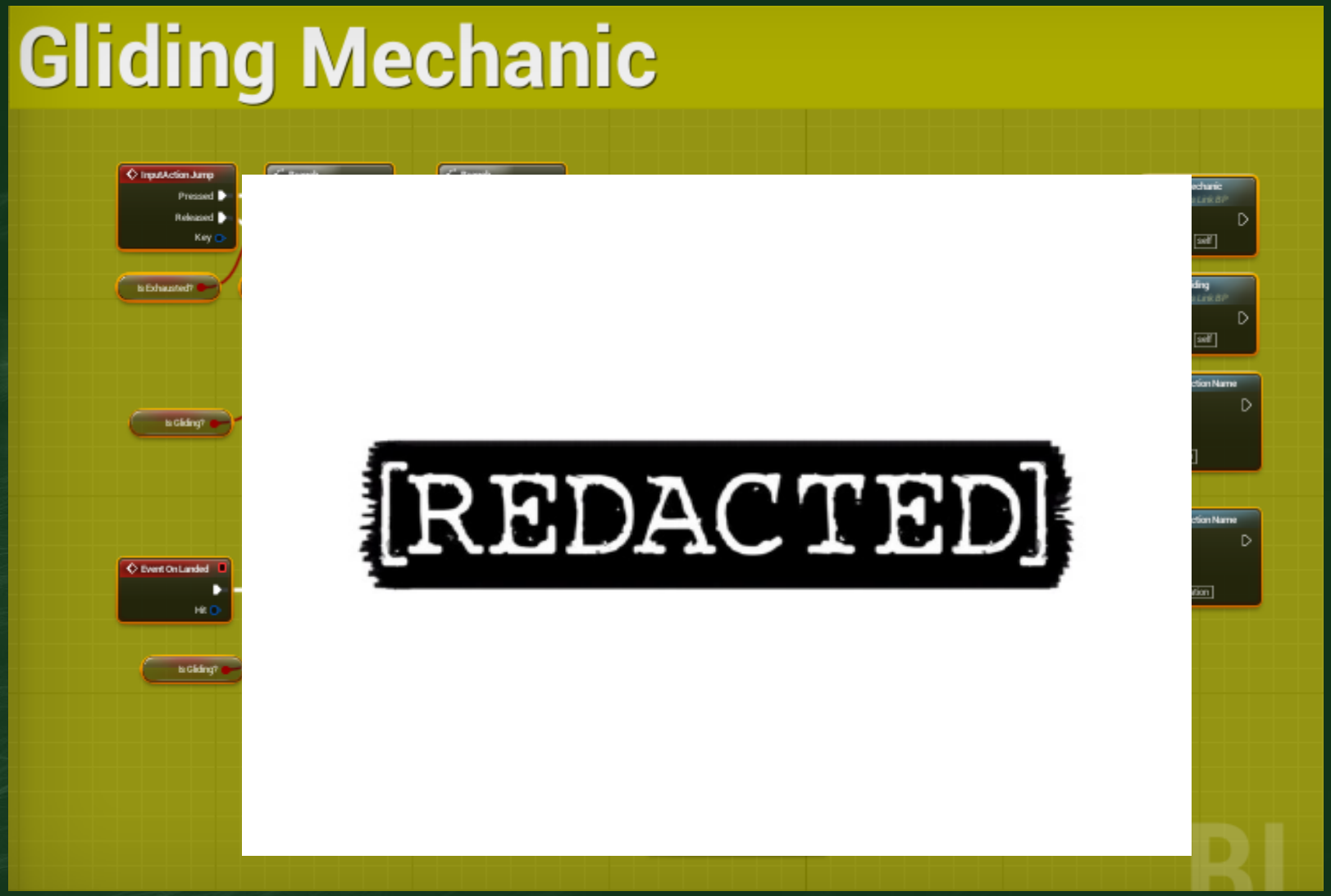
Sprint

Sprint Function

Sprint Function

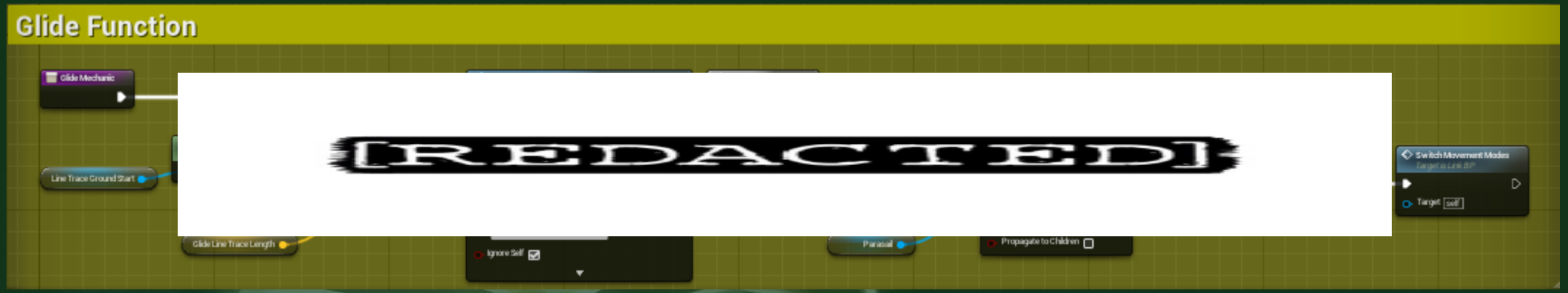
The sprint function checks the player is on the ground and sets the movement to sprinting.

Gliding Mechanic



Gliding Mechanic.

Above is the gliding mechanic that is activated during the jump function, it checks that the player isn't exhausted and also isn't in combat mode.



Gliding Functions.

Above is the gliding function that checks the player is a certain distance from the ground so they can deploy the glider and if they are the gliding movement is activated.

Below is the function to stop gliding which is just a simple boolean check and a enumeration change.

Stamina Depletion

Stamina Regeneration

Stamina Functions.

