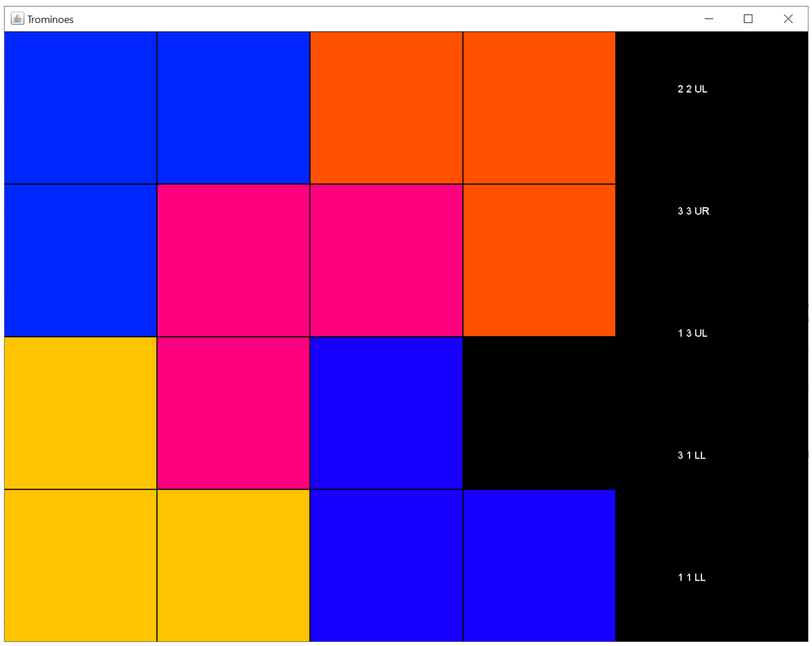
COMP361

Assignment 1

William Kilty

Core:

5.1. & 5.2

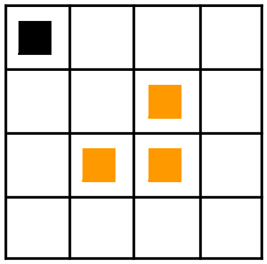


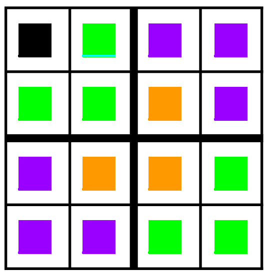
5.5.

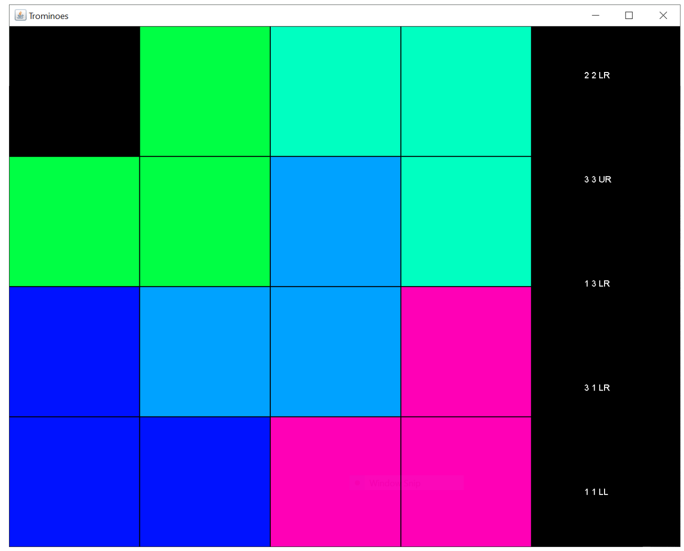
```
boolean checkSumEqual(double[] a, double val) {
      for (int i = 0; i < n; i++) {
            for(int j = i; j < n; j++) {
                  if (a[i] + a[j] == val) return true;
            }
      }
      return false;
}
```

Exercises:

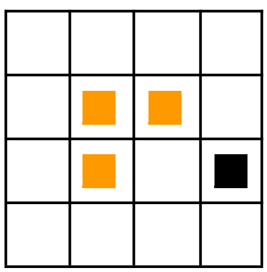
1.

