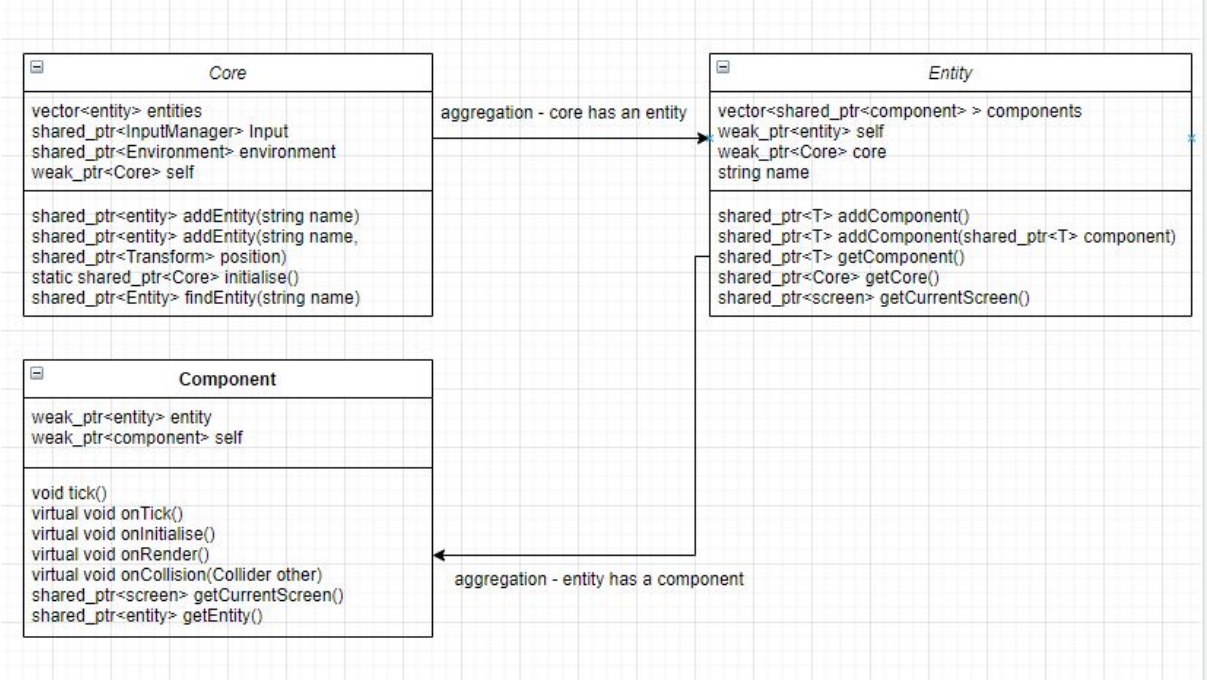
In this project I have designed a bare bones game engine system that is sufficient to build simple 3D games. The system works entirely with code so is not hugely friendly to non programmers however I have often found that GUIs can complicate systems and make it harder to make precisely what you want. Code offers much more precision than a GUI.

The program has been structured with a central core unit that contains a list of all the variables needed in the scene. This unit is where the main game loop occurs and performs operations that require every entity in the scene. In order to run the program the initialise function in core must be called to create the object. Then the user can create all the entities and components they wish to have in their scene in main(). Once they've added everything they wish to have they must call core->start(). This will begin the main game loop.

This engine makes use of an entity component system. All objects in the scene are entity objects which have a list of components attached to them. These components are what run the methods entities use and are where the user can add their own code. Doing this gives the user the ability to code any behaviours they like and apply them to any entity by simply adding the component to it. Components are one of the only parts of the engine to make use of inheritance as all additional components must inherit from the component base class so they can be added to entities.



The above diagram shows the relations in the overarching component entity system. Core has a list of entities and entities have a list of components. The

system is built up using aggregation over inheritance and smart pointers are used to navigate the hierarchy. The component base class has a number of virtual functions which are designed to be overwritten. These allow the user to code in behaviours for these specific events such as once every tick or when a collision occurs.

The specifics of what the engine does and each part of it are contained in the documentation for the engine. It is broken up into all the premade objects that are in the source code and what their methods and attributes are. The documentation has been focused on an end user and as such has not been complicated by adding back end functions the user won't need to make use of.

In making this project I had to research collision detection techniques and decided on using GJK (Winter, 2020) for its efficiency. This collision detection system works by creating a simplex from subtracting the most extreme points on the 2 objects being tested in an arbitrary direction. The direction is recalculated and more points added until the simplex is a 3D tetrahedron. Then if the origin is contained within this simplex then there is a collision. Else there is not. This method is based upon the Minowski difference which was calculated by subtracting every point on one object from every point on the other. While this method works GJK is an optimisation of it which speeds up the process massively.

Overall this engine has been built to be very basic in order to focus on refining the parts it does have. The engine does make building games somewhat simpler by handling the visuals and scene management for the user allowing them to focus on the parts of the game that are less tedious to create. I feel this engine could be improved a large amount given more time to add new features however I made the choice to focus on the features that are required for a functional engine so that I may make these parts effective.

Winter, 2020, GJK: Collision detection algorithm in 2D/3D [online], Winter's Blog, available from: https://blog.winter.dev/2020/gjk-algorithm/ [accessed 01/01/2021]