**BLUEPRINTS**

**Gameplay Mechanics:**

Clearly define and plan the core gameplay mechanics. This includes player controls, interaction systems, and the overall flow of the game.

**User Interface (UI):**

Design and implement a user interface that is intuitive and enhances the player's experience. Consider elements like health bars, inventory systems, and on-screen prompts.

**Character Blueprint:**

Create blueprints for player characters and non-player characters (NPCs). Define their behavior, animations, and interactions within the game world.

**AI Blueprint:**

Develop blueprints for artificial intelligence (AI) to control NPC behaviour. This includes pathfinding, decision-making, and responses to player actions.

**Level Design Blueprint:**

Plan the blueprints for your game levels. This involves designing the layout, placing interactive elements, and defining triggers for events.

**Quest and Dialogue System:**

Implement blueprints for quests and dialogues. Design a system that allows for engaging storytelling, interactive conversations, and quest progression.

**Inventory System:**

Create a blueprint for the inventory system. Define how players collect, manage, and use items throughout the game.

**Save and Load System:**

Implement a blueprint for saving and loading game progress. Ensure that players can continue their journey seamlessly between gaming sessions.

**Audio Blueprint:**

Plan for audio cues and sound triggers using blueprints. This includes environmental sounds, character voices, and music changes based on the game's events.

**Dynamic Weather and Time-of-Day:**

If applicable, design blueprints for dynamic weather and time-of-day systems. This adds realism and variation to the game environment.

**Performance Optimization:**

Consider performance optimization in your blueprints. Unreal Engine 5 provides tools for optimization, such as level of detail (LOD) settings and efficient use of resources.

**Testing and Iteration:**

Regularly test and iterate on your blueprints. Get feedback from playtesting to refine and improve the gameplay experience.

**Documentation:**

Document your blueprints thoroughly. This includes comments within the blueprint graphs, external documentation for collaborators, and guidelines for potential future developers.

# **USER INTERFACE (UI)**

## MAIN MENU

1. Create the Main Menu Widget Blueprint:

a. Create the Main Menu Widget:

Right-click in the Content Browser.

Choose "User Interface" > "Widget Blueprint."

Name it something like "WBP\_MainMenu."

Double-click on "WBP\_MainMenu" to open the Widget Blueprint editor.

b. Design the Main Menu:

In the Widget Blueprint editor, you'll see the Palette on the left and the Details panel on the right.

Drag and drop buttons from the Palette for "Start Game," "Load Game," "Options," "Credits," and "Quit Game."

To add buttons, drag a "Button" from the Palette onto the canvas.

To add text to a button, select the button, go to the Details panel, and find the "Content" section.

2. Create Submenu Widget Blueprints:

a. Create Submenu Widget Blueprints:

Repeat the process for creating Widget Blueprints for each submenu (Graphics Settings, Audio Settings, Controls).

Name them accordingly, like "WBP\_GraphicsSettings," "WBP\_AudioSettings," and "WBP\_Controls."

Double-click on each to open their respective Widget Blueprint editors.

b. Design Submenus:

Design each submenu layout with relevant settings or information.

You can use buttons, sliders, or other UI elements from the Palette.

3. Open Level Blueprint:

a. Open the Level Blueprint:

In the main toolbar, go to "Blueprints" > "Open Level Blueprint."

4. Add Functions for Opening Submenus:

a. Add Functions:

In the Level Blueprint editor, right-click in an empty space.

Choose "Function" and name it something like "OpenGraphicsSettings."

Repeat this for other submenus.

b. Create Widget Nodes:

Inside each function, drag off the execution pin and add a "Create Widget" node.

Select the corresponding submenu Widget Blueprint for each "Create Widget" node.

Connect the "Create Widget" node to "Add to Viewport."

c. Bind Buttons to Functions:

Go back to "WBP\_MainMenu."

Select each button, go to the Details panel, find the "On Clicked" event, and connect it to the corresponding function in the Level Blueprint.

