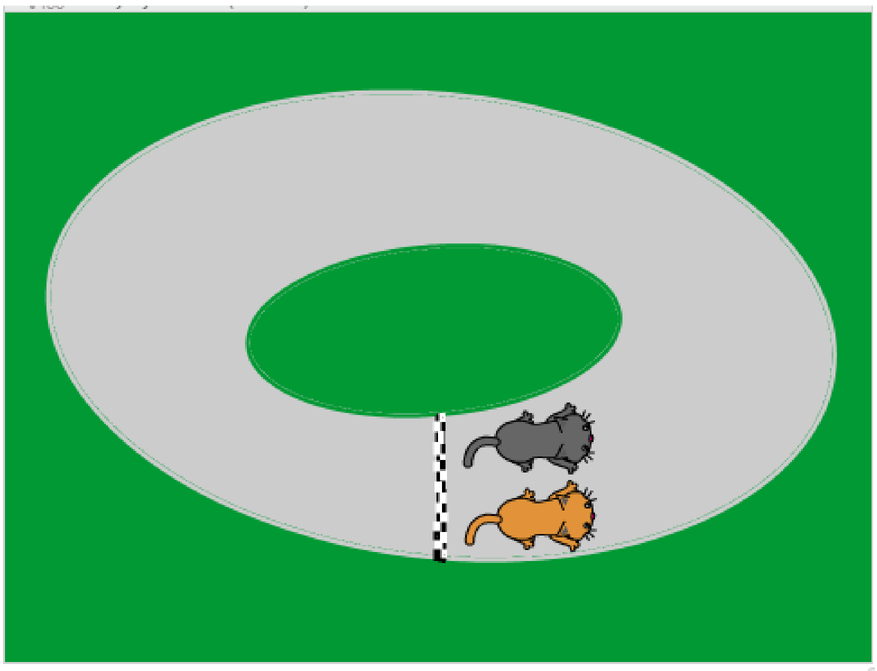
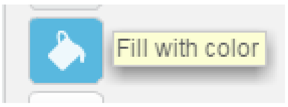
2-player racing game

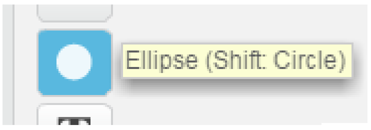
Thanks to https://wiki.scratch.mit.edu/wiki/How_to_Make_a_Two-Player_Racing_Game for this project. See my version online at https://scratch.mit.edu/projects/167215363/.



We are going to make a racing game for two players!

Step 1: Draw the backdrop

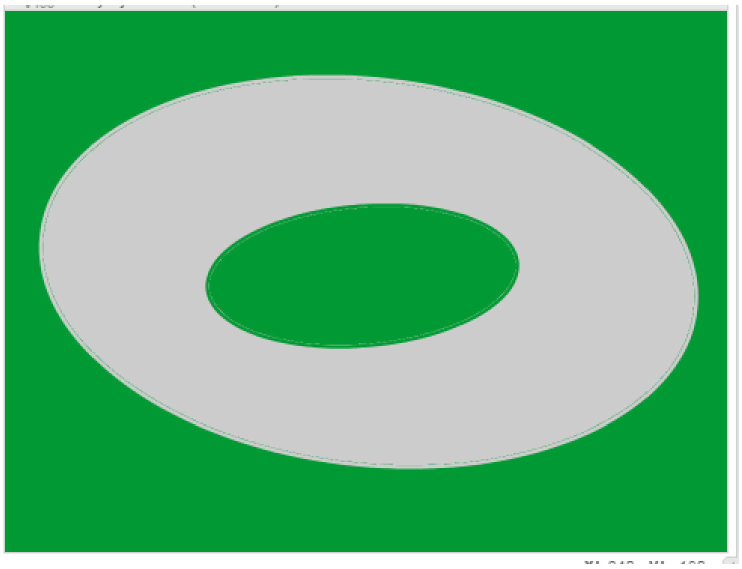
- Start a new Scratch project, and right-click to delete Scratch cat.
- Go to the backdrop, and decide on colours for your track and its surroundings; I chose grey (tarmac) and green (grass).
- Use the Fill with colour tool 

to fill your backdrop with the colour for the grass.
- Now switch to the tarmac colour, and use the ellipse tool


