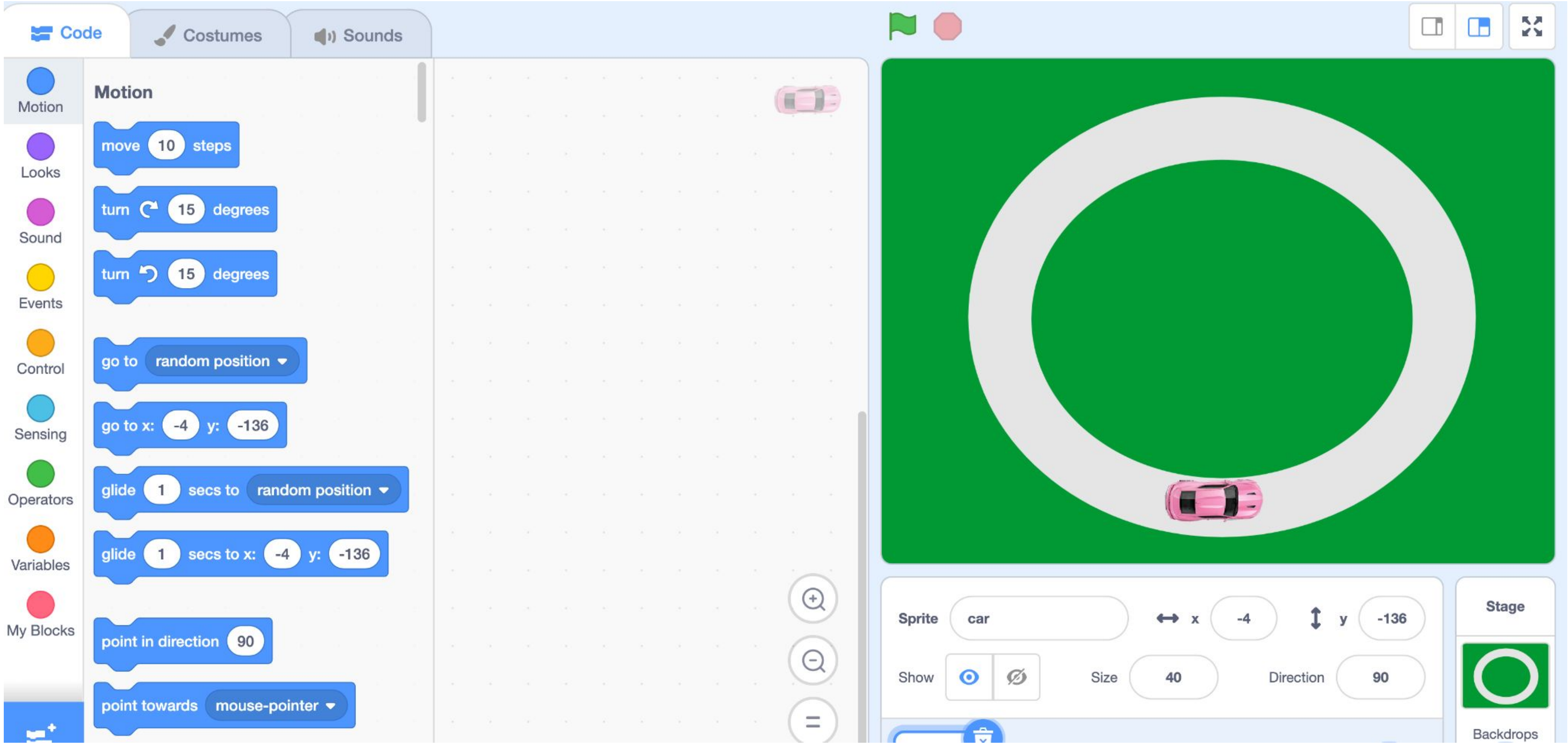
Are you up for a challenge?

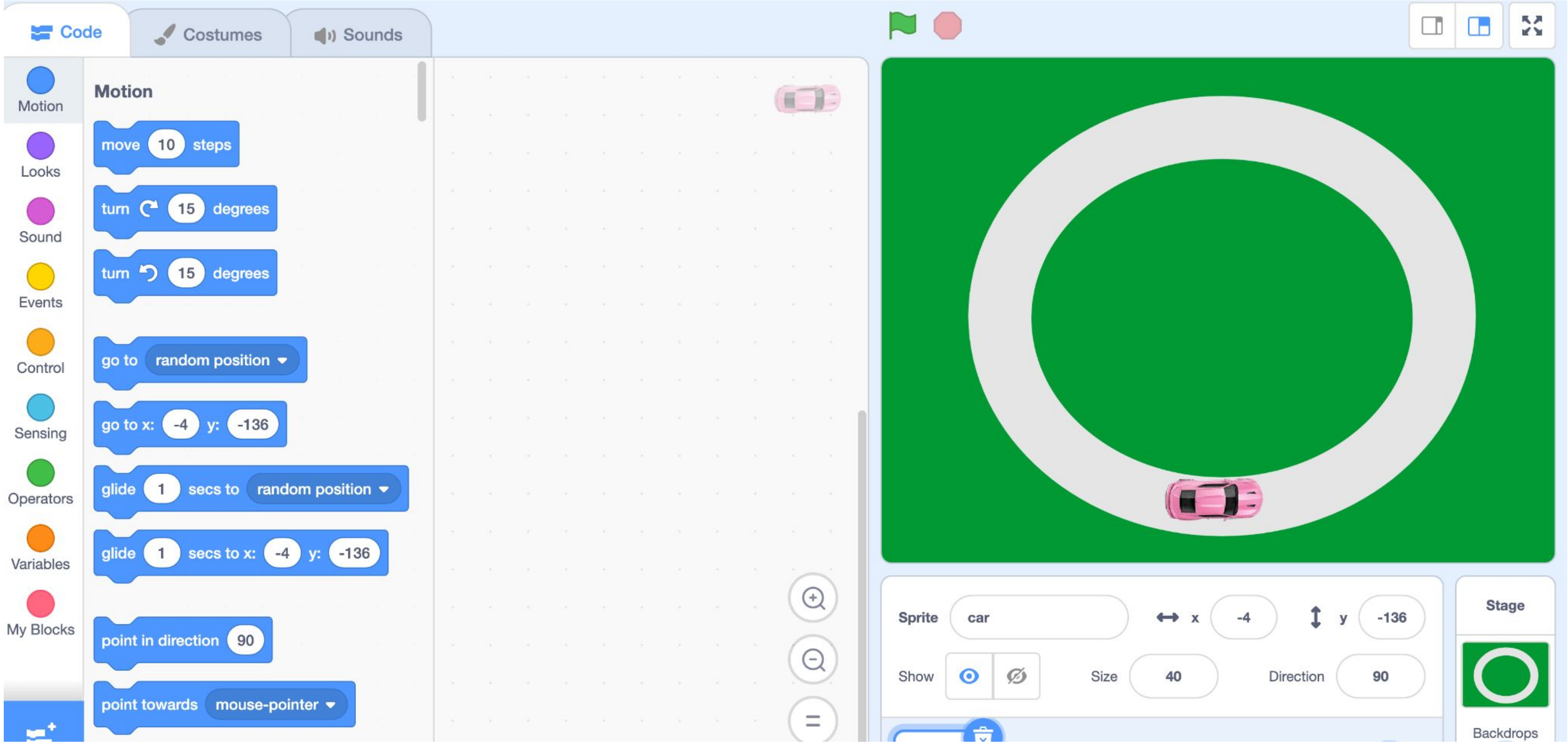
→ Go to Scratch: Click [here]()



Sprite car ↔ x -4 ↕ y -136

Show Size 40 Direction 90

This is Scratch. Scratch is a website which you can use to program things!



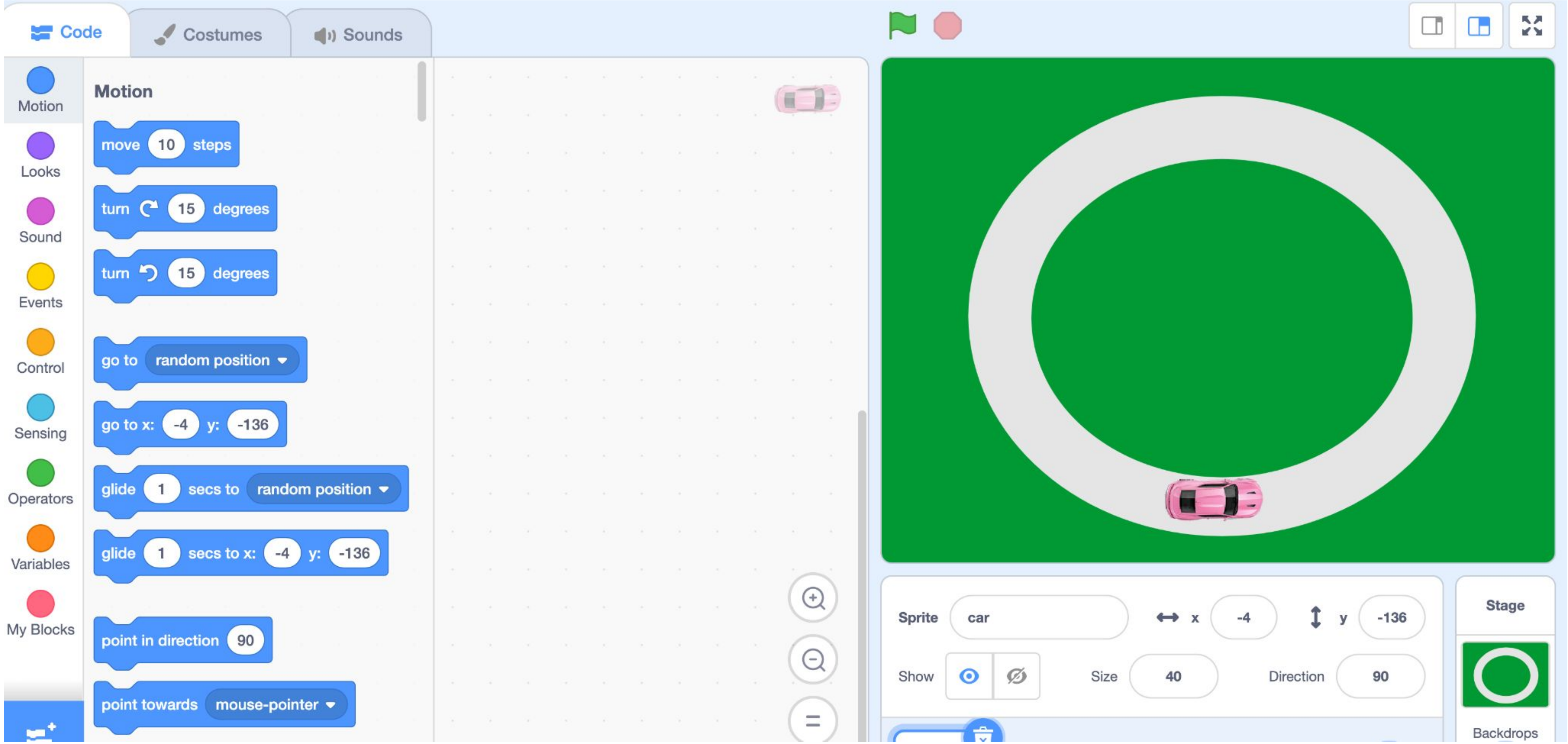
Motion

```
move 10 steps
turn ↻ 15 degrees
turn ↺ 15 degrees
go to random position ▾
go to x: -4 y: -136
glide 1 secs to random position ▾
glide 1 secs to x: -4 y: -136
point in direction 90
point towards mouse-pointer ▾
```

Sprite car ↔ x -4 ↕ y -136

Show Size 40 Direction 90

Programming is a little like building a recipe for a computer. You program ‘code’. And the computer will folow that code step by step.



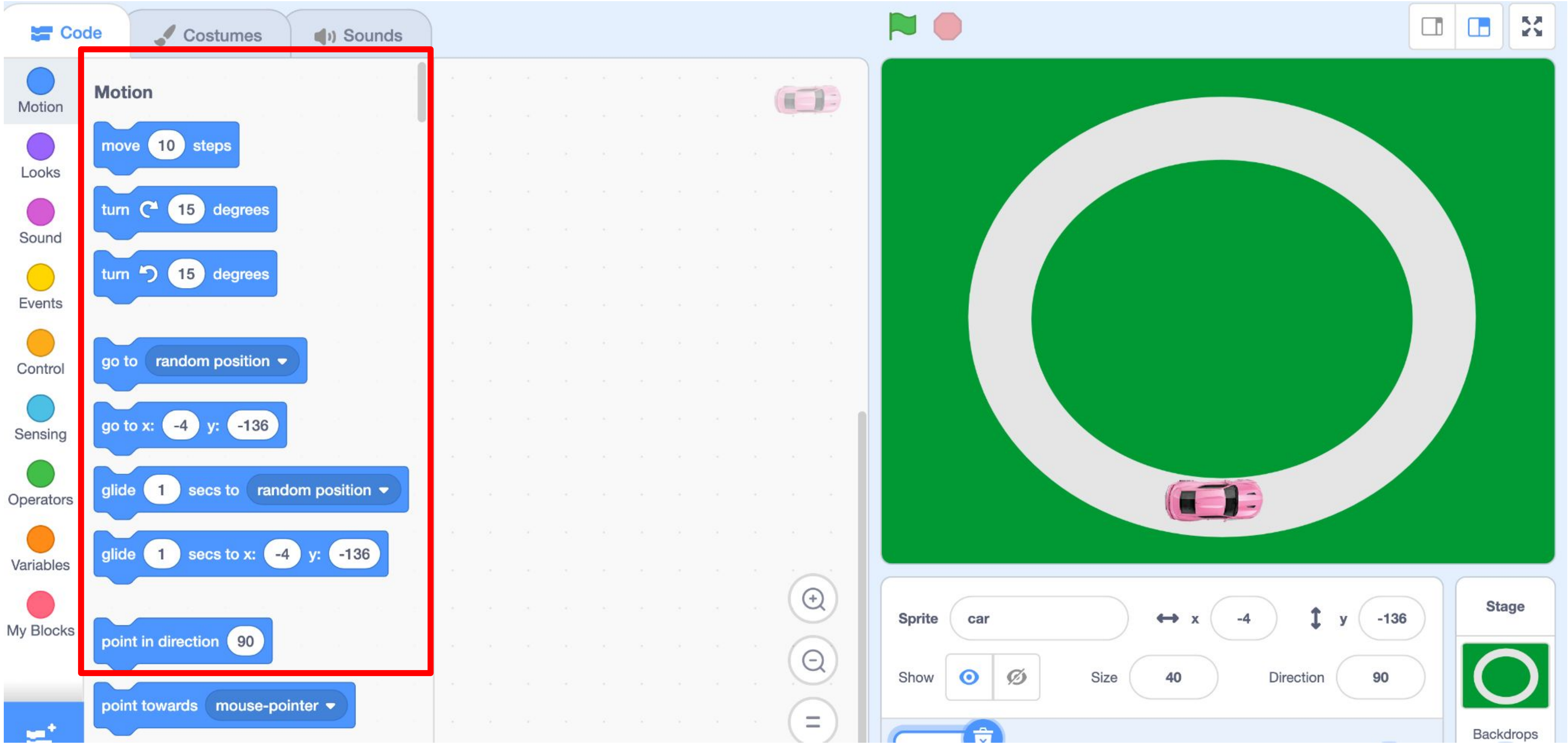
Motion

```
move (10) steps
turn ↻ (15) degrees
turn ↺ (15) degrees
go to (random position ▾)
go to x: (-4) y: (-136)
glide (1) secs to (random position ▾)
glide (1) secs to x: (-4) y: (-136)
point in direction (90)
point towards (mouse-pointer ▾)
```

Sprite car ↔ x -4 ↕ y -136

Show Size 40 Direction 90

These are examples of ‘steps’ or ‘commands’ you can give to the computer. For example: Move 10 steps.

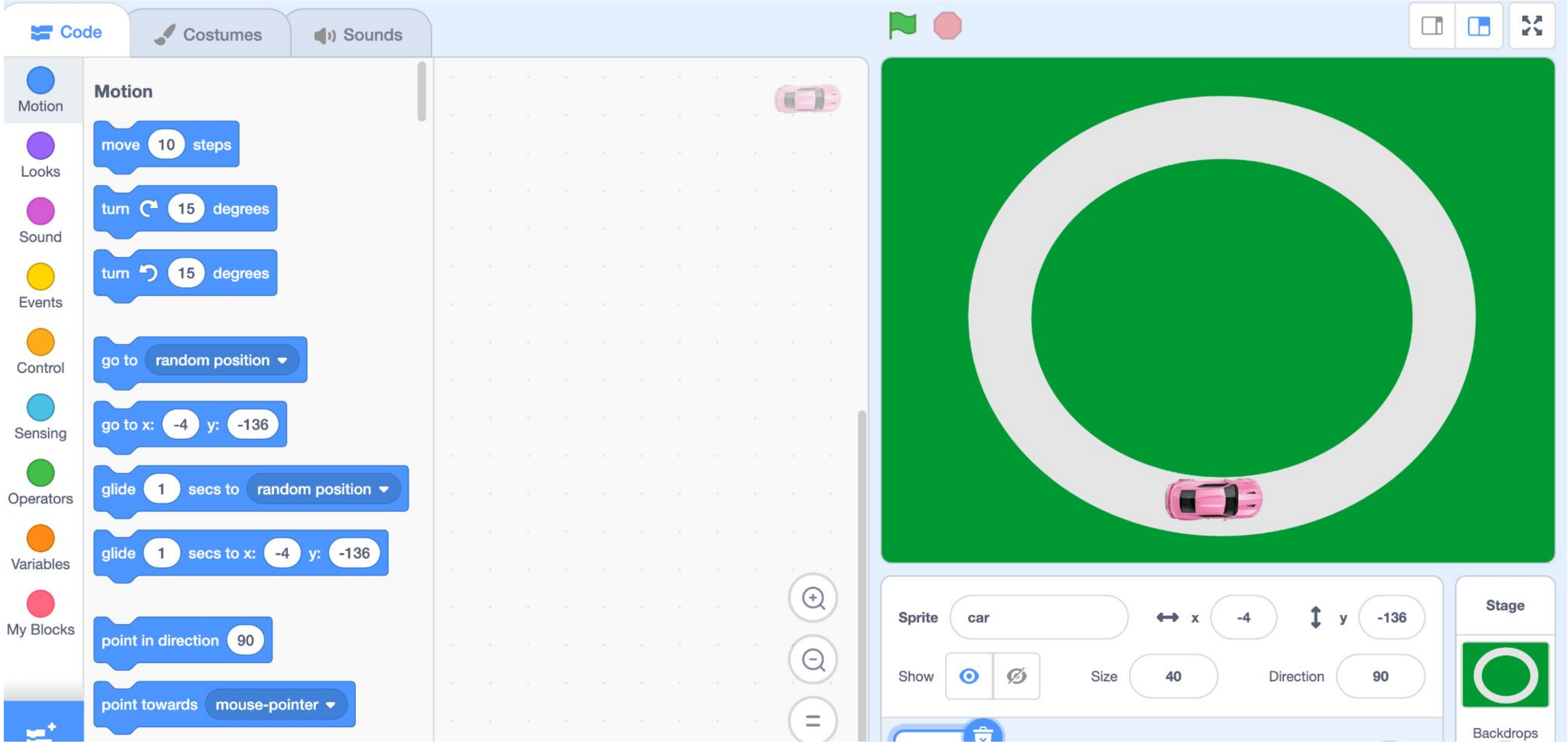


Motion

```
move (10) steps
turn ↻ (15) degrees
turn ↺ (15) degrees
go to (random position ▾)
go to x: (-4) y: (-136)
glide (1) secs to (random position ▾)
glide (1) secs to x: (-4) y: (-136)
point in direction (90)
point towards (mouse-pointer ▾)
```

Sprite: car x: -4 y: -136
Show Size: 40 Direction: 90

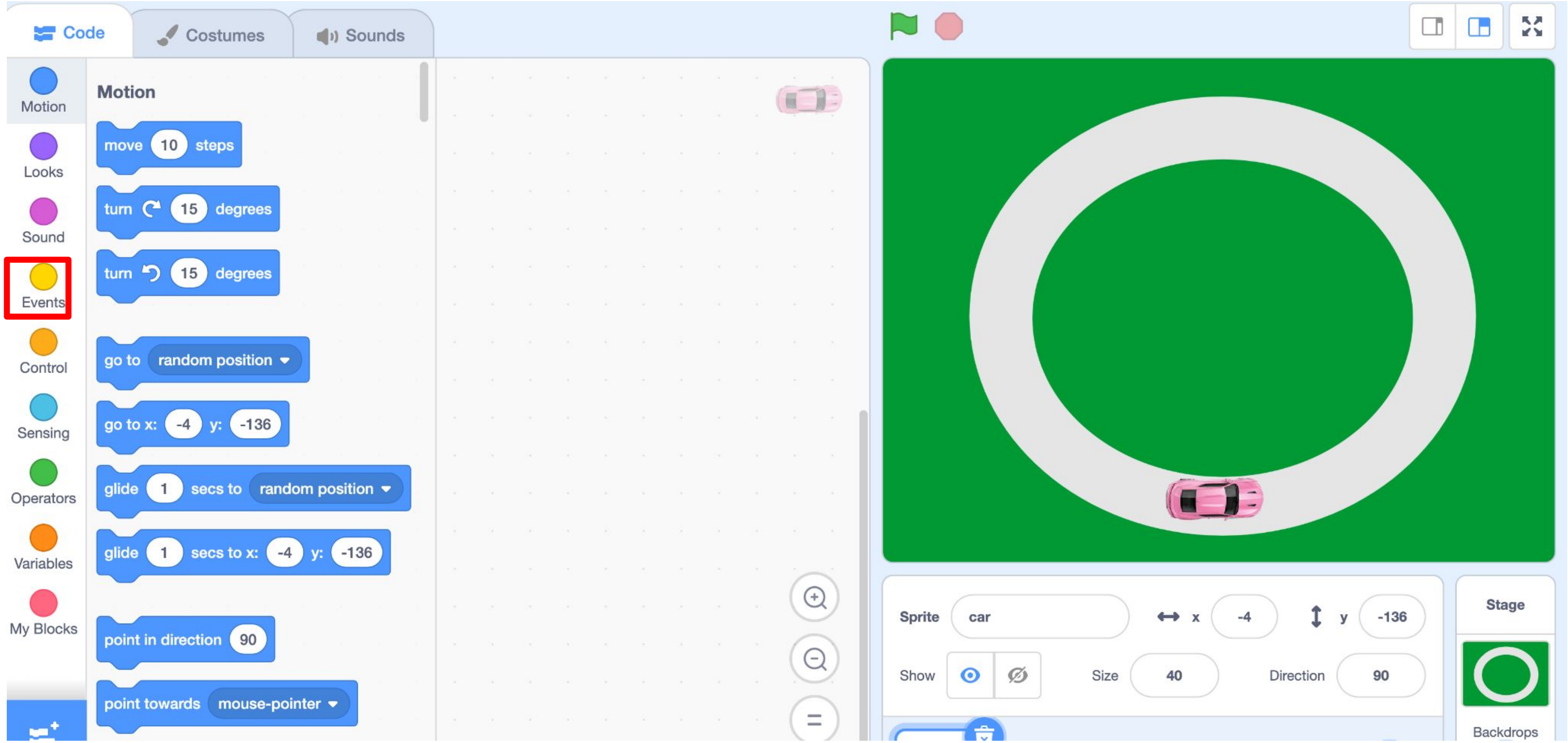
Let’s try moving the car!



Sprite car ↔ x -4 ↕ y -136

Show Size 40 Direction 90

Click on Events



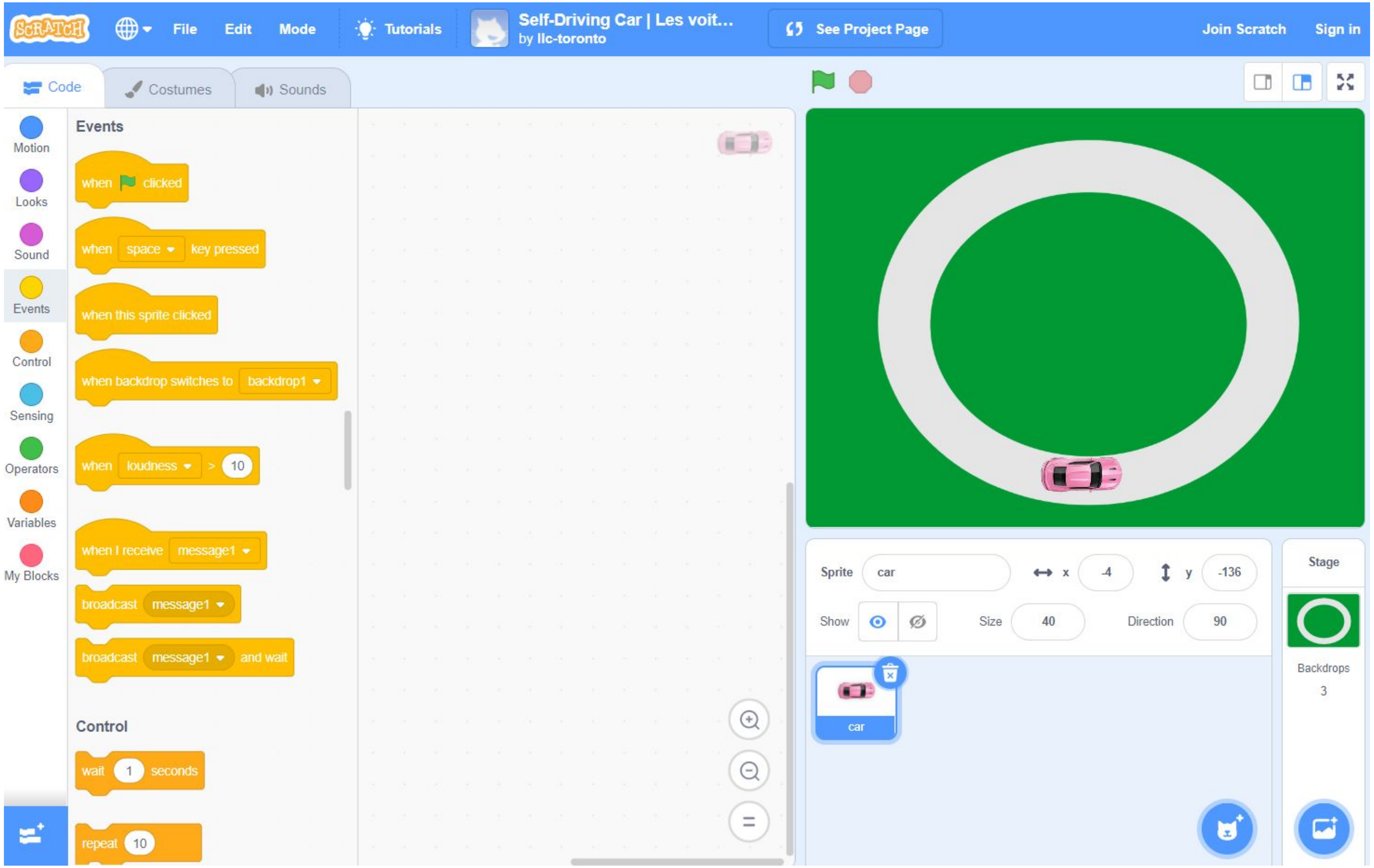
Motion

```
move (10) steps
turn ↻ (15) degrees
turn ↺ (15) degrees
go to (random position ▾)
go to x: (-4) y: (-136)
glide (1) secs to (random position ▾)
glide (1) secs to x: (-4) y: (-136)
point in direction (90)
point towards (mouse-pointer ▾)
```

Sprite car ↔ x -4 ↕ y -136

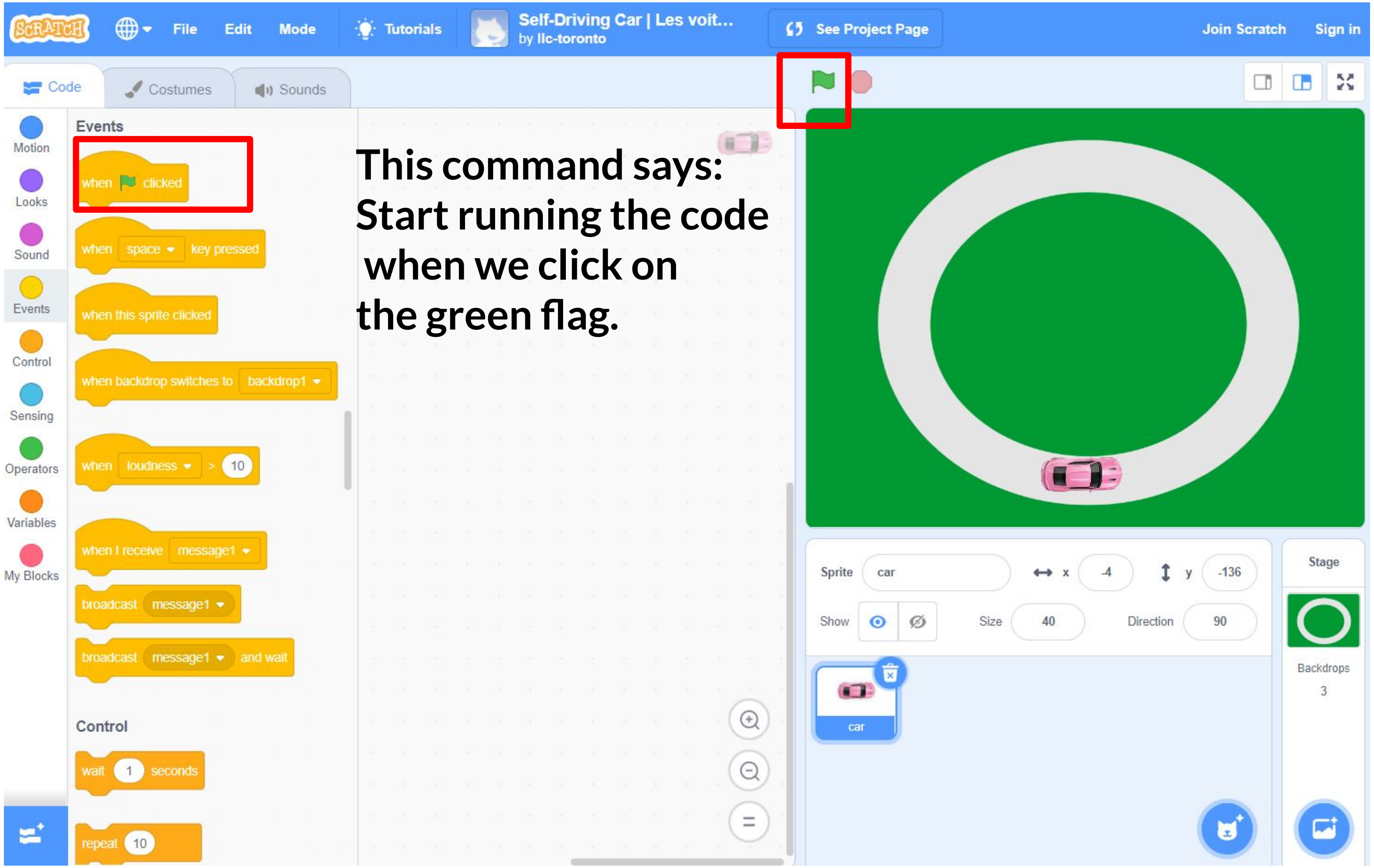
Show Size 40 Direction 90

Now you see this:



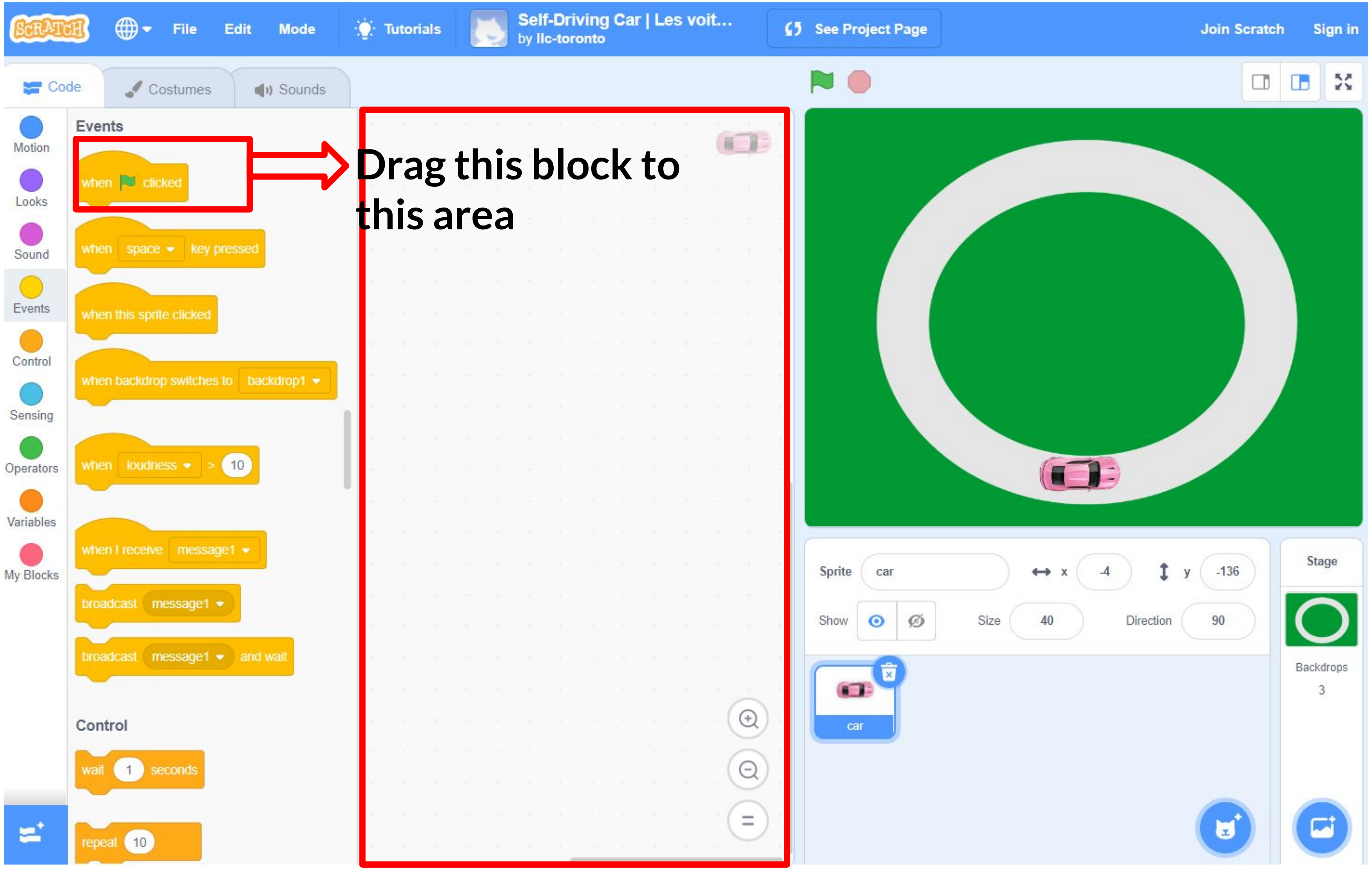
Sprite: car
x: -4
y: -136
Show
Size: 40
Direction: 90

