CPS109 Assignment #1

Owen Goodwin (500909196)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **The Original Images** | | | | | |
|  |  |  |  |  |  |
| cityhall | mario | grasshopper | bup | dog | bunny |

|  |  |  |
| --- | --- | --- |
| **Function** | **Given Parameters** | **Resulting Image** |
| dougFord(img)  where: *img* is the given image | cityhall |  |
| reduce(img)  where: *img* is the given image | bup | (the original image is 128x128, this one is 64x64) |
| nightAndDay(img)  where: *img* is the given image | dog |  |
| negateHalf(img)  where: *img* is the given image | dog |  |
| shift(img,factor)  where: *img* is the given image  *factor* is the number of pixels to shift he image by | bup,24 |  |
| addColour(img)  where: *img* is the given image | mario |  |
| diagonalSplit(img)  where: *img* is the given image | bunny |  |
| combine(img1,img2)  where: *img1 and img2* are the given images | cityhall,mario |  |
| fall(img)  where: *img* is the given image | grasshopper |  |
| greyBox(img,startX,endX,startY,endY)  where: *img* is the given image  *startX* is the left bound of the box  *endX* is the right bound of the box  *startY* is the top bound of the box  *endY* is the bottom bound of the box | dog,100,300,100,300 |  |
| styleBorder(img,col)  where: *img* is the given image  *col* is the colour for the border | cityhall,red |  |
| styleBorder2(img,col,rad)  where: *img* is the given image  *col* is the colour for the border  *rad* is the radius of the circles | cityhall,yellow,45  note: this was done on top of the previous output image |  |
| triColour(img)  where: *img* is the given image | bunny |  |
| greyHare(img)  where: *img* is the given image | bunny |  |
| rot90CCW(img)  where: *img* is the given image | mario |  |
| flipBox(img,startX,endX,startY,endY)  where: *img* is the given image  *startX* is the left bound of the box  *endX* is the right bound of the box  *startY* is the top bound of the box  *endY* is the bottom bound of the box | grasshopper,100,400,100,400 |  |
| recursiveRed(img,startX,endX,startY,endY)  where: *img* is the given image  *startX* is the left bound of the box  *endX* is the right bound of the box  *startY* is the top bound of the box  *endY* is the bottom bound of the box | mario,0,479,0,852 |  |
| subtractCol(img,r,g,b)  where: *img* is the given image  *r, g,* and *b* are the components of the colour to subtract | grasshopper,128,0,128 |  |
| triColour2(img,factor)  where: *img* is the given image  *factor* is the number of pixels to alter before switching methods | bunny,14 |  |

**Source:**

Full function descriptions and documentation is contained within.

cityhall = makePicture("C:\\Users\\Owen\\Documents\\Uni\\CPS109\\cityhall-min.jpg")

mario = makePicture("C:\\Users\\Owen\\Documents\\Uni\\CPS109\\mario-min.jpg")

grasshopper = makePicture("C:\\Users\\Owen\\Documents\\Uni\\CPS109\\grasshopper-min.jpg")

bup = makePicture("C:\\Users\\Owen\\Documents\\Uni\\CPS109\\bup.png")

dog = makePicture("C:\\Users\\Owen\\Documents\\Uni\\CPS109\\dog-min.jpg")

bunny = makePicture("C:\\Users\\Owen\\Documents\\Uni\\CPS109\\bunny-min.jpg")

# This function will cut City Hall in half. Why does that sound familiar?

def dougFord(img):

skyCols = []#Initialize an empty array

for i in range(getWidth(img)-1):#Loop through each column of the image...

skyCols.append(getColor(getPixel(img, i, 0)))#...adding the color of each pixel in the first 10 rows of that colum to that array.

skyCols.append(getColor(getPixel(img, i, 1)))#This is under the assumption that the top bit of the image is sky.

skyCols.append(getColor(getPixel(img, i, 2)))

skyCols.append(getColor(getPixel(img, i, 3)))

skyCols.append(getColor(getPixel(img, i, 4)))

skyCols.append(getColor(getPixel(img, i, 5)))

skyCols.append(getColor(getPixel(img, i, 6)))

skyCols.append(getColor(getPixel(img, i, 7)))

skyCols.append(getColor(getPixel(img, i, 8)))

skyCols.append(getColor(getPixel(img, i, 9)))

n = 0#Counter var

for x in range(getWidth(img)/2):#Loop through each column in the left half of the image

for y in range(getHeight(img)-1):#Loop through each row in the column

setColor(getPixel(img,x,y), skyCols[n])#Set the colour of the current pixel to one of the colours we collected in our array

n = n+2#Increase our counter by 2

if n>=len(skyCols):#If our counter is too high...

