Race Positioning System: Guide

By Solution Studios

Contents:

Introduction:

Firstly, thank you for purchasing the Race Positioning System. (Previously known as the Car Positioning System)

The Race Positioning System can be used to add calculating race standings e.g. 1st, 2nd, 3rd etc, to your scene. The system is written in C# and is modular. Once you have the main things sorted (setting up PositionSensors), you can add what you like to make your game unique.

The majority of your time spent using the system will be using either Custom Editors in the Inspector window for each component, or using the Editor Window. To implement the Race Positioning System into your scene, just open the Editor Window in 'Window>RPS Editor Window' and follow the steps. We also advise using the tutorial in this pdf guide and following it step by step.

The Race Positioning System also includes 4 sets of Position Textures for race standings e.g. 1st, 2nd, 3rd etc. You will find these in a compressed .zip folder within Assets>RPS called 'PositionTextures.zip'.

How it works:

While setting up your scene using the RPS Editor Window you will place many PositionSensors around your track. These PositionSensors are just empty GameObjects with one script attached. The attached script gives a PositionSensor a position number. As you go around the track the position numbers on the PositionSensors should increase.

Any object with an RPS_Postion script attached can then find the nearest PositionSensor and from it, find the nearest position number. This position number gives the object an approximate position around the track. A further calculation is also made to find the percentage distance between the last and next PositonSensors. All of this information together with the lap number (if a RPS_Lap script is attached) is compared with all of the other RPS_Position scripts in the race, putting the objects in order of 1st to last.

An RPS_Lap script can be added to any object with an RPS_Position script to give it a lap system. This has to be done to each RPS_Position script individually in the Inspector window. The RPS_Lap script keeps track of the lap number the object is on and can freeze the position at the end of the race for the finished race standings. When the nearest position number is 0 and then becomes the highest of all position numbers in the next frame, it means the object has passed the start/finish line so the lap number increases. Similarly, if the nearest position number decreases from the highest to 0 then the object has gone back a lap (the wrong way over the start/finish line). This is how the lap system works.

If an RPS_Checkpoints script is added to an object with an RPS_Postion script (again, done in the Inspector window), a checkpoint system is added to that object. And this has to be done to each object individually (but you might not want AI cars to have a checkpoint system as if they miss a checkpoint, they will have to do a whole lap to pass it again unless it is built into the AI to go back). The RPS_Checkpoints script stores a list of checkpoints for each lap done for a RPS_Position script. It stores whether each of these checkpoints has been passed or not. The RPS_Position script then uses this to check if all checkpoints have been passed when going onto the next lap and to limit the nearest position number variable to a checkpoint which has been missed.

If you are using any RPS_Checkpoints scripts in your scene, you will need your own system for detecting when a checkpoint is passed and calling the CheckpointPassed function on the RPS_Checkpoints script(s). In the demo scenes we used a trigger based system. This is explained in more detail in the 'Tutorial – Adding a Checkpoint System' part of this guide.

Tutorial

Introduction:

This is a detailed step-by-step tutorial for using the Race Positioning System in your project. We advise all first-time users to read through this before using the package.

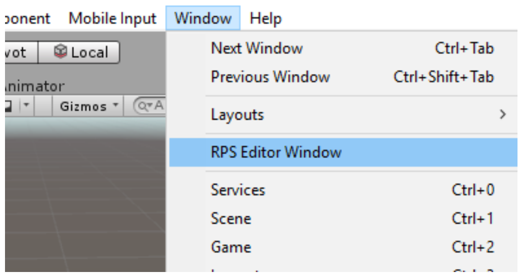
Before you start using the Race Positioning System in your scene, you need a few things setup: a track, a player (car/human/boat/etc), AI objects should also be setup, any checkpoint objects and the start/finish line should be in the scene with appropriate colliders.

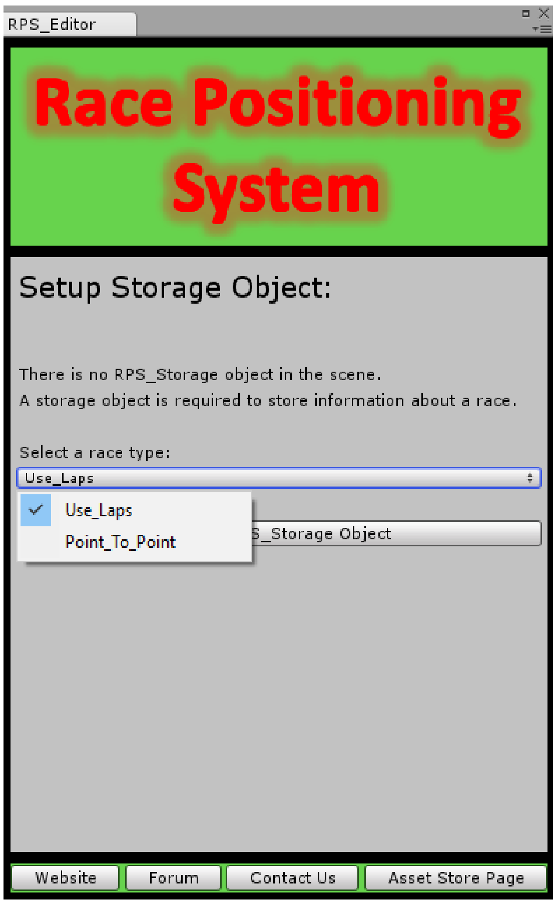
If you are using checkpoints, you can use a trigger based system. Each checkpoint should have a collider attached to it. (with isTrigger = true)

Also, if you want to detect when the player crosses the finish line without relying on the lap system, you should also add a collider (isTrigger = true) to the finish line.

Setting up Position Sensors:

To start using the Race Positioning System in your scene, open the Editor Window. This can be done by pressing 'Window' in the top menu bar and 'RPS Editor Window'.





Setup Storage Object:

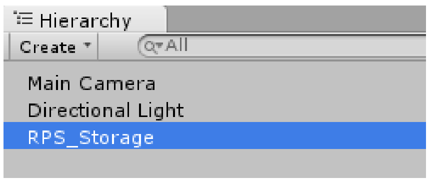
There is no RPS_Storage object in the scene.
A storage object is required to store information about a race.

Select a race type:
Use_Laps
✓ Use_Laps
Point_To_Point
S_Storage Object

The Editor Window should open telling you to setup a storage object. A storage object just stores all of the necessary data for the race in the scene. It also acts as the parent object to all PositionSensors.

