Thesis by Lloyd Phillips

CONTENTS

ABSTRACT

This thesis undergoes the process of a group project undergoing the development of a game. The game is called AVA and it is a 3rd person puzzle walking simulator game produced by 5 artists and a programmer. This piece explains what planning was put into the games development process, what it's like to work with artists and how the final game came to be. Additionally, work that was not in the final game is shown and why it did not make it to the final deliverable. It is hoped that this thesis informs the reader of the process that went on during the development of this project whilst also enlightening issues of communication between artists and programmers.

INTRODUCTION

AVA is a game about a character's journey through their past, which will lead them and the player into the Tibetan culture and rituals to unlock the door to Nirvana. This thesis focuses on the development of a group project were the team have decided to build a game in Unreal Engine. The team consists of 5 artists and a single programmer meaning game mechanics were limited to a single person. This game is a 3rd person puzzle walking simulator game which allows the game to focused more on art in the 10 weeks development time rather than the game itself. The thesis will go over the planning and how the game was planned. The initial designs of UI so that the artists and programmer new what to aim for/produce. Production; what was and wasn't implemented, how, why, and what went wrong. Finally, what it's like working with artists, issues, restrictions and perks that come with it.

PLANNING

The first part of this project was coming together as a team and coming up with initial ideas. The artists underwent a preproduction task of thinking about how they want their game and what they want it to be when it's finished, a selection of different cultures where provided to vote upon on depending on colours, structures and native animals. Once the culture had been chosen the team then went their separate ways in trying to come up with mechanic ideas and how the story might play out. The author's initial design for the game mechanics and how they might work can be found in External Document E (Please refer to ***External Document E – Initial Thoughts and Worries***). Within this document can also be found the initial culture votes, worries that where found with the teams' chemistry and of course the story and mechanics and how they might entwine. The initial design used the lunar cycle and the Tibetan culture research (Provided by one of the artists) to merge the culture, mechanics, and story into one as well as using key items within the culture to come up with some game mechanics and puzzles.

The following part of the planning was to get the artists to complete a GDD (Please refer to ***External Document A – GDD***). This was surprisingly difficult to get them to complete as it was their first time making and designing a game and the process was unfamiliar to them. The GDD is a preproduction document that allows the programmer to understand the functionalities and requirements that the Artists desire and require and get a rough outline of what classes can be implemented. It also works as a focus for the artists to keep on point to the initial design as designs can run astray and take quite large tangents without having a structured document to refer to as reference. This then moves onto the considering of how the mechanics of the game might be developed and how they might work in a theoretical sense. The inspiration for the template of the GDD that was provided to the artists comes from a book by Scott Rogers (2014) called *Level Up*! Which is a structured book on games design and is aimed at more game design than technical design.

From the GDD usually a Technical Design Document (TDD) will be created however in this case the TDD blends in with this thesis. From the GDD the first step is to breakdown the overall game aspects, then gather the desirables from what the artists have said they want in the game. From this it is possible to start to consider how the game will be designed and implemented. What needs to be learned, and what's already known, and what can't be done.

BREAKDOWN

- This is a game about a character called AVA.
- He killed a weasel out of malice.
- The Weasel is now your companion.
- Walking Simulator + Puzzles.
- Puzzles Include:
    - Green Room: Ethics: Praying
    - White: Generosity: Gifting
    - Yellow: Patience: Waiting/Slow Walking
    - Blue: Diligence: Helping
    - Red: Renunciation: giving up an item
    - Black: Wisdom: Solve the library
- Rich Environment
- Interior & Exterior Environments

DESIRABLES GAINED FROM GDD

- 3rd Person Game
- Game World
    - Simple interactable objects
    - Models
    - Textures
    - External Environment
    - Interior Environment
    - Moving Grass
    - Deformed Grass
    - Trees blowing
- Movement to include:
    - Multi Speed movement (Walk, Run, Sprint)
    - Turn Around
    - Jumping
    - Push/Pull
    - Climbing
- Inventory System
    - Collection of Items
    - Discardable
    - Collectables
    - Intractable in world
    - Intractable in game.
- Puzzles
    - Praying
    - Puzzle Completion
    - “Peaceful” particle effect while praying.
    - 4 Piece Puzzle set up
    - Gifting puzzle (apply gift to god)
    - Throw item down whole puzzle
    - Stay away from throne. (Check)
    - Apply scrolls to a match symbol puzzle
- UI
    - Main Menu
    - In Game
    - Controls
    - Inventory System
- Cut Scenes
    - Puzzle Completion
    - Opening Scene
    - Closing Scene
- Inverse Kinematics (IK)
- Highlight of objects.
- Volumetric Lights
- Audio & Sound Effects
- Antelope AI
- Weasel AI
- Lighting

As is apparent the list is quite a substantial size for a game development of 10 weeks and it’s highly likely that fair amount of it will have to be cut from the project to be able to provide a reasonably polished deliverable. An artist’s mind can be quite ambitious without understanding the implications to gameplay and programming requirements for different aspects. Luckily this project had a team of reasonable artists who understood that some things would have to be compromised whilst others will have to be dropped completely.
After this breakdown was completed a full task list was constructed, continuing from that an ABC plan was put into place (Please refer to ***External Document B – ABC Plan***). Plan A was kept to the bare minimum and to as minimal of a deliverable as possible as through previous group projects it was seen that a project can have a very quick turnaround into failing and having no deliverable at all. Plan B was kept to a more desired output that the team would be happy to hand in as a polished deliverable this was the “Target” for the final hand in. Plan C was a highly desired game that if time permitted could implement certain technical and artistic advances.

From the ABC plan a Gantt chart was produced, sadly this was mainly used for tracking days, a more detailed task list and where the team was regarding the tasks, it wasn't used as an actual scheduler. The reasons for this was because of the sporadic working times of some team members as well as changes in management throughout the process. (Please refer to ***External Document C – Gantt chart***)
A final document was put into place to track the work of the team members this was the Production Diary (Please Refer to ***External Document D – Production Diary (Post)***). This helped keep track of team members and to make sure that all the team was working appropriately as they should. It was also helpful with days off and estimating ETA's on work based on time taken to do a job.

GitHub

The project initially used GitHub as our choice of source control, however due to the following reasons:

- Right limitations on the university computers
- issues with file sizes going over 100MB
- Only being able to use Git Desktop
- The gitignore file not doing its job properly

As the project went on sadly the source control had to be manual for the project. However, GitHub was still used as a source control for all the coding. A back up was stored on a separate external HD as well as other backups of various files. An eventuality of having to merge all files across the group had to be done and took a few days which could have been removed if git had been installed on the windows machines at university.
The repo for this project can be found here:
https://github.com/Driven-Mad/MastersProject

INITIAL DESIGNS

UI Design

IN GAME UI

A blocked-out example of the in-game UI was provided for the artists so that they could understand what they were aiming for, keeping in mind the simplistic design and considering the programming restrictions, so they wouldn't go overboard with their design. Based on this, feedback could be given from both ends and a final design could be made keeping both technical and artistic parties happy. The initial game UI had very simple and very minimal design to keep the screen as clear as possible. The design wanted to use the 6 virtues that are assigned to each room as a notification to the player to let them know that the puzzle was complete. Each time a puzzle was complete a flower petal would fill with the color of the room. In figure 1 all images are filled in for simplicity. This kept the UI concise and with the theme of the game.

Figure 1 In Game UI (Initial)

INVENTORY UI

A blocked-out example of the Inventory system (Figure 2) had to be provided to the artists allowing them to understand how the inventory system would be implemented. Initially the team wanted to have two tabs, one for collectables to allow for a more in-depth story to be told through them, whilst the second tab was for items which are found around the palace for different puzzle solutions.

The team wanted items to be picked up, a description of the item to be shown and for the items to be discarded or used. The description would just be a simple sentence and with time would turn into an object viewer and give a greater description or play a short video with some history to the object.

The initial thoughts on the implementation of the inventory system was to have the player hold an array of "Items" and push/pop a list of widgets that held the sprites to the UI slots.

Figure 2 Inventory UI (Initial)

MAIN MENU UI

The main menu also had to be blocked-out (Figure 3) for the artists so they had a good understanding of what was needed. The initial design had a beautiful image as the background and three simple buttons; play, controls and quit. The artists would have to provide the image and the 3 buttons one highlighted, one not. This allowed the indication to the user where their cursor is currently selected. The reason for this is that fonts can be quite a pain within unreal.

Figure 3 - Main Menu (Initial)

The initial controls window that was desired was in Figure 4 where there is a simple sophisticated table which had all possible inputs based the user's desired input method whether that be Keyboard & Mouse, X-Box Controller, or the PS4 controller. This method gave the ability to show the menu to the user without too much change. Which can cause some disorientation to some users when they're switching between multiple menus.

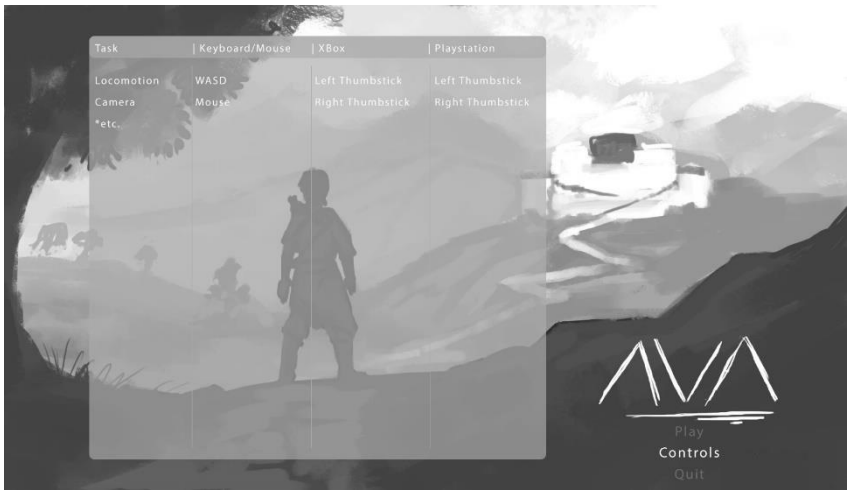


Figure 4 - Instructions (Initial)

Control set up.

The controls were prechosen based on previous games and how the team saw games were played from personal experiences and based on the mechanics that were going to be implement. The full key mapping can be seen in both Figure 5 and Table 1. This controller setup was tested to see what felt more comfortable during gameplay when doing multiple interactions.

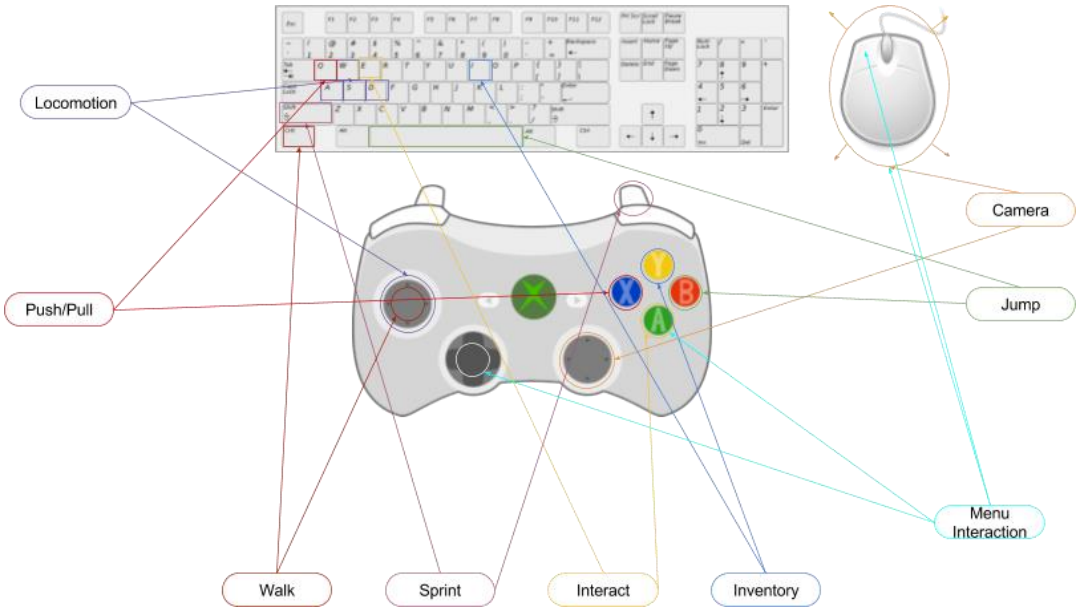


Figure 5 - Control Setup

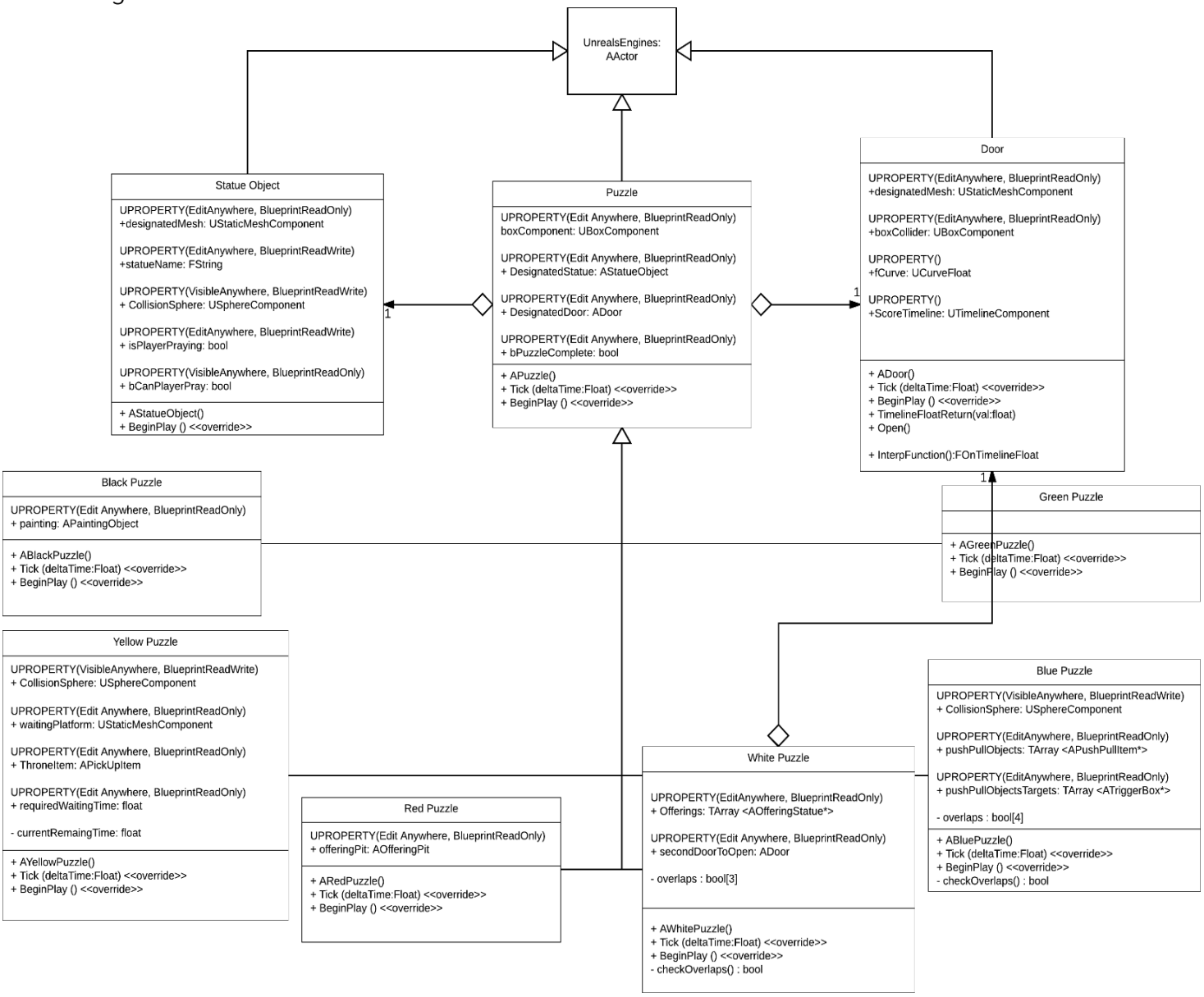
Task	Keyboard/Mouse	Xbox	PS4
Locomotion	WASD	Left Thumbstick	Left Thumbstick
Camera	Mouse	Right Thumbstick	Right Thumbstick
Jump	Spacebar	B	Circle
Push/Pull	Q	X	Square
Interact	E	A	Cross
Open Inventory	i	Y	Triangle
Sprint	LShift	Right Trigger	R2
Walk	Ctrl	Left Thumbstick (Light)	Left Thumbstick (Light)

Table 1 - Control Setup

PRODUCTION

Puzzles

Class Diagram



This is the general structure for the puzzles. The puzzle class itself is abstract and must be instantiated, however unreal doesn't support "Real" abstract classes that inherit anything from the engine. UE4 does allow the change of the UCLASS() to UCLASS(abstract) which will make the class unable to be instantiated as itself and must have children classes to be instantiated (Davidson 2014).

