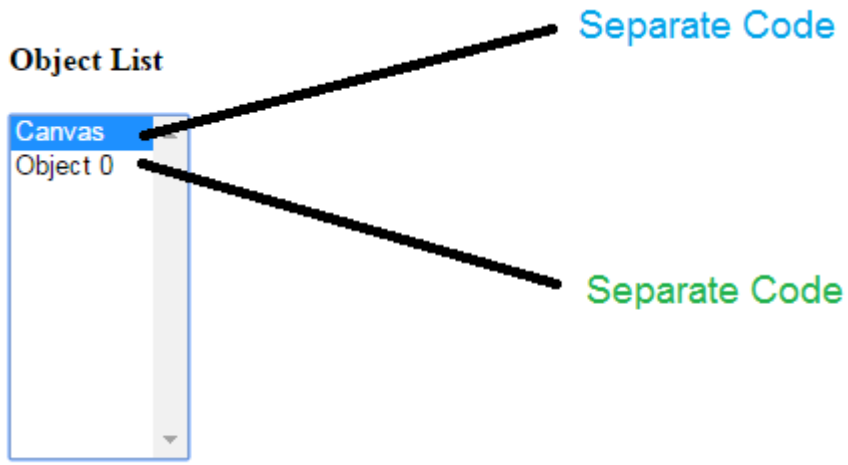
MAKING ART WITH MATH + CODE

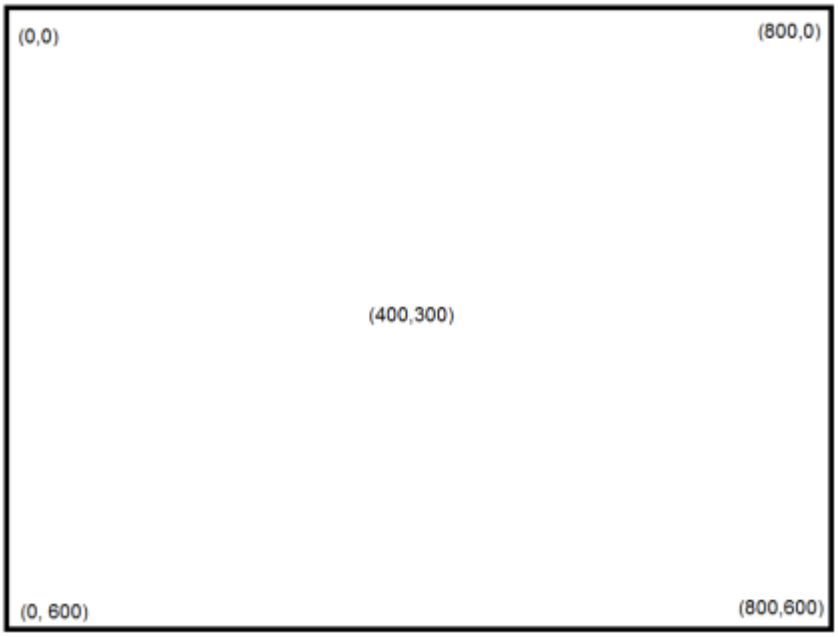
www.researchideas.ca/wmt/c6b6.html

Helpful Tips and Tricks

- Each 'object' has its own blockly code that is saved (and loaded) when you switch between objects



- The canvas is an 800x600 grid (length x width), with (0,0) in the top left and (800,600) in the bottom right. (400,300) is therefore the middle of the canvas



- The ‘add object’ button will add a point in a random location with a random colour

- The ‘create point’ blockly code will create a point at a given location, but with a random colour; you may set the colour manually

- When using the ‘create point’ blockly code, the new(est) point created will be selected after the code is run – it may appear your code has disappeared, but you are looking at the code of the new point (which is empty)

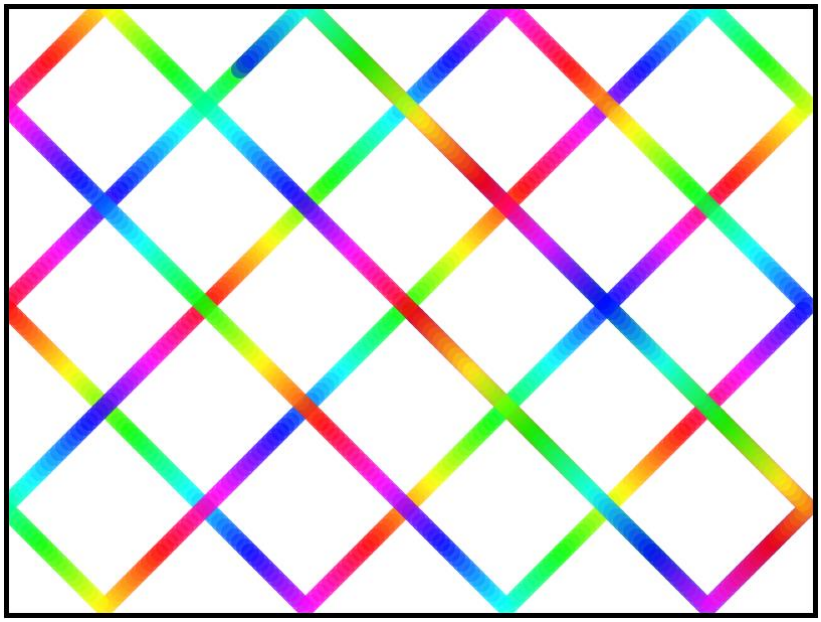
- Blocks that fall under the ‘Canvas Objects → Events’ code work on the currently selected object – if you are working the Canvas code, these blocks do not make sense unless you first create a block to apply it to

- Colour is determined by a Hue/Saturation/Lightness/Alpha selection. Hue ranges from 0-360, while the other attributes range from 0-100. Visit http://hslpicker.com/ for a simple hands on way to see how each attribute affects a colour

