ECS
Entity Component System

Para quem já trabalha com desenvolvimento avançado de sistemas deve conhecer o modelo MVC que significa: Model View Controller que é uma forma de separar os dados para o banco de dados(*Model*), os dados que serão manipulados(*Controller*) e os dados que serão vistos(*Views*).

O ECS(Entity Component System) é similar ao MVC, ou seja, é um padrão de arquitetura de software, mas é utilizado mais comumente para desenvolvimento de games. Em português, ECS significa: **Sistema de Componente e Entidade** . Um **ECS** segue o princípio da "*composição ao invés de herança*" . Em resumo, um ECS consegue separar dados e lógica para que sejam mais fáceis de manipularmos .

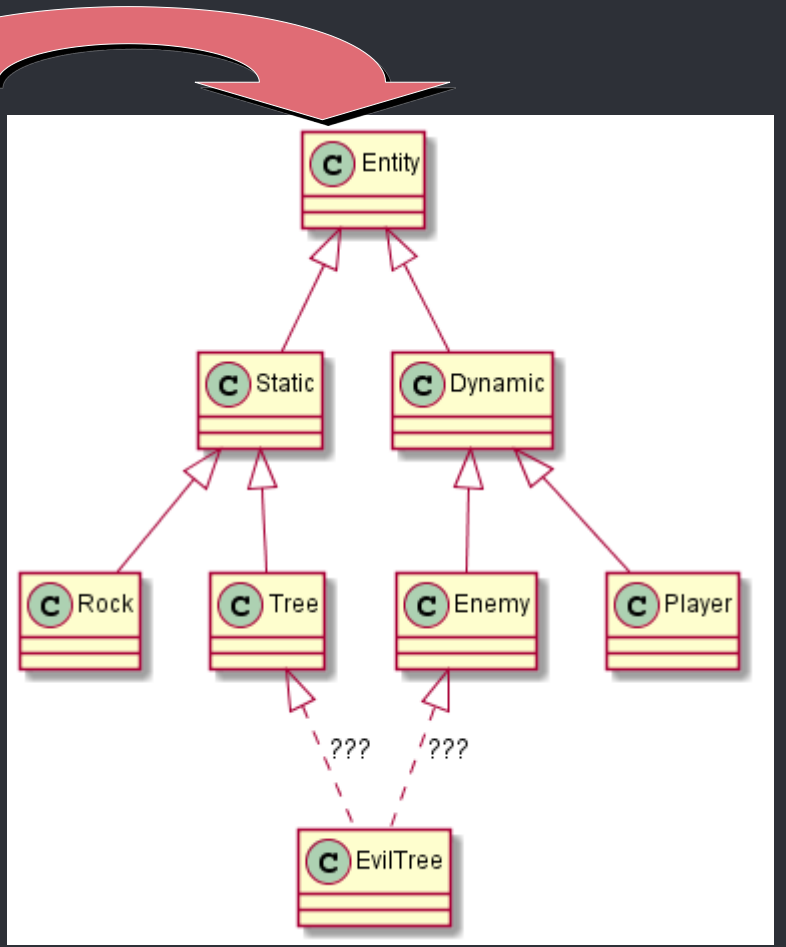
O ECS se divide da seguinte forma:

- **Entity** é tudo aquilo que for: Jogador(Player), Plataforma(que se move), *Míssel*(Bullet), Inimigo(Enemy),... Ou seja, **tudo aquilo que tenha uma posição e possa se mover**(podemos interagir com ela), as Entity em ECS são compostas por Components(Componentes).
- **Component** é tudo aquilo que for: Posição(x e y), Velocidade, Limites de um objeto(Bounding Box), Vida, Dano, Arma, Mapa,... Ou seja, Components são dados. Tudo aquilo que podemos atribuir à uma Entidade(Entity).
- **System** é **tudo aquilo que for código/lógica/função** que move os objetos: Escutar teclas, Mouse, Som, Física, Gravidade,...

A **ECS** *se difere da Programação Orientada à Objetos*, ela é considerada uma **Programação Orientada à Design** .

