MAGBOOK

BUILD A GAME IN 30 MINUTES!

Coding for Kids

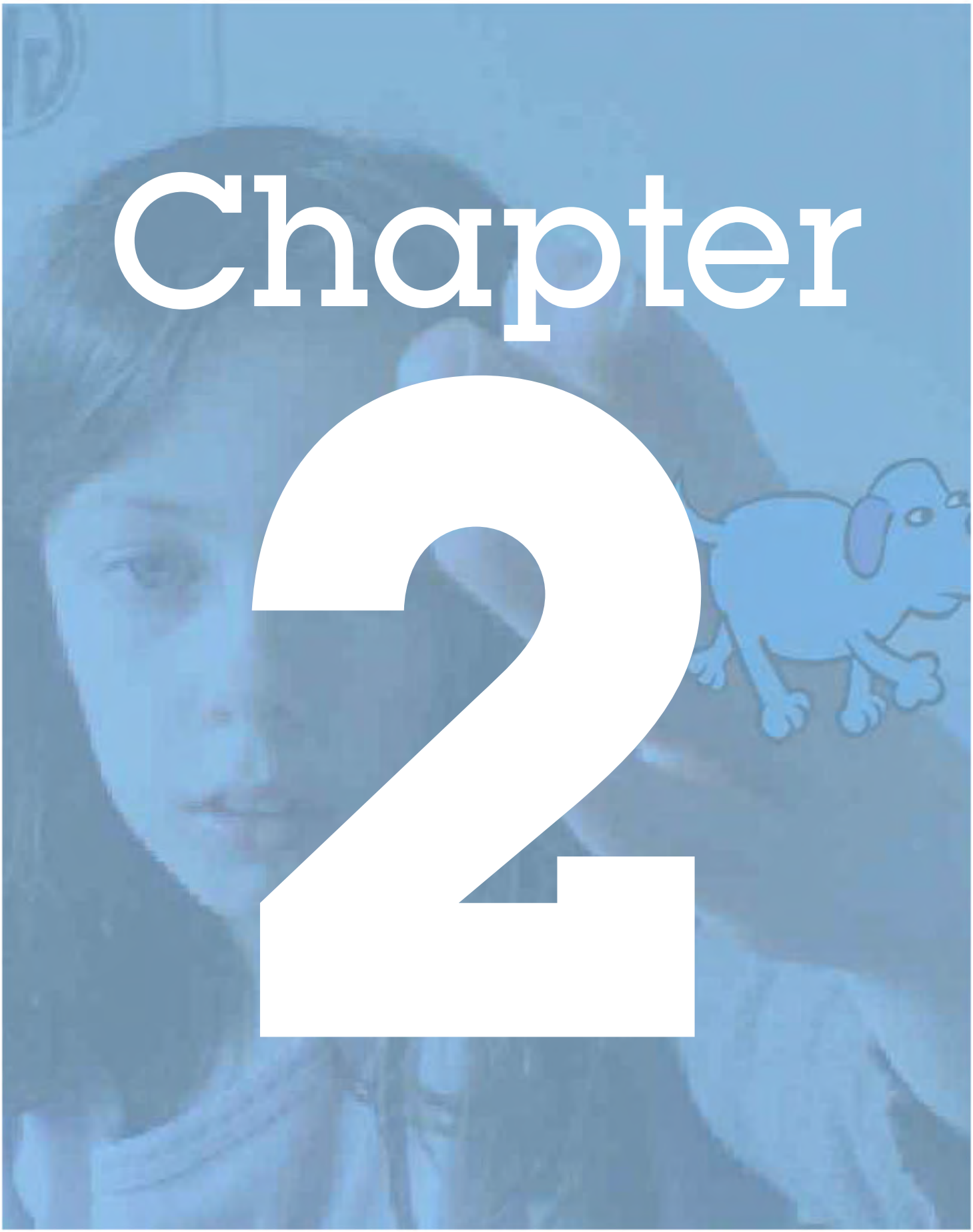
Scratch

Step-by-step friendly advice

Animate stories

Make your own racing game

Create your own quiz



Chapter 2

Contents

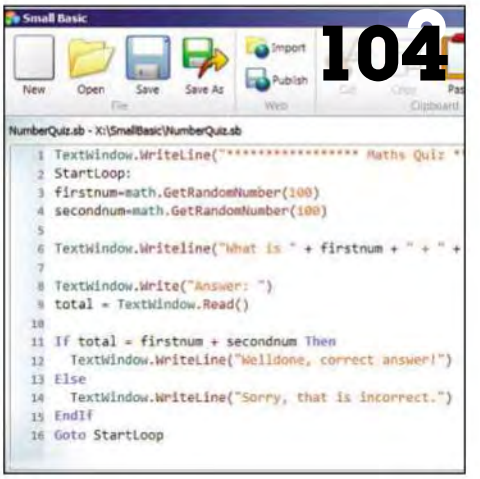
16

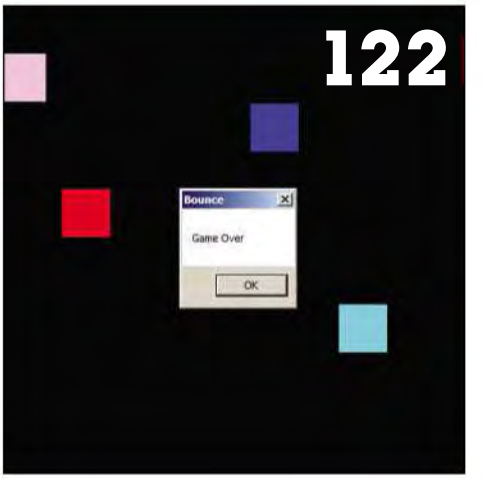
15

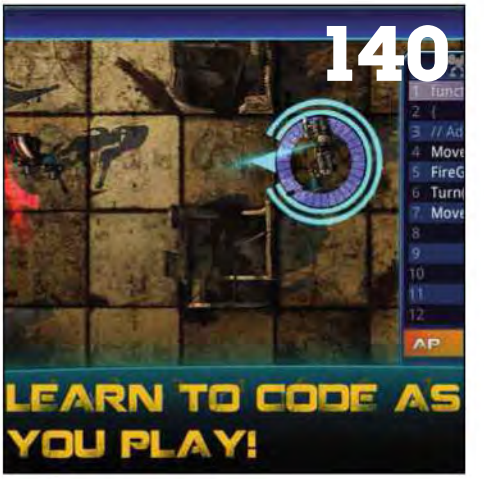
32

SECTION 1
Start coding

SECTION 2
Build your skills







Section 2
Build your skills

By now, you've encountered many of the basic building blocks of computer science, and put them to good use in simple projects. Scratch's drag-and-drop approach is ideal for these first steps into coding, but it's capable of a lot more.

In this chapter, we're going to look at some more advanced Scratch programs, using variables, one-dimensional arrays, subroutines and more. You'll discover how to do more with Scratch graphics, and how to create painting apps, racing games and other programs. Most importantly, all the knowledge you gain here will be relevant later on when we explore more complex ways to code.

Sounds difficult? It won't be. The more you learn about coding, the more logical it becomes, and before long you'll be prepared for what comes next.

IN THIS SECTION

Fun with Scratch graphics

Scratch can do a whole lot more than push spaceships and starfish around the screen. It's time to discover what other exciting features it has to offer

WHAT YOU'LL LEARN

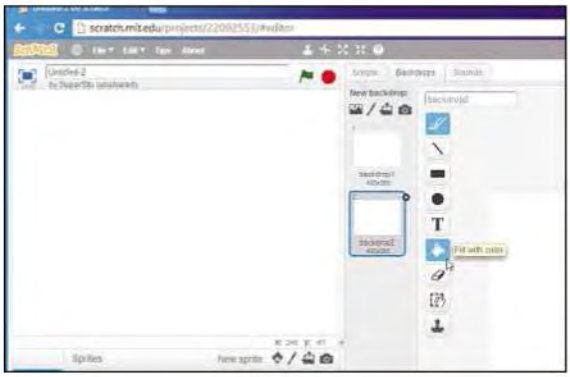
- How to use the pen in Scratch
- How to combine Operator blocks with variables
- How to take input from users, then work with it

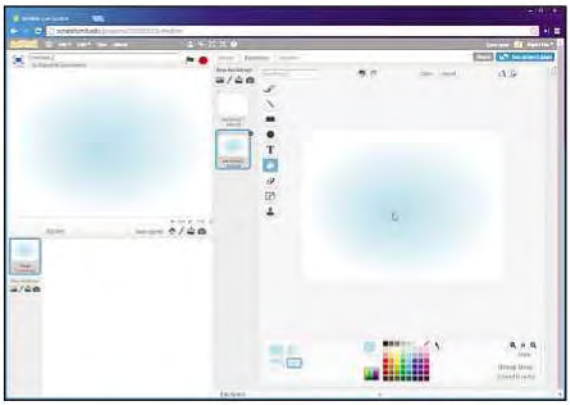
So far, we've used backgrounds, speech bubbles, thought bubbles and sprites to fill the screen, but Scratch also has another way of handling graphics: the pen. This is a really useful tool, which lets a sprite work as a pen, drawing on the screen as it moves. You can set whether the pen is drawing or not, and change the colour, shade and width of the line. What's more, you can control all these things – just like you would the sprite – using script blocks, and even use variables to alter the settings as the program runs.

Here, we're going to show how versatile the pen is by making a random pattern generator program, which takes a series of instructions, including a few random numbers, to create a whole range of fascinating, multicoloured patterns. And once we've got the random pattern generator working, we'll show you how to customise the program to give the user more control.

STEP 1

If it's there, start by deleting the Scratch cat sprite. Now, our first job is to create a really simple background. Go down to the Backdrops area next to the Sprites area and

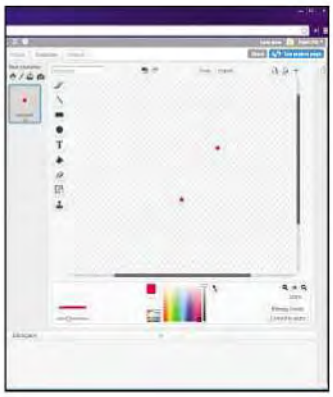




click the Paintbrush icon to 'Paint new backdrop'. Click the Paintpot tool to select 'Fill with color', then choose the round gradient fill from the group of four buttons, bottom left. Choose a nice light blue colour by clicking on it in the Colour Palette, then just click in the middle of the screen. Hey presto, one simple but attractive background.

STEP 2

Next, we need an incredibly simple sprite. Go to the Sprites area and click the Paintbrush icon there to 'Paint new sprite'. Click the Brush to select it from the toolbar on the left, then choose a bright red colour from the Colour Palette. Move the line-width slider to the left of the Colour Palette to get a slightly thicker



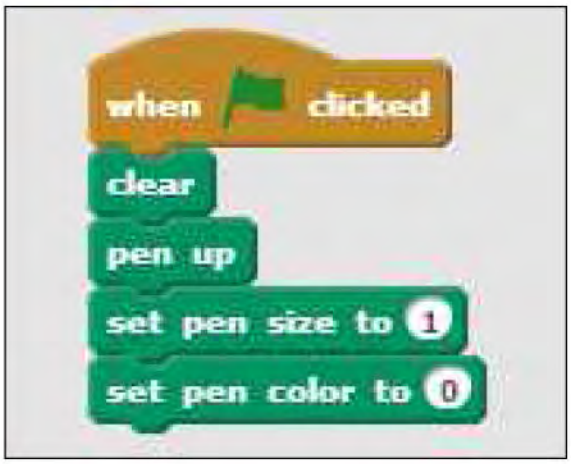
TOP TIP

None of the settings in this project are set in stone, and you can get some interesting results by messing around with them. Try altering the Pen size, the starting Angle or the starting Length in the first block of script, and see what happens.

line. Now, all you have to do is put a single, tiny dot right in the faint cross in the middle of the Painting area. Don't worry if you miss. You can either click the Clear button at the top and redo it, or the 'Set costume centre' button (the last of the three at the top right of the Costumes area) to move the cross. Either way works.

STEP 3

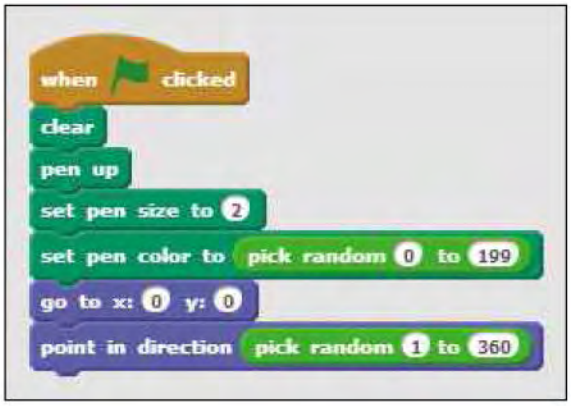
We've built our sprite. It's time to put it to work. This is another project where we can use clones to get a lot of work done very quickly, so we start off building a script to set



our master sprite up, then spawn them. First, pull in the 'when green flag clicked' block, then click on the Pen category. We need to reset the Stage and the status of the pen every time the program starts, so drag in a 'clear' block, then a 'pen up' block, then a 'set pen size to...' block and finally a 'set pen color to…' block.

STEP 4

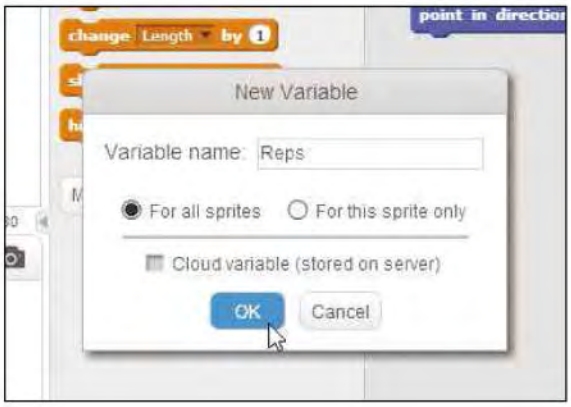
Enter 2 as the value in the 'set pen size to' block, then go to the Operators category and drag a 'pick random x to x' block to the space in the 'set pen colour to' block. Set the values to 0 and 199. This will give us a random starting colour for each pattern we generate. Now go to the Motion



category and pull in a 'go to x: x y: y' block, then a 'point in direction' block. Set the x and y coordinates for the first block to 0 and 0, then drag in another 'pick random x to x' block and drop it on the space in the 'point in direction' block. Set the values to 1 and 360.

STEP 5

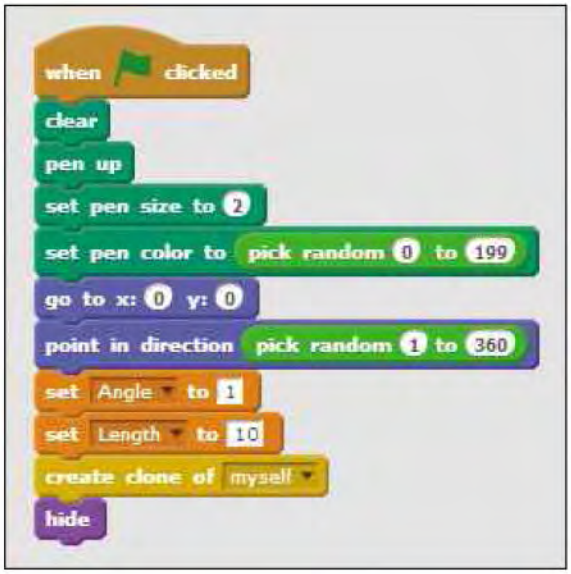
Our sprite will now start off each run in the middle of the Stage, with a random pen colour, ready to move off in a random direction. To control where it moves to next and to make sure it keeps on moving, we're going to use



variables. Click on the Data category and make three new variables, which we'll call Angle, Length and Reps. You'll see three counters appear on the screen. Click the checkbox next to each variable if you want to switch its counter off or on again.

STEP 6

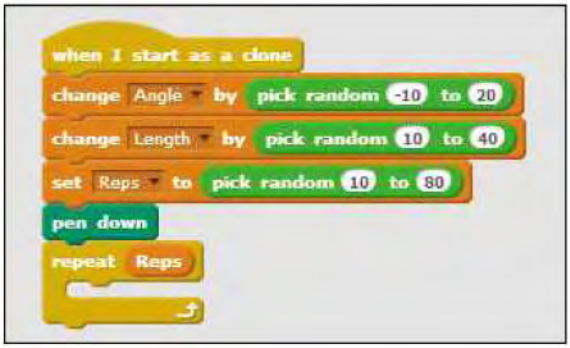
We now need to set the starting values for the first two variables. Drag in two 'set variable to x' blocks and stack them underneath the existing stack. Change the first to read 'Angle'



and 1, and the second to read 'Length' and 10. Now go to the Control category and pull in the 'create clone of' block, and make sure it's set to clone 'myself'. Finally, go to Looks and drag in a 'hide' block.

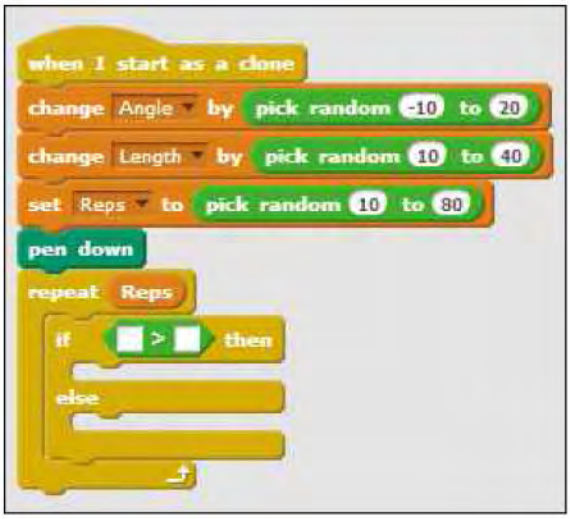
STEP 7 We now start a new stack with the 'when I start as a clone' block from Control. This block will control how the individual clones behave. To make an interesting pattern, we want each clone spawned to behave differently, so we pull in three blocks from the Data category to affect their behaviour. First, we need two 'change variable by x' blocks, then a 'set variable to x' block. Set the first block to 'Angle', the second to 'Length' and the third to 'Reps'.

STEP 8 We want each variable to change by a small amount each time, so that each clone does a slightly different thing. Go to Operators and drag in three 'pick random' blocks, placing each one carefully in the space where the value should go in our three data blocks. The Angle block needs to be set to -10 to 20. The Length block needs to be set to 10 to 40. Finally, the Reps block needs to be set to 10 to 80. When you're done, add the next block: 'pen down', from the Pen category.



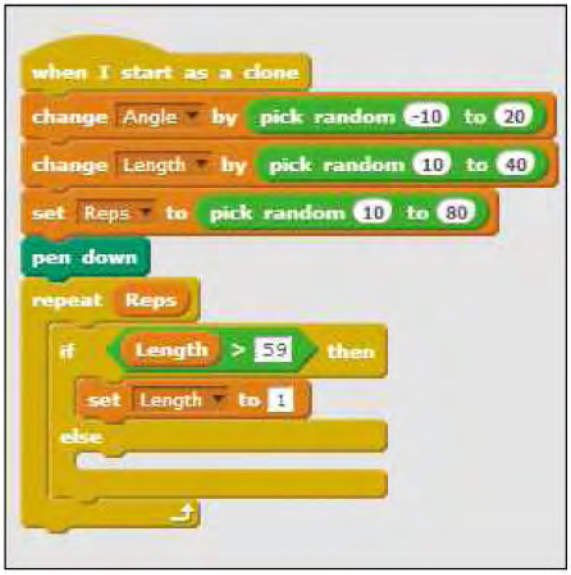
STEP 9 Next, we need to go to the Control category and pull out a 'repeat' block. Normally, we'd set this block to loop the instructions we put inside it a set number of times. This time, however, we're going to control the number of times it loops using a variable. Go to the Data category, find the block for our Reps variable, and drag this into place on the 'repeat' block.

STEP 10 To produce its patterns, our pattern generator continuously changes the length of the lines it draws. If those lines get too long, however, you just end up with a dull pattern. We control the length using an 'if, then, else' block. Drag it



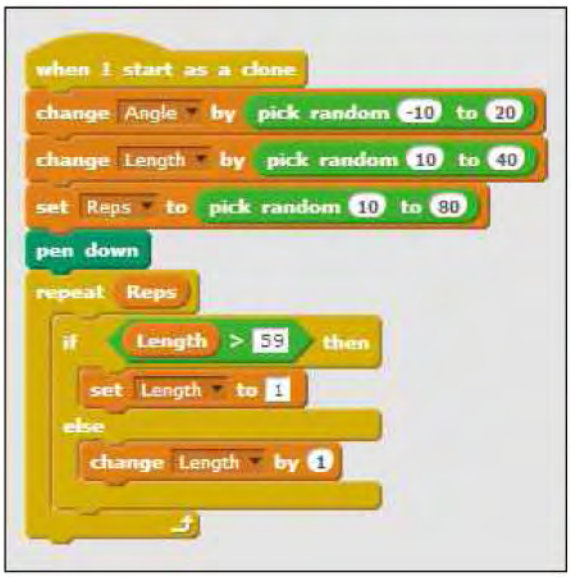
inside the C shape of the 'repeat' block, then go to the Operators and find the block that reads 'x > x'. This operator is looking to see whether the number on the left is larger than the number on the right.

STEP 11 Grab the block for the Length variable from Data, and drop it in the left-hand space. Now type 59 in the right-hand space. Next, drag the 'set variable to x' block out and drop it into the space underneath. Change the variable to Length, and change the value to 1. You're telling the



program to look at the Length variable, see if it's over 59, and, if it is, to reset it back to 1.

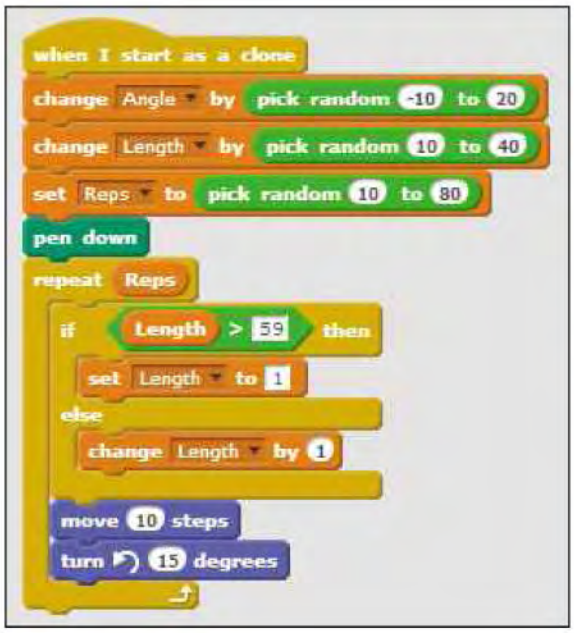
STEP 12 If it's under 59, however, we want the program to increase Length by 1. Get the 'change variable by x' block and drag it into place in the space beneath the 'else'. Set the value to 1.



STEP 13 Now it's time for the bit that tells each clone where it needs to draw. First, go to the Motion category, and drag a 'move x steps' block into the stack, making sure that it fits underneath the 'if, then, else' block, but above the bottom part of the 'repeat' block. Then, underneath it, place a 'turn anticlockwise x degrees'

CONTROLLING THE PEN

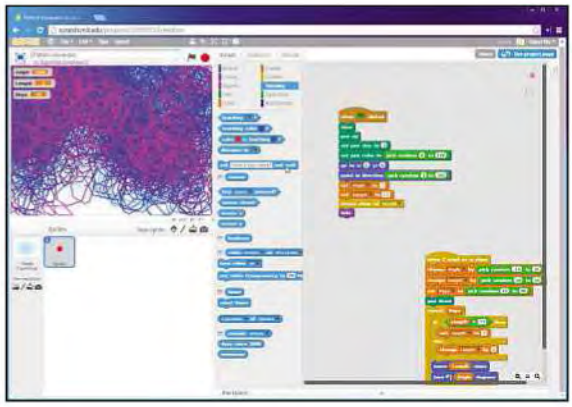
Scratch's pen lays down a line as the sprite it's attached to moves, and you can use the Pen blocks to control when the line is drawn or not, and also how it will look. Drag the 'pen down' block into a sprite, and the pen will draw a line from that point on. Drag the 'pen up' block into play, and the drawing stops. Three blocks control the pen's colour. The one with a block of colour at the end will set the pen to the currently selected colour, while the 'set pen colour to x' block, where x is a value or a variable, will set the pen to the colour associated with that number. The 'change pen colour' block can then adjust that number, moving the colour towards one end or the other of the colour spectrum. The 'set pen shade' and 'change pen shade' blocks set the pen to the brightest (highest) or lowest (darkest) shades of the colour, so that you'll get a bright blue line or a nearly black or white one, while the 'set pen size' and 'change pen size' blocks alter the width of the line to make it thinner or thicker.



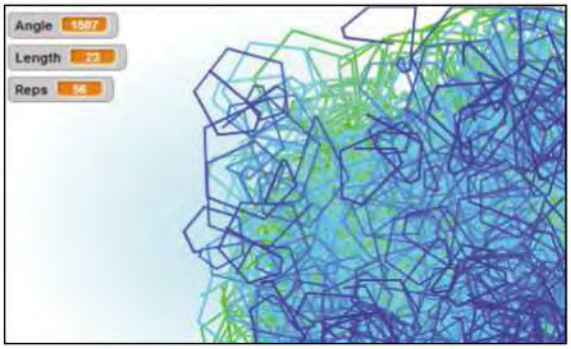
block, which is the one with the arrow pointing to the left.

STEP 14 We actually control these blocks using the variables we made earlier. Go to the Data category, and drag the Length variable, where the value should go in the 'move' block, and the Angle variable where the value should go in the 'turn' block. The program looks at the current Length, and moves that many steps in its current direction. It then looks at the current Angle, and changes its direction by that amount before moving on.

STEP 15 All that code will create just one line, so we need to keep creating clones to make more lines. Go to the Control category, and find the 'create clone of x' block. Drop it in place underneath the last two blocks, and make sure that it's set to 'myself'.



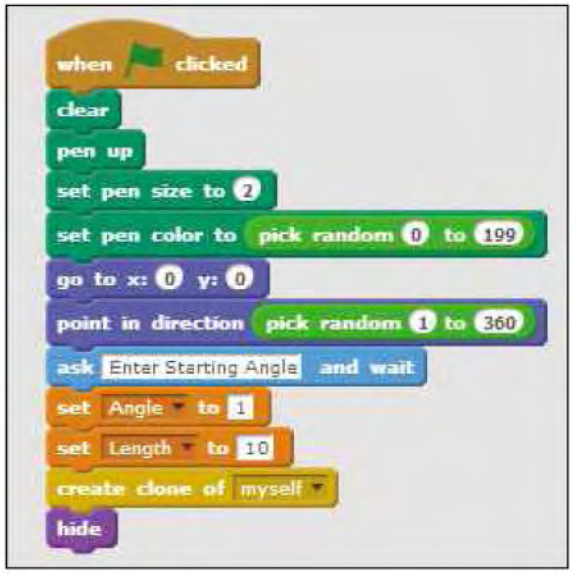
STEP 16 Now we need to make sure the line keeps changing colour. Go back to the Pens category, and find the 'change pen colour by x' block. Drag this in and drop it beneath the 'create clone' box, and you'll have a working pattern generator. Give it a spin!



STEP 17 As it is, our pattern generator is working pretty well, but wouldn't it be great if we could influence how it works and the patterns it makes? We can. There are a couple of ways of doing it, but here we're going to use Scratch's 'ask and wait' block. To make it work, we're going to need to do a bit more scripting. First, go to the Sensing category and

look for the 'ask and wait' block. By default, it has the question 'what's your name?'.

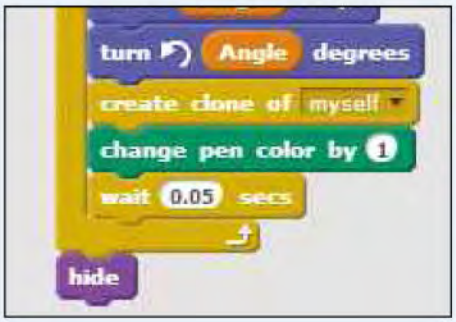
STEP 18 Drag it to the scripts area and into the 'when green flag clicked' stack, putting it just above the orange 'set Angle to 1' block. Change the text to read 'Enter Starting Angle'.