- Holding *alt* while hovering over a block will give you a bit more information about it

Creating Points – Basics

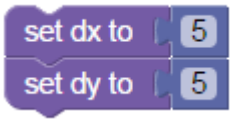
- In this section, we'll learn the basics of creating points and adjusting their properties. After this section, we'll end up with a canvas that looks something like this:



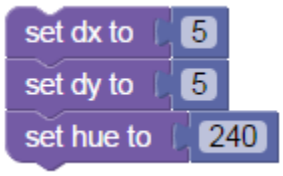
- Let's start by creating a point in the middle – drag the 'create point' block from the *'Canvas Objects'* category onto the blockly work area, and click 'Run Code', which executes any code in the current blockly area

- The blockly area will empty, but a point will form on the canvas. Whenever a new point is created, the object list will select the new point which will contain no blockly code

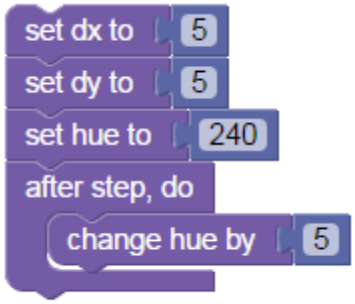
- Let's give our point some motion! Under the '*Canvas Objects*' category, select the '*Motion*' subcategory. Here we can set the x and y values of our point, but if we want our point to continuously move, we want to give it a change in its x or y value every step – the dx and dy blocks do just that. Try setting the dx and dy values using the respective blocks in the *Motion* subcategory, then press Run Code again



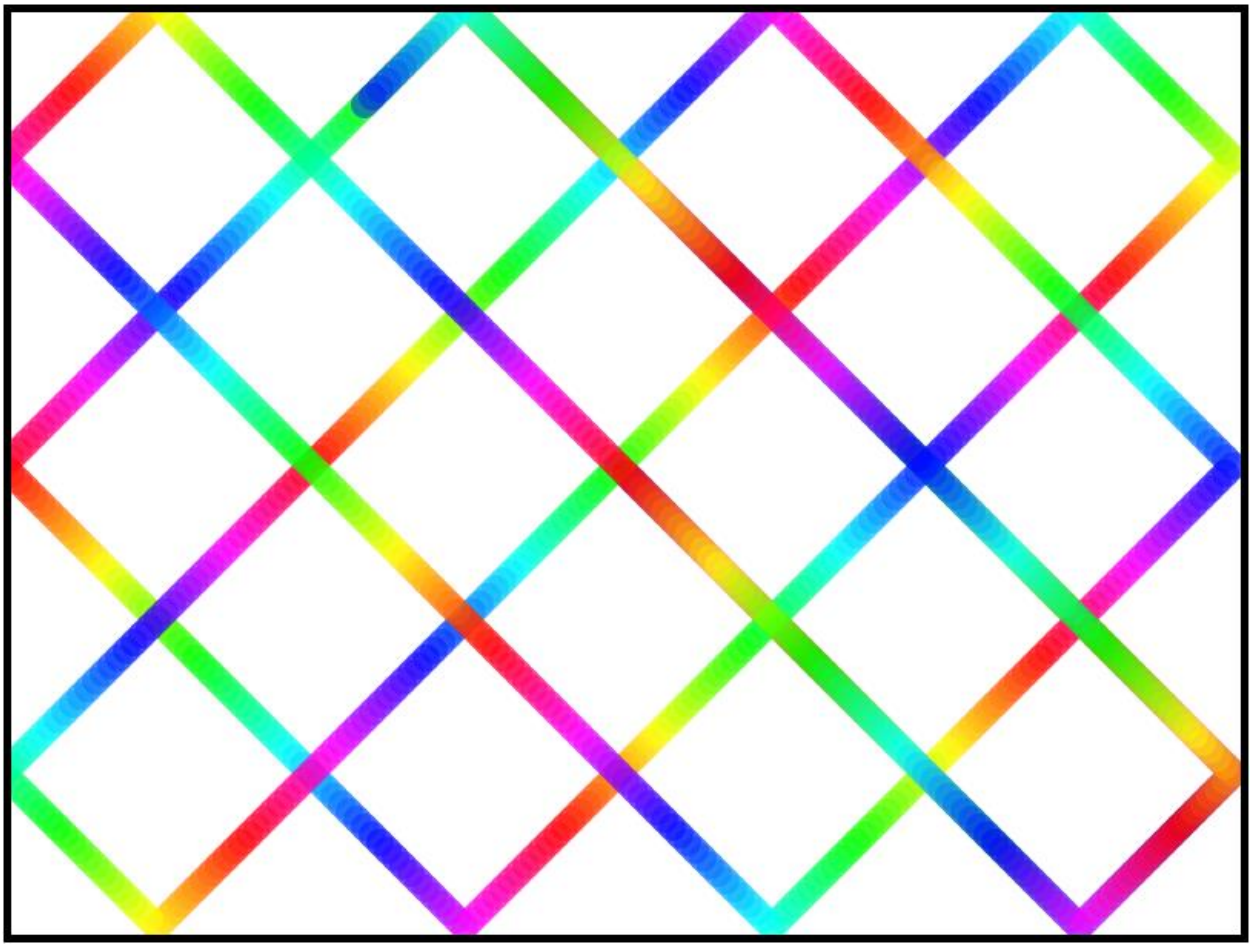
- Our point should now be bouncing around our canvas. We can also change its appearance using the *'Appearance'* subcategory – let's change the hue to red (360) or blue (240). You can mix and match your own colours here: http://hslpicker.com/



