COMP120: Creative Computing

4: Tinkering Graphics III

Learning Outcomes

- **Explain how** to duplicate parts of an image
- **Explain how** to identify the spaces and features of an image
- **Apply** mathematical knowledge to **write** computer programs that manipulate the spaces and features of an image
- **Implement** executable code that can `tinker' graphics

Mirroring

- Mirroring is averaging across pixels
- A mirror will:
  - Define a dimension of inflexion
  - Duplicate pixels from the one side of the inflexion to the other, in reverse order

Source Code: Mirroring (1)

```
private static Bitmap mirrorVertical(string source)
    Bitmap bmp = new Bitmap(source);
    int width = bmp.Width;
    int height = bmp.Height;
    for (int ty = 0, by = height-1; ty < height/2; ty++,
        by--) {
        for (int x = 0; x < width; x++){
            Color p = bmp.GetPixel(x, ty);
            bmp.SetPixel(x, ty, p);
            bmp.SetPixel(x, by, p);
        }
    }
}
```

Note: When defining the initial value for bottom y (by) we minus 1 to avoid the creation of a gap in the mirror

Source Code: Mirroring (2)

```
private static Bitmap mirrorHorizontal(string source)
    Bitmap bmp = new Bitmap(source);
    int width = bmp.Width;
    int height = bmp.Height;
    for (int y = 0; y < height; y++) {
        for (int lx = 0, rx = width-1; lx <
        width/2+1; lx++, rx--) {
            Color p = bmp.GetPixel(lx, y);
            bmp.SetPixel(lx, y, p);
            bmp.SetPixel(rx, y, p);
        }
    }
}
```

Activity #1: Mirroring

In pairs:

- Use your function to repair the temple
- Post your repaired temple on Slack!