HOW THE GENERATOR WORKS

The pattern generator looks complex and does some clever things, but actually how it works is very simple. Each clone appears, draws a line of the current length in the current direction, then turns according to the current angle. It then creates a new clone of itself and changes the colour of the line. The clone then repeats that process, and the Reps variable decides how many times each clone does this before it can stop drawing, turning and cloning. As each clone ends up spawning so many other clones, you end up with lots of clones all drawing at once, but the way the Length and Angle variables are set and changed prevents the lines from going off in completely random directions. You end up with a pattern that's random enough to be unpredictable, but not so random that it's just a total mess.

OTHER IDEAS



1 You can get a better idea of what the program is doing by adding a Show block to reveal the Cone sprites moving as they draw and adding a 'wait' block to the main 'repeat' loop to slow things down a little. Put a 'hide' at the bottom of the stack so you don't spoil the finished pattern. Give it a try!

2 You can also see what happens when the shade changes instead of the colour. Simply drag out the 'change pen colour by 1' block and replace it with a 'change pen shade colour by…' block, setting it to -1 to keep getting darker, or 1 to keep getting brighter.

3 You can also try messing about with the line width. To do it, we've created a new variable called Width, and set the pen size to use that variable in the 'when I start as a clone' stack. We then use an 'if, then, else' block, just like we did with the Length earlier, to control the maximum setting. Take a look at the stack, and see if you can work out how it works.

STEP 19 Now go back to the Blocks Palette and find the 'answer' block. Drag this to the 'set angle to' block, and drop it where you can currently see the 1. When the program runs from now on, it will ask the user to 'Enter Starting Angle'. When the user types this in and presses the Enter key, the program stores what they type as a kind of special variable – the Answer. By telling the 'set angle to' block to use the answer, we tell it to look at and use whatever's stored in Answer. Only one Answer can be stored this way at any time, so we need to use it before we can ask another question.

STEP 20 We could repeat these same three steps for the Length variable, but as this already has a limit set on it, the change wouldn't make much difference to the program or the pattern. Instead, it makes more sense to change the number of repetitions. Drag in a new 'ask and wait' block below the 'set angle to' block, and change the text to Enter Repetitions.

STEP 21 Now go to the second stack – the one that starts with 'when I start as a clone' – and find the block that reads 'set Reps to pick random 10 to 80'. Drag out the 'pick random' operator, and replace it with the 'answer' block. The number of repetitions won't be random anymore, giving the user more control over the pattern. While you're there, it's worth changing the values in 'change Angle by pick random to -5 and 5'. Again, this makes it easier to influence the finished pattern. Try running the program with different combinations of Angle and Repetitions to see what happens. ●

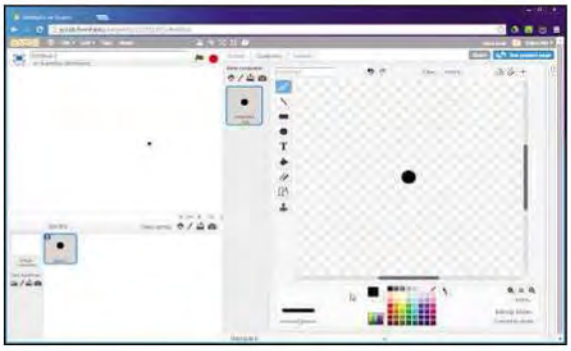
Paint with Scratch

In this project, we're going to push Scratch further than we ever have before by creating our very own Scratch painting program

WHAT YOU'LL LEARN

- How to create sliders to control variables
- How to use sprites as buttons
- How to 'stamp' sprites with the pen
- How to transfer costumes from sprite to sprite

Scratch gets used a lot to make games, quizzes and funny cartoon animations, but with a bit of thought it can be used to make almost any kind of program. We're going to prove that with this next project by building a simple painting program, complete with basic tools to paint lines, a choice of different colours, and even tools to stamp and paint with pictures on the page. It might sound like a lot of hard work, but with Scratch's built-in features it's easier than you'd think.



STEP 1

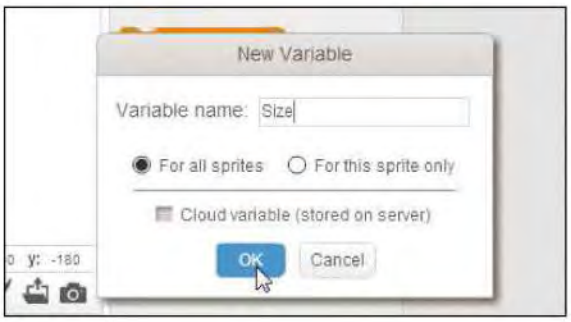
Start off by deleting the Cat sprite, then create a new Dot sprite, just like we did in the last project. Click the 'Paint new sprite' button at the top of the Sprites area. When the Costume Editor loads, increase the line width to the halfway mark, then stick a small blob of paint smack in the centre of the crosshairs. It helps if you press the magnifying glass to zoom into 400%.

STEP 2

That's our main sprite done, so click on the Scripts tab and let's get scripting. First, we're going to make our pointer work like a paintbrush. Our stack kicks off with the 'when green flag clicked' block. Then, go to the Pen category and drag out the 'clear' block. This clears the screen every time our paint program is run. Now go to Control, and drag a 'forever' block into place. If you remember, this block tells the program to keep on doing whatever instructions we put inside it for as long as this stack is running.

STEP 3

Before we go on, we need to create some variables. We're going to use these to control the colour and shade of the paint our brush lays down, as well as the size of the brush. For each one, go to the Data category and click the Make a Variable button, then enter the name of the variable, and set it to work



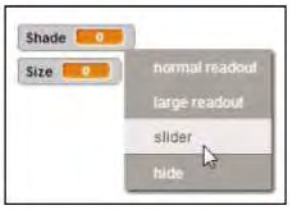
TOP TIP

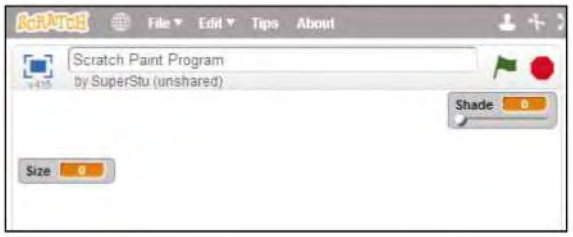
The little black sprite sometimes leaves marks on the screen when painting. If it's annoying you, just add a Hide block to the end of the first 'when green flag clicked' stack.

'For all sprites'. Call the three variables Colour, Shade and Size. You can leave the checkboxes next to Shade and Size ticked, but untick the one next to Colour.

STEP 4

Go over to the Stage area, and right-click on the Shade counter. Select Slider from the dropdown menu that appears, and your Shade

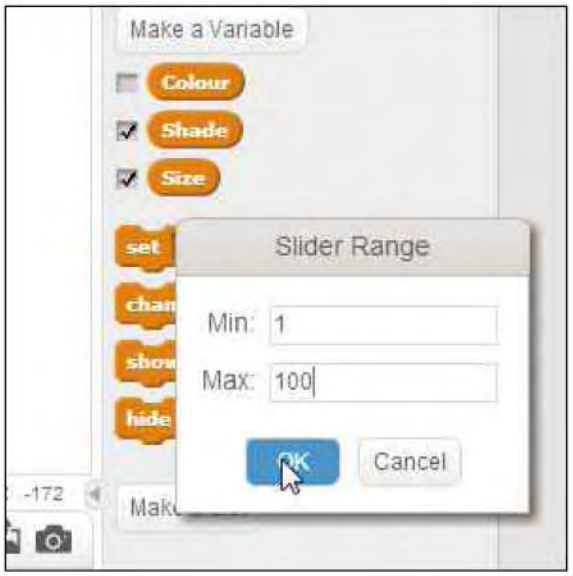




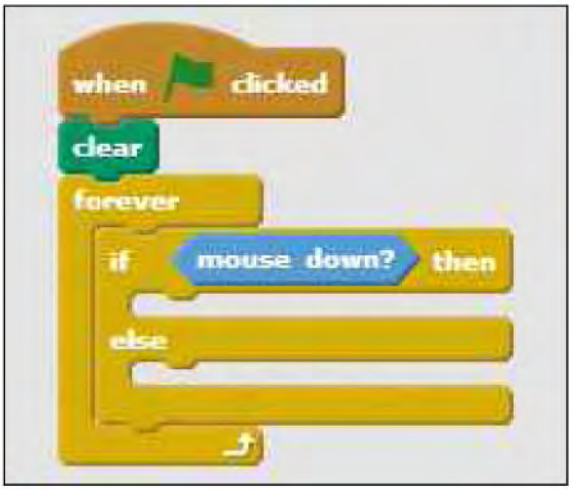
counter turns into a slider, where you can move the slider left and right to alter the value of the Shade variable. You can try it right now if you like, but the really cool thing is that anyone using the program will be able to do the same thing. Now drag the new Shade slider over to the top-right corner of the screen.

STEP 5

Do the exact same thing with the Size counter, then drag the new slider into place beneath the Shade slider. Now right-click again on the Shade slider, and select 'Set



slider min and max' from the menu. Enter 1 as the minimum and 100 as the maximum. Do the same with the Size slider, and enter 1 as the minimum and 20 as the maximum.

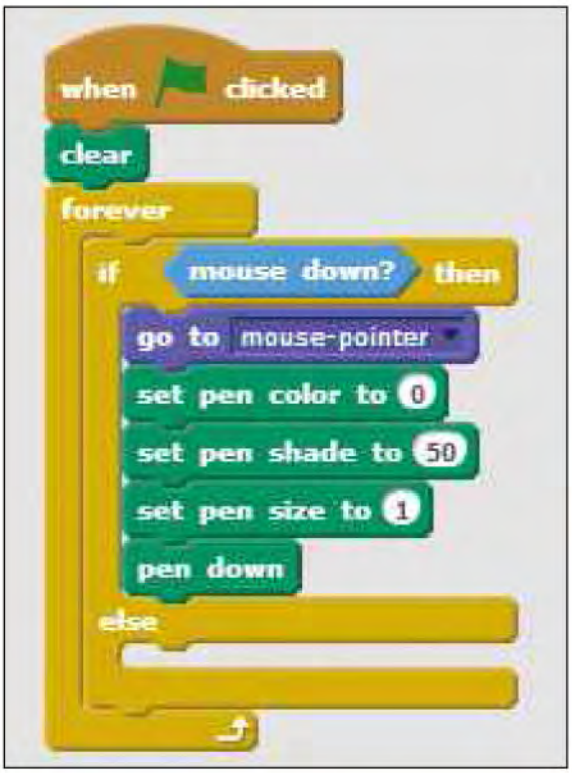


STEP 6

Right, let's get back to our script. Go to Control and find the 'if, then, else' block, and drag it into place inside the C shape of the 'forever' block. Now go to the Sensing category, find the 'mouse down?' block, and drag it into the space after the 'if'.

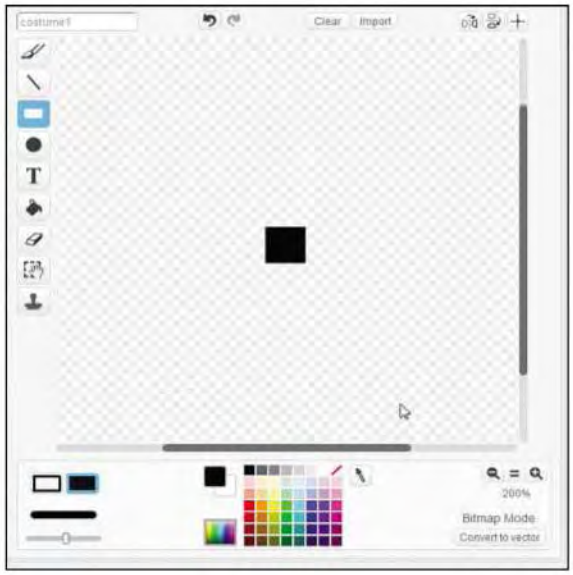
STEP 7

We've set this script to keep an eye on the mouse button, and see if it's being pressed or not. Now we need to tell it what to do if the mouse button is pressed. First, go to Motion, and drag in the 'go to x' block. It should be set to



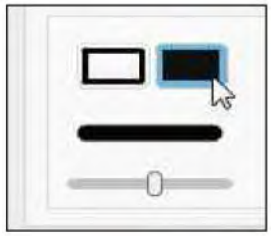
mouse-pointer already. Now go to the Pen category, and drag in the 'set pen color to', 'set pen shade to' and 'set pen size to' blocks, followed by the 'pen down' block.

STEP 8 Instead of setting these blocks manually, we use our variables. Go to the Data category and drag the Colour variable to the value in the 'set pen color to' block. Do the same with the Shade variable and the 'set pen shade to' block, and the Size variable with the 'set pen size to' block.



STEP 9 When the mouse button is pressed, our sprite will now draw a line using our current settings for Shade, Size and Colour. And if it isn't, drag the 'pen up' block from the Pen category and place it inside the space underneath the 'else'. Give the script so far a try. You'll be able to draw a black line by clicking and holding the mouse button as you move the mouse around the screen. You can even change the width of the line by sliding the Size slider up and down.

STEP 10 It works, but wouldn't you like some colour? We can fix that by making a new sprite. Click the 'Paint new sprite' button and use the Rectangle tool to draw a small square over the crosshair. You can either fill it with the 'Fill with color' tool or click on the filled shape option in the bottom right of the Costume Editor. Remember, you can always use the Set Costume Centre tool later if you don't quite get it right.



STEP 11 Click on the Scripts tab, and we can start scripting what this sprite will do. First, we need to reposition it, so grab a 'when green flag clicked' block, then attach a 'go to x: x y: y' block to it. Set x to -220 and y to 160.

STEP 12 Now create a second stack, starting with a 'when this sprite clicked' block. Once that's in place, go to the Data category and pull in two 'set variable to x' blocks. Set the first one to 'Colour' and 0, and the second one to 'Shade' and 0. In effect, we've turned this sprite into a button. Click on it, and it turns the colour of the line to black.

TOP TIP

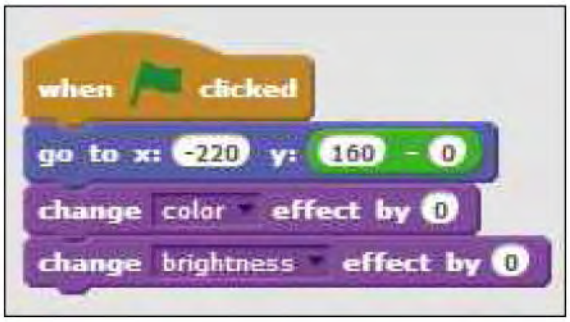
When you're duplicating sprites, try renaming them. It makes things a whole lot easier for you if they look similar, and it can also help anyone remixing your project later on.

STEP 13

That's great, but what about the colours? Well, we could keep making a new sprite for each colour and create a brand-new script for each, but why work when a little scripting will do the work for us? We can keep duplicating this sprite, and make a few changes for each version. Before we duplicate anything, look at the 'when Green flag clicked' sprite. Go to the Operators category and drag the 'x – x' block into the 'Go to x: x y: y' block, where the 160 is at the moment. Set the first value to 160 and the second to 0.

STEP 14

Now go to the Looks category and pull in the 'change x effect by x' block twice. Set the first one to 'color' and set the value to 0, and the second one to 'brightness' and the value



to 0. We don't actually want these two blocks to affect this sprite, but they make things a lot easier once we start duplicating it. Now right-click on the sprite in the Sprites area and select Duplicate.

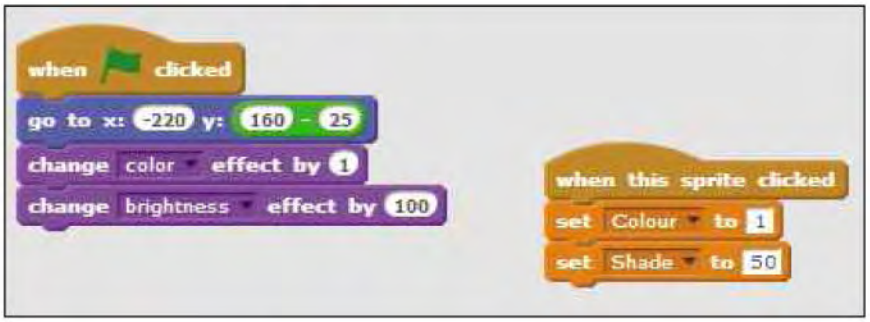
STEP 15

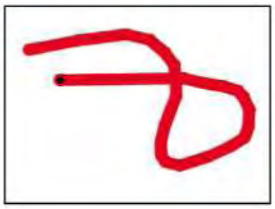
Just click on the new Sprite to highlight it, then it's time to adjust the script. By changing the second value in the Operator block in the 'go to x: x y: y' block, you can quickly change the vertical position of the sprite. Setting it to 25 works well in our example. Now you need to change the colour

of the sprite, by changing the settings for the Color and the Brightness effects. Here, we want a red, so set the Color effect to change by 1, and the Brightness effect to change by 100.

STEP 16

Now we just need to go over to the 'when this sprite clicked' stack and set it up to match. Set the Color to 1, then set the Shade to 50. Now, when you run the script, the



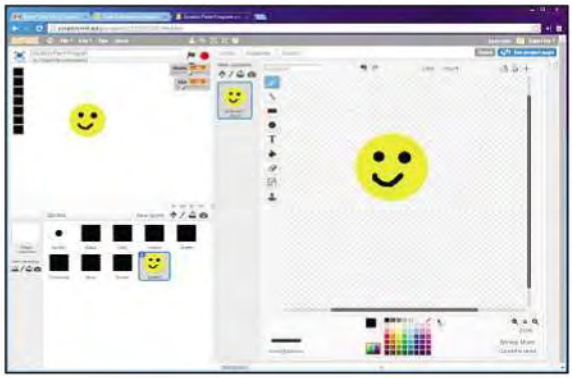


bottom of our two black square sprites will now look red, and produce a red line when clicked.

STEP 17

The beauty of doing things this way is that you can keep on duplicating the last sprite, then change the values to alter its colour, and the colour of the line you'll get once you click on it. Change the value of the y: position, increasing it by 25 each time, then change the Color and Brightness effects to bring in a new colour. Finally, set the Colour variable in the second stack to match. It's best to leave

the Shade variable set at 50, and the Brightness effect at 100. To create this little palette, we've set the rest of the colours to 30, 50, 90, 120 and 150. You can always choose different colours, or duplicate and change as many sprites as you have room for.



STEP 18 It's not just lines you can paint, however. You can also use the Stamp block to paint with the sprites themselves. First, you need to add a new sprite, either by choosing one from the library or by painting one yourself with the aid of the Costumes Editor. Here's one we made earlier.

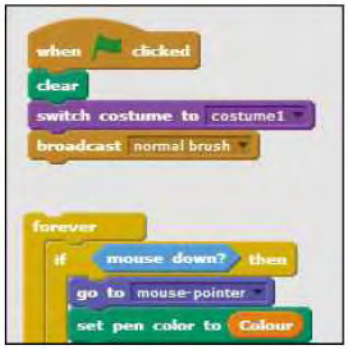
STEP 19 Now right-click on the costume in the list just to the left of the Costume Editor and select Duplicate. Drag the duplicate over Sprite 1 where it sits in the Sprites area, then release the mouse button. This copies and transfers this costume

over to Sprite 1 – your main painting sprite. You'll need it later.

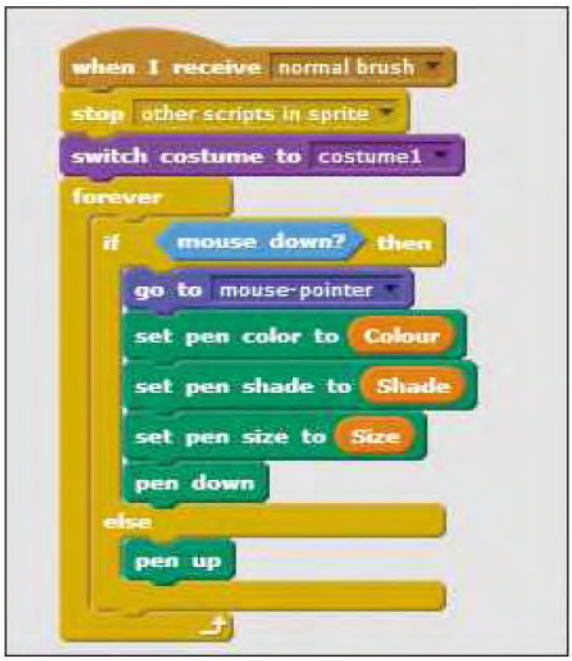
STEP 20 Click the Scripts tab. This sprite needs two stacks of script. The first, which starts with 'when green flag clicked', changes the size of the sprite then moves it so that it fits neatly underneath your Colour Palette. You might need to try different values here before you get it right. The second tells the program to broadcast a message when this sprite

is clicked. Here, we're calling it 'smiley'. Again, you're transforming a sprite into a button.

STEP 21 But what happens when that button is clicked? Well, go back to the first sprite you made. We need to make some changes to its script. Click on the 'forever' block underneath the 'when green flag clicked' block and drag the rest of the stack away from the top two blocks. Now you need to add two more blocks to the top stack: a 'switch costume to' block and a 'broadcast message' block. The costume should be set to 'costume1', and you need a new message, which we'll call 'normal brush'.



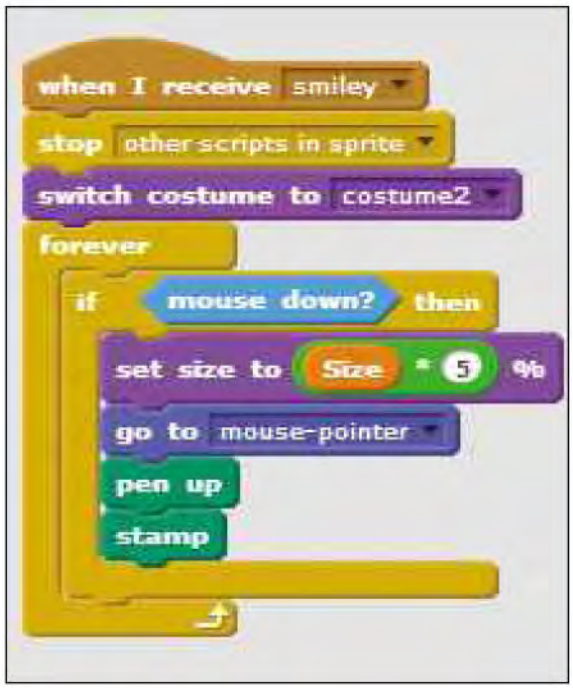
STEP 22 Now get a 'switch costume to' block and place it on top of the rest of the stack. Change the costume to read 'costume1'. Next, get a 'stop' block from Control, set it to



TOP TIP
Sometimes Scratch misbehaves itself, and the sprites you're creating won't appear on the Stage. If this happens, save the project and refresh your browser window. This usually sets things right.

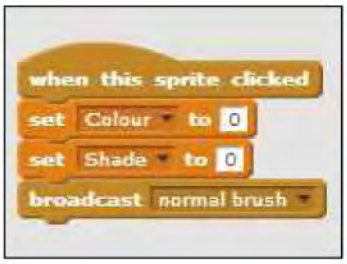
'other scripts in sprite', and place that on top. Finally, get the 'when I receive message' block from the Events category, and click it into place on top. Change the message to 'normal brush'. This script now changes costume to the first costume and draws lines when it gets the normal brush message, just like it did before.

STEP 23 Start a new stack with the 'when I receive message' block, but this time change the message to read 'smiley'. The stack underneath is a variation on the one we've

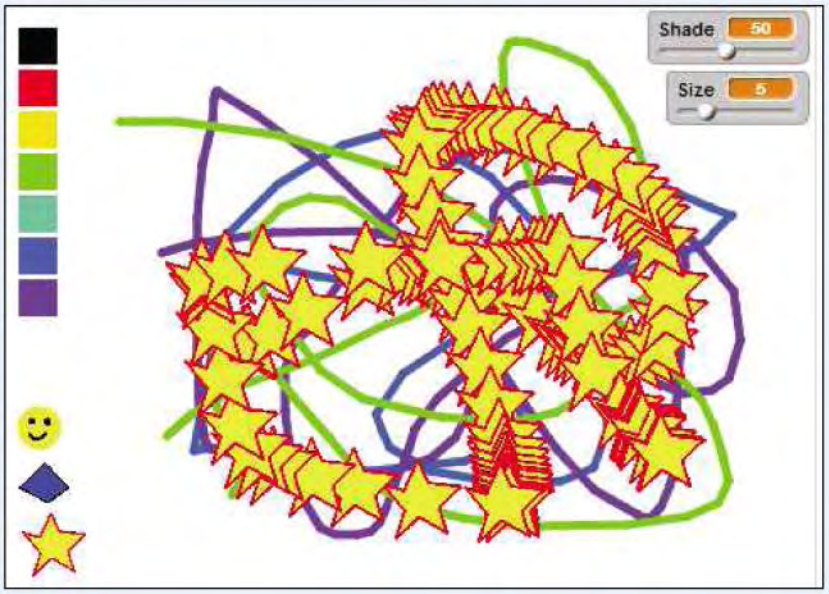


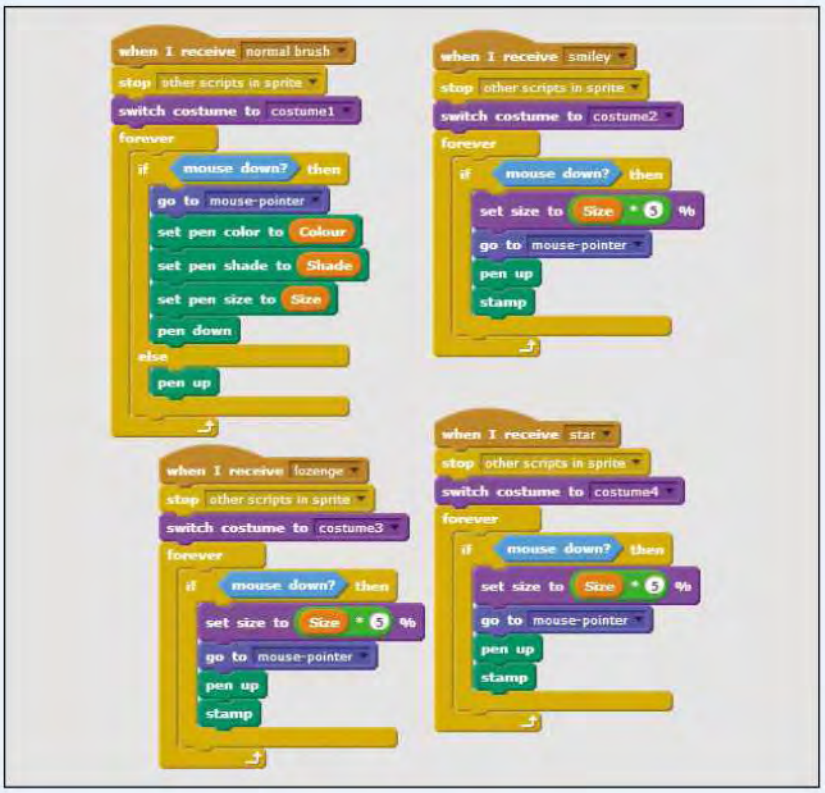
already created to draw lines, but this time it uses the 'stamp' block. This places a copy of the current costume everywhere you click, or draws with the costume if you hold down the mouse button. Note the operator with the Size variable in the 'set size to' block. This allows the Size slider to control the size of the stamp, but adjusts for the fact that the 'set size to' block works differently to the 'set pen size to' block.