- Solid colours aren't quite as interesting as variable ones, so let's change the colour of the line as we go. Under the *Events* subcategory, select the *after step, do* block and place it in the blockly work area. Now go back to the *Appearance* subcategory, and drag a *change hue by* block onto the work area, inside of the *after step, do* block. This will cause the point to change its colour after every step (movement) on the canvas

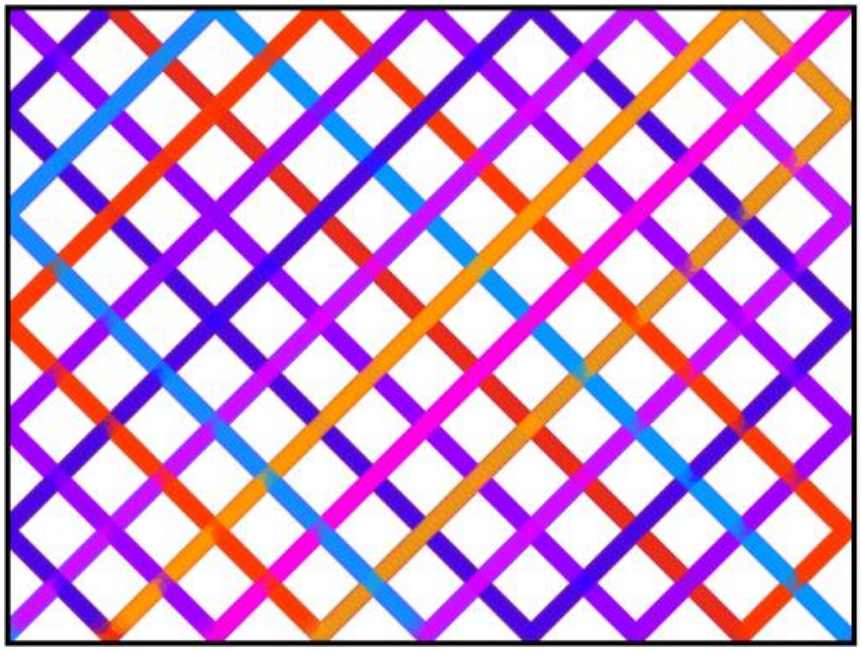


- The point should now be moving along the screen and gradually changing its colour as it moves along. The final product should look something like this:



Using Loops

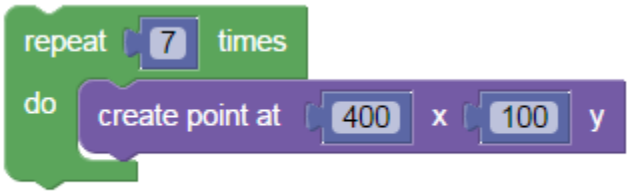
- In this section, we'll introduce the concept of using loops to duplicate code. Here's an idea of what the end product will look like:



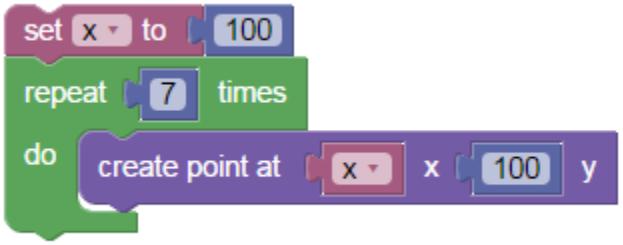
It looks similar to our first product, but we'll have more than just one point creating lines this time.

- Let's make more than one point this time; clear the canvas and drag a *create point* block onto the workspace, but don't run the code yet

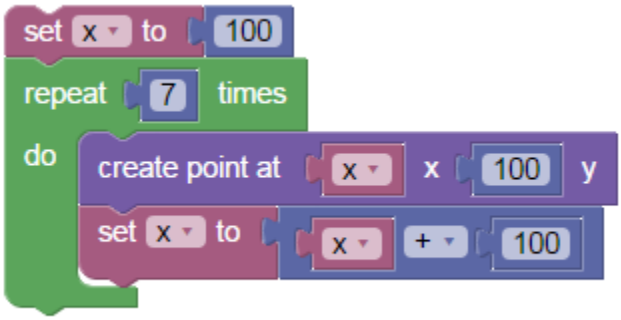
- We can use some loops to help us duplicate code. Go to the *Loops* category, and pull out a *repeat* block onto the workspace like so



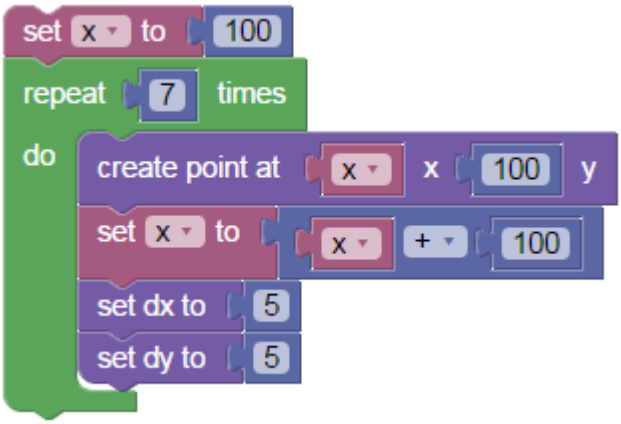
- However, this gives us seven points at the same spot. If we want to put them in different places, we'll have to use a variable that changes with each iteration. In the *variables* section, drag the *set* block onto the workspace, as well as a number block from the *Math* category, and attach it in like so:



