ETGG1801	Lab4: **if and while statements**	Points: **30**
Assigned: **9/11/2017**	Due: ~~**9/22/2017 @ 8am**~~ **9/26/2017 @ 8am**	

Note: In this lab, I want you to complete 2 of these and a part of a 3rd one. If you successfully finish all 3, you get bonus points!

1. (**15 points**) Create a new file called **monte_carlo.py**. This program should load the Great Britain image on ssugames. There are two parts:
   a. **part A:**
      i. Calculate and print precisely how many sea (green) and land (red) pixels are in the image by looking at every single pixel.
      ii. You can access a pixel's color like this[1]:

```
color = surf.get_at((x, y))
red = color[0]                  # 0 – 255
green = color[1]                # 0 – 255
blue = color[2]                 # 0 - 255
```

      iii. Print out these values:
         1. The time it took to do this operation (in seconds)
         2. The number of land pixels (I counted 150,207)
         3. The number of sea pixels (I counted 329,729)
         4. The number of pixels examined (I counted 479,938)
   b. **part B:**
      i. Perform a Monte Carlo estimate of the land / sea values. A monte carlo simulation (in this context) is when you guess a random pixel position and keep track of the contents (land or sea). Then estimate the *total* number of land pixels by taking n percent of the total image area (where n is the fraction of land pixels found so far compared to the number of pixels looked at). You can determine the error (as compared to the correct number) with this formula:

$$errorSea = \frac{|correctSea - estimatedSea|}{totalPixels}$$

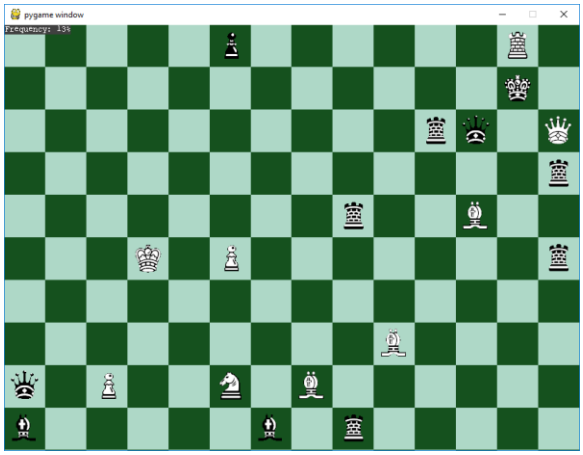
      (The vertical bars indicated absolute value – use the abs python function)
      ii. Run the monte carlo simulation until you have an error of 0.5% of less for land *and* sea. Then print out these values:
         1. The time it took to do the Monte Carlo simulation (usually this was about 1/10 the time of the full test)
         2. The estimated number of land and sea pixels and the error value (both of which should be under 0.5%)
         3. The number of pixels examined (I'd usually have to look at only about 20 pixels to determine this!!)

2. (**15 points**) Create a new file called **checkerboard.py**. This program should:
   a. Generate random values for each of these quantities (using random.xyz functions)
      i. **window_width** (800-1200)
      ii. **window_height** (600-900)
      iii. **number_of_columns** (5 – 15)
      iv. **number_of_rows** (6 – 12)

[1] color is a *tuple* in python (immutable arrays / lists in other languages). The square brackets are used to index a single element of this tuple. We'll explore this in much greater detail in section 6.

v. **colorA** (make it light-ish, but the RGB values should be random)
vi. **colorB** (similar to colorA, but dark-ish)
vii. **frequency** (0.1 – 0.9) [i.e. 10% - 90%]

b. Create a window of the chosen size.
c. Draw a checkerboard pattern. Hint: you might find the mod operator (%) helpful…
d. On approximately frequency % of the squares, draw a token, which is one of the 12 pieces on the pieces.png image on ssugames. Center this image in the square.
e. Draw the frequency you used on top of the screen.
f. Keep the window open for 3 seconds then shut down.



3. (**15 points**) Create a new file called **scroller.py**. This program should:
   a. Create an 800x600 window.
   b. Load the pond and boid image from ssugames. In code, scale it to be the same size as the window (pygame.transform.scale)
   c. Make the boid move smoothly[2] in one direction until either:
      i. It hits a wall – in this case, "bounce" in the opposite direction
      ii. 1 second passes – in this case, make the boid choose a new random direction (up, down, left, or right).
   d. Make the boid face in the direction it's moving (use the pygame.transform.rotate function)
   e. Make the background slowly scroll to the right. Use a combination of the blit command and the pygame.transform.flip (to draw a mirror image) so the background appears to be seamless.
   f. The program should run for 20 seconds and then shut down.
   g. Here's an example of my solution runn ing: **`https://youtu.be/kXJyYA3AGRs`**

[2] For now, get it to look smooth on your machine. In the next section, we'll explore a way to make things move at the same rate on any machine.