Traffic Lights - Scratch

This project will simulate the behaviour of traffic lights. UK traffic lights have a specific sequence which is fairly simple to explain.

1. Red - Traffic should stop
2. Amber & Red - drivers should get ready to go
3. Green - Traffic is free to go
4. Amber - Traffic should prepare to stop

This loops round forever. In general it is used to give multiple incoming lanes of traffic access to a common bit of road. The lights are used to prevent collisions and share the road fairly.

Some pedestrian crossing have a slightly different sequence

1. Red - Cars should stop
2. Flashing Amber - Cars can proceed if the crossing is completely clear
3. Green - Cars can proceed
4. Amber - Prepare to stop

This project will show how you can break up your code into functions. The functions that we shall write are as follows.

1. A function that can generate any combination of the red/amber/green lights using the pen tool.
2. Utility functions that generate specific states (just red, red and amber, etc).
3. Sequence functions that call the utility functions in the correct order. We will end up with two of these, pedestrian and standard.

Step 1 - Write the Draw Light Function

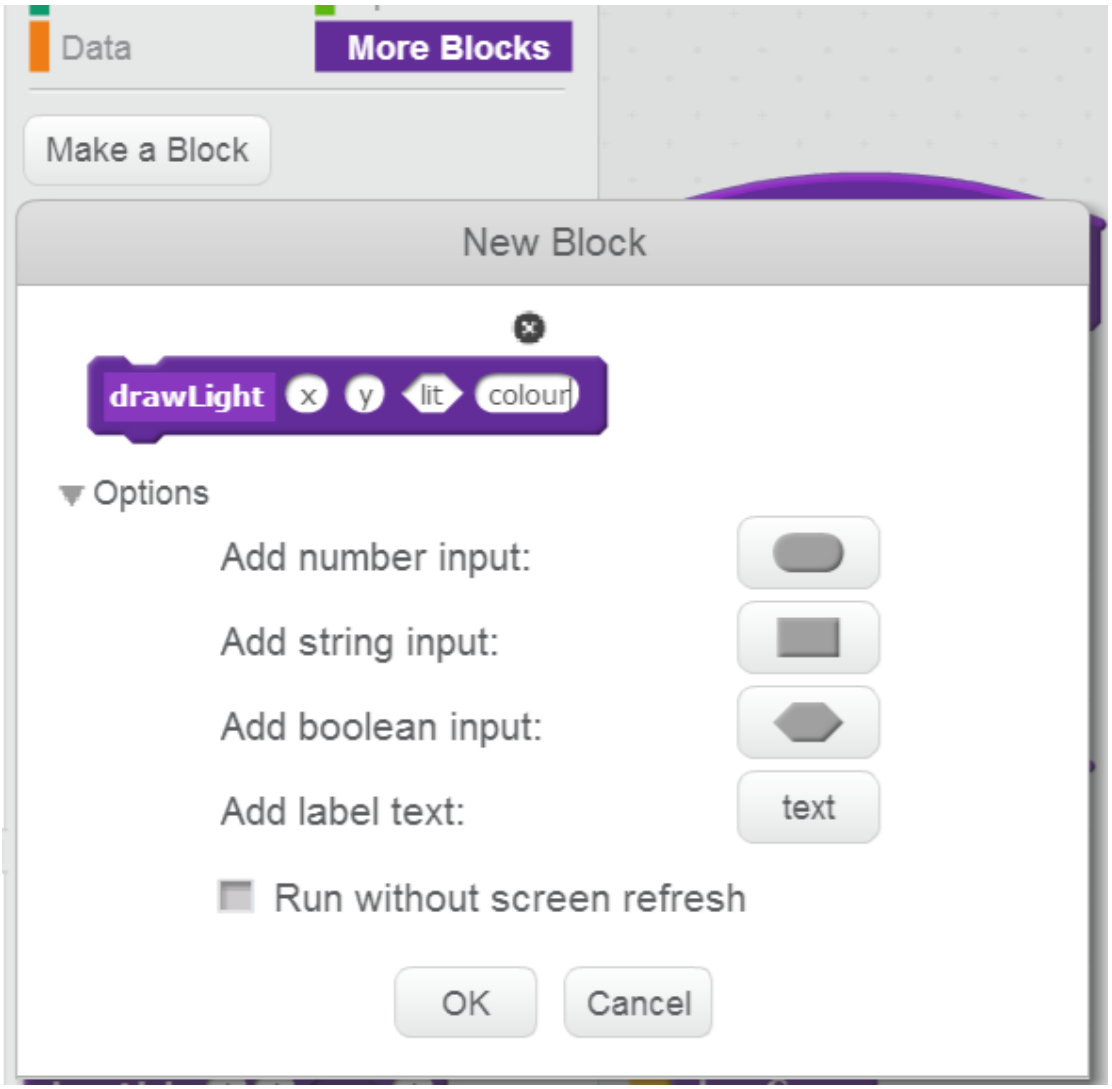
This function will draw a single light using the pen tool. It will need the following arguments.

