Community Experience Distilled

Leverage the power of programming to use the Raspberry Pi to create awesome games

Raspberry Pi Projects for Kids
Second Edition

Daniel Bates

The Raspberry Pi is a single-board mini computer designed to get more people (particularly children) interested in computer programming. It aims to make programming tools and educational programs as accessible as possible, making it very easy to get started with.

This book will guide you through six fun projects that show how programming can be used to be creative. Each project has clear step-by-step instructions and explanations helping children grasp the concepts easily.

You will start by setting up the Raspberry Pi and get to grips with the Scratch programming language to create simple animations. Gain and put to use your understanding of Python to write simple yet useful programs. Create and play a physical game by connecting it to the Raspberry Pi en route to become aware of a number of other possible uses for a very similar circuit/program. Finally, with an understanding of Sonic Pi, you will create your own music.

Who this book is written for
This book is for kids who wish to develop games and applications using the Raspberry Pi. No prior experience in programming is necessary; you need only a Raspberry Pi and the required peripherals.

What you will learn from this book
- Gear up to start programming by setting up the Raspberry Pi and taking a tour of available applications
- Understand the fundamentals of programming and electronics using the Raspberry Pi
- Use the Linux operating system and programming languages such as Scratch and Python to build interesting projects
- Gain a basic understanding of how the Python programming language works by writing simple programs
- Build a fully functioning game and explore how to modify it to create new levels
- Create animations and music to make your games and applications more exciting
- Make computer code interact with the physical world
- Add markers to your personal mapping program
- Get an understanding of Sonic Pi, and discover how to use it to create your own music

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 2 'Animating with Scratch'
- A synopsis of the book’s content
- More information on Raspberry Pi Projects for Kids Second Edition

About the Author

Daniel Bates is a computer science researcher at the University of Cambridge. His day job involves inventing designs for future mobile phone processors and when he gets home, he likes playing games or working on one of his coding projects (or both!). Daniel has been a volunteer for the Raspberry Pi Foundation since 2011 and is enthusiastic about introducing new people to computing. He has previously written Instant Minecraft: Pi Edition Coding How-to and Raspberry Pi Projects for Kids (First Edition), both published by Packt Publishing.
Raspberry Pi Projects for Kids
Second Edition

The Raspberry Pi is a credit card-sized computer designed to make computing accessible to all. With the trend towards making computers easier and easier to use, the art of programming has been in decline. Programming is a powerful tool that lets us tell the computer exactly what we want to do. In much the same way as we use a hammer or screwdriver to help us with a physical task, we can use programming to help us with a mental task. The Raspberry Pi exposes programming software to make it as easy as possible to get started.

After introducing the Raspberry Pi computer and showing you how to set it up, this book will guide you through six separate mini-projects. Each project is fun, visual, and has plenty of scope for personalization. By the end of this book, you will understand and be able to use three different programming languages, and will be able to use them to build creative programs of your own.

What This Book Covers

Chapter 1, Getting Started with Raspberry Pi, shows you what Raspberry Pi is and how you can get one set up and ready to use.

Chapter 2, Animating with Scratch, introduces the Scratch programming language and uses it to create simple (and not-so-simple) animations.

Chapter 3, Making Your Own Angry Birds Game, teaches you how to make your very own computer game using the Scratch programming language.

Chapter 4, Creating Random Insults, explores how random funny phrases can be generated using the Python programming language.

Chapter 5, Testing Your Speed, helps you to connect electronic components to your Raspberry Pi to create a physical game controlled by your computer code written in Python.

Chapter 6, Making an Interactive Map of Your City, teaches you more about Python and shows you how to access Google Maps to create a personal map of your area.

Chapter 7, Building Beats with Sonic Pi, introduces the Sonic Pi application and shows you how the programming concepts learned so far can be applied to the creation of music.
In this chapter, we’re going to use a programming language called **Scratch** to create a simple animation. Along the way, we’ll visit many of the main concepts of programming languages, so if you understand everything that you learn in this chapter, you will be well equipped to start writing programs of your own.