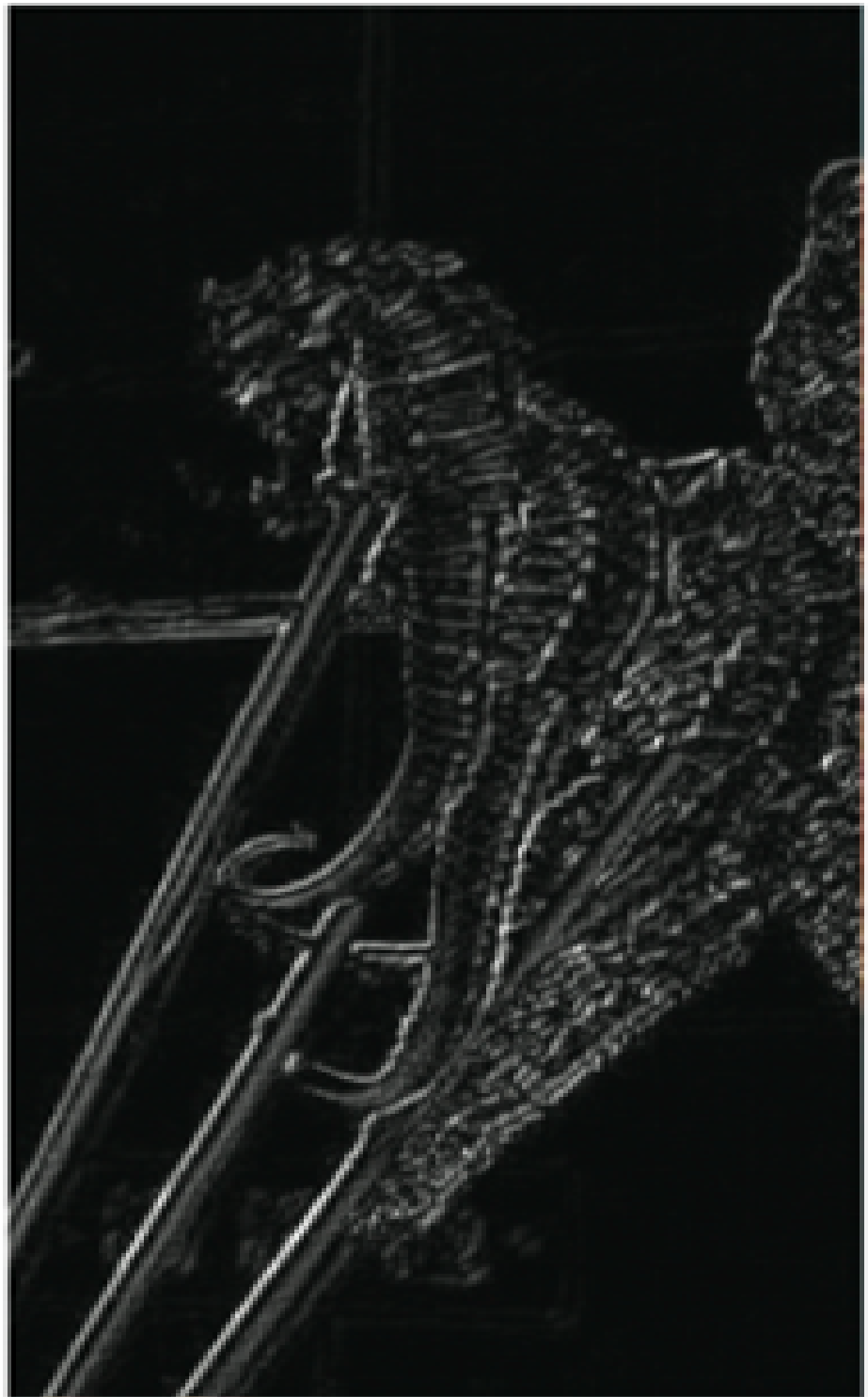
Edge Detection

- Blurring is averaging across pixels
- Edge detection is looking for *differences* between pixels:
  - We draw lines that our eyes see — where the luminance changes
- If the pixel changes left-to-right, or up-and-down, make a pixel black. Else, white.

Source Code: Edge Detection

```
private static Bitmap mirrorHorizontal(string source) {
    Bitmap bmp = new Bitmap(source);
    int width = bmp.Width;
    int height = bmp.Height;
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            Color pixel = bmp.GetPixel(x, y);
            Color nextPixel = bmp.GetPixel(x + 1, y + 1);
            int pixelSum = pixel.R + pixel.G + pixel.B;
            int nextPixelSum = nextPixel.R + nextPixel.G
            + nextPixel.B;
            int diff = Math.Abs(nextPixelSum - pixelSum);
        }
    }
    return bmp;
}
```

Note: This will create an error due to an overflow exception.
You will need to create a method to fix the error.



Source Code: Edge Detection (1)

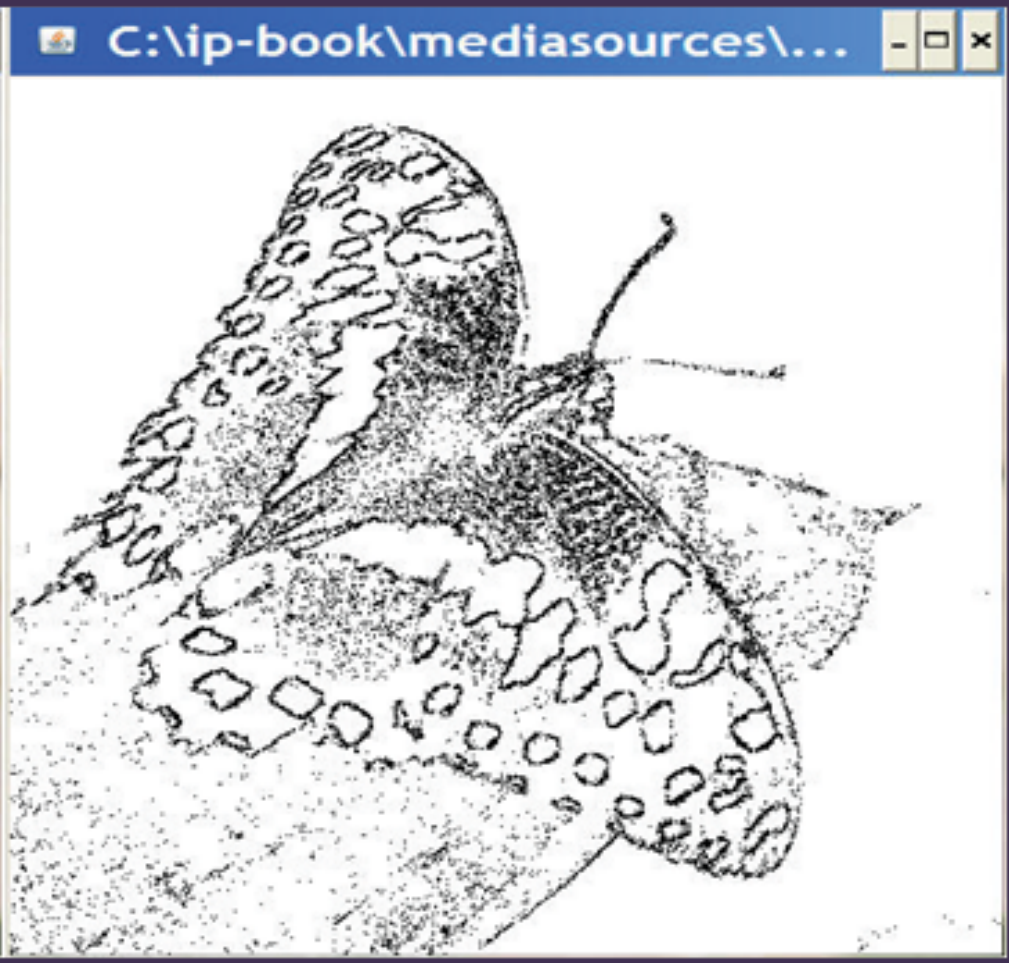
```
private static Bitmap drawBetterEdges(string source) {
    Bitmap bmp = new Bitmap(source);
    int width = bmp.Width;
    int height = bmp.Height;
    Bitmap ebmp = new Bitmap(bmp.Width, bmp.Height);
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            Color here = ebmp.GetPixel(x, y);
            Color right = bmp.GetPixel(x + 1, y)
            Color down = bmp.GetPixel(x, y + 1);
            int hereL = (here.R + here.G + here.B)/3;
            int downL = (down.R + down.G + down.B)/3;
            int rightL = (right.R + right.G + right.B)/3;
...
```

