게임엔진

제11강 게임 플레이 시스템

학습 내용

- 게임 월드의 구성
- 동적 구성 요소 구현: 게임 객체
- 데이터 주도 게임 엔진
- 게임 월드 에디터
- 게임 플레이 기반 시스템
- 런타임 객체 모델 구조

게임 플레이 시스템

- **게임 메카닉(Game Mechanics)**
  - 게임 내에 있는 여러 단위 간의 상호 작용을 규정하는 규칙들
  - 플레이어의 목표
  - 성공과 실패의 기준
  - 플레이어 캐릭터의 능력
  - 가상 월드에 존재하는 NPC 개체들의 수와 종류
  - 전체적인 게임을 통해 겪는 체험의 흐름

  - 게임 메카닉은 일반적으로 설득력있는 이야기와 생동감 있는 캐릭터들과 밀접하게 섞임.
    - 물론, 이야기와 캐릭터가 필수적인 것은 아님. (예: 테트리스)

- **G Factor**
  - 게임 플레이를 구현하는 데 사용되는 소프트웨어 시스템

월드 구성 요소

■ 게임 월드

□ 게임의 배경이 되는 2D 또는 3D 의 가상 게임 세계

□ **“정적 구성 요소”**

- 움직이지 않거나, 게임 플레이에 적극적으로 관여하지 않는 요소들
- 지형, 빌딩, 길, 다리, 등등

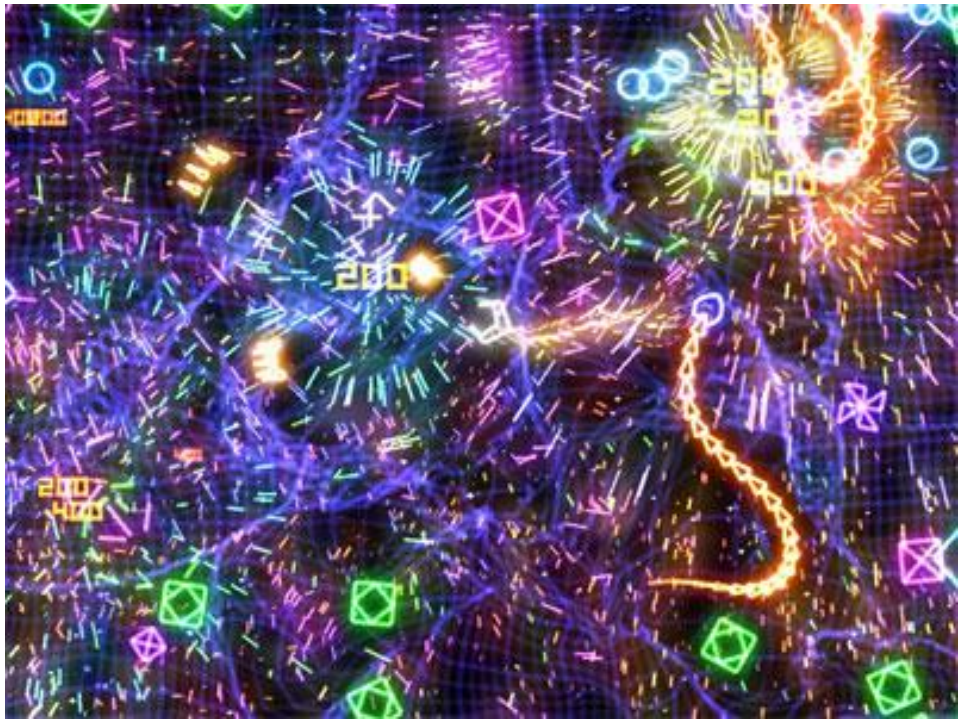
□ **“동적 구성 요소”**

- 움직이이면서 게임의 진행에 직접적으로 개입하는 요소들
- 캐릭터, 탈것, 무기, 아이템, 수집할 수 있는 물건
- 파티클 이미터(Particle Emitter)
- 동적 광원(Dynamic Light)
- 영역(Region) - 중요한 게임이벤트를 검출하는, 눈에 보이지 않는 공간
- 스플라인(Spline) - 물체의 경로를 정의

- **“게임 상태(Game State)”** - 게임 월드의 동적 구성 요소들 전체의 상태를 포괄적으로 지칭

- **동적 구성 요소과 정적 구성 요소의 비율은 게임마다 다름.**

- **일반적으로, 적은 수의 동적 구성 요소들이 넓은 정적 배경 공간 위에서 움직임.**
  - 동적 구성 요소는 CPU 자원 측면에서 정적 요소보다 비쌈.
  - 물론, 동적 구성 요소가 많아야 게임 월드가 “활기차” 보임.
  - 정적 구성 요소가 없는 것도 있음. (예. Asteroids, Geometry War)



■ **동적 요소과 정적 요소의 구별은 항상 명확하지 않음.**

□ 게임 엔진 마다 기준이 다르며, 아예 구분하지 않고 모든 요소를 동적으로 간주하는 엔진도 존재.