The top function shows how the stamina depletes as the player is sprinting and gliding.

The bottom function shows how when the player is exhausted the stamina begins to regenerate.

Exhausted Enum

Gliding Enum

Sprint Enum

Walking Enum

Enumeration Functions.

These are the various enumeration functions that are called upon when a specific movement mode is selected.

The Stamina wheel variable is explained later in the report.

Set Stamina Regeration Macro

Set Stamina Depletion Macro

Set Glide Settings Macro

Stop Gravity Settings Macro

Set to Sprint Macro

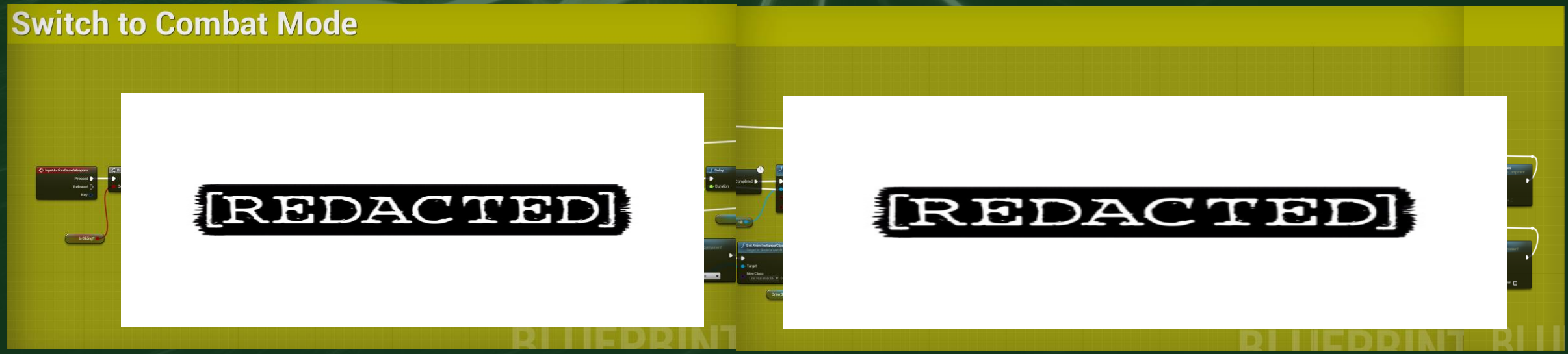
Set to Exhaust Macro

Macro Functions

The macros on the previous page and the one above are all utilized to set the various walk speeds, gravity scale and player air control.
The glide macro also shows how when the glider is activated it slightly launches the player in the air like a gust of wind, this was added for player feedback.

Player Combat

I implemented combat mechanics after play testers said that the game would benefit from combat, especially it being a Zelda game.
Below is the function that switches the game mode to combat it checks that the player isn't gliding this was so the player couldn't transition to a new animation whilst gliding.
During the change to combat mode the player transitions between two animation blend spaces and blue prints. Also before each switch the animation also has a montage that has Link sheathing and unsheathing his weaponry. One thing missing from this blue print is I set the character movement to zero when they are sheathing and unsheathing.



Sword Attack

Shield Block

Attacking and Blocking Functions.

The above function is for when the player attacks, it first checks if the players weapons are drawn and they are not blocking it also freezes the player on the spot so they do not skid around whilst the attack animation is playing.

The blocking function does the exact same as the attacking checks that the player is not attacking and also freezes movement speed and unfreezes whilst blocking and not blocking.

Player Health & Stamina

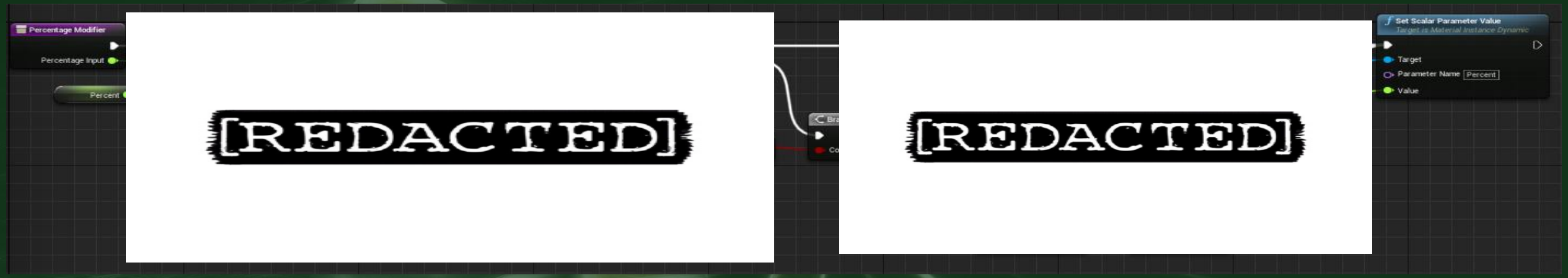
I wanted the health system to replicate that which is used in the Zelda series starting with three hearts and having the ability for the heart to decrease by quarters rather than the standard progression bar the UE4 uses.

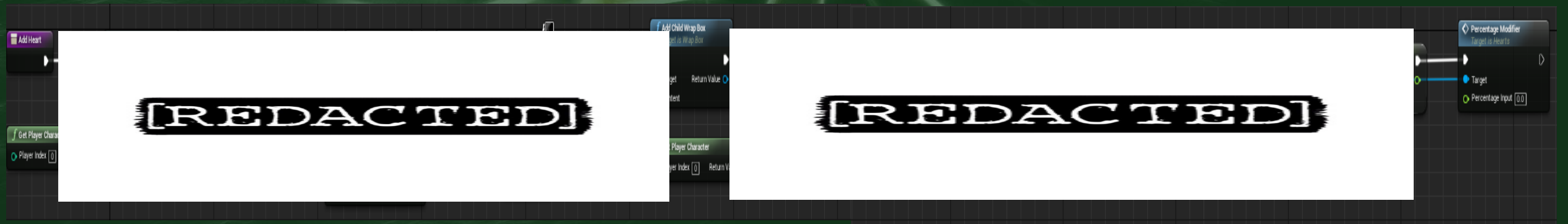
Heart Material.

