SERIES GAME

게임 개발의 조율사 '게임 진행 모듈'

개별 엔진 통합 및 효율적 데이터 관리 위한 필수 도구

글 / 한국전자통신연구원(ETRI) 김현빈 외 가상현실연구부

연재순서

1. 3D 게임엔진에 대해
2. 렌더링 엔진
3. 애니메이션 엔진
4. 사운드 엔진
5. 서버 엔진
6. 게임 인공지능
7. 맵 에디터
8. **게임진행 모듈**

게임 진행모듈이란 개별 엔진을 유기적으로 통합하고, 각종 데이터를 효율적으로 관리하며, 기획을 충실히 반영해 콘텐츠를 개발하는 데에 필요한 모듈을 말한다. 렌더링 엔진, 애니메이션 엔진 등 개별 엔진과 기획서가 충실히 준비됐다고 하더라도 그것을 이용해 성공적인 게임 콘텐츠를 만들기 위해서는 많은 추가 작업이 필요하다. 이것을 요리에 비유하면 바로 이전까지는 요리를 만들 재료만 준비된 것이라고 볼 수 있다. 이 재료들을 알맞게 배합하고, 순서에 따라 조리를 한 다음 보기 좋게 식탁에 배치해야 하는 일이 지금부터 해야 할 일이다. 이런 일을 하는 것이 게임 진행 모듈이다. 이번 호에서는 게임 진행 모듈을 엔진의 통합과 게임 객체의 관리로 나누어서 설명하겠다.

엔진의 통합

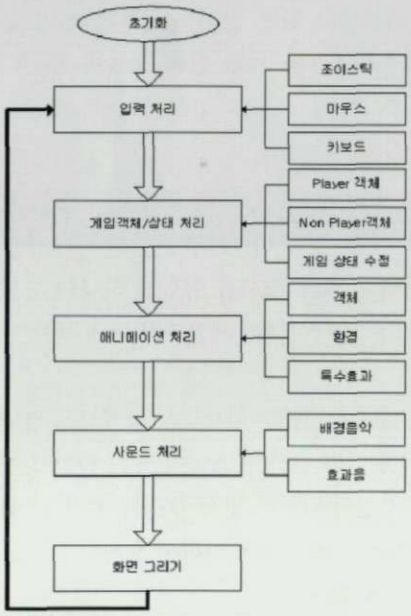
좋은 성능을 가진 엔진이 있다고 하더라도 그 엔진 내의 각 세부 모듈을 통합해 하나의 콘텐츠로 만들기 위해서는 단순히 엔진을 조합하기만 하면 되는 것이 아니라 게임 개발 프로젝트 전체에 대한 이해가 필요하다.

개발하고자 하는 게임이 구상됐다면 게임을 개발하기 위한 플랫폼과 기능들을 알아보고 타당한 지에 대한 결정을 내려야 한다. 엔진을 개발할 능력이 된다면 원하는 게임에 맞는 엔진을 개발해야 하고, 그렇지 않다면 외부에서 엔진을 가져와서 자신의 입맛에 맞게 수정해야 한다. 엔진을 선택하기에 앞서 원하는 기술의 스펙(Spec)을 정리하고, 그 스펙에 따라서 원하는 엔진을 선택해야 한다. 기술 스펙을 결정할 때 고려할 사항은 다음과 같다.

스 펙	설 명	예
플랫폼	게임이 동작될 기반 하드웨어, 소프트웨어 환경을 결정한다.	MS Windows기반의 PC, Linux, 모바일, PDA 등
렌더링 엔진	눈에 보여질 것들의 기능을 결정한다.	2D/3D, 카툰 렌더링, BSP기반 등
사운드 엔진	귀에 들려질 것들의 기능을 결정한다.	스테레오, 돌비 서라운드 등
레벨 에디터	각 레벨을 디자인하기 위해 필요한 기능을 결정한다.	실내/실외, script 스타일, trigger 방식 등

개별 엔진이 어느 정도 완료가 됐거나 통합할 시기가 됐다면 이제 각 엔진에 대한 이해와 정리가 필요한 단계가 됐다고 할 수 있다. 개별 엔진의 기능 중 개발하고자 하는 콘텐츠에서 필요한 기능을 알아내고, 그것을 사용하고 관리할 모듈을 만들어 내어야 한다. 자신이 직접 엔진을 개발했을 경우에는 상관없지만 외부에서 가져온 경우 반드시 필요한 작업이다. 이 경우 자신에게 필요한 부분을 가려낼 뿐만 아니라 부족한 부분을 알아내어 직접 작성을 해야 하기도 한다.

● 기획서에 기반 둔 설계 필요



개발자는 게임의 기획을 충분히 이해하고 숙지한 후 설계에 들어가는 것이 바람직하다. 기획의도와 게임 플레이에 대해 정확하게 이해해야 전체적인 게임 윤곽이 잡힌다. 기획 작업이 어느 정도 된 후에는 콘텐츠의 기본 구조 및 골격을 정리한다. 프로그래머는 실제 프로그래밍 작업에 들어가기 전에 최대한 상세한 부분까지 디자인을 하도록 해야 한다.

