**Tìm hiểu về Data Driven Programming**

## DDP và OOP

Mình sẽ lướt qua định nghĩa, chỉ chủ yếu đi về việc áp dụng chính của các kiểu lập trình khi có yêu cầu bài toán đưa ra (Requirement)

### 1/ Lập trình hướng đối tượng OOP

Kiểu lập trình này thường gần gũi hơn với yêu cầu người dùng và cuộc sống quanh ta. Thường cách để xác định logic của kiểu này sẽ là

* Xác định các đối tượng (objects) trong yêu cầu bài toán
* Xác định các thuộc tính, dữ liệu và quan hệ giữa các đối tượng trong yêu cầu bài toán

### 2/ Lập trình hướng dữ liệu DDP

Với DDP thì yêu cầu bài toán được ra cần được nhìn với khía cạnh là dữ liệu (Data). Trong DDP, kể cả các function xử lý cũng sẽ được gắn liền với dữ liệu, lúc này việc thiết kế data structures của sẽ tốn nhiều thời gian hơn để xây dựng. Dẫn đế việc số lượng dòng code & thời gian thực thi chương trình sẽ được giảm tải, do các xử lý logic đã được gắn trong dữ liệu rồi

### 3/ OOP vs DDP

**OOP** là kiểu lập trình truyền thống của dân lập trình viên

**Ưu điểm**

* Cách thức thiết kế sẽ là ánh xạ giữa yêu cầu bài toán thực tế và ngôn ngữ lập trình nên việc thiết kế tương đối đơn giản
* Bản thân OOP hỗ trợ rất tốt các kỹ thuật Kế thừa, Đa hình nên việc phát triển & mở rộng code cho các ứng dụng này rất linh động

**Khuyết điểm**

* Đối với các yêu cầu chương trình về xử lý dữ liệu, lọc & chuyển đổi dữ liệu thì OOP sẽ tốn nhiều thời gian thiết kế
* Khi data model hoặc data logic thay đổi thì cần người lập trình viên phải chỉnh sửa và maintain data trên chương trình

**DDP** thường được sử dụng nhiều với các úng dụng xử lý , chuyển đổi dữ liệu và các ứng dụng lập trình front-end hiện tại

**Ưu điểm**

* Logic nằm data nên việc thay đổi logic yêu cầu bài toán có để cho chính End User hoặc Business End tự thay đổi và maintain. Miễn không thay đổi structure của data
* Performance ứng dụng sẽ được rút ngắn do xử lý logic đa phần đã điều hướng trong chính dữ liệu

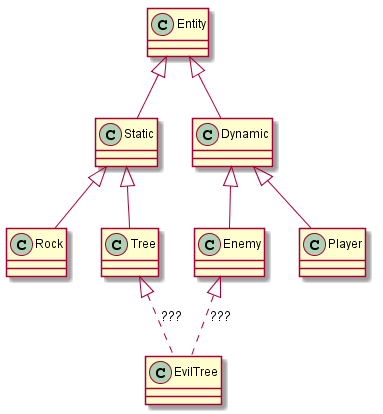
**Khuyết điểm**

* Khi dữ liệu quá lớn, việc thiết kế logic dữ liệu sẽ là thách thử lớn
* Cần sự phối hợp và am hiểu lớn về yêu cầu bài toán. Đôi lúc người lập trình và business user phải ngồi với nhau để làm cấu trúc dữ liệu

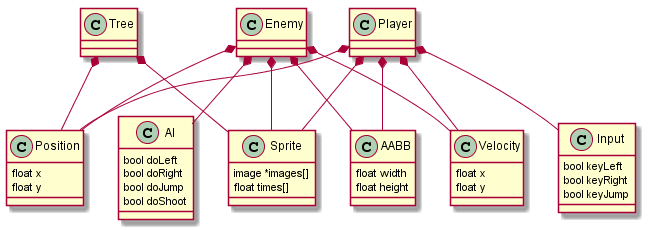
# Hiểu biết cơ bản về hệ thống Component - Entity – Systems