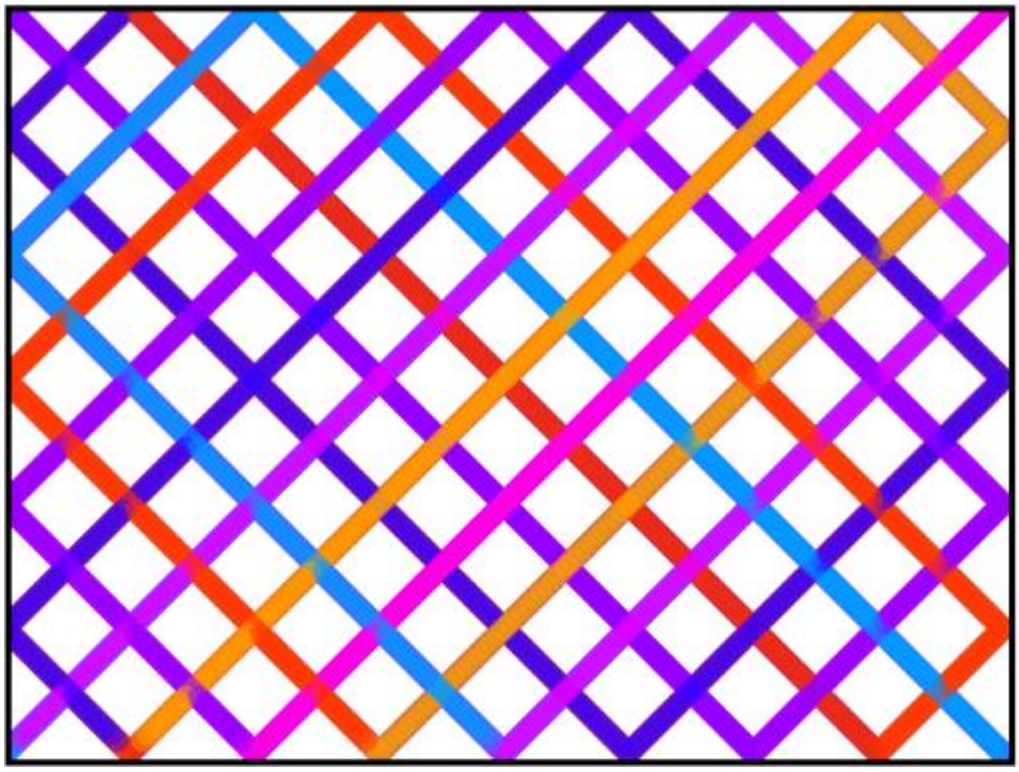
- You can rename the variable (or create a new one) using the dropdown menu in the *set* block. Notice we've replaced the number value of x in our *create point* block to the x variable now. However, our variable needs to change after each iteration, so let's get another *set* block and put it inside the repeat loop. We'll add 100 to x each time so that our points are spaced 100 pixels apart (the addition block is found in the *Math* category)



- Now our x variable will increase by 100 every time we loop, creating seven points spaced out horizontally by 100 pixels each. Finally, let's drag in a bit of motion into the mix – we'll give each point the same dx and dy, but you could always change them later

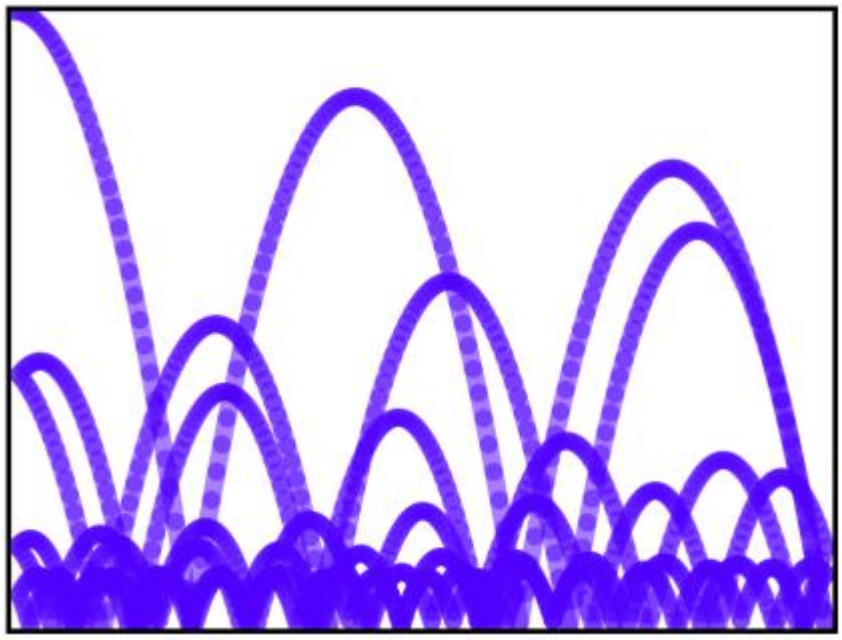


- Press the run code button and let's see what we've created! After some time, the pattern should look something like this:



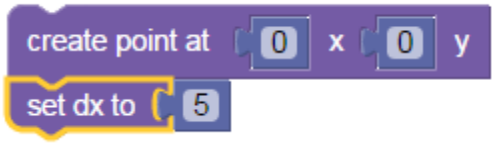
Using After-Step Commands to Simulate Gravity

- After step commands allows you to continuously modify a point after each step it makes. In this example we'll see how to use this command to simulate the effect of gravity.



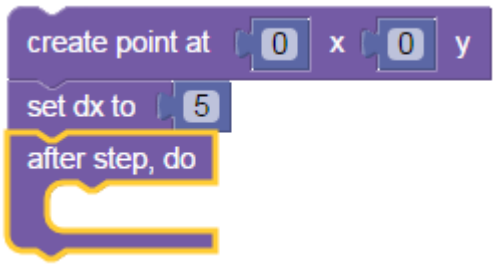
- Begin by clearing the canvas, then placing a create object block onto the Blockly area. Let's set the x and y values to 0 so that the point begins at the top left of the canvas.

