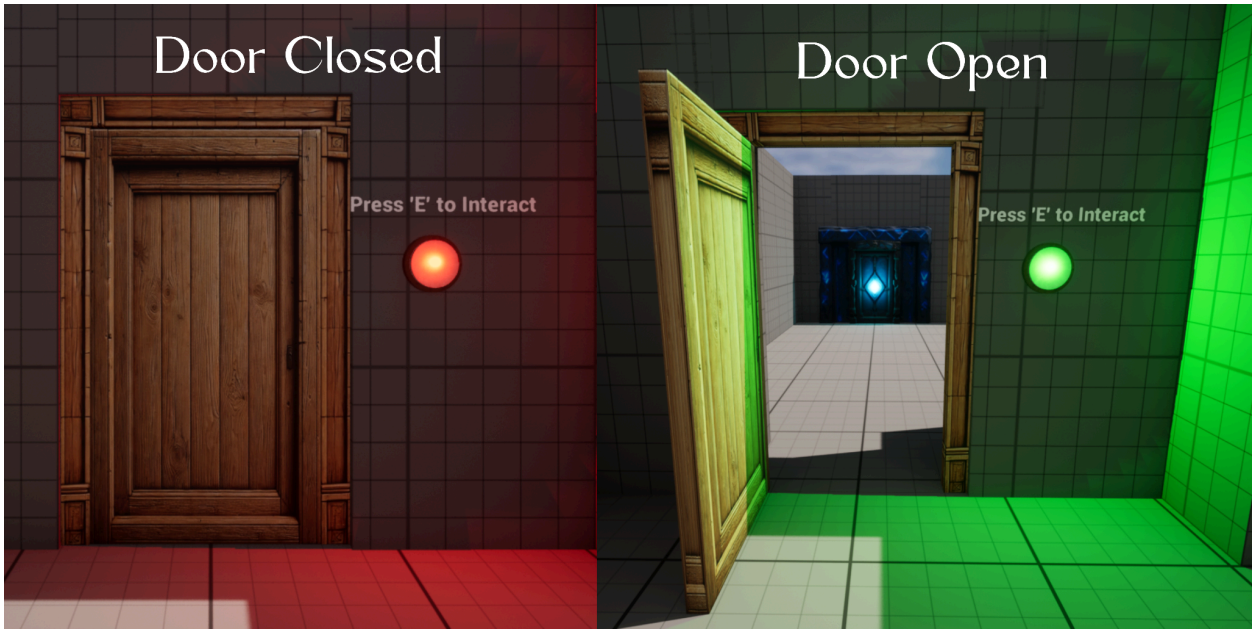
BP Buttoned Door



ViKing Production
Prepared by: Joey Vanlanduyt
Unreal Engine Version 5.1.1

Overview

This project involves the creation of a functional button-door interaction system within Unreal Engine, starting from the First-Person template. The system demonstrates a modular approach by separating the door frame and door into different Blueprints, with the door being the only movable component. Additionally, a button Blueprint interacts with the door to open or close it, providing visual and auditory feedback to the player.

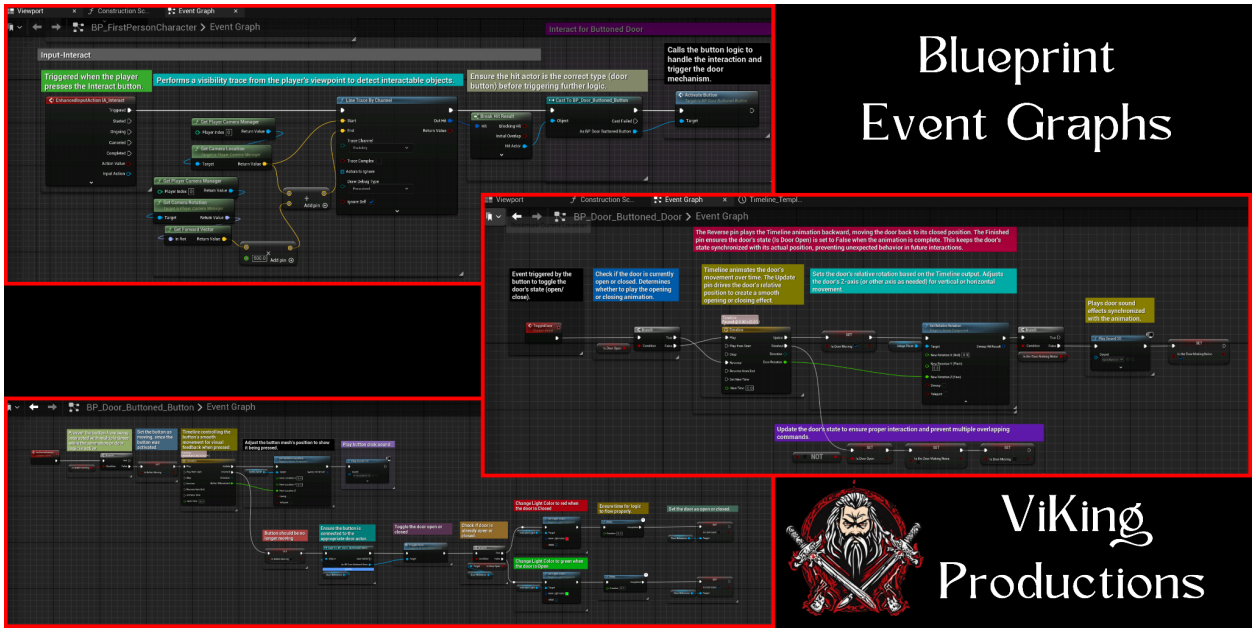
The setup utilizes block meshes for initial prototyping to lay out the environment and interactions. These were later replaced by functional Blueprints to integrate gameplay mechanics. The project showcases key Unreal Engine features, including timelines, collision detection, and modular Blueprint structures, and addresses design challenges effectively. This system is modular and scalable, making it adaptable for more complex scenarios, such as multiple buttons controlling a single door or doors requiring simultaneous button presses for activation.