**Scratch**

In this chapter, we will use Scratch to create our animation. Scratch is a programming language that has been specially designed so that you can make animations and games with ease. Version 1.4 of Scratch is pre-installed with the Raspbian OS but is also available on other computers. You can download it from [http://scratch.mit.edu/](http://scratch.mit.edu/) if you ever want to run your programs away from your Raspberry Pi. Start up Scratch by opening **Menu** at the top of the screen, and navigating to **Programming**, and then **Scratch**.

**Downloading the example code**

You can download the example code files for all Packt books you have purchased from your account at [http://www.packtpub.com](http://www.packtpub.com). If you purchased this book elsewhere, you can visit [http://www.packtpub.com/support](http://www.packtpub.com/support) and register to have the files e-mailed directly to you.
The following screenshot shows the layout of Scratch:

The following are its main sections. I’ll mention the names of these sections throughout the next two chapters, so you might want to refer back to this page.

The following are the elements of Scratch, as shown in the preceding screenshot:

- **Menu (1)**: This is where the options to save and load your projects are. If you ever want inspiration for projects, take a look at the provided examples by navigating to **File | Open | Examples**. Remember to save and back up your progress regularly!

- **Sprite controls (2)**: Every picture in the game is called a sprite. These buttons allow you to copy, remove, grow, and shrink sprites. To use them, click on the button you want, and then click on the sprite you want to affect.

- **Screen layout (3)**: Choose between a small Stage (see 4), a large Stage, and a fullscreen game. The small Stage is better for smaller screens as it allows more space for code.
• **Stage (4):** This is where you will see the effects of all your programming.

• **Sprite list (5):** All the sprites in your project are shown here, and you can easily add new pictures or change existing ones in it.

• **Script area (6):** Each sprite has a number of scripts attached to it, and they are shown in this area. Each script is a short piece of code that controls how the sprite behaves.

• **Blocks (7):** Each block is a programming command that can be connected to other blocks (similar to a jigsaw) to create scripts. Drag a block into the script area to use it, and then drop it next to another block in the script area to join the two.

• **Block types (8):** The blocks are separated into eight different categories, each having different roles in your programs.

### Hello world!

Let's create a very simple program to show how easy it is to produce a visible result.

1. From the **Control** section, drag a **when green flag clicked** block into the script area. Then drag a **forever** block so that it connects to the bottom of the **when green flag clicked** block. Finally, from the **Motion** section, drag a **turn 15 degrees** block into the middle of the **forever** block.

2. Once the blocks are connected, you can move them all around at the same time by clicking and dragging the topmost block. If you drag a block in the middle of a collection of blocks, you will get that block and all the blocks below it.

3. Click on the green flag present at the top-right corner of the screen to run the program.

The following screenshot graphically illustrates the preceding steps:
You should see the cat rotating. Your script should also be highlighted to show that it is active. You can change the rotation amount to any number you like to see the cat spin faster or slower—click on 15, seen in the preceding code block, and type in a new number. You can even choose a negative number, and the cat will spin in the opposite direction. Click on the red stop sign in the top-right corner to stop your program.

Now, I’ll describe how the Raspberry Pi understands your program and knows what to do. It understands that a script should start when the green flag is clicked on because this is the top block. As soon as this has happened, it moves on to the next block, forever. Everything inside the forever block will execute repeatedly until you tell it to stop. In this case, we have told the Raspberry Pi that we want to continuously rotate the cat, and this is what we see. You can see that no blocks can be attached to the bottom of the forever block. If an action keeps on going forever, no later commands will ever run.

## Code tour

There are several types of code blocks available if you want to continue experimenting before we start working on the animation. A full description can be found online at [http://info.scratch.mit.edu/Support/Reference_Guide_1.4](http://info.scratch.mit.edu/Support/Reference_Guide_1.4). A quick tour of the code blocks is as follows:

- **Motion**: This allows us to control where a sprite is on the screen and which direction it is facing. Its options include rotating, moving to any position, and moving in the direction that the sprite is facing.

