

FSM, 클래스 구조

FSM?

Finite State Machine

FSM?

Finite State Machine

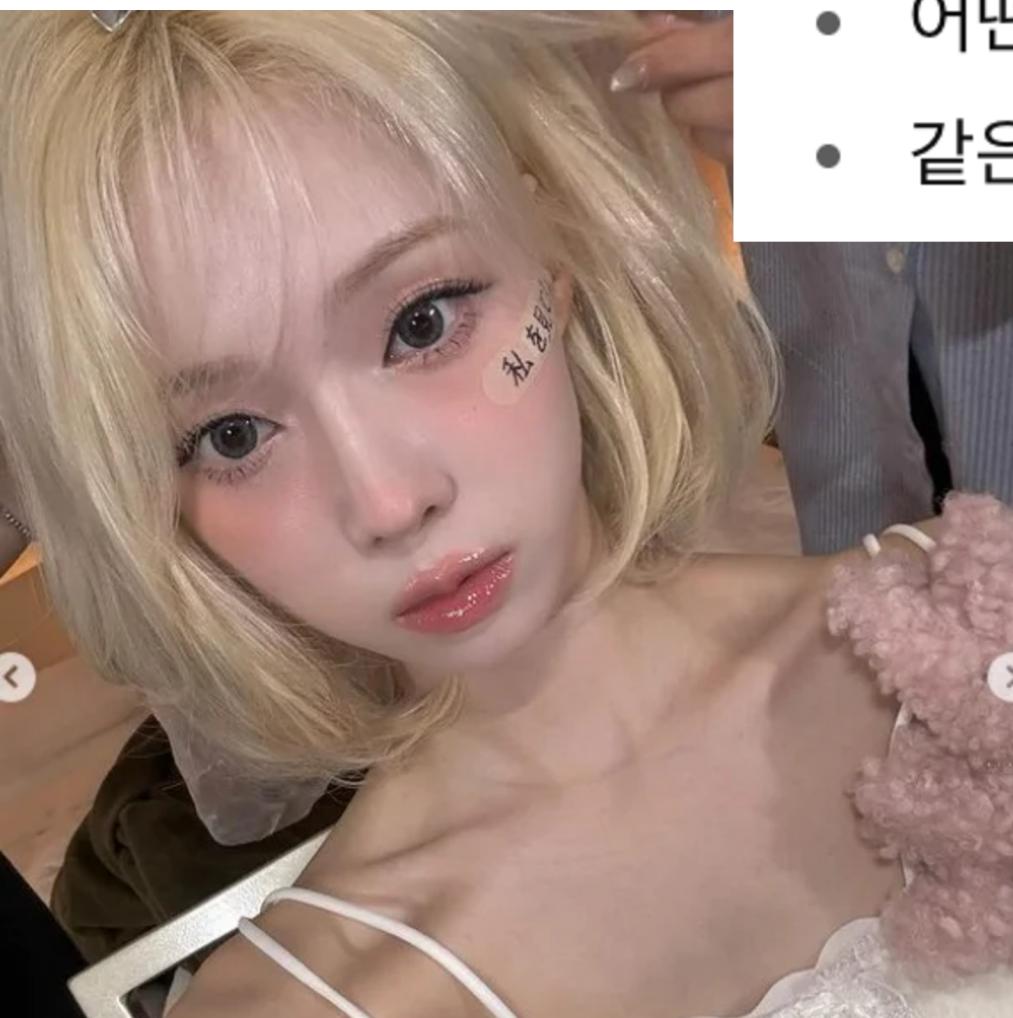
유한 상태 기계

상태?

