

The Technicolor Window

In this lab, you will write custom reporter blocks to perform several useful calculations and computations. You will then use these blocks to make an art project (scroll to the bottom to see the result).

1. Simple Computations


1. Write a custom Snap!! reporter block called "min" that determines which of two numbers (passed as arguments) is smaller and reports that value. If the two numbers are equal, report either one.

Example:  should report **2**

2. Write a custom SNAP! reporter block called "max" that determines which of two numbers (passed as arguments) is larger and reports that value. If the two numbers are equal, report either one.

Example:  should report **4**


3. Write a custom SNAP! predicate block called "between" that determines if a number is between two other numbers (three arguments total). If the first number is equal to either of the other two numbers or is between them, the block should report "true".

Example:  should report **true**, because 4 is between 3 and 10

2. Stepping Things Up

1. Write a custom SNAP! reporter block called "distance to" that computes and reports the distance from a sprite's current position to another point passed in as two arguments. Use the (x position) and (y position) blocks to determine the sprite's position. Remember that the formula for the distance between points (x_1, y_1) and (x_2, y_2) is:

$$\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$$

Example: if sprite is at (1, 0) on the stage, then  should report **5**.

Hint 1: Snap! does not have a "square" block. Instead, you can multiply a number by itself to square it. Hey, maybe you should make the "square" operation a custom block!

Hint 2: The (sqrt of _) block in the Operators tab can be used to calculate the square root.

2. Write a custom Snap! reporter block called "Snap! to range" which takes a number and a range consisting of two other numbers (three arguments total). If the first argument is within the range, it simply reports the same number; otherwise, it reports the number in the range which is closest to the first argument.

Example:  should report **5**, since 3 is closest to 5 in the range 5 to 10.

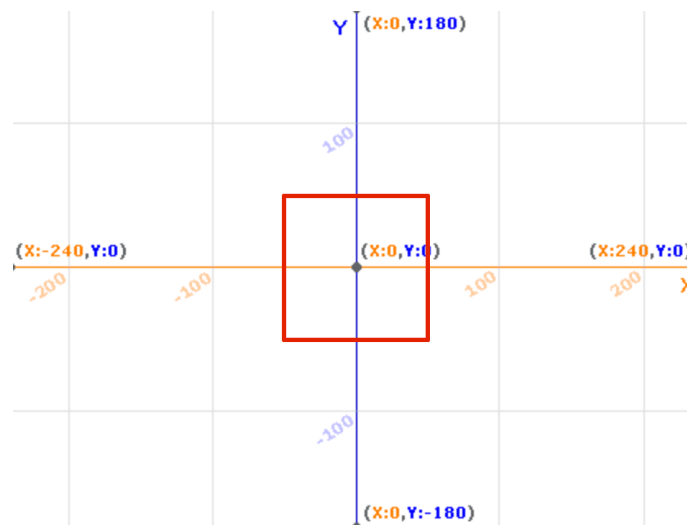
Example: `snap 7 to range 5 10` should report **7** since 7 is already in the range 5 to 10.

Hint: think about how you can use the blocks from part 1 to write this block.

3. Building on our work

1. On the SNAP! stage, imagine a square with corners at $(-50, 50)$, $(50, 50)$, $(50, -50)$, and $(-50, -50)$. In other words, the square covers the area of the stage from $x = -50$ to $x = 50$ and $y = -50$ to $y = 50$.

We have drawn the square on the SNAP! stage below in case you have trouble picturing it. You *do not* need to code this up. By now, we already know that you are an expert at drawing shapes!





Create a custom Snap! block called “go randomly outside square” which will teleport a sprite to a random location OUTSIDE of the square’s area. Looking at the image above, the sprite would go somewhere on the stage outside of the red lines. **Hint:** it may help to think of this in reverse -- how can we tell if the sprite is *inside* the square, using our custom blocks? How can we keep teleporting to random points on the stage until we know that we *are not* inside the square?

Then, add an argument to the block to use in place of “50” -- in technical terms, this argument is half the length of a side of the square. **Hint:** for a given number “num”, you can use $(0 - \text{num})$ or $(-1 \times \text{num})$ to compute its negative. Hey, maybe you should make a “negative of” custom block!

Example: `go randomly outside square 30` will teleport the sprite to a random position on the stage which is not inside the square with corners at $(-30, 30)$, $(30, 30)$, $(30, -30)$, and $(-30, -30)$.

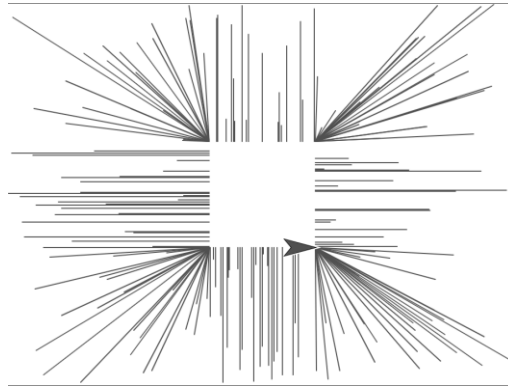
2. Create a custom SNAP! block called “go to square” that takes the same argument as before (i.e. half-side length of a square centered at the stage origin); this block should make the sprite go to the nearest point on the square. If the sprite is already inside the square’s area, then it does nothing. **Hint:** writing this block is simple if we use the “Snap! to range” custom block.