n = n-len(skyCols)#Loop back around

return img

#This function will reduce the size of an image by half

#Ex. If the input image is 128x128, the output image will be 64x64

def reduce(img):

xOld = yOld = xNew = yNew = 0#These represent our current x-y coords in both the new and old images. We start them at 0

newimg = makeEmptyPicture(getWidth(img)/2, getHeight(img)/2)#Create a new empty image, half the size of the old one

while(true):#infinite loop

px1 = getPixel(img, xOld,yOld)#Here we grab 4 pixels, like this....

px2 = getPixel(img, xOld+1,yOld)# px1|px2

px3 = getPixel(img, xOld,yOld+1)# ---|---

px4 = getPixel(img, xOld+1,yOld+1)# px3|px4

r = (getRed(px1) + getRed(px2) + getRed(px3) + getRed(px4))/4#Take the average red value of the 4 pixels

g = (getGreen(px1) + getGreen(px2) + getGreen(px3) + getGreen(px4))/4#Do the same with the green and blue values

b = (getBlue(px1) + getBlue(px2) + getBlue(px3) + getBlue(px4))/4

setColor(getPixel(newimg, xNew, yNew), makeColor(r,g,b))#Set the colour of the current pixel in the new image to a colour created from the average values just taken

xOld = xOld+2#Increase our x-value for the old picture by 2, since we already took care of the pixel beside the old x-value

if(xOld >= getWidth(img)-1):#If that new x-value is beyond the width of the old image...

xOld = 1#Reset to 1. This is because if we reset to 0, our image will come out slanted, and while that may look cool it's not what we want.

yOld = yOld+2#Increase current y-value by 2, for the same reason we do it for the x-value

xNew = xNew+1#Add one to our x-value in the new image

if(xNew >= getWidth(newimg)-1):#Again, if we've gone beyond the bounds of the image...

xNew=0#reset to 0

yNew=yNew+1#go to the next row

if(yOld >= getHeight(img) or yNew >=getHeight(newimg)):#Check if we've gone beyond the height of either image, which would indicate that we're done here

break#end the loop

return newimg

#This function will alternate setting a pixel either lighter or darker, resulting in something quite ugly

def nightAndDay(img):

darken = true#Start off by darkening the pixel. If this is false, we lighten the pixel

for px in getPixels(img):#Loop through each pixel

if darken:#If we're supposed to darken it...

setColor(px, makeDarker(getColor(px)))#...darken it

darken = false#...make the next one lighter

else:#Otherwise, we're supposed to make it lighter...

setColor(px, makeLighter(getColor(px)))#...so do that

darken = true#Make the next one darker

for px in getPixels(img):#Here I'm repeating the same loop simply to make the effect a bit more obvious

if darken:

setColor(px, makeDarker(getColor(px)))

darken = false

else:

setColor(px, makeLighter(getColor(px)))

darken = true

return img

#Sort of similar to the last one, here we negate every other pixel for a true monstrosity

def negateHalf(img):

negate = true#Start by negating

for px in getPixels(img):#Loop through each pixel

if negate:#If we're supposed to negate this pixel...

setColor(px, makeColor(255-getRed(px), 255-getGreen(px), 255-getBlue(px)))#then do it

negate = false#don't negate the next one

else:#Otherwise we're not negating this pixel

negate = true#just chill and negate the next one

return img

#This function will shift an image horizontally by a specified number of pixels ('factor')

def shift(img, factor):

newImg = makeEmptyPicture(getWidth(img),getHeight(img))#Create a new image to work with

pixels = []#Empty array to store all of our pixels in

for px in getPixels(img):#loop through each pixel

pixels.append(px)#add it to the array

#yes, I am totally aware that for loop was totally redundant. I did it anyways.

for i in range(len(pixels)-1):#Loop through each index in our array

j = i + factor#find the pixel x units ahead of this one (if factor is 5, find the pixel 5 ahead of this one)

if(j>len(pixels)-1):#If we go beyond the bounds of the array...

