Table of Contents

1. Overview
2. Class Declaration
3. Fields and Properties
   3.1 Player References
   3.2 Configuration Flags
   3.3 Recording / Replay
   3.4 Scene Management Data
   3.5 Events
4. Unity Callbacks and Flow
5. Vehicle Spawn / Despawn Handling
   5.1 Player Vehicle Spawn
   5.2 AI Vehicle Spawn
   5.3 RCC Camera Spawn
   5.4 Vehicle Destruction
6. Terrain Initialization
7. UI and Display Controls
8. Record / Replay Controls
9. Player Vehicle Registration
10. Behavior and Camera Controls
11. Teleportation and Freeze Mechanics
12. Slow-Motion Handling
13. Integration with RCC_InputManager
14. Integration with BCG_EnterExit (Optional)
15. Cleanup and OnDisable
16. Usage Notes and Best Practices
17. Summary

1. Overview

`RCC_SceneManager` is the central manager for scenes using the Realistic Car Controller (RCC) system. It handles:

- Player vehicle registration and tracking.
- RCC camera management.
- RCC UI dashboard display.
- Optional record and replay functionality.
- Terrain data initialization.
- Events and callbacks related to vehicle spawning, destruction, and behavior changes.

It is typically placed in a **singleton** configuration (`RCC_Singleton<RCC_SceneManager>`), ensuring only one instance is active in the scene. It also manages specialized features such as slow-motion, teleportation, and optional third-party integrations.

---

2. Class Declaration

```
public class RCC_SceneManager : RCC_Singleton<RCC_SceneManager> {
    // Implementation details...
}
```

This class inherits from a generic `RCC_Singleton` base class, ensuring only one `RCC_SceneManager` instance exists at runtime.

---

3. Fields and Properties

This section covers the primary fields exposed by `RCC_SceneManager`. Many of these fields are publicly accessible to allow for dynamic setup and fine-tuning of RCC features.

3.1 Player References

- **`public RCC_CarControllerV4 activePlayerVehicle;`**
  The current player vehicle (car) under user control.

- **`public RCC_Camera activePlayerCamera;`**
  The active RCC camera following the player vehicle.

- **`public RCC_UI_DashboardDisplay activePlayerCanvas;`**
  The active UI dashboard for displaying speed, RPM, gear, etc.

- **`public Camera activeMainCamera;`**
  Reference to the scene's main camera (`Camera.main`) for quick access.

- **`private RCC_CarControllerV4 lastActivePlayerVehicle;`**
  Internally tracks the previous player vehicle to detect when a new vehicle becomes active.

3.2 Configuration Flags

- **`public bool registerLastSpawnedVehicleAsPlayerVehicle = true;`**
  Automatically registers the most recently spawned vehicle as the player vehicle if `true`.

- **`public bool disableUIWhenNoPlayerVehicle = false;`**
  Disables the UI dashboard canvas when there is no active player vehicle.

- **`public bool loadCustomizationAtFirst = false;`**
  When set to `true`, any saved customizations (e.g., paint color, wheels) are automatically applied to a newly registered player vehicle.

3.3 Recording / Replay

- **`public bool useRecord = false;`**
  Enables or disables the recording/replay functionality in the scene.

- **`public List<RCC_Recorder> allRecorders = new List<RCC_Recorder>();`**
  Stores references to all `RCC_Recorder` components found in the scene.

- **`public enum RecordMode { Neutral, Play, Record }`**
  Describes the global record/replay state across the entire scene.

- **`public RecordMode recordMode = RecordMode.Neutral;`**
  The current global state (Neutral, Play, or Record).

3.4 Scene Management Data

- **`private float orgTimeScale = 1f;`**
  Stores the original time scale value to restore after slow-motion toggles.

- **`public List<RCC_CarControllerV4> allVehicles = new List<RCC_CarControllerV4>();`**
  Master list of all `RCC_CarControllerV4` vehicles (player or AI) in the scene.

Note: If using the optional **BoneCracker Games EnterExit** package (`BCG_ENTEREXIT`), there is also:
`public BCG_EnterExitPlayer activePlayerCharacter;`

- **`public Terrain[] allTerrains;`**
  All active `Terrain` objects in the scene.

- **`public Terrains[] terrains;`**
  Stores additional data about each terrain in the scene (e.g., splatmaps, materials).

- **`public bool terrainsInitialized = false;`**
  Set to `true` after the terrain data is fully loaded and processed.

3.5 Events

- **`public delegate void onBehaviorChanged();`**
  `public static event onBehaviorChanged OnBehaviorChanged;`
  Fired whenever the global driving behavior changes (e.g., arcade to realistic).

- **`public delegate void onVehicleChanged();`**
  `public static event onVehicleChanged OnVehicleChanged;`
  Fired whenever the active player vehicle changes.

