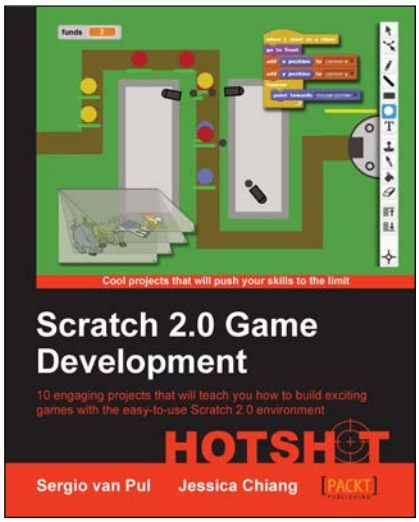
[PACKT] PUBLISHING

Scratch 2.0 Game Development HOTSHOT

Sergio van Pul

Jessica Chiang



Chapter No. 1

"Blowing Things Up!"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.1 "Blowing Things Up!"

A synopsis of the book’s content

Information on where to buy this book

About the Authors

Sergio van Pul is a game designer and artist interested in making interactive media entertainment. He has built games and interactive applications using Flash ActionScript. He has also worked on web designs and stylesheets, using a basic text/script editor to write raw HTML and CSS, as well as using the Drupal CMS to assemble a website. He is also familiar with visual editing tools such as Photoshop and Première.

Sergio has worked as a freelance designer and programmer on a variety of projects, many of which involved interaction and education. During this time, he met people from Scratch Web Foundation, a Dutch organization that promotes digital design and programming knowledge in primary education.

Sergio started using Scratch to teach children about programming and game design. Occasionally, he also uses Scratch as a quick and easy prototyping tool to test game interaction concepts. He likes experimenting with the program and building tutorials, examples, and complete game projects. Some of his material was printed and tested during workshop sessions for Scratch Web Foundation. This book is his first official publication.

I'd like to thank the people from Scratch Web Foundation. In particular, I'd like to thank Joek van Montfort and Helen Fermate for working with me and giving me the drive and opportunity to develop engaging Scratch projects. Some of the projects offered in this book started out as little thought experiments and workshops for Scratch Web. I'd also like to thank Jan-Pieter van Seventer, one of my game-design teachers, for notifying me about an interesting workshop assignment for children. This assignment solidified my interest in Scratch as a teaching tool for interactive media education. His brief note set me on the path that eventually led to the publication of this book. Finally, I'd also like to thank my co-author Jessica Chiang for stepping in and taking some of the burden from me of writing a whole book worth of engaging Scratch projects.

Jessica Chiang is a senior software engineer, online educator, and technology enthusiast.

She has worked with a wide range of interesting and cutting-edge technologies including nuclear detector and unmanned aircraft -control system. Not only an inquisitive learner, she also loves to teach in class as well as online through her website (`shallwelearn.com`) and YouTube channel (`http://www.youtube.com/user/kookoodoll`).

Jessica has self-published an e-book titled *Shall We Learn Scratch Programming: E for Everyone*. This book has been requested by many schools to supplement their computer science and education curriculum; one such school is Jessica's alma mater, the University of California, San Diego.

I would like to thank my husband Dr. Greg Chen for his loving patience and encouragement during the whole writing process. I also want to thank both of my sons, Matt and Vincent, for being the game beta testers. Finally, I want to thank my parents for fostering in me a spirit of curiosity and adventure. Without them, I would have neither started nor completed this book.

Scratch 2.0 Game Development HOTSHOT

Scratch offers a fun way of getting introduced to programming and interactive media design. Within minutes of starting the program, you can see the first results of your work. Visual feedback comes early and often, making high-level, abstract concepts a lot easier to understand.

Even without a specific plan in mind, it's fun to play and experiment with the software. You are always discovering and learning something new, and even failed projects can have funny or spectacular results.

Since Version 2.0, Scratch has moved from a desktop application to an online interface. Scratch 2.0 also includes many new and exciting features, which makes creating more advanced games possible.

This book presents a series of fully-realized interactive projects to work on. It will teach you how to build great games with lots of depth. The final results will be close to production level games. This book not only introduces you to the new features of Scratch 2.0, but also introduces you to interactive media design in general. You can take the lessons learned here and apply them to create games with tool sets other than Scratch.

We hope you enjoy working on the projects in this book. May they inspire you to create even better games!

What This Book Covers

Project 1, Blowing Things Up!, builds a simple game involving a cannon and some targets. You will learn about placing sprites, building scripts, and setting the game in motion.

Project 2, Beating Back the Horde, teaches you how to create multiple enemies and how to move them along a predefined path. You will also learn about drag-and-drop and click mouse controls.

Project 3, Start Your Engines, shows you how to build a keyboard controlled game. The game will showcase simple collisions between objects and how to handle them. You will also learn to create and use a timer.

Project 4, Space Age, shows you how to build a game that is extensive and configurable in terms of level of difficulty. This game comes complete with spaceship, shield, scoreboard, enemy, and an ample supply of ammunition.

Project 5, Shoot 'Em Up, shows you how to build a fast-paced action game with waves of enemies to defeat. You will learn about setting up movement patterns and speeds for both the player character and enemies.

Project 6, Building a Worthy Boss, shows you how to finish your side-scrolling shooter with a memorable boss encounter. You will learn to design an epic finish for a game level.

Project 7, Creating a Level Editor, teaches you how to create a tile editor and automatically build tile-based maps with it.

Project 8, Dungeon Crawl, involves you using the tile editor from the previous project to build an action RPG. You will also learn how to create multiple levels and different enemies.

Project 9, Hunger Run, shows you how to build a fast-paced auto-scrolling platform game. This project explains how horizontal and vertical scrolling work.

Project 10, Sprites with Characters, will dive into creating complex sprites using Scratch 2.0's vector editor. Piece by piece, we will build a robot and add animation scripts. The finished sprite can be imported to other Scratch projects.

Appendix, The New Scratch Interface, will give you an overview of the new Scratch 2.0 interface and will show you some of the new features you can play with.

Project 1

Blowing Things Up!

Scratch is a fun-to-use program that teaches you about animating, programming, and building games. You already know this because you have been making simple games with Scratch for a while, and now you want to learn more. This project will use some of the most important Scratch tools and explain some basic game programming principles.

