**Background Research**

**Assets and Animation**

**What type(s) of animation will our game need?**

Many objects will not be animated at all, such as planets. Other objects may have very simple animations, such as towers and even ships. Therefore, we could just use simple asset switching to create animations. However, if more advanced animations are needed there are a few ways to do this.

**Boids:**

Boids are a type of simple flocking artificial intelligence. They behave similarly to birds and other flocking animals. These may be useful for simulating the behavior of large amounts of enemies/enemy ships where flocking is a good approximation for their behavior. One advantage of this over other methods is that it is very simple to implement and requires only a few rules. However, a big disadvantage is the lack of control inherent in the design and its properties of emergent behavior. Simulating complex attacks and maneuvers would be difficult, although not impossible, to do with Boids. As such it is not recommended for use in the Quasar Engine.

**Forward Kinematics/Inverse Kinematics:**

Kinematics is a method of describing the motion and position of objects in 2D or 3D space. It is useful in the field of animation as a way of succinctly and robustly describing motion. Forward Kinematics involves calculating the position of the end effector (the arm/edge/leaf etc.) of the object based on the angles and positions of all the joints. Inverse kinematics is the opposite and is the process of calculating all the angles and positions of the joints based on the position of the end effector. This can be used to animate the objects we have in our game. Say we want the wing of our ship to move in and out based on speed. This can be simulated with a simple hinge joint. And then we can manually decide the end position of the wing and have an inverse kinematics system calculate the required positions and angles of all the other joints we use in our model to achieve this. One key advantage of this over Boids is that it allows granular flexibility over animating our models and we can animate complex attacks and maneuvers much easier. However, this comes with the disadvantage that the system is much more complex to implement. Another disadvantage is that if it is not constrained it can lead to very unrealistic looking animation. This is, however, less important for the artificial constructs we would be animating than biological creatures. We have preexisting FK and IK code from **Animation and Simulation** which then negates the disadvantage that it is complex to implement.

**What kind of assets will our engine need to deal with?**

A wide variety of assets are required for our game engine to create the game we envision. These include, but may not be limited to

* audio assets (mp3, wav)
* video assets (mp4, avi)
* animation files (bvh)
* script files (.c, .lua, bespoke format for our scripting? )
* Art assets (.png, .jpg, .psd, .obj)
* Vulkan/Project files (.vcxproj)

How these are handled is more a focus of Eason’s work on File Systems. However, this is briefly discussed here. One method is to create individual readers/writers for each file type. From the list, however, we can see how many file types we may need to handle and so practically writing this many parsers is not practical.

Therefore, using existing parsers for each of these file formats makes the most sense. As well as this, deciding on the best format for the needs of the game to prune out unnecessary parsers at an early stage is also prudent.

**Where to get/How to create the required assets?**

There are many free sources online for many of the assets we need. There are also tools that we can use to create many of the assets. Audio assets can be created using software such as **Audacity.** This records the audio from a microphone and then allows many alterations to be made to attributes such as the pitch, tempo and so on. Another audio tool for simple game audio is **Jsfxr**, this tool was created specifically for game jams and has several preset audio types such as Lasers and coin pickups and then allows customization of various parameters to adjust this to the project's requirements. There are also several websites that provide free game audio such as **Mixkit** and **FreeSound**. However, their library of game specific sounds is limited. The game envisioned does not require very complex sounds and the simple beeps of **jsfxr** are likely sufficient.

Art assets can be created using **Photoshop** or one of the various free alternatives such as **Gimp**. These provide advanced photo editing features and none of the simple assets we need. Textures and other 2D assets can be created using either of these. There are also various online websites that provide free assets such as **Poliigon**. However, many of these are too realistic for the game. As such the recommendation is to use a simple photo editor to create any 2D assets.

3D models will also be needed for the project. Since our planets will be voxel based these 3D models will be simple primitives. One option would be to allow the creation of these directly in engine by using **Glut** or using OpenGL's render functions to create cubes. If OpenGL is being used as our rendering engine, then this simplified 3D model creation significantly. However not all models will be this simple. Ships, enemies and towers for example are likely to be more complex. For these purposes using a dedicated 3D modeling package like **Blender** is recommended. Blender is a free, relatively simple and easy to use 3D modelling software. There are other options such as **Maya,** but these are much more complex to complete the rendering process and other setup than blender. And so given our limited time and opportunities to learn new skills for the project it is not advised.

Video files are not likely to play a big role in the project but where they are needed using a screen recorder such as **OBS** is likely sufficient. Cutscenes are out of scope. Vulcan has extensions that support video encoding and decoding and so this is something to consider when choosing between Vulcan and OpenGL.

**How will assets be managed (Storage, preloading, caching etc.)?**

In engines such as **Unity** assets are not loaded in until they are included in the scene. Instead, they are visible from the current asset directory and can be dragged and dropped in as needed. This could be done by periodically probing an “asset” directory and scanning for new files. This then requires less memory to be allocated to the program. However, if a large asset is dragged in the slowdown from creating and allocating the required memory may be significant. As such preloading assets or using caching is a good idea. For small assets when they are detected in the asset folder loading them in would be wise. This could also be done with large assets but should be done asynchronously in that case to prevent slowing down the rest of the engine in the meantime.