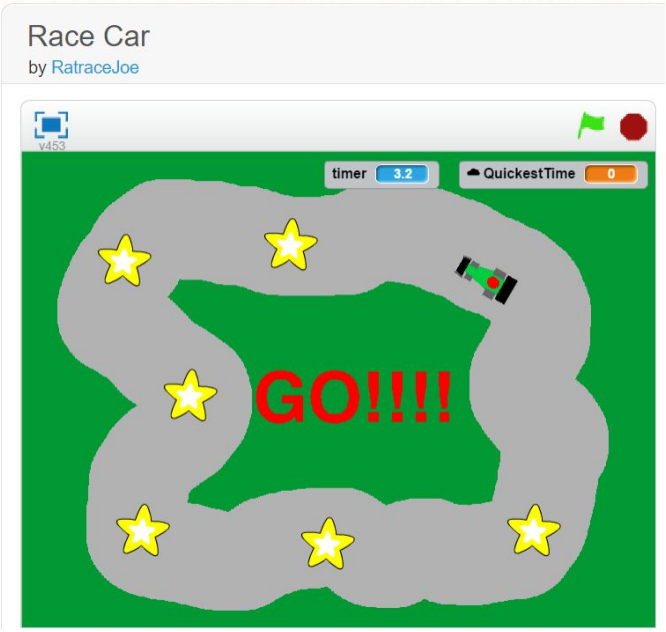
Scratch - Race Car

In this project we will be building a game where the player drives a car from a top down perspective. They player must navigate the track and collect all the stars to stop the timer. We will use cloud variables to store the high score on the internet.

The finished game will look like this
https://scratch.mit.edu/projects/138301593



The player will use the arrow keys to control the car with

- Up and Down arrows to Accelerate and Decelerate
- Left and Right keys to turn
- The space bar will be used to stop and start the game.

Build the Car

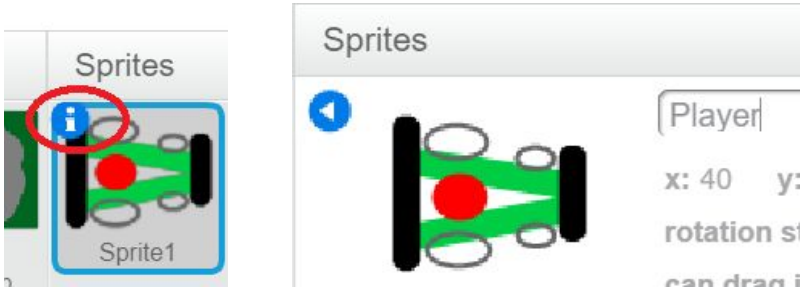
Draw the Sprite

First we will build a car sprite. I will keep it fairly simple.

- Start a new scratch project
- Click on the **Costumes** tab for the default sprite (sprite1, the cat)
- Delete the second costume
- Clear the first costume
- Draw it as shown.

The most important thing is that it points to the right. It's a bit like a formula one car, we have draw a chassis, spoilers, wheels and drivers helmet.

Rename the sprite to **player** by clicking on the **information** button by the sprite and editing the text.



Build Forward and Backward Controls

For now, we will trigger the control loop when the player clicks on the green flag. Later we will use broadcasts to control the game.

- Click on the **Scripts** tab
- From **Events** Drag a **When (green flag) is clicked** block into the code area.
- From **Control** drag a **repeat until <>** block into the event handler. Leave the condition blank for now.
- We want to go forward if the **up key** is pressed. So from **Control** drag an **if <>** block into the loop block.

- From **Sensing** drag a **key <up arrow> is pressed**; you will have to select **up arrow** from the drop down box.
- From **Motion** drag a **move <5> steps**; you can type 5 into the box.

Try Yourself

Build another **if <>** block just after this first one that detects the **down arrow** and moves **-5** steps if it is pressed. This should give us a means of going forwards and backwards.

