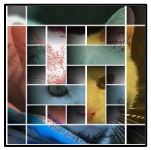
Jack Wagner

Completed



Original

```
# Name:  Jack Wagner
# Date:  10/21/2024
# Title: Cathedral of the Colorful Kitty

from jes4py import *
import math

# **** Helper Functions for Modifying the Source Picture ****
def copy(bigPic,smallPic,targetX,targetY):
    for x in range (targetX, getWidth(smallPic) + targetX):
        for y in range (targetY, getHeight(smallPic) + targetY):
            oldPixel = getPixel(smallPic, x - targetX, y - targetY)
            newPixel = getPixel(bigPic, x, y)
            color = getColor(oldPixel)
            setColor(newPixel, color)

def copyChunk(referencePic, targetPic, chunkLength, startX, startY):
    for x in range (startX, chunkLength):
        for y in range (startY, chunkLength):
            referencePixel = getPixel(referencePic, x, y)
            targetPixel = getPixel(targetPic, x, y)
            color = getColor(referencePixel)
            setColor(targetPixel, color)

def makeChunk(referencePic, targetPic, chunkLength, startX, startY):
    chunk = makeEmptyPicture(chunkLength, chunkLength)
    for x in range (chunkLength):
        for y in range (chunkLength):
            referencePixel = getPixel(referencePic, startX + x, startY + y)
            targetPixel = getPixel(chunk, x, y)
            color = getColor(referencePixel)
            setColor(targetPixel, color)
    return chunk
```

```
def blur(pic):
    blurredPic = duplicatePicture(pic)
    for x in range(1, getWidth(blurredPic) - 1):
        for y in range(1, getHeight(blurredPic) - 1):
            center = getPixel(blurredPic, x, y)
            left = getPixel(blurredPic, x - 1, y)
            right = getPixel(blurredPic, x + 1, y)
            top = getPixel(blurredPic, x, y - 1)
            bottom = getPixel(blurredPic, x, y + 1)
            newRed = (getRed(center) + getRed(left) + getRed(right) + getRed(top) + getRed(bottom)) // 5
            newGreen = (getGreen(center) + getGreen(left) + getGreen(right) + getGreen(top) + getGreen(bottom))//5
            newBlue = (getBlue(center) + getBlue(left) + getBlue(right) + getBlue(top) + getBlue(bottom)) // 5
            setColor(center, makeColor(newRed, newGreen, newBlue))

def scale(source_picture, scale):
    new_width = getWidth(source_picture) * scale
    new_height = getHeight(source_picture) * scale
    new_picture = makeEmptyPicture(new_width, new_height)
    source_x = 0
    for new_x in range(0,getWidth(new_picture)):
        source_y = 0
        for new_y in range(0,getHeight(new_picture)):
            color = getColor(getPixel(source_picture, int(source_x), int(source_y)))
            setColor(getPixel(new_picture, new_x, new_y), color)
            source_y = source_y + 1/scale
        source_x = source_x + 1/scale
    return new_picture

def changeBrightness(inputPic, scale):
    modifiedPic = duplicatePicture(inputPic)
    for varient in range(abs(scale)):
        for pixel in getPixels(modifiedPic):
            r = int(getRed(pixel)+scale)
            g = int(getGreen(pixel) + scale)
            b = int(getBlue(pixel) + scale)
            if r <= 0:
                r=0
            if g <= 0:
                g=0
            if b <= 0:
                b=0
            setColor(pixel, makeColor(r, g, b))
    return(modifiedPic)
```

```
def luminance(pixel):
    redValue = getRed(pixel)
    greenValue = getGreen(pixel)
    blueValue = getBlue(pixel)
    luminance = (redValue + greenValue + blueValue) // 3
    return luminance

def posterizeTo5Colors(sourcePicImport, c1, c2, c3, c4, c5):
    posterizedPic = duplicatePicture(sourcePicImport)
    for pixel in getPixels(posterizedPic):
        darkCounter = 0
        mediumCounter = 0
        brightCounter = 0
        luminanceValue = luminance(pixel)
        if luminanceValue < 50:
            setColor(pixel, c1)
            darkCounter+=1
        elif luminanceValue < 100:
            setColor(pixel, c2)
            mediumCounter+=1
        elif luminanceValue < 150:
            setColor(pixel, c3)
        elif luminanceValue < 200:
            setColor(pixel, c4)
        elif luminanceValue <= 255:
            setColor(pixel, c5)
    return posterizedPic