Note: Once again this will create an error due to an overflow exception. You will need to create a method to fix the error.

Source Code: Edge Detection (2)

```
...
                if (Math.Abs(hereL - downL) > 20 &&
                Math.Abs(hereL - rightL) > 20) {
                    ebmp.SetPixel(x, y, Color.FromArgb(255,
                    255, 255, 255));
                }
                if (Math.Abs(hereL - downL) <= 20 &&
                Math.Abs(hereL - rightL) <= 20) {
                    ebmp.SetPixel(x, y, Color.FromArgb(255,
                    0, 0, 0));
                }
            }
        }
}
```


C:\ip-book\mediasources\...


C:\ip-book\mediasources\...

Activity #2: Edge Detection

In pairs:

- Setup a basic project in PyGame
- Refer to the following documentation
- Refactor the function: `drawBetterEdges(picture)` to use better variable names
- Test your solution

Background Subtraction

- Let's say that you have a picture of someone, and a picture of the same place (same background) without the someone there, could you subtract out the background and leave the picture of the person?
- Maybe even change the background?
- What we most need to do is to figure out whether the pixel in the Person shot is the same as the in the Background shot.
- Will they be the EXACT same colour? Probably not.

Source Code Part A: Background Subtraction

```
 private static Bitmap swapBackground(string front,
 string back, string newBack, int tolerance)
{
    Bitmap bmp = new Bitmap(front);
    Bitmap bbmp = new Bitmap(back);
    Bitmap nbmp = new Bitmap(newBack);
    int width = bmp.Width;
    int height = bmp.Height;
    Color p;
    Color bp;
    Color nbp;
...
```

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

Source Code Part B: Background Subtraction

```
...
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            p = bmp.GetPixel(x, y);
            bp = bbmp.GetPixel(x, y);
            if (GetDistance(p,bp) < tolerance) {
                nbp = nbmp.GetPixel(x, y);
                bmp.SetPixel(x, y, nbp);
            }
        }
    }
    return bmp;
}
```

Note: This source code assumes that you have already created a function called ‘GetDistance’.

Problems

- We've got places where we got pixels swapped that we didn't want to swap
- We've got places where we want pixels swapped, but didn't get them swapped
  - Shirt stripes
  - Shadows
  - etc.

