Programming with Scratch

A step-by-step guide, linked to the English National Curriculum, for primary school teachers

Written by Neil Rickus (Computing Champions)

http://computingchampions.co.uk
@computingchamps
Contents

Activity 1 – Scratch Conversations ................................................................................................................. 3

Optional activity – Broadcast messages ............................................................................................................. 7

Activity 2 – Maths Quiz ....................................................................................................................................... 8

Activity 3 – Maze Game ...................................................................................................................................... 12

Activity 4 – Moving sprite game ....................................................................................................................... 17

Activity 5 – Platform game ................................................................................................................................ 24

License

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license. The license can be viewed at: http://creativecommons.org/licenses/by-nc-sa/4.0/

Acknowledgements

Parts of this work build on the Scratch Planning examples provided by Phil Bagge at http://code-it.co.uk/ in accordance with the Creative Commons Attribution-NonCommercial 3.0 Unported (CC BY-NC 3.0) license. The license can be viewed at: http://creativecommons.org/licenses/by-nc/3.0/
Programming with Scratch

Activity 1 – Scratch Conversations

Computing National Curriculum areas covered (all Key Stage 2):

• design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts
• use sequence, selection, and repetition in programs; work with variables and various forms of input and output
• use logical reasoning to explain how some simple algorithms work and to detect and correct errors in algorithms and programs

Task overview:

• Familiarisation with the Scratch interface
• Create a conversation between two characters using Say blocks

Possible cross-curricular links:

• Literacy – speaking and listening; drama; speech punctuation; feelings and emotions of characters
• History – interviewing historical figures
• Geography – conversation between two people from contrasting environments
• PSHCE – discussing feelings towards an issue, such as bullying
• Science – highlighting misconceptions about a particular topic
Activity 1 – Scratch Conversations

The Scratch website can be accessed at: http://scratch.mit.edu/ Click on Create to load the coding environment.

The Scratch environment has four screen sections:

- **Script Area** – contains your program’s code
- **Block Palette** – contains sections of code (called Blocks), which can be dragged into the Script Area
- **Stage** – where your program’s actions take place
- **Sprites Area** – contains details of the characters (called Sprites) in your program
Our first task is to create a conversation between two characters. Before working at the computer, you may wish to get the children to think about what their characters will say. In class, it is useful to model the first few instructions on an Interactive Whiteboard at the same time as the children.

<table>
<thead>
<tr>
<th>Task explanation</th>
<th>Required steps</th>
<th>Screenshots</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firstly, delete the cat sprite so we can use something more interesting</td>
<td>Click with the right mouse button on the cat in the <em>Sprites Area</em> and select <em>Delete</em></td>
<td>![Image]</td>
</tr>
<tr>
<td>Next, we’re going to choose our characters</td>
<td>Click on the <em>Choose new sprite from library</em> button at the top of the <em>Sprites Area</em></td>
<td>![Image]</td>
</tr>
<tr>
<td>You can move the characters around the screen so they’re where you want them</td>
<td>Select a sprite from either the <em>Animals, Fantasy</em> or <em>People</em> sections and press <em>OK</em></td>
<td>![Image]</td>
</tr>
<tr>
<td>Repeat to add a second <em>Sprite</em></td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>We now need a more exciting background</td>
<td>Click on the <em>Stage</em> (left-hand side of the <em>Sprites Area</em>)</td>
<td>![Image]</td>
</tr>
<tr>
<td>Make sure you select a background to make the <em>Sprites</em> stand out</td>
<td>Click on the <em>Choose backdrop from library</em> button</td>
<td>![Image]</td>
</tr>
<tr>
<td>It’s now time for our sprites to have a conversation. To do this, we need a <em>Green flag</em> to start the program (<em>Script</em>) and a number of instructions in order</td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>Click on your first <em>Sprite</em> in the <em>Sprites Area</em>. Click on <em>Scripts</em> at the top of the <em>Block Pallet</em> if the <em>Script Area</em> is not visible</td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>At the top of the <em>Block Pallet</em>, click on <em>Events</em> and drag a when <em>Green Flag clicked</em> block into the <em>Script Area</em> (on the right of the screen)</td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>Drag two <em>Say</em> blocks from <em>Looks</em> into the <em>Script Area</em> and attach them to the when <em>Green Flag clicked</em> block</td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>Enter some text for the sprite to say</td>
<td></td>
<td>![Image]</td>
</tr>
<tr>
<td>Click on your other <em>Sprite</em> and repeat the above</td>
<td></td>
<td>![Image]</td>
</tr>
</tbody>
</table>
Click on the Green Flag at the top of the Stage to run your program.

