

# 언리얼 엔진의 메모리 관리

(Memory Management in Unreal Engine)

# 강의 내용

---

언리얼 엔진의 메모리 관리 방식을 파악하고,  
언리얼 오브젝트의 메모리를 관리하는 예제 실습

# 강의 목표

---

- 언리얼 엔진의 메모리 관리 시스템의 이해
- 안정적으로 언리얼 오브젝트 포인터를 관리하는 방법의 학습

# 언리얼 엔진의 자동 메모리 관리

# C++ 언어 메모리 관리의 문제점

---

- C++은 저수준으로 메모리 주소에 직접 접근하는 포인터를 사용해 오브젝트를 관리한다.
- 그러다보니 프로그래머가 직접 할당(new)과 해지(delete) 짝 맞추기를 해야 한다.
- 이를 잘 지키지 못하는 경우 다양한 문제가 발생할 수 있음.
- 잘못된 포인터 사용 예시
  - 메모리 누수(Leak) : new를 했는데 delete 짝을 맞추지 못함. 힙에 메모리가 그대로 남아있음.
  - 허상(Dangling) 포인터 : (다른 곳에서) 이미 해제해 무효화된 오브젝트의 주소를 가리키는 포인터
  - 와일드(Wild) 포인터 : 값이 초기화되지 않아 엉뚱한 주소를 가리키는 포인터.
- 잘못된 포인터 값은 다양한 문제를 일으키며, 한 번의 실수는 프로그램을 종료시킴.
- 게임 규모가 커지고 구조가 복잡해질수록 프로그래머가 실수할 확률은 크게 증가한다.

C++ 이후에 나온 언어 **Java/C#** 은 이런 고질적인 문제를 해결하기 위해

포인터를 버리고 대신 가비지 컬렉션 시스템을 도입함.

# 가비지 컬렉션 시스템

- 프로그램에서 더 이상 사용하지 않는 오브젝트를 자동으로 감지해 메모리를 회수하는 시스템.
- 동적으로 생성된 모든 오브젝트 정보를 모아둔 저장소를 사용해 사용되지 않는 메모리를 추적
- 마크-스융(Mark-Sweep) 방식의 가비지 컬렉션
  1. 저장소에서 최초 검색을 시작하는 루트 오브젝트를 표기한다.
  2. 루트 오브젝트가 참조하는 객체를 찾아 마크(Mark)한다.
  3. 마크된 객체로부터 다시 참조하는 객체를 찾아 마크하고 이를 계속 반복한다.
  4. 이제 저장소에는 마크된 객체와 마크되지 않은 객체의 두 그룹으로 나뉜다.
  5. 가비지 컬렉터가 저장소에서 마크되지 않은 객체(가비지)들의 메모리를 회수한다. (Sweep)

