Classes in blue

Methods in red

How it starts path finding:

1. In the Start method of Character a path is requested from the PathFindingAgent. As far as the character is concerned this just means: “Hey this is where I want to go, do whatever you can to create a path”.
2. In the PathFindingAgent.RequestPath method, we check if the player is on the ground. If the player is on the ground then we start the path request; otherwise we store where we want to go and wait until the character is grounded before we calculate a path. This class doesn’t know how to calculate a path either, so it asks PathFinding.RequestPathInstructions. PathFinding is our main man and does actually know how to calculate a path.
3. So now we’re in PathFinding.RequestPathInstructions. This method creates a PathData class. This contains everything we need to know to calculate a path and then calls PathFinding.FindPath. This is where the path calculation actually starts (we made it!).

How the path finding is calculated:

I’m going to use the word ‘node’ often in this section. A node is just a representation of a tile that a player can move through as part of a path.

A number of costs are used to define nodes. These costs are used to determine whether a node is added to the final path list with lower costs being more desirable. They are just numbers that represent certain things:

- ‘C’ is a predetermined weight based on whether the node is a ground node or jump node (ground nodes are more desirable than jumping nodes and have a smaller number meaning they are more likely to be selected as part of a path).
- ‘G’ is C + the previous nodes G score. This is the cost to move from the start node to this node. This cost is basically used to find the path with the least number of jumps. If there is a straight line along the ground from the start node to the end node, this cost is used to find it.
- ‘F’ is how much it costs to move from this node to the end of the path. A lower cost means it is closer to the goal node. This is calculated based on the distance of this node from the final node.
- ‘H’ is the total cost of this node, calculated using G + F. The H score of nodes are compared to determine which node to add to the final path list.

1. So we’re at the beginning of PathFinding.FindPath. We create some variables that are important:
    a. openNodes: a list of nodes that we are currently considering for inclusion in the final path list.
    b. closedNodes: a list of nodes that have already been considered.
    c. pathNodes: the final path.
2. We add the startNode to the openList. This will be the first tile considered.

3. Then we enter the main loop (starts "`while (openNodes.Count > 0)`"). This is the loop that actually calculates the path.
4. We loop through all openNodes (this only contains the startNode at the moment) and we find the node with the lowest h score (the h score is the total cost of the node). Again this will be the startNode as it's the only one in the list but we'll be adding more to the openList in future. We set this node to be out current node and remove it from the openNodes and add it to the closedNodes list so we do not use it again.
5. Now we have a current node we need to check all of its neighbours to find the one that is closest to the end goal and the least expensive to get to. So we loop through all neighbours for our currentNode and calculate the g (cost from start to end node), f (cost from this node to end of path), and h (total cost of node, f + g) costs. We add each of these nodes to the open list (so the one with the lowest cost will become our current node). With this done we go back to step 4 but this time there are four more nodes in the list.
6. Steps 4 and 5 repeat until we reach our end node.
7. With the end node reached, we are out of the main loop and then all we do is create the path by adding all nodes to the pathNodes list (see the comments for more details) and creating a set of instructions. The instructions are used by the player to know whether they can walk to a node or they have to jump.
8. The path is then sent to the player to be processed by calling PathFindingAgent.ReceivePathInstructions.

What happens once has been sent to the player:

1. PathFindingAgent.ReceivePathInstructions stores the path as a 'waitingOrder'. This just means it is processed in the Update method when the character is grounded.
2. In the Update method of PathFindingAgent, once the order has been received, a number of variables are set so the player will start following the path and the currentOrders (received from the path finder) are changed slightly under certain conditions (see comments for details).
3. Now that the PathFindingAgent is setup to use the path, whenever PathFindingAgent.AiMovement is called (by AiController.GetInput, which is called in the FixedUpdate method of the Character class) the character follows the set of instructions. It also does a few things to correct for the character jumping too far etc.