1. x-position.
2. y-position.
3. indicator if the light is in the ‘lit’ state.
4. the colour *if* the light is lit (the light will show as grey if it is *not* lit.

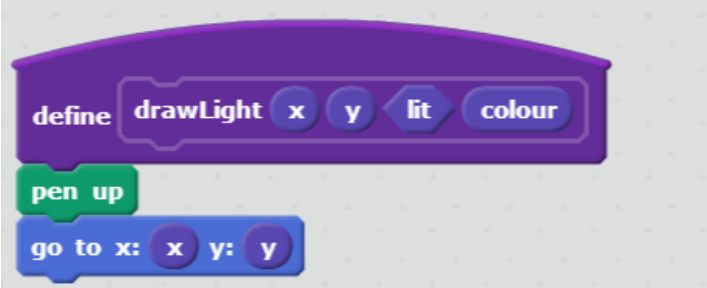
To create this function, under **More Blocks** click on **Make a Block**. Give the function the name **drawLight**.

In order to give the function parameters, we will need to expand the **options** section of the UI and add the following parameters.

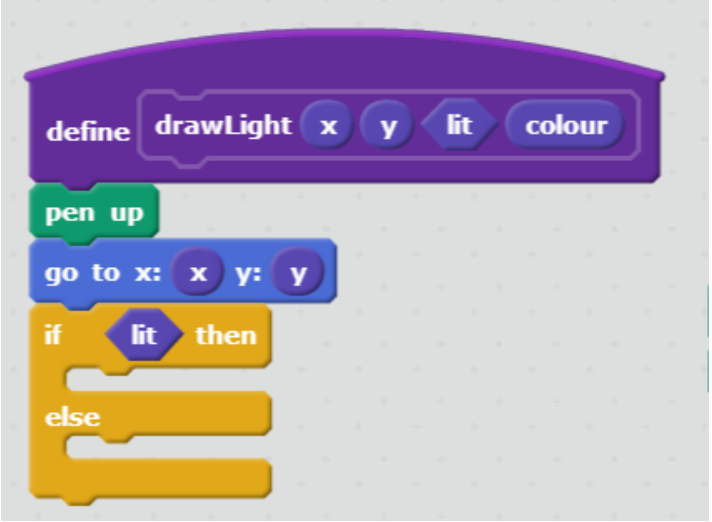
1. Number input called *x*
2. Number input called *y*
3. Boolean input called *lit*
4. Number input called *colour*



Before we do anything to the pen, let us ensure that the pen is up. From **Pen** drag a **pen up** block into our function. Then from **Motion** drag a **go to x: y:** block into our function. Drag the **x** and **y** parameters from the function header into the **x** and **y** parameters of the **go to** block.

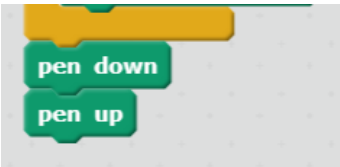


Now we need to set the correct pen colour/shade. From **Control** drag an **if-else** block into our function. Then drag <lit> from the function header into the **if-else** condition.



