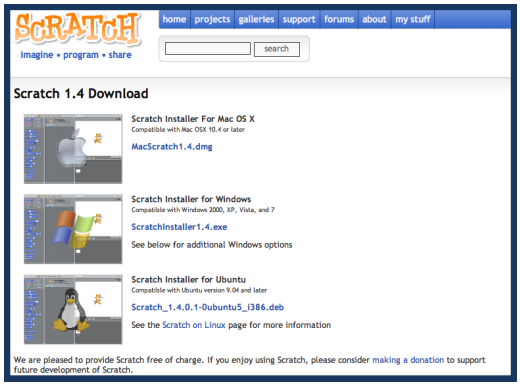
SCRATCH

Step One: Installing Scratch

To get started me must first download Scratch from the internet. (if you already have the program installed on your computer, you can skip to Step Two)

Using your favorite web browser, go to scratch.mit.edu were you will see the download Scratch button.

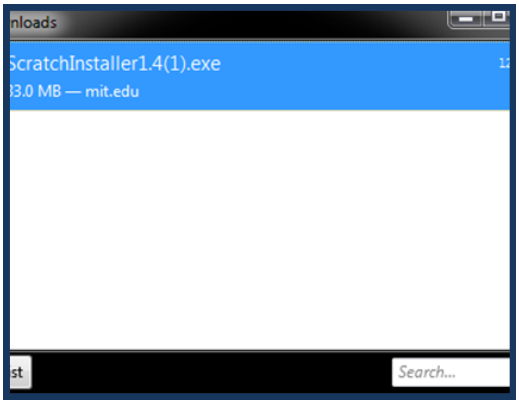
Then proceed to select the download for your specific operating system. If you are using PC, click the PC download. If you care suing Mac, click the mac download.



PC Download

This shows the next window which will open called “Downloads” and you can watch the progress of your file downloading.

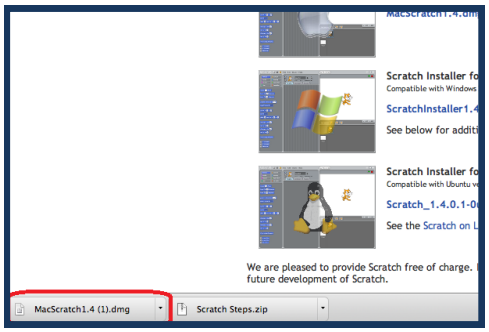
Next you want to install Scratch by double clicking on the ScratchInstaller1.4.exe file. You may be asked at this point if you want to allow this program to make changes to your computer. Click the Yes button.



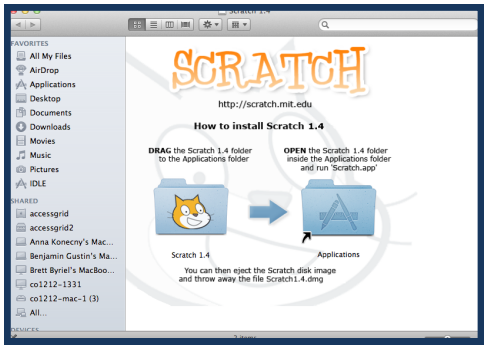
By clicking the yes button you will open the Scratch setup wizard (Figure 3). The wizard makes the install very simple. You can choose to let the program install in the default location on your hard drive or a different location of your choosing. Once you decide where you would like Scratch installed at click the “Install” button.

Mac Download

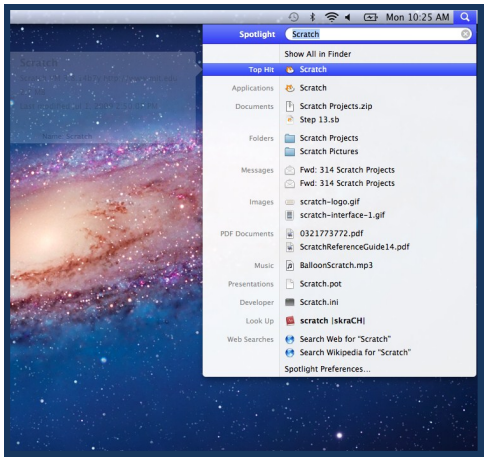
Click on Scrath1.4.dmg you have just downloaded. This should be located somewhere visibly or in the downloads folder for your web browser.



Doing so will bring up this window on your screen. It should look something like the these images. You might need to minimize your web browser to see it. You should click on the scratch icon and drag it to the applications folder as shown.

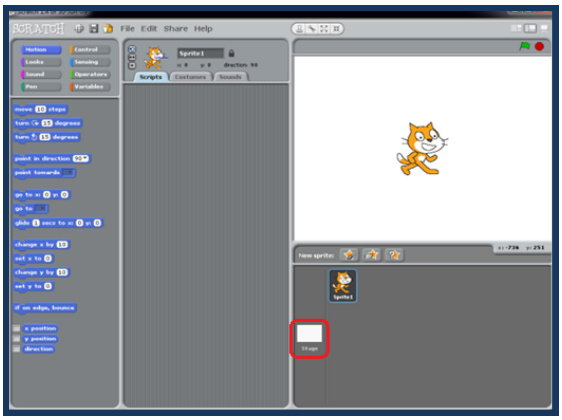


Congrats. You have successfully installed Scratch! You can now successfully run the scratch program. You can find this in your applications folder or by hitting (⌘ +SPACE) and typing in scratch like shown.



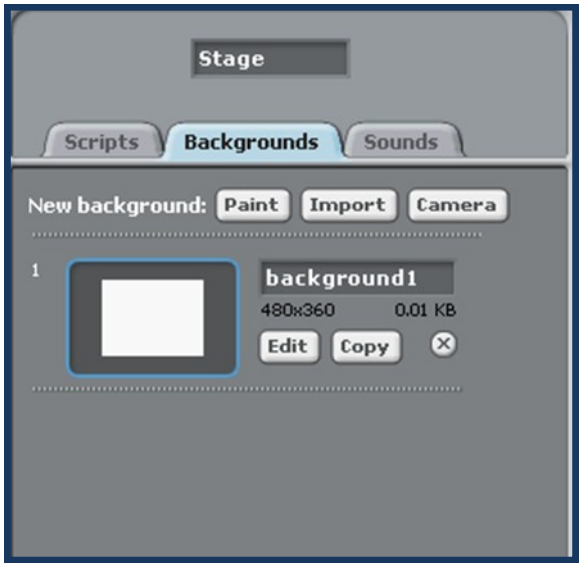
Creating a Background

To create our background we must first click on the small white box near the bottom right of the screen that is labeled “Stage”.



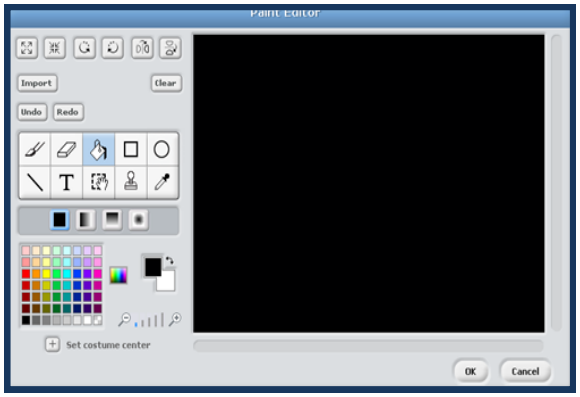
This shows the center section once “Stage” has been clicked. Next we want to edit this background, so click the “Edit” button.

This will open a very basic and simple editing window. The tools in this editor are similar to that of the Paint program we are all familiar with.