Above you can see how the heart function works when it decreases by a certain percentage. Each heart has a value of 4 and so if you decrease 1 or 25% the player loses a quarter of heart as can be seen in the image to the right.

Percentage Modifier

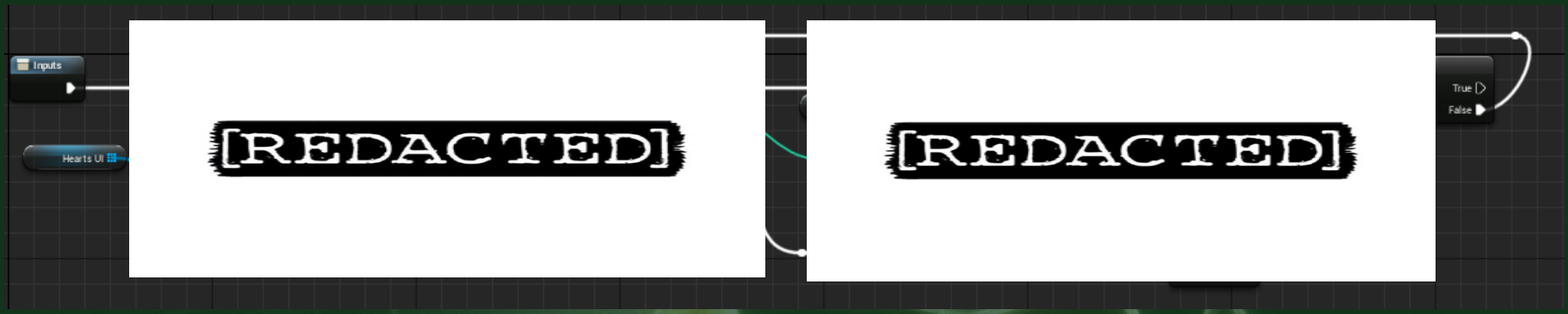
Below shows how the initial percentage modifier works to make sure only a quarter of heart is taken or added each time the player takes damage or receives healing respectively. Also checks if a heart box is full or empty so it can perform this task.





Add Heart function.

The above function shows how the player can gain more then three hearts however it first checks if the player has already got twenty hearts, which is the standard maximum in most Zelda games.



Find Hearts to Subtract Macro.

The blueprint above shows how the game finds hearts to subtract and what percentage when combined with the percentage modifier.

Find Hearts to Add Macro.

The blueprint to the right shows how the game finds hearts to add back to the character after receiving them.

HUD Event Construct and Update Hearts function.

The above functions show that at the start of the game the player starts with three hearts as the index is set at two.

The update hearts function shows how the macros on the pervious slide work with the percentage modifier.

Modify Health.

In the character blueprint is where the modify health function is set. The above image shows how the function works with the HUDS update hearts function.

Event any Damage.

The functions to the left shows that any damage receiving will modify the health from the above function, in the Enemy Blueprint you will see how blocking prevents damage.

I wanted the stamina system to replicate Breath of the Wilds having a circular bar that depletes green and regenerates the same colour if not fully used. But when the player reached zero stamina the bar will turn red and the player will be exhausted until the regeneration event has finished its full cycle.

Samina Material.

Above you can see how the stamina wheel works when it decreases by a certain decimal. Also as seen in the furthest white box that shows flow you can see that when the wheel decreases makes a perfect circle. The image to the right shows what the player will see on the screen during the use of the stamina function.

Samina Function.

Above you can see how the stamina wheel works on the HUD. For this to work I had to use an event tick, which I usually hate using as it can be performance heavy in the engine but because I couldn't find a work around and it wasn't that performance heavy I didn't mind using an event tick.

As you can see it shows how when Link is exhausted is changes from red to green.

Player HUD

I wanted everything to run via the main player HUD so all the widgets all feed to one location and call upon one another. The image on the next page shows what the players HUD looks like with everything running at once except the Quest Journal.

Like all Zelda games the health bar is situated in the top left hand corner of the screen, the stamina wheel is just off from the player character just like the Breath of the Wild as well as the players Mini-Map which is located in the bottom left hand corner.

The left and right side also have two new functions which are the Moral System and Quest List entry respectively. These do not stay on the screen permanently like the hearts and the mini-map. They work via a console command that can slide in the widgets using the M and Q keys. When the player picks up a new quest the widget will automatically slide in and the player has the choice to slide it out if they desire. Same with the Moral System if you enter a new region the morals of that region change so to update the player is slides in.

The stamina bar only appears when the player is either gliding and sprinting that shows it decreasing and regenerating as soon as its back at 100% it disappears.


Kakariko Village
0
Adventure Log
Current Quests
Completed Quests
Failed Quests
50 / 100


Kakairiko Village
0
Lourdes
Press [E] to interact!
30/100
20/100
30/100
Let the hunt begin!
Hunt Spiders:
0/3!
53

Player Mini-Map

The player Mini-Map using a render target from a camera attached to the top of the character that only shows specific items. Such as NPC's as dots, the player counter as a central direction arrow. If an NPC has a quest a ? will appear on the radar, that will or will not move if the NPC is patrolling. If the layer has to find a specific location the yellow arrow will move around the mini-map like a compass, it also shows the player how may steps they have to take to reach a destination. This will also show a coloured circled location on the mini-map to inform the player they are close to the location of whatever they are hunting,

Update Direction Arrow function.