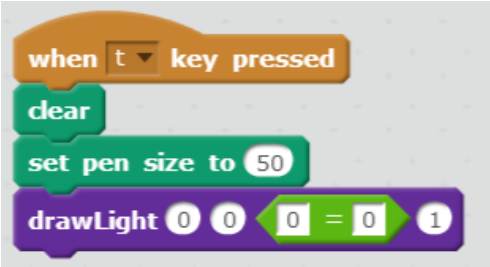
From **Pen** also drag a **set colour to <0>** block into the main **if** statement. Make sure its the colour setter that accepts a number. We also need to set the shade, drag a **set shade to <50>** block in. Drag the **colour** variable from the function header into the colour setter.
From **Pen** drag a **set colour to []** block into the **else** statement. Use the one with the coloured block in for the else. Use the colour dropper to select the grey of the background in the code area. The colour picker is a bit weird *please ask for help if you are stuck*.

Last thing we need to do is to draw the dot on the game area. From **Pen** drag a **pen down** and then a **pen up** block after the **if-statement**.

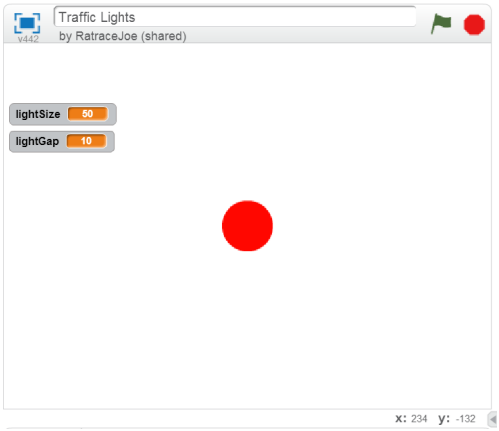


Step 2 - Test the Draw Light Function

Now the function is complete, we should test it.
Make a new event handler, from **Event** drag a **when <t> key is pressed** block into the code area. Use whatever button you want, I have used <t>.
From **Pen** drag a **clear** block into the event handler.
From **Pen** drag a **set pen size to <50>** block into the new event handler.
From **More Blocks** drag a call to our **drawLight** function into the new event handler.
From **Operators** drag a **[] = []** block into the boolean parameter.



That last step is a slight oddity in Scratch, there is no way to just type in TRUE or FALSE. So we have to use some simple operator that will evaluate to TRUE or FALSE as appropriate.
In the test function above, we have tested that zero equals zero. Since zero clearly does equal zero, the operator evaluates to true and the function is called with **lit = true**.
We will see this pattern later on.
Try running the function and you should see something like this.



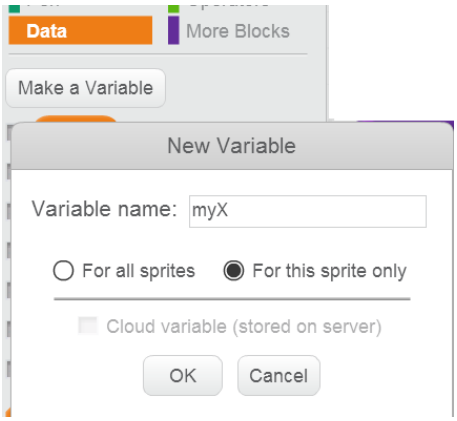
Step 3 - Define some variables

It is good practice to create variables to store values that are used throughout the code. This makes the code easier to read and maintain. For this step we will just create some variables that will be used later.

Under **Data** click on **Make a Variable**. Use this functionality to create the following variables.

- red
- amber
- green
- lightGap
- lightSize

Now create the following 2 variables, but when you click on **Make a Variable** make sure you select **for this sprite only** when you type in the variable name.



The variables should be called

- myX
- myY

These will be used to define the position of an instance of the sprite. We will create them **for this sprite only** so that we can create multiple traffic lights in later steps.