You will have noticed the Sprites talk over each other! To rectify this problem, we need to use a Wait instruction.

Click on your first Sprite and drag two Wait blocks from Control into your existing Script. Change the Wait time to two seconds for each block.

Click on your other Sprite and repeat the process (note the location of the Wait blocks within the script is different for the two Sprites).

Run your program again using the Green Flag.

Congratulations! You’ve created your first program in Scratch. You’ve written a program to achieve a specific goal, sequenced instructions and worked with outputs (the speech displayed on the screen). You’ve also probably corrected errors in your program, which is known as debugging.

**Extension activities**

For each of the activities below, feel free to experiment with a range of features to make your program as engaging as possible.

- Get the Sprite to move towards the character (Hint: have a look in Motion)
- Play a sound instead of using a Say block (Hint: try looking in Sound)
- Change the appearance of the Sprite after they’ve spoken (Hint: use a Next Costume block in Looks)
- Alter the background after one of the characters has spoken (Hint: the required block is in Looks)
- Add a third character (Sprite) to the conversation
Programming with Scratch

Optional activity – Broadcast messages

In Activity 1, *Wait* blocks were used to ensure the characters spoke in turn. Using *Wait* blocks in this way is quite inefficient and creates lots of extra work if we want to add additional events, such as movement, sound or further speech. We’re now going to alter each *Sprite’s script* to use *Broadcast* messages, which pass a message to a *Sprite* to ensure it only undertakes an action when it receives a specified command (rather than relying on a certain period of time).

When working with children, this can be effectively demonstrated by whispering to a few pupils that when they hear a specified code word, such as “Dave”, they have to get up and do five star jumps. “Dave” is then shouted out (i.e. *Broadcast*) to the class, but only the children given the code word know to act upon it.

<table>
<thead>
<tr>
<th>We need to modify our program from Activity 1 to remove the <em>Wait</em> blocks and use <em>Broadcast</em> messages instead</th>
<th>For each of your <em>Sprites</em>, delete your existing <em>Script</em> by clicking with the right mouse button and selecting <em>Delete</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Click on your first <em>Sprite</em> and drag a <em>Broadcast</em> block from <em>Events</em>, along with a <em>Say</em> block and a <em>when Green Flag clicked</em> block</td>
<td>Click on your second <em>Sprite</em> and drag a <em>When I receive</em> block from <em>Events</em>, followed by a <em>Say</em> block</td>
</tr>
<tr>
<td>Continue the conversation, by adding a <em>Broadcast</em> block to your second <em>Sprite’s script</em> (you’ll need to create a <em>new message</em> with a suitable name, such as <em>message2</em>)… followed by adding a separate <em>When I receive</em> block to your first <em>Sprite</em> and another <em>Say</em> block</td>
<td>Continue the conversation by repeating the process</td>
</tr>
</tbody>
</table>
Programming with Scratch

Activity 2 – Maths Quiz

Computing National Curriculum areas covered (all Key Stage 2):

- design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts
- use sequence, selection, and repetition in programs; work with variables and various forms of input and output
- use logical reasoning to explain how some simple algorithms work and to detect and correct errors in algorithms and programs

Task overview:

- Create a Maths game containing a range of multiplication questions
- Add a score to the game using a variable

Possible cross-curricular links:

- All subjects – assessment activities (formative and summative; peer assessment)
- Science – predicting outcomes of experiments
Activity 2 – Maths Quiz

Initially, we need to ask the user a question and allow them to input the answer. Although quizzes can be produced for any topic, it’s best to initially only ask questions with numerical answers, which minimises errors relating to spelling or typing.

