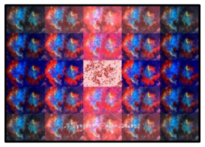
# Keegan Fontaine

Completed



Original

```
'''
  TAKES A WHILE TO LOAD!!!
  There are a log of pixels that need to be copied, and a lot of (inneficient) method calls.
  I'd like to blame the slowness on the fact that it's Python, however my poorly optimized code doesn't help.

  This is quite possibly the most disgusting, hard to read code I have ever written.
  For some reason though, this approach makes sense to me.

  It should work for any gridSize, however I haven't done extensive testing.
  If you decide to test changing it, let me know if you have any issues!

  Also, if you have any questions or concerns, please let me know.
'''


class ImgGridSquare:

  def __init__(self):
    self.gridSize = 5
    self.img = makePicture(getMediaPath("space.png"))
    self.width = min(getWidth(self.img), 200)
    self.height = min(getHeight(self.img), 140)
    self.emptyImg = makeEmptyPicture(self.width * self.gridSize, self.height * self.gridSize)
    self.signature = makePicture(getMediaPath("signature.png"))


  def changeLightness(self, emptyPx, imgPx, offset):
   setColor(emptyPx, makeColor(getRed(imgPx) + offset, getGreen(imgPx) + offset, getBlue(imgPx) + offset))


  def edgeDetect(self, emptyPx, imgPx, threshold):
    x = getX(imgPx)
    y = getY(imgPx)
    if y < self.height - 1 and x < self.width - 1:
      botrt = getPixel(self.img, x+1, y+1)
      thislum = (getRed(imgPx) + getBlue(imgPx) + getGreen(imgPx)) / 3
      brlum = (getRed(botrt) + getBlue(botrt) + getGreen(botrt)) / 3

      if abs(brlum-thislum) > threshold:
        setColor(emptyPx, black)
      elif abs(brlum-thislum) <= threshold:
        setColor(emptyPx, white)


  def colorShift(self, emptyPx, imgPx, offset):
    red = getRed(imgPx)
    blue = getBlue(imgPx)


    if (red < 63):
      red = red + red*offset

    if (red > 62 and red < 192):
      red = red + red*offset * 1.1

    if (red > 191):
      red = red + red * offset * 1.15

    setBlue(emptyPx, blue)
    setRed(emptyPx, red)


  def saturate(self, emptyPx, imgPx, offset):
    r = getRed(imgPx)
    g = getGreen(imgPx)
    b = getBlue(imgPx)

    maxColor = max(r, g, b)
    if maxColor == r:
      setColor(emptyPx, makeColor(r + offset, g - offset, b - offset))
    elif maxColor == g:
      setColor(emptyPx, makeColor(r - offset, g + offset, b - offset))
    else:
      setColor(emptyPx, makeColor(r - offset, g - offset, b + offset))
```

```
def copyPixels(self, xOffset, yOffset, imgEffectList, imgEffectArgs = None):
    for x in range(self.width):
      for y in range(self.height):
        emptyPx = getPixel(self.emptyImg, x + xOffset, y + yOffset)
        imgPx = getPixel(self.img, x, y)

        for i in range(0, len(imgEffectList)):
          # This is very confusing. It simple calls each of the image "filters" I'm passing into this one,
          # with corresponding argument values
          imgEffectList[i](emptyPx, imgPx, imgEffectArgs[i])
          imgPx = emptyPx

def sign(emptyImg, signature):
  for x in range(0, getWidth(signature)):
    for y in range(0, getHeight(signature)):
      signaturePx = getPixel(signature, x, y)

      if(getColor(signaturePx) != white):
        # the xOffset just adds a cool, weird effect to my signature
        xOffset = getWidth(emptyImg) / getWidth(signature)

        setColor(getPixel(emptyImg, x * xOffset, y + getHeight(emptyImg) - getHeight(signature)), white)


def collage():
  img = ImgGridSquare()

  for i in range(0, img.gridSize):
    for j in range(0, img.gridSize):

      '''
        I like to think of these methods not as image effects, but rather as image filters.
        When I think of it like this, It makes sense to use a list in order to collect the effects.
        I think of it as a list of image filters, and then I simply apply the filters to each image copy in
        the correct order.
        It's also easy to edit the image effects applied like this, if ever I decided I'd like to change
        things up.
      '''

      effectsList = [img.changeLightness, img.saturate, img.colorShift]
      argsList = [
        -25 * abs(i - img.gridSize / 2) + 50,
        -25 * abs(j - img.gridSize / 2) + 50,
        -0.5 * abs(i - img.gridSize / 2) + 1
      ]
      # If at the center gridpoint, add an edgeDetect to the effectsList
      if i == img.gridSize / 2 and j == img.gridSize / 2:
        effectsList.insert(0, img.edgeDetect)
        argsList.insert(0, 10)

      # Copy pixels over from my original image to my emptyImg, calling the correct effects, or "filters"
      # on each one.
      img.copyPixels(img.width * i, img.height * j, effectsList, argsList)


  sign(img.emptyImg, img.signature)
  repaint(img.emptyImg)


collage()
```