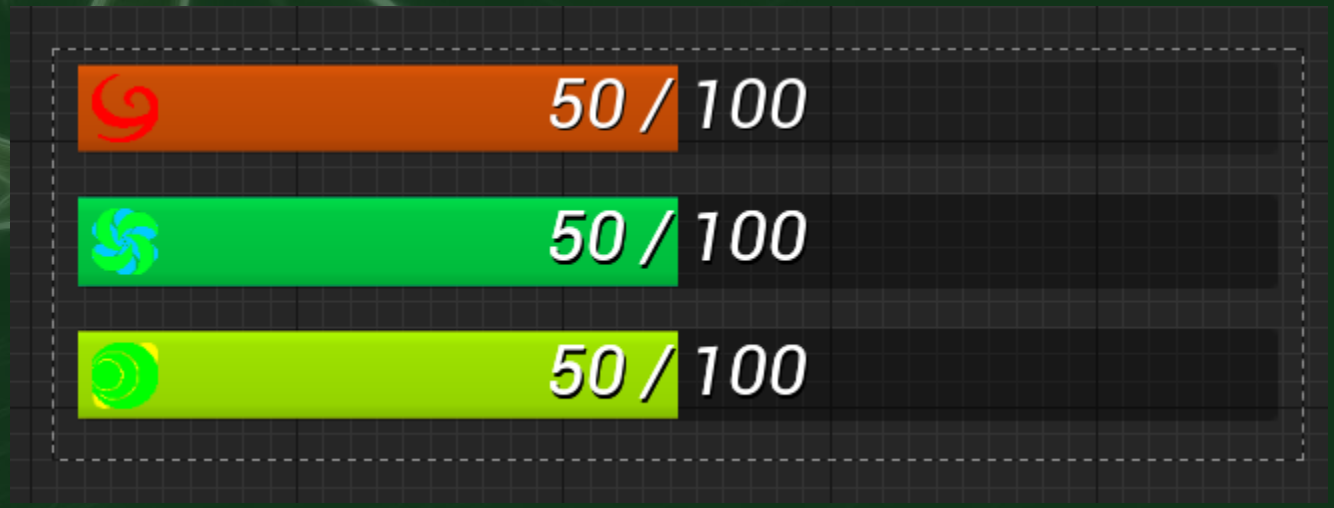
The blueprint to the left is from inside the Quest Manager system and shows how the mini-map arrow updates depending on the character and current goal locations.

Moral System

The moral system is the integral part of this prototype and was on of the first things I began development on. It is essentially just a progress bar that most games use for health, stamina or even magic in RPG's. Because of how simple is was to create an experience system I wanted to focus more on the story arch and narrative of the game creating Quests that will morally test the player. It is also another reason why I wanted to create additional movement controls for the character and also a quest system. As I felt that the Moral system alone would not be a true reflection of my skills in the Unreal Engine.

Moral Experience Bar.

The widget to the right is what the player sees when the moral counter slides in. I have five factions in the game and the bottom two progress bars change colours and the factions symbol also changes. The reason the bottom two have merged symbols is because in the editor the bars do no act as hidden until the game is activated.



Moral Experience Adding and Updating.

The blueprints here show how the moral system updates widget and also gains additional moral points (which are clamped at 100 in the rewards functions of the Quests)

Factions

For the purposes of the prototype I created five factions that the player can interact with. Each faction has their own beliefs and morals which dictates how they present themselves to Link. Each faction has their own symbol so when the player is given a quest or challenge they can easily track which faction they are currently working for.

Bad Boys Club

Bunch of young adults who like to cause trouble for the citizens of Kakariko Village, they were originally children of the Lost Woods before the Forrest Girls faction lead by Saria kicked them out.

Old Man

The original Mayor of Kakariko Village who just wants to retire in his garden but gets a lot of trouble from the Bad Bous Club.

Mayor Kakariko

Mayor of the Village who was forced to distance himself from the Lost Woods residence after the Triforce fractured and corrupted the Forrest dwellers.

Forrest Girls and Saria

The Forrest dwellers of the lost woods became bitter and twisted towards the Hylian folk with the destruction of the Triforce and mutated them into plant people. Their leader Saria has become a matriarch and finds all outsiders untrustworthy. She has memories of Link but he must prove himself to her people before she will entertain him.

Quest System

To fill out my project I started work on a complex Quest system that worked between multiple widgets and other assets such as NPC AI and Interactable Objects. Within this quest system was a Journal that the player could track various quests that they have participated in, currently ongoing or failed.

Journal

The player Journal was easily accessible for the player to track quests. Originally I just had a quest list that appeared at the side of the screen as mentioned earlier when the Q key was pressed. The journal was added to give the game narrative and help tell the story, inside the journal I could fill out a detailed description of events that the NPC's could of said without having to worry about creating elaborate dialog conversations and cutscenes. This would all be done via the Quest blueprints

Updating Journal Description.

This blueprint shows a simple piece of coding that changes the Quest Description.

Current Quests

Current Quests

Current Quests

Quest Name Goes Here!

Region Name

Quest Type

Detailed Quest and story goes here, more information to help the player

Goals:

Reward:

+ OO Moral

+ OO Region

Select Cancel

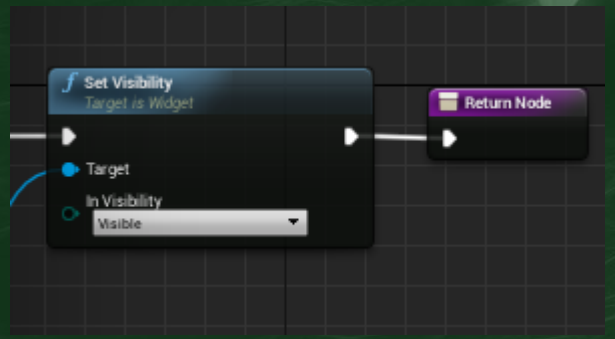
Generate SubGoals & Adding Quest list.

The top blueprint is how the many various SubGoals appear under the Journals Goal box for example if the player had to do more then one task such as find a flower and hunt 2 spiders.

The bottom blueprint demonstrates how a simplified version of the quest appears under its specific category (Current, Completed and Failed).


