

# RACE TRACK GENERATOR - ULTIMATE

## RACE MANAGER

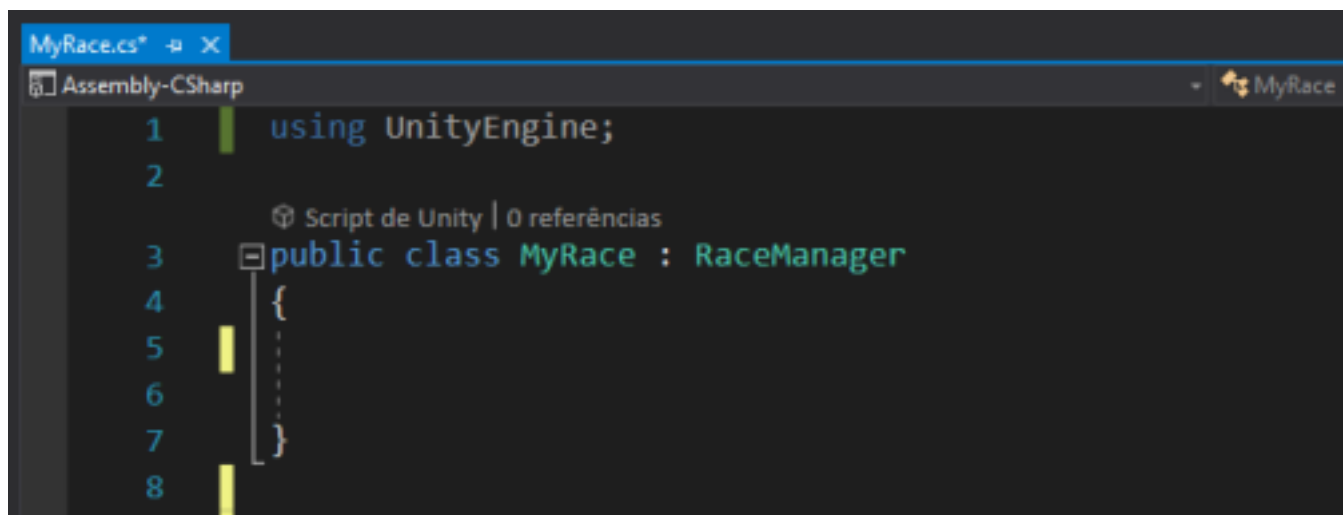
### How to Create a Custom Race Manager Script:

This guide will walk you through the steps to create a custom race manager script using the RTG Ultimate package, allowing you to fully control and customize race behavior in your Unity project.

#### Step 1: Create a New Script

1. Open your Unity project.
2. In the **Project** window, right-click and select **Create > C# Script**.
3. Name your script (e.g., MyCustomRaceManager).

#### Step 2: Extend RaceManager Instead of MonoBehaviour



The screenshot shows a C# script named 'MyRace.cs' in a code editor. The script starts with 'using UnityEngine;' on line 1. On line 3, it defines a public class 'MyRace' that inherits from 'RaceManager'. The class body is currently empty, with lines 4 through 7 showing the opening and closing curly braces. A tooltip above the class name indicates 'Script de Unity | 0 referências'. The file explorer on the right shows the script is part of an 'Assembly-CSharp' project.

```
1 using UnityEngine;
2
3 public class MyRace : RaceManager
4 {
5 }
6
7
8
```

1. Open the newly created script in your C# editor (such as Visual Studio).
2. Modify the script to extend RaceManager instead of MonoBehaviour to inherit all race management functionalities:
3. Create an Empty GameObject in the scene and attach the script to it.

## Step 3: Provide Information in the Inspector

### 1. Cars in the Race:

- This section contains an array where you can add the cars that will participate in the race

Define the number of cars and assign them to the race by dragging and dropping prefabs into the array

- The package comes with player-controlled cars and AI-controlled cars, located in:

1. Assets\RTG Ultimate\Cars\Prefabs\Player Cars (for player cars)
2. Assets\RTG Ultimate\Cars\Prefabs\AI Cars (for AI cars)

- You can include up to two player-controlled cars (for Split-Screen mode), and the remaining cars must be AI-controlled.

