Report on changes made to the SteamBound game

Introduction

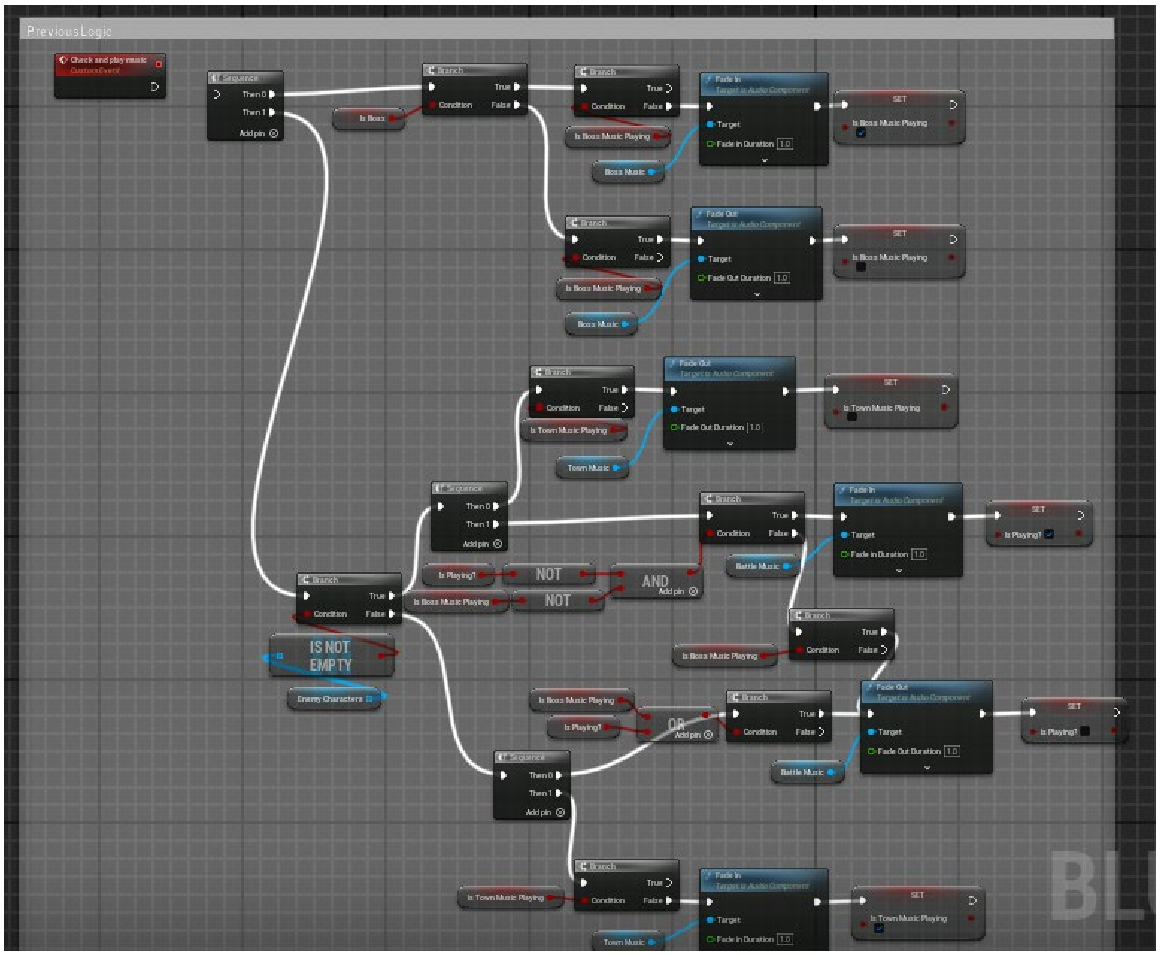
Inspired by one of my Advanced Data Structures class projects, I decided to continue improving the SteamBound game by adding 3 relevant data structures, 3 algorithms and 3 visual representations of each algorithm. Each data structure is added to improve and optimize the code structure. The algorithms, along with their visual representation in-game, are added to practically apply my knowledge of them in Unreal Engine blueprints and to improve the game's accessibility. Below is the detailed documentation of the changes that I made to the original version of the game submitted for the GameOff2024 game jam.

Data Structure 1 : HashMap

The BP_MusicManager blueprint was created to manage the different music in the game, in other words, when to change the music depending on the presence of enemies. In the initial structure, there were many boolean variables and several node redundancies.

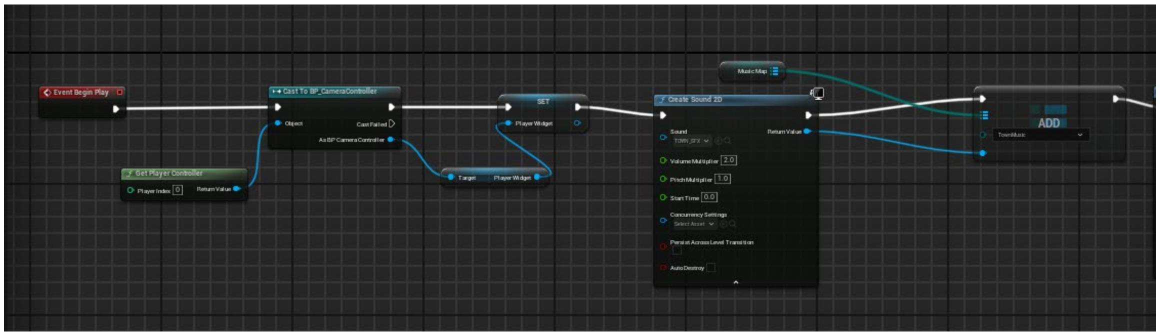
Adding a HashMap avoids these redundancies and allows storing music variables in a Map to allow flexibility and quick search for each music using Keys. Furthermore, instead of creating variables for each song, we can simply initiate the sounds we need using the Create Sound 2D node and add them to the Map.

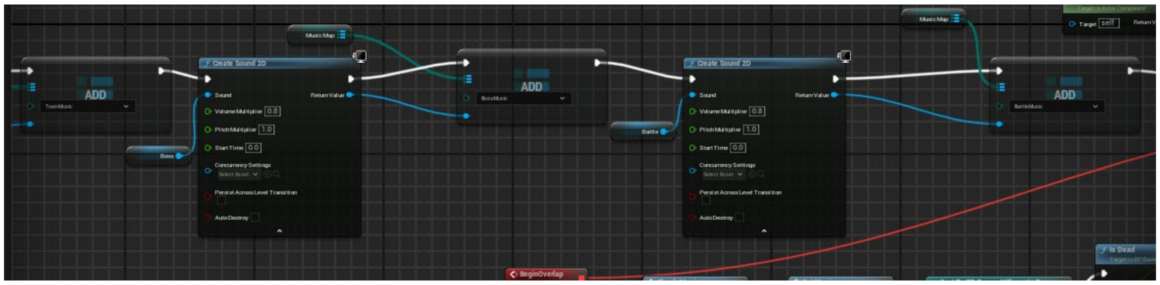
Old implementation :



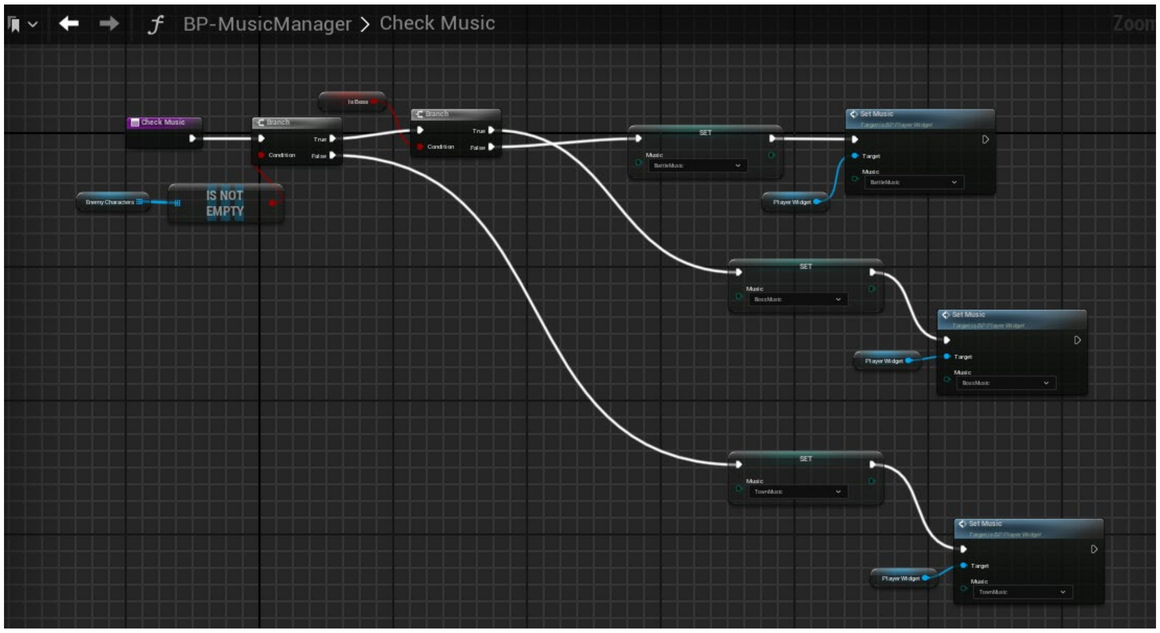
New implementation:

We add the music variables to the Map:

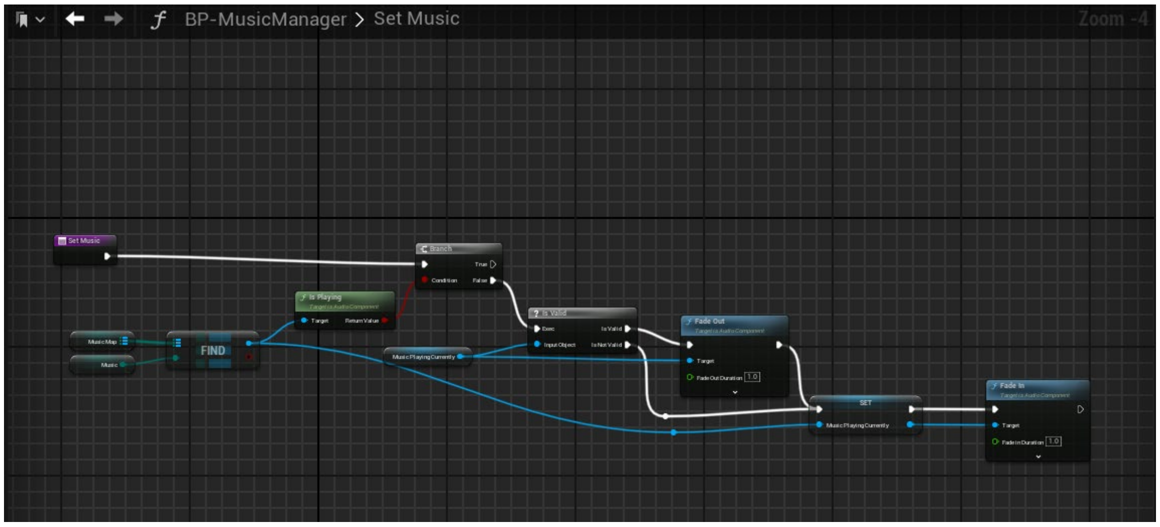




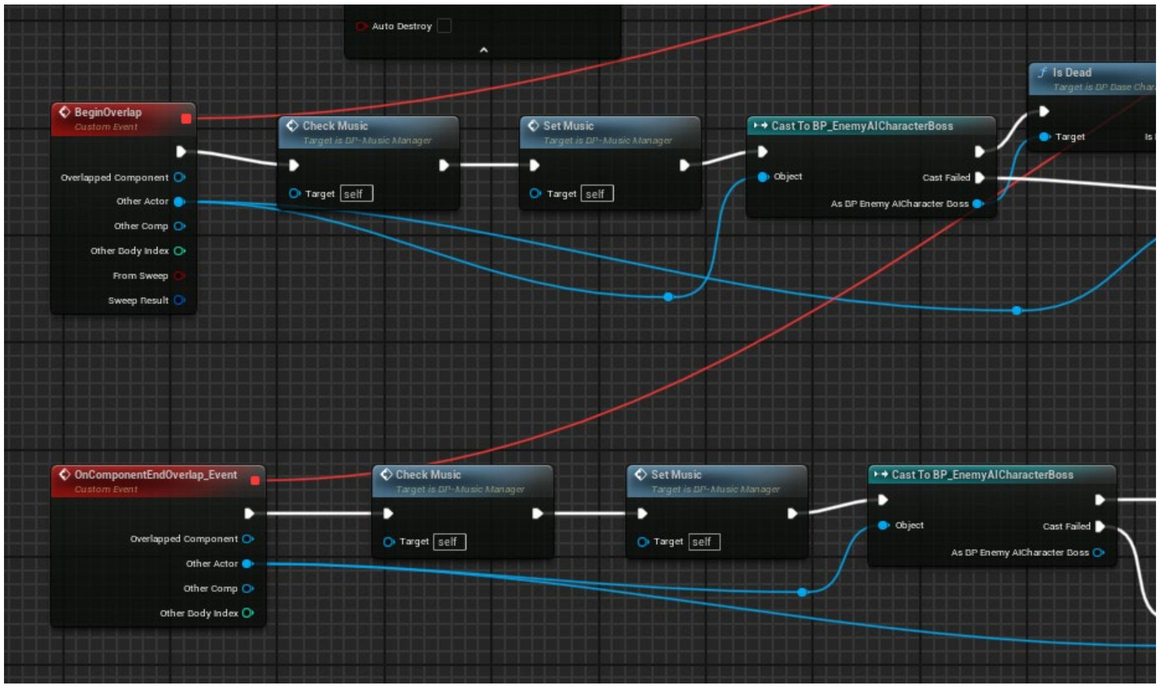
Function Check Music to determine which music should play:



Function Set Music to change the music if needed:



In the game, there are several trigger boxes, so each time the player passes them, we will call the 2 functions to check if a change of music is needed:



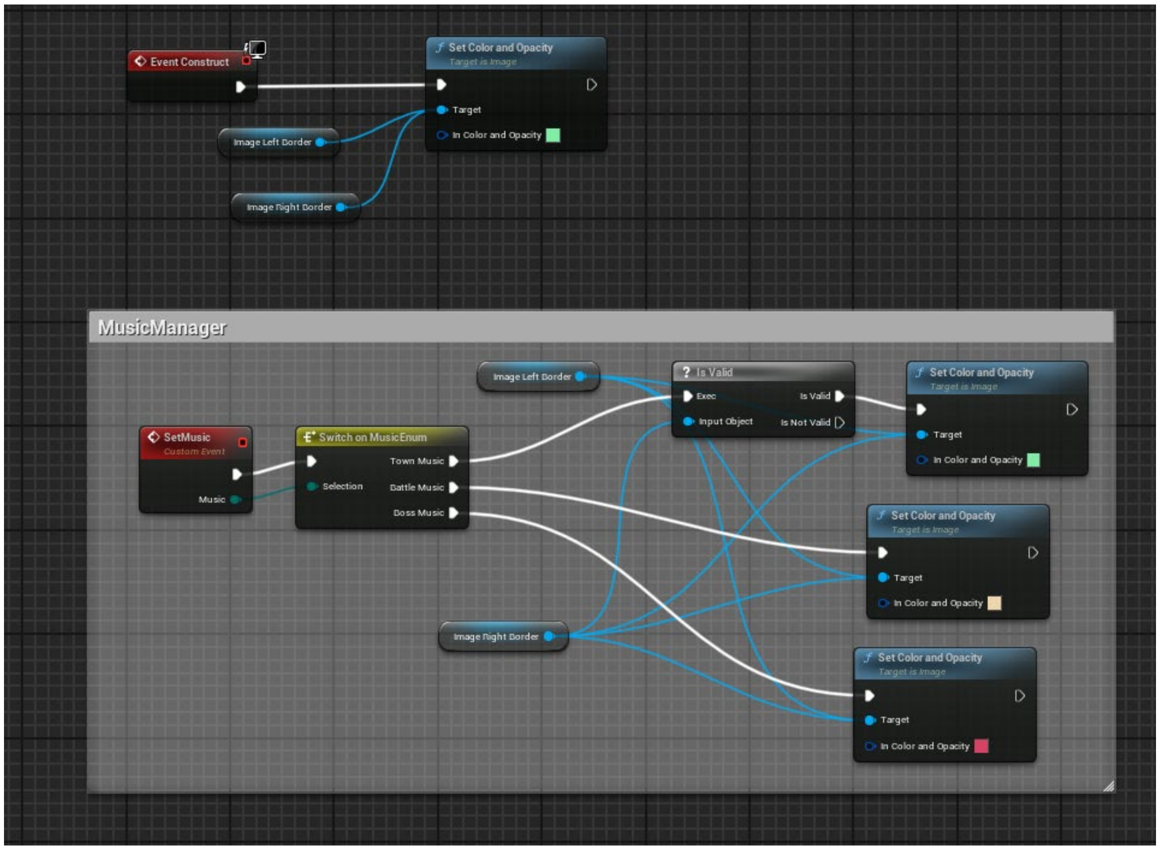
Algorithm : Dynamic Programming

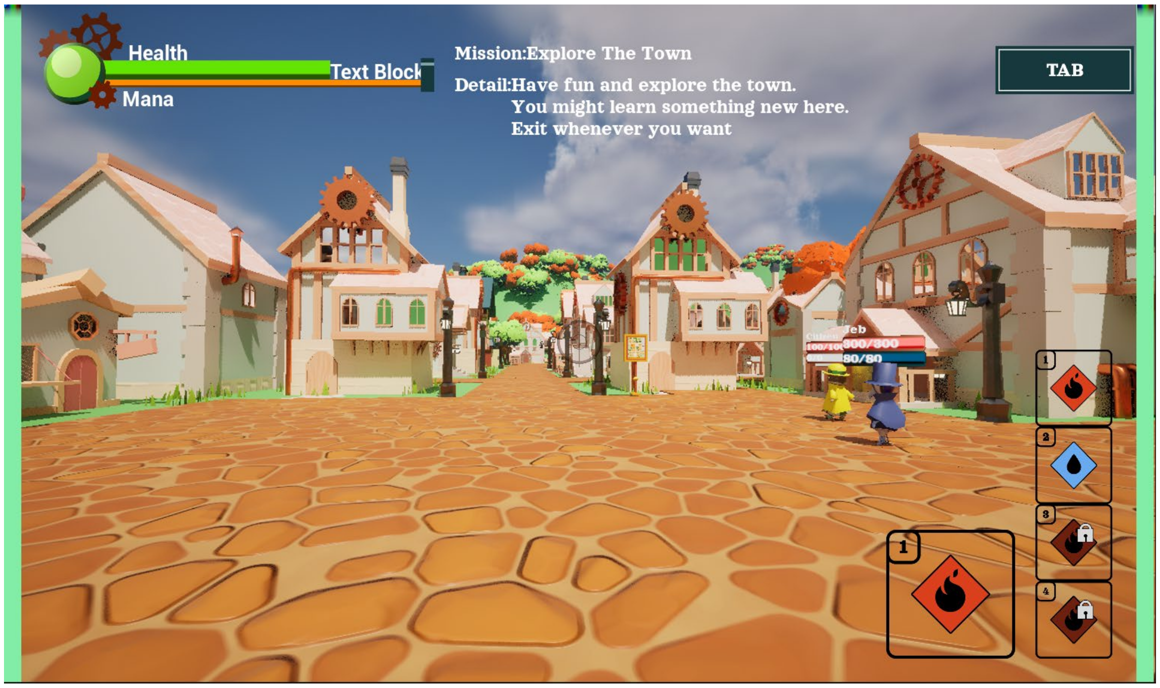
Dynamic Programming is an optimization algorithm that allows you to improve recursive code by reducing the workload of a function, or by grouping several repetitive blocks of code into a single function. The way I applied this algorithm is by creating 2 functions (Set Music and Check Music) that replaced the many nodes used to create the music change logic. Instead of creating several Branches and booleans and continuously checking if a particular music is playing, we can simply call these 2 new functions in the OnComponentBeginOverlap and OnComponentEndOverlap.

Visual representation:

To demonstrate the music changing throughout the game, I modified the PlayerWidget by adding 2 vertical bars on each side of the screen (left and right) that will change color depending on the music playing. By default, the music that plays when the game starts is TownMusic, which is represented by the color green. Then, when the player encounters enemies, the color turns orange. Finally, when the player finds the boss, the color turns red.

Player widget:



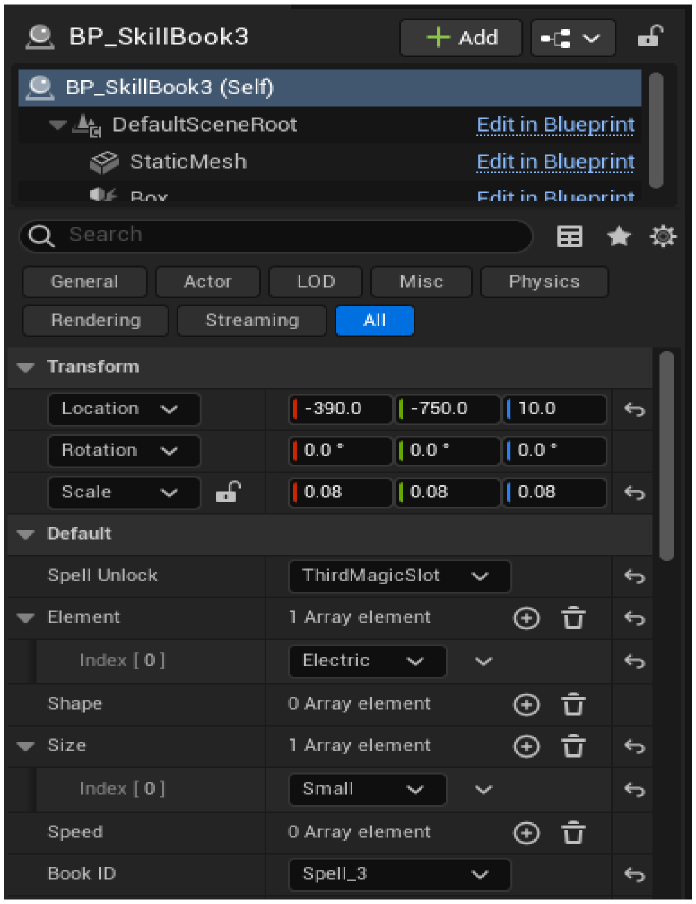

Health
Text Block
Mana
Mission:Explore The Town
Detail:Have fun and explore the town.
You might learn something new here.
Exit whenever you want
TAB
Jeb
800/800
80/80
1
2
3
4
1

Data Structure 2 : Array

In the BP_SkillBook, which are books found in different places in the game, there was a logic to make the player learn certain "Spells" and store them in a progression Map. When you interact with a SkillBook in the game, you can learn one or more spells, and this logic was managed with a progression Map and a single Enum containing the 4 Enums for all the spells, which had to be added to the Map.

In the game, each SkillBook contains a description of a single spell (for example, the Fast spell, which allows you to cast spells at high speed), but our initial plan was to make the player learn more than just one spell with some books, even if the description is for a single spell. To allow the selection of the specific spells we want the player to learn with each book, I decided to create 4 enum arrays for each type of spell: Element, Shape, Size and Speed. This way, in the level editor, we can change the properties of each instance of the Blueprint SkillBook by adding the elements we need to the corresponding spell array, and thus facilitate the job for the level designers and narrative designers.

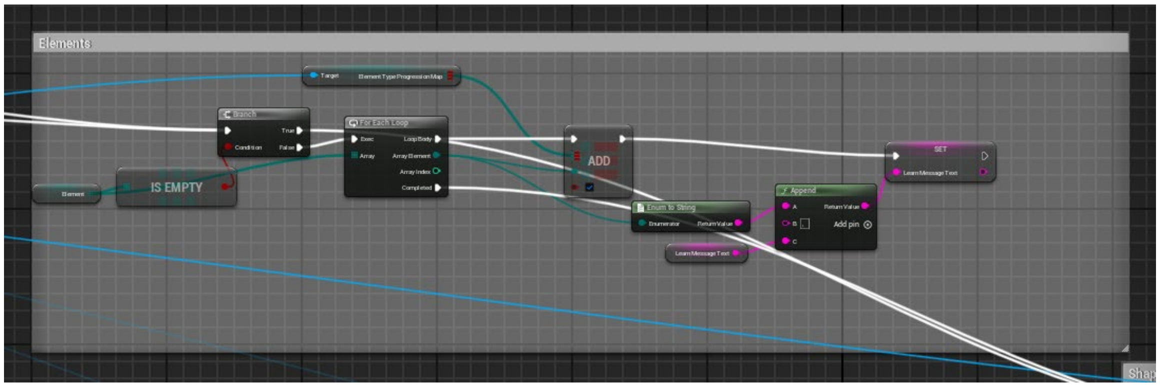
For example:



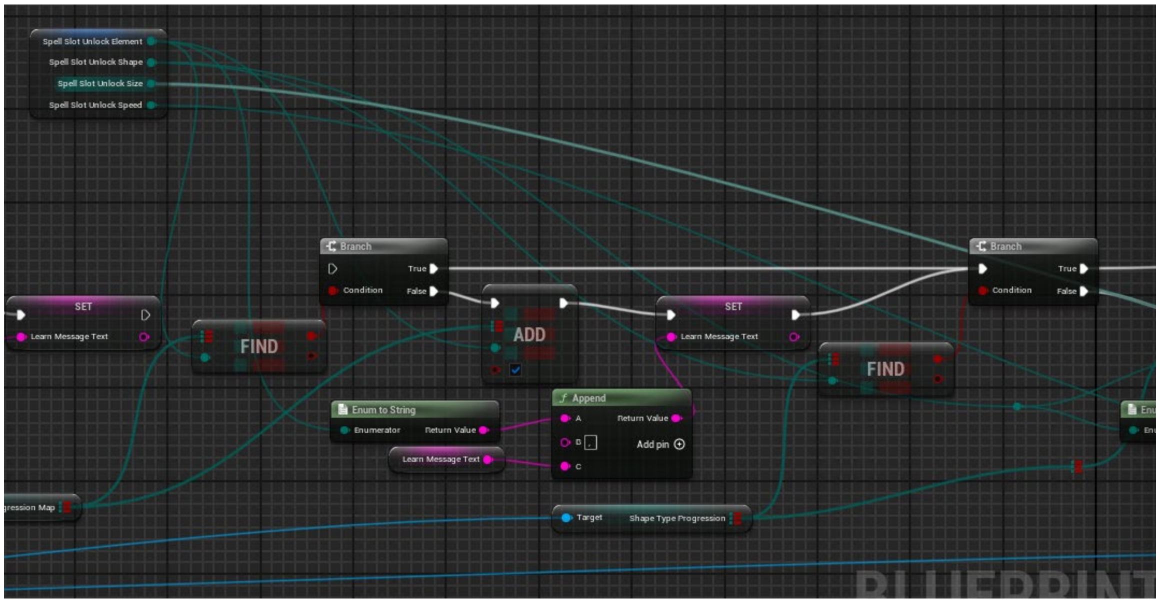
Location	-390.0	-750.0	10.0
Rotation	0.0 °	0.0 °	0.0 °
Scale	0.08	0.08	0.08

Spell Unlock	ThirdMagicSlot
Element	1 Array element
Index [0]	Electric
Shape	0 Array element
Size	1 Array element
Index [0]	Small
Speed	0 Array element
Book ID	Spell_3

Here is the logic for each of the 4 arrays of spells:



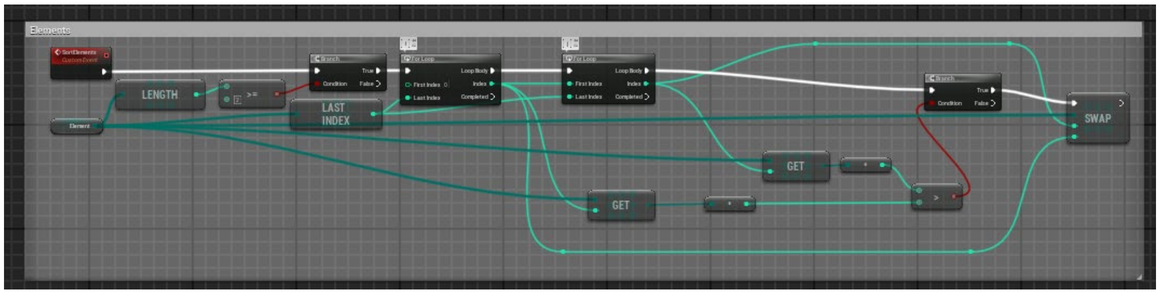
It used to look like this:



Algorithm : Bubble Sort

I implemented the Bubble Sort algorithm to sort the spells the player will acquire when interacting with a SkillBook. This will allow the player to view the spells they have learned in the order they see them in the LoadoutWidget (which is opened by pressing Tab). Bubble Sort sorts the elements in each array in the order they appear in the enums, as well as in the order they appear in the LoadoutWidget.

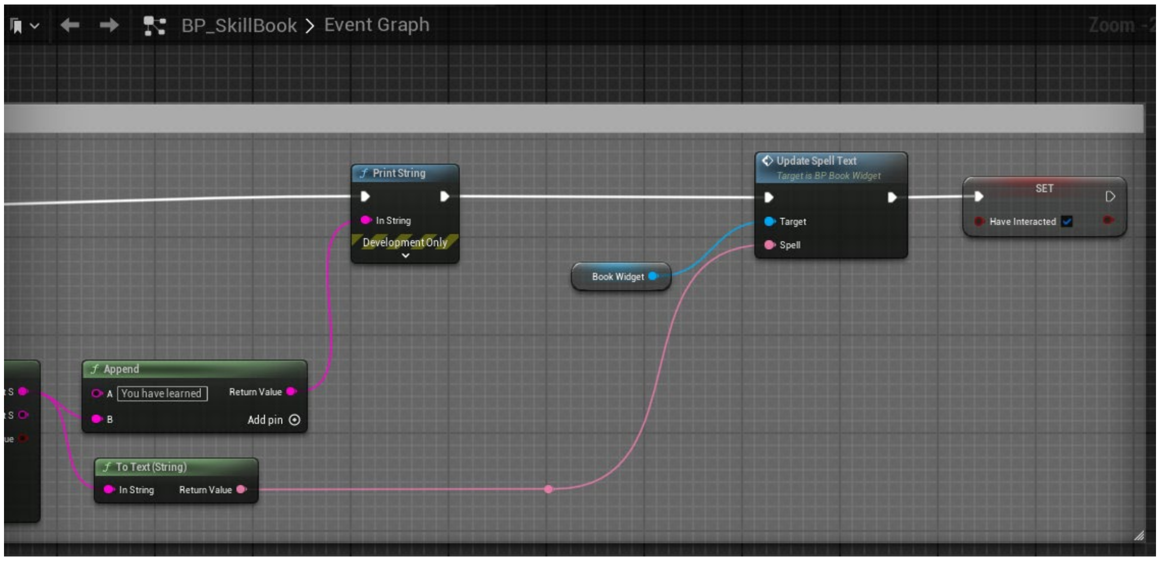
Bubble sort implementation:



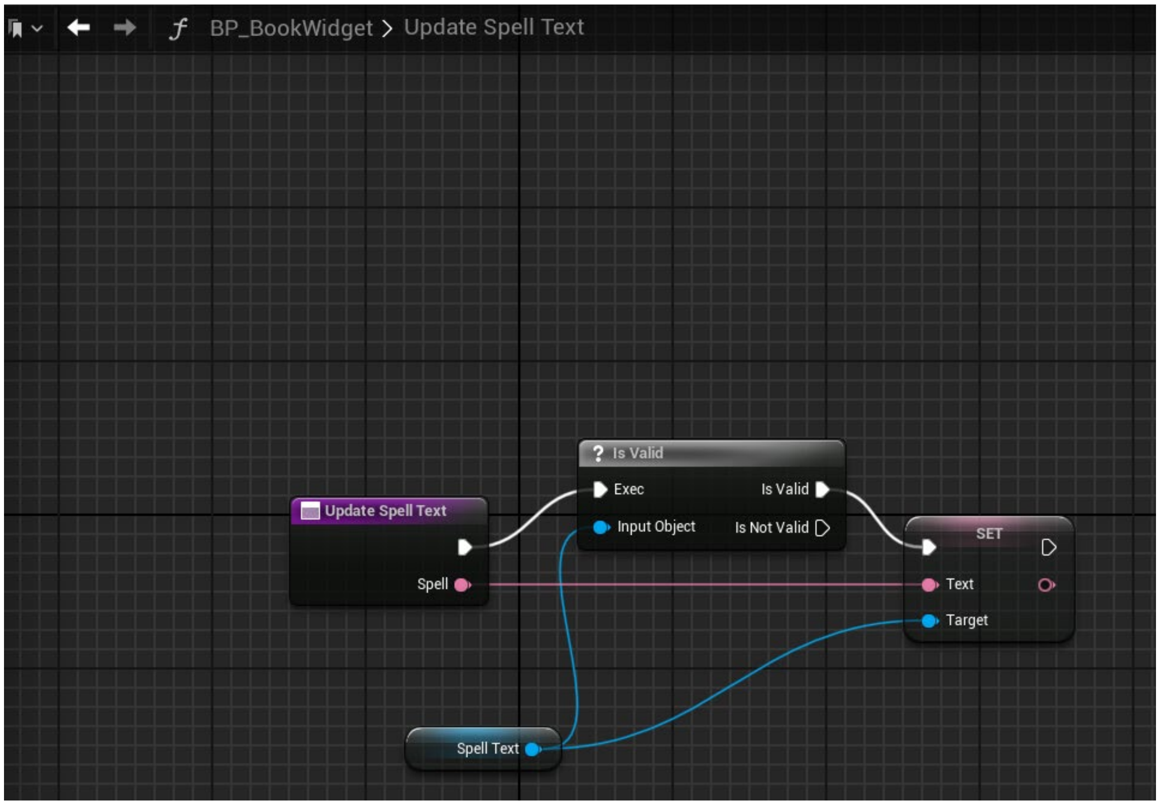
Visual representation:

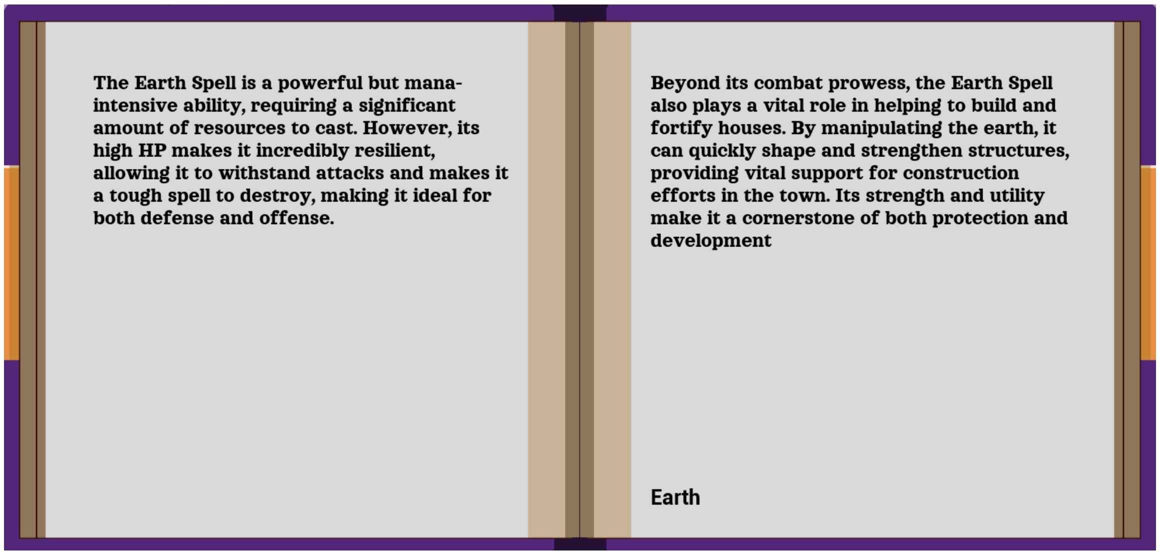
To make the game more accessible and more concise for the player, I added a text in the BookWidget to indicate which spell the player will acquire when opening a SkillBook. Originally, the books only contained a description of the spell(s), but I think that we can make it more precise by adding at the bottom of the BookWidget the name of the spell(s) that the player has just learned. In addition, the spells are sorted in this text (with the bubble sort) according to the default order that can be seen by clicking Tab in the game.

Where I call the function in Skillbook to update the text in the widget:



The function that I created in the BookWidget:





The Earth Spell is a powerful but mana-intensive ability, requiring a significant amount of resources to cast. However, its high HP makes it incredibly resilient, allowing it to withstand attacks and makes it a tough spell to destroy, making it ideal for both defense and offense.

Beyond its combat prowess, the Earth Spell also plays a vital role in helping to build and fortify houses. By manipulating the earth, it can quickly shape and strengthen structures, providing vital support for construction efforts in the town. Its strength and utility make it a cornerstone of both protection and development

Earth

Data Structure 3 : Graph (Dependency graph)

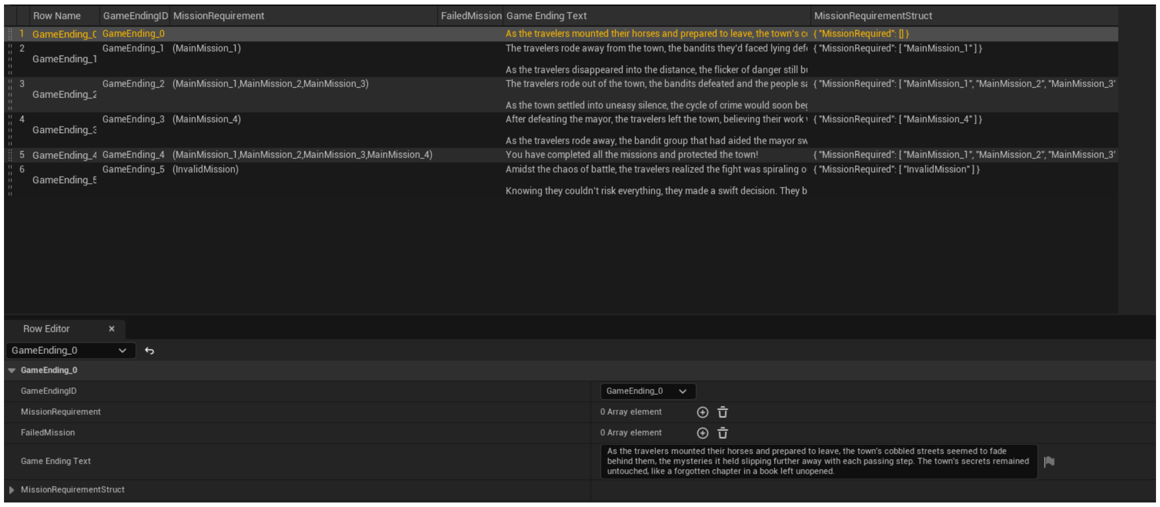
The game has multiple endings, and the conditions required to unlock each ending are the game's different missions. The DT_GameEndings data table shows the missions required to unlock each ending. Initially, there was no Blueprint to manage the endings, and they were managed in the GameEnd Widget and the BP_CameraController, which I think is too scattered. Furthermore, the Widget should not be responsible for managing the endings, nor the CameraController, since the latter should only contain camera-related logic. The initial function to evaluate the ending to display had nothing to store the endings that are already unlocked, which is why I created 2 Maps to store this information.

The first Map stores the endings and their corresponding required missions. It is used to iterate through the missions and check if all required missions are among the missions the player has completed.

The second Map stores the endings with a Boolean value (Unlocked). This is used to check if some endings are already unlocked. If not, we will continue with the checks and when we unlock an ending, we will add it to the Map with Unlocked marked True.

To correctly implement this data structure, I also had to modify the BP_Mission and BP_CameraController blueprints, as well as the GameEnd widget.

Here is the Data Table (the ending 5 was not implemented):



	Row Name	GameEndingID	MissionRequirement	FailedMission	Game Ending Text	MissionRequirementStruct
1	GameEnding_(	GameEnding_0			As the travelers mounted their horses and prepared to leave, the town's c	{ "MissionRequired": [] }
2	GameEnding_1	GameEnding_1	(MainMission_1)		The travelers rode away from the town, the bandits they'd faced lying def As the travelers disappeared into the distance, the flicker of danger still bu	{ "MissionRequired": ["MainMission_1"] }
3	GameEnding_2	GameEnding_2	(MainMission_1,MainMission_2,MainMission_3)		The travelers rode out of the town, the bandits defeated and the people sa As the town settled into uneasy silence, the cycle of crime would soon beg	{ "MissionRequired": ["MainMission_1", "MainMission_2", "MainMission_3'
4	GameEnding_3	GameEnding_3	(MainMission_4)		After defeating the mayor, the travelers left the town, believing their work As the travelers rode away, the bandit group that had aided the mayor sw	{ "MissionRequired": ["MainMission_4"] }
5	GameEnding_4	GameEnding_4	(MainMission_1,MainMission_2,MainMission_3,MainMission_4)		You have completed all the missions and protected the town!	{ "MissionRequired": ["MainMission_1", "MainMission_2", "MainMission_3'
6	GameEnding_5	GameEnding_5	(InvalidMission)		Amidst the chaos of battle, the travelers realized the fight was spiraling o Knowing they couldn't risk everything, they made a swift decision. They b	{ "MissionRequired": ["InvalidMission"] }