---

4. Unity Callbacks and Flow

`RCC_SceneManager` uses standard Unity callbacks (`Awake`, `Start`, `Update`, `OnDisable`) plus coroutines for async operations:

- **`Awake()`**
  - Overrides `Time.fixedDeltaTime` and `Application.targetFrameRate` if specified in RCC_Settings.
  - Instantiates telemetry UI if enabled in RCC_Settings.
  - Subscribes to RCC events for camera spawn, vehicle spawn/despawn, input manager triggers, etc.
  - Locates an `RCC_UI_DashboardDisplay` in the scene if available.
  - Locks cursor if specified by RCC_Settings.
- **`Start()`**
  - Begins gathering terrain data via a coroutine (`GetAllTerrains()`).
- **`Update()`**
  - Monitors changes to the active player vehicle.
  - Optionally disables the UI if there is no valid player vehicle.
  - Updates `activeMainCamera` reference to `Camera.main` if available.

    - Polls the first recorder in `allRecorders` to determine the global record/replay state.
- **`OnDisable()`**

    - Unsubscribes from all event hooks to prevent memory leaks and stale callbacks.

---

5. Vehicle Spawn / Despawn Handling

`RCC_SceneManager` listens for vehicle spawn and destruction events from both player vehicles and AI vehicles.

5.1 Player Vehicle Spawn

- **`RCC_CarControllerV4_OnRCCSpawned(RCC_CarControllerV4 RCC)`**
  Triggered whenever a player vehicle is spawned.
    - Adds the new vehicle to `allVehicles` if not already present.
    - If `useRecord` is true, ensures an `RCC_Recorder` component is attached or created.
    - Optionally assigns the newly spawned vehicle as the active player vehicle if `registerLastSpawnedVehicleAsPlayerVehicle` is `true`.

5.2 AI Vehicle Spawn

- **`RCC_AICarController_OnRCCAISpawned(RCC_AICarController RCCAI)`**
  Triggered whenever an AI vehicle spawns (wrapped by an `RCC_AICarController`).
    - Similar to player vehicles, ensures it is tracked in `allVehicles` and possibly assigned an `RCC_Recorder` if recording is enabled.

5.3 RCC Camera Spawn

- **`RCC_Camera_OnBCGCameraSpawned(GameObject BCGCamera)`**
  Triggered whenever an `RCC_Camera` is created.
    - Assigns the new `RCC_Camera` as `activePlayerCamera`.

5.4 Vehicle Destruction

- **`RCC_CarControllerV4_OnRCCPlayerDestroyed(RCC_CarControllerV4 RCC)`**
  Removes the destroyed vehicle from `allVehicles`.

- **`RCC_AICarController_OnRCCAIDestroyed(RCC_AICarController RCCAI)`**
  Same as above, but for AI vehicles.

Note: After vehicles are destroyed, the system re-checks all existing recorders to remove any with missing vehicle references (`CheckMissingRecorders()`).

---

6. Terrain Initialization

- **`IEnumerator GetAllTerrains()`**
  - This coroutine runs at `Start()` to gather all active `Terrain` objects into `allTerrains`.
  - For each terrain, it captures important data (alphamap width, height, splatmap information, etc.) into `terrains[]`.
  - Sets `terrainsInitialized = true` once complete.

This process is especially useful if you need terrain-specific interactions, such as friction adjustments or specialized wheel physics.

---

7. UI and Display Controls

- **`public void CheckCanvas()`**
  - If `disableUIWhenNoPlayerVehicle` is true, this method hides or modifies the `activePlayerCanvas` when no valid player vehicle exists.
  - If a valid vehicle is present, sets `activePlayerCanvas.displayType` to full display mode (unless in customization mode).

---

8. Record / Replay Controls

The scene manager can optionally control recording and playback of vehicle movements:

- **`public void Record()`**
  Initiates or stops recording on all `RCC_Recorder` instances.

- **`public void Play()`**
  Initiates or stops replay on all `RCC_Recorder` instances.

- **`public void Stop()`**
  Stops both recording and playback modes on all recorders.

- **`private IEnumerator CheckMissingRecorders()`**

    - Ensures no orphaned recorder components exist by removing any with a null `CarController`.

  Implementation detail: By default, the first recorder's mode (Record or Play) is used to reflect the global `recordMode` on the `RCC_SceneManager`.

---

9. Player Vehicle Registration

`RCC_SceneManager` provides several overloads for registering a vehicle as the active player:

1. **`public void RegisterPlayer(RCC_CarControllerV4 playerVehicle)`**

    - Sets the specified vehicle as `activePlayerVehicle`.
    - If `activePlayerCamera` exists, it targets the new vehicle.
    - Optionally applies previously saved customizations if `loadCustomizationAtFirst` is true.
2. **`public void RegisterPlayer(RCC_CarControllerV4 playerVehicle, bool isControllable)`**

    - In addition to above, forces `playerVehicle.SetCanControl(isControllable)`.
3. **`public void RegisterPlayer(RCC_CarControllerV4 playerVehicle, bool isControllable, bool engineState)`**

    - Allows further control to start or stop the vehicle's engine immediately.
4. **`public void DeRegisterPlayer()`**

    - Clears `activePlayerVehicle`.
    - Disables control on the old player vehicle.
    - Removes camera target if `activePlayerCamera` is present.

