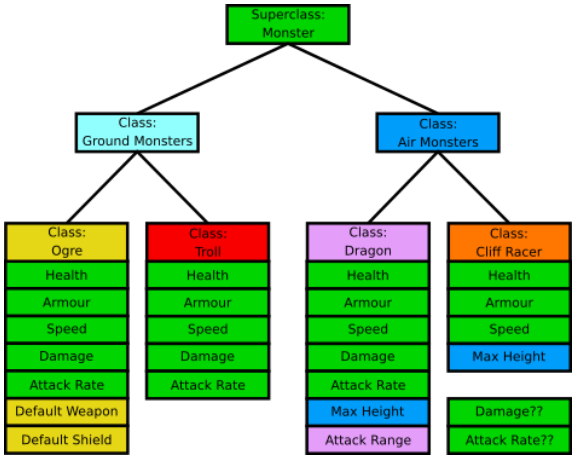
Entity-Component-System introduction:

An Entity-Component-System aims to separate the data that represents an object (or “entity”) from its behaviour. The traditional approach to representing objects in a scene is an object-oriented one; each object has its own class that may inherit traits from another class. For scenes with moderately large numbers of objects, this can become very complicated, very quickly [1].



The diagram above shows a possible configuration of classes in the object-oriented approach. Green attributes are inherited from the superclass – all monsters have health, armour, speed, damage and attack rate. But what if we wanted a monster that didn’t attack at all? Or a monster that walked on the ground but could also fly?

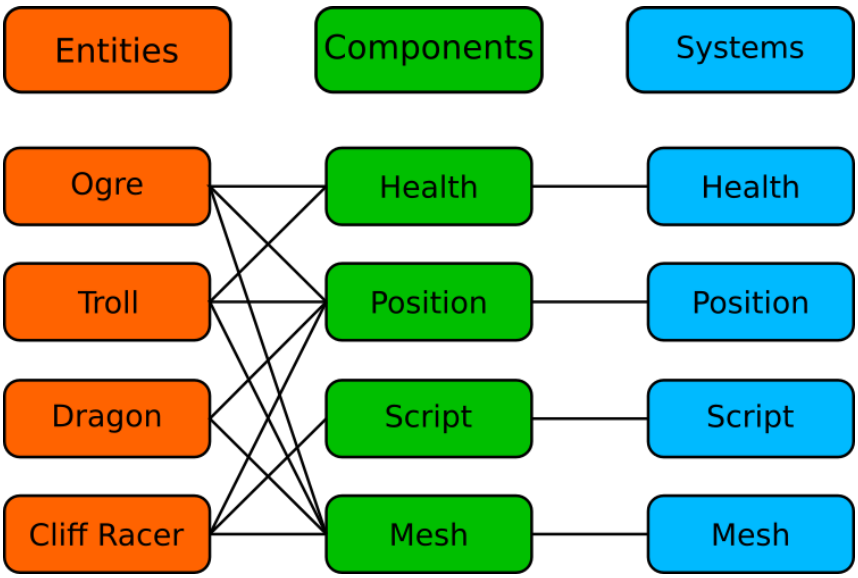
Another difficulty with the object-oriented approach is the relative difficulty in classifying objects based on their attributes. For example, one might want to draw up a list of all monsters that carry weapons. Perhaps keeping track of a list of all object IDs that carry weapons would be an appropriate solution. An Entity-Component-System can handle this by default.

A simple definition of Entity-Component-System:

Entity – the parallel in the object-oriented approach would be the object itself. Entities are containers; they have a unique ID and a collection of components [2].

Component – the object-oriented equivalent would be class attributes and variables. Components are containers for data [2].

System – the object-oriented equivalent would be class methods. Systems act upon components of a specific type, defining the behaviour of an entity [2].



To define an entity, create a unique ID for the entity, then assign components to it. Each component will maintain a list of entities that contain that component, and each entity will maintain its own list of components, containing the relevant data.

When a system acts on a component, it will draw on the list of entities with that component, and act on them. For example, if health needs updating every game tick, the health system will be called every tick, which will then trigger an update of the health attributes of all entities with health. The script system may just batch run all the scripts associated with entities in one go.

The main thing to notice here is that entities are separate from components, which are separate from systems. This avoids all the problems mentioned with the object-oriented approach.

Does the engine being developed need an Entity-Component-System?

One of the benefits of using an ECS is that it can handle large numbers of objects without the code becoming needlessly complicated. Since the game idea in this project will involve waves of enemies, probably in large numbers, and many different types of entity or object, this would simplify the code in the long run. The system doesn't need to be very complicated, as more functionality can be added later.

Third-party libraries:

The EnTT library:

- Open-Source (MIT license)
- Well-documented
- Requires a compiler with support of at least C++17.
- Header-only

Link: **https://skypjack.github.io/entt/**

The EntityX library:

- Open-source (MIT-license)
- Well-documented
- Requires a compiler with support of at least C++11

- Not header-only

Link: **https://github.com/alecthomas/entityx**

Implementation:

The ECS is a data-oriented approach, and so it will require careful use of data structures and memory management [3].

The ECS will need to keep track of many data structures representing entities and components. Access to these data structures will be through an interface that provides basic functionality such as adding and deleting entities, adding and deleting components from the game and from specific entities, and linking systems to component types. More functionality will be added as needed.

References:

[1] – Härkönen, T., 2019. Advantages and Implementation of Entity-Component-Systems.
[2] – Hollmann, T., 2019. Development of an Entity Component System Architecture for Realtime Simulation.
[3] – Nilsson, D. and Björkman, A., 2019. *Profiling a Prototype Game Engine Based on an Entity Component System in C+* (Doctoral dissertation, BS thesis, Malmö University).