

## Six Shades (not fifty!) of Grey: PocketLab Voyager/Scratch Dice

### Introduction

This is a programming project that capitalizes on PocketLab-Scratch Integration. This project makes use of the Scratch random number block to simulate rolling an ordinary six-sided die. The six random but equally likely outcomes are mapped to sprites of six different shades of gray. Voyager's light sensor is then used to determine the value of the die's roll, mapping light sensor values to the corresponding sprite from six images of the face up on the die. A short action video of the author's solution accompanies this lesson. A snapshot from this video is shown in Figure 1.

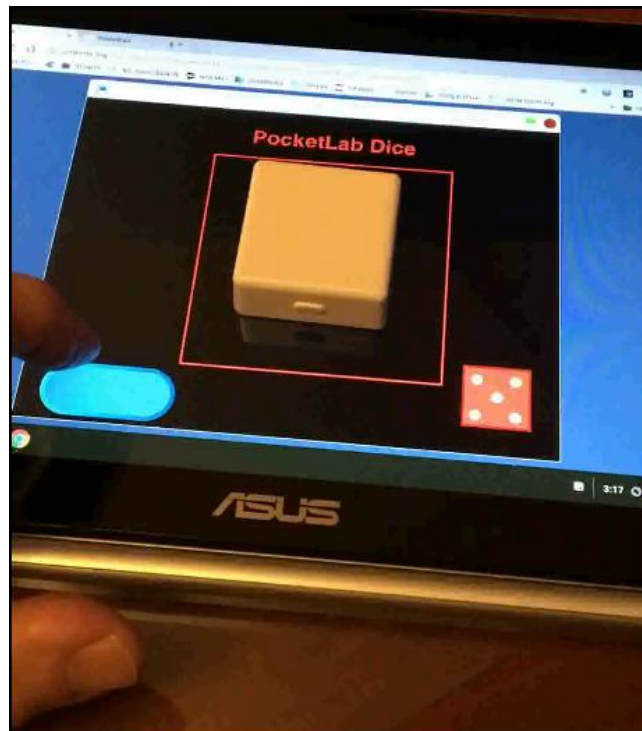


Figure 1

### The Author's Approach

Figure 2 shows the ScratchX canvas. The background is purposely black. The large red square is a sprite placed simply to let the dice player know where to place Voyager. The white circle in this square is *Ball6*, the brightest of the six Balls. All six balls are on top of one another, so *show* and *hide* blocks must be used to show the appropriate ball. *Die1* through *Die5* are images of the six sides of a die, and have been imported as sprites. A pdf file from which the images may be extracted with any imaging software has been included with this lesson. All of the die images are on top of one another, so *show* and *hide* blocks must be used to show the appropriate die. The blue *button* is intended to be clicked by the dice player in order to display a "roll" of the dice. The "PocketLab Dice" text sprite contains the main code driving the random rolls of the die.

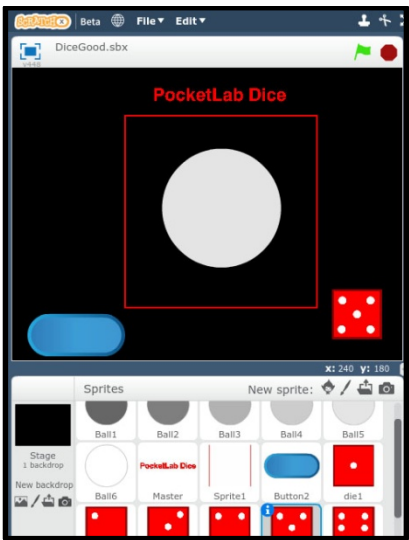


Figure 2

Figure 3 shows the ScratchX code for the Master sprite “PocketLab Dice”. This code continually drives the die roll simulation process. There is a *forever* loop. With each iteration of the loop, a random number *dieRoll* from 1 through 6 is drawn, a series of *nested if* blocks broadcasts a message to the appropriate sprite ball, and there is a two second delay.

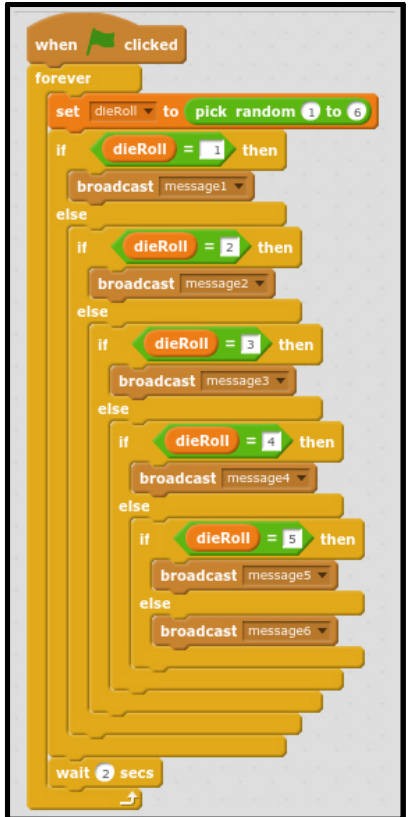


Figure 3

Figure 4 shows two event procedures for the *ball1* sprite. When the green flag is clicked to start the program, the *ball1* sprite is set to the desired size of 325% and then hidden. Upon receiving *message1* from the master sprite, *ball1* is shown for 1 second and then hidden. Similar events exist for *ball2* through *ball6*.