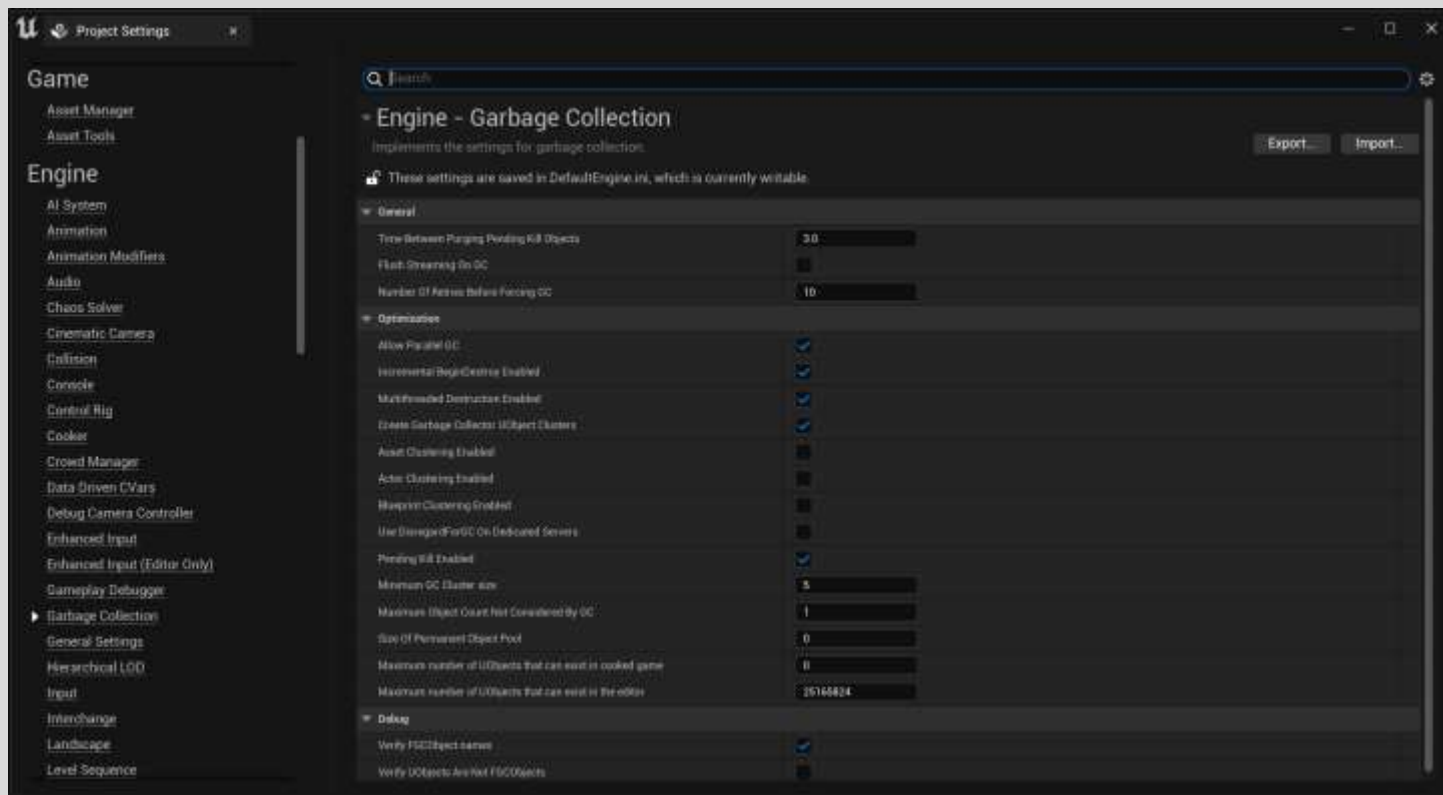
1	1	1	0	0	0
Object	Object	Object	Object	Object	Object

↙ ↑ ↗  
참조되고 있어  
회수 작업으로부터  
살아남을 오브젝트

↙ ↑ ↗  
참조되지 않아  
자동으로 회수될 오브젝트

# 언리얼 엔진의 가비지 컬렉션 시스템

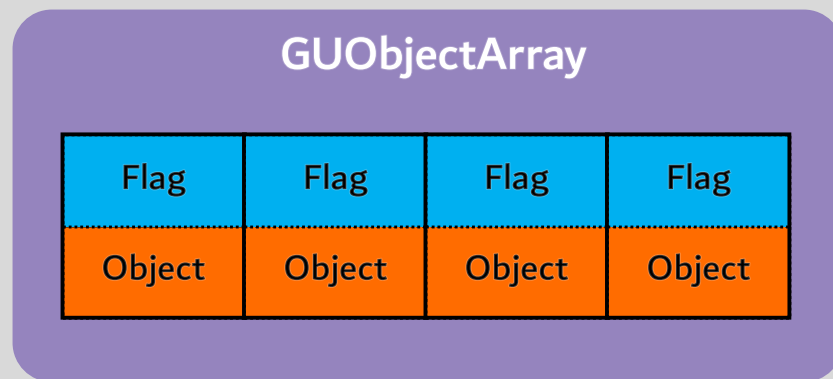
- 마크-스윙 방식의 가비지컬렉션 시스템을 자체적으로 구축함.
- 지정된 주기마다 몰아서 없애도록 설정되어 있음. (GCCycle. 기본 값 60초)
- 성능 향상을 위해 병렬 처리, 클러스터링과 같은 기능을 탑재함.



