# Table of Contents

## 1. Overview

`RCC_Demo` is a **utility manager** script that demonstrates how to handle:

- **Vehicle spawning** in Realistic Car Controller (RCC) demos.
- **Behavior selection** (e.g., arcade, realistic).
- **Mobile control switching** (touch, gyro, steering wheel, joystick).
- **Quality settings** switching.
- **Scene management** (restarting and quitting).
- **(Optional)** Integrates with **Photon PUN** for multiplayer vehicle spawning.

It is typically used in demo or prototyping scenes to quickly switch vehicles, behaviors, and control schemes at runtime.

## 2. Class Declaration

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

- Inherits from `RCC_Core`, which provides a reference to an RCC vehicle (via `CarController`) if needed.
- Can optionally integrate with **Photon Pun** networking if `PHOTON_UNITY_NETWORKING` and `RCC_PHOTON` are defined.

---

## 3. Fields and Properties

- **`public int selectedVehicleIndex = 0;`**
   Tracks which vehicle prefab will be spawned from the `RCC_DemoVehicles.Instance.vehicles` array.

- **`public int selectedBehaviorIndex = 0;`**
   Tracks which behavior profile (in `RCC_Settings.behaviorTypes`) is currently chosen.

---

## 4. Key Methods

This section covers the primary functionality offered by `RCC_Demo`.

### 4.1 Vehicle Selection and Spawning

1. **`public void SelectVehicle(int index)`**

   - Sets `selectedVehicleIndex` to the provided index.
   - Next time `Spawn()` is called, the vehicle at this index is used.
2. **`public void Spawn()`**

   - Spawns a new player vehicle from `RCC_DemoVehicles.Instance.vehicles[selectedVehicleIndex]`.
   - If a player vehicle already exists, destroys the old one first.
   - Preserves **position**, **rotation**, **velocity**, and **angular velocity** from the old vehicle, if available.
   - Ensures the newly spawned vehicle is registered as the active player vehicle (`registerAsPlayerVehicle: true`) with engine running.

// Example usage:
demo.SelectVehicle(1); // Choose 2nd vehicle in the array
demo.Spawn();           // Replace any existing vehicle with the chosen one

3.

### 4.2 Behavior Management

1. **`public void SetBehavior(int index)`**

    - Assigns `selectedBehaviorIndex` to the given `index`.
    - A separate call to `InitBehavior()` is needed to actually apply it.
2. **`public void InitBehavior()`**
    - Calls `RCC.SetBehavior(selectedBehaviorIndex)`, which changes the vehicle handling/behavior profile.
    - These profiles are defined in `RCC_Settings.behaviorTypes`.

```
demo.SetBehavior(2);    // Store the index
demo.InitBehavior();    // Apply behavior #2 to the active vehicle
```

3.

## 4.3 Mobile Controller Selection

```
public void SetMobileController(int index) {
    switch (index) {
        case 0:
            RCC.SetMobileController(RCC_Settings.MobileController.TouchScreen);
            break;
        case 1:
            RCC.SetMobileController(RCC_Settings.MobileController.Gyro);
            break;
        case 2:
            RCC.SetMobileController(RCC_Settings.MobileController.SteeringWheel);
            break;
        case 3:
            RCC.SetMobileController(RCC_Settings.MobileController.Joystick);
            break;
    }
}
```

- Accepts an `int` (`0` to `3`) mapping to **TouchScreen**, **Gyro**, **SteeringWheel**, or **Joystick** modes.
- Updates the current mobile input method used by RCC.

## 4.4 Quality Settings

```
public void SetQuality(int index) {
    QualitySettings.SetQualityLevel(index);
}
```

- Simple wrapper to switch Unity's **QualitySettings** level at runtime (0 = lowest, up to however many levels are defined).

## 4.5 Scene Management (Restart/Quit)

1. **`public void RestartScene()`**
   - Reloads the **current active scene** using `SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex)`.
2. **`public void Quit()`**
   - Closes the application (no effect in Editor).

// Example usage:
demo.RestartScene();  // Reload the current scene
demo.Quit();          // Quit the game/application

3.

---

## 5. Photon Networking (Optional)

If **PHOTON_UNITY_NETWORKING** and **RCC_PHOTON** are defined, the script adds additional Photon-based methods:

- **`public void SelectPhotonVehicle(int index)`**
  - Similar to `SelectVehicle()`, but used in a Photon context.
- **`public void SpawnPhoton()`**
  - Destroys the existing player vehicle with `PhotonNetwork.Destroy(...)`.
  - Instantiates a new **networked** vehicle prefab from `"Photon Vehicles/" + vehicleName`.
  - Registers it as the player vehicle in RCC.

This allows the same demo script to spawn vehicles in a **multiplayer** environment.

---

## 6. Usage Notes and Best Practices

1. **Vehicle Prefabs Array**
   - The script relies on `RCC_DemoVehicles.Instance.vehicles[]`. Ensure you populate that array with valid RCC vehicle prefabs.
2. **Photon**

    - When using `SpawnPhoton()`, ensure you have prefabs in a **Resources/Photon Vehicles/** folder matching the **exact** names in `RCC_DemoVehicles`.
    - Also make sure the prefab has a `PhotonView` and is registered in the Photon PUN settings.
3. **Behavior Initialization**
    - `SetBehavior()` only assigns an index; call `InitBehavior()` to apply changes.

Alternatively, you can call them directly:
 demo.SetBehavior(2);
demo.InitBehavior();

4. **Scene Transitions**
    - `RestartScene()` is convenient for demo scenarios, but in production, you may want a more sophisticated approach (scene loading transitions, save states, etc.).
5. **Layer Management**
    - If the script is used in combination with dynamic spawning (especially in Photon), ensure your layers and tag setups are consistent, so the vehicle physics and camera follow work properly.
6. **Enter/Exit Integration**
    - If **BCG_ENTEREXIT** is defined, the script tries to handle transferring the driver to the newly spawned car. This is specific to **BoneCracker Games** Enter/Exit add-on.
7. **Project Setup**
    - Typically placed on a **Game Manager** or **UI** object in the scene. The methods can be wired up to **UI Buttons** for quick testing.

---

## 7. Summary

`RCC_Demo` is a straightforward script for **RCC demo scenes**—quickly **spawning** vehicles, **changing behaviors**, **switching mobile controls**, **adjusting quality**, and **restarting or quitting** the scene. It also supports **multiplayer spawning** via Photon if configured. This script greatly simplifies the workflow in a demo environment or a prototype, letting developers and testers toggle between different cars and configurations without manual scene changes or complex custom logic.