Color Wheel

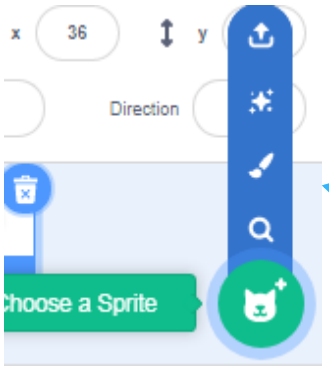
Vineet Srivastava

In this lesson, we will …

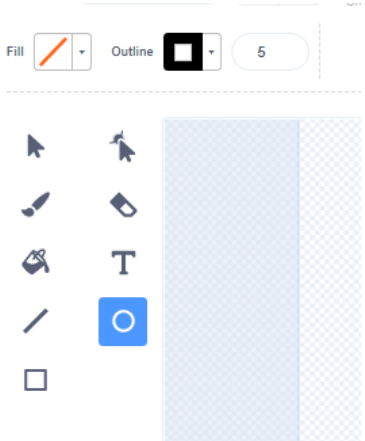
* Build an elegant yet engaging game called the color wheel.
* In this game, the user has to move (turn) a color wheel to make sure the balls coming towards the wheel make contact with the correct color on the wheel.
* In making this game, we will use costume editing in bitmap mode, elementary rotation commands, cloning, duplication and of course, coding logic.

Color Wheel

* We will first make a color wheel.
* Follow the steps below:



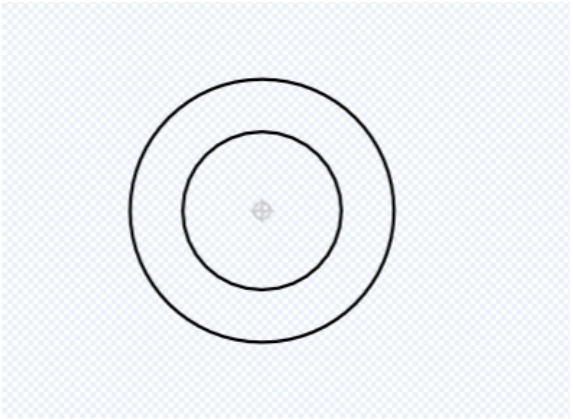
Choose ‘Paint a sprite’

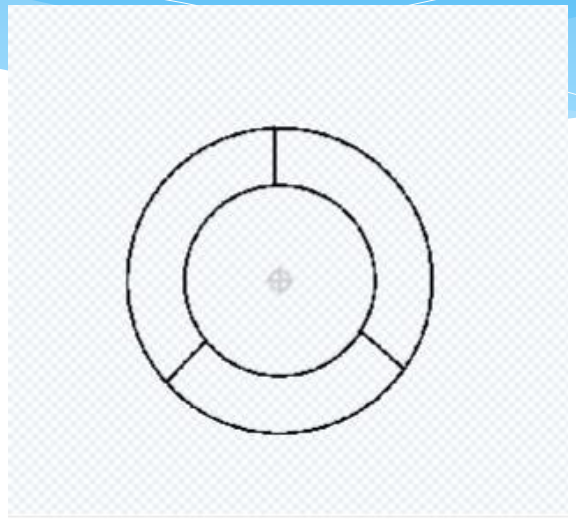


Set FILL to No Fill, Outline to 5.

WIBYTE

Color Wheel





Draw two ‘concentric’ circles – Basically circles with the same centre. Make sure that the circles are nicely centred.

NOTE: CLICK SHIFT WHEN DRAWING TO ENSURE YOU GET A CIRCLE.

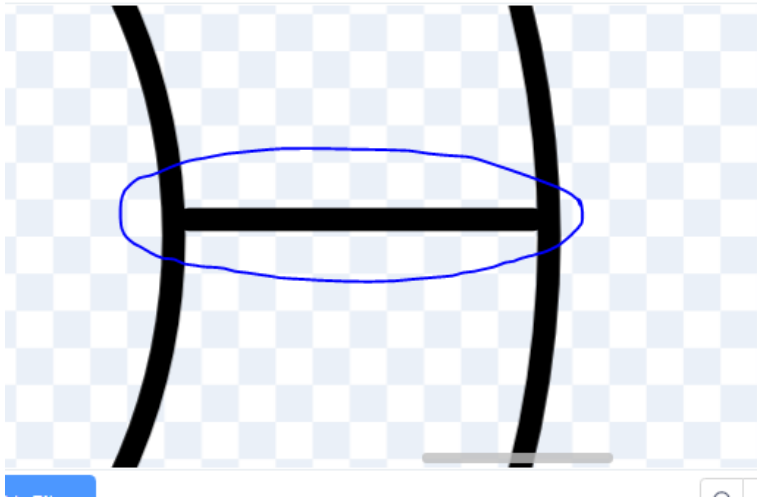
Divide the circle into 3 equal parts as shown above.