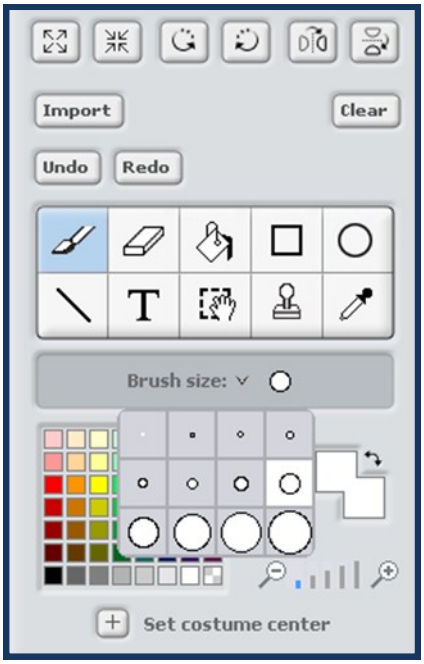


Now we want to make a background that looks like we have many bright stars on a dark night. To achieve this, first we want to make the entire background black.

The default color is already set to black; all we have to do is click the fill tool icon that resembles a spilling paint bucket and then click anywhere in the editing area.

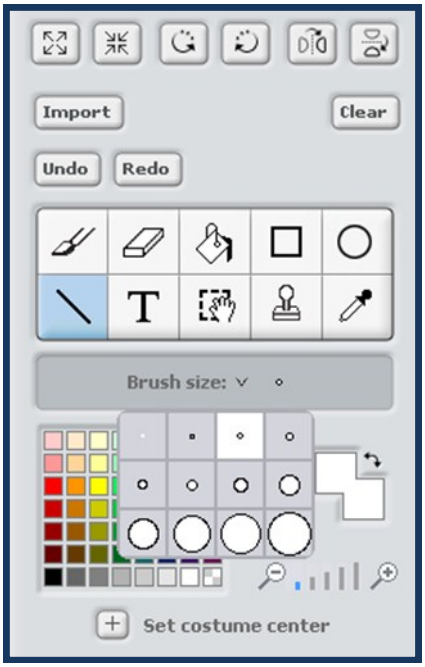


Our starts will consist of a small white dot with two thin lines crossing one another overtop the white dot. First make one white dot by clicking on the paintbrush tool icon, and then select the white color in the color chart that is second in from the bottom right. Next we need to choose the size of the dot by clicking on the shaded button labeled Brush size near the left center of the screen. Click on the middle right hand circle as show in Figure 8.

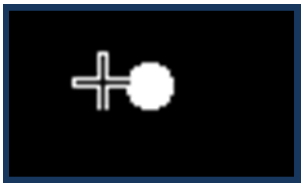


Once you have the small white dot ready, we can start making our stars. Proceed by clicking and release once anywhere in the edit area to make a single white dot. We only need to make one for now because we can copy it for the rest of the stars.

Next we need to make the white lines on our dot to make the stars appear as if they are gleaming. Click on the Line tool icon and then again change the brush size to the circle in the second from the right on the top row as shown in Figure 9. Once you have the line tool ready we can make our star glimmer. When you move

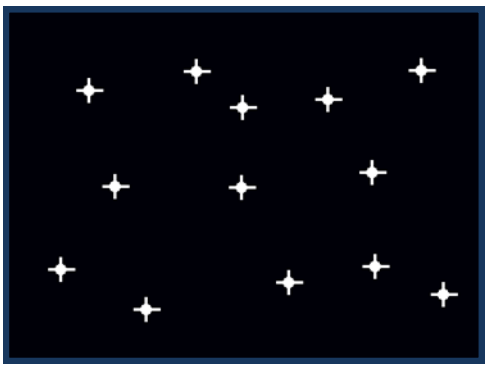


Once you have the line tool ready we can make our star glimmer. When you move your mouse over to the editing area, your curser will change into a plus shape. Like in Figure 10, place the right side of your curser so that it just touches the left side of your dot. Create a small line by clicking and holding your left mouse button then drag you curser to the right so that the left of your curser just touches the right side of the circle. If done correctly your star will look similar to that in Figure 11.



To create the second line of each star will be the same procedure except we will start with our curser on top of the dot and draw down. Figure 12 show the completed star.

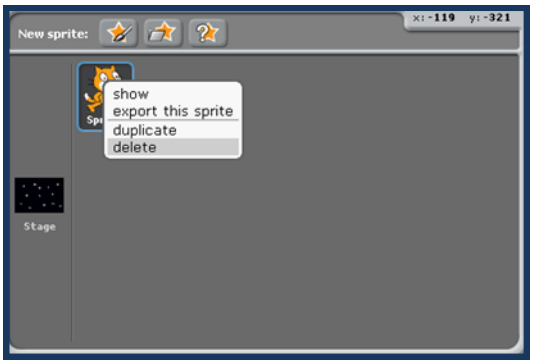
To copy this start all we have to do is select the stamp tool icon on the left which allows up to make a box around our start. The area that you select around your start should be as close to the star as possible without touching it. Once selected you will have a stamp that you can click anywhere on the background and create another star. Do this about 12 to 14 more times and evenly space apart each star



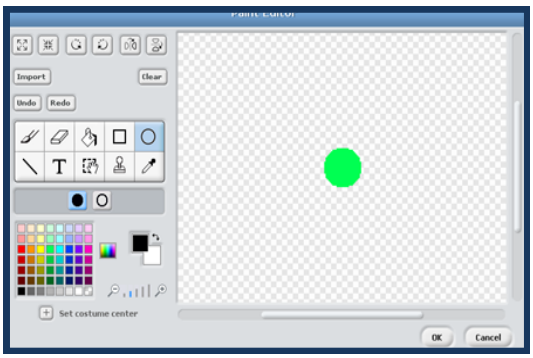
Creating your Sprites

Sprites are arguably the most important part of anyone's project in Scratch. Sprites are the objects that you assign code to. In our game we will use 3 different types of sprites that will all behave differently

Scratch inserts a default sprite that looks like a walking cat, but because we will not use this sprite in our game we must delete it. As shown in figure 14, to delete a sprite you must right click on it thumbnail, in the bottom right of the window, and

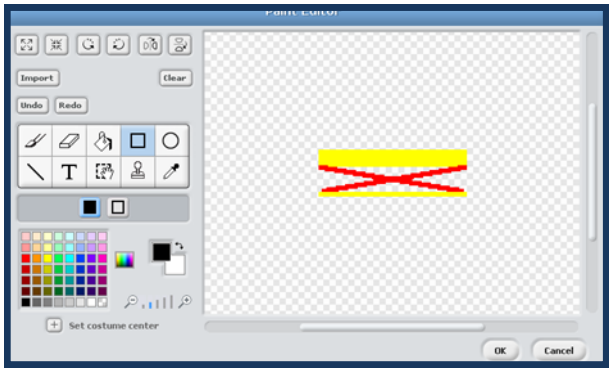


Now we can begin making our own sprites! First sprit we make will be a simple ball. To create a sprite we must click the paint new sprite icon that is right above where the sprite thumbnails are located. This brings you to the paint tool once again. Now that you are familiar with the paint tool in Scratch, creating our ball is as simple as selecting a color and creating a small circle with the ellipse tool as shown.



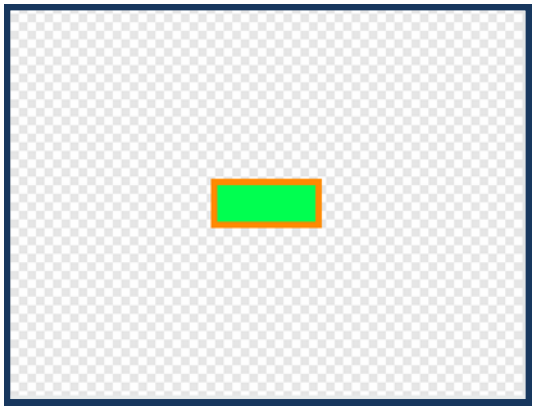
Once you are satisfied with your ball and click ok, then the ball you have created will now appear in the game area of the main screen. Then we can create the next sprite in the same way, but this time we will create a paddle type of figure.