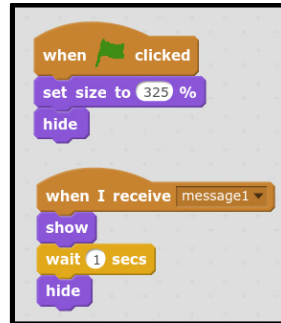


Figure 4

Figure 5 shows the click event code for the blue button. First a message *hideDie* is broadcasted to all of the ball sprites to hide them. Then there is a wait until the light sensor value is less than 15, meaning that the last die face that was on the screen has been hidden, and the black background is now visible. Since the background will be black for 1 second, there is a 1 second delay, after which the light sensor value is grabbed and stored in the variable *pDie*. Then in a series of nested if blocks, the variable *pDie* is compared to actual ranges for each of the light intensities of the six balls. As soon as the correct range is identified, a message of the form *message1x* is broadcasted to the appropriate *diex* image.

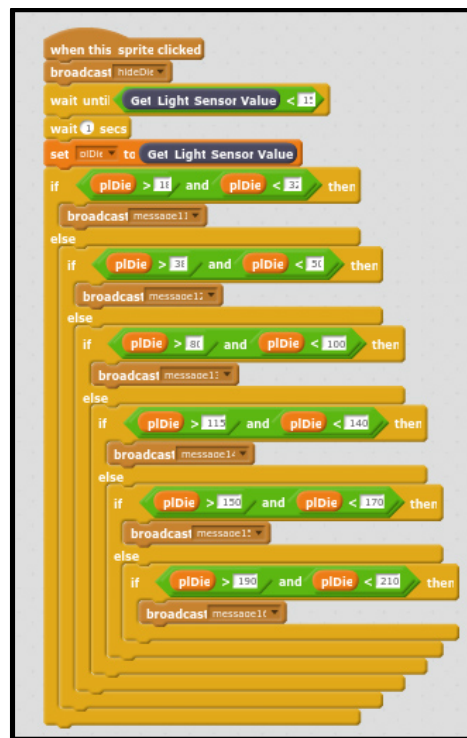


Figure 5

Figure 6 shows the three event procedures for *die1*. There are similar events for *die2* through *die6*. When the program is started, the size of the *die1* face image is set to 50% and then hidden. When *die1* receives *message 11*, it is moved to the front and shown. When *die1* receives the *hideDie* message, it is hidden.

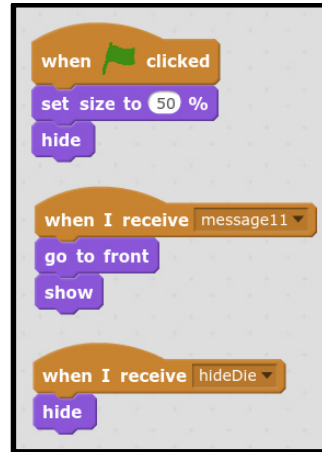


Figure 6

### Determining the Light Intensity Range for Displaying Each Die Face

Figure 7 is a screen print of approximately two minutes of simulated die roll light intensities (at 10 points/second). Each roll is separated by a black screen, which explains the zero light intensity every other second. Observation of this graph suggests that the die rolls are random and equally probable. Light intensities fall into six Lux categories in agreement with those in Figure 5. The Asus Chromebook screen was set to maximum brightness. Clearly, this kind of calibration is required for any particular Chromebook screen used with this lesson.

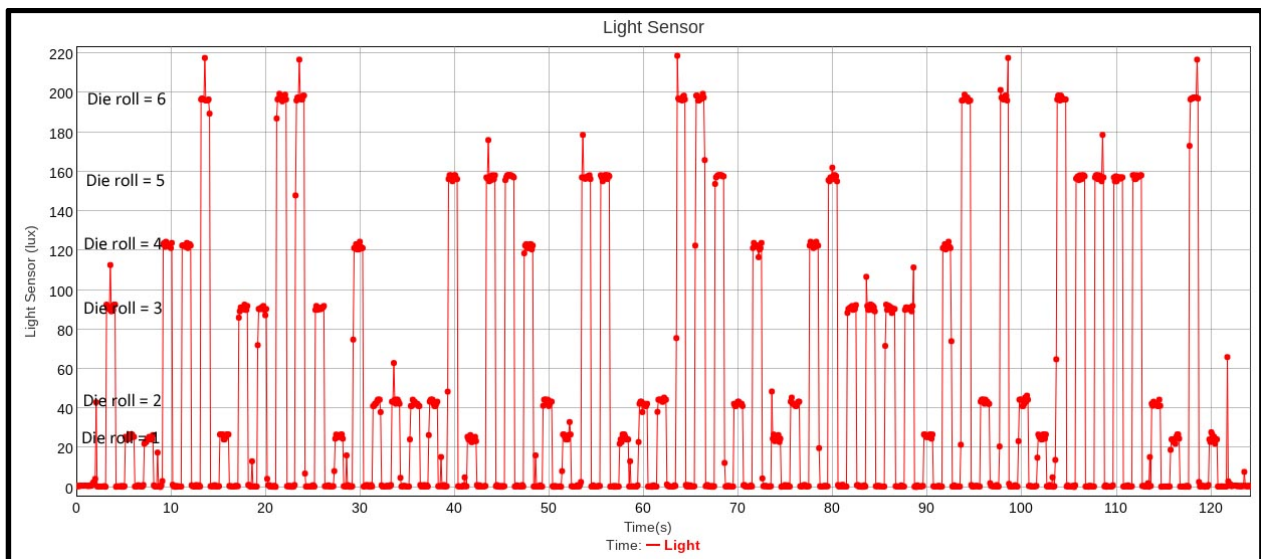


Figure 7