### 2. Number of Laps:

- Set the number of laps for the race in this field.

### 3. Rubber Band AI:

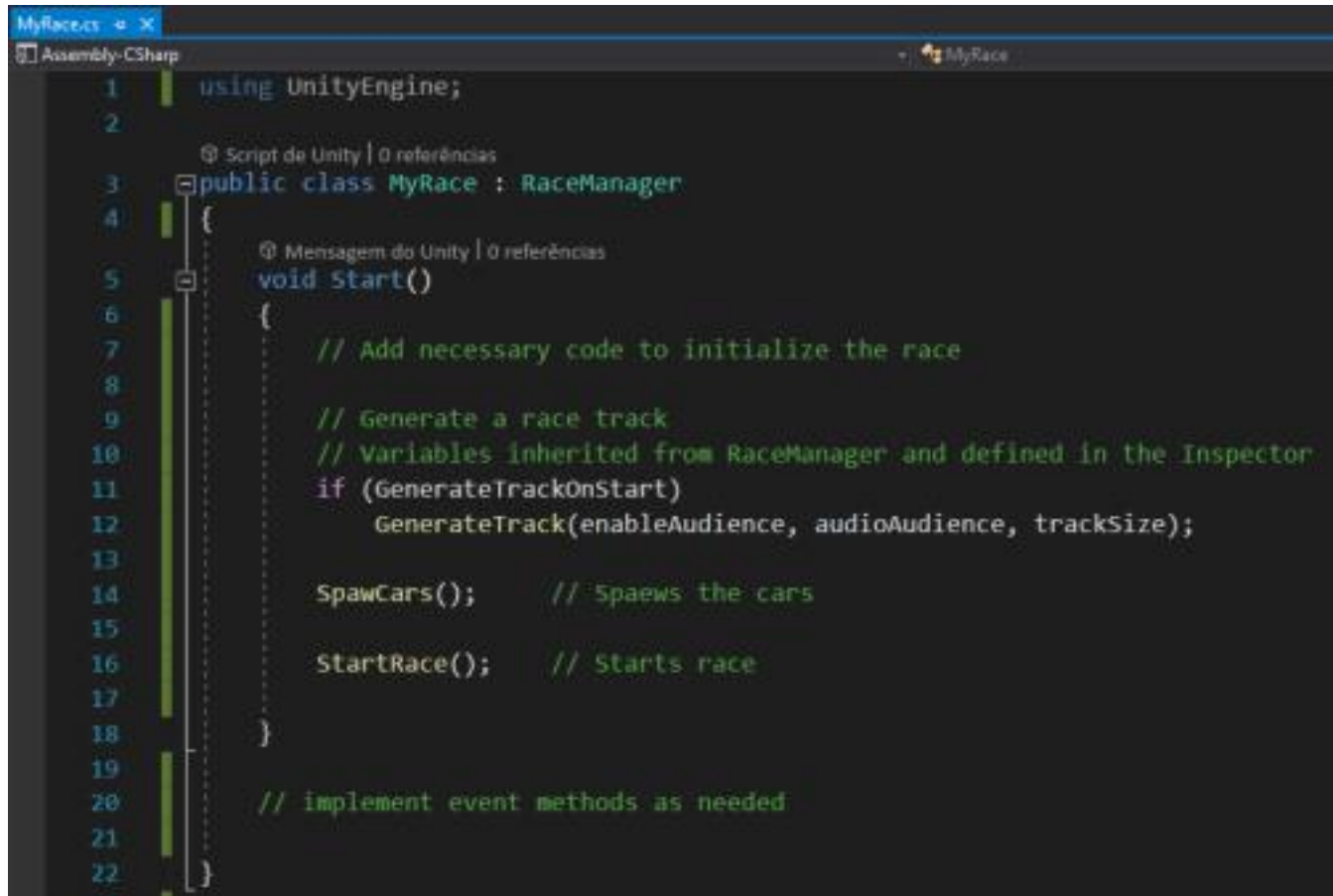
- Enable or disable adaptive AI behavior using this option.
- When enabled, the AI will dynamically adjust its speed to maintain competitive distance with player-controlled cars, providing a balanced racing experience.