def grayScale(inputPic):
    grayscaledPic = duplicatePicture(inputPic)
    for pixel in getPixels(grayscaledPic):
        new_red = getRed(pixel) * 0.299
        new_green = getGreen(pixel) * 0.587
        new_blue = getBlue(pixel) * 0.114
        intensity = new_red + new_green + new_blue
        setColor(pixel, makeColor(intensity, intensity, intensity))
    return grayscaledPic

def addBorder(inputPic, instructions):
    modifiedPic = duplicatePicture(inputPic)
    pixels = getPixels(modifiedPic)
    picWidth = getWidth(modifiedPic)
    picHeight = getHeight(modifiedPic)
    for pixel in pixels:
        if (instructions["top"] == True) and (getY(pixel) <= (instructions["width"] - 1)): # Draw the Top Row
```

```
            setColor(pixel, instructions["color"])
        # Draw the Bottom Row
        if (instructions["bottom"] == True) and getY(pixel) >= (picHeight - instructions["width"]):
            setColor(pixel, instructions["color"])
        if (instructions["left"] == True) and (getX(pixel) <= instructions["width"]): # Draw the Left Border
            setColor(pixel, instructions["color"])
        # Draw the Right Border
        if (instructions["right"] == True) and (getX(pixel) >= picWidth - instructions["width"]):
            setColor(pixel, instructions["color"])
    return modifiedPic

def edgeDetect(inputPic, backgroundColoredPic, foregroundColoredPic, threshold):
    modifiedPic = duplicatePicture(inputPic)
    for pixel in getPixels(modifiedPic):
        x = getX(pixel)
        y = getY(pixel)
        if y < getHeight(modifiedPic) - 1 and x < getWidth(modifiedPic) - 1:
            bottomRightPixel = getPixel(modifiedPic, x+1, y+1)
            thisLuminance = luminance(pixel)
            bottomRightLuminance = luminance(bottomRightPixel)
            if abs(bottomRightLuminance - thisLuminance) > threshold:
                setColor(pixel, getColor(getPixel(foregroundColoredPic, x , y)))
            else:
                setColor(pixel, getColor(getPixel(backgroundColoredPic, x , y)))
    return modifiedPic

def sepiaTint(inputPic):
    modifiedPic = duplicatePicture(inputPic)
    grayScale(modifiedPic)
    for pixel in getPixels(modifiedPic):
        red_value = getRed(pixel)
        blue_value = getBlue(pixel)

        if red_value < 63:
            red_value = red_value * 1.1
            blue_value = blue_value * 0.9
        else:
            if red_value < 192:
                red_value = red_value * 1.15
                blue_value = blue_value * 0.85
            else:
                red_value = red_value * 1.08
                if red_value > 255:
                    red_value = 255
                blue_value = blue_value * 0.93
```

```
        setRed(pixel, red_value)
        setBlue(pixel, blue_value)
    return modifiedPic

def sepiaTintNew(inputPic):
    modifiedPic = duplicatePicture(inputPic)
    grayScale(modifiedPic)
    for pixel in getPixels(modifiedPic):
        red_value = getRed(pixel)
        blue_value = getBlue(pixel)

        if red_value < 63:
            red_value = red_value * 1.1
            blue_value = blue_value * 0.9
        elif red_value < 192:
            red_value = red_value * 1.15
            blue_value = blue_value * 0.85
        else:
            red_value = red_value * 1.08
            if red_value > 255:
                red_value = 255
            blue_value = blue_value * 0.93

        setRed(pixel, red_value)
        setBlue(pixel, blue_value)
    return modifiedPic

def tint(inputPic, r, g, b):
    tintedPic = duplicatePicture(inputPic)
    for pixel in getPixels(tintedPic):
        averageColor = (getRed(pixel) + getGreen(pixel) + getBlue(pixel)) / 3
        newRed = int(r * (getRed(pixel)/255))
        newGreen = int(g * (getGreen(pixel)/255))
        newBlue = int(b * (getBlue(pixel)/255))
        setColor(pixel, makeColor(newRed, newGreen, newBlue))
    return tintedPic

def drawShadowOnChunk(chunk, chunkLength, layerDifference, shadowStart):
    maxshadowDarkness = 0.8
    shadowDarkness = maxshadowDarkness
    shadowLength = (chunkLength * layerDifference)/8

    if shadowStart == "left":
        for shadowX in range(chunkLength):
```

