* The map data is read in from the "FileParser" which obtains the map dimensions and raw data. The data is processed during the "File.ReadAllLines" function where each character is converted into the enumerator-space which makes sharing and accessing easier. Unfortunately the original data pointing to special characters is lost for better maintainability and readability.
* Once the “Level” has called “FileParser” it updates the “TileMeshSettings” for maps dimensions and the per tile resolution using the tilesheet. This data is used for both the mesh and Texture2D generation.
* Any tile sheet can be used and for simplicity sake should be configured within “Level” by adding it to the “TileEnumMapper” which essentially creates a key pair value between the sprites type and its index within the tile sheet. This makes it easier to access tiles by type rather than name all of which is stored in the “TileSheet”.
* Returning to the “Level”, after the initial setup it finally draws the tiles onto a Texture2D which is splatted onto the mesh as a material. Unfortunately this process takes 3 or more seconds on larger map dimensions and optimisation have been considered and partially implemented:
  + Less draw calls by doing line by line rather than tile by tile which would require sprite batching per row.
  + Detecting the most significant tile covering the map i.e. there is more grass then we would set the entire texture to that sprite initially.
  + Unfortunately the chunking system is only partially implemented and does not function correctly. This would have solved the support issue for larger maps exceeding standard texture sizes (only discovered in the labs) where the tile resolution was too high. But loading an 8 or 16 tile resolution will work for large maps even on low end machines.
* While the “FileParser” still retains the map data the “Level” handles retrieving and setting tiles which these function should have been moved to “FileParser”.
* Although the chucking does not work it is still used since it creates one chunk which holds the current map. Both the “TileMap” and “TileMesh” generally handle the map interaction as a whole as pose to per tile. It has functionality that entities can use to transform themselves into the tile map space (Alignment).
* The A\* search algorithm used the same principles for a cost from start along best path and the estimated total cost from start to goal which is done per nodes (tile). (It’s the same A\* we all do with opening and closing nodes and returning the result after)
* A\* was initially chosen as a quick means to add AI path planning and get entities moving but during test seems to work reliable well and fast considering. Alternatively we would have preferred to use D\* mixed with region chunking for best optimisation as well as diversity for maps since A\* does absolutely terrible on maze/less open world maps. Another main drawback of the A\* algorithm and any best-first search really is its memory requirement, since at least the entire open list must be saved.
* Each entity handles its’ own path planning using the generic static A\* class. Thus allows several paths to be planned at once. Initialising an entities plan is simple, specify where it should end up as a vector2 (pre-validation prevents the entity from going beyond the map and non-walk-able tiles). Additionally a speed is also specified and determines how fast the entity moves along the path since this is done with Coroutines rather than loops so it will run on a separate thread preventing game backlogs and stalling while it tries to plan and progress along its’ path.