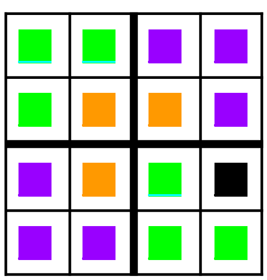


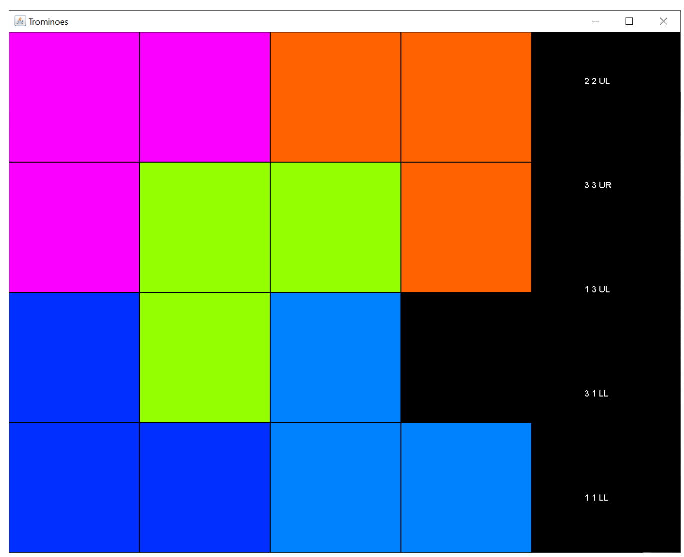




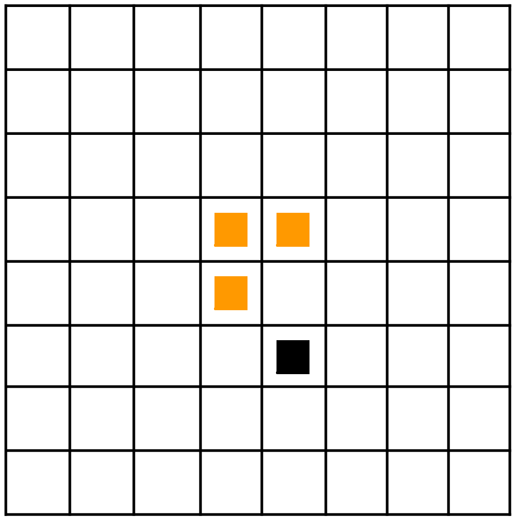
2.