Test

Now that you have forward and backward controls built, try the game by pressing the green flag and pressing the up and down arrow keys. The car should go forward (relative to the direction it is pointing) and backward accordingly. Do not proceed until this is working!

Build Turning Controls

We now need to build the steering wheel for the car. We will be using a similar pattern to the movement, but we will use the **turn <> degrees** block from **Motion**. Here is how to build turning right.

- From **Control** drag an **if <>** block into the game loop
- From **Sensing** drag a **key <right arrow> is pressed** block into the **if <>** block
- From **Motion** drag a **turn clockwise <5> degrees**.

This particular button handler will look like this.

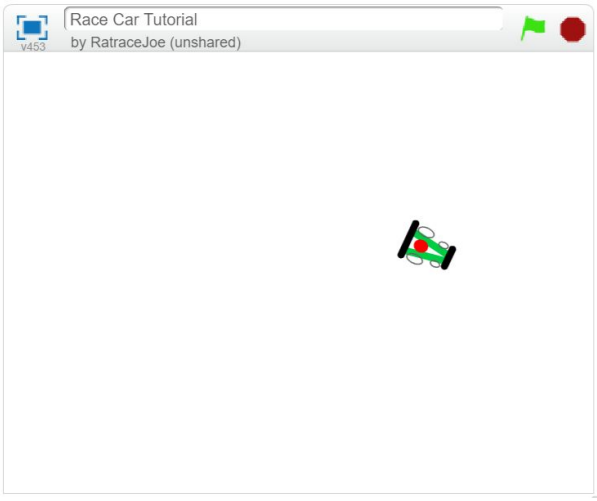
Try Yourself

Build another button handler for the **left arrow** which turns **counter clockwise** by 5 degrees. This should allow us to turn in either direction.

Test

Click the green flag to start the game then try to drive the car. It should be possible to turn in both directions (in addition to going forwards and backwards)

Use the shrink tool to make the sprite a bit smaller so it looks roughly as shown in this screenshot.



That will do for the controls, it is now time to build the track.

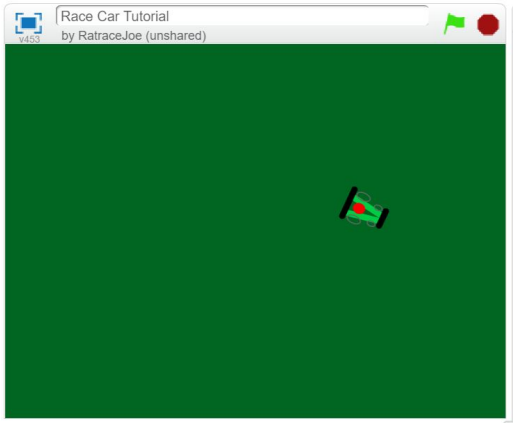
Build the Track

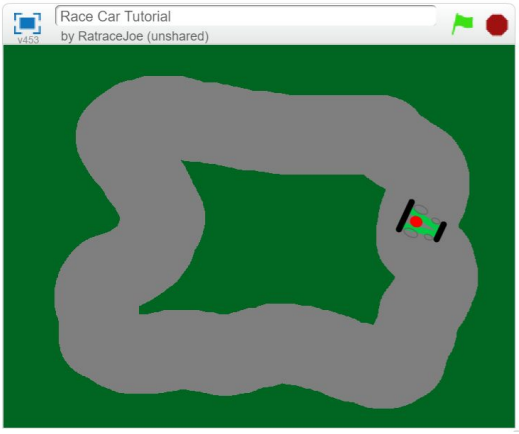
The stage will act as the track, and it will be very simple. We will use one colour as ‘grass’ and another as ‘tarmac’. Then we will write some code to detect when the player is on grass and use it to slow the player down.

Draw the Track

This will simply be a case of drawing a new backdrop.