■ 동적 요소과 정적 요소의 구별은 “최적화” 관점에서 보는 것이 타당.

□ 어떤 물체의 상태가 변하지 않는다면?

- 미리 계산함으로써 런타임 시간 계산을 없앨 수 있음.
- 정적 배경 물체들 - 월드 공간으로 미리 배치 가능
- 라이트맵(Light Map)
- 그림자맵(Shadow Map)
- 정적 환경 차폐 정보(Static Ambient Occlusion Information)
- PRT(Precomputed Radiance Transfer) 구면조화 함수 계수(Spherical Harmonics coefficient)

□ 파괴가능한 배경의 구현?

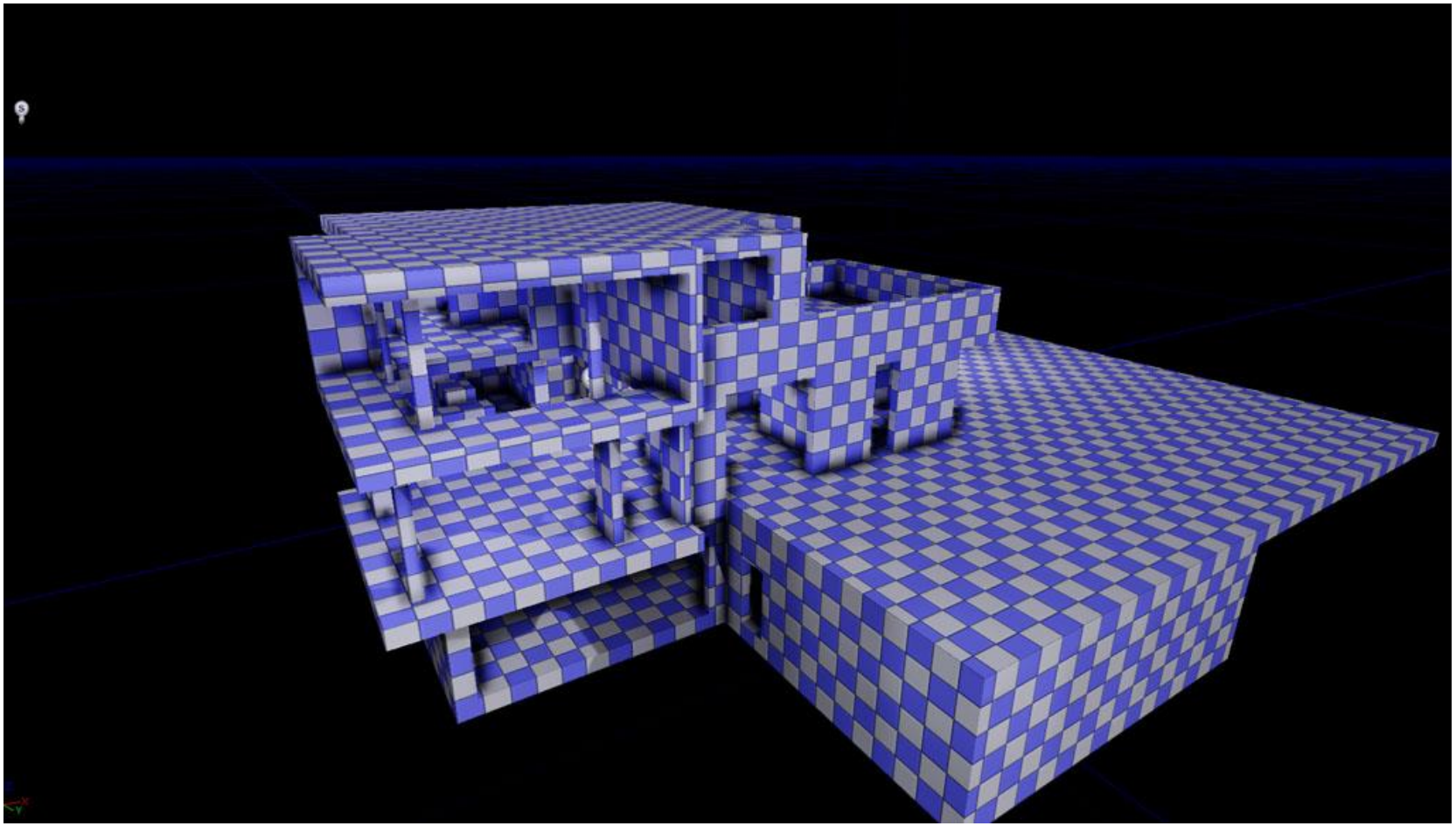
- 세가지 다른 버전(멀쩡한 버전, 손상 입은 버전, 완전히 파괴된 버전)을 미리 준비
- 게임 진행에 따라 교체

정적 지형(Static Geometry)

- 풀, 나무, 빌딩 등, 정적 배경으로 사용되는 물체들을 효과적으로 모델링 및 렌더링
- Geometry Instancing
  - 한 씬내에서 동일한 메쉬의 복사본을 여러 개를 렌더링하는 것.
  - 각 인스턴스의 위치와 방향만 살짝 다르게 줌으로써, 다양한 물체들이 있는 것 같은 느낌을 줌.
  - 그래픽 카드의 가속 기능(Hardware Instancing)을 통해 빠른 렌더링이 가능.