- Next let's give our point some motion along the x axis – set its dx value to 5. This will make the point bounce right to left as it hits the boundaries of the canvas. You can test the code by running it, then clearing the canvas – the code on the canvas is not deleted when the canvas is cleared.

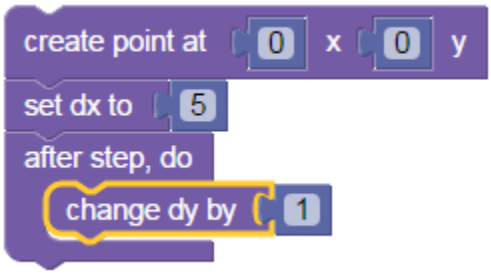


- Now let's give it the pull of gravity. Gravity is a constant acceleration down (or towards the center of the Earth), so after every move the point makes we

want it to be moving down the canvas. Begin by opening the *Events* subcategory in *Canvas Objects*, and drag the *After Step, do* block onto the blockly area. This block tells the point to execute all of these commands after each step it makes.



- We need to tell the point to do something after each step, so we'll need to put a block inside there. In the *motion* subcategory, drag a *change dy by* block inside of the *after step, do* block.

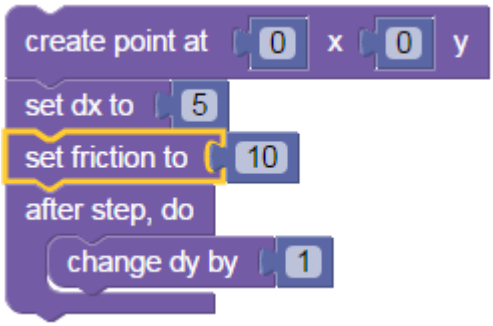


This tells the point that after every step it makes, it needs to increase its dy by 1. By increasing the rate of change of the point's y value, we are giving it *acceleration*, which is what gravity is! Since a y value of 0 is the top of the canvas and a y value of 600 is the bottom of the canvas, increasing the dy value will cause the point to accelerate down the canvas.

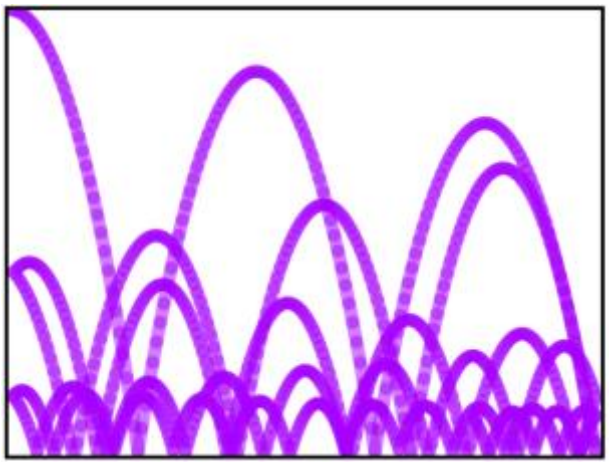
- Press the Run Code button and take a look at what we get...

It isn't quite what a falling ball looks like in real life, is it? Our points initially have zero friction – they bounce *elastically* off of the walls without losing any speed.

- Let's try giving our point some friction. First let's clear the canvas again – you should see the code we were working on appear as we move back to our canvas object. In the *motion* subcategory, drag a *set friction to* block onto the work area.



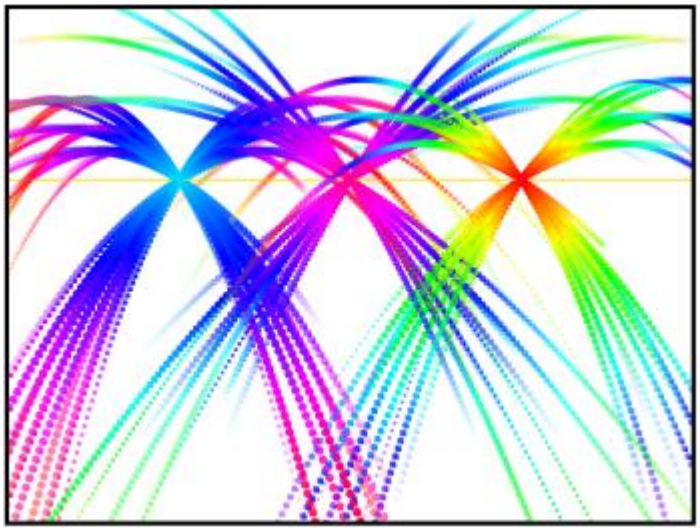
Friction is a value from 0 to 100 that indicates how much of a point's velocity is lost upon hitting an edge of the canvas. Try running the code again, and you should see something similar to this:



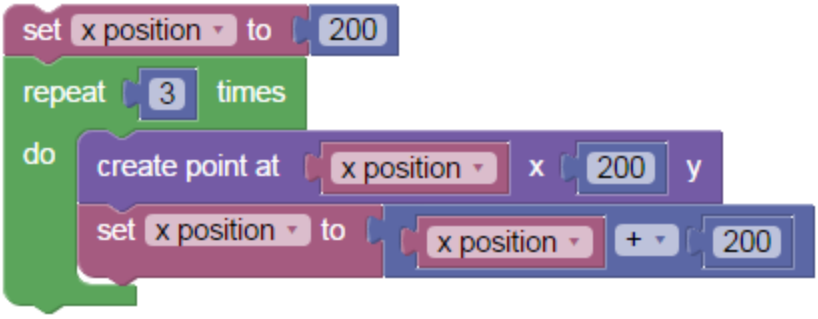
- That's a bit closer to what we would expect the path of a bouncing ball to be like. Try changing the friction or the rate of change for dy and see how they affect the result (e.g. a lower rate of change in dy such as 0.1 will give you a more moon-esque bounce!)