```
            for shadowY in range(chunkLength):
                sourcePixel = getPixel(chunk, shadowX, shadowY)
                sourceRed = getRed(sourcePixel)
                sourceGreen = getGreen(sourcePixel)
                sourceBlue = getBlue(sourcePixel)
                newColor = makeColor(int(sourceRed * (shadowDarkness)), int(sourceGreen * (shadowDarkness)),
int(sourceBlue * (shadowDarkness)))
                setColor(sourcePixel, newColor)
            shadowDarkness = 1 - (maxshadowDarkness * (1 - shadowX / chunkLength))
    elif shadowStart == "top":
        for shadowY in range(chunkLength):
            for shadowX in range(chunkLength):
                sourcePixel = getPixel(chunk, shadowX, shadowY)
                sourceRed = getRed(sourcePixel)
                sourceGreen = getGreen(sourcePixel)
                sourceBlue = getBlue(sourcePixel)
                newColor = makeColor(int(sourceRed * (shadowDarkness)), int(sourceGreen * (shadowDarkness)),
int(sourceBlue * (shadowDarkness)))
                setColor(sourcePixel, newColor)
            # As the loop progresses, the level in whitch rgb values are scaled decreases
            shadowDarkness = 1 - (maxshadowDarkness * (1 - shadowY / chunkLength))

def addShadowToChunk(inputChunk, chunkLength, instructions, gridX, gridY):
    chunk = duplicatePicture(inputChunk)
    chunkLayer = instructions[gridX][gridY]["layer"]

    # Check Adjacent Chunks to See if a Shadow is Needed and Where
    # Checks the chunks to the left and right for possible shadow sources
    for adjacentChunkX in range(gridX - 1, gridX + 1, 2):
        if (0 <= adjacentChunkX <= 6) and (instructions[gridX-1][gridY]["border"]["right"] == True):
            adjacentLayer = instructions[gridX-1][gridY]["layer"]
            layerDifference = chunkLayer - adjacentLayer
            if layerDifference < 0:
                drawShadowOnChunk(chunk, chunkLength, layerDifference, "left")
    # Checks the chunks above and below for possible shadow sources
    for adjacentChunkY in range(gridY - 1, gridY + 1, 2):
        if (0 <= adjacentChunkY <= 6) and (instructions[gridX][gridY-1]["border"]["bottom"] == True):
            adjacentLayer = instructions[gridX][gridY-1]["layer"]
            layerDifference = chunkLayer - adjacentLayer
            if layerDifference < 0:
                drawShadowOnChunk(chunk, chunkLength, layerDifference, "top")

    return chunk
```

```
def reverseChromaKey(source, background, signatureStartingX, signatureStartingY):
    for sourcePixel in getPixels(source):
        x = getX(sourcePixel)
        y = getY(sourcePixel)
        if getRed(sourcePixel) + getGreen(sourcePixel) + getBlue(sourcePixel) // 3 > 30:
            backgroundPixel = getPixel(background, signatureStartingX + x, signatureStartingY + y)
            backgroundColor = getColor(backgroundPixel)
            setColor(sourcePixel, backgroundColor)

# **** Functions That Modify The Source Picture and Create a New Modified Version ****
def makePic1(sourcePic, r, g, b):
    pic1 = duplicatePicture(sourcePic)
    pic1 = sepiaTintNew(pic1)
    pic1 = tint(pic1, r, g, b)
    return pic1

def makePic2(sourcePic, r, g, b):
    origialPic = duplicatePicture(sourcePic)
    darkenedSourcePic = changeBrightness(sourcePic, -8)
    colorShiftedPic = tint(origialPic, r, g, b)
    lineArtPic = edgeDetect(sourcePic, darkenedSourcePic, colorShiftedPic, 5)

    return lineArtPic

def makePicPixelated(sourcePic, r, g, b):
    startingPic = duplicatePicture(sourcePic)
    scaledDownPic = scale(startingPic, 1/7)
    scaledUpPic = scale(scaledDownPic, 7)
    tintedPic = tint(scaledUpPic, r, g, b)
    return tintedPic

def makePicPatriotic(sourcePic, originalRedColor, r, g, b):
    posterizedPic = duplicatePicture(sourcePic)

    newRed = makeColor(229, 124, 108)
    darkRed = makeColor(183, 42, 44)
    white = makeColor(242, 242, 242)
    lightBlue = makeColor(94, 139, 217)
    darkBlue = makeColor(59, 75, 140)

    posterizedPic = posterizeTo5Colors(posterizedPic, darkBlue, darkRed, lightBlue, newRed, white)
    tintedPic = tint(posterizedPic, 227, 133, 108)
    return posterizedPic
```

```