Now you see this:



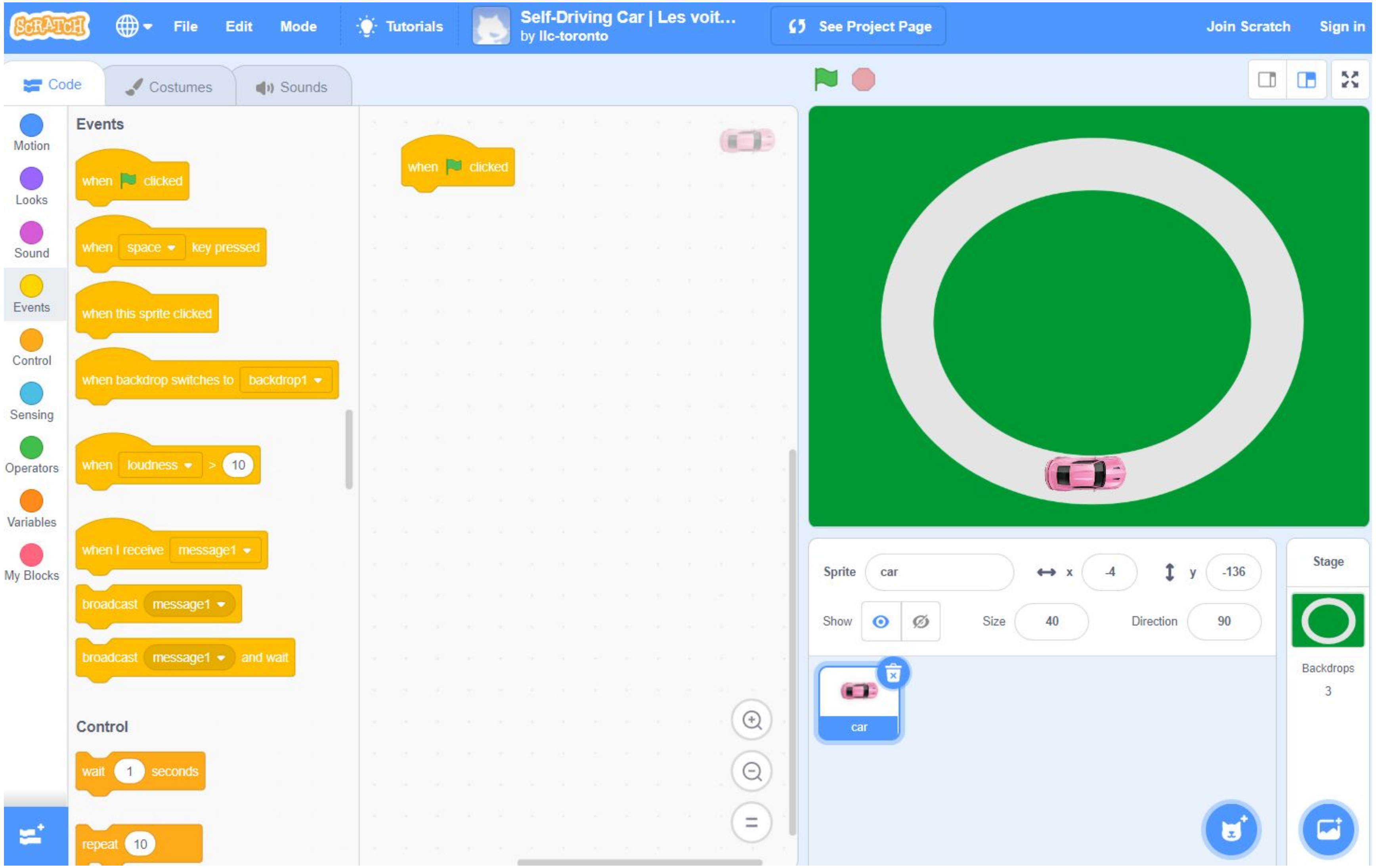
This command says: Start running the code when we click on the green flag.

Now you see this:



Drag this block to this area

Now you see this:

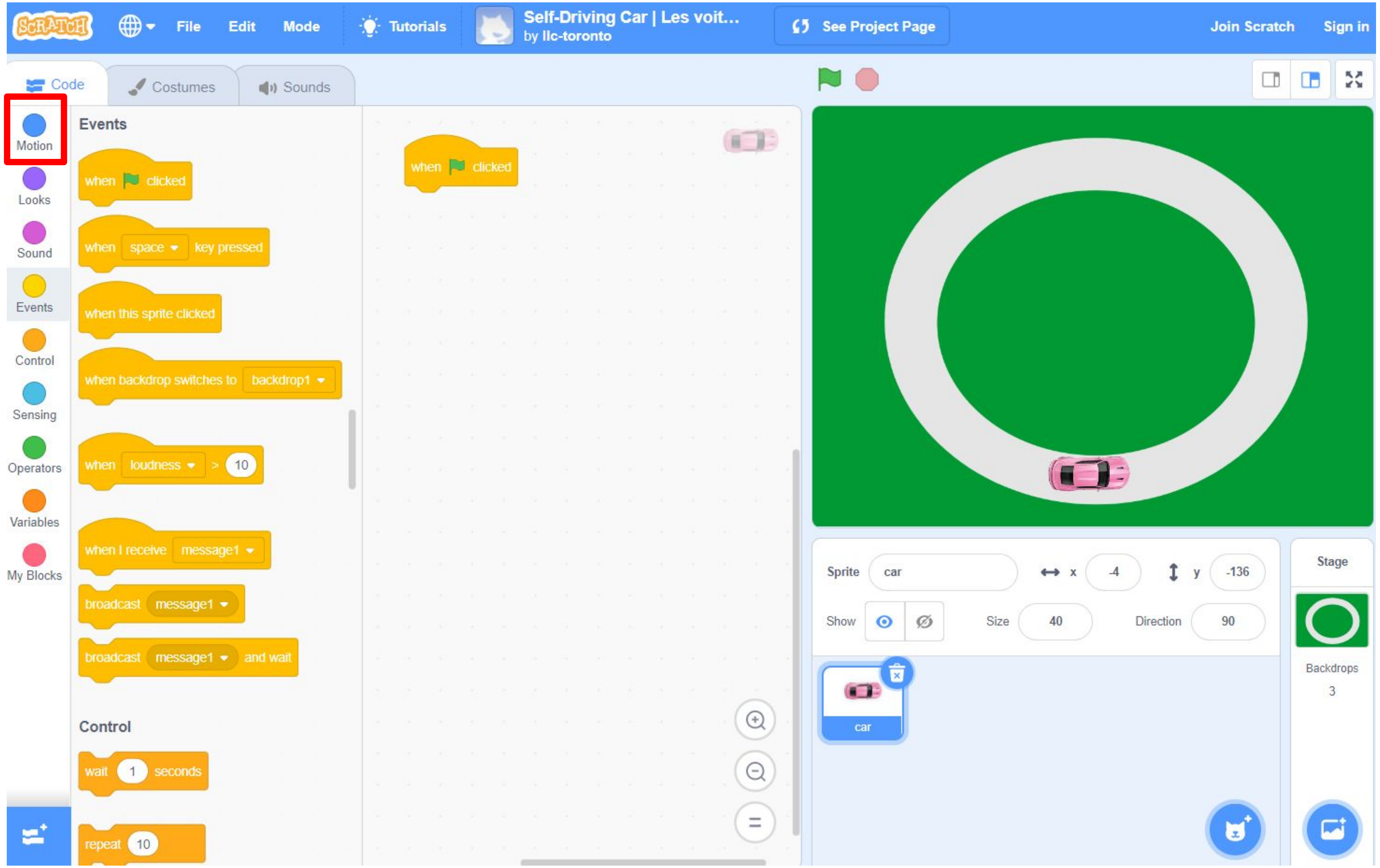


```
when 🏴 clicked
```

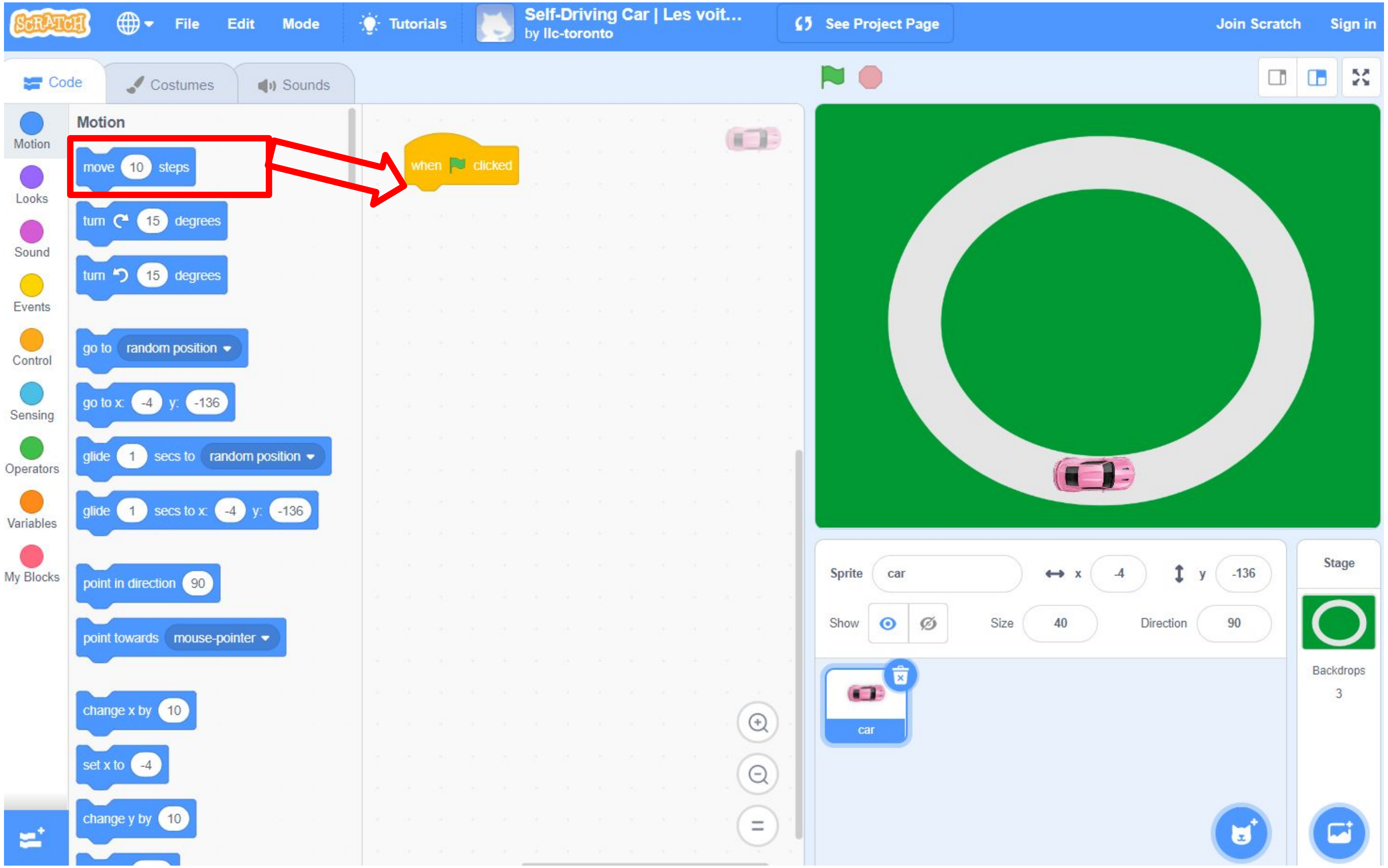
Sprite: car; x: -4; y: -136

Show; Size: 40; Direction: 90

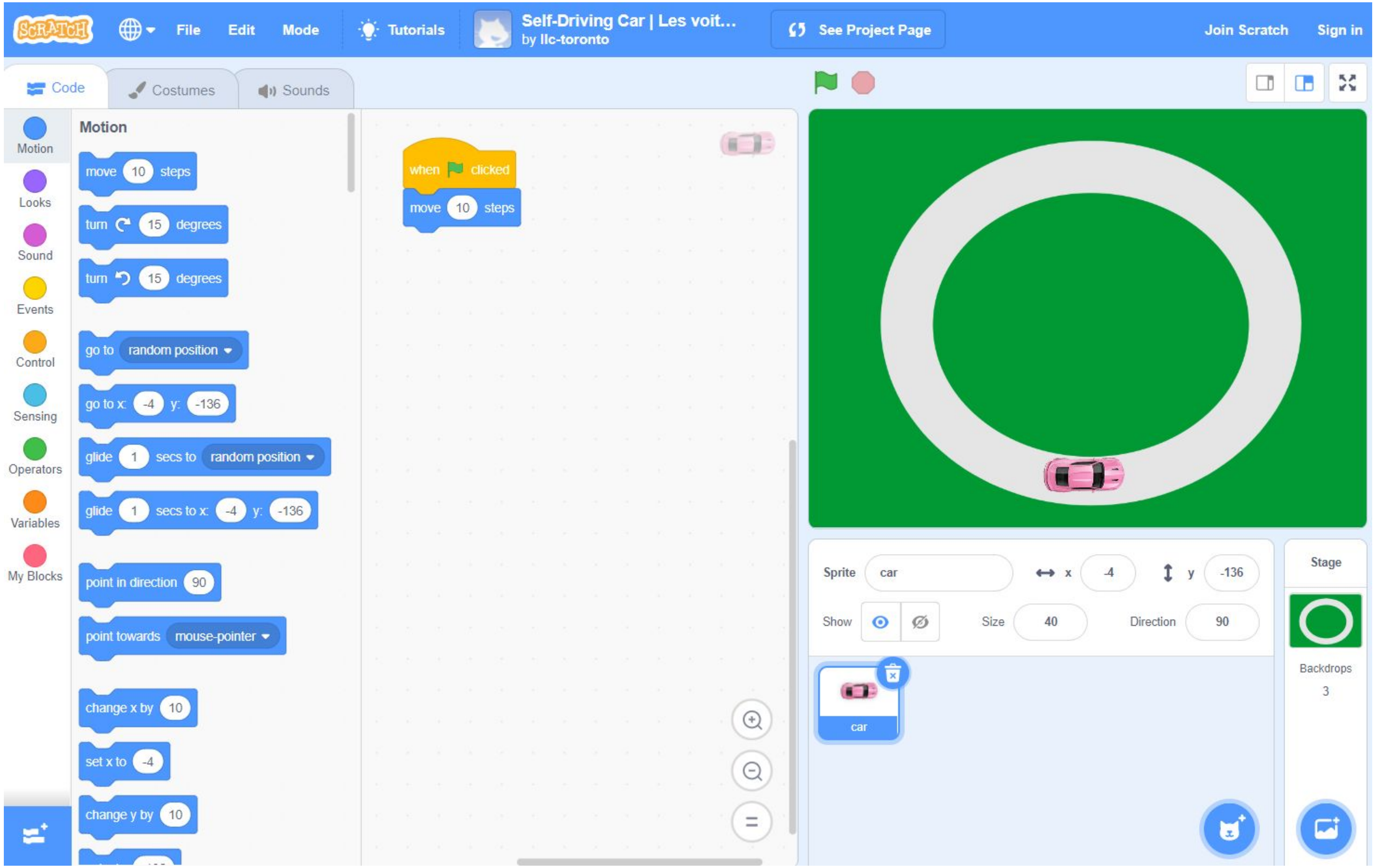
Click on Motions



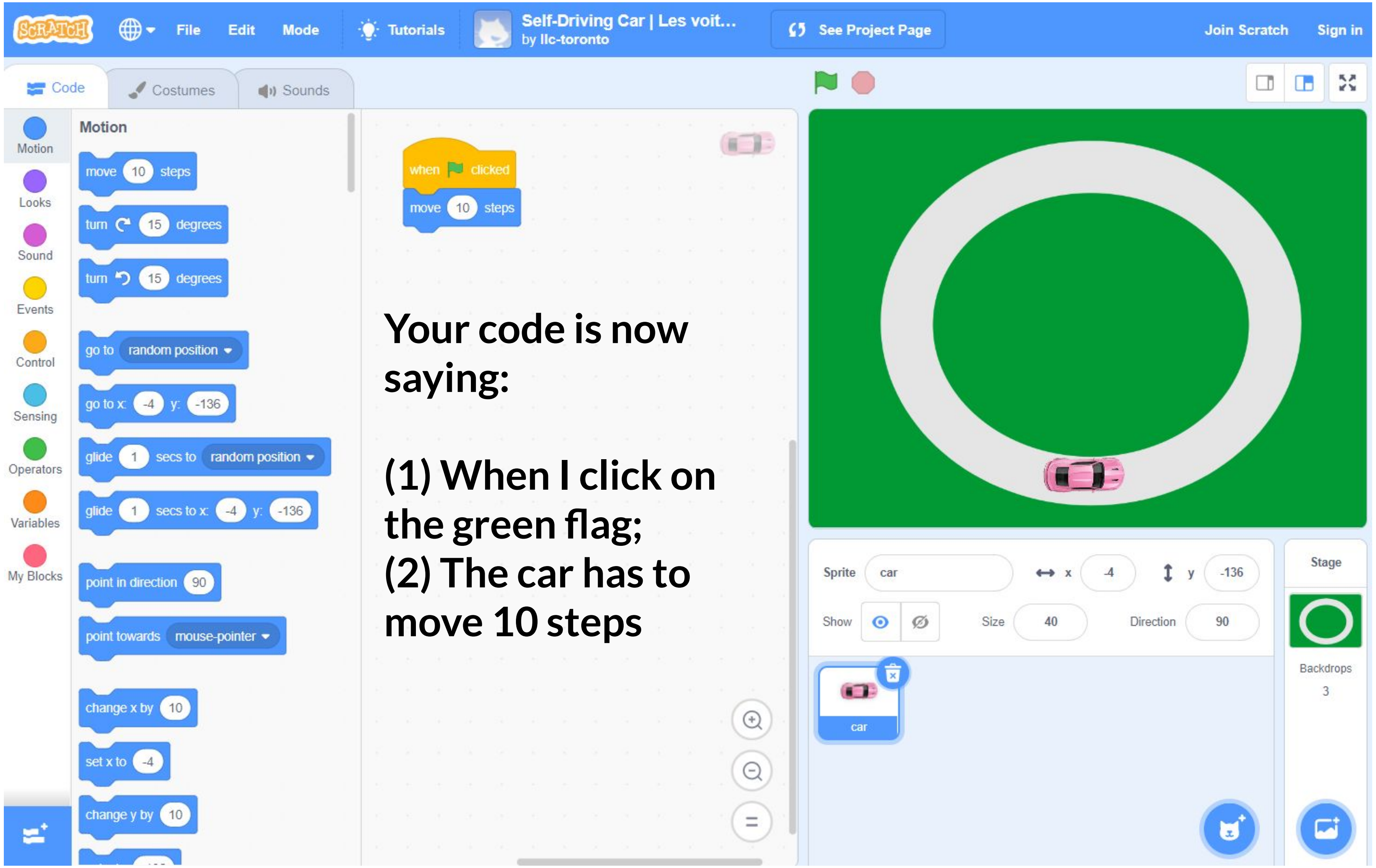
Drag this command under your other command



Drag this command under your other command



Drag this command under your other command

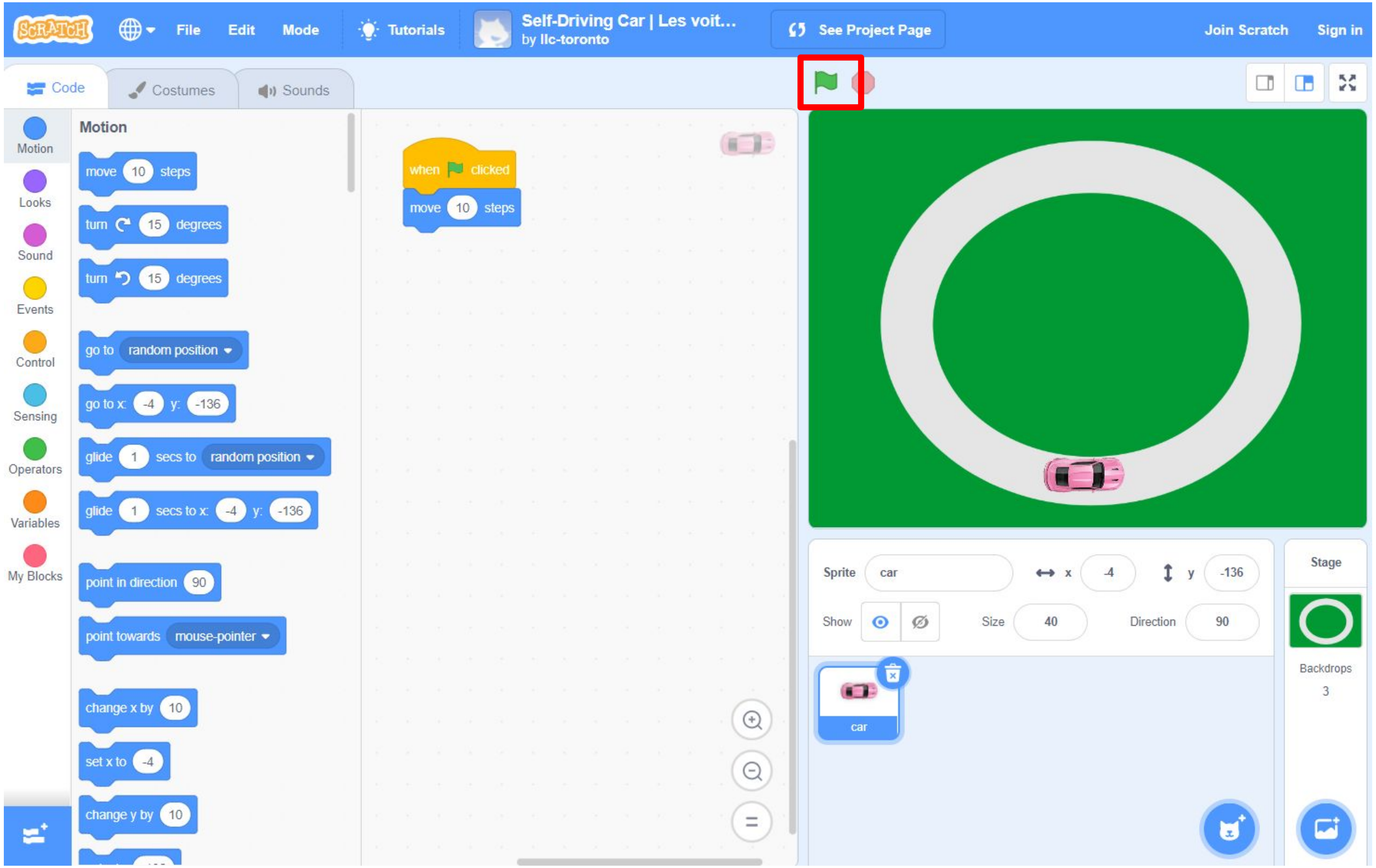


```
when 🏁 clicked
move 10 steps
```

Your code is now saying:

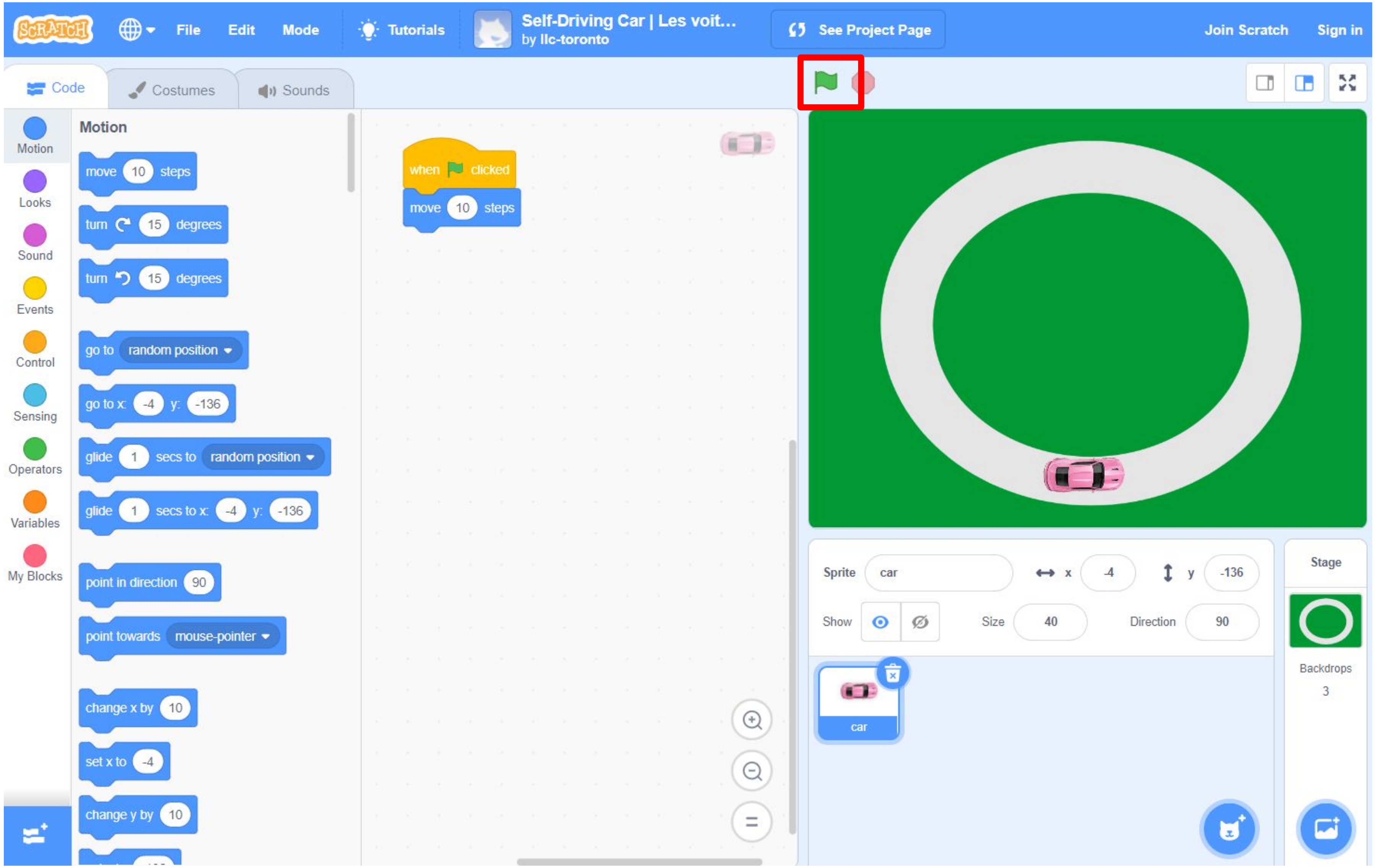
(1) When I click on the green flag;
(2) The car has to move 10 steps

Let’s try it out! Click on the green flag.



when 🏁 clicked
move 10 steps

Did your car move? Great job!

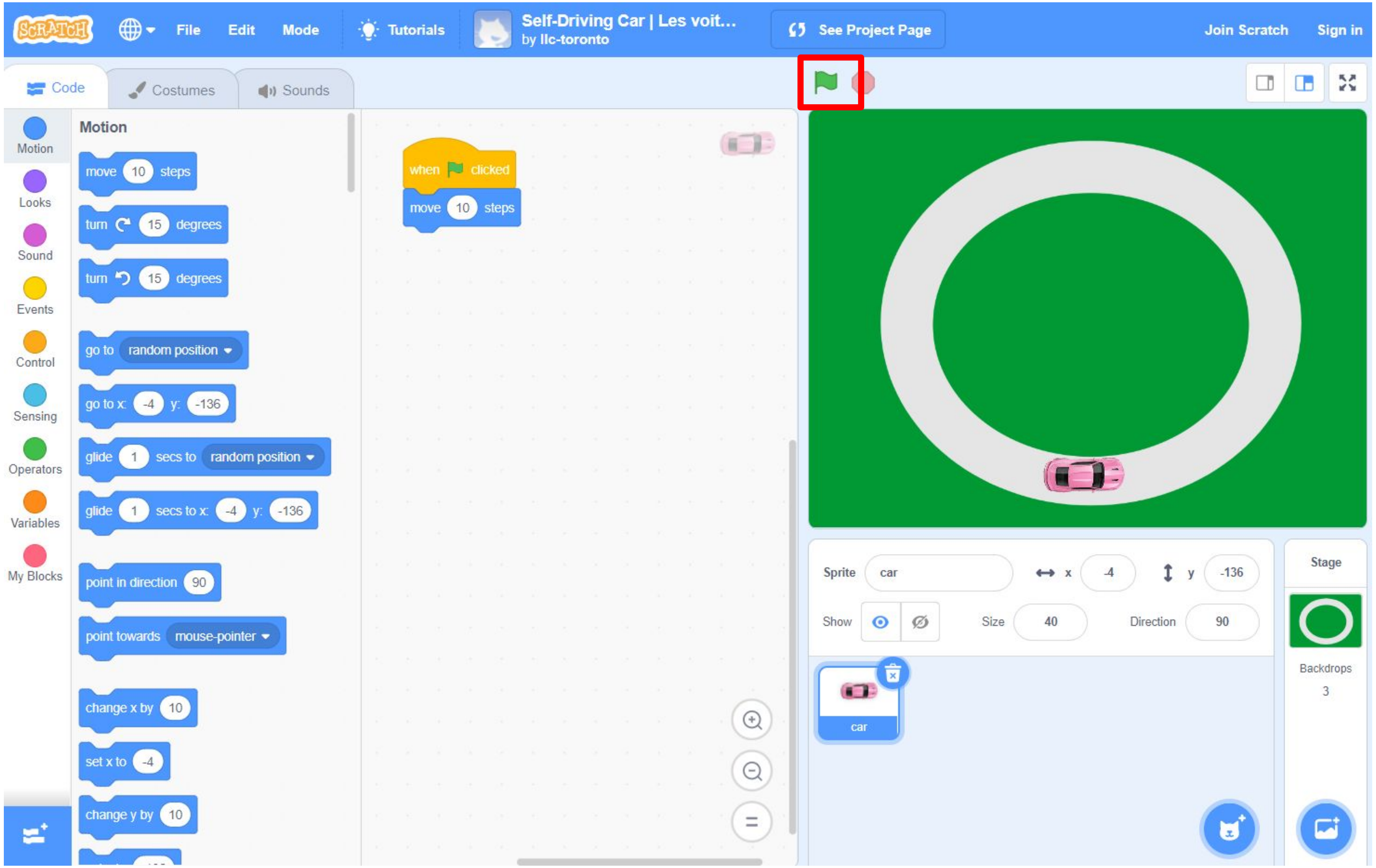


```
when ⚑ clicked
move (10) steps
```

Sprite: car x: -4 y: -136

Show Size: 40 Direction: 90

What happens if you put 100 instead of 10 and press the green flag again?