📌 **FSM에서 "상태(state)"의 정의**

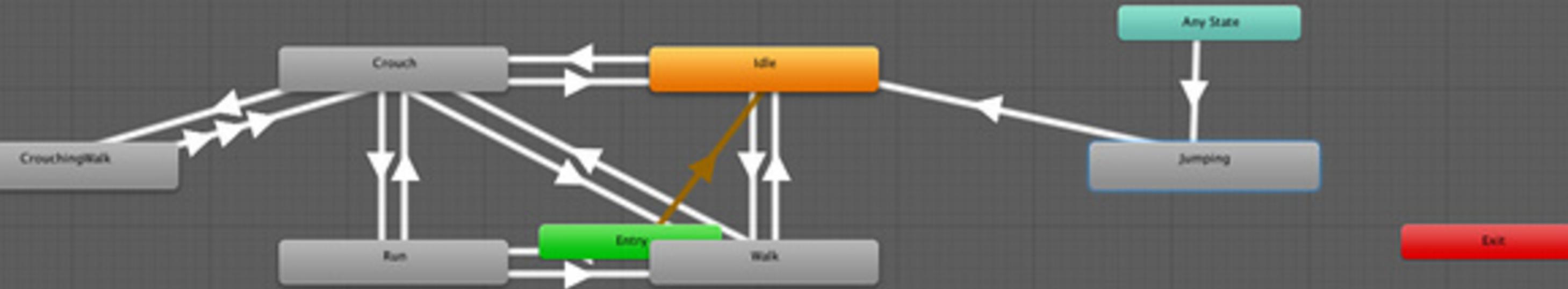
👉 현재 시스템이 가지고 있는 상황·조건·위치를 표현하는 하나의 단위

- 어떤 시점에서 시스템이 어떤 동작을 할 준비가 되어 있는지를 나타냅니다.
- 같은 입력을 받아도, 현재 상태에 따라 다른 출력·동작이 나올 수 있습니다.



FSM의 예시를 찾아보자

Base Layer



노드들을 하나의 클래스로 정의하자



실습 그그

Draw.io를 접속하세요

각 노드들은 어떤 기능을 해야 할까?

각 노드들은 어떤 기능을 해야 할까?

각자 특징에 맞는 일을 해야겠죠?

클래스로 나누는 이유는
은닉성을 가지기 위해

클래스로 나누는 이유는
상호 의존성을 낮추기 위해

SOLID 원칙 (객체지향 설계 5원칙)

이 5가지 원칙은 변경 및 확장에 유연한 소프트웨어를 설계하기 위한 지침으로, 코드의 가독성과 유지보수성을 높입니다. 

1. 단일 책임 원칙 (SRP: Single Responsibility Principle)

- 클래스는 단 하나의 책임, 즉 단 하나의 이유로만 변경될 수 있어야 합니다.

2. 개방-폐쇄 원칙 (OCP: Open/Closed Principle)

- 소프트웨어 요소(클래스, 모듈 등)는 확장에는 열려 있어야 하지만, 수정에는 닫혀 있어야 합니다. 즉, 기존 코드를 수정하지 않고 기능을 확장할 수 있어야 합니다.

3. 리스코프 치환 원칙 (LSP: Liskov Substitution Principle)

- 자식 클래스는 언제나 부모 클래스를 대체할 수 있어야 합니다. 즉, 부모 클래스의 객체가 사용되는 곳에 자식 클래스의 객체를 사용해도 문제가 없어야 합니다.

4. 인터페이스 분리 원칙 (ISP: Interface Segregation Principle)

- 클라이언트는 자신이 사용하지 않는 인터페이스에 의존하지 않아야 합니다. 불필요한 인터페이스를 만들지 않고, 클라이언트에 꼭 필요한 기능만 제공하는 작은 인터페이스로 분리해야 합니다.

5. 의존 역전 원칙 (DIP: Dependency Inversion Principle)

- 자주 바뀌는 구체적인 클래스에 직접 의존하기보다는 추상적인 인터페이스에 의존해야 합니다. 이를 통해 변경에 유연하게 대처할 수 있습니다. 

상태들은 만들었는데 어떻게 사용할까?

상태들은 만들었는데 어떻게 사용할까?

그러한 상태들을 전환, 업데이트를
하는 context(주체)객체를 만들자

실습 그 7



애니메이터 할때 기분은 어떤가요?

애니메이터 할때 기분은 어떤가요?

노드들 간선 연결하고 조건 추가하고

exit설정하고 등등

정말 귀찮죠?



FSM의 단점

확장성이 안좋다.

(새로운 상태가 생기면 노드끼리 연결을
계속 손봐야함)

FSM의 단점

일반적인 if, switch case의 단점인
복잡함이 그대로 전해짐

FSM의 장점

일반적인 if,switch보다
의존성이 낮아서 버그발생률이 낮음

FSM의 장점

구현이 정말 쉽다.

FSM의 장점

직관적이여서 상태가 많지 않은 경우
적절하게 사용 가능함.

FSM의 사용 예시

멀티플레이 게임에서 매치메이킹, 등
네트워크 관련 상태들을 다룰 때 사용
(오프라인, 온라인, 매치메이킹 중, 게임
중 등등)