- Click on the **Stage**
- Click on **Backdrops**
- Use the fill tool to fill the backdrop with a green colour.
- Use a very thick **pen** tool to draw a grey track





Make Grass Affect the Car's Speed

Return to the **Script** tag for the car sprite. Have a look at the code. We are moving **5 steps** in either direction based on the up/down arrow keys. We need to change this number if the car is in contact with the grass. The best way to do this is to introduce a variable.

- Under **Data** click on **Make a Variable**.
- Call the variable **speed**
- From **Data** drag a **set <speed> to 5** block into the start of the green flag event handler (before the game loop!)

This will set this variable to something sensible at the start of the game. The code should look like this:

We now need to detect the grass (or not) and set the speed accordingly. Add the following code inside the game loop.

- From **Control** drag an **if <> else** block into the game loop (alongside our existing **if <>** blocks)
- From **Sensing** drag a **touching colour []** block into the **if <> else** condition
- Use the colour picker to set the colour to **green**. (Click on the little square, then click on the green of the background.
- From **Data** drag a **set <speed> to 1** into the if statement.
- From **Data** drag a **set <speed> to 5** into the **else** part
- From **Data** drag the value of the **speed** variable into the **move <5> steps** block that responds to the **up arrow**.

Try Yourself

Use a combination of the multiplier block from the **Operators** and the speed variable to move **minus speed** when the **down arrow** is pressed.

Test

Click the green arrow again and try driving around. You should find that the car slows right down if you are in contact with the grass. Check that you got the reverse direction working properly still! Drive in and out of the grassy area to check the speed adjust correctly. Once you have tested this, you can continue. The speed variable will be made visible by default, so the game should look like the screenshots below (one on grass, one on tarmac)

Create the Collectable Stars

We now have a playable driving simulator (delusions of grandeur) we need to build the items to collect within the timer. The purpose of the stars will be to ensure that the car has done a complete lap.

Create the sprite and clone it

We will using cloning when the game starts to plop stars around the track

- Click on **Choose a New Sprite from the Library**
- I chose the star shown below, it can be found under **things** in the sprite library
- Click on the **Scripts** tab for the **star** sprite
- From **Events** drag a **When Green Flag is clicked** event handler into the code area.

We will use a series of **goto x: y:** blocks and **clone this sprite** blocks to plop them around the track. Luckily the Scratch editor cleverly populates the **goto x,y** block with the current coordinates of the sprite.

- Move the star sprite to somewhere on the track
- From **Motion** drag a **goto x: y:** block into the green flag handler
    - You shouldn't have to change the values of x/y here, it will be set to where the star is in the game area
- From **Control** drag a **Create clone of myself** into the handler
- Move the star further along the track
- Repeat a few times
- Finally from **Looks** drag a **hide** block after the clone creations.

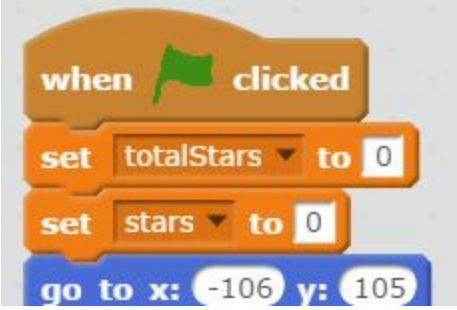
Test

Click on the green flag and you should see a number of stars around the track as shown in the screenshot above.

Handle Star Sprite Game Loop

We will be using a similar repeat until loop in the clone handler to make the stars work. In order to detect that the stars have been collected, we need to create some new variables and initialise them in our green flag handler.

- Under **Data** create two new variables
    - totalStars - this will hold the number of stars created
    - stars - this will hold the number of stars collected
- From **Data** drag a **set <totalStars> to <0>** into the **green flag** handler
- From **Data** drag a **set <stars> to <0>** into the **green flag** handler



Now we need to write the behaviour of the clone when it is created.

