**C# Array Game – Elimination**

**Game Rules**

In Elimination, the player clicks coloured blocks in order to ‘eliminate’ them from the grid. When they do so, each eliminated block is replaced with a new, random colour. Each block is worth 100 points or more and the player must eliminate as many blocks as possible before the timer runs out.

To maximise their points, the player should try to link large groups of coloured blocks together. If a group consists of more than 1 block, each successive block eliminated is worth an additional 100 points, i.e. eliminating 3 single blocks will earn the player 100 + 100 + 100 = 300 points whereas eliminating a group of 3 linked blocks will earn 100 + 200 + 300 = 600 points.

**Approach to the problem**

To start with, I needed to create a 2 dimensional grid of buttons of random colours. Once I had this, I needed a way of detecting a group of linked blocks by checking the clicked block’s neighbours. This algorithm turned out to be the most difficult part of the assignment to implement. When the program could detect and link up groups of the same colour blocks, I developed a scoring system that would reward players for playing more tactically and trying to link together large groups, rather than randomly clicking or earning the same amount of points for eliminating lone blocks. To add to the difficulty, I then added a countdown timer which locks the grid and shows the player their score once it reaches 0.

**Difficulties**

The biggest difficulty I had was implementing a way of identifying a group of linked coloured blocks so that the correct score could be calculated. Initially, I tried doing this with iteration whereby the program searched along the x axis of the clicked button, then its y axis. This worked in some cases, such as 3 blocks in a horizontal line. In testing however, I noticed that in, for example, S-shaped groups, finding neighbours this way would only find blocks that where in the “line of sight” of the clicked block, as it only travelled down its x and y axes and would miss out those that weren’t.

Eventually I worked out that I had to implement the ‘find neighbours’ method recursively. This way, the algorithm would try to move left, right, up and down of the clicked block. If the algorithm could move in any direction (i.e. the neighbouring block is of the same colour), it recursively moves to the next block and in turn checks the left, right, up and down of this second block. By doing this, it continues along all paths of matching colours in every direction, even reaching blocks not in the “line of sight” of the clicked block. Therefore by doing it this way, the game was able to detect groups of blocks of any shape.

**Future Enhancements**

To make the game more interesting and challenging, I would add difficulty levels that change the length of time the player has to play and the size of the grid. I could also create different game modes, such as ones where the player must avoid a certain colour block that will reduce their time and conversely, certain blocks increase their time, eliminate all of its neighbour blocks regardless of their colour etc.