The parent class "Puzzle" is the child of UE4's spawnable class "Actor", this allows the item to be spawned within the game. This helps keep the puzzles in context and updated using UE4's system and makes it easier to get pointers to objects within the scene as reference. A full UML diagram can be seen in ***External Document F - Full Puzzle UML Diagram.*** Each puzzle is represented by a colour and a virtue, this help to narrow down the game mechanic design as well as base the rooms design around the different colours and virtues.

Door Class

Each puzzle requires a door that will open when the puzzle is complete. The door is a mostly a simple class that will run a timeline function that will read in a curve that has been designated by an artist to match their choice of door swing.

First a curve is found and then a timeline is defined. The interp function which is used by the timeline class, declare in the header file, will bind a function to be called that will access the floating point value given from the timeline (See Appendix Code Snippet G). The function that is bound then allows for the value from the timeline to be used. (See Appendix Code Snippet H).
As is visible in figure 6 the curve goes from 0 to -90 over the course of 5 seconds. This value over time will be the doors new yaw value, seen in the above function, to simulate the movement over time.

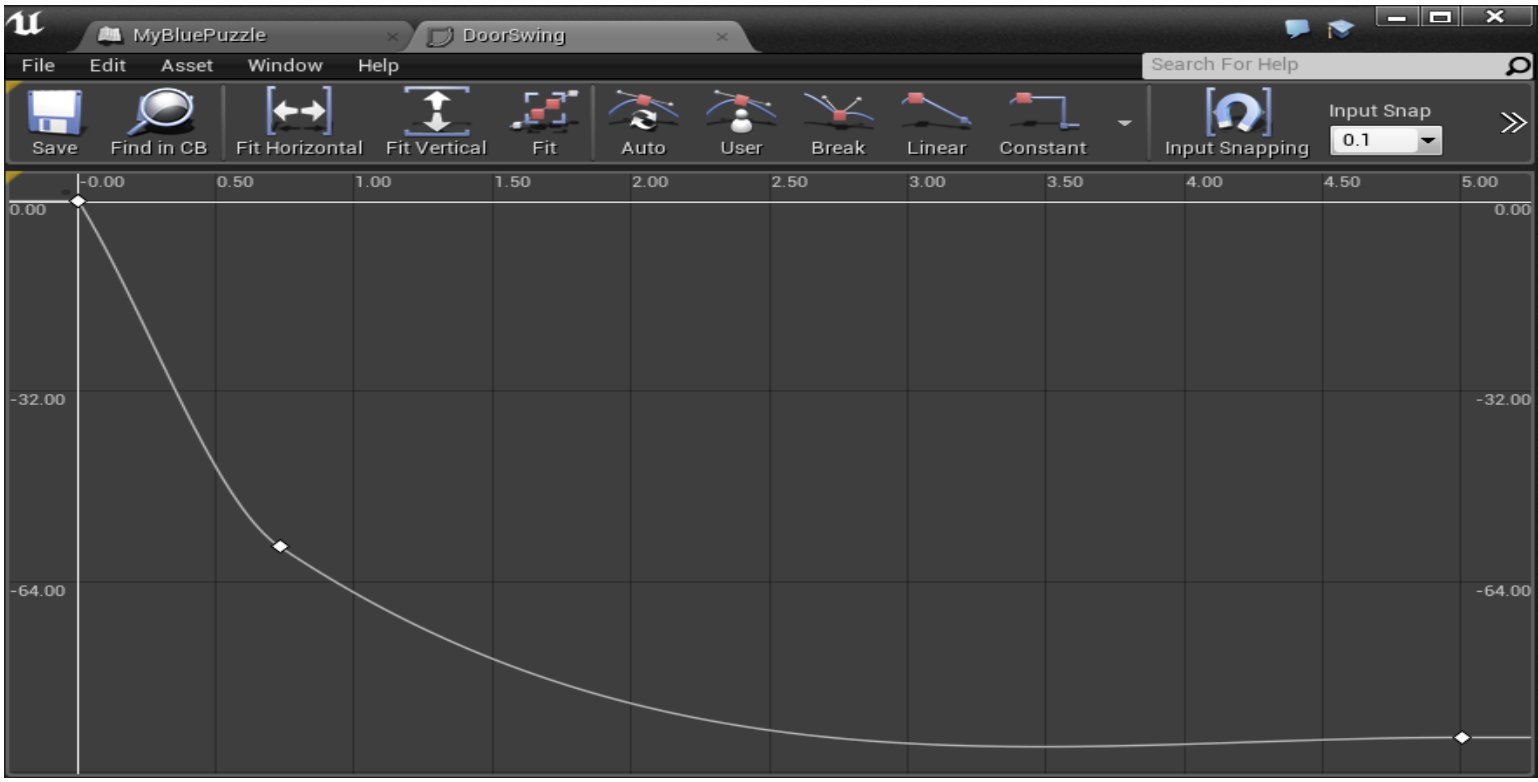


Figure 6 Door Swing Curve

Statue Object Actor Class

The next thing that every puzzle has is a statue which allows the player to pray at and receive some instruction towards the current puzzle that they are in. The instruction is given via a small sequence which will be explained later in this thesis. Each statue has its 3D representation which is the static mesh component "Designated mesh". This allows for the artists to give an appropriate model for each different room. An example of a setup of this actor can be seen in figure 7, where the yellow rooms Tara is placed, and a mat is placed in front for the player to pray if they require the help of the goddess. The functionality for praying at this class is provided in the character class in the interaction function. (See Appendix Code Snippet I)
In this we create an array to store any actors which are overlapping with the characters attached sphere. This sphere fully encapsulates the player allowing triggers to be fired when another actor overlaps with it. The GetOverlappingActors function, provided by unreal, will store all the actors within the array. To proceed, loop through the overlapping actors, and check if one of them is a statue object. If so then the statue variables is player praying and can the player pray at the statue are changed. This makes sure that the player can only pray at any statue once, this helps reduce the length of the game as due to time making these skippable was not available. Watching them


Figure7 - Tara Praying Object

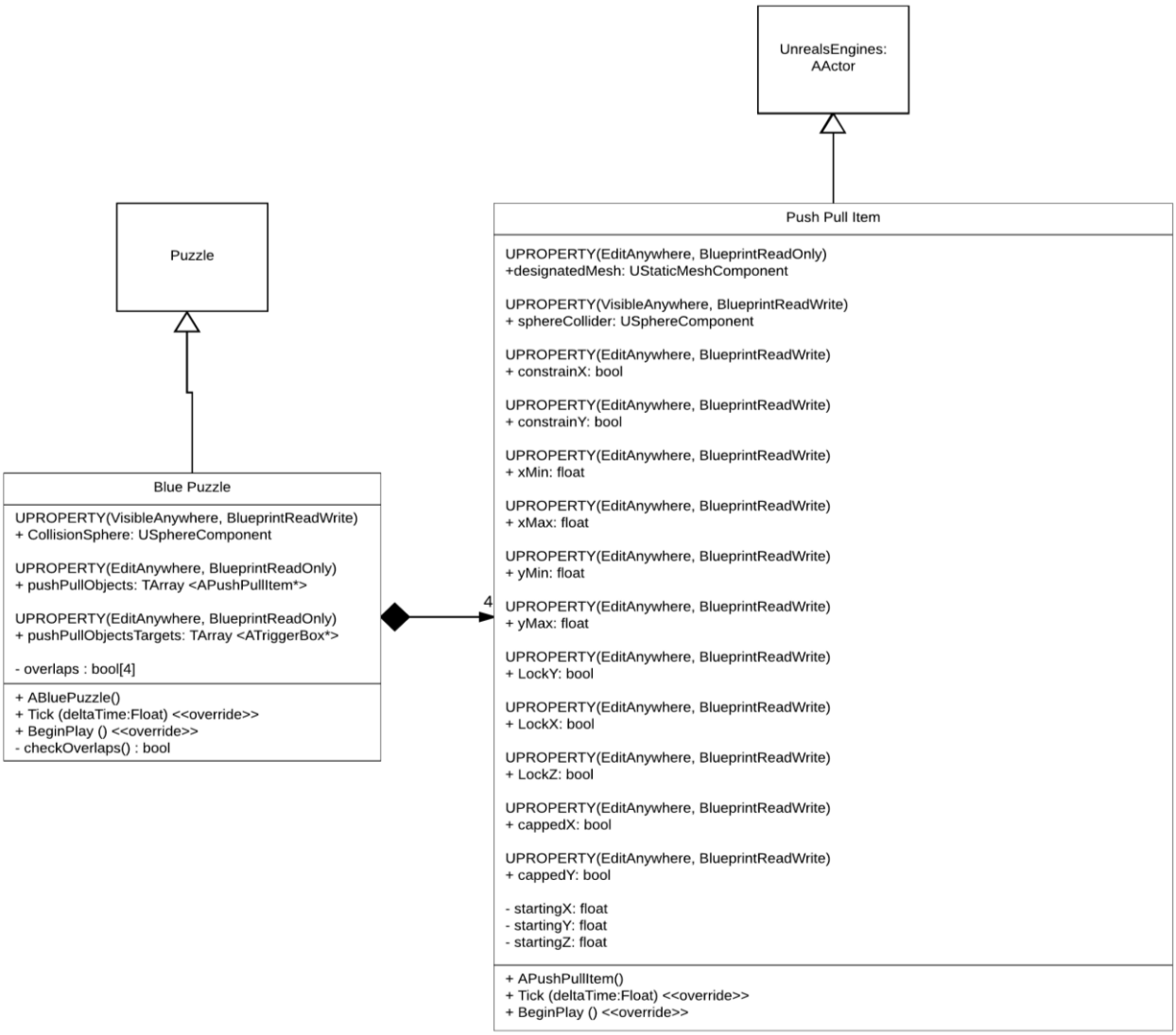
multiple times by accident would increase the time it takes to play substantially. The IsPraying is for the animator within the animation graph so that it can play the appropriate praying animation at the right time.

Green Puzzle

Green puzzle had little functionality so there was not much point in doing a UML diagram for this puzzle as the only thing that was required to complete this puzzle was to have the player pray at the statue to receive hints. It inherits from the puzzle class and only really uses the same functionality as the parent class, except for a slight modification to the tick function. This may be a simple class but it was felt to be one of the more important puzzles as it helps show the player where they can find help. Praying at this statue triggers a small sequence that will show the player the way to the next puzzle and informs them that praying at statues will result in help.

Blue Puzzle

UML Diagram



How the Puzzle Works

This puzzle is the blue room puzzle and is based on diligence. The point is to help the god by putting the lilies in their place (On lily pads). As is visible in figure 8 the yellow dots represent the lilies and the red squares represent the targets (Lily Pads). The player must push the lilies onto the targets to complete the puzzle. To do this the push and pull actors that represent the lilies are pushed onto the lily pads by the player and when it overlaps with a trigger box an overlap event occurs and the puzzle is informed that the Lily is in its place. Pushing all 4 actors onto the targets completes the puzzle and will switch to a small sequence to show the lotus door changing a petals colour to blue. This functionality is based in the Blue Puzzle class update function. (See Appendix Code Snippet J)

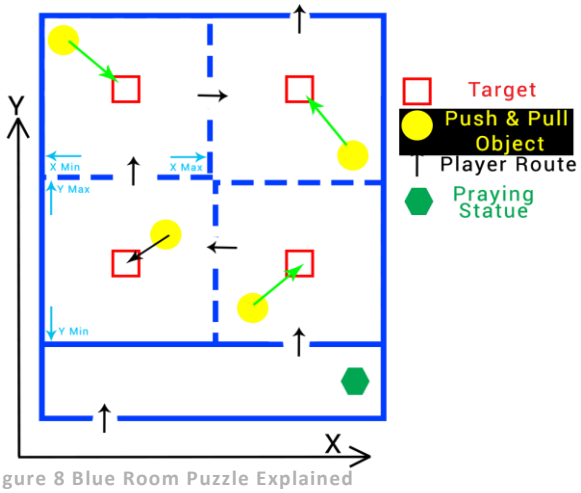


Figure 8 Blue Room Puzzle Explained

From this, much like the praying functionality, each of the trigger boxes stored in the blue puzzle class are checked through a for loop and checks if the push pull objects that are also stored within the class are overlapping with it. If the overlap is true, a stored array of Booleans is set to the same index. Once all overlaps are true, the puzzle is complete.

Issues That Arose

The blue puzzle was the second of the implementations of the puzzles and was where the first of many issues with this project came into effect. When an Actor (A) is attached to another Actor (B), the collisions of Actor B are not passed to Actor A and so provides the first issue: Clipping meshes.

Push and Pull Functionality

The push and pull class is inherited from the Actor class to give it more functionality that is required within this class. This class inherits from the actor class so that the ability to use UE4's AActor::AttachToActor and AActor::DetachFromActor functions is available to be used. These are used to accomplish a relatively smooth transition in attachment and to be able to leave them within the scene. (See Appendix Code Snippet K)

When attached and the player pushes an item it goes through other objects and walls. To fix this and accomplish a push and pull like reaction from an object a few solutions where attempted. To solve the clipping issue solutions that were tried included; trying to modify Actor A's collisions to match that of Actor B's, A simple replacement of Actor A's collisions to be B's, and adding a new collision altogether. Sadly, none of these were successful and so a final addition was added which was a simple and fast hard coded box collision. During this attachment, the players walk speed is reduced to make it appear as the player is pushing slower with the added weight. Once this process is complete the StopPushPull function simply reverses it.

Solutions

The class allowed the artist to input hard values of X&Y Minimum and Maximum values which if met would detach the player from the item they have "pushed or pulled", these figures can be seen in Figure 8. Additionally, a lock entirely on a chosen axis was added because the Unreal's lock to axis did not work due to the hard-coded collision box. Finally, the artists helped by modifying the rooms design and layout to avoid having objects within the scene that weren't necessary during the push and pull portion.

