Table of Contents

---

1. Overview

The **`RCC_CarSelectionExample`** script provides a simple vehicle selection system for a Realistic Car Controller (RCC) demo scene. It:

- Spawns a list of vehicles from a predefined list only once.
- Allows players to cycle through vehicles using “Next” and “Previous” buttons.
- Activates only the selected vehicle, deactivating the rest.
- Remembers the last selected vehicle index across scenes using `PlayerPrefs`.
- Transitions into gameplay by registering the chosen vehicle as the active player’s car.
- Optionally loads another scene once a vehicle is selected.

This system is ideal for simple “showroom” or “garage” style menus, where the user can preview and choose a vehicle before entering the main game scene.

---

2. Class Declaration

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

This class inherits from `RCC_Core`, a base class from RCC. It relies heavily on `RCC_CarControllerV4` for vehicle behavior and `RCC_Camera` for camera management.

---

3. Fields and Properties

- **`private List<RCC_CarControllerV4> _spawnedVehicles = new List<RCC_CarControllerV4>();`**
  An internal list holding references to all spawned vehicles. Instantiated once to avoid redundant spawning.

- **`public Transform spawnPosition;`**
  The position and rotation reference where new vehicles are spawned.

- **`public int selectedIndex = 0;`**
  Tracks the currently selected vehicle in `_spawnedVehicles`. Initially loaded from `PlayerPrefs`.

- **`public GameObject RCCCanvas;`**
  Reference to the RCC UI Canvas, which can be enabled or disabled depending on selection state.

- **`public RCC_Camera RCCCamera;`**
  Reference to the active RCC camera used for orbiting around vehicles and switching to gameplay camera modes.

- **`public string nextScene;`**
  Name of the scene to load once a vehicle is finally selected.

---

4. Core Functionality

Below are the key methods that govern how this script spawns, activates, and finalizes the vehicle selection process.

4.1 Initialization and `Start()`

```
private void Start() {
    if (!RCCCamera)
        RCCCamera = RCCSceneManager.activePlayerCamera;

    selectedIndex = PlayerPrefs.GetInt("SelectedRCCVehicle", 0);
    CreateVehicles();
```

```
}
```

1. **Camera Fallback**
    - If `RCCCamera` is not explicitly assigned, it attempts to grab the active camera from `RCCSceneManager`.
2. **Load Selection Index**
    - Retrieves the previously selected vehicle index via `PlayerPrefs.GetInt("SelectedRCCVehicle", 0)`—defaulting to 0 if no preference is stored.
3. **Create Vehicles**
    - Calls `CreateVehicles()` to spawn the entire vehicle list at scene start.

4.2 Vehicle Creation and Handling

```
private void CreateVehicles() {
    for (int i = 0; i < RCC_DemoVehicles.Instance.vehicles.Length; i++) {
        RCC_CarControllerV4 spawnedVehicle = RCC.SpawnRCC(
            RCC_DemoVehicles.Instance.vehicles[i],
            spawnPosition.position,
            spawnPosition.rotation,
            false,
            false,
            false
        );

        spawnedVehicle.gameObject.SetActive(false);
        _spawnedVehicles.Add(spawnedVehicle);
    }

    SpawnVehicle();

    if (RCCCamera && RCCCamera.GetComponent<RCC_CameraCarSelection>())
        RCCCamera.GetComponent<RCC_CameraCarSelection>().enabled = true;

    if (RCCCanvas)
        RCCCanvas.SetActive(false);
}
```

1. **Batch Spawning**
    - Iterates over `RCC_DemoVehicles.Instance.vehicles[]`, spawning each vehicle at the specified `spawnPosition`.
    - Parameters `(false, false, false)` mean:

        - *doNotRegisterAsPlayerVehicle* = `false`
        - *engineOff* = `false`
        - *isAI* = `false`
          (Interpretation may vary depending on your RCC implementation, but effectively these flags ensure no immediate control or registration.)
2. **Vehicle Deactivation**
    - After spawning, each vehicle is deactivated (`SetActive(false)`) until it is the chosen one.
3. **Storing References**
    - Each spawned vehicle is stored in `_spawnedVehicles` so it can be easily managed later.
4. **Activating the Current Selection**
    - Calls `SpawnVehicle()` to show the vehicle at `selectedIndex`.
5. **Camera and UI Configuration**
    - If an **orbit camera** script (`RCC_CameraCarSelection`) is attached to `RCCCamera`, it's enabled to allow rotational preview.
    - Disables the main RCC Canvas until a vehicle is confirmed selected.

4.3 Selection Navigation: Next and Previous Vehicle

```
public void NextVehicle() {
    selectedIndex++;
    if (selectedIndex > _spawnedVehicles.Count - 1)
        selectedIndex = 0;

    SpawnVehicle();
}

public void PreviousVehicle() {
    selectedIndex--;
    if (selectedIndex < 0)
        selectedIndex = _spawnedVehicles.Count - 1;

    SpawnVehicle();
}
```

- **`NextVehicle()` / `PreviousVehicle()`**
    - Increments or decrements `selectedIndex`.
    - Wraps around if the index goes out of range (using `%` logic effectively).
    - Calls `SpawnVehicle()` to show the newly selected car.