Design Choices and Philosophy:

Modularity: The door and frame were designed as separate Blueprints to allow greater flexibility and reusability in different scenarios. For example, this allows swapping out the door Blueprint with minimal changes if a new door design or interaction logic is required.

Player Feedback: Visual feedback was added to the button via a light component, which switches between red and green to indicate the state of the door (closed or open). Additionally, sound effects were implemented for both the button press and door movement to enhance immersion and ensure feedback across multiple senses.

Iterative Development with Block Meshes: The initial layout was created using block meshes, a key step to test gameplay functionality before investing time in detailed modeling. This iterative approach allowed for efficient testing and refinement of the core mechanics.



Blueprint Logic

1. **Door Blueprint (BP_Door_Buttoned_Door)**

   The Door Blueprint (BP_Door_Buttoned_Door) handles all movement and state management for the door. This Blueprint ensures that the door opens and closes smoothly using a Timeline node to adjust its hinge's rotation over time. Additionally, it uses boolean variables like Is Door Open and Is Door Moving to prevent overlapping interactions

   Custom Event: ToggleDoor:

   - Triggered when the linked button is pressed.
   - Checks the current state of the door (Is Door Open boolean):
     - If True: Plays the Timeline in reverse to close the door.
     - If False: Plays the Timeline forward to open the door.

   Timeline:

   - Drives the door's rotation smoothly over time.
   - The rotation is applied to the door's hinge pivot to simulate realistic movement.

   State Management:

   - Updates Is Door Open and Is Door Moving booleans after the animation completes to prevent overlapping interactions.

2. **Button Blueprint (BP_Door_Buttoned_Button)**

The Button Blueprint (BP_Door_Buttoned_Button) manages the interaction between the player and the door. It uses a custom event, ActivateButton, which is triggered when the player interacts with the button via a line trace. The button includes a Timeline node to animate its movement when pressed, creating a realistic button press effect. Boolean variables such as Is Button Moving ensure that the button cannot be pressed repeatedly before completing its animation. A light component is used to provide visual feedback by changing its color to red or green, indicating the door's current state (closed or open). The button is linked to its specific door through a Door Reference variable, ensuring it controls the correct door in the scene.

Interaction Logic:

- Triggered by the player's line trace when the Interact button is pressed.
- Contains a Door Reference variable to link the button to a specific door Blueprint.

Button Movement:

- Uses a Timeline to animate the button press for visual feedback.
- Changes the button mesh’s position to simulate being pressed.

Indicator Lights:

- Updates the light color based on the door's state (linked via the Door Reference).

The use of indicator lights and synchronized sound effects enhances the player experience by providing immediate feedback about the door's state. These design choices aim to improve usability, ensuring the player understands the outcome of their interaction without ambiguity

3. **First-Person Character Blueprint**

The First-Person Character Blueprint (BP_FirstPersonCharacter) enables the player to interact with the button using an input action mapped to IA_Interact. This Blueprint uses a Line Trace by Channel to detect objects directly in front of the player within a certain range. When the line trace hits a button, it triggers the ActivateButton event within the detected Button Blueprint, initiating the interaction sequence. The line trace logic ensures precise targeting of interactable objects.This Blueprint is essential for integrating player actions into the button-door interaction system.

Enhanced Input Action: IA_Interact:

- Detects when the player presses the interact button.
- Executes a line trace from the camera to detect hit objects within range.
- Casts the hit object to BP_Door_Buttoned_Button to ensure interaction is only possible with buttons.
- Calls the ActivateButton event on the button Blueprint.

Challenges and Solutions

1. **Button Only Working Once**

   Issue: Initial logic did not properly reset the Is Button Moving variable, preventing the button from being pressed multiple times.

   Solution: Added logic to reset the Is Button Moving variable once the button animation completes, allowing the button to be pressed again.

2. **Door Not Opening or Closing Properly**

   Issue: The door would sometimes require two button presses to toggle.

   Solution: Refactored the state management for Is Door Moving and Is Door Open to ensure consistent state changes and avoid overlapping animations.

3. **Sound Overlapping**

   Issue:nDoor sounds would stack and play multiple times.

   Solution: Added a boolean variable (Is Door Making Noise) to ensure sounds only play when appropriate.

4. **Line Trace Detection**

   Issue: Line trace initially had difficulty detecting the button reliably.

   Solution: Adjusted the line trace distance and ensured that all interactive objects were set to the correct collision channel.

5. **Delayed Light Updates**

   Issue: The indicator lights would not update immediately after the door's state changed.

   Solution: Used a small delay (0.3 seconds) before updating the light color to match the door's animation timing.

