Table of Contents

---

1. Overview

`RCC_MobileButtons` is a script that handles **on-screen (mobile) controls** for **Realistic Car Controller (RCC)**. It supports multiple **mobile control schemes**, including:

- **TouchScreen** (tap-based steering with left/right buttons, gas/brake buttons or sliders).
- **Gyro** (device accelerometer for steering).
- **Steering Wheel** (one on-screen wheel to steer).
- **Joystick** (a single joystick for steering input).

`RCC_MobileButtons` creates a **unified** set of input values (`mobileInputs`) that get consumed by the RCC vehicle (`RCC_CarControllerV4`) or the RCC input manager.

---

2. Class Declaration

public class RCC_MobileButtons : RCC_Core {
    // ...
}

- Inherits from `RCC_Core`, giving it access to `CarController`, `Settings`, etc.
- Expected to be placed on a UI manager GameObject in your scene, along with references to UI elements.

---

3. UI Elements and Configuration

3.1 Controller Buttons

Exposed public fields for the on-screen buttons:

- **`RCC_UI_Controller gasButton`**
  Tap-based gas pedal (`input` in [0..1]).
- **`RCC_UI_Controller gradualGasButton`**
  Slider-based throttle for more fine control.
- **`RCC_UI_Controller brakeButton`**
  Brake pedal.
- **`RCC_UI_Controller leftButton, rightButton`**
  Steering buttons if using a TouchScreen layout.
- **`RCC_UI_Controller handbrakeButton`**
  For toggling or pressing the handbrake (drift).
- **`RCC_UI_Controller NOSButton, NOSButtonSteeringWheel`**
  Nitrous oxide boost buttons in different layouts.

All these buttons are of type `RCC_UI_Controller`, which has an `input` property returning a float in `[0..1]` depending on user touch.

3.2 Gradual Throttle vs. Tap Throttle

public bool useGradualThrottle = false;

- If `true`, the script displays `gradualGasButton` instead of `gasButton`.
- Allows the user to have a slider-like pedal for more nuanced control.

3.3 Steering Wheel and Joystick

- **`public RCC_UI_SteeringWheelController steeringWheel;`**
  An on-screen wheel that rotates based on user drag, returning a steering input float.

- **`public RCC_UI_Joystick joystick;`**
  A joystick that typically returns an X-axis for steering (and possibly Y for other purposes).

---

4. Internal Input Processing

4.1 Collected Variables

Within `Update()`, the script reads from:

- **`gasButton`** or **`gradualGasButton`** → `throttleInput`
- **`brakeButton`** → `brakeInput`
- **`leftButton`**, **`rightButton`** → combined steering input.
- **`steeringWheel`** → `steeringWheelInput`
- **`joystick`** → `joystickInput`
- **`gyroscope`** → `gyroInput` (accelerometer-based if `Gyro` layout is chosen).
- **`handbrakeButton`** → `handbrakeInput`
- **`NOSButton`** or **`NOSButtonSteeringWheel`** → `boostInput`

The script then merges these into a single set of **steering** and **throttle** values.

4.2 Static `mobileInputs`

public static RCC_Inputs mobileInputs = new RCC_Inputs();

- The final result of the above button inputs are stored in this **static** `RCC_Inputs` object.
- Other RCC scripts (like `RCC_InputManager`) can read from `RCC_MobileButtons.mobileInputs.throttleInput` etc., to apply them to the vehicle.

---

5. Lifecycle Methods

5.1 `Start()`

- Stores the original position of the **`brakeButton`** (e.g., used in **Gyro** layout to reposition the brake).
- Calls `CheckMobileButtons()` which enables or disables buttons based on `Settings.mobileControllerEnabled`.

5.2 `OnEnable()` and `OnDisable()`

- **`OnEnable()`**: Subscribes to `RCC_SceneManager.OnVehicleChanged` event. If the player changes vehicles, we may need to re-check which buttons to show.
- **`OnDisable()`**: Unsubscribes from the above event to avoid memory leaks.

5.3 `Update()`

1. If `mobileControllerEnabled` is **false**, it returns early (no mobile input).
2. Switches on `Settings.mobileController` to pick the layout (TouchScreen, Gyro, SteeringWheel, Joystick).
3. Reads inputs from the relevant UI elements.
4. Merges them into final values → `mobileInputs`.

