Complex game systems evaluation

Changes to the original brief:

The original brief for my complex game system was for it to be a graphics card-based pathfinding solution, as well as a graphics card-based node graph generation tool. While creating the system I found it extremely difficult to get the astar pathfinding algorithm to work effectively on the graphics card, upon searching for a solution I made the change from the graphics card to multithreading on the CPU, this would provide an increase in performance compared to using the main thread on the CPU whilst being easier to implement than the graphics card solution.

The other change that was made to the original brief was to the node generation system, on the brief it was going to be on the graphics card. This was changed to be on the CPU as it is not a live updating solution extra processing time is not an issue as it saves the node graph to a scriptable object and does not have to be processed twice for the same environment layout.

Problems encountered:

The immediate problem that was encountered when trying to implement the a star pathfinding algorithm to the graphics card through a compute shader was creating a class that could be sent to the graphics card. This was a problem because it could not have a reference to itself in the class for a parent which a star requires for calculating path costs.

The other problem with getting the class structure was the graphics card not allowing non blittable types, this means that the class must require either using C# type marshalling which I could not get to work, or to only have standard system types such as int and float. The node class that I had after the mesh generation had references to parents and to an edge class giving the connections to other nodes, so the graphics card would not allow this, this was one of the reasons why I decided to switch to multithreaded cpu pathfinding so there were more memory options.

The last issue that lead to the switching from gpu to multithreaded cpu was that the graphics card would not accept a non-static number for array sizes, In my generation I wanted the user to be able to select the connection amount for each node, so they could have more options, because of the graphics card not allowing a input to define array sizes, the two options were to have a larger array size resulting in wasted memory on a large scale to accommodate larger array sizes for connection amounts or to swap to multithreading cpu, when looking at these options during development this was when I did swap as the graphics card had to many problems for me to solve in the time frame.

The initial problem when starting development on this system was to generate a node graph off a mesh on an object. As I could not find any resources on creating on a system like this, I decided to look through unity's documentation. The lack of information on how to make a system like this was a problem that was encountered because I had no way of knowing what worked, when I eventually got all the mesh points in world space to use for pathfinding, I noticed that objects that have mesh

points underneath them were used as well (such as a cube), because of this the agents would not move properly. This was solved by doing a y check to see if any other node was within a certain distance of that node in the x and z coordinates then returning the node with a greater y.

This system for doing a y check on nodes proved to not work however as it would not take rotation into account and when an object was rotated the system would delete needed nodes. This was solved by getting the mesh objects normal direction and doing a dot product check of the negative normal and the direction from one node to the other. This would take rotation in account and be able to still remove nodes that would be underneath the object that agents should be walking over.

Performance of the system:

Unfortunately, while the systems node generation works well and performs with decent speed for the generation, the accompanying Ai agent system is slower than its prebuilt unity counterpart.

In a small test scene with unwalkable walls and a flat surface, one thousand unity navmesh agents were able to find random paths along the environment at an average of 256fps on low obstacle avoidance.

Compared to my system that would average 56fps performing the same task of obstacle avoidance with one thousand agents. However, my system can outperform the unity counterpart, when my agent's avoidance is turned off, in the same test my agents averaged 307fps with one thousand agents pathing to random points.

The conclusion for the performance is my agent's obstacle avoidance is inadequate and in standard tasks it would be simpler for the navmesh component to be used for the average user. However, in certain cases where "dumb" ai are needed my system would be a viable alternative use.