Table of Contents

1. Overview
2. Class Declaration
3. Record / Replay Flow
4. Data Structures
   4.1 RecordedClip
   4.2 PlayerInput
   4.3 PlayerTransform
   4.4 PlayerRigidBody
5. Runtime Lists and State
6. Key Methods
   6.1 Record()
   6.2 SaveRecord()
   6.3 Play()
   6.4 Play(RecordedClip _recorded)
   6.5 Stop()
   6.6 Replay() Coroutine
   6.7 Revel() Coroutine
7. Update and FixedUpdate Logic
8. Usage Notes and Best Practices
9. Example Integration
10. Summary

---

1. Overview

`RCC_Recorder` is a **Record / Replay** component within the **Realistic Car Controller (RCC)** ecosystem. It captures **vehicle input, transform, and rigidbody data** frame-by-frame while in *Record* mode, and later replays it in *Play* mode. This allows you to:

- **Save a driving session** for debug or demonstration.
- **Replay** that session to replicate exact motions and inputs.
- Potentially export or share replay data via `RCC_Records.Instance`.

It's typically attached to an RCC vehicle (`RCC_CarControllerV4`) and integrated with a higher-level manager or UI that toggles recording / playback states.

---

2. Class Declaration

public class RCC_Recorder : RCC_Core {

```
    // ...
}
```

- Inherits from `RCC_Core`, ensuring access to `CarController`, `Settings`, etc.
- Typically, you'll have one `RCC_Recorder` per vehicle that you want to record.

---

3. Record / Replay Flow

1. **Record**
    - Call `Record()` to start capturing data.
    - On every `FixedUpdate()`, the script stores the vehicle's input and transform info in arrays.
    - Stopping record (`Record()` again, toggling back to Neutral) finalizes the data into a `RecordedClip`.
2. **Replay**
    - Call `Play()` to replay the last `recorded` clip (or pass a specific clip).
    - The vehicle is put into **`externalController`** mode, meaning it ignores player inputs.
    - Each frame's input, position, and velocity are applied over time until the clip ends or is manually stopped.

---

4. Data Structures

4.1 `RecordedClip`

```
[System.Serializable]
public class RecordedClip {
    public string recordName;
    public PlayerInput[] inputs;
    public PlayerTransform[] transforms;
    public PlayerRigidBody[] rigids;

    public RecordedClip(PlayerInput[] _inputs,
                        PlayerTransform[] _transforms,
                        PlayerRigidBody[] _rigids,
                        string _recordName) {
        // ...
    }
}
```

- `recordName`: A label (e.g., “Demo_1”).
- `inputs`: Array of **PlayerInput** per frame.
- `transforms`: Array of **PlayerTransform** (position/rotation).
- `rigids`: Array of **PlayerRigidBody** (velocity/angular velocity).

When a recording finishes, these arrays are saved in `RCC_Records.Instance.records`.

4.2 `PlayerInput`

```
public class PlayerInput {
    public float throttleInput;
    public float brakeInput;
    public float steerInput;
    public float handbrakeInput;
    public float clutchInput;
    public float boostInput;
    public float fuelInput;
    public int direction;
    public bool canGoReverse;
    public int currentGear;
    public bool changingGear;
    public RCC_CarControllerV4.IndicatorsOn indicatorsOn;
    public bool lowBeamHeadLightsOn;
    public bool highBeamHeadLightsOn;
    // Constructor...
}
```

- Stores all **driving inputs** from the vehicle.
- Light states (low/high beams) and indicator states are also included.

4.3 `PlayerTransform`

```
public class PlayerTransform {
    public Vector3 position;
    public Quaternion rotation;
    // ...
}
```

- The vehicle’s **position** and **rotation** for each recorded frame.

4.4 `PlayerRigidBody`

```
public class PlayerRigidBody {
    public Vector3 velocity;
    public Vector3 angularVelocity;
```

```
    // ...
}
```

- The vehicle's **velocity** and **angularVelocity** for each frame (to preserve momentum during replay).

---

5. Runtime Lists and State

`List<PlayerInput> Inputs`, `List<PlayerTransform> Transforms`, `List<PlayerRigidBody> Rigidbodies`

- Temporary lists that accumulate data during `Mode.Record`.
- Once recording stops, a `RecordedClip` is built from them, then stored in `recorded`.

`public RecordedClip recorded`

- The **latest** clip that was saved or loaded for replay.

`public enum Mode { Neutral, Play, Record }`

- The recorder's current state.
- **`Mode.Neutral`**: Not recording or playing.
- **`Mode.Play`**: Replaying a clip.
- **`Mode.Record`**: Capturing data.

---

6. Key Methods

6.1 `Record()`

```
public void Record() {
    if (mode != Mode.Record) {
        mode = Mode.Record;
    } else {
        mode = Mode.Neutral;
        SaveRecord();
    }

    if (mode == Mode.Record) {
        Inputs.Clear();
        Transforms.Clear();
        Rigidbodies.Clear();
```

```
    }
}
```

- Toggles between **Record** and **Neutral**.
- If entering Record mode, clears old data.
- If exiting Record mode, calls `SaveRecord()` to finalize.