Debugging tools, such as Print String nodes and visual line trace debugging, were instrumental in identifying and resolving issues like line trace detection, state variable mismatches, and sound stacking. For instance, Print String was used to confirm whether the Is Button Moving variable was properly resetting at the end of the button's animation.

Final Implementation

- **The system now works seamlessly:**

  The player presses a button to toggle the door's state.

  Smooth animations and sounds play in sync for both the door and button.

  Indicator lights provide immediate visual feedback for the door's state.

- **This implementation demonstrates a modular and reusable design:**

  Doors and buttons are separate Blueprints, allowing for flexibility in level design.

  Multiple buttons and doors can be placed in the level, each configured independently.

Key Learnings

- **Blueprint Modularization**:

  Separating the door and frame into distinct Blueprints allows for easy updates and customization.

- **State Management**:

  Properly managing state variables (Is Door Open, Is Button Moving, etc.) is crucial to prevent unintended behavior.

- **Prototyping**:

  Using block meshes for initial layout simplifies iteration and testing before adding functionality.

- **Debugging Tools**:

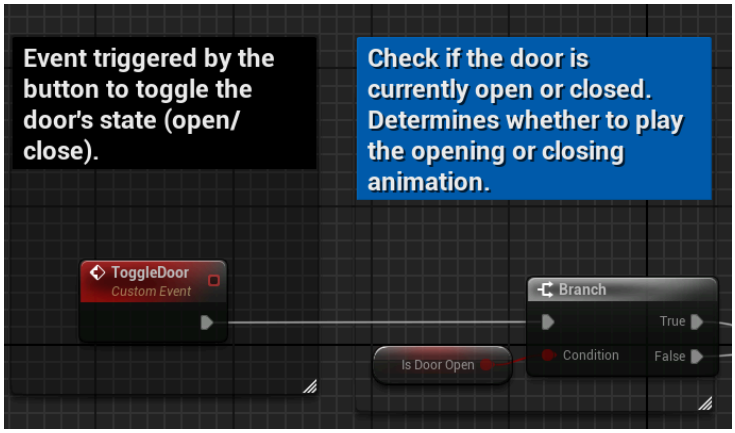
  Debugging (e.g., using Print String and visualizing line traces) was essential for identifying and resolving issues.

Specific Blueprint Information

Door Blueprint (BP_Door_Buttoned_Door)

The Door Blueprint handles the animation and state management for the door. It uses a Timeline to smoothly animate the door's movement and ensures proper synchronization between its state variables.