STEP 24 Finally, you need to go through each sprite that you use in the palette and add a 'broadcast message' block at the end of the 'when this sprite clicked' stack. Set the message to 'normal brush' and you're all set to draw away. ●



MESS AROUND





Why stop at one sprite? You can keep adding or painting new sprites and using them to paint with by following steps 18 to 24. You can even drag or duplicate scripts from one sprite to another to save time, adjusting whatever settings and message names need to be altered. Why not try different shapes and patterns, and see what works and what doesn't?

My Scratch racing game

Putting a car on the track is the easy part of this project, but to put it up against some rival racers we'll need to make use of Scratch's lists

WHAT YOU'LL LEARN

- How to use Motion blocks and variables to move a car around a track
- How to control how a sprite behaves with colour
- How to time things in Scratch
- How to make and use lists

This top-down racing game seems pretty simple, with just a few sprites racing around a flat 2D track. In fact, there's a little more to it. Most racing games feature some kind of opposition, which usually means coding in artificial intelligence routines to control where they drive, how fast they go, and what they do when they're near to the player's car. This is the sort of problem professional game developers face every day, but with Scratch it's different. Not only are these routines very difficult to code, but the code required might slow Scratch down to a crawl.

That's why this racing game – we'll call it Ghost Racer – cheats. Instead of controlling the rival racers, it memorises how the player races, then puts cars on the track that follow the same patterns. It all works thanks to lists, which store the player's direction and speed in memory, then recall that information to control three 'ghost' cars.

This is going to be a pretty complex project, and before you tackle any complex project you should jot down what you need to do, as this will help you structure the program. In this case, we need to:

1. Build a racetrack
2. Put a car on the track
3. Create controls for how the car will speed up, slow down and steer on the track
4. Define what will happen when the car comes off the track
5. Track lap times
6. Let the player know when they have a new record
7. Create lists to capture what the car does
8. Put our ghost racers on the track
9. Use the information stored in the lists to keep those racers on the track
10. Define what happens when our car hits a ghost racer

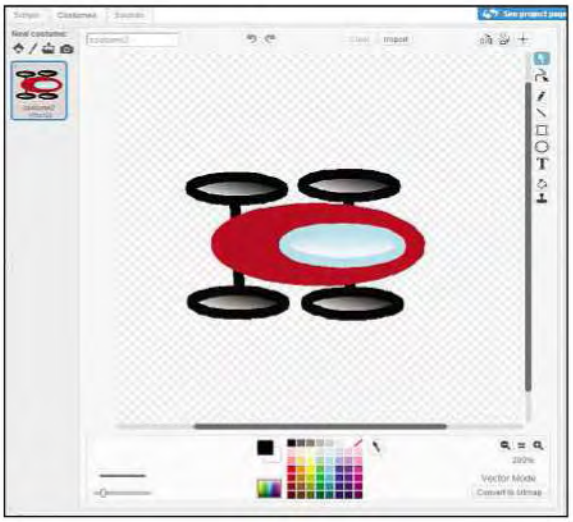
That's quite a lot to be getting on with, so let's get started.

STEP 1

Before we do anything else, we need a track. You can paint this yourself by clicking on the 'Paint new backdrop' button and using the Backdrop Editor. Use the Paintcan tool to fill the whole space with green, then paint on a black road with a nice, thick, black brush. Set the size of the brush to maximum, and paint your track straight on to the green background.

STEP 2

Next, we need a sprite for our car, Why not click on the 'Paint new sprite' button and make your own? Here's one we've made from a series of ellipses, but you can make anything you fancy. Just have fun, and let your imagination run wild.



STEP 3

Before we can put our car on the track, we need to create some variables. Go to Data, and create two new ones. Have them marked 'For all sprites', and call them 'lap count' and 'speed'.

STEP 4

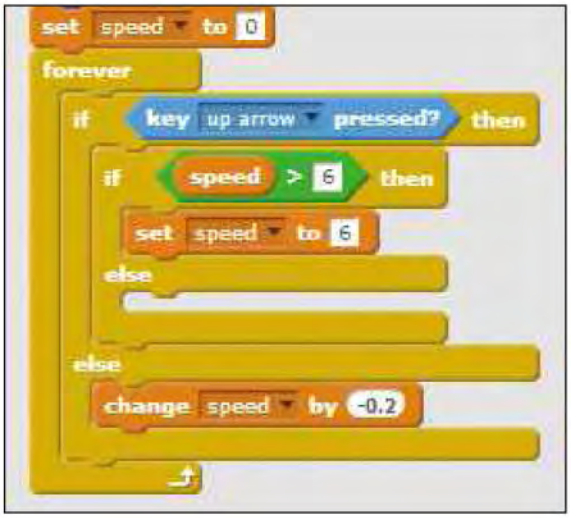
Now we start assembling our main control stack for the player's car. This first set of blocks changes the car sprite to a suitable size for our course (10%), puts it in the right place (x: 0, y: 98), pointing it in the right direction (-90), and sets the Speed variable to 0. Our car is in position – it just needs some code to make it go.

STEP 5

Start by pulling in a Forever loop and adding it to the bottom of the stack. Inside that goes an 'if, then, else' block. Drag a 'key x pressed?' block from Sensing into the space next to the 'if', and set the key to the up arrow. Now grab a 'change variable by x' block from Data, and pull this into the space underneath the 'else'. Set the variable to 'speed' and the value to -0.2. This tells the car to slow down gradually if the up arrow key isn't being pressed.

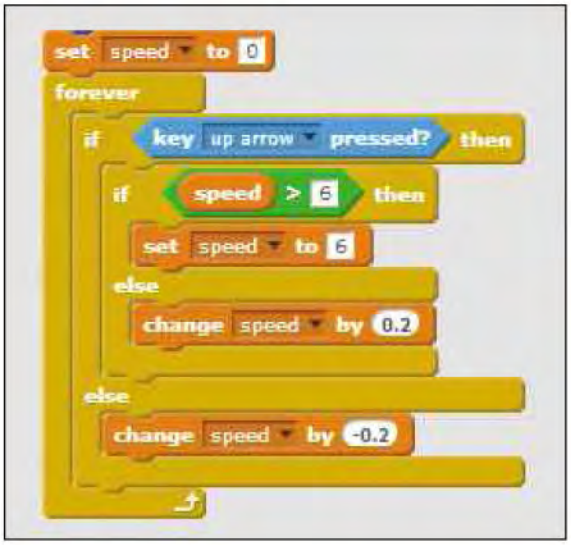
STEP 6

If it's being pressed, we want the car to accelerate, but if it goes too fast, players won't be able to control it. That's why we now drag in another 'if, then, else' block and nest it inside the 'if' section of the last one. For the 'if' condition, we use the 'x > x' operator block and the Speed variable to tell the program that if the Speed variable is higher than 6 it needs to keep speed set to 6. Whatever happens, the speed can't creep up above this level.



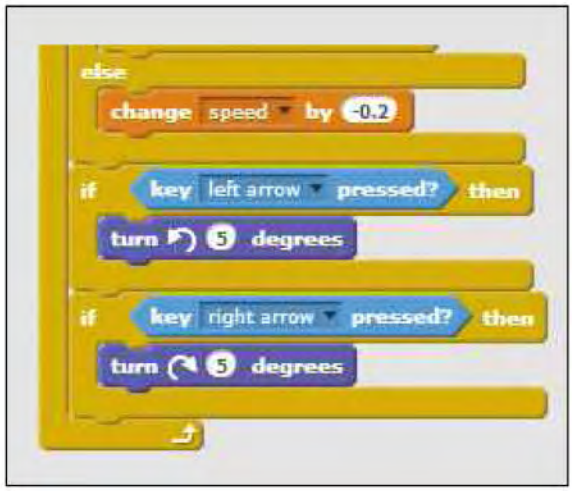
STEP 7

Next, add a 'change variable by x' block into the space below the 'else', and set the variable to 'speed' and the value to 0.2. If the current value of speed is below 6 and the up arrow key is being pressed, the value will go up by 0.2. When we've done a little more work, the Speed variable will control the speed of the car, so it will steadily get faster.



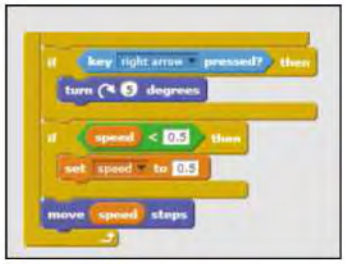
STEP 8

What it can't do, however, is steer left or right. Let's fix that. We add two 'if, then' blocks to the bottom of the stack inside the 'forever' loop. The top one uses the 'key x pressed?' block from Sensing to check whether the left arrow key is being pressed, and, if it is, use the 'turn anti-clockwise x degrees' block from Motion to turn 5 degrees anti-clockwise. The bottom 'if, then' block does the same with the right arrow key, but with the 'turn clockwise x degrees' block. Our car can now steer left and right.



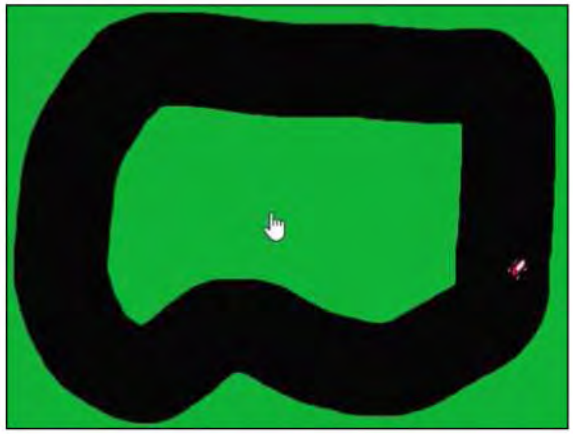
STEP 9

Now we just need to add a few more blocks. The 'if, then' block that goes in next uses the 'x < x' operator to see if the Speed variable is lower than 0.5. If it is, it sets it to 0.5. This stops our car from going backwards if the up arrow key isn't pressed. The next block, the 'move speed steps' block, simply tells the car to move forward in its current direction by the current value of speed. If speed is 6, it will move forwards 6 steps. If it's set to 3, it will move forwards 3 steps. Click the green flag above the Stage, and you can take your car sprite for a spin.



STEP 10

That's got the car moving, but we've still got a problem. At the moment, the car travels just as fast whether it's on the track or not. We can, however, fix this using an 'if, then, else' block and a special Sensing block called 'touching x color?'. Drag the 'touching x color?' block into the space next to the 'if', and click on the little colour square. The normal pointer should change into a hand with a pointy finger. Move the hand over the green grass area of the backdrop and click it. The little square should now match the colour. Now drag a 'change variable by x' block into the space underneath, and set the variable to 'speed' and the value to -0.2.

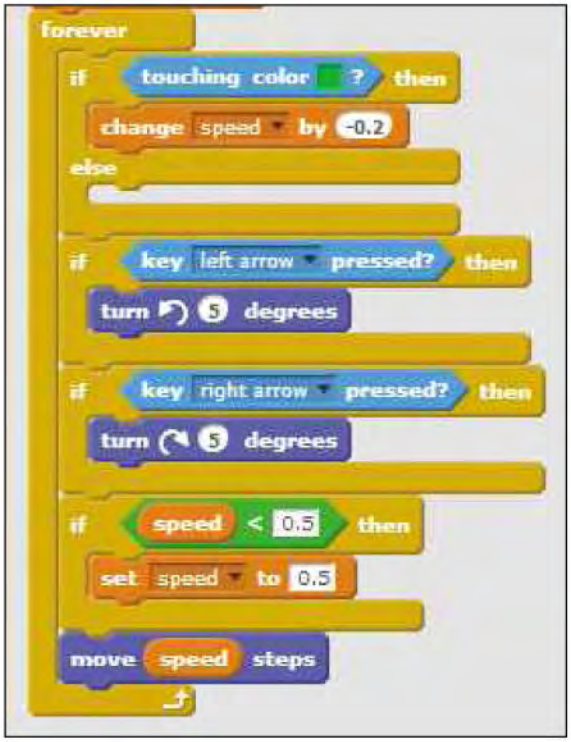


STEP 11

Now it's time for a little bit of rearranging. Click and hold on the 'if key up arrow pressed?' block at the top of the stack inside the 'forever' block, and drag it out of place. Next, drag your new 'if touching x color?' block and place it where the rest of the stack used to be.

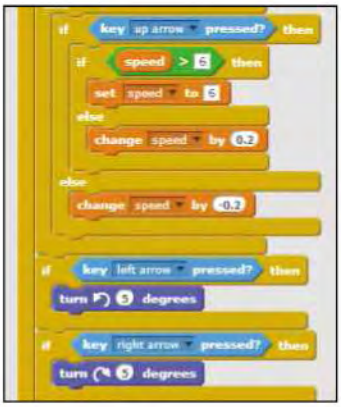
ABOUT LISTS

Lists work a bit like variables, but they enable us to store, then use several different bits of data at once. You can use them to hold different pieces of text, like a list of names or a list of replies that a user makes when we ask them to type something in, or we can use them to hold numbers, like a list of answers to sums or a list of coordinates in a game. All we need to do is tell the program where to store the information – as, say, item1 in 'my shopping list' – then we can use the same details to retrieve the information when we need it. In regular programming languages, lists are normally called 'arrays'.

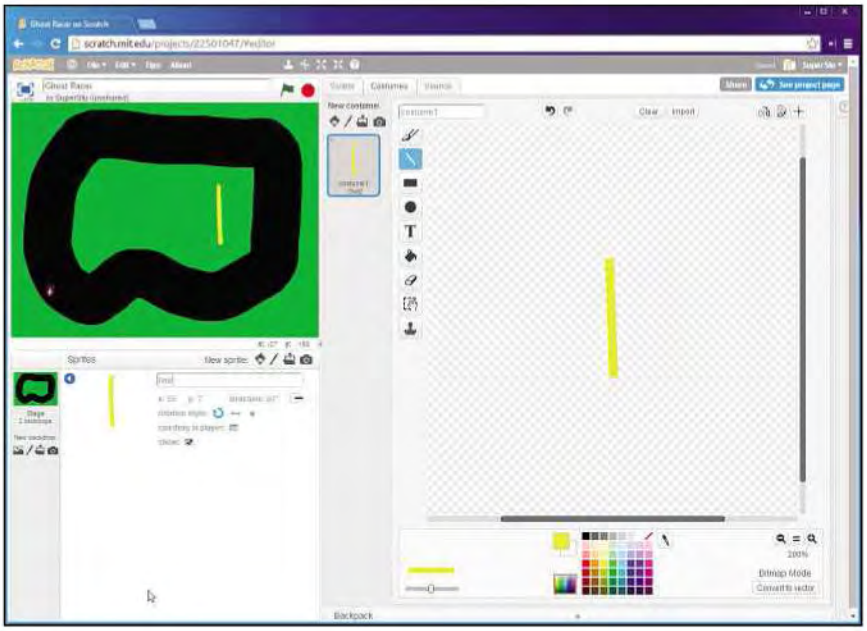


STEP 12 Now grab and drag the 'if key left arrow pressed?' block in the bottom half of your stack away from the blocks above. You need to place this inside the 'forever' block, but outside the 'if, then, else' block.

STEP 13 Finally, grab the rest of the stack and put it in position inside the space beneath the 'else'. You now have a car that accelerates, slows down and steers, and that will drive faster on the track than it will on the surrounding grass. Go on, give it a try!

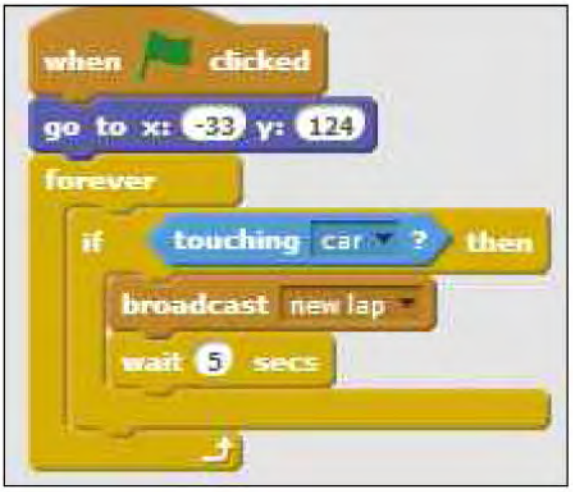


STEP 14 We have a car that moves, but we don't really have a game. Let's say that the goal in Ghost Racer is to get the lowest lap times. We need a way of tracking when the car completes a lap, and how long it's taken to do it. To do the first bit, we're going to create a new sprite. Just use the 'Paint new sprite' button and the Line tool to draw a yellow line going vertically down the screen. You might

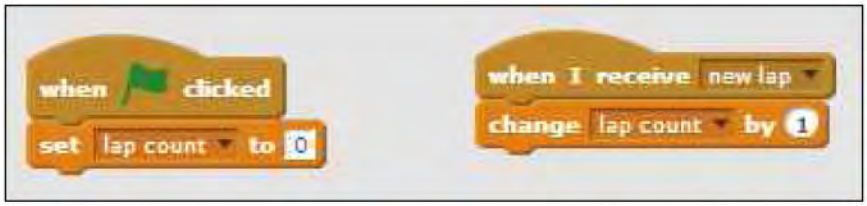


want to rename it 'line' and name our car sprite 'car' to make things easier later on.

STEP 15 Once it's finished, grab and drag the sprite into position so that it's just ahead of the car. Now click the Scripts tab. The line sprite's script is pretty simple. We use a 'go to x: y:' block to set its position. The block should have the right values in when you drag it out. The 'if, then' block uses a 'touching x?' operator to see if the line sprite is touching



the car sprite, and if it does it will broadcast a new message, which we call 'new lap'. The 'wait 5 secs' block makes sure the new lap message isn't triggered several times as the car crosses the line. Finally, the 'forever' block makes sure that the program is always looking to see whether the car is crossing the line.

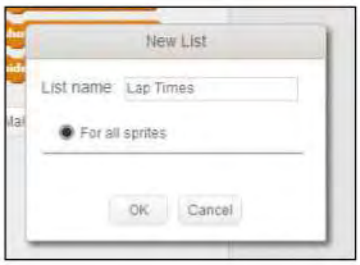


STEP 16

In Scratch, you can add scripts to the backdrop, just as you can to any sprite. Click on the Course backdrop next to the Sprites area, and start a new stack with a 'when green flag clicked' block. For now, just add a 'set variable to x' block and set it to 'lap count' and 0. Now start off a second stack with a 'when I receive message' block, setting the message to 'new lap'. Go to Data and grab a 'change variable by x' block and stack this underneath, and set the variable to 'lap count' and the value to 1. This will help us keep track of how many laps our car has finished.

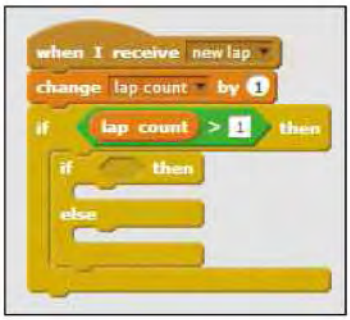
STEP 17

To time our laps, we can use Scratch's own built-in Timer function. Go to the Sensing category and click on the checkbox next to the 'timer' block to make it visible. You might want to move the counter that appears so that it's in the top-left corner of the Stage. Now go to the Data category and create our first list. Click the 'Make a List' button, and call the list 'Lap Times'. Untick the checkbox next to Lap Times to hide the list from view.



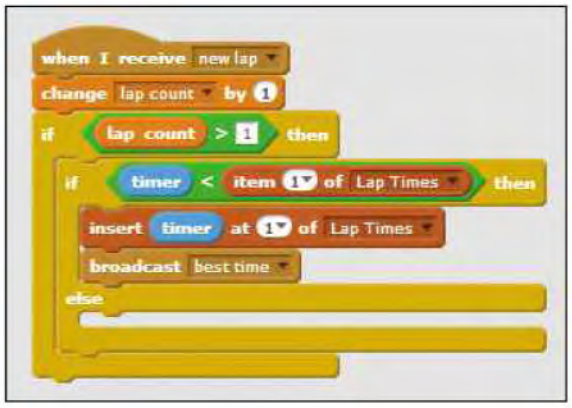
STEP 18

Let's get back to our backdrop script. Add an 'if, then' block to the bottom of the 'when I receive new lap' stack, and drag an 'x > x' operator into place after the 'if'. Drag the block for Lap Times into the first space, and type 1 into the second. This will tell the next few blocks to only start working if the car has completed its first lap. Now grab an 'if, then, else' block and drag this inside.



STEP 19

Pull an 'x < x' operator to the space next to the 'if'. Now go to Sensing, and find the 'timer' block. Place this in the first position on the operator. Now go to Data and find the block that says 'item 1 of Lap Times'. Drag this into the second position.

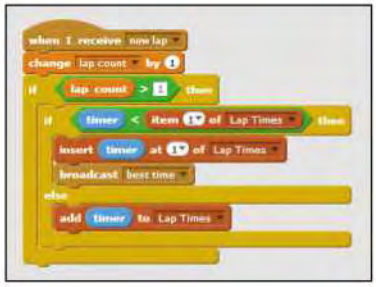


STEP 20

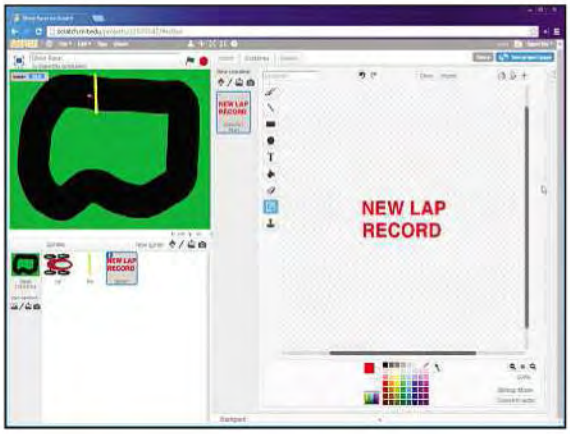
Find the block that says 'insert thing at 1 of Lap Times'. Put this in place in the first gap for the 'if, then, else' block. Drag the 'timer' block into the space where it says 'thing'. Underneath, place a 'broadcast message' block. Select a new message and call it 'best time'.

STEP 21

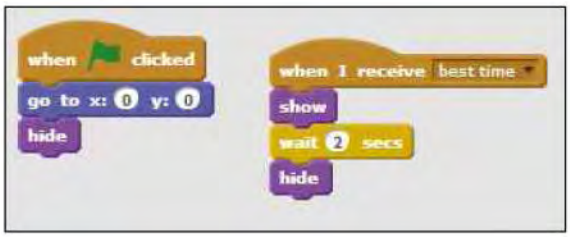
Now find the block that says 'add thing to Lap Times'. Drag this into position in the second gap of the 'if, then, else' block. Again, drag the 'timer' block into the space where it says 'thing'. What we've just done is tell the program to look at the timer, and see whether the timer for that lap is faster than the best time we've seen so far. If it is, it replaces the best time. If it isn't, it's just added to the list of times. Finally, go to the Sensing category and find



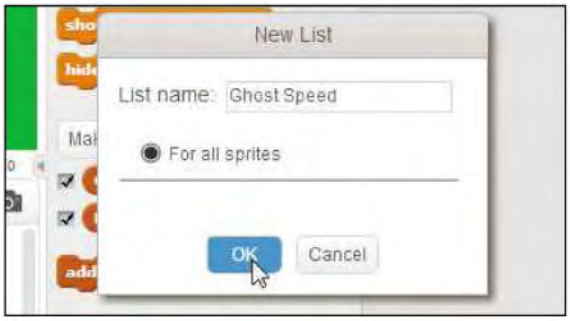
the Reset Timer block. Add it to the bottom of this stack. This resets the timer to 0 for the start of every lap.



STEP 22 Now create a new sprite by clicking the Paint new sprite button. All we need to do here is use the Text tool and type New Lap Record in a bright colour, as you see here.

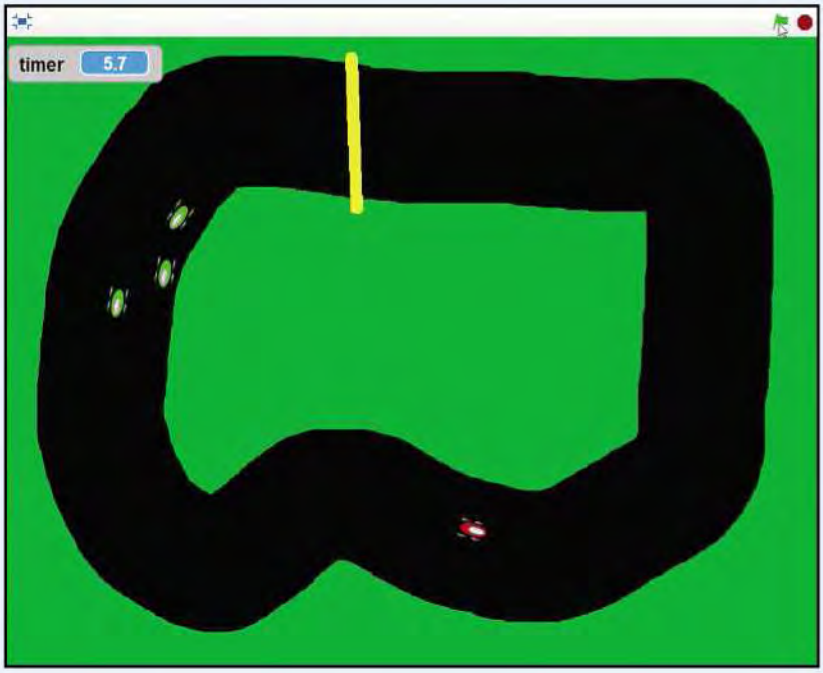


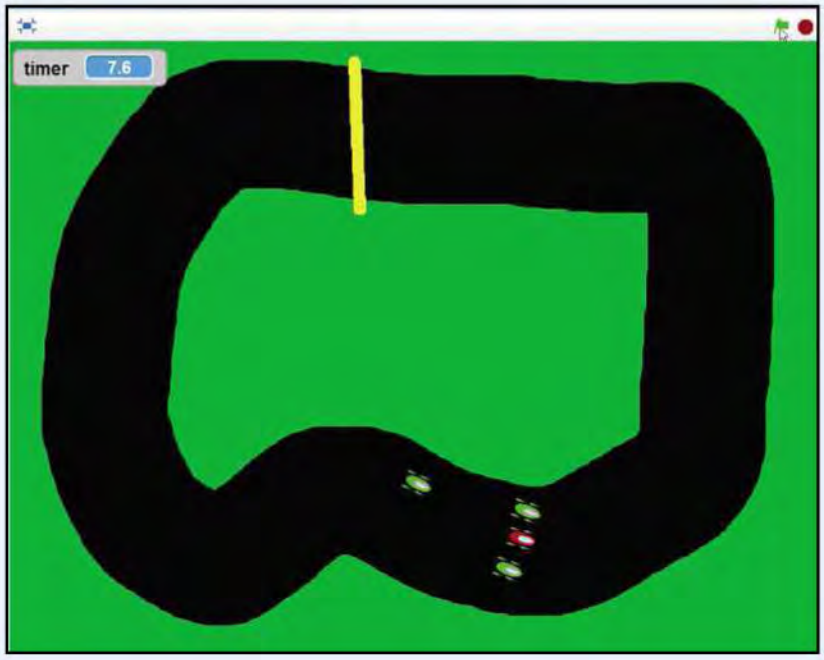
STEP 23 Now click on the Scripts stack and add these two shorts scripts. The first centres the New Lap Record text in the centre of the screen, then hides it. The second waits for the Best Time message, then flashes up the New Lap Record text in the centre of the screen for a couple of seconds, then hides it again.