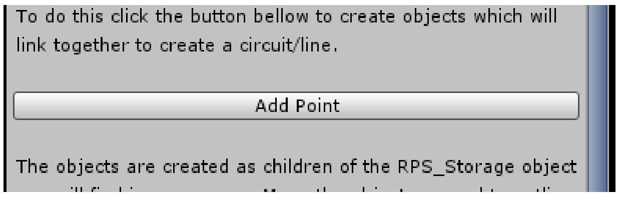
Select the appropriate race type from the drop down, either 'Use_Laps', allowing you to place PositionSensors in a circuit, or 'Point_To_Point', where PositionSensors are placed in one line.

Then press 'Create RPS_Storage Object' to add the object to your scene. You should now see the RPS_Storage object in the Hierarchy view. The object can be ignored for now.



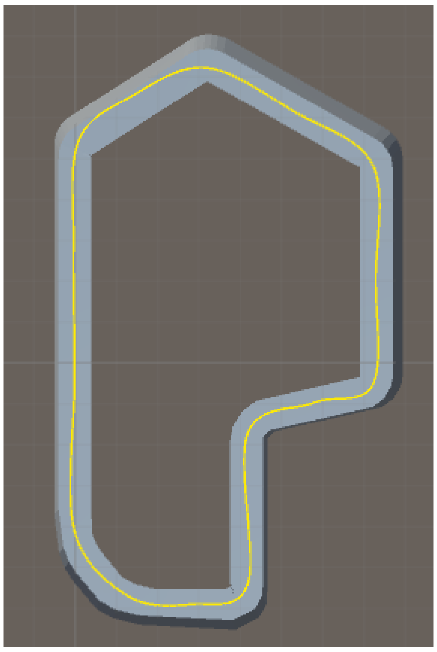
The editor window will now show some more text, explaining that PositionSensors need to be placed around the track. Press the 'Start' button to continue.

Next, you will see even more text. Just follow the instructions carefully.
This step will create a curved line which will follow your track, to add points for the line to pass through press the 'Add Point' button.


To do this click the button bellow to create objects which will link together to create a circuit/line.

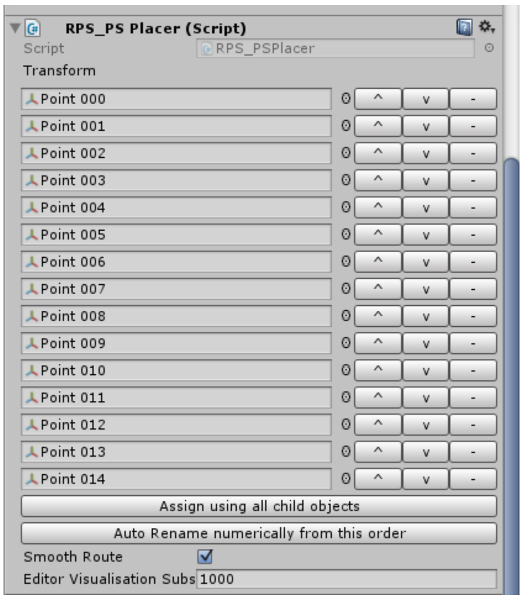
Add Point

The objects are created as children of the RPS_Storage object



This will create an empty gameObject which is a child of the RPS_Storage object. Use the scene view to position the object near the start line of your track. Place more of these objects around your track giving the outline. Try to keep the line in the middle of the track the whole way around.

More advanced users might want to use the RPS_PSPlacer scripts inspector. You will find the script attached to the RPS_Storage object and it is a modified version of the WaypointCircuit script in the Standard Assets:


RPS_PS Placer (Script)

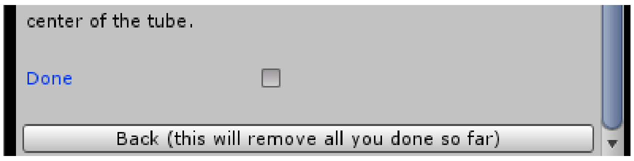
Script	RPS_PSPlacer
Transform	
Point 000	^ v -
Point 001	^ v -
Point 002	^ v -
Point 003	^ v -
Point 004	^ v -
Point 005	^ v -
Point 006	^ v -
Point 007	^ v -
Point 008	^ v -
Point 009	^ v -
Point 010	^ v -
Point 011	^ v -
Point 012	^ v -
Point 013	^ v -
Point 014	^ v -

Assign using all child objects

Auto Rename numerically from this order

Smooth Route	✓
Editor Visualisation Subs	1000

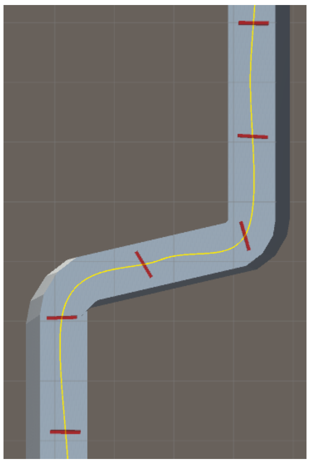
Once you have finished this tick the 'Done' toggle on the editor window.


center of the tube.

Done ☐

Back (this will remove all you done so far)

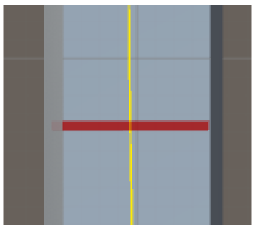
Scroll down in the editor window to see what to do next.
In the scene view you will now see red cuboids around your track. These represent where the groups of PositionSensors will be placed:

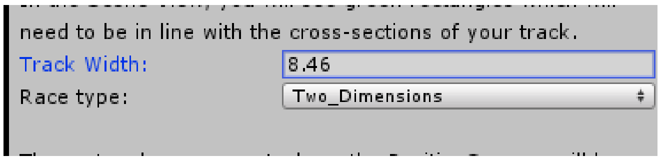


You can still move the position of the curve by moving the Point gameObjects in the scene view. Don't worry about the position or rotation of the cuboids too much as that can be changed for each one individually in the next step.

If your track is for a 3D race e.g. airplanes racing, you can change the type of position sensor in the drop down from 'Two_Dimensions' to 'Three_Dimensions', but the majority of races e.g. cars/boats/humans will use two dimensions not three, so this does not need to be changed.