j=j-len(pixels)-1#Loop back around to the beginning

temp = pixels[i]#Swap the two pixels

pixels[i] = pixels[j]

pixels[j] = temp

i = 0#counter var

for px in getPixels(newImg):#Loop through each pixel in our blank new image

setColor(px, getColor(pixels[i]))#Get the colour from the current index in the array, and set it to the current pixel

i = i+1#Increase our counter

return newImg

#This function will check if the sum of the R,G,and B values of a pixel is over a specified value, and if it is, turn that pixel green.

def sum(img, value):

for px in getPixels(img):#Loop through each pixel

if(getRed(px)+getGreen(px)+getBlue(px)>value):#if the sum of r+g+b is greater than the specified value,

setColor(px, makeColor(0,255,0))#Set the colour of the current pixel to green

return img

#This function will add colour to a pencil drawing of Mario

def addColour(img):

for px in getPixels(img):

if getX(px) < 250 and getX(px) > 200 and getY(px) < 630 and getY(px) > 360 and getRed(px) > 100 and getGreen(px) > 100 and getBlue(px) > 100 and getRed(px) < 150 and getGreen(px) < 150 and getBlue(px) < 150:

setColor(px, makeColor(0,0,255))

elif getX(px) < 310 and getY(px) < 630 and getY(px) > 385 and getRed(px) > 100 and getGreen(px) > 100 and getBlue(px) > 100 and getRed(px) < 150 and getGreen(px) < 150 and getBlue(px) < 150:

setColor(px, makeColor(0,0,255))

elif getX(px) < 310 and getX(px) > 170 and getY(px) < 630 and getY(px) > 475 and getRed(px) > 100 and getGreen(px) > 100 and getBlue(px) > 100 and getRed(px) < 175 and getGreen(px) < 175 and getBlue(px) < 175:

setColor(px, makeColor(0,0,255))

elif getY(px) < 360 and getY(px) > 170 and getRed(px) > 90 and getGreen(px) > 90 and getBlue(px) > 90 and getRed(px) < 140 and getGreen(px) < 140 and getBlue(px) < 140:

setColor(px, makeColor(131,92,59))

elif getY(px) < 700 and getY(px) > 610 and getRed(px) > 90 and getGreen(px) > 90 and getBlue(px) > 90 and getRed(px) < 140 and getGreen(px) < 140 and getBlue(px) < 140:

setColor(px, makeColor(131,92,59))

return(img)

#This is a bit of an odd one. I can't really put into words what exactly the purpose of it is.

def diagonalSplit(img):

newImg = img#Create a copy of the input image

i = 1#Counter var

x = getWidth(img)-i#Our x will start at the far right side of the image, since we start i at 1. As i increases, our starting x position will decrease.

y = 0#Starting at the top of the image

while(true):#Infinte loop

px1 = getPixelAt(img, x, y)#Get the pixel at the current x-y coord

px2 = getPixelAt(newImg, y, x)#swap the x-y coord, and get the pixel at that new location in the copy of the image

setColor(px2, getColor(px1))#Set the color of the pixel from the copy to the color of the pixel form the original

x = x-1#Move left 1 pixel

if x<0:#If we've gone beyond the left edge of the image...

i = i+1#Increase our counter

x = getWidth(img)-i#Get our new starting x position

y = y+1#Move down to the next row

if y == getHeight(img) or x<0:#If we've reached the bottom of the image, or our starting x position is less than 0, we're done!

break#Get outta here

return newImg

#Here we will combine two images

def combine(img1, img2):

maxW = getWidth(img2)#We set the maximum width and height to the width and height of the second image...

maxH = getWidth(img2)

if getWidth(img1)>getWidth(img2):#...however we check if the width and height of the first image are greater and adjust our maximums accordingly.

maxW = getWidth(img1)

if getHeight(img1)>getHeight(img2):

maxH = getHeight(img1)

imgOut = makeEmptyPicture(maxW,maxH)#Create a new blank image with the maximum dimensions, this will ensure both images can fit on our output image

x = 0#Starting at position (0,0)

y = 0

while(true):#Infinite loop

if (x <= getWidth(img1)-1) and (y <= getHeight(img1)-1) :#Check if we are in the bounds of image 1

setColor(getPixelAt(imgOut, x, y), getColor(getPixelAt(img1, x,y)))#if so, set the pixel at (x,y) to the colour at the same position in image 1

else:

setColor(getPixelAt(imgOut, x, y), getColor(getPixelAt(img2, x,y)))#if we are outside the bounds of image 1, we resort to using the colour at that position in image 2

x = x+1#Move right one pixel

if x > maxW-1:#Check if we have gone beyond the bounds of the output image

x = 0#If so, reset our x to 0

y = y+1#and move down one row

if y > maxH-1:#check if moving down 1 row put us outside again

break#If so, we're done! Woohoo!