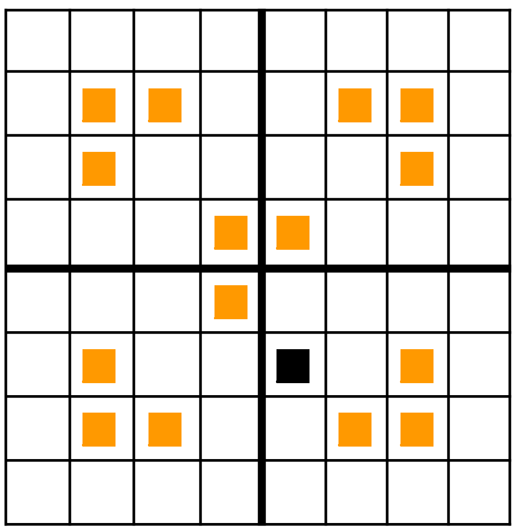


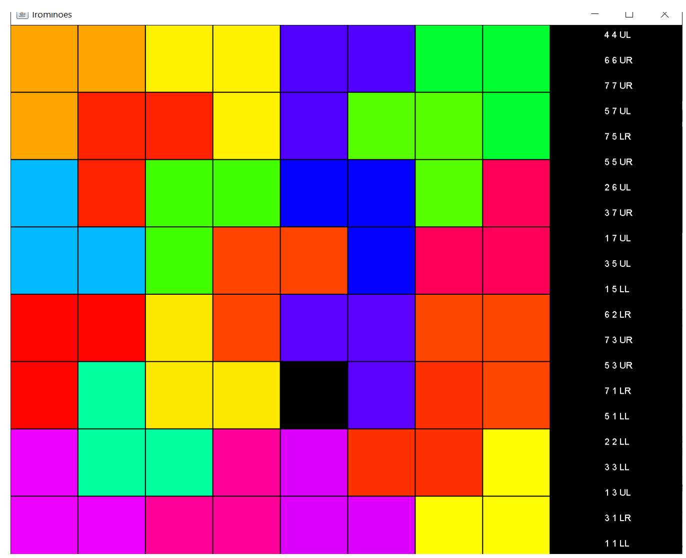


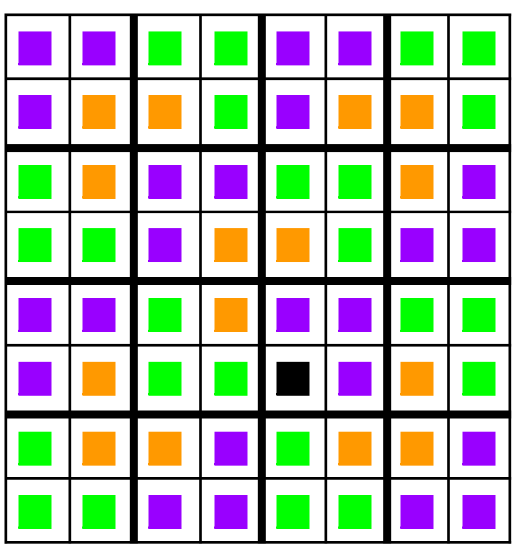


3.