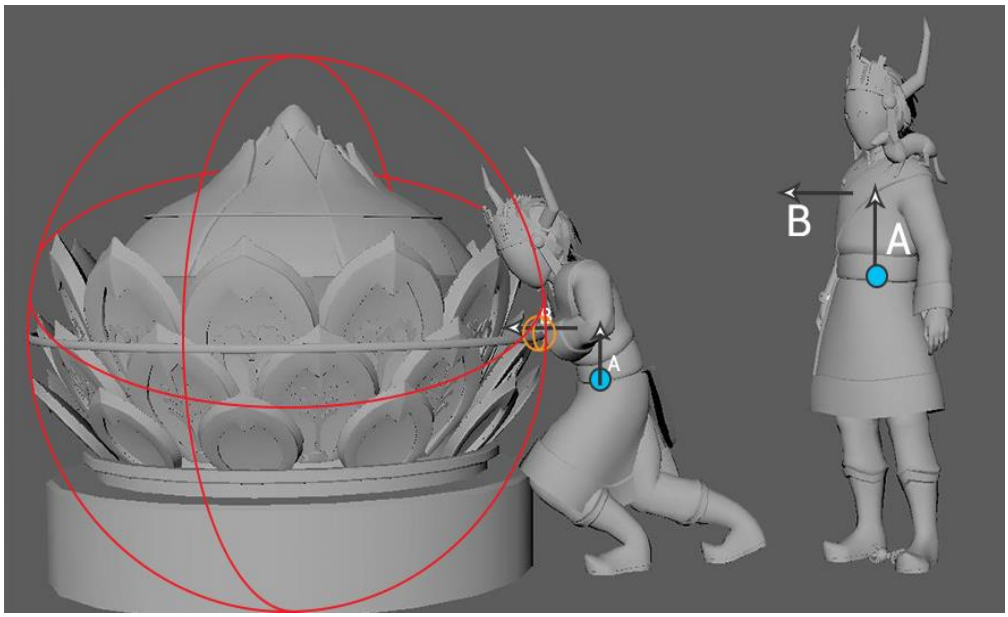


Figure 9 Push Pull Explained

During this process, the character class needed to see if it could interact with a push and pull object. To accomplish this a line trace was cast forward X amount from the player by getting the actors forward vector (FV) and getting the actors location (L). The ray starts at the actor's location (Blue Dot in Figure 9) and an offset A is added, the end of the ray is calculated by L + (FV * Trace Distance B). (See Code Snippet L)

The trace distance allows for the item to be only attached when it is in range giving mostly full control on the attachment process. The reason for the required offset is because the actor's location is set at the centre of the mesh and the artists wanted something to match so that only something that hit the trace line would be able to be pushed/pulled to increase believability and adjust mesh sizes of the push and pull items accordingly. This helped to minimise the attachment process giving odd results such as the player having its back to the object but still attached.

The sphere component of the push and pull actor can be moved to be placed where it is required for the player to attach, this could be the orange circle in figure 9 where it's at a very specific point, or it could be in a more varied placed such as the large red area. This does not mean, for example like IK (Inverse Kinematics) where the hand would attach exactly like a human's hand, this means that the player will "Collide" the trace line with this circle and that would allow them to attach to that object. The push and pull actor is used in the black puzzle and can be used over and over within the game if the game was

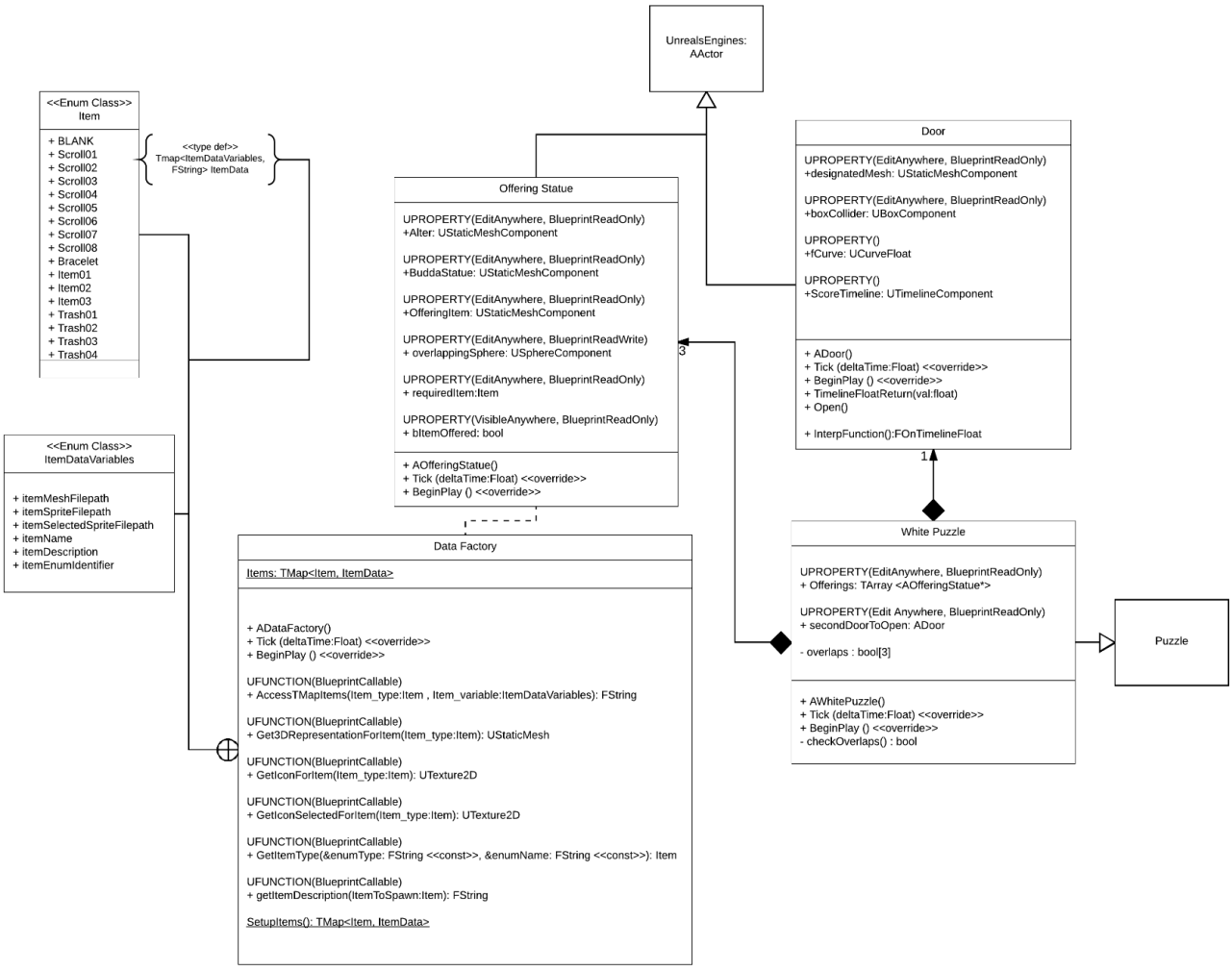
extended to move platforms to jump around on for example. However, this is currently not the case as the development of further puzzles and mechanics would take time.

Current Bugs

Current issues on the bug list regarding the push and pull mechanic is that it allows for the user to approach from the side and still attach at a certain range of angles (roughly 30 degrees' either way). The player can still only go forward and backwards but this makes it look worse and less believable. This could be solved by snapping the user to the object within a certain threshold however this may not be implemented in the deliverable due to shortage of time.

White Puzzle

UML Diagram



How the Puzzle Works

This puzzle is the white room puzzle and is based on generosity. To complete the puzzle the player must generously offer up to the Buddha statues an offering, in this case a mandala which will be represented by our class “Pick Up Item”. As is visible can see in figure 10 the yellow dots represent the three Pick Up Item actors that the player need to pick up and place in their inventory. The red squares represent the Buddha statues which will be the class “Offering Statue”. The player must search the rooms and collect all 3 items and offer them up as a gift to each of the Buddha statues. They do this by entering their inventory, selecting the appropriate item and click “use”. At the end of this puzzle the game will switch to a small sequence to show the lotus door changing a petals colour to white.

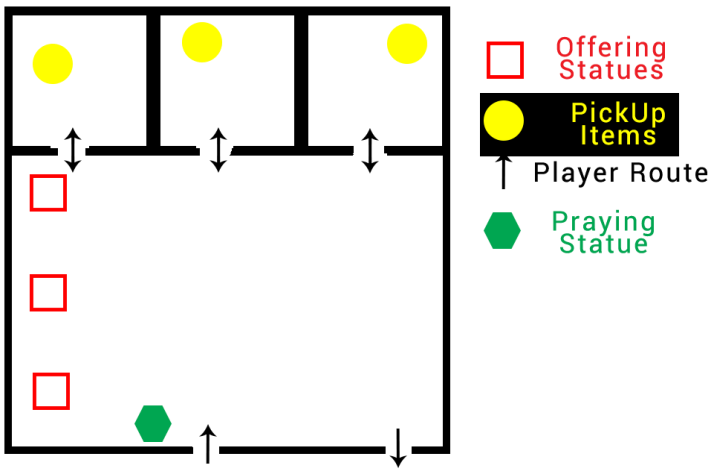


Figure 10 White Room Puzzle Explained

Issues That Arose

There were a few issues that arose when it came to light by implementing this puzzle. The main ones being on the Offering Statue class the Buddha Statue and Alter meshes would constantly put their relative locations to world locations which

would remove them from their proper places. Acknowledgement that the puzzle was complete had to be achieved so the player new they were progressing. A second door was also required for this puzzle as it lead to the great hall before going into the next room.

Data Factory Class

The data factory class allows the ability to store all the data on various items that are within the scene that go in the inventory system. This includes different file paths to various assets as well as properties required for the game. First, two enum classes were created which were exposed to blueprints via the UENUM (Blueprint Type) allowing the use of these within the blueprint system. Next a type definition map called Item Data was created which will be used to store string data variables based on the enums Item Data Variables.

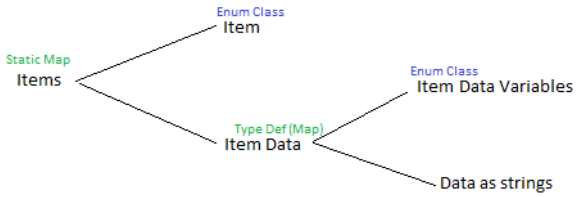


Figure 11 Data Factory Tree

```
typedef TMap<ItemDataVariables, FString> ItemData;
```

Then a map of the enum class Item and the type definition Item Data is created.

```
static TMap<Item, ItemData> Items;
```

This links items to a set of data within the map. This will allow the eventuality of accessing data values via strings and the enums themselves, it also makes it easier to create re-usable functions for all items rather than individual items. To clarify the data flow can be seen in figure 11 and shows the data factory structure better in terms of the maps that are being used.

Note: Inventory & The pick up actor will be explained later in this thesis. An example of an item being set up can be seen in Appendix Code Snippet M.

Figure 12 Offering Statues

Offering Statue

Each offering statue requires a single item to be offered to it. Once the appropriate Pick Up Item actor overlaps with the overlapping sphere, much like how overlapping actors are found in previous functionality shown, the item is destroyed and the Offering Item static mesh becomes the 3D representation mesh of the Pick Up Item actor. As the item is offered a Boolean is also triggered to inform the puzzle that the item has been offered successfully. This is calculated in a similar way to the blue puzzle with overlaps being the triggering notify of events. (See Appendix Code Snippet N)

Solutions

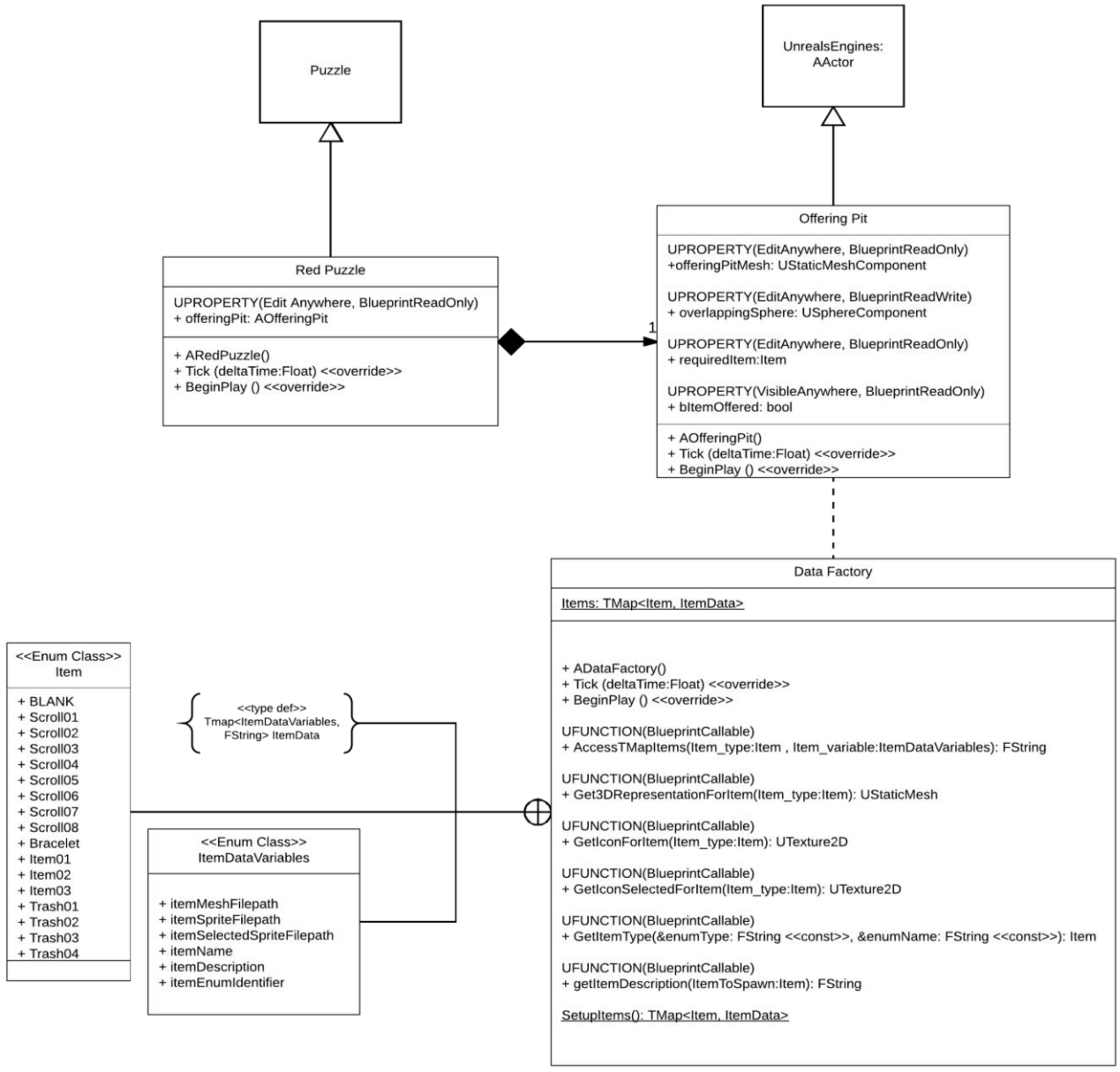
Regarding the two meshes Currently setting them as attached to the Root Component works for now, however it's rather random and cannot be guaranteed as a fixed as this an unreal bug and requires rebuilding of the class to 100% fix which time is not allowing. To solve the acknowledgment issue once the item has been offered the Buddha statue will hold the item in their hands as acknowledgment that the player has offered the item successfully. As seen in figure 12 the green circles where on the left an item has been offered to the statue and on the right the item has not been offered. The second door was added to the child actor as this was not considered during the designing of the puzzles. Leading to when the puzzle is complete, the door on the right-hand side will open and lead into the great hall, whilst also a second door will open to the red puzzle room across the hall.

Current Bugs

The only known bug for this puzzle is the Buddha statue and alter, which currently seems to be okay however still noting it here in the case that it happens during testing.

Red Puzzle

UML Diagram



How the Puzzle Works

This puzzle is the red room puzzle and is based on the virtue of renunciation. The player must offer up a precious item that the character AVA holds dear to their heart to show the willingness renouncing worldly possessions. From the beginning of the game the character holds in their inventory a bracelet that Ava has had since they were born. The player must go to the offering pit and offer up the bracelet which will drop down the pit. A short cinematic will play which shows Ava taking off the bracelet and dropping it into a pit. Upon offering this item the player has completed the puzzle and the lotus door sequence will play and a petal will change to red. A layout of the red room can be seen in figure 13. The bracelet will no longer be on the player's wrist nor will it be in the inventory.