STEP 24 We have the car moving and we're tracking lap times, but how about some rivals on the track? Well, first we need to create two new lists. For each, go to Data, and

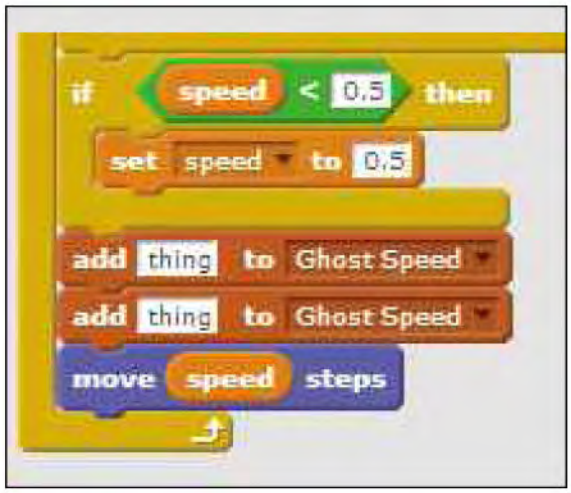
HOW THE GHOSTS WORK





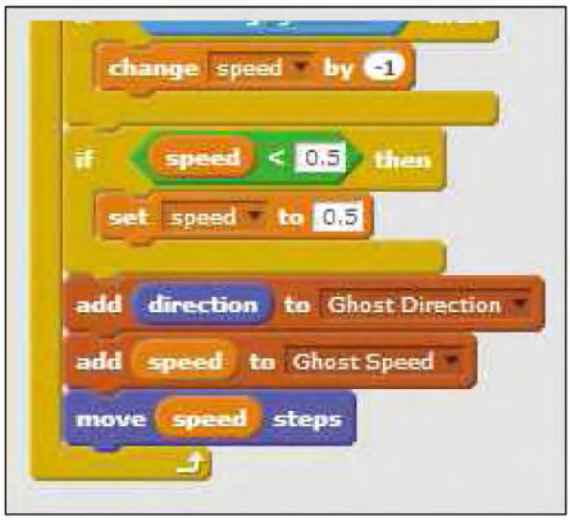
Each time the player car moves, its direction is stored in the Ghost Direction list, while its speed is stored in the Ghost Speed list. When a new ghost is spawned, it looks to item 1 on each list and moves in the direction the Ghost Direction list tells it to at the speed the Ghost List list provides. The Item variable then increases value by 1, so the next time the ghost wants to move it takes its direction speed from item 2 on the list, then item 3, and so on. While its speed isn't quite the same, it's basically copying the player's driving.

click the 'Make a List' button. Call the first list Ghost Direction and the second list Ghost Speed. Untick the checkboxes next to both so the two lists don't clutter up the screen.



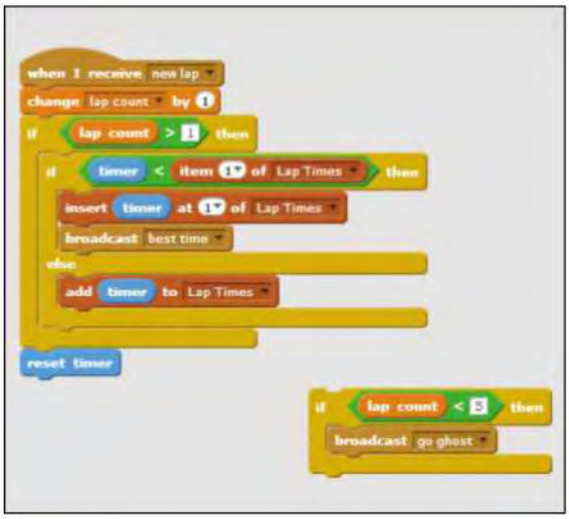
STEP 25 These two lists are going to store data for the player's car's speed and direction, so that the ghost cars can follow in the player's footsteps – or tyre tracks, if you like. To make this happen, we just need to add a couple of new blocks to the player's car's script. Click on the car sprite and scroll down to 'Move speed steps'. Now go to the Data category and drag in two 'add thing to list' blocks, just above it.

STEP 26 First, let's deal with the top one of the two new blocks. Go to the Motion category and look for the 'direction' block at the bottom of the screen. Drag this in where it currently says 'thing' and change the list at the end to 'Ghost Direction'. Now go to the Data category and drag the



Speed variable to where it currently says 'thing' on the second block. Drop it there, and change the list to read 'Ghost Speed', if it doesn't already.

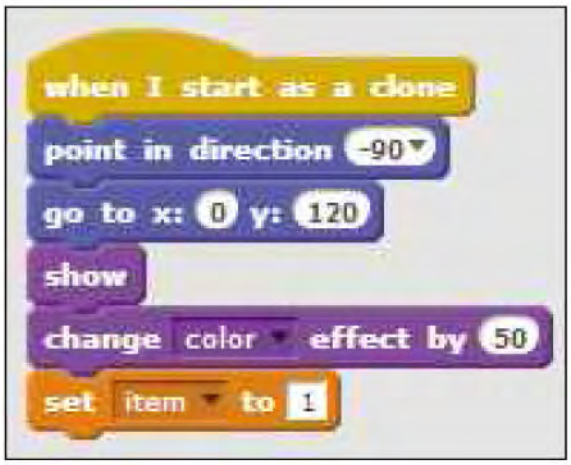
STEP 27 We also need to adjust the script for the backdrop. Look at the 'when green flag clicked' block, and add two 'delete x of list' blocks from the Data category. Set both to delete 'all', and set one to delete all of 'Ghost Speed', and the other to delete all of 'Ghost Direction'.



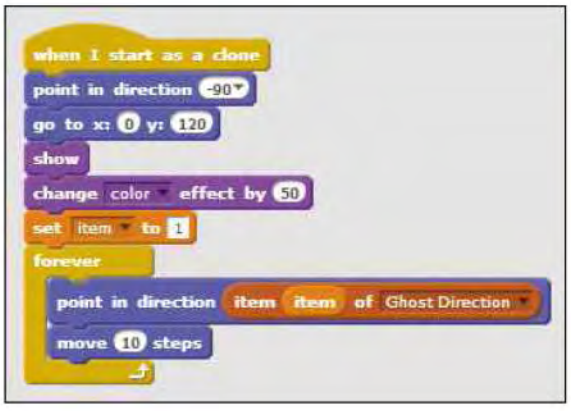
STEP 28 For the 'when I receive new lap' block, we need to create a new 'if, then' block. We set this up so that if the lap count is under 5 then it will broadcast a new message, which we'll call 'go ghost'. Drag this in under the 'if, then, else' block, but still within the 'if lap count > 1' block. We'll use this to spawn new ghost cars once per lap during the second, third and fourth laps.

STEP 29 Now duplicate the main car sprite and call the duplicate 'ghost'. Make double-sure that you have the ghost sprite selected, then delete all the blocks of script in the Scripts area. We need to add three all-new ones. The first tells the ghost car to hide. The second tells the ghost car to

keep listening for the 'go ghost' message. When it receives it, it has to wait 4 seconds, then create a clone of itself.

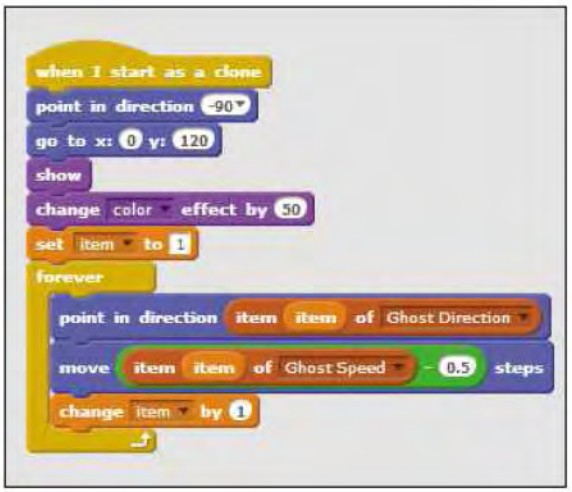


STEP 30 This last script is a bit trickier. First, we need to create a new variable called 'item', and make sure it's set to 'For this sprite only'. Start a new stack with 'when I start as a clone'. Drag a 'show' block into place, followed by a 'change color effect by 50', and two blocks to get the clone in the right position and heading in the right direction. Now go to Data and add a 'set variable to x' block, changing the variable to 'item' and the value to 1.

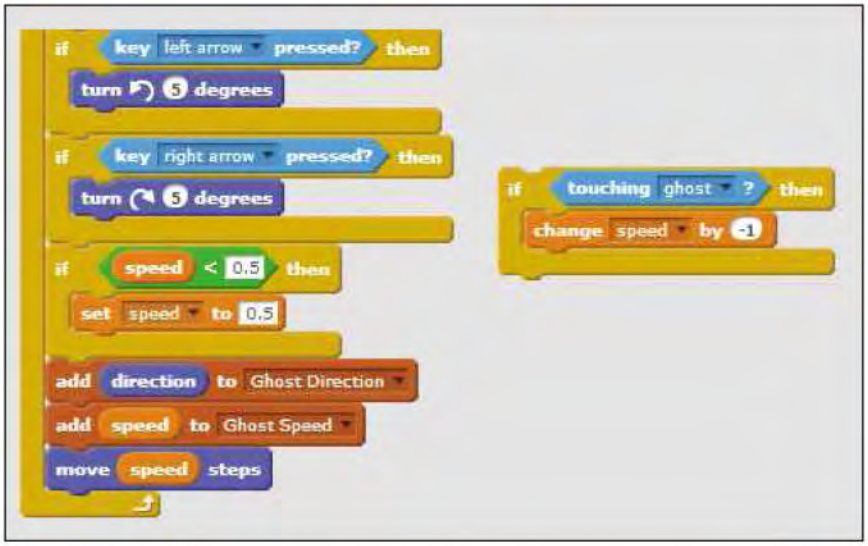


STEP 31 Next add a 'forever' loop. Inside it sits three more blocks. The first is 'point in direction' from the Motion category. Look at where the value normally sits, and drag in an 'item 1 of list' block from the Data category. Place the Item variable in where the item number would usually be, and change the list to 'Ghost Direction'.

STEP 32 The second block is almost a repeat, but using a 'move x steps' block and the Ghost Speed list. However, we want to slow down our ghosts a bit so our player has to make their way through them. That's why we use an 'x – x' operator block, with 'Ghost Speed' in the first space and 0.5 in the second. Finally, we use a 'change x by x' block to 'change Item by 1'.



STEP 33 We're nearly finished, but we'd like to have some kind of penalty for when the player's car hits a ghost car. All it takes is one final 'if, then' block in the scripts for the main car sprite. This tells the car sprite that if it touches the ghost sprite, it has to change its speed by -1. Drag this into place above the 'if speed <0.5 then' block, and you're all finished and ready to race! ●



My Scratch Quiz

Harness the power of lists to build your own brilliant quiz

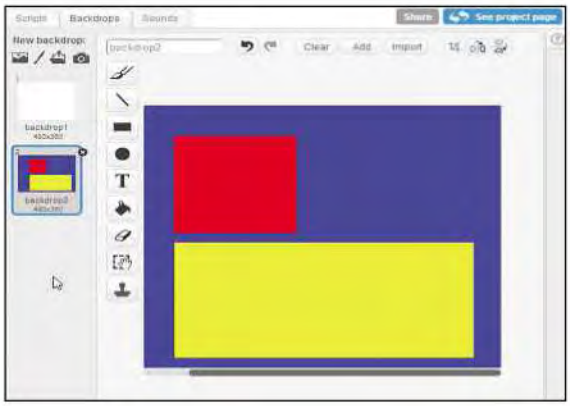
WHAT YOU'LL LEARN

- More about lists and what they can do
- How to use lists to hold and access information
- How to build clickable buttons
- How to use sound and animation to give feedback

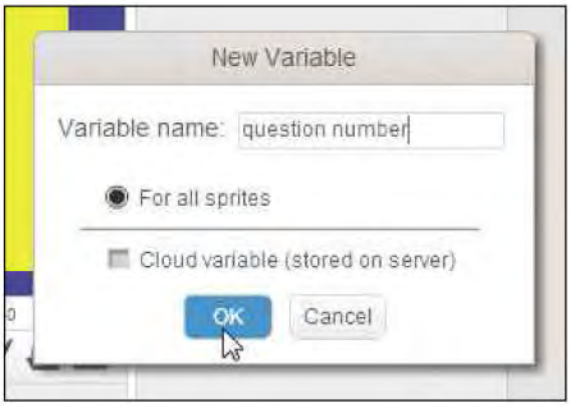
In the last project we used lists to store information from our player's race car and use it with our ghost cars. Now we're going to use lists again to store questions and answers for a quiz. It's a simple multiple-choice quiz, but there's plenty going on beneath the surface. So we're going to plan out the questions and answers first.

On some paper, write down each question and three possible answers – make sure you know which answer is correct. Since we're merely starting off this project, we need only five sets of questions and answers, but you can add as many as you like thereafter. This is one of those projects where, once the basics are up and running, it's easy to add to it.

STEP 1 Start off the new project by getting rid of the Scratch cat, then click on the 'Paint new backdrop' icon. The backdrop we want is simple: just a blue background, a red square and a yellow rectangle as shown. Use the paintbucket to fill in the background, then use the Rectangle tool with the 'filled' button clicked in to make the two shapes. When you've finished, mouse over to the old empty background and click the 'x' button in the top-right corner to remove it.



STEP 2 Now we need to create our first variables. Click on the Scripts tab then the Data category. Click the Make a Variable button and call the variable 'question number'. Make sure that 'For all sprites' is switched on, then click OK.

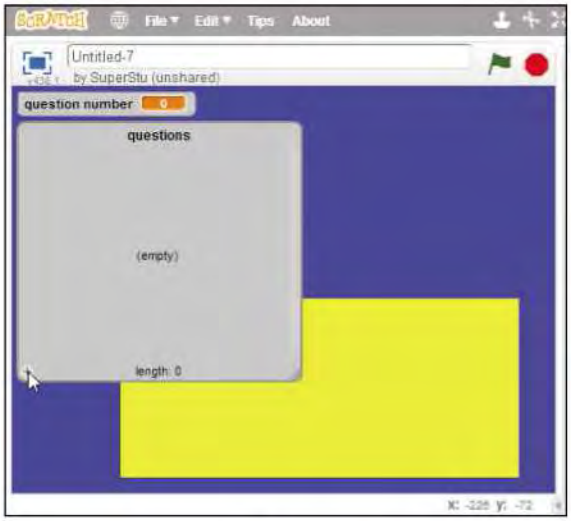


Now repeat the same steps to build a second variable, but call this one 'score'.

STEP 3 It's time to start adding questions. Click the Make a List button, then enter the name as 'questions'. Now click on the bottom-right corner of the new list and drag it out a little. This will make it easier for you to see what you're doing. With that done, press the '+' button in the bottom-left corner to start adding items to the list.

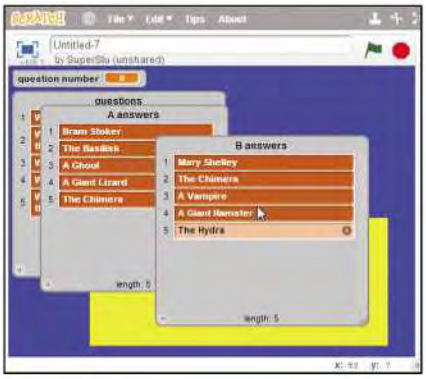
TOP TIP

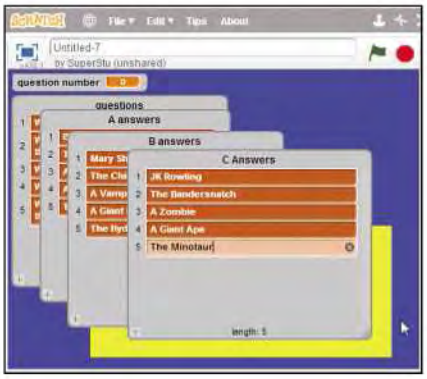
On some screens you might struggle to see what you're typing. Find out if the browser you're using has a Zoom setting. Zooming in to 100% or more will provide a clearer view.



ABOUT LISTS

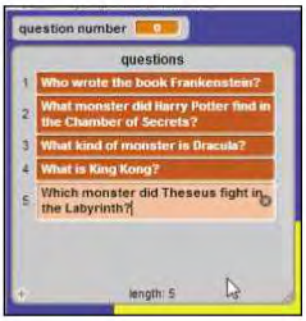
Scratch's lists are what programmers call 'single dimensional arrays'. Each one has an index number (for example, item 1 of the questions list) that points to one entry (Who wrote the book, Frankenstein?). However, in this project we use a variable (question number) to work across five different lists and pull out the right information from each one. As long as the lists have the correct information in the correct order, it's all under control. In effect, we're faking what programmers call a 'multi-dimensional array'.





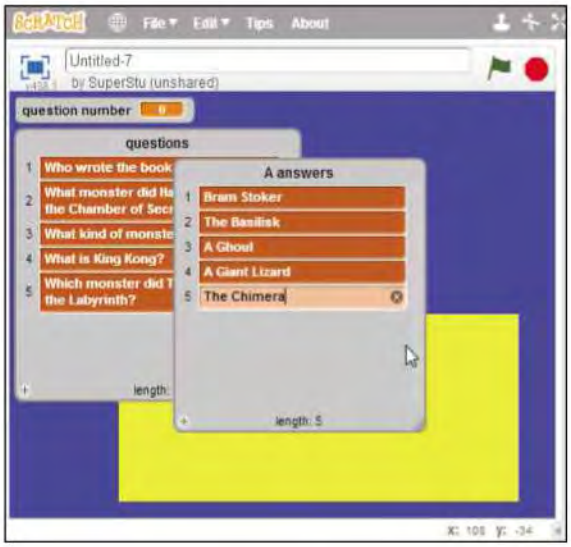
STEP 4

A box will appear with a blinking text cursor. All you need to do is type in your question then press the Return key. If you need to remove a question or you add one too many, click the 'x' button within the little text box.



STEP 5

Once that list is done, click the Make a List button again to create a new list. Call this one 'A answers', and enter your first set of multiple-choice answers. Each one needs to be a possible answer for the question with the same number.

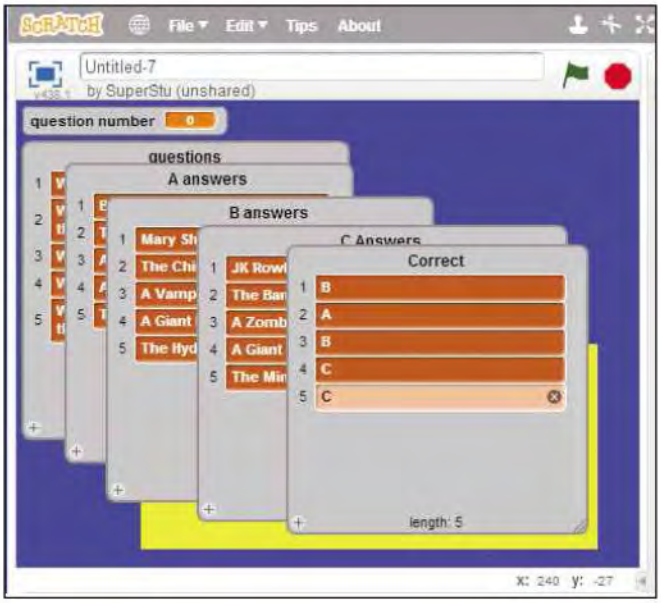


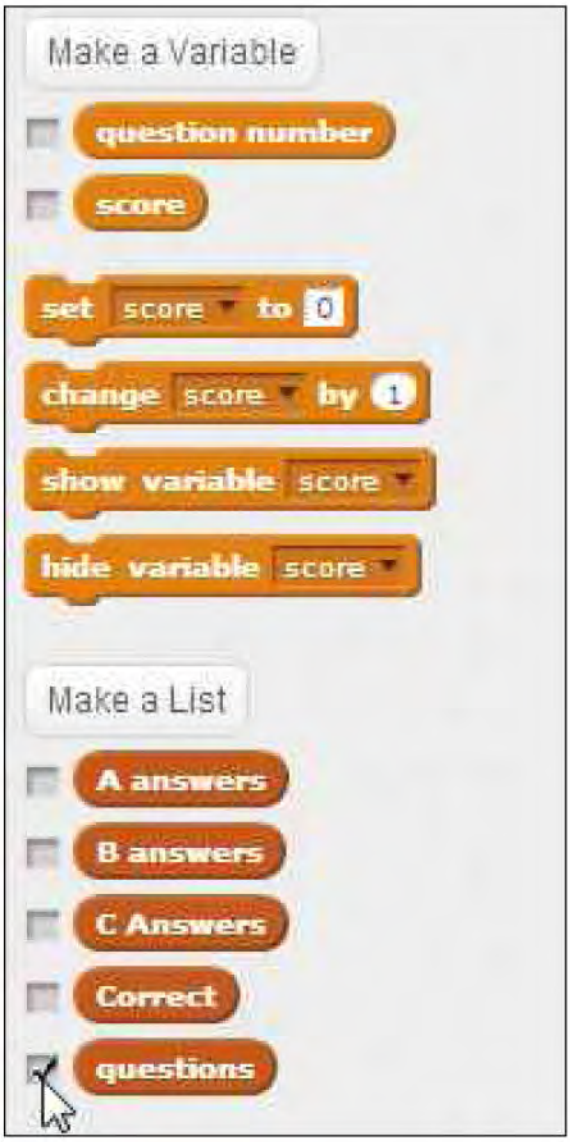
STEP 6

When you've finished your first set, make a new list, call it 'B answers' and enter your next set of answers. Once that's done, create another new list and call it 'C answers'. Enter your final set of answers, and we have the questions and answers ready to go.

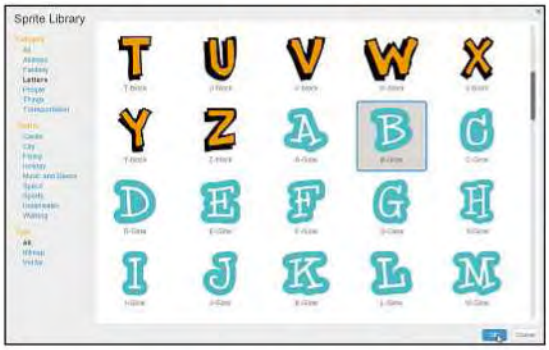
STEP 7

All the same, we still have one more list to build. This one is called 'Correct', and it lists which answer – A, B or C – is the correct answer for each question. Make sure you pick the right one!





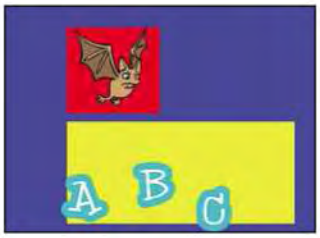
STEP 8 Of course, nobody wants to see a quiz with all the questions and answers on the screen. Untick the boxes next to all your new lists and variables to remove them from the stage.



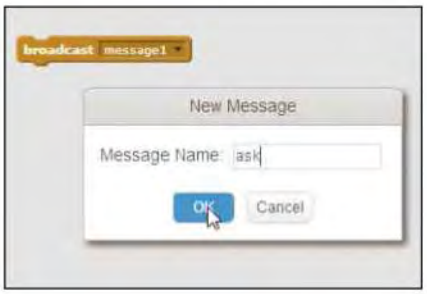
STEP 9 Now it's time to add some buttons. Click the 'Choose sprite from library' button then click on the Letters category. Click on the A-Glow letter, then click OK. Repeat to add the B-Glow and C-Glow letters and all three will appear on the Stage. These are our answer buttons.

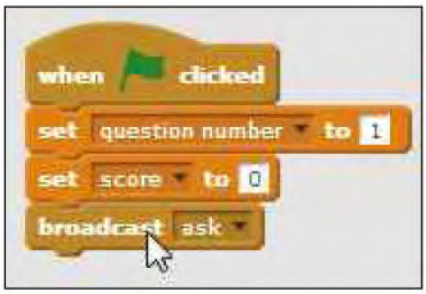
STEP 10 If we leave the buttons where they are, we won't be able to use them or read the multiple-choice answers. Drag them into position as shown. There's a little trial and error involved, so you may need to come back and adjust them later. Remember: you can always use the Grow and Shrink buttons in the toolbar to make a sprite bigger or smaller to fit in all three.

STEP 11 Now to add our question master: the ever-cheerful bat. Click the 'Choose sprite from library' button, then on the Animals category to find him. Click on him and then press Add. Now drag him into position inside the red box.



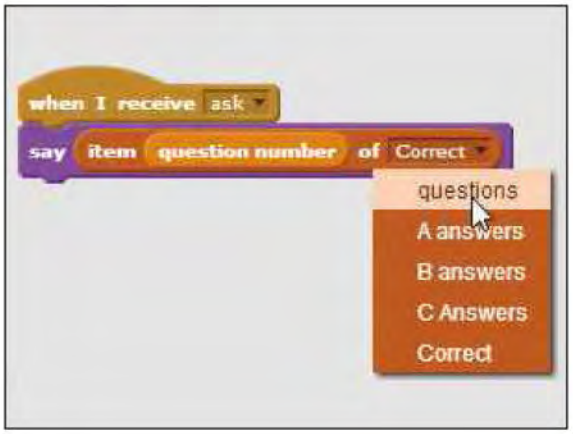
STEP 12 Click on the Stage to select it. It's time to start building the first script. Pull in a 'when green flag clicked' block from Events, then go to Data and pull in the 'set variable to x' block. Set the variable to 'question number' and enter the value as 1. Now pull in another 'set variable to x' block, but set the variable to 'score' and leave the value as 0.



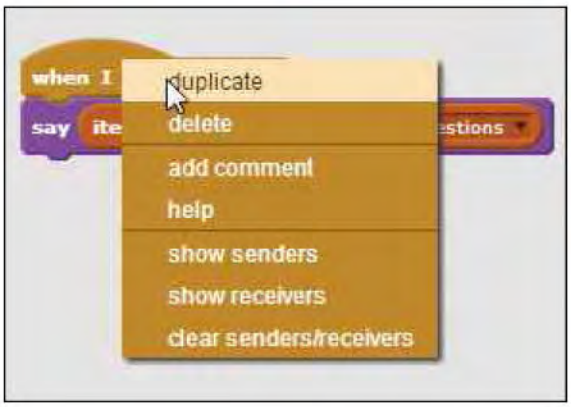


STEP 13 Go to the Events category and drag in a 'broadcast' block to the Scripting area. Select New Message and enter 'ask'. Then drag the block into position at the bottom of the stack.

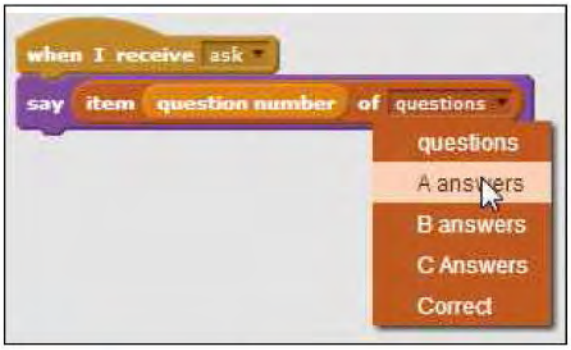
STEP 14 Now click on the Bat sprite. Go to Events and grab the 'when I receive block'; set it to 'ask'. Now go to Looks and pull in a 'say' block. Add it to the bottom of your new stack.



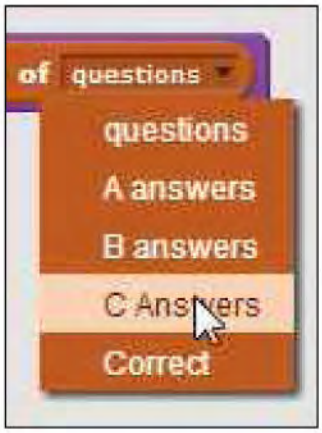
STEP 15 This next bit is a little complicated. Click on the Data category and look for the block that says 'Item 1 of list name', where list name will probably read Correct. Pull this into place in the 'say' block. Now look again at the Data blocks and pull in the block for the 'question number' variable. Drop this where the number 1 is at the moment. Finally, set the list name at the end to 'question'. This tells Scratch to look at the current value of the 'question number' variable, and say the question attached to that number.