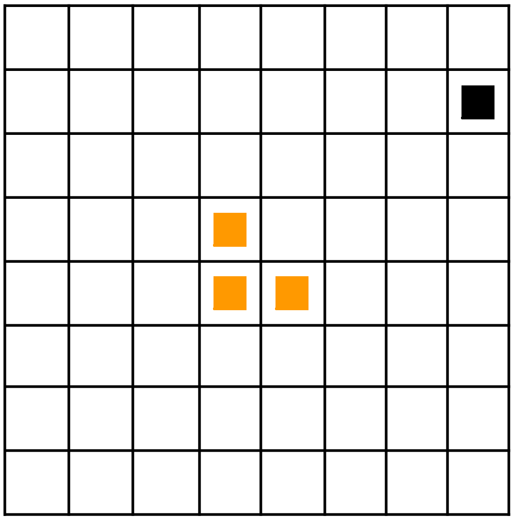


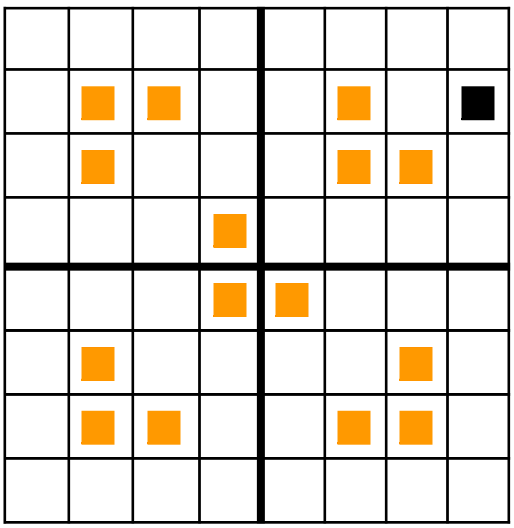


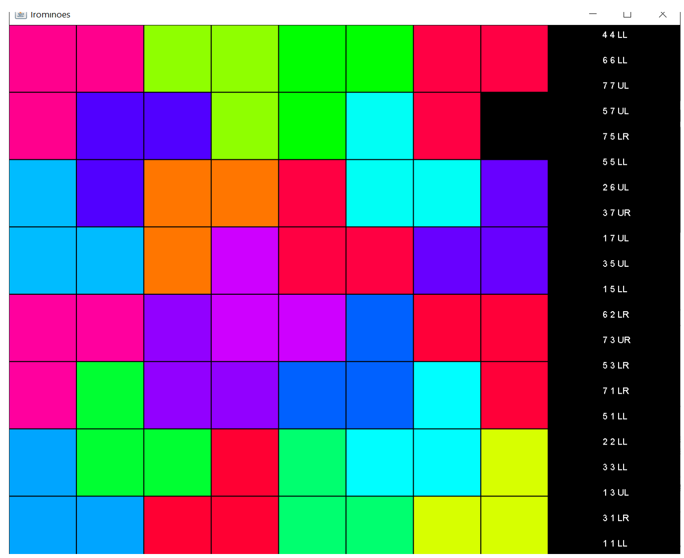

Irominoes
4 4 UL
6 6 UR
7 7 UR
5 7 UL
7 5 LR
5 5 UR
2 6 UL
3 7 UR
1 7 UL
3 5 UL
1 5 LL
6 2 LR
7 3 UR
5 3 UR
7 1 LR
5 1 LL
2 2 LL
3 3 LL
1 3 UL
3 1 LR
1 1 LL

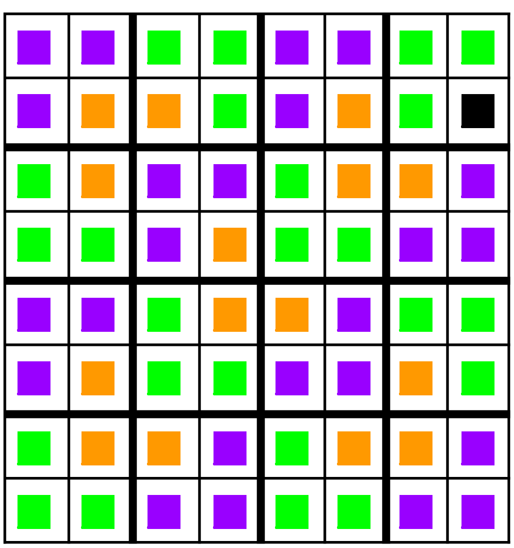


4.






Irominoes
4 4 LL
6 6 LL
7 7 UL
5 7 UL
7 5 LR
5 5 LL
2 6 UL
3 7 UR
1 7 UL
3 5 UL
1 5 LL
6 2 LR
7 3 UR
5 3 LR
7 1 LR
5 1 LL
2 2 LL
3 3 LL
1 3 UL
3 1 LR
1 1 LL



5.

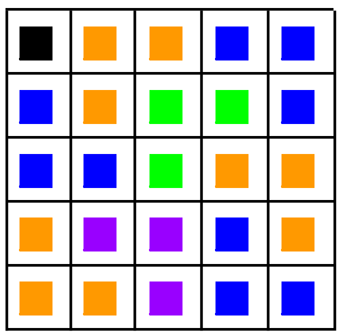
$$T(n) = \begin{cases} 1 & n = 2 \\ 1 + 4T(\frac{n}{2}) & n > 2 \end{cases}$$

$\alpha = log_2 4 = 2$

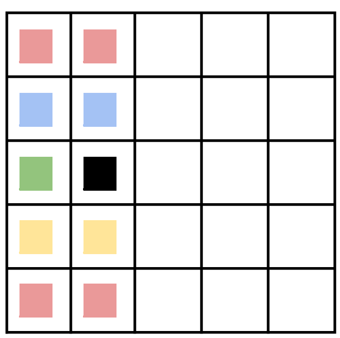
$f(n) \in O(n^{\alpha-\epsilon}) = 1 \in O(n^{2-2}) = 1 \in O(1)$

In the master theorem, case 1 applies, so T(n) ∈ Θ(n^2),
where T(n) == c_n

6.



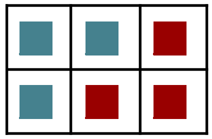
7.



Looking at the green tile highlighted, there are only two possible ways to fit a tromino into its space. Either via the blue highlighted spaces, or via the yellow. In both situations, we end up with two tiles in a row, which cannot be tiled by a tromino. These are highlighted in red.