Try to create the sprite shown if figure 16 using the paint tool. The paddle must be a rectangular shape, so using the rectangle tool is very helpful. Remember that the paddle must be larger than the ball you created.



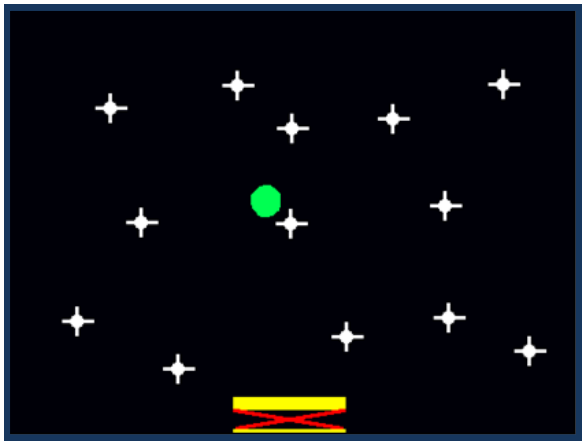
Once you finish with the paddle you must drag the paddle to the bottom of the screen as shown.

Now that you understand how the paint tool works, creating the blocks will be quite simple. The blocks are nothing more than a multicolored rectangle. The fill color of the blocks will be green and the outline of the blocks will be orange. Figure 18 show how the completed blocks should look.



Now that you have finished designing your sprites, then your screen should look similar to ours in Figure 19.

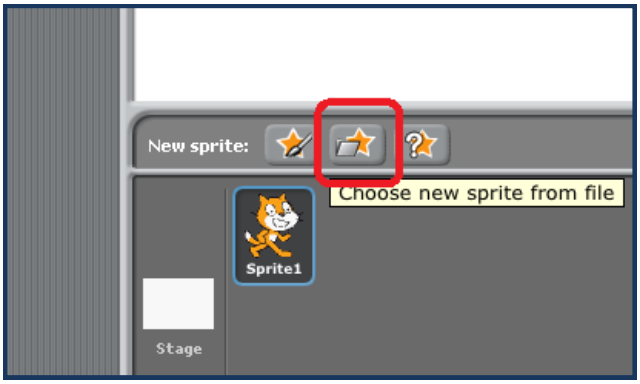
This is the general look of our game! Next you will be creating the code for each sprite, which makes them interact with each other the user.



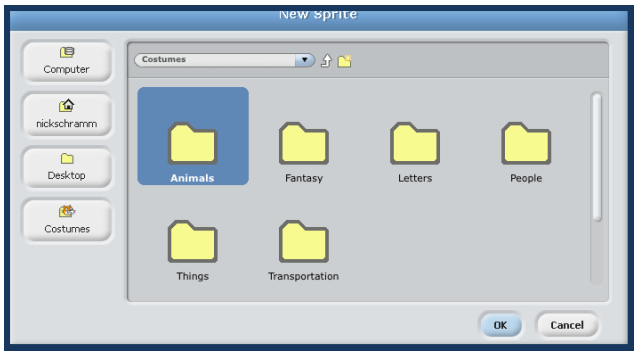
Using Existing Sprites

Drawing sprites are hard. It is hard to get a good image from the simple paint options that Scratch gives. We are going to explore the possibilities with using Sprite files that already exist.

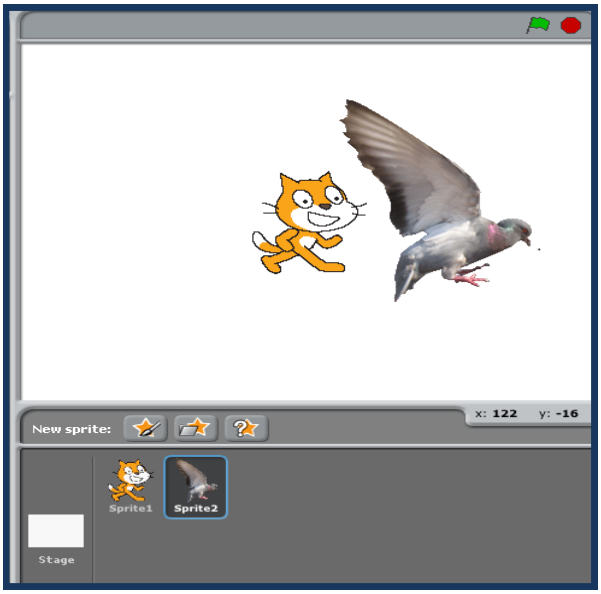
The first step in creating a sprite with a existing image is to click “Choose new sprite from file” button. It is located as shown.



This will open the file explorer. This can be used to select any sprite or image you can think of. For this short example we use a bird. Double click on Animals and then select the bird image. Next Press Ok.

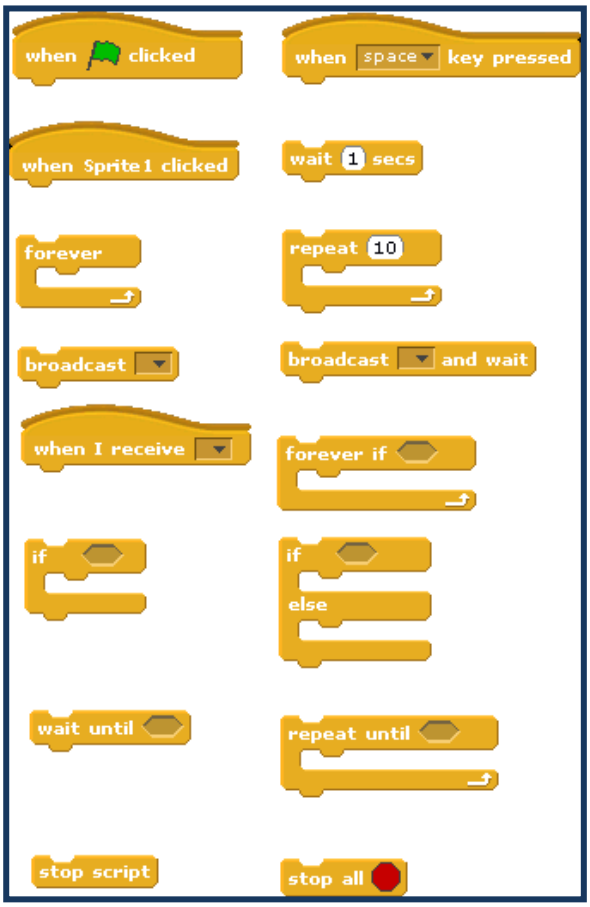


This selection shows the creation of your new sprite, a pretty (or ugly) bird. At the bottom an icon for a new sprite has been added as well as the bird sprite has been added to the stage..



Control Blocks

We've discussed some of the basic control operations in Scratch, but those have mostly dealt with beginning your program or some commands. All of these control operations you may have noticed have a wave along the top indicating no commands precede them. Now let's discuss the other control operations. These operations determine when your program will execute certain commands. One of the common ones we will use are the if-statement blocks and the if-else-statement blocks. In this case the if statement will execute the statements inside it's block if the condition in the hexagon is met. The hexagon, like other functions in Scratch can be filled with statements that are shaped hexagonally, such as the and, or, and not operations in operators, which we will discuss later.