5. Add Functionality for Start Game, Load Game, Credits, and Quit Game:

a. Add Start Game Function:

Create a function in the Level Blueprint for starting the game.

Implement the necessary logic for starting or loading the game.

………………….

Implementing the Start Game Logic:

1. Open the Level Blueprint:

Go to "Blueprints" > "Open Level Blueprint."

2. Create a New Function:

Right-click in the graph, choose "Function," and name it something like "StartGame."

3. Add Logic to Start the Game:

Inside the "StartGame" function, you'll define the logic to initiate the game. This can include actions like:

Loading the next level.

Setting up player characters.

Initializing game state variables.

Example Blueprint Nodes:

You might use nodes like "Open Level" to load a new level.

You can also set up any necessary variables or conditions for your game's starting state.

4. Bind Button to StartGame Function:

Go back to your main menu widget (WBP\_MainMenu).

For the "Start Game" button, find the "On Clicked" event in the Details panel.

Connect this event to the "StartGame" function in the Level Blueprint.

5. Testing:

Press the "Play" button to test if clicking "Start Game" initiates the logic you've implemented.

Implementing the Load Game Logic:

1. Create Load Game Function:

In the Level Blueprint, create another function named "LoadGame."

2. Add Logic to Load the Game:

Inside the "LoadGame" function, define the logic to load a saved game. This might include:

Retrieving saved data.

Loading the appropriate level or state.

Initializing game variables based on the saved data.

Example Blueprint Nodes:

You might use nodes like "Load Game from Slot" to load saved data.

Set up the necessary variables to restore the game state.

3. Bind Button to LoadGame Function:

Go back to your main menu widget (WBP\_MainMenu).

For the "Load Game" button, find the "On Clicked" event in the Details panel.

Connect this event to the "LoadGame" function in the Level Blueprint.

4. Testing:

Press the "Play" button to test if clicking "Load Game" initiates the logic you've implemented.

Additional Tips:

Use nodes like "Save Game to Slot" when saving the game.

Make sure to handle error checking and edge cases in your logic, such as checking if there's a saved game to load.

Consider creating separate Blueprint classes for game state management or utilizing a Game Instance for persistent data.

…………………

Connecting Event to Function in Level Blueprint:

1. Open WBP\_MainMenu Widget Blueprint:

Double-click on "WBP\_MainMenu" in the Content Browser to open the Widget Blueprint editor.

2. Find the "Start Game" Button:

In the Widget Blueprint editor, find the "Start Game" button on the canvas.

3. Select the Button:

Left-click on the "Start Game" button to select it.

4. Go to the Details Panel:

On the right side of the screen, find the "Details" panel. If it's not visible, go to "Window" and make sure "Details" is checked.

5. Find the "On Clicked" Event:

In the Details panel, scroll down until you find the "Events" section.

Look for the "On Clicked" event.

6. Bind the Event to Function:

Click on the empty box next to "On Clicked" to open the Graph for this event.

7. Connect to the "StartGame" Function:

Inside the graph for the "On Clicked" event, you'll see nodes representing the actions to be performed when the button is clicked.

Drag off from the execution pin (small white triangle) of the "On Clicked" node.

Begin typing "StartGame" in the search bar and select the "StartGame" function from the Level Blueprint.

8. Compile and Save:

Once you've connected the "On Clicked" event to the "StartGame" function, click the "Compile" button at the top of the Blueprint editor.

Then, click the "Save" button.

9. Testing:

Press the "Play" button in the editor to test your main menu.

Clicking the "Start Game" button should now trigger the logic inside the "StartGame" function in the Level Blueprint.

Additional Tips:

Make sure the "StartGame" function has been defined in the Level Blueprint before attempting to connect it.

If the Level Blueprint is not open, go to "Blueprints" > "Open Level Blueprint" and ensure you're working in the correct Level Blueprint.

………………..

b. Add Credits Function:

Create a function in the Level Blueprint for opening the Credits screen.