<table>
<thead>
<tr>
<th>Task explanation</th>
<th>Required steps</th>
<th>Screenshots</th>
</tr>
</thead>
<tbody>
<tr>
<td>As with the first task, delete the cat Sprite so we can use something else</td>
<td>Click with the right mouse button on the cat in the Sprites Area and select Delete</td>
<td><img src="image1.png" alt="Screenshot" /></td>
</tr>
<tr>
<td>Next, we’re going to choose our character (you only need one!)</td>
<td>Click on the Choose new sprite from library button at the top of the Sprites Area</td>
<td><img src="image2.png" alt="Screenshot" /></td>
</tr>
<tr>
<td></td>
<td>Select a sprite from either the Animals, Fantasy or People sections and press OK</td>
<td><img src="image3.png" alt="Screenshot" /></td>
</tr>
<tr>
<td></td>
<td>Feel free to also change the background</td>
<td><img src="image4.png" alt="Screenshot" /></td>
</tr>
<tr>
<td>We need our character to introduce the game when the program starts</td>
<td>Drag a when Green Flag clicked block and a Say block into the Script Area (if you can’t see the Script Area, click on the Scripts tab at the top of the window)</td>
<td><img src="image5.png" alt="Screenshot" /></td>
</tr>
<tr>
<td></td>
<td>Change the text to introduce your game</td>
<td><img src="image6.png" alt="Screenshot" /></td>
</tr>
<tr>
<td>Our quiz needs a question to ask the user</td>
<td>Drag an Ask block from Sensing and attach it to your Script</td>
<td><img src="image7.png" alt="Screenshot" /></td>
</tr>
<tr>
<td></td>
<td>Change the text to ask a multiplication question</td>
<td><img src="image8.png" alt="Screenshot" /></td>
</tr>
<tr>
<td>The program has to display a different message depending on whether the answer is correct or incorrect. We do this using Selection</td>
<td>Drag an if, then, else block from Control and attach it to your Script</td>
<td><img src="image9.png" alt="Screenshot" /></td>
</tr>
</tbody>
</table>
Following this, we need to check the Input and use Selection to choose the route through our program, which is:

*If the answer is correct* (2x5=10), *then* display a "well done" message, *else* display an "unlucky" message.

Drag an equals block from Operators into the top of the *if, then, else* block.

Place Answer from Sensing in one side of the equals block, followed by the correct answer on the other side.

Drag a Say block into the first half of the *if, then, else* block. Change the text to "Well done".

Drag another Say block into the second half of the *if, then, else* block. Change the text to "Unlucky".

Run your program using the Green Flag (you enter your answer in the box that appears at the bottom of the Stage).

Add some additional questions using the same process.

Add further Ask and *if, then, else* blocks to your Script containing different questions and answers.

Run your program and check it behaves as expected.

---

Well done! You’ve made another program in Scratch. This time you’ve also used *selection* and worked with both inputs and outputs. We’re now going to add a score to our game using a variable.

A variable is similar to a box. It can contain anything the computer can store, such as numbers or text. Its contents can be changed, or varied (hence the name variable), and we can find out the contents of the box at any time.

We first need to create a variable to store the score.

Click on Data at the top of the Block Palette and select *Make a Variable*.

Give the variable a name, such as “Score” and click OK.
If the tick box next to the variable name is selected in the Block Palette, the variable is shown in the corner of the Stage

When the quiz starts, we have to ensure the score starts at zero, rather than continuing from the previous game

Drag the Set, to block from Data to immediately below the when Green Flag clicked block in your script (this ensures it’s the first instruction run)

Every time the user gets an answer correct, we need to increase the score by one

Drag a Change, by block from Data into the first half of your if, then, else blocks. Place it before the “Well done” message

Repeat the process for the other if, then, else blocks in your Script

Run your program and check the score starts at zero and it increases by one when a question is answered correctly

Good work! You’ve now used a variable to record the score. You have also output the variable to the screen for the user to see.

Extension activities

For each of the activities below, feel free to experiment with a range of features to make your program as engaging as possible.

- Adjust the score to decrease when the user gets a question wrong (Hint: change the score by -1)
- Make the character move or dance when an answer is correct (Hint: look in Motion)
- Get the character to include the user’s answer when they speak – e.g. “Well done. The answer was 10” (Hint: you’ll need to use a Join block from Operators)
- Change the background when the score reaches ten (Hint: add an if, then block to check the score and perform an action if Score = 10)
Programming with Scratch

Activity 3 – Maze Game

Computing National Curriculum areas covered (all Key Stage 2):

- design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts
- use sequence, selection, and repetition in programs; work with variables and various forms of input and output
- use logical reasoning to explain how some simple algorithms work and to detect and correct errors in algorithms and programs

Task overview:

- Guide a character around a maze without bumping into the sides

Possible cross-curricular links:

- Maths – angles and lines of symmetry (within the maze design)
- Art – sprite and maze design
- History / Literacy – famous mazes (e.g. Hampton Court); Myths and Legends (e.g. Theseus and the Minotaur)
### Activity 3 – Maze Game

Our first task is to make our character look like they're moving as they travel around the maze. We also need to control the character using the keyboard.