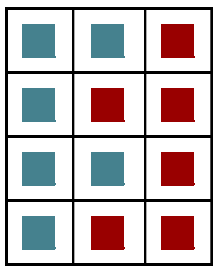
Completion:

8. We start with our base-case of i = j = 1:

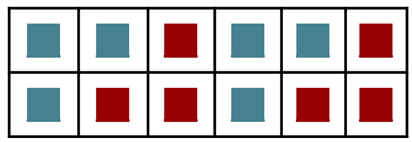


Now, any multiples of i or j will simply tile this pattern.

For example, if i = 2:



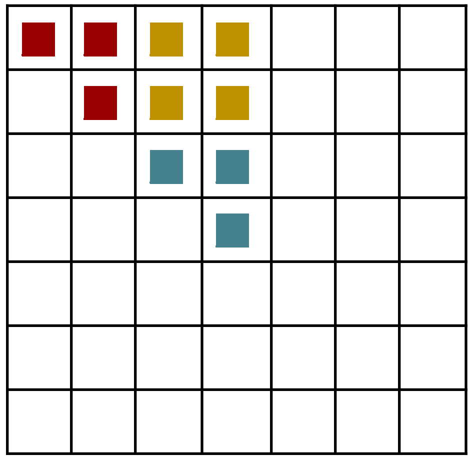
Or, if j = 2:



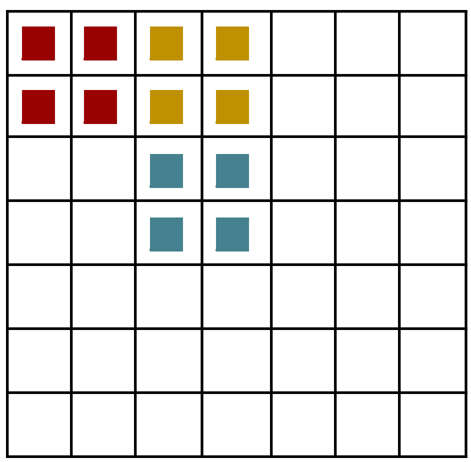
This works for any positive integers i, j

```
void tile2x3(int i, int j) {
      for (int x = 0; x < i; x++) {
            for(int y = 0; y < j; y++) {
                  placeTromino(x*2-1, y*3-2, Dir.UL);
                  placeTromino(x*2-1, y*3-1, Dir.LR);
            }
      }
}
```

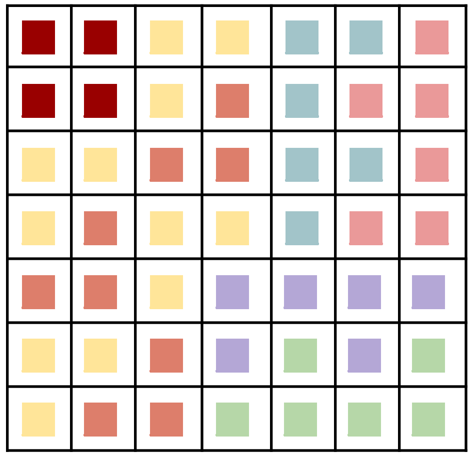
9. Due to symmetry; rotating and flipping the board around its centre leaves only these tiles as unique deficient boards:



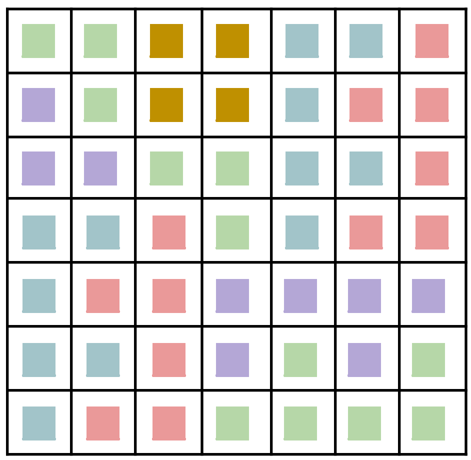
Note the three different colours. This is because any missing tile within a colour can be tiled by a single tromino in the centre of that coloured section. This means there are only three base cases with which to tile a deficient 7x7 board.