# 가비지 컬렉션을 위한 객체 저장소

---

- 관리되는 모든 언리얼 오브젝트의 정보를 저장하는 전역 변수 : GUObjectArray
- GUObjectArray의 각 요소에는 플래그(Flag)가 설정되어 있음.
- 가비지 컬렉터가 참고하는 주요 플래그
  - Garbage 플래그 : 다른 언리얼 오브젝트로부터의 참조가 없어 회수 예정인 오브젝트
  - RootSet 플래그 : 다른 언리얼 오브젝트로부터 참조가 없어도 회수하지 않는 특별한 오브젝트

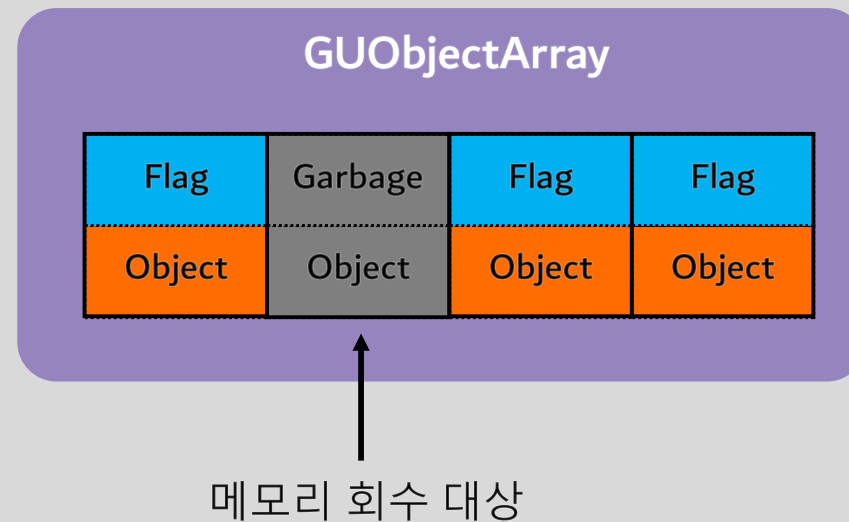


가비지 컬렉터는 **GUObjectArray**에 있는 플래그를 확인해 빠르게 회수해야 할 오브젝트를 파악하고 메모리에서 제거함



# 가비지 컬렉터의 메모리 회수

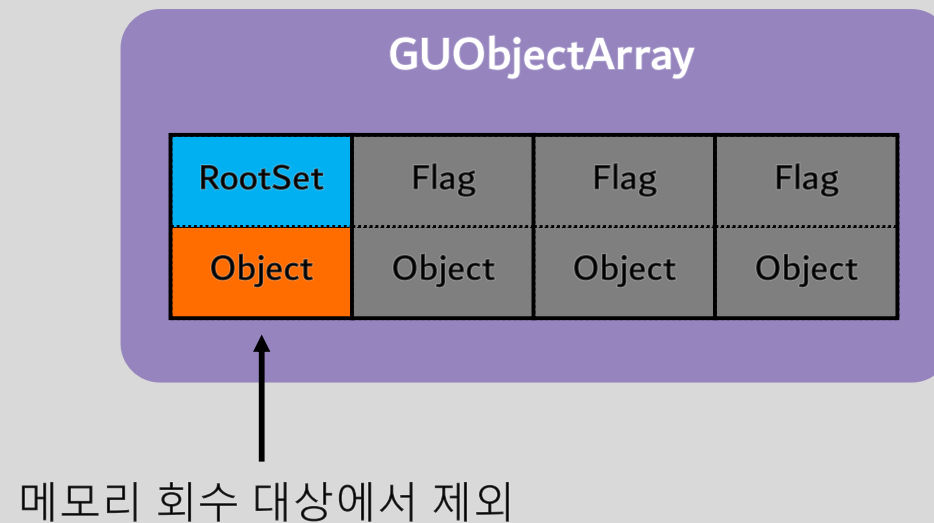
- 가비지 컬렉터는 지정된 시간에 따라 주기적으로 메모리를 회수한다. ( 기본 값 60초 )
- Garbage 플래그로 설정된 오브젝트를 파악하고 메모리를 안전하게 회수함.
- Garbage 플래그는 수동으로 설정하는 것이 아닌, 시스템이 알아서 설정함.



한 번 생성된 언리얼 오브젝트는 바로 삭제가 불가능함.

# 루트셋 플래그의 설정

- AddToRoot 함수를 호출해 루트셋 플래그를 설정하면 최초 탐색 목록으로 설정됨.
- 루트셋으로 설정된 언리얼 오브젝트는 메모리 회수로부터 보호받음.
- RemoveFromRoot 함수를 호출해 루트셋 플래그를 제거할 수 있음



콘텐츠 관련 오브젝트에 루트셋을 설정하는 방법은 권장되지 않음

# 언리얼 오브젝트를 통한 포인터 문제의 해결

---

- 메모리 누수 문제

- 언리얼 오브젝트는 가비지 컬렉터를 통해 자동으로 해결.
- C++ 오브젝트는 직접 신경써야 함. (스마트 포인터 라이브러리의 활용)