Step 1: **Event: ToggleDoor**

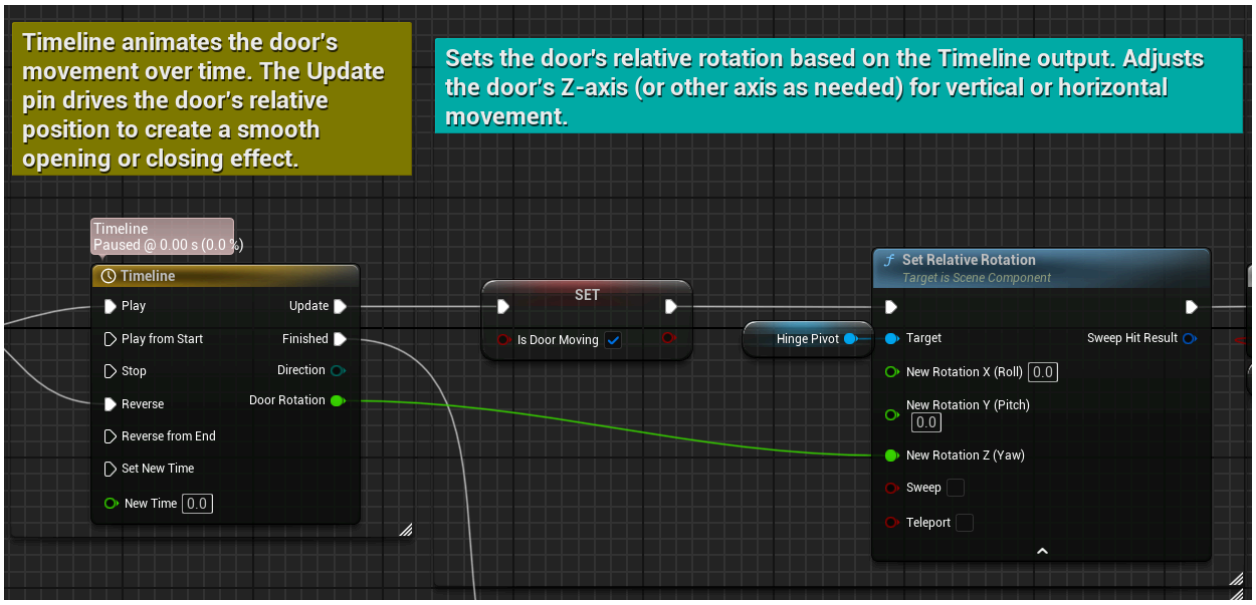


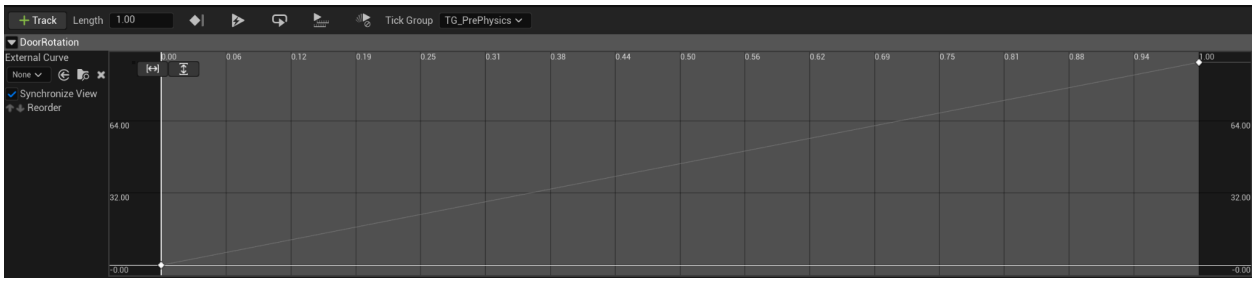
This event is called whenever the associated button triggers an interaction.

A **Branch Node** checks the Is Door Open boolean:

- If True, the Timeline is played in reverse, closing the door.
- If False, the Timeline is played forward, opening the door.

Step 2: Timeline Animation



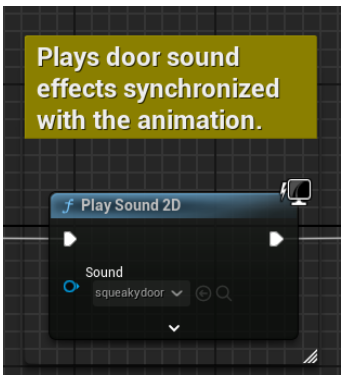


The Timeline node drives the rotation of the door's hinge pivot over time.

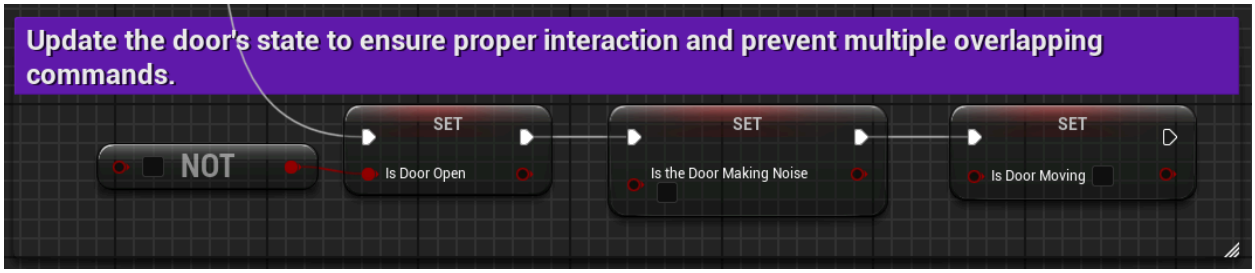
The output of the Timeline is connected to a **Set Relative Rotation Node**, which adjusts the door's rotation along the Z-axis to simulate opening or closing.

Step 3: **Sound and State Management**

A **Play Sound 2D Node** plays the door's movement sound (e.g., creaking).

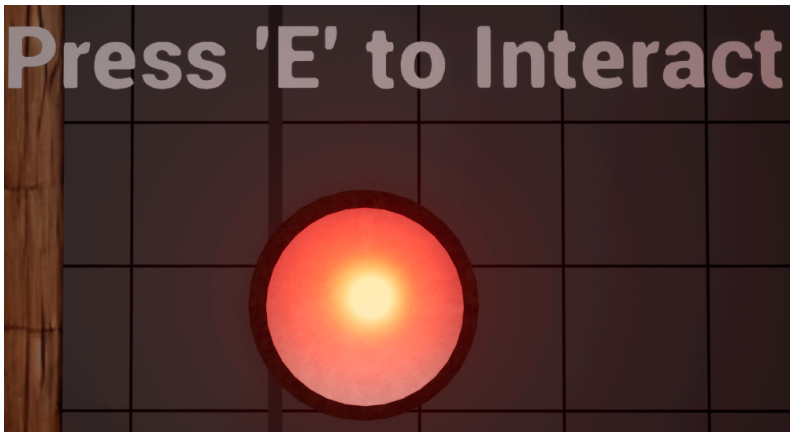


After the Timeline finishes, the Is Door Open boolean is updated to reflect the door's new state.