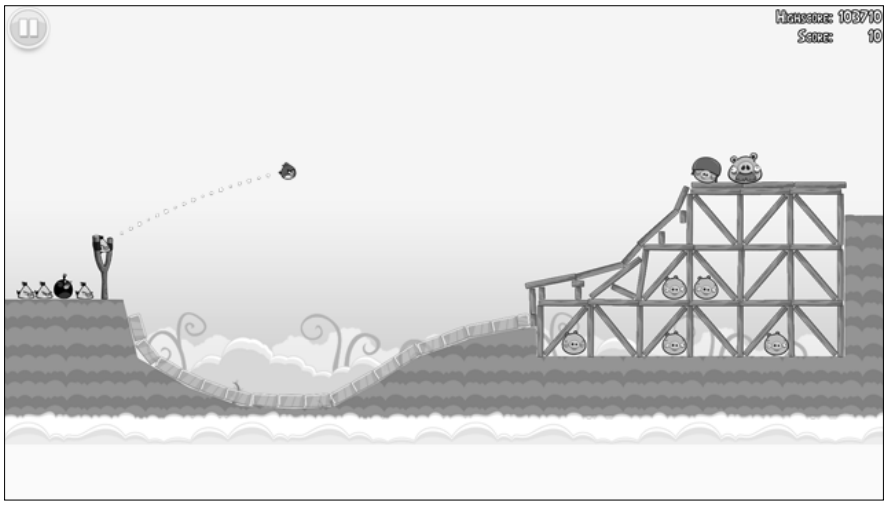
Mission briefing

We will make an artillery game. You might know this type of game from the very popular **Angry Birds** series, but this is actually a very old concept, dating back to the earliest computers. It was an obvious choice for imaginative programmers to turn military calculations into a game, because computers were originally used to calculate missile trajectories.

Why is it awesome?

We won't be able to guide any real missiles (luckily) with the scripts in this game. Instead of using proper mathematical calculations, we will use some simple tricks to get the desired results.

In games, it is rarely necessary to be absolutely realistic. Sometimes, bending the rules of reality creates more spectacular results; take Angry Birds, for instance:



We won't build a game as sophisticated as Angry Birds straight away. Our example will be more bare bones but still fun to play. In later projects, we will look back at this first example, and you will be challenged to add new things to this game to make it more interesting.

Your Hotshot objectives

In this project we will be:

- Creating a new project
- Starting scripts
- Adding targets
- Creating a parabolic shot
- Creating a landscape

While doing this, you will learn about (among other things):

- Drawing with Scratch
- Using variables
- The xy-coordinate system
- Operators and conditions (what has to happen and when)
- The very useful cloning feature to quickly duplicate objects

Mission checklist

To get started, go to the Scratch website (`scratch.mit.edu`) and start a new project by clicking on the **Create** button at the top of the page. If you already have a Scratch account, it might be useful to log in first, so that you can save your work in your account. If you are new to Scratch and are unfamiliar with the interface, have a look at *Appendix*, *The New Scratch Interface*.

Creating a new project

We need to make sure that we're logged in and ready to get to work on a new project. We'll then draw some sprites and assemble our awesome cannon (and cannonballs). The Scratch menu bar gives us the option to explore the existing projects from other users (using **Explore**) or create a new project ourselves (using **Create**):

Prepare for lift off

You are presented with a new project, including the **Scratch cat** as usual. We won't use the cat, so you can right-click on the sprite, and choose **delete**. A **sprite** is the official name for a 2D computer image. Most Scratch projects are built using sprites. You can find an overview of all the sprites used in a project in the bottom-left corner of the screen, underneath the **stage**.

Engage thrusters

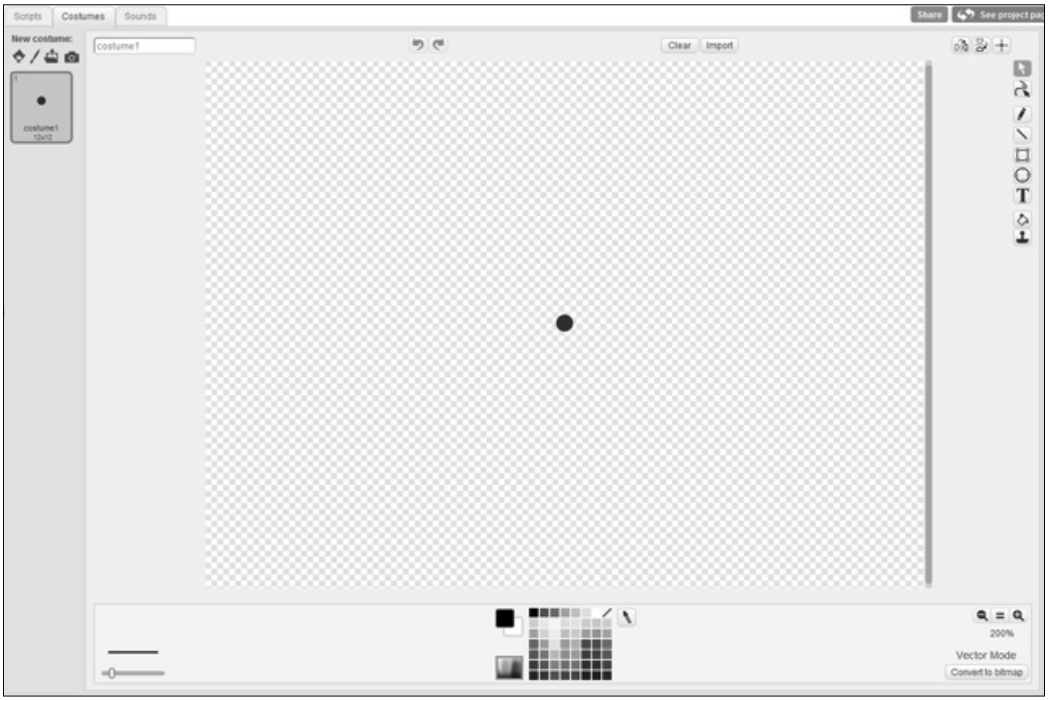
We will draw our own sprites for this game. Let's start with a simple cannonball!

