Table of Contents

1. Overview

`RCC_AICarController` is the **AI driver** component for **Realistic Car Controller** (RCC). It directs the assigned vehicle to **follow waypoints**, **chase** a target, or **follow** a target. It also provides **raycast-based obstacle avoidance**, reversing logic, and optional **speed limiting**. The AI sets external inputs on the underlying `RCC_CarControllerV4`, effectively taking control away from user input.

2. Class Declaration

```
[RequireComponent(typeof(RCC_CarControllerV4))]
public class RCC_AICarController : RCC_Core {
   // ...
}
```

- Inherits from **`RCC_Core`**, granting usage of shared RCC utilities and references.
- Requires an `RCC_CarControllerV4` on the same GameObject, so it can directly drive the vehicle.

---

3. Core Purpose and Features

1. **Navigation**
   - **FollowWaypoints** using a container of waypoints.
   - **ChaseTarget** (e.g., a player or other object).
   - **FollowTarget** from a distance.
2. **Raycast-based** obstacle detection and avoidance.
3. **Reversing** logic if stuck.
4. **Speed limiting**, **lap counting**, **brake zones**.
5. **External** input feed to the `CarController` (`CarController.externalController = true`).

---

4. Fields and Configuration

4.1 Waypoint Navigation

- **`public RCC_AIWaypointsContainer waypointsContainer`**: Holds a list of waypoints for the car to follow in **FollowWaypoints** mode.
- **`public int currentWaypointIndex`**: The current waypoint we're heading towards.
- **`public int lap`**: Tracks how many laps are completed if the waypoint list is looped.

4.2 Target Following / Chasing

- **`public string targetTag = "Player"`**: AI searches for objects with this tag to chase or follow.
- **`public enum NavigationMode { FollowWaypoints, ChaseTarget, FollowTarget }`**

- `public Transform targetChase`: The actual transform the AI is chasing/following.
- `public List<Transform> targetsInZone`: Potential chase/follow targets within a detection radius.

4.3 Obstacle Avoidance

- `public bool useRaycasts = true`: If enabled, the AI casts forward and angled rays to detect obstacles.
- `public float raycastLength = 3f` and `public float raycastAngle = 30f` define the range and spread of rays.
- `public LayerMask obstacleLayers`: Layers considered obstacles.
- `private bool raycasting = false`: True if any ray hit an obstacle.
- `private float rayInput = 0f`: Additional steering from the raycast to avoid the obstacle.

4.4 Steering/Throttle/Brake Inputs

- `public float steerInput, throttleInput, brakeInput, handbrakeInput` store final values.
- Computed by `Navigation()` each frame, then fed into `RCC_CarControllerV4`.

4.5 Speed Limits and Lap Tracking

- `public bool limitSpeed`: If true, we clamp at `maximumSpeed`.
- `public float maximumSpeed = 100f`: Speed limit for the AI.
- `public bool stopAfterLap` / `public int stopLap = 10`: AI can stop after completing a certain number of laps.

4.6 Brake Zones

- `public List<RCC_AIBrakeZone> brakeZones`: Lists brake zones within detection range.
- `public RCC_AIBrakeZone targetBrake`: The closest zone we must slow down in.

4.7 Events

- `public static event onRCCAISpawned(RCC_AICarController RCCAI)`: Fired when AI car is enabled/spawned.
- `public static event onRCCAIDestroyed(RCC_AICarController RCCAI)`: Fired when AI car is disabled/destroyed.

---

5. Initialization

5.1 Awake()

- If `waypointsContainer` is null, tries to find **`RCC_AIWaypointsContainer`** in the scene.
- Creates an internal `NavMeshAgent` named `"Navigator"` so the AI can get direction from Unity's pathfinding logic.

5.2 OnEnable()

- **`CarController.externalController = true`** so the vehicle is AI-driven.
- Invokes **`OnRCCAISpawned(this)`** event.

---

6. Update Flow

6.1 Update()

- If AI is active (`CarController.canControl`), updates:
  - Possibly resets `maximumSpeed` if `limitSpeed` is false.
  - Moves the `navigator` transform to the front wheels area.
  - **`CheckTargets()`** if in chase or follow mode.
  - **`CheckBrakeZones()`** for upcoming slow-down areas.

6.2 FixedUpdate()

1. If `CarController.canControl` is false, return.
2. **`FixedRaycasts()`** if `useRaycasts` is true, tries to avoid obstacles by steering away.
3. **`Navigation()`** sets `steerInput`, `throttleInput`, `brakeInput`, `handbrakeInput` for the AI logic.
4. **`CheckReset()`** if stuck, switch to reversing.
5. **`FeedRCC()`** inputs to `CarController`.