프로그램 설계가 끝났으면 실제 프로그래밍 작업에 들어간다. 일반적으로 게임을 진행할 때에 처리되는 과정은 다음의 그림과 같다. 이 과정이 계속 반복된다고 해서 진행 루프(loop)라고 부르며, 매번 새로운 화면을 프레임(frame) 단위로 갱신하므로 프레임이라고 부르기도 한다.

● 프로젝트 및 소스 프로그램 관리

프로젝트 관리는 프로그램, 디자인, 사운드, 기획 등 전체적인 관리를 하는 것을 뜻한다. 요즘은 게임 개발 시에 한두 명이 개발하는 것이 아니라 많은 사람이 개발에 참여하고 영화와 같이 각 담당에 대한 책임과 이들간의 체계가 자리 잡혀져 가고 있다. 이를 보조하기 위해 여러 가지 도구가 있으므로 활용하면 좀 더 체계적이고 수월한 관리가 가능하다. 예를 들어 MS Project나 Lotus Notes, 웹 게시판 등을 이용해 관리할 수도 있다.

종종 프로그래밍이나 파일 수정 중에 파일들을 잃어버리는 수가 있다. 이러한 경우에 소스 관리 프로그램을 사용한다면 가장 최근의 백업으로 복구시킬 수 있다. 여러 명이 프로그래밍을 하다 보면 같은 파일을 여러 사람이 동시에 수정할 수도 있다. 이럴 경우 자신이 수정한 내용이 없어지거나, 다른 사람이 수정한 내용과 겹쳐서 문제를 야기시킬 수 있다.

코드를 공유하기란 쉬운 일이 아니다. 만약 한 프로그래머가 수정한 파일을 동시에 다른 프로그래머가 수정을 하고, 그것을 여러 사람이 저장한 후에 프로그래머가 다시 저장한다고 하자. 이와 같이 하다 보면 본래 수정한 코드가 사라질 수 있다. 이러한 이유로 소스를 관리해야 하는 문제가 생긴다. 정책적으로 클래스나 모듈을 되도록 연결성이 적고 독립적으로 동작하도록 만들고, 필요할 때마다 백업을 해두는 것이 좋지만 궁극적인 해결책은 되지 못한다. 이러한 문제를 해결해줄 수 있는 여러 가지 도구들이 있다. 예를 들면 CVS나 MS Source Safe 등이 좋은 예이다. 이들은 수정할 파일을 체크아웃(check out)해 사용하게 함으로써 자신이 수정 중임을 표시하고, 수정이 완료되면 체크인(check in)을 해 다른 사람이 수정한 내용과 중복이 되지 않도록 도와준다. 만약 같은 부분을 수정했을 경우에는 그 부분을 사용자가 직접 수정해야 한다. 이뿐만 아니라, 여러 가지 버전을 관리하는 버전 관리 기능, 수정된 내용을 시간대별로 보여주는 history 기능 등이 있어서 편리하게 사용할 수 있다.

● 코드 명명규칙(code naming convention)

여러 사람들이 각 모듈을 나누어 맡아서 작성하다 보면 변수나 클래스 이름 등을 짓는 방식이 모두 다름을 알 수 있다. 예를 들어 GameObject 클래스를 만든다고 하자. 어떤 사람은 클래스 이름을 CGameObject로 만든다. 또 어떤 사람은 TGameObject로 만든다. 실제 하는 기능은 똑같고 공유돼야 할 클래스들이다. 또한 함수 이름도 마찬가지이다. 위치를 정하는 함수가 있을 때 어떤 사람은 setLocation()함수라 이름짓고 LocationSetup()이라고 이름짓기도 한다. 서로 똑같은 기능을 수행하는 함수이다. 이렇게 되면 전체적인 소스의 일관성이 없어지고, 가독성이 떨어져서 유

지보수에 시간이 걸린다. 이러한 이름들은 시작 시 미리 규칙을 정해 두고 모두 그 규칙을 따르게 하는 것이 좋다. 식별자(identifier)의 이름을 지을 때 다음과 같은 점을 고려해 이름을 짓는 습관을 들이도록 한다.

첫째, 프로그래머가 기억하기 쉬운 이름을 만든다. 둘째, 다른 사람들이 읽기 쉬운 이름을 만든다. 셋째, 같은 기능을 하는 식별자는 같은 이름 규칙에 따라서 이름을 짓는다.

게임 객체의 관리

게임의 디자인이 끝났으면 필요한 자료를 정리해야 한다. 프로그램 속에 직접 프로그래밍해야 할 부분과 외부에서 가져 올 부분을 정리한다. 되도록 외부에서 데이터를 관리하도록 한다. 외부 데이터는 맵 에디터나 캐릭터 에디터 등 부가적인 툴(tool)을 이용해서 만들 수도 있고, 사운드나 동영상 등 외부 상용 툴을 이용해서 만든 데이터도 포함이 된다.

