Description: This project is an individual project designed to reinforce the theories and practices learned in class. You will be implementing the A* algorithm in Unity3D in a path planning context. As a first step, you will create a terrain centered at (-500,0,-500). The path cost from a start point to an given point will be calculated as the distance, plus the terrain height at the given point location. We will be using a grid system for the nodes of the graph, see Figure 1, and this grid can be visualized using the following c# code (attached to a gameObject, typically the main camera),

```csharp
void Start () {
    for(int i=0;i<10;i++) {
        for(int j=0;j<10;j++) {
            GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
            cube.transform.position = new Vector3(i*10f, 0.5f, j*10f);
            cube.transform.localScale = new Vector3(9.9f,1f,9.9f);
        }
    }
}
```

Figure 1: Visualization of the scene when running the grid code. The size of the grid in the X,Y, and Z dimensions are 10, 0.5, and 10. The cubes are scaled to 9.9 so you can see the divisions between grid elements.

The cost of the A* algorithm is computed as \( f(n) = h(n) + g(n) \) where \( g(n) \) is the cost to reach the node and \( h(n) \) is the estimated cost to get from the node to the goal. For this project, add to the \( g(n) \) cost, the height of the terrain. The height of the terrain at an arbitrary Vector3 can be computed as,

```csharp
Vector3 gpos = new Vector3(0,0,0);
float h = Terrain.activeTerrain.SampleHeight(gpos);
```

You can assume that the start point, or root of your tree, is position Vector3(0,0,0). You can also “hard code” the end point in your code as Vector(0,0,80).
**Deliverables** You are responsible for submitting,

1. A script file that computes the A* cost between the start point and the end point and visually “draws” the path in the scene. This drawing could be dynamically created primitives, or you could use the Debug.DrawLine method. Your A* can be computed once on start, or if you feel adventurous, you can compute the A* path every couple seconds.

2. Screen shots in JPG or PNG form that illustrate your method working with various terrain heights. Your scene should include multiple textures and trees. See Figure 2 for examples.

![A* example 1](image1.png) ![A* example 2](image2.png)

Figure 2: Visualization of the complete A* algorithm running in a scene.

**Guidance** Some concepts that could be helpful in this project. Use of the comparable interface if you choose to declare your own class of objects, i.e.

```csharp
public class GridObj : System.IComparable

and

public int CompareTo(obj: Object) : int
```

Another structure that might be helpful is ArrayLists or 2D arrays. They can be declared in the following way.

```csharp
public List<T> frontier = new List<T>();
public int[,] grid = new int[10,10];
```

If you are interested in computing the A* algorithm in real time every couple of seconds, you may want to look into the Invoke and InvokeRepeating methods.

**Submission** Submit the files on Blackboard.