FIRST CODING
creative coding for kids

Amazing Mazes

Introduction

Scratch is a visual programming language suitable for children and teens to create their own interactive stories, games and animations. Projects can also be uploaded and shared with the Scratch community via the Scratch website for others to enjoy.

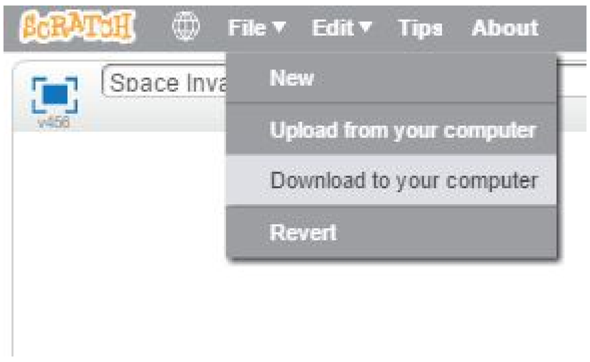
In this exercise we'll learn about how to make a maze game and how we can randomise the actions of certain elements to make the game unpredictable when being played.

Let's get started by first loading Scratch. You can either install Scratch on your computer or load it via your web browser by navigating to https://scratch.mit.edu/

Top Tip

Don't forget to save your work regularly. It's good practice to get in the habit of saving any progress you've made. So when you're feeling great after solving a coding problem, save it, save it, save it!

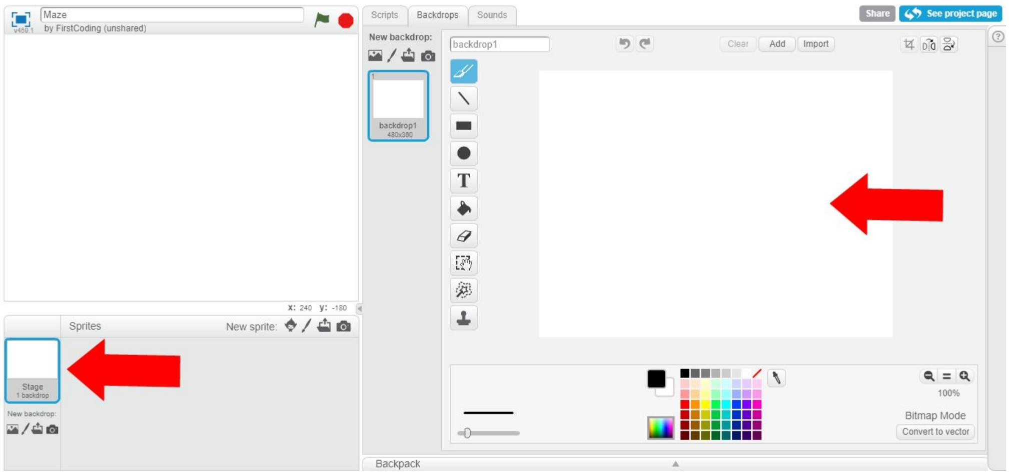
The save options may differ slightly between Scratch versions but head on up to the File option in the top left and you'll find it.