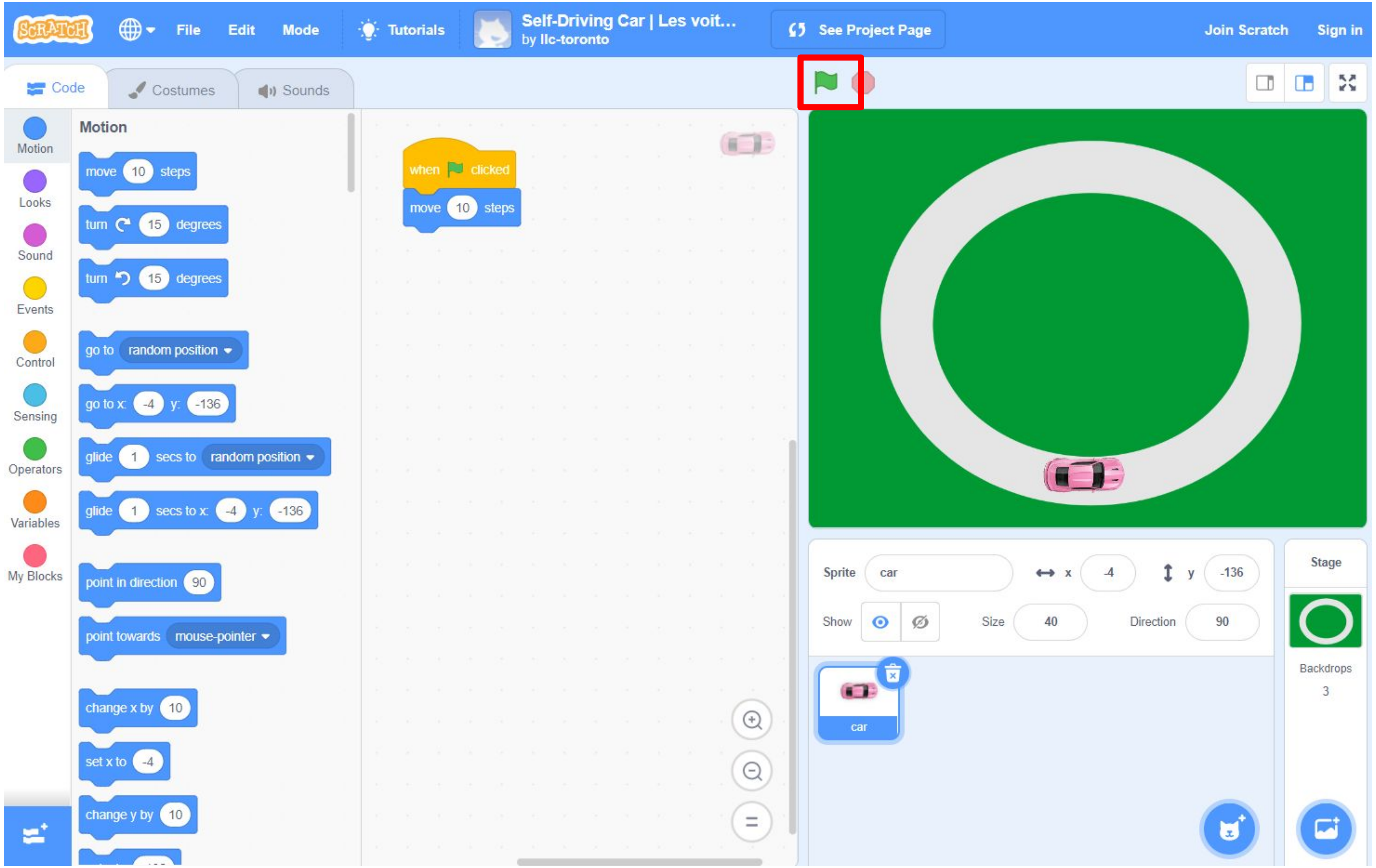


```
when 🏁 clicked
move (10) steps
```

Sprite: car x: -4 y: -136

Show Size: 40 Direction: 90

And -100?

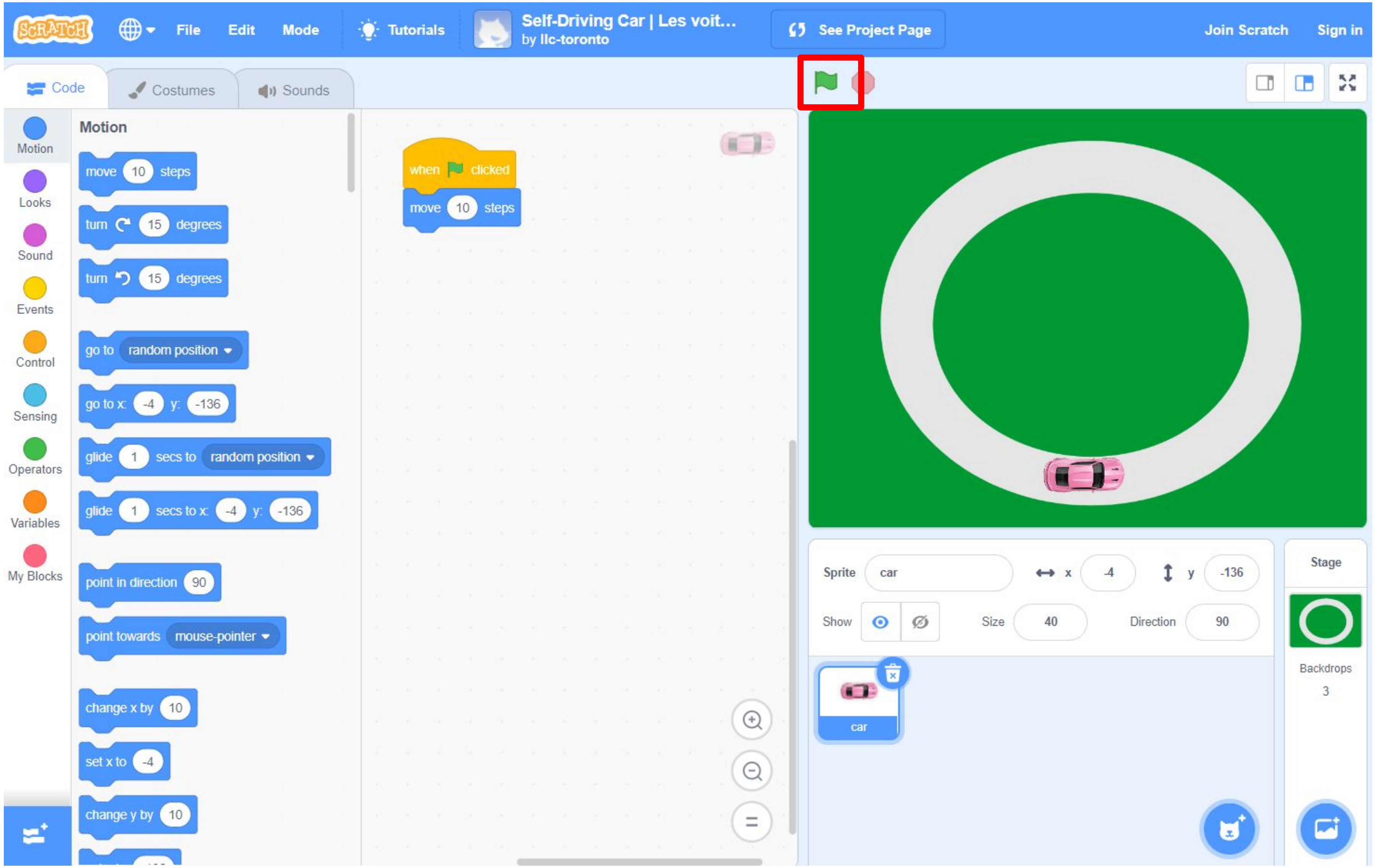


```
when 🏴 clicked
move 10 steps
```

Sprite: car x: -4 y: -136

Show Size: 40 Direction: 90

You can now move the car!

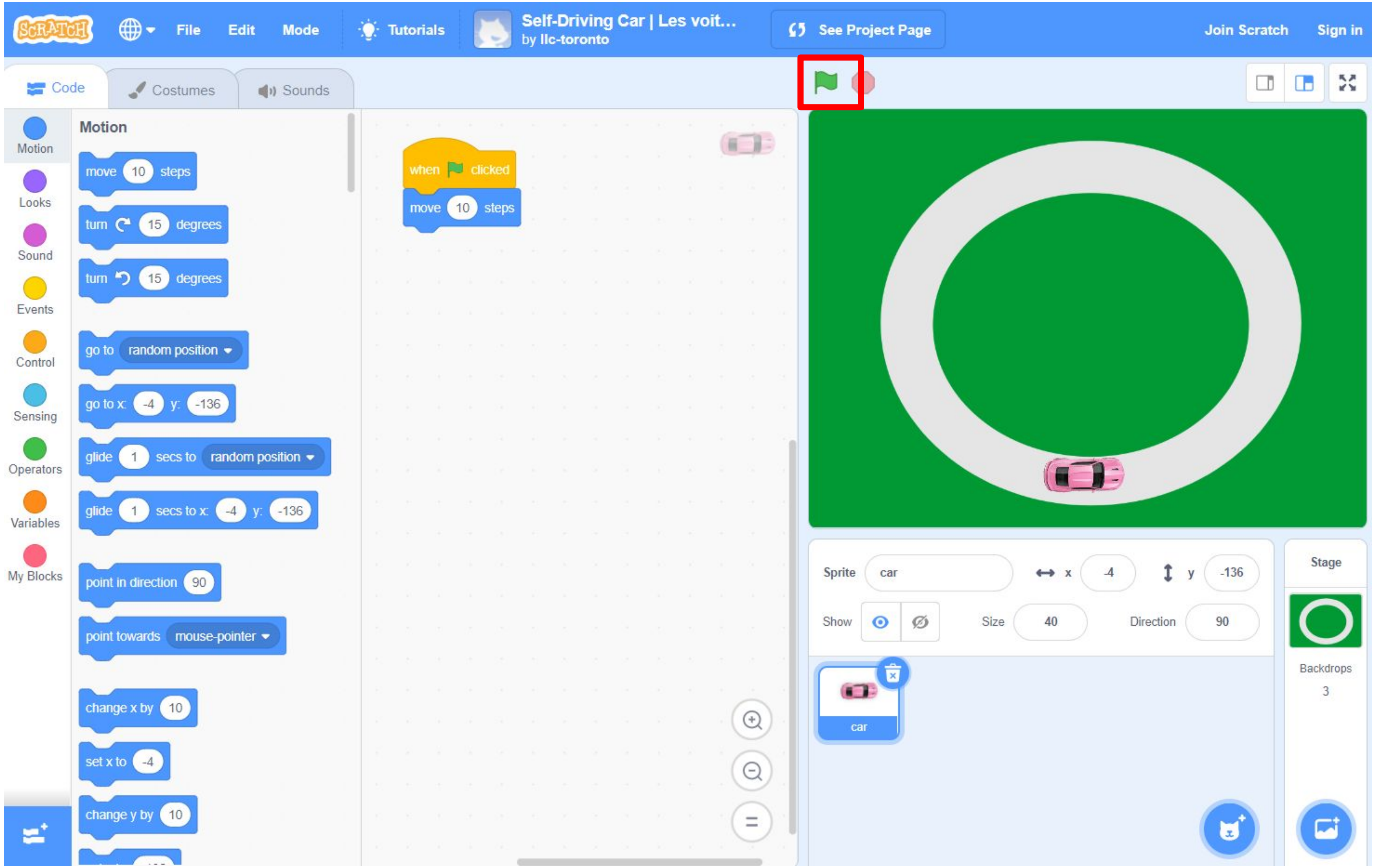


```
when ⚑ clicked
move 10 steps
```

Sprite: car x: -4 y: -136

Show: ● Size: 40 Direction: 90

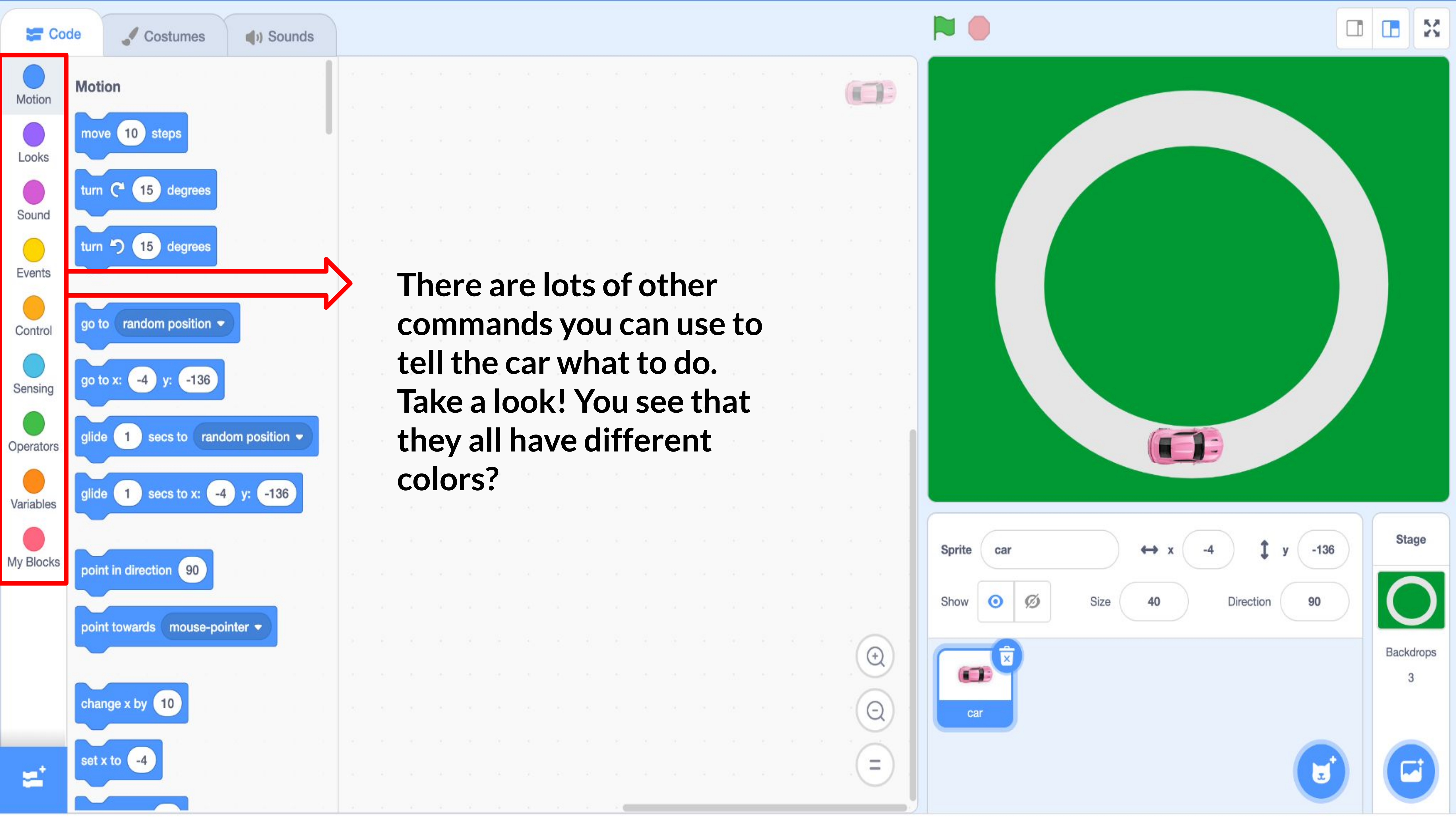
Let’s see what Scratch can do more!



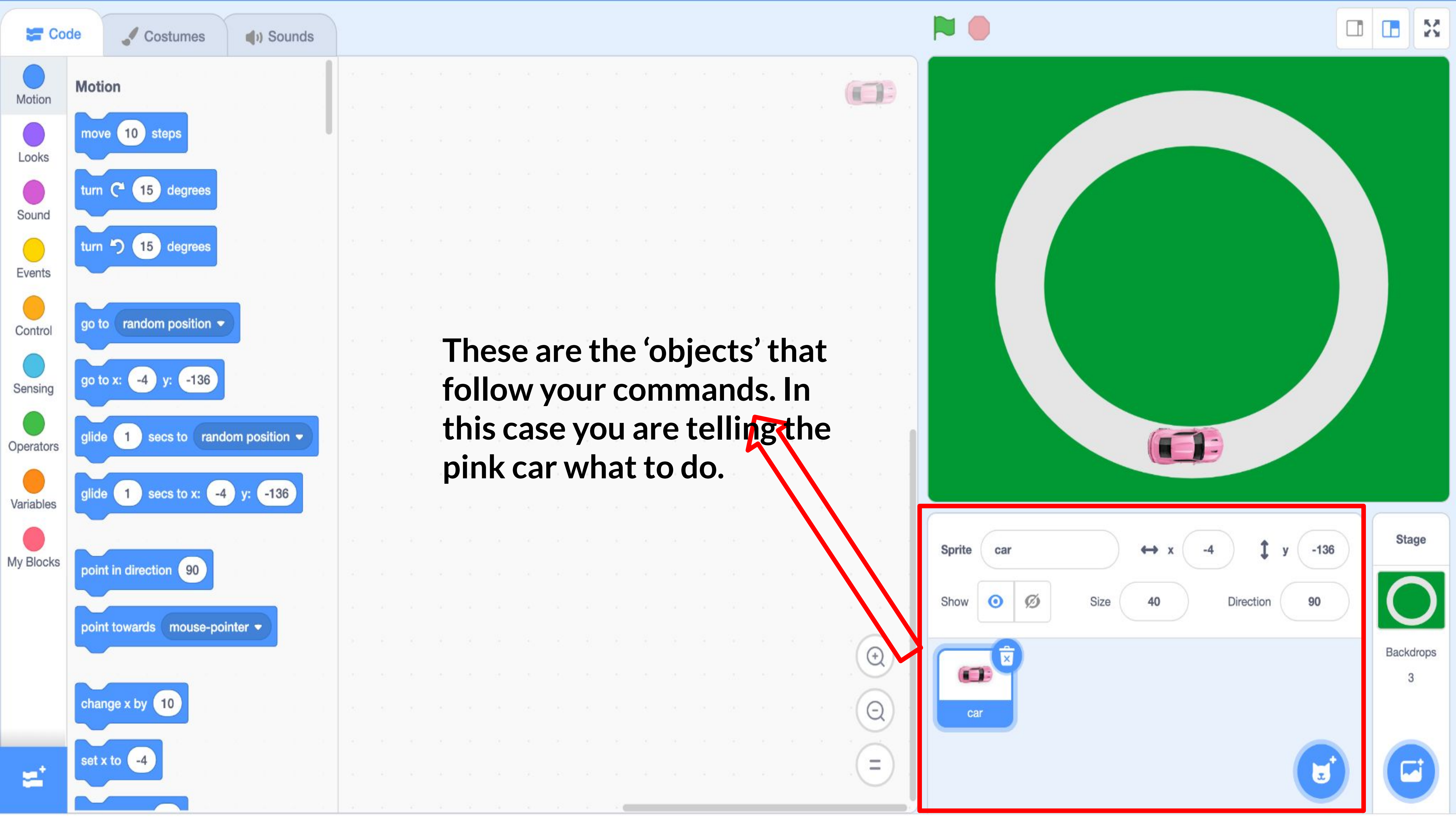
```
when 🏁 clicked
move 10 steps
```

Sprite car ↔ x -4 ↕ y -136

Show Size 40 Direction 90



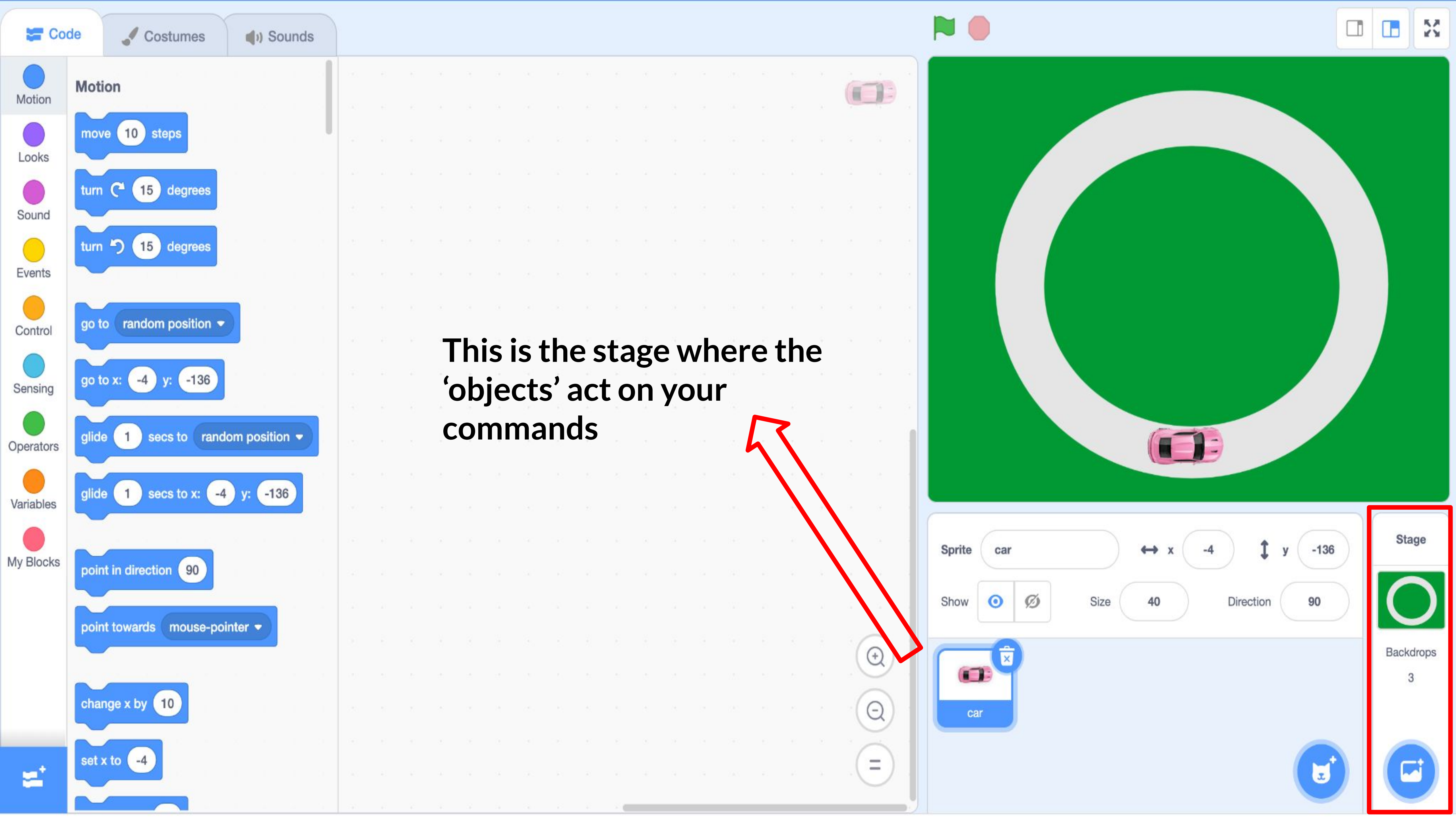
There are lots of other commands you can use to tell the car what to do. Take a look! You see that they all have different colors?



Sprite car x -4 y -136

Show Size 40 Direction 90

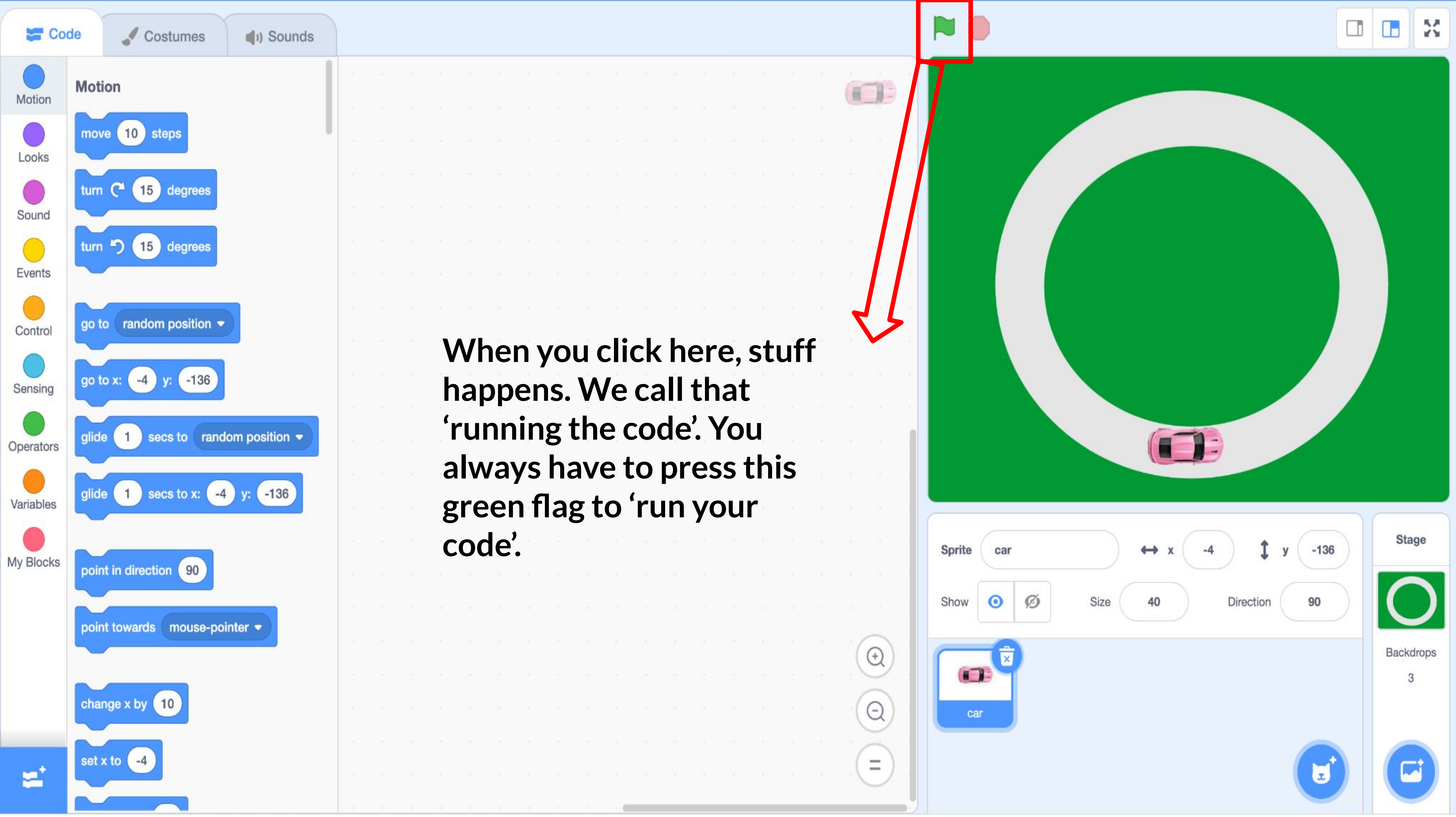
These are the 'objects' that follow your commands. In this case you are telling the pink car what to do.



Sprite car x -4 y -136

Show Size 40 Direction 90

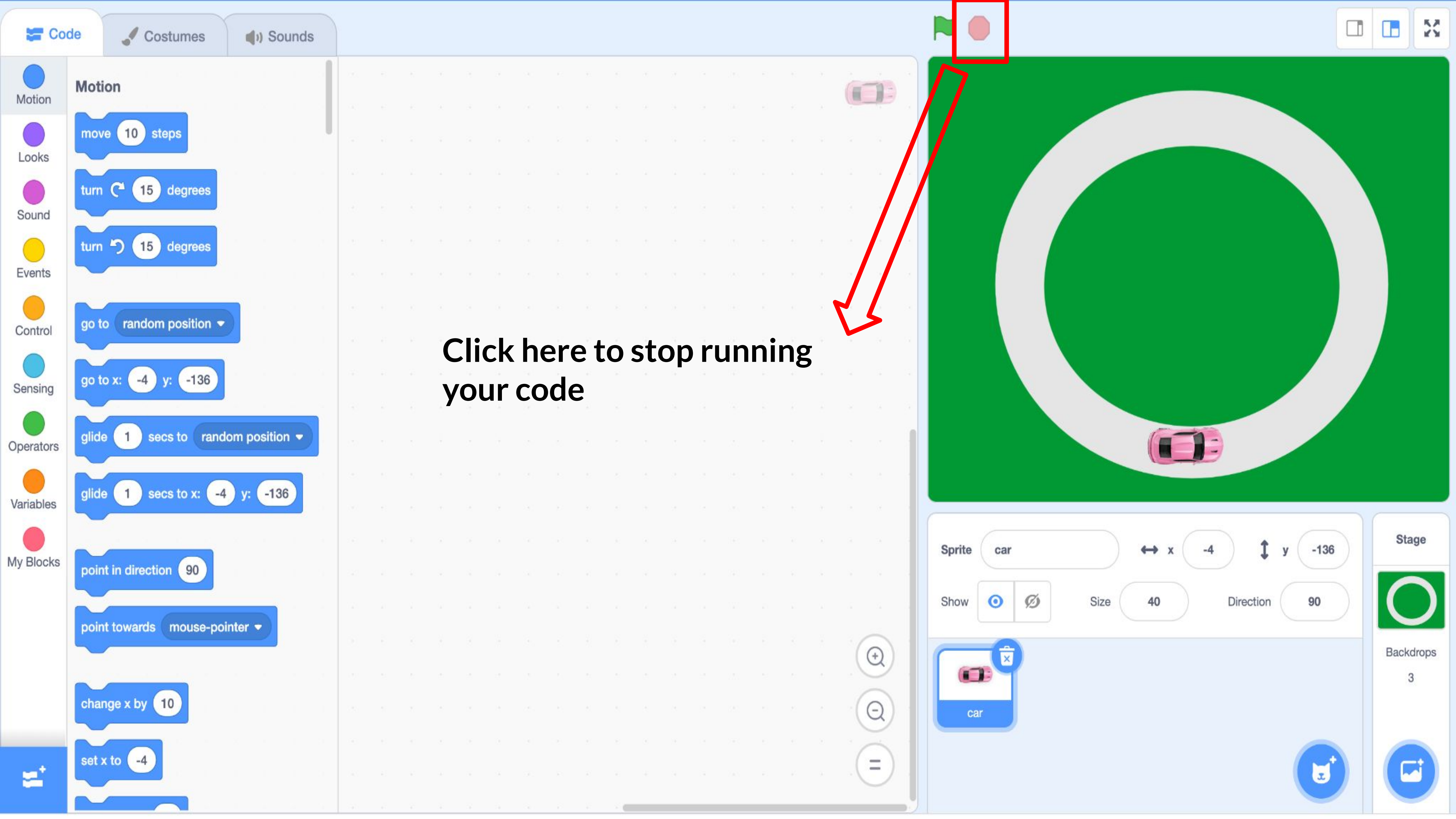
This is the stage where the ‘objects’ act on your commands



Sprite car ↔ x -4 ↕ y -136

Show Size 40 Direction 90

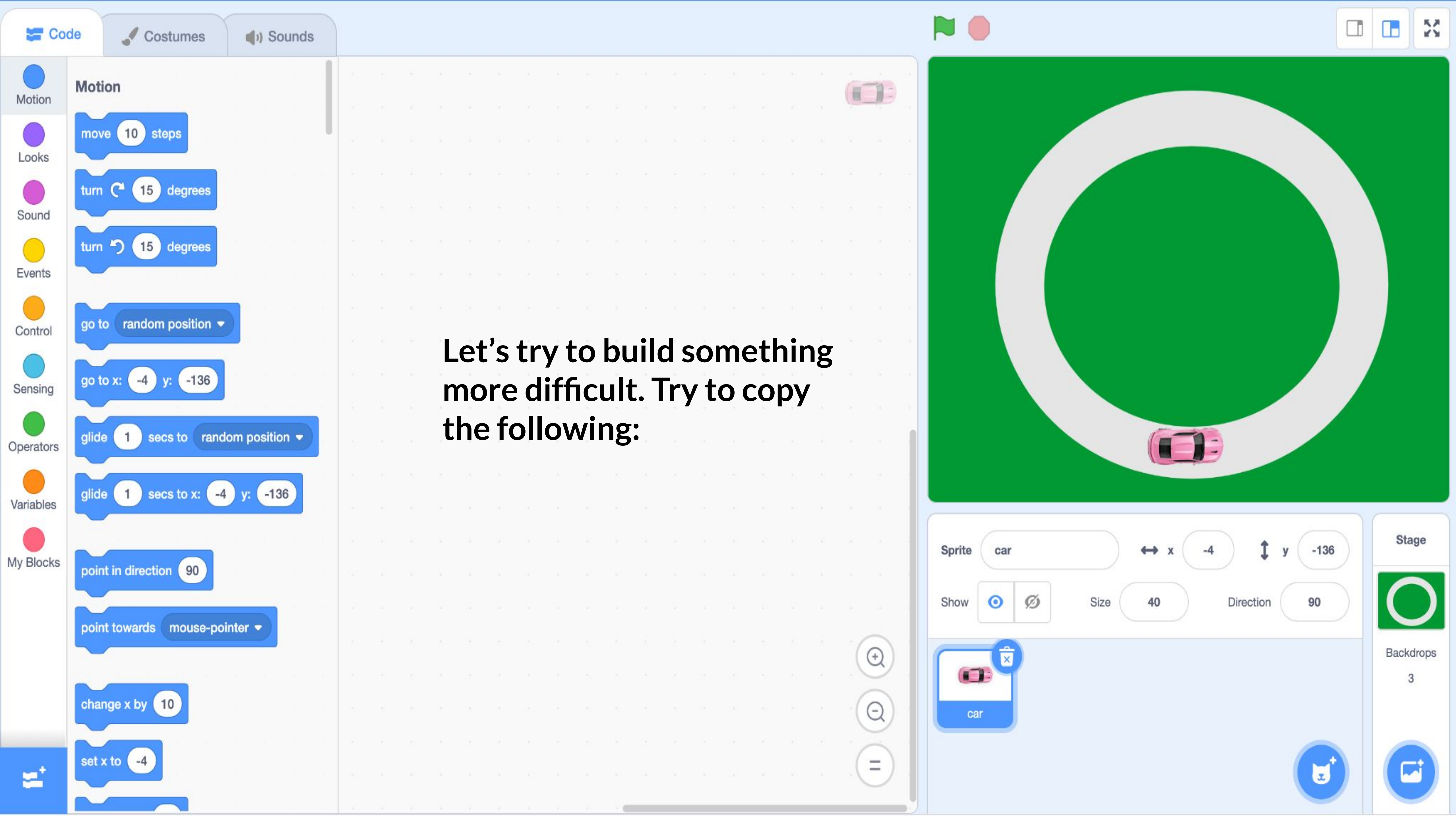
When you click here, stuff happens. We call that ‘running the code’. You always have to press this green flag to ‘run your code’.