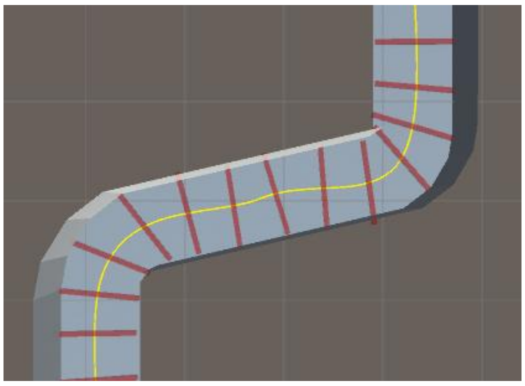
Next, you need to set the width of the cuboids to match the width of your track use the 'Track Width:' field to do this.

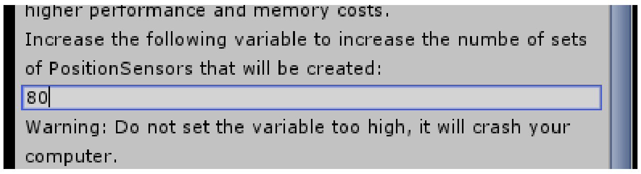




need to be in line with the cross-sections of your track.
Track Width: 8.46
Race type: Two_Dimensions

And now you need to increase/decrease the number of cuboids so there are 2-3 around each corner. The fewer cuboids, the better for performance, the more cuboids the better the accuracy (We advise numbers bellow 200 on mid-range computers and less than 50 on mobile platforms). To do this, change the number in the integer field:





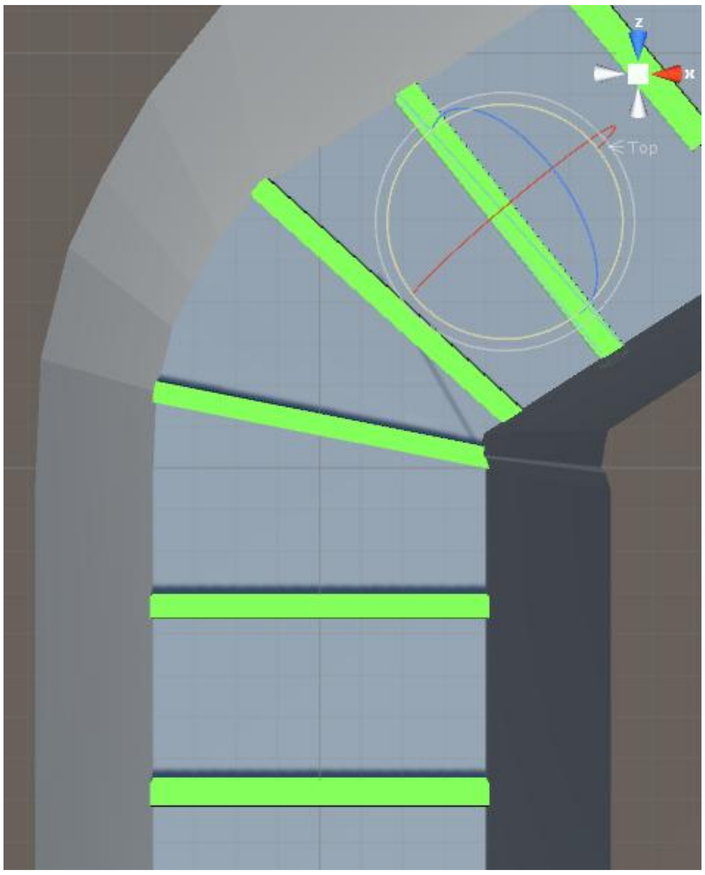
higher performance and memory costs.
Increase the following variable to increase the numbe of sets of PositionSensors that will be created:
80
Warning: Do not set the variable too high, it will crash your computer.

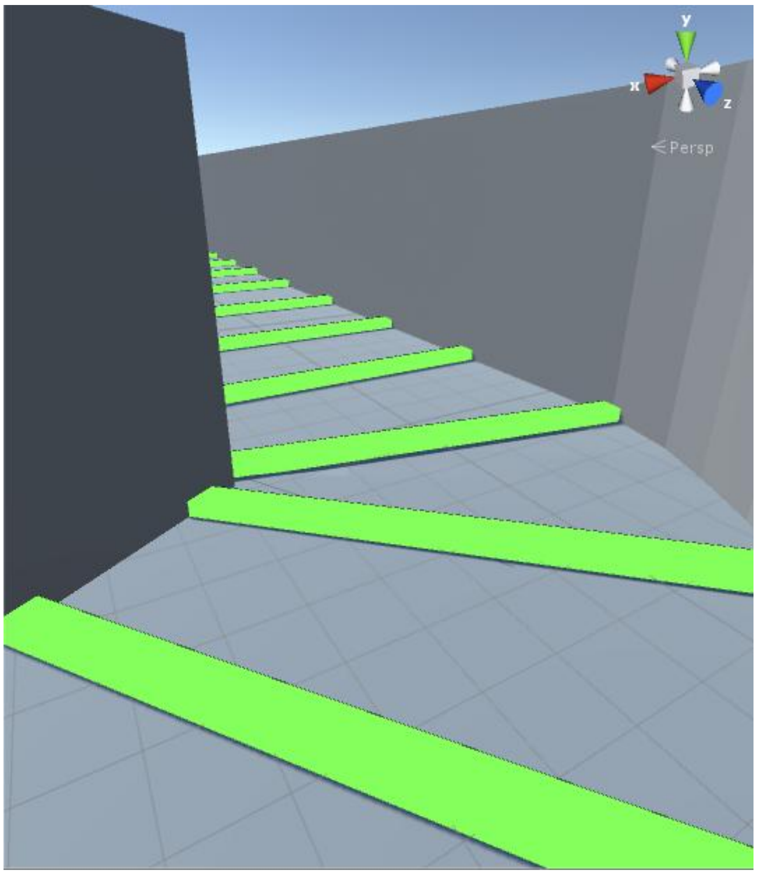
When you are happy with this, click the 'Next' button at the bottom of the editor window.

Now you will see the yellow line around your track has disappeared, and all of the cuboids have turned green. This means you can now position each cuboid individually and can no longer use the curve around the track to position them.
You can also delete any of the cuboids in areas where there are more than needed.

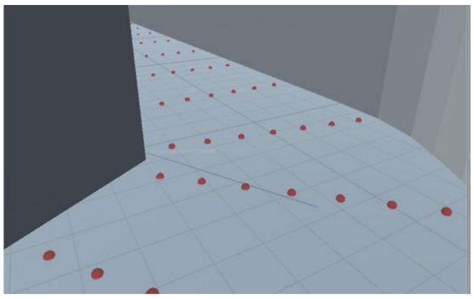
IMPORTANT: Do not change the order of the cuboids as it is being stored in an array and represents the positions around the track from start to finish.

