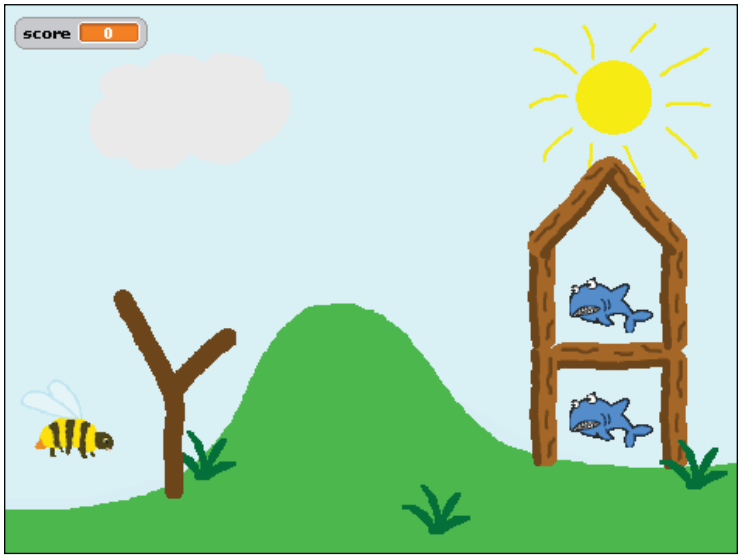
2
Making Your Own Angry Birds Game

In this chapter, we are going to make our own version of the popular Angry Birds game. What's more, when we're finished, we will be able to add all sorts of new rules and enemies to keep the game fresh. The following screenshot shows a completed version of our game:

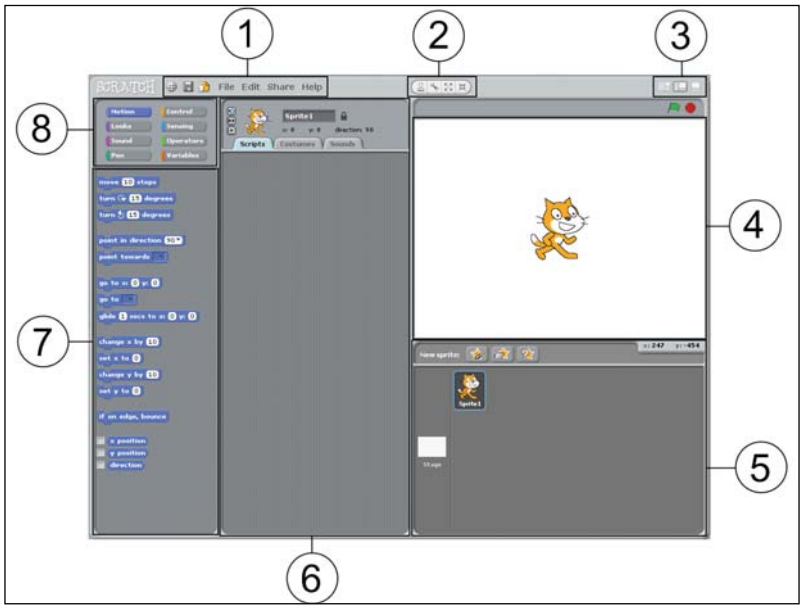


If you haven't played Angry Birds before, here's a quick description of how the game works. The player launches a bird through the air using a slingshot and attempts to hit all of the pigs at the other end of the level. In order to make things more challenging, the pigs are often hidden behind hills or inside flimsy buildings that the player must knock down.

By creating our own version of the game, we have the freedom to change whatever we like. We can change the level design, decrease gravity, fire the bird faster (or bee, in our case), change all of the characters, and add new power-ups and prizes. The sky is the limit!