Sprite car x -4 y -136

Show Size 40 Direction 90

Click here to stop running your code



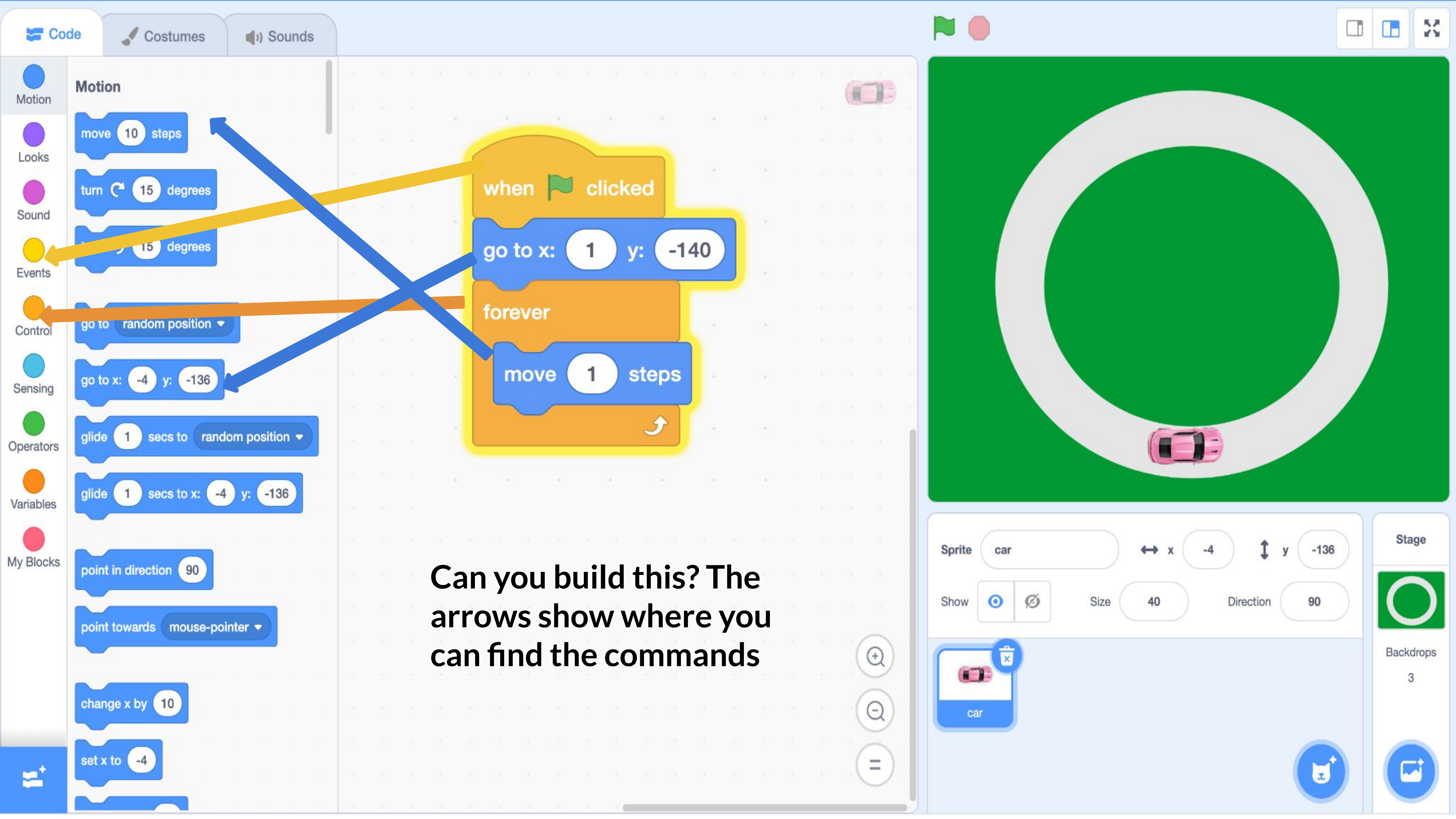
Let's try to build something more difficult. Try to copy the following:

Motion

```
move (10) steps
turn ↻ (15) degrees
turn ↺ (15) degrees
go to (random position ▾)
go to x: (-4) y: (-136)
glide (1) secs to (random position ▾)
glide (1) secs to x: (-4) y: (-136)
point in direction (90)
point towards (mouse-pointer ▾)
change x by (10)
set x to (-4)
```

Sprite: car x: -4 y: -136

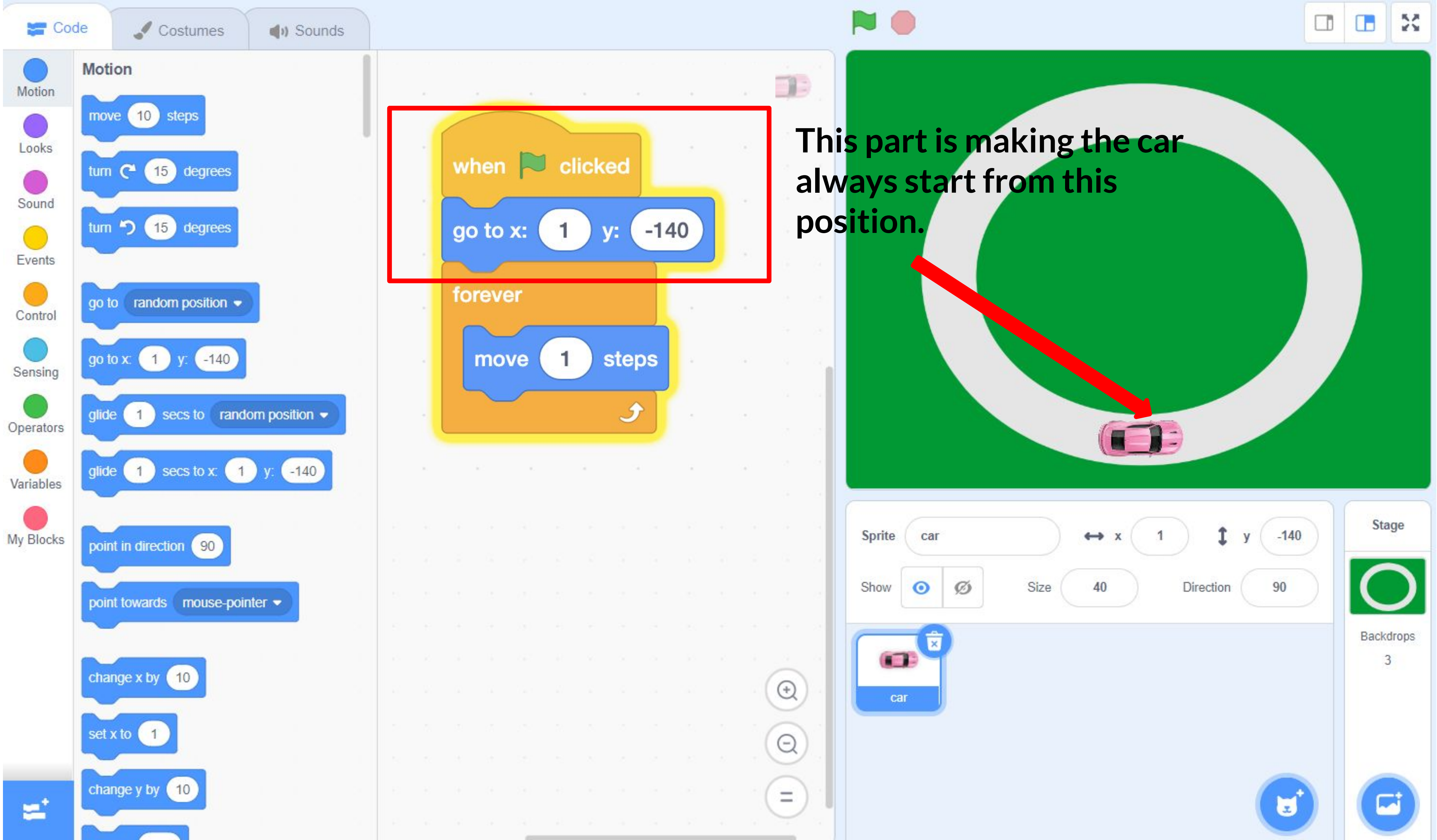
Show Size: 40 Direction: 90



```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
```

Sprite: car x: -4 y: -136
Show Size: 40 Direction: 90

Can you build this? The arrows show where you can find the commands

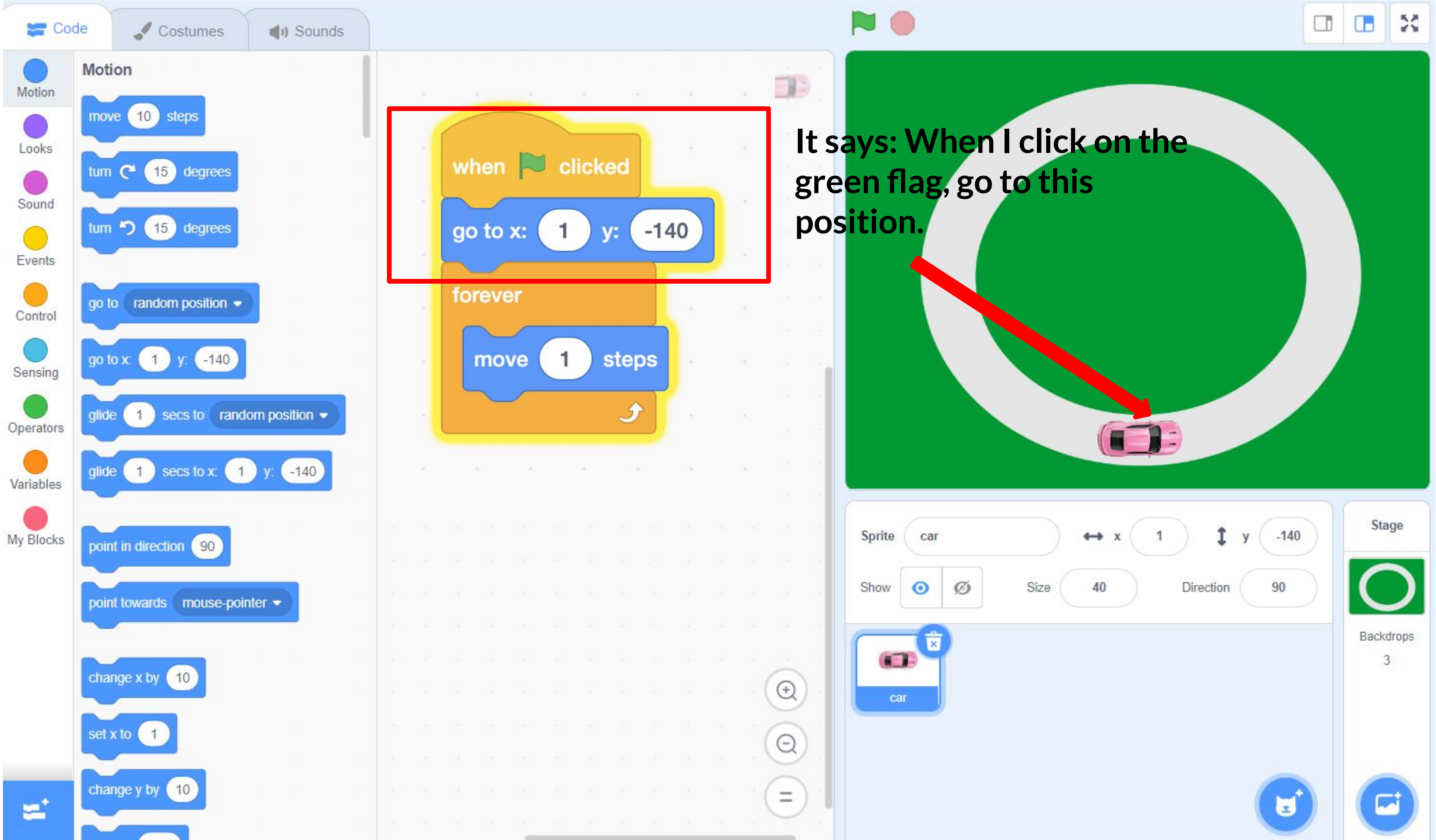


```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
```

Sprite: car   x: 1   y: -140

Show   Size: 40   Direction: 90

This part is making the car always start from this position.

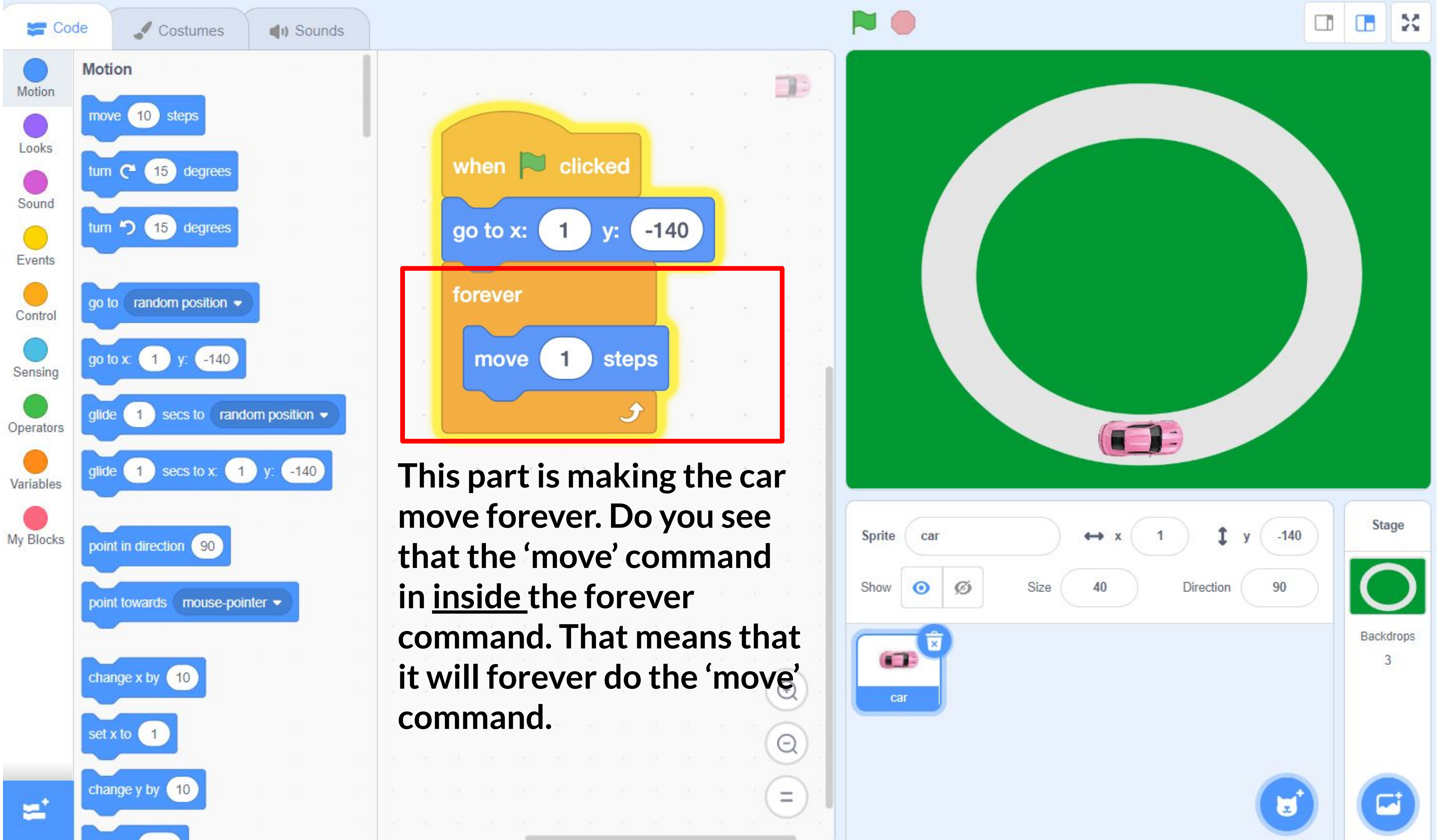


```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
```

Sprite: car x: 1 y: -140

Show: 👁 Size: 40 Direction: 90

It says: When I click on the green flag, go to this position.

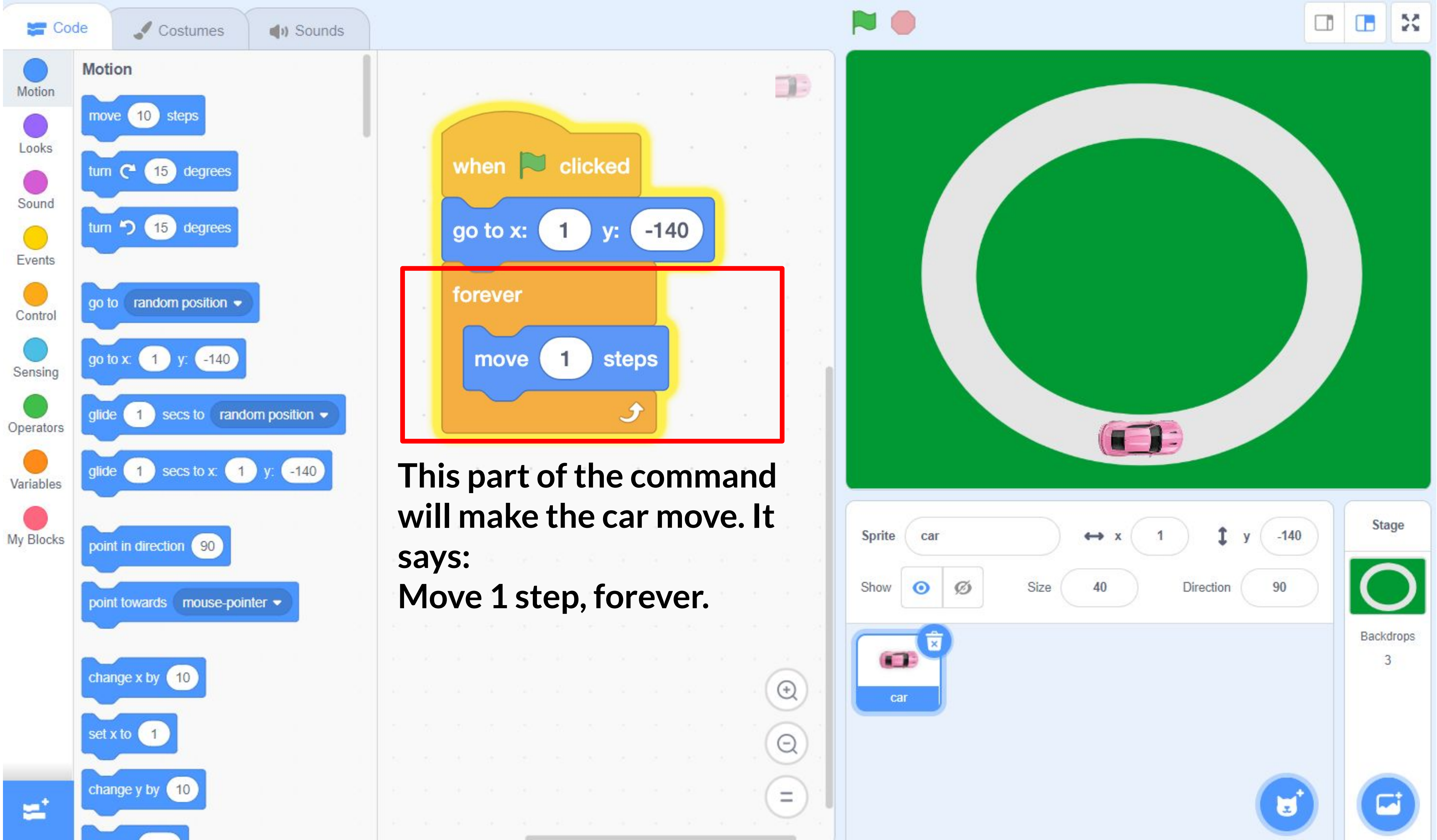


```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
```

Sprite car ↔ x 1 ↕ y -140

Show Size 40 Direction 90

This part is making the car move forever. Do you see that the 'move' command in <u>inside </u>the forever command. That means that it will forever do the 'move' command.



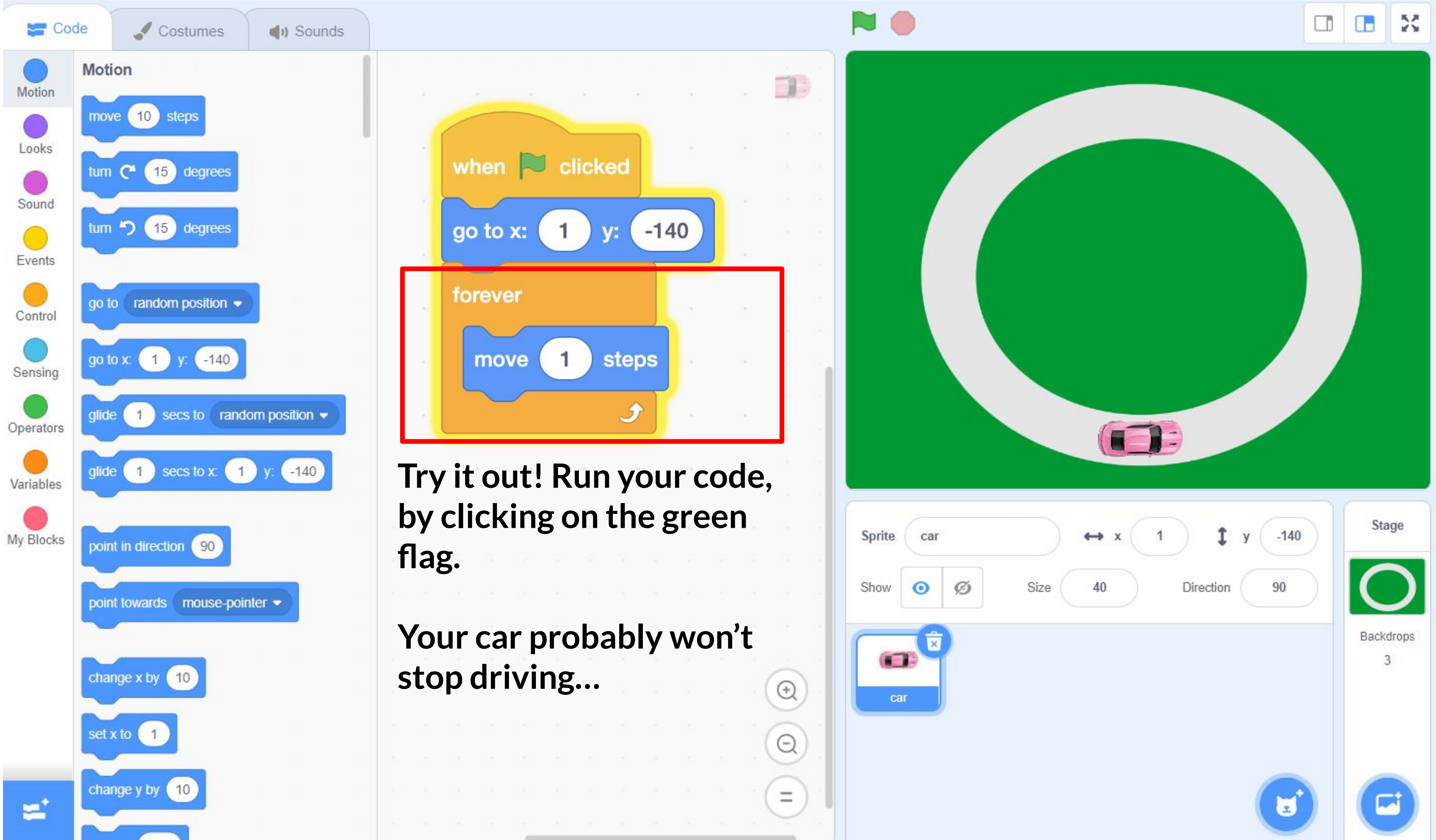
```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
```

Sprite car ↔ x 1 ↕ y -140

Show Size 40 Direction 90

This part of the command will make the car move. It says:

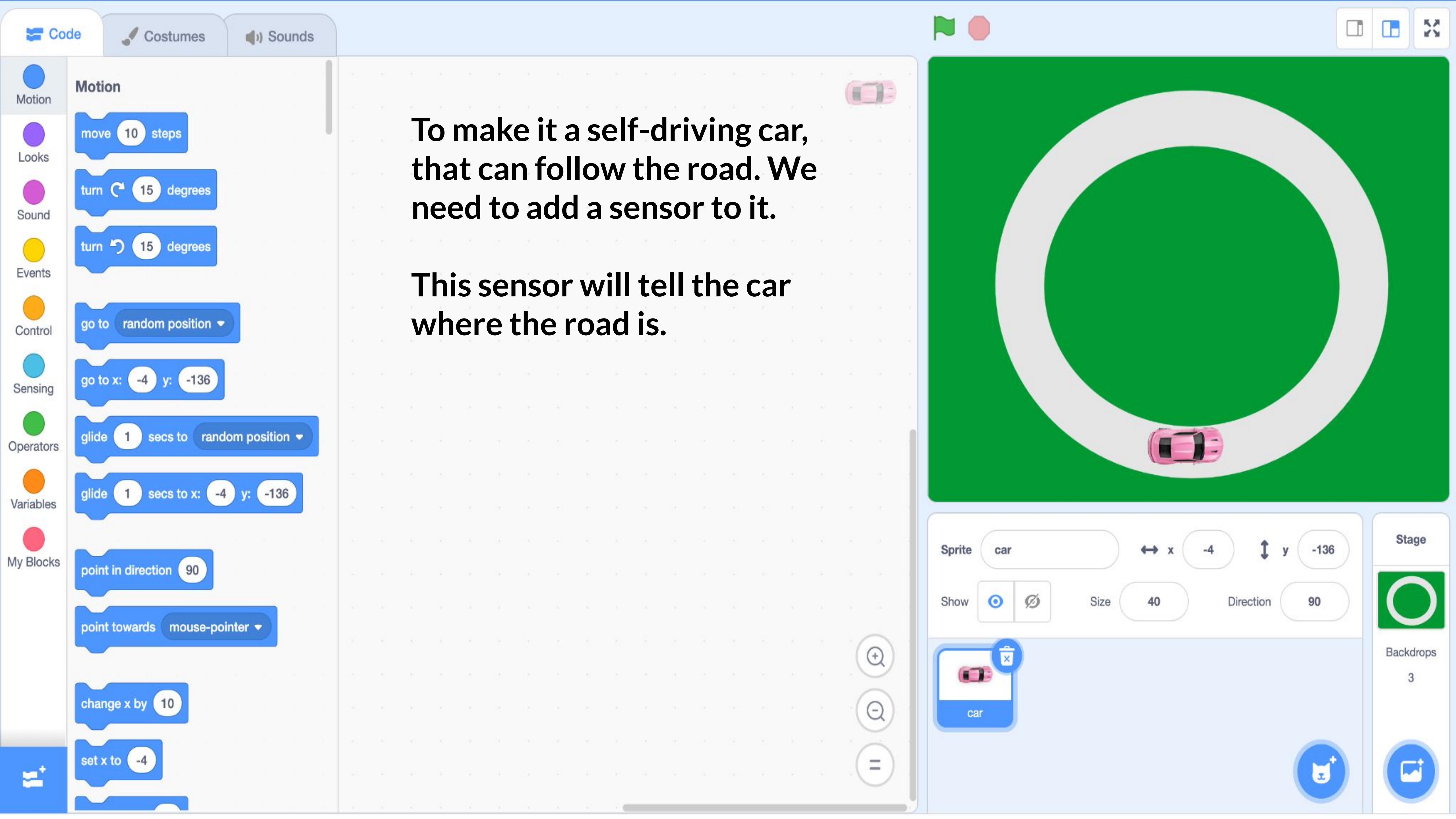
Move 1 step, forever.



```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
```

Try it out! Run your code, by clicking on the green flag.

Your car probably won't stop driving…

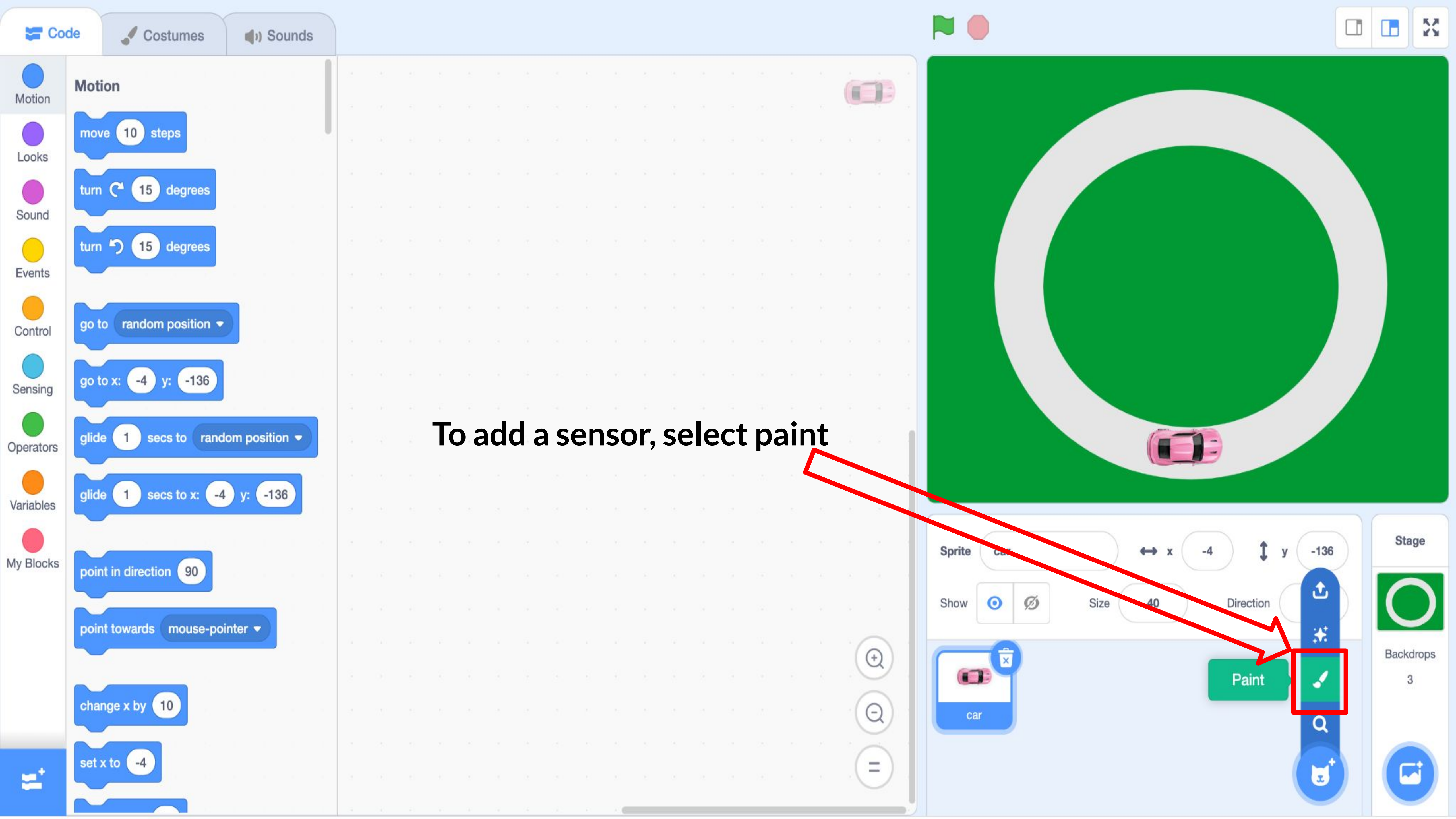


Sprite car x -4 y -136

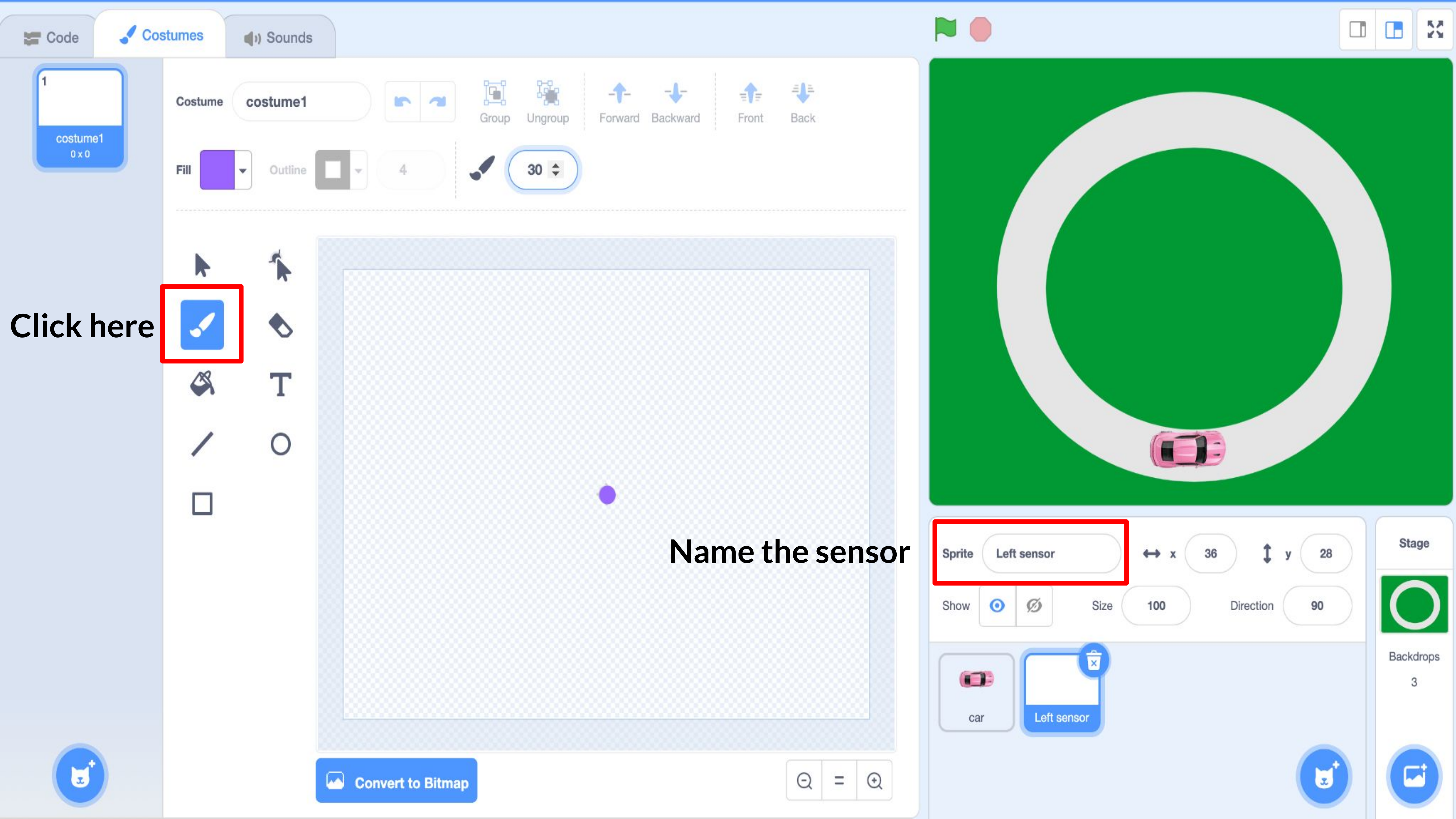
Show Size 40 Direction 90

To make it a self-driving car, that can follow the road. We need to add a sensor to it.