Example 1: If a sprite is at (100, 120) on the stage, then  will cause it to move to (50, 50), the top right corner of the square.

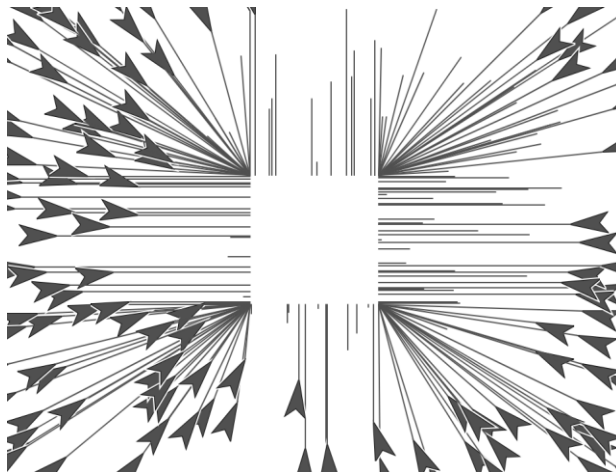
Example 2: If a sprite is at (35, -60) on the stage, then  will cause it to move to (35, -50), a point on the bottom side of the square.

4. Putting it all together

1. Finally, we are ready to make a program! Write a script so that when the green flag is pressed, all pen marks are cleared, and a loop runs where the sprite repeatedly draws a line from a random location somewhere outside the square centered at the origin with side length 100 (i.e. "half side length" 50) to a point on the perimeter of the square. Play around with the number of iterations of the loop (i.e. number of lines drawn) until you can see the outline of our imaginary square on the stage.

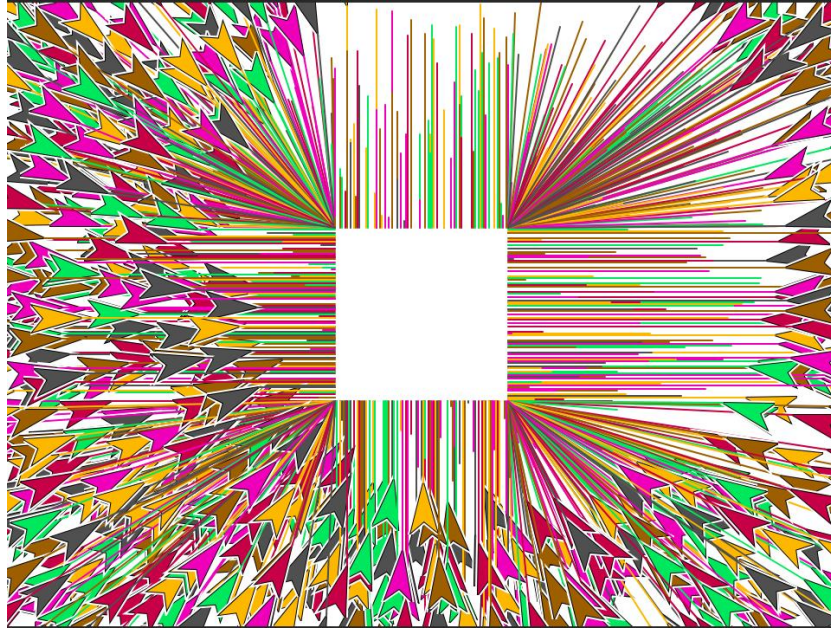


2. Add code to the program so that if the sprite is at a distance of more than 150 steps away from the point (50, 50), it will point towards the center of the stage (use the [point towards _] block in the Motion category) and stamp itself (use the [stamp] block in the Pen category) before it proceeds with the line drawing from the previous step. There should be no stamp marks within a radius of 150 steps from (50, 50). **Hint:** we already wrote a block to calculate distance to a point.



Introduction to Computer Science

3. Add code so that the sprite hides itself once it is finished drawing (you will probably also want to make it show itself when the program begins so it is easier to re-run).
4. **Optional Step** Replace all the constants in your script (square side length, number of lines drawn, stamp distance from the corner of the square) with custom reporters which simply report the same constants. What is the benefit of this? If you want to change any parameters of your artwork like the "window" size later, it is simpler since you only need to edit the custom blocks!
5. Duplicate this sprite a few times (right click the sprite in the sprite corral and select "duplicate" -- the new sprite will have the same scripts) and give the clones different pen colors of your choosing. Re-run the program to observe your art with more color!



6. **Optional Step** Feel free to adjust the colors, line thicknesses, number of lines drawn per sprite, window size, and the "no stamp" radius before submitting your work.

Grading Rubric

Lab 3.4 Criteria	Earned	Points
1.1: min block		2
1.2: max block		2
1.3: between block		2
2.1: distance to block		3
2.2: Snap! to range block		3
3.1: go randomly outside square block		3
3.2: go to square block		3
4.1: repeated random line drawing		3
4.2: stamping		2
4.3 to end: final touches		2
Project total		25