Make sure that the 3 lines drawn this way a completely touching the circle. (See next slide)

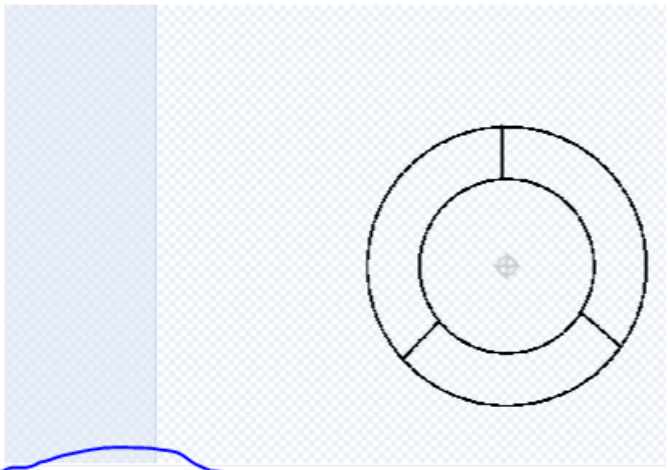
WIBYTE

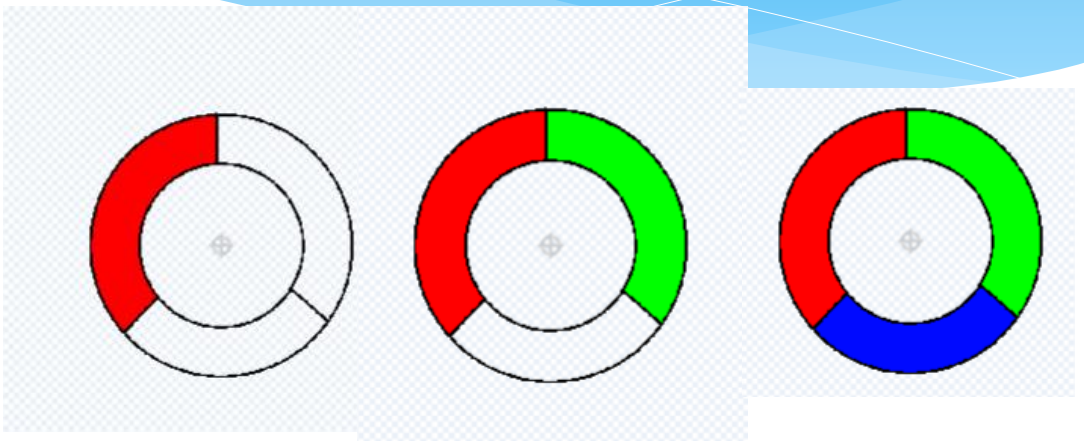
Make sure there are no gaps

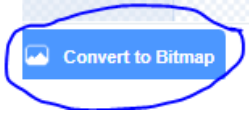
* Zoom-in on all the 3 lines to make sure there is no gap between the lines and the circles. (Do this on all 4 lines).



Completing the color wheel

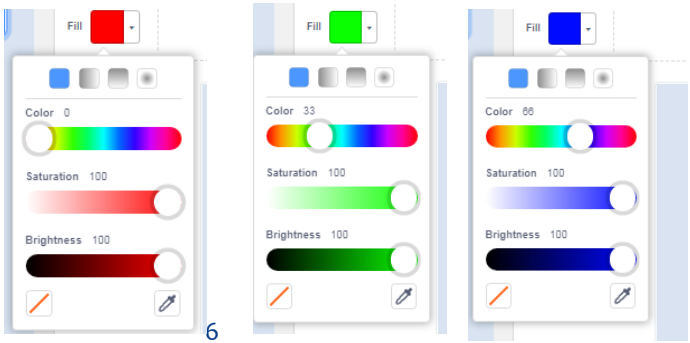






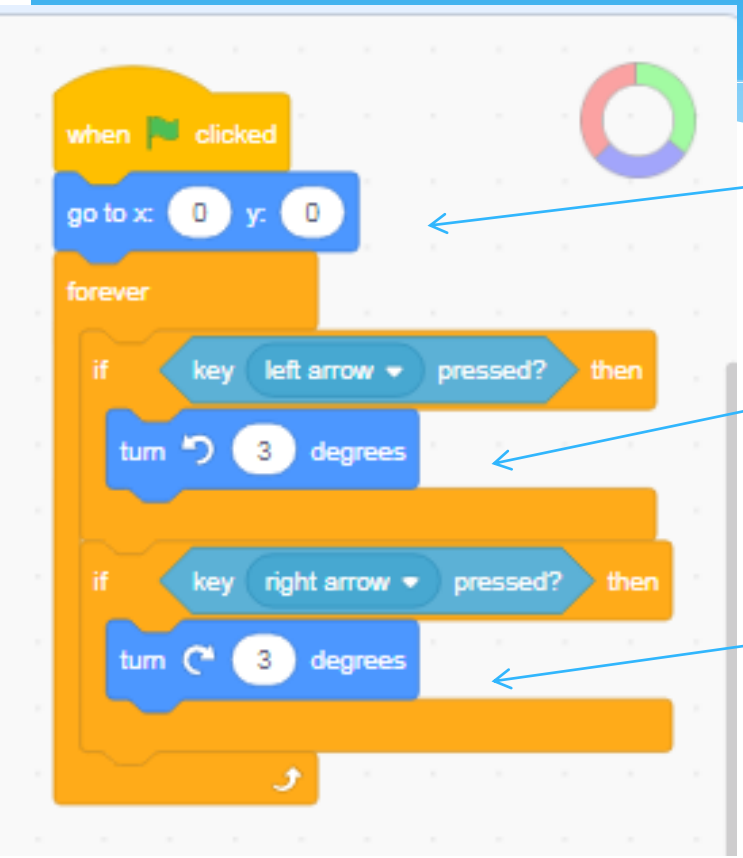
Convert to bitmap mode

Click the fill tool, choose a colour and click on one of the 3 segments.