This sensor will tell the car where the road is.



To add a sensor, select paint



Click here

Name the sensor

Costume: costume1

Brush size: 30

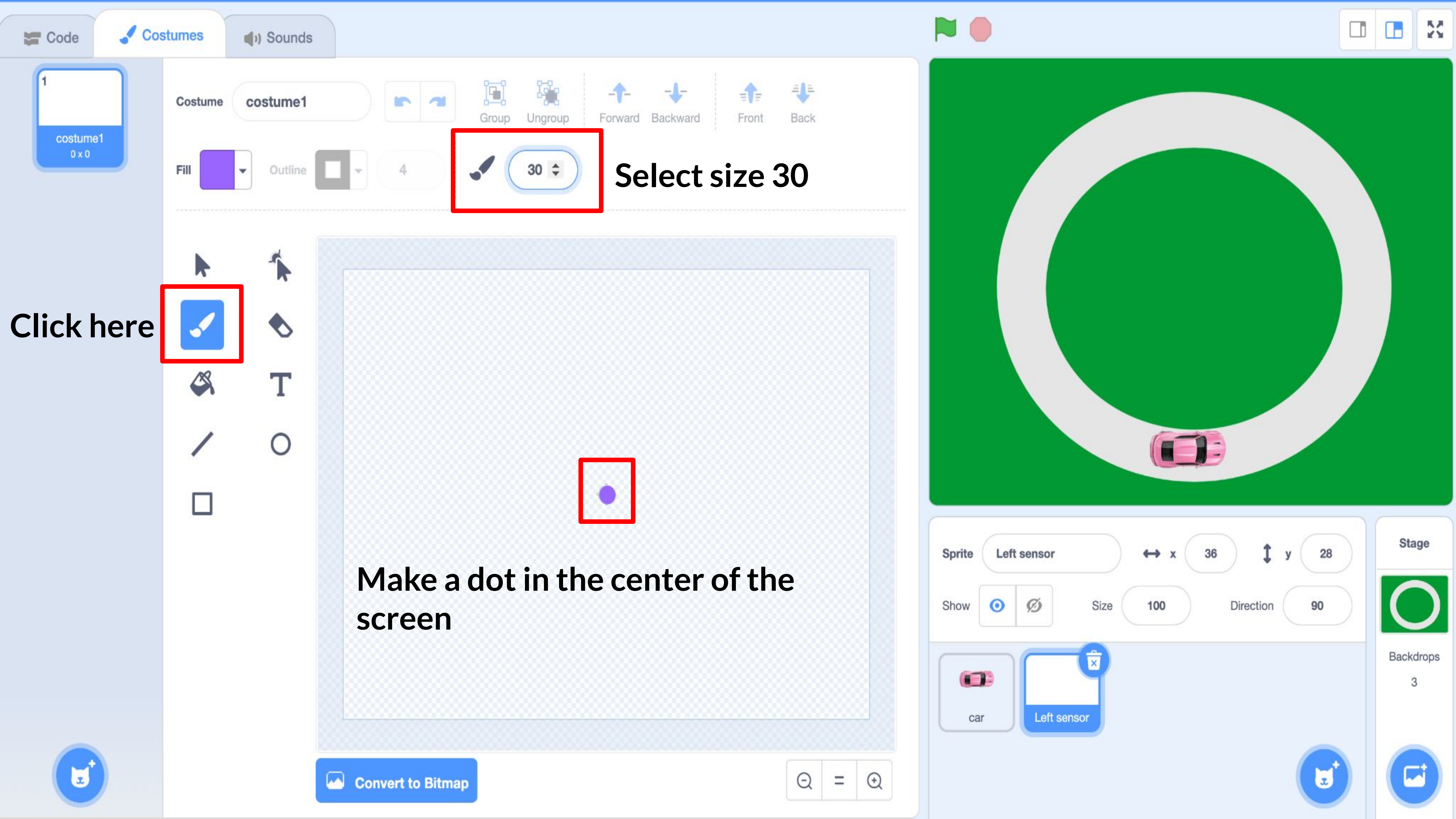
Sprite: Left sensor

x: 36

y: 28

Size: 100

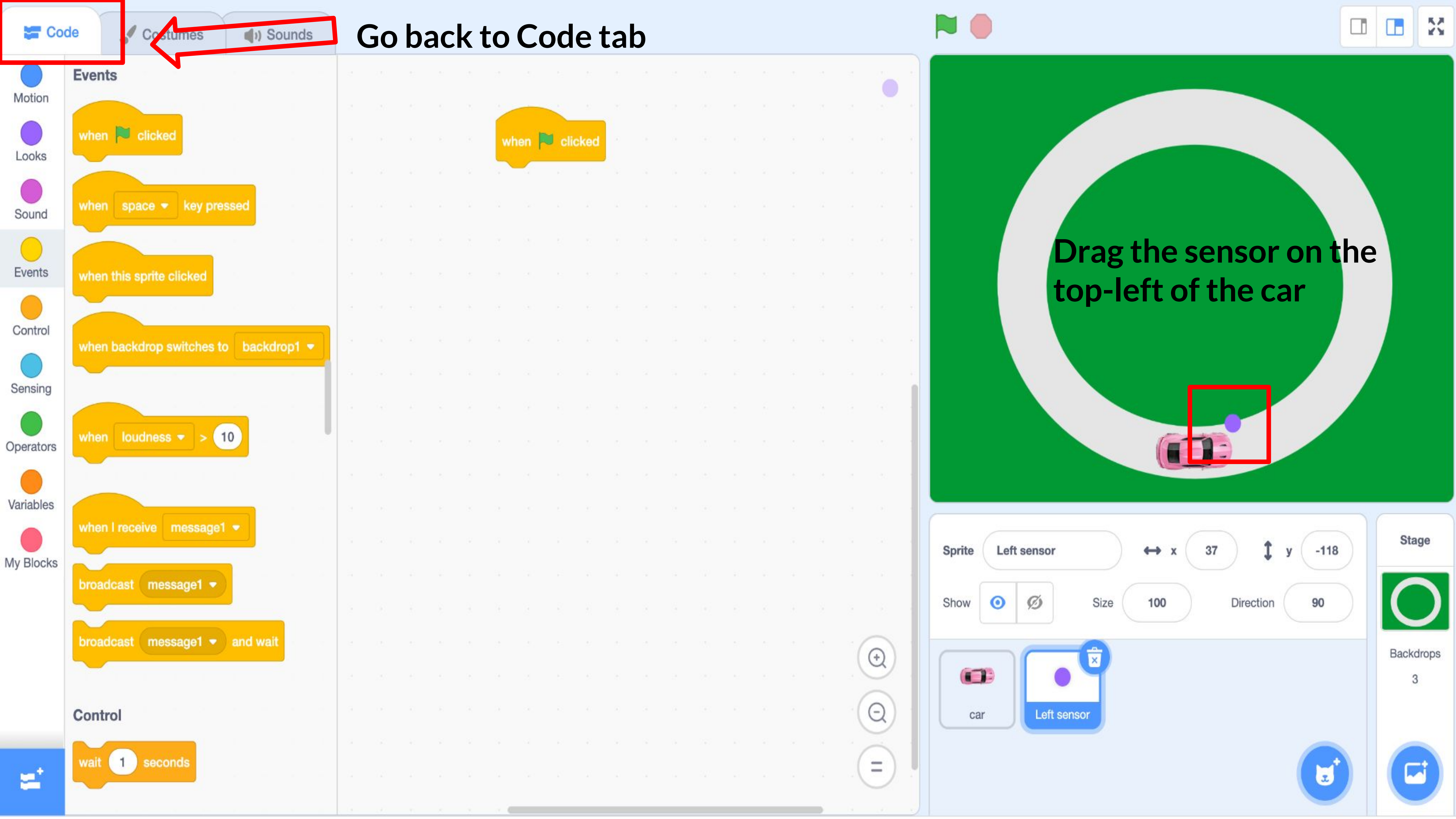
Direction: 90



Select size 30

Click here

Make a dot in the center of the screen



Go back to Code tab

Drag the sensor on the top-left of the car

Sprite: Left sensor

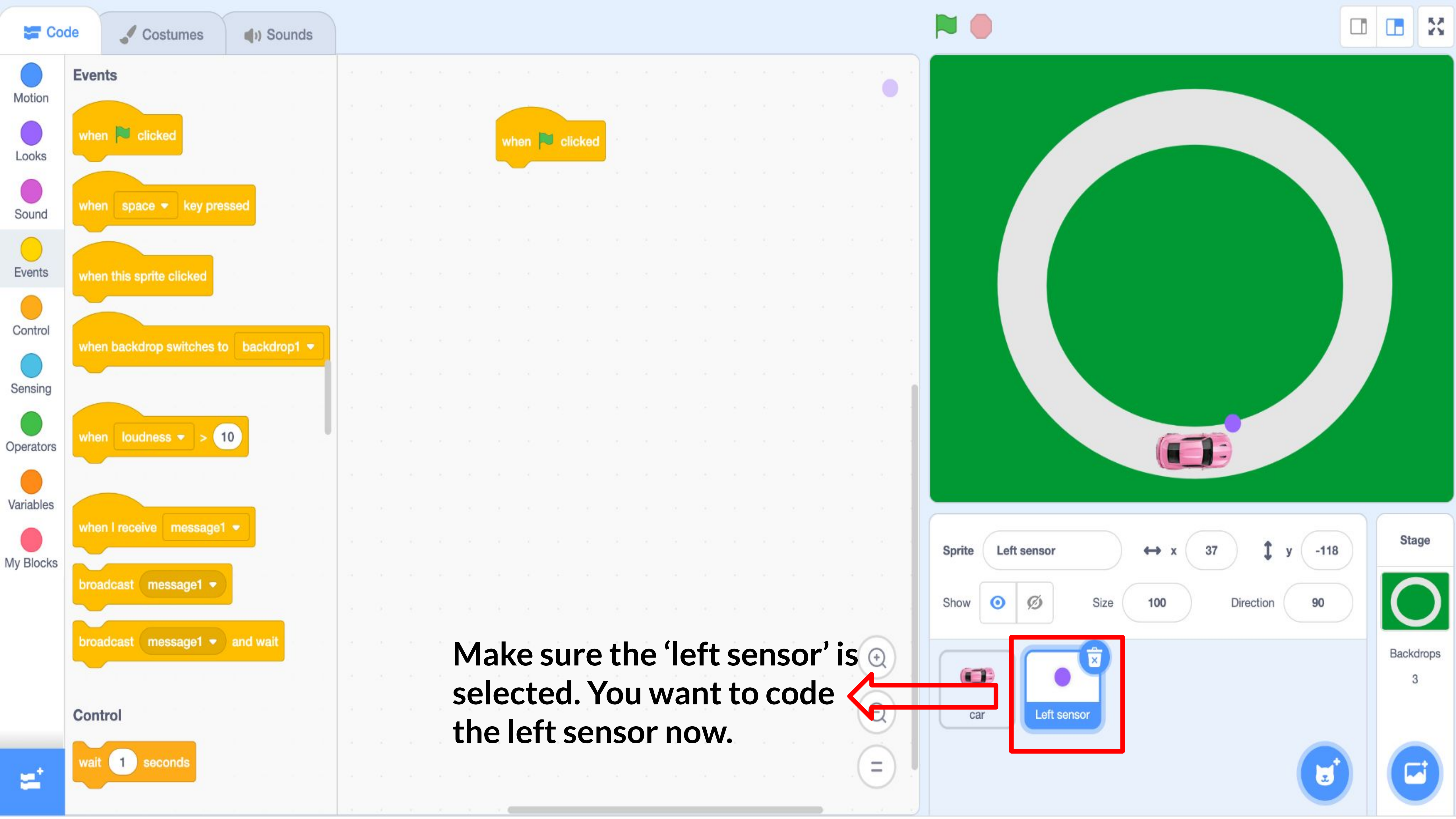
x: 37

y: -118

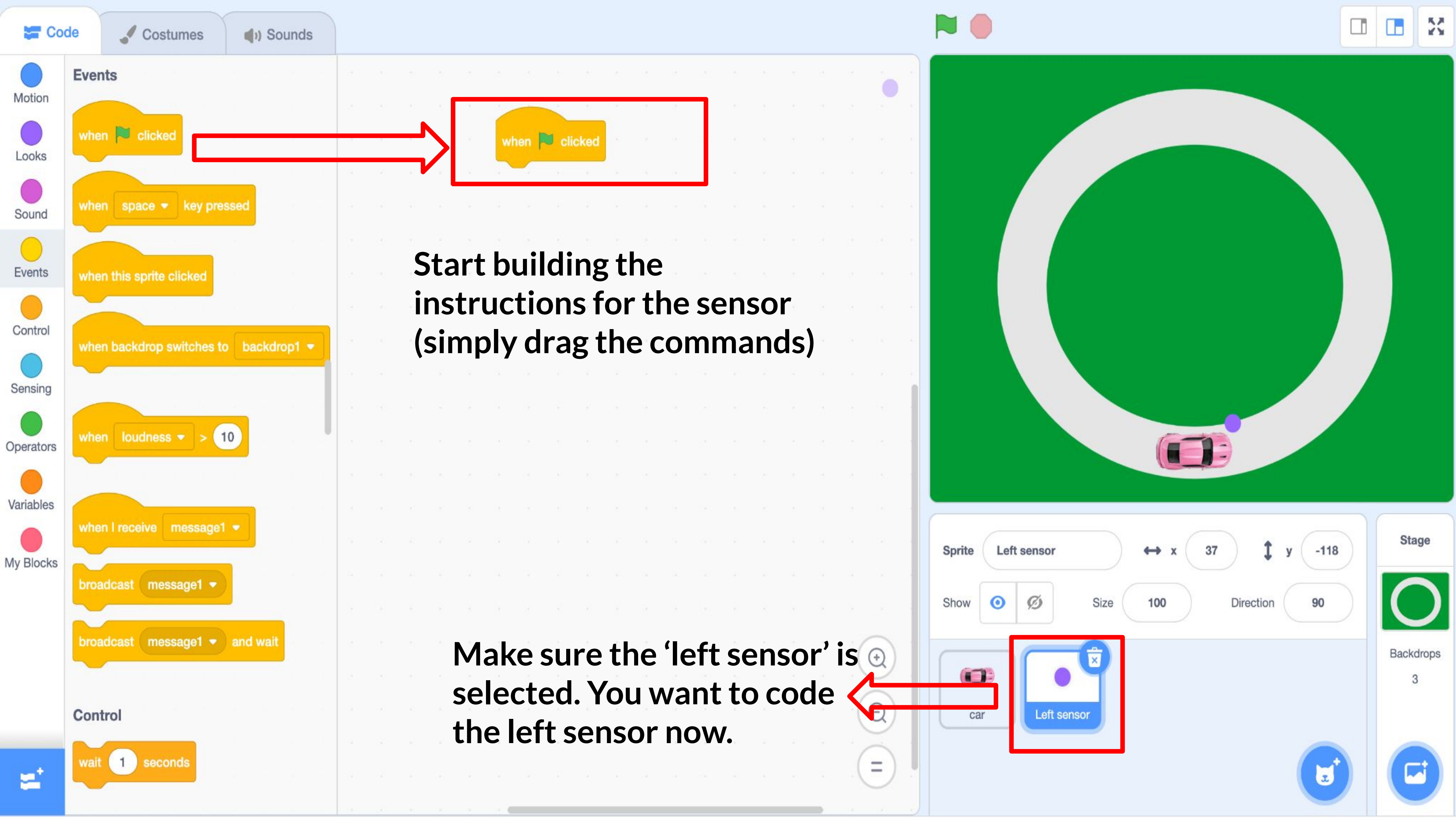
Show

Size: 100

Direction: 90

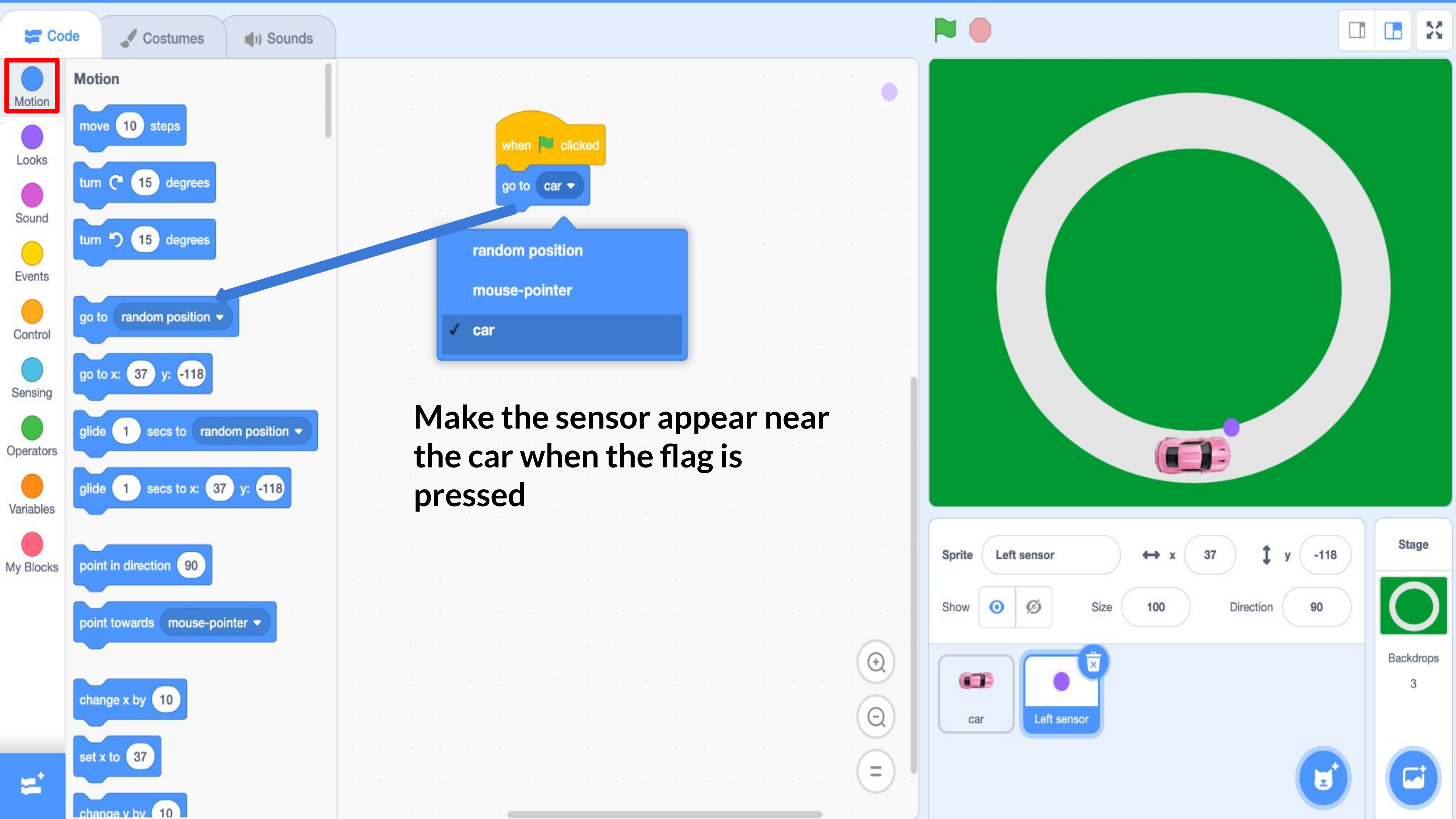


Make sure the ‘left sensor’ is selected. You want to code the left sensor now.

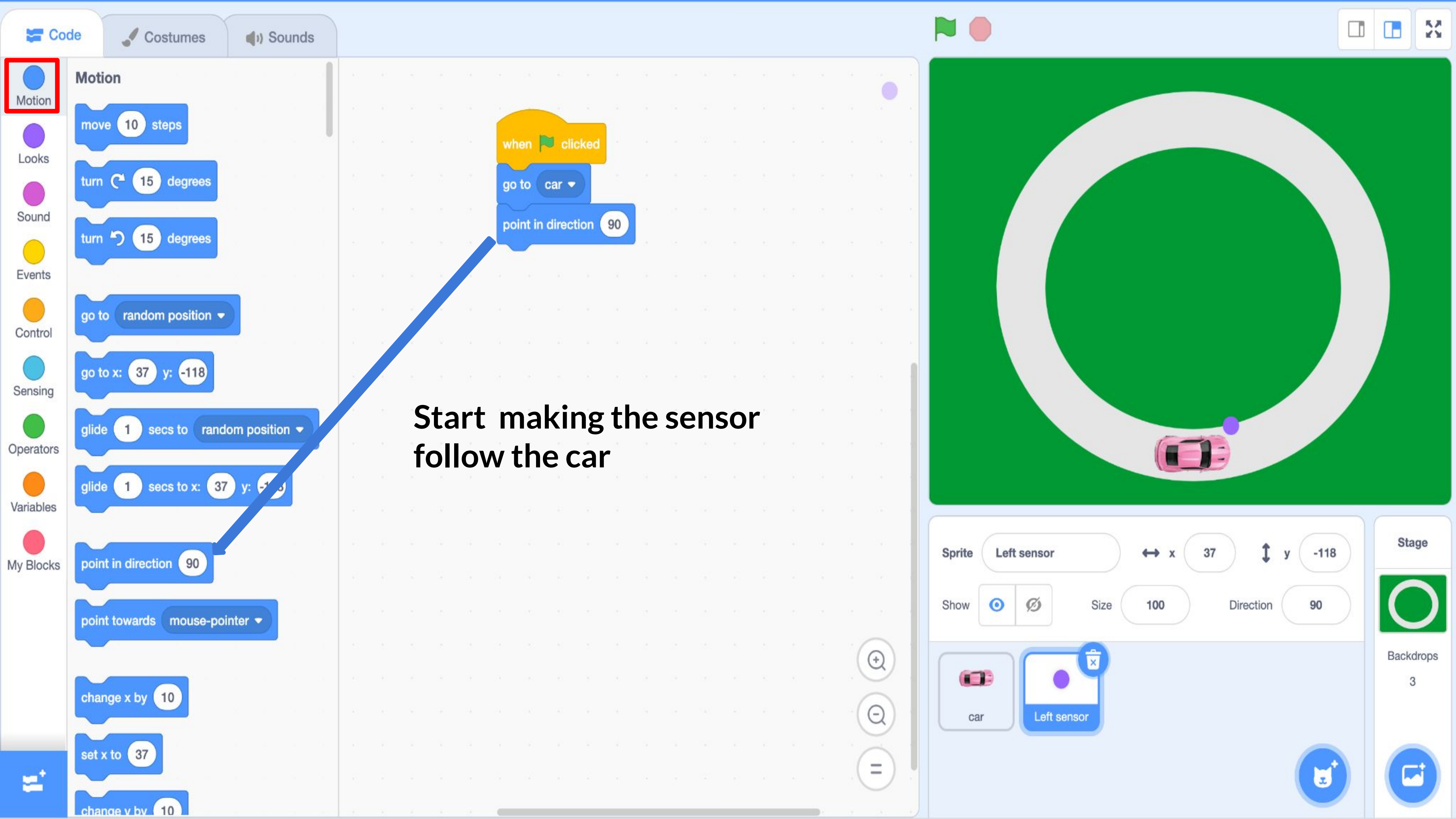


Start building the instructions for the sensor (simply drag the commands)

Make sure the ‘left sensor’ is selected. You want to code the left sensor now.



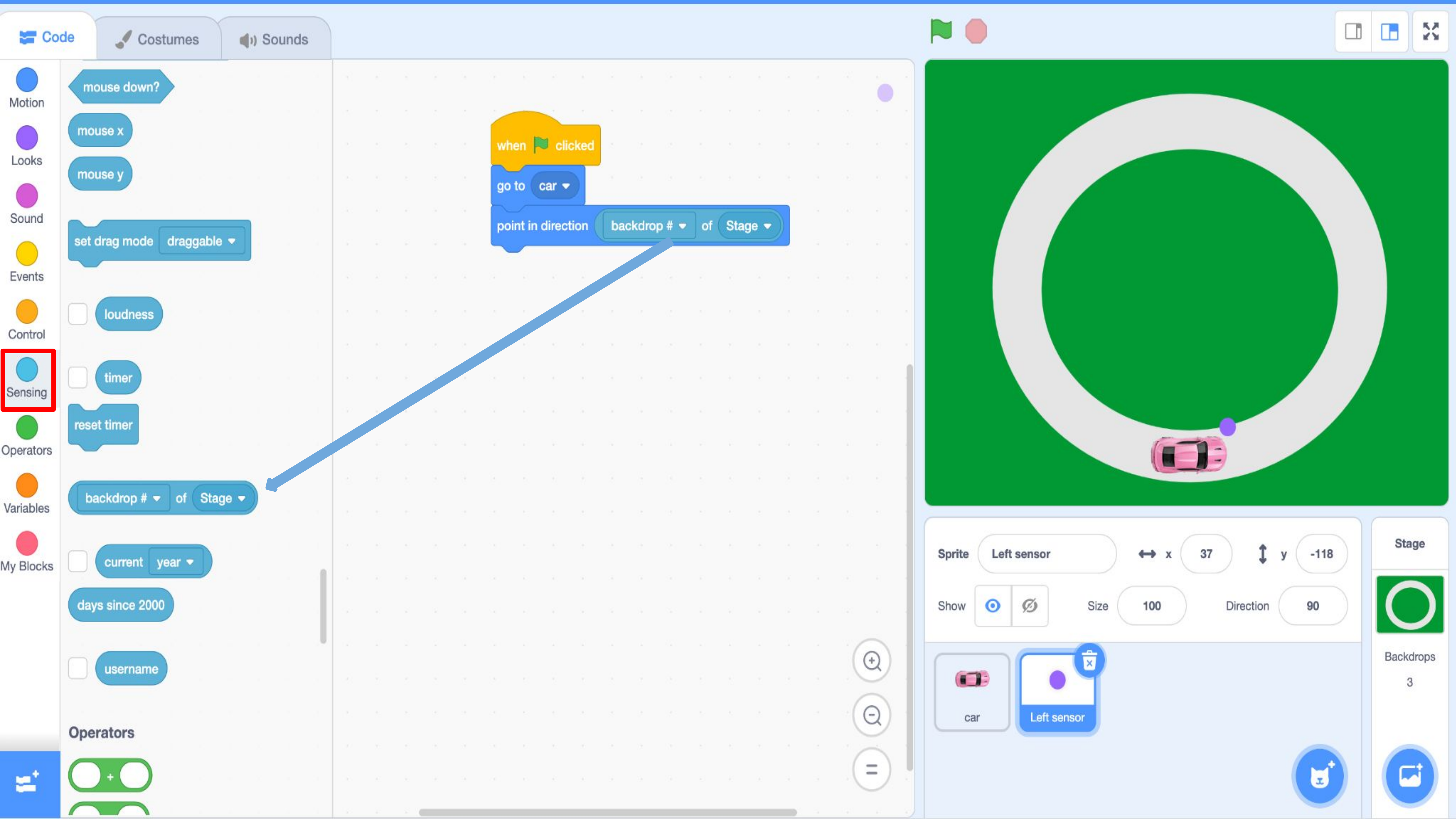
Make the sensor appear near the car when the flag is pressed



```
when 🏳 clicked
go to car ▾
point in direction 90
```

Sprite: Left sensor, x: 37, y: -118, Show, Size: 100, Direction: 90

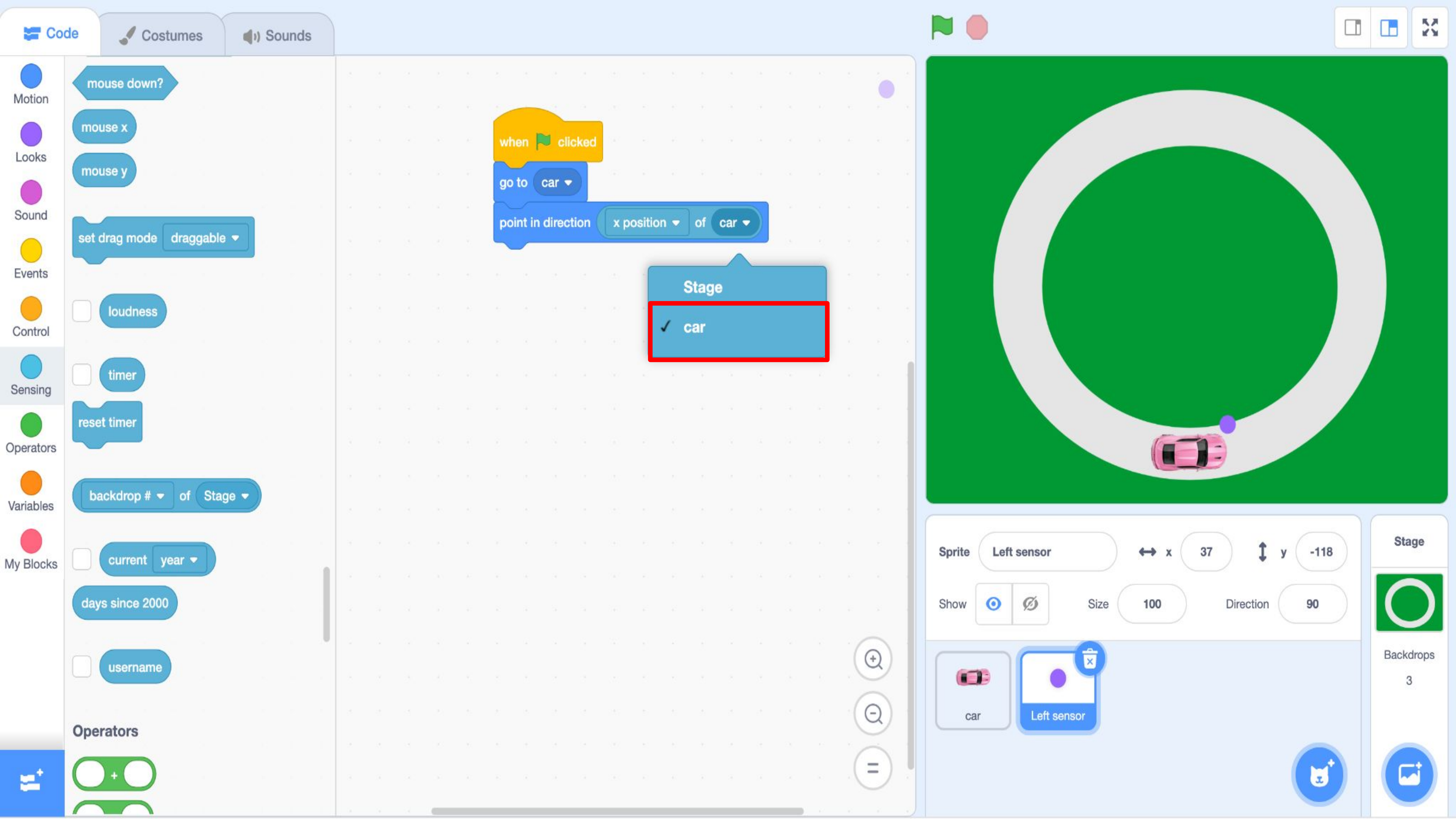
Start making the sensor follow the car



```
when ⚑ clicked
go to (car ▾)
point in direction ([backdrop # ▾] of (Stage ▾))
```