4.4 Spawning the Selected Vehicle

```
public void SpawnVehicle() {
    for (int i = 0; i < _spawnedVehicles.Count; i++)
        _spawnedVehicles[i].gameObject.SetActive(false);

    _spawnedVehicles[selectedIndex].gameObject.SetActive(true);
    RCCSceneManager.activePlayerVehicle = _spawnedVehicles[selectedIndex];
}
```

- **Disable All → Enable One**
  - Loops through `_spawnedVehicles` and disables all vehicles.
  - Activates only the vehicle at `selectedIndex`.
- **Set Active Player**
  - Temporarily sets `RCCSceneManager.activePlayerVehicle` to the chosen car.
  - This is mostly for preview purposes since full control is not enabled yet.

4.5 Finalizing Selection

```
public void SelectVehicle() {
    RCC.RegisterPlayerVehicle(_spawnedVehicles[selectedIndex]);
    _spawnedVehicles[selectedIndex].StartEngine();
    _spawnedVehicles[selectedIndex].SetCanControl(true);

    PlayerPrefs.SetInt("SelectedRCCVehicle", selectedIndex);

    if (RCCCamera && RCCCamera.GetComponent<RCC_CameraCarSelection>())
        RCCCamera.GetComponent<RCC_CameraCarSelection>().enabled = false;

    RCCCamera.ChangeCamera(RCC_Camera.CameraMode.TPS);

    if (RCCCanvas)
        RCCCanvas.SetActive(true);

    if (!string.IsNullOrEmpty(nextScene))
        OpenScene();
}
```

1. **Registering as Player**
   - Calls `RCC.RegisterPlayerVehicle()` to fully designate the selected car as the active player vehicle.
2. **Enable Control**
   - Starts engine (`StartEngine()`) and sets control (`SetCanControl(true)`).
3. **Persist Selection**

    - Writes `selectedIndex` to `PlayerPrefs` so it can be retrieved if the user reenters the menu or goes to a new scene.
4. **Camera and UI**
    - Disables the orbit camera script and switches to a TPS (third-person shooter) or chase camera mode.
    - Re-enables the main RCC Canvas for driving HUD.
5. **Scene Transition**
    - If `nextScene` is set, calls `OpenScene()` to load it.

4.6 Deselecting Vehicles

```
public void DeSelectVehicle() {
    RCC.DeRegisterPlayerVehicle();

    _spawnedVehicles[selectedIndex].transform.position = spawnPosition.position;
    _spawnedVehicles[selectedIndex].transform.rotation = spawnPosition.rotation;

    _spawnedVehicles[selectedIndex].KillEngine();
    _spawnedVehicles[selectedIndex].SetCanControl(false);

    Rigidbody vehicleRigidbody =
_spawnedVehicles[selectedIndex].GetComponent<Rigidbody>();
    vehicleRigidbody.ResetInertiaTensor();
    vehicleRigidbody.velocity = Vector3.zero;
    vehicleRigidbody.angularVelocity = Vector3.zero;

    if (RCCCamera && RCCCamera.GetComponent<RCC_CameraCarSelection>())
        RCCCamera.GetComponent<RCC_CameraCarSelection>().enabled = true;

    RCCCamera.ChangeCamera(RCC_Camera.CameraMode.TPS);

    if (RCCCanvas)
        RCCCanvas.SetActive(false);
}
```

- **De-register Vehicle**
    - Resets control by calling `RCC.DeRegisterPlayerVehicle()`.
- **Reset Transform**
    - Moves and orients the deselected vehicle back to `spawnPosition`.
- **Engine and Control Off**
    - Calls `KillEngine()` and `SetCanControl(false)`.
- **Physics Reset**
    - Clears velocity and angular velocity to avoid residual movement in the preview area.
- **Orbit Camera**
    - Re-enables the selection orbit camera script for the user to continue cycling vehicles.

- **Canvas**
    - Disables the UI if returning to a selection state.

4.7 Scene Transition

```
public void OpenScene() {
   SceneManager.LoadScene(nextScene);
}
```

A straightforward method to load the specified `nextScene` using Unity's `SceneManager`.

---

5. Usage Notes and Best Practices

1. **Vehicle Prefabs**
    - The array of vehicles (`RCC_DemoVehicles.Instance.vehicles`) must be set up correctly. Each prefab should have `RCC_CarControllerV4` attached.
2. **Spawn Position**
    - `spawnPosition` should be carefully placed in your scene so that vehicles are visible and not colliding with other objects.
3. **Orbit Camera**
    - Ensure the `RCC_CameraCarSelection` script is on your RCC camera if you want the user to freely rotate the camera around the previewed car.
4. **Canvas Management**
    - The `RCCCanvas` is hidden during selection to provide a clean UI. Once the vehicle is selected, it is activated to display driving HUD elements.
5. **Scene Flow**
    - If `nextScene` is left empty, the script will just enable the player vehicle and remain in the current scene, effectively letting you test or play in place.
6. **PlayerPrefs for Persistence**
    - Only the vehicle index is saved (`"SelectedRCCVehicle"`). If you need more data (e.g., custom colors or upgrades), store them separately or use a more advanced system.
7. **Extensibility**
    - You could integrate this script with your own UI logic (e.g., adding UI elements for selecting paint jobs, rims, etc.).
    - Also, consider hooking into `RCC_Customization_API` if you want to load or save custom stats for the selected car.

6. Summary

`RCC_CarSelectionExample` is a straightforward demonstration of how to create a **vehicle selection** or **showroom** interface in Realistic Car Controller. By spawning all vehicles up front, deactivating them, and allowing quick cycling with an orbit camera, players can easily preview vehicles. Once a choice is confirmed, the script handles final registration, enabling controls, and optionally transitions to the main gameplay scene.

This approach keeps spawning logic minimal and reuses each vehicle object for more efficient memory usage, ensuring a smooth preview-to-play experience for your RCC-based projects.