The cannonball will be the main actor in this game because it will be the object that "blows things up". There are a few ways to add a new sprite to the stage. We can draw a sprite, select a sprite from the Scratch library, or import it from our hard drive. It's also possible to take a picture with a webcam.

To draw sprites, we perform the following steps:

1. Click on the little paintbrush icon between the stage and the sprites window. This will open the **costumes** tab.

2. Check if the editor is in **Bitmap Mode** or in **Vector Mode**. To create our drawing, we will select **Vector Mode**.

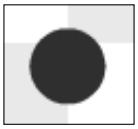


3. Select the Ellipse tool (the circle) seen on the right-hand side of the drawing canvas. Click and hold the left mouse button and drag it. You will probably end up with something that's not quite a circle.

4. Undo your drawing with the back arrow button at the top of the drawing canvas. When things go wrong, you can always go a few steps back. You can even clear the entire screen (using **Clear**) and start over.

5. To create a perfect circle, hold *Shift* while dragging the mouse. Don't make your circle too big. It has to be able to move about the stage freely without bumping into the edges all the time.
6. Next, click on the crosshair button at the bottom of the toolbar. This button lets you center your image. Now click on the center of your cannonball.
7. We name the new sprite `cannonball`.

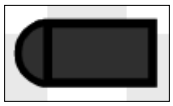
Your cannonball is now done and ready for business. It will look like a simple grey dot as in this screenshot:



Let's move on to creating a cannon to shoot from. The cannon will consist of two parts: the back of the cannon, which serves as the pivot point, and the barrel. The following are the steps to create them:

1. Create a new sprite by clicking on the paintbrush icon.
2. Start again by creating a circle. This circle should be slightly larger than your cannonball.
3. Next, select the Rectangle tool that is right above the Ellipse tool.
4. Draw a rectangle that is as high as the circle but about three times as wide.
5. Click on the Select tool and drag the rectangle to overlap the right half of the circle.
6. Make sure that the center point of the cannon is placed at the center of the circular element, near the back of the cannon.
7. We name this sprite `cannon`.

8. Just keep in mind that the cannon consists of two separate shapes. You can color them separately, or you can color them together by first selecting both the shapes while holding the *Shift* key.



Objective complete – mini debriefing

We've now got ourselves a cannon and some ammunition to shoot with.

Starting scripts

Let's have some fun making the cannon shoot its cannonball. It's always a good idea to script and test the interactive parts of your game as early as possible. Without scripts, it's just a bunch of pretty pictures! They might be nice to look at, but they won't keep the player entertained for long.

Engage thrusters

We have two objects to script. **Object** is an official programmer word that means something that performs an action in a program.

In this case the objects are visible. Our sprites are our objects. We have a cannonball and a cannon. The player will be able to control the direction of the cannon. The cannonball will fly away in a certain direction based on which way the cannon is pointing. So the way things are controlled is:

player → cannon → cannonball

Let's create a short script for the cannon. This script will save the direction the cannon is pointing to, so that the cannonball will know in which direction to fly.

We have to create a **variable** to store the direction of the cannon. If you're unfamiliar with variables, read the information box on the following page. To create a variable, follow these steps:

1. Click on the **Data** category. This is where you can create variables.
2. We will now click on the **Make a Variable** button.
3. Name the variable `direction` and make it available **for all sprites**.
4. We start the program with the **when <green flag> clicked** block. This is the easiest way to set any program in motion.
5. Underneath it, we will place a **forever** block, because we will check the cannon's direction indefinitely.
6. If you want, you can tick the checkbox to make the variable visible on stage. Then, you can see the direction that the cannon is facing in, at all times.
7. Put a **set () to ()** block in the **forever** block and select **direction**.
8. View the **Motion** category and look down at the list of blocks to find the built-in cannon **direction** variable. Place it in the open space. It may look superfluous to send the built-in variable value to a self-made variable. We do this because Scratch can't send built-in sprite variables to other sprites directly. Our self-made variable can be used in all sprites.

This is all the scripting that has to be done for the cannon. The following is the finished script:

About variables

Variables are an important part of programming. They allow you to store information for later use and to transfer information between different objects. A variable consists of two things; a name by which it is recognized and the word/number/object that it stores.

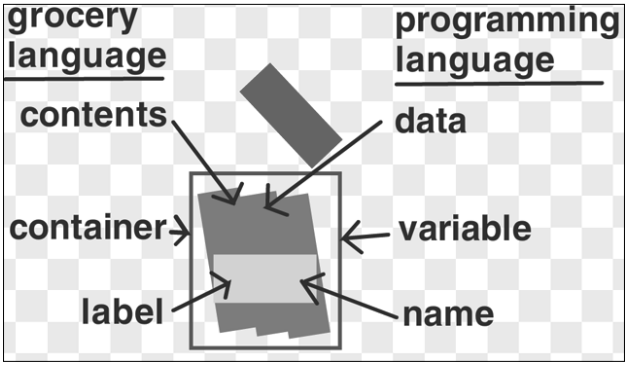
The easiest way to think about it is to compare the variable to a jar of pickles in a grocery store. The store clerk is the computer. He handles this jar and all the other jars that are available in the store. The jar is a variable.

On the face of it they all look the same, like glass containers. It's hard to distinguish one jar from another. That's why every jar has a label with a word on it. This is the name of the jar/variable. Most probably, the name of the jar will say something about what's in the jar. The jar of pickles, for example, will be called "Pickles".

You move up to the counter and ask the clerk, "How many pickles are in the Pickles jar?" and the clerk checks the jar, counts the pickles, and says, "There are 9 pickles in the jar." You now know something about the content of the Pickles jar. You feel like having a snack and decide to buy two pickles. After you paid and received the pickles, you ask again, "How many pickles are in the Pickles jar?" and the clerk counts the contents of the jar again (just to make sure) and answers, "There are 7 pickles in the jar."

It's most common to store numbers in a variable, because computers like to work with numbers, but variables can also contain names or even whole objects. Scratch keeps it simple, though. Scratch variables can only contain numbers or names/words.

