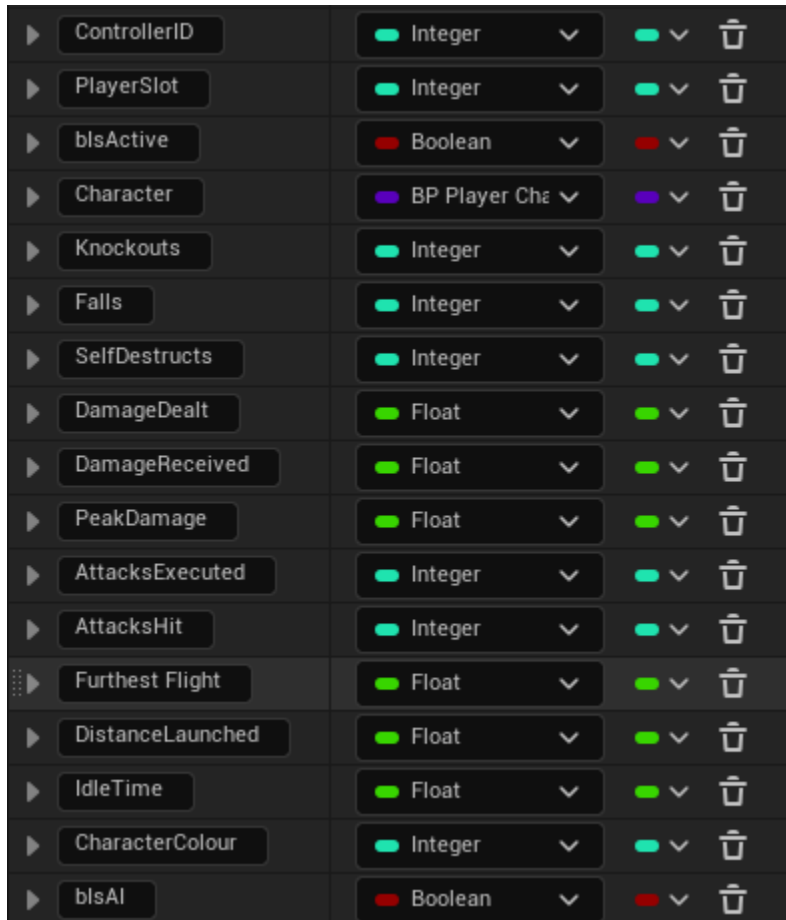


Game Manager System

This document will go over my code for the Game management system for Rift City Rebels. This system is responsible for tracking the player selections, such as which character each player selected, which stage was selected, and which game mode was selected. This system also tracks their match statistics, such as how many knockouts they received, and how much damage they received or dealt.

To track the player data, I created a struct to track this information, and made a Map in the game mode using each player's controller ID as the key, and this struct as the value.