---

6. Enabling and Disabling Buttons

- **`DisableButtons()`**: Hides all references (gas, brake, steering wheel, joystick, NOS, etc.).
- **`EnableButtons()`**: Shows relevant buttons based on whether `useGradualThrottle` is `true` or `false`.
    - For instance, if `useGradualThrottle` is `true`, hides `gasButton` and shows `gradualGasButton`.

---

7. Mobile Layout Setups

`Update()` calls one of these methods depending on the chosen layout:

7.1 TouchScreen Layout

```
private void SetTouchScreenLayout() {
    // Steering is done via leftButton & rightButton
    // Gas / brake are separate buttons
    // No gyro or steering wheel usage here
}
```

- `gyroUsed` is disabled.
- Left/right buttons are visible.
- If the car uses NOS, `NOSButton` is shown.
- SteeringWheel, joystick, etc. are hidden.

7.2 Gyro Layout

```
private void SetGyroLayout() {
    // Steering via device's accelerometer
    // Hide left/right buttons
    // Reposition brake button where left used to be
}
```

- Enables the **gyroscope** (`Accelerometer.current`) via the new InputSystem.
- Disables on-screen left/right.
- Leaves gas/brake, and possibly NOS.

7.3 Steering Wheel Layout

```
private void SetSteeringWheelLayout() {
    // The "RCC_UI_SteeringWheelController" is used for steering
    // No left/right buttons or gyro
    // Possibly a separate NOS button if car supports it
}
```

- Hides the normal NOS button in favor of `NOSButtonSteeringWheel`.
- If `steeringWheel` isn't active, it's shown.

7.4 Joystick Layout

```
private void SetJoystickLayout() {
    // The "RCC_UI_Joystick" is used for steering
    // Hide left/right buttons, steering wheel, or gyro
}
```

- Shows joystick, repositions brake button to original position.
- If `canUseNos` is `true`, shows the standard NOS button.

---

8. Example Integration

1. **Add `RCC_MobileButtons`** to a GameObject with references to your UI.
2. Configure:
    - Gas button or Gradual Gas.
    - Brake, Left, Right, etc.
    - Steering Wheel or Joystick if needed.
3. In **RCC_Settings**, set:
    - `mobileControllerEnabled = true`.

    - `mobileController = RCC_Settings.MobileController.TouchScreen` (for example).
4. At runtime, `RCC_MobileButtons.mobileInputs` gets updated each frame.
5. `RCC_InputManager` checks if `mobileControllerEnabled` is true and if so, uses these values instead of standard input.

---

9. Usage Notes and Best Practices

1. **Controller Switching**
    - You can switch `Settings.mobileController` at runtime (e.g., from a UI dropdown).
    - `RCC_MobileButtons` automatically reconfigures which UI elements are visible.
2. **NOS Support**
    - `canUseNos` is set by checking the active player vehicle's `useNOS` property. If the car supports it, the button is visible.
3. **Gyro Sensitivity**
    - Controlled by `Settings.gyroSensitivity`. Adjust for a comfortable tilt range.
    - If the accelerometer is unavailable (like in the Editor or some devices), `gyroInput` will be zero.
4. **Testing**
    - In the **Unity Editor**, you likely have to simulate touches or rely on a device build to test real tilt/gyro.
    - You can also toggle these layouts at runtime for debugging.
5. **Edge Cases**
    - Ensure `RCC_UI_Controller` references are not null. The script gracefully handles them by ignoring if `button == null`.
6. **Performance**
    - This is typically lightweight as it's just reading UI values once per frame.

---

10. Summary

`RCC_MobileButtons` streamlines the process of **mobile on-screen controls** in **Realistic Car Controller**. By exposing different layouts (TouchScreen, Gyro, Steering Wheel,

Joystick), it can accommodate various **driving preferences** on mobile devices. It reads input values from UI elements (buttons, sliders, wheels, or joystick) and compiles them into one consistent `RCC_Inputs` object (`mobileInputs`). Other RCC systems then interpret these inputs to steer, accelerate, brake, or apply NOS on the active vehicle.