WIBYTE

Code for the color wheel



Send the color wheel to (0, 0)

If the Left Arrow is clicked, move the wheel anti-clockwise by a small amount.

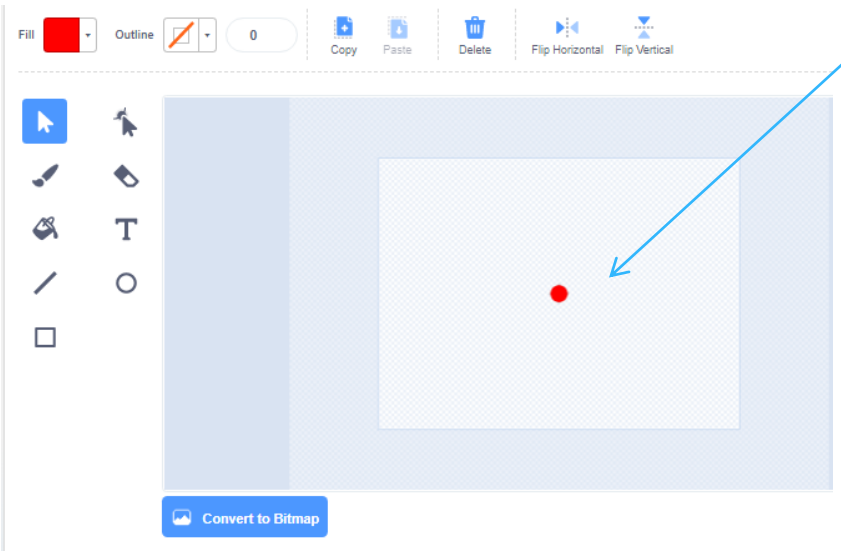
If the Right Arrow is clicked, move the wheel clockwise by a small amount.

NOTE:

1. Do not turn by very large numbers else the game is not pleasant to play.

Create a ball sprite

* Once again, use PAINT a sprite and create a small circle (dot).
* MAKE sure this circle is centered properly.

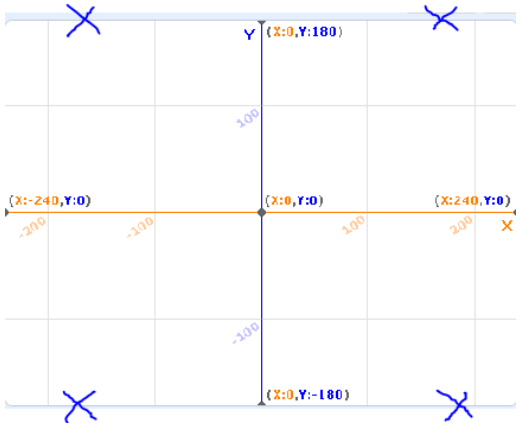


Also, for aesthetic reason, use one of the 3 colors that was used for the color wheel.

Code for the ball sprite

* In our game, there will be several copies of the ball.
* We will use cloning to get these.
* Furthermore, we want the ball sprite to start from one of these four corners:
  * x = 180, y = 180
  * x = -180, y = 180
  * x = 180, y = -180
  * x = -180, y = -180
* Notice, these are marked as

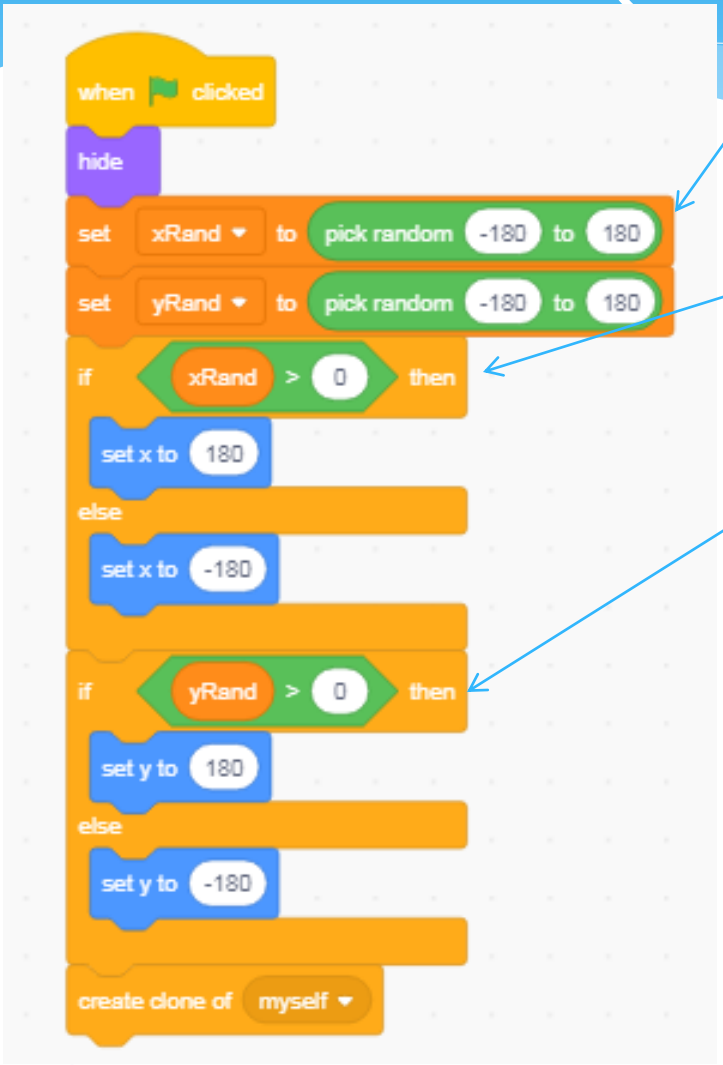
'x' on the x-y grid backdrop.



