COMP120: Creative Computing

3: Tinkering Graphics II

Learning Outcomes

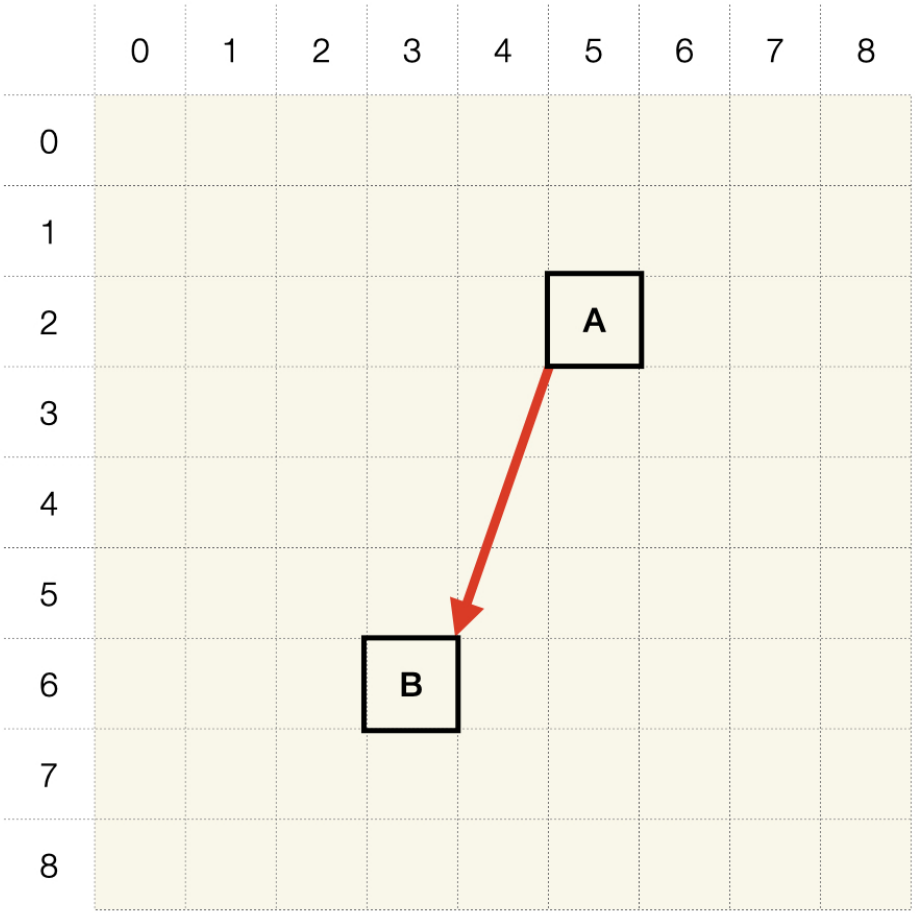
- **Explain how** conditional logic can manipulate the output of a computer program
- **Apply** mathematical knowledge to **write** computer programs that manipulate pixels in a surface
- **Trace** existing computer programs

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:
- $\sqrt{(red_1 - red_2)^2 + (green_1 - green_2)^2 + (blue_1 - blue_2)^2}$

Distance between 2 points



Using the Cartesian coordinate system:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Pos A = **5,2**
Pos B = **3,6**