- From **Events** drag a **when I start as a clone** block into the code area.
- From **Data** drag a **change <totalStars> by <1>** block into the clone handler. This will allow each clone to count itself into the total.
- From **Looks** drag a **show** block in
- From **Control** drag a **repeat until <>** block into the **clone handler** block. This will be the game loop similar to the one for the car.
- From **Control** drag a **delete this clone** block AFTER the game loop. The assumption is that the clone will no longer be required when the game is finished.

The star will sit on the track until it comes into contact with the car, so we will use an **if <>** block inside the loop to detect collision with the car. Upon colliding with a star, the score should be incremented and then we will check to see if all of the stars have been collected.

- From **Control** drag an **if <>** block into the loop
- From **Sensing** drag a **touching <Player>** block into the condition. This assumes you renamed the player sprite correctly earlier.
- From **Data** drag **change <stars> by <1>** block into the **if <>** block
- From **Control** drag another **if <>** block after the score counter.
- From **Control** drag a **delete this clone** block after our new **if <>** block but inside the **if <touching player>** block.

Use a combination of operators and variable values to check if the number of stars collected is equal to the total stars. If it is, then the game is over and we can indicate this to the user.

- From **Operators** drag a **[] = [] (equals)** block into the **if <>** statement
- From **Data** drag **stars** and **totalStars** into the two blocks of the equality statement. It does not matter what order they are in.
- For now we will simply say “Well Done” when the last star has been collected. From **Looks** drag a **Say <Well Done> for <4> seconds** into the **if <>** block.

Test

Try playing the game now, it should be possible to collect all the stars. Upon collecting the last one, the game will say “Well Done” for 4 seconds and then the final star will disappear.

Handle the Timer

We now have a functioning game, but let us make it more exciting by using a timer. The timer should be reset to zero when the game is started, and stopped when the last star is collected.

First we should create some new variables. One of these is a particularly interesting type of variable, a **cloud variable**. These have a single value that is shared across anyone playing this game. It allows you to compete against friends from across the internet.

- Under **Data** click on **Make a Variable** and give it the name **thisTime**. We will use this variable to store the time for the current attempt. It will be a normal **For all sprites** variable.
- Under **Data** click again on **Make a Variable** and give it a name **bestTime**. This time however tick the box **Cloud Variable**

You will only be allowed to create cloud variables if you are logged into Scratch on a verified account. If you are not, then do not make it a cloud variable and it will just persist for the given user.

We need to reset the timer when the game stars

- From **Sensing** drag a **reset timer** block into the **green flag** handler of the **star** sprite.

We now need to use the timer to set **thisTime** when the game ends.

- From **Data** drag a **set <thisTime> to <0>** block into the **game loop** just before the **Well Done** message.
- From **Sensing** drag the value of **timer** into the **thisTime** data setter.
- From **Control** drag an **if <> else** block after the timer setter.
- Duplicate the **Well Done** say block into both parts of the **if <> else** block. We will change the message to say **new best time** in the **if <>** part.

The game end code now looks like this

The **if <>** statement will be used to evaluate the **bestTime** against the time we have just set (**thisTime**). If **bestTime** has not been set yet, we should assume the new time is the best. So this means we need to use an OR block.

- From **Operators** drag an **<> OR <>** block into our new **if <> else** condition.
- Use Operators to check for either of the following two conditions
    - **bestTime = 0**
    - **thisTime < bestTime**
- From **Data** drag a **set <bestTime> to <thisTime>** block into the correct part of the **if <>** block.