Scratch

In this chapter, we will use Scratch to create our game. Scratch is a programming language that has been specially designed to make animations and games with ease. Scratch Version 1.4 comes as standard with the Raspberry Pi but is also available on other computers. You can download it from `http://scratch.mit.edu/` if you ever want to play your game away from your Raspberry Pi. Start up Scratch by double-clicking on its icon on the desktop (it should have the picture of a cartoon cat). The following screenshot shows the Scratch layout:



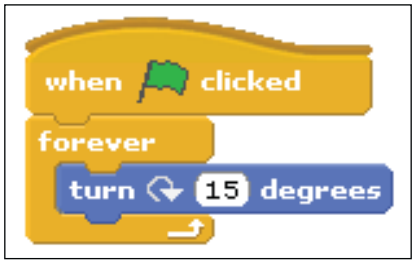
The following are its main sections:

- **Menu** (**1**): This is where the options are to save and load your projects. If you ever want inspiration to code 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, 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 of the sprites in your project are shown here, and you can easily add new pictures or change existing ones.
- **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 (like 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. From the **Motion** block type, drag a **turn 15 degrees** block into the script area (this example uses the clockwise turn block), and do the same for the "**when the green flag clicked**" and "**forever**" steps from the **Control** section. Connect them together by dragging one block close to the other. You should see a white highlight where the block needs to be placed. Release the mouse button and the block will snap into place. Click on the green flag present at the top-right corner of the screen to run the program.



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. You could also try adding other types of motion blocks within the **forever** block. Click on the red stop sign in the top-right corner to stop your program.

This is how the Raspberry Pi understands your program and knows what to do. It understands that the script should start when the green flag is clicked. 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 at the bottom of the **forever** block. If something keeps 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 on the game. A full description can be found online at `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 in 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 some 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 them. 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. We will use it later to work out when a bird in the game hits something.
- **Sound**: This enables us to play sound. You can add new sounds from the **Sounds** tab in the script area.

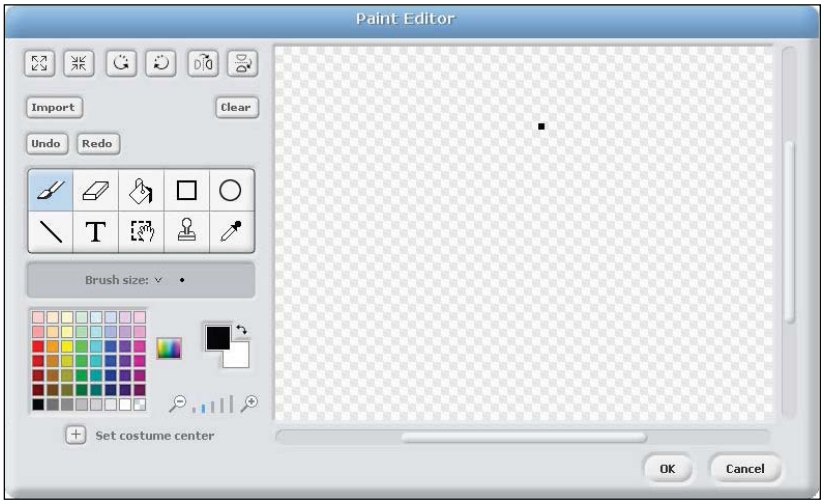
- **Operators**: These are simple mathematical functions, such as add and subtract. Note that some of the blocks are of different shapes; they show which blocks fit together and will be important later.
- **Pen**: This enables us to allow a sprite to draw a line to show where it has been.
- **Variables**: These allow us to give names to pieces of information so they can be accessed from multiple places. As an example, we will create a variable to hold the game score.

Creating a character

To start our game, we will need a character to fling through the air. Angry Birds, of course, used birds as its main characters, but we can use whatever we like.

At the top of the sprite list, you should see the three buttons shown in the previous screenshot. The first lets you draw your own character, the second lets you use an existing image (including a wide range of images included with Scratch), and the third gives you a random image from Scratch's selection.

If you click on the first button, you will be shown the following window; it has plenty of easy-to-use options for creating your own drawings. Hover your mouse cursor over any of the buttons to see what they do.

