1단계_12. 언리얼 엔진의 메모리 관리

수강일	@2023/04/04
이름	전영재
멘토	Min-Kang Song,현웅 최
멘티	
작성 상태	Not started
단계	1단계
강의 시청 여부	☑
이수 여부	☐

Contents

강의 제목

강의 목표

- 언리얼 엔진의 메모리 관리 시스템의 이해
- 안정적을 언리얼 오브젝트 포인터를 관리하는 방법 학습

언리얼 엔진의 자동 메모리 관리

C++언어의 메모리관리의 문제점?

저수준으로 메모리주소에 직접 접근할수있기에, 프로그래머가 직접 할당(new) 해지(delete)를 해줘야함

못하면 다음과 같은 문제가 생길 수 있다.

1. 메모리 누수
2. 허상 포인터 : 다른곳에서 해제된 주소를 가리키는 포인터
3. 와일드 포인터 : 초기화 되지않은 엉뚱한 주소를 가리킴

⇒ 한번의 실수로 전체 프로젝트가 뻗을수있다.

그래서 C++이후 언어들은 포인터를 버리고 가비지컬렉션 시스템을 도입했다.

가비지 컬렉션

더이상 사용하지 않는 오브젝트를 자동으로 감지해 메모리를 회수하는 시스템

일반적으로 마크-스윕 방식으로 사용함.

무슨방식인진 찾아보자.(강의듣자)

언리얼 엔진의 가비지 컬렉션

자체적으로 마크-스윕 방식의 가비지 컬렉션을 구축했다.

지정된 주기마다 몰아서 없애도록 설정되어 있음. 이걸 GCCycle이라한다.

이런 가비지 컬렉터가 돌아가는 것만으로도 적지않으 리소스 소모가 드는데, 성능향상을 위해 언리얼은 병렬처리, 클러스터링 같은 기능을 탑재했다.

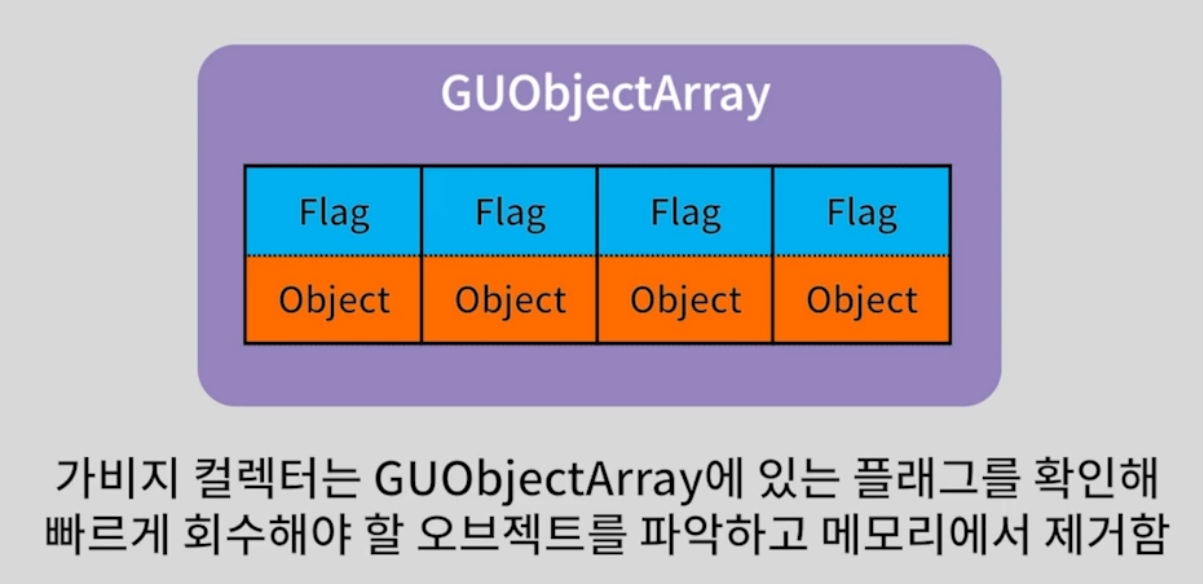
가비지컬렉션을 위한 객체저장소

언리얼엔진은 관리하는 모든 언리얼 오브젝트의 정보를 저장하는 전역변수인 GUObjectArray가 존재한다.

이 Arra의 각 요소에는 플래그가 설정되어 있다.

Garbage플래그 : 다른 오브젝트로부터의 참조가 없어 회수예정인 오브젝트

RootSet플래그 : 다른 오브젝트로부터의 참조가 없어도 회수하지 않는 ㄴ특별한 오브젝트



요약하면 언리얼 가비지컬렉터는 GUObjectArray의 플래그를 확인해 빠르게 회수할 오브젝트를 파악하고 메모리에서 제거한다.

메모리회수

지정된 시간마다 주기적으로 메모리 회수한다.

Garbage플로그로 설정된 오브젝트를 파악하고 안전하게 회수하는데, 이 Garbage플래그는 수동으로 설정하는게 아니라 시스템에서 알아서 설정한다.

⇒ 그래서 오브젝트를 삭제할때는, delete로 직접삭제하는게 아니라 참조를 없앰으로써 가비지컬렉터가 삭제하게 냅두는 것이다.

루트셋플래그의 설정

만약 중요한 오브젝트가 있다면 AddToRoot함수를 통해 루트셋플래그를 설정해주면 보호받을수있다.