Update Detail Window
IsValid
Selected Quest
SetText (Text)
SetText (Text)
Set Color and Opacity
SetText (Text)
In Visibility
Hidden
[REDACTED]


Set Visibility
SetText (Text)
Set Visibility
SetText (Text)
[REDACTED]

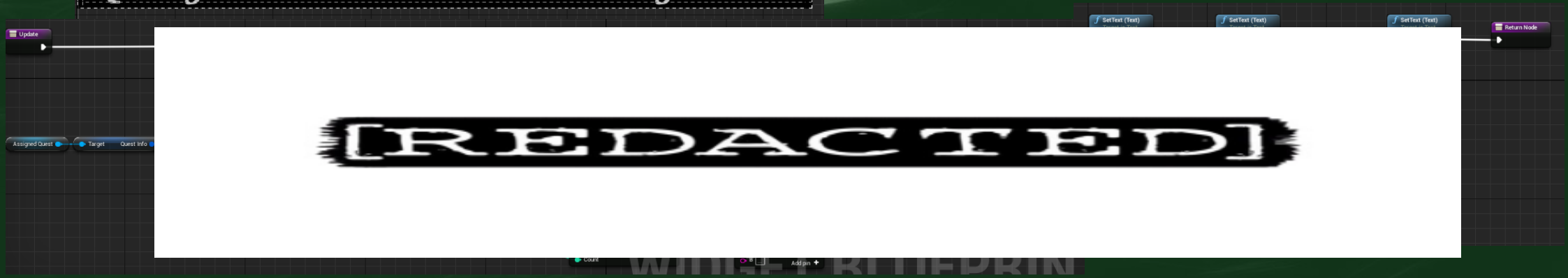


Updating the Journal.

This long blueprint is what goes into updating every aspect of the Journal from the quest type, the moral givers, region allocation, quest details and rewards.

Category Access.

Pressing the Quest Category will rotate the Hylian symbol 90 degrees and then reveal the below Quest entry. Upon clicking the Quest entry the journal updates to the corresponding Quest information.



NPC

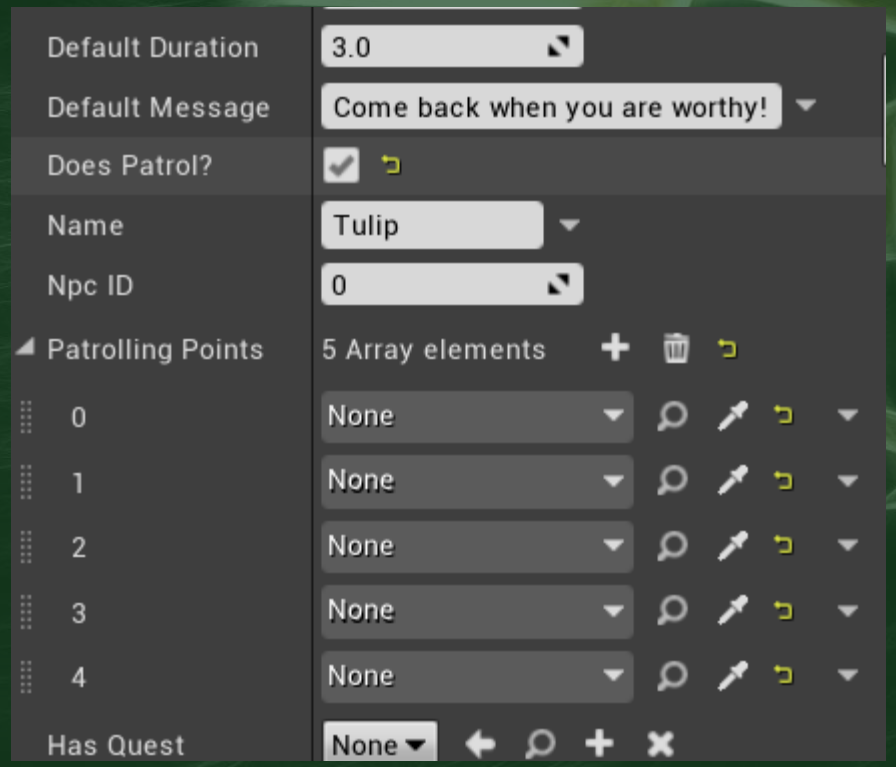
As I was going to be using various NPC's that had similar characteristics, I created a Master blueprint in which I could create children that had the same core instructions but had editable instructions and own features.

NPC Interaction

The interaction with the AI is quite simple, using a blueprint interface I created a interaction tool that worked along side the player characters collision box, so when they enter the AI space and leave it would trigger and event that allowed the player to interact with either an NPC or an object such as a collectable or a sign post that had writing on it.

NPC Movement

The movement of the NPC was quite easy to achieve as I had worked with patrolling points before and was a simple blueprint that just worked with a billboard function that allowed me to place random points in the map and then assign the patrolling NPC with a set destination and in what order they walked to specific points.



NPC Individuality.

To the left is the editable NPC characteristics. I can name each NPC, give them an ID if there are multiple AI in one quest. Change their default message and how long it is show for. Set their Quest, set the prerequisite to the quest and also a message for after the quest is done. And finally set patrol points which is only allocated if the Does Patrol function is set to True.

Enemy AI

As with my NPC, I made the AI as a master blueprint so if I wanted to create different enemies with various individual characteristics I could, such as; health, damage, speed and look/animation.

Enemy Launch.

