COMP120: Creative Computing: Tinkering

2: Tinkering Graphics II

Learning Outcomes

- **Explain how** conditional logic can manipulate the output of a computer program
- **Apply** mathematical knowledge to **write** computer programs that manipulate pixels in a surface
- **Trace** existing computer programs

Distance Between Colors

Sometimes we need to measure when something is `close enough´:

- Distance between two points in the Cartesian coordinate system:
- $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- Distance between two colours in the RGB colour representation system:
- $\sqrt{(red_1 - red_2)^2 + (green_1 - green_2)^2 + (blue_1 - blue_2)^2}$

Activity #1: Color Distance

In pairs:

- Setup a basic project in Visual Studio
- Use the distance equation from the previous slide to write a function which accepts a two colours and returns the distance
- Test your solution
- Then, post your solution on Slack

Numeric Return Values

```
public static int GetDistance(Color first, Color second)
{
    int redDifference;
    int greenDifference;
    int blueDifference;

    redDifference = first.R - second.R;
    greenDifference = first.G - second.G;
    blueDifference = first.B - second.B;

    return redDifference * redDifference +
    greenDifference * greenDifference +
255blueDifference * blueDifference;
}
```

Expected Output: Color Distance

You can use predefined color names with this method:

```
Color Color1 = Color.FromKnownColor(KnownColor.Pink);
```

Expected Values:

```
Console.WriteLine (GetDistance(WHITE, BLACK));
- 195075
Console.WriteLine (GetDistance(WHITE, PINK));
- 6673
Console.WriteLine (GetDistance(BLACK, PINK));
- 143098
Console.WriteLine (GetDistance(MAGENTA DARKGOLDENROD));
- 82533
```

NOTE - You need to create the color objects and pass them into the function, don't just use a colour in capitals. just for illustration.

Activity #2: Colour Tolerance

In pairs:

- Setup a basic project in PyGame
- Implement the function `closeEnough((colour, colour), tolerance)` that returns a boolean value
- Test your solution
- Then, post your solution on Slack

Boolean Return Values

```
def close_enough(colour_base, colour_comparitor):
    if distance(colour_base, colour_comparitor) < 50.0:
        return True
    else:
        return False
```

Note: This source code excerpt will not work in PyGame.

Tolerance-based Pixel Manipulation

```
def turnRed():
  brown = makeColor(42,25,15)
  file="/Users/guzdial/Desktop/mediasources/katieFancy.jpg"
  picture=loadPicture(file)
  for pixel in getPixels(picture):
    color = getColor(pixel)
    if distance(color, BROWN) < 50.0:
      red=getRed(pixel)*2
      green=getGreen(pixel)
      blue=getBlue(pixel)
      setColor(pixel,makeColor(red,green,blue))
  return picture
```

Note: This source code excerpt will not work in PyGame.

Red Eye

- When the flash of the camera catches the eye just right (especially with light colored eyes), we get bounce back from the back of the retina.
- This results in `red eye'
- We can replace the "red" with a color of our choosing
- First, we figure out where the eyes are (x,y)


C:\Documents and Settings\Mark Guz...
Zoom
X: 109 Y: 91 R: 129 G: 97 B: 76

Activity #3: Red Eye

In pairs:

- Setup a basic project in PyGame
- Refer to the following documentation
  - `http://www.pygame.org/docs/ref/rect.html`
- Implement the function: `removeRedEye(picture, area, colour)`
- Test your solution
- Then, post your solution on Slack


C:\Documents and Settings\Mark Guz...
Zoom
X: 183 Y: 97 R: 0 G: 0 B: 0

Posterization

- Posterization is simply reducing the number of colours in an image
- We look for a range of colours, then map them to a single colour, e.g:
  - If red is between 63 and 128, set it to 95
  - If green is less than 64, set it to 31
- The end result is that a bunch of different colours, get set to a few colours
- Beware of naive solutions with a large number of `if' statements

Zoom

X: 0 Y: 0

R: 155 G: 153 B: 94 Color at location:

Zoom

X: 0 Y: 0

R: 159 G: 159 B: 95 Color at location:

Calculating Luminance in RGB

To do this, we may need to determine the luminance of a pixel:

- Luminance is the overall brightness of a pixel
- In RGB, it is the *mean* average value of each component:
  - $lum = (red + green + blue)/3$

Activity #5: Black and White

In pairs:

- Setup a basic project in PyGame
- Refer to the following documentation
- Implement the function: `makeGreyscale(picture, colourCount)`
- Test your solution
- Then, post your solution on Slack

Source Code: Black and White

```
def blackAndWhitePosterize(picture):
  for pixel in getPixels(picture):
    red = getRed(pixel)
    green = getGreen(pixel)
    blue = getBlue(pixel)
    luminance = (red + green + blue) / 3
    if luminance < 64:
      setColor(pixel, BLACK)
    else:
      setColor(pixel, WHITE)
```

Sepia Tone

- Pictures that are sepia-toned have a yellowish tint to them that we associate with older pictures.
- It's not directly a matter of simply increasing the yellow in the picture, because it's not a one-to-one correspondence:
  - Instead, colors in different ranges get mapped to other colours.
  - We can create such a mapping using IF statements
- The end result is that a bunch of different colours, get set to a few colours
- Beware of naive solutions with a large number of `if' statements

Sepia Tone

- First, we're calling greyScaleNew (the one with weights).
- We then manipulate the red (increasing) and the blue (decreasing) channels to bring out more yellows and oranges.
  - It's perfectly okay to have one function calling another.
  - Why are we doing the comparisons on the red? Why not? After greyscale conversion, all channels are the same!
- The end result is that a bunch of different colours, get set to a few colours
- Why these values? Trial-and-error: Tinker the values!

Source Code: Sepia (1)

```
def sepiaTint(picture):
  #Convert image to greyscale
  makeGreyscale(picture)

  #loop through picture to tint pixels
  for p in getPixels(picture):
    red = getRed(p)
    blue = getBlue(p)

    #tint shadows
    if (red < 63):
      red = red*1.1
      blue = blue*0.9
...
```

Note: This source code excerpt will not work in PyGame.

Source Code: Sepia (2)

```
...
    #tint midtones
     if (red > 62 and red < 192):
       red = red*1.15
       blue = blue*0.85

     #tint highlights
     if (red > 191):
       red = red*1.08
       if (red > 255):
         red = 255

       blue = blue*0.93

     #set the new color values
     setBlue(p, blue)
     setRed(p, red)
```

Note: This source code excerpt will not work in PyGame.

Activity #6: Sepia Tone

In pairs:

- Setup a basic project in PyGame
- Refer to the following documentation
- Refactor the function: `sepiaTint(picture)` to use constants rather than literals
- Tinker with the values of the constants to test your solution
- Then, post your solution on Slack