def makePicGrayscaled(sourcePic):
    defaultPic = duplicatePicture(sourcePic)
    grayscaledPic = grayScale(defaultPic)
    return grayscaledPic

# **** Functions Used to Assemble the Final Collage ****
def getInstructions(sourcePic, colorsList, picsList, tintedPicsList, borderWidth, borderColor):
    strongLightChange = 5
    softLightChange = 2
    instructions = []
    for column in range(0,7):
        newColumn = []
        for row in range(0,7):
            newColumn.append({
                # Determines how high or low the layer is (0-4) (4 is the heighest aka the outer frame)
                "layer": 10,
                "border": {
                    "color": borderColor,
                    "width": borderWidth,
                    "top": True,
                    "bottom": True,
                    "left": True,
                    "right": True},
                "pic1": sourcePic})
        instructions.append(newColumn)

    # Customize the instructions to generate the propper illusion of depth
    for xCord in range(len(instructions)):
        for yCord in range(len(instructions)):
            # Config chunks occupying the edges (tinted source pic)
            if (yCord == 0):
                if (1 <= xCord <= 5):
                    instructions[xCord][yCord]["pic1"] = tintedPicsList[2]
                    instructions[xCord][yCord]["border"]["left"] = False
                    instructions[xCord][yCord]["border"]["right"] = False
                    instructions[xCord][yCord]["layer"] = 4

            elif (yCord == 6) and (1 <= xCord <= 5):
                instructions[xCord][yCord]["pic1"] = tintedPicsList[3]
                instructions[xCord][yCord]["border"]["left"] = False
                instructions[xCord][yCord]["border"]["right"] = False
                instructions[xCord][yCord]["layer"] = 4

            elif (xCord == 0) and (1 <= yCord <= 5):
                instructions[xCord][yCord]["pic1"] = tintedPicsList[0]
```

```
            instructions[xCord][yCord]["border"]["top"] = False
            instructions[xCord][yCord]["border"]["bottom"] = False
            instructions[xCord][yCord]["layer"] = 4

        elif (xCord == 6) and (1 <= yCord <= 5):
            instructions[xCord][yCord]["pic1"] = tintedPicsList[1]
            instructions[xCord][yCord]["border"]["top"] = False
            instructions[xCord][yCord]["border"]["bottom"] = False
            instructions[xCord][yCord]["layer"] = 4

        # Config the chunks containing the first pic
        if (xCord == 2) and ((1 <= yCord <= 3) or (yCord == 5)):
            instructions[xCord][yCord]["pic1"] = picsList[0]
            if 1 <= yCord <= 2:
                instructions[xCord][yCord]["border"]["bottom"] = False
            if 2 <= yCord <= 3:
                instructions[xCord][yCord]["border"]["top"] = False
        # Config the chunks containing the second pic
        if (xCord == 4) and ((3 <= yCord <= 5) or (yCord == 1)):
            instructions[xCord][yCord]["pic1"] = picsList[1]
            if 3 <= yCord <= 4:
                instructions[xCord][yCord]["border"]["bottom"] = False
            if 4 <= yCord <= 5:
                instructions[xCord][yCord]["border"]["top"] = False
        # Config the chunks containing the third pic
        if (yCord == 2) and ((3 <= xCord <= 5) or (xCord == 1)):
            instructions[xCord][yCord]["pic1"] = picsList[2]
            if 3 <= xCord <= 4:
                instructions[xCord][yCord]["border"]["right"] = False
            if 4 <= xCord <= 5:
                instructions[xCord][yCord]["border"]["left"] = False
        # Config the chunks containing the fourth pic
        if (yCord == 4) and ((1 <= xCord <= 3) or (xCord == 5)):
            instructions[xCord][yCord]["pic1"] = picsList[3]
            if 1 <= xCord <= 2:
                instructions[xCord][yCord]["border"]["right"] = False
            if 2 <= xCord <= 3:
                instructions[xCord][yCord]["border"]["left"] = False
        # configure all the inner layers
# Additional Border Instructions
instructions[1][0]["border"]["left"] = true
instructions[1][6]["border"]["left"] = true
instructions[5][0]["border"]["right"] = true