<table>
<thead>
<tr>
<th>Task explanation</th>
<th>Required steps</th>
<th>Screenshots</th>
</tr>
</thead>
</table>
| We need to choose a *sprite* with more than one look / appearance, or *Costume*. This allows us to make the *sprite* appear as if it's moving. | Delete the Cat *sprite*.  
Click on the *Choose new sprite from library* button at the top of the *Sprites Area*.  
Select a sprite from *Animals* containing more than one costume (possible *Sprites* include Crab, Butterfly1, Dinosaur1, Elephant, Monkey1) and press *OK*.  
Leave the background white. | ![New sprite](image1) ![New sprite](image2) ![New sprite](image3) ![New sprite](image4) |
| To give the impression of movement, we need to use *repetition* to constantly switch between *costumes*. We do this using a *forever loop*. | Drag a *when Green Flag clicked* block into the *Script Area*.  
Attach a *forever* block from *Control*.  
Place a *Wait* block from *Control* and a *next costume* block from *Looks* inside your *forever loop*.  
Run your program using the *Green Flag*. Alter the number of seconds in the *wait* block so it looks like your *sprite* is moving. | ![New sprite](image5) ![New sprite](image6) ![New sprite](image7) ![New sprite](image8) |
| During the maze game, our *sprite* is going to constantly move forward. We also use *repetition* through a *forever loop* to achieve this. | Drag a second *when Green Flag clicked* block and a *forever* block into the *Script Area*.  
Insert a *Move* block from *Motion* into the *forever loop*.  
Run your program. You can alter the number of steps in the *Move* block if your *sprite* is moving too fast. | ![New sprite](image9) ![New sprite](image10) ![New sprite](image11) ![New sprite](image12) |
Next, we need to move the character when certain keys are pressed. We use both repetition and selection to check for keyboard input.

| Drag a further *when Green Flag clicked* block and a *forever* block into the Script Area |
| Place the *forever* loop after the *when Green Flag clicked* block, followed by an *if, then* block inside the loop |
| Put a *key pressed* block from *Sensing* in the top of the *if, then* block and change the key to the *right arrow* |
| Put a *turn right* block within the *if, then* block |
| Repeat this process to make the *left arrow* turn the sprite left when pressed |
| Run your program using the *Green Flag* and experiment with the key presses. You can adjust the amount of turn in the *turn degrees* block if required |

You've produced yet another program in Scratch. Within the program, you've used repetition to make the character move across the screen. Repetition has been combined with selection to enable keyboard input, which alters the location of the sprite output to the screen.

The next stage in producing our game is to create the maze. We also need to program our sprite to perform a sequence of instructions if it touches the maze wall.

| Firstly, we need to draw our maze |
| Click on the *Stage* icon to the left of the *Sprites Area* |
| Select the *Backdrops* tab at the top of the *Blocks Palette* |
| Choose either the *Paintbrush* or *Straight line* tool from the icons on the left of the window |
| Select a colour and make the line width thicker by dragging the bar at the bottom of the screen |
Draw your maze by clicking and dragging with the left mouse button on the white backdrop (don’t worry if your sprite is too big at this stage)

Ensure your maze has a coloured border all the way around the edge

<table>
<thead>
<tr>
<th>Your <em>Sprite</em> is probably too large to move around the maze, so we need to make it smaller</th>
<th>Click on the <em>Shrink</em> icon above the <em>Block Palette</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Click on your <em>Sprite</em> a number of times until it can comfortably navigate around your maze</td>
<td></td>
</tr>
</tbody>
</table>

We now need to modify the game so the *Sprite* cannot go through the maze’s walls. We can again use *repetition* and *selection* to check whether the *Sprite* is touching the wall:

*If* the *Sprite* is touching the wall, *then* we want it to say “Ahhh” for a period of time, followed by ending the game

<table>
<thead>
<tr>
<th>We now need to modify the game so the <em>Sprite</em> cannot go through the maze’s walls. We can again use repetition and selection to check whether the <em>Sprite</em> is touching the wall:</th>
<th>Once again, drag a <em>when Green Flag clicked</em> block</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>If</em> the <em>Sprite</em> is touching the wall, <em>then</em> we want it to say “Ahhh” for a period of time, followed by ending the game</td>
<td>Place a <em>forever</em> loop after the <em>when Green Flag clicked</em> block, followed by an <em>if, then</em> block inside the loop</td>
</tr>
</tbody>
</table>

Drag a *touching color* block from *Sensing* into the top of the *if, then* block

Click on the colour within the *touching color*, followed by clicking on your maze outline on the *Stage* (this will alter the *touching color* to be the same as your maze)

Place a *Say* block within the *if, then* block and modify the block to say “Ahhhhh” for 0.2 seconds

Add a *Stop* block from *Control* into the *if, then* block (this will end the game)

Run your program using the *Green Flag*. You may wish to adjust some of the game’s sections, such as the *Sprite’s* speed, size or the layout of the maze
The *Sprite* currently remains in the same position when we restart the game. Each time the game starts, we ideally want the *sprite* to return to a specific starting position and point upwards.