Collision Effects

- Points may have an effect that occurs when another point collides into it. There are currently two effects – fireworks and spirals. Here's a sample of what the fireworks effect looks like:

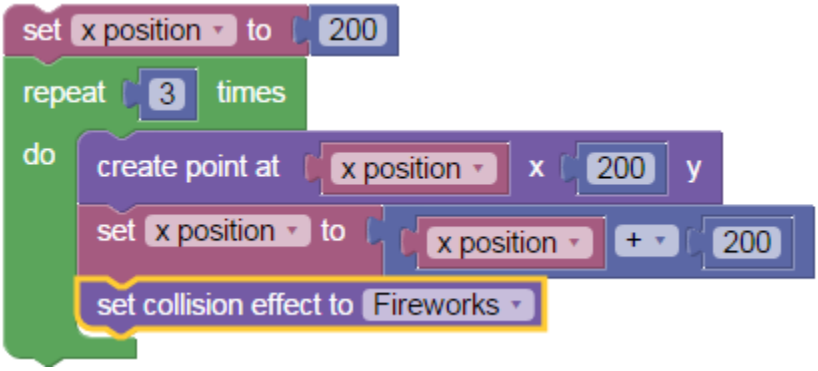


- For this exercise, the idea is to have three static points that have this firework effect, and one moving point that will trigger them. Let's start by creating three points using a loop as we have used before:

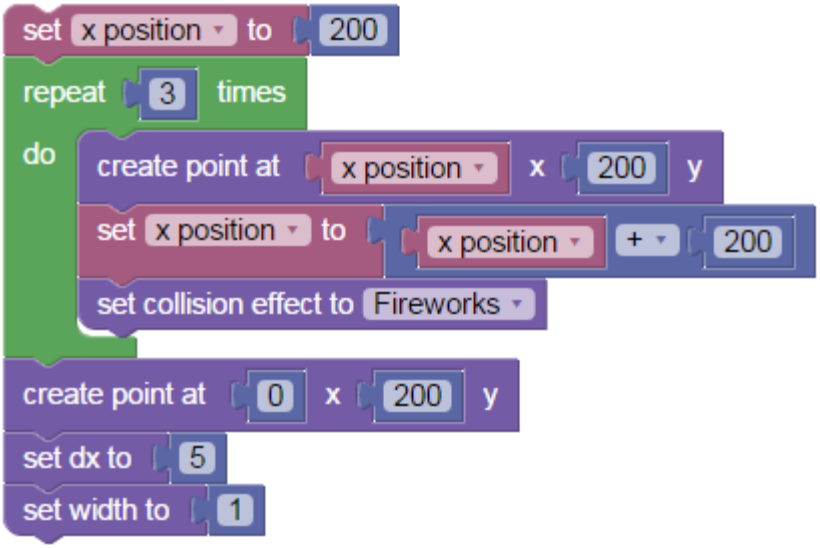


This will create three points for us at (200, 200), (400, 200), and (600, 200) respectively.

- Now let's set the effect for each of these points. In the *Canvas Effects* category, select the *fireworks* subcategory and drag the *set collision effect to* block inside the repeat loop like so:



- Finally, let's make a point that traverses across these three points so that it triggers the fireworks:



If you don't want to see the line, you can set its opacity to 0 so that it is invisible. Here we've simply set the width to 1 to make it quite small.

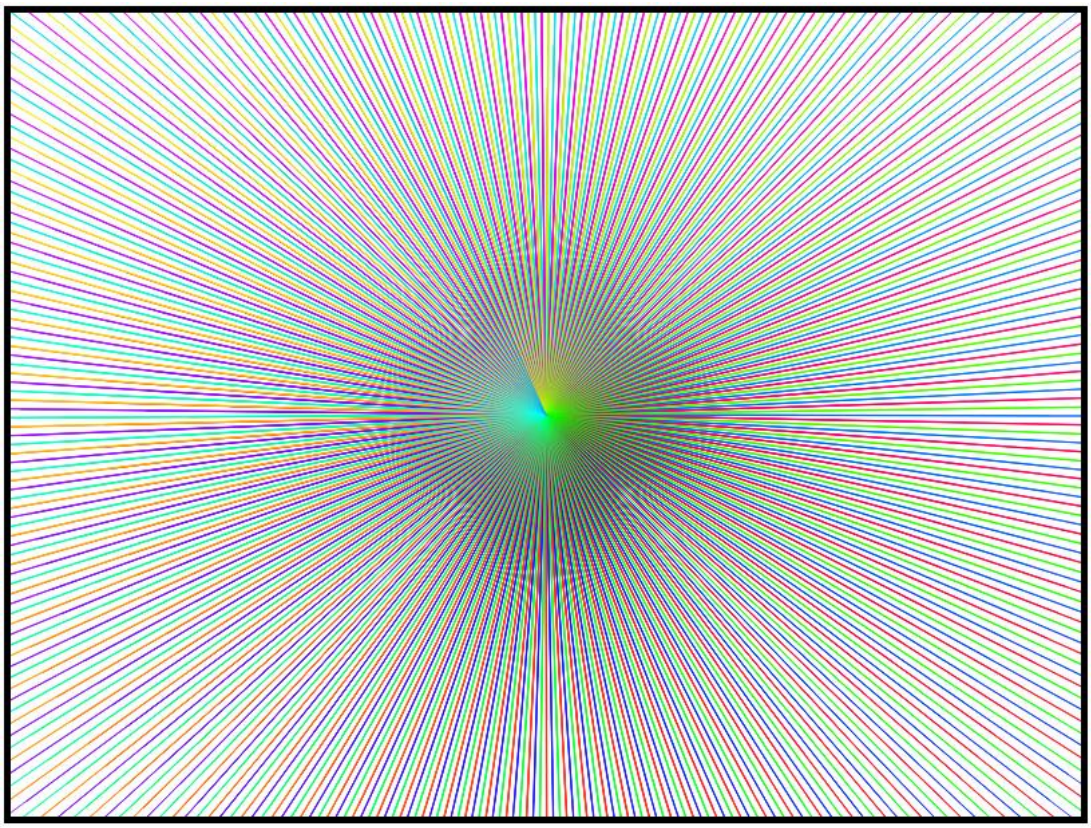
- The *fireworks* subcategory has various attributes that you can modify – try setting some of them by placing the block just after the *set collision effect* block inside the repeat loop.

