The program displays three tiles made up of 5x5 array of points on the left, and a grid made up of 10x10 array of tiles on the right. The tiles on the left are essentially the interface.

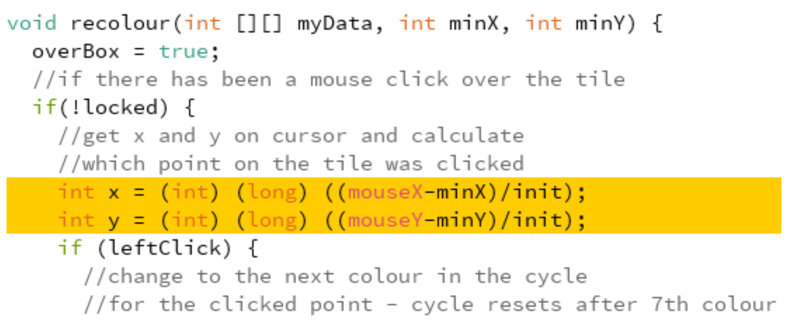
The top two tiles are for recolouring tile's A and B – to do this, the user simply clicks or right clicks one point on a tile, and it will update the interface and grid with the colour change. Left clicking will cycle through white, grey, red, yellow, green, blue, and black; right clicking will reverse through the cycle. Clicking a point at the end of the cycle (white or black) will loop it back to the opposite end of the cycle.

The third tile, below the recolouring tiles, is used to change the order tiles are displayed on the grid. By default, only tile A is shown, but left or right clicking on the reorder tile will cycle through an order of patterns. The order cycles as follows:
only A tiles; only B tiles;
A and B tiles repeating on each row; B and A tiles repeating on each row;
A and B tiles alternating on each row; B and A tile alternating on each row;
A and then mirrored A tiles in a repeating line; B and mirrored B tiles in a repeating line;
A and mirrored A tiles, then B and mirrored B tiles repeating; B and mirrored B tiles, then A and mirrored A tiles repeating.
Like recolour, the cycle moves to the opposing end when it reaches an end.

A notable algorithm I used is one for calculating which point in a recolour tile was clicked for its colour to be changed. It's a simple yet effective algorithm but took some time to get it working as expected. The minx/minY points are used to specify whether it's reading from the location of recolour tile A or B. mouseX and mouseY give the location of the mouse of the screen.

By taking away the 'min' value from the corresponding 'mouse' value, it essentially gives the mouse coordinates as if the min point were the origin (0, 0). Dividing this by the initial value (16 – the size of a single point on the screen) it gives a close decimal value to the actual number of the point in the array.

Converting the decimal to long and then to int acts like a rounding technique, except rather than rounding to the nearest whole number, it just rounds the number down (same as just removing the decimal off the end of the number) and converting to int is necessary as that is the numerical format that the array can read.



```
void recolour(int [][] myData, int minX, int minY) {
  overBox = true;
  //if there has been a mouse click over the tile
  if(!locked) {
    //get x and y on cursor and calculate
    //which point on the tile was clicked
    int x = (int) (long) ((mouseX-minX)/init);
    int y = (int) (long) ((mouseY-minY)/init);
    if (leftClick) {
      //change to the next colour in the cycle
      //for the clicked point - cycle resets after 7th colour
```