Activity #3: Background Subtraction

In pairs:

- Setup a forms project in Visual Basic
- Implement chroma key as a form of background subtraction
- Use your previous experience with isolating colour to implement this.
- Test your solution

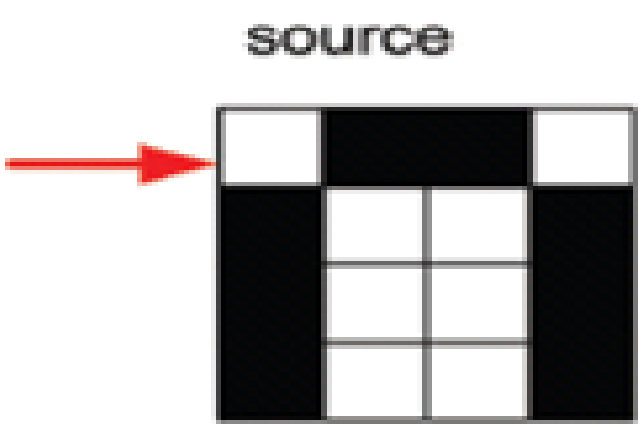
Source Code - Part 1: Collage

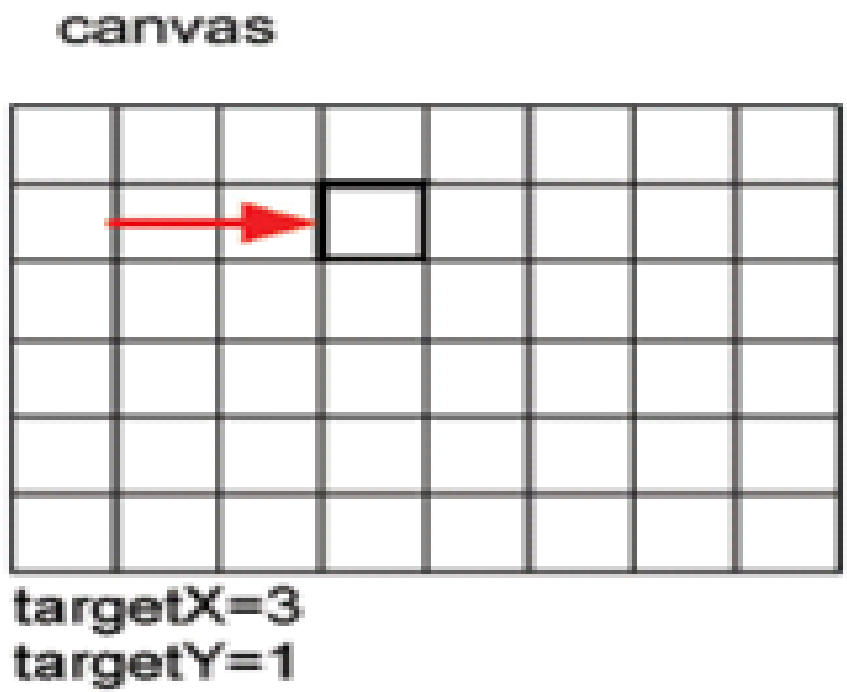
```
private static Bitmap swapBackground(string source,
string canvas, int targetX, int targetY)
{
    Bitmap bmp1 = new Bitmap(source);
    Bitmap bmp2 = new Bitmap(canvas);
    int width1 = bmp1.Width;
    int height1 = bmp1.Height;
    int width2 = bmp2.Width;
    int height2 = bmp2.Height;
...
```

