Building Games and Animations With Scratch

By Andy Harris

Computers can be fun—no doubt about it, and computer games and animations can be especially appealing. While not all games are good for kids (in this way, computer games are no different than books or movies), games are clearly appealing to kids.

It might be possible to channel this fascination into something that supports the learning mission rather than detracting from it. Used properly, games and animations can be used as an interesting way to motivate history lessons, demonstrate ideas, explore the world, and (especially) reinforce math ideas. The best way to make games work for good is to see gaming as a creative endeavor. It's fine to play games made by others, but it's much more fun to build your own.

Game programming has long been known as one of the most challenging forms of computer programming. The demands of gaming are challenging, and the need to constantly push the limits of the hardware requires significant mathematics. Still, there can be ways to teach game development to kids.

Introducing Scratch

Game development does not have to be difficult or expensive. The Lifelong Kindergarten Lab at **Massachusetts Institute of Technology** developed a remarkable free tool called **Scratch**, which enables even young students to create remarkable games and animations with no previous programming knowledge. Along the way, they learn quite a bit about math, science, and logic.

In this article, I will introduce you to Scratch and show you how to use it in your homeschool. It will be great fun. However, this is not a tutorial. I'll show you exactly how to get started, but then the rest of the article is designed as a series of challenges. I'll give you hints and examples, but part of the fun is working together with a friend or parent to discover how everything works.

The first step is to download and install Scratch. This program is available for free for all major operating systems: **scratch.mit.edu**. While you're at the MIT website, look over all the great resources there. The kids will probably just start playing, but parents may want to

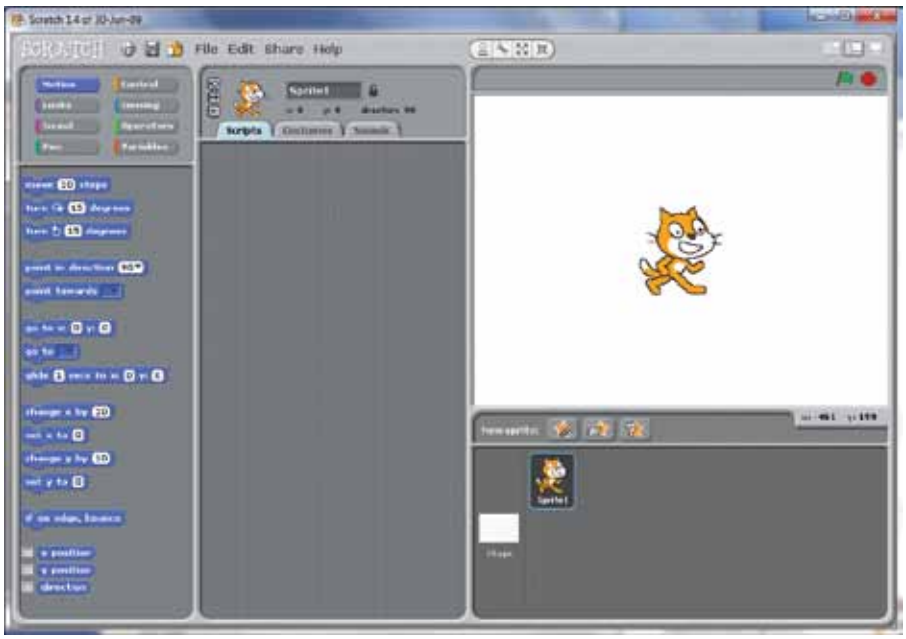

Figure 1

Scratch . . . allows us to introduce extremely sophisticated ideas in math, science, and computer programming in a way that's fun for kids and adults to learn together.