### 4. Race Track Definitions:

- Choose whether to enable "**Generate Track on Start**".
- If enabled, set the parameters for the new track, such as size, and specify whether there will be an audience (crowd).

## Step 4: Insert Necessary Code in Start Method

Inside your custom script, include the following code in the Start() method to set up the race:



```
1 using UnityEngine;
2
3 [Script de Unity | 0 referências]
4 public class MyRace : RaceManager
5 {
6     [Mensagem do Unity | 0 referências]
7     void Start()
8     {
9         // Add necessary code to initialize the race
10
11         // Generate a race track
12         // Variables inherited from RaceManager and defined in the Inspector
13         if (GenerateTrackOnStart)
14             GenerateTrack(enableAudience, audioAudience, trackSize);
15
16         SpawCars(); // Spawns the cars
17
18         StartRace(); // Starts race
19     }
20
21     // implement event methods as needed
22 }
```

1. **GenerateTrack** (bool enableAudience = true, bool audioAudience = true, int trackSize = 1):

This method will generate a race track

```
// Generate a race track if the 'GenerateTrackOnStart' option is enabled in the Inspector.
if (GenerateTrackOnStart)
    GenerateTrack();
```

**optional parameters:**

**GenerateTrack** (bool enableAudience = true, bool audioAudience = true, int trackSize = 1)

bool enableAudience -> Defines if there will be an audience in the stands

bool audioAudience -> Defines if there will be crowd noise

int trackSize (1,2 or 3) -> Defines if the generated track will be (1) small (2) large (3) Random

2. **SpawCars():**

Spawns the cars registered in the Inspector but does not start the race.

### 3. **StartRace**(float timeToStart = 0):

Cars have been spawned, but are not enabled to race

StartRace() enables all cars to start race, then triggers the "On\_Race\_Start" event

**optional parameters:**

timeToStart: How long to wait before starting

3 - 2 - 1 - GO!

If you add the DisplayText UI to your scene, you can easily include a Countdown at the start of the race and also display messages on the screen that will be displayed for a specified period of time.

DisplayText UI can be found in "Assets\RTG Ultimate\Race Manager\UI"

To implement the Countdown at the start of the race

Just insert this code:

```
//Spawn the cars that are registered in the Inspector
SpawCars();

// if the 'displayText UI' was added to the scene
if (displayText)
{
    // Show Countdown to start the race -> Parameter: Number of seconds to Countdown
    displayText.StartCountDown(3);
}

StartRace(4); // Parameter: Number of seconds to start the race
```

To display texts:

**ShowMessage**(string msg, float duration = 0, int fontsize = 32, string msg2 = "", int fontsize2 = 22)

Displays a message on the screen, either permanently or for a specified duration

Example:

```
displayText.ShowMessage(car.name + " was Eliminated!", 2, 22);
```

Parameters:

**msg**: message to be displayed

**duration(optional)**: Time interval for which the message will be displayed, in seconds (0 = permanent)

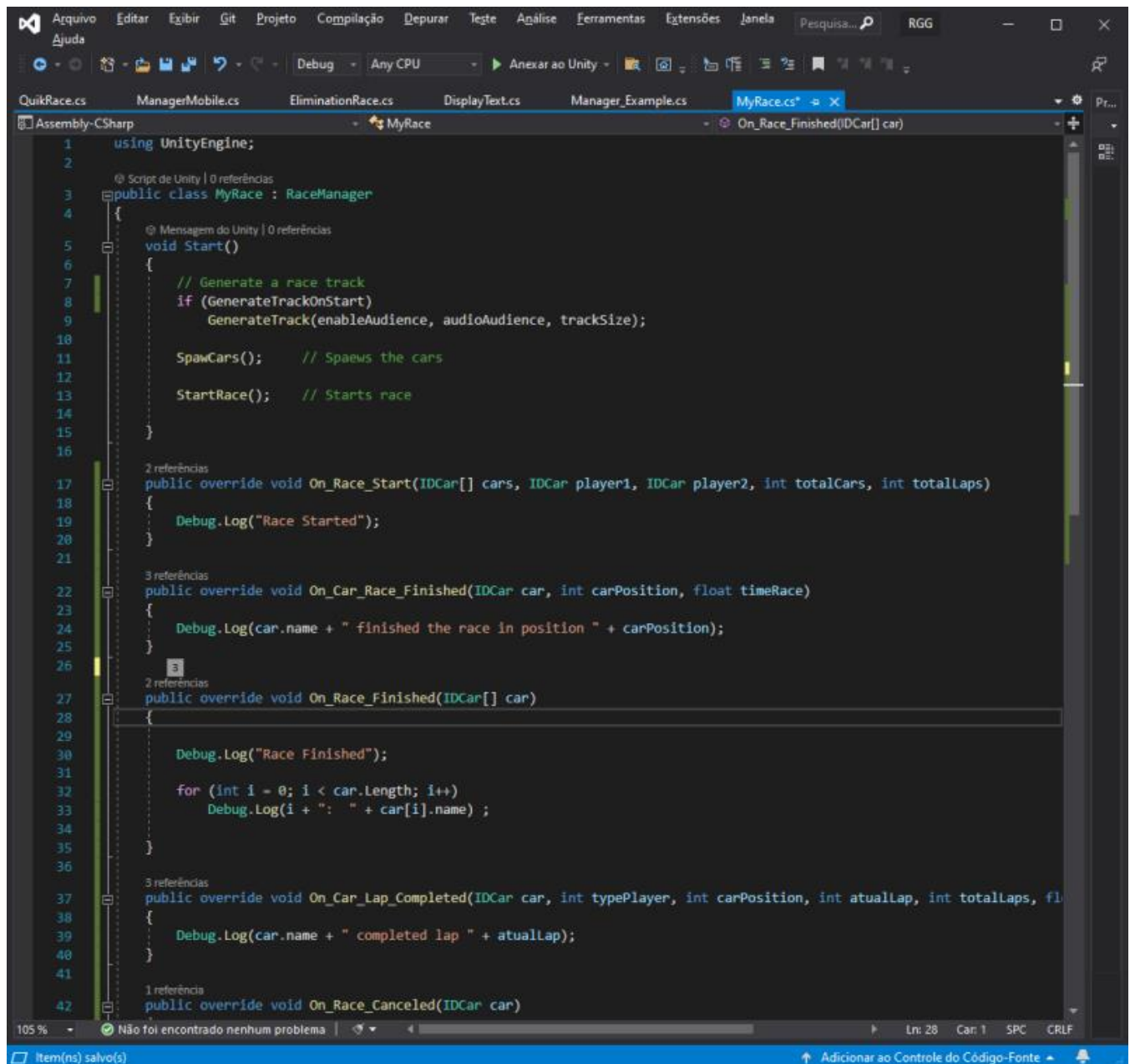
**fontsize(optional)**: Font size for the msg text

**msg2(optional)**: secondary message to be displayed below msg

**fontsize2(optional)**: Font size for the msg2 text

## Step 5: Understand the Inherited Methods and Their Usage

Your custom script inherits various methods from the RaceManager class. These methods are essential for controlling race behavior and responding to game events.



```
1 using UnityEngine;
2
3 @ Script de Unity | 0 referências
4 public class MyRace : RaceManager
5 {
6     @ Mensagem do Unity | 0 referências
7     void Start()
8     {
9         // Generate a race track
10         if (GenerateTrackOnStart)
11             GenerateTrack(enableAudience, audioAudience, trackSize);
12
13         SpawCars(); // Spawns the cars
14
15         StartRace(); // Starts race
16     }
17
18     2 referências
19     public override void On_Race_Start(IDCar[] cars, IDCar player1, IDCar player2, int totalCars, int totallaps)
20     {
21         Debug.Log("Race Started");
22     }
23
24     3 referências
25     public override void On_Car_Race_Finished(IDCar car, int carPosition, float timeRace)
26     {
27         Debug.Log(car.name + " finished the race in position " + carPosition);
28     }
29
30     2 referências
31     public override void On_Race_Finished(IDCar[] car)
32     {
33         Debug.Log("Race Finished");
34
35         for (int i = 0; i < car.Length; i++)
36             Debug.Log(i + ": " + car[i].name);
37     }
38
39     3 referências
40     public override void On_Car_Lap_Completed(IDCar car, int typePlayer, int carPosition, int atualLap, int totallaps, fl
41     {
42         Debug.Log(car.name + " completed lap " + atualLap);
43     }
44
45     1 referência
46     public override void On_Race_Canceled(IDCar car)
```

## Game Events:

These methods are triggered when certain situations occur during the race, and you can use them to control the flow of your game. For example, you could display your game's menu screen on the 'On\_Race\_Finished' event or show the new track record information on the 'On\_Car\_Lap\_Completed' event after checking the lap time provided as a parameter.