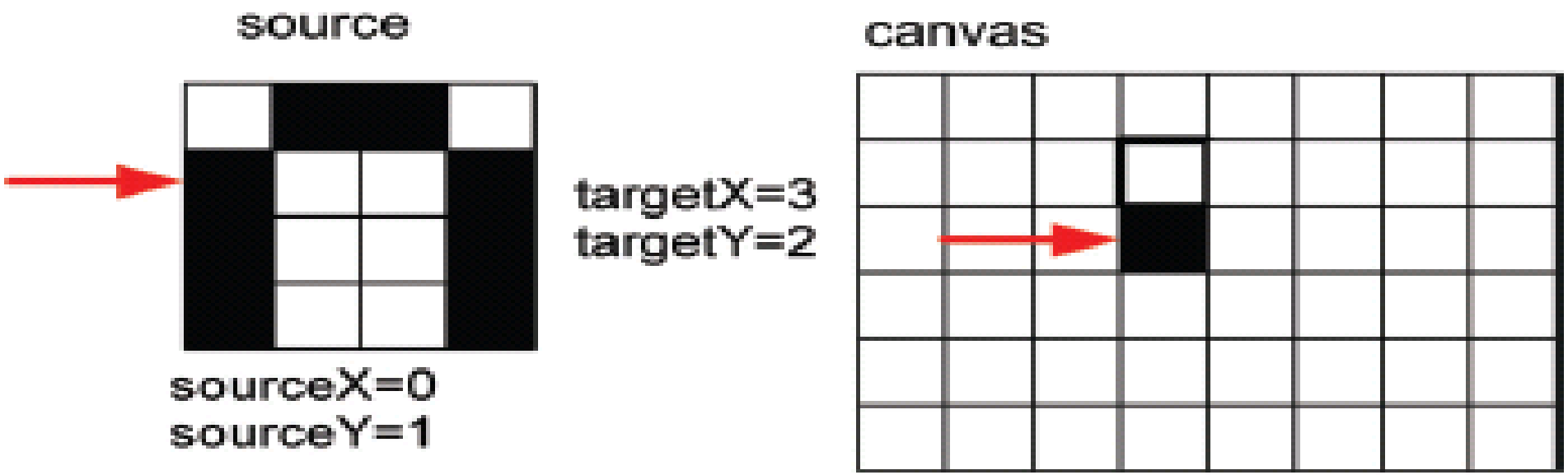
Note: **Source** represents the image to overlay and the **Canvas** is the background image. **targetX** and **targetY** are the offsets from the top left of the canvas.

