Programming Assignment 2
Detailed Instructions

Overview

In this programming assignment, you're building a lazy professor graph traversal game.

To give you some help in approaching your work for this assignment, I've provided the steps I implemented when building my assignment solution.

Step 1: Build the Graph

For this step, you're building the graph.

I provided a stub for a `GraphBuilder` script, but I didn't attach the script to a game object. Attach the script to a game object in the scene so it will run when the game starts.

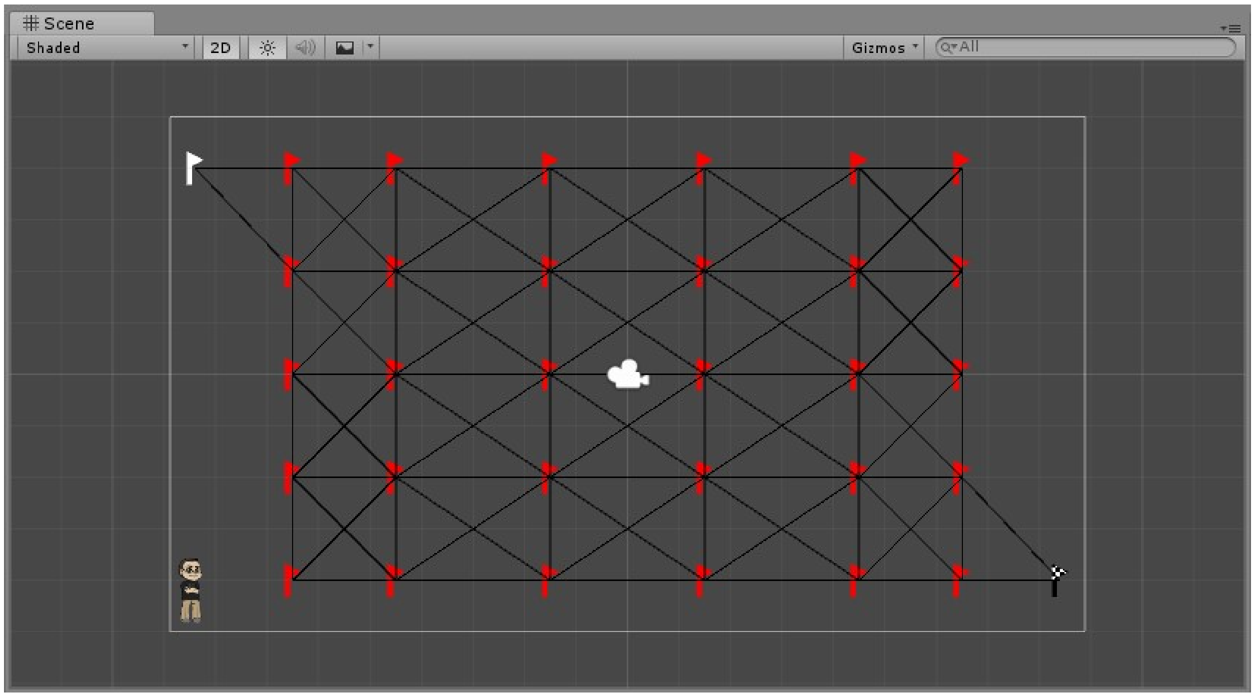
Add the code to the script as indicated by the comments in the `Awake` method. Using the `Awake` method ensure the graph is built before anyone (like the traveler) accesses it in their `Start` method.

Make sure you add the Start object, the End object, and all the Waypoint objects to the graph. You'll probably find the tags for those objects useful as you do that.

Add all the edges to the graph, making sure you only add an edge between two nodes if they're within 3.5 units horizontally and 3 units vertically of each other. Make sure the weight of an edge is set to the distance between the two nodes.

Attach the EdgeRenderer script I provided to the main camera. When you run the game, you should see all the edges for the graph drawn in the scene.

Your graph should look like the picture below.



Step 2: Find Shortest Path from Start to End

For this step, you're finding the shortest path (in other words, the path with the least cost) from the start to the end in the graph.

Although we know that breadth-first search guarantees that the first path we find will have the least number of edges, it doesn't guarantee that the first path we'll find will have the minimum cost in terms of the edge weights.

One pathfinding algorithm that's guaranteed to find an optimal solution in terms of total cost is called Dijkstra's algorithm (named after Edsgar Dijkstra, a giant in the Computer Science domain).