- 댕글링 포인터 문제

- 언리얼 오브젝트는 이를 탐지하기 위한 함수를 제공함 ::IsValid()
- C++ 오브젝트는 직접 신경써야 함. (스마트 포인터 라이브러리의 활용)

- 와일드 포인터 문제

- 언리얼 오브젝트에 UPROPERTY 속성을 지정하면 자동으로 nullptr로 초기화 해 줌.
- C++ 오브젝트의 포인터는 직접 nullptr로 초기화 할 것 (또는 스마트 포인터 라이브러리를 활용)

# 회수되지 않는 언리얼 오브젝트

---

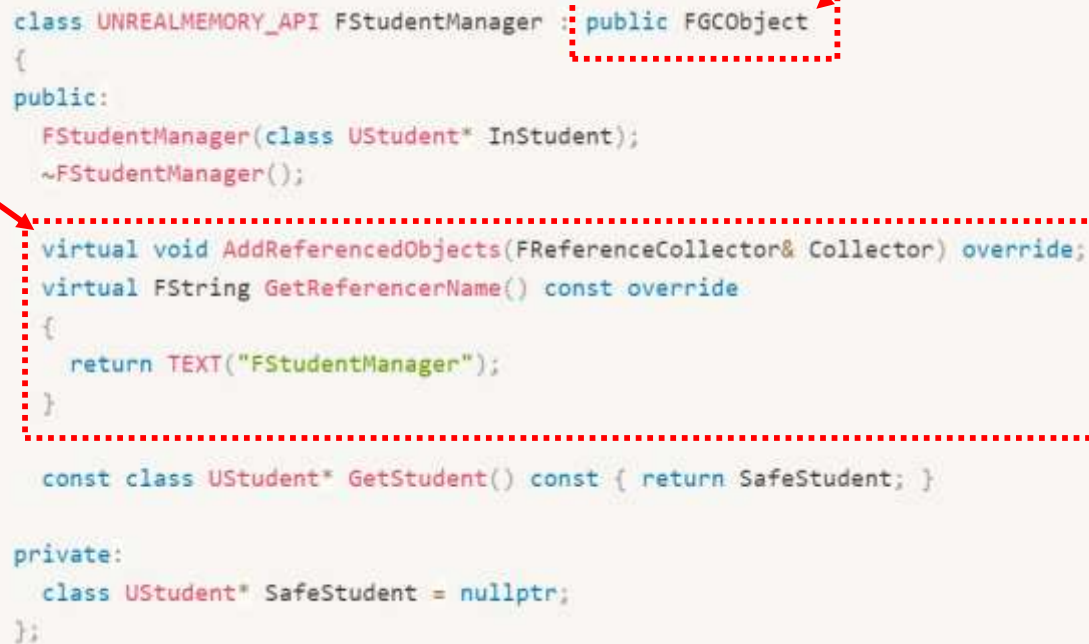
- 언리얼 엔진 방식으로 참조를 설정한 언리얼 오브젝트
  - UPROPERTY로 참조된 언리얼 오브젝트 ( 대부분의 경우 이를 사용 )
  - AddReferencedObject 함수를 통해 참조를 설정한 언리얼 오브젝트
- 루트셋(RootSet)으로 지정된 언리얼 오브젝트

( 오브젝트 선언의 기본 원칙 )

오브젝트 포인터는 가급적 **UPROPERTY**로 선언하고,  
메모리는 가비지컬렉터가 자동으로 관리하도록 위임한다.

# 일반 클래스에서 언리얼 오브젝트를 관리하는 경우

- UPROPERTY를 사용하지 못하는 일반 C++ 클래스가 언리얼 오브젝트를 관리해야 하는 경우
- FGObject 클래스를 상속받은 후 AddReferencedObjects 함수를 구현한다.
- 함수 구현 부에서 관리할 언리얼 오브젝트를 추가해 줌.



```
class UNREALMEMORY_API FStudentManager : public FGObject
{
public:
    FStudentManager(class UStudent* InStudent);
    ~FStudentManager();

    virtual void AddReferencedObjects(FReferenceCollector& Collector) override;
    virtual FString GetReferencerName() const override
    {
        return TEXT("FStudentManager");
    }

    const class UStudent* GetStudent() const { return SafeStudent; }

private:
    class UStudent* SafeStudent = nullptr;
};
```