Implement the logic to show the Credits submenu.

c. Add Quit Game Function:

Create a function in the Level Blueprint for quitting the game.

Implement the logic to quit the game.

6. Create Options Submenu:

a. Design Options Submenu:

Open "WBP\_Options" and design it with buttons for Graphics Settings, Audio Settings, and Controls.

Add a "Back" button.

b. Bind Buttons to Functions:

Bind the buttons in "WBP\_Options" to the corresponding functions in the Level Blueprint.

Create functions in the Level Blueprint for each option (e.g., "OpenGraphicsSettings").

7. Implement Graphics, Audio, and Controls Submenus:

a. Open Graphics, Audio, and Controls Widgets:

Open "WBP\_GraphicsSettings," "WBP\_AudioSettings," and "WBP\_Controls."

Design each submenu with relevant settings.

Add a "Back" button to each.

b. Bind Back Button:

In each submenu, bind the "Back" button to a function that closes the current submenu.

Create a function in the Level Blueprint to handle going back.

8. Test in the Editor:

a. Playtest:

Press the "Play" button and test the interactions between the main menu and submenus.

START GAME BUTTON:

**Step 1: Open Your Project in Unreal Engine 5**

* Open Unreal Engine 5.
* Load your existing project or create a new one.

**Step 2: Create a Main Menu Widget Blueprint**

* Right-click in the Content Browser.
* Choose "User Interface" > "Widget Blueprint."
* Name it "WBP\_MainMenu."
* Double-click to open the widget.

**Step 3: Design the Main Menu UI**

* Add a button for "Start Game" to the main menu.
* Design any additional UI elements you want in the main menu.

**Step 4: Set Up Widget Graph for Interaction**

* Open the Widget Blueprint Graph.
* Create an event for the "Start Game" button.
* Add nodes to execute actions when the button is clicked.

**Step 5: Implement the "Start Game" Button**

* Drag off the "On Clicked" event of the button.
* Add a "Open Level" node.
* Connect the "Open Level" node to the execution pin of the "On Clicked" event.
* In the "Open Level" node, type the name of the level/map you want to open when the game starts.

**Step 6: Test Your Main Menu**

* Press the "Play" button to test your Main Menu widget in the editor.
* Click the "Start Game" button to verify that it opens the designated level.

**Tips:**

* Ensure the "Start Game" button is visually appealing and stands out.
* Use clear labeling on the button to guide the player.
* Consider adding visual feedback, like hover effects or animations, to make the button interactive.

## **OPTIONS MENU**

**Step 1: Open Your Project in Unreal Engine 5**

* Open Unreal Engine 5.
* Load your existing project or create a new one.

**Step 2: Create Main Menu Widget Blueprint**

* Right-click in the Content Browser.
* Choose "User Interface" > "Widget Blueprint."
* Name it "WBP\_MainMenu."
* Double-click to open the widget.

**Step 3: Design the Main Menu UI**

* Add a button labeled "Options."
* Create three variables (GraphicsSettingsWidget, AudioSettingsWidget, ControlsSettingsWidget).
* Add event handlers for the "Options" button.

**Step 4: Set Up Widget Graph for Options Button**

* Open the Widget Blueprint Graph.
  1. On the "Options" button click event:
  2. Create a custom event (e.g., ShowOptionsPage).
  3. Create a switch node to decide which options page to display.

**Step 5: Create Graphics Settings Widget**

* Create a new Widget Blueprint named "WBP\_GraphicsSettings."
* Design the Graphics Settings UI (follow previous graphics settings guide).
* Add a "Back" button to return to the main menu.

**Step 6: Create Audio Settings Widget**

* Create a new Widget Blueprint named "WBP\_AudioSettings."
* Design the Audio Settings UI (follow previous audio settings guide).
* Add a "Back" button to return to the main menu.

**Step 7: Create Controls Settings Widget**

* Create a new Widget Blueprint named "WBP\_ControlsSettings."
* Design the Controls Settings UI (follow previous controls settings guide).
* Add a "Back" button to return to the main menu.