Finally you will need to align all of the cuboids with the track. This is the most time consuming process. Make sure the first and last cuboid are quite close together either side of the start/finish line if you are using laps.



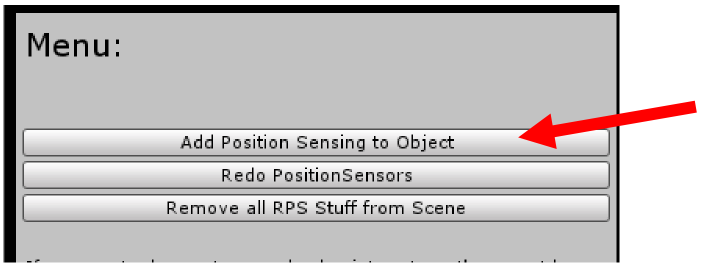


Once you have done this, press the 'Finished' button in the editor window and your PositionSensors will be setup. (Each red dot is now a position sensor)

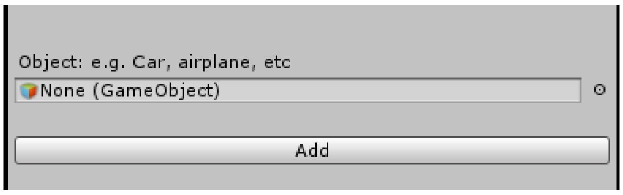


Adding Position Sensing to Objects:

In the editor window press the 'Add Position Sensing to Object' button.


Menu:
Add Position Sensing to Object
Redo PositionSensors
Remove all RPS Stuff from Scene

Then drag and drop your moving object (e.g. car/airplane/boat/human/etc) from the Inspector into the field in the window and press the 'Add' button.



This adds two scripts to the gameobject. An RPS_Position script which calculates the objects position and race position at runtime, and a RPS_Inspector script which can be used to add further scripts for UI elements, checkpoints and laps. It also does some extra data storing things in the background with the RPS_Storage object.

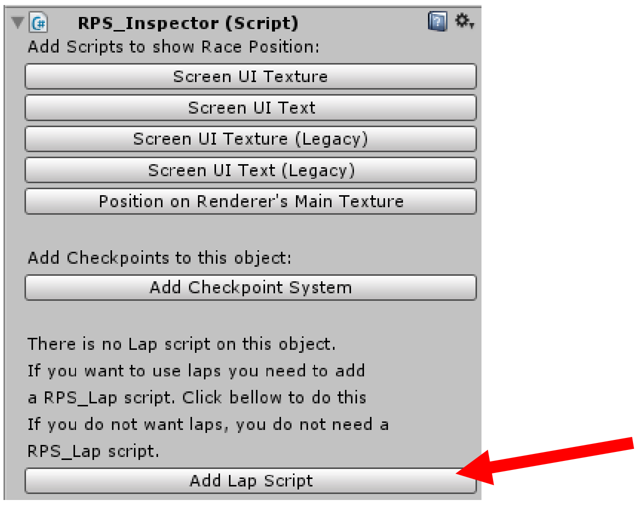
This process needs to be repeated for every object in the scene which you want to have a race position including both player and AI objects.

You have now finished using the editor window and can close it, the rest needs to be done in the Inspector.

Adding a Lap System:

If your race needs laps, you will need to add and RPS_Lap script to every object involved in the race. This can be done in the inspector.

Click on each object in the race (one at a time) and find the RPS_Inspector script. (It should have been added in the previous step). It should look like this:

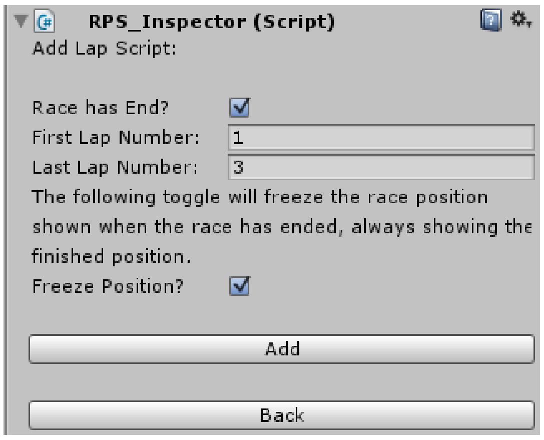


RPS_Inspector (Script)
Add Scripts to show Race Position:
Screen UI Texture
Screen UI Text
Screen UI Texture (Legacy)
Screen UI Text (Legacy)
Position on Renderer's Main Texture

Add Checkpoints to this object:
Add Checkpoint System

There is no Lap script on this object.
If you want to use laps you need to add
a RPS_Lap script. Click bellow to do this
If you do not want laps, you do not need a
RPS_Lap script.
Add Lap Script

Click the 'Add Lap Script' button. Before it is added you will need to set some variables.



RPS_Inspector (Script)
Add Lap Script:

Race has End? ✓
First Lap Number: 1
Last Lap Number: 3
The following toggle will freeze the race position shown when the race has ended, always showing the finished position.
Freeze Position? ✓

Add

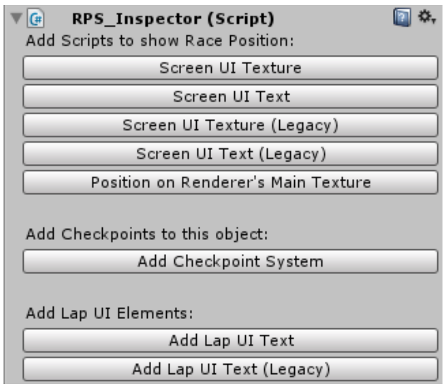
Back

Race has End? – If your race has a set number of laps e.g. 3 laps, then this should be ticked. If your race has an infinite number of laps then this should not be ticked.
First Lap Number: - The lap number to start the race on. If the objects start just before the start/finish line you might want to make it 0 instead of 1, but any number still works the same.
Last Lap Number: - The lap number that the race finishes on. If the first lap number is 1 and the last lap number is 3 then there are 3 laps (laps 1/3, 2/3 and 3/3).
Freeze Position? – If this is ticked then the race position will stop being calculated as the object crosses the line otherwise it will continue to be calculated and might change e.g. car 1 finishes 1st before car 2, then car 2 overtakes car 1 so the position of car 1 changes to 2nd as it is still being calculated, even though the race has ended.