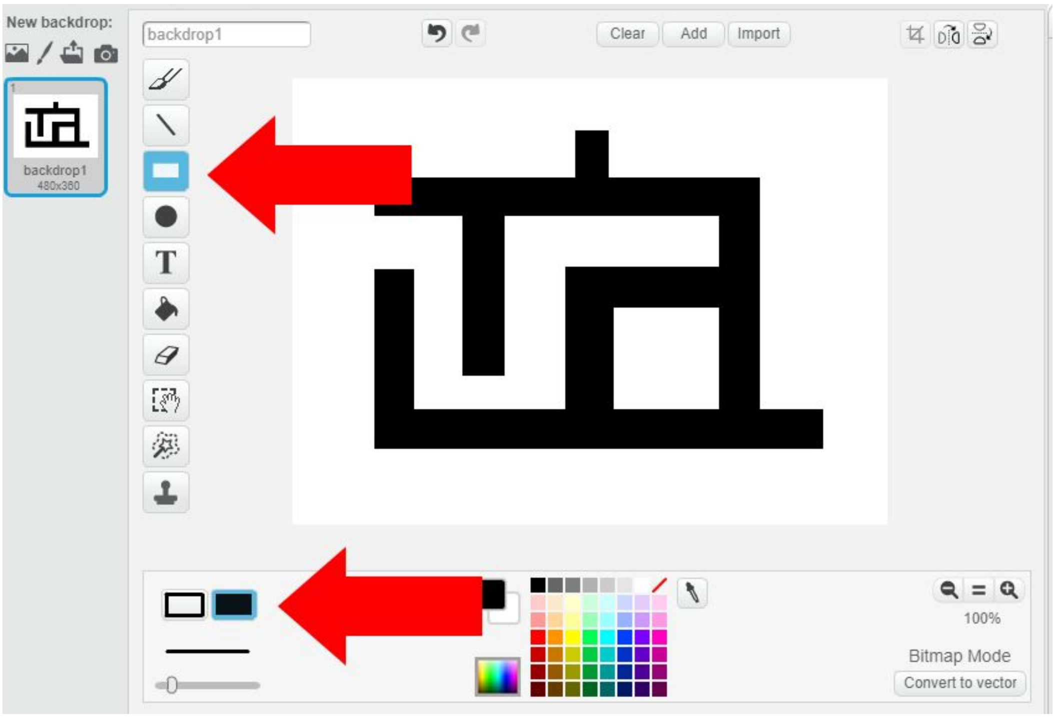
Let's get started

Step 1: Drawing the maze

We first need to draw the maze layout for our game. As this is managed as a backdrop to our project, make sure the Stage Backdrop is selected and this will allow us to draw our simple maze layout in the editor on the right hand side.

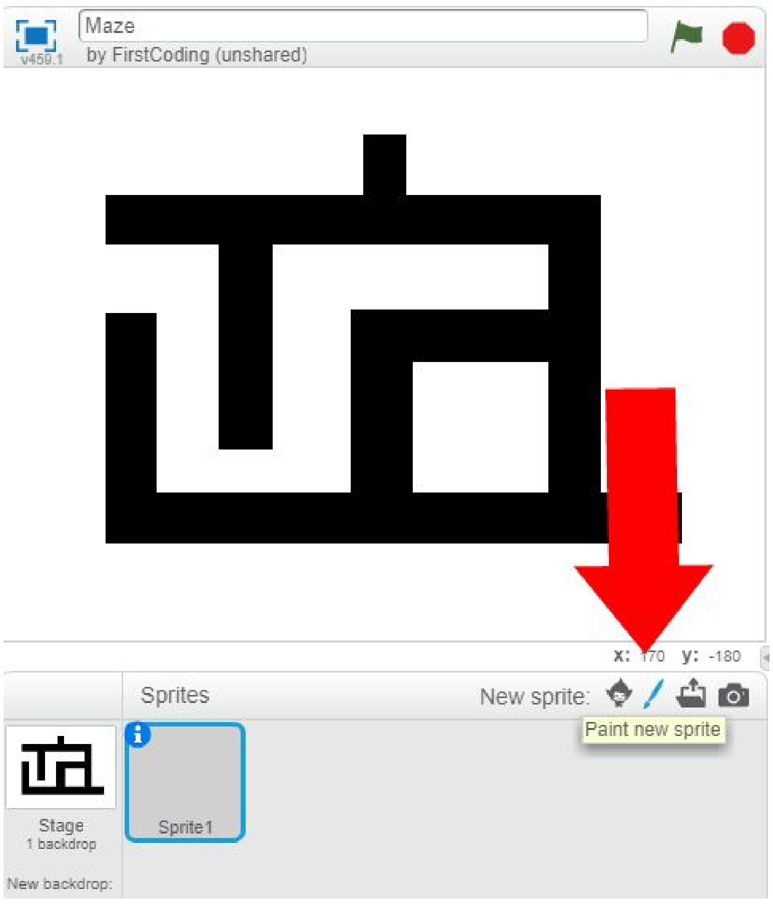


For this exercise we'll start off by drawing a simple maze. More difficult layouts can be drawn at a later date once we know our game works. For the meantime though use the rectangle drawing tool with the fill colour option selected and we can then start drawing the maze.