---

10. Behavior and Camera Controls

- **`public void SetBehavior(int behaviorIndex)`**
    - Updates the global behavior in `Settings` (e.g., switching from arcade to realistic).
    - Fires `OnBehaviorChanged` event to notify subscribers of the new behavior.
- **`public void ChangeCamera()`**
    - Cycles through different camera modes (hood, orbit, chase, etc.) on the active camera.

---

11. Teleportation and Freeze Mechanics

`RCC_SceneManager` supports teleporting vehicles:

- **`public void Transport(Vector3 position, Quaternion rotation)`**
    - Teleports `activePlayerVehicle` to the specified position and rotation.
    - Resets velocity and wheel torque.
    - Invokes a **freeze** coroutine for 1 second, during which player input is disabled and velocity is clamped to prevent glitching.
- **`public void Transport(RCC_CarControllerV4 vehicle, Vector3 position, Quaternion rotation)`**
    - Same as above, but allows teleporting a *specific* vehicle rather than the active vehicle.
    - Temporarily toggles `isKinematic` on the Rigidbody to prevent physics spikes during teleport.
- **`private IEnumerator Freeze(RCC_CarControllerV4 vehicle)`**
    - Disables control and zeroes out velocity for 1 second post-teleport.
    - Ensures stable re-orientation before re-enabling control.

---

12. Slow-Motion Handling

- **`private void RCC_InputManager_OnSlowMotion(bool state)`**
    - If `state` is `true`, sets `Time.timeScale` to a fraction (e.g., `0.2f`) for a slow-motion effect.
    - If `state` is `false`, restores `Time.timeScale` to `orgTimeScale`.

---

13. Integration with `RCC_InputManager`

`RCC_SceneManager` hooks into these input callbacks:

- **`RCC_InputManager.OnRecord`**
  Calls `Record()` to toggle record mode.
- **`RCC_InputManager.OnReplay`**
  Calls `Play()` to toggle replay mode.
- **`RCC_InputManager.OnSlowMotion`**
  Toggles slow-motion on/off.

These integrations allow centralized control of record/replay and slow-motion through user input or a custom UI.

---

14. Integration with BCG_EnterExit (Optional)

If **BCG_ENTEREXIT** is defined (BoneCracker Games' Enter-Exit system):

- **`public BCG_EnterExitPlayer activePlayerCharacter;`**
  Holds the reference to the currently active character.

- Event hooks like `BCG_EnterExitPlayer.OnBCGPlayerSpawned` / `OnBCGPlayerDestroyed` are used to track spawning and destruction of the player avatar.

- The manager can then coordinate the player avatar with vehicles (e.g., setting the camera to the newly entered vehicle).

---

15. Cleanup and `OnDisable`

When the object is disabled or destroyed, `RCC_SceneManager`:

- Unsubscribes from all static events (`OnBCGCameraSpawned`, `OnRCCPlayerSpawned`, etc.).
- Prevents further event callbacks from referencing destroyed objects or this manager instance.

---

16. Usage Notes and Best Practices

1. **Singleton Pattern**
   Ensure only one `RCC_SceneManager` is present in each scene. Having multiple managers can lead to conflicts in event handling.

2. **Automatic Registration**
   If you want to manually control which vehicle is the player, set `registerLastSpawnedVehicleAsPlayerVehicle = false` and use `RegisterPlayer(...)` at the appropriate time.

3. **UI Dashboard**

   - If `disableUIWhenNoPlayerVehicle` is true, keep in mind that UI toggles may hide the entire HUD, including essential speedometers, gear displays, etc.
   - For a custom HUD, consider using the events `OnVehicleChanged` to show or hide UI elements.

4. **Slow-Motion**
   This can affect all physics-based objects in the scene. Use with caution if other time-dependent logic is in place.

5. **Record / Replay**

   - Each vehicle can have its own `RCC_Recorder`.
   - Ensure that the right mode (Record or Play) is set globally or individually, depending on your requirements.
   - For large scenes, continuously recording can generate substantial data. Manage this carefully or provide in-game toggles.

6. **Behavior Switching**
   Calling `SetBehavior(int behaviorIndex)` will override the behavior set in RCC_Settings. If you want to revert to the default behavior, disable `Settings.overrideBehavior`.

7. **Teleportation**

   - The built-in transport methods reset wheel torque and enforce a freeze period, which is usually enough to avoid physics glitches.
   - If you have complex AI scripts, ensure they handle or ignore the freeze period properly.

---

17. Summary

`RCC_SceneManager` orchestrates key features in Realistic Car Controller projects:

- Tracks and manages **player vehicle** references, spawns, and despawns.
- Handles **UI** for speed, RPM, gear indicators, and toggles them based on vehicle presence.
- Optionally controls **record / replay** for advanced debugging or replay features.
- Initializes **terrain data** for advanced wheel and friction setups.
- Provides convenience methods for **teleportation**, **slow-motion**, and **behavior switching**.

Proper use of `RCC_SceneManager` ensures a cohesive, unified approach to scene-wide vehicle management, camera behavior, and specialized features like recording and slow motion. It is a core component for any Realistic Car Controller–based setup.