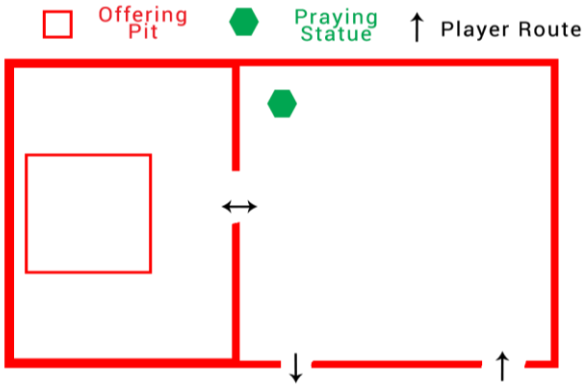


Figure 13 Red Room Explained

Issues That Arose

This was the first time a full cinematic would be played however the solution to that will be explained further on in the thesis. The biggest issue was adding to the realism of the game regarding what would happen to the bracelet once the cutscene had played and how the bracelet would work in general. Another issue that arose was that the bracelet could be dropped anywhere within the palace and it is required to be able to complete this puzzle.

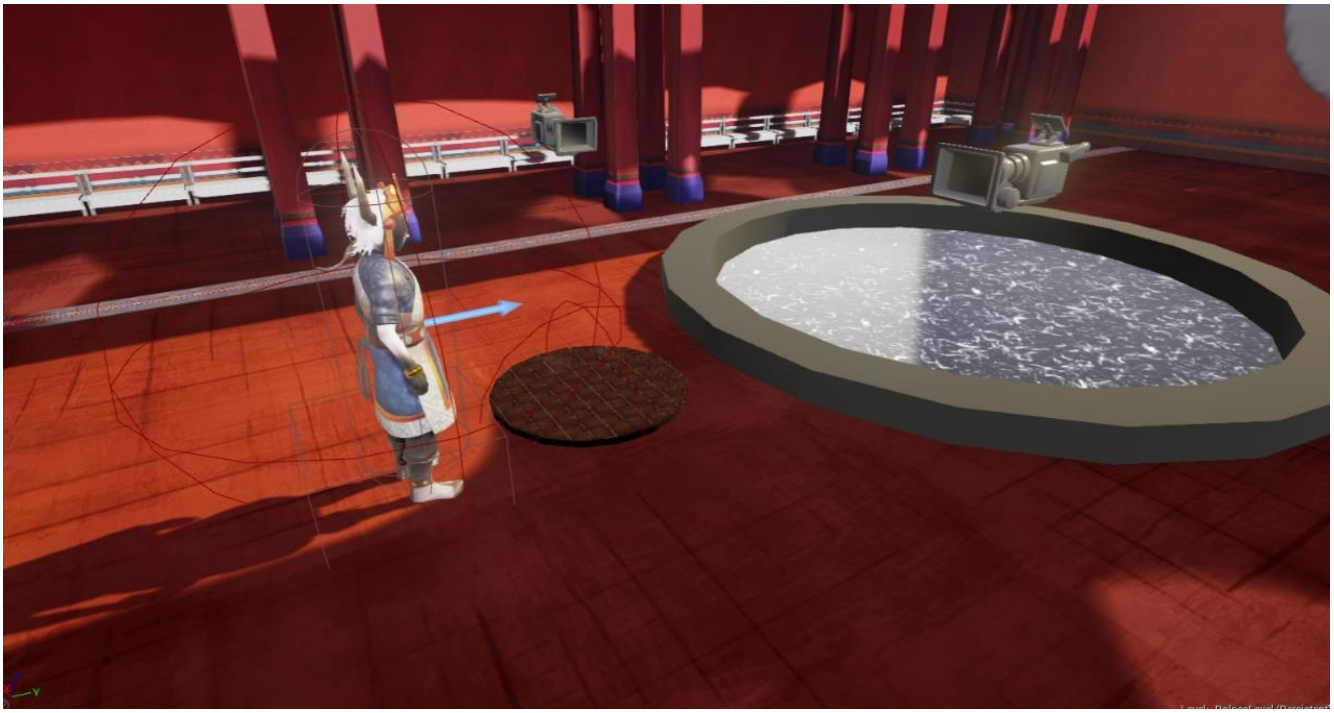


Figure 14 Red puzzle pit

Offering Pit

The offering pit works very much like the offering statues, an item is required, specifically the bracelet, and must be offered up to the Offering Pit actor to sate the puzzle. This class could have been avoided completely and the two merged however this class was a quick addition at the time as ease was needed over efficiency. Once the bracelet overlaps with the overlapping sphere then the item is destroyed in game, and the cut scene is played. As seen in figure 14 the pit is set up with the giant cylindrical mesh being the pit and the small mat indicates where the player should stand to offer up the item.

Solutions

After a long discussion with one of the artists it was decided that the bracelet would be an item that was attached to the rig and they would animate it within the animation itself, once the cut scene complete the material of the bracelet would be replaced with a transparent one. This kept the solution rather simple and saved messing around with sockets attached to the rig within unreal engine.

To solve the issue with the dropping the bracelet anywhere, the plan is to have it much like within Pokémon when a player tries to use the bicycle indoors and professor oak says that it is not possible to use the item here. The plan is to do much the same however have the Tara say, “This item is precious to you, you cannot just throw it away like its trash”. This solution has not been implemented yet however should be within the final deliverable.

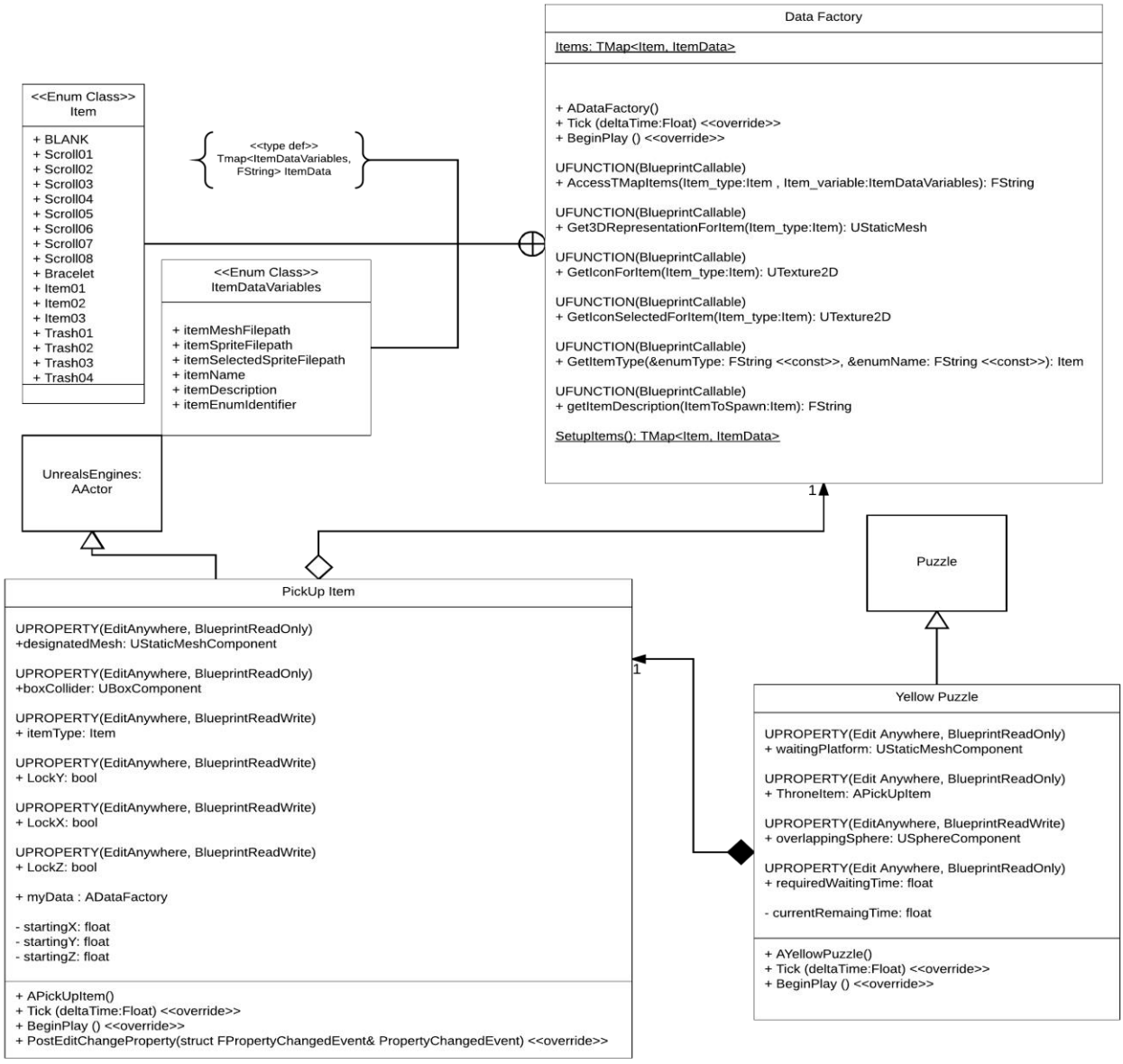
Current Bugs

Current known bugs with this is that the collision circle for the offering is too small and that the player sometimes just drops the item to the floor. They can pick it up again and try again, however this should work first time, simply increasing the size of the collision sphere should fix this.

A further bug includes the cloth simulation where the sequence snaps the player into place the cloth sim is then broken randomly and clips in the arm. The plan is to change this so that the sequence doesn’t control the player and instead to switch to the puzzle itself having it manage the player, that way the player can be lerped between the two positions rather than snapped. However, if time does not allow the bug has been recorded here.

Yellow Puzzle

UML Diagram



How the Puzzle Works

This puzzle is the yellow room puzzle and is based on the virtue of patience. The target for getting this puzzle complete is to get a pick up item actor from the throne, but when the player gets close to it, the item disappears. The player must wait in one place for a total of 10 seconds and upon waiting for the 10 seconds a cut scene is triggered where the weasel gets the item for the player. The point to this puzzle is to show that patience is a virtue and that with patience reaps rewards. A layout of the room can be seen in figure 15.

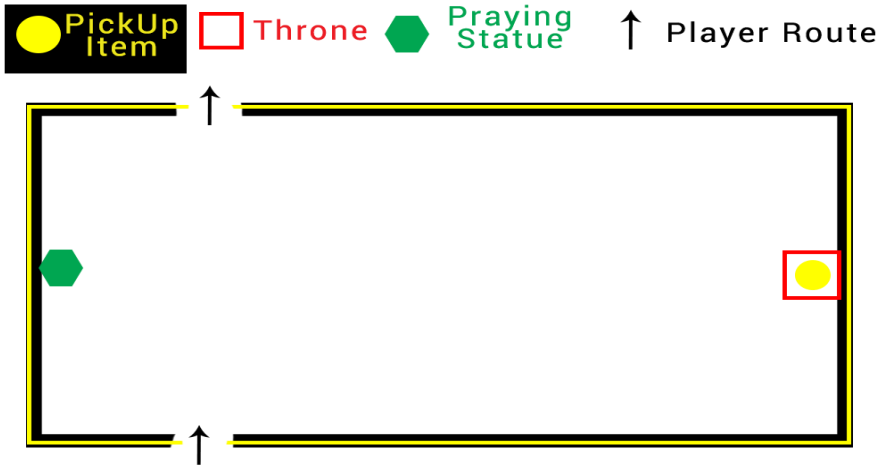


Figure 15 Yellow Room Explained

This puzzle uses a timer, and once the player has been overlapping with the correct sphere for the right amount of time the puzzle is complete. The throne item has to be destroyed, so that it is removed from the world, and so that the rigged one can be replaced with it. The rigged one has an invisible texture till this point. Upon completing this the rigged throne item's material is changed to the right one, set to invisible at the end, and then finally the item added to the inventory. (See Appendix Code Snippet O)

Issues That Arose

The main things that arose within this cut scene is that the player had to wait in a specific spot for a certain amount of time and it was not clear how to accomplish this.

Pick Up Item Actor

The Pick Up actor is a class that inherits from the actor class and uses the data factory to adjust all of its variables. This actor can be spawned and manipulated by anything else within the game however is used within the inventory system that is attached to the UI blueprint. Items can be added to the game and simply change the item time and the variables will change on the fly. When the player interacts bIsPickingUp is set to true which is used for the animator if they wanted to play an animation of picking up. bPlaySoundPickup is a notify to allow for a sound to be played when they pick up an item. (See Appendix Code Snippet P

Figure 16 Yellow Room Waiting

Solutions

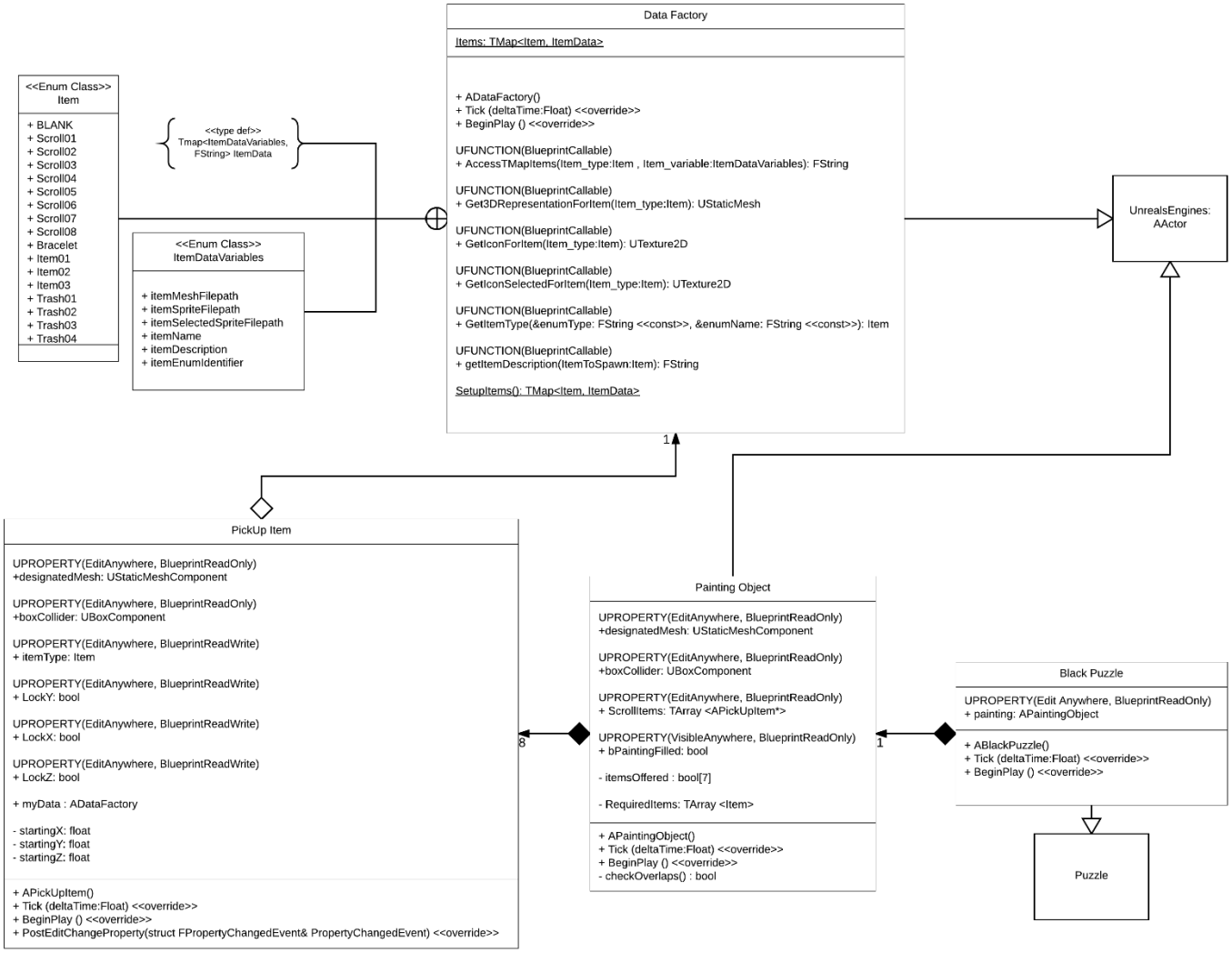
As seen in figure 16 the solution to having the player wait in a single spot was to indicate it with a mat for the user to stand on. The Tara gives hints to how to complete the puzzle also and through user testing most the players understood how to complete this.