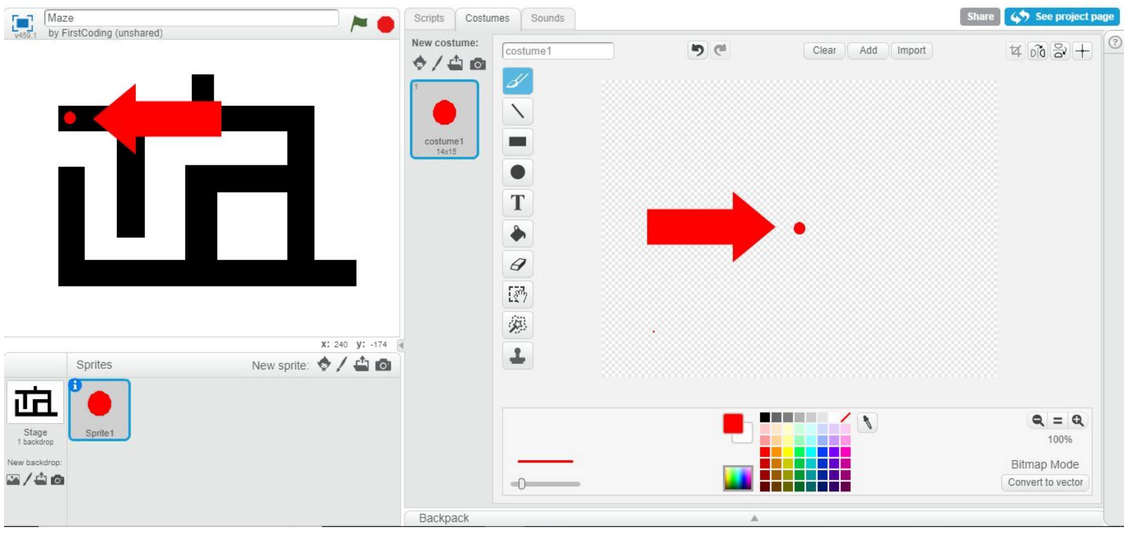


Step 2: Creating our player

We need to create sprite that we'll direct around the maze. This is done by selecting the Paint New Sprite option as shown below:

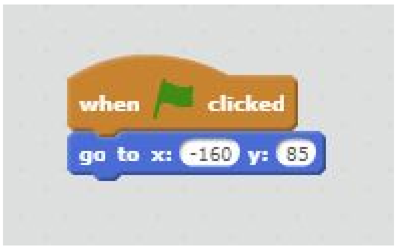


Then from within the editor we can design our sprite. In this exercise we're going to keep it nice and simple by just creating a red dot. It is this red dot that we'll be programming so that we can guide it around the maze.



Step 3: Programming our player

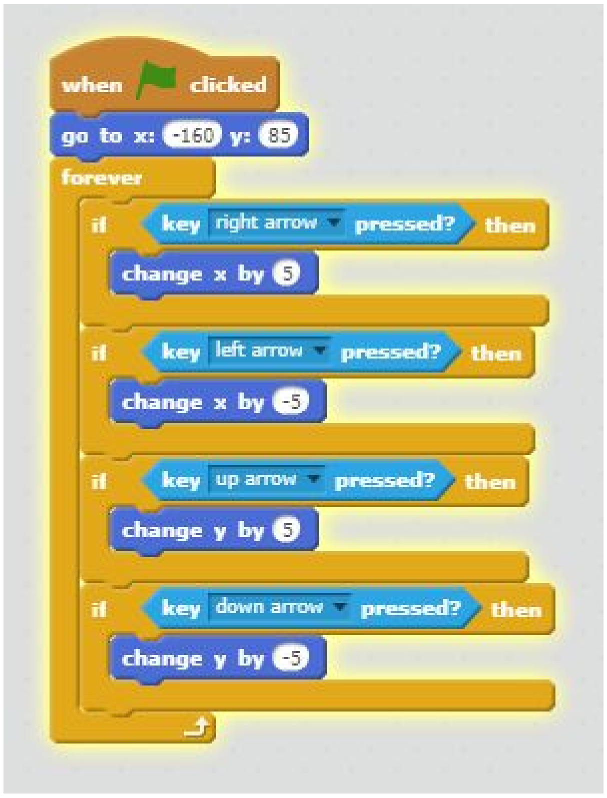
Like with all Scratch programs, everything starts with the click of the green flag, so locate the ‘When Green Flag Clicked’ block and add the ‘Go To’ block found in the Motion section. By adding the starting coordinates of our sprite, this ensures that we start at exactly the same place every time we click the green flag to start a new game.



```
when clicked
go to x: -160 y: 85
```

We can now add the standard set of blocks that will allow us to control the sprite via the arrow keys on the keyboard. Be sure to keep all the IF blocks sat within a FOREVER block as this will ensure our program is always checking to see when a key is press.