Sending ball to one of the 4 corners (Method 1)

* There are really two decisions to be made:
  * Does the ball go to LEFT side or the RIGHT side?
  * Does the ball go to the TOP or the BOTTOM?
* We will use two variables for this purpose, call them, xRand and yRand, and assign both of them to separate random numbers.
* The idea is simple, kind of like ‘tossing a coin’ twice to answer the two questions above.

Sending ball to one of the 4 corners (Method 1)



Pick up random numbers from -180 to 180. For xRand and yRand.

If xRand > 0, send the ball to right side (set x to 180). Else send the ball to left side (set x to -180)

Similarly, If yRand > 0, send the ball to the top (set y to 180). Else send the ball to the bottom (set y to -180)

Notice that there is a 50 % chance that the number that has been picked up is more than 0. (Since 0 is exactly in the middle of -180 and 180).

Taken together, this ensures that the ball is equally likely to appear from any of the 4 corners.

After sending the ball to one of the corners, create a clone.

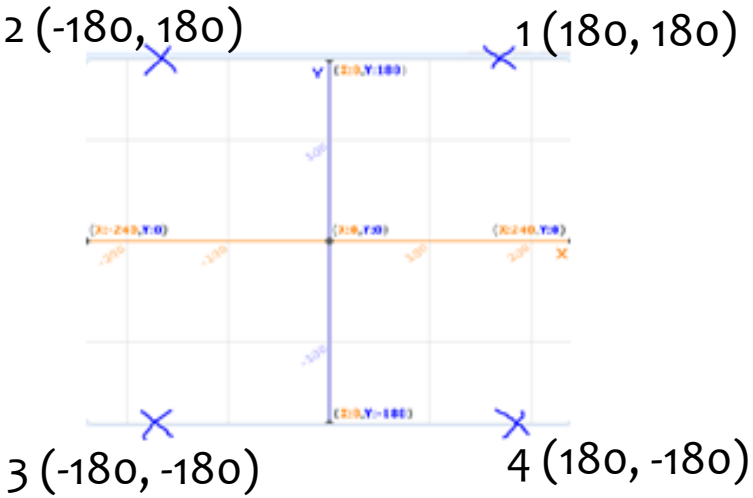
WIBYTE

Food for thought

* In fact, we need not take -180 and 180 as the range for random numbers.
* We can take any other number, for example, say -100 to 100. The important point is the condition for determining whether we should go to the right corner or left corner (or top/bottom corner). This condition is basically to compare against 0.
* Since 0 is exactly mid-way between -180 and 180, we ensure that in *half* the cases the ball starts from left corner.
* Also, this ensures that the ball is equally likely to start from one of the 4 corners.
* Point to ponder, what will happen if we used:
  * If (xRand > 45) instead of (xRand > 0). Try this.

Sending ball to one of the 4 corners (Method 2)

* Another method can be helpful here. Say we label the 4 corners as

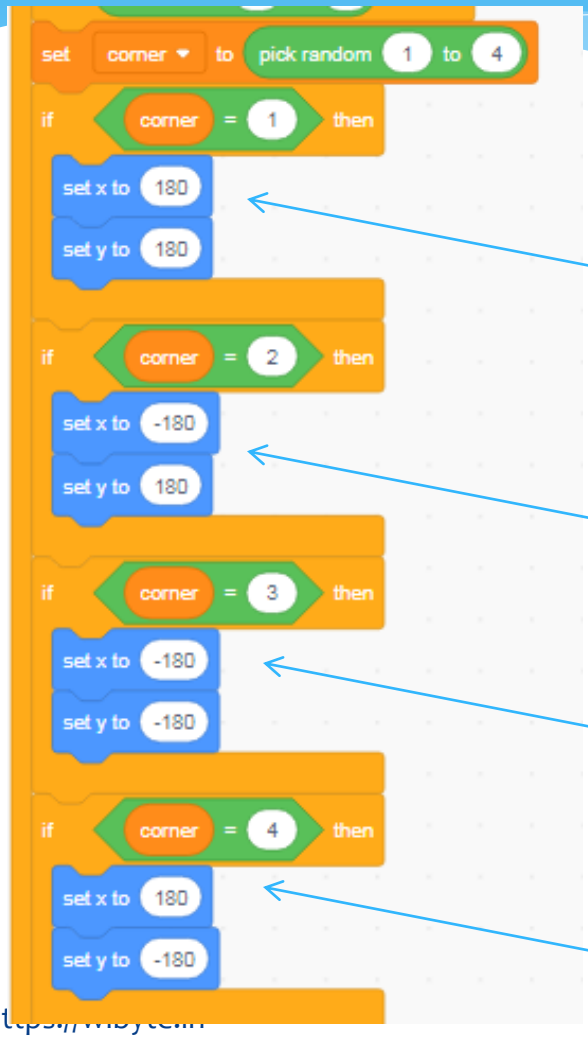


Now, we can use a variable called ‘CORNER’.