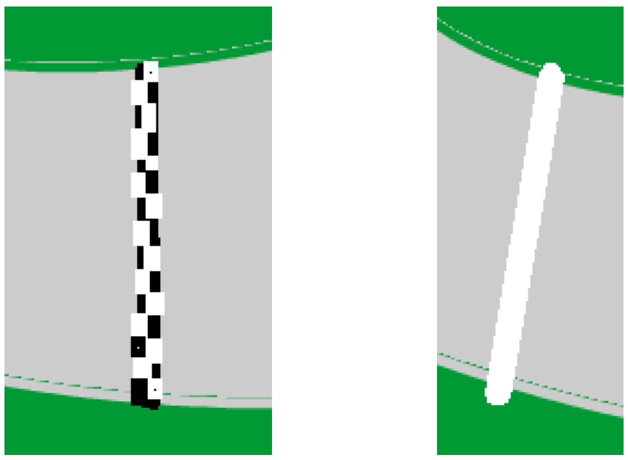
to draw the outside of the track. You can drag and stretch the box around it until it is the right size, and in the right place.

- Use the Fill with colour tool again to colour it all in grey (or your track colour, if different).
- Now switch the colour back to green (you can use the Pick up colour tool if you need to

) and draw another ellipse to be the centre of the track. If you are going to write code to keep your sprites on the track, it will be easier if you have used the same colour for both the inside and outside.
- Don’t make your track too narrow! Remember two sprites will be moving on it.
- You should end up with something that looks a bit like this:



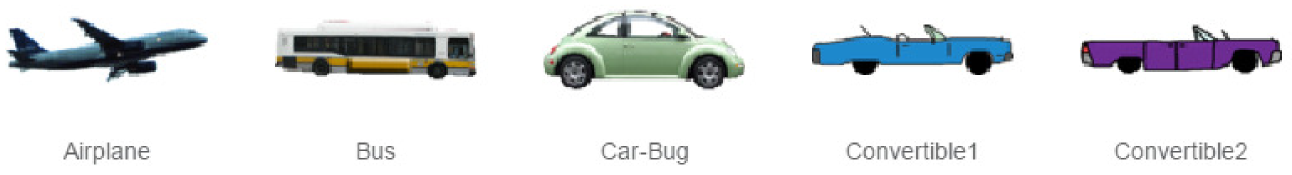
- The last thing we need on the backdrop is a finishing line. You can do a fancy black and white one if you like (use the zoom to make it easier), or just a simple line, but make sure it is a different colour from everything else.



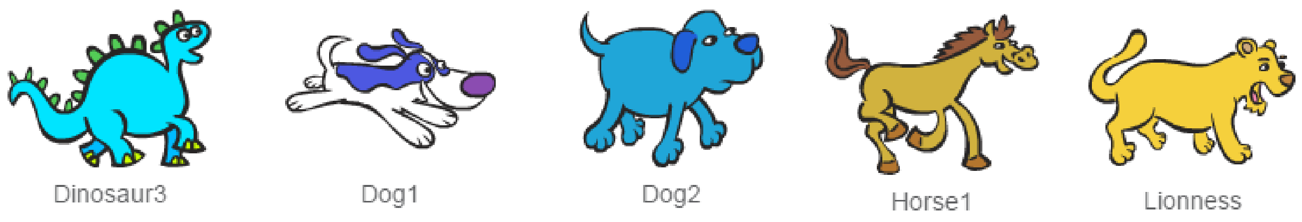
Save your project

Step 2: Choose your sprites

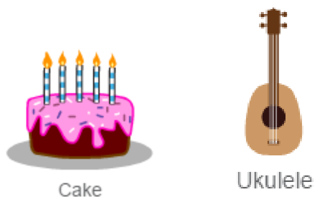
- Look in the sprite library for two sprites who could race against each other. There is a Transportation section with several cars and other vehicles:



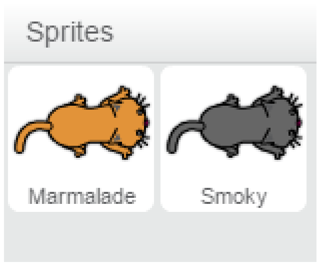
- Or you could choose animals:



- Or something quite random:

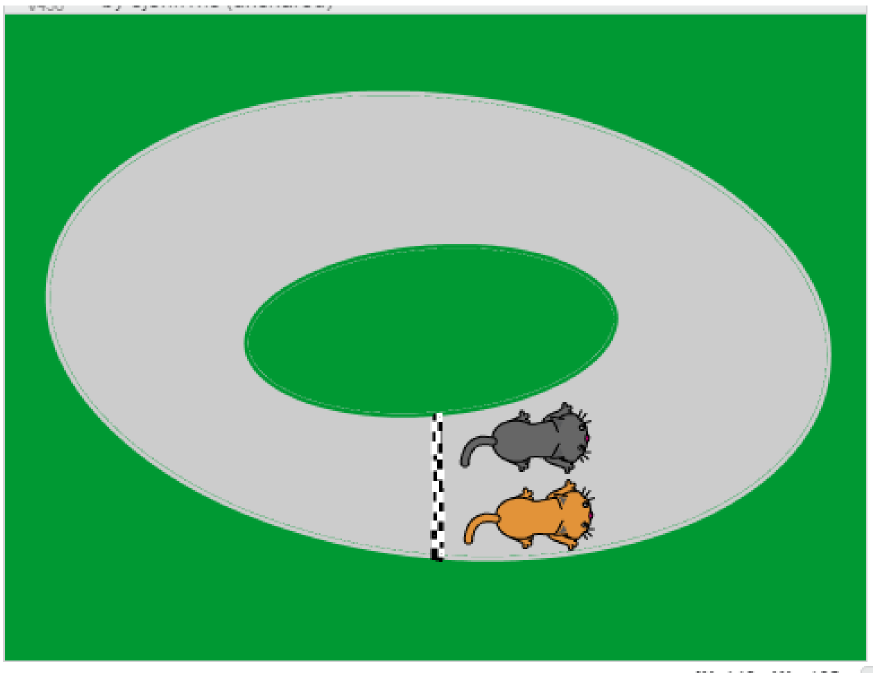


- I chose the cat seen from above, duplicated it and changed the colour of the second one. You can give them names if you like:



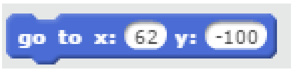
- You will probably need to shrink your sprites to make them fit easily on the track

. Position them just after the finish line like this (they mustn't be touching the finish line because that is how we test for the end of the race):

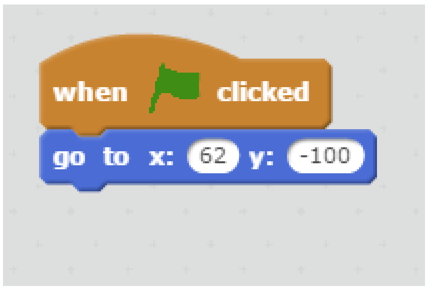


Save your project

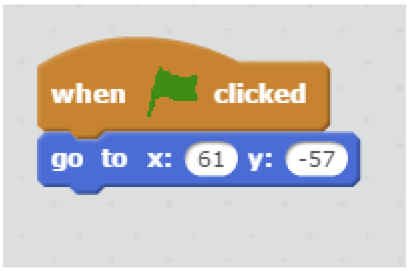
Step 3: Code your first sprite

- One sprite will be moved by the up, down, left and right arrow keys, and the other one by the W, A, S and D keys (so you and the other player can both use half of the keyboard comfortably). The code is very similar.
- On your first sprite, start by coding its starting position – we have placed it exactly where we want it to start, so the block 

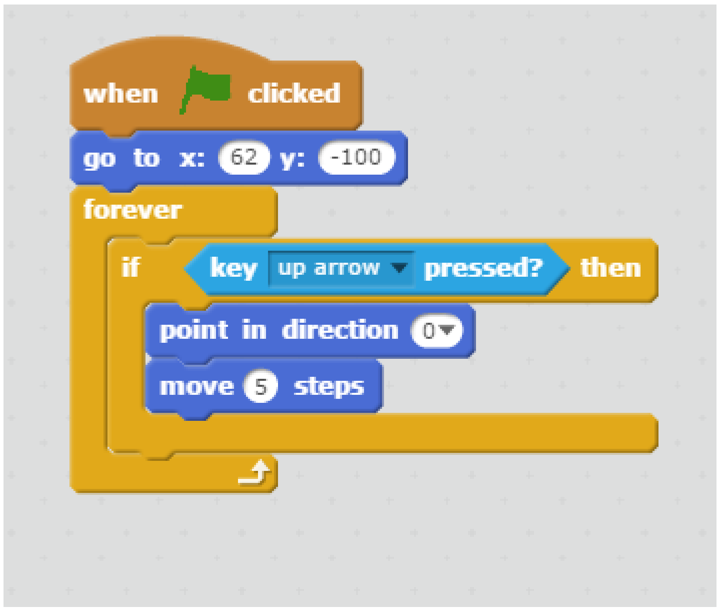
will have the correct numbers in it already. Drag this block into your code, and add the green flag 'hat' block, and then do the same for the second sprite.
- Your code should look something like this, although your numbers will be different – that is fine:


```
when [flag] clicked
go to x: 62 y: -100
```