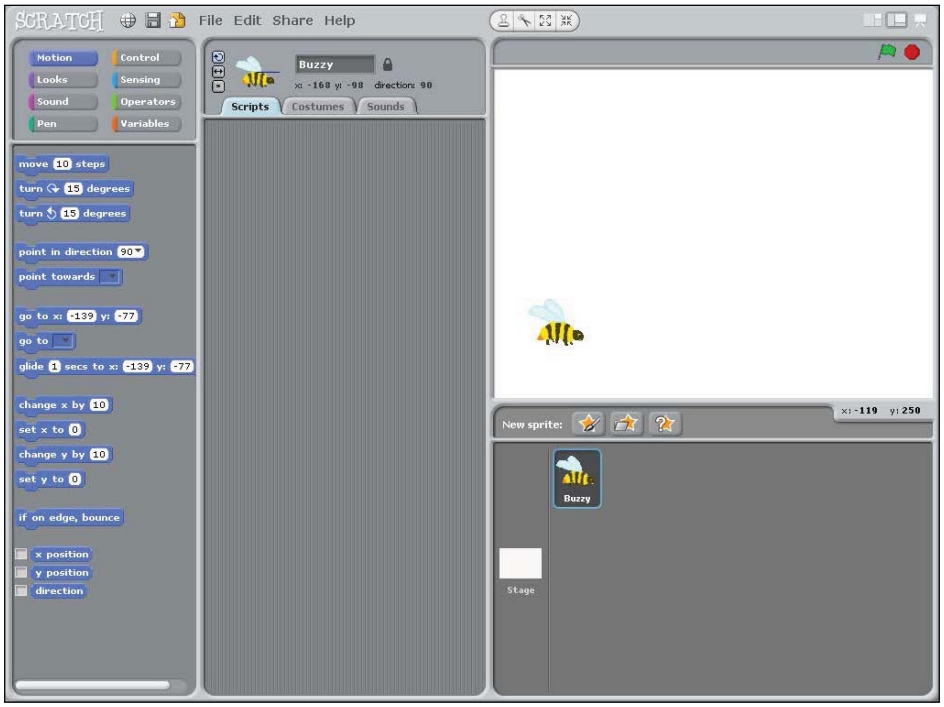


The second button brings up a fairly standard file explorer with lots of neatly categorized images. This is the option I will use, but feel free to do something different.

Once you have drawn or selected a sprite, click on **OK** to add it to the game. If you choose not to use the default cat character, right-click on it in the sprite list and click on **Delete** (this will also delete any code you have created for the cat). You can navigate to **Edit** | **Undelete** to bring the cat and its code back.

Now that you have a main character, drag it within the Stage to roughly where you think will be a good starting position, and resize it by clicking on the shrink button in the sprite controls and then repeatedly clicking on the sprite. I suggest making the sprite quite small so there is plenty of room around it to fly. Now would also be a good time to give your character a name—there is a textbox at the top of the script area that should say something similar to `Sprite 2`, which you can change to whatever you like.

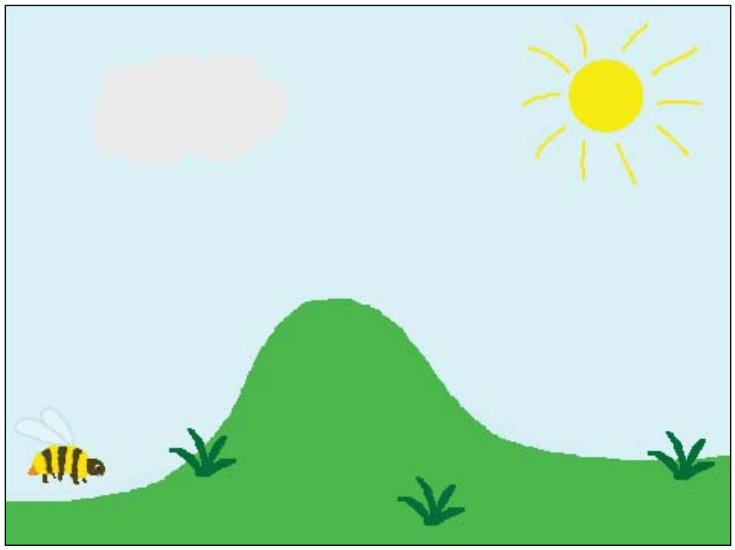
Your screen should now look something like the following image but with your own character instead of the bee that I have used:



Creating a level

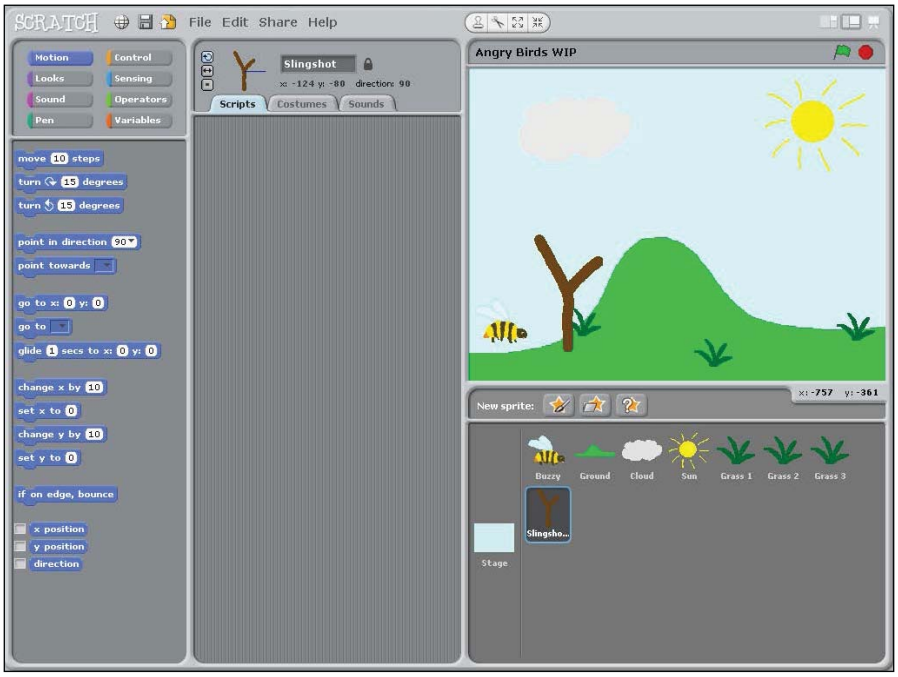
Now, let's make the game look a little more interesting by adding some scenery with the following steps:

1. At the left of the sprite list, you'll see a white rectangle called Stage. Click on it and then select the **Backgrounds** tab in the script area. Again, you have the option of drawing your own background or using a pre-existing image, but this time, I recommend creating your own so that you can make the level fun to play.
2. Click on the **Edit** button. Try to keep your background as simple as possible; it will be easier to add extra objects (for example, the ground, trees, and clouds) as additional sprites later because then you will be able to move them around more easily. It is perhaps easiest to simply fill the background with a solid sky blue color (and maybe some distant mountains).
3. Now back in the Sprite list, create sprites for all of the scenery you want in your game. At minimum, this will be the ground, but you can add all sorts of little details. With each sprite you create, remember to position it on the Stage, make sure it is the size you want, and give it a descriptive name. Remember that you can duplicate sprites using the left button in the **Sprite Control** area. When you have finished, you might be left with something like the following screenshot:



I have put a hill in the middle of the level to make it more challenging to hit the enemies on the right-hand side of the screen.

When you are happy with your level design, draw a picture of a slingshot and add it to the left-hand side of the Stage. Give it the name `Slingshot` so we are able to find it easily later on. Your Scratch window should now look as follows:



Moving the character

Now, let's start adding some code and making the game interactive! In this section, we'll do everything necessary to launch our main character using the slingshot.

Initialization

The first thing we want to do is make sure the position of our main character resets every time we start the game. Click on the main character and create the following script in the script area:

The code snippet states that when the green flag is clicked, the current sprite (the main character) will move to the same position as the slingshot.

Test that your code works by clicking on the green flag. You should see your character jumping to the same position as the slingshot. You may find that the character is behind the slingshot; if you would prefer for it to be in front, simply click on it on the Stage and drag it a short distance. Interacting with any sprite in this way will put it on top of all other sprites.

Moving with the keyboard

Now, let's allow the player to move the character around using the keyboard so that they can aim their shot. We are mostly going to be making use of this code block (from the **Sensing** section) but with different keys:

Before you read any further in this book, take a minute to have a look around the available code blocks. Can you find any useful blocks that we could combine with this block to move a sprite up, down, left, or right? This block is a strange shape; how can we connect it with the motion blocks?

There are actually a few different ways to do this, but in this book, we will use the following code block:

Hopefully, this looks fairly sensible to you. If the left arrow key is pressed, do *something*. That *something* may be a bit confusing, however, so here's a quick explanation.

The position of every sprite on the screen is given by two numbers (or coordinates). The x coordinate tells you how far left or right the sprite is, and the y coordinate tells you how far up or down the sprite is. The center of the Stage is at `(0,0)`, that is, both the x and y coordinates are zero. The x coordinate increases from left to right and the y coordinate increases from bottom to top. You can see the current coordinates of any sprite underneath its name in the script area, and the coordinates of the mouse are shown just under the Stage.

