# Table of Contents

## 1. Overview

**`RCC_InputManager`** manages **player inputs** for the Realistic Car Controller (RCC) system. It translates raw Unity InputSystem actions or mobile UI inputs into normalized values (throttle, brake, steering, handbrake, etc.) that RCC components (e.g., `RCC_CarControllerV4`, `RCC_Camera`) consume.

Key responsibilities:

- Reading **Unity InputSystem** actions via the **`RCC_InputActions`** asset.
- Firing **events** in response to button presses (e.g., toggling headlights, changing camera).
- Supporting **mobile** input when enabled via RCC settings.
- Optionally handling **gyro steering**.

## 2. Class Declaration

public class RCC_InputManager : RCC_Singleton<RCC_InputManager> {
   // ...

}

- Inherits from a **singleton** base class (`RCC_Singleton<RCC_InputManager>`), ensuring only one instance is active.

---

## 3. Purpose and Functionality

- **Unified Input Layer**: All user inputs (keyboard, gamepad, mobile UI, gyro) funnel into a single place.
- **Events**: When a user presses certain buttons (e.g., toggle headlights), it triggers events so other scripts can react.
- **Continuous Axes**: Throttle, brake, steering, handbrake, and boost values are stored in an `RCC_Inputs` struct, updated each frame.

---

## 4. Fields and Properties

### 4.1 Input Storage (`RCC_Inputs`)

public RCC_Inputs inputs = new RCC_Inputs();

- Holds the current frame's input values:
  - `throttleInput`
  - `brakeInput`
  - `steerInput`
  - `handbrakeInput`
  - `boostInput` (e.g., NOS)
  - `clutchInput`
  - `orbitX`, `orbitY`, `scroll` (camera orbit and zoom)

These values are read by the car controller or camera each frame.

### 4.2 Unity InputActions (`RCC_InputActions`)

private static RCC_InputActions inputActions;

- An **InputSystem**-generated class that defines mappings for each action (throttle, brake, turn, headlights, etc.).
- Instantiated once on-demand in `GetInputs()` and then **enabled**.

### 4.3 Gyro Usage

public bool gyroUsed = false;

- Indicates whether gyro steering is currently in effect (particularly relevant for mobile).
- Not heavily demonstrated in this snippet but can be extended to read device gyroscope data.

---

## 5. Events and Delegates

`RCC_InputManager` triggers a variety of **static events** in response to user button presses or toggles. For example:

- **`OnStartStopEngine`**: Called when the user toggles the engine.
- **`OnLowBeamHeadlights`**, **`OnHighBeamHeadlights`**: Called when headlights are toggled.
- **`OnChangeCamera`**: Called when the camera is switched.
- **`OnIndicatorLeft`**, **`OnIndicatorRight`**, **`OnIndicatorHazard`**: Left, right, or hazard signal toggles.
- **`OnGearShiftUp`**, **`OnGearShiftDown`**: Gear changes.
- **`OnNGear(bool state)`**: Neutral gear toggle (pressed or released).
- **`OnSlowMotion(bool state)`**: Slow-motion toggle.
- **`OnRecord`**, **`OnReplay`**: For recording or replaying vehicle movements.
- **`OnLookBack(bool state)`**: Look-back camera toggle (pressed or released).
- **`OnTrailerDetach`**: Detach trailer from the vehicle.

These events allow **loose coupling**: any other script can subscribe to them and react without direct references to the input manager.

---

## 6. Lifecycle

### 6.1 `Awake()`

```
private void Awake() {
   gameObject.hideFlags = HideFlags.HideInHierarchy;
   inputs = new RCC_Inputs();
}
```

- **Hides** the game object from the hierarchy.
- Initializes `inputs`.

### 6.2 `Update()`

```
private void Update() {
    if (inputs == null)
        inputs = new RCC_Inputs();

    GetInputs();
}
```

- Ensures `inputs` is never null.
- Calls `GetInputs()` every frame to update the input values.