Step 4 - Setup Basic Event Handler

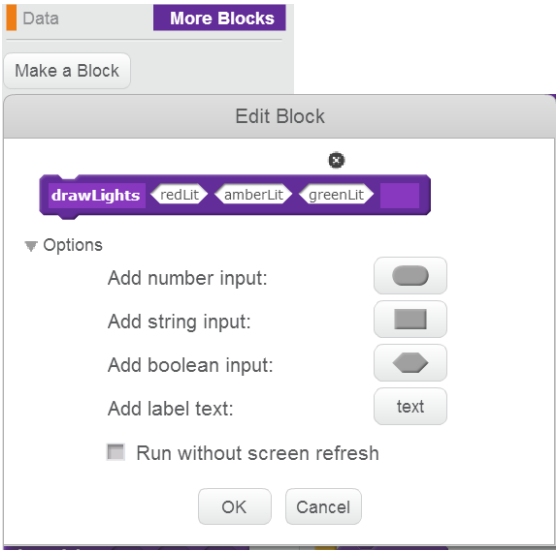
In order to set-up these variables, we will build an event handler for the <space> key. Under **Events** drag **when <space> key is pressed** into the code area.
From **Looks** drag a **hide** block into this event handler.
From **Data** drag a set of 7 **set <varName> to <0>** blocks into our handler and setup the variables as shown.

- myX = 0, myY = 0. This sets the traffic light to be centered around the middle.
- red = 0, numerical value of a shade of red
- amber = 30, numerical value of a shade of amber
- green = 70, numerical value of a shade of green
- lightSize = 50, sets the size of each light
- lightGap = 10, sets the gap between each light (vertical spacing)

In later stages we will add calls to our various draw lights functions, but we can now use these variable values to define sensible behaviour.

Step 5 - Create Draw Lights Function

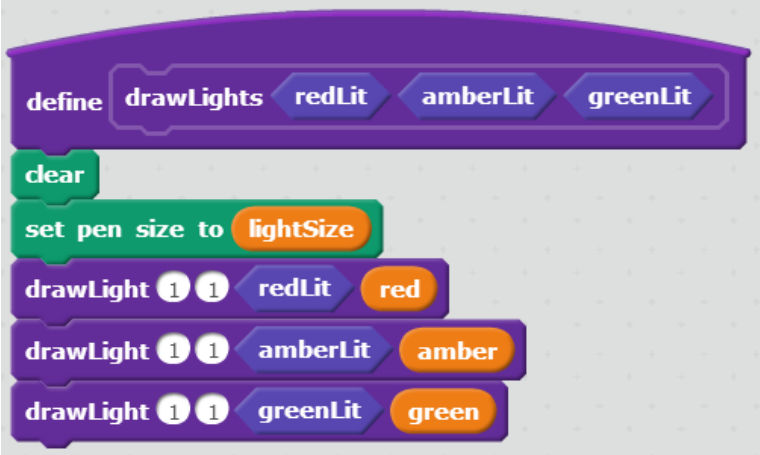
This function will draw 3 lights, positioning them in a vertical line. The middle light will be positioned at the **origin** of the traffic light (as defined by **myX** and **myY**).
Under **Mode Blocks** click on **Make a New Block**. Give it a name **drawLights** and 3 boolean parameters **redLit**, **amberLit** and **greenLit**. These boolean parameters will be used to indicate if the red/amber/green lights should be lit.



First thing we should do is clear the pen, from **Pen** drag **clear** into our new block. Then from **Pen** drag a **set pen size to <0>** in. From **Data** drag **lightSize** into the new pen size setter.

To draw the 3 lights, we need to call our **drawLight** function 3 times. From **More Blocks** drag 3 calls into our function.