Sprite: Left sensor x: 37 y: -118

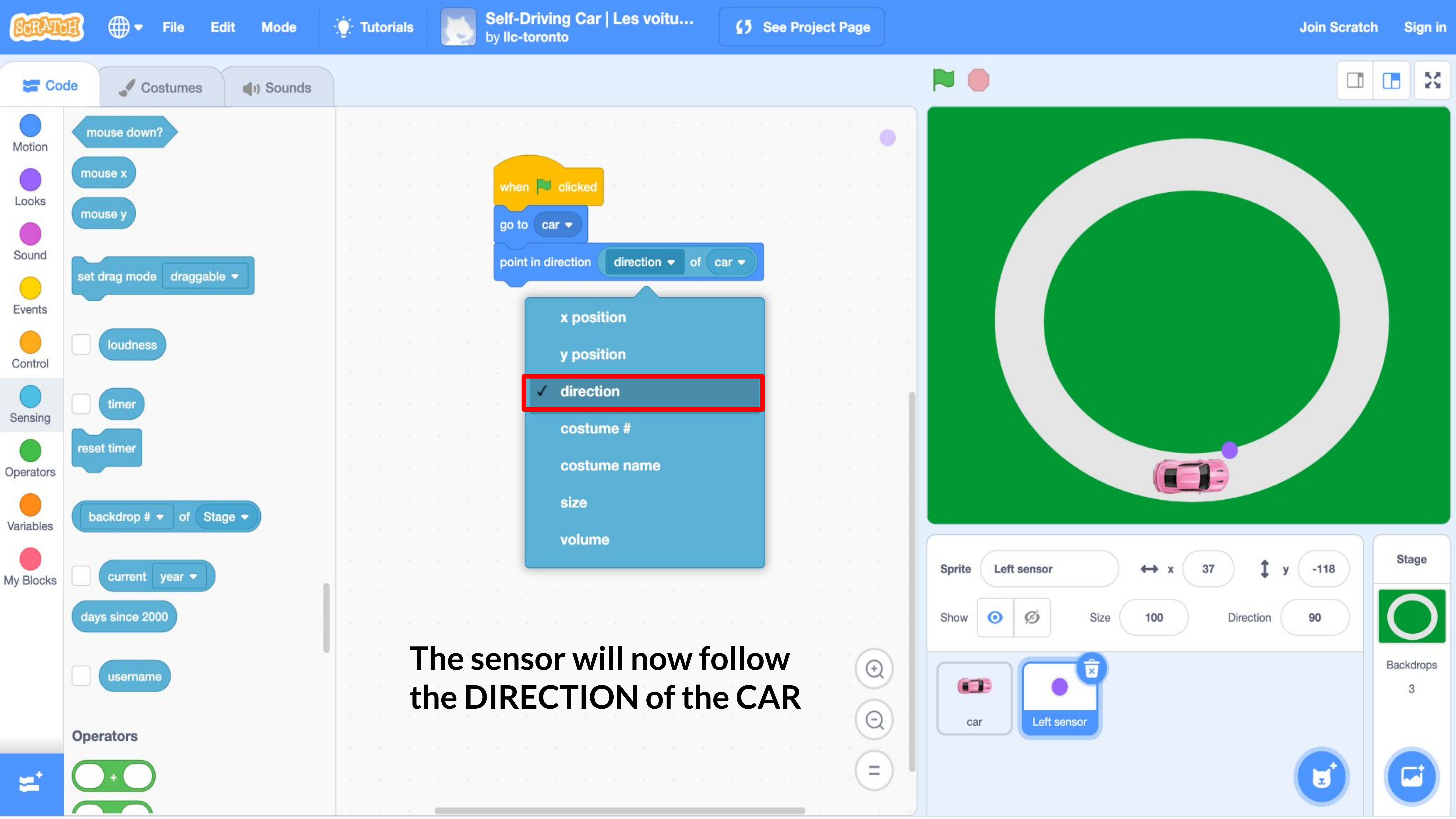
Show Size: 100 Direction: 90



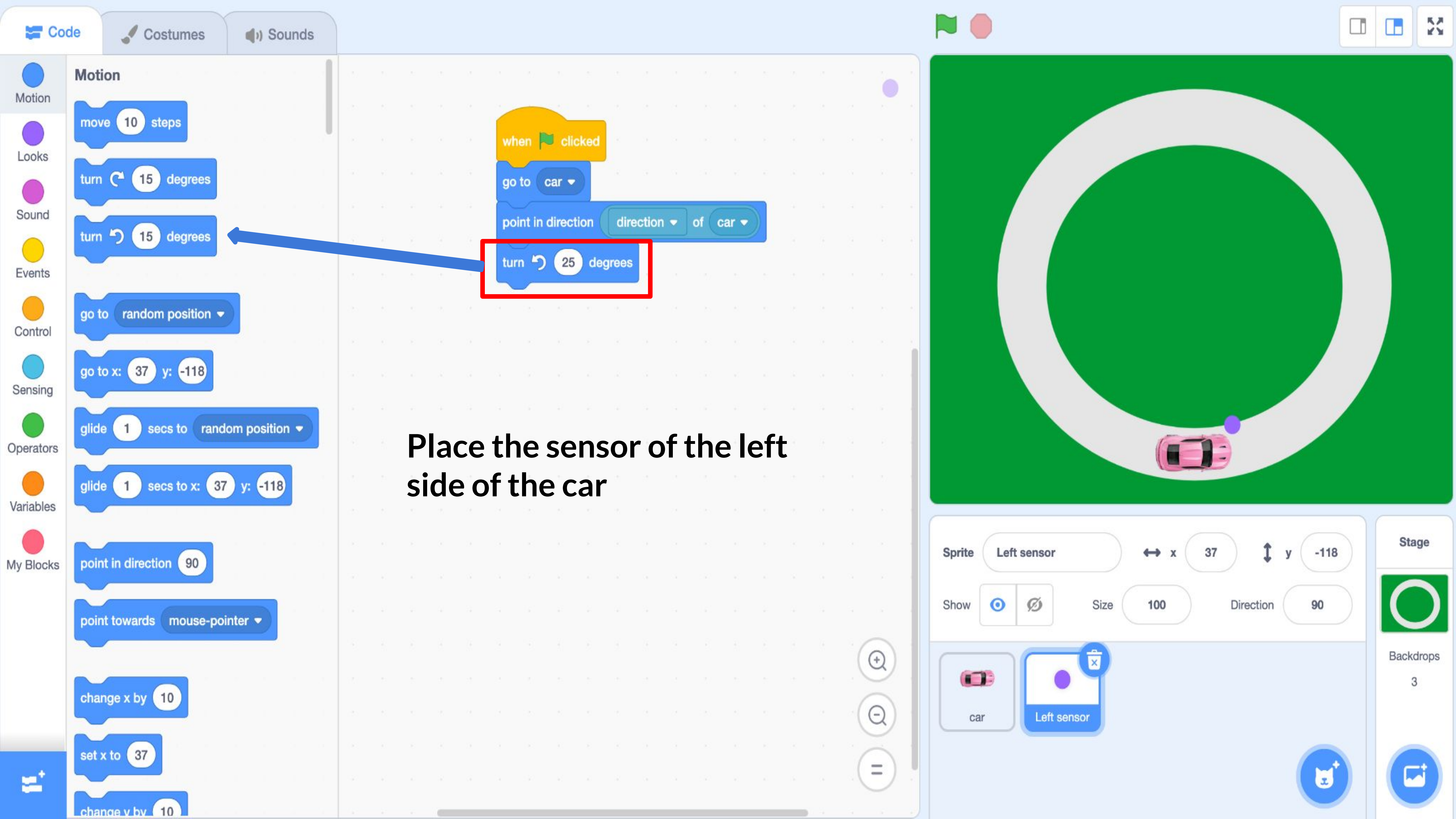
```
when 🏴 clicked
go to (car ▾)
point in direction ((x position ▾) of (car ▾))
```

Stage
✓ car

Sprite: Left sensor x: 37 y: -118
Show Size: 100 Direction: 90



The sensor will now follow the DIRECTION of the CAR

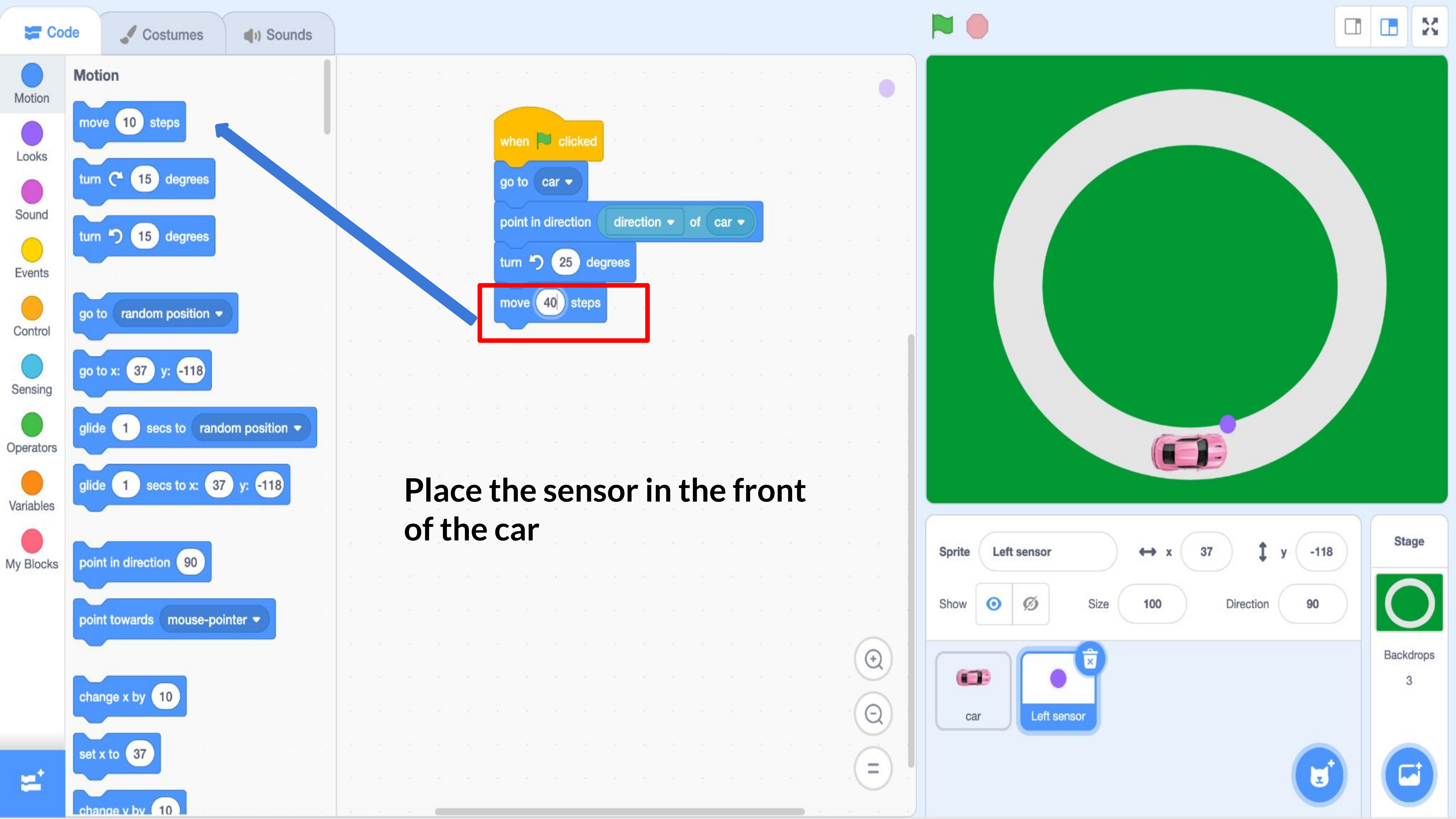


```
when ⚑ clicked
go to (car ▾)
point in direction ((direction ▾) of (car ▾))
turn ↺ (25) degrees
```

Sprite: Left sensor   ↔ x: 37   ↕ y: -118

Show: ● ○   Size: 100   Direction: 90

Place the sensor of the left side of the car

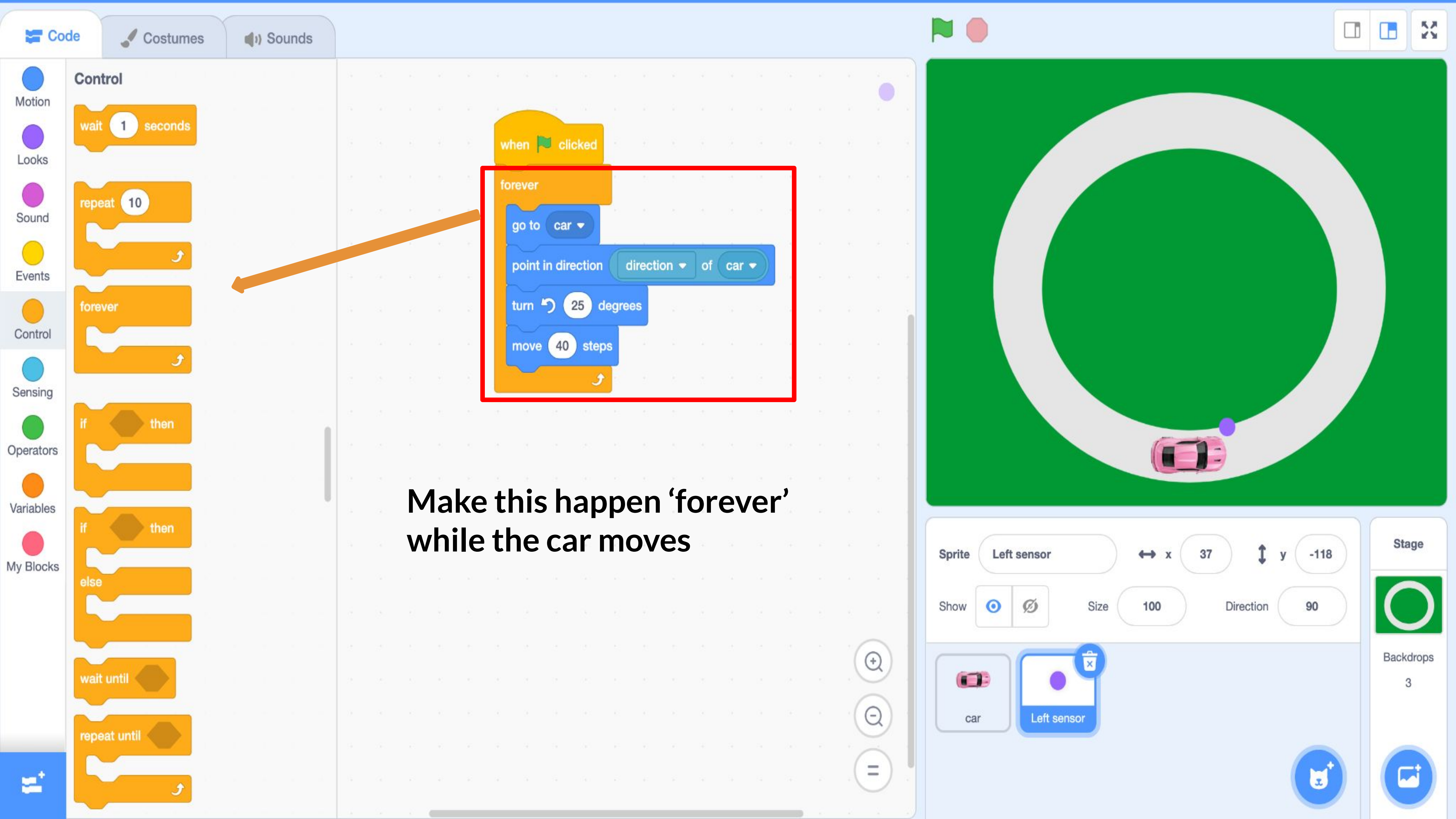


```
when 🏁 clicked
go to car ▾
point in direction (direction ▾ of car ▾)
turn ↺ 25 degrees
move 40 steps
```

Sprite: Left sensor x: 37 y: -118

Show: Size: 100 Direction: 90

Place the sensor in the front of the car

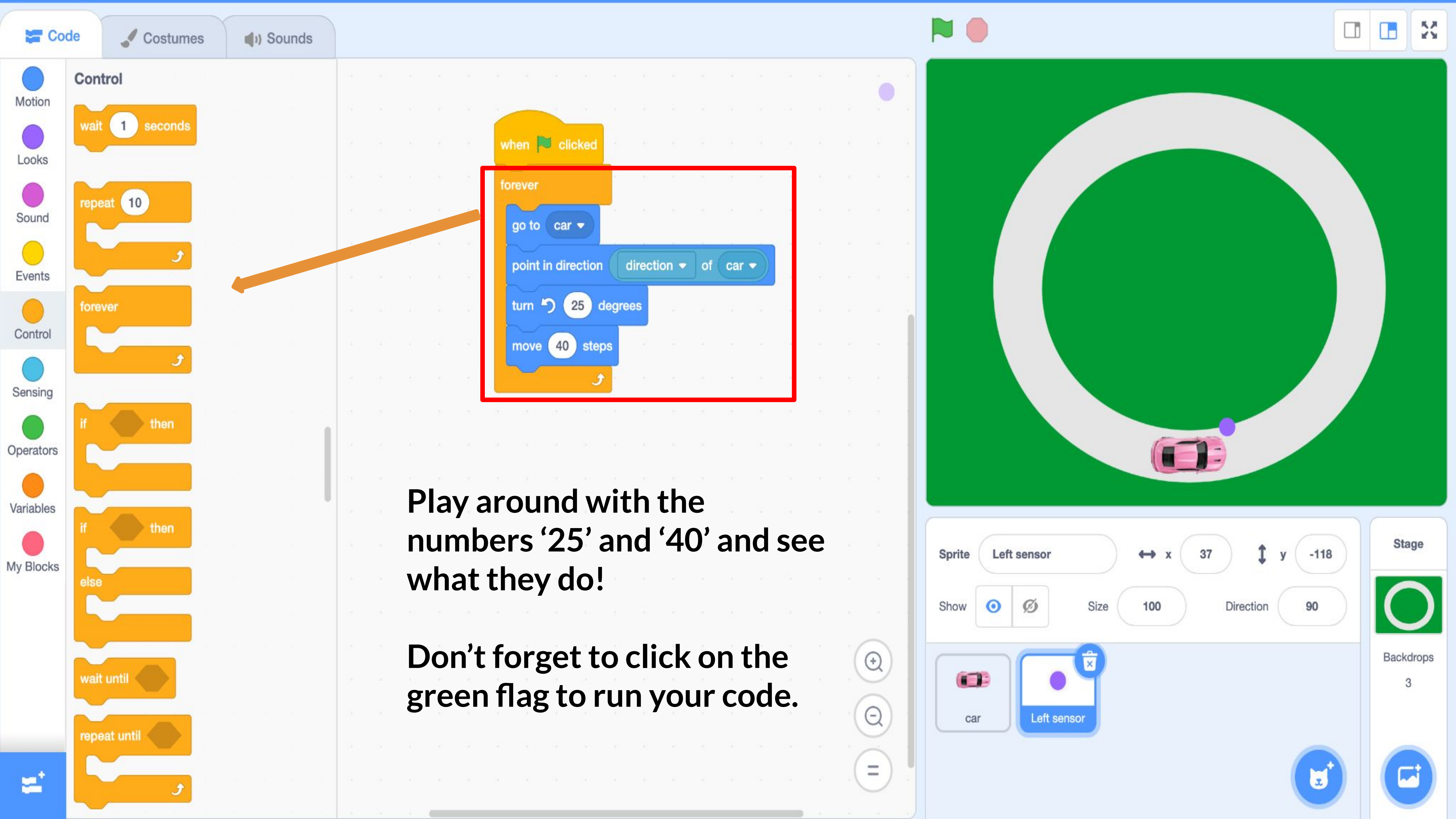


```
when ⚑ clicked
forever
    go to car
    point in direction (direction of car)
    turn ↺ 25 degrees
    move 40 steps
```

Sprite: Left sensor x: 37 y: -118

Show: ● ○ Size: 100 Direction: 90

Make this happen ‘forever’ while the car moves



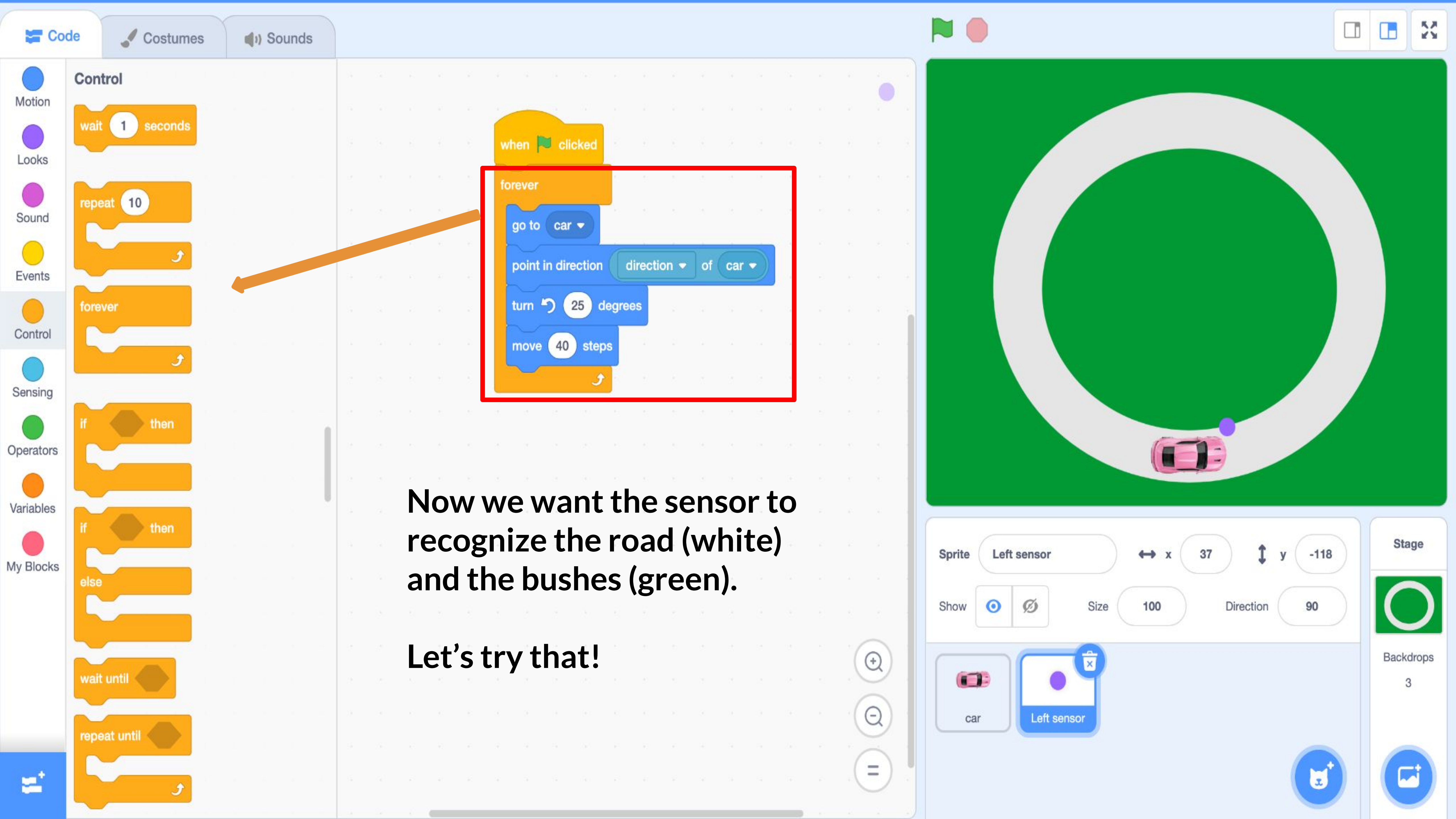
```
when 🏴 clicked
forever
    go to (car ▾)
    point in direction ((direction ▾) of (car ▾))
    turn ↺ (25) degrees
    move (40) steps
```

Sprite: Left sensor ↔ x 37 ↕ y -118

Show: ● ⊘ Size 100 Direction 90

Play around with the numbers ‘25’ and ‘40’ and see what they do!

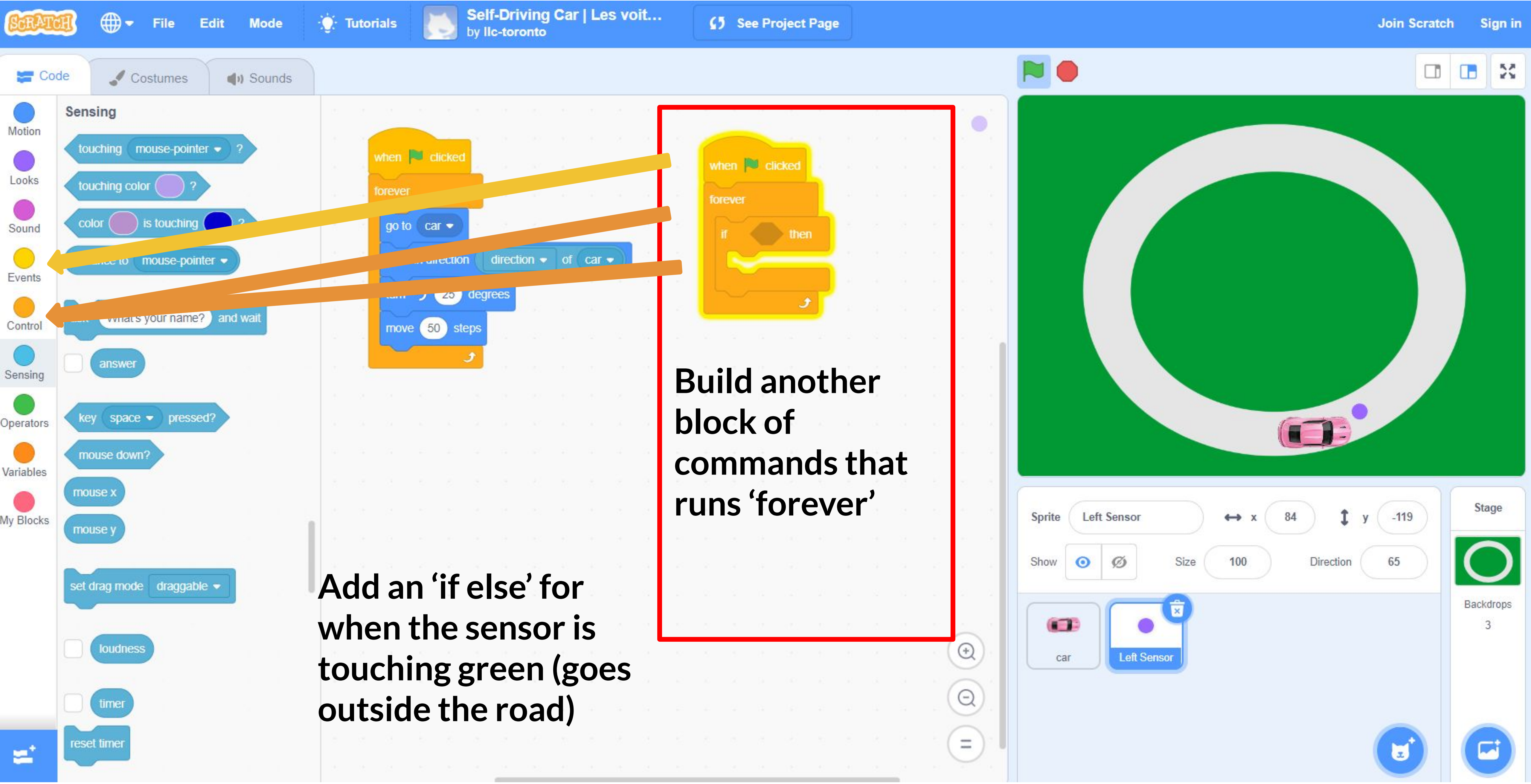
Don’t forget to click on the green flag to run your code.



```
when ⚑ clicked
forever
    go to car ▾
    point in direction (direction ▾ of car ▾)
    turn ↺ 25 degrees
    move 40 steps
```

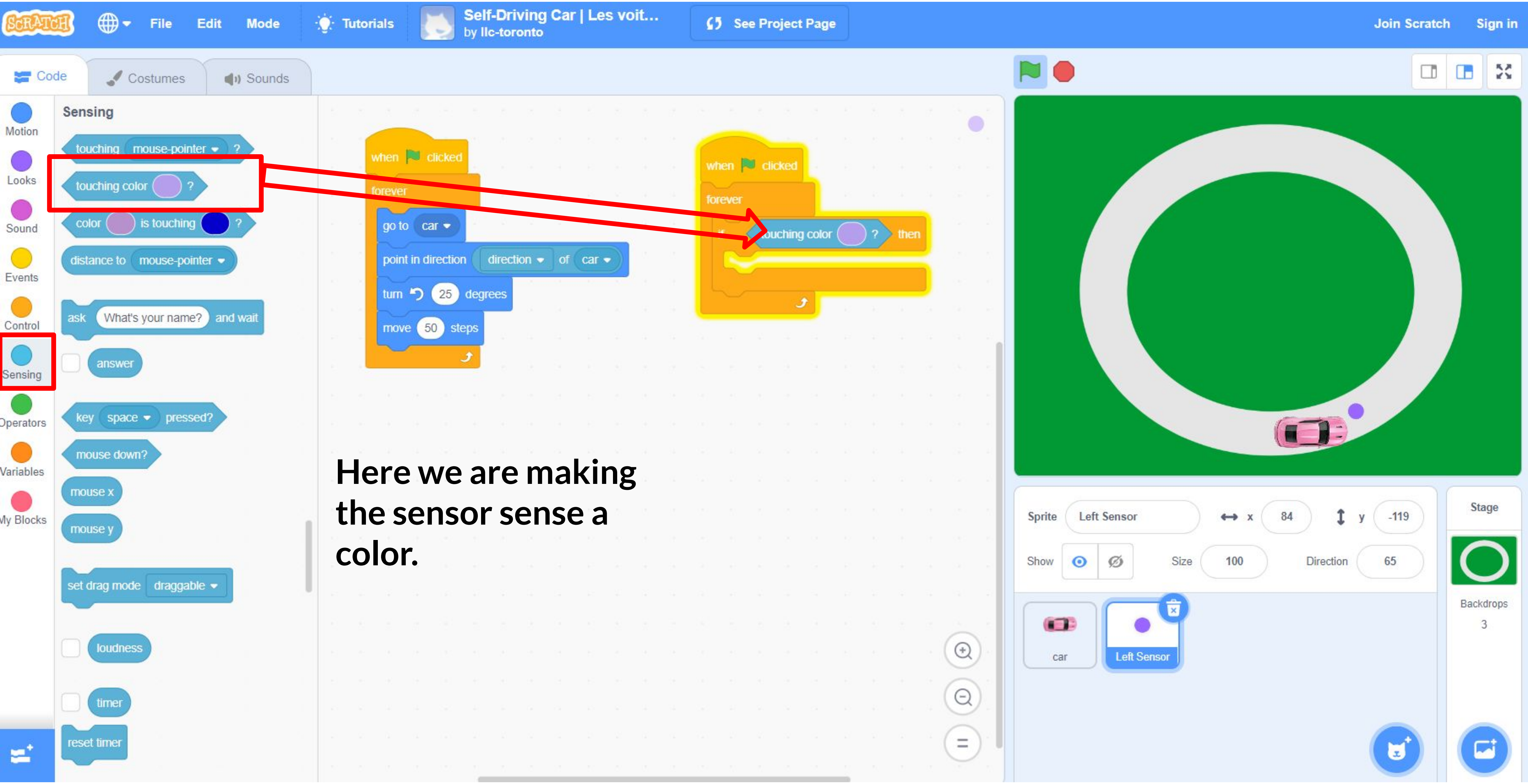
Now we want the sensor to recognize the road (white) and the bushes (green).

Let’s try that!

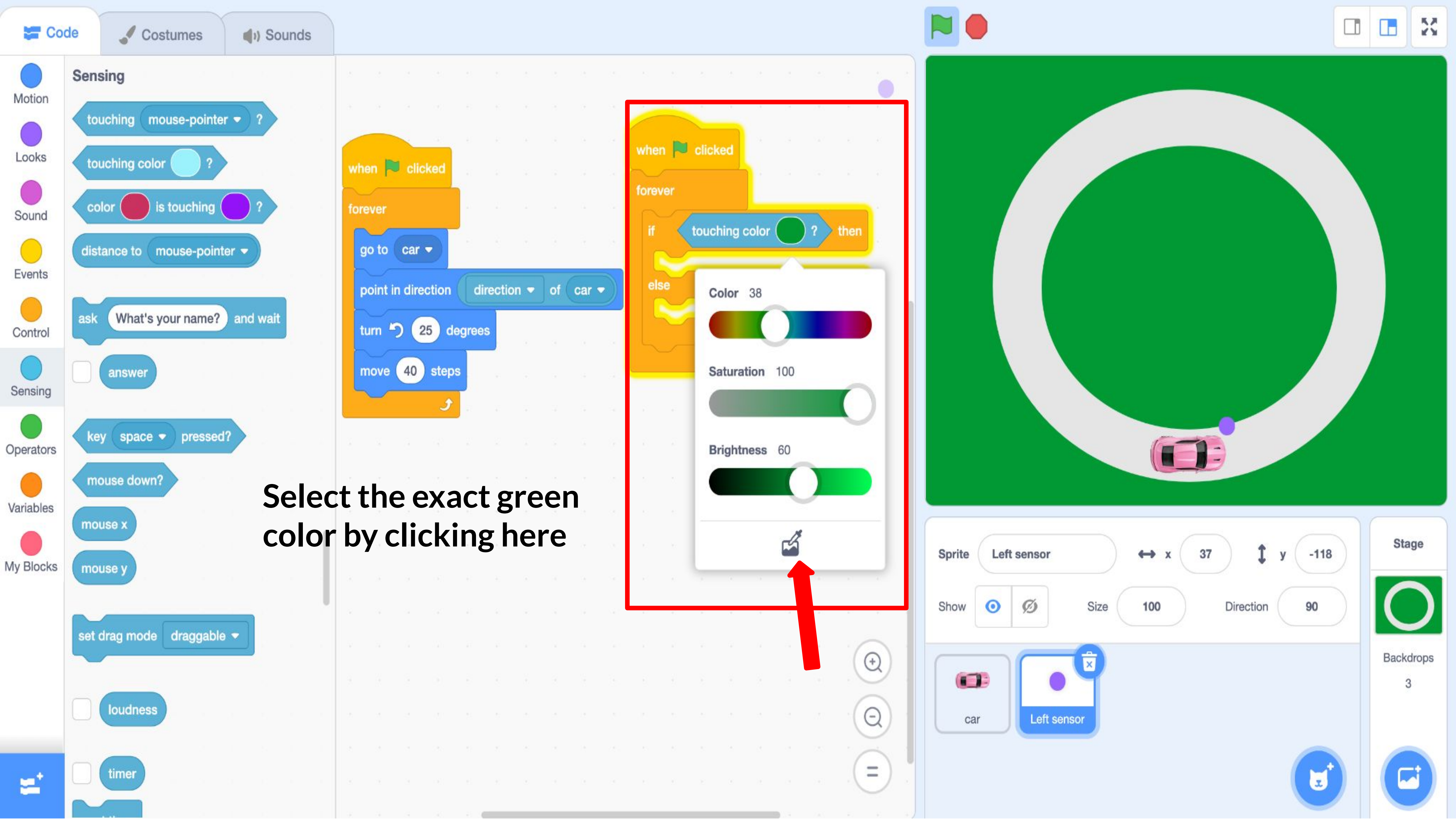


Build another block of commands that runs 'forever'

Add an 'if else' for when the sensor is touching green (goes outside the road)



Here we are making the sensor sense a color.



