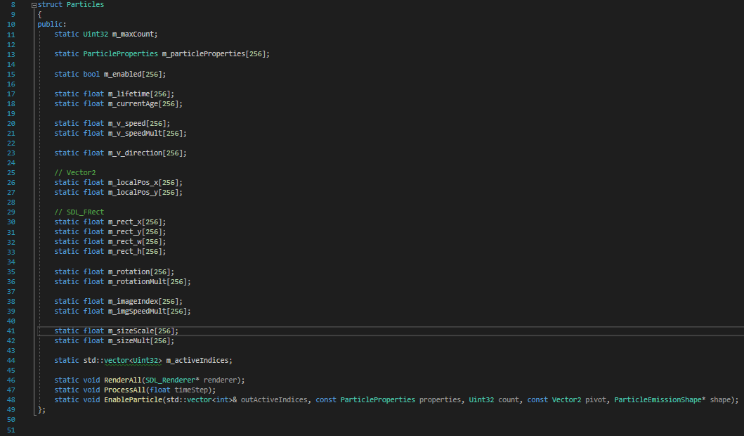
DOD Space Shooter

This is my space shooter game. Unfortunately, due to me having misunderstood some of the core concepts in the beginning of this course, there wasn’t much in terms of DoD that I ended up implementing. But I am much wiser now, and one system *did* manage to end up in the final product; a particle system that consists of multiple arrays representing each “instance’s” properties.

**The Particle System**

The particle system in this game runs on parallel arrays that when aligned vertically each represent a property of a single particle. When a particle system is looking to play an effect, it will call EnableParticle, where the struct looks through its own indices to find an index where **m\_enabled** is *false*. From there, the particles that are available will be assigned the particle system’s ParticleProperties, and the particle multithread takes care of processing their movement until their lifetimes are up, disabling them.

In retrospect, had I not misunderstood the concept of DoD from the start, I could have well designed this game in a way more data-oriented fashion. I could have had a system where all objects in the game are ordered in a similar fashion to this particle system, with texture pointers aligned with each other, and same with their positions, rotations, collision radii *(wouldn’t even need to create/delete collider classes in this scenario!)*, behavior-related stuff and so on, skipping OOP almost entirely. Done right, it could very much benefit the CPU’s cache, as the number of cache misses could be greatly reduced due to lack of reallocation and change in block sizes.

Had I designed the game this way I could’ve also tried utilizing intrinsics for aspects like object speed/position updates, which at best could reduce the number of CPU cycles for transform updates by as much as 8 times; in a scene with 20 active objects, this may potentially even rival the number of cycles it takes for the CPU to gather something from main memory!