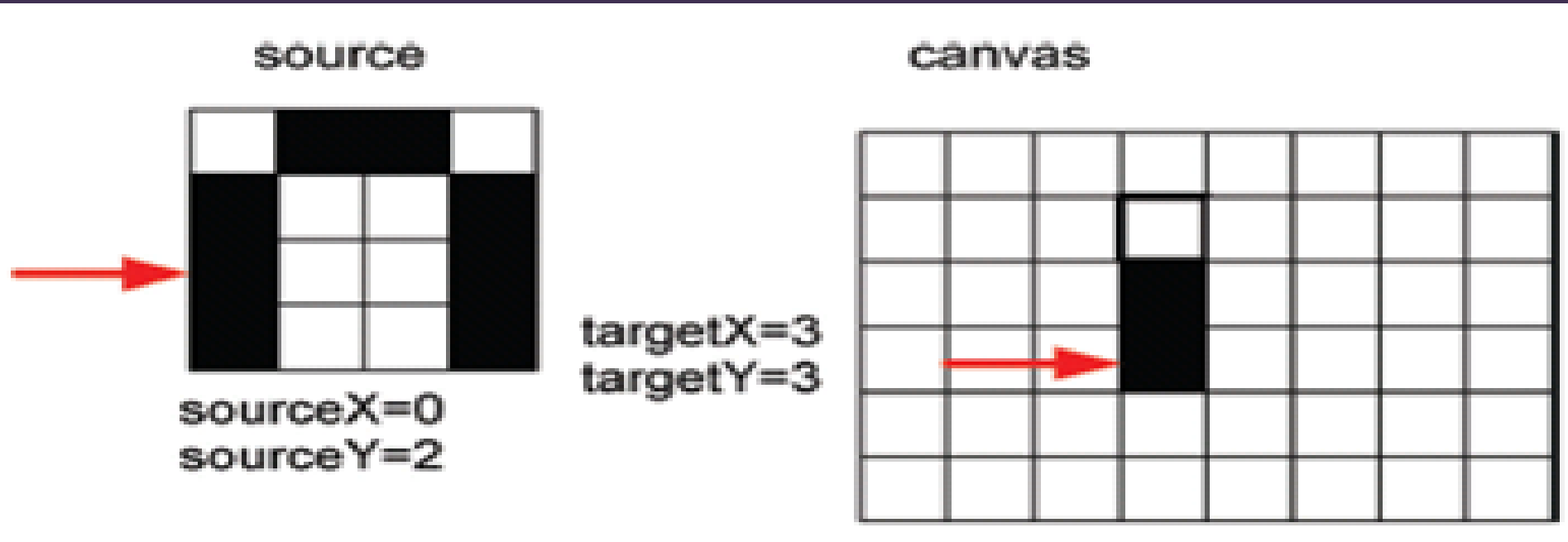
Source Code - Part 2: Collage

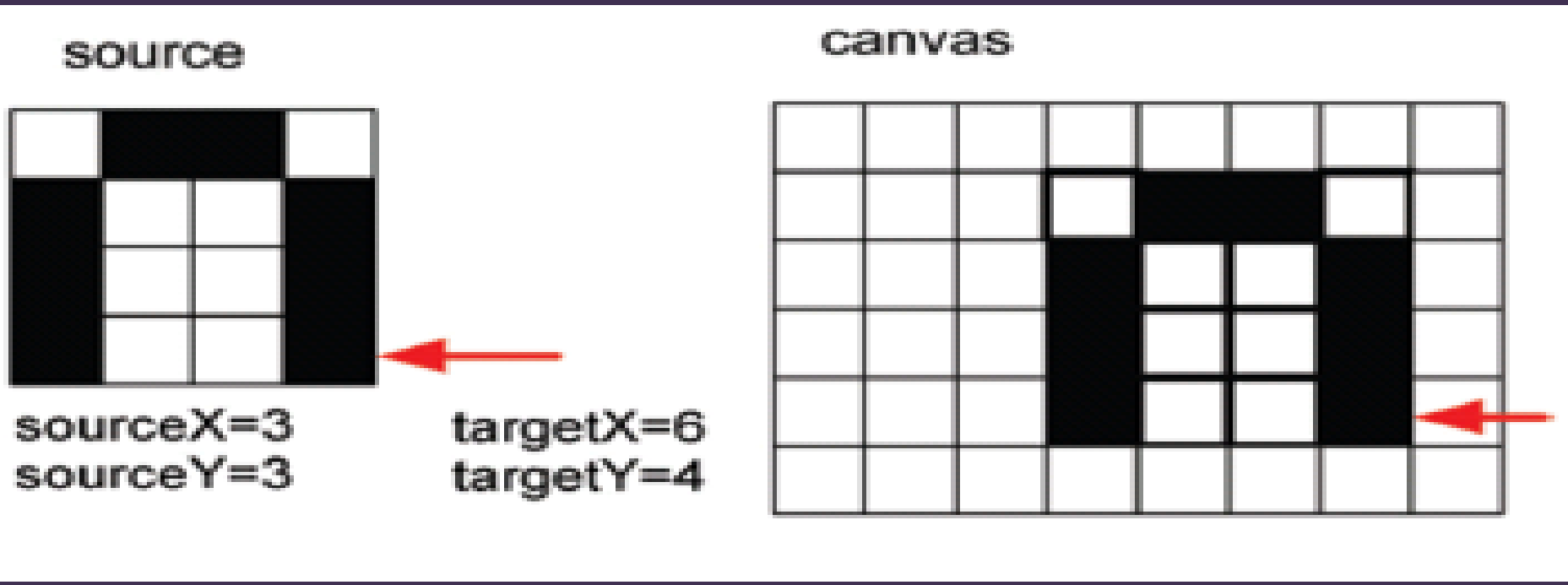
```
...
    for (int y = 0; y < height2; y++) {
        for (int x = 0; x < width2; x++) {
            if (x >= xpos && x < width1+xpos) {
                if (y >= ypos && y < height1+ypos) {
                    bmp2.SetPixel(x, y, bmp1.GetPixel
                    (x-xpos, y-ypos));
                }
            }
            bmp2.SetPixel(x, y, bmp2.GetPixel(x, y));
        }
    }
    return bmp2;
}
```

