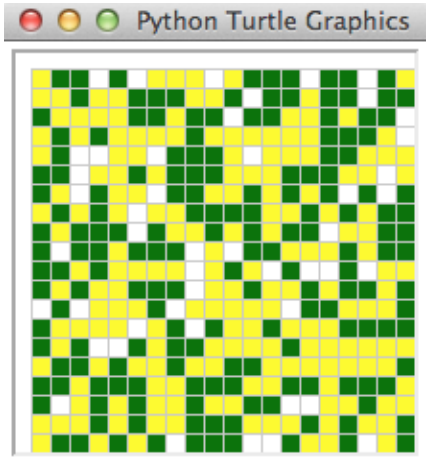
C.S. 112: Lab # 6
Schelling Lab
Due: November 9, 2015

Read Project 9.1 in the textbook. Notice that, as promised at the top of page 447, the author has stopped telling you precisely what the parameters should be. This is a purposeful decision, as when you use computers to carry out computation in real life no one will be telling you the parameters. Similarly, no one will tell you how to break down a complex problem into a sequence of functions (as described in chapter 7), but I do expect you to follow the guidelines there and introduce several helper function whenever you have code which logically belongs in a helper function.

That said, let's think a bit about *how* one might decide on an appropriate design and appropriate parameters. Clearly, this lab is meant to build on section 9.2 and the Game of Life code (from March 30) in a major way. You will need to use a turtle to draw a grid and then initialize that grid with squares of two types. Simultaneously, you will keep track of which information is in which grid square via a 2D list.

Consider introducing a function which draws all the grid lines and a separate function to initialize the grid. Initialization here means for each square in the grid you select randomly whether it is a Plain-Belly Sneetch house, a Star-Belly Sneetch house, or a vacant house. Follow the suggestion in the book to make the proportions 45%, 45%, and 10%. Please draw Plain-Belly Sneetch houses `yellow` and Star-Belly Sneetch houses `darkgreen` so that the pictures formed by all students will match. Your initialized grid will be drawn as



In order to ensure that what you draw fits on your computer screen, you should copy and paste the function `createSquares` from the the Game of Life code. This has the effect of overriding the way Python represents the turtle. Now, instead of appearing as a small arrow it will appear as an appropriately sized square. The act of drawing a colored square will involve the `stamp()` function from the `turtle` module. The variable SCALE is a global constant, i.e. a global variable which is used in several

functions and is never changed by any of them. Consider using `rows = 20, columns = 20`, and `SCALE = 10`. It's okay to model your code off the game of life code, but do **NOT** call your function `life` since that's not a descriptive name.

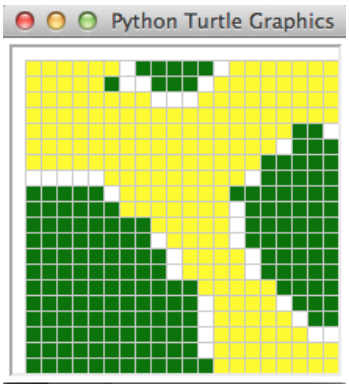
At this point you should be thinking of the grid *both* as something a turtle is drawing and as a 2D list H. You must decide what to store in the 2D list, and it will initially be filled via your initialization function. In the Game of Life code every entry was 0 or 1, and global constants were used to provide descriptive variable names (so that it is clear at a glance when 1 means ALIVE and when it's a number). You could do something similar, but you will need 3 different constant values. Warning: you can't follow the Game of Life code too closely, e.g. because its initialization function has a parameter which is the initial configuration (e.g. a glider) whereas yours determines the initial distributions of yellow, green, and white squares randomly. Your `main()` method will also require code similar to the Game of Life code, since you need to set up the screen, set initial coordinates, and hide the turtle in the same way. Consider using `screen.tracer(100)` since we'll be doing much more drawing in this lab. I do *not* want you to make the 2D list a global variable. Globals are only appropriate if they are constant. If several different functions need to access the 2D list, make it a parameter and/or return value for the functions.

The project makes it clear you need to consider several tolerance values between 0 and 1 to answer the reflection questions. In your function which does the bulk of the computation, consider making tolerance a parameter. This function will need to fill and refill a 2D list over and over again, so needs a parameter for number of rounds. When you call your function from `main` please call it with 100 rounds. The function will need a loop to repeat `rounds` many times and each time will need to iterate through every entry in the 2D list.

Each entry (r, c) looks at its 8 neighbors and determines the number k of them that belong to the other group of individuals (e.g. Star Belly Sneetches), and the total number T of non-vacant neighbors (beware: all 8 neighbors might be vacant, so $T = 0$ could occur). If the proportion k/T is larger than the tolerance threshold, the individual will move to one of the vacant squares and the square at (r, c) will become vacant. Consider introducing a helper function, like we did in the Game of Life code. After this move has been represented in the 2D list, use your turtle to draw squares of the appropriate colors in the two houses involved in the move. So that all moves are made simultaneously, make use of the `copy.deepcopy` method and only update the grid after all entries (r, c) have been considered.

Question 9.1.7. *Why do we use deepcopy rather than copy? What would go wrong if we only used copy?*

After many iterations, if segregation has occurred, you might get a picture like:



Your code should be drawing all the time and you should be able to see the green and yellow squares moving as your picture transforms from the first image above to the second. The reflection questions ask about segregation. You may consider this word to mean that there are fewer than 4 clusters in your final picture. In addition to the six reflection questions in the book, please answer:

Question 9.1.8. *Do your answers to the reflection questions depend on the size of the grid?*

To answer this, consider grid sizes with 10 rows and columns and with 30 rows and columns. When answering any of the questions you **may not** make use of online resources. In particular, for 9.1.5 you must think of an example on your own rather than googling.