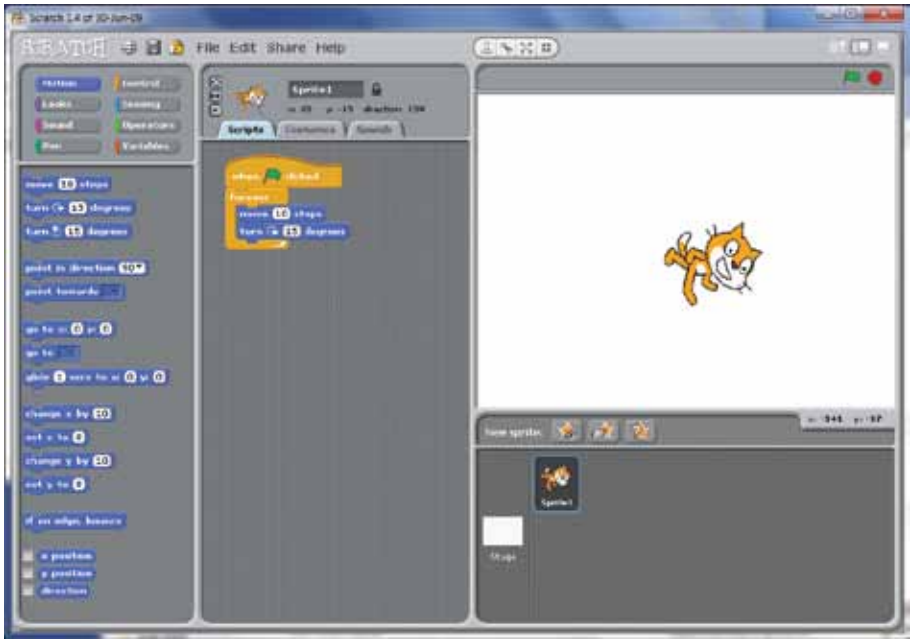

Figure 2

learn about the many useful help features available at the MIT site. The best way to learn Scratch is to just start using it.

Scratch, like most 2D gaming and animation tools, is based on the notion of sprites. A sprite is an image with special properties; it can move, turn, and bump into other sprites (and often it can do much more, but those are the basics).

If you look at the right-hand panel you'll see a cat in a white box. The white box is your stage, and the cat is your first sprite. (The default sprite is called Scratch the cat, the mascot of the Scratch project.) The panel farthest to the left contains a series of buttons. Each of these buttons indicates something the current sprite can do. Try this experiment: Be sure the blue Motion buttons are visible, and double-click the Move 10 Steps button repeatedly. Watch what happens to the cat. He moves!

Now take the Move 10 Steps button and drag it into the large center area. It will just sit there. Now take one of the Turn 15 Degrees buttons, and drag it into the same center area. If you get close to a button that's already in the center area, you'll see a white bar, indicating you can join the two buttons together. Do that, so that the two buttons are joined. (It doesn't matter which order they are in for now.) You've put two commands together to make something new. If you double-click this new combination, it will both move and rotate that cat. Congratulations! You're now a computer programmer. It gets a little harder than this, but not much.

There are a couple more steps to do before you can call this first animation "finished." The blue buttons are all about changing the motion of the cat, but there's much more you can do. The buttons in the upper-left corner all let you access different trays of commands. Click the yellow Control button to see the commands about controlling the general flow of behavior. Many of the most important commands are here. Find the one that says Forever. It looks to me kind of like an alligator mouth. (Don't use Forever If; that's a different command.)

Take the Forever block and drag it to the combination you've just made in the center area. If you get it close enough, the "mouth" opens up and the Forever block surrounds the other elements. If you double click this, the cat will quickly spin around! Give it a try!

Finally, it's nice to add a cleaner way to start and stop your programs. Within the same set of Control commands, you should see a button that says When <green flag> Clicked. This button looks kind of like a hat, because it's meant to indicate the beginning of a block of code. Drop it right on top of your other code. When you're finished, your code should look like Figure 2.

Sliding Back and Forth

I've walked you through the first project, but this is about gaming, so the rest of the exercises are positioned as gaming challenges. I will provide you with a sample and some hints, and your job is to figure out how to make the sample. It's best to work with a partner, and you can look things up online, but really you just need some imagination and willingness to play around. If you get stuck anywhere, I have solutions to all the exercises on my website (**www.aharrisbooks.net/scratchTutorial**).