Since we want to move left when the left arrow key is pressed, we have to change x by a negative amount. In this case, it has the same effect as subtracting 5.

We will need one of these code blocks for each arrow key:

- The left arrow key should change x by -5
- The right arrow key should change x by 5
- The up arrow key should change y by 5
- The down arrow key should change y by -5

Finally, since we want the player to be able to press each button multiple times to continue adjusting their position, we need to put all of these blocks inside one big **forever** block. The **forever** block should be connected at the bottom of the existing script so the player can adjust the character's position after the position has been reset. Your code should now look as follows:

Once again, test your code out by clicking on the green flag. You should be able to move your character around by pressing the various arrow keys.

Launch!

Now that we've got the character in the right position, let's launch it! First, let's think about what we want to happen when the launch happens. We want to stop the player moving the character (so they can't cheat), and instead, we want to start moving it with a speed and direction dependent on how far from the slingshot the player is.

Since we are moving into a new phase of the game, it is a good idea to use a separate script when we launch. This will help keep each script relatively small and manageable. Add the following code to the **forever** block, where all of your other keyboard-handling blocks are, as follows:

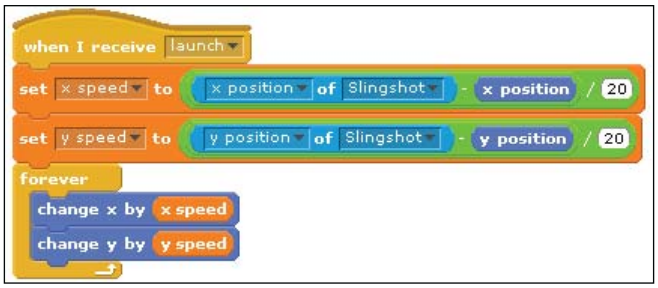
Here, **launch** is the name of a message. When the space key is pressed, launch is sent to all of the other scripts, and if any of them are waiting for that particular message, they will start to run. We also stop the current script so that we stop repeatedly checking which keys are being pressed and the player can't continue to move the character around.

Before we create the second script, we want to be able to calculate how fast to fling the character. To do this, we are going to store the speed in a variable. Variables allow us to store one value each and can be shared between different scripts. In this case, we're using a number, but variables can also store text. We are actually going to use two variables to store the speed: one for up-down speed and the other for left-right speed. The reasons for this will become clear later.

Click on **Variables** and then on **Make a variable**. The following window should pop up. Call your variable `x speed`, and make sure it is valid only for this sprite. Then do the same to create another variable called `y speed`. You can choose whether or not a variable is shown on the Stage by clicking on the little box next to it in the code block area.

Flight

Now that we have these variables, we can create the second script, which will control our flight through the air. The code for this is shown in the following screenshot. It's a little complicated, but try to work out what it does as you build the script up in the script area. I'll explain how it works shortly. (For the two long blocks that are almost identical, it is possible to create one, then right-click on it and duplicate it to save effort in creating the second one.)



The preceding script waits until it receives the **launch** message from the first script. Only then does it start. We set the **x speed** variable to a value that is relative to the distance between the character and the slingshot. I divide the value by **20** to make sure that the flight isn't too fast, but you may prefer a different value here. We then do exactly the same to compute the **y speed** value. Once the speed has been computed, we repeatedly move the object according to our speed.

We're now in a good place to test if everything is working. Click on the green flag, move around, and then launch using the Space bar. You should see your character fly in a straight line across the screen. You may want to try launching from different positions to see how this affects your speed and direction.

One thing that you may have noticed is that your character flies directly through the middle of the slingshot, not the part that it should actually be fired from. This is easy to fix. Click on **Slingshot** in the sprite list, choose the **Costumes** tab in the script area, click on **Edit**, and click on **Set costume center**. You can now drag the crosshairs around to choose a more sensible launch position. Once you have finished, click on **OK**. The slingshot will probably need repositioning on the Stage, but your character's flight should now follow a better path.

Adding physics

The next thing for us to do is to give our character a more interesting flight path. The game would be too easy (and no fun) if we just flew in a straight line through all the obstacles.