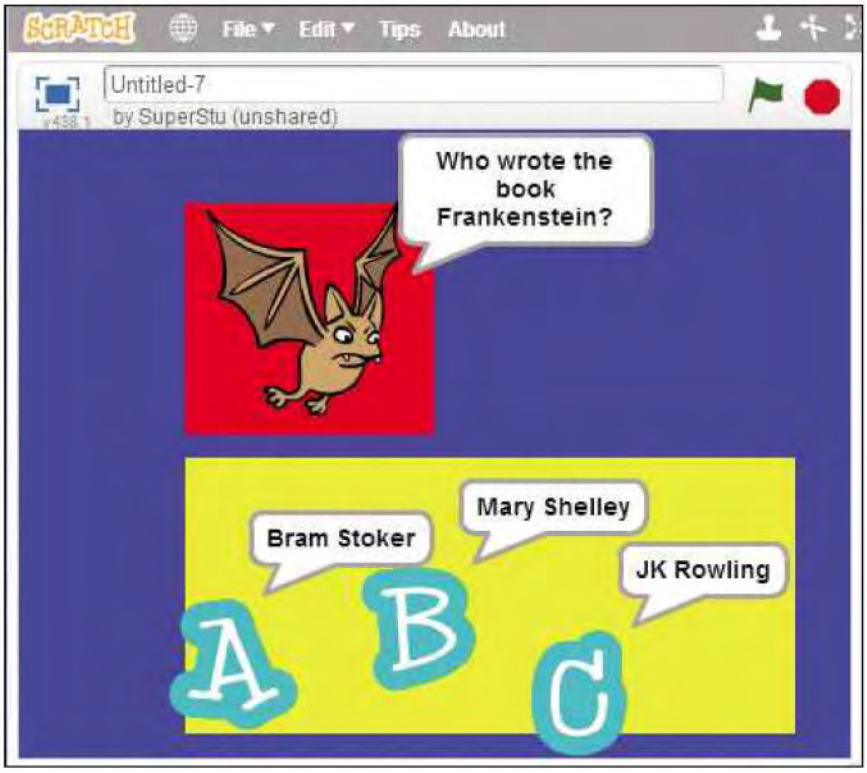


STEP 16 The good news is that, having built this block, we can reuse it. Right-click on it then click Duplicate and drag it onto the A-Glow sprite. Repeat for the B-Glow and C-Glow.

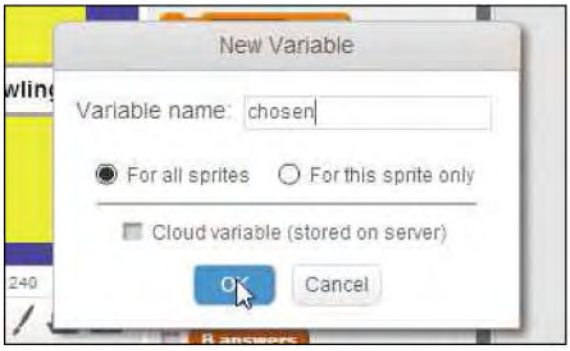


STEP 17 All we need to do now is adjust the script for each sprite. Click on the A-Glow sprite and change the questions list to read 'A answers', then click on the B-Glow sprite and change it to read 'B answers'. You can probably work out what to do with the C-Glow sprite, right?



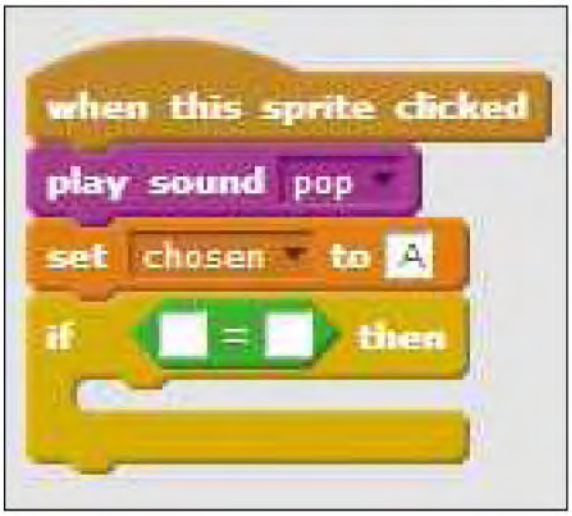


STEP 18 Now press the Green Flag button to see the quiz in action so far. You should see your first question plus the first three multiple-choice answers.

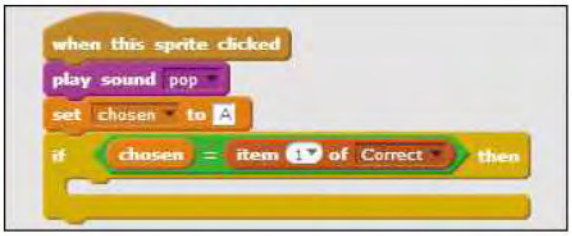


STEP 19 It's time to set up our buttons to answer the questions. First, create a new variable by clicking on the Data category then the Make a Variable button. Call this one 'chosen'.

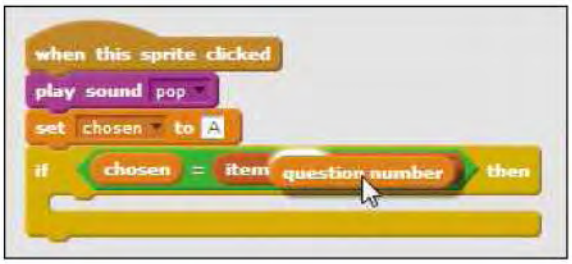
STEP 20 Now click on the A-Glow sprite, go to Events and drag in a 'when this sprite clicked' block. Below it, add a 'play sound pop' block from Sound, then go to Data and drag in a 'set variable to x' block. Change the variable to read 'chosen' and change the value to read A.



STEP 21 Next, pull in an 'if, then' block. Go to the Operators category and pull the 'x = x' operator into place next to the 'if'.

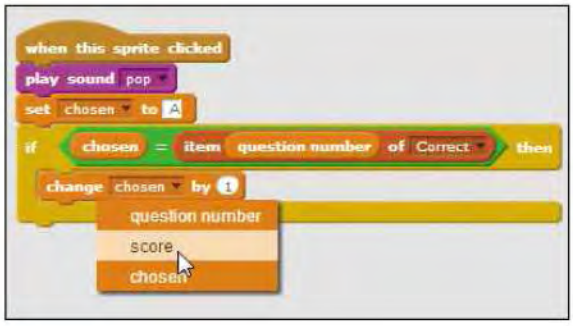


STEP 22 This bit is also pretty complicated, so take it slowly. Find the chosen variable in Data, and drop it in the first slot of the 'x = x' operator. Now find the block that reads 'item 1 of Correct' and drag that into the second slot. If the list name doesn't read Correct, change it to Correct.

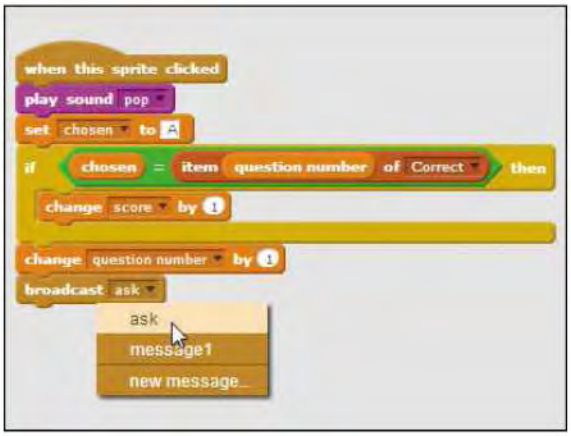


STEP 23 Now grab the block for the 'question number' variable; put that into the slot next to 'item'. This tells Scratch to check whether the chosen variable has the same value as the

Correct item for that question. In other words, if the player has clicked the button with the right answer.

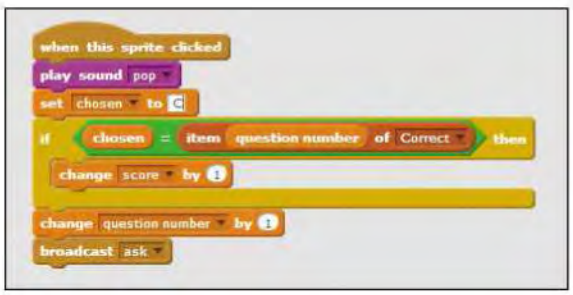


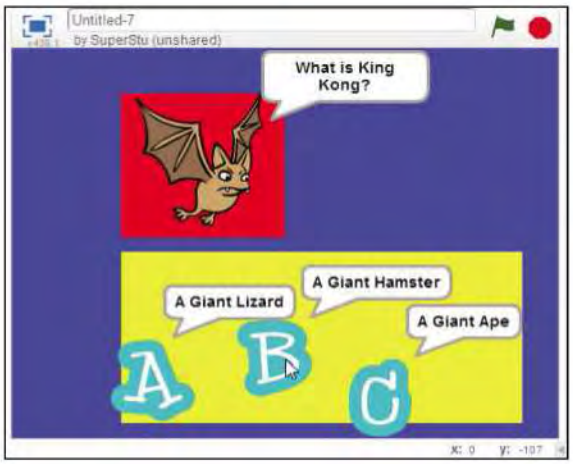
STEP 24 Now pull in a 'change variable by x' block, and set the variable to 'score' and the value to 1. This just tells Scratch that if the conditions have been met – that the player has clicked the right answer – to increase their score by 1.



STEP 25 This stack needs two last blocks. First, pull in another 'change variable by x' block and set it to 'question number' and 1. Now drag a 'broadcast' block to the bottom of the stack, and set it to broadcast the 'ask' message. This basically tells all our sprites to deliver the next question and its possible answers.

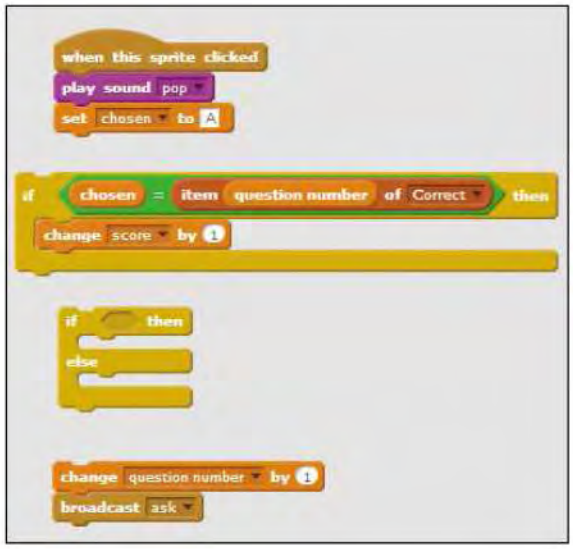
STEP 26 Again, having finished this stack for one sprite, we can duplicate it and use it with the others. Right-click it, select Duplicate, and drag it across to the B-Glow and C-Glow sprites. Now click on each one in turn and adjust the script to match. Change the value in the 'set chosen to' block to B for the B-Glow sprite and C for the C-Glow sprite.



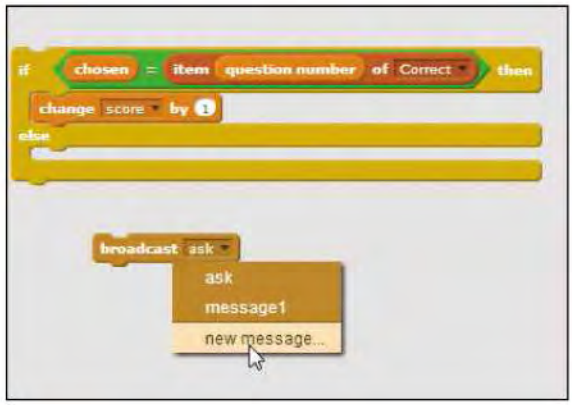


STEP 27 Press the Green Flag button to give the quiz a go. Clicking on an answer should now take you to the next question.

STEP 28 It works – but wouldn't it be great if your players could find out if they had got the answer right or wrong? Let's add two new sprites: a tick (Button 4) and a cross (Button 5) from the buttons category. Once they appear on the Stage, move them into position in the top-right corner. It doesn't matter if they overlap. In fact, it works better if they do.



STEP 29 Now we're going to dismantle the block of script we made earlier. Click on the A-Glow sprite and pull the 'change question number' and 'broadcast' blocks from the bottom of the stack, then the 'if, then' block. With that done, drag in an 'if, then, else' block.

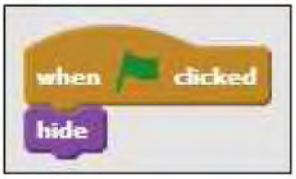


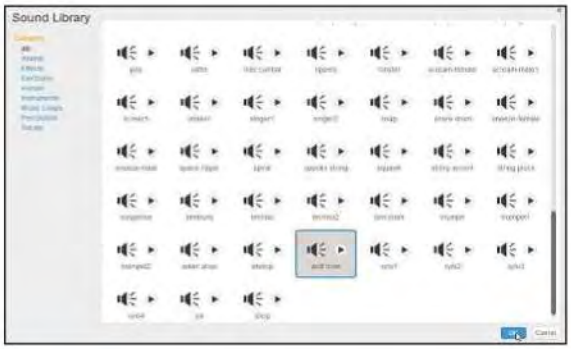
STEP 30 Drag the 'if chosen = item question number of correct' block we assembled earlier into the slot next to the 'if', and the 'change score by 1' block into the C shape beneath. Now drag in a new 'broadcast' block, select New Message and call the message 'tick'. Drag it into the C shape beneath the 'if'. Pull in a second new 'broadcast' block and call this one 'cross'. Drag this into the C shape beneath the 'else'.

STEP 31 Finally, reattach the 'change question number' and 'broadcast ask' blocks to the bottom of the stack. Now, go to the B-Glow and C-Glow sprites and right-click and delete the stack that this stack will replace. With this done, you can just right-click to duplicate our new stack, then drag

it over to the B-Glow and C-Glow sprites. Remember to change the value in the 'set chosen to' block to B for the B-Glow sprite and C for the C-Glow sprite.

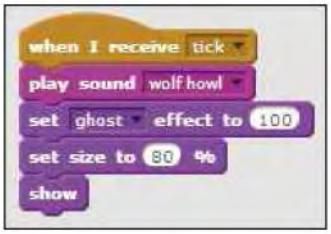
STEP 32 Now click on the 'tick' sprite, and add a new script with a 'when green flag clicked' block at the top and a 'hide' block, from Looks, beneath. This hides the sprite when we don't want to see it.





STEP 33 Click on the Sounds tab, then on the 'Choose sound from library' button to add a sound. Here we're using a spooky wolf howl.

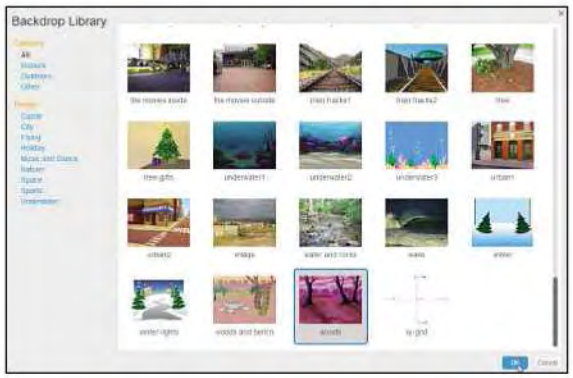
STEP 34 Now start a new stack with a 'when I receive' block, followed by a 'play sound' block from Sound. Set the sound to the wolf howl. Below this add a 'set x effect to x' block and a 'set size to x %' block. In the first block, set the effect to 'ghost' and the value to 100. In the 'size' block set the value to 80%. Finally, add a 'show' block from Looks.

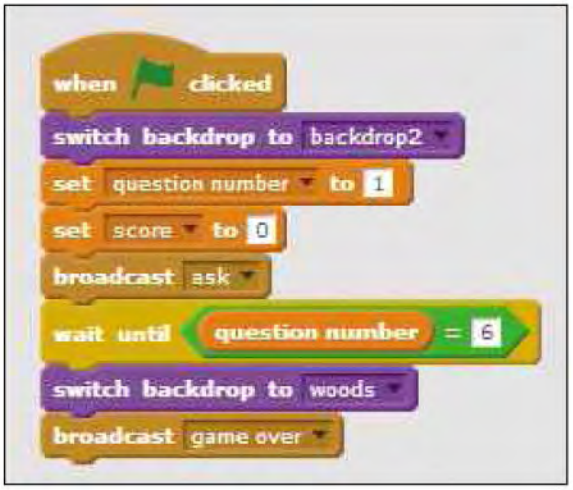


STEP 35 Add two 'repeat x' blocks to the bottom of the stack. In the first one, add a 'change x effect by x' block and set the effect to 'ghost' and the value to -5. In the second, add a 'change size by x' block and set the value to 2. Finally, add a 'hide' block to hide the sprite when it's finished showing.

STEP 36 Having built these two stacks, we can reuse them for the cross. Duplicate each one and drag it over. Change the message in the top block of the larger stack to 'cross', and give it a different sound if you like. Here we've opted for a piercing screech.

STEP 37 All we need now is a cool way to end our quiz. We'll start with a quick backdrop switch. Click the 'Choose backdrop from library' button and select something appropriate. Here we're using the woods. We don't want to start with this backdrop, though, so click on the Stage, then on the Scripts tab, and add a new 'switch backdrop to x' block into our 'when green flag clicked' stack. Set it to our main backdrop: backdrop2.

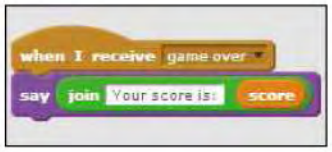




STEP 38 Now add a 'wait until' block to the bottom of the stack and put an 'x = x' operator block in the slot. Click on the Data category and pull the 'question number' variable into the first slot of the operator. Now work out how many questions you have, add 1 to the figure, then type that in as the second value. To finish this stack, add another 'switch backdrop to' block and set it to 'woods', then add a new 'broadcast' block. Call this one 'game over'.

STEP 39 Click on the Bat sprite and assemble one last stack of blocks. This one starts with a 'when I receive' block; set this to 'game over'. Now grab a 'say' block from Looks and pull in the 'join hello world' block from Operators. Place it where you'd normally enter text. Type 'Your score is:' with a space at the end where it says 'hello'.

STEP 40 Finally, drag the in 'score' variable and drop it over 'world' in the 'join' block. When the player has answered all the questions, the backdrop changes and the bat dishes out their final score. ●



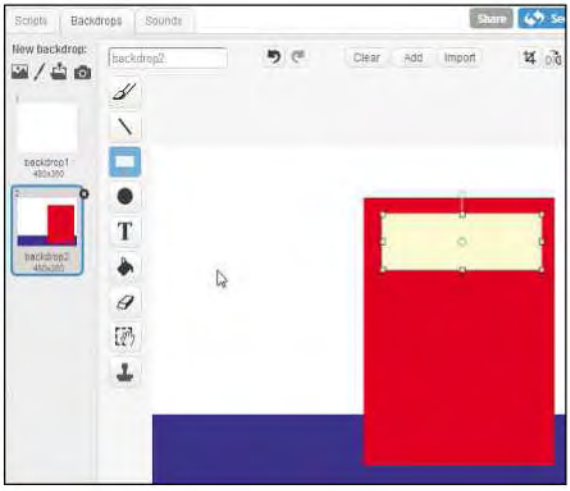
My Incredible Scratch App

Learn how to build an app with this fun-filled, monkey-themed timer

WHAT YOU'LL LEARN

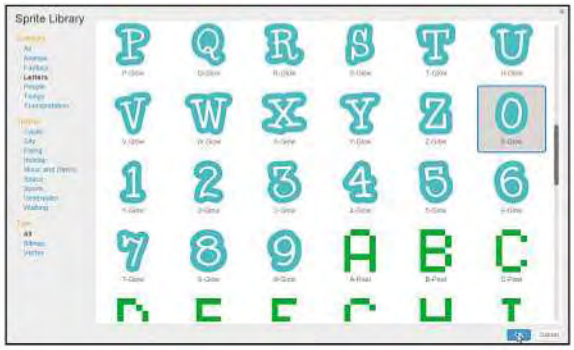
- How to build buttons with animated effects
- How to use cloning to make a numerical display
- How duplicating scripts can save you plenty of time

Scratch is great for forming simple arcade games or cartoon animations, but you can also use it to build smartphone-style apps. This one – a programmable timer with a dancing monkey – is both simple and a little bit silly, but it gives you a taste of what's involved. Building apps by hand can be a long-winded process, but by using duplication we can cut down the workload.



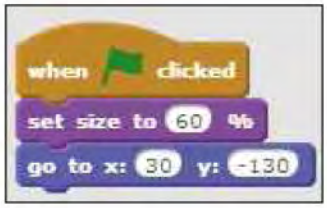
STEP 1 Start by ditching that under-appreciated Scratch cat, then click the 'Paint new backdrop' button. The scene you want to paint is constructed from three filled rectangles. Start with the blue one (the floor) then the red one (our timer) and the yellow one (our screen).

STEP 2 Now we want to put our buttons into place. Click the 'Choose new sprite from library' button and put them in in turn. Here we're using the 0-Glow, 1-Glow, 2-Glow and so on, buttons from the Letters category. Just keep on adding

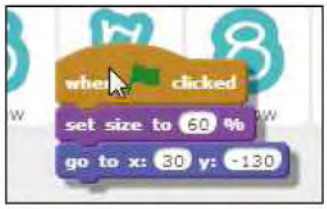


them and pressing OK until they're in a jumbled mess in the middle of the stage.

STEP 3 The buttons need to be resized and put in the correct position. We could do this manually, but we'll get a more precise result using a script. Click on the 0 sprite then set up the stack as shown, with the value of the 'set size box' to 60% and the positions on the 'move to' block set to x: 30 and y: -130.



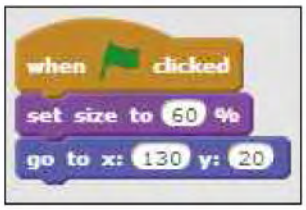
STEP 4 You can now drag and drop this stack onto each of the number sprites. This, of course, will mean each one turns up in the same position, but you can adjust the x: and y: values in each 'move to' block to get them in the right place.



TOP TIP

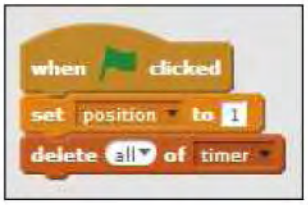
The exact positions of buttons may need tweaking, so press the Green button, see where the 0 turns up and make adjustments. This then gives you coordinates to work from for your other buttons.

In our case, the four rows of buttons have the y: coordinates -130, -80, -30 and 20 (all 50 pixels apart), while the three columns have the x: coordinates 30, 80 and 120. Once you know that, you can work out the rest.



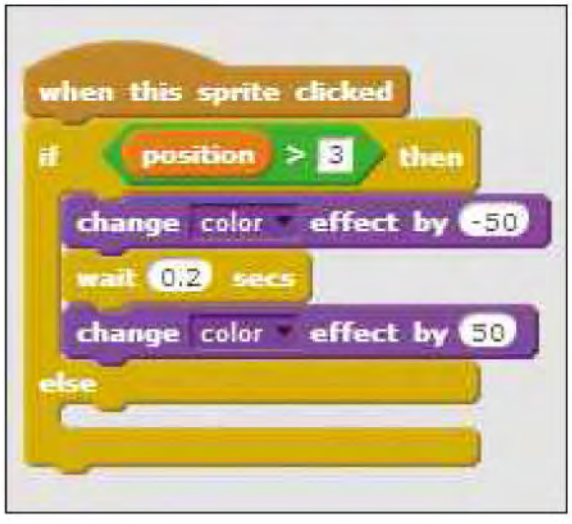
STEP 5

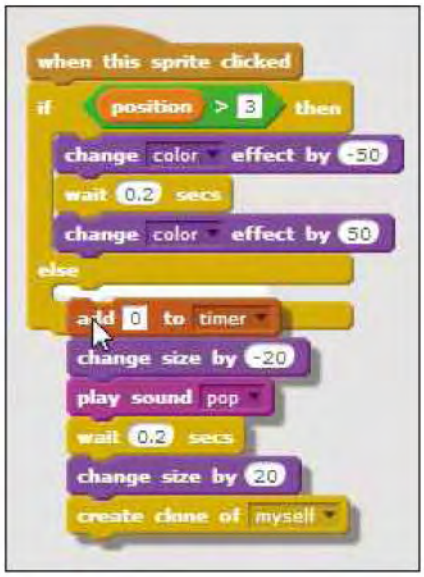
Now click on the Stage, then on the Data category. Make two new variables – 'countdown' and 'position' – and one new list, which we'll call 'timer'. We also need to add a quick stack to clear the decks each time the app launches. It's a 'when green flag clicked' block, followed by a 'set variable to x' block with 'position' as the variable and 1 as the value. Finally, pull in a 'delete x of list' block, and set x to 'all' and the list to 'timer'.

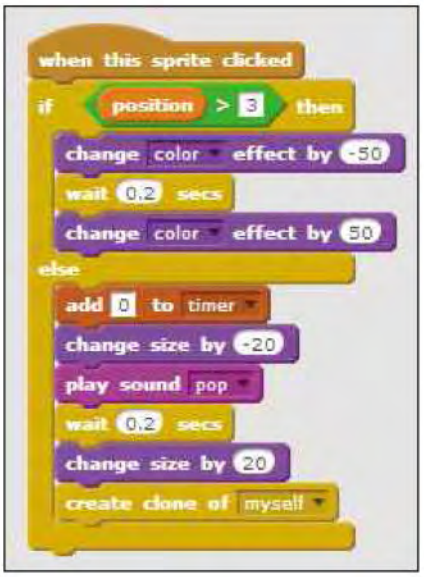


STEP 6

Right, time to start scripting our buttons. This stack starts with a 'when this sprite clicked' block, followed by an 'if, then, else' block, where we place an 'x > x' operator block into the slot after the 'if'. In the C-shape beneath, we have two 'change x effect by x' blocks from Looks separated by a 'wait control' block set to 0.2 seconds. Both 'change x effect' blocks need to be set to color, but the top one is set to -50 and the bottom to 50. This part of the stack tells Scratch to check if we already have three numbers in our display, then flash colours quickly, but to not add on any more numbers. Three is enough.







STEP 7

We'll now prepare a few blocks ready to go into the C-shape under the 'else'. Go to Data and get the 'add x to list' block, with the value set to 1 and the list set to 'timer'. Then we need a 'change size' block from Looks, set to -20, then a 'play sound' block from Sound, set to 'pop'. After that we have a 'wait' block from Control set to 0.2 secs, another 'change size' block set to 20 and, finally, a 'create clone of' block from Control set to 'myself'. Assemble that lot, then drag it into position.

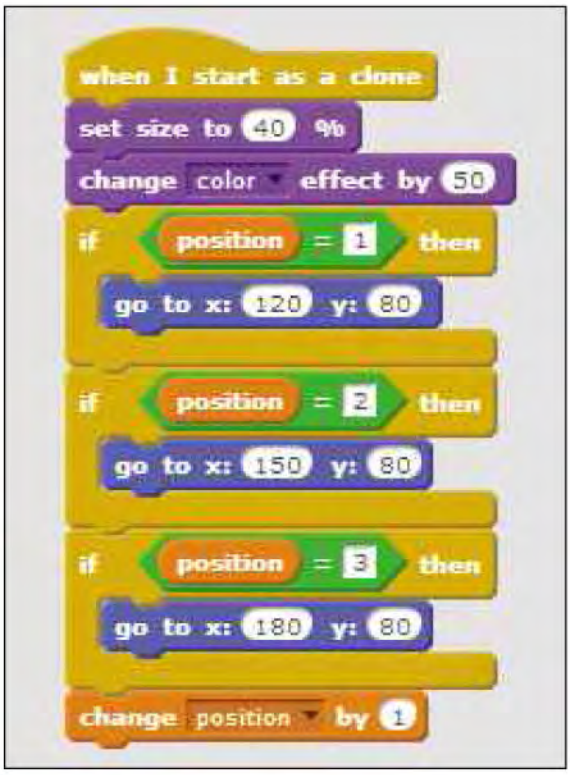
STEP 8

Again, we can save ourselves time and effort by duplicating. Drag this stack onto each number in turn, then go through and adjust each script. All you need to change is the

'add x to list' block (at the top of the stack under 'else'). Change it to 1 for the 1-Glow sprite, 2 for the 2-Glow sprite, 3 for the 3-Glow sprite and so on, up to 9.

STEP 9

The buttons will work, but we want to put some numbers on our display. We're going to cut another corner here by using cloning. Click the 0-Glow sprite again. The start of this script, after 'when I start as a clone', tells the new clone to be smaller than the Button sprite and a different colour.