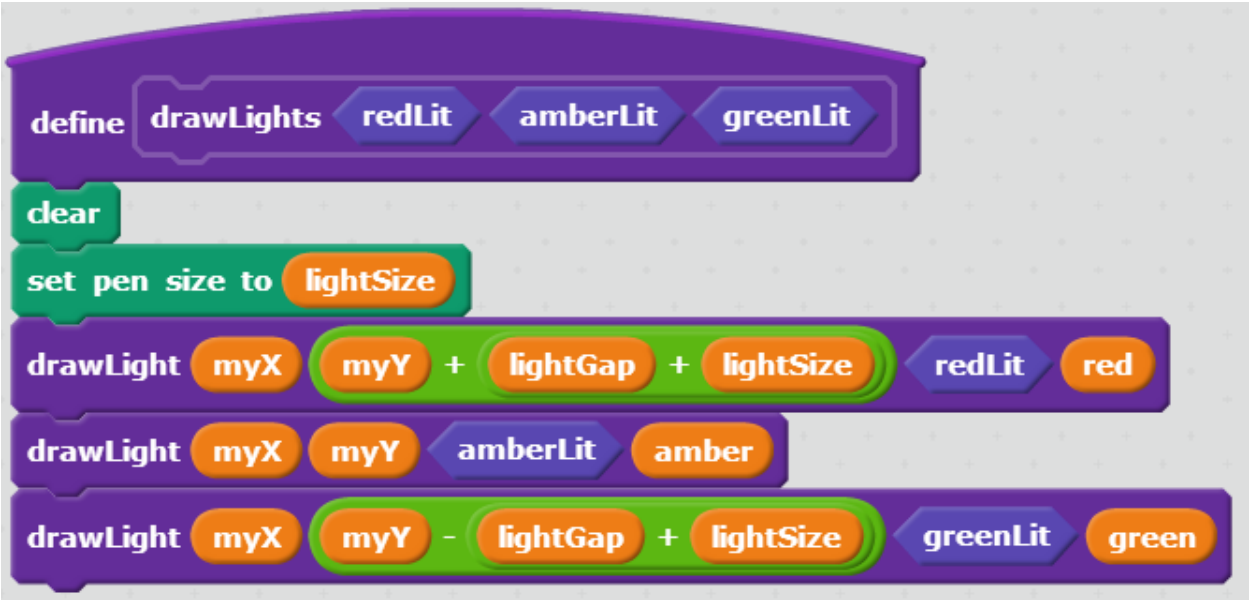
Each call will pass one of our boolean inputs into the **drawLight** function. From the function header drag red/amber/green lit indicators into the drawLight functions. From **Data** drag the values of **red/amber/green** into the colour spaces for the **drawLight** functions.



The x-value in each case is the defined **myX** variable. From **Data** drag **myX** into each x variable. The y-values need to be set as follows.

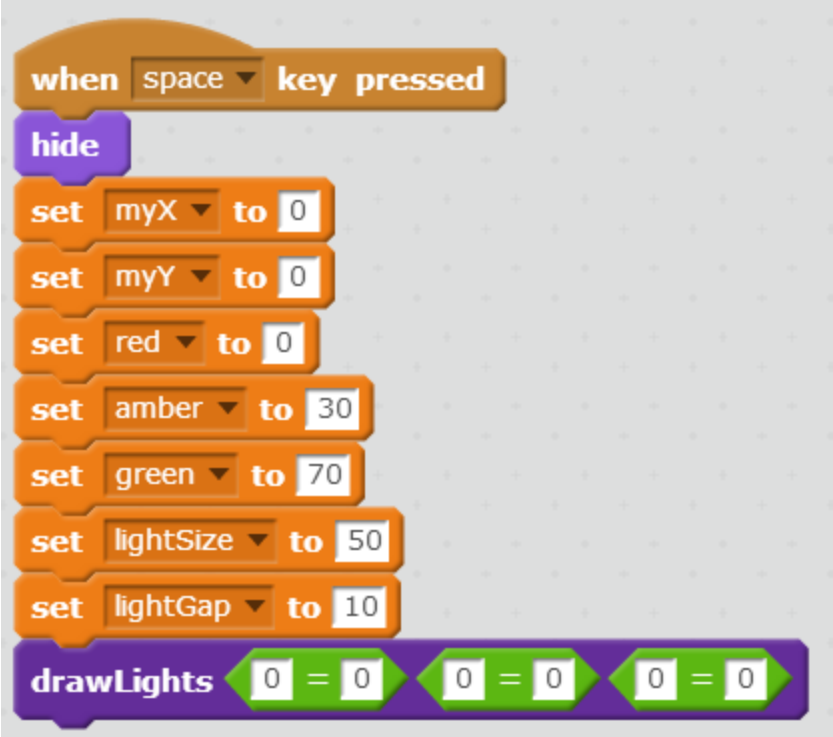
- RED y = myY + (lightGap + lightSize)
- AMBER y = myY
- GREEN = myY - (lightGap + lightSize)

