Setting up Parameters

1. Open Parameters.py
2. Set whatever you want for P, N, K, E, T
3. Set IMG to be pygame.image.load(<Path to Image>)
4. Set OUTPUT_DIR to be where you want the pictures of every 1000 generations of the genetic search to be saved.
5. Set IMG_WIDTH to be the width of your image in pixels
6. Set IMG_HEIGHT to be the height of your image in pixels

By default, the Parameters are set to something that will run.

Running the Search Algorithm

Run this from command line on CAEN:

python Search.py

The program will print to stdout what the best fitness is every 1000 children.

Note on Poor Approximations

From my testing, it appears that not accounting for the alpha values in the fitness function skews the approximations. Specifically, pygame's surface.draw does not draw with alpha (http://stackoverflow.com/questions/6339057/draw-a-transparent-rectangle-in-pygame). This is what I believe is the source of the problem.

Consider this extreme example: I fill a canvas with a black rectangle with alpha 255 (fully opaque). Then, I take an exact copy of the original image pixel-by-pixel, but set all the alphas to 0 (i.e. all pixels are transparent) and overlay that on the black rectangle.

Since pygame.surface won't account for the alpha values, it'll just draw the original image on top of the black rectangle. We get a perfect fitness score for an awful black rectangle. This appears to be exactly the problem in most of the animations. The RGB values in the higher generations are close to the colors in the original image, but there's nothing accounting for the blending.

Notice that the Michigan "M" seems to be approximated better. I hypothesize that is because there is no blending needed to draw the Michigan "M". It's still a bit "muggy" though, because the alpha values are being mutated randomly when the fitness function doesn't actually keep it in check. This leads to incorrect blending.
What needs to happen is to be able to compare the blended image with the original image, but pygame doesn't seem to have this functionality.

If the mutations were ineffective, the Michigan "M" should not be approximated that well either. The mutations should at least be reasonable because the "M" shows up.

Finally, I am rather confident in the correctness of the genetic search algorithm - except for the provided fitness function.

Also see @186 on Piazza.