Perhaps, it's better to illustrate the explanation of variables with the following screenshot:



We will go on to create a script to move the cannonball:

1. Click on the cannonball sprite to open its script editor.
2. The cannonball will also be triggered with a **when <green flag> clicked** block. Its actions will also be contained inside a **forever** loop.
3. Place a **move () steps** block inside the **forever** block. That's the basic necessity to set the sprite in motion.

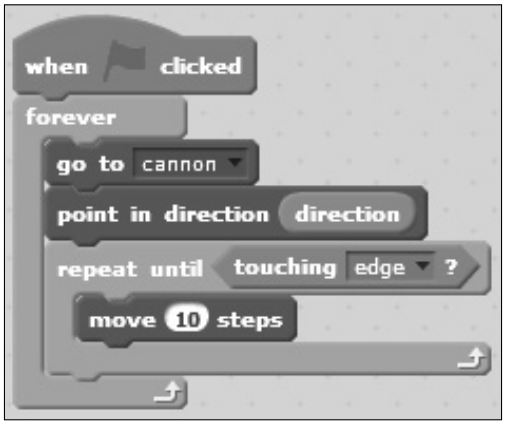
Now we build a few more controlling scripts, so the cannonball actually goes where we want it to go:

1. Place a **go to ()** block on top of the **move () steps** command and select **cannon**. This makes the cannonball hop back to its starting position.
2. Make the cannonball copy the cannon direction with a **point in direction ()** block and by insetting the **direction** variable. Note that this is our added variable (orange background) from the **Data** category, not the built-in (blue background) variable from the **Motion** category. The variable with the orange background is the saved cannon position. In this case, the variable with the blue background is the current direction of the cannonball, which wouldn't change anything when applied to itself.

To make the cannonball move forward, instead of constantly resetting, we use another kind of loop with a condition:

1. Put a **repeat until ()** block around the **move** command.
2. Then, place a **touching ()** condition block in the vacant space and select **edge**.

Now the cannonball will angle itself in the same direction as the cannon, and it will keep moving forward until it reaches the edge of the stage. At that point, the script repeats and the cannonball is reset to its starting position in the cannon. The following is the completed script:



We're not building any fancy controls at this point. Click on the **i** button in the top-left corner of the sprite in the **Sprites** window:

This will switch the sprites view to the **sprite properties** screen. Here you can view and edit some information about your sprites. An important one is the sprite name; you can name the sprite here if you didn't already do so in the sprite editor during its creation. Notice the sprite name in the top field of the properties screen:

What we are really looking for at this moment is the little **direction** tool located on the right side of the window.

Objective complete – mini debriefing

Click on and drag the little blue pin to change the direction that the cannon is facing in. Try it and see how the cannonball shoots in different directions depending on the direction of the cannon.

Don't forget to click on the **green flag** at the top of the stage (if you haven't already) to activate both scripts!

Adding targets

This game will be no fun without something to shoot at and blow up. So, we are going to create some targets for the cannonball to hit. We will first draw a new sprite. Then, we are going to use a very cool new feature of **Scratch 2.0**: the ability to create copies of a base object. This can save a lot of time when you want to have multiple objects that work the same way. This is often the case in games. Think of all the enemies you've squashed or all the coins you've picked up in various action games.

Engage thrusters

We will first draw a traditional archery style target, with a circular disk of red and white rings placed on a simple wooden stand, shown as follows:

To create the target, follow these steps:

1. Create another new sprite with the Paintbrush button.
2. Select the Ellipse tool and make sure the fill color is red and the border color is white.
3. Adjust the line thickness to create a fairly thick line.
4. Draw a vertical oval shape.

Don't worry too much about the size. We will adjust the proportions later. It's easier to draw big shapes first, so you can easily see the details and relative placement. When the drawing is complete, you can scale it down to the desired size. First, we need to create two more oval shapes.

Method 1

The first method to create these shapes is as follows:

1. Click on the Ellipse tool.
2. Place your cursor over the existing oval at the top-left edge of the red fill.
3. Click on it and drag to the lower-right edge of the red fill to draw another oval. This oval will fit neatly inside the first one.
4. Repeat these steps to create a third, even smaller oval.

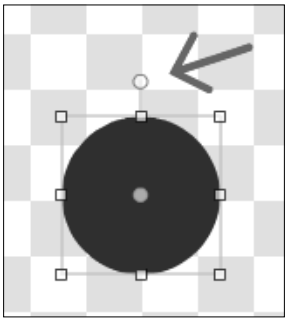
Method 2

The second method to create these shapes is as follows:

1. Click on the **Duplicate** button in the toolbar.
2. Click on the oval shape you have already made.
3. When you move the cursor, you will see a transparent "ghost" of the circle, moving along with the cursor.
4. Click anywhere on the drawing canvas to place a copy of the oval there.
5. Use the scaling widgets you see around the shape when it's selected, to scale the copied circles down to the right size.
6. Drag them inside each other to create the finished target disk.

Now that we have our oval shapes, we can continue building our target!

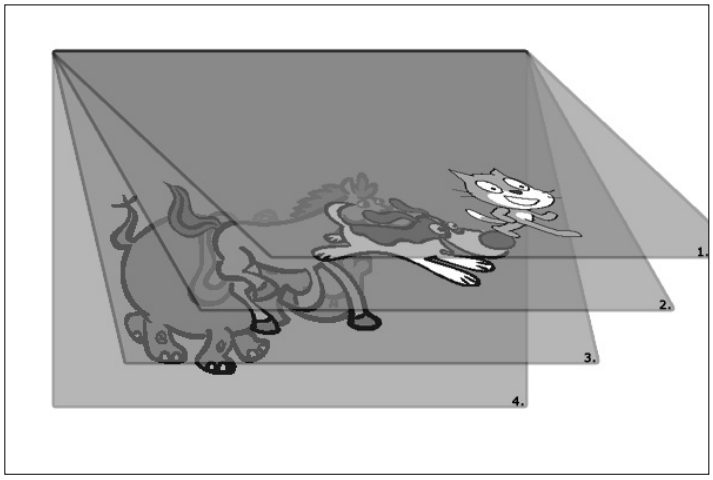
1. Click on the appropriate sample from the color swatches to change the line color to brown.
2. Click on the Line tool and draw a vertical line, about as high as the target disk.
3. Draw another line diagonally down from the upper tip of the first line.
4. The two legs of the target stand are complete.
5. Click on the Select tool and drag a box around the entire target disk. This will select all three shapes as a single object.
6. Click on and drag the **rotation pin** at the top of the selection box and rotate the disk upward.