The image shows a C++ code snippet for the `FStudentManager` class. The class is defined as `class UNREALMEMORY_API FStudentManager : public FGObject`. The `public` section includes a constructor `FStudentManager(class UStudent* InStudent);`, a destructor `~FStudentManager();`, and two virtual functions: `virtual void AddReferencedObjects(FReferenceCollector& Collector) override;` and `virtual FString GetReferencerName() const override;`. The `GetReferencerName` function returns `TEXT("FStudentManager")`. A `const` function `GetStudent()` returns `SafeStudent`. The `private` section contains a `UStudent* SafeStudent = nullptr;`. Two red arrows point to the `public FGObject` inheritance and the `AddReferencedObjects` function implementation.

시스템을 구축하면서 필요한 상황이 발생할 수 있음.

# 언리얼 오브젝트의 관리 원칙

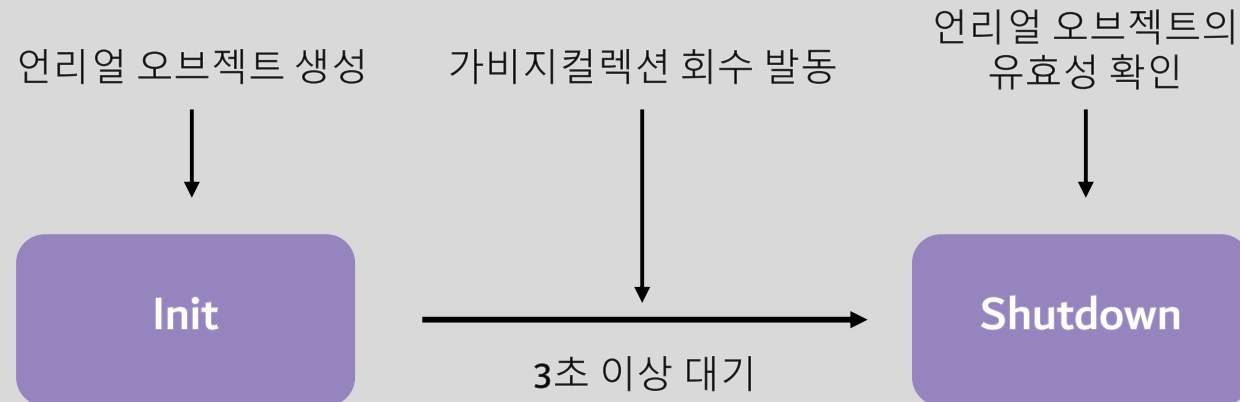
---

- 생성된 언리얼 오브젝트를 유지하기 위해 레퍼런스 참조 방법을 설계할 것
  - 언리얼 오브젝트 내의 언리얼 오브젝트 : UPROPERTY 사용
  - 일반 C++ 오브젝트 내의 언리얼 오브젝트 : FGCOject의 상속 후 구현
- 생성된 언리얼 오브젝트는 강제로 지우려 하지 말 것
  - 참조를 끊는다는 생각으로 설계할 것
  - 가비지 컬렉터에게 회수를 재촉할 수는 있음 ( ForceGarbageCollection 함수 )
  - 콘텐츠 제작에서 Destroy함수를 사용할 수 있으나, 결국 내부 동작은 동일함. ( 가비지컬렉터에 위임 )

# 언리얼 엔진의 메모리 회수 예시

# 가비지 컬렉션 테스트 환경 제작

- 프로젝트 설정에서 가비지 컬렉션 GCcycle 시간을 3초로 단축 설정
- 새로운 GameInstance의 두 함수를 오버라이드
  - Init : 어플리케이션이 초기화될 때 호출
  - Shutdown : 어플리케이션이 종료될 때 호출
- 테스트 시나리오
  - 플레이 버튼을 누를 때 Init 함수에서 오브젝트를 생성하고
  - 3초 이상 대기해 가비지 컬렉션을 발동
  - 플레이 중지를 눌러 Shutdown 함수에서 생성한 오브젝트의 유효성을 확인





정리

# 언리얼 메모리 관리 시스템

---

1. C++ 언어의 고질적인 문제인 포인터 문제의 이해
2. 이를 해결하기 위한 가비지 컬렉션의 동작 원리의 이해와 설정 방법
3. 다양한 상황에서 언리얼 오브젝트를 생성하고 메모리에 유지하는 방법의 이해
4. 언리얼 오브젝트 포인터를 선언하는 코딩 규칙의 이해