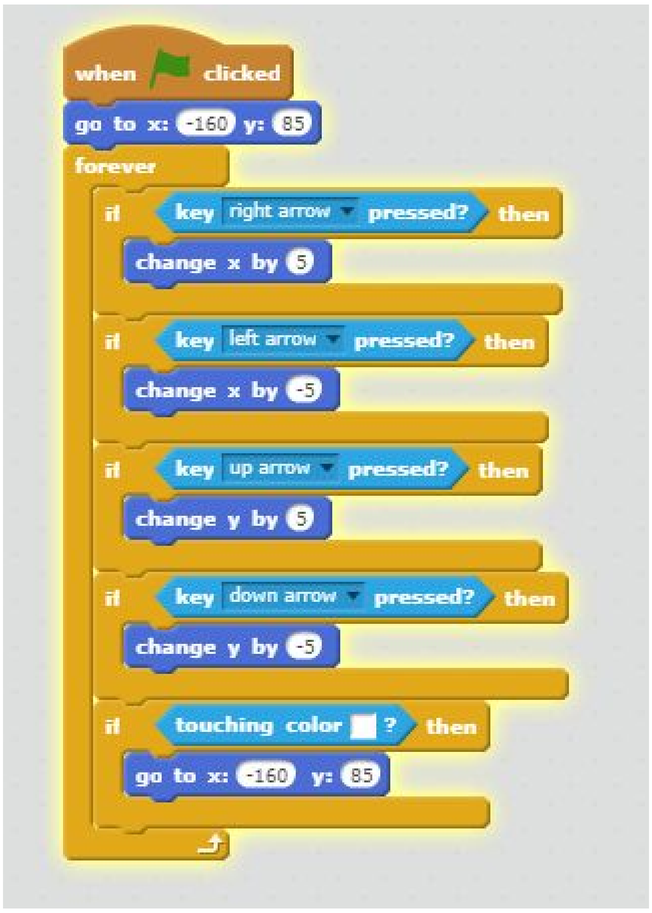
Set the ‘Change By’ block values to only move them 5 pixels at a time. If we allow our sprite to move too fast it will make the game difficult to play. Experiment with different values to see what speed you feel comfortable with.



```
when clicked
go to x: -160 y: 85
forever
    if key right arrow pressed? then
        change x by 5
    if key left arrow pressed? then
        change x by -5
    if key up arrow pressed? then
        change y by 5
    if key down arrow pressed? then
        change y by -5
```

To make the game more challenging we’re going to add a feature whereby if we accidentally touch any walls of the maze we will automatically get taken back to the start. This is achieved by adding in another IF block to our FOREVER loop. This time we're continually checking to see what colour our sprite is touching. If it is touching the colour white then the program knows that we've made a wrong move in the game.

The colour sensing block can be found in the Sensing section. Use the same 'Go To' block as before to ensure we get taken back to the start when the colour white is sensed.

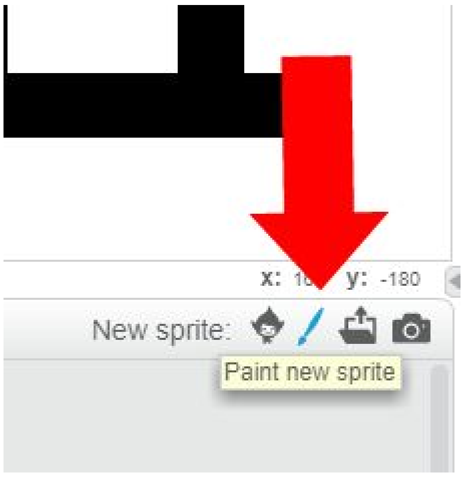


```
when clicked
go to x: -160 y: 85
forever
    if key right arrow pressed? then
        change x by 5
    if key left arrow pressed? then
        change x by -5
    if key up arrow pressed? then
        change y by 5
    if key down arrow pressed? then
        change y by -5
    if touching color ? then
        go to x: -160 y: 85
```