Use a combination of values from **Data** and the **subtraction** and **addition** blocks from **Operators** to build the following.

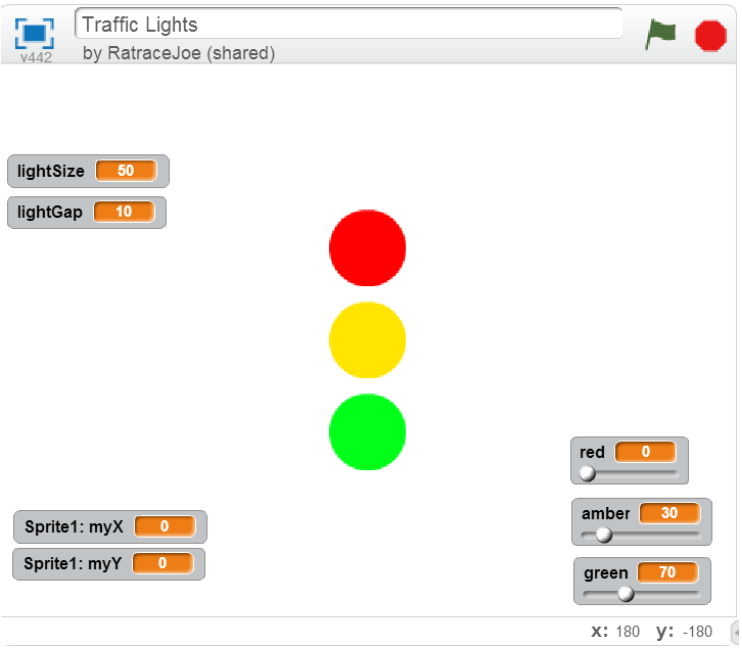


Step 6 - Test Draw Lights Function

In order to test the function, add a call to **drawLights** to the end of our initial event handler. Use the **0 = 0** from operators to pass in TRUE to red/green/amber lit parameters.



Run the test and you should see the following.

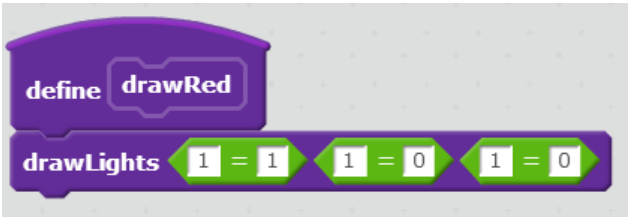


Step 7 - Create Draw Red Utility Function

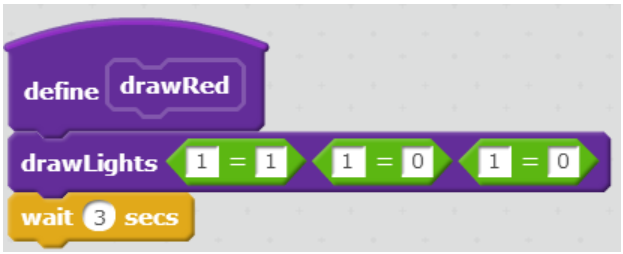
In this step we will create a single utility function to draw the lights with just the RED light on. We will need functions for all the different states, but I will leave it to you to determine exactly what to write there.

To start the drawRed, under **More Blocks** click on **Make a Block**. Give it the name **drawRed** it won't need any parameters.