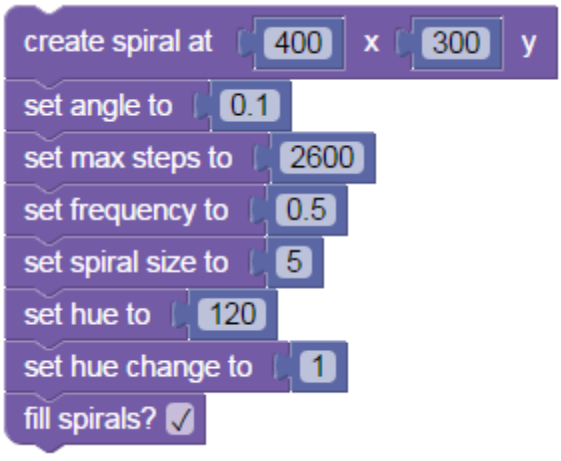
- Spirals work similarly to fireworks, with the obvious exception that they create spirals instead. Check out the next section to see more about them!

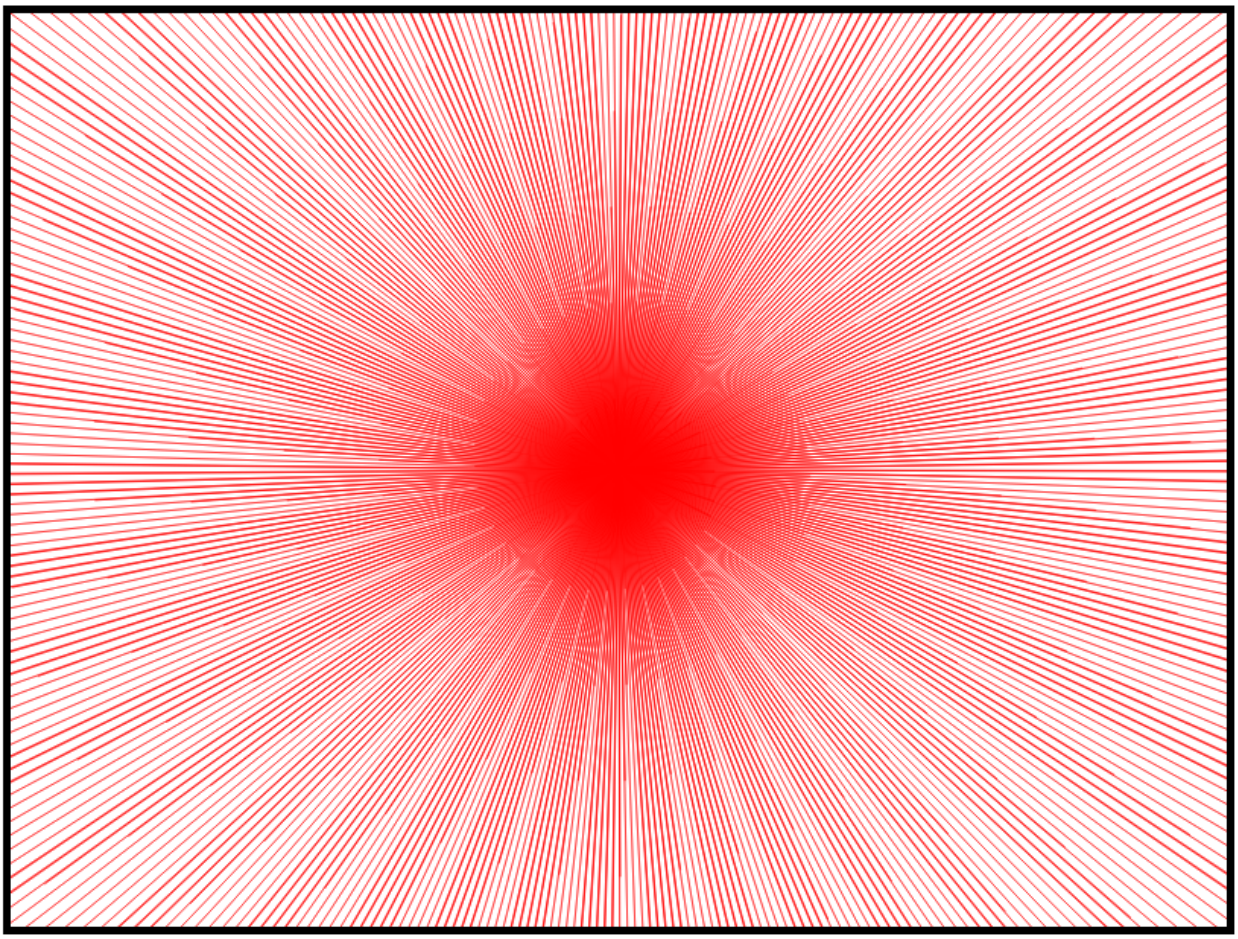
Spirals

- Spirals can be used as effects like the fireworks, but you can also create standalone spirals. Simply drag the *create spiral* block from the *Canvas Effects* category onto the workspace to create one.