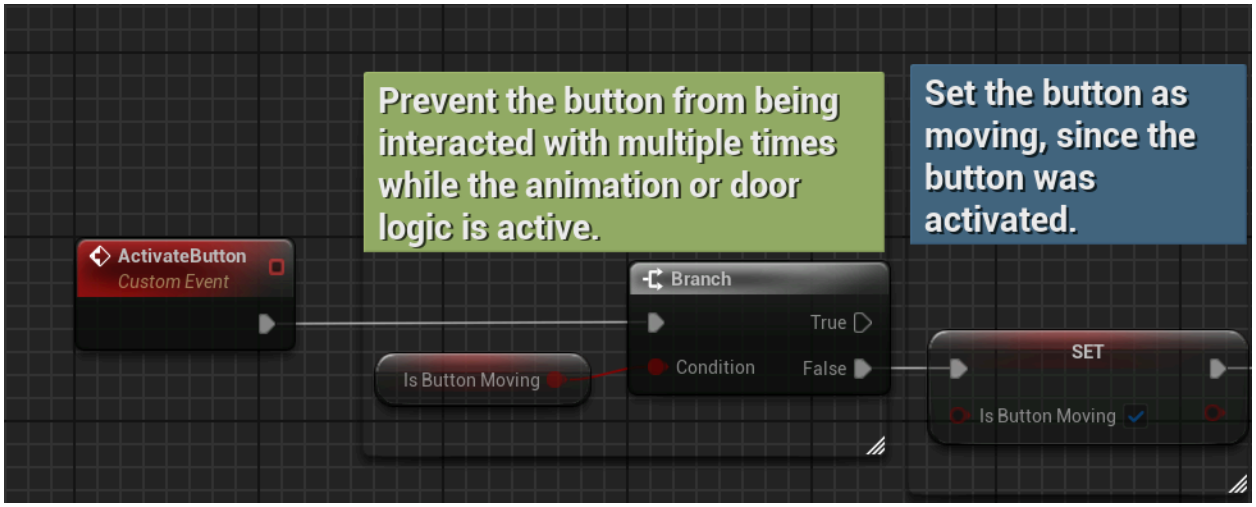
The Is Door Moving boolean ensures that no other interactions can occur while the door is in motion, preventing overlapping animations or sounds.

2. Button Blueprint (BP_Door_Buttoned_Button)



The Button Blueprint manages player interaction and provides visual and audio feedback for pressing the button.

Step 1: **Custom Event: ActivateButton**

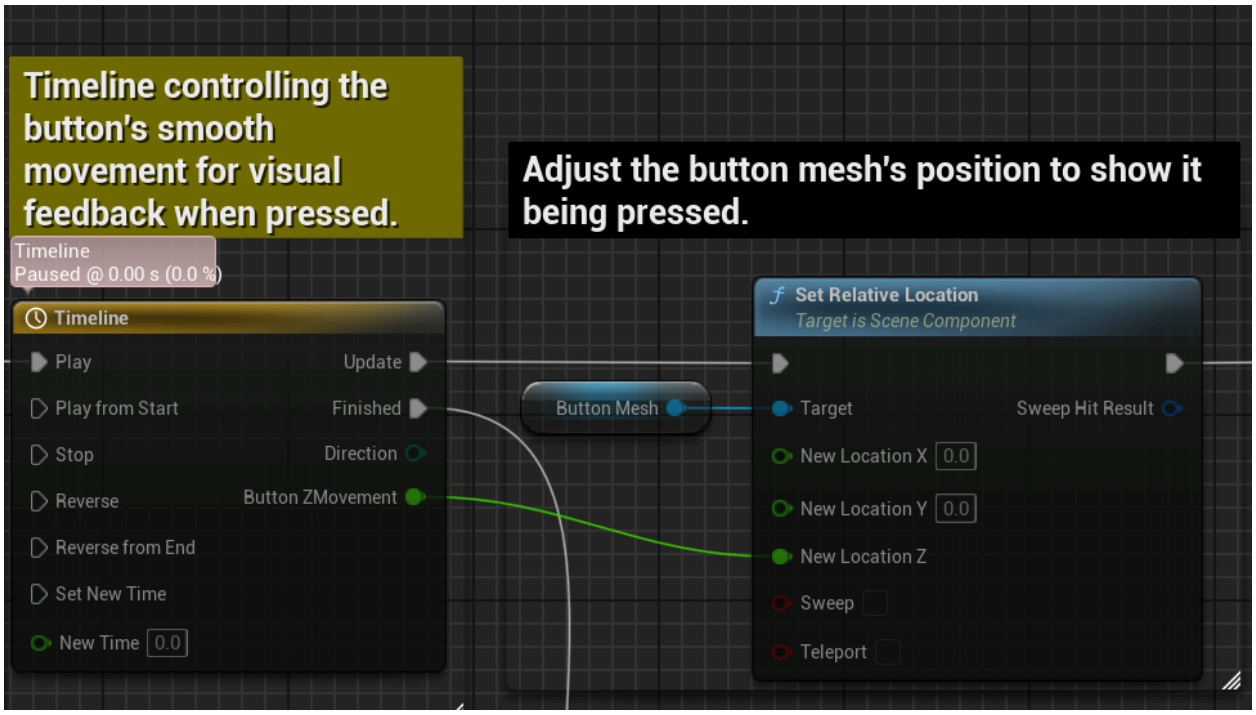


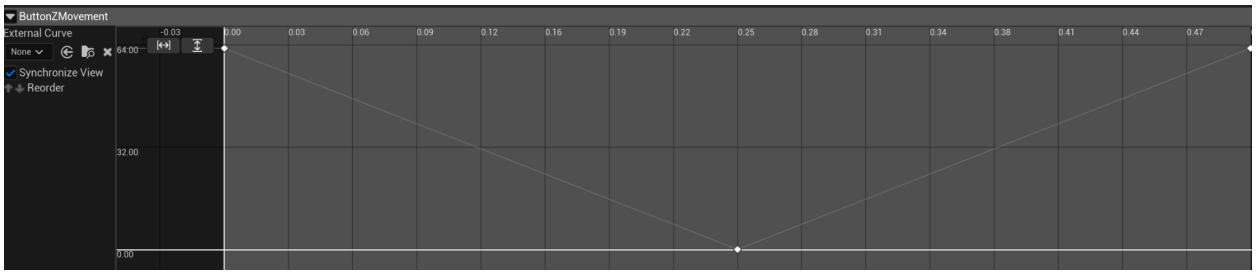
Triggered by the First-Person Character's IA_Interact action.