Dijkstra's algorithm is like breadth-first search, but we'll approach it by keeping references to graph nodes (waypoints) with some extra information in a sorted linked list, then build the shortest path by exploring each node exactly once. When we get to the target, we're done.

The discussion of Dijkstra's algorithm on Wikipedia is really good; in fact, I've modified the algorithm from Wikipedia and provided that detailed algorithm below for you to use as the basis for your `Search` method. I put this method in my `Traveler` class, since the Traveler is the game object that will follow the path.

```
Create a search list (a sorted linked list) of search nodes (I provided a
    SearchNode class, which you should instantiate with Waypoint. I also
    provided a SortedLinkedList class)
Create a dictionary of search nodes keyed by the corresponding graph node.
    This dictionary gives us a very fast way to determine if the search
    node corresponding to a graph node is still in the search list

Save references to the start and end graph nodes in variables

For each graph node in the graph
    Create a search node for the graph node (the constructor I
        provided in the SearchNode class initializes distance to the max
        float value and previous to null)
    If the graph node is the start node
        Set the distance for the search node to 0
    Add the search node to the search list
    Add the search node to the dictionary keyed by the graph node

While the search list isn't empty
    Save a reference to the current search node (the first search node in the
        search list) in a variable
    Remove the first search node from the search list
    Save a reference to the current graph node for the current search node in
        a variable
    Remove the search node from the dictionary (because it's no longer in the
        search list)

    If the current graph node is the end node
        Display the distance for the current search node as the path length
            in the scene (Hint: I used the HUD and the event system to do
            this)
        Return a linked list of the waypoints from the start node to the end
            node (Hint: The lecture code for the Searching a Graph lecture
            builds a linked list for a path in the ConvertPathToString
            method)

    For each of the current graph node's neighbors
        If the neighbor is still in the search list (use the dictionary to
            check this)
            Save the distance for the current graph node + the weight of the
                edge from the current graph node to the current neighbor in a
                variable
            If the distance you just calculated is less than the current
                distance for the neighbor search node (Hint: You can retrieve
                the neighbor search node from the dictionary using the
                neighbor graph node)
                Set the distance for the neighbor search node to the new
                    distance
                Set the previous node for the neighbor search node to the
                    current search node
                Tell the search list to Reposition the neighbor search node.
                    We need to do this because the change to the distance for
                    the neighbor search node could have moved it forward in
                    the search list
```

Hint 1: Build a smaller scene with a Start waypoint, an End waypoint, a few "normal" waypoints, and a Traveler. Do your testing/debugging on this smaller scene so you don't have to deal with so many waypoints as you make sure your code works. Be sure to attach your GraphBuilder script to the main camera in this scene.

Hint 2: Setting the ID (in the Inspector) for each waypoint in the scene makes it much easier to tell what waypoint you're looking at as you test/debug. I did this for you in the scene I provided, but you'll have to do it yourself in the smaller scene you build.

Hint 3: Write a method that converts the search list to a string so you can easily look at the order of the nodes in the search list. I didn't put every piece of information about each search node into the string when I built one, I just included the waypoint ID and the distance for each search node.

Step 3: Have the traveler follow the path

For this step, you're having the traveler follow the path you found from the start node to the end node. Move the traveler to the start node, then have them follow the path. Each time the traveler collides with a waypoint along the path, turn that waypoint green.

The Ted the Collector game from the Arrays and Lists lesson in the first module of the More C# Programming and Unity course (the second course in the specialization) uses similar ideas for moving a collector to a series of pickups. If you haven't taken that course yet, that game is also covered in Chapter 9 of my Beginning C# Programming with Unity book on Amazon. If you don't have the book, you can simply download the Ted the Collector code from http://www.burningteddy.com/Books/Default.aspx and look through it on your own.

Step 4: Stop drawing edges and blow up intermediate waypoints

For this step, you stop drawing the graph edges in the scene. You're also blowing up all the waypoints the traveler visited EXCEPT the start and end waypoints. This all happens when the traveler reaches the end waypoint in the graph.

For the graph edges, you should consider using the event system I've provided. I've provided an Explosion prefab that you should use as you see fit for blowing up the intermediate waypoints.

That's the end of this assignment. When you submit your assignment, make sure you include the "real" scene for your build, not your test scene.