**Step 8: Implement Switching between Options Pages**

* 1. In the main menu's Widget Graph, connect the "ShowOptionsPage" event:
  2. Use a "Create Widget" node to create the appropriate options page (Graphics, Audio, or Controls).
  3. Use the "Add to Viewport" node to display the widget on the screen.

**Step 9: Add Back Button Functionality**

* 1. In each Options page (Graphics, Audio, Controls):
  2. Add a "Back" button with an event handler.
  3. Use the "Remove from Parent" node to close the current options page.

**Step 10: Test Your Main Menu**

* Press the "Play" button to test your main menu in the editor.
* Click the "Options" button to verify that it opens the correct options page.
* Test the "Back" button in each options page to ensure it returns to the main menu.

**Tips:**

* Ensure proper labeling and tooltips for each UI element.
* Use consistent naming conventions for variables and functions.
* Consider adding visual feedback, like button animations.
* Test the options pages on various screen resolutions to ensure they scale appropriately.

## **GRAPHICS PAGE:**

**Step 1: Open Your Project in Unreal Engine 5**

* Open Unreal Engine 5.
* Load your existing project or create a new one.

**Step 2: Create a Graphics Settings Widget Blueprint**

* Right-click in the Content Browser.
* Choose "User Interface" > "Widget Blueprint."
* Name it "WBP\_GraphicsSettings."
* Double-click to open the widget.

**Step 3: Design the Graphics Settings UI**

* + Add UI elements for each setting:
  + Sliders for quality settings (e.g., Texture Quality, Shadow Quality).
  + Dropdowns for presets, resolution, Fullscreen mode.
  + Buttons for Apply Changes, Reset to Defaults.
  + Text or tooltips for help information.

**Step 4: Set Up Widget Graph for Interaction**

* Open the Widget Blueprint Graph.
* Create variables for each setting (e.g., TextureQuality, ShadowQuality).
* Add event handlers for UI elements (OnValueChanged, OnClicked).
* Bind UI elements to corresponding variables.

**Step 5: Implement Resolution Settings**

* Add a dropdown for resolution.
* Create a function to update supported resolutions.
* Use the "GetSupportedScreenResolutions" function to populate the dropdown.
* Bind the resolution dropdown to a variable (e.g., SelectedResolution).

**Step 6: Implement Graphics Quality Presets**

* Add a dropdown for quality presets.
* Populate it with options (Low, Medium, High).
* Create a function to apply settings based on the selected preset.

**Step 7: Implement Individual Graphics Settings**

* Add sliders for individual graphics settings.
* Create functions to update corresponding quality variables.
* Bind sliders to quality variables.

**Step 8: Implement Apply Changes Button**

* Create a function to apply all selected settings.
* Inside this function, use nodes to update engine settings based on chosen values.
* Add confirmation dialogs using UMG widgets if needed.

**Step 9: Implement Reset to Defaults Button**

* Create a function to reset all settings to default values.
* Update UI elements to reflect default values.
* Use nodes to reset engine settings.

**Step 10: Implement Preview Window**

* If applicable, create a preview window within the widget.
* Use dynamic material instances or post-process volumes to preview changes in real-time.

**Step 11: Save/Load Presets**

* Implement functions to save and load user presets.
* Use file I/O or save game objects to store and retrieve preset data.

**Step 12: Implement Help/Tooltip Information**

* Add text or tooltips near each UI element.
* Create functions to display help information dynamically.

**Step 13: Implement Accessibility Options**

* Add options for colorblind modes or other accessibility features.
* Create functions to apply these changes to the game.

**Step 14: Implement Benchmark/Performance Test**

* Create a benchmark function to measure performance.
* Provide feedback or suggest optimal settings based on benchmark results.

**Step 15: Test and Iterate**

* Test the graphics settings in various scenarios.

Collect user feedback and iterate on the design for clarity and usability.