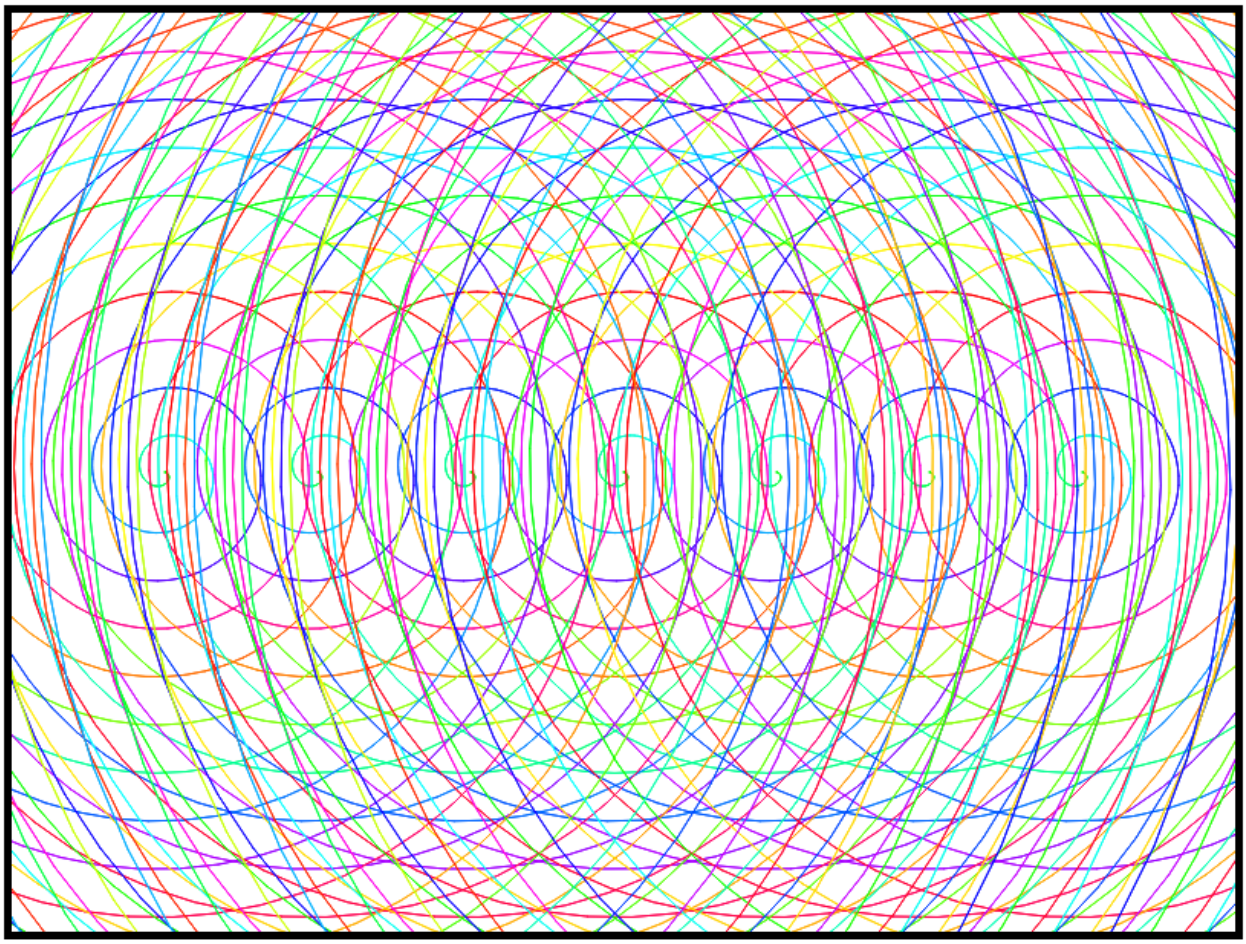
- Spirals, like fireworks, have several attributes that you can modify to create different types of spirals. Try modifying some of the properties and see what kind of spirals you can come up with! Here are a few examples.

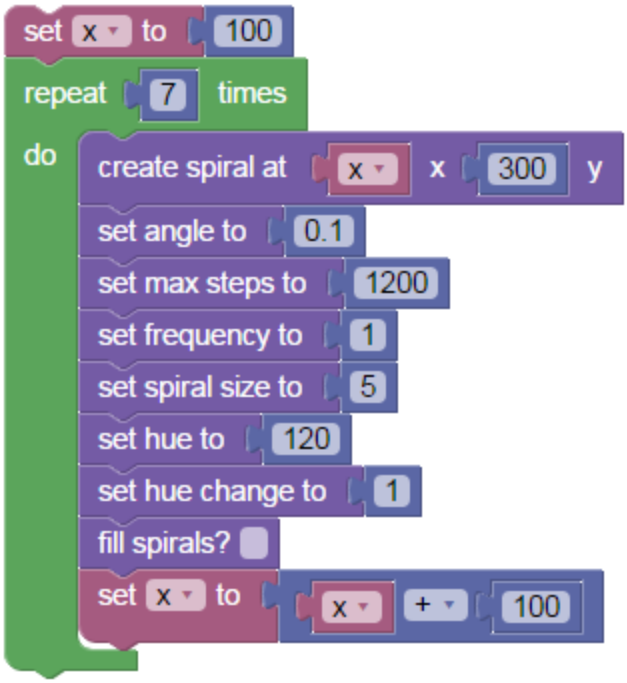






```
create spiral at 400 x 300 y
set angle to 0.3
set max steps to 800
set frequency to 0.5
set spiral size to 10
set hue to 0
set hue change to 0
fill spirals? ✓
```




set x to 100
repeat 7 times
do
create spiral at x x 300 y
set angle to 0.1
set max steps to 1200
set frequency to 1
set spiral size to 5
set hue to 120
set hue change to 1
fill spirals?
set x to x + 100