ASSIGNMENT - A5

Problem Statement :-

Write a C++/Java program to draw a 4x4 chessboard rotated 45° with the horizontal axis. Use Bresenham's algorithm to draw the lines. Use ~~tee~~ seed file algorithm to fill black squares of rotated chessboard.

Objectives :-

1) To learn and understand seed file and boundary file algorithm.
2) To implement seed file algorithm recursively and non-recursively.
3) To implement rotation of polygon using 2-D transformation.

Outcomes :-

1) Student will be able to implement seed file algorithm.
2) Students will be able to implement rotation of 2-D figures using transformation.

Theory :-

Polygon filling is process of coloring in a fixed area or region when regions are defined at pixel level we have algorithms like.

- Boundary fill.
- Flood fill.

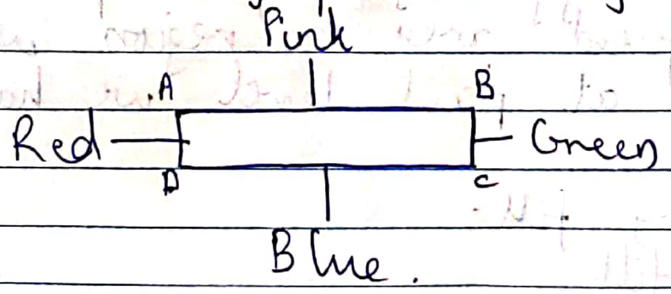
- Edge fill
- Fence fill.

Boundary fill algorithms:

1) This algorithms needs one point which is surely inside the polygon.
2) This starting point is called seed point which is nothing ~~that~~ but a point from which the filling process starts.
3) It is a recursive method. The algorithms checks to see if seed pixel has a boundary color pixel or not.
4) If no, then fill the pixel boundary color and boundary make a recursive call to each of neighbouring pixel.
5) This algorithm work for any shaped polygon and fills that polygon with boundary color but it uses high power of recursive calls & takes more time and memory.

Flood Fill Algorithms:

1) This algorithms also begins with seed point which must be surely inside the polygon original color is previous or old colour.
2) If yes, fill that pixel with ~~old~~ new colour and uses each of pixel neighbouring call



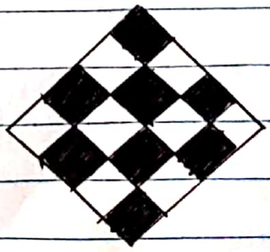
3) Sometimes we want to fill an area that is not defined with a single colour boundary. Flood filled helps in their case as we do not check boundary line.

4) The flood fill algorithm is something else called as seed file algorithms or forest fill algorithm.

Algorithm :

```
Boundary (x, y, new color) {
        current =  getpixel (x, y);
        if ((current ! = new color) && (current ! = boundary
                                                      color))
        {
              putpixel (x, y, new color)
              boundary (~~int~~ x+1, y, new color)
              boundary (x, y+1, newcolor)
              boundary (x-1, y, new color)
              boundary (x, y-1, new color)
           }
        }
```

Test Case :

	Input	Output	Status.
1.	Length :100		Success.

Conclusion :

Thus the seed file and boundary fill algorithm were implemented successfully.