Your first challenge is to make Scratch the cat walk back and forth. Here's my sample: **aharrisbooks.net/scratchTutorial/backForth.html**.

Note that the first time you try to open this file, you might be prompted to install or update **Java**. This is a technology often used in web programs. The form of Java used in this example is completely safe. Please allow the use of Java for the example programs in this tutorial.

The first example involves making the cat walk back and forth on the screen. There are a few main ideas to notice:

- Scratch starts moving when you click the green flag.

Used properly, games and animations can be used as an interesting way to motivate history lessons, demonstrate ideas, explore the world, and (especially) reinforce math ideas.

- He will move forever. (That was a hint.)
- If Scratch ever hits an edge, he should bounce. (That was another hint.)
- He should only face left or right. (This one is tricky. Look for a set of tiny buttons on the top center panel.)

You'll see some buttons that control the cat's orientation. Don't be afraid to experiment!

Shall We Dance?

After you've made the cat glide, it's time to make him dance. The goal is something like this: **aharrisbooks.net/scratchTutorial/dance.html**.

Note that Scratch seems to be changing appearance. Sprites have images. Right now your sprite always looks the same, but he has two images, and you can add as many as you want. We call the sprite images "costumes." If you look at the center panel (where you've been making scripts), you'll see a Costumes tab. Click on this tab, and you'll see two costumes. Click the copy button next to one of the costumes and then click the Edit button.

Scratch includes a complete paint program! You can change your sprite image or draw your own. You can also load an image from the library or take a picture of yourself with a web cam and use that as the foundation of an image. I'll stop here and let you play, because these features alone can keep you busy for some time.

> . . . Part of the fun is working together with a friend or parent to discover how everything works.

Once you have a number of images in a sprite, you can swap between them pretty easily. Look for the Next Costume button under the Looks tab. This will display the next costume in the list and cycle through the list of costumes indefinitely.

If you swap images in a Forever loop, the images might change too quickly to be realistic. (My sprite looks a little frantic at full speed.) Look for a way to slow things down (there's a wait command in controls—just saying . . .). The default wait time is one second. That might be too slow. Is there a way you can change the speed of animations?

Tell a Joke

The next challenge is to use Scratch to tell a simple joke. As usual, here's a working example. Warning: It's a really bad joke: **aharrisbooks.net/scratchTutorial/joke.html**.

Here's what this animation does:

- It has two sprites.
- Each sprite starts invisible and off stage.
- One sprite walks in and calls the other.
- The second sprite comes in.
- They tell a joke.
- The joke deserves a rim shot (and gets one).
- The sprites walk off stage (. . . and none too soon).

Planning and preparation are important parts of this project. First, find a better joke than mine that you want to tell. Choose or create some characters and a background. Once you've decided on the characters, the plot, and the scene, you'll need to investigate a few new tricks to make the program work.

You've probably used the Move 10 Steps command before, but this command is not the only way to manage motion. Two numbers determine the position of the sprite: X relates to the