STEP 10

Now we'll use our 'position' variable to track where the numbers go. We use three straight 'if, then' conditional blocks from Control, each one with an 'x = x' operator block after the 'if'. This tells Scratch to put the new clone in one place if it's the first button pressed, to the right if it's the second, or even further to the right if it's the third.

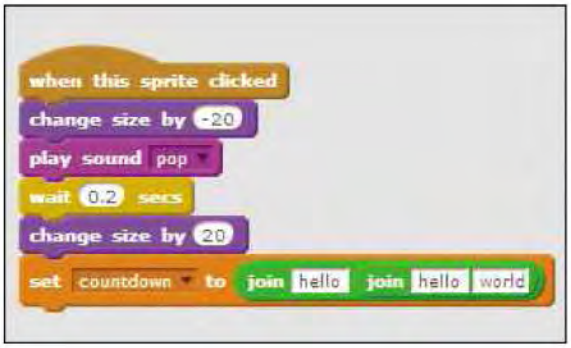
STEP 11

You've guessed it – we can duplicate this script too. Drop it onto each of the other nine number sprites. This time it doesn't even need any adjustment.

STEP 12

We now add a new sprite – the tick from the Things category – to play the part of the Go key on our timer. Just drag this one into position, then start work on this stack of

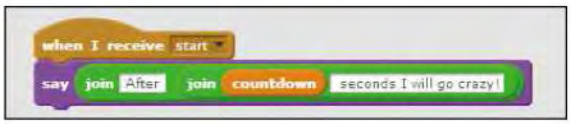
script. It begins easily enough with a 'when this sprite clicked' block, followed by a 'change size' block, a 'play sound' block, a 'wait' block and another 'change size' block. The tricky bit is the 'set variable to x' block at the bottom. You can set the variable to 'countdown', but for the value you need to drag in one 'join hello world' block from the operators, then drag another one in and drop it where the first one says 'world'. The result should look like this.



TOP TIP

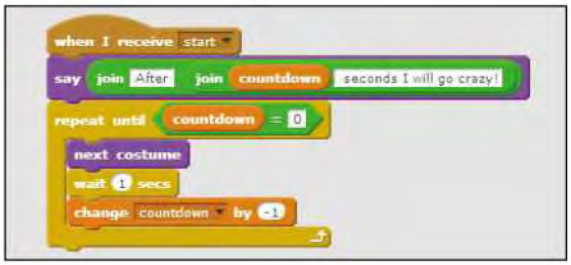
Again, you might have to experiment a bit to get the positioning right. Set up the script in step 10 then press the Green flag and try pressing the 0 key three times to see where the numbers turn up. If they're not right, change the x and y coordinates in the three Go to x: x y: y blocks.

STEP 13 The remaining 'hello' and 'world' slots are now filled with three 'item x of list' blocks from Data, with the list in each one set to 'timer'. The first should read item 1, the second item 2 and the third item 3. Finally, add a 'broadcast' block and give it a message. Call it 'start'.



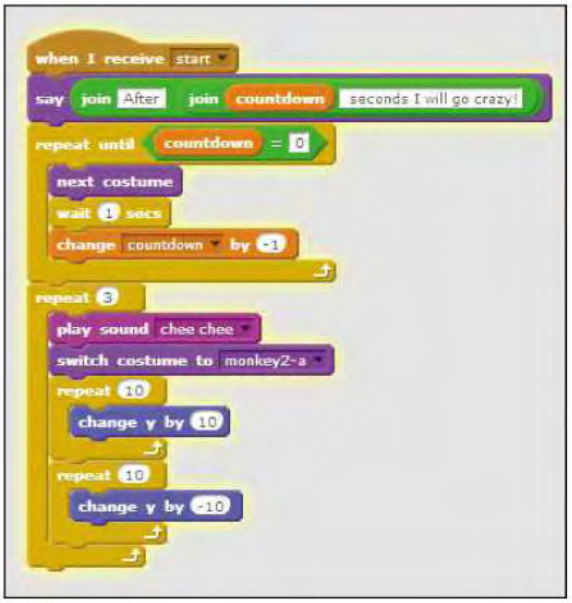
STEP 14 It's time to add our final sprite – this cheeky monkey, who's going to act as our alarm. Bring him in from the Sprite Library, drag him into position, then start on his stack of script. The first bit uses a 'say' block with two 'join' operators, just like we used in step 12. Change the first bit of text to 'After ' (remember the space at the end), then go to Data and drag the 'countdown' variable into the second place. In the last bit, type ' seconds I will go crazy!'. Remember the space at the beginning.

STEP 15 But how will our Monkey go crazy. This next section uses a 'Repeat until' loop, using an 'x = x' operator with the 'countdown' variable as the first value and 0 as the second. The blocks within the loop tell Scratch to cycle



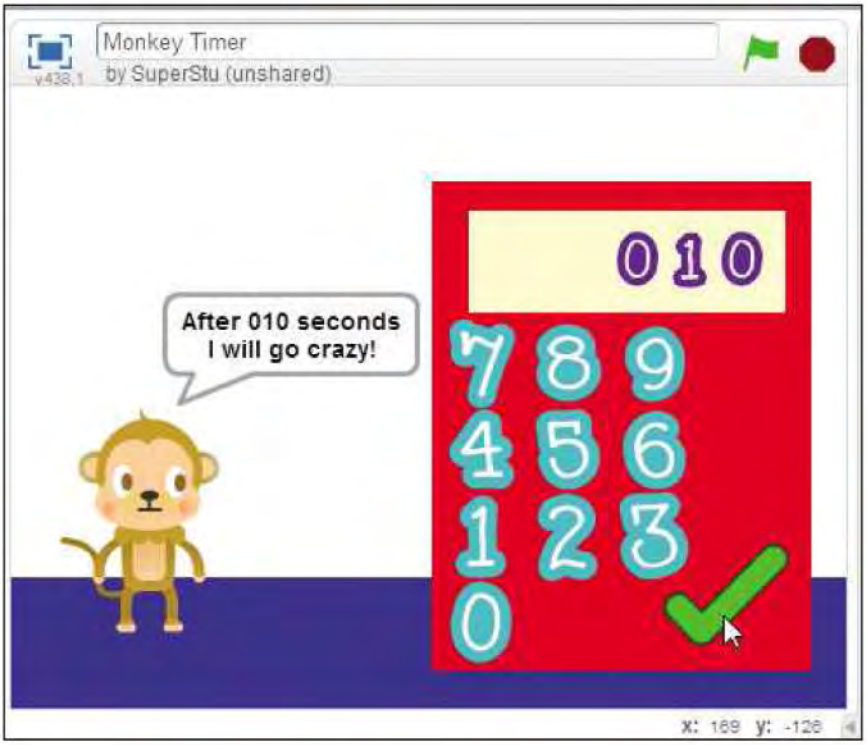
through the monkey's costumes, wait a second, then decrease the value of the 'countdown' variable by 1. This makes the script count down until it reaches 0.

STEP 16 This second section is our alarm. There's a 'repeat x' block set to 3, and inside that we put a 'play sound' block with the monkey's 'chee chee' noise, followed by a 'switch



costume' block set to 'monkey2-a'. The two 'repeat x' blocks below tell Scratch to move the monkey up by one pixel 10 times, then down by one pixel 10 times, so that it looks as if he's jumping up and down.

STEP 17 That's it. Press the Green Flag button and click on three of the number buttons in a row. Click the tick, and the timer begins. When it reaches 0, the monkey goes bananas!



Put yourself in the program

Get Scratch working with your webcam, and you can put your own face in a program or add simple sound and motion controls

WHAT YOU'LL LEARN

- How to use a webcam for motion controls in Scratch
- How to trigger effects with noise
- How to bring webcam photos into programs
- How to record your own sounds and use them

We might mostly use webcams for making video calls, but they can also be put to good use in your latest Scratch program. Webcams can be used to capture sprites and backgrounds, allowing you to put just about anyone or anything into your program, or set a game inside your living room. Meanwhile, take a look inside the Sensing script blocks and you'll find a series of blocks that enable you to sense motion using the camera or noise from a microphone, so that your scratch programs can respond to movement, a shout or a handclap from the user. Sounds good? We'll see how it's done with a trio of bite-sized projects.

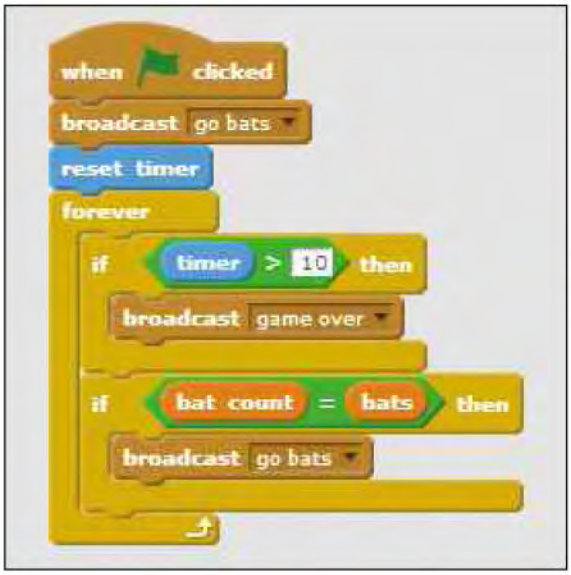
Project One: Bat Burster

Our first webcam project is a simple game where we'll despatch a swarm of deadly vampire bats with a pointy finger (although, in practice, the game will work with just about anything you can wave in front of the screen).

STEP 1 Kick off the project by hitting the 'Choose a new sprite from library' button, and selecting this cartoon bat. Leave the Stage with the default blank backdrop. We'll be filling it in a minute. Now create three variables and call them 'bats', 'score' and 'bat count'. You can set them all to For all sprites.

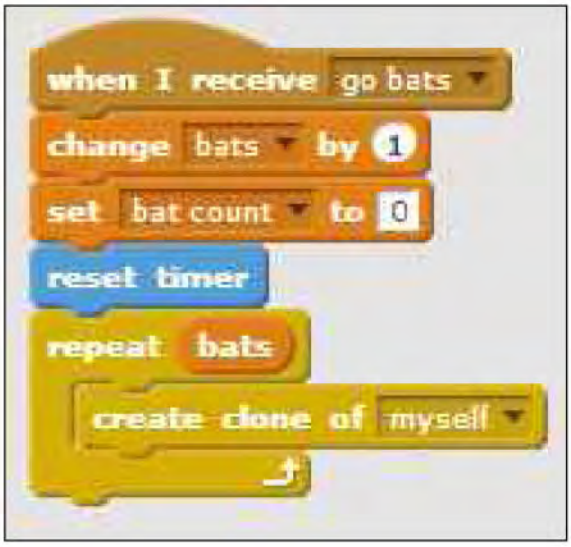
STEP 2 This initial stack sets the three variables to their starting values and hides our initial bat. However, it also uses the 'turn video on' block, which you'll find in the Sensing category. This starts the webcam up, and once your webcam kicks in you should see yourself or your surroundings on the screen.

STEP 3 Now click on the Stage, as we want to add a stack here. This one controls the general flow of the game, broadcasting a new message, which we'll call 'go bats' and resetting the timer at the start of each new wave of bats. We allow 10 seconds for the player to get rid of each wave. Once 10 seconds is past, the program will broadcast a 'game over' message. On the other hand, we want to check whether the player has got rid of the last wave of bats. The Bats variable controls how many bats are released in each wave. The bat count tracks how many bats the player has



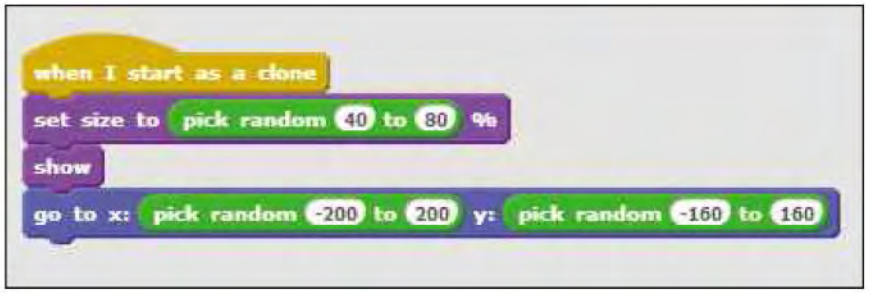
squished. When the two numbers are equal, all the bats are gone, so the game calls in a new wave with 'go bats'.

STEP 4 Here's the block that kicks in when that message is triggered. First, it adds 1 to the Bats variable, so that each wave has more bats than the last. Then it resets the Bat count variable to 0 and resets the timer. You've seen the



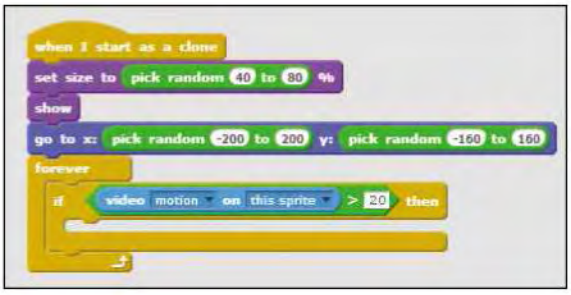
next loop before. It uses the Bats variable to control how many times the loop repeats, and so how many clones of the bat sprite will be created.

STEP 5 The next stack starts with the 'when I start as a clone' block, then adds a random element to the size of the bats, so that we'll get some small ones and some big ones. The Show



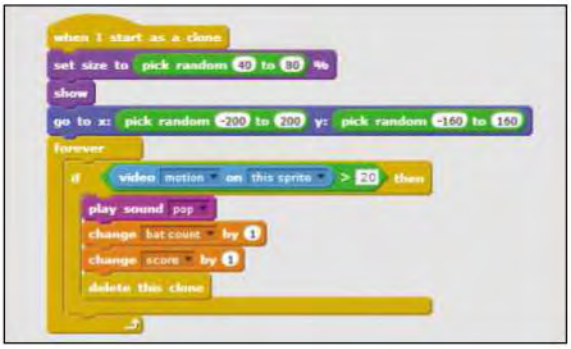
block reveals each clone, then the 'go to' block places each bat in a random location. The x and y settings, -200 to 200 on the horizontal axis and -160 to 160 on the vertical, stop the bats appearing too close to the edges.

STEP 6 The second half of this stack uses a Forever loop, then an 'if, then' block, so that the program is constantly checking to see whether the condition we're about to set is being met. In this case, we use the 'x > x' operator block, and drag in the 'video motion on this sprite' block. This



checks whether any motion underneath or around the sprite in question reaches a certain level, which we've set to 20. There's a bit of trial and error here, as so much depends on your webcam and your lighting. Once you have the game up and running, you might want to adjust this variable until the program works properly.

STEP 7 When there's enough motion to trigger the 'if, then' block, we get a pop sound, the Bat count variable goes up by 1 and the player's score goes up by 1. Then the clone is deleted.



Another bat bites the dust! Give the game so far a go, and see how well these primitive motion controls work out.

STEP 8 Finally, we need to set what happens when the player runs out of time. This last stack brings the original bat sprite to the front, changes the size, then shows a Game Over message for two seconds. It then tells the player what

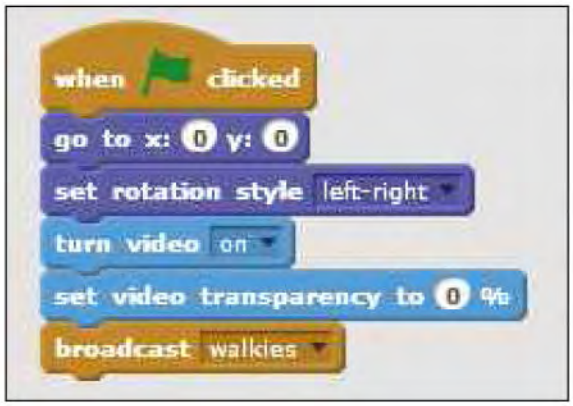
they scored. Here, the Join operator does the work, combining the text in the box after join with the current value of the Score variable as one message. Finally, the 'stop' block, set to 'all', brings our program to a close.

Project Two: Good Dog!

Our second project shows off the directional sensitivity of the Video sensing blocks. We've already seen the 'video motion on this sprite' block, but you can also set this to sense video direction. This works the same way as Scratch's sprite direction, so 0 is up, 90 is right, 180 is down and -90 is left. In this case, we're just going to focus on moving left and right, with an animated dog we can push along in either direction with a gesture.

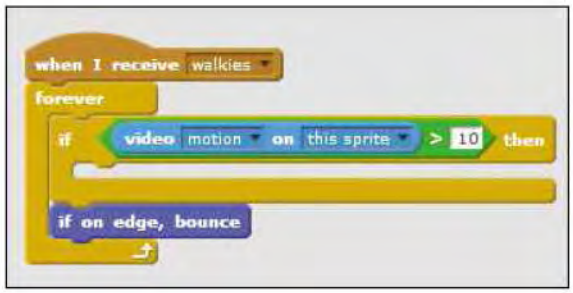
STEP 1 The project begins by importing a sprite from the library. Here, we're going to use Dog2, though any sprite with a walking animation will work. Again, there's no need for any backdrops, as the video feed from the webcam will fill in instead.

STEP 2 The first stack puts the dog in position and ensures the rotation style is set to 'left-right', so that our dog only moves along a horizontal line. We then grab two video-friendly blocks from the Sensing category. The top one switches on the webcam. The second sets video



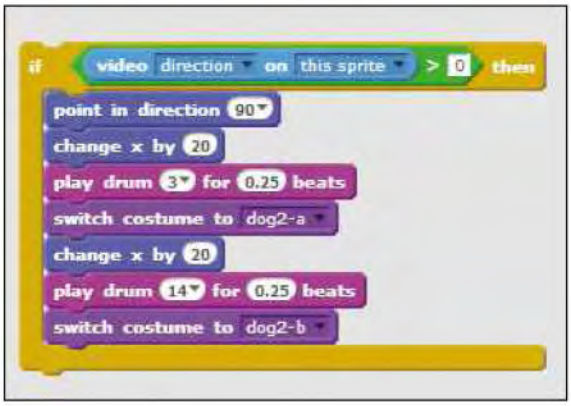
transparency to 0, so that the dog appears on top of the video feed, with no visible background whatsoever. Finally, the 'broadcast message' block sends a new message, 'walkies', which will start the stack that controls the dog's movement.

STEP 3 Here's the start of the 'walkies' stack. The 'forever' block keeps the blocks inside at work while the program is running, and we use an 'if, then' block just to check whether there's any motion on the webcam. If your webcam is being oversensitive, you can use the value here to make it a bit less sensitive. Just keep pushing the value up so



that the dog only moves when you want it to. Finally, the 'if on edge, bounce' block makes the dog turn around if or when he hits the edge of the screen.

STEP 4 We can now start constructing the rest of the stack. It's controlled by an If, then loop with an 'x > x' operator. Into this goes a 'video x on x' block from the Sensing category, which checks whether the direction of any movement around the sprite is moving left or right. If it's moving right, this chunk of script makes the dog face right and walk right,



using the kind of animation we used in our cartoon project earlier on.

STEP 5

Here's the second chunk of the stack. This does exactly the same thing, but it checks for movement going left, then turns the dog around and puts him walking left. Notice the

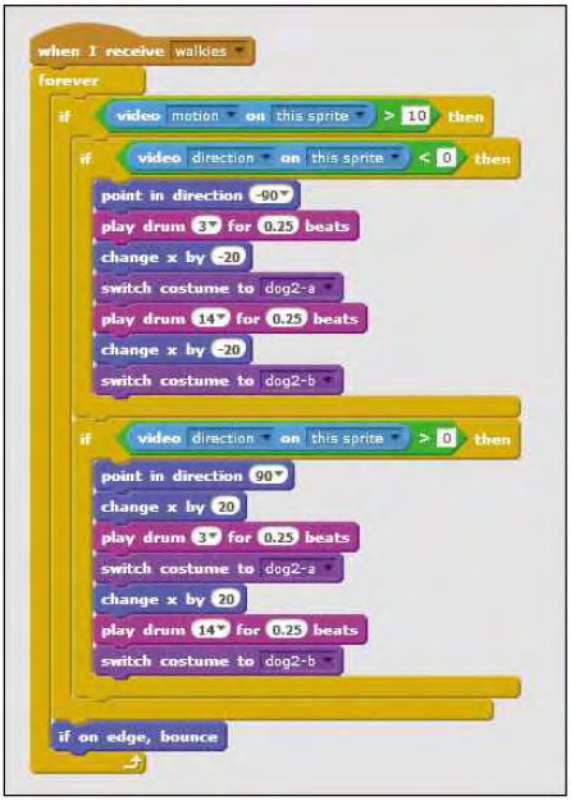
drum beats. These help control the speed of the walk and add a little noise to the animation, too.

STEP 6

The two chunks stack together, then the whole caboodle fits inside the first 'if, then' block. Give the program a spin. The dog should move left or right according to the

ABOUT VIDEO TRANSPARENCY

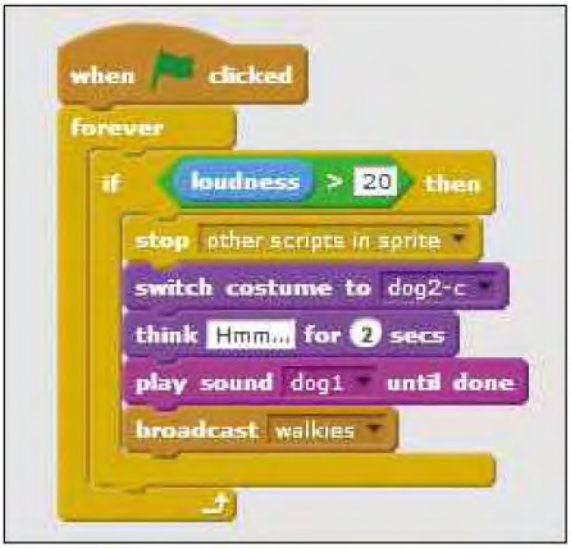
You can use the 'set video transparency to x' block to decide how much or how little of the video feed will be visible in the finished program. Set it to 0 and any sprites will be overlaid directly on the view from your webcam. Set it to 100 and you'll get the backdrop of the Stage with no video showing through. You can still use your webcam for motion control, but you won't see anything it's pointing at.



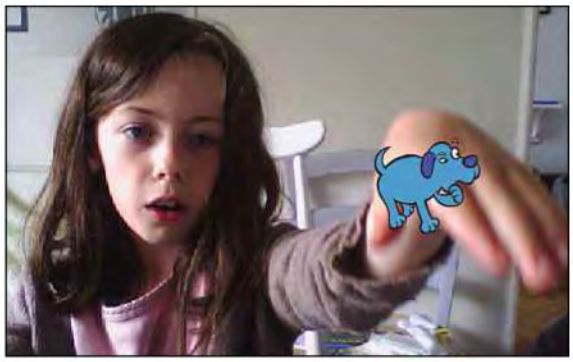
movement of your hand. Remember, if it's not working, keep adjusting the value of the 'video motion sensing' block at the top of the stack.

STEP 7

We're not finished yet, though. Did you know that Scratch can listen through your webcam or our laptop's microphone as well as watch? This last stack uses the 'loudness' block from Sensing. If this goes above the threshold we







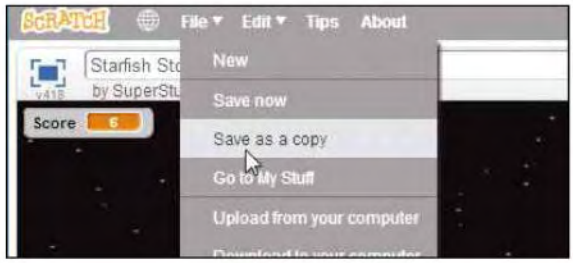

▶ Have fun taking your animated dog for walkies with simple hand gestures!

set using the 'x > x' operator, it stops all the other scripts running for a minute, changes the dog's costume to put it in a listening pose, makes him think 'hmm…' and then makes him woof. Broadcasting the walkies message gets him back moving left and right again.

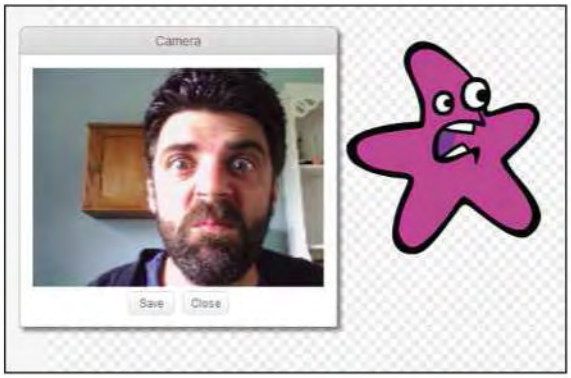
Project Three: Become a Space Invader

A webcam is great for simple motion control projects, but it's also brilliant if you want to bring objects, places or people from real life into your programs. Here, we're going to adapt our earlier Starfish Storm project and show you how to transform your mum, dad or mates into menacing space invaders.

STEP 1 First, you need to open up your Starfish Storm project. Go to the My Stuff page and click the See Inside button underneath the project. We don't want to permanently change the original, so click on the File menu and select Save as a Copy.



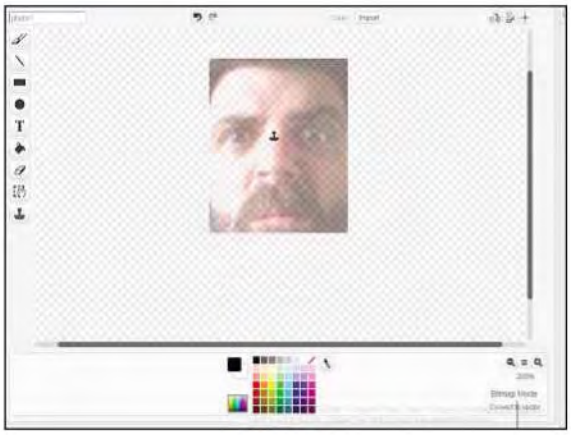
STEP 2 Now click on the main Starfish sprite, then on the Costumes tab. Look above the list of costumes and click on the Camera button to create a 'New costume from camera'. A window will appear showing you the current view from your webcam. Get yourself into position, then click the Save button.



STEP 3 You'll want to edit this costume before you start using it. Choose the Select button from the toolbox on the left, then drag the box over the square or rectangular area of the image you'd like to keep. When you're done, press the Ctrl+C

keys at the same time. This copies the selected chunk of photo into memory.

STEP 4 Now press the Clear button to empty the editing screen, then press Ctrl+V at the same time. Your selected chunk of photo will pop back into view. Position it over the cross in



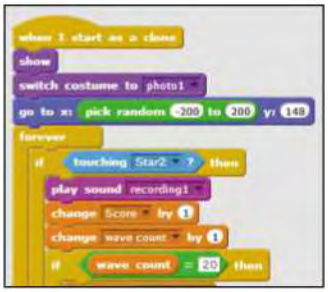
the middle of the Selecting screen, click to place it, then tap the Esc key.

STEP 5 You can edit the new costume if you want, adding neat little touches like devil horns, glasses, weird eyebrows or a nice moustache. For now, though, we'll leave it as it is. In this case, there's only one costume and there's no need to fiddle around with 'switch costume' blocks. If you need to, though, you can always add them to your script stacks, making sure they point to the costume you've just created (called 'photo1' unless you change it).

STEP 6 You don't have to stop at the webcam photos, however. Why not bring in samples of your own voice? Click on the Sounds tab, then the microphone-shaped button, 'Record new sound'. You'll see the Sound Editor. Press the round Record button to record, the square Stop button to stop and the Play button to play things back.

STEP 7 You can also click and drag on the soundwave in the window to highlight a section of your sound, then use the Edit menu to trim or copy sections. You can also add simple fade-in, fade-out or volume effects. Here, we're just cutting out unwanted sections from our sample.

STEP 8 When you've finished tinkering, you can simply change the 'pop' noise the invaders make when they're blasted to your new sample (recording1 unless you change it). Every time they're shot, you'll hear the noise – better make it a blood-curdling scream, then!