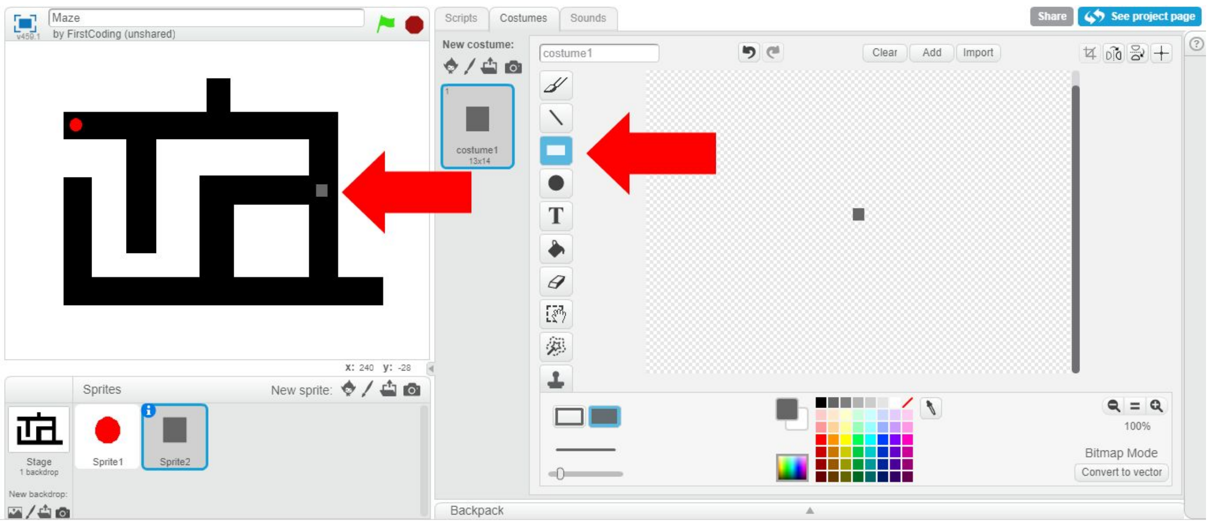
Step 4: Obstacles to avoid

To further add challenges to the game we'll create obstacles to avoid. We'll make these obstacles move around the maze on their own accord, programming them make a random decision as to what direction to turn when required.

Start by creating another new sprite by selecting the Paint New Sprite option.

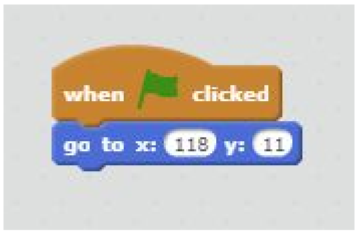


Then using the editor create a simple obstacle. It's best not to make it too big as it will have trouble navigating the maze. For this exercise we'll draw a small grey square using the Rectangle tool as shown:



Step 5: Programming the obstacles

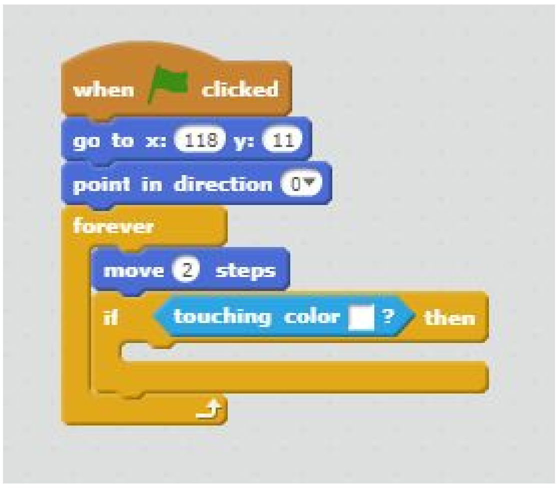
Just as before we first need to ensure that the moving obstacles start at the same position every time the green flag is clicked to start a new game. In the Scripts tab of our obstacle sprite, drag the 'When Green Flag Clicked' block along with a 'Move To' block from the Motion section as shown below:



```
when [flag] clicked
go to x: 118 y: 11
```

