Ship Programming Ideas:

Ship Component System:

A basic component will consist of these main data types:

* String - Component Name
* String – Component Description
* Color - Primary Colour
* Color - Secondary Colour
* Color - Accent Colour
* Float - Durability
* Float – MaxDurability
* Mesh – ComponentMesh
* GameObject – shipGameObject
* Enum – ComponentType
* Float – PowerContribution – Can be negative or positive, all power contribution on all components of a ship added together to see overall power net.

A component can have the componentType enum values:

* Small – Utility Features – support based items.
* Medium – Armour, shields, components used in collaboration with large components.
* Large – Large attachments such as weapons or drills.
* Expansion – Passive ship size increase, for example extra weapon bays etc or storage.
* Upgrade – Small benefits and ability increases to large/medium components.
* Special – Unique components such as hull, reactor etc.

This component can be expanded with new functionality. This can include the addition of multiple new variables such as the reactor could have a heat value assigned to it. Components will also have the base functions/events (which can be inherited and modified):

* OnActivate – When this particular component is activated
* OnDeactivate – When this particular component is deactivated
* OnShipStart – When the ship is started
* OnShipShutdown - When the ship is shutdown
* OnShipUpdate – When the ship is running, basic update function

The ship will make use of this event system in order to make the most of the modularity of this component system.

Ship Thruster Particle Position Mapping

When particles are created, they must be given a starting position. This can be achieved in a variety of ways. Using point caches for this purpose has been tiresome as every ship prefab would require a new point cache as a point cache cannot be exposed as a parameter in the VFX Graph used for the thruster particles. The better way to do this while making it modular is to recreate the texture map that is baked into the point cache as this can be exposed as a parameter and this means it can be assigned to the ship prefab and the particles will map to the correct positions.

Based on the tests completed surrounding the “Set Position from Map” Node in the VFX Graph, the attribute map works in the following way:

* Each of the colour channels (R, G, B) represents a corresponding axis (X, Y, Z) used for positioning within a normalised cube (1x1x1). For the thrusters, only the R and G colour channels will be used as the surface is flat, and does not make use of the Z axis, but for 3D objects, this would not be the case.
* The number of pixels in the image represents the number of particle spawn points with their RGB value setting the position of that particle within the normalised cube. If you wanted 64 particle spawn points, you would create an 8 x 8 texture. This value would then be multiplied with a user defined spawn rate which could be based on the velocity or a constant for each ship type.
* Because the shape of the thrusters is not a primitive shape, some calculations will be needed in order to work out ranges of valid spawn points within the normalised values, this could be by working out the bounds of the mesh passed in.

With this in mind, the editor extension should have the following workflow:

1. The user enters the whole ship mesh and it creates a sub mesh for triangles that have the user defined thruster material that is also input. This expands functionality to other uses and removes the need for me to create a thruster sub mesh in blender myself for each ship, thus saving time and preventing strange import transforms from blender. The main mesh now becomes the thruster sub mesh.
2. The program calculates the size of the whole position map texture by calculating the surface area of all thruster triangles in this sub mesh. This will be done by having the user input a density value or having a constant value across all thrusters.
3. Sub meshes are created for every thruster that is non-touching, this can be calculated by working out whether vertices share an edge.
4. The number of pixels for each thruster sub mesh is calculated by using the percentage of the surface area that thruster provides. This should help ensure an even distribution of spawn positions for particles.
5. For every pixel for each thruster sub mesh, random positions are calculated that conform to the faces of the thruster mesh. The particles should not spawn outside of the mesh face.
6. From here, the calculated positions are placed into their respective colour channels (more on this above) and the pixel colour is calculated and applied in the position map texture.