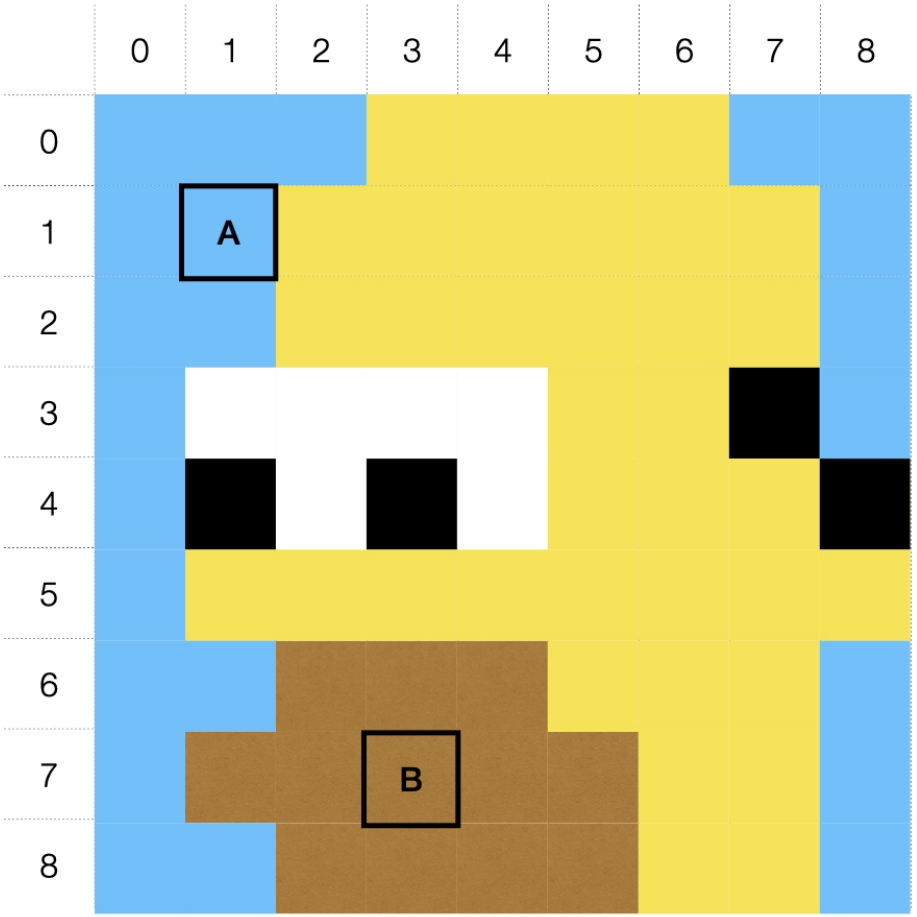
$$d = \sqrt{(3 - 5)^2 + (6 - 2)^2}$$

$$d = \sqrt{(-2)^2 + (4)^2}$$

$$d = \sqrt{20}$$

d= **4.472136**

Distance between 2 colours



In the **RGB** representation system:

$$\sqrt{(red_1 - red_2)^2 + (green_1 - green_2)^2 + (blue_1 - blue_2)^2}$$

Colour A = **86,193,255**
Colour B = **172,118,49**

$$d = \sqrt{(86 - 172)^2 + (193 - 118) + (255 - 49)^2}$$

$$d = \sqrt{(-86)^2 + (75)^2 + (206)^2}$$

$$d = \sqrt{55457}$$

d= **235.4930**

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 +
    blueDifference * 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 Windows forms project in Visual Studio
- Implement the function `closeEnough((colour1, colour2), tolerance)` that returns a boolean value
- Test your solution
- Then, post your solution on Slack

Boolean Return Values

```
public static bool closeEnough(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;

    if ((redDifference * redDifference + greenDifference
    * greenDifference + blueDifference * blueDifference)
    < 300)  {
        return true;
    } else {
        return false;
    }
}
```

Note: This source code should be used in conjunction with a function call to activate it.

Tolerance-based Pixel Manipulation

```
public static bool turnRed(Color newColour) {
    Color brown = Color.FromKnownColor(KnownColor.Brown);
    int redDifference;
    int greenDifference;
    int blueDifference;
    redDifference = newColour.R - brown.R;
    greenDifference = newColour.G - brown.G;
    blueDifference = newColour.B - brown.B;

    if ((redDifference * redDifference + greenDifference
    * greenDifference + blueDifference * blueDifference)
     < 4000) {
        return true;
    } else {
        return false;
    }
}
```

Tolerance-based Pixel Manipulation

```
Color p;

for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        p = bmp.GetPixel(x, y);
        if (turnRed(p)) {
            bmp.SetPixel(x, y, Color.FromKnownColor
            (KnownColor.Red));
        }
    }
}
```

Note: This uses the function from the previous screen. To achieve the outcome that you want you may need to tinker with the tolerance level to quite a high number.

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

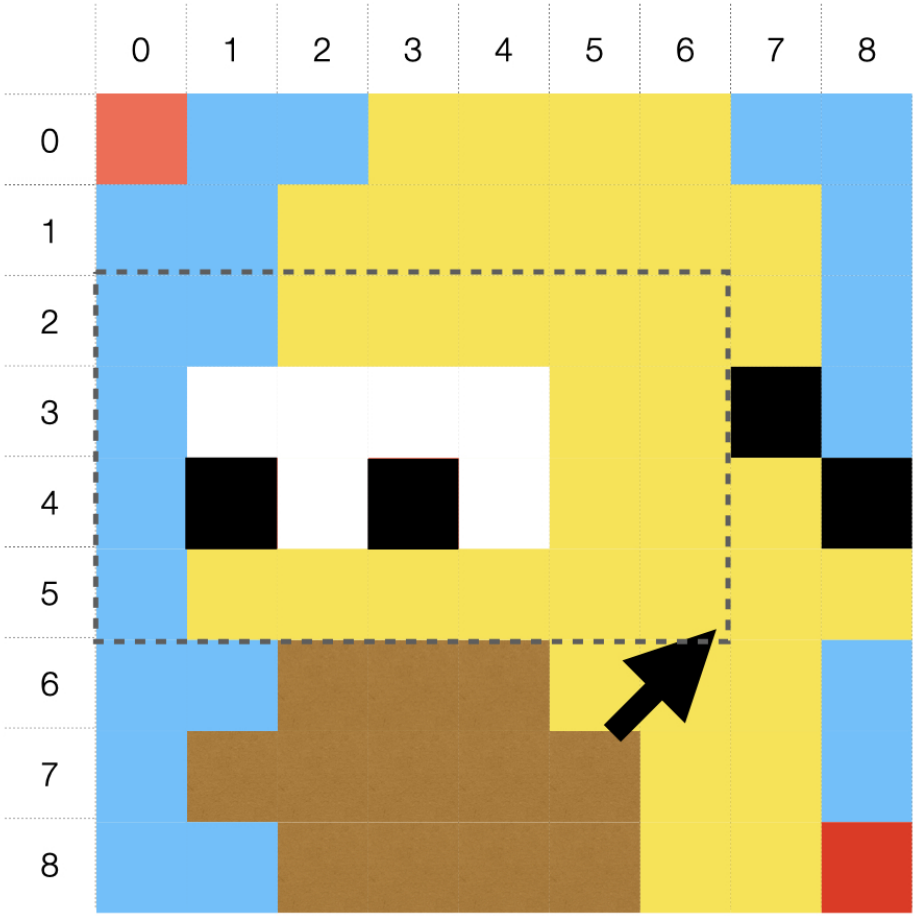
In pairs:

- Setup a Windows form project in Visual Studio
- You will need to create a method to sample a specific area of the image. I suggest you sample the mouse position.

```
private void pictureBox1_MouseUp(object sender,
MouseEventArgs e)
{
    mouseX = e.X;
    mouseY = e.Y;
}
```

- Implement the function: `removeRedEye(picture, area, colour)`
- Test your solution
- Then, post your solution on Teams

Create a sampling area using the mouse



Top Left: **0,2**

Bottom Right: **6,5**


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 Windows form project in Visual Studio
- 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

```
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        p = bmp.GetPixel(x, y);
        int r = p.R;
        int g = p.G;
        int b = p.B;
        int luminance = (r + g + b) / 3;
        // start at 64 to see the difference
        if (luminance < 104)    {
            bmp.SetPixel(x, y, Color.FromArgb(255, 0, 0,
             0));
        }
        if (luminance >= 104)   {
            bmp.SetPixel(x, y, Color.FromArgb(255, 255,
            255, 255));
        }
    }
}
```

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)

```
public void sepiaTint(picture) {
    //Convert image to greyscale
    makeGreyscale(picture);
    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            p = picture.GetPixel(x, y);

            int r = p.R;
            int g = p.G;
            int b = p.B;

            if (r < 63)
            {
                r = r*1.1;
                b = b*0.9;
            }
...
```

Source Code: Sepia (2)

```
...
                if (r > 62 and r < 192) {
                    bmp.SetPixel(x, y, Color.FromArgb(
                    255, 255, 255, 255));
                }
                if (r > 191) {
                    r = r*1.08;
                    if (r > 255) {
                        r = 255;
                    }
                    b = b*0.93;
                }

            }
        }
}
```

Note: This requires that you incorporate a greyscale function as well.

Activity #6: Sepia Tone

In pairs:

- Setup a Windows form project in Visual Studio
- 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