By the way, my friend and I talked through the 447 project and worked out our solution. If you want to do it this way, it may be easier to work together if you both follow a similar general idea.

The frog tracked in memory (or two registers) by the top left corner (in the below example, [0,0]). The other pixels are automatically created and destroyed when the frog moves. The frog always leaves behind a yellow trail, because that is the color of stones… The only occasion upon which the frog will not leave behind this trail is when it initially starts.

(0,0)	(0,1)
(1,0)	(1,1)

As for the stones, we've decided that each stone should have the following things in memory:

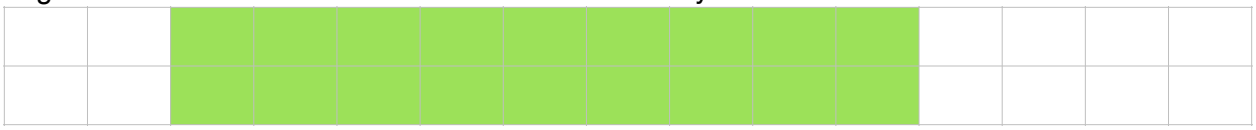
Int Velocity: (randomly generated at start of program)
Array flush_array:
Boolean toRandomize:
Boolean flush:

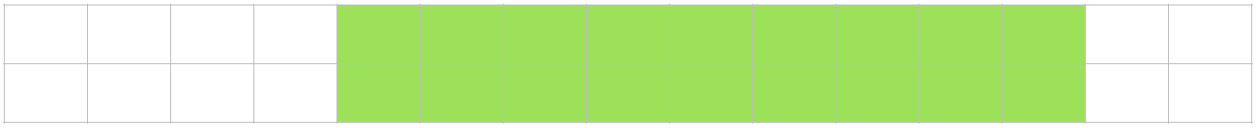
We'll start with the movement of stones because that is the easy part. In order to moved stones, we'll just call a moveRight() or moveLeft() method on all of the rows associated with that rock. We'll feed in velocity into $a0, which is the number of pixels that our rock will move left/right every time these functions are called.

The only tricky part about moving a rock is when our frog is on the rock. Luckily, because of the way that we wrote our moveRight/Left() functions, it should shift the frogs correctly on the screen. You just need to make sure that when you also update your memory/registers so that you know the correct position of the frog after you've shifted the rock.

Spawning a rock is a little more difficult, and in order to do this we've created a set of variables associated with each rock. The process of spawning a rock begins as soon as a rock is one move away from the start of its journey.

So when a rock is moving left to right with a velocity of 2 (skips two LEDS per move), then you begin to randomize. When the rock is two blocks away:





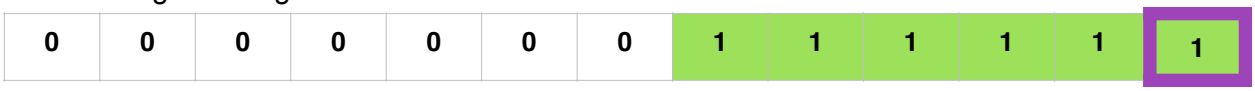
When you randomize, you pick a random number from 1 to 10. If the number happens to be 10, then you take the value in flush and begin to move that rock onto the screen. Flush will hold a rock between the size 8 and 12.

Regardless of what our boolean value **flush** equals, when shifting you will always grab the rightmost value of the array (the flush bit) and introduce that as the newest pixel in your maze. So as long as flush is false, you will just be creating more red pixels. However, when flush is true, you shift your flush array to the right by 1 every time you get the right move pixel.

After moving to the right 1:

After moving to the right 2:

After moving to the right 3:



After moving to the right 4:

After moving to the right 5:

After moving to the right 6:

After moving to the right 7:

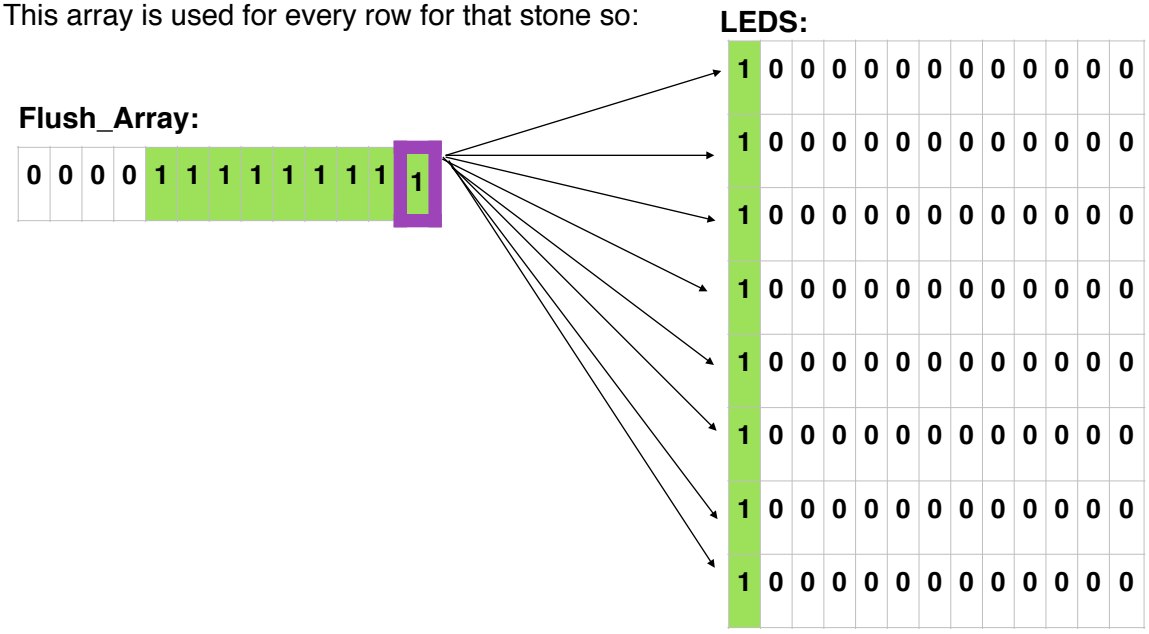
After moving to the right 8:

After moving to the right 9:

After your flush array contains only 0s, you pick a random value from 8 and 12 to be your rocks length and then set your flush array to reflect that. So if you pick '9', you'll set up your flush array to look like this:

The only caveat is that you need to make sure that your array starts as far right shifted as possible <u>while still leaving a 0 towards the rightmost position</u>

This array is used for every row for that stone so:



This is the general premise of how you'll have the loop working:

```
for (int i = 0; i <= velocity; i++):
      for each row in stone:
            grab pixel in flush_bit and introduce into the LEDs
      shift flush_array once to the right
if(flush_array is empty):
      random_size = random(8,12)
      addToFlushArray(random_size)
```