Give it a value between 1 and 4, each with equal likelihood.

Use this value to determine which corner is chosen.

Sending ball to one of the 4 corners (Method 2)



Notice, CORNER is equally likely to be 1, 2, 3 or 4.

If corner is 1, choose (180, 180)

If corner is 2, choose (-180, 180)

If corner is 3, choose (-180, -180)

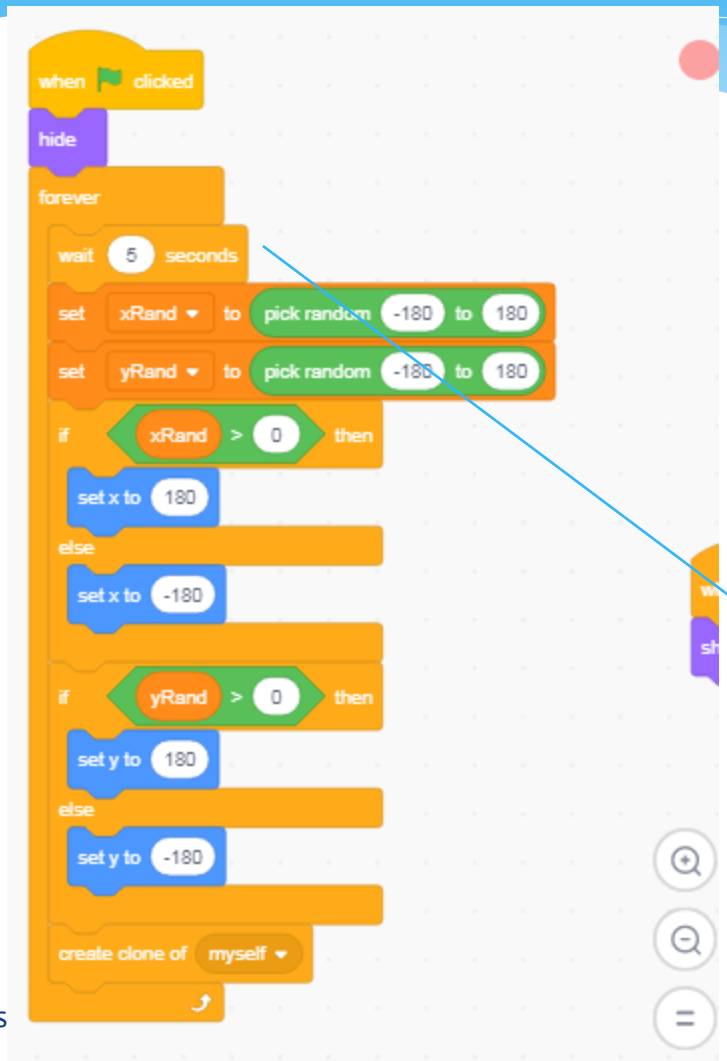
If corner is 4, choose (180, -180)

WIBYTE

Food for Thought

* In Method 2, can you think of some way to increase/decrease the chances of the ball appearing from any one of the corners?
* HINT: Think, can you make the corner take values from, say, 1 to 10 and use these to determine which corrner the ball goes to.

To make this appear regularly

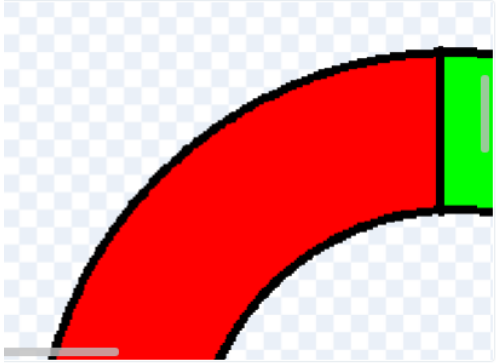


We can wrap the entire code in a forever loop to make sure the ball keeps going to a corner, every 4 to 5 seconds.
Also, we can add a bit of randomness