이들 데이터를 정리하는 데에 있어서 중요한 것은 데이터의 구조와 사용 목적, 조건, 제한사항 등이 데이터 작성자와 프로그래머간에 충분히 협의돼야 한다는 것이다. 그렇지 않으면 만든 데이터를 사용하기 위해 후처리를 해 주어야 할 수도 있고, 최악의 경우 데이터를 전혀 사용하지 못하고 버릴 수도 있다. 간단한 샘플 데이터를 미리 만들어서 테스트해 보는 것도 좋은 방법이다.

메모리 관리는 게임 내의 객체를 만들고 삭제하는 것을 말한다. 최근 주 메모리의 용량과 가격이 낮아져서 메모리는 그리 신경 쓰지 않지만, 게임은 그래픽, 사운드, 동영상 등 상대적으로 덩치가 큰 자료를 수시로 로드해야 하기 때문에 메모리 관리가 필수적이다. 예를 들어 MMORPG의 경우 실제 대륙에 맞먹는 방대한 게임월드를 처리하기 때문에 한꺼번에 모두 메모리에 로드해 실행하기 힘들다. 이럴 경우 동적으로 자료를 로드하고 삭제하는 일이 필요하게 된다.

C++에서 메모리를 할당하고 삭제하는 new와 delete는 사용하기에는 편리하나 성능면에서 요구사항을 만족하지 못하는 경우가 많다. 특히 새로운 객체를 만들 경우 생성자에서 초기화해야 할 데이터가 많은 경우 더욱 느려지게 된다. 또한 빈번한 메모리 추가 삭제로 인해 메모리 단편화가 생기며, 성능 저하의 원인이 된다. 메모리 관리자는 게임 객체별 메모리 생성 및 삭제를 담당하며, 필요시 디버깅 정보의 추가, 데이터의 암호화와 로드/저장 기능을 처리한다.

게임을 프로그래밍하기 위해 프로그래머는 게임 환경에 나타나는 모든 종류의 객체들을 정의해야 한다. 우리는 이것을 게임 객체(game object)라고 부른다. 게임 객체는 게임이 진행되는 동안 생성되고 동작되고, 삭제된다. 화면 크기나 사용자 인터페이스와 같이 게임 진행에 직접적으로 관련되지 않은 것은 여기에 속하지 않는다. 이것은 움직이는 캐릭터가 될 수도 있고 그 캐릭터가 발사한 총알이 될 수 있다.

게임 객체는 개발자에 따라서 여러 가지 다른 이름으로 불리기도 한다. 예를 들어 Quake에서는 Entity로 나타나고, Unreal에서는 Actor, Age of Empire에서는 Entity로 불린다. 게임 객체는 C++에서 class형태로 표현된다.

객체를 설계할 때도 상속관계를 이용해 설계하도록 한다. 객체 내에 공통적인 속성을 부모 클래스로 하고, 자신만의 특징적인 속성만을 정의하고 처리하도록 하면 된다. 가상함수를 사용해 부모 클래스에 모든 가능한 함수 호출을 두고 실제 구현은 자식 클래스에서 구현해 주면 다른 특성을 가지는 객체들을 같은 인터페이스로 처리하는 데에 도움이 된다.

게임 객체를 디자인할 때에는 미리 게임 객체의 성격과 게임 속에서 나타나고 처리돼야 할 것들에 대해서 고려해 두도록 하는 것이 좋다.

● 자료 주도형(data-driven) 객체 설계

각 객체마다 손으로 일일이 프로그래밍 작업을 하는 것은 쉽지만, 속성을 테스트해보기 위해 숫자나 문자열 등을 바꿀 경우 매번 컴파일해야 하고, 때로는 추가적으로 많은 프로그래밍을 요구한다. 이러한 작업은 프로그래머가 아닌 게임 디자이너가 쉽게 고쳐보고 테스트해 볼 수 있도록 만들어야 한다.

모든 자료를 프로그램 상수가 아닌 파일이나 데이터베이스로부터 읽어 들이도록 만든다. 초기 작업 시에는 텍스트 파일(text file)을 사용하는 것이 도움이 될 것이다. 하지만 텍스트 파일은 다루기는 쉽지만 일반적으로 읽어오는 데에 시간이 바이너리 파일(binary file)에 비해 상대적으로 많이 걸리므로 최종 버전에서는 바이너리 파일이나 데이터베이스를 사용하는 경우가 많다.

스크립트 파일(script file)을 이용해 게임의 인공지능을 구현하면 테스트하거나 수정하기 용이하다. 이러한 데이터끼리도 상속관계를 갖게 하면 다루기에 편리하다. 기본적인 데이터는 한 곳에서 관리하고 특징적인 데이터를 따로 분리해 사용한다.