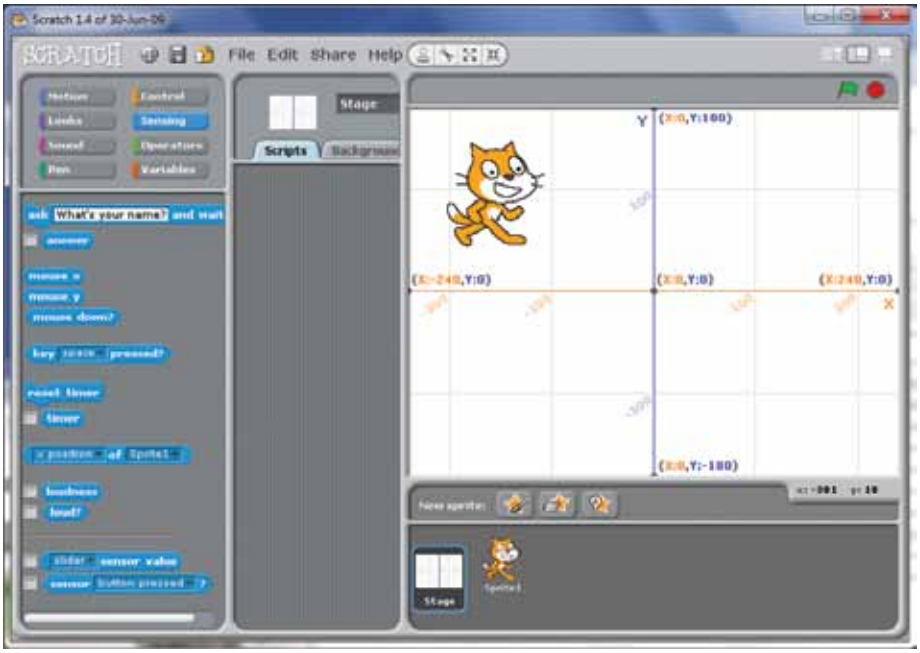

Figure 3

side-to-side position, and Y relates to the bottom-to-top position. The center of the screen is (0, 0). You can see how the screen is organized by selecting the stage.

Go to the Backgrounds tab and import a background. In the main directory of the backgrounds you'll see a special background image called the XY-grid. This tool shows the XY coordinate system and is very handy when you're trying to figure out where you are on the screen. Of course, you can replace it with some other background when you're ready to move on. Figure 3 shows the Scratch editor with the grid showing.

The Move To button in the Motion panel moves a sprite to a particular spot immediately. It's easiest to determine positions if you start with the grid as a background. Also, when you first move to the Motion panel, the Move To and Glide To buttons are pre-set with the current position.

You can't make a sprite completely leave the stage, but you can hide and show sprites. When the animation starts, both sprites should be invisible.

Move your first sprite to the center of the scene and say something (if you want).

The easiest way to synchronize between two or more sprites is by using the message mechanism. Any sprite can send a message. (There's a Broadcast command on the Control menu.) Choose New to create a new message.

Every sprite can also listen for messages. The Control panel includes a special command called When I Receive. You can use this command to listen for any messages broadcast by any sprite. This works much like actors on a stage. In my example, when the program starts, the cat and dog are both hidden off stage. When the cat is finished with his line, he broadcasts the "CallDog" message. The dog is patiently listening for "CallDog" and moves onto the screen when he hears his message. Messages represent an extremely powerful mechanism. My joke program uses a number of other messages to pass control from one sprite to the other.

My example also uses a sound effect. Sounds are attached to sprites and are modified through the Sounds tab. You can record your own sound or use one of the built-in sound effects. The Sounds panel has buttons for playing back sounds. Mess around a bit to get the effect you're looking for.

Up, Down, Left, and Right

The difference between an animation and a game is user control (Okay, there are a few other factors too, such as a goal and an obstacle, but go with me here.) Your next job is to make a sprite that moves up when you press the Up arrow, down when you press the Down arrow, and (I think you're catching on) left and right when the appropriate arrows are pressed.

Here's an example: **www.aharrisbooks.net/scratchTutorial/moveKeys.html**.

User input involves getting some kind of signal from the user, normally from the keyboard or the mouse. This is done with a combination of two types of buttons: (1) if statements and (2) sensors. When you use the word *if* in English, you're normally testing to see if something is true. That's exactly how if statements work in Scratch. On every frame (several times a second), we need to check to see if a key is being pressed. This means you need a Forever loop. Inside that loop, you'll need an If command. (Both are found in the Controls tab.) Note that If has a little hexagon shape inside it. This indicates you need a condition (a true or false test).