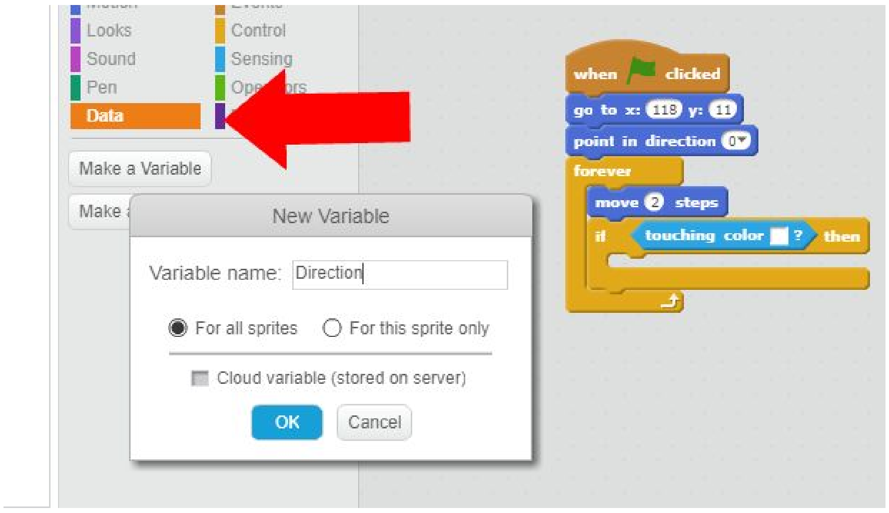
Then using a FOREVER loop we can set our obstacle on its way by moving it a few pixels at a time. Again, experiment with different values in the 'Move' block to quicken or slow its movements.

We can also incorporate another IF block to sense when we reach a wall of the maze. We do this in exactly the same way as before, by sensing for the colour white. This time however whenever the wall of the maze has been reached by our moving obstacle we'll be getting it to turn and continue on its way.



```
when [flag] clicked
go to x: 118 y: 11
point in direction 0
forever
    move 2 steps
    if touching color [ ]? then
```

In order to get our obstacle to randomly decide the way it's going to turn, we first need to create a variable. Head to the Data section as shown below and create a new variable called Direction.



New Variable

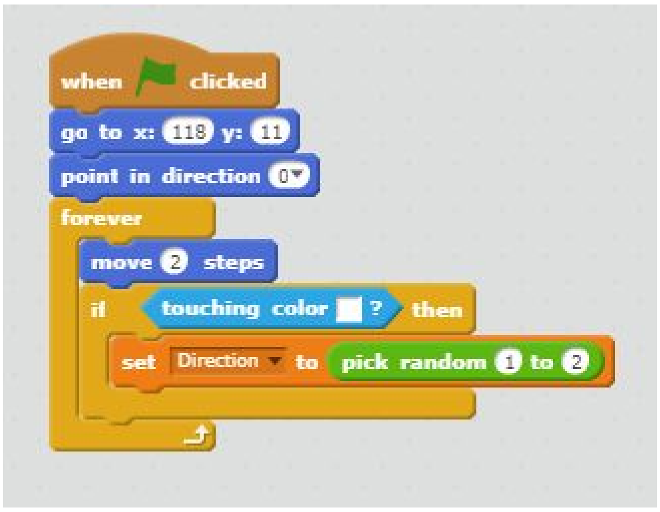
Variable name: Direction

[✓ For all sprites] For this sprite only

Cloud variable (stored on server)