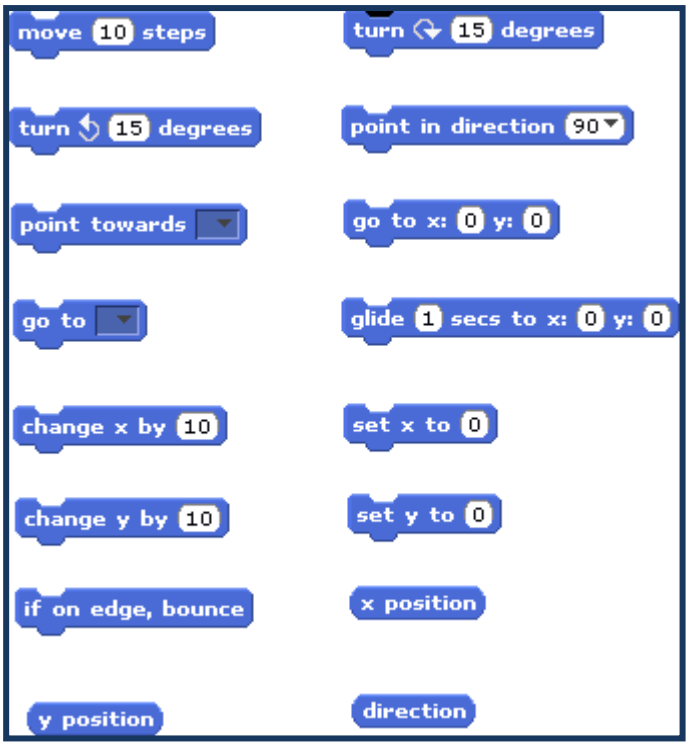
The if-else-statement will behave in a similar way except that if the condition in the hexagon is not met Scratch will skip over the first set of code and move down and execute the code under the else block. Keep in mind that one of the two blocks will be executed. The "forever" block runs indefinitely and the "forever if" blocks will run the block of code placed inside the brackets until the condition is met, and in higher level languages these are commonly referred to as loops. While we don't use them extensively in this tutorial loops are extremely powerful tools in Scratch and most other programming languages. The other commands involve broadcasting, waiting, and terminating statements. Terminating statements speak for themselves; they will stop the program from running causing the user to need to start over. Broadcasting and waiting are not discussed in this tutorial.

Now that we've discussed some of the key control functions lets discuss the motion functions. As we've already shown you can use the move function to cause a sprite to move on the screen, however there are some other useful functions included in the motion section.

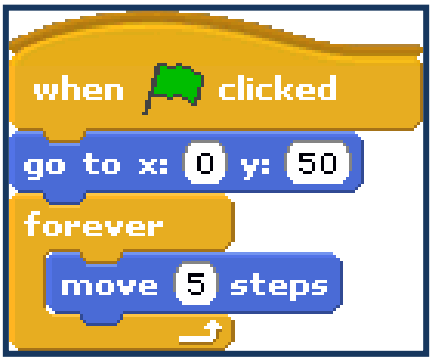
Motion

Motion blocks deal exclusively with moving a sprite or changing the direction of the sprite. Most of these functions speak for themselves and if elaboration is needed it will be in the index in the back, however there is one thing worth noting in Scratch. Direction is handled in an atypical fashion. If a sprite is moving strait up it's direction is 0 and if it's moving exactly to the right it's angle is 90 so any sprite moving up and to the right will have a direction from 0 to 90. If it's moving down and to the right it's direction will be between 90 and 180. If the sprite is moving up and to the left it will between 0 and

-90 and anything moving down to the left will be between -90 and -180.

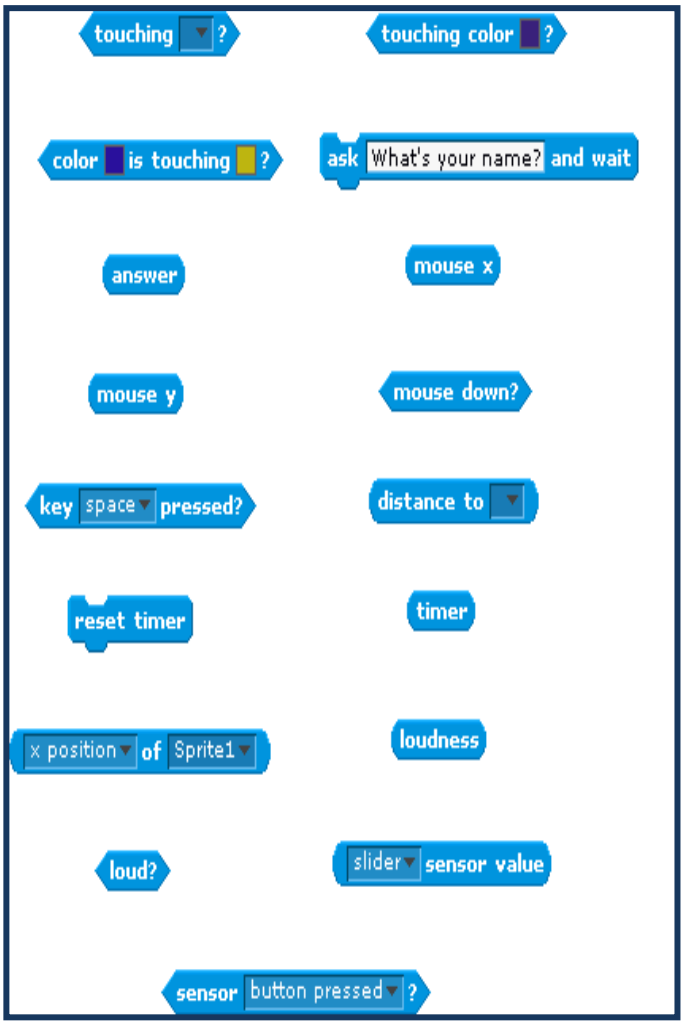


To begin, lets add a go to 0 and y to 50 to the code blocks for the ball sprite so that each time the program is started the ball is reset to the middle of the screen. Add this code block underneath the flag function so that each time the flag is pressed the first thing it does is reset the ball.



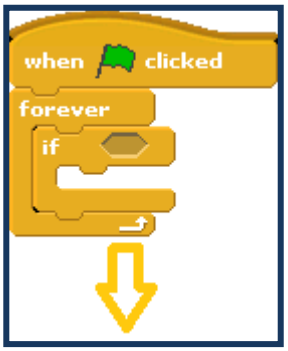
Sensing

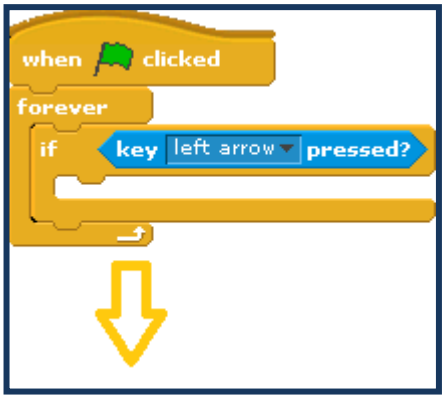
Before we proceed further we'll talk about the sensing functions. Sensing and motion commands work hand in hand in “Breakout” and many practical programs because it allows for the condition of the sprite to change when it senses a particular event has occurred. In Breakout we’ll be using these blocks to sense when the ball hits another sprite or object. In some cases it allows for the condition of the sprite to change when user input dictates it. Using these we can allow the player to take control of the paddle at the bottom of the screen to bounce the ball.

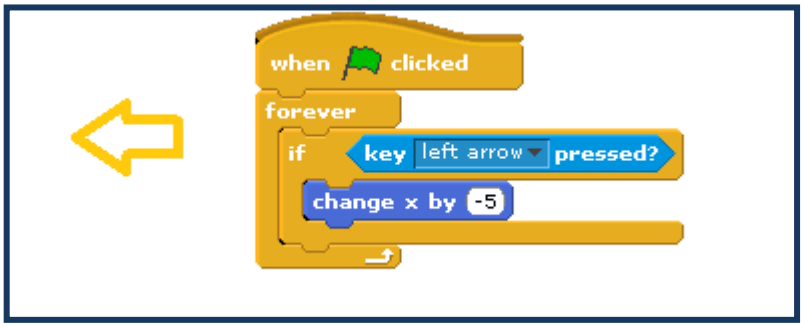


The Paddle

That's why now we're going to set all the code for the paddle in one go. To begin click on your paddle sprite in the bottom left and attach a “forever” block to the flag block. Now place an “if” command block inside the “forever” block. Notice how there's an octagonal blank spot on the “if” statement. As described earlier this is the condition to be met before the blocks inside the “if” block are executed. Now, lets click on the sensing button on the left. We're interested in using the “key pressed?” sensing block. Notice again the hexagonal shape of this block and drag it into the hexagonal blank spot on the “if” block. When the blank spot light up on the edges release you mouse to place it as your if condition. Now, place a “change x by” block from the motion category inside the “if” block and set the value to -5. You may have noticed the drop down menu inside the “key pressed?” block. This allows you to choose a button on a standard keyboard to satisfy the condition. For our exercise choose the left arrow key from the drop down menu.
You've just programmed your first if statement! To complete the code for the paddle create a second if statement and place it directly under the first, but still within the forever statement. Use the same sensing block (“key pressed?”) and this time choose right arrow from the drop down menu. Place a “change x by” block inside the if statement and change the value to 5. The paddle is now ready and coded for Breakout to try it out press the flag and use the arrow keys to move it left and right.