ANIMATE YOURSELF

You can always create simple animations by using the webcam to make a number of costumes of the same face in different poses or animations. You can then use an animation script like the one here to switch between the costumes at the right speed.

Share your projects

Scratch isn't just about building projects, but about sharing your projects with the Scratch community. The great thing is that it's very easy to do

WHAT YOU'LL LEARN

- How to share projects
- How to use tags to help other users find your work
- What studios are, and why you should use them

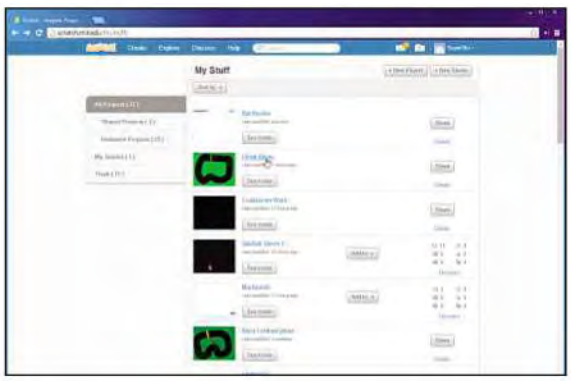
When you finish a project and you're proud of it, it's only natural to want to share it, whether that's with your family, your friends or all the other kids at school. Scratch makes that easy, but it also goes much, much further. Community is an important thing in Scratch, and enthusiastic scratchers, as they're called, like to share their work with other scratchers, not just so they can try a program out or play a game, but so that they can look at the code, maybe learn from it, or even suggest improvements. In fact, scratchers will even remix each other's projects, making them work more efficiently, or adding features and ideas that the original author might not have thought about.

Because Scratch itself is web-based, sharing a project is extremely easy. All it takes is a few clicks, and your projects could be being enjoyed and remixed by a legion of keen scratchers.

Sharing a project

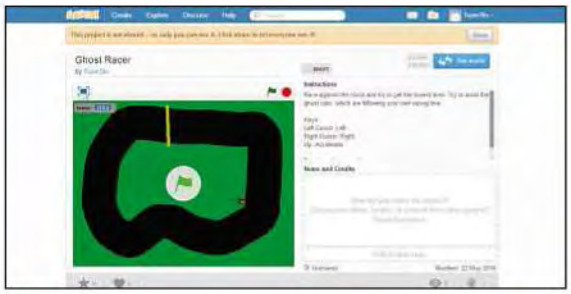
STEP 1 You can actually share a project at any time while you're working on it. Look at the top right of the main Scratch project interface and you'll see two buttons: 'Share' and 'See project page'. Click Share and your project will be shared, and accessible by anyone.

STEP 2 All the same, it's more useful if you set up a proper project page with any notes on the program, and any information on the controls or how to use it. If you're in the project



interface, just click on the 'See project page' button. If you're in the My Stuff page, just click on the project's name, where it's written in blue.

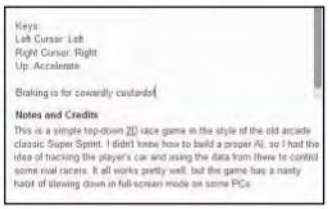
STEP 3 The left-hand side of the project page is dominated by a window, where clicking the green flag button will start your program running. On the right-hand side are two



textboxes, with one for instructions, and one for notes and credits. Click in a box to start typing. Use the Instructions box to tell users what a project does or how to win the game, and also note down any keys that need to be pressed.

STEP 4

Use the Notes and credits box to mention any ideas about the program, and to credit any games, programs, projects or scratchers that inspired it or have helped while you were trying to code it. The Scratch community is a caring, sharing kind of place, so if you've found something useful by looking at someone else's program it's only fair to give them credit.



STEP 5

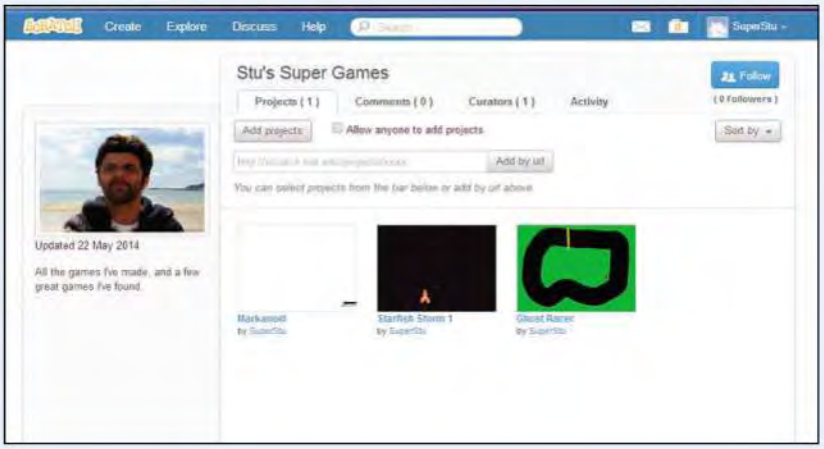
You can enter Project Tags in the box to help other scratchers find your project. You can only post a maximum of three, so think about them carefully. Basic categories like games or animation are already set up for you to pick, so try to give one to every project. Here, we're going for games, 'racing' and 'cars'. When you're done, press the Share button at the top and your project is shared.

STEP 6

The bar along the bottom has options that scratchers can use when they look inside or try your project. The Star allows them to bookmark your project, so they can find it easily later. The Heart is used to tell you and fellow scratchers that they like your project. You can see who has liked or bookmarked your project from the numbers.

The Studios button allows a scratcher to put your project in a studio (see above), while the Embed button is used to embed the project in another website, like a school website or a blog. Finally, 'Report this' is used to report any content that might break the law or upset someone. Still, you wouldn't do anything like that, would you?

ABOUT STUDIOS



Studios are collections of projects, organised by a type or a theme, or just by the fact that whoever runs the studio likes them. Anyone can create a studio by going to the My Stuff page and clicking the '+ New Studio' button. You can change the picture, add a description, then start adding projects to your studio. These don't have to be your own projects. If you like someone's work, you can add it to your studio and point other scratchers to it. Scratch users can follow other users and keep track of their new projects, but also follow studios, so by setting up a good studio full of good projects you can help promote the best.

Once you've created a studio, you can also invite people to work as curators on it. This empowers them to add projects or remove them from your studio, so you could have a whole team of you working together, adding your own projects to a studio or finding great projects to share. If you have a few friends using Scratch, it's a great idea.

STEP 7

The two icons at the right of the bar tell you how many scratchers have viewed your project, and keep track of any remixes. Click on this button and you can see any remixes listed, plus any remixes of those remixes. Produce something really good, and you might start a whole family of offshoots.

STEP 8

Once you've shared a project, others can leave comments on it. The Scratch community tends to be friendly, but if you get mean or snarky comments, you can turn off commenting with a quick tick in the checkbox. The two areas on the right list the most recent remixes, or any studios this project has been added to.

Remix your projects

If you can see inside a project, you can change and potentially improve it. Welcome to the wonderful world of Scratch remixes!

WHAT YOU'LL LEARN

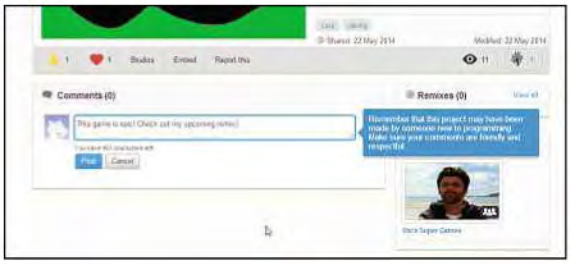
- What are Scratch remixes
- How to make your own remix

As we mentioned before, the Scratch community isn't just about sharing completed projects, but about sending projects out into the community so others can help you with a problem, improve your code, or add exciting, new features. It's even possible to take a project you like and use it as the basis for your own project; it's not stealing as long as you make sure the author gets credit. Scratch makes this easy with its concept of remixing.

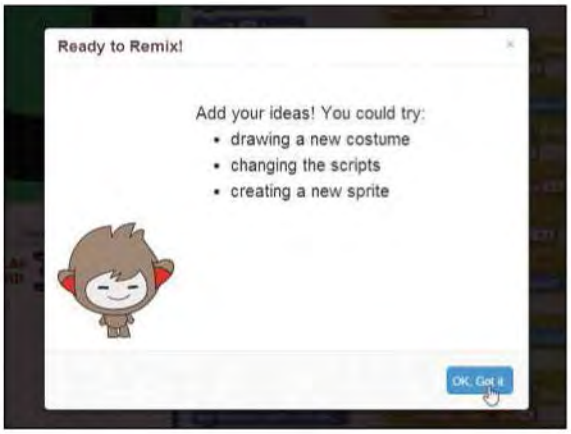
Remixes are versions of a project, usually made by a different scratcher, that might alter the original project. Scratch automatically tracks remixes as they're created, so that everyone can see which projects started where, and how many remixes a project has inspired.

Remixing the Ghost Racer game

STEP 1 Another scratcher, NinjaBee, has found our Ghost Racer game. She's played around with the game a little, and wants to see how it ticks. Having left a cheeky comment, all she needs to do is click the See inside

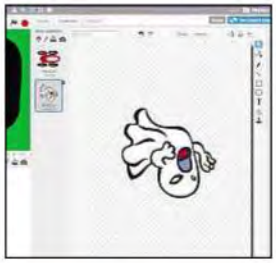


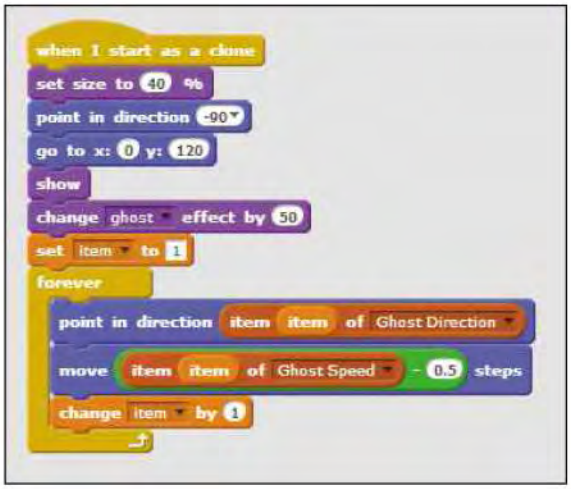
button to have a look at all the backdrops, sprites and code.



STEP 2 She can now see the code and even change it, but she can't save any changes to our original version. If she wants to do anything to the game, she needs to press the orange Remix button and start her own remix. If she does so, she'll see this message.

STEP 3 NinjaBee has decided to take the game's title literally, and change the costume for our ghost car sprites to ghosts. She's turned the costume upside down, as otherwise the ghost will be upside down when it first appears.





STEP 4 Now she's messing around with the ghost's script. She's added a 'Set size to x?' block to make the ghosts larger and changed the colour effect to a ghost effect. We've now got big ghosts you can actually see through racing around on the track.

STEP 5 That's not all. She altered the Go Ghost script, so that the ghosts only wait two seconds before spawning, and changed the operator block and the values in the

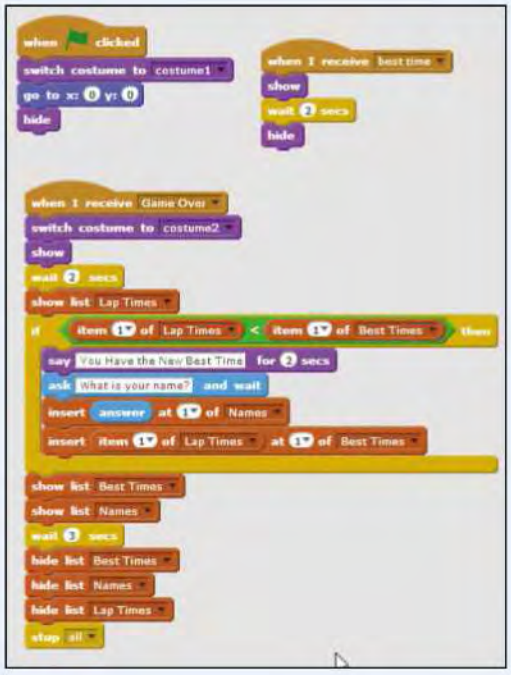
'move' block at the bottom of the 'when I start as a clone' script. The ghosts are now travelling faster than the car.

STEP 6 And now she's changed the script for the car sprite so that, when a ghost catches up with it, it plays the 'zoop' sound and loses speed altogether. You're no longer trying to race your way through the ghost cars, but instead trying to speed away from actual ghosts!

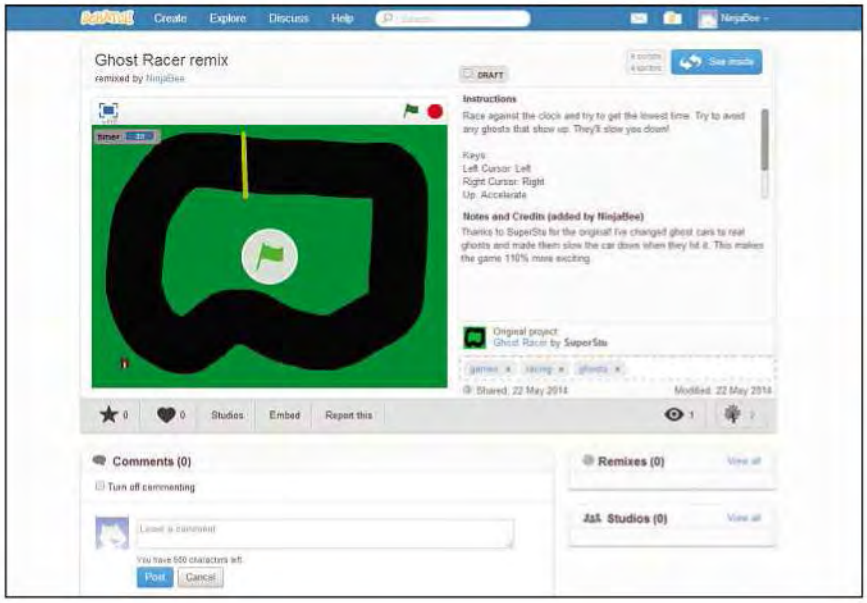
MAKE YOUR OWN REMIX

Why not make your own remix of the Ghost Racer game (or KillerBee's alternative)? Change the top speeds of the car or the speed of the ghost cars. Change the variables to put more ghosts on the track. Try different backdrops with different track layouts, or add two types of ghost car with different speeds. Try adding a brake control or engine noises to the game. The beauty of remixing is it's easy to come up with your own version.

Here, we've made a few changes to the scripts for the Stage and the New Lap Record sprite, as well as giving the New Lap Record a new costume to say Game Over. See if you can see what we've done, and work out what we're trying to do. Is there any way you could do it better? Why not give it a try?



STEP 7 With her remix finished, KillerBee is off to the project page, where she changes the instructions, notes and credits before sharing her remix with a click of the Share button. The finished project page looks something like this.



My awesome Scratch game

It's time to harness all of our Scratch skills to make one final and (fairly) awesome retro-style arcade game

WHAT YOU'LL LEARN

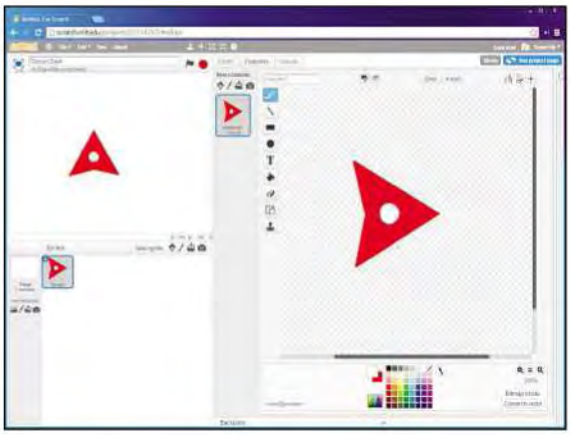
- How to improve performance
- How to make your own blocks
- Add a new wave message

Colour Clash is a retro-style arcade game where the player pilots a triangular starship that leaves a destructive, red trail. Your objective is to destroy the fiendish, blue starships by enveloping them in your trail. Your ship runs on energy, which you capture from the blues when you destroy them. Run out of it and it's game over. Destroy enough of the blue ships, however, and you'll launch a new wave, giving you a handy energy boost and even more blue starships to destroy. The only complication? Hitting the blue starships saps your energy. You might survive the odd brush, but do it too often and you'll find yourself out of gas.

OK, so it's not exactly Minecraft, let alone Super Mario, but what do we need to make it work?

1. A red starship that leaves a deadly trail
2. Blue starships to destroy
3. An energy meter
4. A way of launching new waves and setting how many blue starships will be in each wave
5. A way of keeping score
6. When a blue starship hits the energy trail, they need to be destroyed, boosting the score and the player's energy
7. When the red starship hits a blue starship, it needs to lose energy
8. When all the blue starships are destroyed, we need a new wave
9. When the red starship runs out of energy, we need to end the game and say 'game over'.

That's a pretty good working framework, so let's put it into action.

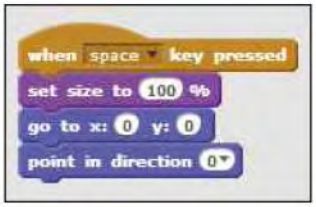


STEP 1

We'll start by setting up the sprite the player will control. Press the 'Paint new sprite' button and create it line by line. Our design is deliberately simple, but if you want something different, go ahead.

STEP 2

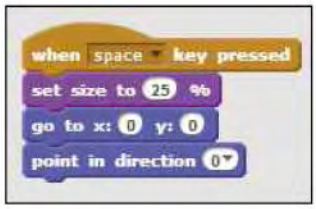
We'll start by getting the movement right. We're going to handle our ship using the mouse, and control its speed with a variable, so we need to go to Data and hit the Make a Variable button, and call that variable 'speed'. Once



TOP TIP

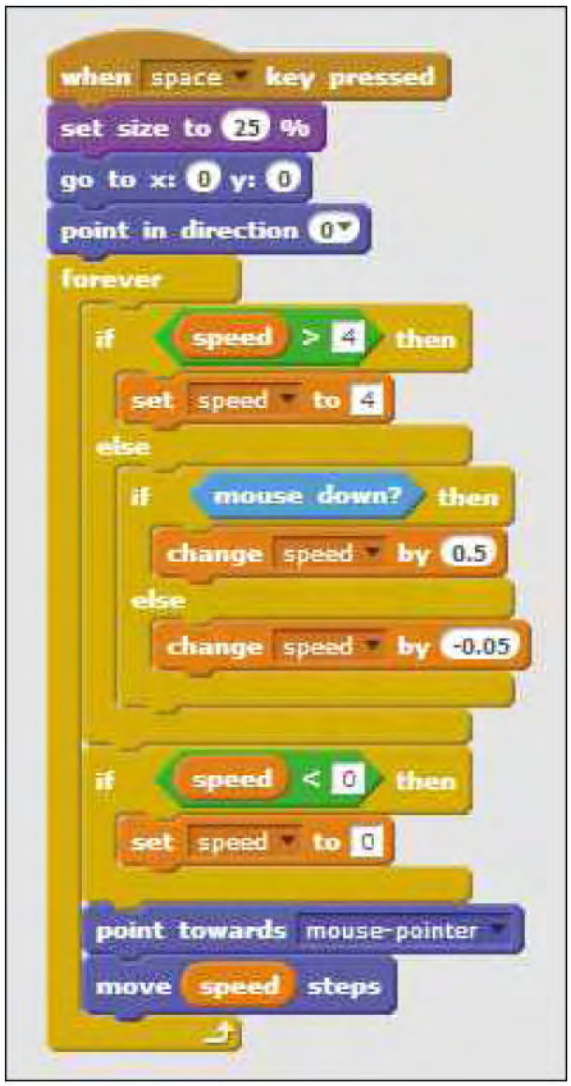
As Energy is crucial in this game, you might want to tick the checkbox next to the variable to make it visible on the screen. Right-click on the counter to put it into Large readout mode.

that's done, we start off with the necessary blocks to set the size, the starting position and the starting direction.



STEP 3

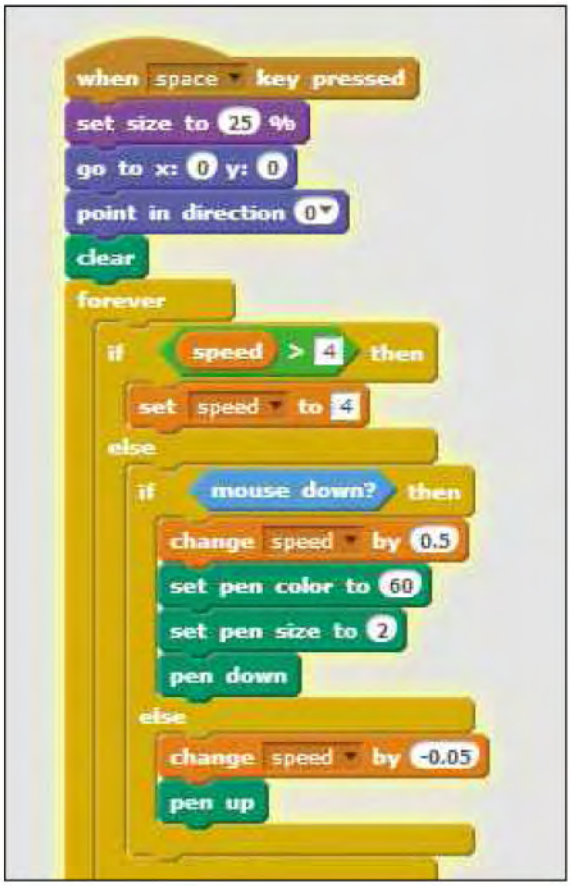
This is the basic movement stack for the ship. It's actually a variation of the script we used in the racing game, but instead of using the cursor keys to control direction, the ship points in the



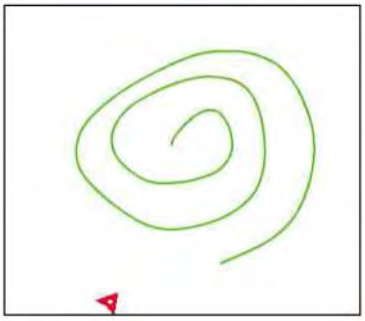
direction of the pointer and accelerates when we press the mouse button. If you take a good look at the stack, you can see how we use the Speed variable to control the speed of movement, and how pressing the mouse button (the 'if mouse down' condition) increases the speed to a maximum of 4.

STEP 4

Now we need to put the trail in place. For this, we use Scratch's Pen feature, just like we did in the Pattern Generator project earlier. You can see where we've put the three green pen blocks so that, when the mouse if pressed, a line is



drawn with the pen colour set to 60 and the size to 2. We've also put a 'pen up' block below the 'else', so that the pen won't draw if the ship isn't accelerating, and a 'clear' block near the top of the stack, to clear the Stage of lines every time we run the program. Give the program so far a try. Press the green flag, then press the space key.



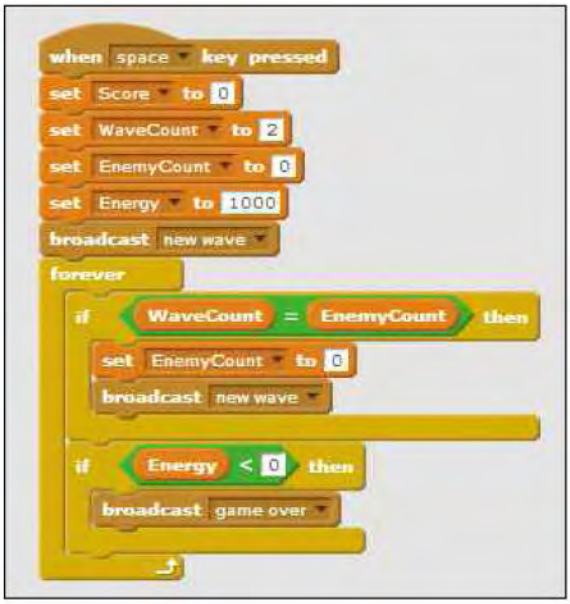
STEP 5

With the ship up, speeding around the Stage and leaving a trail, it's time to start work on the rest of the game. First, click on the 'paint new backdrop' icon and fill the Stage with

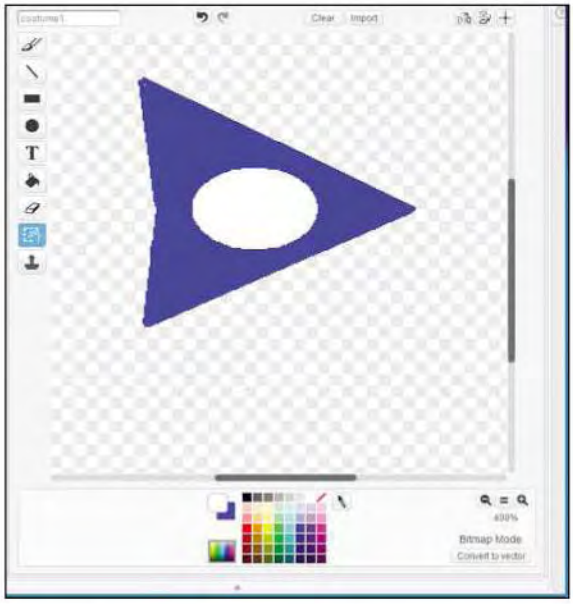
black paint using the Paintcan tool. We're going to need this later. Now click on the Scripts tab, as we're going to control some of the core game functions from the Stage itself. Now we need to set up four new variables, which all of our sprites will share. We'll call them 'EnemyCount', 'Energy', 'WaveCount' and 'Score'. Set them all to 'For all sprites' as you make them.

STEP 6

Here's the Stage's main stack. As you can see, the first chunk just sets up the different variables, then broadcasts a new message, which we'll call 'new wave'. The second chunk is a 'forever' loop that checks for two conditions.



In this game, we track the number of enemies launched in each wave with the variable WaveCount and the number of enemies destroyed by the player with the variable EnemyCount. The first 'if, then' block checks whether all the enemies in the wave have been destroyed, then launches a new wave. The second tracks the player's Energy, and if it slips below 0 broadcasts a 'game over' message, which will launch a whole Game Over routine.

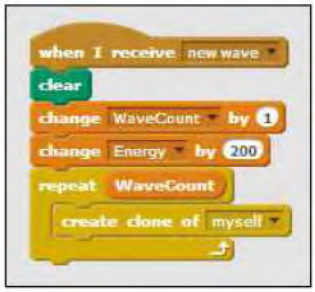


STEP 7

OK. Let's put some enemies on the screen. Again, the sprite is a pretty simple design, and the first stack isn't too complex, either. It tells the sprite to scale to 30%, then hide, waiting to spawn a small army of clones.

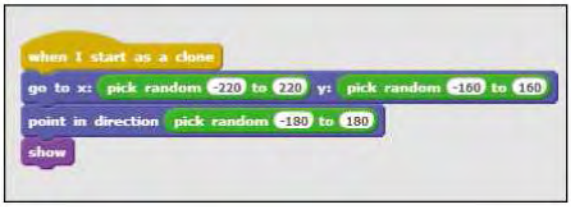
STEP 8

This is where our second stack kicks in. This waits for the 'new wave' message. This first clears any pen lines from the Stage, then changes WaveCount so that this new wave will spawn one more enemy ship than the last wave. The 'change Energy' block gives the player's Energy level a boost, and after that the 'repeat' block creates clones according to the current value of WaveCount. If WaveCount = 3, then three clones will be spawned. If WaveCount = 4, four clones will be spawned, and so on.



STEP 9

But what happens when they're spawned? First of all, we want them to spawn in a random location, facing a random direction. As always, we do this by putting 'pick



IMPROVING PERFORMANCE

Certain things really slow Scratch down. Have too many 'forever' loops in a program and you can slow it to a crawl. Keep applying different colour or ghost effects to too many sprites or clones, and your whole program can slow down. If a program you're writing is running badly, have a good look through and see if you can find ways of making it more efficient. For example, you might find you have two sprites set to detect each other when you only need one, or you might have instructions running when a sprite is still that only need to be running when it's moving. By fixing these issues, you could speed up your program. We call this process optimisation.

random' blocks in the x, y and 'direction' spots on the 'go to' and 'point in direction' blocks. The 'show' block then reveals the brand-new clone.