6.2 `SaveRecord()`

```
public void SaveRecord() {
    recorded = new RecordedClip(
        Inputs.ToArray(),
        Transforms.ToArray(),
        Rigidbodies.ToArray(),
        // Name pattern:
        RCC_Records.Instance.records.Count.ToString() + "_" + CarController.transform.name
    );
    RCC_Records.Instance.records.Add(recorded);
}
```

- Creates a new **`RecordedClip`** from the current session data.
- Appends it to `RCC_Records.Instance.records`.

6.3 `Play()`

```
public void Play() {
    // If no clip is recorded, do nothing
    // Toggle between Neutral/Play
    // If now in Play, start coroutines Replay() and Revel() to apply the data
}
```

- If `recorded` is null, returns.
- Toggles **`Mode.Play`** and sets `CarController.externalController = true`.
- Starts two coroutines:
    - **`Replay()`** (applies input each frame)
    - **`Revel()`** (applies rigidbody velocity each frame)

6.4 `Play(RecordedClip _recorded)`

```
public void Play(RecordedClip _recorded) {
    recorded = _recorded;
    // Similar logic to Play()
}
```

- Same as `Play()`, but loads a **specific** `RecordedClip` rather than the last recorded.

6.5 `Stop()`

```
public void Stop() {
    mode = Mode.Neutral;
    CarController.externalController = false;
}
```

- Cancels current recording or playback, returning to normal user input.

6.6 `Replay()` Coroutine

```
private IEnumerator Replay() {
    for(int i = 0; i < recorded.inputs.Length && mode == Mode.Play; i++) {
        // Set CarController inputs from recorded.inputs[i]
        yield return new WaitForFixedUpdate();
    }
    // End playback
    mode = Mode.Neutral;
    CarController.externalController = false;
}
```

- Steps through each frame in **`recorded.inputs`** and applies them to the car.
- Each iteration waits for `WaitForFixedUpdate()` to match the physics cycle.
- Once complete, returns to neutral state.

6.7 `Revel()` Coroutine

```
private IEnumerator Revel() {
    for(int i = 0; i < recorded.rigids.Length && mode == Mode.Play; i++) {
        CarController.Rigid.velocity = recorded.rigids[i].velocity;
        CarController.Rigid.angularVelocity = recorded.rigids[i].angularVelocity;
        yield return new WaitForFixedUpdate();
    }
    // End playback
    mode = Mode.Neutral;
    CarController.externalController = false;
}
```

- Synchronizes the vehicle's **Rigidbody** velocity each frame to ensure accurate replay of dynamics.

---

7. Update and FixedUpdate Logic

- `Awake()`: Initializes the **Lists** (`Inputs`, `Transforms`, `Rigidbodies`).
- `FixedUpdate()`:
    - If `Mode.Record`, it appends current input/transform/rigidbody data to the lists.
    - If `Mode.Play`, ensures `CarController.externalController = true`.
    - If `Mode.Neutral`, does nothing special.

---

8. Usage Notes and Best Practices

1. **Synchronized Framerate**
    - Data is recorded and replayed in **FixedUpdate()**. This helps keep the physics consistent.
    - If your game runs at variable `fixedDeltaTime`, playback speed could vary.
    - For consistent replays, keep a stable fixed timestep (or store the actual delta time with each frame).
2. **Multiple Vehicles**
    - Each vehicle can have its own `RCC_Recorder` if you want to record multiple cars simultaneously.
    - You can store multiple `RecordedClip`s in `RCC_Records.Instance`.
3. **Trim / Edit**
    - If you want partial replays or skipping frames, you can manipulate the `RecordedClip` arrays before re-playing.
4. **External Controller**
    - During playback, `CarController.externalController = true` means the car **ignores player input**.
    - Stopping playback sets `externalController = false`, restoring normal controls.
5. **Storage**
    - By default, recorded data is saved in-memory to `RCC_Records`. You can write your own logic to save or load from disk.
6. **Performance**
    - Long recordings can produce large arrays. Keep track of memory usage. Consider compressing or limiting capture frequency if needed.

---

9. Example Integration

1. **Add `RCC_Recorder`** to your vehicle prefab or instantiate it at runtime.

Trigger Recording (e.g., via a UI button):
RCC_Recorder myRecorder = myVehicle.GetComponent<RCC_Recorder>();
myRecorder.Record(); // toggles record on/off

2.

Trigger Playback:
myRecorder.Play(); // replays the last 'recorded' clip

3.

Stop:
myRecorder.Stop(); // cancels record or playback

4.

---

10. Summary

`RCC_Recorder` enables **full record and replay** functionality for RCC vehicles. It:

- Collects **player inputs**, **transforms**, and **rigidbody** states during **Record** mode.
- Plays them back exactly, overriding normal user input, in **Play** mode.
- Stores session data in **`RecordedClip`** objects.
- Integrates with **`RCC_Records`** for global record management.

This system is perfect for **replay cameras**, **demonstrations**, **ghost cars** in time trials, or **automated AI** demonstration within Realistic Car Controller.