if (x <= getWidth(img2)-1) and (y <= getHeight(img2)-1):#Check if we're in the bounds of image 2 now

setColor(getPixelAt(imgOut, x, y), getColor(getPixelAt(img2, x,y)))#if so, set the color at (x,y) in the output image to the color of the same position in image 2

else:

setColor(getPixelAt(imgOut, x, y), getColor(getPixelAt(img1, x,y)))#Otherwise, we resort to taking the color from image 1

x = x+1#move right one pixel again

if x > maxW-1:#Same out-of-bounds checks as above

x = 0

y = y+1

if y > maxH-1:

break

return imgOut

#Assuming the input has some green leaves in it, this function will result in some lovely fall colours. Maybe.

def fall(img):

for px in getPixels(img):#Loop through each pixel

if(getGreen(px)>getRed(px) and getGreen(px)>getBlue(px)):#Check if green is the dominant colour in that pixel

setColor(px, makeColor(getGreen(px)+5,getRed(px),getBlue(px)))#if so, we swap the red and green, and add 5 to the new red

return img

#This function takes 4 ints as input, which we will use to define a rectangle

def greyBox(img, startX, endX, startY, endY):

for px in getPixels(img):#Loop through each pixel

if getX(px) in range(startX,endX) and getY(px) in range(startY,endY):#Check if we are in the bounds of that arbitrary rectangle

i = (getRed(px)+getGreen(px)+getBlue(px))/3#If so, we greyscale that pixel

setColor(px,makeColor(i,i,i))#This calculation is pretty standard, nothing fancy

return img

#This function will make a border of little circles around the edge of an image, in a specified colour.

def styleBorder(img,col):

x = 0#starting at the top left of the image

y = 0

while(y<getHeight(img)):#Loop through each y value top to bottom

addOvalFilled(img,x,y,10,10,col)#Add a circle (radius 10)

y = y+10#Move down 10 pixels for the next one

while(x<getWidth(img)):#Loop through each x value left to right

addOvalFilled(img,x,y-10,10,10,col)#add a circle

x = x+10#Move right 10 pixels

while(y>0):#Loop through each y value bottom to top

addOvalFilled(img,x-10,y,10,10,col)#I think you know how this works by now

y = y-10

while(x>0):#Loop through each x value right to left

addOvalFilled(img,x,y,10,10,col)#If you haven't realized by now that we are making our way around the edges of the image counterclockwise, adding circles as we go, I'm ashamed of you.

x = x-10

return img

#This function is IDENTICAL to the previous one, with the only exception being that you now specify the radius ('rad'), instead of us setting it to 10 for you. Documentation not needed.

def styleBorder2(img,col,rad):

x = 0

y = 0

while(y<getHeight(img)):

addOvalFilled(img,x,y,rad,rad,col)

y = y+rad

while(x<getWidth(img)):

addOvalFilled(img,x,y-rad,rad,rad,col)

x = x+rad

while(y>0):

addOvalFilled(img,x-rad,y,rad,rad,col)

y = y-rad

while(x>0):

addOvalFilled(img,x,y,rad,rad,col)

x = x-rad

return img

#The name of this function is a lie. Basically it checks if any of the components (r, g, b) are greater than the other two, and if so, sets the pixel to that color.