instructions[5][6]["border"]["right"] = true
instructions[0][1]["border"]["top"] = true
```

```
    instructions[6][1]["border"]["top"] = true
    instructions[0][5]["border"]["bottom"] = true
    instructions[6][5]["border"]["bottom"] = true
    # Layer Instructions
    # Layer 0 (lowest layer (aka source image))
    instructions[0][0]["layer"] = 0
    instructions[0][6]["layer"] = 0
    instructions[6][0]["layer"] = 0
    instructions[6][6]["layer"] = 0
    for xGrid in range(1,6,2):
        for yGrid in range(1,6,2):
            instructions[xGrid][yGrid]["layer"] = 0
    # Layer 1
    instructions[1][2]["layer"] = 1
    instructions[4][1]["layer"] = 1
    instructions[2][5]["layer"] = 1
    instructions[5][4]["layer"] = 1
    # Layer 2
    instructions[2][1]["layer"] = 2
    instructions[2][3]["layer"] = 2
    instructions[4][3]["layer"] = 2
    instructions[4][5]["layer"] = 2
    instructions[3][2]["layer"] = 2
    instructions[5][2]["layer"] = 2
    instructions[1][4]["layer"] = 2
    instructions[3][4]["layer"] = 2
    # Layer 3
    instructions[2][2]["layer"] = 3
    instructions[4][2]["layer"] = 3
    instructions[2][4]["layer"] = 3
    instructions[4][4]["layer"] = 3
    return instructions

def makeCollage(sourcePic, fullPicsList, colorsList, instructions):
    collage = duplicatePicture(sourcePic)
    sourceWidth = getWidth(sourcePic)
    chunkLength = int(sourceWidth / 7)

    for gridX in range(0,7):
        for gridY in range(0,7):
            referencePic = instructions[gridX][gridY]["pic1"]
            chunk = makeEmptyPicture(chunkLength, chunkLength)
            collageStartX = gridX * chunkLength
            collageStartY = gridY * chunkLength
```

```
            chunk = makeChunk(referencePic, chunk, chunkLength, collageStartX, collageStartY)
            chunk = addShadowToChunk(chunk, chunkLength, instructions, gridX, gridY)
            chunk = addBorder(chunk, instructions[gridX][gridY]["border"])

            copy(collage, chunk, collageStartX, collageStartY)

    return collage

# **** Final Main Function To Assemble and Render the Collage ****
def collage():
    setMediaPath() # Path to project folder
    sourcePic = makePicture(getMediaPath("cat.jpg"))
    sourceWidth = getWidth(sourcePic)
    sourceHeight = getHeight(sourcePic)
    borderWidth = 3
    borderColor = white

    # Create the 4 main colors
    color1 = makeColor(137, 196, 244)
    color2 = makeColor(128, 197, 183)
    color3 = makeColor(239, 202, 93)
    color4 = makeColor(227, 133, 108)

    # Color Shifted Pictures
    coloredPic1 = tint(duplicatePicture(sourcePic), 137, 196, 244)
    coloredPic2 = tint(duplicatePicture(sourcePic), 128, 197, 183)
    coloredPic3 = tint(duplicatePicture(sourcePic), 239, 202, 93)
    coloredPic4 = tint(duplicatePicture(sourcePic), 227, 133, 108)

    # Modified Pictures
    pic1 = makePic1(sourcePic, 137, 196, 244)
    pic2 = makePic2(sourcePic, 128, 197, 183)
    pixelatedPic = makePicPixelated(sourcePic, 239, 202, 93)
    patrioticPic = makePicPatriotic(sourcePic, color4, 227, 133, 108)
    grayscaledPic = makePicGrayscaled(sourcePic)

    # Signature
    signature = scale(makePicture(getMediaPath("signature.jpg")), 0.4)
    signatureStartingX = sourceWidth - sourceWidth // 8 - getWidth(signature)
    signatureStartingY = sourceHeight - getHeight(signature)

    # Assemble Lists For All of The Modified Pictures Along With the Color Pallete
    colorsList = [color1, color2, color3, color4]
    picsList = [patrioticPic, pixelatedPic, pic2, pic1]
    tintedPicsList = [coloredPic4, coloredPic3, coloredPic2, coloredPic1]
```

```
# Assemble the Final Product
instructions = getInstructions(sourcePic, colorsList, picsList, tintedPicsList, borderWidth, borderColor)
baseCollage = makeCollage(sourcePic, picsList, colorsList, instructions)
expamdedCollage = makeEmptyPicture(sourceWidth + borderWidth*2, sourceWidth + borderWidth*2, borderColor)
copy(expamdedCollage, baseCollage, borderWidth, borderWidth)
reverseChromaKey(signature, expamdedCollage, signatureStartingX, signatureStartingY)
copy(expamdedCollage, signature, signatureStartingX, signatureStartingY)

explore(expamdedCollage)
```