## **AUDIO SETTING**

**Step 1: Open Your Project in Unreal Engine 5**

* Open Unreal Engine 5.
* Load your existing project or create a new one.

**Step 2: Create an Audio Settings Widget Blueprint**

* Right-click in the Content Browser.
* Choose "User Interface" > "Widget Blueprint."
* Name it "WBP\_AudioSettings."
* Double-click to open the widget.

**Step 3: Design the Audio Settings UI**

* Add sliders for Master Volume, Music Volume, Sound Effects Volume, and Voice Volume.
* Add a checkbox for Dialogue Subtitles.
* Design labels next to each slider for clarity.

**Step 4: Set Up Widget Graph for Interaction**

* Open the Widget Blueprint Graph.
* Create variables for each setting (MasterVolume, MusicVolume, SoundEffectsVolume, VoiceVolume, bDialogueSubtitles).
* Add event handlers for UI elements (OnValueChanged for sliders, OnCheckStateChanged for checkboxes).

**Step 5: Implement Sliders for Master Volume, Music Volume, Sound Effects Volume, and Voice Volume**

* For each slider, create a custom event to update the corresponding volume variable.
* Use the "Set Sound Mix Class Override" node to adjust volume based on the slider value.
* Bind the sliders to their respective volume variables.

**Step 6: Implement Checkbox for Dialogue Subtitles**

* Create a custom event for the checkbox to toggle subtitles.
* Use a boolean variable (bDialogueSubtitles) to track the state.
* Use the "Set Subtitle Enabled" node to enable or disable subtitles based on the checkbox state.
* Bind the checkbox to the bDialogueSubtitles variable.

**Step 7: Apply Changes and Reset to Defaults**

* Add buttons for "Apply Changes" and "Reset to Defaults."
* Create functions for applying changes and resetting settings.
* In the "Apply Changes" function, use nodes to save the current settings or apply them immediately.
* In the "Reset to Defaults" function, set volume variables and checkbox state to default values.

**Step 8: Test Your Audio Settings**

* Press the "Play" button to test your Audio Settings widget in the editor.
* Adjust sliders and checkboxes to ensure they affect the game's audio as expected.
* Test the "Apply Changes" and "Reset to Defaults" buttons.

**Step 9: Implement Save/Load Presets (Optional)**

* If desired, add buttons for "Save Preset" and "Load Preset."
* Create functions to save and load user audio presets using file I/O or save game objects.

**Tips:**

* Ensure proper labeling and tooltips for each UI element.
* Use consistent naming conventions for variables and functions.
* Consider adding visual feedback, like volume bars, to make adjustments more intuitive.
* Test your audio settings across different sound scenarios to ensure they work in various game contexts.

## **CONTROLS**

**Step 1: Open Your Project in Unreal Engine 5**

* Open Unreal Engine 5.
* Load your existing project or create a new one.

**Step 2: Create a Controls Settings Widget Blueprint**

* Right-click in the Content Browser.
* Choose "User Interface" > "Widget Blueprint."
* Name it "WBP\_ControlsSettings."
* Double-click to open the widget.

**Step 3: Design the Controls Settings UI**

* Add buttons or dropdowns for control-related settings.
* Design labels next to each control setting for clarity.

**Step 4: Set Up Widget Graph for Interaction**

* Open the Widget Blueprint Graph.
* Create variables for each setting (e.g., Sensitivity, InvertYAxis).
* Add event handlers for UI elements (OnClicked for buttons, OnSelectionChanged for dropdowns).

**Step 5: Implement Sensitivity Slider**

* Add a slider for controlling sensitivity.
* Create a variable (e.g., Sensitivity) to store the sensitivity value.
* Use the "Set Mouse Sensitivity" node to adjust sensitivity based on the slider value.
* Bind the slider to the Sensitivity variable.