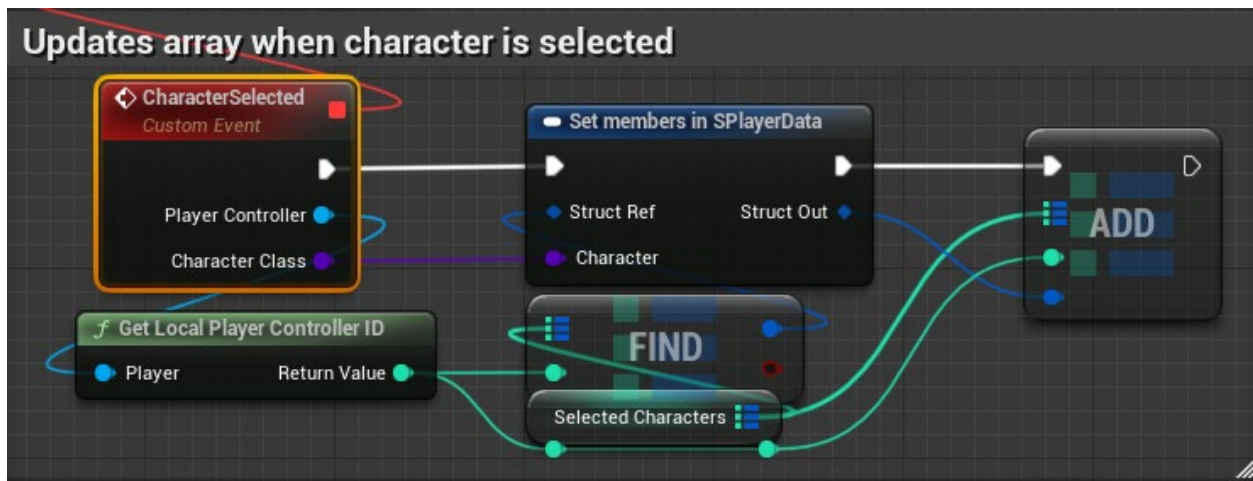


The image shows a screenshot of the Unreal Engine Variable Inspector. It displays a struct with 18 members, each with a name, a data type, and a color-coded icon. The members are: ControllerID (Integer, cyan), PlayerSlot (Integer, cyan), blsActive (Boolean, red), Character (BP Player Char, purple), Knockouts (Integer, cyan), Falls (Integer, cyan), SelfDestructs (Integer, cyan), DamageDealt (Float, green), DamageReceived (Float, green), PeakDamage (Float, green), AttacksExecuted (Integer, cyan), AttacksHit (Integer, cyan), Furthest Flight (Float, green), DistanceLaunched (Float, green), IdleTime (Float, green), CharacterColour (Integer, cyan), and blsAI (Boolean, red). Each member has a dropdown arrow and a trash icon to its right.

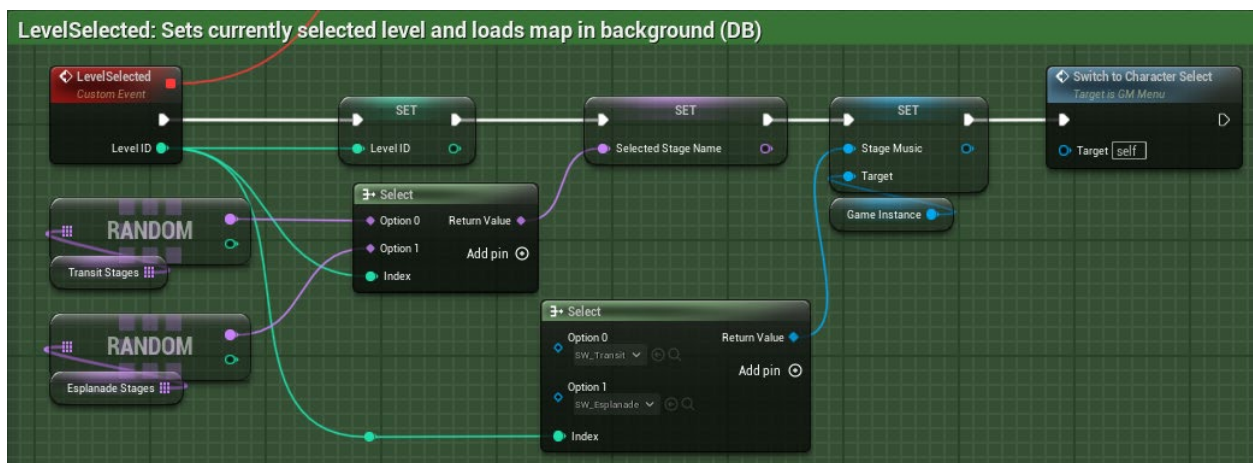
▶ ControllerID	Integer	▼	▼	🗑️
▶ PlayerSlot	Integer	▼	▼	🗑️
▶ blsActive	Boolean	▼	▼	🗑️
▶ Character	BP Player Char	▼	▼	🗑️
▶ Knockouts	Integer	▼	▼	🗑️
▶ Falls	Integer	▼	▼	🗑️
▶ SelfDestructs	Integer	▼	▼	🗑️
▶ DamageDealt	Float	▼	▼	🗑️
▶ DamageReceived	Float	▼	▼	🗑️
▶ PeakDamage	Float	▼	▼	🗑️
▶ AttacksExecuted	Integer	▼	▼	🗑️
▶ AttacksHit	Integer	▼	▼	🗑️
▶ Furthest Flight	Float	▼	▼	🗑️
▶ DistanceLaunched	Float	▼	▼	🗑️
▶ IdleTime	Float	▼	▼	🗑️
▶ CharacterColour	Integer	▼	▼	🗑️
▶ blsAI	Boolean	▼	▼	🗑️

Whenever a character selection is made an event dispatcher is called which the Game Mode subscribes to. The Game Mode then updates the struct matching that player with

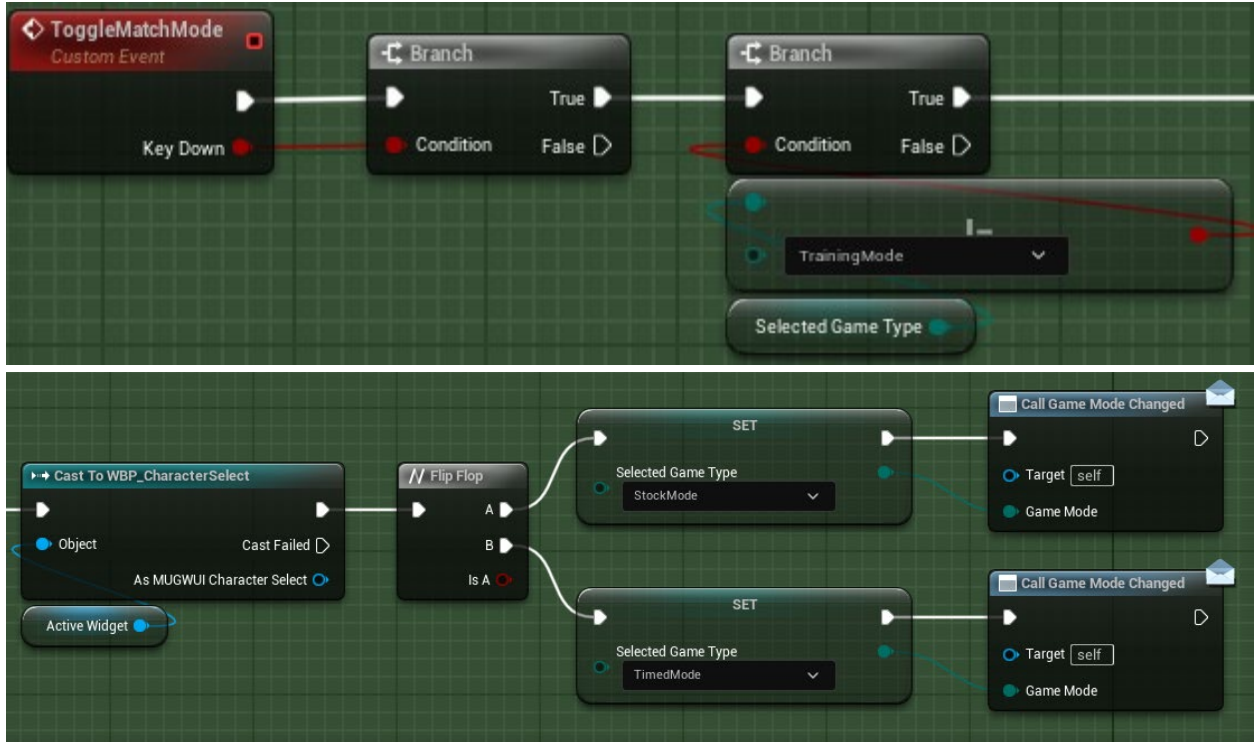
their character selection.



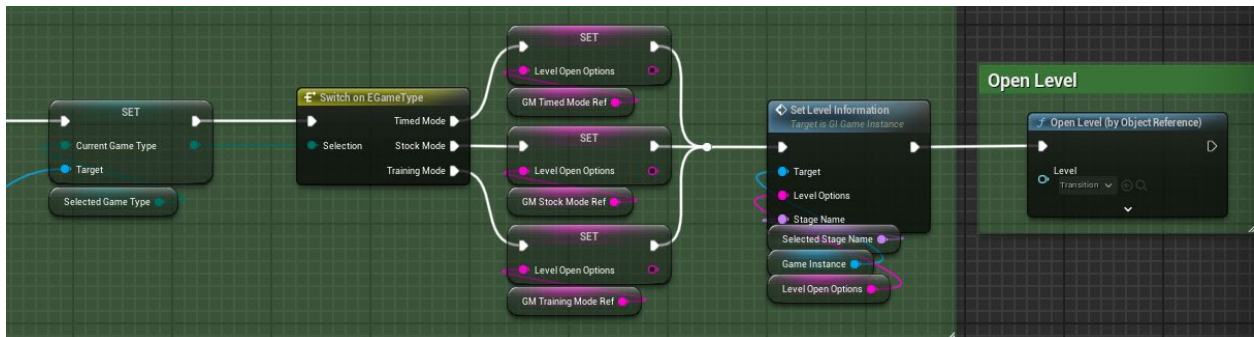
The game mode also stores a reference to the level that is selected using an Integer for the level ID. There are multiple variants of each stage using different lighting, so a random variant is selected from an array. This level reference is then used to tell the Game Instance which music track to use, and is later used to load the level.



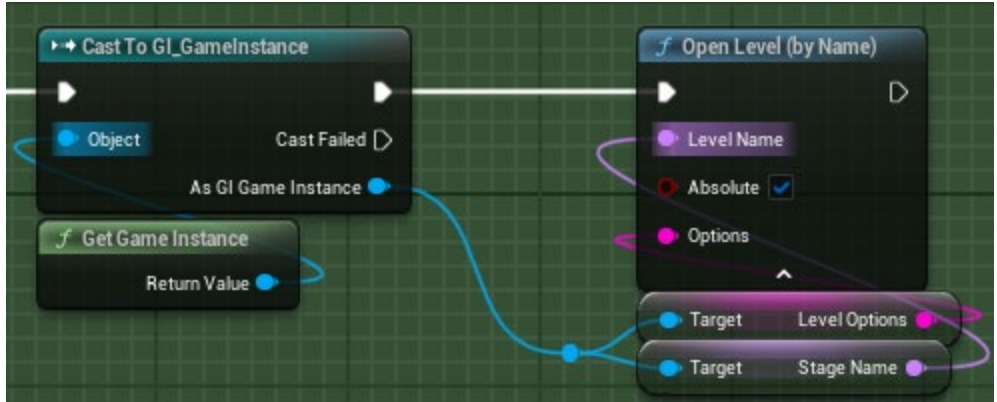
A similar approach was used to load the correct game mode. Rift City Rebels has both a timed and stock mode, both with different Game Modes to handle their unique rules. The game mode is selected using a toggle in the character select, with the default being timed mode, so a Flip Flop node is used, and an enum was created to track the different modes.



