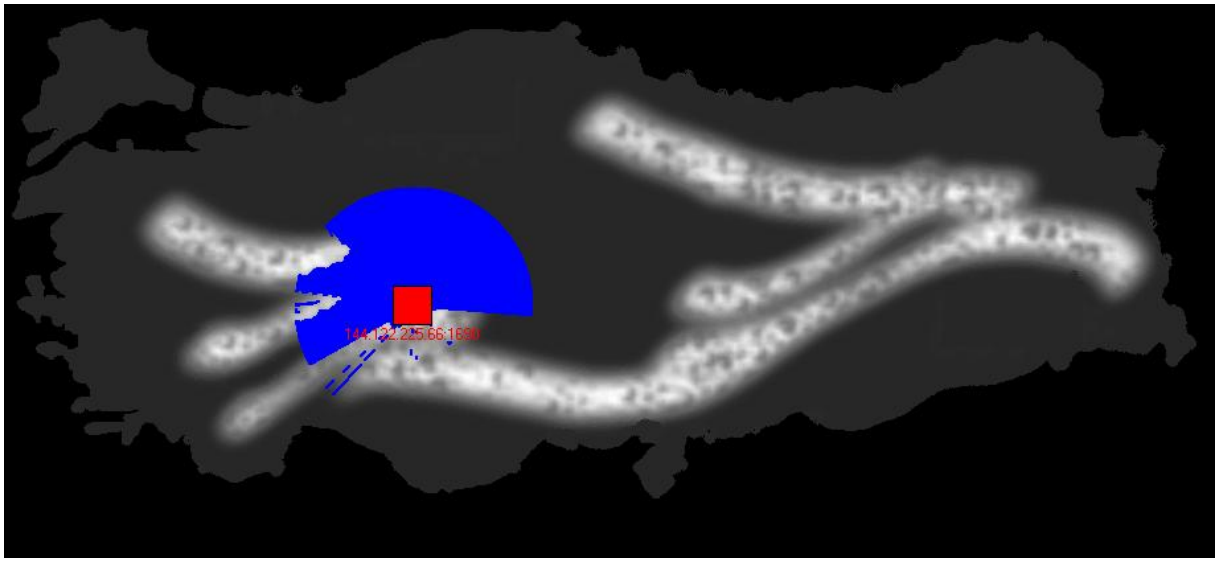
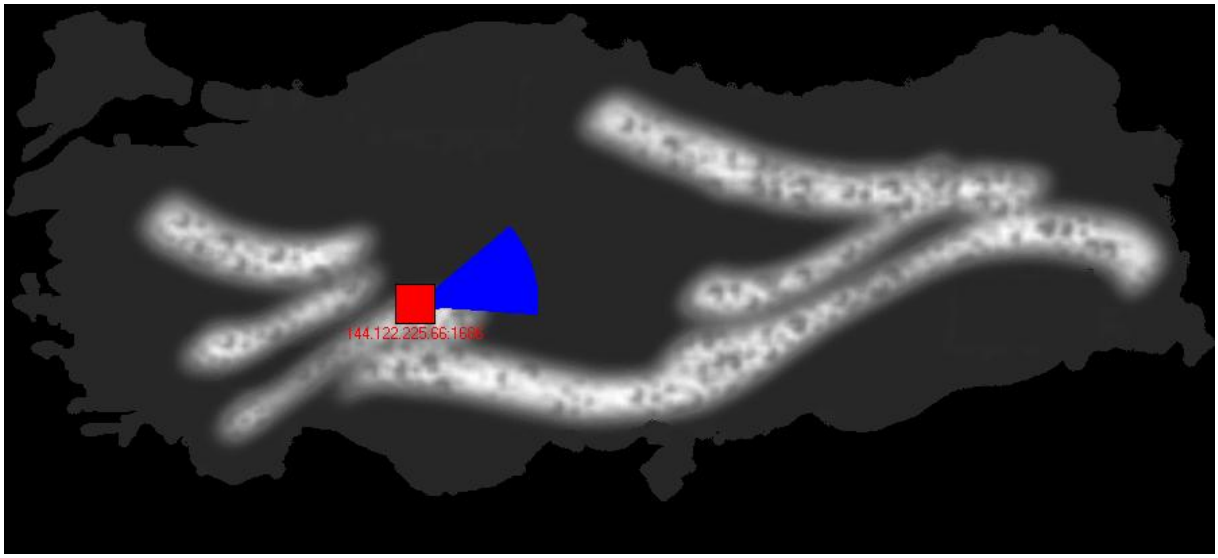
MECAC WEEKLY REPORT (May 12 – May 25)

This week we have worked on view angle integration in area of interest computation. Before view angle integration, the area of interest computation was as below. Here blue points are visible around red client. The result of area of interest computation includes non occluding points. Note that view angle is not relevant in this result.



When we have included view angle to area of interest computation resulting area is drawn with blue in below picture. Computation of the view angle involves knowing the client direction and there is couple of trivial geometric calculations in between.



Given a direction of the client we first compute the angle that is between 0-360. We have normalized this angle considering singularity points in unit circle.

```
if (takeIt)
{
    Vector3 direcVec = clientDirections[client.Key];
    double direcAngle = vectorAngle(direcVec.X, direcVec.Z);
    Console.WriteLine("direction angle: " + direcAngle);

    double angleMax = direcAngle + viewAngle;
    double angleMin = direcAngle - viewAngle;

    double potentialAngle1 = vectorAngle((float)(potential.X - pointOnMap.X), (float)(potential.Y - pointOnMap.Y));

    Console.WriteLine("potential angle: " + potentialAngle1);

    double potentialAngle2 = potentialAngle1 - 360.0;


    if( (potentialAngle1>=angleMin && potentialAngle1<=angleMax) || (potentialAngle2 >= angleMin && potentialAngle2 <= angleMax) ) {
        Console.WriteLine("ciziyor");
        visiblePoints[numVisiblePoints++] = new System.Drawing.Point(potential.X, potential.Y);
        e.Graphics.DrawRectangle(pen, potential.X, potential.Y, 1, 1);
    }
}
```

Code snippet 1 : Integration of view angle information to area of interest computation

When we find all visible points given in Figure 1, we check every point if the point is in the view range. To compute the view range, we first determine the angle that the direction vector makes with x-axis.

We then extract a range of angles from the direction angle considering an offset of 40 degrees per side. Total of 80 degrees are visible for each client. Then we extract the angle between the visible point and the client point has been computed using vectorAngle method.

To be consistent, we check if potential angle is within the range we find at the first step. We also check 360 complement of the same angle.

```
private double vectorAngle(float x, float z)
{
    if (x == 0 && z == 0)
    {
        return 0;
    }
    else
    {
        double asin = radianToDegrees(Math.Asin(z / Math.Sqrt(x * x + z * z)));

        if (x >= 0 && z >= 0)
        {
            // 1 bolge
            return asin;
        }
        else if (x < 0 && z >= 0)
        {
            // 2 bolge
            return 180 + asin;
        }
        else if (x < 0 && z < 0)
        {
            // 3 bolge
            return 180 - asin;
        }
        else
        {
            // 4 bolge
            return 360 + asin;
        }
    }
}
```

VectorAngle is a method that we have added to the server component interface. This method computes the angle between a 2-D vector with X-axis. We have returned the angle within the range of [0-360].

This range is specifically needed considering our future calculations with angle ranges.

To normalize the angle to [0-360] range we check four quadrants unit circle and add an offset angle to arcSin of the actual angle.