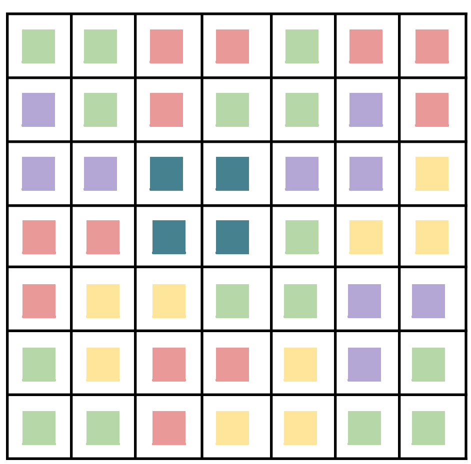
We just need to include these extra tiles to make complete squares. Then it’s a simple case of creating a tiling for the rest of the board



Case 1



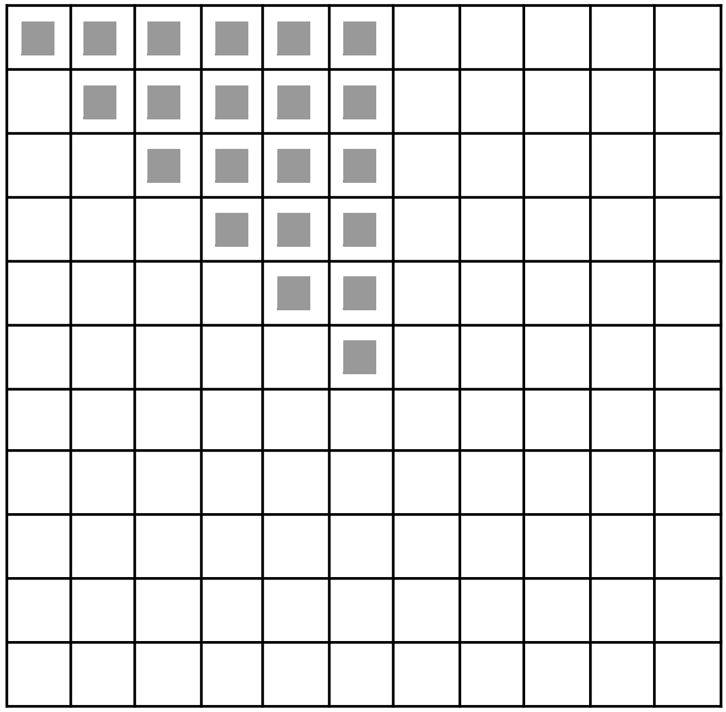
Case 2



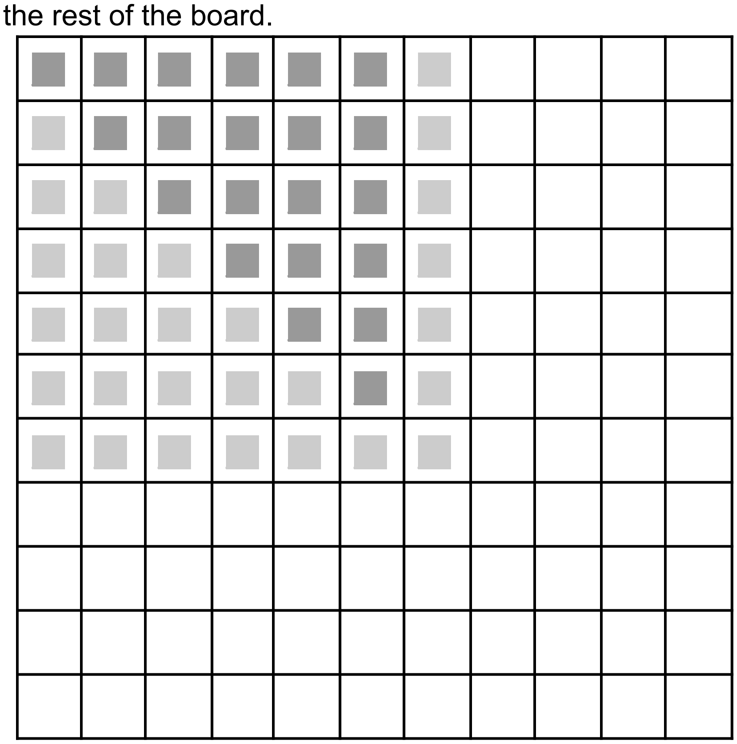
Case 3

These three cases allow a tiling of any deficient 7x7 board with Trominoes.