# Intro:

Một cách truyền thống để xây dựng một game là sử dụng lập trình hướng đối tượng(OOP). Mỗi thực thể trong game là một đối tượng(Object), được tạo ra từ khuôn mẫu lớp(class) cho phép mở rộng thực thể qua tính đa hình. OOP khá hiệu quả đối với những game cỡ nhỏ có ít đối tượng trong game. Khi game lớn có nhiều đối tượng, mỗi đối tượng có nhiều chức năng, đặc điểm khác nhau việc áp dụng OOP sẽ dẫn tới một nhược điểm rất lớn. Đó là khi bạn cần thay đổi 1 đối tượng nào đó sẽ kéo theo một loạt các đối tượng liên quan phải thay đổi theo, điều này làm mất nhiều thời gian và khó tránh khỏi việc bỏ quên update đối tượng liên quan nào đó. Trong phát triển game việc phải thay đổi đối tượng nào đó sẽ thường xuyên xảy ra theo thời gian.



Để giải quyết vấn để này, hệ thống Component - Entity - Systems ra đời. Các lập trình viên bắt đầu xây dựng các đối tượng thông qua các thành phần(component) thay vì kế thừa từ một đối tượng gốc(base object) . Điều này mang lại những lợi ích lớn so với lập trình hướng đối tượng OOP : Dễ dàng để thêm 1 thực thể phức tạp mới. Dễ dàng để mở rộng một thực thể mới bằng cách thêm mới các thành phần của chúng. Dễ dàng sửa lỗi, bảo trì và nâng cấp. Mỗi component sẽ có một chức năng riêng biệt và không lặp lại. Do đó ta dễ dàng khoanh vùng chức năng nào còn thiếu sót để hoàn thiện nó.



## 1.Component

Mỗi component giống như một cấu trúc dữ liệu struct. Nó không có phương thức chỉ có khả năng chứa dữ liệu. Các component là độc lập với nhau. Để quản lý tập trung các component thành danh sách trong entity thì các component phải có 1 điểm chung là cùng kế thừa từ một asbtract component. Và component là đối tượng nhỏ nhất để tạo nên entity.

## 2.Entity

Tất cả đối tượng trong game đều là những thực thể (entity), và mỗi thực thể là duy nhất (có một UniqueID).Một entiy là tập hợp nhiều component có cùng UniqueID. Và những đặc điểm (property), trạng thái (state), phản ứng (behaviour) của các entity được quyết định bởi các Component.

## 3.System

Là những những module con, hoạt động độc lập nhau, xử lý một “behaviour” nào đó trong game. Như đã nói ở trên các component chỉ chứa dữ liệu và không có phương thức gì. Để một thực thể có khả năng di chuyển công việc đó được thực hiện bởi system. Điều đó có nghĩa là mỗi component sẽ có một system để thực hiện hành vi của nó.

Component - Entity - System(CES) là một phương pháp lập trình rất hay, đáng để tìm hiểu và sử dụng. Các dự án lớn và đa phần các Game engine, framework đều được hiện thực bằng phương pháp này để tăng tính dễ bảo trì và nâng cấp. Tuy nhiên nó cũng có một số nhược điểm mà chúng ta cần phải lưu ý :

* Phí tổn cho các System : Như các bạn biết, trong vòng Update(), các System sẽ phải loop qua tất cả các Entity và Update cho các Component, chi phí cho việc này sẽ tăng cao khi số lượng các Component và System nhiều lên. Tất nhiên người ta cũng đã có giải pháp khác phục, đó là lập các Group. Các System sẽ chỉ loop qua các entiy chứa các component mà system đó “lo”, việc nhóm các entity đó lại gọi là tạo các Group. Ngoài ra còn có các giải pháp sâu hơn cho PhysicSystem, RenderSystem – những system chính và “ngốn” performance nhất.
* Liên lạc giữa các System: Các system không phải lúc nào cũng độc lập hoàn toàn với nhau, có những system cần phải nhận thông tin từ system khác. VD: khi một monster entity bị chết, thì UISystem cần biết để cập nhận thông tin UI chẳng hạn. Người ta giải quyết vấn đề này bằng ObserverPattern.