---

7. Navigation Logic

7.1 FollowWaypoints Mode

1. Requires **`waypointsContainer`** not null and has at least one waypoint.

2. If distance to next waypoint < that waypoint's radius, increment `currentWaypointIndex`.
3. If we cycle through all waypoints, increment `lap`. If `stopAfterLap` is true and `lap` >= `stopLap`, we call **`Stop()`**.
4. Sets `navigator.SetDestination(nextWaypointPos)`.
5. Adjusts throttle/brake based on distance or target speed in the waypoint.
6. If speed is high, reduce throttle or add brake proportionally to turn angle.

7.2 ChaseTarget Mode

1. Must have a **`targetChase`** transform. If none, calls **`Stop()`**.
2. `navigator.SetDestination(targetChase.position)`.
3. Usually sets `throttleInput = 1f`, unless speed is too high (then reduce).
4. If turning, also blend in brake.

7.3 FollowTarget Mode

1. Also sets `navigator.SetDestination(targetChase.position)`.
2. Distance-based logic: if close to the target (`stopFollowDistance`), reduce or zero throttle and raise brake.
3. If far, accelerate more.
4. If speed is high, also reduce throttle.

---

8. Obstacle Avoidance (Raycasts)

- Casts 5 rays from the front: center, ±(raycastAngle/3), ±(raycastAngle).
- Any hits on **`obstacleLayers`** sets **`rayInput`**, a steering offset.
- If absolute `rayInput` > 0.5f, we ignore the normal navigation steer for that moment.
- If `raycasting` is true, an `obstacle` is set to the hit object.

---

9. Reversing Logic

- If speed < 5 and velocity is small for >2 seconds, we go **`reversingNow = true`**.
- If stuck for 4 seconds or speed >25, we revert to forward.
- While reversing, we set `throttleInput=0, brakeInput=1` so the car goes backward (car's `direction` is effectively -1).

---

10. Feeding Inputs to RCC

```
private void FeedRCC() {
    CarController.throttleInput = ...
    CarController.brakeInput = ...
    CarController.steerInput = ...
    CarController.handbrakeInput = ...
}
```

- If not changing gears or cutGas, passes the AI's `throttleInput, brakeInput, steerInput, handbrakeInput` to the `RCC_CarControllerV4`.
- This effectively **drives** the vehicle from the AI logic.

---

11. Target and Brake Zone Management

- **`CheckTargets()`**: Every 1 second, we do a sphere overlap of radius `detectorRadius` around the AI. If objects have `targetTag`, add them to `targetsInZone`. We remove any that left range or is inactive, then pick the closest.
- **`CheckBrakeZones()`**: Similarly collects `RCC_AIBrakeZone` references and picks the closest as `targetBrake`.

---

12. Stopping the AI

- **`Stop()`** sets `throttleInput=0, brakeInput=0, steerInput=0, handbrakeInput=1`. The car halts.

---

13. Disabling the AI

- On `OnDisable()`, sets `CarController.externalController = false`. The user could then drive it.
- Invokes **`OnRCCAIDestroyed(this)`** event.

---

14. Usage Notes and Best Practices

1. **Attach** this script to a car with `RCC_CarControllerV4`.
2. **Waypoints**

  - If using **FollowWaypoints**, ensure an `RCC_AIWaypointsContainer` with at least one waypoint is present.

3. **Chase** or **Follow**
   - The script tries to find an object with `targetTag` within `detectorRadius`. If none found, it halts.
4. **Raycasts**
   - `useRaycasts` can be toggled off for simpler pathing. Ray-based obstacle avoidance is basic but helps avoid collisions.
5. **Performance**
   - For many AI cars with large `detectorRadius`, the sphere overlap every second can be costly. Consider chunking or culling.
6. **Stop After Lap**
   - If `stopAfterLap` is true and `lap >= stopLap`, the AI calls `Stop()`.
7. **Interaction with** `RCC_CarControllerV4`
   - The `CarController` is placed in **external controller** mode. The AI sets `throttleInput` etc. every frame.
8. **Smoothed Steering**
   - If `smoothedSteer`, we interpolate steer from old to new over time.

---

15. Summary

`RCC_AICarController` is a straightforward **AI driver** for **Realistic Car Controller** vehicles. By setting **NavigationMode**:

- **FollowWaypoints**
- **ChaseTarget**
- **FollowTarget**

the script uses a **NavMeshAgent** plus optional **raycast** obstacle avoidance to produce steering, throttle, brake, and handbrake inputs. It also handles reversing if stuck, speed limiting, brake zones, lap counting, and vantage targeting. This gives you a simple but effective way to make AI-driven RCC cars.