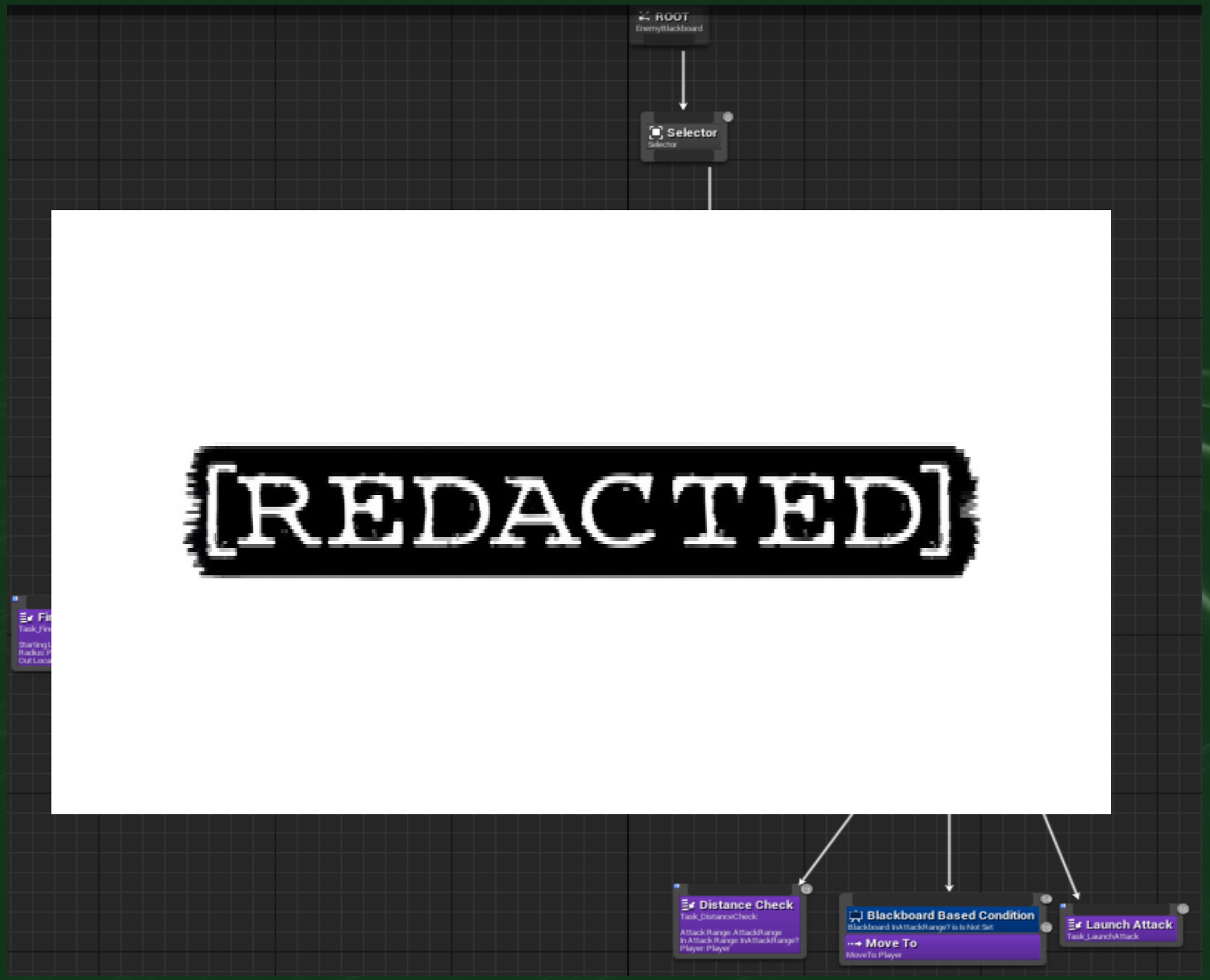
Above is the begin play/launch for the enemy that sets the health of the enemy and checks the various key selectors of the behaviour tree.

Enemy UI.

Above is set up for the Enemy's UI which shows there health as a bar and the enemy name.

Enemy Health.

Basic health set up for the Enemy.



AI Behaviour Tree.

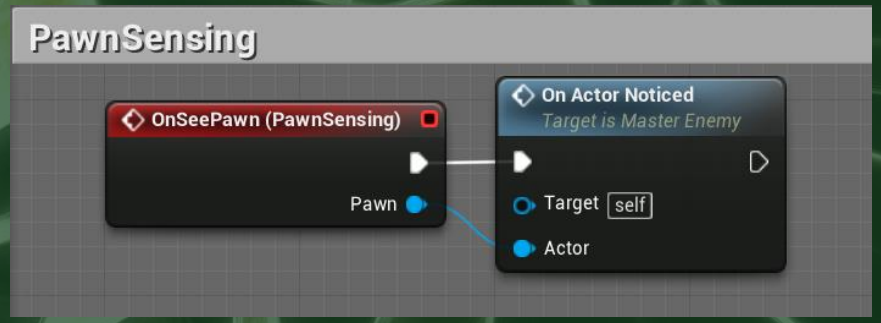
To the left is the Enemy AI Behaviour Tree that shows the various sequences and checks that the Enemy blueprinting goes through in order to patrol, chase the player when noticed, attack the player and also when to retreat when the player has gained distance on the Enemy.

Enemy Patrol

For the enemy I didn't want them to move linear like the NPC's did and I wanted them to have freedom of movement so this is where a behaviour tree comes in handy. Below are functions in the Enemy blueprint and behaviour tree task blueprints.

Enemy Sight Check Player

The Enemy AI uses a pawn sensing ability that links to the behaviour tree and once it has noticed the player character its movement speed increases to a sprint and begins to chase the player.



Enemy Distance Check

The below blueprint checks if the player character firstly is in range to be chased and then if they are in range of an attack.

Enemy Launch Attack

Most of the Enemy instructions for the attack phase lie within the Enemy master blueprint. The main execute though is attached to the behaviour tree through a blueprint task.

