Table of Contents

---

1. Overview

`RCC_OverrideInputsExample` shows how to **manually set** (override) the inputs of a Realistic Car Controller vehicle. By supplying a custom `RCC_Inputs` struct (throttle, brake, steering, etc.), you can take control away from the normal player input system. This is useful for:

- **AI control**
- **Cutscenes** or scripted events
- **Remote inputs** (e.g., networking or debugging tools)

---

2. Class Declaration

```
public class RCC_OverrideInputsExample : RCC_Core {
    // ...
}
```

- Inherits from `RCC_Core`, giving it access to `CarController` (if needed), `Settings`, etc.

---

3. Purpose and Functionality

- Allows a developer to override the standard user input and forcibly supply custom input values each frame.
- Provides UI sliders to adjust throttle/brake/steering/handbrake/NOS in real-time.
- Demonstrates how to **activate** and **deactivate** overriding logic using `OverrideInputs(RCC_Inputs newInputs)` and `DisableOverrideInputs()` on the target vehicle.

---

4. Fields and Properties

4.1 Target Vehicle

public RCC_CarControllerV4 targetVehicle;
public bool takePlayerVehicle = true;

- **`targetVehicle`**: The specific vehicle we want to override.
- **`takePlayerVehicle`**: If `true`, automatically assigns `RCCSceneManager.activePlayerVehicle` each frame, ignoring manual assignment.

4.2 Input Override Struct

public RCC_Inputs newInputs = new RCC_Inputs();
private bool overrideNow = false;

- **`newInputs`**: The custom `RCC_Inputs` containing override values for throttle, brake, steer, etc.
- **`overrideNow`**: Tracks whether overriding is currently in effect.

4.3 UI Elements

public Slider throttle;
public Slider brake;
public Slider steering;
public Slider handbrake;
public Slider nos;
public TextMeshProUGUI statusText;

- Each `Slider` modifies a corresponding field in `newInputs`.
- `statusText` displays whether overriding is active or not.

---

5. Key Methods

5.1 Update()

```
private void Update() {
    // 1) Read slider values and store in newInputs.
    // 2) Optionally update targetVehicle from RCCSceneManager if takePlayerVehicle is true.
    // 3) If overrideNow is true, call targetVehicle.OverrideInputs(newInputs).
    // 4) Update status text to reflect whether override is Enabled or Disabled.
}
```

- Each frame, sets `newInputs.throttleInput = throttle.value`, etc.
- If `overrideNow` is true, applies these to the vehicle via `OverrideInputs()`.
- If `statusText` is assigned, updates a label (e.g. “Status: Enabled” vs. “Disabled”).

5.2 EnableOverride()

```
public void EnableOverride() {
    overrideNow = true;
    targetVehicle.OverrideInputs(newInputs);
}
```

- Called by UI button or code.
- Sets `overrideNow = true`, then immediately applies the override.

5.3 DisableOverride()

```
public void DisableOverride() {
    overrideNow = false;
    targetVehicle.DisableOverrideInputs();
}
```

- Returns control to normal input.
- Calls `DisableOverrideInputs()` on the vehicle, allowing the standard `RCC_InputManager` to resume.

---

6. Usage Notes and Best Practices

1. **Continuous vs. Instant**
    - When `overrideNow` is true, **every frame** the car uses `newInputs`.
    - If you only need a one-time override (like a short control script), you can enable override for a set duration or event.

2. **Slider Ranges**
    - Ensure the Unity `Slider` for **steering** is set to **min = -1** and **max = 1** in the Inspector, while **throttle**, **brake**, **handbrake**, and **nos** are **0..1**.
3. **Conflicts**
    - Disabling override (`DisableOverride()`) is crucial if you want the user to regain control.
    - If multiple scripts try to override inputs at once, the last call in each frame typically wins. So manage them carefully or maintain a single source of truth.
4. **Auto-Finding Player Vehicle**
    - If `takePlayerVehicle` is true, it **always** overwrites `targetVehicle` with `RCCSceneManager.activePlayerVehicle`.
    - If you prefer a specific vehicle reference, set `takePlayerVehicle = false`.
5. **Use Cases**
    - **AI** or **Autopilot** demonstration.
    - **Testing**: Quickly see how the vehicle responds to certain inputs without physically pressing keys.
    - **In-Game Cinematics**: Force the car to drive in a specific path or state.

---

7. Example Integration

- **Unity UI**: Place a canvas with Sliders for throttle, brake, steer, etc., and a Toggle or Button for enabling/disabling override.

Script Reference:
RCC_OverrideInputsExample overrideScript = gameObject.AddComponent<RCC_OverrideInputsExample>();
overrideScript.takePlayerVehicle = false;
overrideScript.targetVehicle = someCarControllerV4Reference;

-

Enable:
overrideScript.EnableOverride(); // from a button or code

-

Disable:
overrideScript.DisableOverride();

-

8. Summary

`RCC_OverrideInputsExample` is a **simple demonstration** of how to **manually control** an RCC vehicle by bypassing the normal input system. It:

- Reads UI slider values into an `RCC_Inputs` struct.
- Applies them each frame to a chosen vehicle when overriding is active (`OverrideInputs(newInputs)`).
- Reverts to normal inputs when disabled (`DisableOverrideInputs()`).

This approach is invaluable for scripting cutscenes, AI logic, or debugging custom control schemes in **Realistic Car Controller**.