**Step 6: Implement Invert Y-Axis Checkbox**

* Add a checkbox for inverting the Y-axis.
* Create a boolean variable (e.g., bInvertYAxis) to track the state.
* Use the "Invert Axis" node to invert the Y-axis based on the checkbox state.
* Bind the checkbox to the bInvertYAxis variable.

**Step 7: Implement Custom Key Bindings (Optional)**

* If desired, add buttons or dropdowns for custom key bindings.
* Create variables for each key binding.
* Use the "Set Input Key Function" node to bind specific functions to custom keys.

**Step 8: Apply Changes and Reset to Defaults**

* Add buttons for "Apply Changes" and "Reset to Defaults."
* Create functions for applying changes and resetting settings.
* In the "Apply Changes" function, use nodes to save the current settings or apply them immediately.
* In the "Reset to Defaults" function, set variables to default values.

**Step 9: Test Your Controls Settings**

* Press the "Play" button to test your Controls Settings widget in the editor.
* Adjust sliders and checkboxes to ensure they affect controls as expected.
* Test the "Apply Changes" and "Reset to Defaults" buttons.

**Step 10: Accessibility Options**

* Add options for controls accessibility (e.g., customizable button size or color).
* Create functions to apply these changes to the game.

**Tips:**

* Ensure proper labeling and tooltips for each UI element.
* Use consistent naming conventions for variables and functions.
* Consider adding visual feedback, like slider bars or toggle animations.

Test your controls settings in various scenarios to ensure they work in different game contexts.

# **CREDITS**

**Step 1: Open Your Project in Unreal Engine 5**

* Open Unreal Engine 5.
* Load your existing project or create a new one.

**Step 2: Create a Credits Widget Blueprint**

* Right-click in the Content Browser.
* Choose "User Interface" > "Widget Blueprint."
* Name it "WBP\_Credits."
* Double-click to open the widget.

**Step 3: Design the Credits UI**

* Design the Credits page with text fields for contributors' names, roles, or any other relevant information.
* Consider adding a scroll box if you expect a long list of credits.

**Step 4: Set Up Widget Graph for Interaction**

* Open the Widget Blueprint Graph.
* Create variables for text fields if needed.
* Add any additional elements like buttons if you want interactivity (e.g., "Back to Main Menu").

**Step 5: Populate the Credits Text**

* Inside the Widget Blueprint Graph, create a function to populate the credits text.
* Use the "Set Text" node to set the text of your text fields.
* You can hardcode the credits or dynamically fetch them if they are stored elsewhere (e.g., in a data table).

**Step 6: Add Scroll Functionality (Optional)**

* If you have a long list of credits, consider adding a scroll functionality.
* Add a Scroll Box to your UI.
* Use the "Add Child" node to dynamically add text elements as credits.
* Implement scrolling using the Scroll Box's properties or additional buttons.

**Step 7: Create a Button to Return to Main Menu**

* If you want users to return to the main menu, add a button.
* Create an event handler for the button click.
* Use the "Open Level" node to load the Main Menu level or widget.

**Step 8: Test Your Credits Page**

* Press the "Play" button to test your Credits widget in the editor.
* Check if the credits text is displayed correctly.
* Test the button to return to the main menu.

**Step 9: Add the Credits Widget to the Main Menu**

* Open your Main Menu Widget Blueprint.
* Add a button for "Credits."
* Create an event handler for the Credits button.
* Use the "Create Widget" and "Add to Viewport" nodes to display the Credits widget.

**Tips:**

* Ensure clear labeling for buttons and text fields.
* Use consistent naming conventions for variables and functions.
* Consider adding transitions or animations for a polished look.

Test your Credits page on different screen sizes to ensure readability.

# QUIT GAME BUTTON

**Step 1: Open Your Project in Unreal Engine 5**

* Open Unreal Engine 5.
* Load your existing project or create a new one.

**Step 2: Create a Main Menu Widget Blueprint**

* Right-click in the Content Browser.
* Choose "User Interface" > "Widget Blueprint."
* Name it "WBP\_MainMenu."
* Double-click to open the widget.