The Ball

You may have noticed that although you can move the paddle; now the ball reacts to neither the paddle or the edges. Lets fix that. Start by reselecting the ball sprite in the bottom right. Click the motion button and look for the block labeled “if on edge, bounce” and place this under your “move” block. Notice now that if you run the program when the ball hits any edge it automatically bounces off.

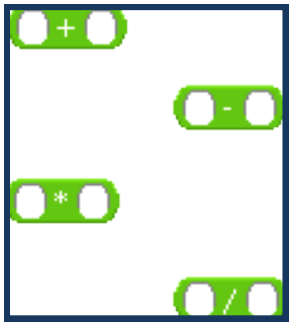
Place an “if” block underneath your new “if on edge, bounce” block. To make the ball respond to the paddle we'll go back to the sensing block and choose a new one called “touching?” and place it in the hexagonal blank spot on the “if” block. Then choose the paddle from the drop down menu.

Finally, pick a “turn degrees” block from the motion section and place it within the “if” statement and set the value of the “turn degrees” block to 50. Notice that now when the ball makes contact with the paddle it bounces off! The current code is a bit unrefined and we will be coming back to clean it up later (so the ball bouncing looks more believable) once we've discussed some of the other block types.

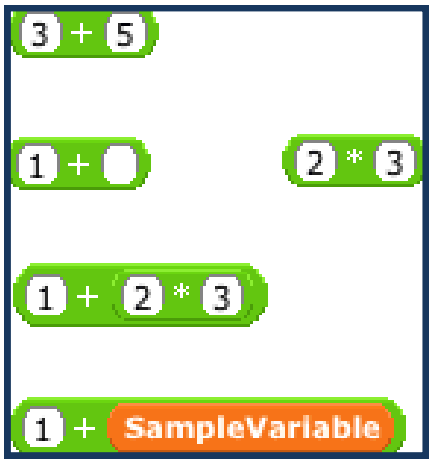
Operators

Value Operators

These operators are known for their circular or oval shape. These are the basic value operators. They are addition, subtraction, multiplication and division and look something like this.



The biggest thing is these operators correspond to a value. They can fit in anywhere there are oval shapes. Since each operator takes circle shapes themselves, this means they can be stacked. Shown in the examples are several combinations. Notice we have an that we have an addition of the result of a multiplication. Another note that variables can be used as one of these value operators.



Advanced Value Operators

- 

A value between the the two values passed to it.
- 

This is a String operation. Combines the two strings.
- 

Finds the letter of a String at a certain position.
- 

Finds the remainder when divided by a number. 12 mod 5 = 2
- 

Round- Rounds down to the nearest number.
- 

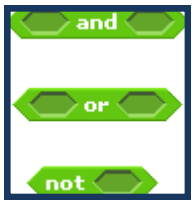
This can be used to do several advanced math functions.

Value Operators

Boolean operators are true/false values. These are important because they are used program control. Now there are two kinds of boolean operators. Boolean operators using values and boolean operators using other boolean operators.

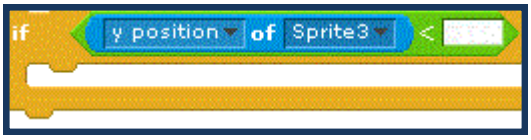
Boolean operators using values are greater than, less than and equal to. These operators take a value and turn it into a boolean.

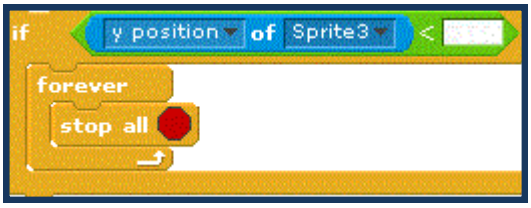
Boolean operators using other boolean operators are AND, OR, NOT. These operators take existing boolean value and return a boolean value. As seen these can be stacked within each other.



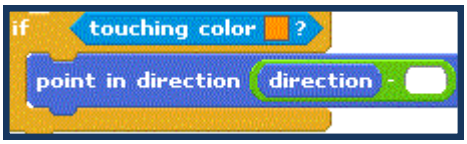
Now we are going to add a game over condition. What we want to stop the game when the ball touches the bottom edge.

We are going to accomplish this by comparing the position of the ball with the position that corresponds to the bottom edge. To do this we have an if statement with a less than operator. In the left side we insert the block with the y position of Sprite 3 (The Ball). Then inside the if statement we put at forever loop with a stop all inside it.





Then below that if statement, we are going to insert another if block. This block will contain a point in direction then the operator direction—55. We chose the orange color because orange is the outline of all the blocks. If we want to check to see if the ball touches the blocks, we must see if it touches orange. Then to similar that it bounces, we point the ball in whatever direction it was facing minus 55.



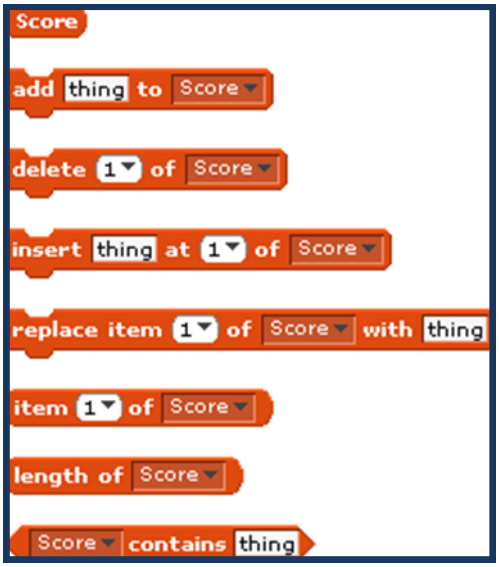
Variables

The variable section contains very few blocks compared to the other sections, but that doesn't mean that variable aren't important. In fact variable are arguably one of the most important sections. When you navigate to the variables tab you may notice there are no blocks. Well, this is because you have to create a variable first. Let's try making the "Score" variable by clicking "make a variable." Notice that now the blocks you see to the right are available to you. In this case the "Score" block will return the current value of the variable this is important because a value can be changed and saved for use later on in the program.

The "set to" block manually changes the value of the variable to the input value and the "change by" block increments or decrements by the value in the blank (to decrement use negative numbers). The "show variable" and "hide variable" blocks are used to show or hide the banner on screen that displays the value of the variable.

List

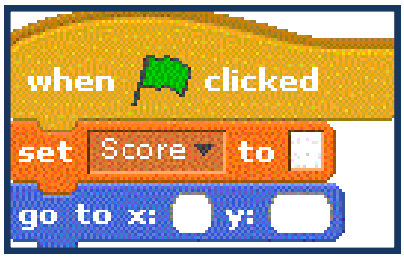
You may have notice a button directly under "create variable" called "create list." We don't use any lists in breakout, but they are an important tool so we will briefly describe their purpose. A list can store several variables inside of it. For instance in terms of Breakout we could maintain a list of type "player" in which we could monitor each players' score. At the press of a particular button we could switch players by using the "item of" block. Say for instance player one is Billy and player two is Jill. First we add Billy and Jill to the list using the "add to" block. If Billy is going to play we could use a sensing block to make it so that if he presses the "B" key on the keyboard then the list chooses item 1 from the list, however if Jill presses "j" then we get item 2 from the list. Lists have many powerful functionality in Scratch and other programming languages.