7. Click on and drag the shape over the target stand. The center of the disk should line up with the top of the stand.

About layers

Perhaps, your shapes are overlapping in the wrong way, with the stand on top of the disk, instead of underneath it. This means that the shapes are sorted in the wrong order. A Scratch drawing consists of separate layers, like sheets of transparent papers stacked on top of each other. The sheets on the top will cover the sheets underneath. This way you can create the illusion of depth, by placing objects that should be far away at the bottom of the stack.



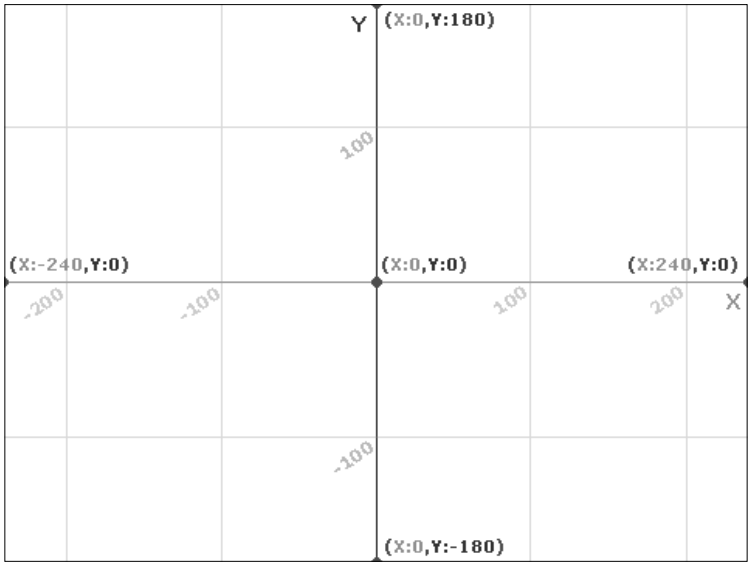
8. Change the order of shapes with the Forward a layer and Back a layer buttons, if needed.

9. Select the entire drawing by dragging a box around it with the Select tool.
10. Scale the entire drawing down to the desired size. You can scale proportionately (horizontally and vertically, evenly) if you hold a **corner widget** while dragging.

Creating multiple targets

We will write a script for the target we just drew; this script will place copies of the object at random locations on the stage. To do this we will use the new **clone** block. This is one of the most exciting new features of Scratch 2.0. Instead of manually copying your sprites N number of times, you can just use a script to do this work for you. It can save a lot of time when creating and editing objects.

1. This script will start with a **when <green flag> clicked** block, just like the earlier ones.
2. Attach a **go to x: () y: ()** block. Fill in the numbers `-100` and `0`.



About X-Y coordinates

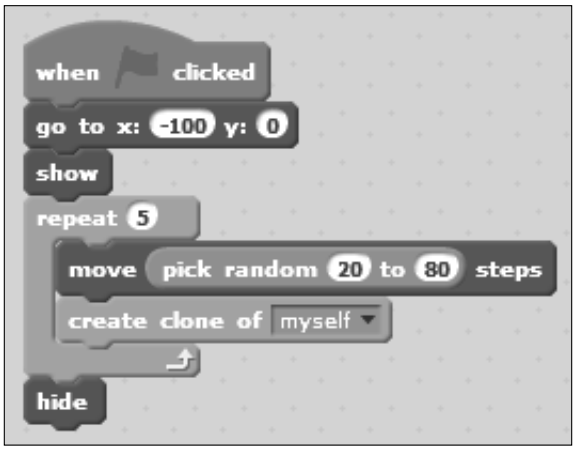
X stands for the horizontal position, that is, how far left or right something is. Y stands for the vertical position, that is, how high or low something is. This way the computer can easily save the position of any object on the stage. Look at the bottom-right corner of the stage. Here you will see the current position of the mouse shown as X and Y coordinates. This can be a helpful tool when deciding where you want objects to appear on stage with a script. Just point to the right place, look at the numbers, and put them at the right places in the script.

The center point of the Scratch stage has the coordinates **(X:0,Y:0)**. The horizontal positions range from **-240** to **+240**. And the vertical positions range from **-180** to **+180**. If you look at the assignment for the previous target, you will notice that the target is placed somewhat left of the center (-100) and on the center line vertically (0).

3. Next, add a **show** block. Yes, the sprite is already visible now, but at the end of the script we will make it disappear. This block makes sure it appears again in time when the script runs.

We are going to make five target clones. We will let the target sprite step right five times and create a clone of itself at each step.

1. Attach a **repeat ()** block and fill in `5`.
2. Inside the **repeat ()** block place a **move () steps** block.
3. Instead of a fixed number, use a **pick random** `20` **to** `80` block to make the spots where a target will appear a little unpredictable and more interesting.
4. Then, attach the new **create clone of ()** block underneath the **move** command inside the **repeat** block. Select **myself**.
5. Finally, use the **hide** option on the original sprite. Place this block at the end, outside the loop.



So now we have a cannon, a cannonball, a bunch of randomly created targets, but still no exciting game. The cannonball can fly through the air, but it doesn't do anything when hitting a target; it just passes right through. This can be easily fixed.

We'll continue scripting the target first. After a clone is created, you can start running a script on the clone. This is a new way of initiating a script.

1. Start a new script in the target object with a **when I start as a clone** block.
2. Attach a **wait until ()** block. This will pause the script until something happens.
3. Place a **touching ()?** condition block inside the slot. Select **cannonball**.

4. The next step is to attach a **wait () secs** block.
5. Fill in a very short time of `0.05` seconds. This might seem a little useless, but it will give the other scripts (specifically the cannonball script in this case) time to respond before the target disappears.
6. The last step is to **delete this clone**.