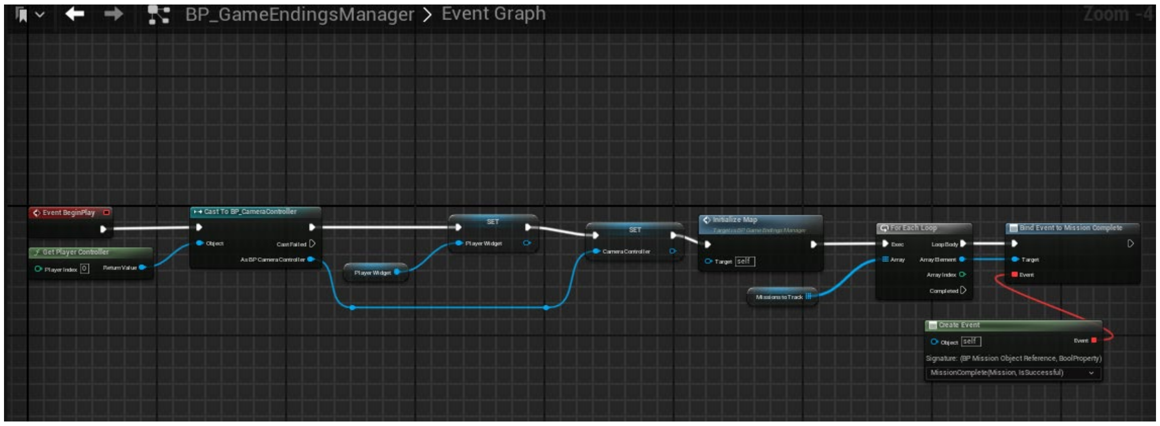
Row Editor

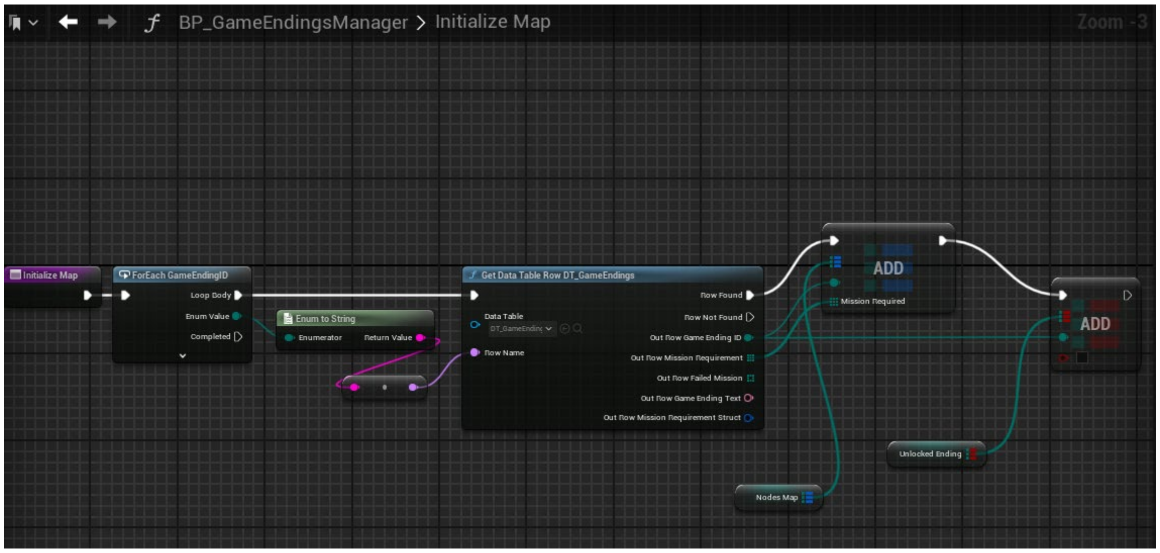
GameEnding_0

GameEnding_0

GameEndingID	GameEnding_0
MissionRequirement	0 Array element
FailedMission	0 Array element
Game Ending Text	As the travelers mounted their horses and prepared to leave, the town's cobbled streets seemed to fade behind them, the mysteries it held slipping further away with each passing step. The town's secrets remained untouched, like a forgotten chapter in a book left unopened.
MissionRequirementStruct	

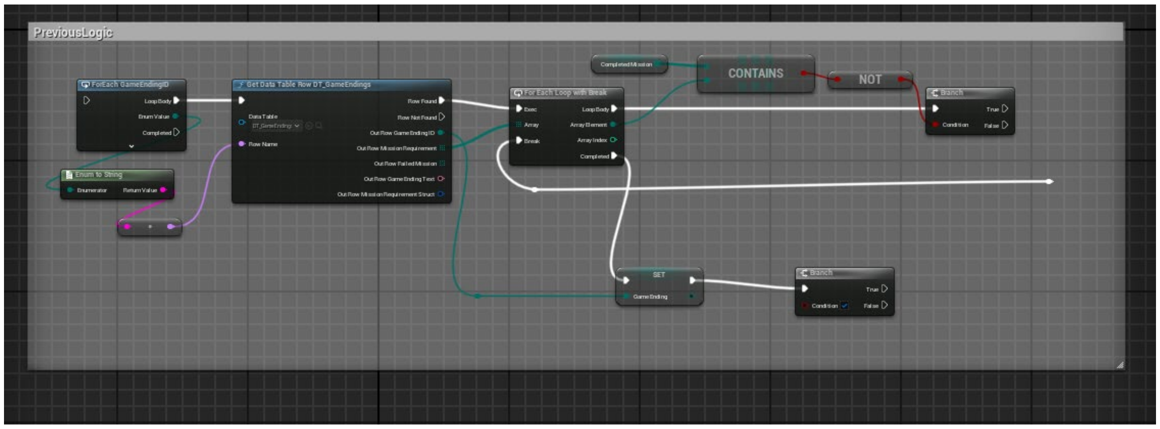
New Blueprint GameEndingsManager created for the graph:





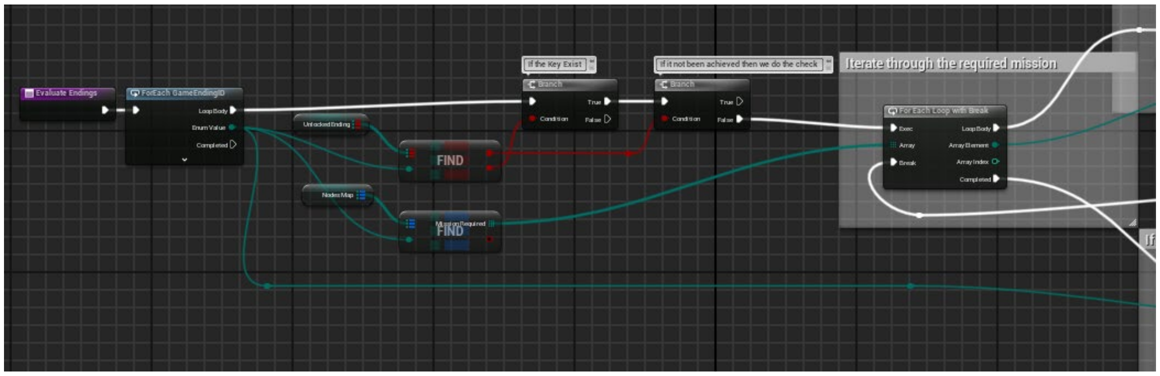
Graph:

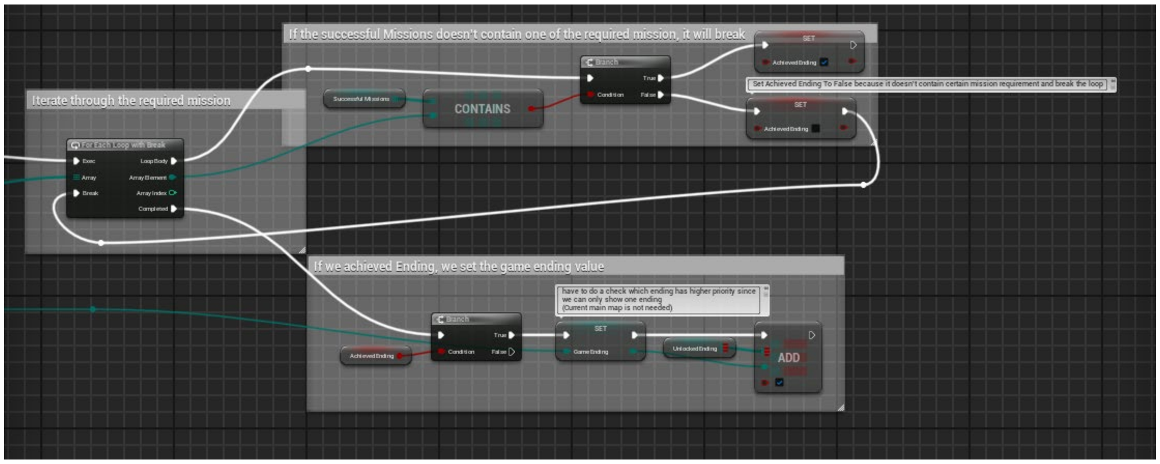
Old implementation:



New implementation:

The EvaluateEndings function is called in the MissionComplete function (which was already created previously), which is called each time the player completes a mission.