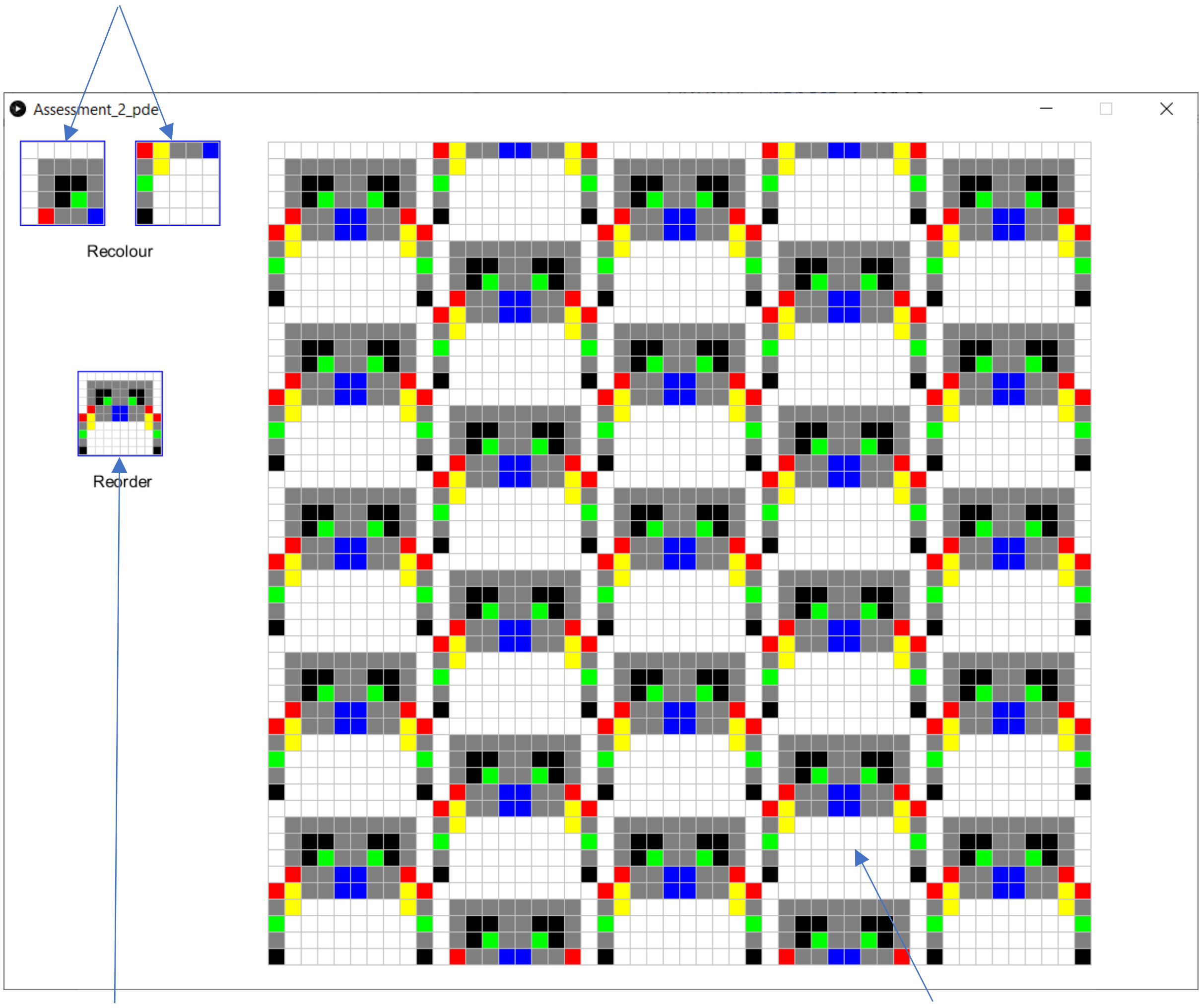
Complicated algorithms for drawing the different points in the grid and interface were avoided as processing allows me to simply give a larger translation before the actual drawing in order to move the grid wherever necessary on the screen.

With that taken care of, the only thing I felt I needed to specify in a drawing algorithm was drawing the points with distance from the translated point without overusing the translate function. This algorithm basically uses the counts of the 2-way loops (i and j) to get the distance each point should be drawn. For every loop, it draws the next point the initial value (16) away from the previous, as the initial value is also the size of each point.

```
        default:
          fill(255);
      }
      rect(init+(init*i), init+(init*j), init, init);
    }
  }
```

The pattern shown below is the second to last one, where A and B tiles are mirrored and repeated.

These are the recolour tiles, clicking a square will change the colour of that square and update the grid and interface references which include the tile



This tile will change the arrangement of the tiles on the grid each time it is clicked and update the grid and interface view

The grid shows the full view of the current tile arrangement with their currently chosen colours

I believe the program as it currently is works well. However, there are some improvements that can be made. One big improvement would be the colour selection and application to the tiles, which could be improved by first selecting a colour from a palette and then clicking on a single tile or click-dragging across multiple tiles in order to change their colour.

Tile rearranging could be improved by having a 4x4 grid of tiles on the interface section and allowing the user to change any of the tiles to be A or B to create their own arrangements. This could be taken a step further by allowing users to create more than 2 tiles and add as many as they would like.

A final improvement that could be made would be the ability to rotate individual tiles (such as in the above mention rearrange improvement) or to rotate the grid as a whole.