Once you have set these variables click the 'Add' button. Do not forget to do this!
Then do the same for all the other objects in the race.

Adding UI elements:

Select the object you want to add a UI element using and look in the inspector for the RPS_Inspector script. Click the button to select the appropriate feature and type in the necessary information into the fields as described bellow. If you want Lap UI then make sure you have added a lap script in the previous step. Remember to click the 'Add' button when you have filled in the fields!
(To add a legacy UI element use exactly the same steps as bellow)



UI Image on screen showing a 1st, 2nd or 3rd texture:

Click the 'Screen UI Texture' button and fill in the fields.

UI Image – The image which should be changed for the different race positions.
Race Sprites – The array of sprites which should be assigned to the UI Image for the different race positions. If there are not enough race sprites for all of the positions, errors may appear at runtime.

UI Text on screen saying the race position in text format:

Click the 'Screen UI Text' button and fill in the fields.

UI Text – The UI Text object which should be changed for the different race positions.
Show Endings – Toggle should be ticked to add the 'st', 'nd' or 'rd' prefixes after the race position number, e.g. when ticked it might show '1st' intead of just '1' when it isn't ticked.

Setting the texture on the material of an object's renderer to a 1st, 2nd or 3rd texture:

You will have to create your own renderer for this first. The easiest way to do this is create a Quad and make it a child of the moving object. Remember to remove the collider component. The material attached must be able to have a main texture assigned. The best shader for this is Unlit/Transparent.
If you want the renderer to face the camera create a script using transform.LookAt. (However, the standard Unity Quad might be rotated 180 degrees to far so you can't see it. If this happens just add the lookAt script to a parent of the Quad where the quad is rotated 180 degrees (y axis) compared to the parent.)

Then, click the ‘Position on Renderer’s Main Texture’ button and fill in the fields.

Renderer – The renderer component of the object which should have the texture changed.
Race Textures – The array of textures showing 1st, 2nd, 3rd ... etc. They will be displayed on the renderer. If there are not enough textures for all of the positions, errors may appear at runtime.

UI Text showing the Lap Number as text e.g. ‘1/3’:

Click the ‘Add Lap UI Text’ button and fill in the fields.

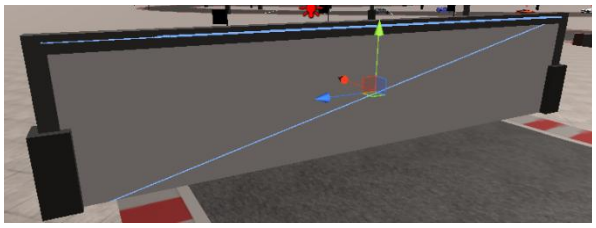
UI Text – The UI Text component to change the text on
Show Total Number of Laps – If unticked it will display a single number. If ticked it will add the / followed by the total number of laps to the text.
Change Text When Finished Race – When the race has finished should the text be changed from e.g. ‘3/3’ to ‘Finished’ or should it be left as ‘3/3’
Change To – What should the text be changed to when the race has finished. The default is ‘Finished’.
Change Font Size – When the text is changed when the race has finished it might be longer than the text it was replacing. For example, the word ‘Finished’ is much longer than ‘3/3’. To make it fit correctly in your UI, you can change the font size to make the word ‘Finished’ smaller so it still fits your UI.
Font Size – The font size to change the text to when the race finishes.

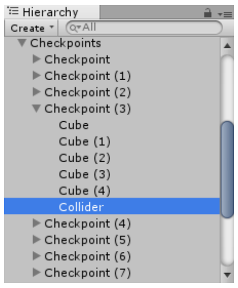
Adding a Checkpoint system:

If you have any problems implementing a checkpoint system following the steps bellow, take a look at our demo scenes.

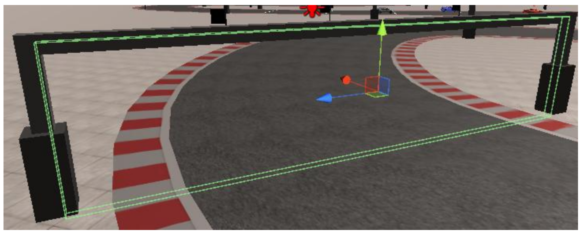
The following steps show a method for adding a trigger based checkpoint system:

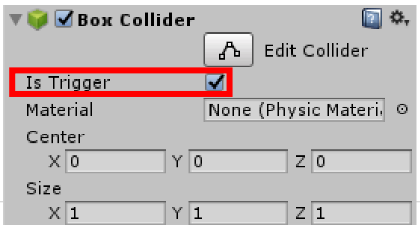
1. Add your checkpoint models to your scene
2. Each model needs a collider to detect when your object hits it. You can add a cube child object to do this, resize it to fit (within) your model, remove the renderer component and tick the isTrigger box on the collider component. You should also add a checkpoint collider at the end of a lap just before the start/finish line if you are using laps.





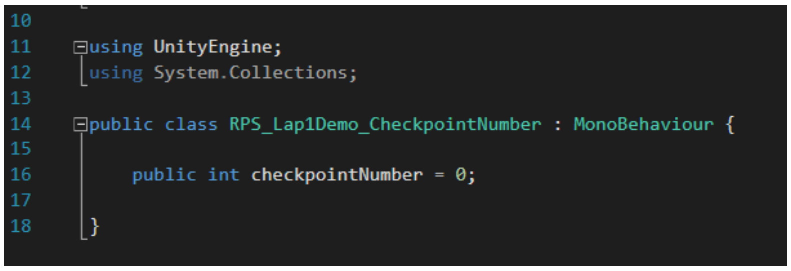
- Checkpoints
  - Checkpoint
  - Checkpoint (1)
  - Checkpoint (2)
  - Checkpoint (3)
    - Cube
    - Cube (1)
    - Cube (2)
    - Cube (3)
    - Cube (4)
    - Collider
  - Checkpoint (4)
  - Checkpoint (5)
  - Checkpoint (6)
  - Checkpoint (7)





Is Trigger ✓
Material None (Physic Materi
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

3. Create a new script which you can attach to each checkpoint. The script just needs one variable (an interger) which represents the checkpoint number (first checkpoint has number 0). The variable needs to be public so it can be accessed by other scripts. Look at the RPS_Lap1Demo_CheckpointNumber script for guidance. (Or just use that script instead of making a new one)