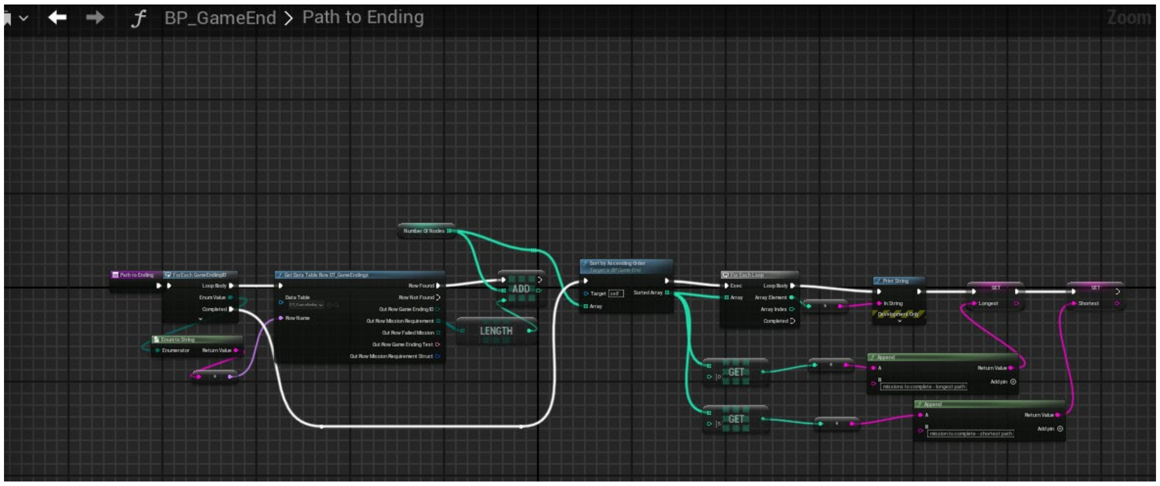

Iterate through the required mission
For Each Loop with Break
Exec
Array
Break
Loop Body
Array Element
Array Index
Completed
If the successful Missions doesn't contain one of the required mission, it will break
Successful Missions
CONTAINS
Branch
Condition
True
False
SET
Achieved Ending
Set Achieved Ending To False because it doesn't contain certain mission requirement and break the loop
SET
Achieved Ending
If we achieved Ending, we set the game ending value
have to do a check which ending has higher priority since we can only show one ending (Current main map is not needed)
Achieved Ending
Branch
Condition
True
False
SET
Game Ending
Unlocked Ending
ADD

Algorithm : Bubble sort with a customized function

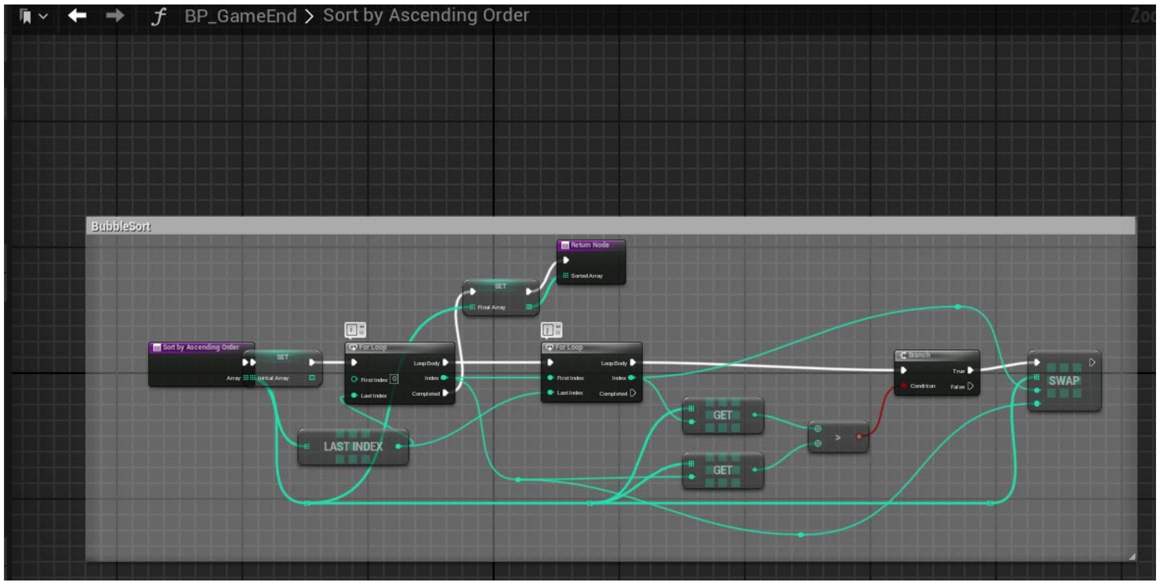
Since the player has no knowledge of the possible endings of the game and does not have a list of missions to do or the missions completed, I created an algorithm that finds the longest ending (which requires the most missions to unlock) and the shortest ending (which requires 0 mission). When the GameEnd widget is displayed, the player will be able to see if they have successfully unlocked the shortest and longest endings, as well as information about the number of missions to unlock them. To make it easier for players to understand, I put the visual representation of this algorithm in the GameEnd widget.

Bubble Sort sorts the lengths of the required missions' arrays in ascending order, and the PathToEnding function will take the sorted elements and assign them to two strings, one showing the shortest path to ending and one showing the longest.

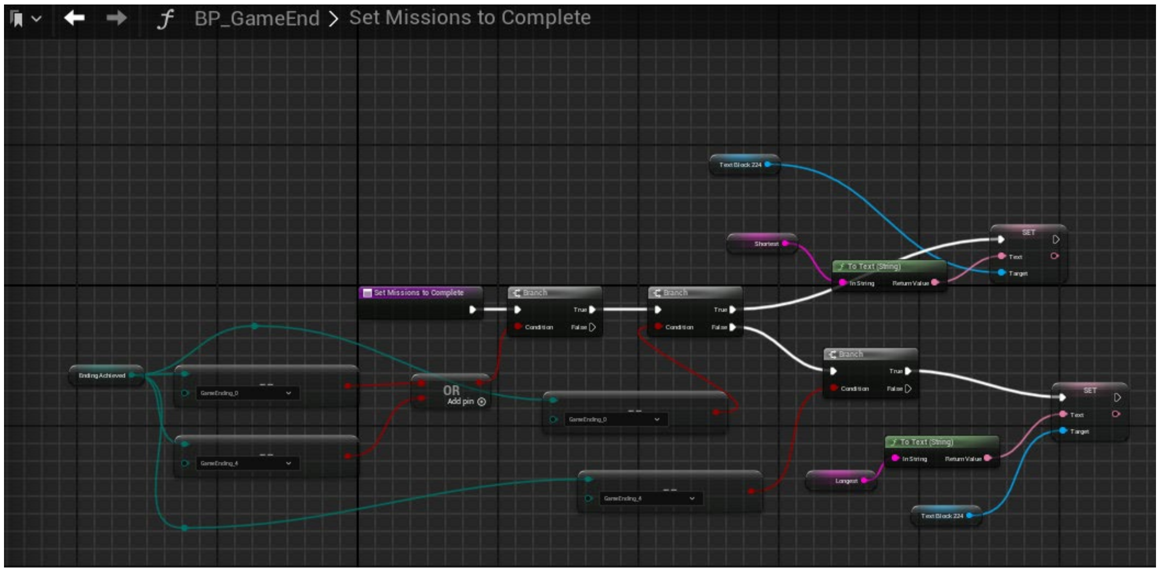
The function PathToEnding:



The function Bubble sort:



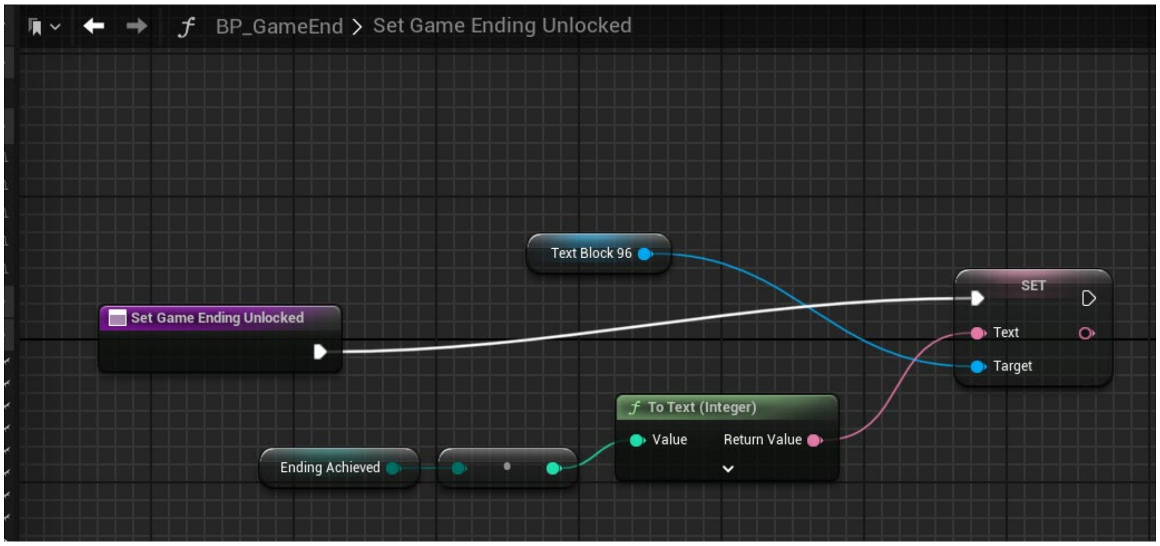
The function to show the text of each ending (short and long):



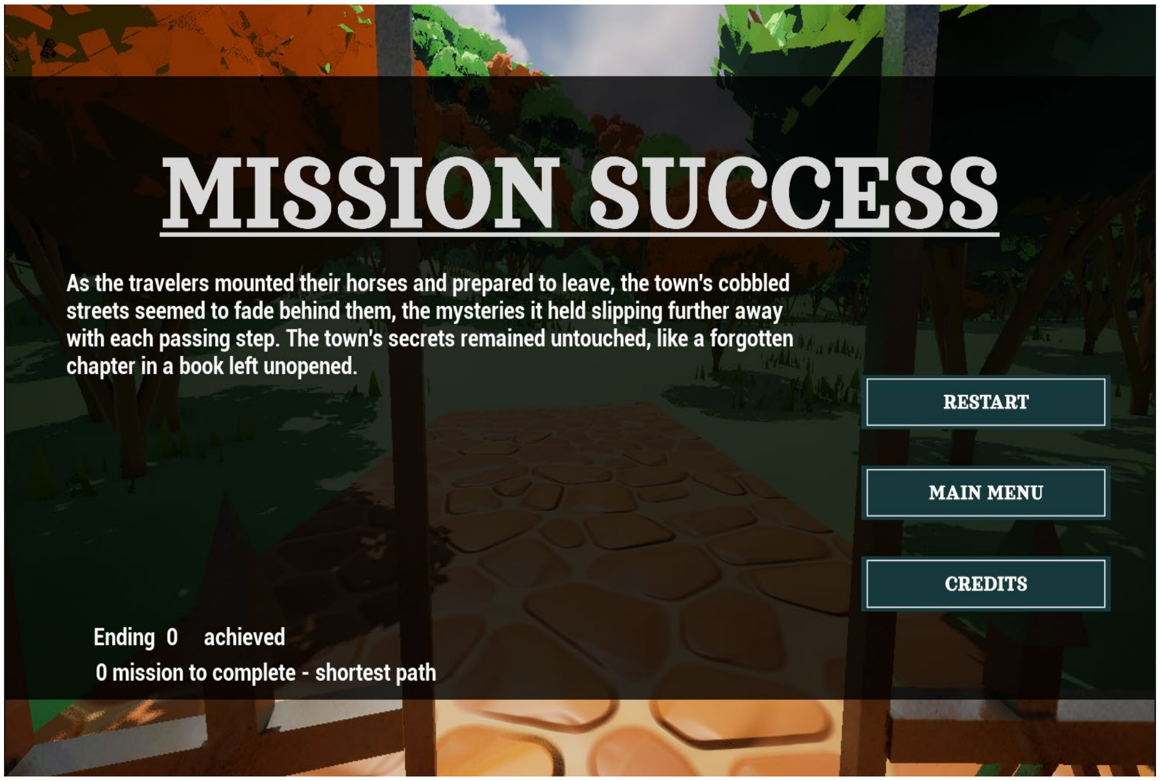
Visual representation:

As soon as the GameEnd is triggered, there will be a widget that announces the end of the game and the description of the end unlocked. However, nothing shows which ending exactly we have just unlocked. This is why I added a function in the GameEnd widget that displays the end unlocked (0 to 4).

Function that shows the ending achieved (the number):



Shortest path to ending:

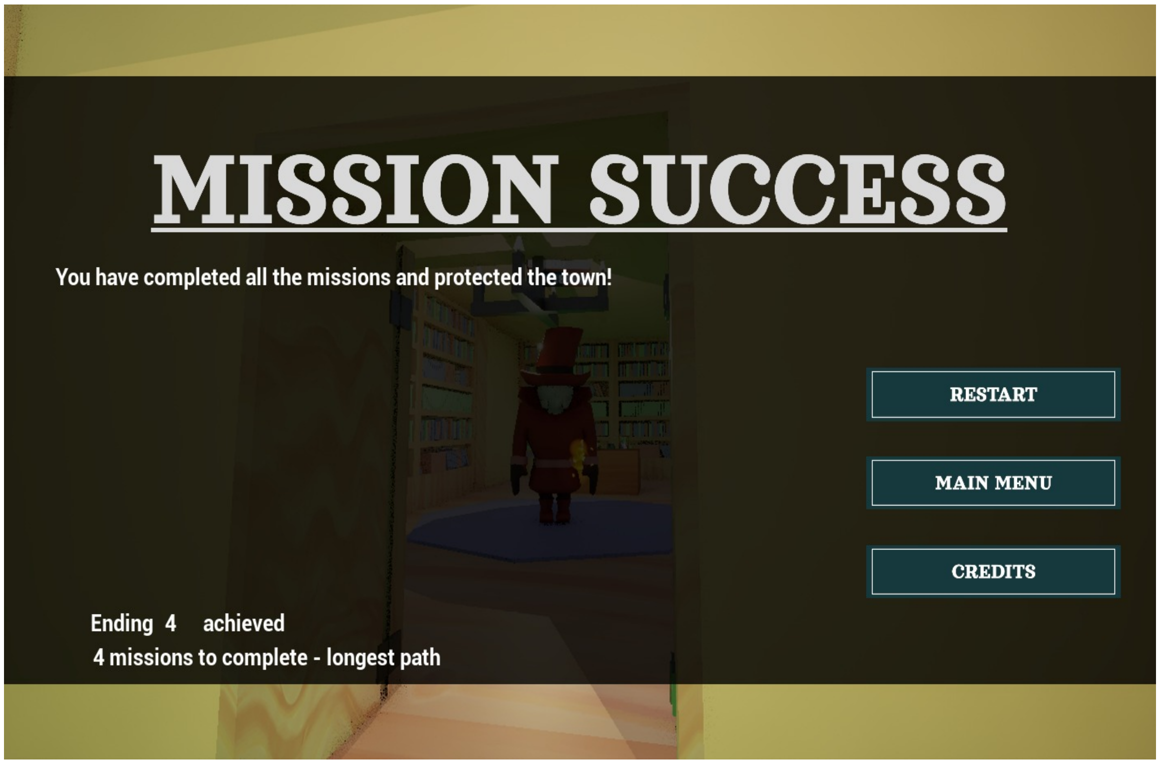

MISSION SUCCESS

As the travelers mounted their horses and prepared to leave, the town's cobbled streets seemed to fade behind them, the mysteries it held slipping further away with each passing step. The town's secrets remained untouched, like a forgotten chapter in a book left unopened.

Ending 0 achieved

0 mission to complete - shortest path

Longest path to ending:


MISSION SUCCESS

You have completed all the missions and protected the town!

Ending 4 achieved
4 missions to complete - longest path