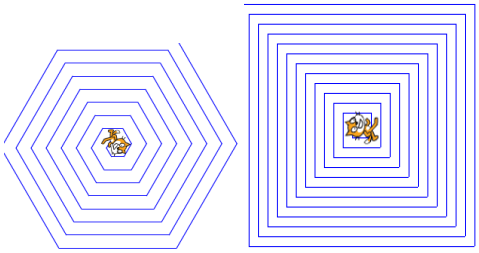
BEA Math Camp Name:
7 July 2015

In the previous session, we saw the application of two important programming concepts: the **repeat** block (or loop), which is useful when we know how many repetitions we need to make and the **variable.** The next project will further refine these ideas. With repeat loops, we may not always know how many repetitions to make. To address this there is a **repeat until** block. The repetitions continue till some condition is fulfilled (or violated). Finally, computer programs can also make decisions, using the **if** and the **if – then –else** blocks.

Project 2: Drawing Spirals

1. Modify the script you made to draw polygons, so that each polygon is now drawn as a spiral, as shown:



2. Notice how the *length* of the side of the polygon now gets smaller with each turn – till it stops. Hint: Recall how the *Side* was made into a variable, allowing it to be changed. A similar idea will allow you to decrease the length of the side.

3. At a certain point, the length of the side is small that the script should be stopped. In the examples above, *the length is decreased and another layer to the spiral is drawn*. This continues as long as **the length is > 4**. This can be done by using the **repeat until** block (found under the control blocks).

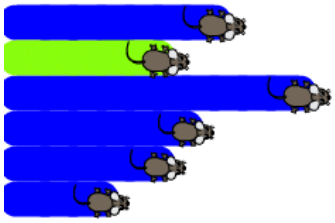
4. Experiment with increasing the number of sides – what do you observe?
You may need to scale the length of side drawn by the number of sides of your figure, to keep the figure within the screen.

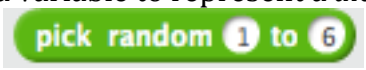
5. Make sure your figure spirals inwards – if it does not, (i.e. you get a funny squiggle – Why does this happen?), you will need to stop the script before this happens. Once you figure out why this happens, you can use the **if block** (Control), with the **stop this script** block (Control).

Project 3: **Rolling Dice and mouse races**:

1. In this project, we start probability experiments (simulations). Imagine a race between 6 critters (*sprites*). Depending on the throw of a die, simulating each roll, by choosing a random number between 1 and 6.

2. If the number rolled is , say 5, then the critter representing 5 moves 20 steps, leaving a thick line behind. Each die roll is made by pressing a key (e.g. space key).



3. Set up 6 sprites. The scripts for all of them are nearly identical, so, write one of them and then copy them over to the other five. Start sprite 1 near the top left of the screen. Create a variable to represent a *die roll* (make sure it is *for all sprites*) and set its value to 

. This should be triggered by an *event*, like the pressing of (say) the space key.

4. Next use an **if block** to check if the value of the *die roll* variable is 1 (or any of the other five possible outcomes. If it is 1 then increase the value of the variable representing 1 by 1, putting the pen down (choose a large value for the pen thickness) and moving 20 steps.

5. Create another sprite that is the finish line – as soon as a sprite touches the finish line, *broadcast* a message, telling all the other sprites that the race is over. This can be done by using another **if block.**

6. When the sprites receive the message, they all show their values (i.e. the number of times each of the other die faces appeared.

Project 3 Challenge:

Modify the above program to include two dice. Now the 11 sprites should represent the **SUM** of the two dice (which are all possible outcomes: 2,3,….12).