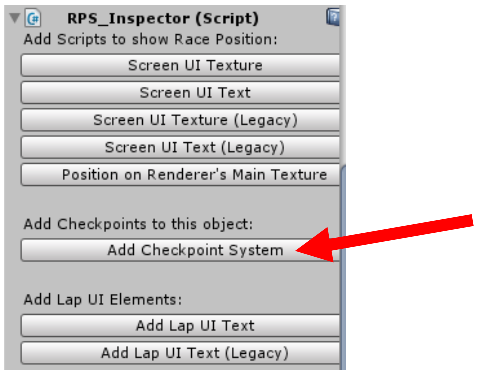
```
using UnityEngine;
using System.Collections;

public class RPS_Lap1Demo_CheckpointNumber : MonoBehaviour {

    public int checkpointNumber = 0;

}
```

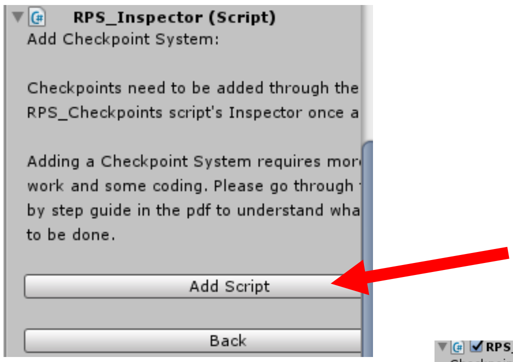
4. Attach the script to ever Checkpoint Collider object (the collider objects NOT the checkpoint parent objects). Set the CheckpointNumber variable in the inspector going 0, 1,2 ... around the track from start to finish.
5. Now you need to create another script to go on the player object. This will need to detect when a checkpoint is hit using OnTriggerEnter and it will find the checkpoint number of the checkpoint it has hit using GetComponent. Look at the RPS_Lap1Demo_CheckpointCollider script for help with this. (If you have lots of triggers in your scene add a tag to all checkpoint colliders and check this in OnTriggerEnter)
   The script then needs to call the CheckpointPassed function on the RPS_Checkpoints script on the player object (which we haven't added yet). This can be done by assigning the RPS_Checkpoints script in the inspector (once we have added it) and calling the function through it as done in RPS_Lap1Demo_CheckpointCollider example script.
6. Attach the script to the player object and add a RPS_Checkpoints script. This should be done using the RPS_Inspector script in the Inspector of the player object:



RPS_Inspector (Script)
Add Scripts to show Race Position:
Screen UI Texture
Screen UI Text
Screen UI Texture (Legacy)
Screen UI Text (Legacy)
Position on Renderer's Main Texture

Add Checkpoints to this object:
Add Checkpoint System

Add Lap UI Elements:
Add Lap UI Text
Add Lap UI Text (Legacy)



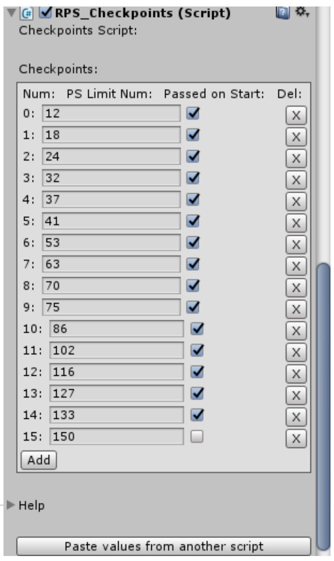
RPS_Inspector (Script)
Add Checkpoint System:

Checkpoints need to be added through the RPS_Checkpoints script's Inspector once a

Adding a Checkpoint System requires mor work and some coding. Please go through by step guide in the pdf to understand wha to be done.

Add Script

Back



RPS_Checkpoints (Script)
Checkpoints Script:

Checkpoints:

Num:	PS Limit Num:	Passed on Start:	Del:
0:	12	✓	X
1:	18	✓	X
2:	24	✓	X
3:	32	✓	X
4:	37	✓	X
5:	41	✓	X
6:	53	✓	X
7:	63	✓	X
8:	70	✓	X
9:	75	✓	X
10:	86	✓	X
11:	102	✓	X
12:	116	✓	X
13:	127	✓	X
14:	133	✓	X
15:	150	☐	X

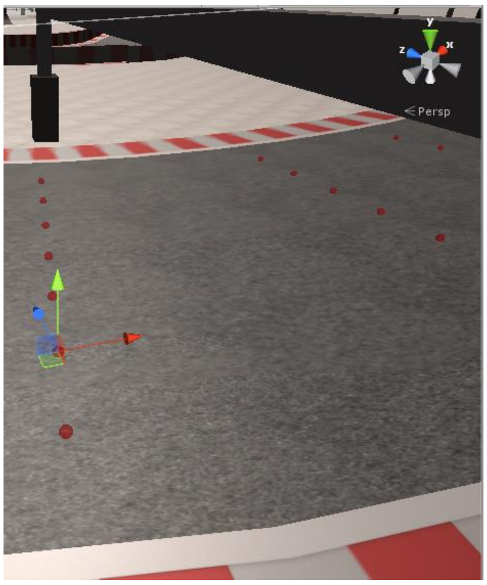
Add

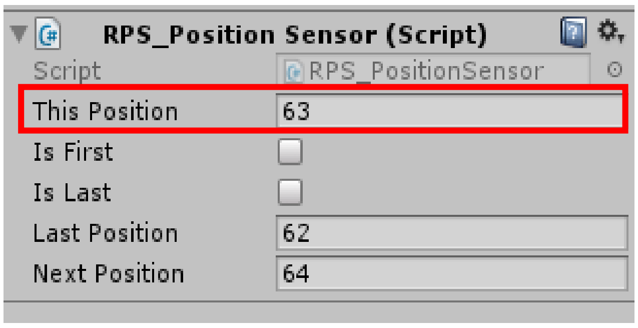
Help

Paste values from another script

7. Remember to assign the RPS_Checkpoints script to the field in the script attached to the player object for detecting collisions with the checkpoints.
   Then look at the RPS_Checkpoints script in the Inspector and fill in all of the fields. Click the 'Add' button to add all of your checkpoints.
   Tick the 'Passed on Start' for checkpoints already passed when the race starts. e.g. in a race with laps, if the objects start before the start/finish line then all checkpoints should be ticked apart from the last one (which is just before the start/finish line).