- **Control**: This allows us to choose when other blocks of code should run. In the preceding example, we saw how to decide when a script should start and how to repeat a block; however, it is also possible to execute a block only if a given condition is true.

- **Looks**: These enable us to decide what a sprite will look like. Each sprite can have multiple images or costumes associated with it, and these blocks can be used to switch between the two. It is also possible for the sprites to talk or change in size or color.

- **Sensing**: This enables us to allow a sprite to detect its surroundings.

- **Sound**: This enables us to play sound. You can add new sounds from the Sounds tab in the script area.

- **Operators**: These are simple mathematical functions, such as add and subtract. Note that some of the blocks are of different shapes; they show which blocks fit together and will be important later.
• **Pen**: This enables us to allow a sprite to draw a line to show where it has been.

• **Variables**: These allow us to give names to pieces of information so that they can be accessed from multiple places. We will go into more detail on this later in the chapter.

If you ever add a block which you no longer want, you can either drag it back to the **Blocks** area, or right-click on it and select **delete**.

### Some more interesting movements

A rotating cat is fun, but isn't particularly interesting, is it? Let's see if we can do something a little better.

1. Drag a **move 10 steps** block from the **Motion** section and place it anywhere inside your existing **forever** block. The cat should now move in a circle when you click on the green flag, rather than just spinning on the spot.

2. Adjust the numbers in the **move** and **rotate** blocks until you are happy with the cat's movement. A larger number in the **move** block will make the circle larger and the motion faster. A smaller number in the **rotate** block will also make the circle larger, but this time it will take longer to complete a rotation. If the cat moves to a position you don't like, you can always drag it around on the Stage.

3. If you like the path the cat is taking, but think it is moving too quickly, you might like to try adding in a **wait 1 secs** block (from the **Control** section) inside your **forever** block, and reducing its number to something very small, such as 0.01. I find the following to be a good combination:

   ![Diagram: when clicked forever turn 10 degrees move 5 steps]

4. Now, cats don't usually move around in circles like this, so let's choose something a little more appropriate. At the top of the script area, click on the **Costumes** tab, then click on **Import**. You should see a whole selection of different images to use, including animals, people, and things.
5. Choose an image of something that you think would be more likely to move in the circular motion you created, such as a fish, bird, or an airplane.
6. Click on **OK** when you're happy with your selection, and you will be returned to the main Scratch screen.
7. You should now be able to see the image you chose, and also a couple of slightly different versions of the cat image.
8. We won't need the cat images anymore; remove them by clicking on the small X symbol next to each one.

**Setting the scene**

We now have an image flying around in empty space. Let's add a background to make it look a bit better.

1. In the sprite list, on the left-hand side, you will find a special sprite called **Stage**. This serves as the background of the animation. Click on it, and the script area will update to show you information on the Stage. You should see that there is one available background simply called **background1**, and it is a white rectangle (if you don't see this, click on the **Backgrounds** tab at the top of the script area).
2. Just like we did before, click on **Import**, and choose a background that you think suits your sprite.
3. When you've finished, click on X next to the white background and you should be able to see your sprite and the new background on the stage. I chose an airplane as my sprite, so I've chosen a background which has some sky for it to fly around in:
4. As you can see, my plane looks like it has already crashed! I need to put it in a better position on the stage. If you also need to move your sprite to a different location, click on the sprite on the stage and drag it to a better position.

5. I also want to make my plane a little smaller. This can be done using the final button in the sprite controls section (as shown here). The button next to it can be used to make sprites larger.

6. Click on one of the buttons, and then click on your sprite repeatedly until it is the size you want it to be. You might want to adjust its position again when you are happy with its new size. Here's an illustration of my stage now:

As you can see, I've moved the plane into the sky, and made it smaller so that it appears to be further away.
Another way to animate

Remember how we earlier removed two slightly different cat images? They were there to allow a different way of animating, that is, switching between different images, or costumes. The following are the steps needed to switch between images and costumes:

1. At the top of the sprite list, click on the middle button. It says Choose new sprite from file when you hover your mouse cursor over it.