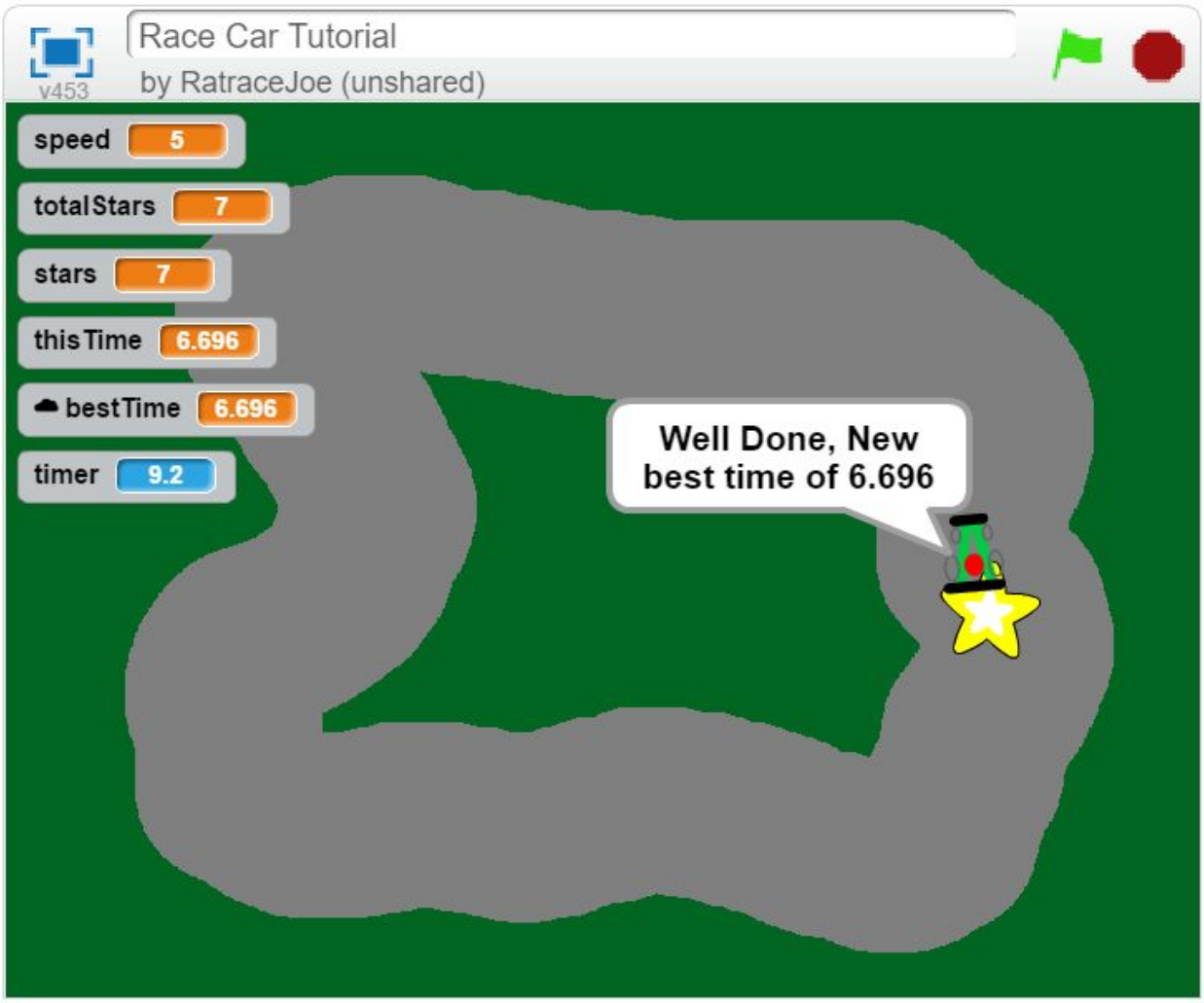
- Change the message for new best time using the string building block from **Operators**.
- Under **Sensing** check the box next to **timer** to make the timer value visible in the game area. This will be useful for the player.

The game ending code should now look like this

Test

Play the game a few times, make the first attempts deliberately slow so that you can beat your best time and check that the timer works correctly. The game should look something like this now.

If you were able to use cloud variables, invite your friends to access your game on their machine and see if they can beat your best time.



CONGRATULATIONS

YOU HAVE NOW FINISHED THE RACE CAR TIMED LAP GAME