1. Drag your *Sprite* to your preferred starting position on the *Stage*.
2. Add a `go to x, y` block from *Motion* to the start of your last script, which now contains the starting point’s co-ordinates (the co-ordinates are also displayed at the bottom right of the *Stage*).
3. You should also add a `Point in direction` block from *Motion*.
4. Run your program using the *Green Flag*. *Debug* your program as appropriate.

Brilliant! You’ve created a game involving a *Sprite* interacting with the background, that responds to a number of *inputs* (the keyboard and touching a colour). *Repetition* and *selection* have again been used, along with instructions in *sequence*. You’ll be beginning to see how programs can be built by *decomposing* them into smaller parts.

**Extension activities**

For each of the activities below, feel free to experiment with a range of features to make your program as engaging as possible.

- Include sound effects when the character moves / hits the wall (Hint: look in *Sound*)
- Change the *Sprite* to a ghost if it touches the wall (Hint: you’ll need to add a new *Costume*)
- Add a score that increases for every second the character avoids touching the side (Hint: combine a *variable* for the score with a *Wait* block)
- Instead of ending the game when the character touches the wall, get it to automatically change direction (Hint: remove the *Stop* block and look in *Motion*)
- Add objects, such as coins, for the character to collect (Hint: add some extra *Sprites* and *Hide* them when touched)
- Include a second level (Hint: you’ll need an end point, of a different colour, in your first maze, which, when touched, should change to your second maze background)
Programming with Scratch

Activity 4 – Moving sprite game

Computing National Curriculum areas covered (all Key Stage 2):

- design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts
- use sequence, selection, and repetition in programs; work with variables and various forms of input and output
- use logical reasoning to explain how some simple algorithms work and to detect and correct errors in algorithms and programs

Task overview:

A leading game manufacturer wants to see how well you can program.

They have asked you to design a game that involves moving a character around obstacles or away from enemies.

Your game can be for one or two players.
Activity 4 – Moving sprite game

If you wish, you can plan your moving sprite game and start programming immediately. Alternatively, you can use the example game specification below, or the more detailed instructions overleaf, which break down the necessary steps for each requirement.

Within school, pupils should be given a planning template and asked to solve the programming challenge by splitting it into smaller parts. This is known as decomposing. This would typically involve specifying the required steps, or script, to complete each section of the program.

Example game specification

Basic: A frog has to move across the road from his pond. The frog is controlled using the keyboard. Butterflies automatically move along the road. If the frog collides with a butterfly, he becomes injured and the game ends. The frog should always start from his pond at the bottom of the screen.

Intermediate: The frog starts with a score of 100, which should decrease by 5 for every second it takes to cross the road. The frog starts with three lives. Every time the frog collides with a butterfly, he loses a life. Once the frog has lost all his lives, the game ends. A “game over” message or screen should be displayed, along with an appropriate sound.

Advanced: Each butterfly’s speed should increase as the game progresses. Once the frog crosses the road and reaches a certain point on the other side, he should progress to the next level. He should also get an extra 100 points.
## Detailed instructions

For each sentence of the *Basic example game specification*, the required steps are outlined below, although a similar result can often be achieved using other blocks or scripts.

<table>
<thead>
<tr>
<th>Task explanation</th>
<th>Required steps</th>
<th>Screenshots</th>
</tr>
</thead>
</table>
| **Text:** A frog has to move across the road from his pond  
*Required steps:* We need a frog sprite, in addition to a suitable background | Delete the Cat sprite  
Click on the *Choose new sprite from library* button at the top of the *Sprites Area*  
Select the *Frog* sprite from *Animals* and click OK  
Click on the *Stage* (left-hand side of the *Sprites Area*)  
Click on the *Upload backdrop from file* button  
Select the file containing the game’s background and click Open | ![New sprite](image1)  
![New backdrop](image2) |
| **Text:** The frog is controlled using the keyboard  
*Required steps:* Our sprite is going to constantly move forward using a *forever* loop. We also need to move the character when certain keys are pressed | Drag a *when Green Flag clicked* block and a *forever* block into the *Script Area*  
Insert a *Move* block from *Motion* into the *forever* loop  
Drag a further *when Green Flag clicked* block into the *Script Area*  
Place a *forever* loop after the *when Green Flag clicked* block, followed by an *if, then* block inside the loop  
Put a *key pressed* block from *Sensing* in the top of the *if, then* block and change the key to the *right arrow*  
Put a *turn right* block within the *if, then* block | ![when clicked](image3)  
![when clicked](image4)  
![if then](image5)  
![if then](image6) |
Repeat this process to make the left arrow turn the sprite left when pressed.