나중에 필요없으면 RemoveFromRoot로 없애면된다.

근데, 사실 이방법은 컨텐츠 만들때 권장되진 않는다.

언리얼 오브젝트를 통한 포인터 문제의 해결

1. 가비지 컬렉터를 통해 자동으로 메모리 문제를 해결하기에 메모리 누수문제가 없다.
    a. 단 C++오브젝트는 직접 신경써야한다.(스마트포인터 라이브러리 활용하던가)
2. 댕글링 포인터문제 해결가능

a. 사용하는지 판단하기 위해, ::IsValid()함수를 제공한다.

b. 단, C++오브젝트는 직접 신경써야한다.

3. 와일드 포인터문제

a. 오브젝트를 UPROPERTY속성을 지정해주면 자동으로 초기값을 nullptr로 초기화 해준다.

b. 마찬가지로 C++오브젝트는 직접 nullptr로 초기화 해줘야 한다.

회수되지 않는 언리얼 오브젝트

UPROPERTY로 참조된 오브젝트는 가비지컬렉터가 회수하지 않는다.

또는 AddReferencedObject 함수를 통해 참조 설정하면 회수안해간다. 근데 자주 안쓴다.

루트셋으로 지정한 오브젝트도 회수안한다. - 진짜 중요한데만 쓰는건데, 잘안쓴다.

⇒그래서 가급적 오브젝트 포인터는 UPROPERTY로 선언하고 메모리는 가비지컬렉터가 관리하도록 위임하는게 좋다.

UPROPERTY를 사용하지 못하는 경우에서 언리얼 오브젝트를 관리해야 하는경우, FCCObject 클래스를 상속받은후, AddReferencedObjects 함수를 구현하면 된다. →컨텐츠 제작에서 잘쓰진 않는다.

언리얼 오브젝트의 관리원칙

1. 생성된 언리얼 오브젝트를 유지하기 위해 레퍼런스 참조방법을 설계할것

a. 언리얼 오브젝트 내의 언리얼 오브젝트 : UPROPERTY 사용

b. C++ 오브젝트 내의 언리얼 오브젝트 : FGCObject 상속 후 구현

2. 생성된 언리얼 오브젝트는 강제로 지우려 하지 말 것

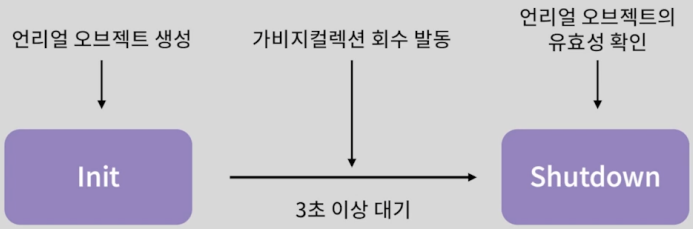
a. 참조를 끊는다는 생각으로 설계할 것.

b. 콘텐츠 제작에서 Destroy함수를 사용할 수 있으나, 결국 내부 동작은 동일하다.(이거도 플래그 설정하고 나중에 가비지컬렉터가 가져가는 방식이거든)

실습

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



즉, UPROPERTY가 붙은 오브젝트, 안붙은 오브젝트 하나씩 만들고 3초후에 가비지 컬렉터가 회수하고 가면 포인터가 어떤 형태로 남아있을지 확인하는 예제다

ㄴ> 오브젝트의 IsValidLowLevel을 써서 로그 찍고, nullptr인진도 확인하는 함수만들어서 로그찍을거다.

CheckUObejctIsValid()와 CheckUObjectIsNull() 을 만들었다

코드 작동 결과,

둘다 nullptr은 아니라고 나왔다. 하지만 Nonprop은 유효하지 않고 PROPERTY붙은애는 유효하다고 나왔다.

⇒ 즉, 이게 nullptr만 보고 섣불리 판단하면 댕글링포인터 문제가 생긴다는 걸 보여주는 예시인 것이다.

똑같이 TArray에 넣어도 UPROPERTY를 붙였냐에 따라 위와 같은 결과가 나오더라.

다음은 일반 C++에서 언리얼 오브젝트를 넣을때 어떻게 해야되는지 알아보자.

1. 소멸자는 일부러 삭제했다.
2. 생성자는 접두사 F를 붙이고 수정했다.
3. …음 이해안된다.

4. 아무튼 이렇게 만든 C++클래스를 MyGameInstance에서 새로운 오브젝트를 NewObject로 만들어주고, 이 오브젝트의 내부에 있는 언리얼 오브젝트가 살아있는지 확인해봤다.

→ UPROPERTY를 달아줄수가 없어서 유효하지 않다고 나오더라.

5. 이걸 해결하기 위해선, FGCObject를 상속받아서 AddReferencedObject() 함수와 GetReferencerName() 함수를 구현해줌으로써 해결할 수 있다.

AddReferencedObject(오브젝트) //이렇게 오브젝트를 등록함으로써 해결했다.

📌 Summary

- C++언어의 고질적인 포인터 문제의 이해
- 이를 해결하기 위한 가비지 콜렉션의 동작원리 이해와 설정방법
- 다양한 상황에서 언리얼 오브젝트를 생성하고 메모리에 유지하는 방법 이해
- 언리얼 오브젝트 포인터를 선언하는 코딩 규칙의 이해
- ⇒ 왠만하면 UPROPERTY쓰면된다

0n. ~~~ 강의 과제

❓ **Q1.**

❓ **Q2.**

❓ **Q3.**

📖 **Reference**