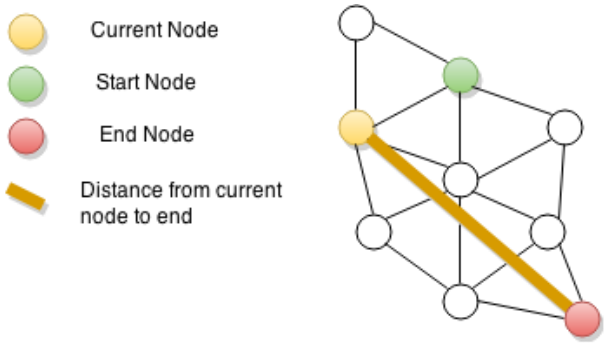
Exercise – AStar

In the Dijkstras shortest path exercise, we added a gScore value and had the open list sorted by gScore. AStar is a quick and simple modification the the dijkstras algorithm. When sorting the open list, we will prioritise nodes that are closer to our target end node.

Unlike the Dijkstras algorithm, AStar can only pathfind to a single known end location. Using this end location, we can calculate a huristic (hScore)

The huristic is usually the squared distance from the current node to the target node.



The open list should be sorted by the "FScore" which is another variable stored within the node. The FScore stands for "Final Score" which is calculated by adding the nodes "GScore" and "HScore" together.

You'll also need to update the Node structure to store the hScore and fScore as well.

```
struct Node
{
	// pointer to the origional node in the graph
	Graph::Node *pNode;

	// information required for pathfinding
	BFS_Node	*parent;
	int		degreesOfSeporation;
	float		gScore;
	float		hScore;
	float		fScore;

};
```

Let's take a look at the altered pseudo code. You'll notice the highlighted lines are those that have changed from the Dijkstras algorithm.

```
Procedure FindAStar(startNode, endNode)

    Let openList be a List of Nodes
    Let closedList be a List of Nodes

    Add startNode to openList

    While openList is not empty

        Sort openList by Node.fScore

        Let currentNode = first item in openList

        // Process the node, do what you want with it. EG:
        if currentNode equals endNode
            break out of loop

        remove currentNode from openList
        Add currentNode to closedList

        for all Edges e in currentNode
            Let n = e.connection
            Add n to openList if not in closedList
            n.gScore = currentNode.gScore + e.cost
            n.hScore = distance from n to endNode
            n.fScore = n.gScore + n.hScore
            n.parent = currentNode


    // Calculate Path
    Let path be a List of Vector2
    Let currentNode = endNode;
    While currentNode != NULL
        Add currentNode.position to path
        currentNode = currentNode.parent
```

Exercise 1:

Implement the above pseudo code. You should be able to see in the demo provided a visual for how the different algorithms select from the open list.

Exercise 2:

You may have noticed with your above implementation that some paths returned are not always the shortest, and that some have some obscure kinks which is obvious to us that its not the shortest path. This is due to only selecting the nodes with the shortest fScore. Its possible that we select a

node whose children have already been processed, and, that it may be faster to get to that child node from this current node than the one which first added it to the open list. In this case, we want to make one more chance to our algorithm, so that we update that child nodes gScore, hScore, fScore and parent to come from this current node, rather than the one that originally processed it.

Update your algorithm to contain the below fix.

```
Procedure FindAStar(startNode, endNode)

    Let openList be a List of Nodes
    Let closedList be a List of Nodes

    Add startNode to openList

    While openList is not empty

        Sort openList by Node.fScore

        Let currentNode = first item in openList

        // Process the node, do what you want with it. EG:
        if currentNode equals endNode
            break out of loop

        remove currentNode from openList
        Add currentNode to closedList

        for all Edges e in currentNode

            Let n = e.connection
            Let gScore = currentNode.gScore + e.cost

            If n.gScore < gScore or n.parent is null
                n.gScore = currentNode.gScore + e.cost
                n.hScore = distance from n to endNode
                n.fScore = n.gScore + n.hScore
                n.parent = currentNode

            Add n to openList if not in closedList


    // Calculate Path
    Let path be a List of Vector2
    Let currentNode = endNode;
    While currentNode != NULL
        Add currentNode.position to path
        currentNode = currentNode.parent
```