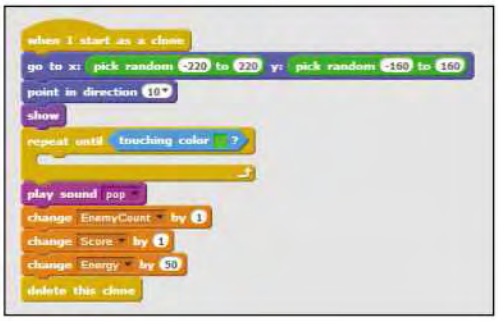
STEP 10

Next, we're going to use a 'repeat until' block. This block keeps on doing the same things until a certain condition is met. Here, it's when the sprite touches something green, like the line being drawn by the player's spaceship. To set the block to this colour, run the

program and trace some green lines around the Stage. Stop the program, then click on the little square in the 'touching color x?' block. Now, being very careful, click on a green line to sample the colour. If you look, you'll see that the colour of the little square changes colour as it's over certain colours. Just watch for it to turn green as you move the mouse around, then click.

STEP 11

When the ship hits the trail, we want it to disappear, adding 1 to the EnemyCount and Score variables, and giving the player's ship a small energy boost. All it takes are three 'change variable by x' blocks and a 'delete this clone' block, which we put last to make sure the other blocks have time to run. We also add a 'play



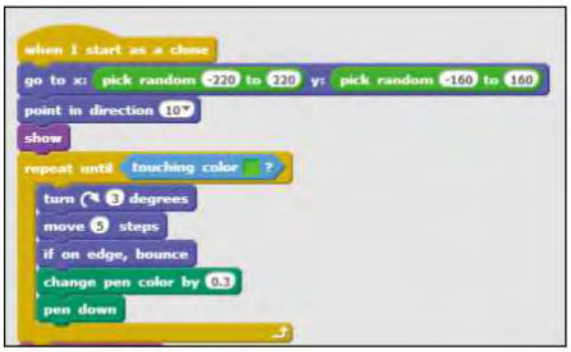
MAKING BLOCKS

Making your own blocks to define subroutines can be incredibly useful. You save yourself work as you don't need to keep adding the same blocks over and over again, but you can also make your program more efficiently. Think carefully about your program, and ask yourself whether there are instructions that it might be doing over and over again, and whether you can cut these down by producing custom blocks, and using those instead.

sound' block and set it to the 'pop' effect, which is added automatically to every new sprite. This whole stack attaches to the bottom of the 'repeat until' loop, as we don't want any of these things to happen until our cloned enemy ship hits a green line.

STEP 12

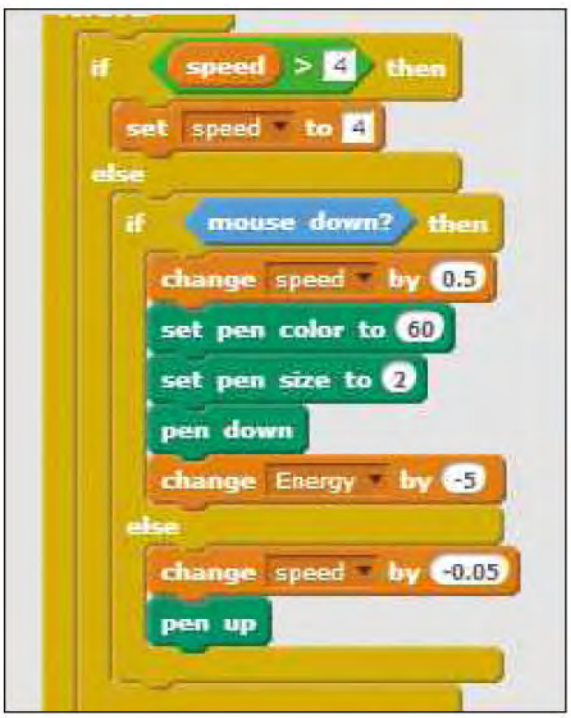
Until that happens, we just want the enemy ships to move. That's handled by a Turn block and a Move block from the Motion category, along with an If on edge, bounce block. We



can change the behaviour of the ships by using different values here. For now, 3 for the 'turn' block and 5 for the 'move' block gives us nice, graceful turns.

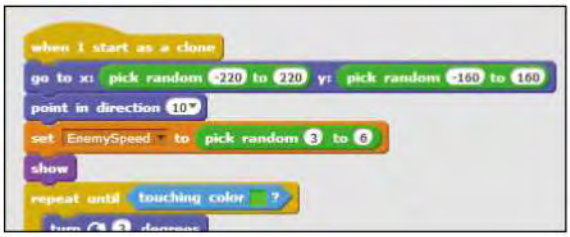
STEP 13

We've actually got the basics of the game in place now, but if you try it you'll soon notice one thing: it's all too easy. What we need is risk – the risk that the player's ship will run



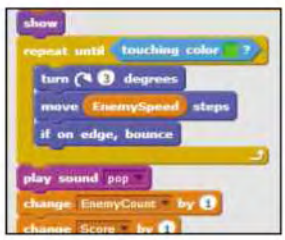
out energy, and that this will mean game over. That's easily done. First, add a new 'change variable by x' block to the player ship's main stack, right underneath the 'pen down' block. Set it to 'Energy' and set the value to -5. Now, whenever the ship is accelerating, it's losing energy.

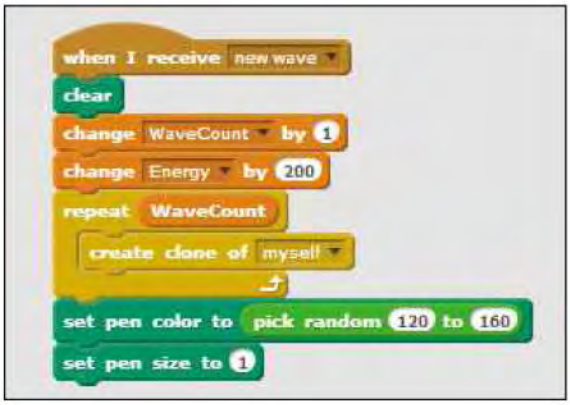
STEP 14 We also want energy to drain away when it hits one of the blue ships. Add another 'if, then' block to the stack, and set it to change 'Energy by -50' when this sprite is touching the enemy sprite.



STEP 15 Now let's make those enemies a little less predictable. First, click on the enemy sprite in the Sprites area, then go to Data and add a new variable. We'll call this 'EnemySpeed', and we need to set it to 'For this sprite only'. Next, slide a new 'set variable to x' block into place near the start of the 'when I start as a clone' stack, just beneath the 'go to' and 'point in direction' blocks. Set it to use the new EnemySpeed variable, and use a 'pick random' block for the value, set to minimum and maximum values of 3 and 6.

STEP 16 Just grab the marker block for the EnemySpeed variable and use it to replace the value in the 'move x steps' block. Each cloned enemy ship will now have its own random EnemySpeed value, which controls how fast that ship will move.

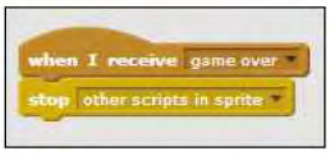




STEP 17 Let's add one final visual flourish. Go to the Pen category and add a 'set pen colour to' block and a 'set pen size to' block to the 'when I receive new wave' stack. Use a 'pick random' block for the value of the first block, set to a minimum of 120 and a maximum of 160. Use a value of 1 for the 'set pen size'.

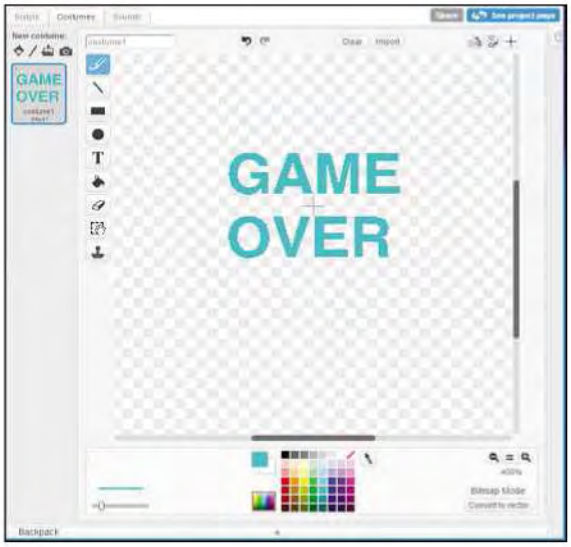
STEP 18 Next, add a 'change pen color' block and a 'pen down' block to the 'When I start as a clone' stack. Drop them carefully inside the 'repeat until' loop. Now, each wave of enemy ships will leave a trail with a different colour, and that colour changes as the ship flies around. The colour is set for the whole wave in the 'when I receive new wave' block, then changed by the 'change pen color' block by the stack that moves each clone.

STEP 19 We now have a reasonably tricky game. The next step is to set what happens when the player's Energy hits 0, and the game is over. First of all, we want the sprites to stop what they're



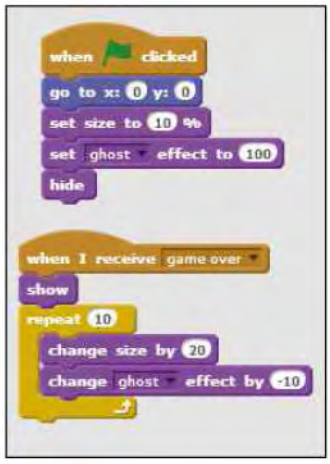
doing, so that the game grinds to a halt. All you need to do is add this stack to each of the sprites, including the Stage, where the 'stop' block should read 'stop other scripts in sprite'.

STEP 20 It's time to create a new sprite using the 'Paint new sprite' button. For this one, you just use the Text tool and type 'Game', then hit the Return key, and type 'Over'. You can



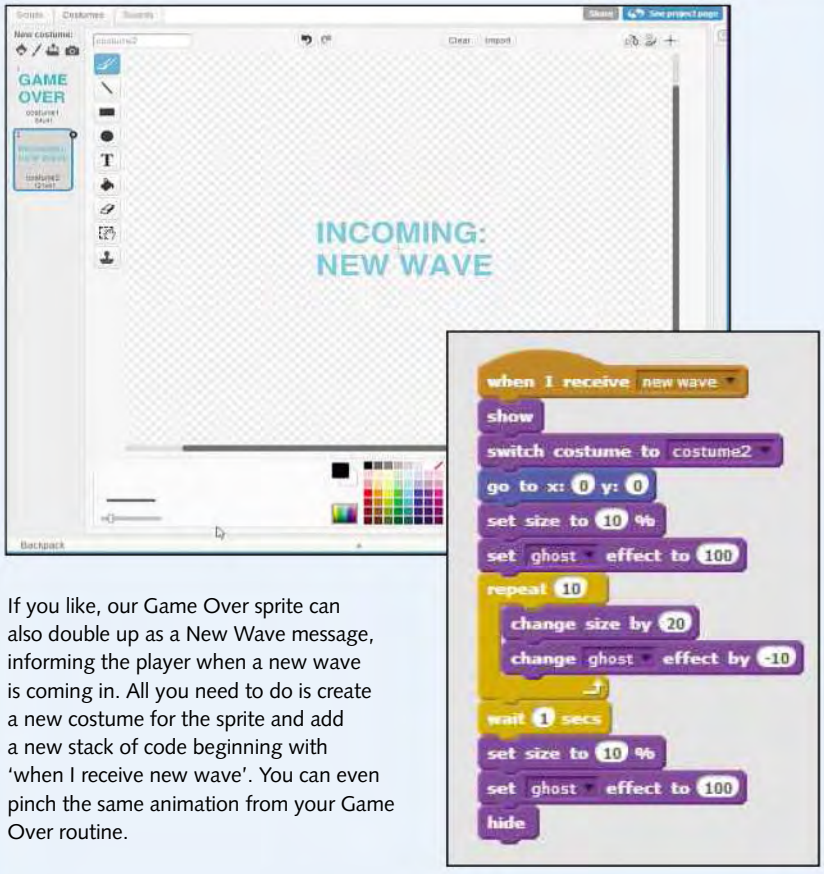
choose whatever colour you like. When you're done, click on the Scripts tab, and we'll start putting together the main Game Over script for our game.

STEP 21 The first stack puts the Game Over message in position at the centre of the screen and reduces the size to 10%. It also adds a Ghost effect, set to 100 to make the message entirely transparent. The Stage is set for a slick effect. The second stack makes the message bigger and less transparent with every repeat, until it pops out, large as life, in the middle of the screen.



STEP 22 That's good, but wouldn't it be better to tell the player what they scored? In fact, wouldn't it be better still if we had some kind of high score table? To pull that off, we're

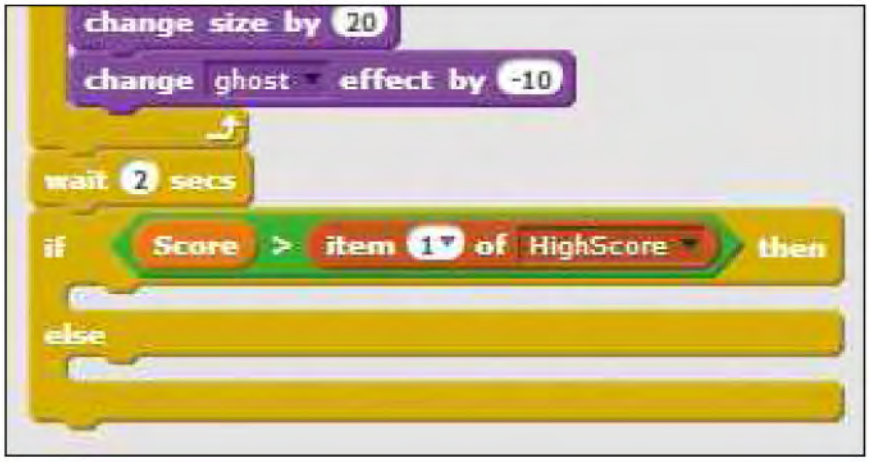
ADD A NEW WAVE MESSAGE



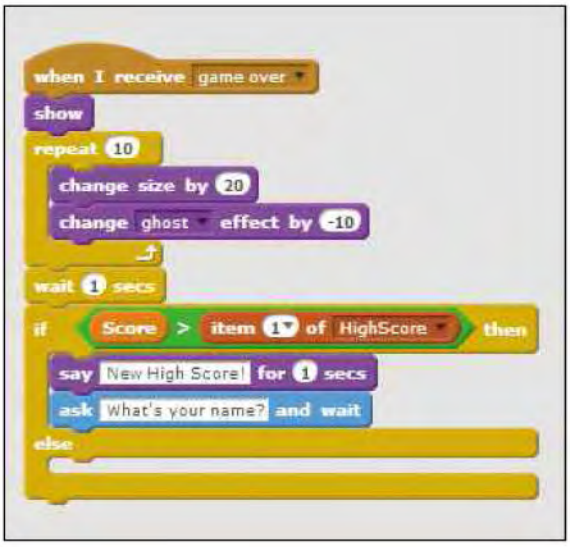
If you like, our Game Over sprite can also double up as a New Wave message, informing the player when a new wave is coming in. All you need to do is create a new costume for the sprite and add a new stack of code beginning with 'when I receive new wave'. You can even pinch the same animation from your Game Over routine.

going to have to use lists. Start by making two new lists, which we'll call HighScore and Names. Untick the checkbox next to each list's block to hide it from view.

STEP 23 Pull in an 'if, then, else' block, then add the 'x > x' operator next to the 'if'. Stick the block for the Score variable into the space for the first value, then grab the 'item x of list'

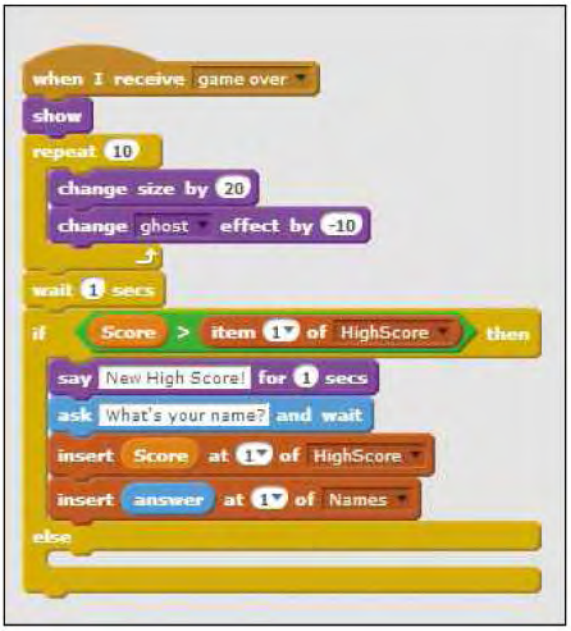


block and put that in where the second should go. Keep the item at 1 and make sure the list is set to 'HighScore'. All we're doing is telling the program to look at the current score and at the current high score, and if the score is higher, to do what we tell it to next.



STEP 24 What's that? Well, first of all we want it to say 'New High Score!', then we want it to ask the player for their name. For that, we use a regular 'say for x secs' block, followed by an 'ask and wait' block from the Sensing category.

STEP 25 The next two blocks then insert the new high score at the top of the HighScore list, and the name that's typed in at the top of the Names list. As long as nothing goes wrong, the two will be linked, so whoever has the highest score – and so the score at position 1 of HighScore – will have their name at position 1 of Names.

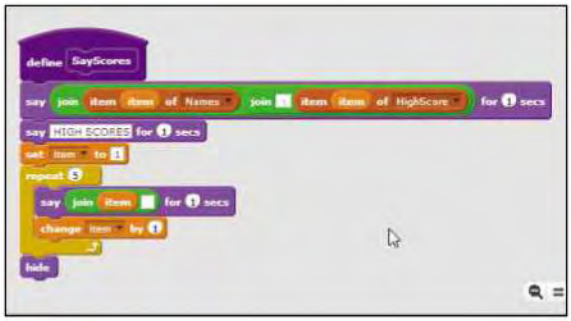


STEP 26 It's time to show you one last Scratch trick. We want the game to end with a little high-scores table whether the player has a new high score or not, but why bother adding the same blocks twice? Scratch has a useful Make a Block feature, where we can create a useful stack of code from other blocks, then call that stack – or subroutine in programming language – with a single block. Go to More Blocks, click the Make a Block button and type 'SayScores' into the box. You'll see that a new block, 'define SayScores', has appeared in the Scripts area.

STEP 27 This first block simply tells the player their score. As you can see, we've used the 'join' operator here. This useful block can be used to join some text and a

variable, or even two variables, in a sentence or a statement. In this case, it says 'Score: ' followed by the current value of the Score variable.

STEP 28 You might remember this next trick from the Racing Game project. Create a variable called 'Item', then add these blocks of script to the bottom of the stack.



The trickiest bit is the 'say' block, which uses multiple 'join' operators to say the current value of the Item variable, followed by the name of the player it refers to, followed by that player's score. You might need a couple of tries to get this right, so make sure it matches what you see here.

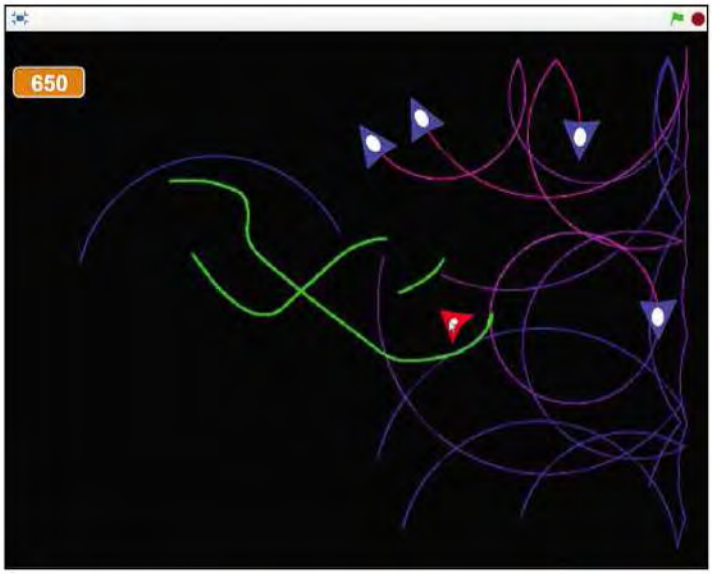
STEP 29 We've defined what the SayScores subroutine is and what it does. Now we can put it to good use. Just drag our new SayScores block into position both at the bottom of the 'if, then' section of the 'if, then, else' block, and the 'else' section. We now get a readout of high scores at the end of every game. Why not give the game a go now? It should be ready for you to play!

EXPERIMENT

There are loads of different things you can try out to make this game even better and more exciting. Why not add some sounds? Or how about adding some music? You could also try out different values for the movements of your ship and the enemy ships, or try different amounts of Energy added or lost when you finish a wave or when you collide.

Every change you make will have an impact on the game. If you want to preserve the original, make sure you use Scratch's Save a Copy feature to save one version as your best version, and another version as a testbed for your tinkering. Go on, have some fun and see what you can come up with!

▼ **Colour Clash is now ready for action. Give it a go!**



Your next steps in coding

You're well on your way to mastering Scratch, so where do you head next on your programming adventure?

It might be simple, but Scratch can create a wide variety of games and animations. Eventually, though, you'll want to create something that can work on a bigger screen or at a higher resolution, or can use more advanced graphics or even 3D. That's not a problem if you're ready to move onto a proper textual programming language, but if you're not there are alternatives.

Alice

Like Scratch, Alice is a graphical coding environment designed to introduce novices to the joys of coding. Its tiles work a little like Scratch's blocks, and it's not difficult to move between the two. However, while Scratch is designed to handle 2D games and animations, Alice works with 3D models in 3D scenes. That doesn't mean you can create the next Pixar movie with Alice, but it does mean you'll get a good introduction to 3D graphics as well as simple logic. In fact, the instructions used in Alice are designed to reflect the standard statements used in mainstream programming languages like C++, C# and Java. It's a good choice if you want to work with something that's a bit like Scratch, but gives you more to get your teeth into.

● www.alice.org

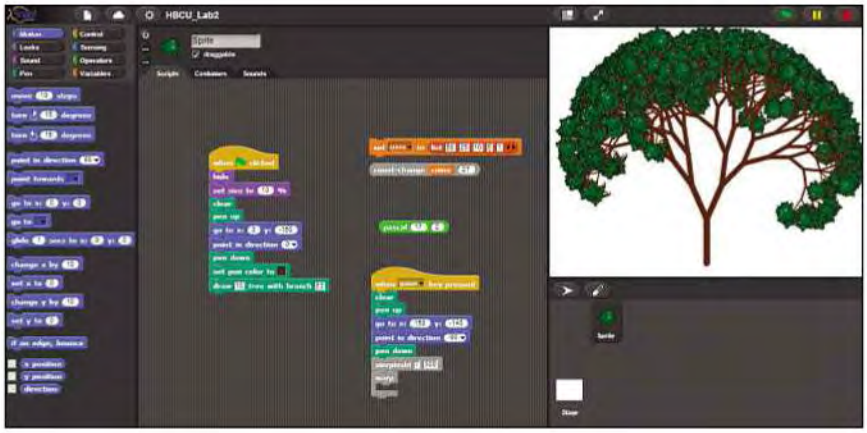
▼ Like Scratch, Alice is aimed at programming novices, but instead of 2D it works with 3D models in a 3D environment.

▲ MIT App Inventor is similar to Scratch, but with the bonus that you can produce apps that run on Android smartphones.

MIT App Inventor

Developed with the aid of Google, MIT App Inventor might be seen as a 'grown up' Scratch. It's another block-based programming tool, and if you're used to Scratch you'll find a lot that's familiar. It's a lot more complex, not to mention intimidating to start with, but the effort could be worth it. That's because App Inventor produces apps that can run on Android smartphones, so you can use it to produce functional programs. It also means you can play with the built-in features of a smartphone, including the motion sensors, cameras and the touchscreen. If you're starting to outgrow Scratch, App Inventor could be for you.

● appinventor.mit.edu/explore/



▲ SNAP! is a great step up from Scratch with extra capabilities for more advanced coding.

SNAP!

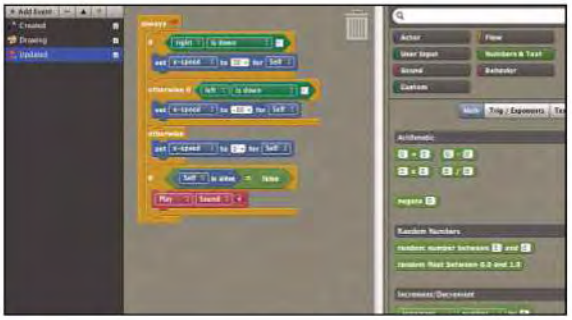
Formerly known as BYOB (or Bring Your Own Blocks), SNAP! is based on Scratch and could be described as an extended version. It runs along similar lines and uses many of the same blocks, but adds extra capabilities for more advanced coding. Like Scratch, it can run in an ordinary browser window, and runs in JavaScript. SNAP! supports more technical features like first-class functions, functional programming and recursion, which won't mean much to novice coders, but will eventually mean a lot to budding computer scientists. Its user interface and general style make it feel like a Scratch for older students. SNAP! programs can be made faster and more efficient than their Scratch equivalents; you can probably replicate most of what you can do in SNAP! in Scratch, but it might take a little longer and require some ingenuity. It's a great step up from Scratch.

● **snap.berkeley.edu/**

Stencyl

Stencyl builds on the principles of Scratch, but adds a mass of features aimed at budding games designers, including editors for sprites, and the tiles and terrain that make up the scenery, plus script blocks that support more of the specific needs of games. Like SNAP!, Stencyl allows you to make your own blocks and share them. It's a little more complex to learn than Scratch, but there are enough similarities to soften the learning curve. Best of all, Stencyl games can run on iOS and Android tablets and smartphones, although the free version will only let you test them. Stencyl has been used to develop successful Flash games, not to mention games on the iTunes App Store and the Google Play Store. If you want to get further as a games developer, Stencyl may be your ticket to the big time.

● **www.stencyl.com/**

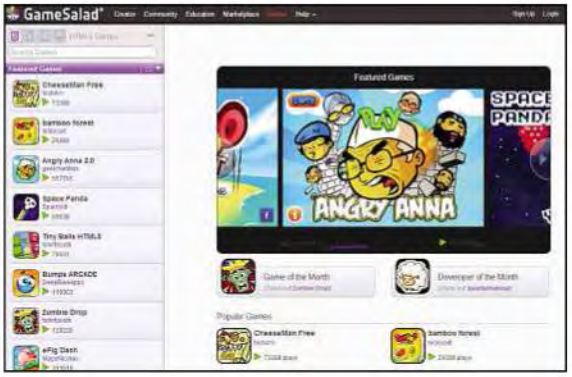


▲ Stencyl is aimed at budding games designers, with the benefit that Stencyl games can run on iOS and Android smartphones.

GameSalad Creator

GameSalad Creator is another visual coding tool for games development, where you set up scenes, bring in sprites (or actors), and set up the game logic in a special backstage panel. With its more object-oriented approach and its rules, physics-based properties and behaviours, it works in a different way to Scratch, but it's possible to produce some impressive 2D games, as you can see from the efforts showcased in GameSalad's Featured Games Gallery. What's more, GameSalad games will run on Android, iOS and Kindle devices, although you can't publish Android or Windows games to an app store without an expensive Pro licence. ●

● **gamesalad.com/creator**



▲ Some impressive 2D games have been produced using GameSalad Creator, as showcased here.

TEXTUAL CODING

These graphical coding environments are great for learning, and great for putting programs together fairly quickly. In the end, though, most novice programmers will have to get their hands dirty with some good, honest code. There are various ways of making the leap, but a handful of environments have been created specifically to help new programmers – and particularly young new programmers – get stuck in. We'll be looking at one of these, SmallBASIC, in the next chapter, but it's also worth looking at Hackety Hack, which uses a graphics toolkit, shoes and a series of lessons to teach you the rudiments of a useful language, Ruby. KidsRuby is another alternative, providing an extremely simple environment for learning the language.