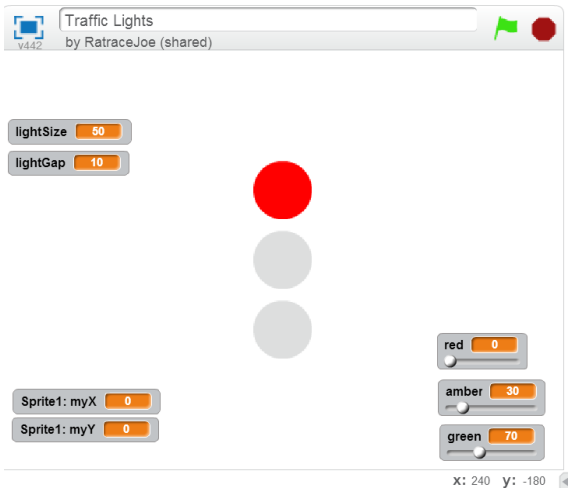
Into our new function, drag a call to **drawLights**. Then use the 0=0 from **Operators** to pass TRUE into RED and 0=1 to pass FALSE into AMBER and GREEN. Pay close attention to the operators, the function will now look like this.



When a red light is shown, we shall put in a fixed delay of 3 seconds. From **Control** drag a **wait <3> seconds** block after the **drawLights** call.



Feel free to call the drawRed function at the end of our <space> event handler. The game area should now look like this.



Step 8 - Create Other Utility Functions

Use the same procedure as the previous step to generate the following functions. These should be sufficient to generate the standard sequence listed at the start of this tutorial.

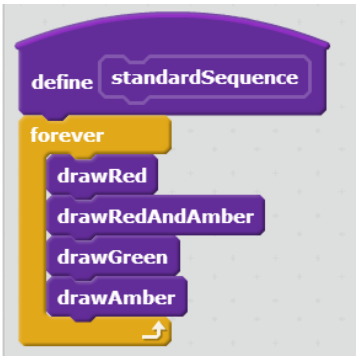
- drawRedAndAmber - lights the red and amber lights, waits 1 second
- drawGreen - lights the green light, waits 3 seconds
- drawAmber - lights the amber light, waits 1 second.

Step 9 - Create Run Standard Sequence Function

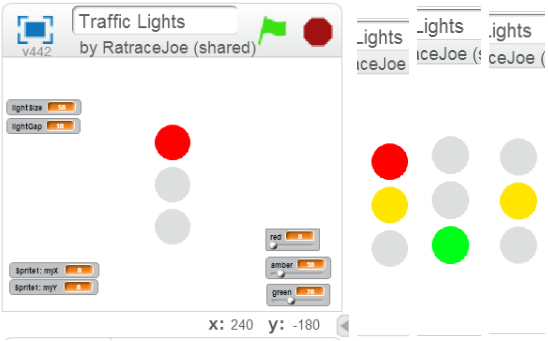
Now that we can get the lights into the correct state with the appropriate delay, we will now build the function that loops round the standard sequence.
Under **More Blocks** click on **Make a New Block**. Give it the name **standardSequence**.
From **Control** drag a **forever** loop into the the new function.
Remember the sequence is as following

- drawRed
- drawRedAndAmber
- drawGreen
- drawAmber

The function will look like this.



Replace the call at the end of the <space> event handler with a call to our new function. The outptu should look like this.



Step 10 - What Next?

- Try and define a function that will show flashing amber.
- Define another function that generates the sequence for the pedestrian crossing.
- Try and create clones of the sprite, each with their own x and y values, to yield 2 sets of lights.
- Setup broadcasts that allow the 2 sets of lights (standard sequence) to be doing the opposite, when one is green, the other should be red. Figure out how to control this to simulate a real road junction.
- Create a new sprite that simulates a green/red man (pedestrian signalling)
- Get a single set of lights using the pedestrian sequence, and a green/red man light set. Put a button in the game area that allows a pedestrian to request a red light. Then trigger a red light.