A **Branch Node** ensures the button isn't already moving by checking the Is Button Moving boolean. If False, the interaction proceeds.

The Is Button Moving variable is set to True to prevent re-triggering during the current interaction.

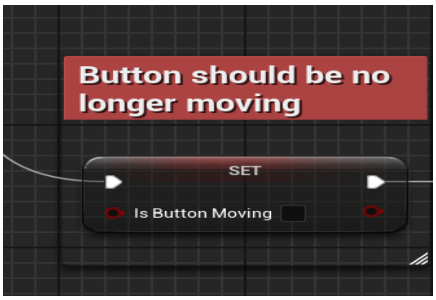
Step 2: **Button Animation**



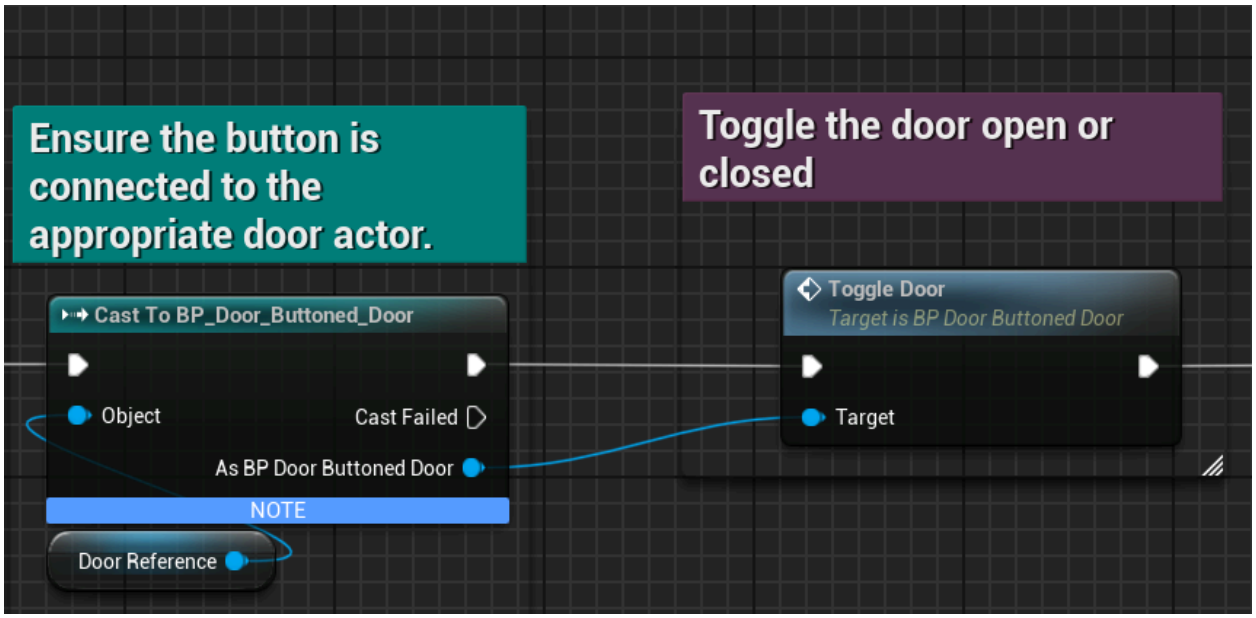


A Timeline node animates the button being pressed by adjusting its Z-location with a **Set Relative Location Node**.

The Finished output of the Timeline resets the Is Button Moving boolean, allowing the button to be pressed again.