Current Bugs

The player is currently able to pick up the item from the throne even as invisible, the solution to this is to simply move the collision sphere away from the item and out of the players reach. The reason this is not simply a static mesh is because it interacts with the inventory.

Black Puzzle

UML Diagram



How the Puzzle Works

This puzzle is the black room puzzle and is based on the virtue of wisdom. The target for getting this puzzle complete is to gather from around the library 7 of 8 pickup items, which are based on the auspicious symbols of the Tibetan culture, and place them in the painting at the back of the room. The 8th is already in the painting to show the player what they are doing. Once all items have been placed on the painting object then the puzzle will be complete. The Final petal will turn black on the lotus door, and Tara will explain to the player that having completed all the virtues it's finally time to go to Nirvana through the lotus door.

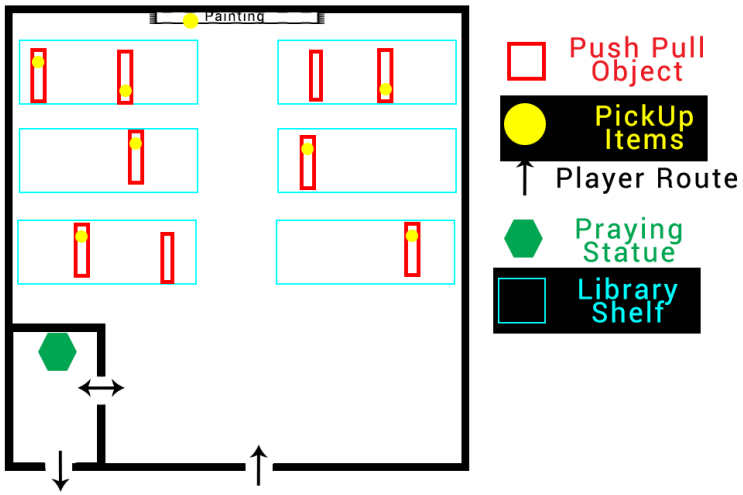


Figure 17 Black Room Puzzle Layout

Issues That Arose

Issues that arose with this puzzle was in the push and pull objects and how because of the hands not touching the handle, how unrealistic it was. The implementation of the painting object, and having 8 pick up items attached to it was also an issue at the beginning.

Painting Object Class

The painting object allows the placing of 8 items in the object like the way that the offering objects are collected. Once all items have been offered the painting object is complete and the puzzle is done. The object holds 8 pick up actors, which are all Item::Blank initially. Upon using the item on the painting, the items are swapped out and will appear as a part of the painting. A finished "painting" can be seen in figure 18. This is obviously just a stand in to show the point of the puzzle, artists will work on an actual mural eventually.

Figure 18 Ava completing the puzzle

Solutions

To solve the issue of the push and pull items, the draws were redesigned to match the animation. As in figure 19 the old draws can be seen in the background, and the new one attached to Ava is the redesigned version. This was so that the items matched to the animation as closely as possible. Originally the pickup items in this room had a lot of issues with being attached to the push and pull objects as well as physics on them. To solve this, physics were removed from all pick up items, and added to the class was the ability to block on certain axis' to stop the item from moving about, this has been previously explained in detail.

Figure 19 Ava pushing and pulling shelf in library

Current Bugs

Current bugs for this room is that the player can attach from a weird angel.
Another is that the items can be picked up from the outside. This will be solved by reducing the size of the pick up items collision boxes.

Last Notes

This puzzle is a great example of the re-usability of the pick up items, the inventory system and the push and pull objects. It also shows that these can be combined and used in various ways to create a wide range of puzzles if the game was to be expanded. Other puzzles could be developed in a similar way to this by re-using a lot of the classes already implemented to expand and broaden this game. An example could be to have push and pull crates to be moved to jump onto to get to higher places where items could be waiting to be offered to a different deity within the Tibetan culture.

Sequences

Sequencer Explained

Sequencer Editor is Unreal's inbuilt cinematic creator to create in game cinematics, it is brand new as of version 4.12 replacing their original matinee cinematic maker from UDK. The sequencer is a multi-track editor that works with cameras called Cine Cameras, and cuts between them via the user's choice in track. The cameras are animated using key framed animation within the level and upon each of their own tracks. Other assets such as actors, sounds and animations can also be played by adding tracks through the sequencer. (Epic Games, 2017). When a sequence is created the sequence itself remains within the scene as an actor, it is possible to create versions of this within blueprints, however having a pointer to the actor within the scene stops a lot of issues that can confuse the engine and throw out bugs. With the help of Bourrelly (2016) this was a successful implementation as the tutorial provided by him clearly explains how to set up a simple sequence and use the sequencer effectively.

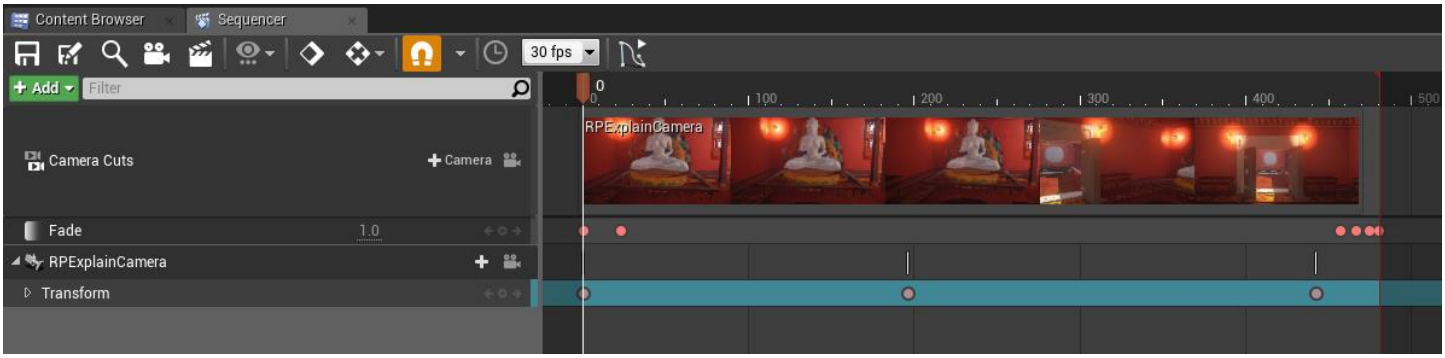


Figure 20 Tara Sequence

Our Sequences

In figure 20 an example of the sequencer for one of the Tara's, it has a single camera and the camera has had its transform key framed. It uses curves to transition between the two frames creating a smooth transition between the two. A great added feature is the addition of the fade to black track. This allows the ability to fade our cinematics to black to hide unwanted arte-facts or returning to the main camera.

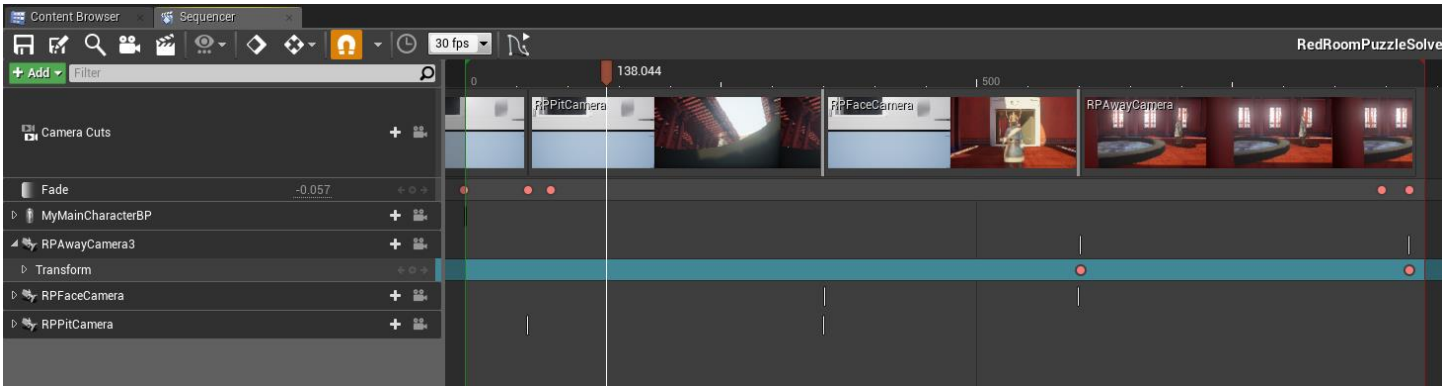


Figure 21 Cinematic Sequence

In figure 21 an example of the sequencer for the cut scene, there is three different cameras which have been cut together in the camera cuts track. This allows a smooth transition between cuts rather than a snapping of the camera when moving it. In this scenario the player has been added however due to a bug with cloth sim this will be removed and the character will be moved by other means.

One issue that was found with this is that if the camera cuts goes all the way to the last frame then the camera can get stuck and will not return to the game, cutting the sequence a few frames short with a fade to black fixed this. This was found after reading about a different bug by Cinebeast (2016) with issues playing animations from the character.

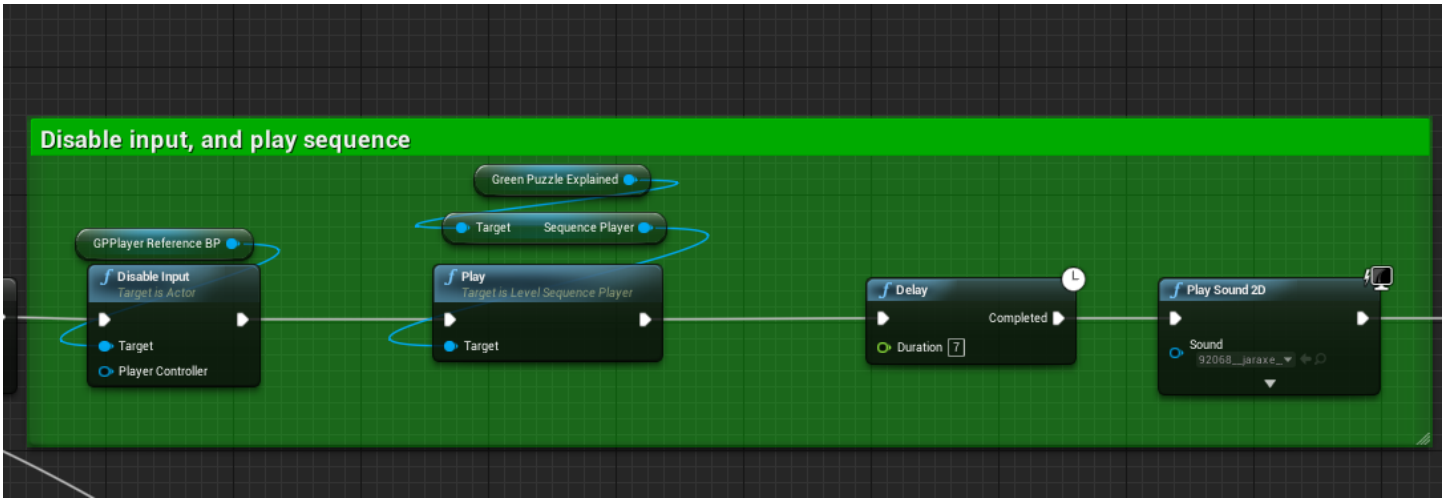


Figure 22 Playing the Tara Sequence

In figure 22 a sequence is played via a reference to a sequence actor within the scene; in this case the green puzzle explained sequence. From that the sequence player is accessed and simply calling the play function from the class plays the sequence. Now because of the way the sequencer works the player still has user input so this must be disabled via the blueprint before the sequence starts playing.

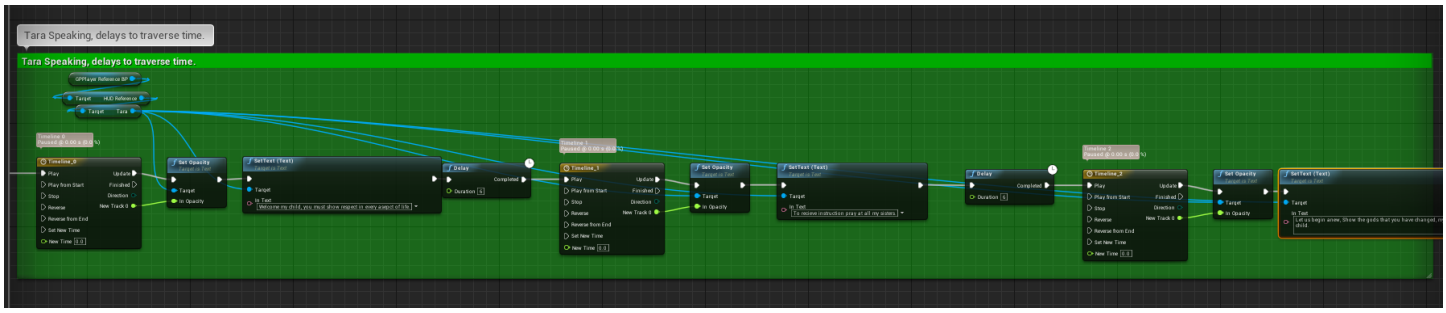


Figure 23 Tara speaking to the player.

As the Tara speaks to the player delays, timelines and sounds needed to be matched with that of the artists requirements for camera cuts. A delay is added in figure 22 to match the time of when the Tara starts speaking through the UI. Timelines are added to add fades in opacity to each of the new sets of texts so that it's not so harsh to the user in switching between sentences of Tara. These can all be seen in detail as apart of figure 23. Delays are also added in between the sentences again to match the camera movement of the sequence.

Unfortunately, in game cinematics that use something like the character class can't have full animations played through this without getting stuck at the end or looping over and over (Cinebeast, 2016). This has been resolved in newer versions as stated by developers in the Cinebeast post (2016), however this project couldn't switch version due to software limitations. This meant that the characters animation had to be animated within Maya, imported into UE4 and then played via a montage through the various puzzles once they had been completed. In figure 24 delays and timelines had to be added to match the animation of the cameras in the sequencer to the animation of the character. The montage and sequence had to only be played once and so Booleans and conditions were met to make sure this happened.

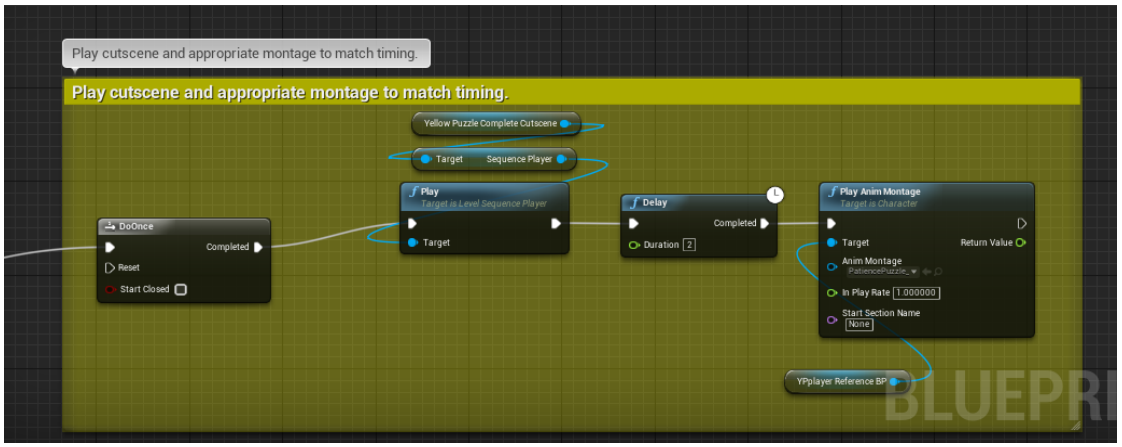


Figure 24 Playing a montage.