2. Find an image you like and that has multiple versions of it available, such as crab1-a and crab1-b. Choose the former, and click on OK.

3. Now, go into the Costumes tab and Import the second version of the same sprite, but this time, do not remove the costume that you already had. If your chosen sprite has more than two versions of it, repeat this process until you have all of them. You should see a screenshot similar to this one when you're done:

4. Now that we have all the costumes we want, let's write a script that cycles through them to create an animation. Click on the Scripts tab at the top of the script area and build the following script in it. You'll notice that your script for the first sprite isn't here. Each sprite has its own collection of scripts that decides how it should behave. You can always return to see a particular sprite's scripts by clicking on that sprite in the sprite list:
5. The **next costume** code block can be found in the **Looks** section of the blocks area. Click on the green flag to see what your animation looks like – you may want to adjust the time taken between switching costumes, so as to make your animation look better.

6. Finally, in the same way as we did for our first sprite, let's choose a sensible size and position for this second sprite. Drag it to a better position on the stage, and use the grow or shrink buttons to change its size.

Both of the animation methods we have met so far can be combined. It is possible to change a sprite's costume, and move it around the screen. This is as easy as adding a second script to a sprite. If we give both the scripts we have written so far to the same sprite, it will cycle through its costumes while it moves in a circle. Try it out if you like!

**Interactive animation**

We now have two different animated sprites, each doing their own thing. One of the special features of Scratch is that it makes interactive animation simple – we can program a sprite to react to you!

Create a new sprite in the same way as you did before using the **Choose new sprite from file** button. For this animation, we're going to have a simple conversation with the user, so choose an image of something that can talk. Find a good position and size for the new sprite, then build the following script in its script area:
There are a few new code blocks here. Let's go through them one by one:

- **ask and wait**: This gives the sprite a speech bubble, asking a question in a box. This block can be found in the **Sensing** section. The script will stop until the user has typed an answer into a text box on screen.

- **answer**: The answer to the question is stored in this block, and is also in the **Sensing** section.

- **join**: This takes two pieces of text and merges them into a longer sentence.

- **say**: This behaves a bit like **ask** did — the sprite uses a speech bubble to convey the text that is within its box. The **say** option can be found in the **Looks** section; there is also **think** which behaves similarly, but it has a thought bubble instead of a speech bubble. The number shown at the end of the **say** block determines how many seconds the speech bubble should be shown for.

- **stop all**: This ends all the scripts of all the sprites. This is not a necessary step, but is useful to end the scripts which run **forever**, and show that the animation has finished. This block can be found in the **Control** section.

Now let's continue the conversation. Add the following blocks between **say** and **stop all**, which you already have:

```
ask What's your favourite number? and wait
say join join I like the number answer too for 2 secs
```

This is fairly similar to the previous section of code. The only clever part is that we want to join three pieces of text, but a single **join** block only allows two. To get around this, we join the first two pieces of text, **I like the number** and **answer**, in one block, and then join the whole first join block with **too!**

**Variables**

Now, what if we want to use the player's name again to say **goodbye** at the end? We previously accessed the name using the **answer** block after we asked for the player's name. However, we have since asked another question, so **answer** now holds the player's favorite number.
The solution to this is to use variables. A variable is used to store a piece of information, and this information can be changed at any time. The answer variable is a special variable which is automatically set whenever a question is asked.

1. Click on Variables and then on Make a variable. The following window will pop-up:

2. Call your variable name and click on OK. You will see that a few new block options have appeared at the left-hand side of the screen. You will also see that name appears with a tick next to it—this means that the current value of the name is being displayed on the stage. We don't need this to be done, so click on the tick to remove it.