WIBYTE

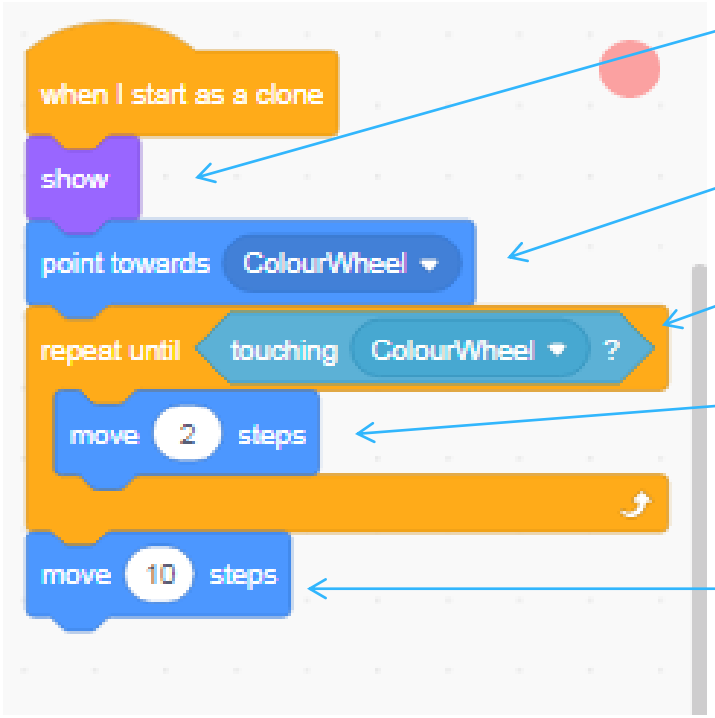
Making the ball move

* Once the ball is started as a clone, we want it to move towards the color wheel.
* We also want to make sure that the ball is touching the correct color on the color wheel.
* However, see the color wheel carefully, zoom in if needed: There is an outer boundary of BLACK outside every colour.
* This means that the ball, when moving towards the colour wheel, will first touch Black than any other colour.



WIBYTE

Code for the ball to move



Remember, the BALL was HIDDEN when FLAG was clicked, hence we need to use SHOW now.

Point towards the color wheel

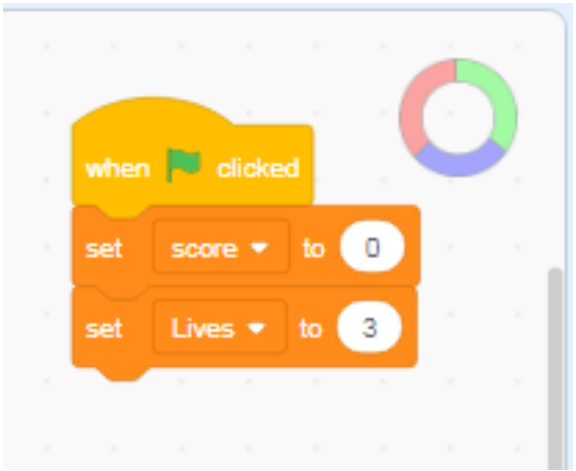
Keep moving towards the color wheel UNTIL touching the color wheel

We can use a variable here to be able to change the speed here if we wish to.

Once touching the color wheel, move 10 steps – basically this gets the ball to sort of 'jump' over the black boundary of the color wheel.

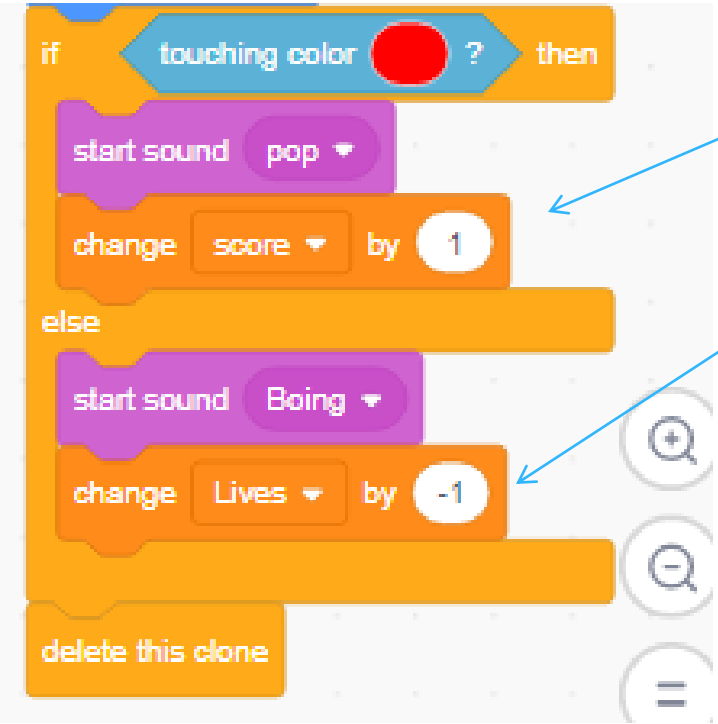
Create score and lives

* Create two variables, score and lives, in the Color wheel sprite.
* Score increases by 1 every time the ball touches a correct color.
* Lives decrease by 1 every time the ball touches a wrong color.



WIBYTE

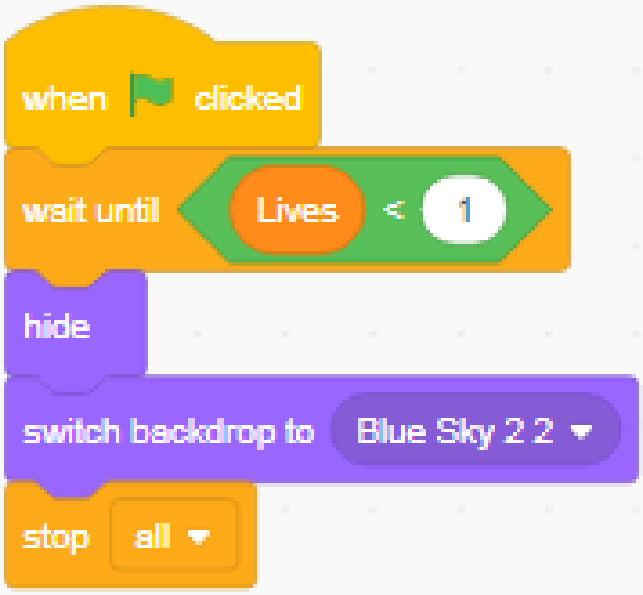
Determine if touching the correct colour



* Create a variable called SCORE and increase it by 1 if the ball is touching the right color.
* If not touching the correct color, reduce lives by 1.
* Finally delete this clone.
* Notice this is just a part of the same 'When I start as a clone' loop.