Lotus Door

The ending of each of the puzzles is to play a sequence of the lotus door (Figure 25). It's a simple shot of the final door that the player is aiming to go through. As the user completes each puzzle the door will add a new petal to the lotus at the top to match the colour of the room they have just completed. Once the puzzle is complete any text from the UI is cleared and the users input is re-enabled. This sequence is about 5 seconds long, but shows the players that they are progressing.

This final lotus door holds a reference to all the puzzles within the scene. As the game progresses and the player completes each of the puzzles the appropriate element in the array of materials of the door is changed to the appropriate material to match the colour of the completed puzzle. Delays are added to counter act the fades and if another cinematic must be played first. For example, the red room has a delay of 32 seconds due to the cinematic that is being played before this one (Figure 26).

Once all the puzzles were complete a final Boolean is set to let the door know that the player can now progress to the next stage of the game. A trigger box is placed in front of the door and when the player overlaps with it the final cut scene is triggered. Much like the rest montages, delays and such are all matched to the camera movements. Along with this a timeline is added to the doors so that they open at an appropriate time within the cut scene (Figure 27).

Figure 26 Lotus Door, updating materials

Once this cut scene has finished the game will be ended and currently returns to the main menu, it is hoped that credits will be added to this before the game is delivered.

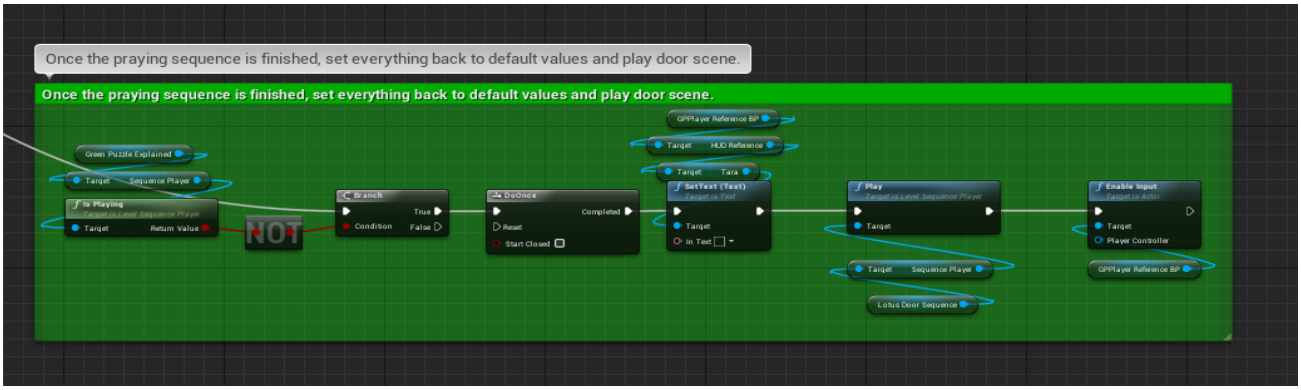


Figure 25 Playing of the lotus door sequence

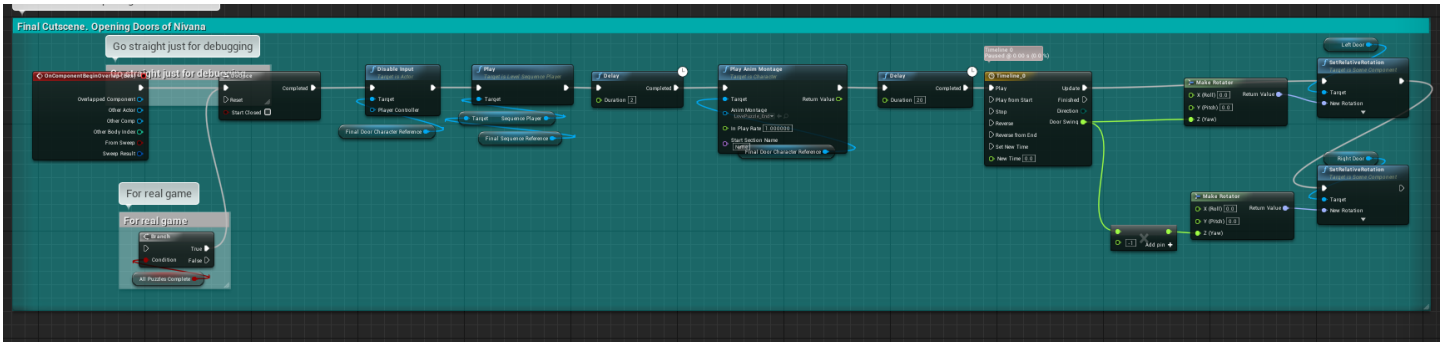


Figure 27 Playing the final Cutscene

Inventory

Spawning an item

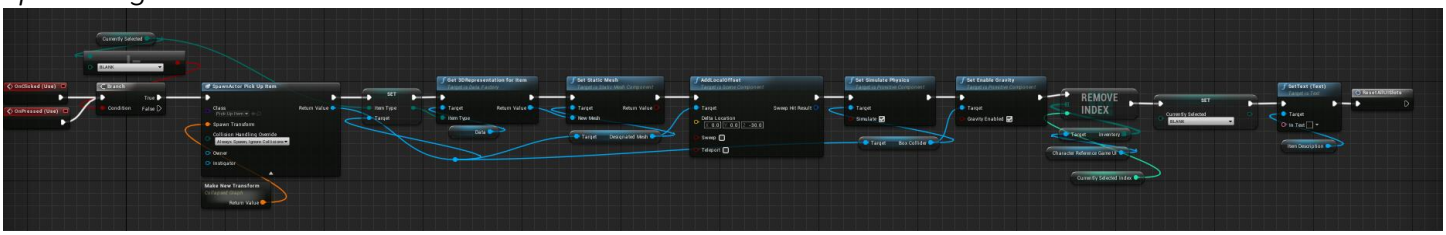


Figure 28 spawning an item from the inventory

To spawn an item, UE4's blueprint node "Spawn Actor" is used. By selecting the class "Pick Up Item" from the drop-down menu the item can be spawned. To do this within the inventory the player will select the item within their inventory and select the use button. A reference to that item name is retrieved and from that, set the item type of the new pick up item that will be spawned. From the item name, various data variables can be accessed via the data factory. The reason the function was not done in C++ was because the Pick Up class includes the data factory, and to forward declare the pickup item could have caused issues down the line. The UI inventory part of the system is all in blueprints so this helped in removing items from the player inventory and the UI. To make sure this remained as realistic as possible, physics & gravity is enabled on the item so that it falls to the ground. The item is then removed from the player's inventory, and the currently selected item is set to blank in the UI and remove the item description to blank. Reset all UI Slots is then called.

Reset All UI Slots

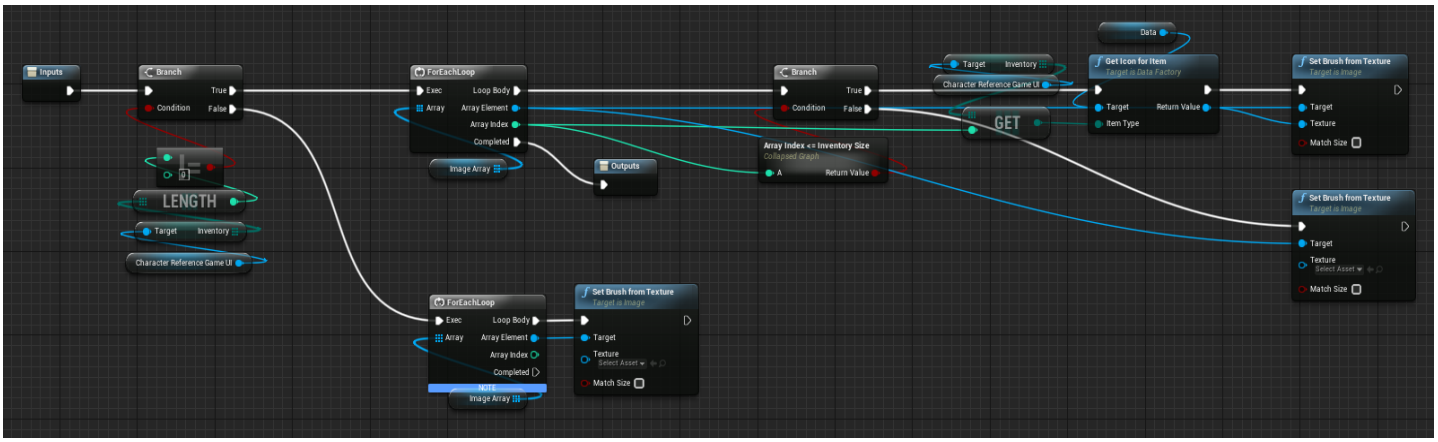


Figure 29 reset all UI Slots function

The reset all UI Slots keeps the UI up to date, and keeps the inventory as clean as possible. It works by checking if the length of the characters inventory exceeds 0, if it doesn't it keeps all the textures blank. However, if it does exceed 0, it does a full loop of the inventory and getting the item gets the icon for the image and applies that as the slots brush. Once it exceeds the inventory size it continues through the slots loop and finishes by setting all the brushes back to blank.

Mouse Input

Each slot within the inventory has its own function (Figure 30) to check if the mouse has been pressed on it. Upon pressing it via the player, all UI slots are reset to make sure they're all up to date and then the slot press function is called passing in the index to the slot that the player selected. Due to an issue were a controller is present, the mouse is not portrayed as an input from any of the controller inputs. How this is going to be fixed is currently unknown as it was presumed that the input would be matched by unreal through the thumb sticks. A quick thought to this would be to simply change between the slots using is key pressed from the character controller, the same issue is within the main menu.

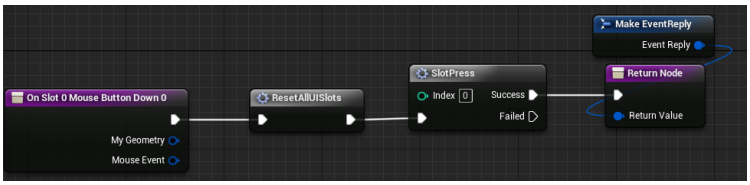


Figure 30 Mouse Button Down

Slot Selections

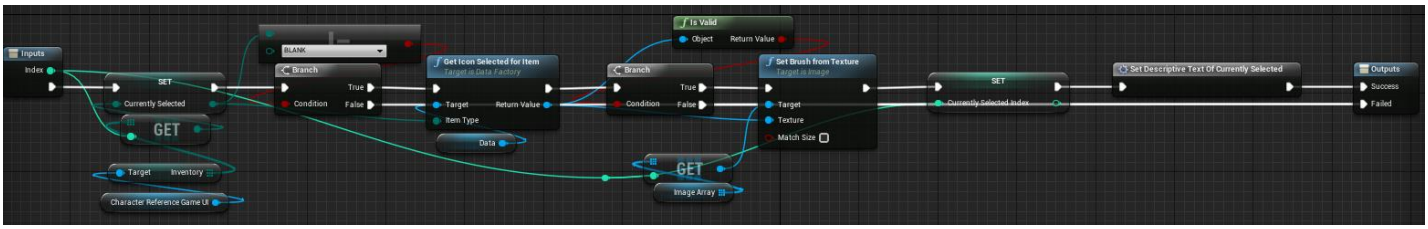


Figure 31 Slot selection

Once the player has selected the slot, the “Currently selected item” is set to be the equivalent to the characters inventory. If it is not blank the hover icon is retrieved and once validated the brush is changed to the new texture. The description of the item is then set on screen once that has been achieved.

Particle Simulations

Torch Light

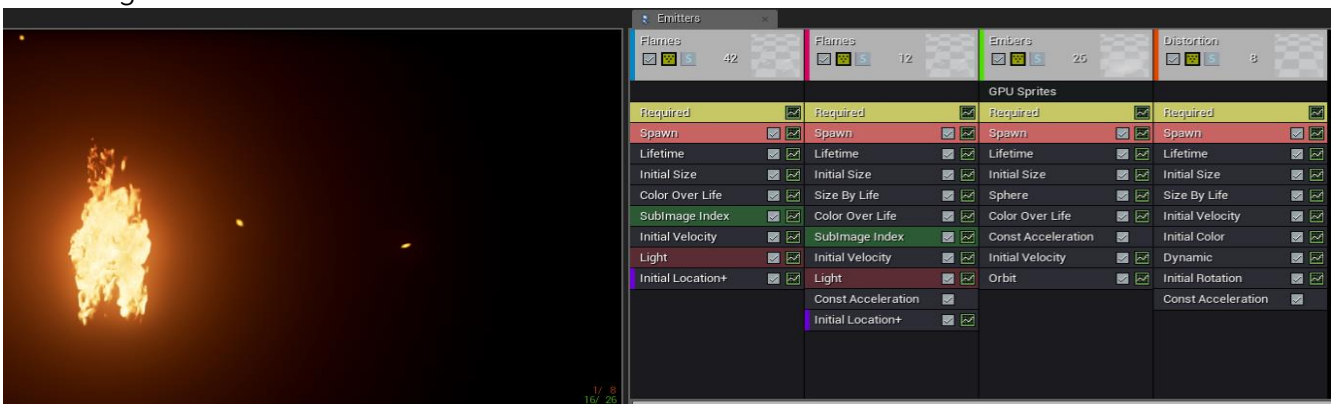


Figure 32 Torch Light Particle System Set up

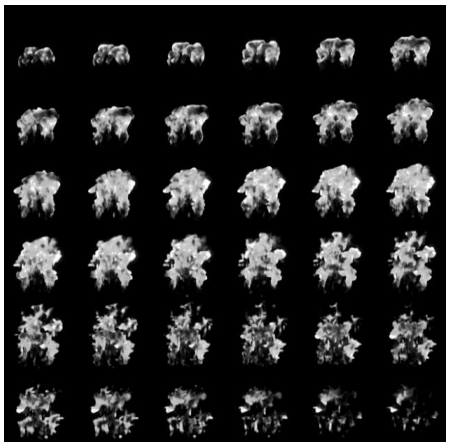


Figure 33 Particle Sprite Sheet

Figure 34 Torch in use

The torch light uses unreals own sprite sheet for fire provided in the starter content. The way the sprite sheet works is by rendering out the fire simulation onto a 6 X 6 sprite sheet (Figure 33). As you can see the sprite sheet goes from starting of the flame all the way to full flame and then to dissipating. The particle system then plays the sprite sheet which simulate the fire on each particle. This particle system has 4 emitters, the first two sets being flames. One for the body of the torch whilst the second being the spattering of fire. The third being the spattering of sparks/embers and the final being a distortion to simulate the heat that comes off the flames. The distortion part is simply a material that has some refraction on it. The particle system being in use can be seen in figure 34 and adds a nice light to the environment and a bit more life.

Pickup Item Indicator

The pickup item indicator particle system (Figures 35 & 36) uses a simple sprite with a random size and a color change over time from yellow to orange. This adds a nice colour change to this so that the its not a bland particle system. The velocity is ranges between 5 to -5 on X & Y axis, and 0 to 5 on the Z. This gives the implication of a random choice of direction and keeps the particles ascending. An orbit node was added so that the particles orbited around. This gave the item a point of focus for the particle to flow around.