### 1. **On\_Race\_Start**(IDCar[] cars, IDCar player1, IDCar player2, int totalCars, int totalLaps)

- Triggered when the race starts.
- **Parameters:**
  - cars: Array of all cars participating in the race (set in the Inspector).
  - player1: The car controlled by the first player.
  - player2: The car controlled by the second player, if any.
  - totalCars: Total number of cars in the race.
  - totalLaps: Total number of laps required to complete the race.
- Example use:  
Display player positions and lap information on the UI.

### 2. **On\_Race\_Finished**(IDCar[] cars)

- Triggered when the race ends.
- **Parameters:**
  - cars: An array of all cars in the race, sorted by their finishing order.
- Example use:  
Display the race results or show a summary screen.

### 3. On\_All\_CarPlayer\_Finished()

- Triggered when the only or last human-controlled car finishes the race (player1 and player2).  
All cars that have not yet finished the race are AI-Cars.
- Example use:  
End the race and bring up the main screen a few seconds after the player finishes the race:

```
2 referências
public override void On_All_CarPlayer_Finished()
{
    /*
     Occurs when the only or last human-controlled car finishes the race (player1 and player2).
     All cars that have not yet finished the race are AI-Cars.
    */

    //Code Example:
    // Force finish the race in 8 seconds, even if some AICar has not finished yet
    Invoke("ForceFinalize", 8);
}

0 referências
void ForceFinalize()
{
    FinalizeRace();

    /*
     Ex: Invoke MainMenu
    */
}
```

### 4. On\_Car\_Lap\_Completed(IDCar car, int typePlayer, int carPosition, int currentLap, int totalLaps, float lapTime)

- Triggered every time a car completes a lap.
- **Parameters:**
  - car: The car that completed the lap.
  - typePlayer: The type of car (0 = AI, 1=player1, 2=player2).
  - carPosition: The car's position at the moment it completed the lap.
  - currentLap: The current lap number the car is on.
  - totalLaps: Total laps required to finish the race.
  - lapTime: The time it took for the car to complete the lap.
- Example use:  
Update lap time display using  
TimeSpan.FromSeconds(lapTime).ToString(@"mm\:ss\.fff").

5. **On\_Car\_Race\_Finished**(IDCar car, int carPosition, float raceTime)

- Triggered when a car finishes the race.
- **Parameters:**
  - car: The car that finished the race.
  - carPosition: The position the car finished in.
  - raceTime: The time it took for the car to complete the race.
- Example use:  
Show the car's final position and race time on the leaderboard.

6. **On\_Car\_Change\_Race\_Position**(IDCar car)

- Triggered when a car changes its position during the race (when overtaking or being overtaken by another car).
- **Parameters:**
  - car: The car that changed its position.
- Example use:  
Update the display of the car's race position in real-time.

## Control Events:



These methods represent player control actions but are not related to controlling the car itself.

### 1. **On\_Select\_Pressed(IDCar car)**

- Triggered when the player presses the Select button while the game is paused.
- **Parameters:**
  - car: The car belonging to the player who pressed the button.
- Example use:  
Restart the race or return to the main menu.

### 2. **On\_Menu\_Pressed(IDCar car)**

- Triggered when the player presses the Menu button (B/Xbox or Esc key).
- **Parameters:**
  - car: The car belonging to the player who pressed the button.
- Example use:  
Pause or resume the race and display the menu.

## Refer to Example Scenes

Explore the example scenes included with the package to see practical implementations of these event methods. These scenes can serve as a base for your custom scripts.

## Watch Tutorial Videos

For more detailed guidance, please refer to the tutorial videos available at:

<https://masterpixel3d.com/rtg>.