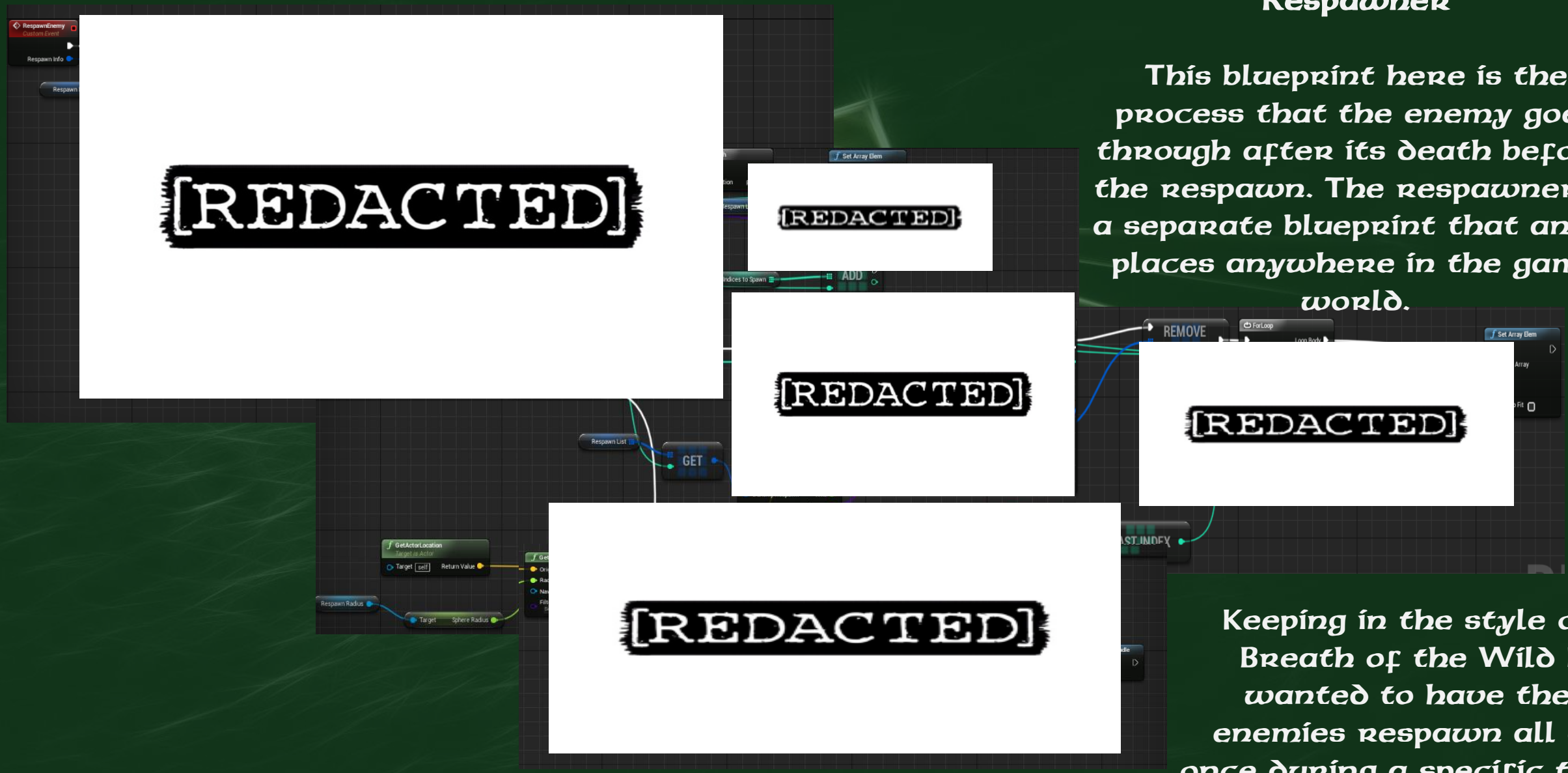
Launch Attack.

Below is the blueprint script that calls upon the launch attack checking the actor location is in range and then selects a montage for the animation to change to an attacking animation.

Enemy Attack Notify

The attack notify actions checks the attack distance of the player and enemy then if they are inline the enemy attacks the player modifying the players health the Attack damage is a individual to each Enemy class. One final check is that if the player is blocking they will take no damage.

Enemy Death and Respawner



Respawner

This blueprint here is the process that the enemy goes through after its death before the respawn. The respawner is a separate blueprint that an be places anywhere in the game world.

Keeping in the style of Breath of the Wild I wanted to have the enemies respawn all at once during a specific time.

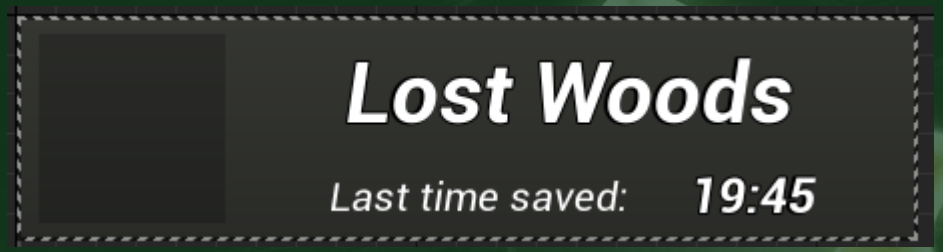
In-Game Mechanics

I also wanted to learn how to create a proper save and load system. I worked on creating a system that features the game starting with the player being able to new game or load a previous save state as soon as the game launches. Then when the player selects the save function they can choose to save to a slot and continue their current game.

Event being play and show save widget.

The above blueprint launches the save/load widget that has he new game and load game functions. The below blueprint launches the save function when the save button is pressed.

Save



Save Slot.

This image to the left is a generic saved file. The update function when saving the game saves the Region and also the time the game was saved. The dark image was supposed to be an in-game screenshot but I couldn't figure out how to get it to work.