Adding Score

Now we are going to add a score to our game. First we what are going to need to do is create are variable called Score. This is done in the variables tab.

Now under ball tab we are going to initialize the score to zero. This must be done so the score does not accumulate from pervious game. What we do is simple insert set Score block and type in 0 for a value.

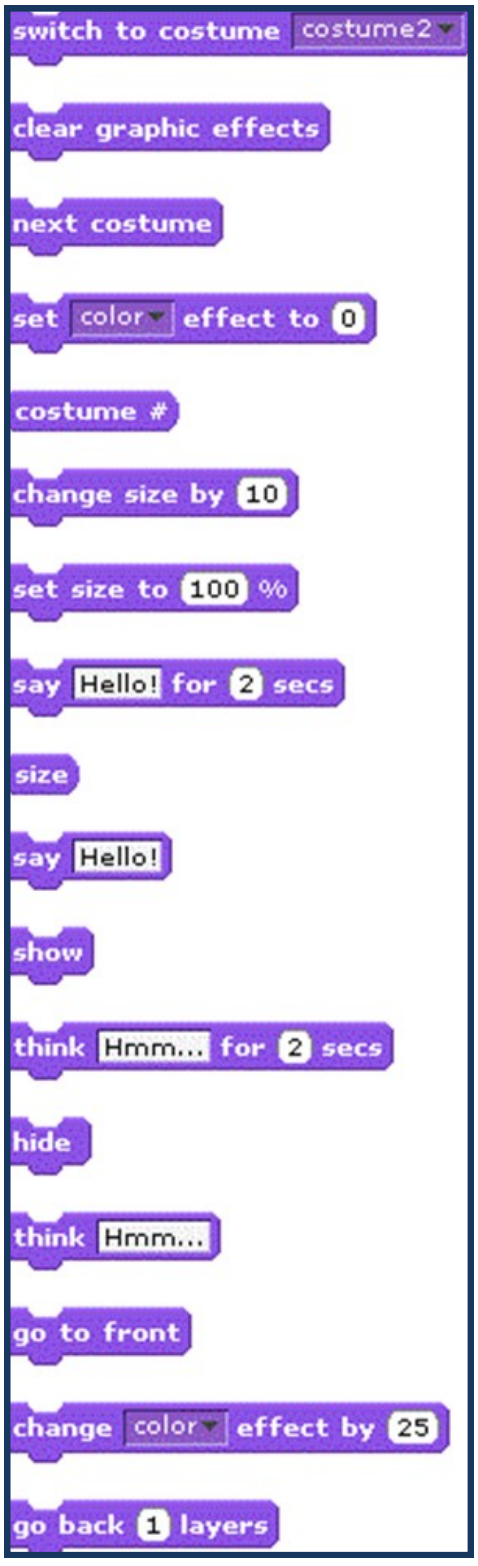


Sound

The sound tab has a few less functions than some of the other categories. The main block is the “play sound” block which is usually then edited through use of the other blocks. Anything from the instrument playing the sound, to the volume, to the tempo your sound is playing at can be edited. Keep in mind if the same sprite is going to be producing more than one sound that a “stop all sounds” block will have to be used and a new “play sound” block put in place.

Looks

We're about to get our first experience with the "Looks" blocks so let's run through them quick. Looking to the left it comes as no surprise that many of the blocks…change the look of the sprite. Some of these blocks are more obvious like allowing you switch costumes (a concept we'll discuss at the end of the tutorial) to altering the size of the sprite. Some that may seem a bit more complicated are the "say" and "think" blocks, which cause the sprite to express what's typed in the blank as text in the program. In addition the "set effect to" block allows you to alter a sprite in any number of ways, and all these modifications can be cleared with the "clear graphic effects" block. Two blocks we'll use a lot for Breakout are "show" and "hide," which cause the sprite to disappear or reappear.

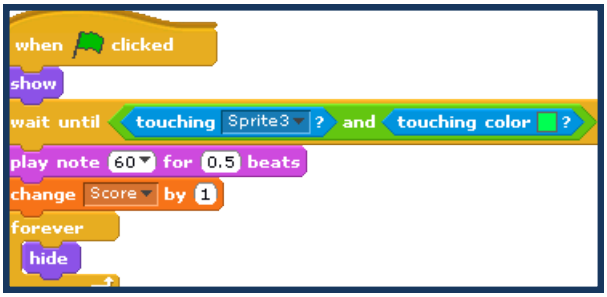


The Blocks

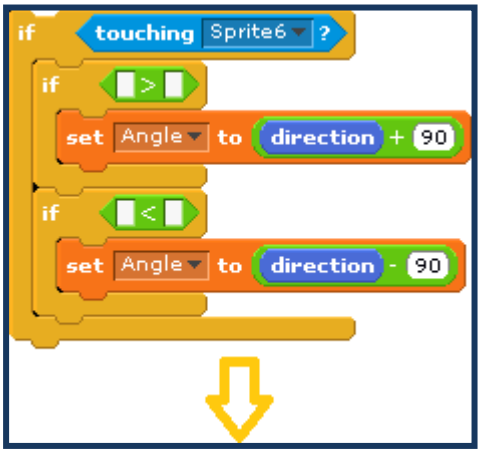
Now we are going to insert the code for the block sprites. The first thing is an initialization page. Create a When State Clicked block and then add a purple show block like is shown here.

Next we are going to insert code for when the ball collides with the block. We want to create a wait until block with the conditions show. Then we are going to add a Change Score block with a 1 for the score. Then this will be followed by a forever block with hide in it.

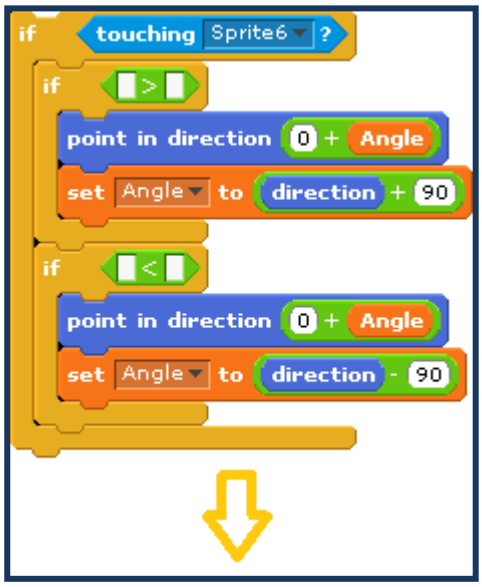
Next likes spice up our game with a bit of sounds. Create a play note block from the sound tab. For values we picked 60 and 0.5, but use whatever suits your taste.



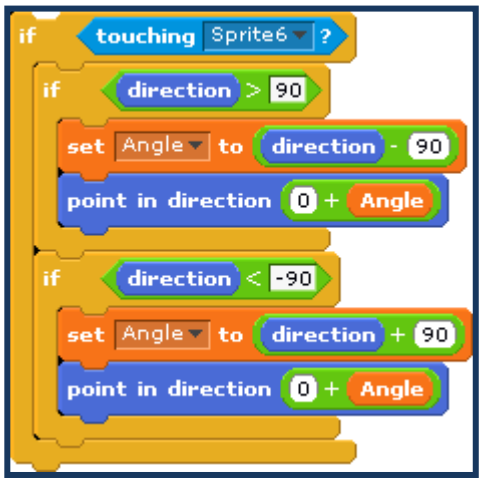
Notice that the areas you can type numbers on the plus and minus operators also have curved edges, which means they can also be filled by variables. Go to the motion tab and towards the bottom select the “direction” block. Scratch knows that because you are selecting the ball sprite that this is the direction of the ball sprite. Place the direction block in the first blank on both the minus and plus statements you placed earlier. For the second blank manually type in 90 for both the plus and the minus blocks. Note that these blocks will save the value of the balls current direction plus or minus 90 depending on the current position of the ball (the current position will be dealt with later).