Getting the game to end

* We can end the game when Lives become 0.
* We will use 'WAIT UNTIL' for this purpose:



Notice:

1. We do not put when <Lives = 0> (See next slide for explanation)
2. We do not need a forever here, since this event (lives < 1) is expected to happen only once.

WIBYTE

Food for thought

* In the ending condition, we have put WAIT UNTIL (LIVES < 1) and not (LIVES = 0).
* To see this, consider a situation where Lives = 1 and 2 balls suddenly hit an incorrect colour. Sudddely Lives will become -1 and since we are checking for Lives = 0, the game may not stop.
* Using a condition (LIVES < 1) guards us against such conditions.

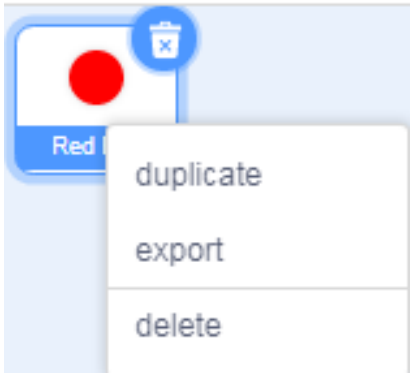
Change Backdrop when game gets over

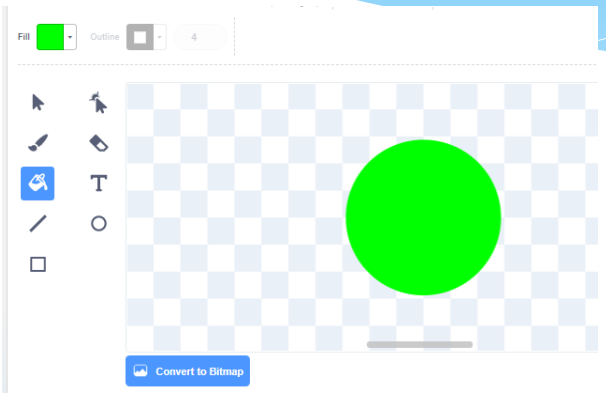
* Just add anew backdrop

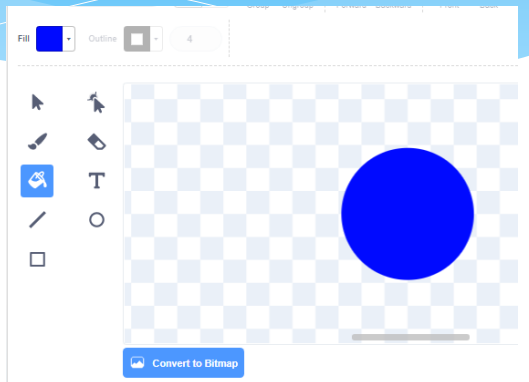
Now to get the other balls

* Having built all this much, it is now easy to extend the code to have two additional balls.
* We will do this by duplicating the RED ball and changing the costume.
* Also, we need to change the color which is considered the ‘correct’ color for this ball.