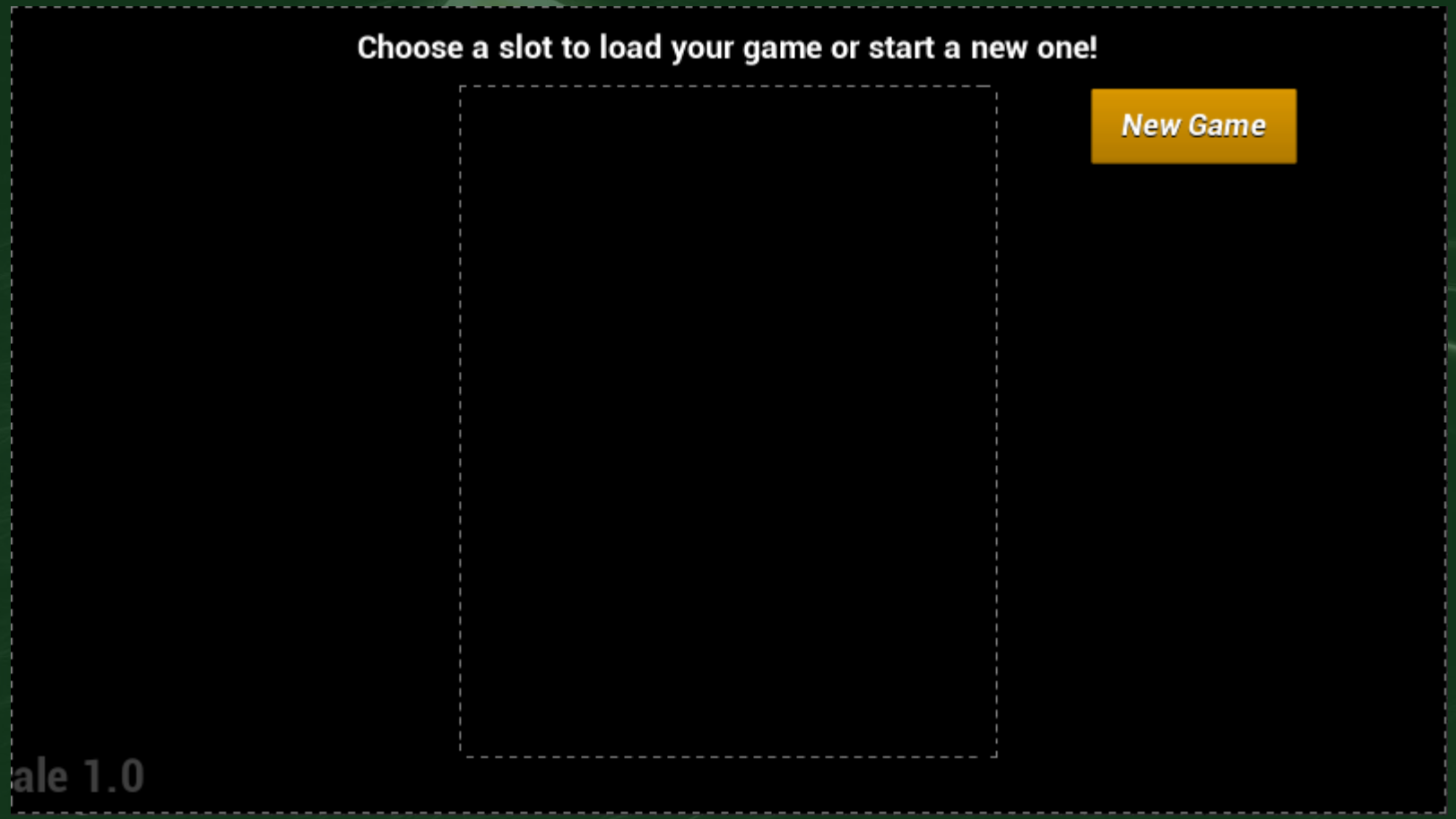
Generate Save Slots.

When the save file launches up this blueprint runs to generate the amount of slots available for the player, which is editable in the player blueprint.

Saving the Game.

This blueprint is the whole save function that locates the slot and then calls between the player variables and the save variables and makes sure everything is saved to its current statistics.

Load & New Game Menu



Main Menu Screen.

The load and new game slot use the same wiget the only difference is the orange button changes from saying "New Game" to "Continue" when it is updated and you select save game whilst playing the game. The centre panel is where all the save slots appear once they have been activated.

Loading a Save Slot.

Similar to the save slot the load function works with the player settings and sets all the current statistics to the previously saved files.

Reflective

I wasn't completely satisfied with this project which can be seen in the final hand-in as during the last build there was a bug that cause the game not to open. Upon restoring to an original version of the project I had located the bug written into the NPC and Quest system that wouldn't allow the Question mark above the NPC to show when they had a task available. After fixing this however the moral system decided to break and it meant the game would only add moral to the percentage bar and not negate anything, this causes quite a bit of stress so in the end I had to scrap a lot of the project and handed in pretty much bare bones and it now looks and feels like a Games Design basic package that anyone could pick up and make their own.

Everything else in the project works fine such as character movement mechanics, enemy AI behaviour tree and the quest system, but sadly the main part of the game being the moral system was corrupt and I didn't have enough time to fix this. Which is quite upsetting because before the build the game worked fine and I was able to get some positive feedback on the game, hindsight is a wonderful thing and if I knew this was going to happen I would of recorded the game testing and have used that to aid my physical hand in.

I also believe that I came back to the project too soon as during the first few weeks I had some mental health issues that put my project on hold for the duration of the course which meant I initially was going to fail the project. Fortunately for me the University and my lectures understood the situation and aided me as much as they could, but personally speaking as I mentioned above I think I may have come back to soon. This was mainly a finical decision as I couldn't fund myself for another year and decided to power on through the project no matter the outcome

Learning Outcome

This project was a giant learning curve for me as I got to experiment more within the Unreal Engine then I have in previous projects. I learnt how to implement parent classes more efficiently meaning that I could be more creative with my child class actors and they were easily editable within the Unreal level editor. I also improved my character development skills researching and learning how to create functions that work along side macros making my blueprints a lot stronger.

In previous projects I have used behaviour trees for AI using simple sight and attack commands but in this project I learnt more technical skills when it came to working with behaviour trees such as enemy patrolling, distance checking and creating a respawn system on death.

Reference for Used Assets

- Advanced Village Pack – Unreal Engine Market place
- Forest Girl Model and Animations - www.Mixamo.com
- Hylian Shield - www.themodelers-resource.com
- Link Character from Breath of the Wild - www.themodelers-resource.com
- Master Sword - www.themodelers-resource.com
- Spider Model – Unreal Engine Market Place (Infinity Blade Adversaries Pack)