Select the exact green color by clicking here

```
when 🏁 clicked
forever
    go to car
    point in direction (direction of car)
    turn ⟲ 25 degrees
    move 40 steps
```

```
when 🏁 clicked
forever
    if <touching color 🟩 ?> then
    else
```

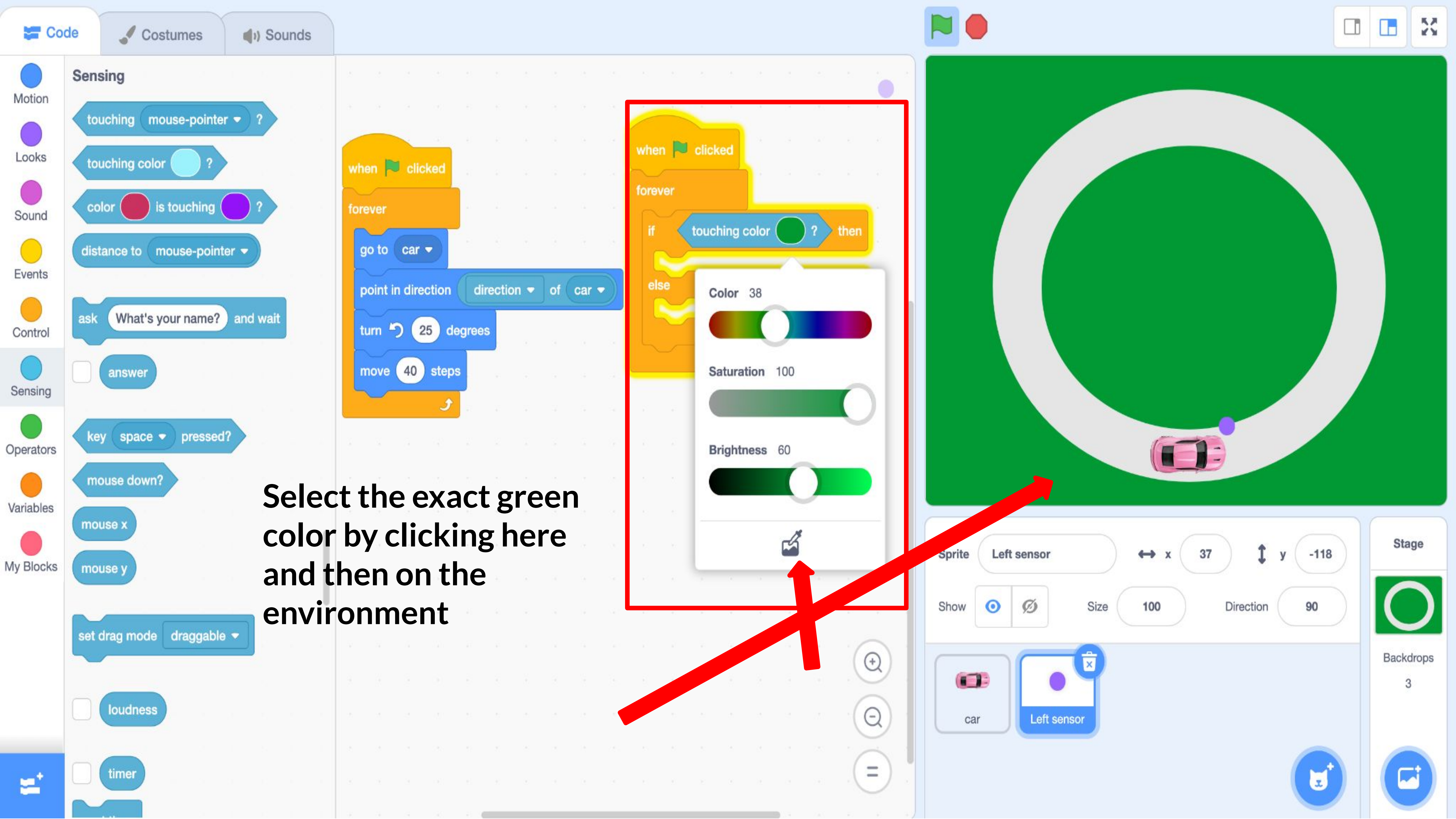
Color 38

Saturation 100

Brightness 60

Sprite Left sensor ↔ x 37 ↕ y -118

Show Size 100 Direction 90



Select the exact green color by clicking here and then on the environment

```
when ⚑ clicked
forever
    go to car ▾
    point in direction direction ▾ of car ▾
    turn ↺ 25 degrees
    move 40 steps
```

```
when ⚑ clicked
forever
    if touching color ● ? then

    else

```

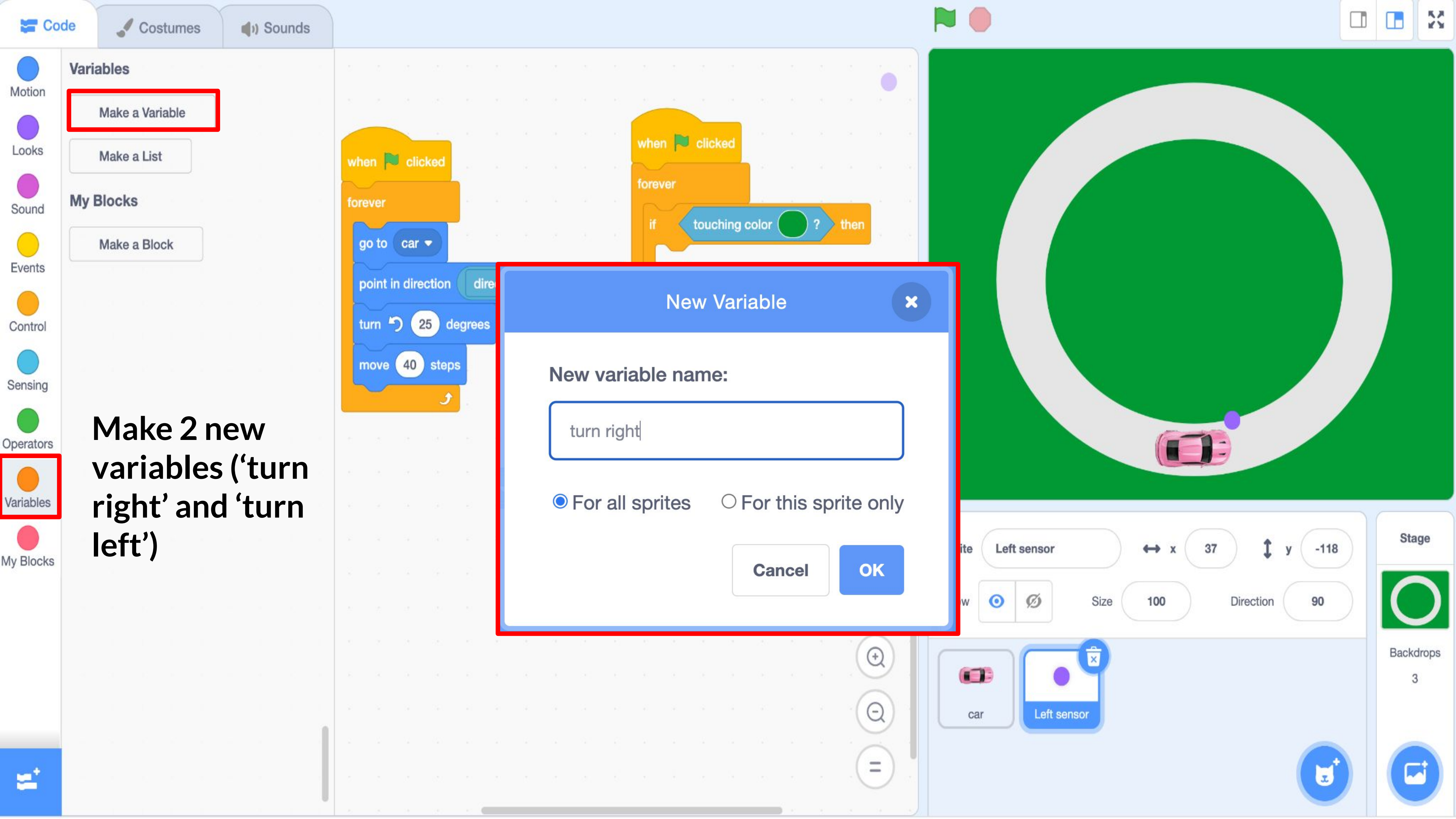
Color 38

Saturation 100

Brightness 60

Sprite Left sensor ↔ x 37 ↕ y -118

Show Size 100 Direction 90



New Variable

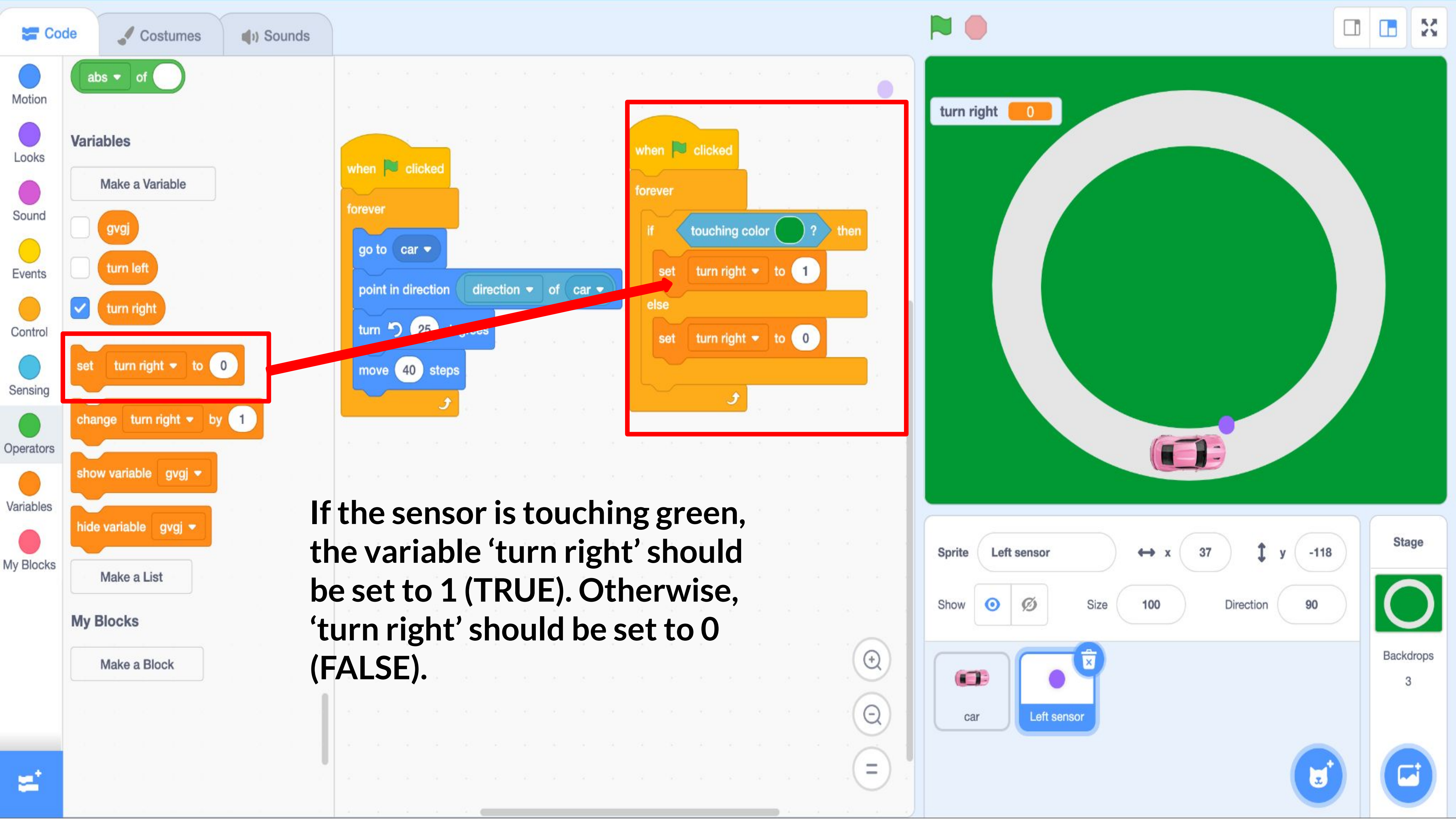
New variable name:

turn right

● For all sprites ○ For this sprite only

Cancel OK

Make 2 new variables ('turn right' and 'turn left')



☐ gvgj
☐ turn left
☑ turn right

```
set turn right to 0
```

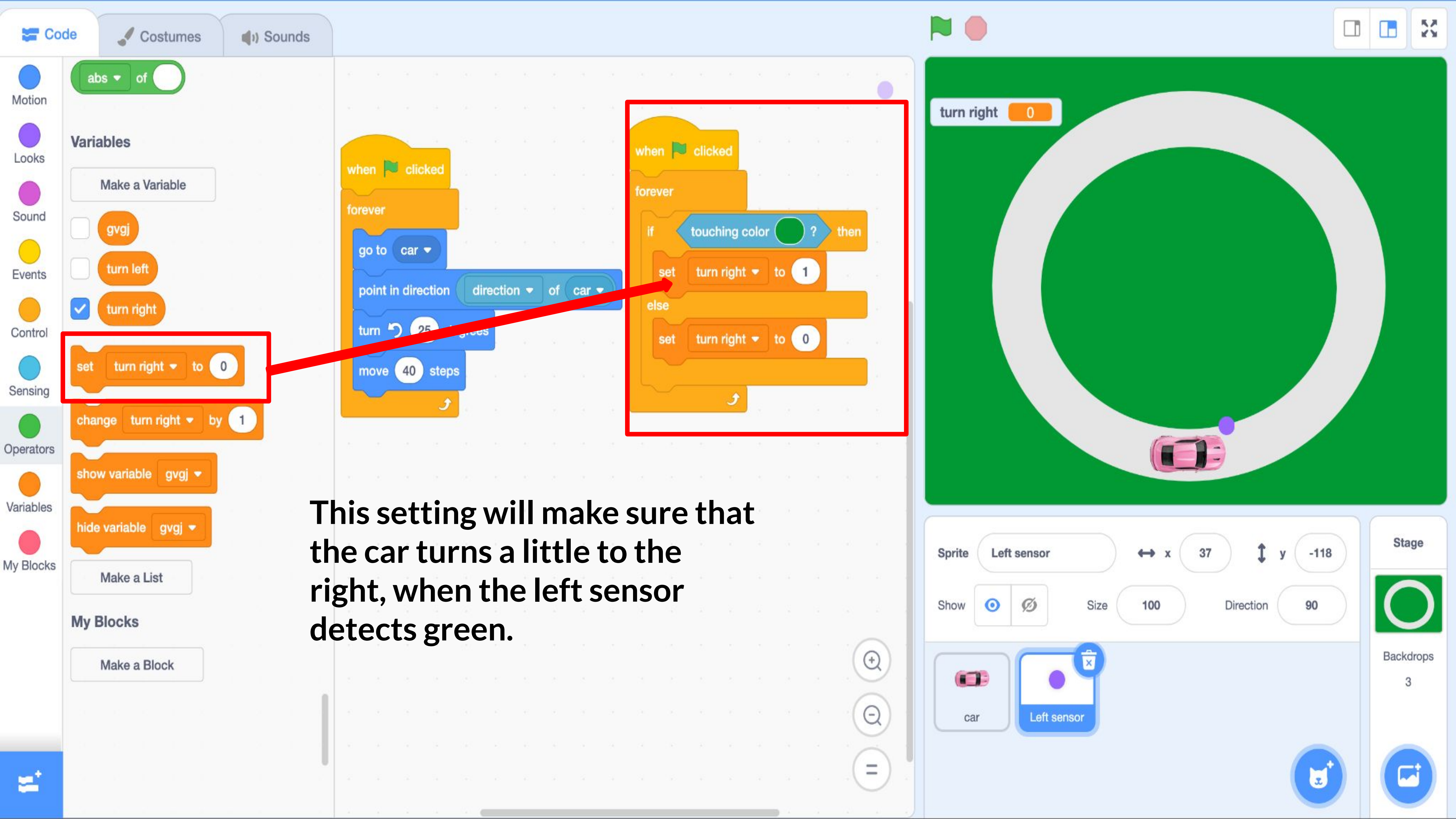
```
when 🏁 clicked
forever
    go to car
    point in direction (direction of car)
    turn ↺ 25 degrees
    move 40 steps
```

```
when 🏁 clicked
forever
    if touching color 🟩 ? then
        set turn right to 1
    else
        set turn right to 0
```

turn right 0

Sprite Left sensor ↔ x 37 ↕ y -118
Show Size 100 Direction 90

If the sensor is touching green, the variable 'turn right' should be set to 1 (TRUE). Otherwise, 'turn right' should be set to 0 (FALSE).



□ gvgj
□ turn left
☑ turn right

```
set turn right to 0
change turn right by 1
show variable gvgj
hide variable gvgj
```

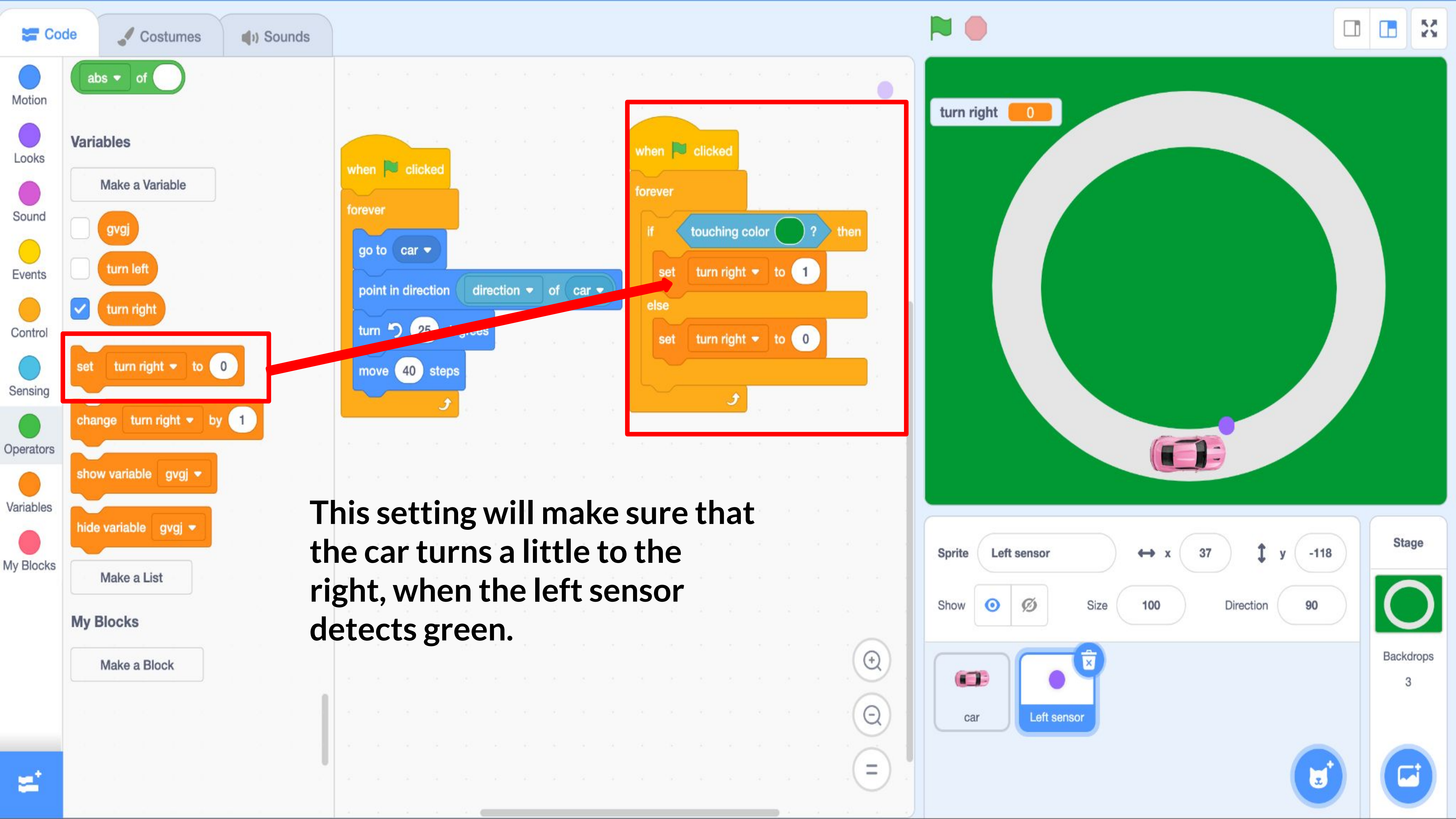
```
when 🏴 clicked
forever
    go to car
    point in direction (direction of car)
    turn ↺ 25 degrees
    move 40 steps
```

```
when 🏴 clicked
forever
    if touching color 🟩 ? then
        set turn right to 1
    else
        set turn right to 0
```

turn right 0

Sprite: Left sensor x: 37 y: -118
Show: Size: 100 Direction: 90

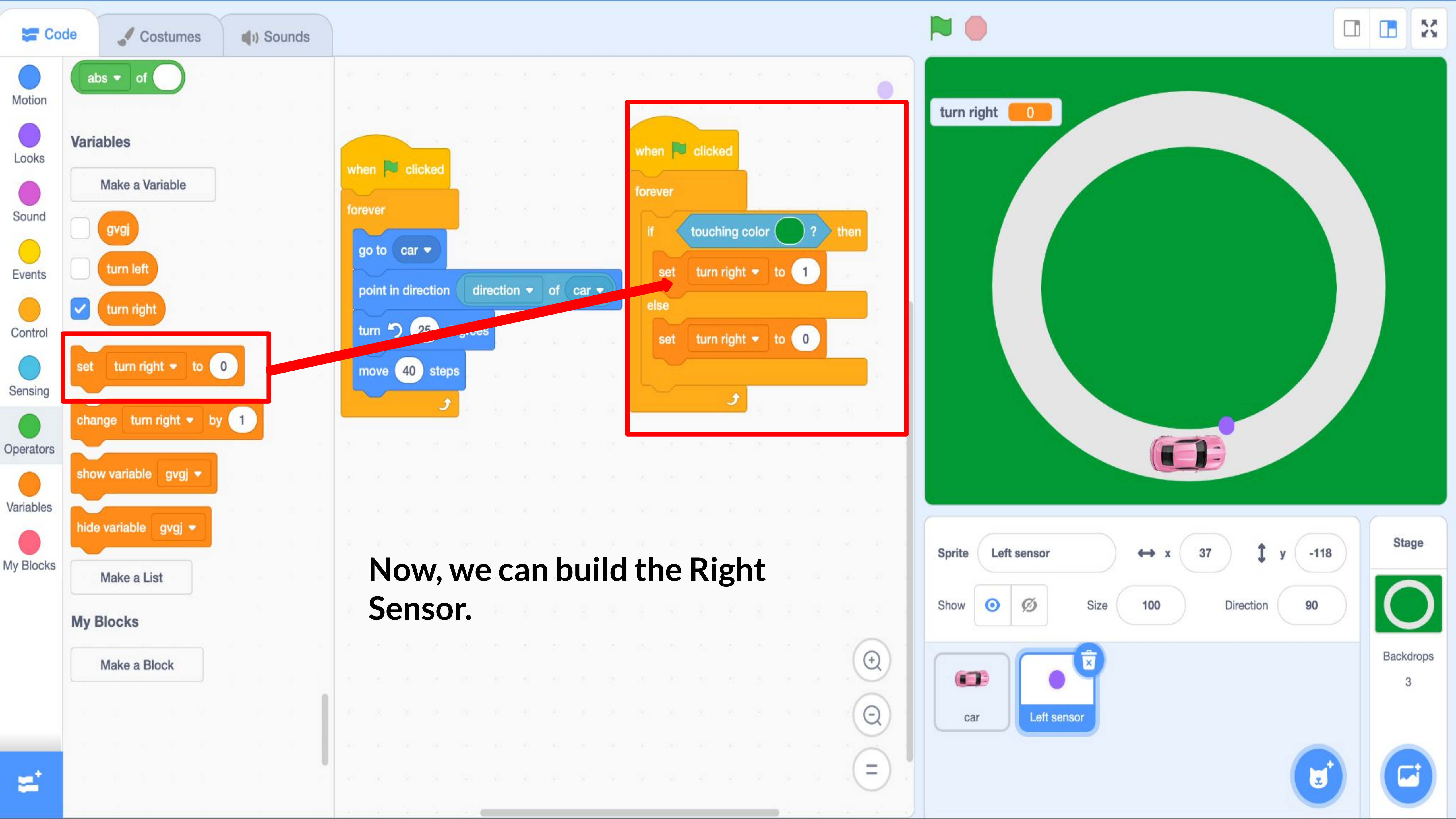
This setting will make sure that the car turns a little to the right, when the left sensor detects green.



```
when 🏁 clicked
forever
    go to car
    point in direction (direction of car)
    turn ↺ 25 degrees
    move 40 steps
```

```
when 🏁 clicked
forever
    if <touching color 🟩 ?> then
        set turn right to 1
    else
        set turn right to 0
```

This setting will make sure that the car turns a little to the right, when the left sensor detects green.

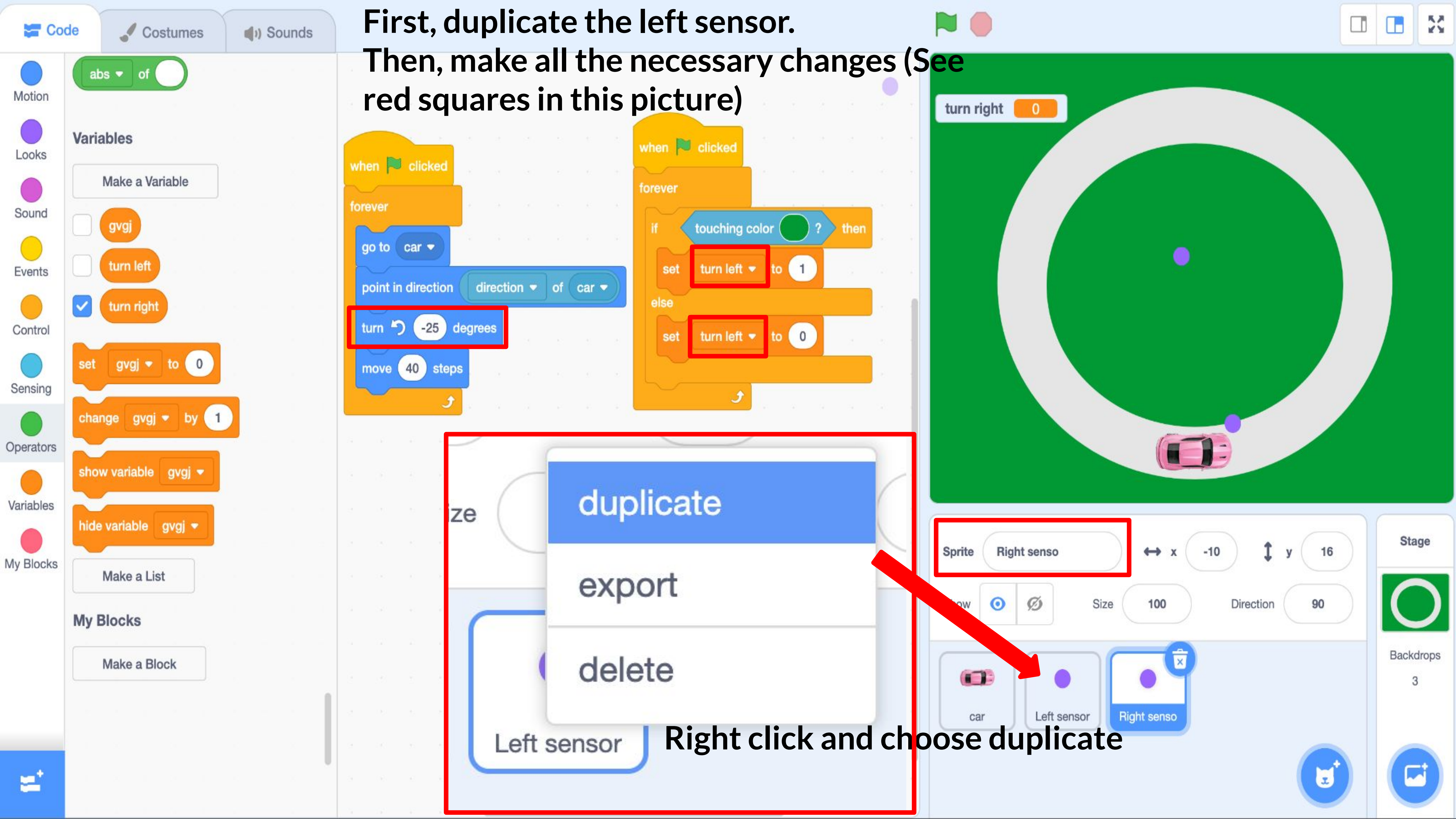


```
when 🏁 clicked
forever
    go to car
    point in direction (direction of car)
    turn ↺ 25 degrees
    move 40 steps
```

```
when 🏁 clicked
forever
    if <touching color ●?> then
        set turn right to 1
    else
        set turn right to 0
```

Now, we can build the Right Sensor.