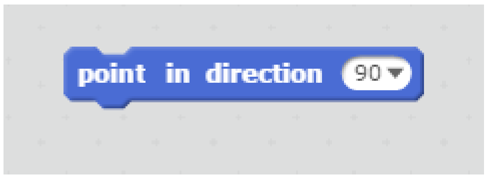
and


```
when [flag] clicked
go to x: 61 y: -57
```

- Next we need a forever loop, and some code that checks for a key press, points in that direction, and moves our sprite. Let's add code for the up arrow to the code we have started for sprite 1:


```
when [flag] clicked
go to x: 62 y: -100
forever
    if <key up arrow pressed?> then
        point in direction 0
        move 5 steps
```

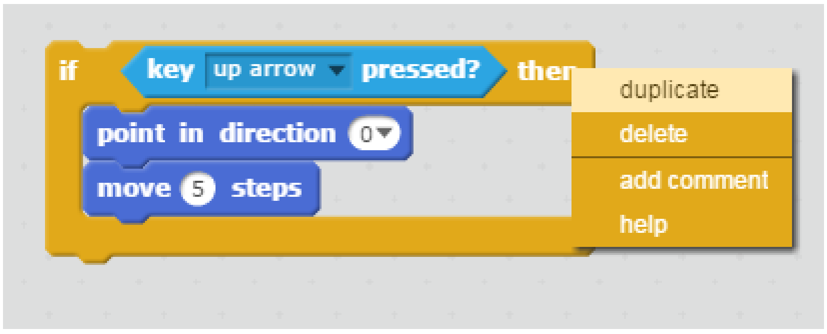
- Test your code! You should find that sprite 1 responds to the up arrow, but nothing else happens (because we haven't coded it yet). Run it a second time.
- When you test your game, you will notice that the sprites may not face the right way at the start. Add this block before the forever loop to each sprite:

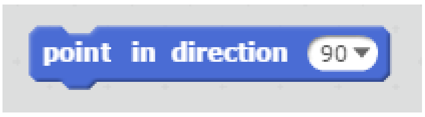

```
point in direction 90
```

- Does that fix it?

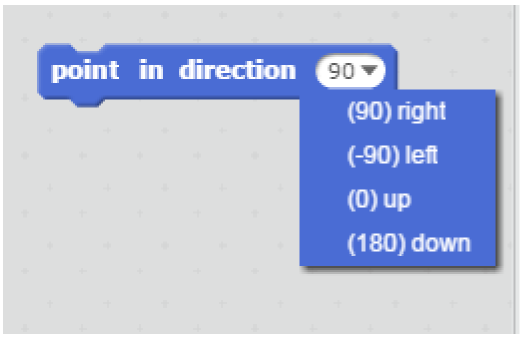
Save your project

- Ok, let's add code for the left, right and down arrows to our first sprite. This needs to go inside the forever loop, test for the press of each key, point in the right direction, and move the sprite.
- Can you add the code without reading any further? If you are an experienced Scratcher, you will realise that 'duplicate' (right-click on your code) will come in handy:



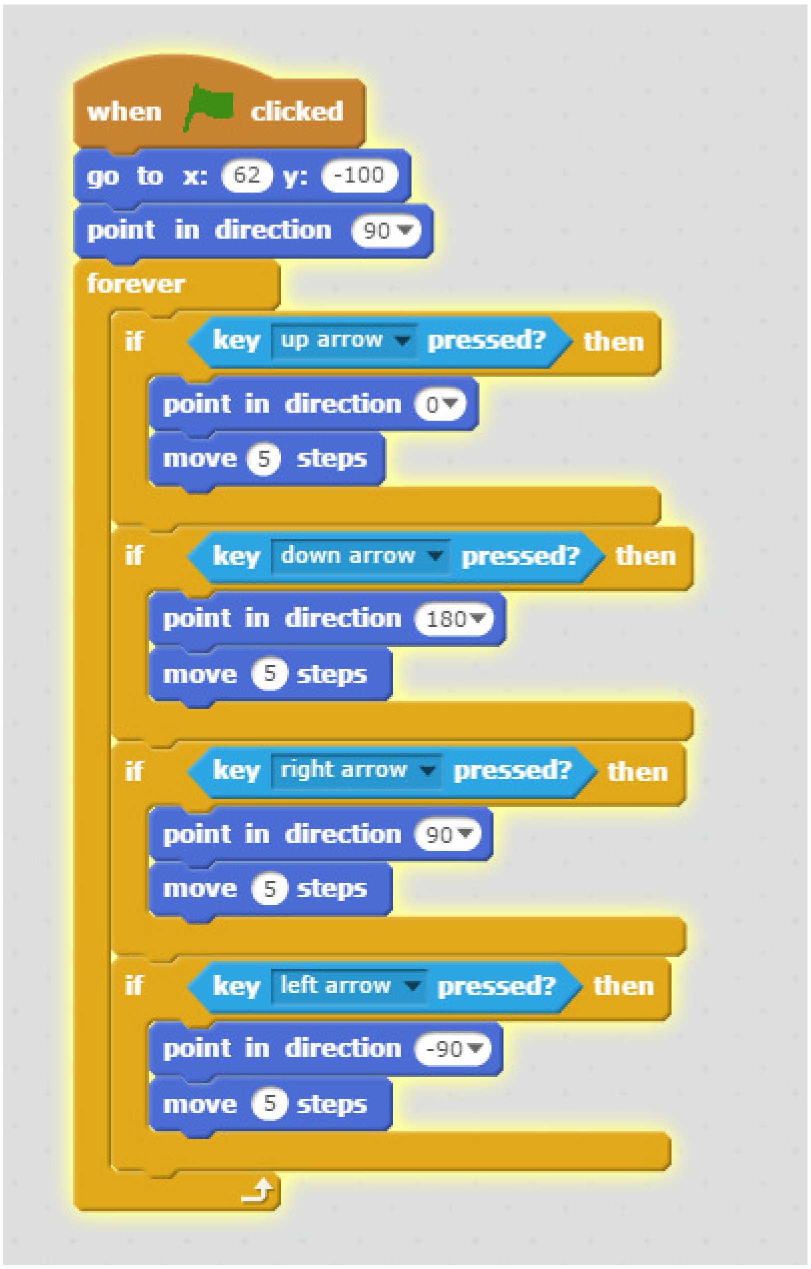
- Don't worry if you need to check it, the full code is on the next page. Be careful with the 

block – you must make sure that the direction you choose matches the key that you have pressed:



- Test your project – and keep testing until Sprite 1 works in all four directions!

- Here is what your code should look like:



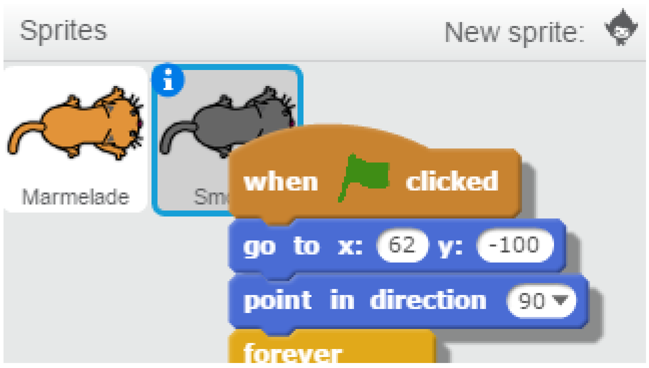
```
when [flag] clicked
go to x: 62 y: -100
point in direction 90
forever
    if key up arrow pressed? then
        point in direction 0
        move 5 steps
    if key down arrow pressed? then
        point in direction 180
        move 5 steps
    if key right arrow pressed? then
        point in direction 90
        move 5 steps
    if key left arrow pressed? then
        point in direction -90
        move 5 steps
```

Save your project

Step 3: Code your second sprite

- Remember that our second sprite is going to move with the WASD keys, which are laid out on the keyboard in a similar way to the arrow keys. W will move the sprite up, A will move left, D will move right, and S will move it down.