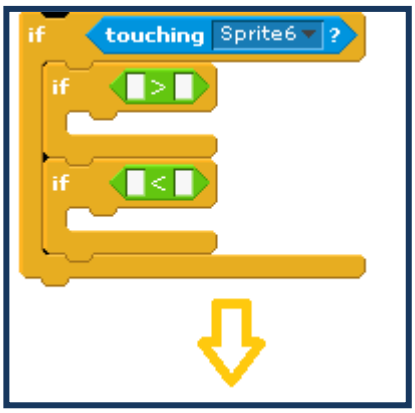
Navigate back to the motion tab and select the “point in direction” block and place one of these blocks under each “set to” block you placed earlier. Similar to the “set to” blocks earlier we will fill the blank spaces in the “point in direction” block with an operator. In this case fill the blanks in both “point in direction” blocks with a plus “+” operator. On both plus “+” blocks set the first blank to 0 manually. For the second blank on both go to the variable tab and select the “Angle” block (that is there because you created the Angle variable) and place one of these blocks in the second blank on both of the plus



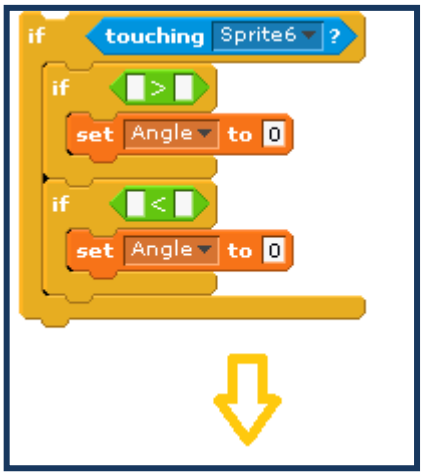
To finish the code for the bouncing manually enter 90 into the second blank of the greater than “>” block and enter -90 into the second blank of the “<” block. Go to the motion section and grab the “direction” block at the bottom and place that in the first blank of both the greater than “>” and less than “<” block. Now the code for the bouncing is complete feel free to try it out.



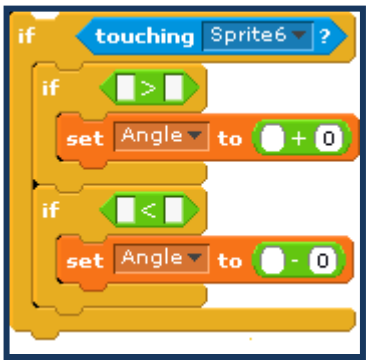
Now that most of the mechanics have been programmed for Breakout lets go back and refine the code for the ball bouncing off the paddle. To begin place two empty “if” blocks inside the “if” statement that has the “if touching” the paddle condition. In the top “if” statement place a greater than or “>” operator block in the condition blank. In the bottom “if” statement place a less than or “<” operator block in the condition



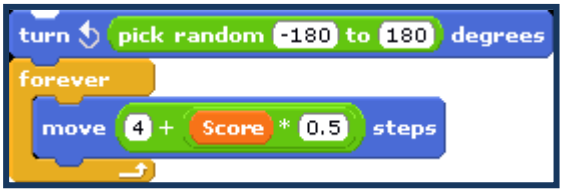
Now go to the variable section and select the “make a variable” button. Variable are great for tracking certain things like score, but just like in math they can be used to store values. Name your new variable “Angle” then drag a “set to” block into both the first and second “if” blocks you placed in the code for the ball sprite. Use the drop down menus on both “set to” blocks select your new Angle variable.



Navigate to the operator section again and select the minus “-” operator and move it into the area on your “set to” block that allows you to type in a value. Notice instead of typing in your own fixed number you can place other numbers maintained by the program instead (these are denoted by curved edges and behave similarly to the hexagonal blanks in “if” statements). Do the same thing for the “set to” block in the second “if” statement, but this time use a plus “+” operator.

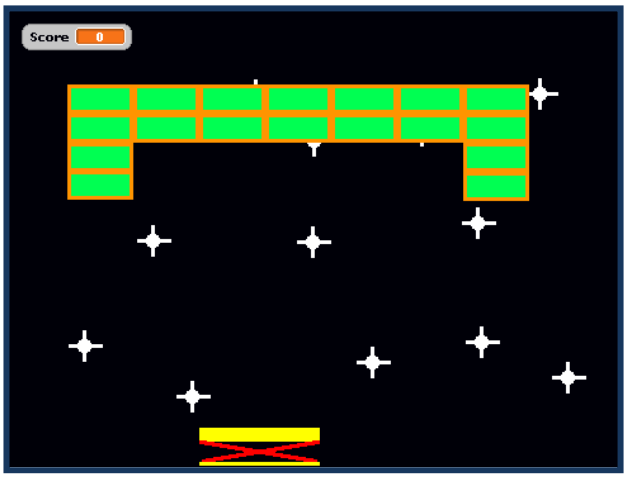


We're going to add two other quick refinements to our game. Before the "forever" block place a "turn" block and choose a "pick random" block from the operators section and put -180 in the first blank and 180 in the second blank so what direction the ball moves in at the beginning of the game is random. Then we'll modify the "move" block just inside the "forever" block so that as score increases the ball moves faster.



To accomplish this place a plus block "+" over the 5 inside the block now. Then place a multiply block "*" inside the second slot of the plus block "+". Set the first blank to 4 and place a "Score" block from the variable section inside the first blank of the multiply block "*". Now set the second blank in the multiply block "*" to the rate you want the speed of the ball to increase by each time the player removes a block. The recommended value is 0.2. The new block and modified block are shown on the right, however the "forever"

Now that the code for the ball is complete lets finish setting up the play field. Begin by right clicking one block on screen and duplicating it. Move this block where you would like and continue duplicating blocks until you have as many as you like. As an example we've provided a sample play field on the right. Be sure to check the starting position of the ball so that it doesn't spawn in the blocks.

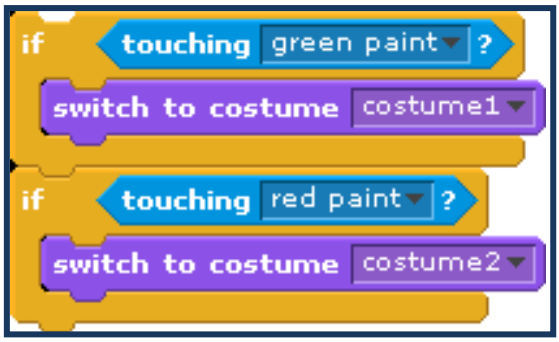


Once this is complete ensure you place a victory condition in any of your code. This can be accomplished with an "if" block with a condition dependent on score. Based on our example set the condition of your "if" block by placing an equals block "=" from the operator section in the condition and setting the first blank to the "Score" block in the variable section and setting the second blank to the number of blocks you have on the screen. Inside the "if" block make sure to place a "stop all" block and preceding that place a "say" block with something informing the player that they won the game. In our example this block was placed in the code with ball within the forever block.