Figure 35 particle system in effect

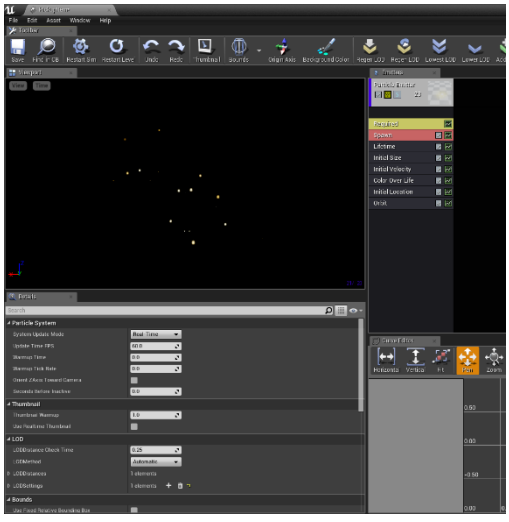
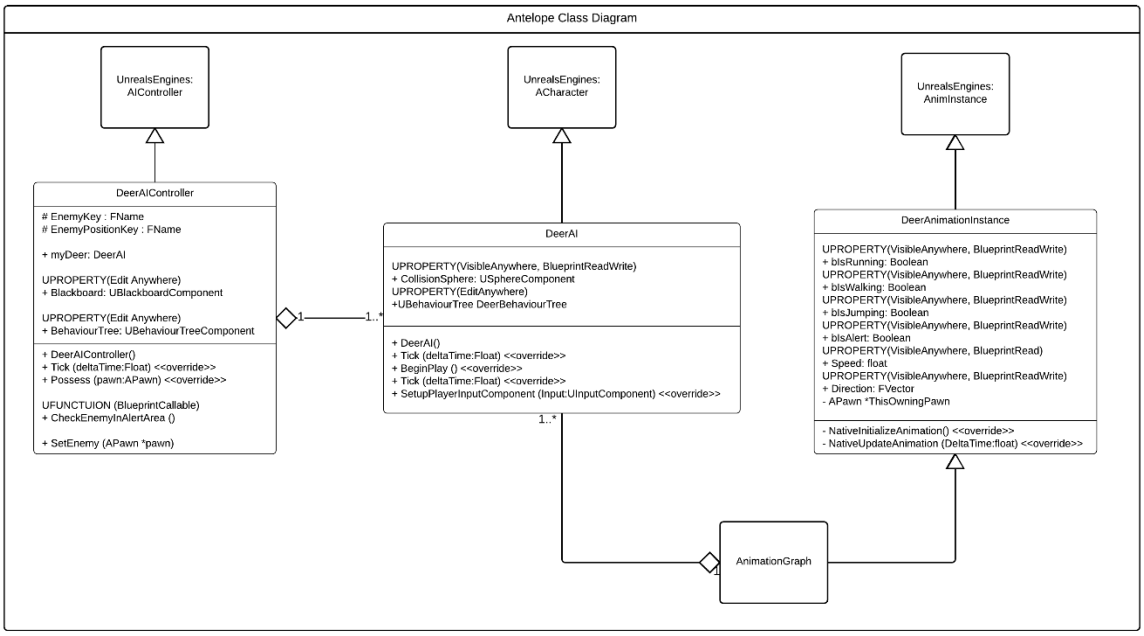


Figure 36 Particle System in editor

WHAT DIDN'T MAKE IT TO FINAL GAME

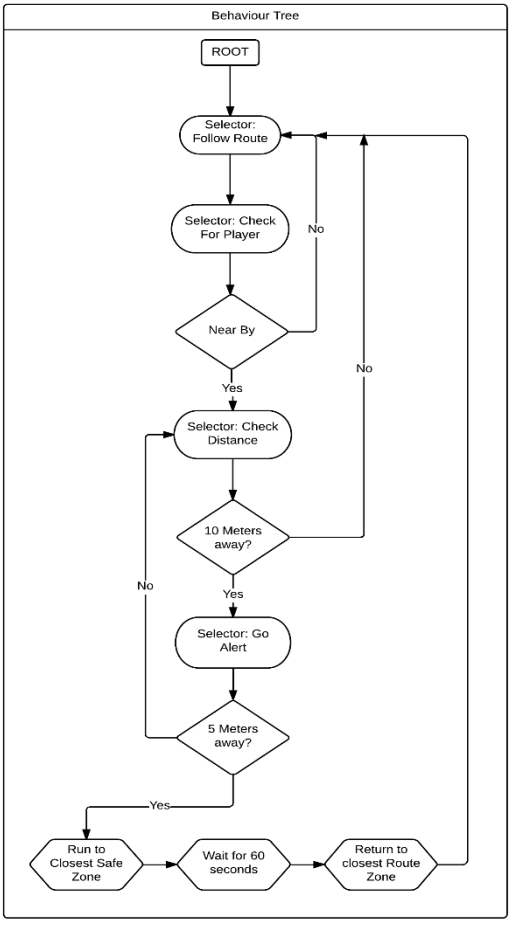
Antelope

Class Diagram



The AI controller would be built up so that it would host the behaviour tree and blackboard components. The behaviour tree component will work as the link between how to control the AI to the behaviour tree. The blackboard component works as a data storage/accessor for the AI which variables are written too for the behaviour tree to access. In the character class, the behaviour tree component will work as the brain for the AI entity. The animation instance will work as rules for transitioning from one animation loop to another.

Behaviour Tree



The behaviour tree uses a range of selectors and sequences to imitate the behaviour of an Antelope. The aim of this would be to use two types of inherited waypoint classes. One which is the route the AI follows, whilst the others are safe zones out of the players reach. The AI will follow a predefined route marked out by the waypoint route class. Upon following this the AI will check if the player is close by. If the player is close by the AI will go alert. If the character goes further than 10 meters it will return to the beginning of the node graph. However, if the player gets closer more towards 5 meters away. The AI will run to the closest safe zone, wait 60 seconds and then return to a route waypoint to continue its route. The functionality with this was helped by Looman (2015) in their zombie survival game in which they had AI zombies. The setup of this in C++ was very good as a reference to help implement this AI as there is little documentation on the C++ AI side unless you search through the engine.

Look for Player function:

This function would use a sphere component (A Large one) to check if the player is colliding with the AI. If colliding the AI will go into alert mode and then start looking at the distance value between the AI and the player.

If Player is found function:

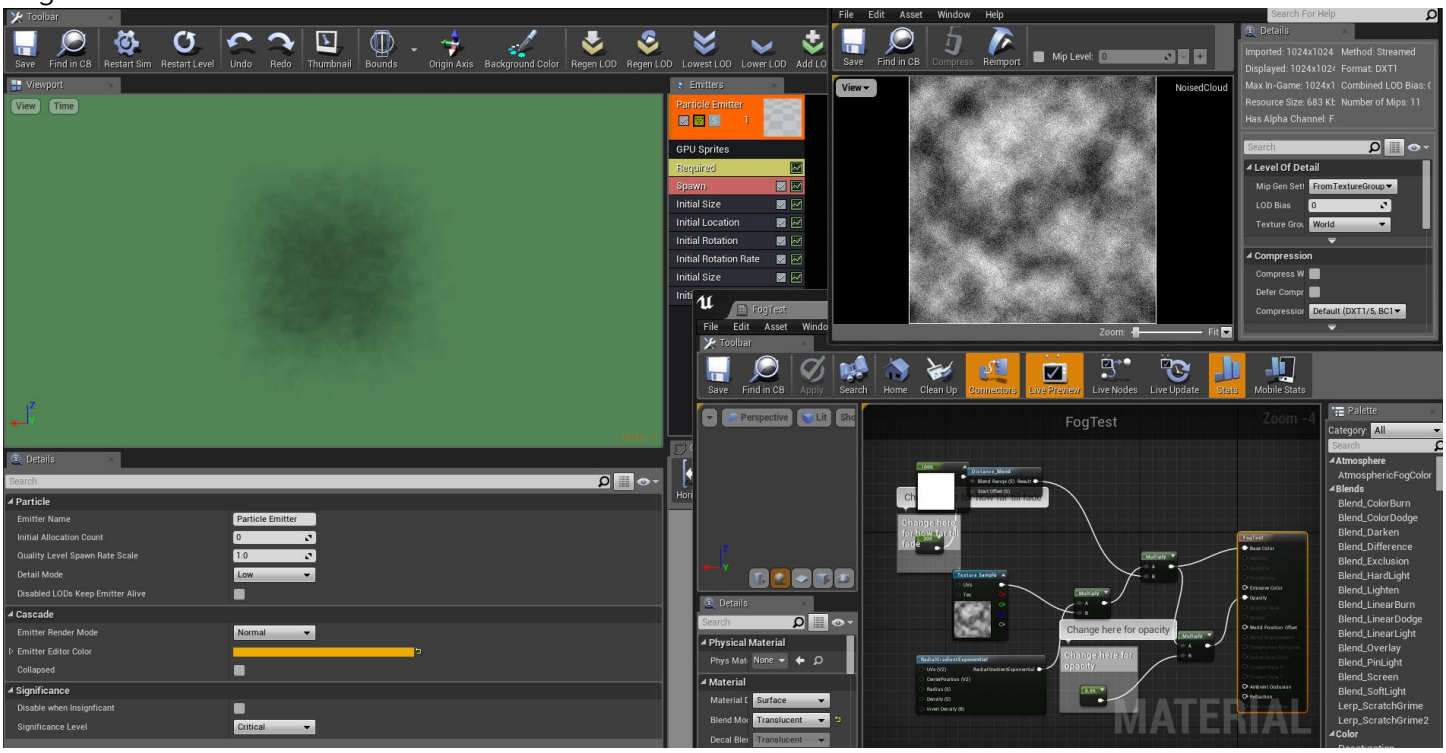
This function would take in the distance value from the Look for Player function if the player is found, the AI will search for the closest safe zone waypoint. Upon which it will start traversing the landscape to escape the player. It will wait for 60 seconds, upon once complete will return to one of the closest route waypoints.

Why it didn't make it in

The reason the Antelope didn't make it into the game is due to 3 weeks prior to the deadline the external environment being dropped to allow for the polishing of the internal environments instead. This also meant that the antelope was not fully tested and could not be polished to a deliverable standard.

Particle Simulation

Fog



The artists wanted to add some noise to the scene, not so much that it is noticeable, but a bit of smoke/fog to remove a bit of the "Cleanness" from the atmosphere. Unreal already provides an ambient dust particle effect which does add a bit of dirt to the atmosphere however this was not enough as it was quite minuscule. The main reference for this was a short video by DokipenTech (2015) and previous knowledge. First a cloud texture was created in photoshop and noise added to it to give the dusty effect that was required. This was imported to unreal and a shader was created. The shader will allow the particles to fade as the player gets closer to them.

The particle system itself has only one emitter on the GPU that uses the material as the particles. The emitter only emits the once in a single burst of 500 particles, the particle sprite size is between 100 and 1000 in size and are dispersed in a 1000^3 area, they move only ever so slightly. It adds a nice noise to the screen as seen in figure 37 which was a very quick test and placement. The right-hand side of the screenshot is clean and the one on the left is a bit dirty.

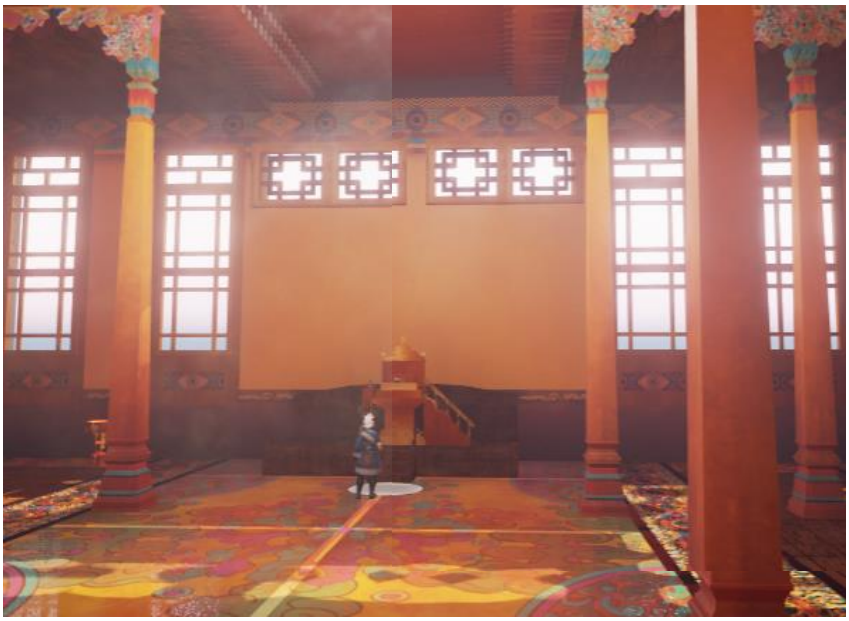


Figure 37 Particles within the scene. (Left) And without particles (Right)

Why it didn't make it in

Due to lack of time the particle system couldn't be added to the game as the artists didn't have time to add a decent texture or modify the colours. The area of effect had to be altered in each room as well which the artists wouldn't have been able to do. This meant that the particle system couldn't be polished to the standard of everything else in the game.

Candle Light

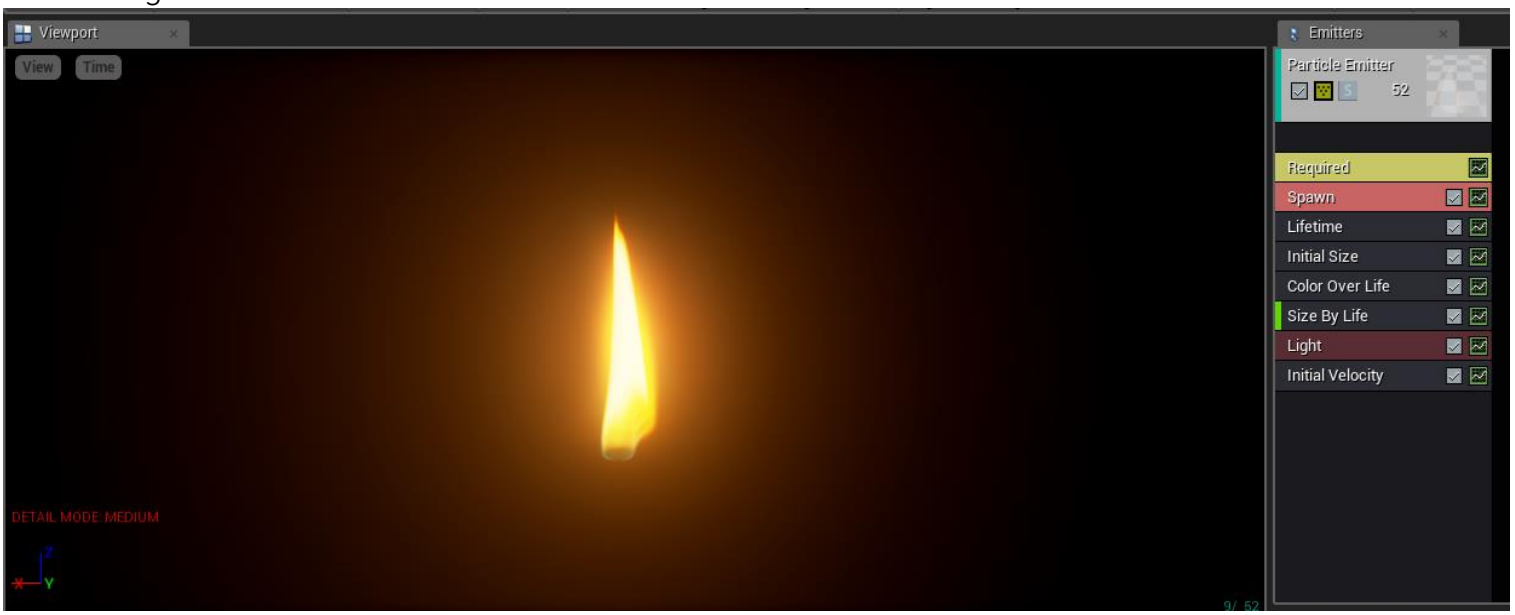


Figure 38 Candle Light Particle System Set up

The candle light particle simulation uses a single emitter as seen in figure 38. The flame itself is simply an image (Adona 2011), and the image is spawned roughly 3-5 particles at any one time. The size changes randomly on spawn which gives the flame the flickering, while also the lights brightness is sharply switched between 0 and 1 to give it the flickering we so desired. The light gives a soft orange glow that really adds to the light. Inspiration and assistance for this particle system came from the tutorial by Epic Games (2004).