Duplicate the red ball twice





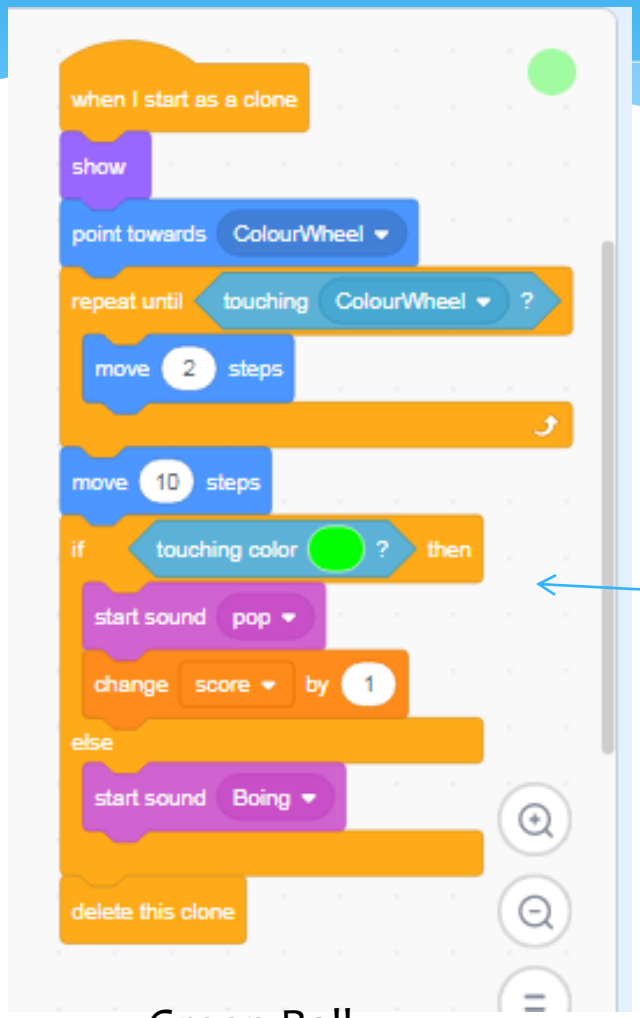


Right click and duplicate the RED ball

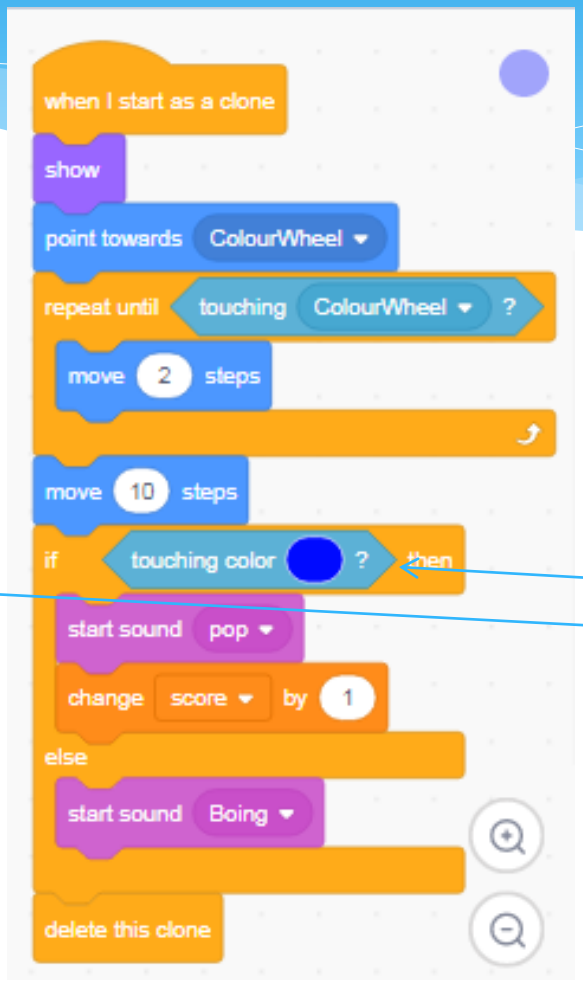
Change the costume to ‘GREEN’

Change the costume to ‘BLUE’

Update the touching conditions



Green Ball



Blue Ball

Notice the updated color

WIBYTE

And you are all set!

* With this, you are all set for your Independent Activity – Color Wheel game.
* Use this opportunity to sharpen your concepts of cloning, loops, variables etc.
* Most importantly, enjoy ☺

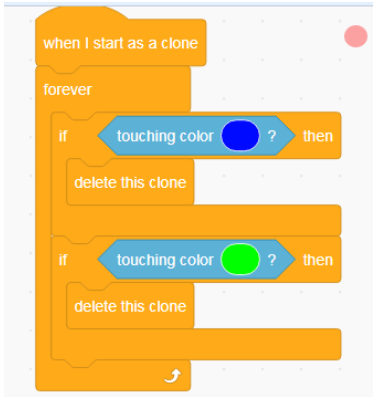
Extra Innings

Use of Lists

* In our game, we used two variables xRand and yRand for determining which corner does the ball come from.
* An alternative (and somewhat simpler) implementation of this is using lists. You can take a look here: https://projects.raspberrypi.org/en/projects/catch-the-dots/1
* (We will learn about lists in a latter class)

Balls falling on each other

* As the game progresses, we sometimes have a situation where the balls fall on each other. This leads to confusion and also erroneous scoring (since hidden behind a red may be a green ball).
* One simple method of avoiding this is to add a bit more code to the 'WHEN I START AS A CLONE'.
* Notice this set of codes will delete one of the two balls that are touching each other.



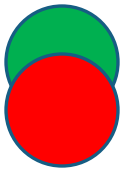
Red Ball

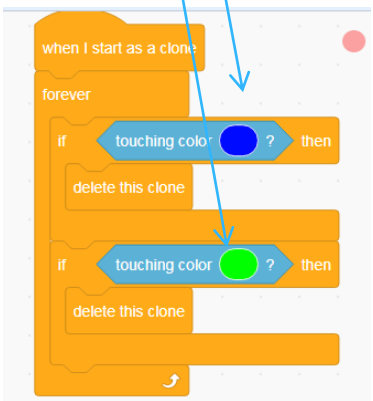
Green Ball

WIBYTE

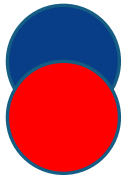
Balls Falling on each other (Examples)

This must be the color of the SPRITE, not the color wheel



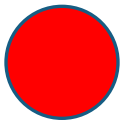


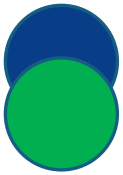
Example 1
(Red touching Green:
Red will get deleted.)



Example 2
(Red touching Blue:
Red will get deleted.)

This must be the color of the SPRITE, not the color wheel





Example 3
(Green touching Blue:
Red will get deleted.)

WIBYTE

Ideas to spice up your game

* Balls move at different speeds
  * Eg. Green moves faster than red, but also scores more points.
* Balls appear more and more frequently.
* Balls speeding up.
* Winning streaks
  * E.g. if you get 10 correct answers in a row, you may get an extra life.
* A magic mode, where you are required to touch ‘another’ color, not the same color.
* Different levels!