The PS Limit Number is the most important part of this. If the object is quite far around/along a track it will have a higher position number. If a checkpoint is missed the position number will be limited to the number assigned to the missed checkpoint in the field on the RPS_Checkpoints script. This means if a object is 1st in a race it might not show 1st if it has missed a checkpoint earlier on, this stops players from cutting corners in a race.
To assign the correct Position Number to each checkpoint, find the checkpoint in the scene. You should see red dots around the track (PositionSensors) which you created earlier. Find the next set of red dots after the checkpoint. Select one of them by clicking on it (or use the Hierarchy if this doesn't work as they are all children of RPS_Storage) and look in the inspector. The number you need is in the 'This Position' field of the RPS_PositionSensor script.




RPS_Position Sensor (Script)

Script	RPS_PositionSensor
This Position	63
Is First	☐
Is Last	☐
Last Position	62
Next Position	64

Do this for all checkpoints and enter the numbers into the column in the RPS_Checkpoints script. If you are using laps, for the last checkpoint just before the start/finish line the position number you use should still be the one just after the collider which should be the last set of PositionSensors of the lap (before they start from 0 again). E.g. in our screenshot of our RPS_Checkpoints script above, the last position number is 150 as we have position numbers going from 0 (at the start) to 150 (at the end) around our track.

You can do the same for each object in the race if you need to. However, in most games the AI objects have it built into their AI to pass through the checkpoints so a RPS_Checkpoints script isn't needed to check this and only the player object needs it.

Your checkpoints system should now be working.

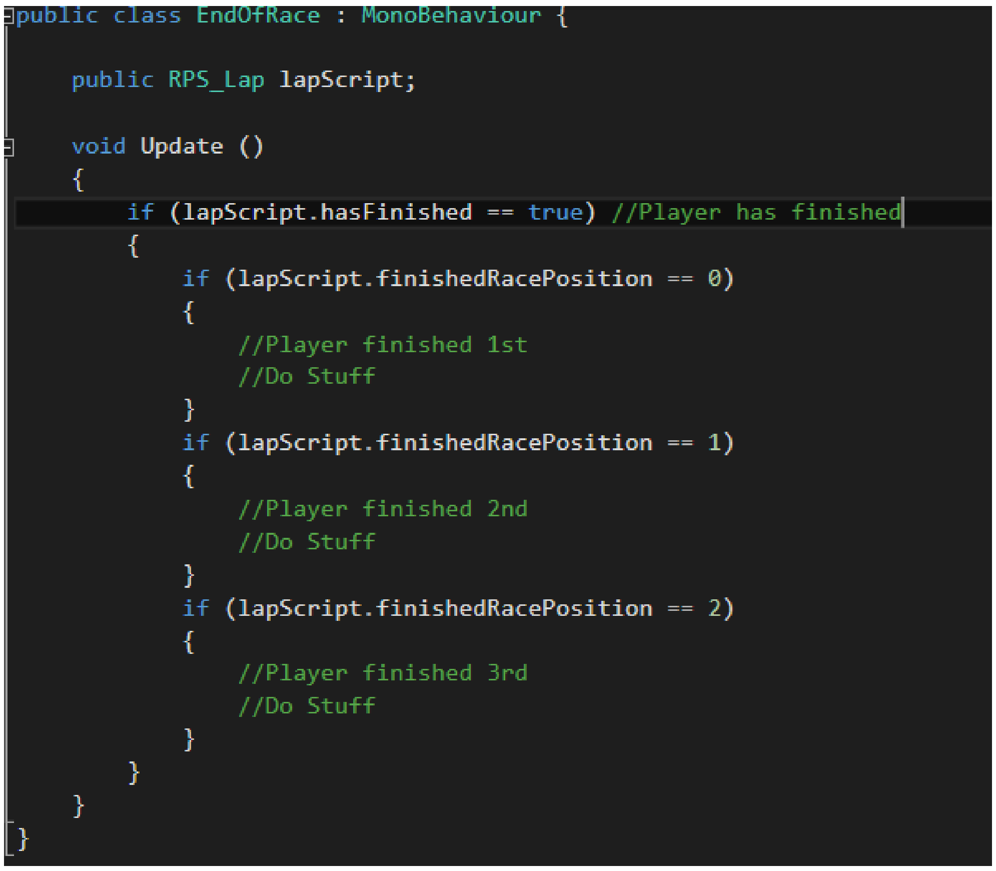
End of Race Scripts:

At the end of your race you will probably want something to happen. For example, you might want some UI to appear for the player to move on to the next scene or something similar. You might also need to know the place the player finished in e.g. 1st, 2nd, etc.

If you are not using laps in your race, you will need to do slightly more coding for the end of the race as normally the RPS_Lap script would handle most of it. You will need a trigger on your finish line and a script on it detecting when the object hit it. The number of objects which hit it could be counted until the player's object hits it, so you know the players finished race position. (You could also get the players position from the RPS_Position script, but it would be less accurate in this situation). Then you could add code for UI to appear etc. The '3 Dimensional' demo scene is a good example for this.

If you are using laps then the process is slightly easier. Create a new script for detecting when the race has ended and add a public variable for the RPS_Lap script on the player object which can be assigned in the inspector. In the Update function/void, you can use the bool RPS_Lap.hasFinished to detect when the race has finished. You can then use the variable RPS_Lap.finishedRacePosition to get the players race position. When the variable is 0, the player finished 1st, when variable is 1, player finished 2nd, etc.

E.g. in C#:



```
public class EndOfRace : MonoBehaviour {

    public RPS_Lap lapScript;

    void Update ()
    {
        if (lapScript.hasFinished == true) //Player has finished
        {
            if (lapScript.finishedRacePosition == 0)
            {
                //Player finished 1st
                //Do Stuff
            }
            if (lapScript.finishedRacePosition == 1)
            {
                //Player finished 2nd
                //Do Stuff
            }
            if (lapScript.finishedRacePosition == 2)
            {
                //Player finished 3rd
                //Do Stuff
            }
        }
    }
}
```

Scripts Overview:

Here are some short descriptions of the main scripts which you should know about:

- **RPS_Checkpoints.cs –** This script must be attached to each object with an RPS_Position script which you want a checkpoint system to apply to. It stores lists of checkpoints and whether they have been passed or not. The RPS_Position script can then access this script to limit the known position around the track to any checkpoints that might have been skipped. The 'CheckpointPassed' function must be called on this script by whatever method you choose when a checkpoint is passed.
- **RPS_Inspector.cs –**This script just stores variables for each object involved in a race, allowing the use of some editor features. It has no effect at runtime.
- **RPS_Lap.cs –** This script can be attached to any object with an RPS_Position script. It adds a lap system to the object, keeping track of the current lap number. No objects should have more than one attached.
- **RPS_LapUI.cs –** This script can be attached to any object with a RPS_Lap script. It is used to display the current lap number of an object through an object with a Unity UI Text component.
- **RPS_LegacyLapUI.cs -** This script can be attached to any object with a RPS_Lap script. It is used to display the current lap number of an object through an object with a Legacy GUIText component.
- **RPS_LegacyScreenUI.cs –** This script can be attached to any object with a RPS_Position script. It is used to display the current race position of an object through different textures (showing 1st, 2nd, 3rd etc) or text through an object with a Legacy GUITexture or GUIText component.
- **RPS_Position.cs –** This script can be attached to any object, to calculate the object's current race position. This script does the majority of the work to calculate the object's position based on PositionSensors around the track. It is required by all Lap and UI scripts.
- **RPS_PositionSensor.cs -** This script is attached to all PositionSensors in the race scene (when you set them up in the editor window). It stores the position of the PositionSensor in relation to the other PositionSensors before and after it. The RPS_Position scripts constantly find the nearest RPS_PositionSensor script in order to calculate the current race position.
- **RPS_PositionTexture.cs –** This script can be attached to any object with a RPS_Position script. It is used to display the current race position of an object through different textures (showing 1st, 2nd, 3rd etc) through an object's renderer. E.g. A plane or quad might be put above a car to show its position. This script changes the texture on the plane to show the car's position.
- **RPS_ScreenUI.cs -** This script can be attached to any object with a RPS_Position script. It is used to display the current race position of an object through different textures (showing 1st, 2nd, 3rd etc) or text through an object with a Unity UI Image or Text component.
- **RPS_Storage.cs –** This script stores main variables for all objects and scripts involved in a race. This links some objects together allowing the use of some editor features. It is required at runtime as many other scripts access its variables.

These scripts are all commented on, if you wish/need to understand them.

If you add any new variables to the scripts above, they will not appear in the Inspector as the new fields are not coded into the editor scripts. This is why we advise creating your own scripts and accessing the scripts above through GetComponent rather than by modifying them.

Useful Variables and Functions:

RPS_Postion Script:

RPS_Position.currentLapNumber – The current lap number being used to compare with other RPS_Position scripts to calculate race standings. A variable with type int.

RPS_Position.lapsGoneBack – The current number of laps the object has gone back (in the wrong direction) to compare with other RPS_Position scripts to calculate the race standings. A variable with type int.

RPS_Position.freezePosition() – Can be called to freeze the race position being calculated. This is done by the RPS_Lap script if it is enabled in the Inspector. If you are not using laps, it can be called to freeze the position at the end of the race (or anytime during the race). A function with type void.

RPS_Position.unfreezePosition() – The opposite of the previous function. Can be called to unfreeze the race position being calculated. A function with type void.

RPS_Position.raceFinished() – When using laps, it is called by the RPS_Lap script. When not using laps it can be called to tell the RPS_Position script that the race has ended. A function with type void.

RPS_Position.getRacePosition() – Returns the current race position. 0 = 1st, 1 = 2nd, ... etc. A function with type int.

RPS_Position.nearestPosition() – Returns the nearest position number from the nearest position sensor. This doesn't include the limits set by the RPS_Checkpoints script when a checkpoint is missed. A function with type float.

RPS_Position.nearestPositionLimited() – Returns the nearest position number from the nearest position sensor. This does include the limits set by the RPS_Checkpoints script when a checkpoint is missed. A function with type float.

RPS_Lap Script:

RPS_Lap.hasFinished – Has the race finished according to the RPS_Lap script. (Has the finish line been crossed on the last lap by <u>the object it is attached to</u>). To check if all objects have finished the race, this variable should be checked on all RPS_Lap scripts. A variable with type bool.

RPS_Lap.finishedRacePosition – The race position that an object finishes the race at. 0 = 1st, 1 = 2nd, ... etc. Should use the previous variable to check the race has finished first, as if the race hasn't finished it will be 0 (even if the object isn't 1st). A variable with type int.

RPS_Lap.currentLapNumber() – The current lap number. It includes the number of laps gone back. A function with type int.

RPS_Lap.maxLapNumber() – The current lap number without the number of laps gone back. It is the highest lap number which has been reached by the object throughout the race. A function with type int.

RPS_Lap.lapsGoneBack() – The number of laps the object has gone back (wrong way across start/finish line). A function with type int.

RPS_Checkpoints Script:

RPS_Checkpoints.checkpointLaps – A C# list of the class checkpointLap. A new checkpointLap is added when the object goes onto a new lap. A checkpointLap has an array of all the checkpoints on that lap and whether they have been passed or not. A variable with type List<RPS_Checkpoints.checkpointlap>.

RPS_Checkpoints.limitedPosition – The position that the RPS_Position script is being limited to if a checkpoint has been missed. A variable with type float.

RPS_Checkpoints.CheckpointPassed (int checkpointNumber) – A function which must be called when a checkpoint is passed. Read the tutorial to see when and how this is used. It tells the RPS_Checkpoints script when a checkpoint has been passed with the checkpoint's number as a parameter. A function with type void.

RPS_Checkpoints.nextCheckpointNumber() – Returns the checkpoint number of the next checkpoint for the object to cross. A function with type int.

RPS_Checkpoitns.lastCheckpointNumber() – Returns the checkpoint number of the last checkpoint the object has crossed. A function with type int.

RPS_Checkpoints.checkpointMissed() – Returns whether a checkpoint has been missed or not. A function with type bool.

RPS_Checkpoints.checkpointNumberMissed() – If a checkpoint has been missed, this function returns the number of the first checkpoint which has been missed. Use the previous function before this to make sure a checkpoint has actually been missed. A function with type int.

RPS_Checkpoints.checkpointLap Class:

RPS_Checkpoints.checkpointLap.lapNumber – The lap number of this set of checkpoints. A variable with type int.

RPS_Checkpoints.checkpointLap.hasPassed – A C# list containing a bool for every checkpoint on the lap and if it has been crossed or not. A variable with type List<bool>.

Contact Us:

You can contact us by emailing:

solution_studios@outlook.com

Or you can submit a contact form on our website here:

http://solution-studios-for-unity.moonfruit.com/contact-us/4583955951

Also, if you open the RPS Editor Window there is a button at the bottom which will take you to the unity forum page. We check it regularly so you can also contact us there.

And we would appreciate it if you could leave a review on the Asset Store page.

Thank you!