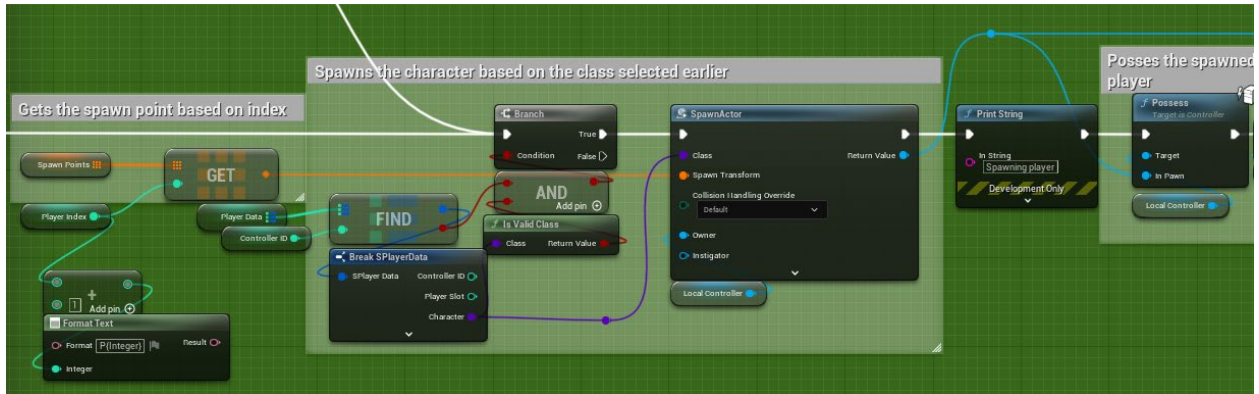
This is then used to set a string reference in the GameInstance to load the level options for the stage.



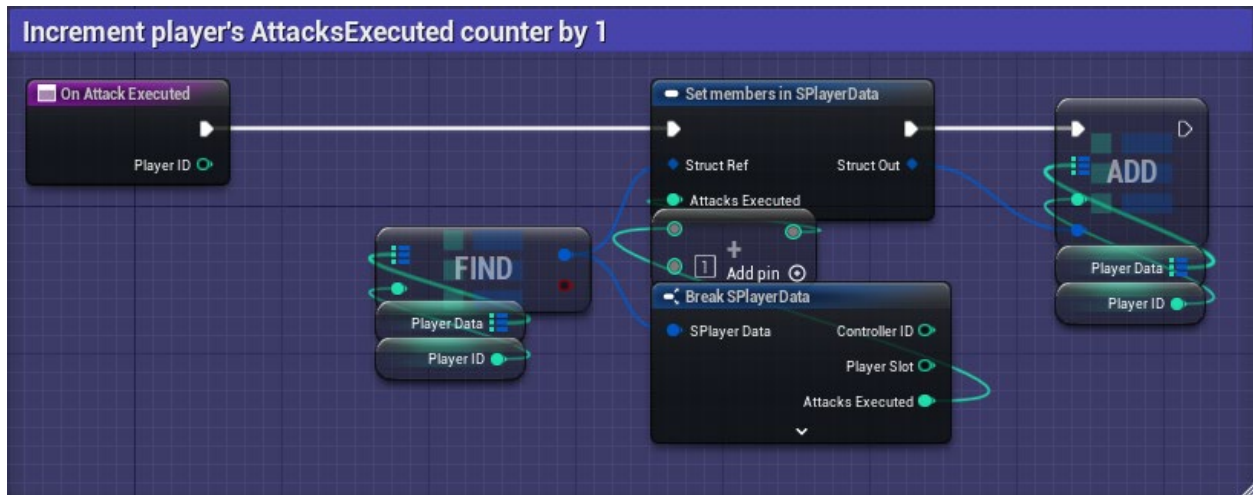
This string is retrieved by the Game Mode for the loading screen to load the stage with the correct game mode.



Once the players load into the map, the selected game mode takes. It takes the data in the map of structs from the Game Instance and uses it to spawn the players characters and assign them to the correct controller.



Throughout the match, the Player's performance stats get regularly updated for things like damage dealt, damage taken, how many times they were knocked out, how many times they knocked out others, etc. This gets updated using event dispatchers in the base player class. For the sake of brevity, I will only go over one as an example, though the others follow a similar pattern.



In this example, the *OnAttackExecuted* dispatcher was called, providing the player's ID. The manager class then searches through the Player Data to find the corresponding entry, and increments the *Attacks Executed* member. This is then read back to the Map to overwrite the previous entry with this new, updated Struct.