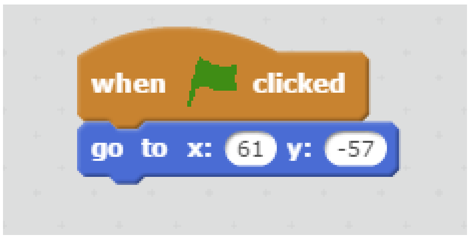
- Can you duplicate the code from your first sprite, and make these changes on the second sprite? To copy the code, hold down the mouse button over the ‘hat’ block on sprite 1, and drag it over the second sprite, then let go:



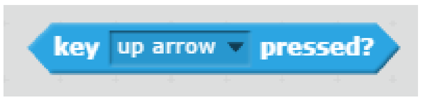
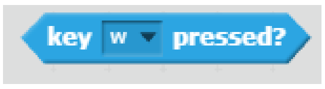
```
when [flag] clicked
go to x: 62 y: -100
point in direction 90
forever
```

(you can’t see the mouse in this picture, but it was over the ‘when’)

- Remember that we already had the starting position for Sprite 2; mine was:

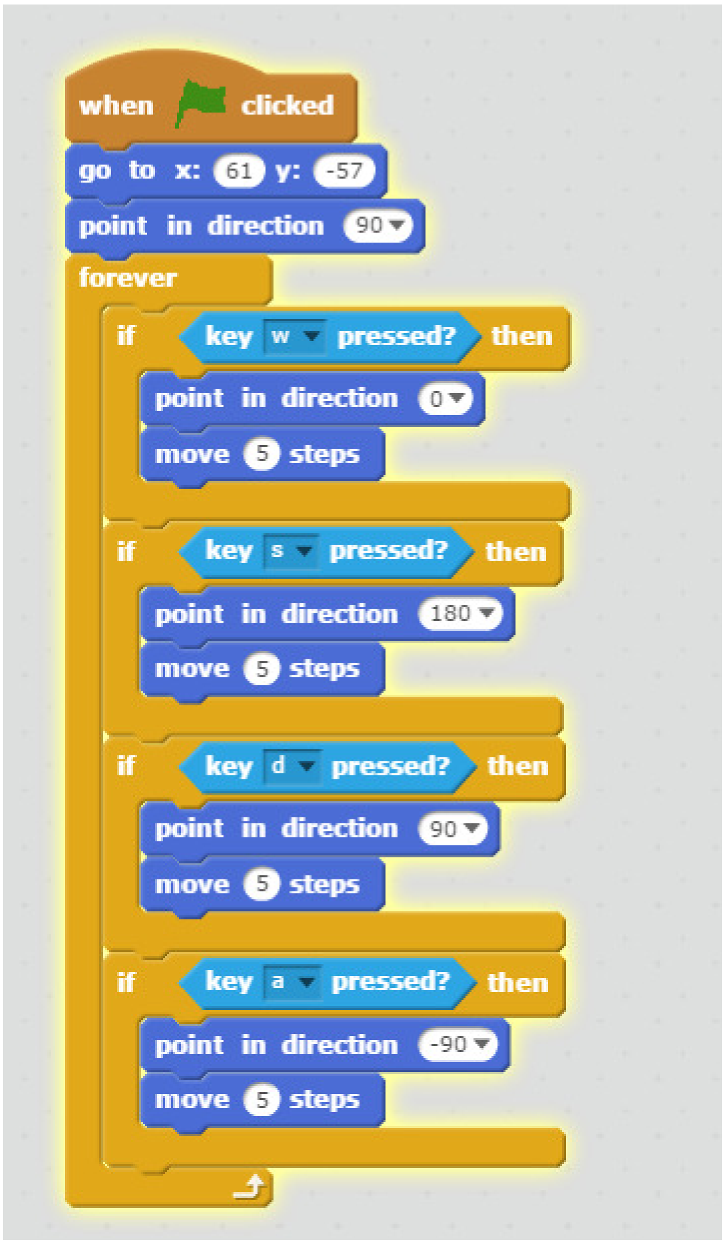


```
when [flag] clicked
go to x: 61 y: -57
```

- So we can throw away the first two blocks from Sprite 1, and join the rest of the code onto the ones we have for Sprite 2.
- Now all we have to do is change the sensing code blocks  `key up arrow pressed?` to the letter keys as described above, so this one becomes  `key w pressed?`. If we get this right, everything else should work!
- Test your project. Make sure you test everything, up, down, left and right for Sprite 1, and up, down, left and right for Sprite 2. Does it all work?

