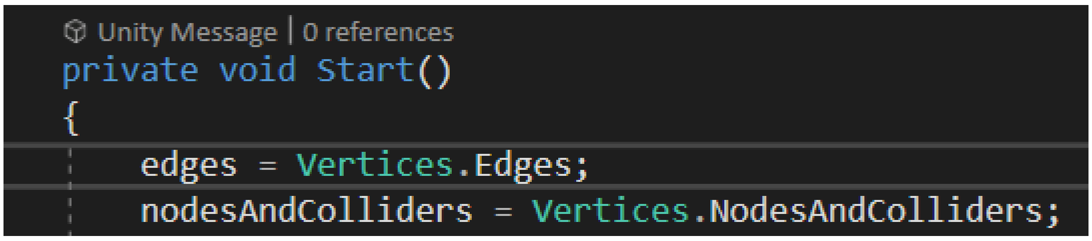
Node Map Implementation

The code consists of 3 main parts:

1. Data - the vertices class holds 3 lists (nodes, colliders and edges) and a dictionary. There is also the enum NodeStates.
2. Game manager is essentially an input handler, currently only handling mouse clicks on nodes using raycast.
3. Node controllers - consists of the classes “Node”, “ConnectingNodes” and “States”. It uses the data to handle the transition between node states.

When playing the Map scene, the Vertices class assigns the list values of nodes, colliders and edges that we populate in the Unity inspector to static lists as well as a static dictionary of the colliders and their respective nodes. This happens on “Awake”. When other classes need one of these lists, instead of creating similar lists and populating them in the inspector, they cache the Vertices list locally during “Start”. This is done in order to minimise external calls to the Vertices class and leave as much “work” as possible outside of the editor.



```
Unity Message | 0 references
private void Start()
{
    edges = Vertices.Edges;
    nodesAndColliders = Vertices.NodesAndColliders;
```

Next, the States class maps the colours of node states and the enum NodeStates to 2 dictionaries in “Awake”. The colours can be changed through the Unity inspector by the game designer. Using these dictionaries, the class manages the colours and states of the nodes as well as transitioning between them. In “Start” we set the state and colour of all the nodes to locked, except for the “root” node, which we set as open. The code also prevents a case of multiple roots.

The ConnectingNodes class populates each node’s “ConnectedNodes” list by checking for an overlap between the colliders of one edge and the 2 nodes it touches. It’s done after going through every edge in the list Vertices.Edges.

When the game manager detects a player clicking on a node (no matter its state). If it detects the player clicking on a node’s collider, it calls the “NodeClicked()” method in the class Node. This method either transitions a node to the complete state, or prints an appropriate message that the node clicked was not in the “open” state.

If the node transitions to “completed”, the Node class goes over all the connected nodes, and transitions only those that are locked to the “open” state.