Step 3: **Door Reference and Interaction**

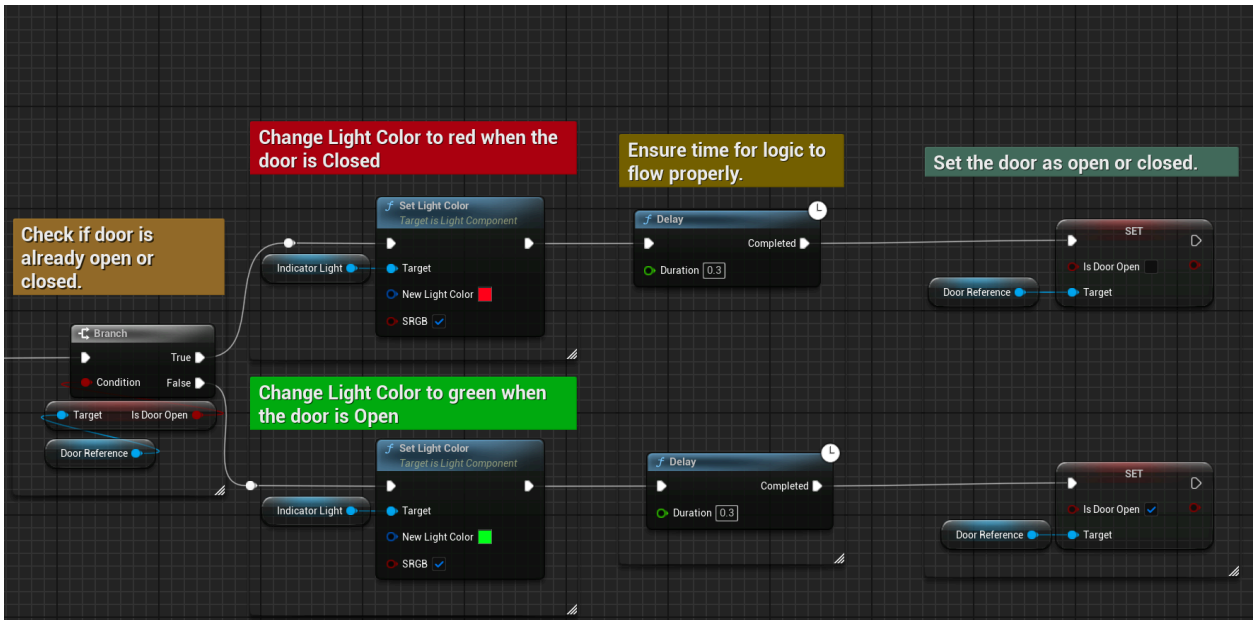


The Button Blueprint contains a Door Reference variable, which links the button to its corresponding Door Blueprint.

A **Cast to BP_Door_Buttoned_Door Node** validates that the linked door is correct and calls the ToggleDoor event on the Door Blueprint.

\

Step 4: Indicator Lights



The button includes a light component that changes color to reflect the door's state:

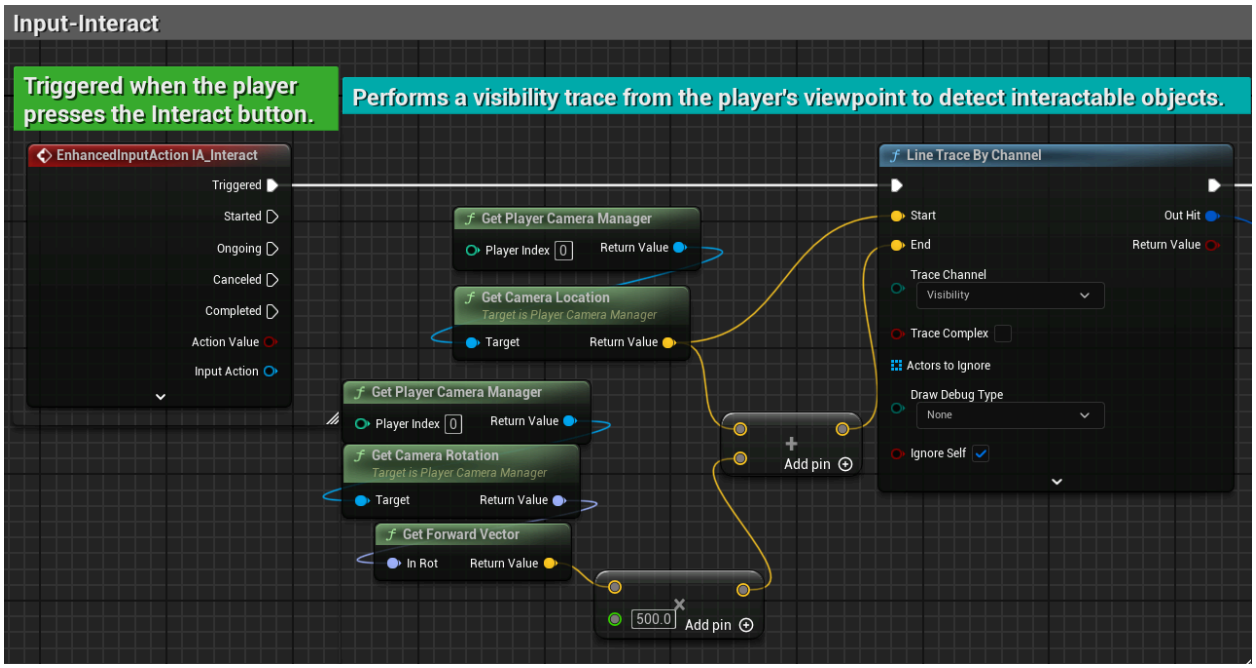
- **Red Light**: Indicates the door is closed.
- **Green Light**: Indicates the door is open.

A **Branch Node** checks the door's state via the Door Reference and updates the light color accordingly.

3. First-Person Character Blueprint

The First-Person Character Blueprint handles player interaction with the button through a line trace system.