**Step 3: Design the Main Menu UI**

* Add a button for "Quit Game" on your main menu.
* Optionally, add a TextBlock or another widget for the warning message.

**Step 4: Set Up Widget Graph for Interaction**

* Open the Widget Blueprint Graph.
* Create an event handler for the "Quit Game" button (OnClicked).
* Optionally, create an event handler for other relevant buttons (e.g., "Options" or "Play").

**Step 5: Implement Quit Game Functionality**

* 1. Inside the "Quit Game" event handler:
  2. Use the "Quit Game" node to gracefully exit the game.
  3. Optionally, before quitting, show a warning dialog.

**Step 6: Implement Warning Dialog**

* Create a custom widget for the warning dialog.
* Design the dialog with a message, buttons (Yes/No), and any additional information.
* Open the Widget Blueprint Graph of the warning dialog.

**Step 7: Set Up Widget Graph for Warning Dialog**

* Create variables to store relevant data (e.g., YesButtonClicked).
* Create event handlers for button clicks (Yes and No).
* In the "Yes" event handler, set the variable (YesButtonClicked) to true.
* In the "No" event handler, set the variable to false.
* Add logic to handle what happens when "Yes" or "No" is clicked.

**Step 8: Implement Warning Dialog Display**

* 1. In the "Quit Game" event handler (Step 5):
  2. Before quitting, create an instance of the warning dialog.
  3. Show the dialog and wait for the player's response (waiting for YesButtonClicked to be true or false).

**Step 9: Apply Changes**

* 1. In the warning dialog, after the player makes a choice:
  2. If "Yes" is clicked, proceed with quitting the game.
  3. If "No" is clicked, hide or destroy the warning dialog without quitting.

**Step 10: Test Your Main Menu**

* Press the "Play" button to test your Main Menu widget in the editor.
* Click the "Quit Game" button to trigger the warning dialog.
* Confirm the warning dialog functionality by clicking "Yes" or "No."

**Tips:**

* Ensure clear labeling and messaging in your warning dialog.
* Consider adding animations or transitions to make the dialog visually appealing.
* Test thoroughly to ensure the warning dialog behaves as expected.

You may customize the warning dialog further based on your game's theme and aesthetics.

## NAVIGATING ON A BOAT:

Step 1: Set Up Your Project:

* Open Unreal Engine and create a new project with the desired settings.
* Choose template or start with a blank project.

Step 2: Create the Boat Model:

* Import or create a 3D model of your boat.
* Ensure the boat has collision geometry for accurate interactions.

Step 3: Set Up Player Character:

* Create a new Blueprint class for your player character (e.g., "BP\_PlayerCharacter").
* Open the Blueprint and add a Skeletal Mesh component for your character (the player).
* Attach the boat model to the character as a child component.

Step 4: Implement Boat Controls:

* Inside the Blueprint, add Input functions for boat controls (e.g., MoveForward, MoveRight).
* Add nodes to control the boat's movement based on player input.
* For simplicity, use the "Add Movement Input" node for basic forward and sideways movement.
* Adjust boat speed and responsiveness by tweaking input scales.

Step 5: Set Up Water Material:

* Create a water material or use one from the marketplace.
* Apply the material to the water surfaces in your level.

Step 6: Implement Buoyancy:

* Add a Skeletal Mesh Component for the boat in the Blueprint.
* Enable "Simulate Physics" for the boat component.
* Add a Box or Capsule Collision Component to act as the buoyancy volume.
* Adjust the buoyancy volume to cover the submerged part of the boat.
* Tweak the physics settings (Mass, Damping, etc.) for realistic buoyancy.

Step 7: Adjust Camera for Boat:

* Add a Camera Boom Component to the Blueprint.
* Attach a Spring Arm Component to the Camera Boom.
* Position the camera above and behind the boat for a good view.
* Adjust parameters like Arm Length, Socket Offset, and Lag for smooth camera movement.