We can now complete our short conversation by storing the player's name in the name variable, so that we can access it again when it's time to say goodbye. Here is the script used for the whole conversation:

As you can see, instead of using the answer block directly, we store its value in the name variable, and then use name throughout the script.
Movement

Now let's add a bit of movement to this animation. We're going to make the sprite do backflips, and the number of backflips will be decided by the player's favorite number. Here's the code needed to perform a single backflip:

![Code block]

You may find it useful to build this script, and then click on it to run it once, before adding it to the existing script. This is a good way to test that it works properly. Here's a quick explanation of what's happening in this code:

- **glide**: This is a smooth form of motion. We give it a duration and a position to move to. A shorter duration means quicker movement. The position is described using x and y coordinates. The value of x axis increases as you move to the right on the stage, and y increases as you move up the stage. You can see the current coordinates of the mouse cursor at the bottom-right of the stage. In this case, we want to move upwards at the start of the flip and downwards at the end, so we add 50 to the sprite's current y position at the start, and subtract the same 50 when we're finished.

- **repeat**: This makes all of the code blocks inside it run a certain number of times. It is a lot like **forever**, which we have already seen, but it will eventually stop. In this case, we make 24 small rotations to make the flip appear smooth. You might have noticed that 24 rotations of 15 degrees makes up the full 360 degrees of a circle, so the sprite will finish the right way up at the end.
Now, that was one flip. We want to do a certain number of flips, so we will want to put the whole script, shown in the preceding image, inside another `repeat` block, and repeat it as many times as the player tells it to. In order to have access to the player's favorite number, we'll need to insert our new code into our previous script at the right point: after the question has been asked, but before saying goodbye. Here's what the code should look like now:

![Code snippet]

### Keeping count

Now, let's have the sprite count the flips as it's moving so we know it does the right amount. For this, we're going to need a new variable to keep track of how many flips have taken place so far. Create a new variable called `flips` in the same way as you created `name` earlier. We will want to set the number of flips to zero before the sprite starts flipping; we will want to increase the number of flips by one after each flip, and also we'll also want the sprite to tell us how many flips it has done so far.
The following image shows you what the code for flipping should look like. All the other code should still be present in your script, but I'm not showing it in this image, so as to help you focus on the section that has changed:

```
set flips to 0
repeat answer
  glide 0.1 secs to x: x position y: y position + 50
  repeat 24
    turn down 15 degrees
  end
  glide 0.1 secs to x: x position y: y position - 50
  change flips by 1
  think flips for 1 secs
```

As you can see, there are three new code blocks, which match up to the following three things we wanted to do:

- We set flips to 0 at the start
- We change flips by 1 after each flip (this adds 1 to the current value of flips)
- The sprite displays how many flips it has done so far in one second

**If-then-else**

Repetition is one way to control which code blocks run; the If-then-else method is another. In the Control section, you will see a few different types of repeat and forever blocks, and also a couple of blocks that say if. These blocks allow us to optionally run any code blocks inside them, depending on the result of a test. The else version of the block gives us options: if the test passes, only the blocks in the first gap run, and if the test fails, only the blocks in the second gap run. We're going to use this block in our script.

```
if flips > 5
  say Wow, I'm dizzy now! for 2 secs
else
  say Easy! for 2 secs
```
Here, you are shown some code, which makes use of the **if-then-else** code block. Our test is performed to figure out whether the number of flips is more than five. If it is, the sprite tells us how dizzy it is. If it isn’t, then the sprite says that it was easy to do all the flips. We can put as many code blocks as we like within each of the gaps in the **if-then-else** block but, for this example, we only have one block in each.

This code needs to go in after all the flips have finished, but before saying goodbye. Here’s an illustration of the final, completed script:
Summary

In this chapter, we explored a few different ways of creating animations in Scratch. Along the way, we used a wide selection of the available code blocks. Many of these blocks are very similar to those used in other programming languages—you will recognize them in later chapters.

Importantly, you should recognize that this is your program, and you are free to change it as you like. You can change the way any of the sprites look, you can change any of the text or numbers, and you can even change which blocks are used to change the behavior of the program.

In the next chapter, we will continue the theme of making things interactive by building an entire game in Scratch – a version of Angry Birds which you will be able to modify to suit you.
Where to buy this book


Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.