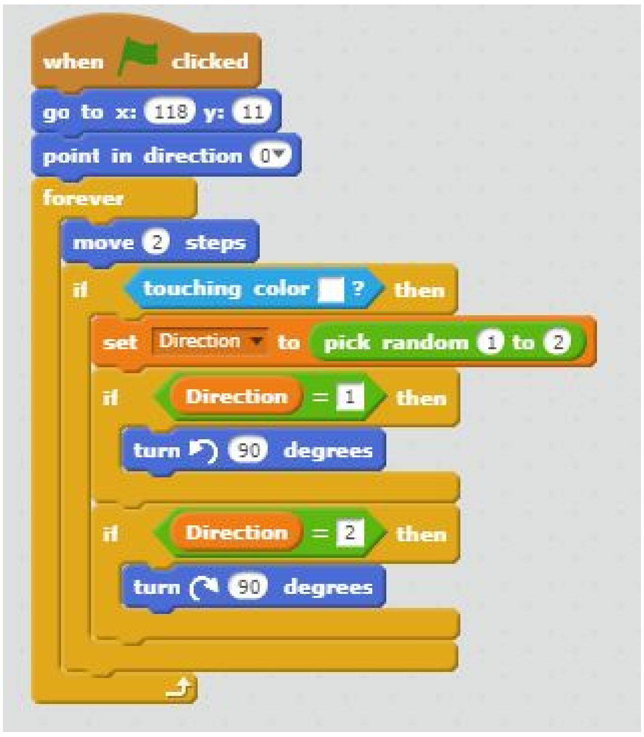
OK Cancel

We can then add our Direction variable into our IF statement by using the 'Set' block (also found in the Data section). We are going to set it to a random value between 1 and 2 and we do this by using the 'Pick Random' block found within the Operators section.



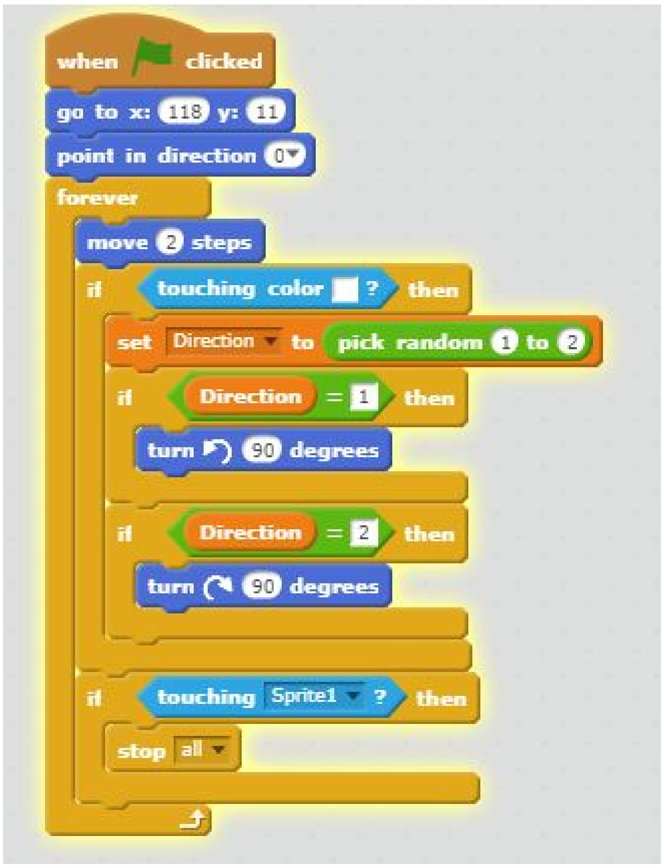
```
when 🏴 clicked
go to x: 118 y: 11
point in direction 0▾
forever
    move 2 steps
    if < touching color ⬜ ? > then
        set Direction ▾ to pick random 1 to 2
```

Setting up a couple of IF statements based on the value that our Direction variable was randomly set to.If it was set to a 1, then our obstacle turns 90 degrees to the left. If our Direction variable is set to 2, then our obstacle turns 90 degrees to the right. Experiment with different degree values in the ‘Turn’ block to see how our automated obstacle behaves.



```
when 🏴 clicked
go to x: 118 y: 11
point in direction 0▾
forever
    move 2 steps
    if < touching color ⬜ ? > then
        set Direction ▾ to pick random 1 to 2
        if < Direction = 1 > then
            turn ↺ 90 degrees
        if < Direction = 2 > then
            turn ↻ 90 degrees
```