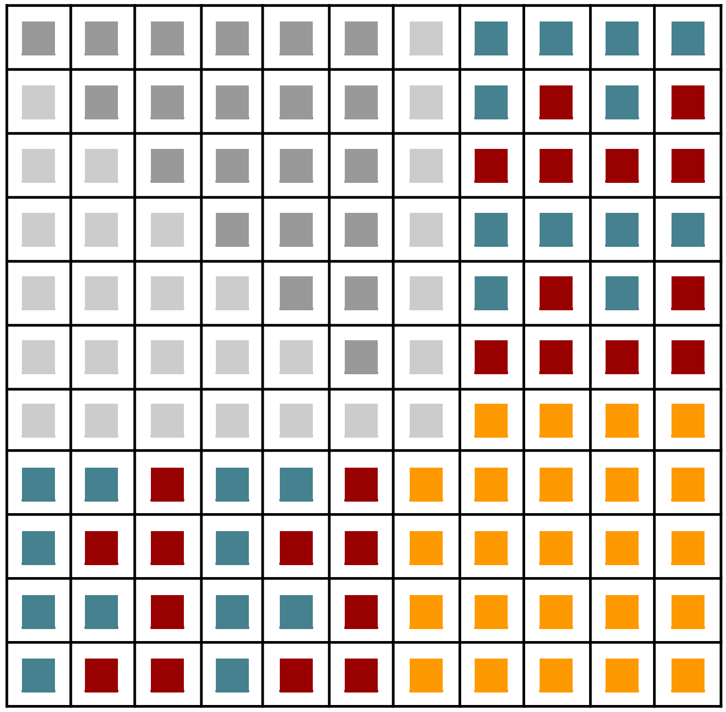
10. Similarly to 7x7, utilising symmetry allows only a limited number of deficient 11x11 boards



Upon closer inspection, it can be noted that all these cases fit within a 7x7 subgrid of the board. This means we can use a 7x7 tiling to fit each of these cases. It’s then just a case of filling out the rest of the board.



And filling out the rest of the board, it turns out, can be achieved through a combination of the question 8 and question 6 solutions as follows:



Using a 5x5 board missing the top left spot to tile the lower right of the board, and then filling in the remaining spaces with 2x3 mappings, we get a completely tiled 11x11 deficient board.

```
void tile11x11(int x, int y) {
      If ((10-y) > x || x > 5) {
            rotateAndFlip(11, x, y);
            return;
      }
      tile7x7(x, y-4, 0, 4); //tile a 7x7 offset by (0,4)
      tile5x5(6, 0); //tile a 5x5 offset by (6,0)
      tile2x3(2, 2, 0, 0); //tile a 2x3 offset by (0,0)
      tile3x2(2, 2, 7, 5); //tile a 3x2 offset by (7,5)
}
```

11. Tiling an nxn deficient board where n is odd and nxn % 3 = 1:

```
void tileOdd(int n, int x, int y, int xOff, int yOff) {
      If (n <= 5 || n*n % 3 != 1) return;
      If (n == 7) tile7x7(x, y, xOff, yOff);
      If ((n-1-y) > x || x > (n+1)/2) {
            rotateAndFlip(n, x, y, xOff, yOff);
            return;
      }
      tileOdd(n-4, x, y-4, 0+xOff, 4+yOff);
      tile5x5(n-3+xOff, yOff);
      tile2x3(2, 2, xOff, yOff);
      tile3x2(2, 2, xOff+n-4, yOff+5);
}
```

16x16 as required by marking schedule: