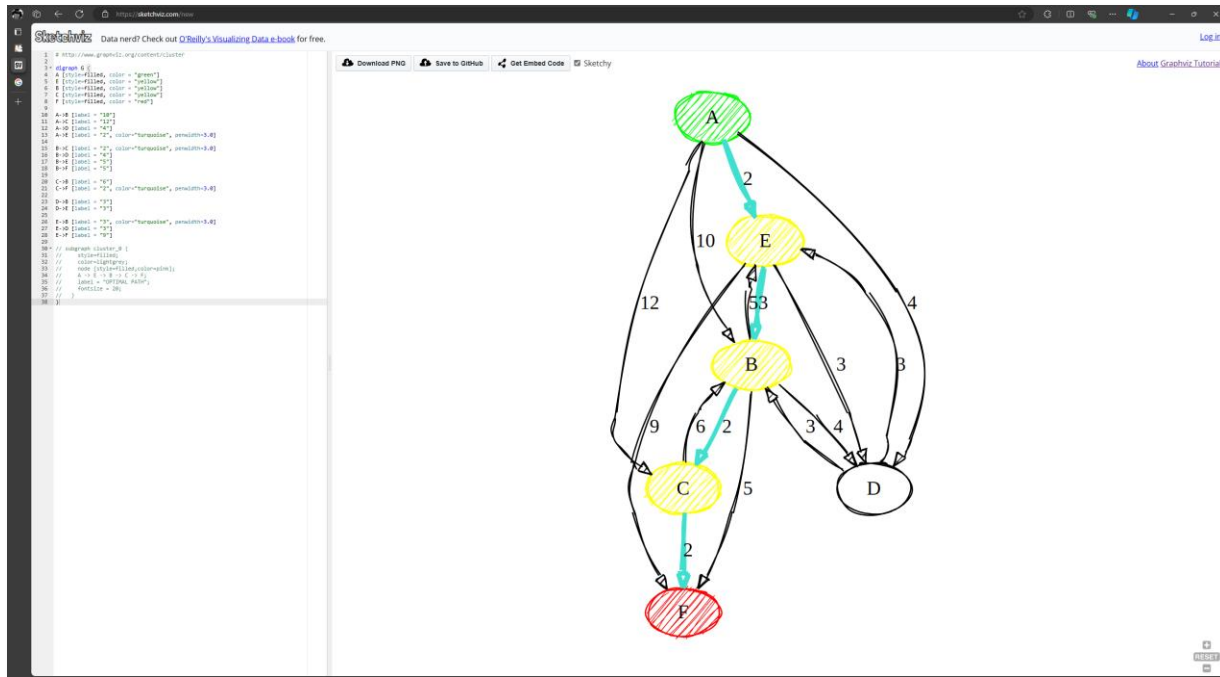# LAB – 8

PATH FINDING

Fig 1.1: Graph on a visualizer website (SketchViz) with optimal path via Dijkstra Algorithm
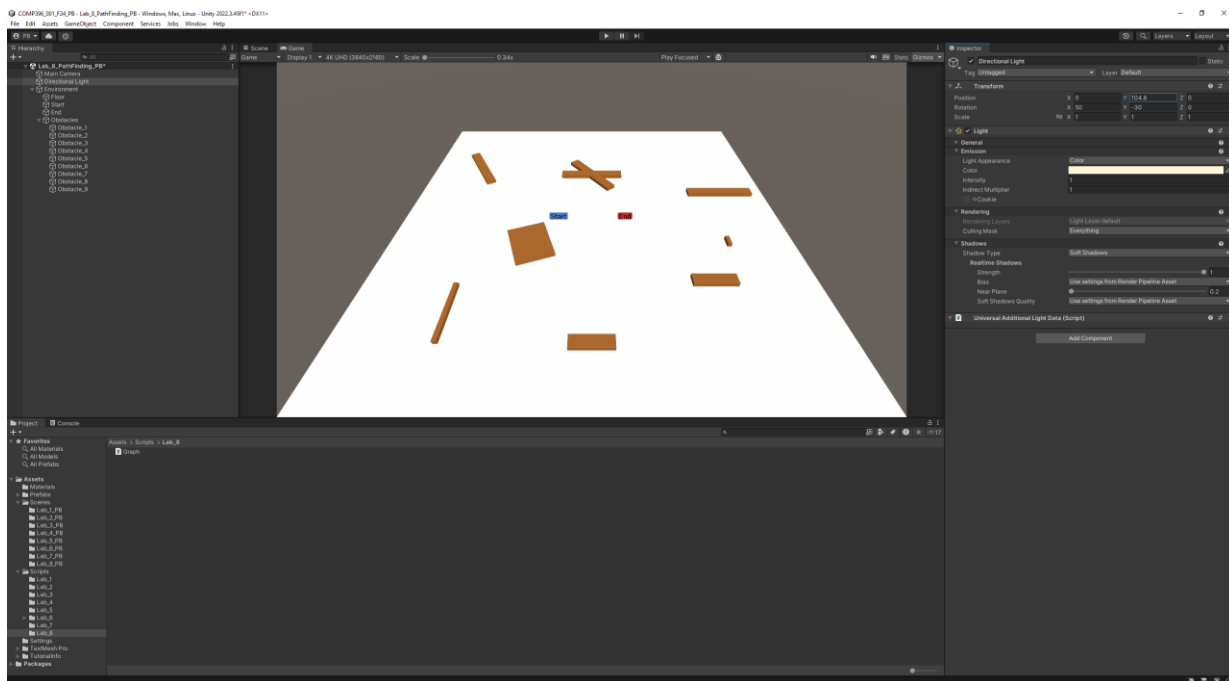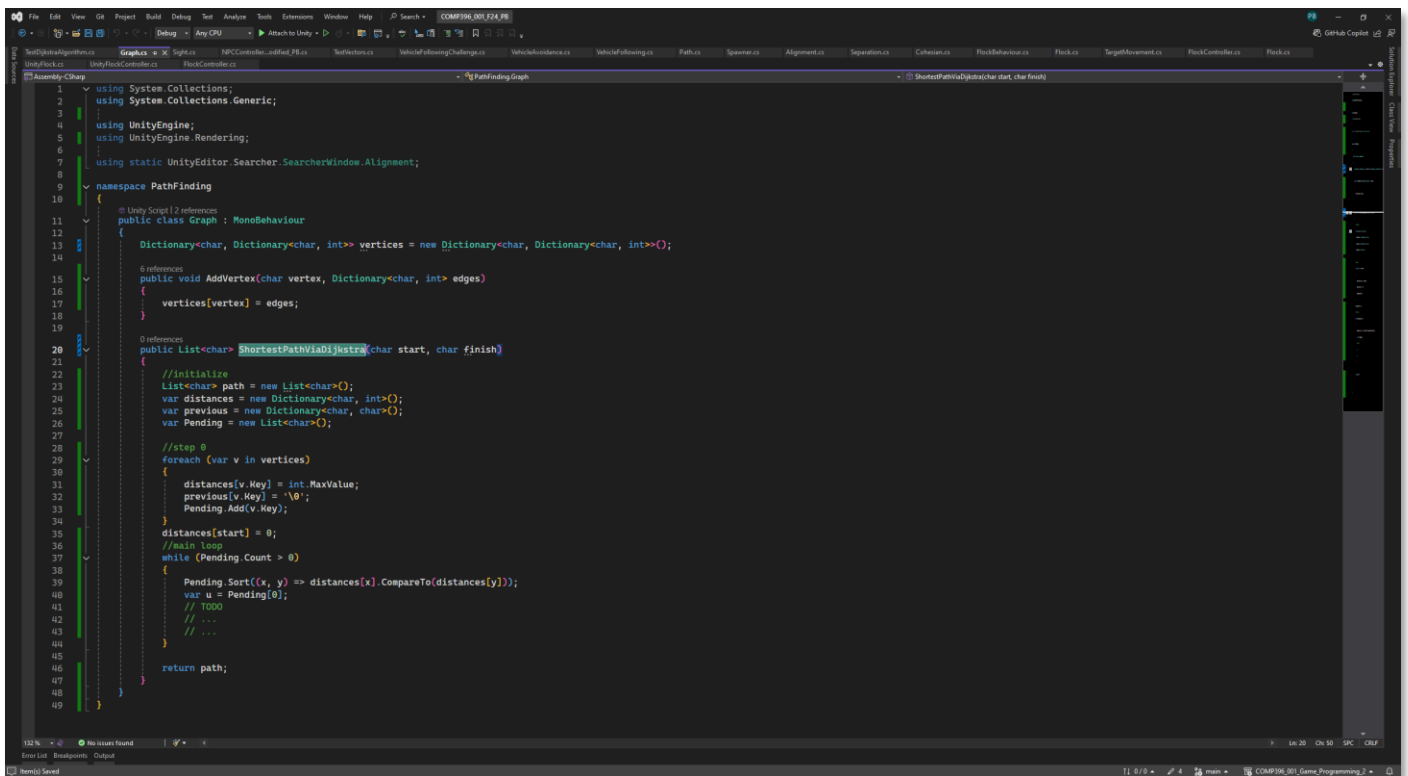

Fig 1.2: Screenshot showing Objects Placed

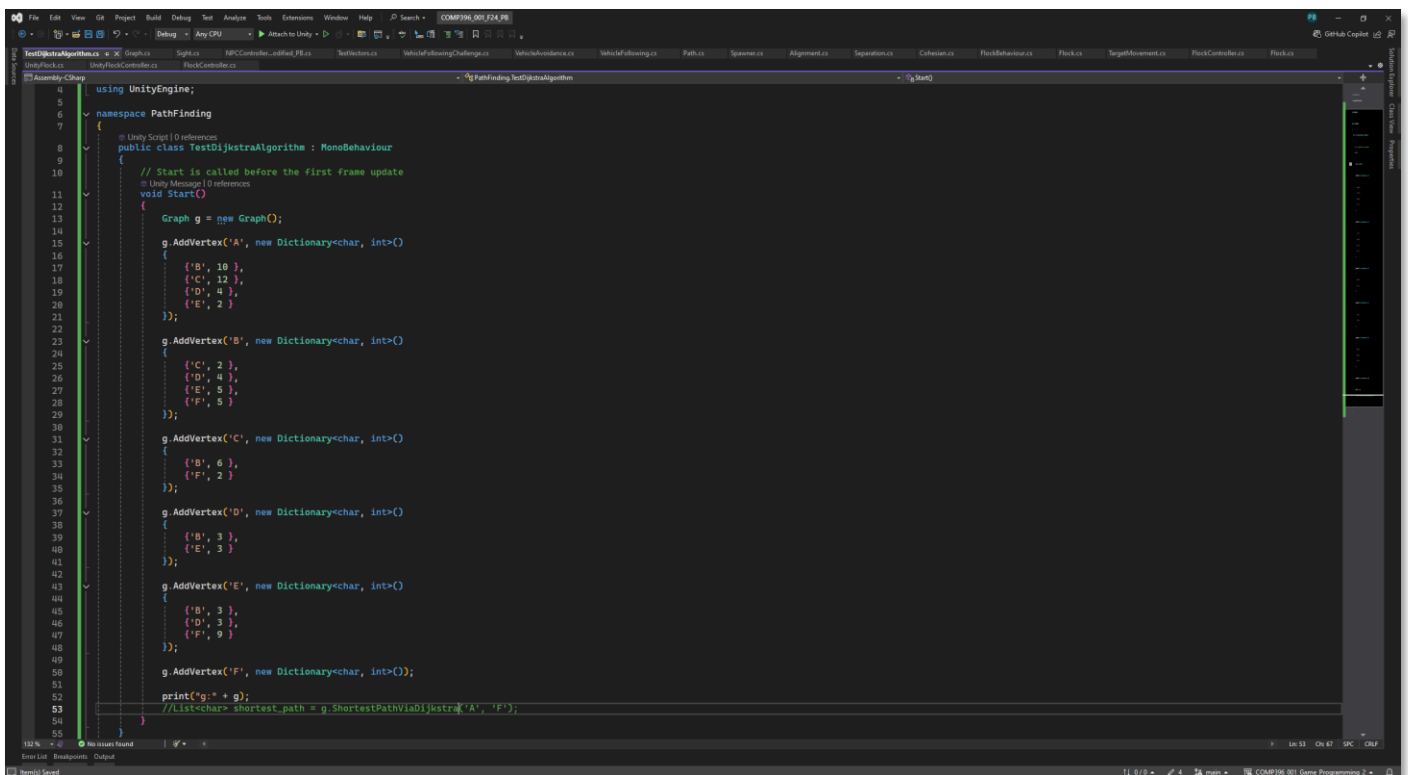Fig 1.3: Screenshot showing Graph Template
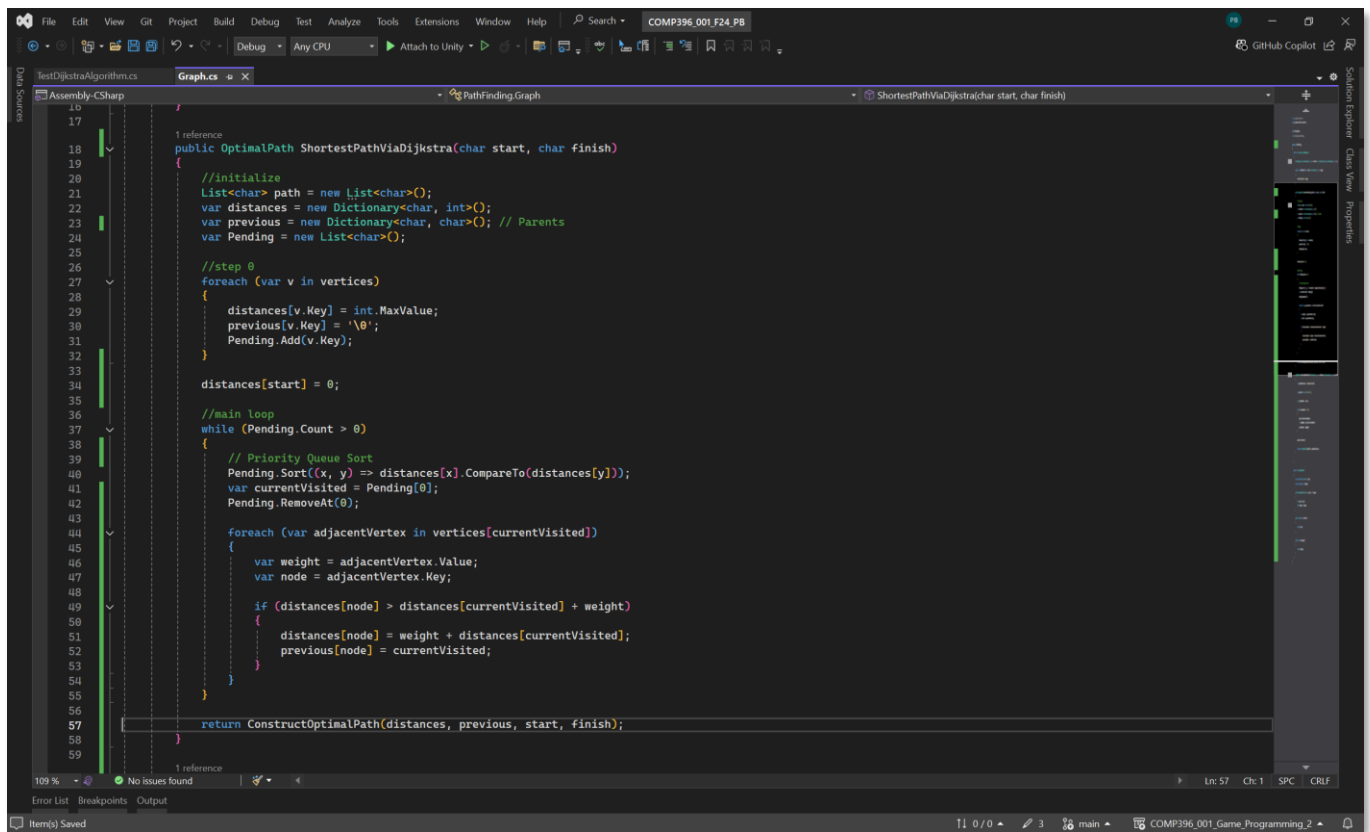

Fig 1.4: Screenshot showing Dijkstra Algorithm File

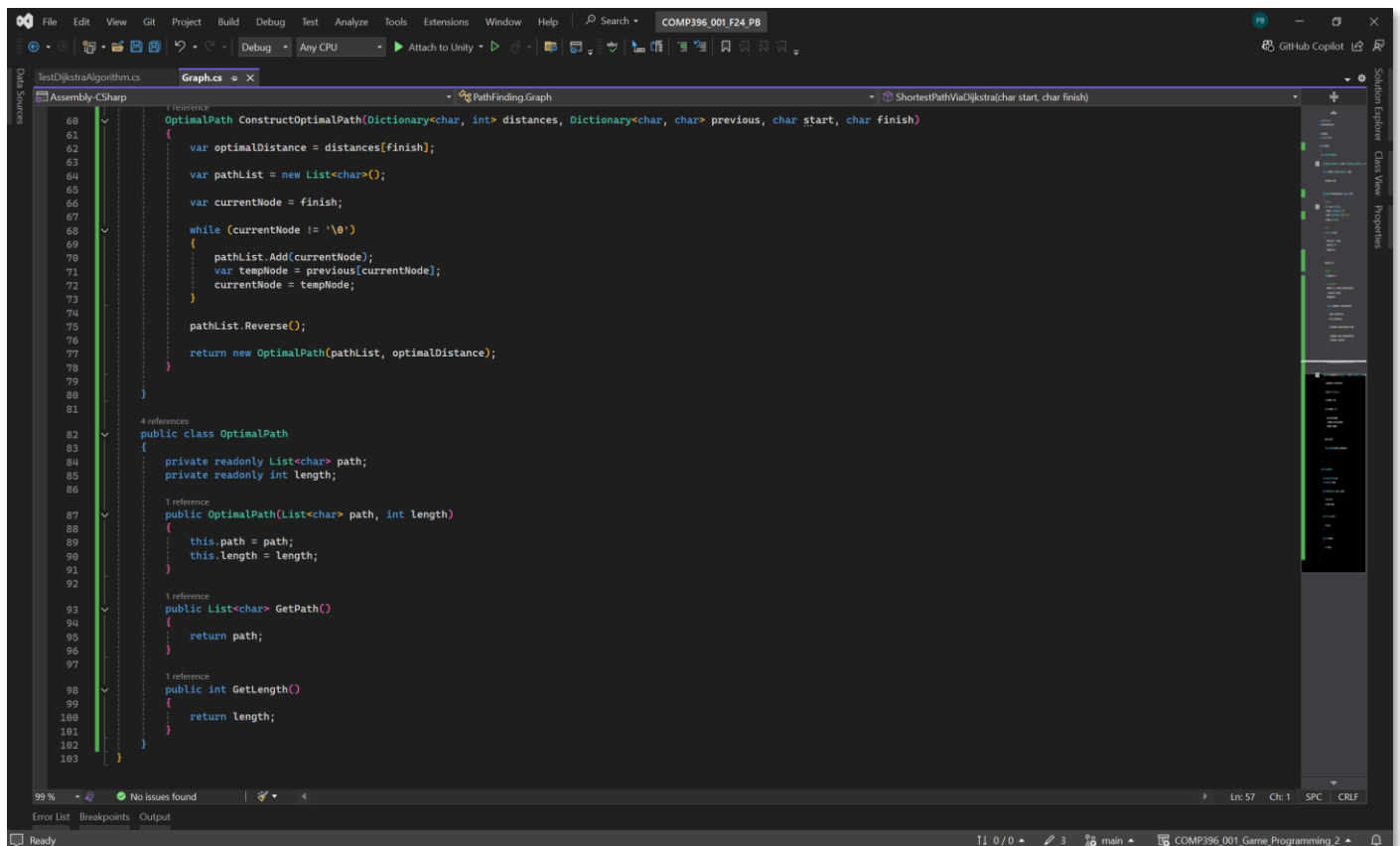Fig 1.5: Screenshot showing Implementation of Dijkstra Algorithm

```csharp
public OptimalPath ShortestPathViaDijkstra(char start, char finish)
{
    //initialize
    List<char> path = new List<char>();
    var distances = new Dictionary<char, int>();
    var previous = new Dictionary<char, char>(); // Parents
    var Pending = new List<char>();

    //step 0
    foreach (var v in vertices)
    {
        distances[v.Key] = int.MaxValue;
        previous[v.Key] = '\0';
        Pending.Add(v.Key);
    }

    distances[start] = 0;

    //main loop
    while (Pending.Count > 0)
    {
        // Priority Queue Sort
        Pending.Sort((x, y) => distances[x].CompareTo(distances[y]));
        var currentVisited = Pending[0];
        Pending.RemoveAt(0);

        foreach (var adjacentVertex in vertices[currentVisited])
        {
            var weight = adjacentVertex.Value;
            var node = adjacentVertex.Key;

            if (distances[node] > distances[currentVisited] + weight)
            {
                distances[node] = weight + distances[currentVisited];
                previous[node] = currentVisited;
            }
        }
    }

    return ConstructOptimalPath(distances, previous, start, finish);
}
```


Fig 1.6: Screenshot showing Supporting structure of Dijkstra Algorithm PT- 1

```csharp
OptimalPath ConstructOptimalPath(Dictionary<char, int> distances, Dictionary<char, char> previous, char start, char finish)
{
    var optimalDistance = distances[finish];

    var pathList = new List<char>();

    var currentNode = finish;

    while (currentNode != '\0')
    {
        pathList.Add(currentNode);
        var tempNode = previous[currentNode];
        currentNode = tempNode;
    }

    pathList.Reverse();

    return new OptimalPath(pathList, optimalDistance);
}

public class OptimalPath
{
    private readonly List<char> path;
    private readonly int length;

    public OptimalPath(List<char> path, int length)
    {
        this.path = path;
        this.length = length;
    }

    public List<char> GetPath()
    {
        return path;
    }

    public int GetLength()
    {
        return length;
    }
}
```
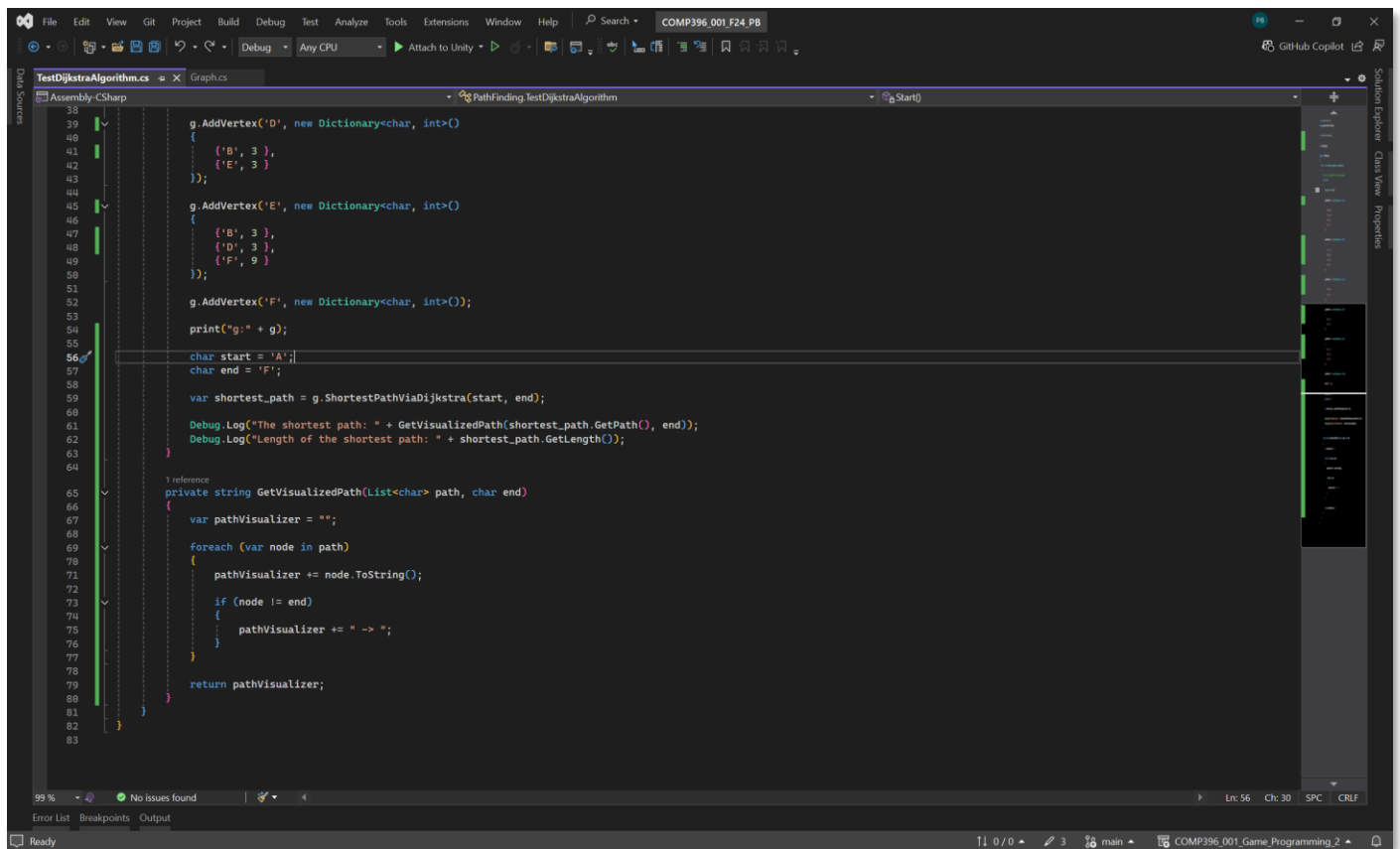
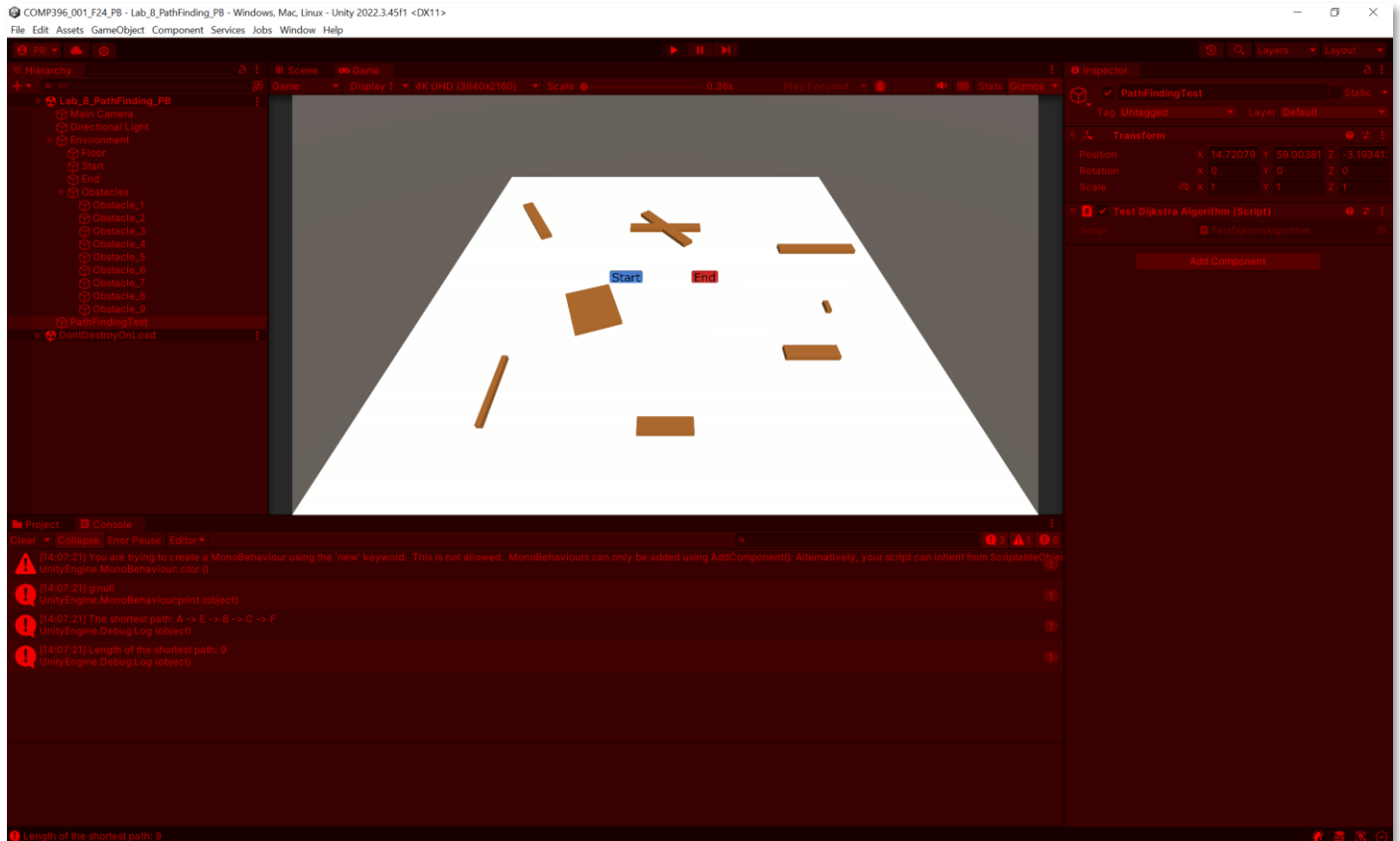*Fig 1.7: Screenshot showing Supporting structure of Dijkstra Algorithm PT- 2*

*Fig 1.8: Screenshot showing Execution of Dijkstra's algorithm for "**A**" to "**F**"– (Path and Length in console)*
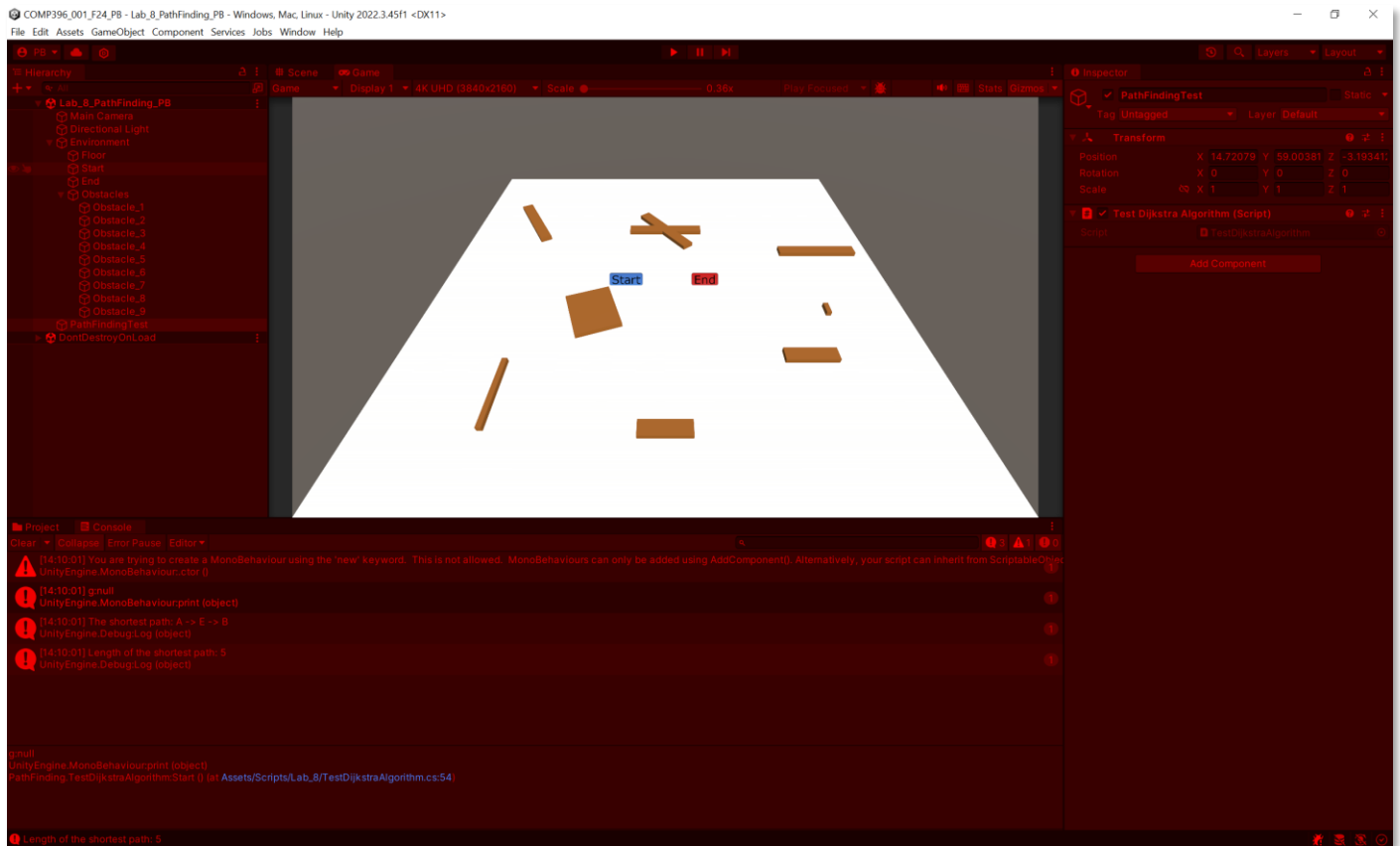
*Fig 1.9: Screenshot showing Execution of Dijkstra's algorithm for "**A**" to "**B**"– (Path and Length in console)*
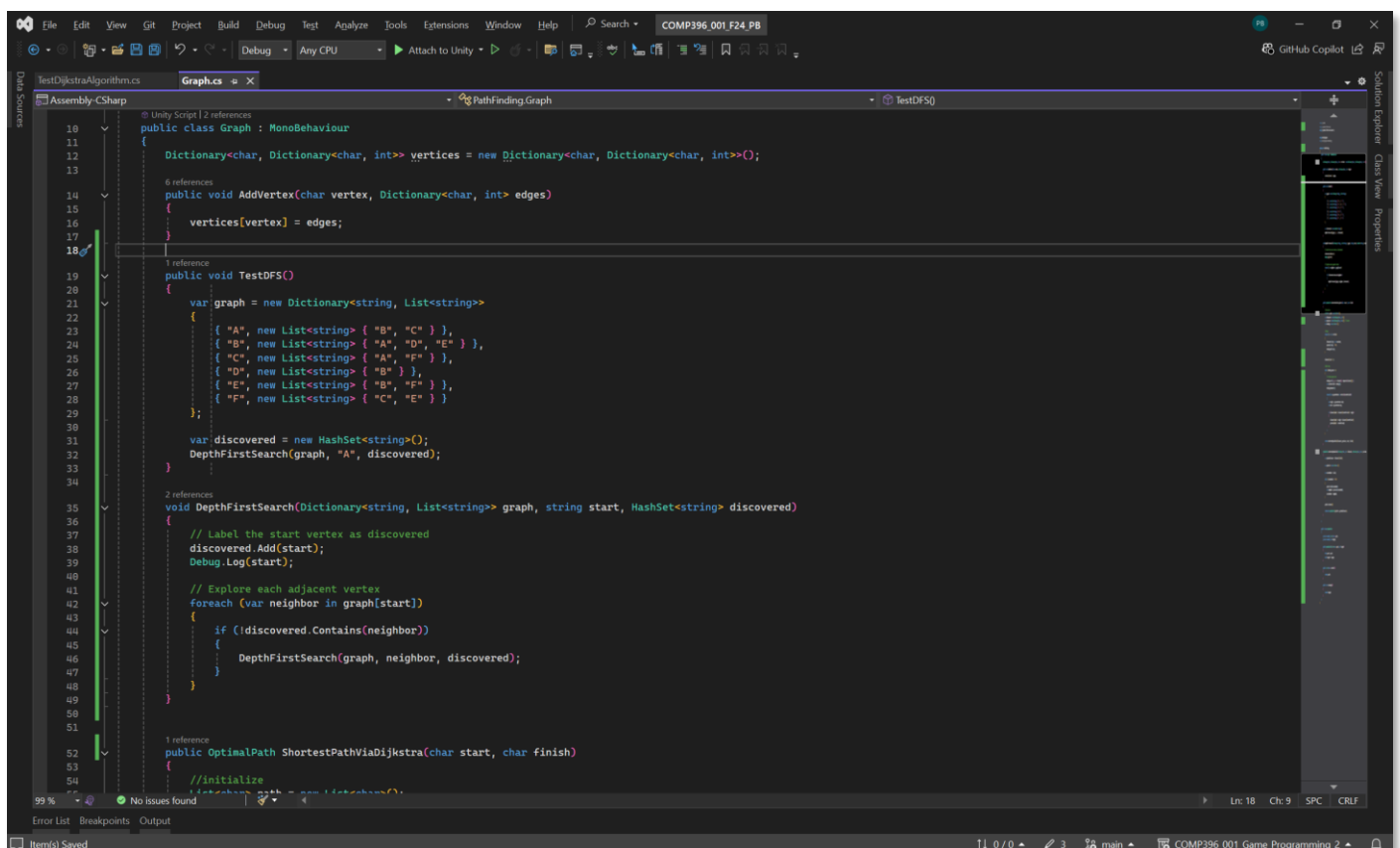

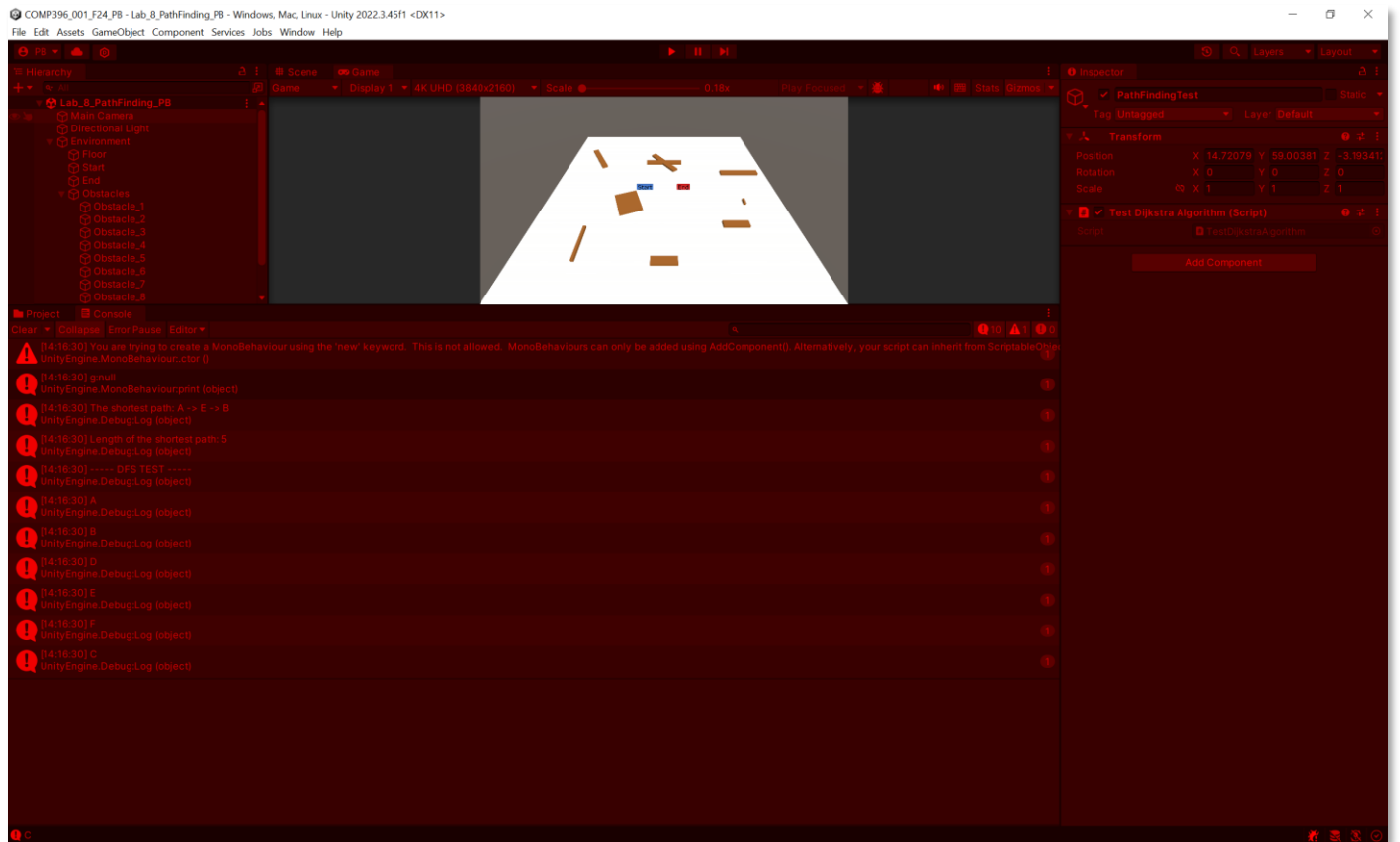*Fig 1.10: Screenshot showing BFS Search implementation*

Fig 1.11: Screenshot showing BFS Search Execution in console