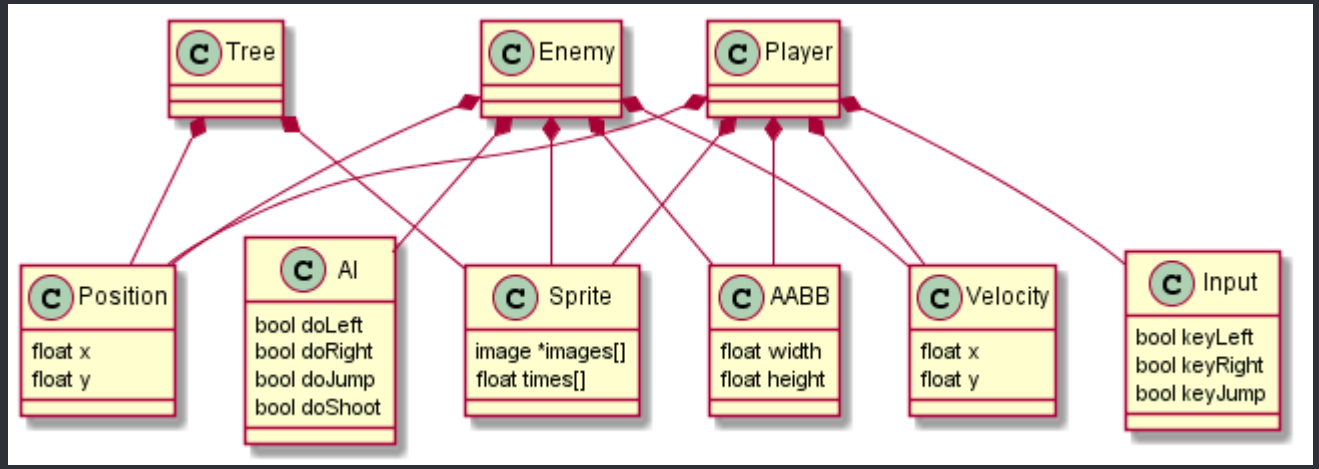
Qual a diferença?

Veja esse diagrama, ela é um estruturação de uma **POO/OOP**



Note que a árvore ela é estática na vida real, *mas em Games podemos ter uma árvore que ande, que seja um inimigo, ou algo do tipo, logo, ela deveria herdar características da classe* Dynamic também . Imagine um game com um monte de coisas diferentes, o quão trabalho será para você incluir algoritmos.

Já uma **Orientação à Design(OOD)** seria similar à imagem abaixo:



Isso diminui significativamente o números de classes específicas para determinada ação .

Em alguns games vocês poderão ler o termo **EC** somente, isso quer dizer que é diferente de *ECS*, ou seja, os **Components** também inclui a lógica do game.

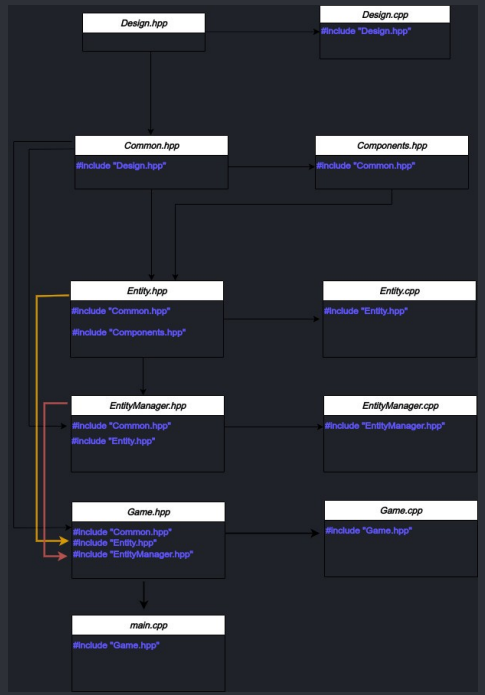
Em termos de arquivos a arquitetura se resumiria em:

- main.cpp
  - Systems.cpp
    - Entitys.cpp
      - Components.cpp

> Alguns desses arquivos podem ter sub-arquivos, por exemplo: EntityManager.cpp , que seria uma de gerenciar nossas entidades.

Assim como MVC, dificilmente você vai notar essa hierarquia de arquivos tão bem definidas como nesse exemplo, geralmente todos esses arquivos tem bastante outros arquivos, principalmente games que possui alta complexidade, como: GTA, Fortinite, Free Fire,... e entre outros .

Além disso também há uso de outro classe para manipular os dados de um objeto, utiliza-se uma classe de vectors, geralmente chamada de Vec2, que representa o desenho do código, ela faz uso de functors para identificar os operadores estabelecido pela palavra chave operator como vimos no Curso de C++ Moderno Avançado e a organização dos arquivos dessa maneira exemplo:




```cpp
class Vec2 {
  public:
    float x, y;

    Vec2();
    Vec2(float xin, float yin);

    bool operator == (const Vec2 & rhs) const;
    bool operator != (const Vec2 & rhs) const;

    Vec2 operator + (const Vec2 & rhs) const;
    Vec2 operator - (const Vec2 & rhs) const;
    Vec2 operator / (const Vec2 & rhs) const;
    Vec2 operator * (const Vec2 & rhs) const;

    void operator += (const Vec2 & rhs);
    void operator -= (const Vec2 & rhs);
    void operator *= (const float val);
    void operator /= (const float val);

    float dist(const float val);
};
```

Vamos ver um exemplo disso na prática nos próximos vídeos.

Fonte da imagens utilizadas:
https://gamedev.net/articles/programming/general-and-gameplay-programming/understanding-component-entity-systems-r3013/