First, duplicate the left sensor.
Then, make all the necessary changes (See red squares in this picture)

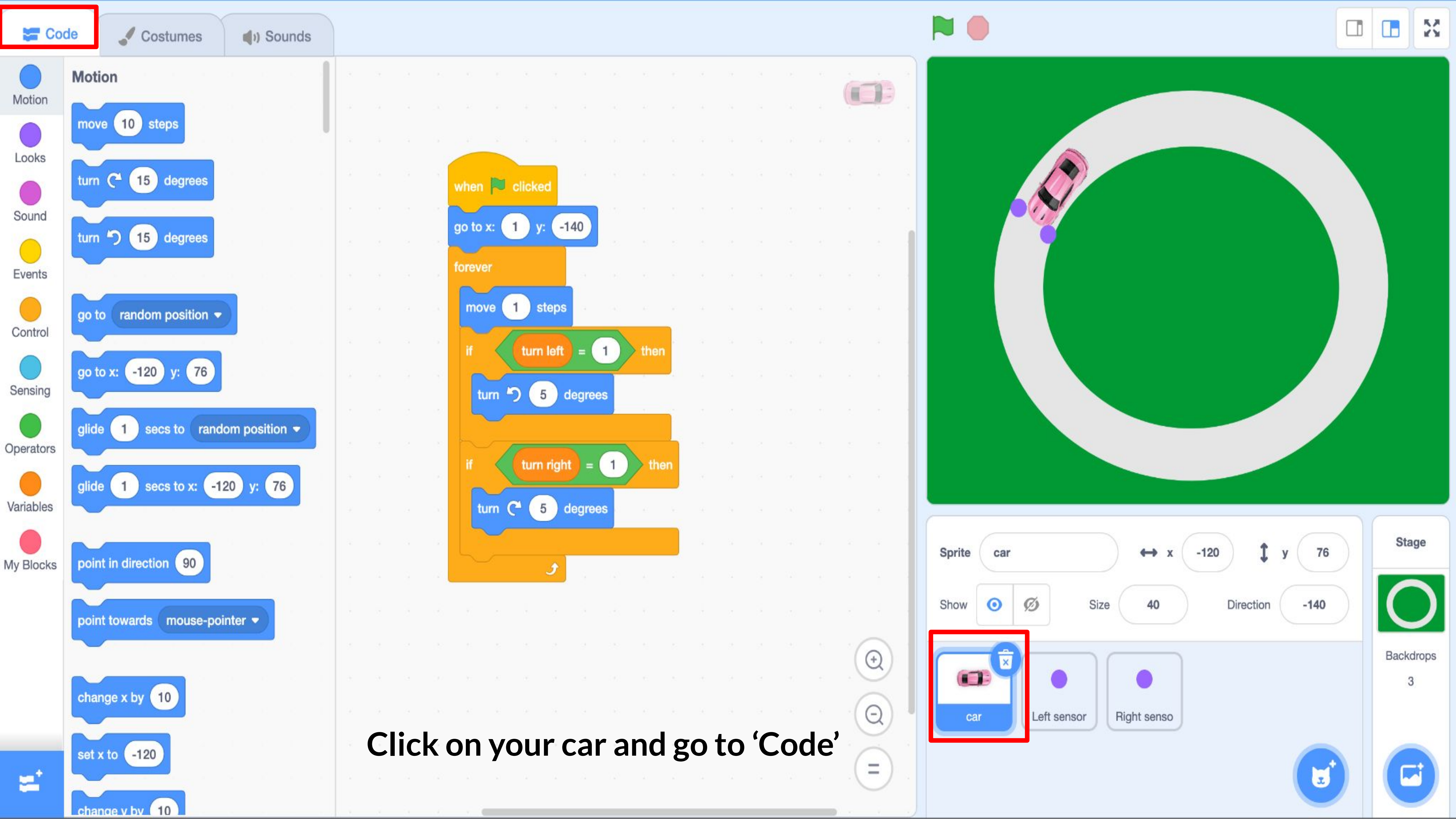


```
when 🏴 clicked
forever
    go to car
    point in direction (direction of car)
    turn ↺ -25 degrees
    move 40 steps
```

```
when 🏴 clicked
forever
    if touching color 🟩 ? then
        set turn left to 1
    else
        set turn left to 0
```

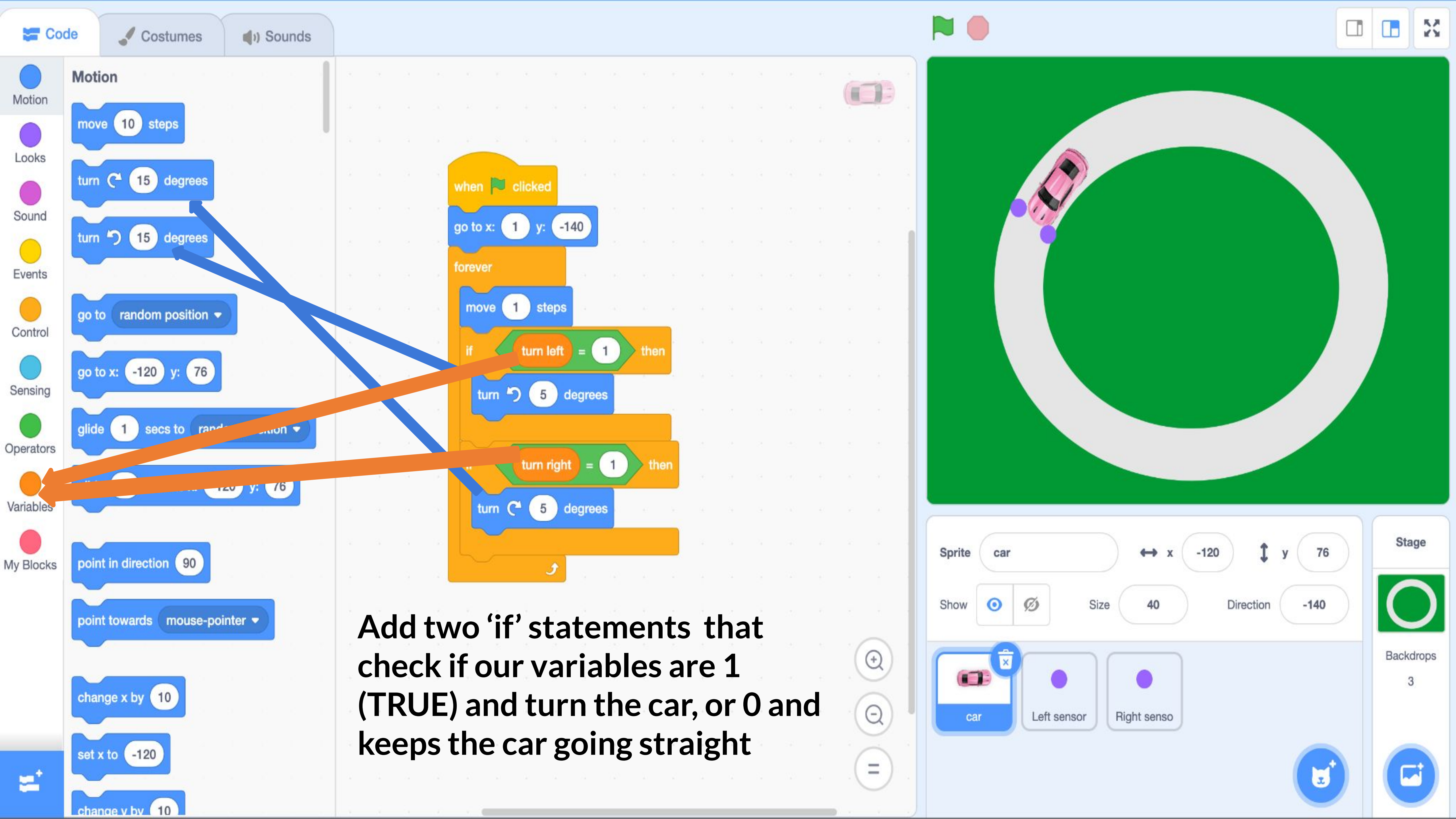
Sprite: Right senso; x: -10; y: 16; Size: 100; Direction: 90

Right click and choose duplicate



```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
    if turn left = 1 then
        turn ↺ 5 degrees
    if turn right = 1 then
        turn ↻ 5 degrees
```

Click on your car and go to ‘Code’

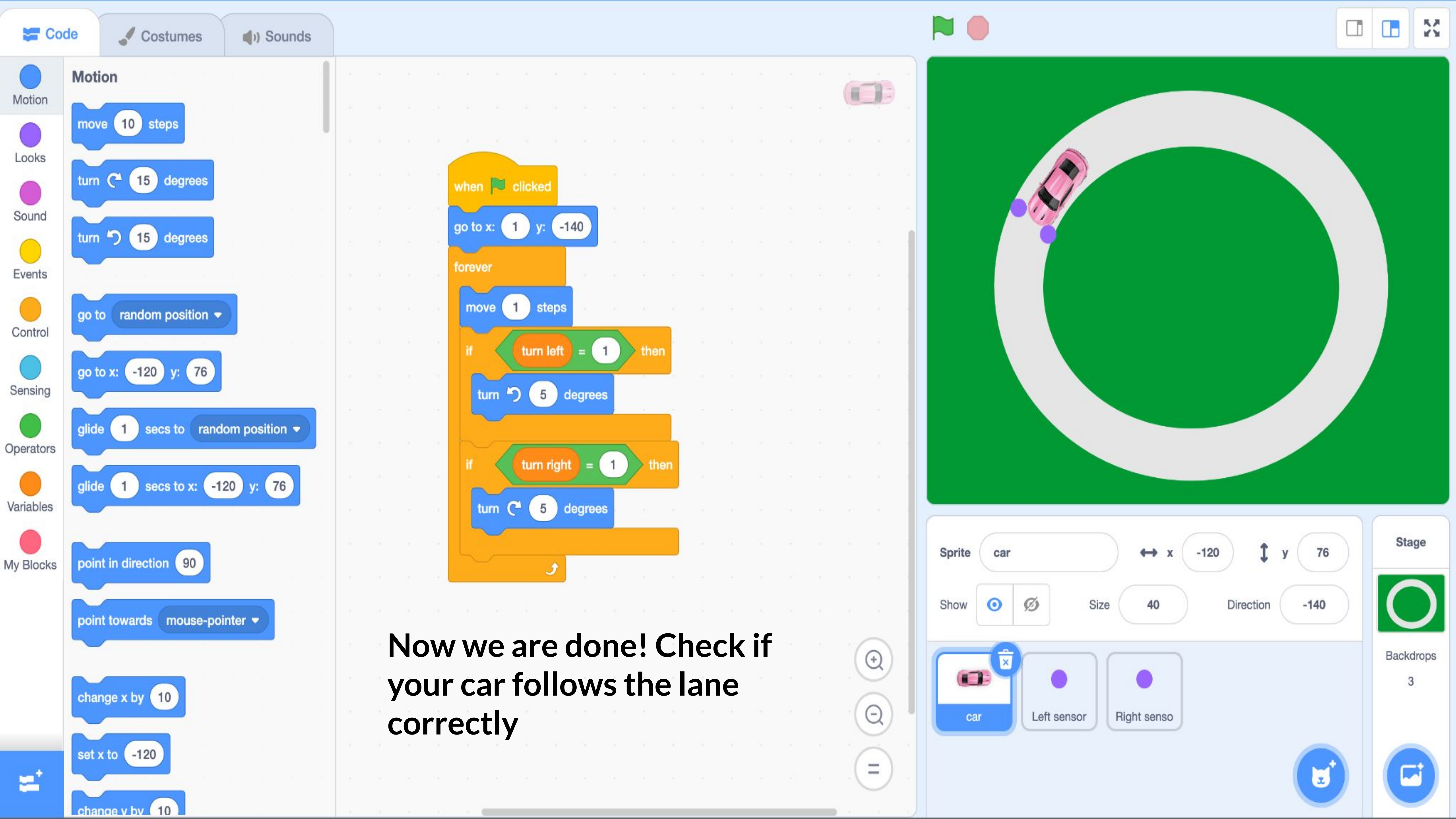


```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
    if turn left = 1 then
        turn ↺ 5 degrees
    if turn right = 1 then
        turn ↻ 5 degrees
```

Sprite car ↔ x -120 ↕ y 76

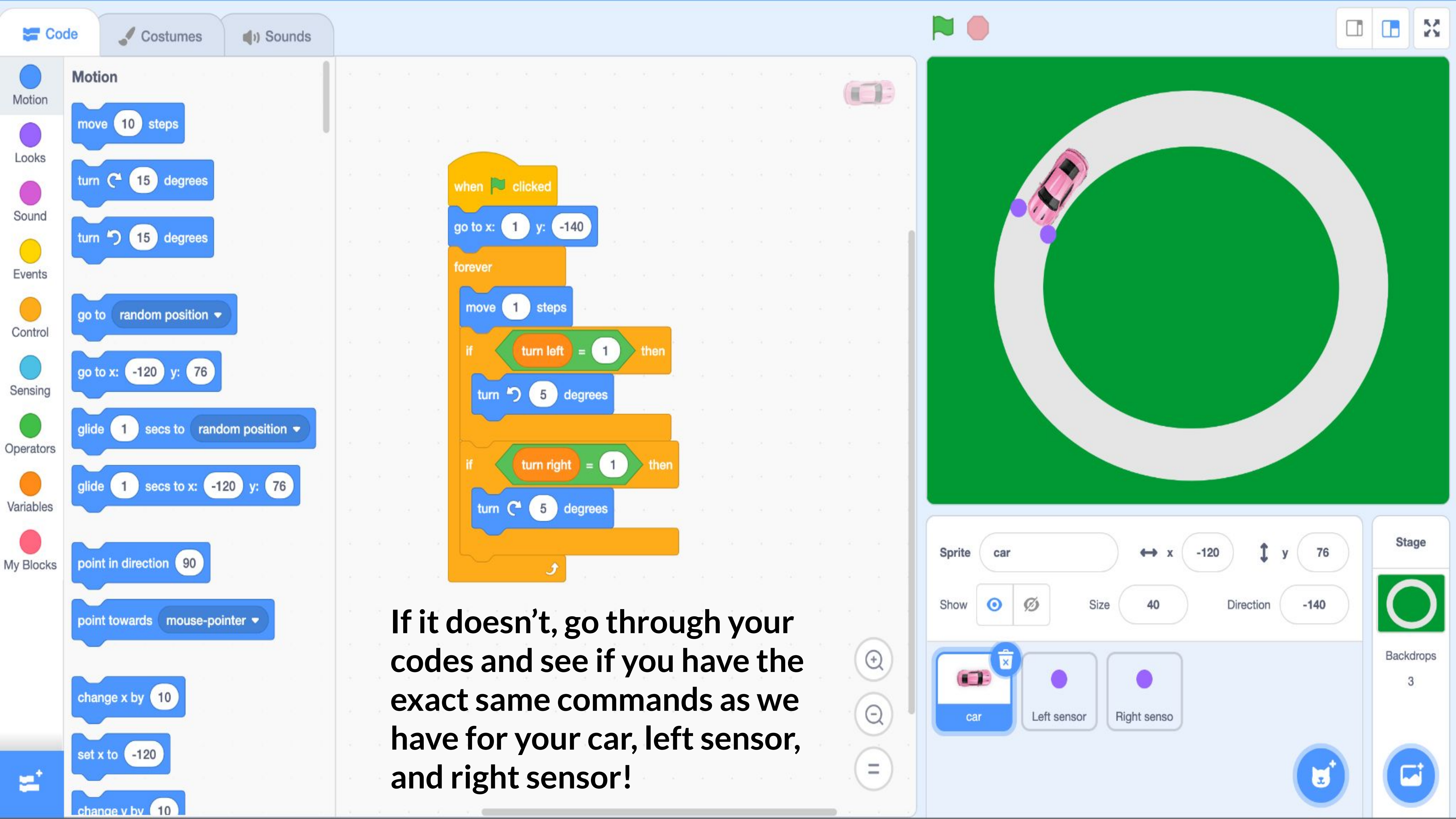
Show Size 40 Direction -140

Add two ‘if’ statements that check if our variables are 1 (TRUE) and turn the car, or 0 and keeps the car going straight



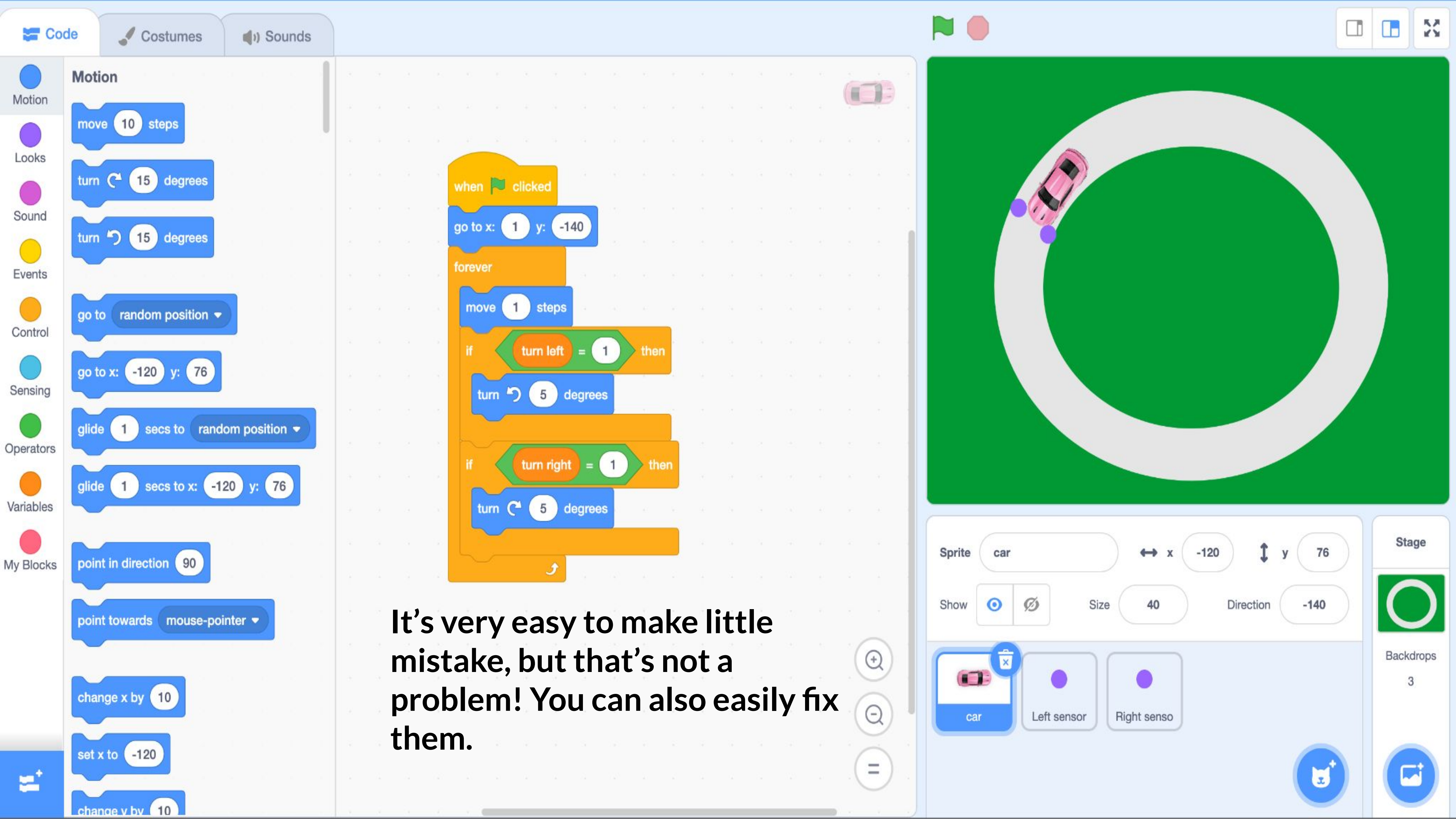
```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
    if turn left = 1 then
        turn ↺ 5 degrees
    if turn right = 1 then
        turn ↻ 5 degrees
```

Now we are done! Check if your car follows the lane correctly



```
when 🏳 clicked
go to x: 1 y: -140
forever
    move 1 steps
    if turn left = 1 then
        turn ↺ 5 degrees
    if turn right = 1 then
        turn ↻ 5 degrees
```

If it doesn't, go through your codes and see if you have the exact same commands as we have for your car, left sensor, and right sensor!



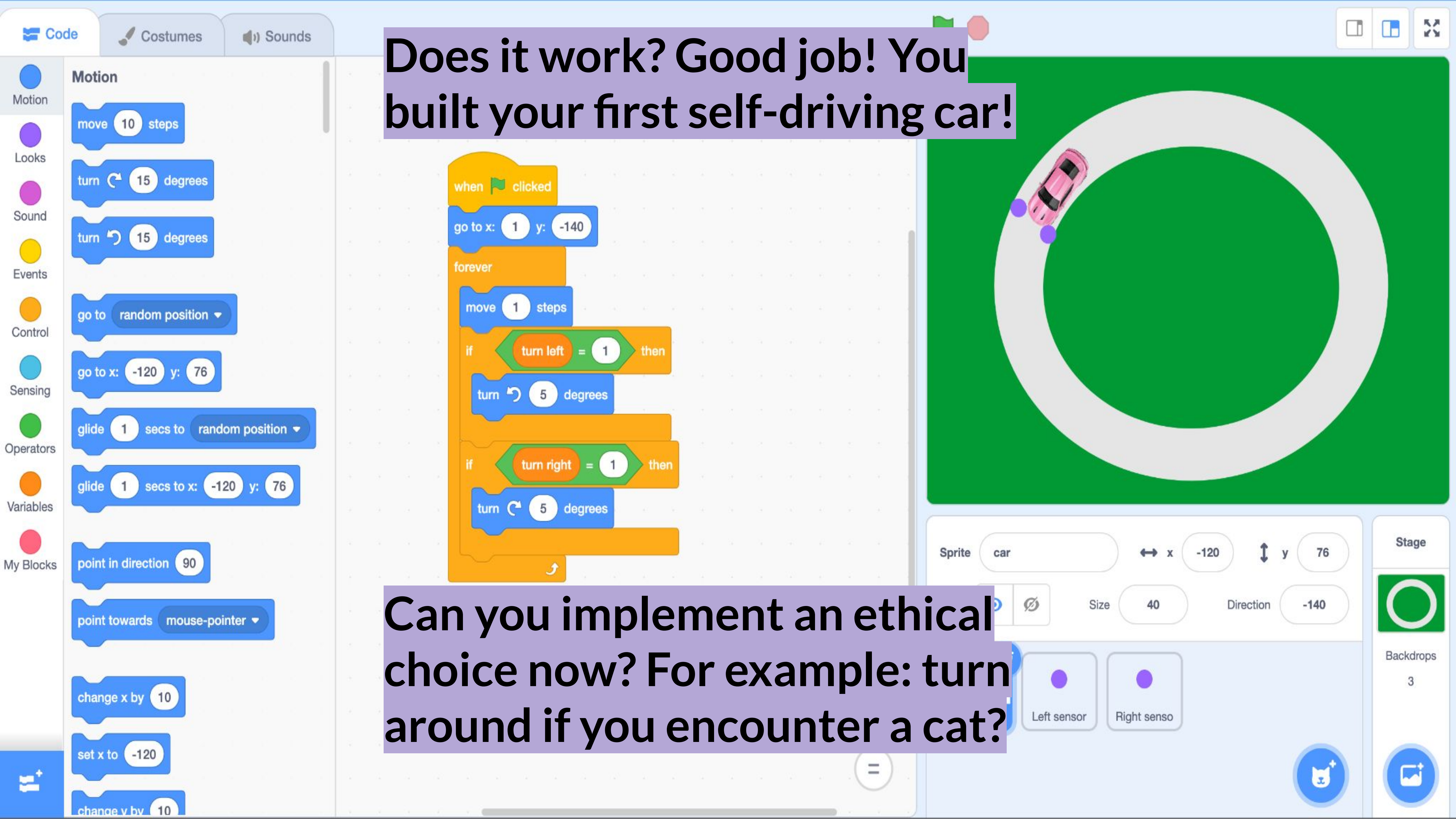
```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
    if <(turn left) = 1> then
        turn ↺ 5 degrees
    if <(turn right) = 1> then
        turn ↻ 5 degrees
```

Sprite: car   ↔ x: -120   ↕ y: 76

Show   Size: 40   Direction: -140

It’s very easy to make little mistake, but that’s not a problem! You can also easily fix them.

Does it work? Good job! You built your first self-driving car!



```
when 🏁 clicked
go to x: 1 y: -140
forever
    move 1 steps
    if turn left = 1 then
        turn ↺ 5 degrees
    if turn right = 1 then
        turn ↻ 5 degrees
```

Sprite car ↔ x -120 ↕ y 76

Size 40 Direction -140

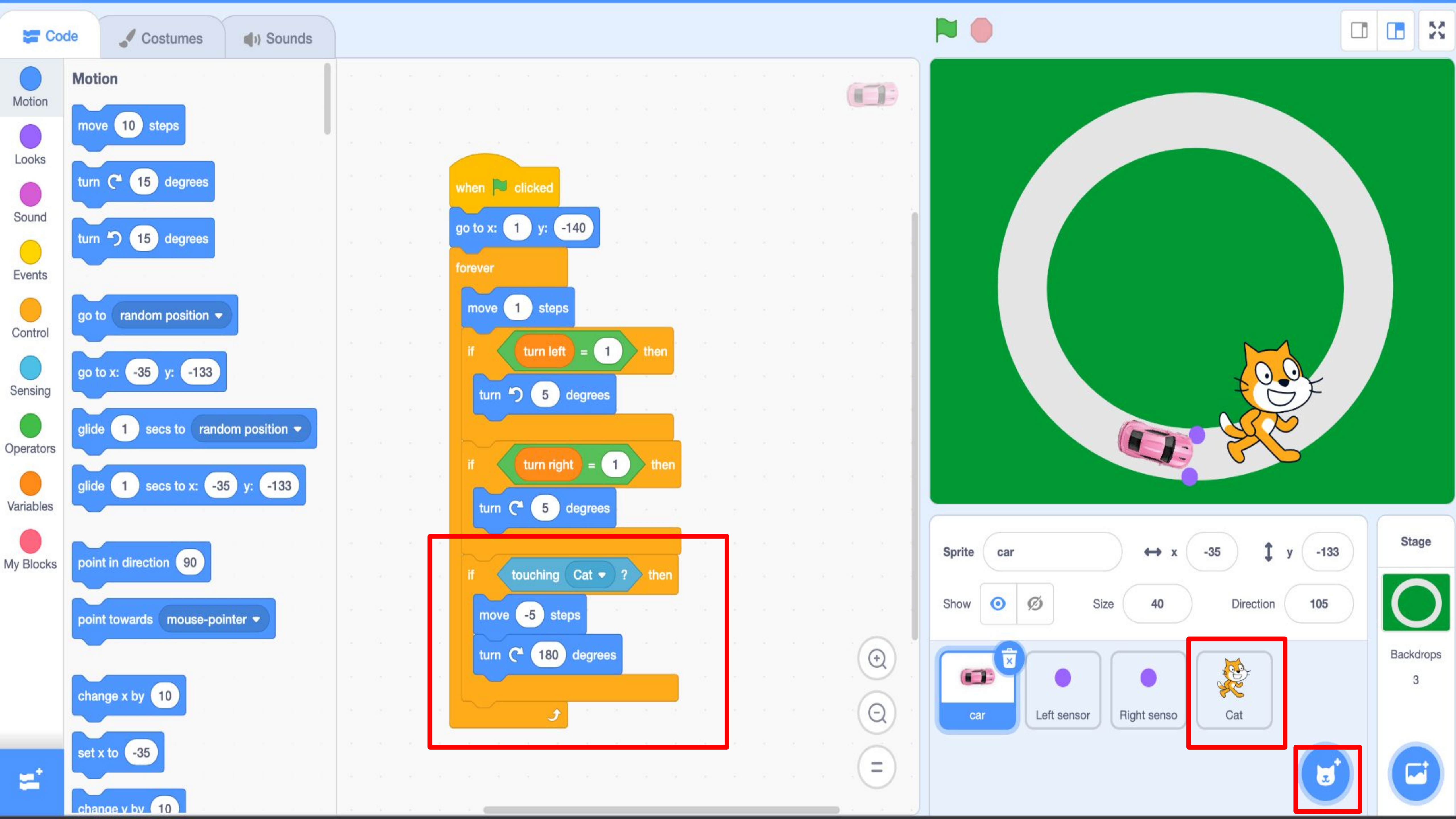
Left sensor

Right senso

Stage

Backdrops 3

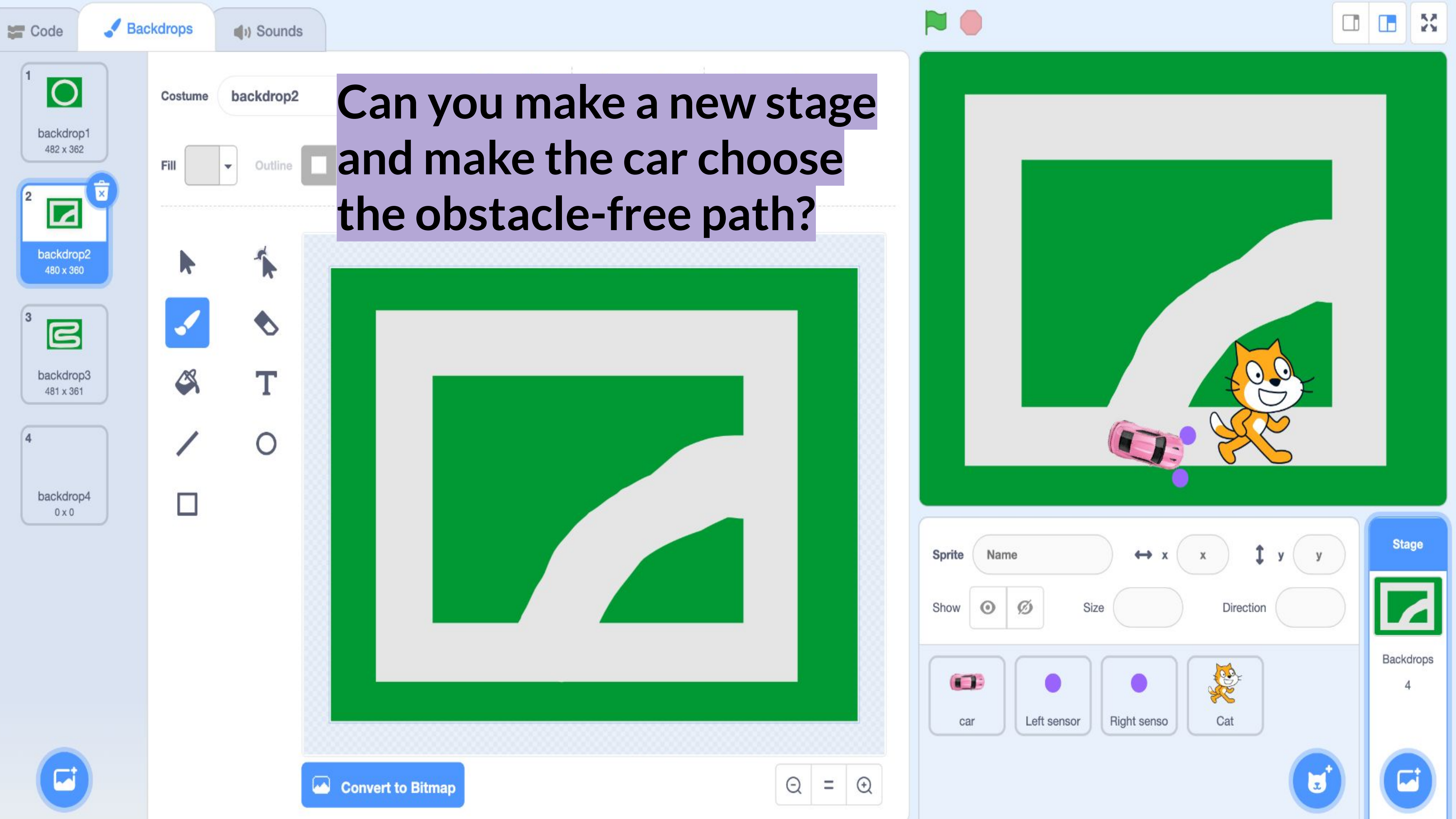
Can you implement an ethical choice now? For example: turn around if you encounter a cat?



```
when 🏴 clicked
go to x: 1 y: -140
forever
    move 1 steps
    if turn left = 1 then
        turn ↺ 5 degrees
    if turn right = 1 then
        turn ↻ 5 degrees
    if touching Cat ? then
        move -5 steps
        turn ↻ 180 degrees
```

Sprite car ↔ x -35 ↕ y -133

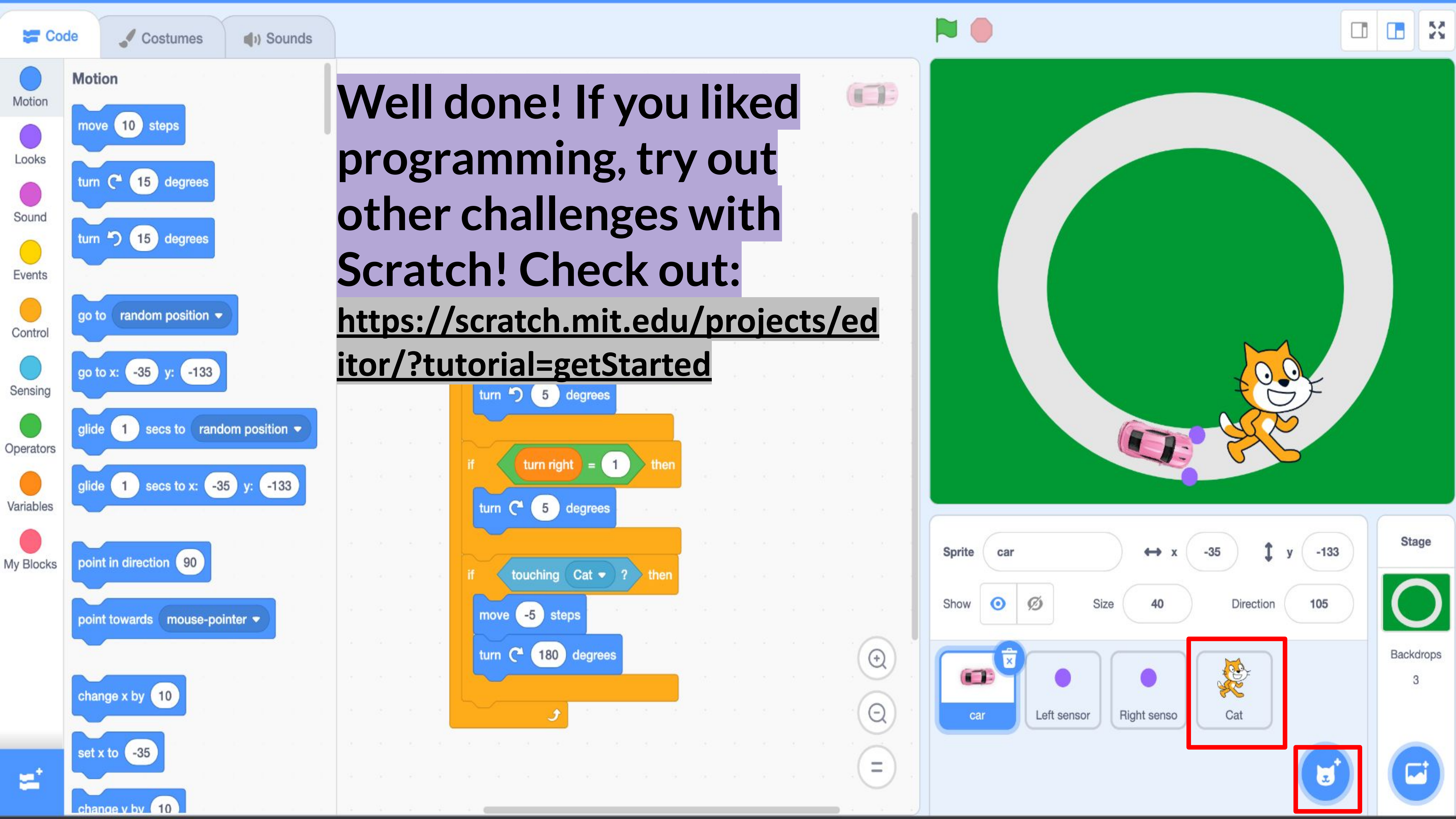
Show Size 40 Direction 105



Can you make a new stage and make the car choose the obstacle-free path?

Well done! If you liked programming, try out other challenges with Scratch! Check out:

https://scratch.mit.edu/projects/editor/?tutorial=getStarted



```
turn ↺ 5 degrees
if <(turn right) = 1> then
    turn ↻ 5 degrees
if <touching Cat ?> then
    move -5 steps
    turn ↻ 180 degrees
```

Sprite: car | x: -35 | y: -133

Show | Size: 40 | Direction: 105