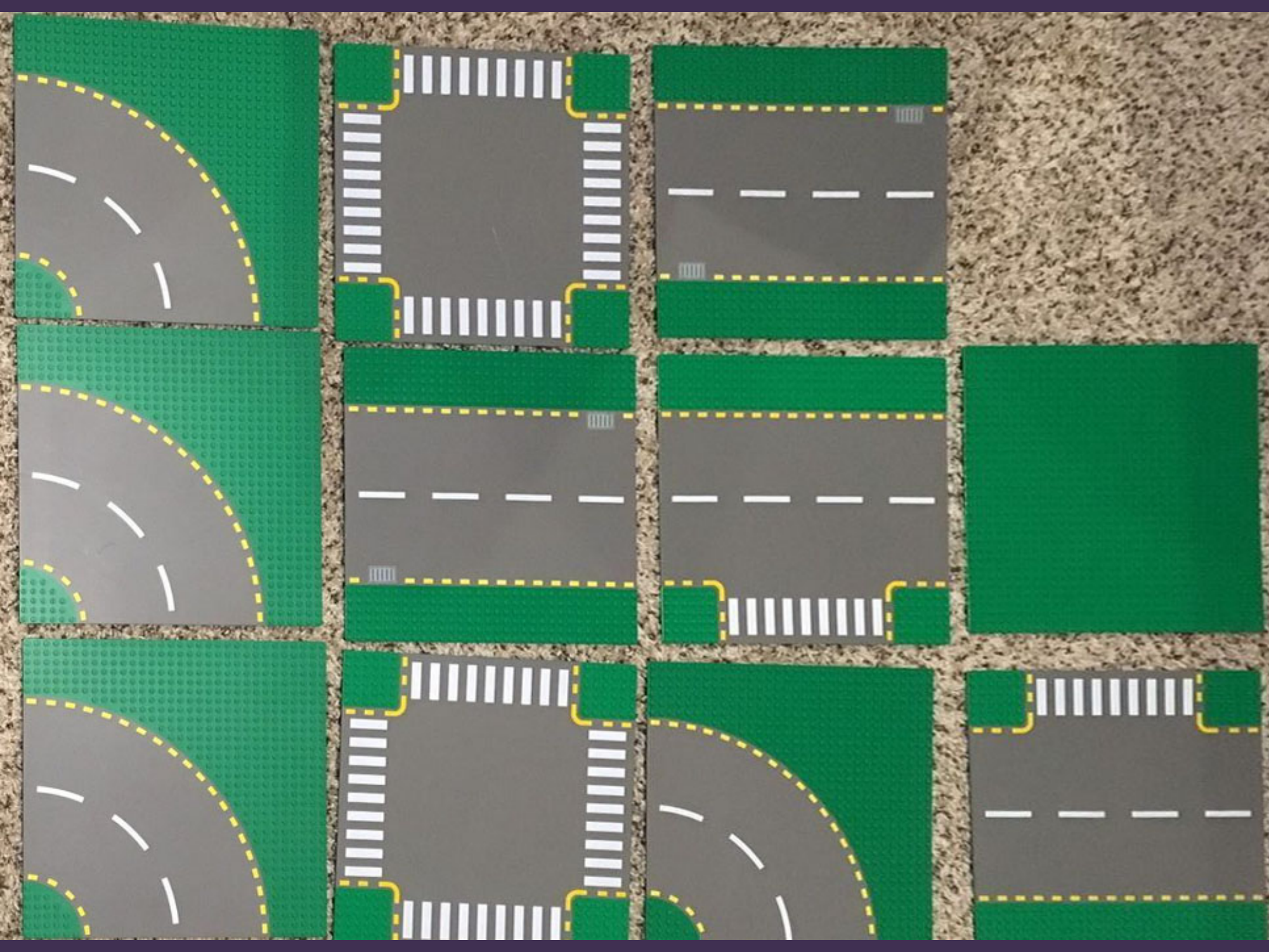

source
sourceX=0
sourceY=0


canvas
targetX=3
targetY=1


source
canvas
targetX=3
targetY=2
sourceX=0
sourceY=1


source
canvas
targetX=3
targetY=3
sourceX=0
sourceY=2


source
canvas
sourceX=3
sourceY=3
targetX=6
targetY=4



Activity #4: Collage

In pairs:

- Find some smaller images online
- Integrate the collage algorithm into your tinkering graphics project
- Create a routine that can layout a series of images in a defined sequence
- Use this to create a collage of the images you found

Sprite Sheets and Animations

Review this article on Code Project by Sasha Djurovic which is a forms application for creating sprite animations

`https://www.codeproject.com/Articles/1896/Sprite-manipulation-in-C`

Activity #5: Sprite Sheets

In pairs:

- Find a sprite sheet online
- Take the project code and see if you can manipulate it to create your own forms application
- Animate something

What Next?

If you have implemented all of these algorithms, then use the rest of the workshop to:

- You now have a range of image manipulation algorithms at your disposal
- You only need to successfully implement and repurpose a set of these to do well on your Tinkering Graphics assignment!
- So, finish implementing the algorithms needed to complete your coursework
- Tidy up your code, ready for next session's peer-review activity
- Extend the code beyond the brief as appropriate to your particular game