Step 8: Implement Docking Mechanism:

* Create Dock Blueprint classes for interaction points.
* Add collision components to represent docking locations.
* Implement logic in the player character to detect and interact with docks.

Step 9: Handle Environmental Challenges:

* Create obstacles and challenges in your level.
* Implement collision detection and response to handle obstacles.
* Add logic for the player to navigate around or through environmental challenges.

Step 10: Test and Iterate:

* Playtest your level to ensure smooth boat navigation.
* Iterate on controls, physics settings, and environmental interactions for better gameplay.

# TRIGGER RAIN

### 1. Create Weather Blueprint:

a. Create BP\_WeatherController:

Right-click in the Content Browser.

Choose "Blueprint Class."

Select "Actor" as the parent class.

Name it "BP\_WeatherController."

b. Add Components:

Open BP\_WeatherController.

In the Components panel (left side), click "Add Component."

Add a "Static Mesh" component for the sky (you can adjust the sky dome or sphere mesh).

Add a "Particle System" component for rain effects.

c. Create Variables:

In the My Blueprint panel (left side), click "Add Variable."

Name it "bIsRaining," set its type to Boolean.

Add another variable named "RainIntensity," type Float.

d. Create Custom Events:

Click on the "Event Graph" tab.

Right-click in the graph and select "Custom Event."

Name it "StartRain."

Right-click again, add another event named "StopRain."

Add one more named "ChangeRainIntensity."

### 2. Level Blueprint Setup:

a. Open Level Blueprint:

Open the level where the meeting event occurs.

Go to the "Blueprint" menu, and select "Open Level Blueprint."

b. On Player Meet Event:

Drag a reference to the BP\_WeatherController into the level blueprint.

On the event graph, create an event (e.g., "OnPlayerMeet").

Drag off the execution pin and call "StartRain" on BP\_WeatherController.

Connect a "Delay" node to simulate heavy rain for a duration.

c. After Delay - Reduce Rain Intensity:

After the delay, drag a reference to BP\_WeatherController.

Call "ChangeRainIntensity" with a lower intensity value.

Optionally, call "StopRain" if you want the rain to completely stop.

### 3. Boat Interaction Blueprint:

a. Create BP\_BoatInteraction:

Right-click in the Content Browser.

Choose "Blueprint Class."

Select "Actor" as the parent class.

Name it "BP\_BoatInteraction."

b. Boat Interaction Logic:

Open BP\_BoatInteraction.

On the event graph, create an event (e.g., "OnBoatStart").

Drag off the execution pin and call "StartRain" with a higher intensity on BP\_WeatherController.

Connect a "Delay" node to simulate heavy rain during the boat ride.

c. After Delay - Reduce Rain Intensity:

After the boat ride delay, drag a reference to BP\_WeatherController.

Call "ChangeRainIntensity" with a lower intensity value.

Optionally, call "StopRain" if you want the rain to completely stop.

## 4. BP\_WeatherController Modifications:

a. Modify Rain Intensity Event:

Open BP\_WeatherController.

In the "ChangeRainIntensity" event, find the Particle System component in the Components panel.

Adjust particle system parameters (like emitter size or spawn rate) based on the "RainIntensity" variable.

b. Rain Intensity Calculation:

In your level blueprint or BP\_BoatInteraction, you can calculate the rain intensity based on game events.

Set the "RainIntensity" variable on BP\_WeatherController with the calculated intensity.

### 5. Testing:

a. Playtest:

Hit the "Play" button and observe the changes in rain intensity based on player interactions.

Additional Tips:

To find the Components panel: In the Blueprint editor, it's on the left side. If it's not visible, go to the "Window" menu and ensure "Components" is checked.

To adjust particle system parameters: In the Components panel, select the Particle System component, and its details will be in the Details panel on the right side.

To drag references to other blueprints: In the level blueprint, you can drag a reference from the node you're working on (like "Event Begin Play") to another node to create a connection.

## UI Interface.

Main Menu