Unity. Assembly Definition

정의

컴파일러 / 어셈블러 / 컴파일

이전에 배웠던 컴파일러와 어셈블러에 대한 이해부터 다시 되새겨 보자. (어셈블러 언어 vs 컴파일러 언어 vs 인터프리터 언어 (velog.io))

- 어셈블러 (Assembler) : 어셈블리어 (MOVE, ADD 등등 - 기계어 (0/1) 로 변환하기 쉬움.) 로 작성된 프로그램을 기계어로 바꿔주는 프로그램.
- 컴파일러 (Complier) : 고급 언어로 작성된 프로그램을 번역해서 어셈블리(어) / 기계어로 바꿔주는 프로그램.
- 모든 프로그램은 기계어로 변환되어 컴퓨터가 읽을 수 있도록 변환되어야 함.

코드를 실행하는 것을 우리는 보통 **컴파일** 이라고 하는데, 이는 "어떤 언어의 코드 전체를 다른 언어로 바꿔주는 과정"을 의미한다. 엄밀히 말해서, 컴파일러는 이러한 컴파일 과정을 수행해주는 프로그램 / 소프트웨어를 의미한다.

또한, 모든 High Level Language 의 코드는 컴파일러를 통해 기계어로 변환되어야 한다. 그 과정에서 어셈블리어 (= 어셈블리) 로 변환하는 과정을 거칠지 말지는 컴파일러의 몫.

유니티에서는 컴파일러가 어셈블리로 변환하는 과정을 거치는 듯 하다.

Assembly Definition

Unity의 "*Assembly Definition*"은 프로젝트 내의 **스크립트 코드**를 **별도의 어셈블리**로 구성할 수 있게 해주는 기능이다.

기본적으로 Unity는 모든 스크립트를 **하나의 어셈블리 파일로 컴파일**한다. 그러나 프로젝트가 커지면 컴파일 시간이 길어질 수 있고, 코드의 모듈화나 재사용성이 떨어질 수 있다. 이런 문제를 해결하기 위해 Assembly Definition을 사용할 수 있습니다.

Assembly Definition의 주요 기능

1. **컴파일 속도 향상**: 별도의 어셈블리로 코드를 분리하면, 변경이 발생한 어셈블리만 다시 컴파일하면 되므로 전체 컴파일 시간을 줄일 수 있다.
    - 특히, Unity 에서는 Script 가 하나 변경될 때마다, 그것을 Load 하거나 등록하는 시간이 존재하는데. 이것이 컴파일 때문에 오래 걸린 것일 경우, 이 시간을 줄일 수 있을 것으로 보인다.

2. **코드 모듈화**: 특정 기능이나 컴포넌트를 별도의 어셈블리로 분리하면, 그 어셈블리를 다른 프로젝트에서도 쉽게 재사용할 수 있다.
3. **의존성 관리**: Assembly Definition 파일에서는 다른 어셈블리에 대한 **의존성**을 명시적으로 설정할 수 있습니다. 이를 통해 코드의 의존성을 명확하게 관리할 수 있다.

사용 방법

1. Unity Editor에서 `Assets` 메뉴를 열고 `Create > Assembly Definition`을 선택하여 새 Assembly Definition 파일을 생성한다.
2. 생성된 `.asmdef` 파일을 선택하면, Inspector 창에서 해당 어셈블리의 설정을 변경할 수 있다. 여기에서는 어셈블리의 이름, **의존성**, 플랫폼 별 설정 등을 지정할 수 있습니다.
3. `.asmdef` 파일을 생성한 후에는 **해당 폴더와 하위 폴더에 있는 모든 C# 스크립트가 해당 어셈블리로 컴파일**됩니다.
4. **다른 어셈블리에서 이 어셈블리를 참조하려면, 참조하려는 어셈블리의 `.asmdef` 파일 설정에서 `Assembly Definition References` 섹션에 필요한 어셈블리를 추가해야 한다**. => 이게 Assembly Definition 이 포함되어 있는 경우, 다른 코드에서 이를 참조하지 못했던 이유임.