Why it didn't make it in

This didn't make it to the game because of the environment. The candle light works best in darker environments which in testing was what was being worked with. This can be seen in figure 39, the light is just subtle enough and has a nice glow to add the atmosphere of the environment. As you can see in figure 40 in production the environment was far too bright for this to be noticeable and made the particle effect redundant. To solve this a torch light was implanted instead, which had a larger flame.

Figure 39 Candle light in dark environment

Figure 40 Environment

WORKING WITH ARTISTS

Restricting the artists was one of the more difficult areas within this project as there's no definitive list of "This is how many polys a model should have" or "This is the texture size map it should have". The list below was generated based on previous experience and suggestions from other game developers from a forum post on the epic games official forums (Whammy 2014). As this game will not be optimised with Level of Detail (LOD) the poly counts of items should remain quite low. Luckily Mip-Map generation of the textures is automatic, however still having 4K textures everywhere can be very expensive and aims and limitations where still added.

Poly Counts & Textures sizes

	Models (K = 1000)		**Textures**		**LOD**	
Item	Aim	MAX	Aim	MAX	Required?	To What Level?
AVA	15K	35K	2048 X 2048	4096 X 4096	No	0
Weasel	10K	20K	1024 X 1024	2048 X 2048	No	0
Antelope	10K	20K	1024 X 1024	2048 X 2048	No	0
Grass	10	20	128 X 128	512 X 512	No	0
Rocks	200	500	256 X 256	512 X 512	Yes	2
Trees	500	1500	512 X 512	1024 X 1024	Yes	2
Mountains	200	500	512 X 512	1024 X 1024	No	0
Decorative Assets	200	500	256 X 256	512 X 512	Yes	2
Buildings Assets	500	1000	512 X 512	1024 X 1024	Yes	2
Palace (Whole)	2000	5000	1024 X 1024	2048 X 2048	Yes	2

LOD means that the model requires it to be halved and exported X level amount. For example, with Rocks:
LOD0 would be a 500-poly model
LOD1 would be a 300-poly model
LOD2 would be a 50-poly model
This would then save performance with rendering when these items are in the background and too far to see the detail.

Sounds Specifications

PCM, ADPCM, DVI ADPCM
Format: .WAV
Bit Rate: 16
Speaker Channels: Mono, Stereo, 2.1, 4.1, 5.1 6.1, 7.1

Communications between Artists and Programmers.

During the project, it was found that there is an issue with the communication between artists and programmers and the misunderstanding of one another's jobs. There was a lot of "Can you just" statements without realising the implications of the request that was asked. An example of this from programming to artist, would be the programmer asking the artists to "Just" change a model so that it fit with the push and pull animation, this meant that the artist had to re-design the model and that the weighting of the model wouldn't work in the design. To a programmer it was simply adding a handle to the mesh and solve a large issue with a small change, however to the artist it was destroying their work. The same can be done in reverse, asking a programmer to "Just" implement a push and pull system. To an artist it is simply attaching the item to the player, to the programmer however there are issues with collisions, bugs, work arounds, restrictions in the implementation, it took time and it wasn't understood that it would cause a lot of different issues within the game mechanics.

CONCLUSION & EVALUATION

The overall game environment was polished with plenty of objects to fill a room and keep the rooms alive. Beautifully detailed textures were applied to walls, pillars and floors to make the rooms more colourful and keep the user interested. The animation of the character was smooth and transitioned very nicely from pose to pose, this included the weasel and how he was animated alongside this. The weasel added some extra life to the game as a companion to the player making the game seem less lonely and a bit fuller of life. The cinematic sequences animations all worked very nicely to give some story and life to the character, Ava. The design work, murals and beginning video added some past to our game giving the user some purpose to play the game. The design work itself was pretty and detailed and followed similar themes to that of the environment. The game mechanics all integrated the Tibetan culture as much as it could along with providing simple but fun puzzles for the player to do. Cut scenes themselves added to the story as the player progressed through the game.
The team has worked successfully together in producing a short game but in quite a large environment. The environment is not overly empty and with all the cut scenes the game doesn't feel like the player would be playing on their own either. The aim was to produce a game, and as a team, accomplished that. The game plays from beginning to end, the team have all learned different things, and on an individual basis have blossomed and learnt as the project progressed.
The planning out of the game was quite a success as the initial goal was overly ambitious and with mistakes made and issues arising time was lost. Having the ABC plan allowed the quick dropping of items and could still produce a high-quality game.
There was an issue during crunch time with the shaders as one of the artists made a mistake with parenting of shaders, whilst the programmer made a mistake in the shader itself by mixing up the blue and green channels of a normal. This caused a lot of issues with lighting as well as the UE4's "Brush" objects not calculating static lighting, all the walls and floors had to be converted to static meshes due to UV's being odd, the issue is still unclear, only the culprit was known.
This game could be improved by finishing the final garden (Figure 41) which the player could run around in before going through a final door instead. However, time did not allow for this to be finished as the rest of the palace had to be polished. Initially an external environment leading up to the palace was desired, which would have plenty of animals through AI, foliage and a training part on how to do all the puzzles, again, this was overly ambitious and only tests where implemented. Somethings that where desired but wasn't originally in the plan would be things such as IK (Inverse Kinematics) which places the actors foot in the right place, for example going up a slope and the foot match to the slope. This could also be done to attach a hand to a push and pull item.

Figure 41 Unfinished Garden

This project has not necessarily been overly complex, although still very difficult at times, but certainly has a lot of content for 11 weeks. The project was an overall success however the last push to finish it was very difficult as the level had to be separated out into multiple maps and optimised as there was too much within it. This term was called level streaming and mean that the only rooms that where processed was the room in front of the player, the room the player is in, and the room that the player is going too. This helped reduce processing power and lighting calculations freeing up space on the CPU and GPU and allowing the artists more freedom with their lighting. After this the final build was done which took 18 hours to get started as unreals packaging system is at acceptable standard, however some errors are not very clear, and are most of the time blank making it very hard to track down issues. Once this was completed and tests where done, the final stage was building lighting which took close to 4 hours at half the quality the team wanted.

REFERENCES

2017. Keyboard PNG. PNG. Clip Art All. Available from: http://clipartall.com/clipart/1472-keyboard-clipart.html [Accessed 16 May 2017].

Adona, N., 2011. Candle Light Tutorial Using VraySU 1.48.66 up. Photogrpah. Blogspot. Available from: http://nomeradona.blogspot.co.uk/2011/04/candle-light-tutorial-using-vraysu.html [Accessed 10 August 2017].

Ahkâm, 2017. Mouse Png. PNG. Free Icons Png. Available from: http://www.freeiconspng.com/img/23280 [Accessed 16 May 2017].

Bourrelly, F, 2016. How to create movies in Unreal engine 4.12. *New* Sequencer Tutorial. Video. YouTube. Available from: https://www.youtube.com/watch?v=RqWaa08V_dw [Accessed 31 July 2017].

Cinebeast, 2016. Transitioning from a Sequence into gameplay. UE4 Answer hub. 26 September 2016. Available from: https://answers.unrealengine.com/questions/495697/transitioning-from-a-sequence-into-gameplay.html [Accessed 1 August 2017].

Davidson, M. J., 2014. How do I implement Pure Virtual Methods?. UE4 Answer hub. 11 April 2014. Available from: https://answers.unrealengine.com/questions/28225/pure-virtual-methods.html [Accessed 20 July 2017].

DokipenTech, 2015. Unreal Engine 4 Atmospheric Fog FX Setup demo. Video. Unreal Engine: Youtube. Available from: https://www.youtube.com/embed/sc7Jj_9fNYs [Accessed 14 June 2017].

Epic Games, 2017. Sequencer Overview. Unreal Engine. Available from: https://docs.unrealengine.com/latest/INT/Engine/Sequencer/Overview/ [Accessed 2 August 2017].

Eric, 2017. Xbox Controller PNG Free Download. PNG. PNG Mart. Available from: http://www.pngmart.com/image/32169 [Accessed 16 May 2017].

Looman, T., 2015. Survival Sample Game: Section 3. Epic Games. Available from: https://wiki.unrealengine.com/Survival_Sample_Game:_Section_3 [Accessed 12 July 2017].

Rogers, S., 2014. Level Up! The Guide to Great Video Game Design. 2nd ed. USA: Wiley.

Unreal Engine, 2004. Candle Fire Particle - Video. Video. Epic Games. Available from: https://wiki.unrealengine.com/Candle_Fire_Particle_-_Video [Accessed 10 August 2017].

Whammy, 2014. Thread: Recommended poly count for player models and weapons. Epic Games Forums. 07 December 2014. Available from: https://forums.unrealengine.com/showthread.php?18120-Recommended-poly-count-for-player-models-and-weapons&highlight=character+count [Accessed 11 May 2017].

APPENDIX

External Document A – The GDD

External Document B – ABC Plan

External Document C – Gantt Chart

External Document D – Production Diary (Post)

External Document E – Initial Thoughts and Worries

External Document F - Full Puzzle UML Diagram

Code Snippet G –

```
static ConstructorHelpers::FObjectFinder<UCurveFloat> Curvy(TEXT("/Game/TemporaryContent/DoorSwing.DoorSwing"));
if (Curvy.Object)
{
	fCurve = Curvy.Object;
	ScoreTimeline = CreateDefaultSubobject<UTimelineComponent>(TEXT("TimelineScore"));
	InterpFunction.BindUFunction(this, FName{ TEXT("TimelineFloatReturn") });
}
```

Code Snippet H

```
void ADoor::TimelineFloatReturn(float val)
{
	designatedMesh->SetRelativeRotation(FRotator(0 , val, 0));
}
```

Code snippet I –

```
TArray <AActor*> overlappingActors;
overlappingSphere->GetOverlappingActors(overlappingActors);
for (int32 overlappingActorIndex = 0; overlappingActorIndex < overlappingActors.Num(); overlappingActorIndex++)
{
	AStatueObject* const overlappingTestSO = Cast<AStatueObject>(overlappingActors[overlappingActorIndex]);

	if (overlappingTestSO && overlappingTestSO->bCanPlayerPray)
	{
		overlappingTestSO->bCanPlayerPray = false;
		overlappingTestSO->isPlayerPraying = true;
		bIsPraying = true;
	}
}
```

Code snippet J –

```
for (size_t i = 0; i < pushPullObjects.Num(); i++)
{
	TArray <AActor*> overlappingActors;
	pushPullObjectsTargets[i]->GetOverlappingActors(overlappingActors);
	for (size_t index = 0; index < overlappingActors.Num(); index++)
	{
		APushPullItem* const overlappingTest = Cast<APushPullItem>(overlappingActors[index]);
		if (overlappingTest && overlappingTest == pushPullObjects[i])
		{
			overlaps[i] = true;
		}
```


```
		}
	}
	if (checkOverlaps())
	{
		bPuzzleComplete = true;
	}
```

Code Snippet K –

```
void AMainCharacter::PushPull()
{
	attachedPushPullItem = Cast<APushPullItem>(results.GetActor());
	if (attachedPushPullItem)
	{
		GetCharacterMovement()->MaxWalkSpeed = pushPullSpeed;
		FAttachmentTransformRules attachRules = FAttachmentTransformRules(EAttachmentRule::KeepWorld, false);
		attachedPushPullItem->AttachToActor(this, attachRules);
		MoveIgnoreActorAdd(attachedPushPullItem);
		GetCharacterMovement()->bOrientRotationToMovement = false;
		bIsPushPulling = !bIsPushPulling;
	}
	if (!bIsPushPulling)
	{
		StopPushPull();
	}
}
```

Code Snippet L –

```
FRotator tempRot = GetActorRotation();
FRotator tempYaw = FRotator(0, tempRot.Yaw, 0);
FVector tempFV = FRotationMatrix(tempYaw).GetUnitAxis(EAxis::X);
FVector tempActorLocation = FVector(GetActorLocation().X, GetActorLocation().Y, GetActorLocation().Z + characterArmHeight);
FVector end = (tempActorLocation + (tempFV * pushPullTraceCheckDistance));
FHitResult results;
FCollisionQueryParams query = FCollisionQueryParams(FName(TEXT("trace")), false, this);
bool hit = GetWorld()->LineTraceSingleByChannel(results, tempActorLocation, end, ECC_Visibility, query);
```

Code Snippet M –

```
ItemData a;
a.Add(ItemDataVariables::itemDescription," A symbol of the infinite connections in life that bind us all together.");
a.Add(ItemDataVariables::itemEnumIdentifier, "Scroll01");
a.Add(ItemDataVariables::itemMeshFilepath, "/Game/Ava/Modelling/Interactives/Scroll1.Scroll1");
a.Add(ItemDataVariables::itemSpriteFilepath, "/Game/Ava/UI/Knot.Knot");
a.Add(ItemDataVariables::itemSelectedSpriteFilepath, "/Game/Ava/UI/Knot_Hovered.Knot_Hovered");
a.Add(ItemDataVariables::itemName, "The Endless Knot");
tItems.Add(Item::Scroll01, a);
```

Code Snippet N –

```
APickUpItem* const overlappingTest = Cast<APickUpItem>(overlappingActors[overlappingActorIndex]);
if (overlappingTest && overlappingTest->itemType == requiredItem)
{
	bItemOffered = true;
	ItemData e = ADataFactory::Items[requiredItem];
```


```
		UStaticMesh* mesh = Cast <UStaticMesh>(StaticLoadObject(UStaticMesh::StaticClass(), NULL, e[ItemDataVariables::itemMesh-
Filepath].GetCharArray().GetData()));
		OfferingItem->SetStaticMesh(mesh);
		overlappingTest->Destroy();
}
```

Code Snippet O –

```
bool characterFound = false;
TArray <AActor*> overlappingActors;
overlappingSphere->GetOverlappingActors(overlappingActors);
for (size_t overlappingActorIndex = 0; overlappingActorIndex < overlappingActors.Num(); overlappingActorIndex++)
{
		AMainCharacter* const overlappingTest = Cast<AMainCharacter>(overlappingActors[overlappingActorIndex]);
		if (overlappingTest)
		{
			characterFound = true;
			currentRemaingTime -= 0.1;
			if (currentRemaingTime <= 0.f)
			{
				overlappingTest->Inventory.Add(ThroneItem->itemType);
				ThroneItem->Destroy();
				bPuzzleComplete = true;
			}
		}
}
if (!characterFound)
{
		currentRemaingTime = requiredWaitingTime;
}
```

Code Snippet P –

```
APickUpItem* const overlappingTestPUI = Cast<APickUpItem>(overlappingActors[overlappingActorIndex]);
if (overlappingTestPUI && !overlappingTestPUI->IsPendingKill() && overlappingTestPUI->bCanPlayerPickUp)
{
		bIsPickingUp = true;
		bPlaySoundPickup = true;
		Inventory.Add(overlappingTestPUI->itemType);
		overlappingTestPUI->Destroy();
		bIsPickingUp = false;
}
```