\* 미리 컴파일된 헤더 사용해서 빌드 속도를 올렸다.

\* 가변 인자 템플릿 부분 특수화를 사용해서 입력 키 시스템을 구현했다.

\* Modern C++을 적극 활용하려고 하고 있다. 새로운 표준 헤더가 추가되면 사용 해보려고 공부한다.

\* C++17 filesystem 헤더를 사용해서 경로 관리자를 설계했다.

\* 싱글턴 클래스 템플릿을 만들어서 사용하고 있다. 멀티 쓰레드 환경에서도 안전하게 사용하기 위해 mutex 헤더를 추가하고 call\_once를 사용 중이다.

\* chrono 헤더를 사용해서 타이머 시스템을 설계했다.

\* 타일 맵의 마지막 인덱스 참조를 방지하기 위해 limits 헤더의 epsilon을 사용 중이다.

\* random 헤더를 사용해서 메르센트위스터 방식의 난수 시스템을 사용하고 있고, 정규 분포와 베르누이 분포를 사용하고 있다.

\* memory 표준 헤더를 추가해서 unique\_ptr, shared\_ptr, weak\_ptr을 사용하고 있고 경우에 따라 어떤 스마트 포인터를 사용해야 하는지 알고 있다. 프로젝트에서 사용하는 모든 포인터들을 모든 포인터를 스마트 포인터로 사용 중이다. weak\_ptr을 최소한으로 사용해 lock 비용을 줄이고 있다. 또한, enable\_shared\_from\_this 인터페이스 클래스를 사용해 필요한 곳에 자기 자신의 control 블록을 전달해 주고 있다.

\* texture, animation, sound 같은 데이터들은 각각 매니저들을 통해 shared\_ptr로 프로토 타입을 갖고 있도록 설계했고, 오브젝트들은 해당 데이터를 마찬가지로 shared\_ptr로 들고 있도록 설계해서 메모리 낭비가 낭비되는 것을 방지했다.

\* 엑셀 .csv 파일로 texture, animation, sound 정보들을 한번에 읽어 들이도록 처리했다. 데이터가 추가되더라도 코드 수정하고 컴파일할 필요없이, csv 데이터 시트 상에서 처리하는 것이 가능해졌다.

\* DirectXTK 오픈소스를 내 프로젝트에 연동해 사용 중이다. Audio 헤더를 사용해서 XAudio2 시스템을 추가했다.

\* Core 모듈은 모든 매니저들을 초기화하고 게임 루프를 실행시킨다.

\* Core의 로직에서는 입력처리, 업데이트, 충돌검사, 렌더링 등의 작업이 수행된다. 각 함수는 다음과 같이 위임된다. Core -> Scene -> SceneComponent -> Layer -> Object -> Collider

\* 충돌 매니저는 오브젝트간의 충돌을 관리한다. 현재 Point, Rectangle, Circle, Pixel 충돌체가 존재한다.

\* 각 충돌체에는 Enter, Stay, Leave 방식의 3가지 콜백 함수를 들고 있도록 설계했고, 각 콜백 함수를 등록할 때 functional 헤더의 function 클래스를 활용하고 있다. 또한 람다를 활용해 전역 함수나 클래스 정적 멤버 함수, 클래스 멤버 함수 모두 동일한 인터페이스로 콜백 함수를 등록할 수 있게 하였다. 이 덕분에 bind로 this, \_placeholder 등을 전달해서 여러 버전의 함수를 오버로딩 해야하는 것과 템플릿 함수로 만들어야 했던 것을 피할 수 있게되어 가독성을 높일 수 있었다.

\* Pixel 충돌체는 이미지를 이용해 처리하기 때문에 비용이 비싸다. 그래서, 충돌 매니저에서 shared\_ptr로 프로토 타입을 갖고 있도록 설계했고, Pixel 충돌체를 사용하는 객체에서 이 충돌체를 참조하는 방식으로 사용하도록 설계했다. 이를 통해 메모리 낭비를 줄일 수 있게 되었고 복사 생성시에도 스마트포인터만 복사하면 되기 때문에 전에 비해 효율이 좋아졌다고 할 수 있다.

\* 이동 시맨틱을 적극 활용하고 있다. 따라서 STL 알고리즘이나 vector의 size가 capacity보다 큰 경우에 이동 연산이 적용 될 수 있게 되었다.

\* 참조를 적극 활용해 불필요한 복사가 생기지 않도록 하고있다.

\* Tag 시스템을 설계했다. Tag 클래스를 상속받는 클래스는 tag, activation, enablement 3가지 프로퍼티를 갖게 된다. 각각의 프로퍼티들은 이름, 객체 소멸, 활성화 등을 관리한다. tag는 각종 Find 연산에 사용되기 때문에, 공유 자원을 관리할 떄 매우 유용하다. activation은 객체의 life를 나타내는 bool인데 false가 되면 해당 객체는 Scene 시스템에서 제거되게 되고 메모리 해제 된다. enablement는 해당 객체에 작업을 수행할지를 결정하는 bool 변수다. enablement가 false가 되면 해당 객체는 아무 작업도 하지 않는다.

### Scene system의 모듈들을 외부에서 참조하는 경우 weak\_ptr을 사용한다.

### shared\_ptr, weak\_ptr를 파라미터로 사용하는 함수에서 작업량에 따라 !, expired로 걸러준다.

### Initialize, Create는 엄격하게 Logic은 느슨하게