Look into the Sensing tab and you'll see a great number of tests. The one you're looking for is Key <space> Pressed. Now place a movement command, and when you press the appropriate key, you'll move in the indicated direction. Figure 4 shows this in action.

Figure 4

Figure 5

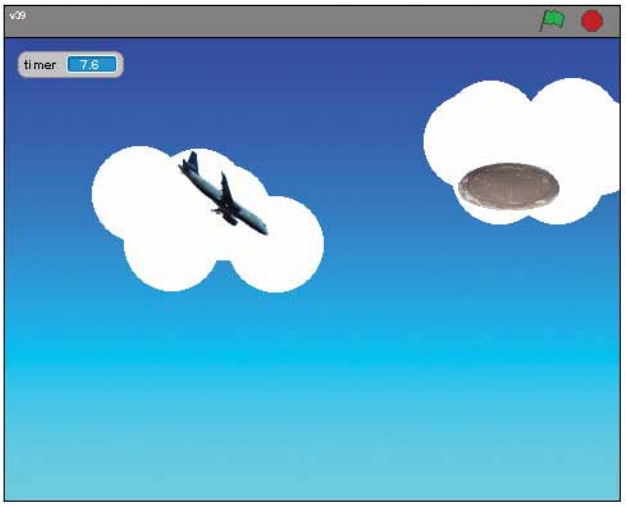

Figure 6

Here are a few tips:

- Be careful not to put if statements inside each other. While this is legal, it probably won't do what you want.
- For this exercise, use the Change X By and Change Y By commands. These are very powerful and will allow you to have complete control of your character.
- Have fun and experiment with other sensors and changes.
- Modify my code so you can make the sprite also go up and down.

Fly Away!

Game developers have two major types of motion. Sometimes (as in the last example) you directly control the X and Y position of a sprite. Other times, it's better to think about motion as speed and direction. Scratch makes it pretty easy to use this method as well. As an example, look at the following airplane game: **www.aharrisbooks.net/scratchTutorial/plane.html**.

In this game, you control an airplane. The Up arrow moves you forward, Down moves you backwards, and the Left and Right arrows turn the aircraft. (I know airplanes don't move backwards, but this is my game, and I'll make up my own reality, thank you very much.)

For the most part, this game is much like the other motion. Just think through how you will change the movement.

Oh, No! It's a UFO!

The airplane example is cool, but it's not really a game yet, because we don't have anything to smack into. What kind of self-respecting game doesn't have some sort of mayhem? Take a look at the next example to see some fun: **www.aharrisbooks.net/scratchTutorial/ufo.html**.

> It's fine to play games made by others, but it's much more fun to build your own.

Now we have a UFO (I actually used the manhole cover built into the Scratch library, but now it's a UFO. Because I said so, that's why.) The UFO has interesting behavior. Whenever I hit the UFO with the plane, the UFO resets (or moves to a new random spot on the screen). Here are a few tips for getting this behavior:

- First, build the UFO sprite. Make sure you've named your sprites. That becomes important when you start working with collisions. You can change the name of a sprite in the text box that appears near the top of the screen when the sprite is active.
- Add some code in the UFO sprite to check for collisions. Remember, you'll need to do this every frame, so you'll need a Forever loop.
- When the UFO collides with the plane, change the X and Y position of the UFO. (You can also play a sound effect if you want, of course.)
- Look in the Operators panel for a Pick Random command.
- You may want to review the XY-grid background to see what your random range should be.
- You can set the position in one command or separate X and Y. I think the separate commands are easier, because the random stuff will make the code very long.
- The message mechanism can be a good way to clean up your behaviors; consider using it.

We've Got All the Time in the World

In every game, there has to be some sort of goal and some sort of obstacle. The goal of the airplane game could be running into the UFOs, but what's the obstacle? Time is one of the easiest obstacles to manage. The next goal is to make a timer. Here's an example in Figure 5: **www.aharrisbooks.net/scratchTutorial/timer.html**

The timer is pretty easy to use. It's always been there, but you might not have seen it before. Look in the Sensing tab. There's a button called Timer with a little check box next to it. Click the check box, and you'll see the current number of seconds since the timer began. That's not a helpful thing by itself, but you also have a Reset Timer button. Think about how you can be sure the timer resets at the beginning of a game. (Remember, the game always starts right after you click on the green flag.)