Run your program using the Green Flag and experiment with the key presses. You can adjust the amount of turn in the turn degrees block. You also can alter the number of steps in the Move block if your sprite is moving too fast.

Text: Butterflies automatically move along the road.

**Required steps:** We need to add some additional butterfly sprites to move horizontally across the screen. When they reach the edge of the screen, they should turn around and continue moving.

Click on the Choose new sprite from library button at the top of the Sprites Area.

Select the Butterfly1 sprite from Animals and click OK.

Drag a when Green Flag clicked block and a forever block into the Script Area.

Insert a Move block from Motion into the forever loop, in addition to a wait block from Control.

Add an if on edge, bounce block from Motion to your forever loop.

Finally, add a set rotation style block from Motion before your forever loop (this stops our butterfly turning upside down when it touches the stage edge).

Run your program using the Green Flag and ensure your butterfly moves automatically. Experiment with altering the value of the move and wait blocks.

Additional butterfly sprites can be added by repeating the process above.
**Text:** If the frog collides with a butterfly, he becomes injured and the game ends.

**Required steps:** We need to detect if the frog is touching a butterfly using repetition and selection:

*If the frog sprite is touching the butterfly sprite, then we want the frog to say “Owwww” for a period of time, followed by ending the game.*

---

<table>
<thead>
<tr>
<th>Required steps</th>
<th>Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drag a <em>when Green Flag clicked</em> block into the Script Area.</td>
<td><img src="image" alt="Script Diagram" /></td>
</tr>
<tr>
<td>Place a <em>forever</em> loop after the <em>when Green Flag clicked</em> block, followed by an <em>if, then</em> block inside the loop.</td>
<td><img src="image" alt="Loop Diagram" /></td>
</tr>
<tr>
<td>Put a <em>touching</em> block from Sensing in the top of the <em>if, then</em> block and change the <em>sprite</em> to <em>Frog</em>.</td>
<td><img src="image" alt="Frog Block Diagram" /></td>
</tr>
<tr>
<td>Put a <em>Say</em> block from Looks into your <em>if, then</em> block and alter the block to say “Owwww” for 0.5 seconds.</td>
<td><img src="image" alt="Say Block Diagram" /></td>
</tr>
<tr>
<td>Add a <em>Stop</em> block from Control into the <em>if, then</em> block (this will end the game).</td>
<td><img src="image" alt="Stop Block Diagram" /></td>
</tr>
<tr>
<td>Run your program using the <em>Green Flag</em> and ensure your script works as expected.</td>
<td><img src="image" alt="Green Flag" /></td>
</tr>
<tr>
<td>Repeat this process for your other butterfly sprites.</td>
<td><img src="image" alt="Repeat Process" /></td>
</tr>
</tbody>
</table>

---

**Text:** The frog should always start from his pond at the bottom of the screen.

**Required steps:** The frog *Sprite* currently remains in the same position when we restart the game. Each time the game starts, we ideally want the *sprite* to return to a specific starting position and point upwards.

---

<table>
<thead>
<tr>
<th>Required steps</th>
<th>Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drag your frog <em>Sprite</em> to your preferred starting position on the <em>Stage</em> (in the pond).</td>
<td><img src="image" alt="Boot Sprite" /></td>
</tr>
<tr>
<td>Place a <em>when Green Flag clicked</em> block into the Script Area.</td>
<td><img src="image" alt="Boo Block Diagram" /></td>
</tr>
<tr>
<td>Add a <em>go to x, y</em> block from <em>Motion</em> below the <em>when Green Flag clicked</em> block. You should also add a <em>Point in direction</em> block from <em>Motion</em>.</td>
<td><img src="image" alt="Motion Block Diagram" /></td>
</tr>
<tr>
<td>You may wish to also set the starting position for your butterfly sprites.</td>
<td><img src="image" alt="Set Starting Position" /></td>
</tr>
</tbody>
</table>
Brilliant! You now have a moving sprite game, which has been produced by decomposing the program specification into smaller tasks. A child undertaking a task such as this independently has met the coding requirements of the Key Stage Two Computing National Curriculum.

For the Intermediate and Advanced example game specification, step-by-step instructions are not provided. However, guidance for how to complete each part of the specification is outlined below. As with the Basic guidance, similar results can often be achieved in a number of ways.