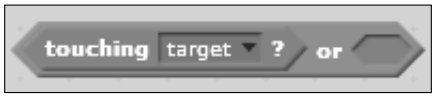
Cannonball collisions

The targets disappear when hit by the cannonball, but the cannonball can go on through multiple targets. This makes the game a bit too easy. It would be better if the cannonball is stopped by hitting a target as well. So a new cannonball has to be aimed and shot for each target.

Making the cannonball disappear on contact with a target just requires a little addition to the existing script. The cannonball is already reset to its original position when hitting the stage edge. We can use this already existing script and also check for hitting a target.

1. Click on the cannonball sprite in the **Sprites** view to see its scripting panel.
2. Grab an **() or ()** operator block.
3. Pull the **touching <edge>?** block from the script and place it in one of the **() or ()** slots. It doesn't matter which one.
4. Also get a new **touching ()?** block. Place it in the other slot and have this condition checking for `target`. (Have you already properly named your target sprite?)
5. Place this entire conditional structure in the now vacant condition slot in the existing script.

The cannonball gets reset when it touches the edge or a target. It doesn't matter that the target is a clone. It is still called a "target".

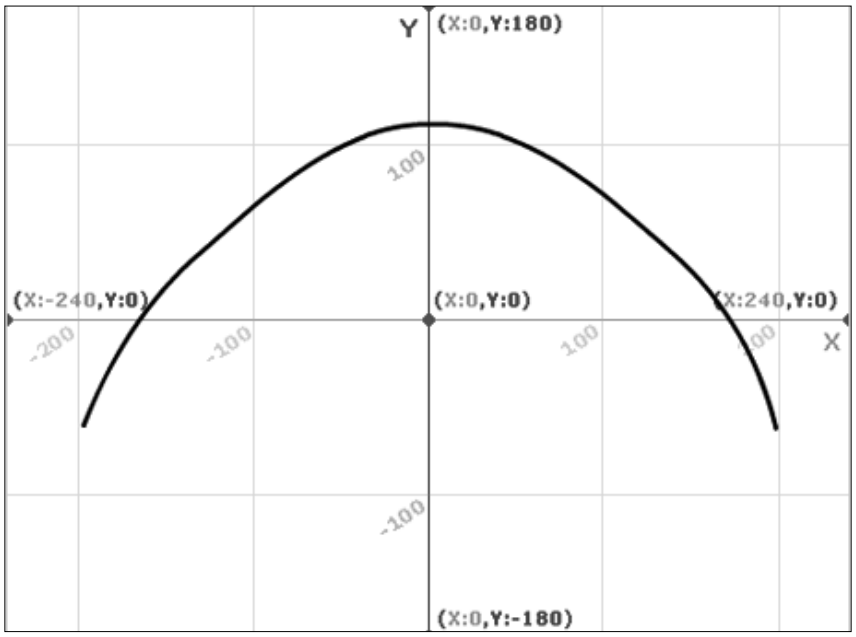


Objective complete – mini debriefing

Currently the cannonball is moving on in a straight line. In reality, a cannonball doesn't move like that (but we'll be fixing this). It is heavy, and what goes up must come down. You can try it yourself with a ball or a stone. Throw it upwards in front of you and see what happens. Just be careful with the neighbor's windows!

Creating a parabolic shot

The ball will move upwards in the direction you threw it. At some point it will start to slow down, stop, and then start falling down. The path the ball follows is called a parabola. This is what we expect of gravity. It's always around us, so we don't pay too much attention to it. If you would throw the ball in space, which has no gravity, the ball would move on forever in a straight line, as it does now in our game. On Earth, the ball will move up for a while, then slow down, and eventually fall down again. This movement path is described in the following diagram:



In this case, the cannon and cannonball should be on the ground, so we want to simulate some gravity. Not only will the parabola trajectory look more interesting, it will also make the game more challenging. You could try hitting a target by shooting the cannonball at a steep angle and have it hit a target as it drops down. It is very useful to shoot over the hills, which we will do in the last stage of this project.

Engage thrusters

To simulate gravity, you could use a realistic mathematical formula. But it can be hard to figure out, and in a simple game, it often isn't needed. We are going to create a good-looking parabola trajectory with just a simple calculation and some trickery.

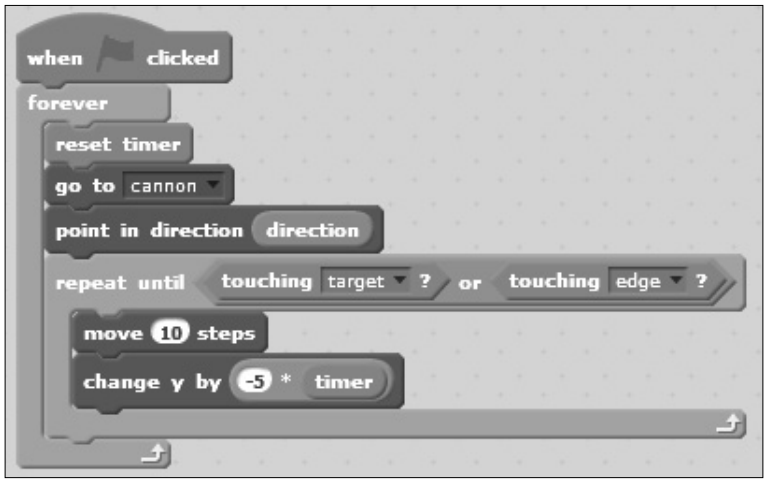
To simulate the pull of gravity, we are going to use the built-in **timer** variable. This timer will start counting seconds when the game is started, but we can reset it to start counting again. We will be using the increasing number as a constantly increasing pull, which will eventually start dragging the cannonball downwards.

The steps to create the trajectory are as follows:

1. We select the cannonball sprite to add to its scripts.
2. After the **move () steps** command, add a **change y by ()** block. This will cause the cannonball to vertically move a bit after taking its steps.
3. Place a **() * ()** (multiplication) operator in the box.
4. Then put the **timer** variable in one of the operator slots.
5. In the other slot, fill in a number manually. Try a few numbers, just for the fun of it, and see what happens.

   In my opinion, the number **-5** has the best result. But if you disagree, you're free to choose another number. Just make sure that the number is negative, because a positive number will cause the cannonball to float ever faster upwards.