---

## 7. Collecting Inputs (`GetInputs()`)

This is the **core** function that reads the actual input values each frame.

```
public void GetInputs() {
    if (inputActions == null) {
        inputActions = new RCC_InputActions();
        inputActions.Enable();
        // Bind all performed/canceled events for button inputs...
    }

    if (!Settings.mobileControllerEnabled) {
        // Desktop / standard input
        inputs.throttleInput = inputActions.Vehicle.Throttle.ReadValue<float>();
        inputs.brakeInput = inputActions.Vehicle.Brake.ReadValue<float>();
        inputs.steerInput = inputActions.Vehicle.Steering.ReadValue<float>();
        inputs.handbrakeInput = inputActions.Vehicle.Handbrake.ReadValue<float>();
        inputs.boostInput = inputActions.Vehicle.NOS.ReadValue<float>();
        inputs.clutchInput = inputActions.Vehicle.Clutch.ReadValue<float>();

        inputs.orbitX = inputActions.Camera.Orbit.ReadValue<Vector2>().x;
        inputs.orbitY = inputActions.Camera.Orbit.ReadValue<Vector2>().y;
        inputs.scroll = inputActions.Camera.Zoom.ReadValue<Vector2>();
    } else {
        // Mobile input
        inputs.throttleInput = RCC_MobileButtons.mobileInputs.throttleInput;
        inputs.brakeInput = RCC_MobileButtons.mobileInputs.brakeInput;
        inputs.steerInput = RCC_MobileButtons.mobileInputs.steerInput;
        inputs.handbrakeInput = RCC_MobileButtons.mobileInputs.handbrakeInput;
        inputs.boostInput = RCC_MobileButtons.mobileInputs.boostInput;
    }
}
```

### 7.1 Desktop / Standard Input

- If **mobileControllerEnabled** is `false`, reads axis values via `inputActions.Vehicle.*`.
    - `Throttle` / `Brake`: float in `[0..1]` or `[-1..1]`, depending on the action definition.
    - `Steering`: float in `[-1..1]`.
    - `Handbrake`, `NOS` (boost), `Clutch`: float in `[0..1]`.
- Also captures **camera orbit** (`Orbit` action) and **zoom** (`Zoom` action) from the `Camera` map.

### 7.2 Mobile Input

- If **mobileControllerEnabled** is `true`, it defers to `RCC_MobileButtons.mobileInputs`, which is a separate system managing on-screen buttons/joysticks.
- `gyroUsed` (if set) could be read elsewhere or integrated into `RCC_MobileButtons`.

---

## 8. Callback Methods

Example of a callback binding:

```
inputActions.Vehicle.StartStopEngine.performed += StartStopEngine_performed;

private void StartStopEngine_performed(InputAction.CallbackContext obj) {
    OnStartStopEngine?.Invoke();
}
```

When the **`StartStopEngine`** action is triggered (e.g., user pressed a key), the local handler fires and then calls `OnStartStopEngine?.Invoke()`. The same pattern applies for **trailer detach**, **look back**, **record**, **replay**, etc. By design, it's using `performed` or `canceled` events to provide **boolean** toggles (for hold vs. release).

---

## 9. Usage Notes and Best Practices

1. **Single Instance**

    - As a singleton, only one `RCC_InputManager` should be present. Usually added automatically by RCC if not found.
2. **Modular Events**
    - Subscribing scripts (e.g., `RCC_CarControllerV4`, `RCC_Camera`) handle `OnIndicatorLeft`, `OnChangeCamera`, etc.
    - This approach keeps the input manager separate from the actual vehicle logic.
3. **Mobile vs. Desktop**
    - The code automatically chooses between **desktop input** (`InputActions`) and **mobile input** (`RCC_MobileButtons`).
    - Toggle `Settings.mobileControllerEnabled` at runtime if you want to switch input modes on the fly.
4. **Customizing InputActions**
    - If you add or change actions in **`RCC_InputActions`**, re-generate that class in the Unity Editor.
    - Update this script to handle new events or axes if needed.
5. **Performance**
    - Input checks happen in `Update()`, which is standard for user inputs. This is generally lightweight.
6. **Integrating with Other Systems**
    - If you want to integrate with custom UI or other controllers, you can reference `RCC_InputManager.Instance.inputs` directly or raise new events.

---

## 10. Summary

`RCC_InputManager` provides a **unified input interface** for Realistic Car Controller. It reads inputs from Unity's **new InputSystem** or RCC's **mobile control** logic and **broadcasts events** for feature toggles like headlights or shifting gears. By storing continuous input states (throttle, brake, steer, etc.) in `RCC_Inputs`, it makes these values easily consumable by any RCC component, decoupling **input logic** from **vehicle and camera logic**.