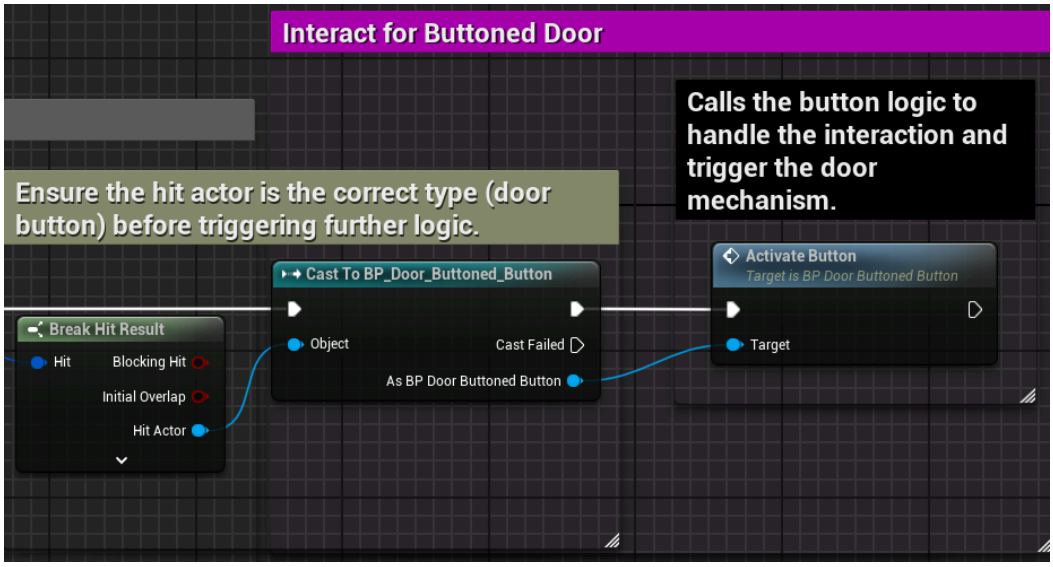
Step 1: **Input Action: IA_Interact**



When the player presses the interact key (e.g., “E”), the IA_Interact action is triggered.

A **Line Trace by Channel Node** performs a line trace from the player’s camera forward, checking for interactive objects within a specified distance.

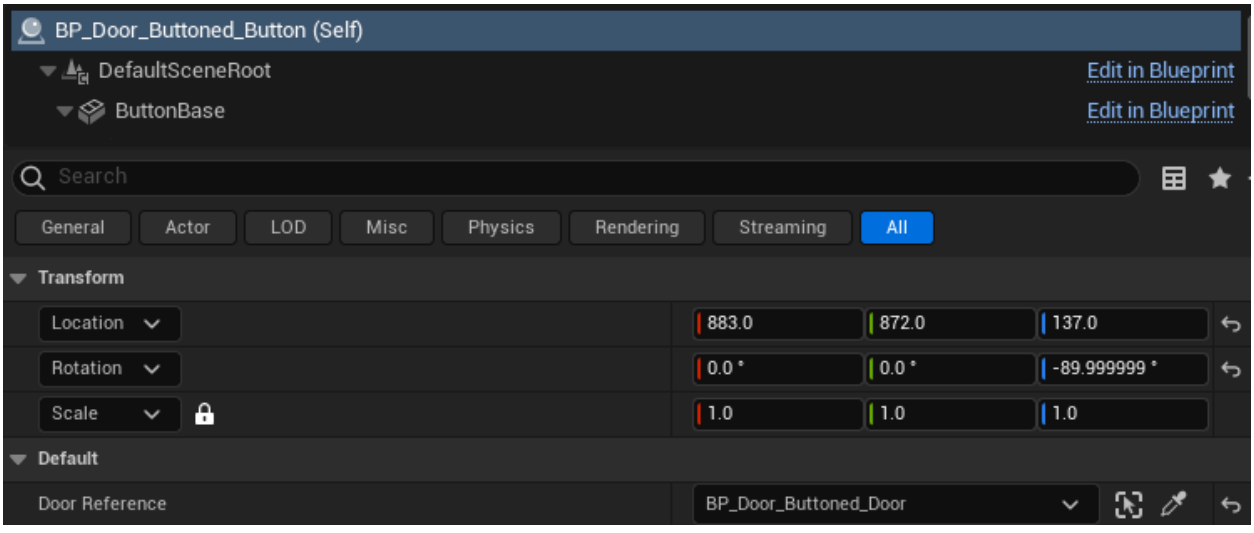
Step 2: **Object Detection and Casting**



The line trace checks for hits on objects.

A **Cast to BP_Door_Buttoned_Button Node** validates whether the hit object is a button.

If the cast is successful, the button's ActivateButton event is called.



Lastly, within the **Details** panel of the selected **BP_Door_Buttoned_Button** instance, ensure the correct door is assigned in the **Door Reference** property under the **Default** category. This links the button to its corresponding door, allowing the interaction to function as intended.

Additional Design Elements

Block Mesh Layout:

Block meshes were used during the prototyping phase to layout the button, door, and frame.

These placeholders allowed for testing functionality and interaction before replacing them with the final Blueprints.

Using block meshes for prototyping allowed rapid testing of interaction placement and functionality before committing to detailed assets. This approach minimized wasted effort and enabled iterative adjustments to the door and button setup without distractions from visuals.

Sound Management:

Door sounds were limited using a boolean (Is Door Making Noise) to prevent stacking or overlapping audio during rapid interactions.

Final Result

This project showcases a robust and modular approach to implementing a button-door interaction system in Unreal Engine. From initial block mesh prototyping to the final Blueprint implementation, the design prioritizes usability, scalability, and player feedback. Challenges encountered during development were resolved methodically, demonstrating a solid understanding of Unreal Engine's features and problem-solving skills.

This modular design can easily be expanded to support more complex scenarios. For example, multiple buttons could control a single door for cooperative puzzles, or a single button could control multiple doors for timed challenges. The separation of the door and button into distinct Blueprints ensures that this system can be adapted without significant rework.