6. To reset the timer after the cannonball has hit something, add a **reset timer** block at the start of the loop.

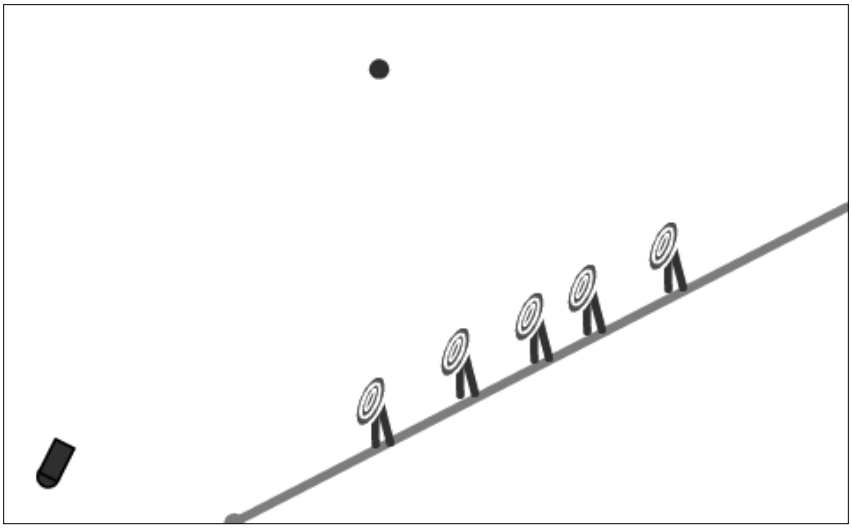


Objective complete – mini debriefing

Test the game again and turn the cannon to point at different angles. See how the parabolic trajectory of the cannonball responds.

Creating a landscape

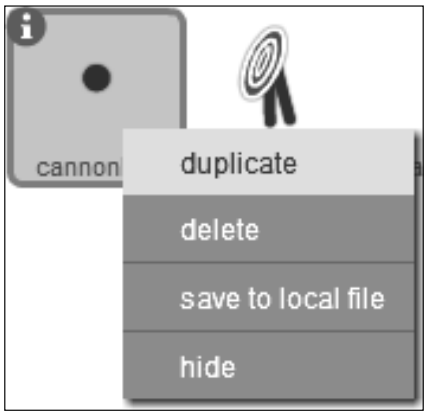
As the final stage of this project we will create a hill for the targets to sit on. The landscape will have a diagonal slope; we have to change the parabola trajectory to shoot at different points on the hill. The finished game will look like the following screenshot:



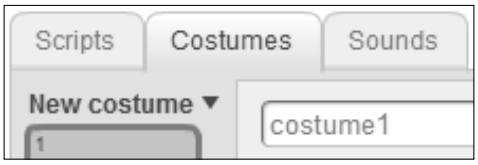
Engage thrusters

We could manually draw landscapes as sprites. But if we want to create many different levels, it could take a lot of work. Instead, we will make a drawing tool to create the hill.

1. First, create a copy of the cannonball by right-clicking on the sprite in the **Sprites** view and choosing **duplicate**, shown as follows:

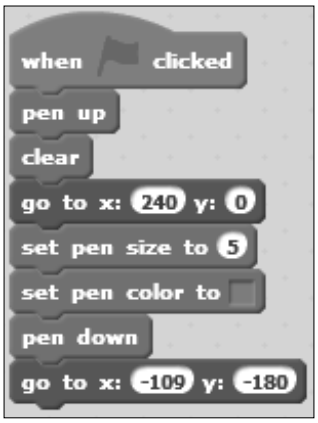


2. Click on the **i** button on the copied sprite and change its name to `drawing tool`.
3. Throw away the scripts in the copied sprite. Those are only useful for the cannonball, not for the drawing tool.
4. Switch to the **Costumes** tab and change the color of the drawing tool to dark green using the color swatches. This is mostly to make it visually different from the cannonball, so that we don't confuse the two.



5. Go back to the **Script** tab so that we can create a new script for the drawing tool.
6. We start the script again with a **when <green flag> clicked** block.
7. Attach a **pen up** block to make sure the tool doesn't draw anything while moving to its starting point.

8. Add a **clear** block. This might not look useful now, but like with the **hide** and **show** commands in the target sprite script, this will help clear the screen once we want to restart the game.
9. Move the tool to its starting point with a **go to x: () y: ()** block.
10. Fill in `240` after **x** and `0` after **y**, so the tool starts at the right edge of the stage.
11. Next, set the pen size to `5` using a **set pen size to** block. Again you might want to create a thinner or thicker line.
12. Choose a green color for a natural-looking hill in the **set pen color to …** block.
13. Place the drawing tool on the stage with a **pen down** block. We are now ready to start drawing the landscape.
14. Then, we move the pen diagonally to the bottom edge with a **go to x:** `-100` **y:** `-180` block. Because the pen is down, it will draw a line between its start and end point.



To make full use of our newly drawn landscape, we have to set the targets down on them. But how do we do that on such an uneven surface? The solution is to slowly move the targets down and use another collision test to decide when the targets have reached the landscape and should stop moving. We add the following script to the target sprite:

The steps to perform this test are as follows:

1. Click on the target sprite so you can view its scripts.
2. Add a **repeat until ()** block at the start of the clone script.
3. Inside the **repeat until** loop place a **change y by ()** block and fill in a negative number to make the targets move down.

   I filled in the number `-4` for a fairly slow speed. You may take bigger steps, but then the targets could end up being stuck partly through the landscape. Not a big issue, but it might not look as nice.

4. To stop the targets from moving when they reach the landscape, add a **touching color ()?** condition to the **repeat until ()** block.
5. Click on the color box and then point and click on the green line that you've drawn.