#But what if they're equal? ????????????

def triColour(img):

for px in getPixels(img):#Loop through each pixel

if(getRed(px)>getGreen(px) and getRed(px)>getBlue(px)):#if red is the dominant color...

setColor(px,makeColor(255,0,0))#set it to red

elif(getGreen(px)>getRed(px) and getGreen(px)>getBlue(px)):#if green is the dominant color...

setColor(px,makeColor(0,255,0))#set it to green

elif(getBlue(px)>getRed(px) and getBlue(px)>getGreen(px)):#if blue is the dominant color...

setColor(px,makeColor(0,0,255))#set it to purple

#kidding

return img

#This will change Archie's ginger fur to grey

def greyHare(img):

for px in getPixels(img):#Loop through each pixel

if 100<getRed(px)<200 and 85<getGreen(px)<190 and 60<getBlue(px)<175:#Check if the components are within the rnage that would signify the colour of his fur

i = (getRed(px)+getGreen(px)+getBlue(px))/3#greyscale that pixel

setColor(px,makeColor(i,i,i))

return img

#This will rotate an image 90 degrees counterclockwise. Rotating it the other way would require just a few simple changes to this

def rot90CCW(img):

imgOut = makeEmptyPicture(getHeight(img),getWidth(img))#Make an empty picture we will fill later

x = y = 0#start at (0,0)

while(y<getHeight(img)):#Loop

setColor(getPixelAt(imgOut,y,x),getColor(getPixelAt(img,x,y)))#set the color at the flipped coordinates in the empty image to the color at the proper coordinates in the original image

x = x+1#move to the next pixel right

if x>=getWidth(img)-1:#Check if we've gone out of bounds

x = 0#if so, reset to 0

y = y+1#and move down one row

return imgOut

#This one allows the user to specify a rectangle like the greyscale one from earlier, but now we're gonna fli the box's contents horizontally

def flipBox(img, startX,endX,startY,endY):

imgOut = img#Copy of the original image to work with

for px in getPixels(img):#Loop through each pixel

if(getX(px) in range(startX,endX) and getY(px) in range(startY,endY)):#Check if the current pixel is within the box

setColor(getPixelAt(imgOut,endX-getX(px)+startX,getY(px)),getColor(px))#Set the pixel (in the output image) at the opposite position of the current one (in the input image) to the color of the current one

#|12345678|

#Looking at the line above, we would take the color of pixel #1 in the original image and apply it to pixel #8 in the output image

#Afterwards, we would repeat the process, applying the color of #2 to #7, and so forth

return imgOut

#Who doesn't love recursion?

#Who doesn't love recursion?

#Who doesn't love recursion?

#Who doesn't love recursion?

#Who doesn't love recursion?

#Who doesn't love recursion?

#Base case: This function takes 4 ints in order to define a rectangle, which we will make redder recursively because why not

def recursiveRed(img,startX,endX,startY,endY):

for px in getPixels(img):#Loop through each pixel

if (getX(px) in range(startX,endX) and getY(px) in range(startY,endY)):#Check if the pixel is within the bounds of the box

setColor(px, makeColor(getRed(px)+20,getGreen(px),getBlue(px)))#Increase the pixel's r value by 20

if startX==endX==startY==endY:#Base case: our box has become 0x0

return img#We're done

else:#keep going!

return recursiveRed(img,startX,endX/2,startY,endY/2)#Do it again with a box half the size

#This takes in a color as r, g, and b values, and subtracts that from each pixel.

def subtractCol(img,r,g,b):

for px in getPixels(img):#Loop through each pixel

setColor(px,makeColor(getRed(px)-r,getGreen(px)-g,getBlue(px)-b))#Subtract the given rgb values from the pixel's rgb values

return img

#No relation to the first triColour. Basically, we go through each pixe and switch up the pixel's rgb values. The way this is done is influenced by the int 'factor'

def triColour2(img, factor):

i = 0#Counter

phase = 0#Int that determines the order we switch the r, g and b

for px in getPixels(img):#Loop through each pixel

if phase == 0:#Phase = 0: B,R,G

setColor(px, makeColor(getBlue(px),getRed(px),getGreen(px)))

elif phase == 1:#phase = 1: G,R,B

setColor(px, makeColor(getGreen(px),getRed(px),getBlue(px)))

elif phase == 2:#Phase = 2: B,G,R

setColor(px, makeColor(getBlue(px),getGreen(px),getRed(px)))

elif phase == 3:#Phase = 3: G,B,R

setColor(px, makeColor(getGreen(px),getBlue(px),getRed(px)))

elif phase == 4:#Phase = 4: R,B,G

setColor(px, makeColor(getRed(px),getBlue(px),getGreen(px)))

i = i+1#Increase our counter each time

if i%factor==0:#Check if the counter is divisible by the factor

phase = phase+1#If so, use the next phase

if phase==5:#if phase = 5, loop back to 0

phase = 0

#Checking for divisibility means that if the factor is 62, we will change phases every 62 pixels

return img