Save your project

- If you need to check your code for Sprite 2, it should look like this:



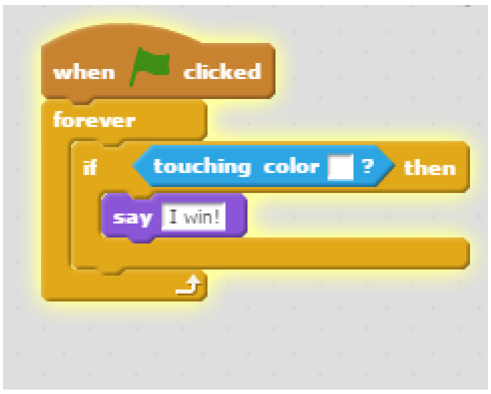
```
when [flag] clicked
go to x: 61 y: -57
point in direction 90
forever
    if key w pressed? then
        point in direction 0
        move 5 steps
    if key s pressed? then
        point in direction 180
        move 5 steps
    if key d pressed? then
        point in direction 90
        move 5 steps
    if key a pressed? then
        point in direction -90
        move 5 steps
```

- Test your project with a friend! Try it a few times, does it feel like a fair race? Is there anything you want to tweak?

Save your project

Step 4: Winning the race

- The first sprite to reach the finish line is the winner, so add this block of code to both sprites (you can change what they say, if you like):



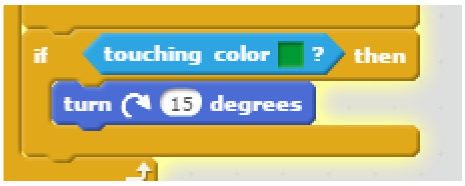
```
when clicked
forever
    if touching color ? then
        say I win!
```

- Make sure the colour you are testing for is the colour of your finish line (I used white), and that you aren't using this colour anywhere else in the project.
- Test your project, making sure both sprites have been given a chance to win, and you see the result you expect.

Save your project

Step 5: Keeping to the track

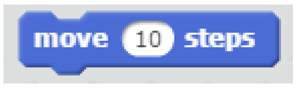
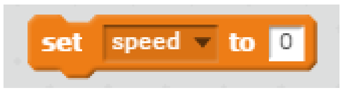
- You have probably found that it makes no difference whether our sprites stay on the track or not – I thought there ought to be a penalty for touching the grass. But if we stop our sprite from moving when it touches the grass, it will just get stuck forever? And we can't easily make it jump backwards a bit, because we don't know which direction it is going in…
- I decided that grass would confuse the cats and make them spin in circles, so I added this code inside the forever loop that checks for winning:

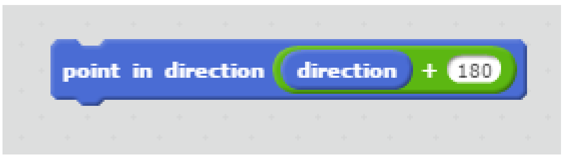


```
if touching color ? then
    turn 15 degrees
```

- Try it out. The sprite will rotate when it goes on the grass, but it can still move.

More ideas

- We could add a timer, that stops when one sprite wins.
- Do you think your sprites move too fast, or too slowly? Change the number in the 
  `move (10) steps`
  block to something higher or lower, and see if you prefer it (remember to change it in every direction!). This could be a bit boring, so make a variable called speed, use this in each move block instead, and use the 
  `set [speed ▾] to [0]`
  block to assign a value at the beginning.
- Following on from the suggestion above, try randomly assigning a speed to each sprite at the start. How much variation do you need, for a good game?
- The sprite that starts nearer the outside of the track has further to go to win than the sprite that starts on the inside. How could you make this fairer – make it slightly faster, maybe?
- Suppose you did want the sprite to jump backwards a bit when touching the grass, you can be a bit clever with direction. Try something like:

  ```
  point in direction ((direction) + (180))
  ```

  This will make the sprite face backwards, until you next press a movement key. Do you prefer that to spinning in circles?
- Imagine the track was on a slope, and your sprites went faster when they were going to the left (or up, down, right)? They could also go slower when going to the right (or down, up, left). How would you code this? To be fair, it would have to be the same change on each sprite. Does it make the game more exciting, or more annoying?
- This project has no sounds…