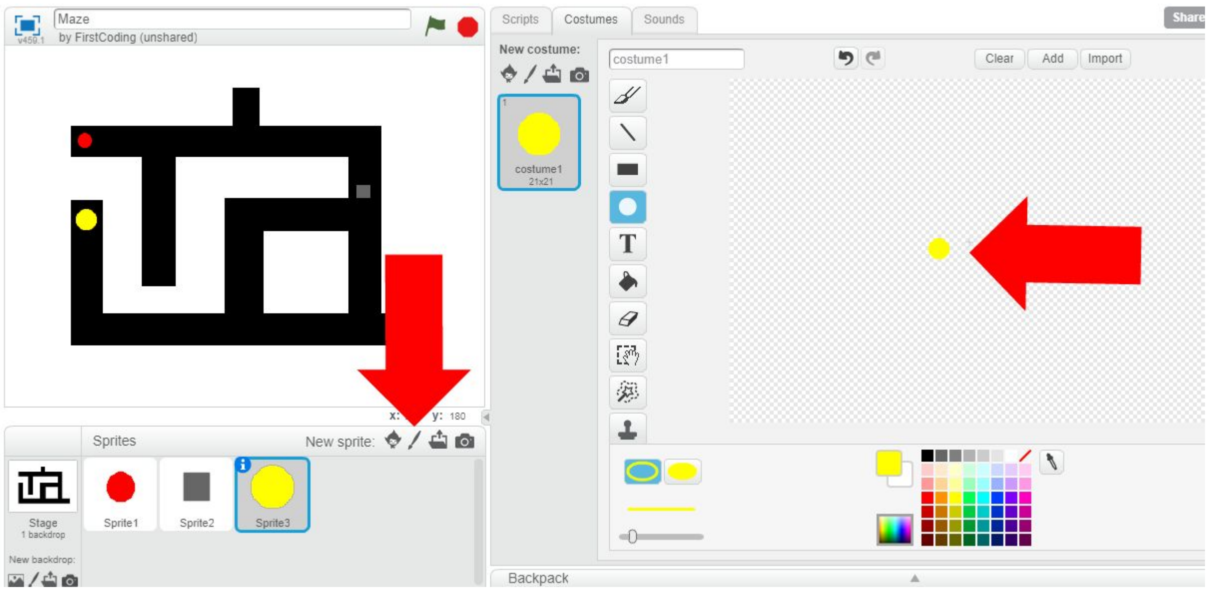
Finally we need to factor in our Game Over scenario. In order for the game to finish we need to detect when our automated block touches our player. This is achieved by another IF block whereby it checks to see if it’s collided with us. If it has then it stops the game.



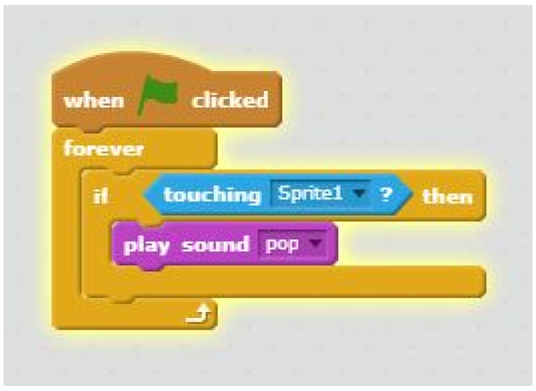
```
when ⚑ clicked
go to x: 118 y: 11
point in direction 0
forever
    move 2 steps
    if touching color ? then
        set Direction to pick random 1 to 2
        if Direction = 1 then
            turn ↺ 90 degrees
        if Direction = 2 then
            turn ↻ 90 degrees
    if touching Sprite1 ? then
        stop all
```

Step 6: Goal

The final element of our game is creating a target for us to aim for. This is once again achieved by creating a new sprite and designing something suitable in the editor on the right. Once again we're keeping things nice and simple in this case by drawing a yellow circle.



We can program the target to play a sound when we come into contact with it after navigating the maze and successfully avoiding the obstacle. This is achieved by the follow blocks:



```
when [flag] clicked
forever
    if <touching Sprite1 ?> then
        play sound pop
```

Click on the green flag and give the maze game a try. How can we make it more challenging? Here are a few ideas:

- Add more moving obstacles. We can duplicate the obstacle we've already created but remember to amend the starting coordinates for each new sprite.
- Add a timer and a leader board to record the fastest times.
- Randomly change the position of the goal when a new game starts
- Change the maze layout each time a new game starts.