Something needs to happen when the timer is finished. What you'll need is another condition. If you want the game to stop after 10 seconds, you'll need to compare the timer to the value 10. Look in the Operators tab and you'll see the > (greater than) operator from math class. See if you can figure out how to make a condition that triggers when the timer is greater than 10. You will probably want the game to be longer in real life, but it's much easier to test the game with a short time span at first. You can make it longer once you know it's working right.

There are a number of things you could do when the timer reaches 10, but

the easiest is to simply stop all the scripts. Look in the Controls panel for the appropriate button.

Just a Little More . . . Let's Add the Score!

Now we're getting close to creating a game. However, it would be nice to know how many UFOs we touched during the game. That's pretty easy to do, but we need some way to count how many touches have happened. This introduces a wonderful new concept called a variable. A variable is a special place in memory that holds some kind of value.

Look at the Variables tab and you'll see a simple set of controls, but there's a lot more there than meets the eye. Click on the Make a Variable button, and a little dialog pops up. We want to call this new variable Score, and it will be available to all sprites. When you're finished creating the variable, a bunch of new command buttons appear in the Variables tab.

Use the check box next to the Score button to determine if the score is displayed on the screen.

You can now add code to your program to reset the score to 0 when the game begins and change the score by 1 (or a thousand—video games practically invented grade inflation) every time you hit a UFO. Your final game ought to look like Figure 6. You can play it here: **www.aharrisbooks.net/scratchTutorial/ufoHunter.html**.

But Wait . . . There's More!

You have now learned how to a game. But I've walked you through *my* game. It's time for you to make your own. You should have all the basic features you need. Here are a few ways to make it your own:

- Add your own graphics and sound effects, and change the theme.
- Give the user's sprite more realistic motion.
- Make the target move around on the screen somehow.
- Add new things to get in the way. Maybe you lose a life when you hit one, and the game ends when you run out of lives.
- Build an entirely new game based on these ideas and your imagination.

Okay, Programmers, I'll Talk to You Now

If you're already a computer programmer, you probably can see what's going on here. Scratch is a very clever environment that helps us teach all the main ideas of programming. With the exercises outlined in this article, I've introduced the following:

- Sequential programming
- Conditions and loops
- Event-driven programming (through messages and sensing)
- Simple functions (through-message passing)
- Object-oriented design (each sprite is an instance of a sprite class)
- Variables

The examples I've shown are still quite simple, but Scratch actually supports quite sophisticated programming paradigms. You might want to experiment with some of these things:

- Add your own DX and DY variables to add a basic physics model.
- Add gravity (objects slowly fall down when released).
- Add a "jet pack" behavior that provides upward thrust on a key press.
- Use lists to keep track of multiple objects.
- Create an RPG game with characters, monsters, and interactions.
- Build an orbital physics simulation.

Really, we can do all of these things. Scratch's best feature is that it allows us to introduce extremely sophisticated ideas in math, science, and computer programming in a way that's fun for kids and adults to learn together. Build some great games, and let me know what you've learned! Send me your best games and I'll post them on my website for others to enjoy!

Andy Harris is a homeschool dad, father of four great kids, and husband to the greatest homeschool teacher ever. He has taught all ages of students, from kindergarten to university level. Andy is the author of a number of well-known books, including **HTML/XHTML/CSS: All in One for Dummies**, **Game Programming—The L Line, PHP6/MySQL Programming for the Absolute Beginner**, *and* **Beginning Flash Game Programming for Dummies**. *For more information about his books, to see where he is speaking next, or to just say hi, please stop by his website:* ***www.aharrisbooks.net***.