Computers are very precise about color. Keep this in mind if your color collision doesn't work. Most likely, the actual color of the object will be slightly different from the color that you checked for. You can't see it with the naked eye, but the computer can tell the difference based on the color number. The finished script should look like the following screenshot:

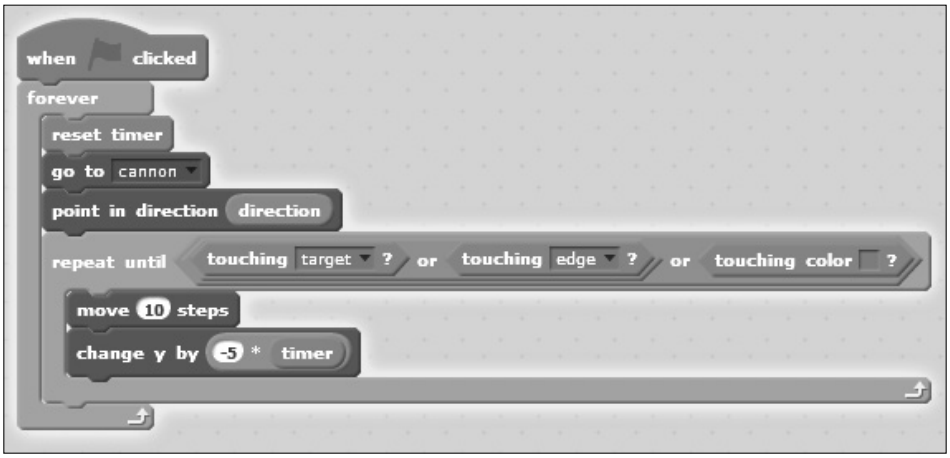
As a final step, change **y:** `0` to **y:** `180` in the other target sprite script. This will place the target at the top of the stage and make sure that it doesn't end up inside or underneath the landscape.

Now that we have another object on the stage, that is, the landscape, there is one more thing we have to do to finish the game. When the cannonball hits the ground, it should stop instead of moving straight through the landscape. This is similar to the addition we added after including the targets. Now we only need to add another collision check for the cannonball to respond to.

The following are the steps for this collision check:

1. Get another **() or ()** operator block.
2. We will check for hitting the green landscape color, just like we just did with the targets.
3. In the other slot, we will place the entire condition check, like we made earlier. So our latest **() or ()** block will become the outermost block in the construction.
4. Place the entire construction back in the **repeat until ()** slot and we're done.

The cannonball will now respond to hitting the stage edge, a target, and the landscape.

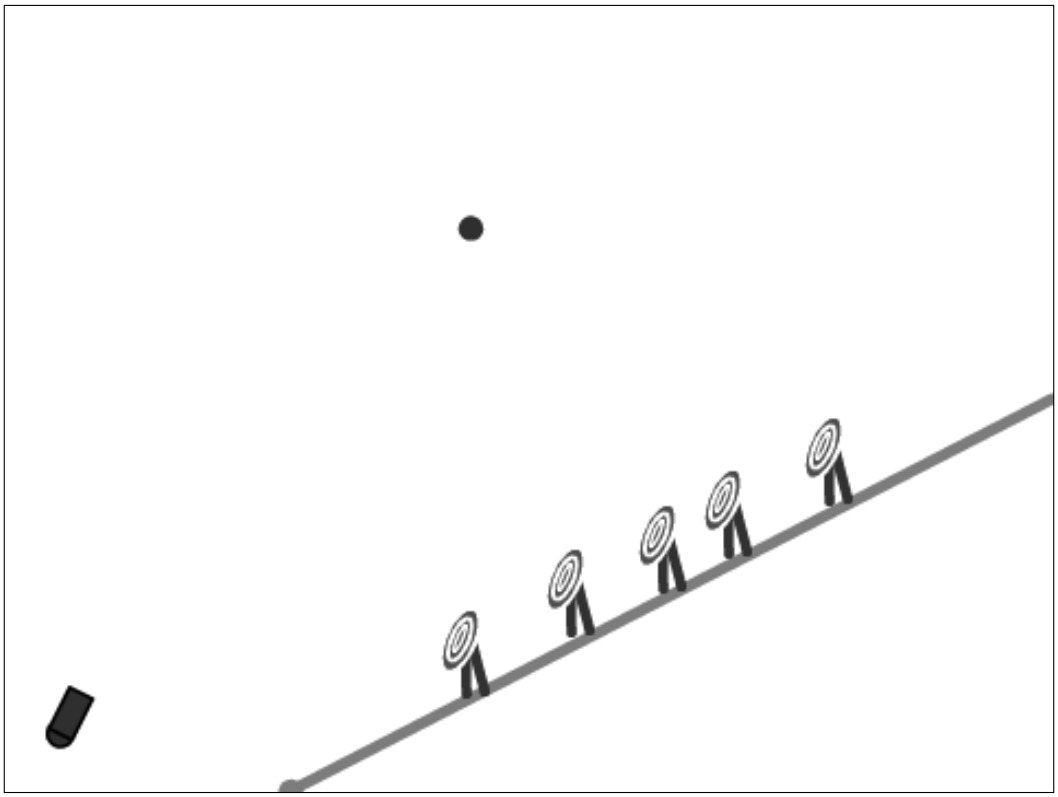


Objective complete – mini debriefing

That's it! We should now have an automatic landscape complete with a functioning cannon, firing script, and targets.

Mission accomplished

Your first game is now done, well, sort of. You learned how to draw sprites and how to add scripts to them to make them interactive. We used the new cloning feature to quickly copy the same sprite, including its functionality. We worked with collisions between objects, some variables, and simulated gravity to create a simple, but challenging game. The finished game will look like the screenshot that follows:



Hotshot challenges

It's still far from the Angry Birds example that we mentioned at the start, but all the basic elements are there. Test your game, and play with the numbers a bit to see how they can affect the playability! Have a go with the following tasks:

- If you increase the cannonball speed to very high, it will become very fast, but hard to control.
- Try increasing or reducing the number for the simulated gravity effect.
- Do you still know which number that is? It is somewhere in the cannonball script.
- Can you change the number so it feels like the game takes place on the moon, where gravity is not as strong as it is on Earth?

In later projects, we will look back at this game and you will be challenged to expand and improve it, based on the things you will learn in those projects. The game might not look like anything special just yet, but with some effort and imagination, you could make it into an exciting game to rival Angry Birds.

Where to buy this book

You can buy Scratch 2.0 Game Development HOTSHOT from the Packt Publishing website: `http://www.packtpub.com/scratch-2-game-development-hotshot/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

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

www.PacktPub.com