
Table of Contents

---

1. Overview

`RCC_UI_Controller` is a **UI component** within Realistic Car Controller (RCC) designed to collect a **float** input from Unity's **UI** system. It can operate as either:

- A **button** (`Button` component), where pressing gradually increases an `input` value (0 to 1) and releasing decreases it.
- A **slider** (`Slider` component), where the user can directly set a float between 0 and 1.

Other RCC scripts (e.g., `RCC_MobileButtons`) typically reference `RCC_UI_Controller` objects to get the **current input** values.

---

2. Class Declaration

```
public class RCC_UI_Controller : RCC_Core, IPointerDownHandler, IPointerUpHandler {
    // ...
}
```

- Inherits from **`RCC_Core`**, giving access to common RCC settings (`Settings`), etc.
- Implements **`IPointerDownHandler`** and **`IPointerUpHandler`** to detect press and release events on UI elements.

---

3. Purpose and Functionality

- **Abstracts** user touch or mouse events into a single `input` float in the range `[0..1]`.
- **Optionally** acts like a **button**: Pressing gradually increases `input` from 0 up to 1 at a rate defined by `Settings.UIButtonSensitivity`; releasing decreases it at `Settings.UIButtonGravity`.
- **Alternatively** acts like a **slider**: Directly sets `input` to the slider's value.

---

4. Fields and Properties

- **`Button button`**: The attached `Button` component if using button mode. Could be `null` if in slider mode.
- **`Slider slider`**: The attached `Slider` component if using slider mode. Could be `null` if in button mode.
- **`public float input = 0f`**: The **current** input value in `[0..1]`.
- **`private float Sensitivity`**: Shortcut to `Settings.UIButtonSensitivity`.
- **`private float Gravity`**: Shortcut to `Settings.UIButtonGravity`.
- **`public bool pressing = false`**: Indicates whether the user is currently pressing (finger/mouse down).

---

5. Behavior in Different UI Contexts

5.1 Button Mode

If a `Button` component is found:

- **Pressing** (`OnPointerDown`) starts incrementing `input` each frame (`input += Time.deltaTime * Sensitivity`).
- **Releasing** (`OnPointerUp`) triggers `pressing = false`, so `input` will decrement each frame (`input -= Time.deltaTime * Gravity`) until 0.
- **`input`** is **clamped** between 0 and 1.

5.2 Slider Mode

If a `Slider` component is found:

- The script directly reads `slider.value`.
- **Pressing** means we keep using `slider.value`, though typically a slider can also be dragged.
- Once released or not dragging, the script sets `input` to 0 (by design, if the user's pointer is no longer on it).
- `slider.value` is also forcibly set to `input` to keep them in sync.

---

6. Event Handling

Implements **`IPointerDownHandler`** and **`IPointerUpHandler`** to detect UI pointer events:

6.1 `OnPointerDown(PointerEventData eventData)`

```
public void OnPointerDown(PointerEventData eventData) {
    pressing = true;
}
```

- Sets `pressing = true`, causing the input to rise (button mode) or reflect slider value (slider mode).

6.2 `OnPointerUp(PointerEventData eventData)`

```
public void OnPointerUp(PointerEventData eventData) {
    pressing = false;
}
```

- Sets `pressing = false`, so in button mode the input will fall back to 0 over time.
- In slider mode, typically the script sets `input = 0` if not pressing.

---

7. Processing Input in LateUpdate()

```
private void LateUpdate() {
    // If button is present but not interactable => pressing = false, input = 0
    // If slider is present but not interactable => pressing = false, input = 0
    // If slider => input = pressing ? slider.value : 0f
    // If button => increment or decrement input depending on pressing
```

```
    // clamp input in [0..1]
}
```

- The method checks if the UI component is **interactable**:
    - If not, sets `input = 0` and `pressing = false`.
- If in slider mode (`slider != null`), `input` is taken from `slider.value` only if `pressing` is `true`; else reset to 0.
- If in button mode (`button != null`), `input` is incremented or decremented.

This ensures consistent, real-time update of the float input within the correct range.

---

8. Lifecycle Methods

8.1 `Awake()`

```
private void Awake() {
    button = GetComponent<Button>();
    slider = GetComponent<Slider>();
}
```

- Detects whether the GameObject has a `Button` or a `Slider`.

8.2 `OnEnable()`

```
private void OnEnable() {
    input = 0f;
    pressing = false;
    if (slider) slider.value = 0f;
}
```

- Resets `input` to 0 and clears `pressing`.
- If a slider is present, sets its `value` to 0.

8.3 `OnDisable()`

```
private void OnDisable() {
    input = 0f;
    pressing = false;
    if (slider) slider.value = 0f;
}
```

- Ensures the input is **zeroed** when deactivated (no leftover values).

---

9. Usage Notes and Best Practices

1. **UI Setup**
   - Attach **`RCC_UI_Controller`** to a UI GameObject containing either a `Button` or a `Slider`.
   - If both exist, it will choose the first found or treat it as a slider if both are present (which is usually not recommended).
2. **Range**
   - **`input`** is always `[0..1]`. If you need negative values (e.g., steering), consider other scripts (like `RCC_UI_Joystick`) or a custom approach.
3. **Custom Sensitivity**
   - You can tweak `Settings.UIButtonSensitivity` or `Settings.UIButtonGravity` in **RCC_Settings** to change how quickly the input rises/falls in button mode.
4. **Integration with Other Scripts**
   - Typically read by `RCC_MobileButtons` or your custom code to drive the vehicle or other logic.
   - Example: `float gas = gasButton.input;`
5. **Disable Interactability**
   - If the `Button` or `Slider` is **not interactable**, the script will treat it as `input = 0`.
   - This is useful for disabling certain inputs at runtime.

---

10. Summary

`RCC_UI_Controller` is a simple but flexible script for **on-screen vehicle controls** in the Realistic Car Controller framework. It can interpret a **Button** press as a **gradually increasing** float from 0 to 1, or let a **Slider** directly set the input value in real time. This unified approach allows for easy integration into any RCC-based mobile or UI control scheme, providing a straightforward **`input`** property that other RCC scripts can read each frame.