Intermediate example game specification

<table>
<thead>
<tr>
<th>Text</th>
<th>Required steps</th>
</tr>
</thead>
</table>
| The frog starts with a score of 100, which should decrease by 5 for every second it takes to cross the road | • Create a variable for the score  
• Set the variable to 100 when the game starts  
• Use a forever loop to change the score by -5 and wait one second |
| The frog starts with three lives. Every time the frog collides with a butterfly, he loses a life | • Create a variable for the lives  
• Set the variable to 3 when the game starts  
• When the frog touches a butterfly, change the score by -1 |
| Once the frog has lost all his lives, the game ends                  | • If the lives variable is zero, stop the game                                |
| A "game over" message or screen should be displayed                  | When the lives variable reaches zero, either:  
• Say a specific message  
• Change the backdrop to a special “game over” screen |
| Along with an appropriate sound                                       | • Play a sound when the lives variable reaches zero |

Advanced example game specification

<table>
<thead>
<tr>
<th>Text</th>
<th>Required steps</th>
</tr>
</thead>
</table>
| Each butterfly’s speed should increase as the game progresses       | • Create a variable for the speed for each butterfly sprite  
• Set the variable to 0 when the game starts  
• Use a forever loop to change the move speed by a certain amount after a specific period of time, such as every few seconds |
| Once the frog crosses the road and reaches a certain point on the other side, he should progress to the next level | • Create a sprite or coloured area at the top of the screen  
• When the frog touches the area or sprite, the backdrop should change  
• The starting position of both the frog and butterflies may also need to be altered |
| He should also get an extra 100 points                              | • When changing the backdrop, change the score variable by 100 |

Written by Neil Rickus. Licensed under CC BY-NC-SA 4.0
Extension activities

For each of the activities below, feel free to experiment with a range of features to make your program as engaging as possible.

- Include sound effects when the character moves / hits the enemies (Hint: look in Sound)
- Change the Sprite to something different if it touches a butterfly (Hint: you’ll need to add a new Costume)
- Get the butterflies to change direction (Hint: use a turn block)
- Make the frog bounce across the stage if he touches the side (Hint: look in Motion)
- Add objects, such as food, for the character to collect (Hint: add some extra Sprites and Hide them when touched)
- Have different enemies, such as dogs or dinosaurs, on each level (Hint: only show certain sprites when a certain backdrop is selected)
- Rather than your frog automatically moving forward, program the up and down arrows to move the sprite forwards and backwards respectively
- Include additional levels
Computing National Curriculum areas covered (all Key Stage 3):

- use two or more programming languages, at least one of which is textual, to solve a variety of computational problems
- design and develop modular programs that use procedures or functions
- understand simple Boolean logic [for example, AND, OR and NOT] and some of its uses in circuits and programming

Task overview:

A leading game manufacturer was impressed with your moving sprite game. They now want you to produce a platform game.

They have asked you to make a one-player game, which involves moving a sprite along various platforms. The sprite is likely to have to jump over gaps between platforms, in addition to avoiding obstacles and enemies.
Activity 5 – Platform game

If you wish, you can plan your platform game and start programming immediately. Alternatively, you can use the example game specification below, or the more detailed instructions overleaf, which break down the necessary steps for each requirement.

Within school, pupils should be given a planning template and asked to solve the programming challenge by splitting it into smaller parts. This is known as decomposing. This would typically involve specifying the required steps, or script, to complete each section of the program. They should also identify sections of code that can be reused and included within a procedure.

Example game specification

Basic: A sprite has to move along the ground or on platforms. The sprite is controlled using the keyboard. The sprite should fall to the ground if not on a platform. The sprite should be able to jump onto platforms and be able to jump a range of different heights. The sprite should start on the left hand side of the screen and move right along the ground / platforms. The code should be implemented using procedures where possible.

Intermediate: The sprite should move realistically, including when jumping or falling. The sprite starts with three lives. The sprite should have to avoid enemies and lose a life if he collides with an enemy. Once the sprite has lost all his lives, the game ends. A "game over" message or screen should be displayed, along with an appropriate sound.

Advanced: Once the sprite reaches a certain point on the right hand side of the screen, a "congratulations" message must appear and he should progress to the next level. The sprite has a set amount of time to complete the level. Levels should scroll over more than one screen where possible.
## Detailed instructions

For each sentence of the *Basic example game specification*, the required steps / blocks are outlined below, although a similar result can often be achieved using other blocks or scripts.