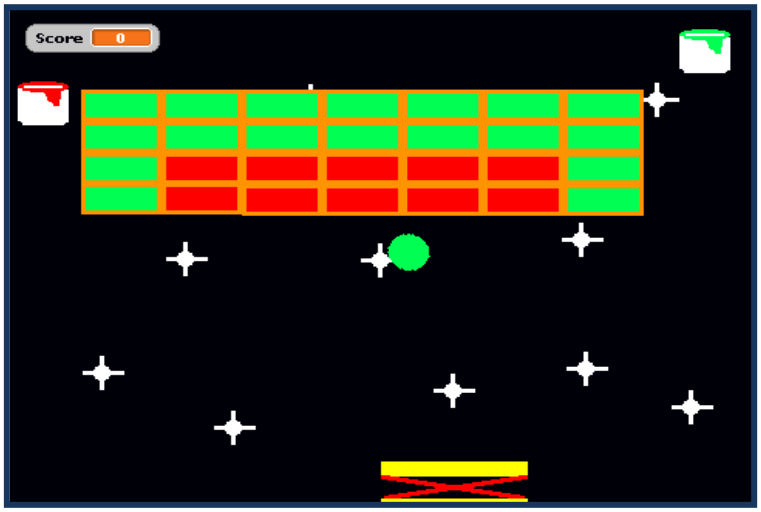
Extras

One other concept we'll introduce at the end of our tutorial is the use of costumes. To begin we'll design two sprites that will change the costume on the ball sprite. In our example we have two paint cans. One of these paint cans contains the original color (light green) that the ball starts as. The other paint bucket will be red. Now we'll select the ball sprite and select the costumes tab located above the code. You should see the default costume which we set as the light green ball. Click on the copy button next to that costume to create a second costume of the same shape. Now click the edit button on your new costume (which should default to the name costume 2) and use the paint bucket tool to fill the ball with red instead of light green.

Now we'll create a few pieces of extra code to handle this. Inside the “forever” block in the code for the ball sprite create two “if” blocks. The condition for the first one will be “if touching” the first paint can or other sprite you'd like to change the color of the ball. The condition for the second if block will be “if touching” the other color changing sprite. Now go to the looks tab in the top left and select the "switch to costume" block and place one inside each "if" block. If you want the ball to turn green when it hits the green paint can use the drop down menu to select costume 1 and for the red paint can choose costume 2. Now when the ball contacts the red paint can it changes to red, and when it hits the green paint can it switches back to green. We've also chosen to place a "switch to costume" block before the "forever" block to always reset the ball to green at the beginning.



Now that the ball turns red lets put it to use. Right click one of your bricks and duplicate it as you've done before, but now go to the costume tab and edit the default costume. Using the paint bucket tool fill the inside of the block with red. Then select the script tab again and switch the second condition in your "wait until" block to the color red instead of light green. Now duplicate your new red block 9 more times and fill the gap left by your green blocks to create one solid rectangle (look below). Now when the ball is red you can remove red blocks and when the ball is green you can remove green blocks. Be sure to also change the victory condition in the script section for the ball sprite so that victory is achieved when the score is equal to 28, because you now have 28 blocks.



Scientific Applications

Now that Breakout is completed you should have a solid understanding of many of the key component in Scratch. Now that you understand Scratch let's talk about Scratch's practical application in education. Scratch is a programming language, it's basic, however it's also very graphical and it's easy to see the result of your code. This makes Scratch a simple to learn programming language that is also fun and visually appealing making it a great way to introduce younger audiences to very complex programming concepts. In addition Scratch can be used to model scientific and mathematical concepts very easily. Perhaps while working through Breakout you started to formulate other ways certain functions could be used or you tweaked something, because you preferred the game to behave in a certain way. If that's the case, great! You are on your way to thinking about and programming your own content in Scratch. If not, we've provided some alternate examples to show what Scratch is capable of doing that perhaps you hadn't thought about yourself.

Chemical Reactions

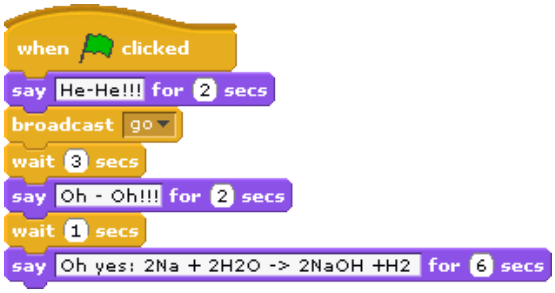
One great example shown left involves a chemical reaction. This chemical reaction involves the combination of Na2 and 2 H2O's. Each chemical is its own sprite and upon contact a chemical reaction occurs. After the reaction the byproducts are sent out and the resulting chemicals from the reaction remain. Coding this in Scratch is very simple and we'll take a quick look at and analyze the code.

H--H

Na+ O--H⁻

O--H⁻ Na+

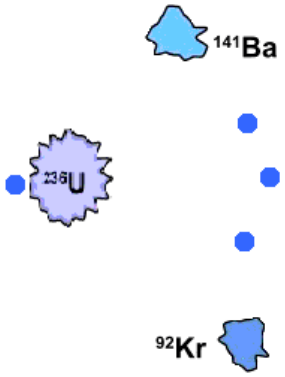
At first glance this code isn't anything you haven't seen before. A simple green flag to begin the code and a "say" and "wait" block. Then there's a block we haven't encoutered, "broadcast." Broadcast is named so because it can be viewed as, say, a radio signal. The programmer decides the signal, in this case "go," and sets other code to wait to begin until it "hears" the broadcast signal.



Now lets look at the code for the NaNa sprite. Notice that in this case the code does have a "when i receive" block that is waiting for "go" to be broadcasted. Once this occurs the code begins executing. The other code in this example is nothing new, you can see that that when the sprite comes into contact with sprite 3 it hides itself and then broadcasts a new signal to reveal other sprites and start other lines of code

Physics

The code for this first sprite is very simple. Essentially it states that when the flag is clicked to move to the right until it's touching sprite 2. The "wait" block is only being used as a means to slow the rate of speed. Once this sprite is touching sprite 2 this sprite hides itself and broadcasts "hit" to initiate the code for the other sprites.



The code for the uranium atom is also very simplistic. Once it receives the "hit" broadcast it changes costume to uranium 236, which then switches costume to what looks like an explosion. Following the explosion the sprite hides itself and broadcasts split, which causes the sprites showing the by-products of the breakdown of uranium 236.

Now, lets take a look at the code for Barium 141. All of the byproduct sprites share very similar code, but it's still reasonably complicated. Notice this code doesn't begin to run until the "split" signal is received. At this point the sprite shows itself and sets two previously created variables (u and v) to 15. Then the code enters the "repeat" statement, which as indicated will execute 140 times. Inside the block the sprite is moved to coordinates (u,v) and then u and v are both incremented by one. Essentially this code moves the sprite up to the right. Once it has finished moving the sprite is then hidden.

Mathematics

Scratch is also perfect for simulating mathematics. Say for instance you have one sprite at x position 60 and another at - 60. Perhaps when the green flag is pressed sprite 1 leaves sprite position -60 moving right at 5 steps per second while the sprite on the right leaves 5 seconds later and moves left at 7 steps per second. This problem sound familiar? Perhaps both of your sprites could be trains? Now that they're both moving we can create variable to track their x-positions and when they equal both can stop to show where they intersect. Or the program could run until either train hit the opposite side they started from to determine which train would reach the other train's station first.

Dece t
i

Other subjects

Outside of strictly technical fields Scratch can also be used to create short animations to help any student grasp a concept. English could have an animation outlining I before E except after C by showing the word “Deciet” (incorrectly) then making the I slide down and right while the E takes it’s place, and finally ending by moving the I up to make “Deceit.” Social Studies could animate any number of situations using Scratch.

Conclusion

Throughout this tutorial we’ve introduced you, trained you, and proven the usefulness of Scratch. We began with the basics: installing, creating your first project, background, and Sprites. Then, we moved on to the code: Controls, Motion, Sensing, Variables, Operators, Looks, and Sounds. As we learned these thing we constructed the game “Breakout” to show off what Scratch can accomplish. Finally, we showed how Scratch can be used in a wide variety of situations to assist in teaching and learning a concept. Scratch can be a powerful programming language, but like any other programming language you just need to know, and see, how it can be applied to multiple situations.