<table>
<thead>
<tr>
<th>Task explanation</th>
<th>Required steps</th>
<th>Screenshots</th>
</tr>
</thead>
</table>
| **Text:** A sprite has to move along the ground or on platforms  
*Required steps:* We need a suitable *sprite* to control and also a *sprite* to act as our ground / platforms. The background should be kept to a single colour.  
| Delete the Cat *sprite*  
Click on the *Choose new sprite from library* button at the top of the *Sprites Area*  
Select the *sprite* click OK  
Click on the *Paint new sprite from library* button at the top of the *Sprites Area*  
Draw a simple *sprite* containing a flat ground and a number of coloured obstacles. The *sprite* should fill the whole screen  
*Rename the *sprite* appropriately, such as "ground"* | ![New sprite:](image) ![New sprite:](image) |
| **Text:** The sprite is controlled using the keyboard  
*Required steps:* Our program is going to constantly check for when certain keys are pressed. Unlike previous programs, we're going to manually alter the X co-ordinates, rather than using move.  
To move the *sprite* left and right, we can use the following code. Note the use of both *not* and *and* operators in the final *if* statement.  
| ![Diagram](image) |
| **Text:** The sprite should fall to the ground if not on a platform  
*Required steps:* We need to *create* and *set a variable* for the "gravity". We also need an extra *if* statement in the *forever* loop to check if the *sprite* is on the ground and reduce the Y co-ordinate appropriately if *not* touching ground.  
| ![Diagram](image) |
Text: The sprite should be able to jump onto platforms and be able to jump a range of different heights. The code should be implemented using procedures where possible.

Required steps: We can create a block, or procedure (in More Blocks) named "jump" to reuse the same code for different height jumps.

We can assign different key presses to pass different values to the procedure.

Text: The sprite should start on the left hand side of the screen and move right along the ground / platforms.

Required steps: We can define the sprite's starting position at the beginning of our program using goto x y.

The final code is shown right.

Well done! You now have a platform game, which includes the use of procedures and Boolean operators. A child undertaking a task such as this independently has met some of the coding requirements of the Key Stage Three Computing National Curriculum.

For the Intermediate and Advanced example game specification, blocks of code are not provided. However, guidance for how to complete each part of the specification is outlined below. As with the Basic guidance, similar results can often be achieved in a number of ways.
Intermediate example game specification

<table>
<thead>
<tr>
<th>Text</th>
<th>Required steps</th>
</tr>
</thead>
</table>
| The sprite should move realistically, including when jumping or falling | • Give the impression of movement using next costume  
• Slowly increase and decrease the value of the Y co-ordinates during the jump |
| The sprite starts with three lives. The sprite should have to avoid enemies and lose a life if he collides with an enemy. Once the sprite has lost all his lives, the game ends | • Create a variable for the lives  
• Set the variable to 3 when the game starts  
• When the sprite touches an enemy, change the score by -1 |
| Once the sprite has lost all his lives, the game ends               | • If the lives variable is zero, stop the game                                                                                               |
| A "game over" message or screen should be displayed                 | When the lives variable reaches zero, either:                                                                                                 |
|                                                                    | • Say a specific message  
• Change the backdrop to a special “game over” screen                                                                                   |
| Along with an appropriate sound                                     | • Play a sound when the lives variable reaches zero                                                                                            |

Advanced example game specification

<table>
<thead>
<tr>
<th>Text</th>
<th>Required steps</th>
</tr>
</thead>
</table>
| Once the sprite reaches a certain point on the right hand side of the screen, a "congratulations" message must appear and he should progress to the next level | • Create a sprite or coloured area at the right hand side of the screen  
• When the main sprite touches the area, the message sprite should be displayed, followed by changing the ground / platforms sprite |
| The sprite has a set amount of time to complete the level            | • Create a variable for the time and reduce it by one every second  
• When the variable reaches zero, display the "game over" message / screen                                                                 |
| Levels should scroll over more than one screen where possible       | • Create the adjoining part of the level sprite and hide it when the game starts  
• As the sprite moves right, move both the background sprite left at the same rate as the sprite |
Extension activities

For each of the activities below, feel free to experiment with a range of features to make your program as engaging as possible.

- Include sound effects when the character moves / hits the enemies (Hint: look in Sound)
- Change the Sprite to something different if it touches an enemy (Hint: you'll need to add a new Costume)
- Allow the player to select their Sprite at the start of the game (Hint: you could use a range of Costumes or independent sprites)
- Get the enemies to move at random (Hint: look in Operators)
- Add objects, such as food, for the character to collect (Hint: add some extra Sprites and Hide them when touched)
- Have an end of level boss to get past (Hint: only show certain sprites when certain co-ordinates are reached)
- Give the sprite ability to fire custard pies or a similar, non-threatening, weapon at enemies (Hint: add an additional sprite, which moves from the main sprite when a certain key is pressed)
- Include additional levels