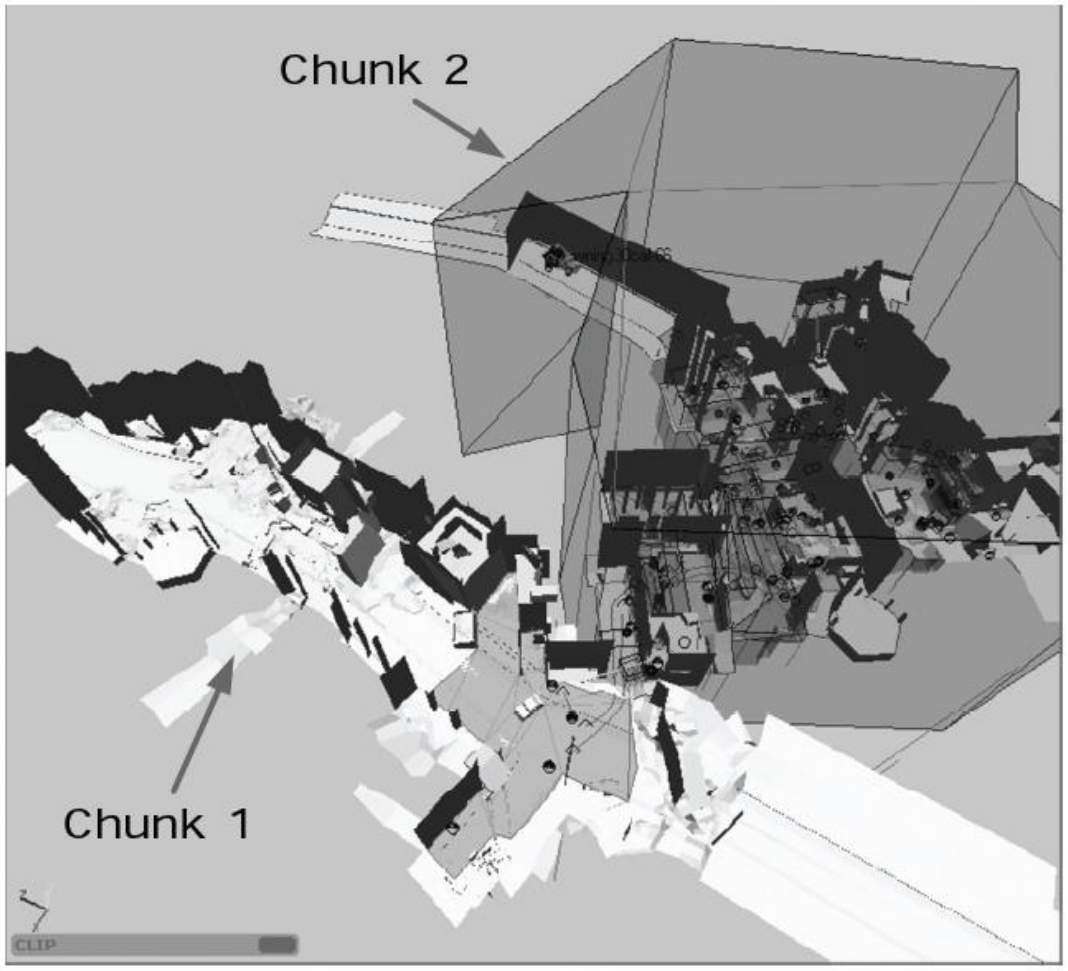
Brush Geometry

- 평면들로 둘러싸인 볼록한 입체들이 모여서 이루어진 형상
- 빠르고 만들기 쉽고, BSP 렌더링과 잘 어울림.
- 배경의 충돌 데이터를 만드는데 많이 사용됨.



월드 덩어리(World Chunk)

- 게임의 배경이 매우 넓은 경우, 여러 개의 지역으로 쪼개는 것이 일반적.
- World Chunk = 레벨 = 맵 = 스테이지 = 에어리어
- 게임의 흐름에 따라 청크 사이를 플레이어가 이동해 나가게 됨.



레벨

- 초창기 게임기들의 한정된 메모리 제약 조건 내에서 다양한 게임 플레이를 제공하는 목적으로 만들어진 것.
- 최근의 게임들에서는 월드 청크가 바뀌는 것을 유저가 체감하기 어려워짐.

다양한 청크 구조

- 별형 구조(Star Topology) - 플레이어가 시작하는 허브 지역을 중앙에 두고, 이 허브에서 다른 지역들에 마음대로 접근하는 형태
- 그래프형 구조 - 각 지역이 일정하지 않은 방식으로 연결

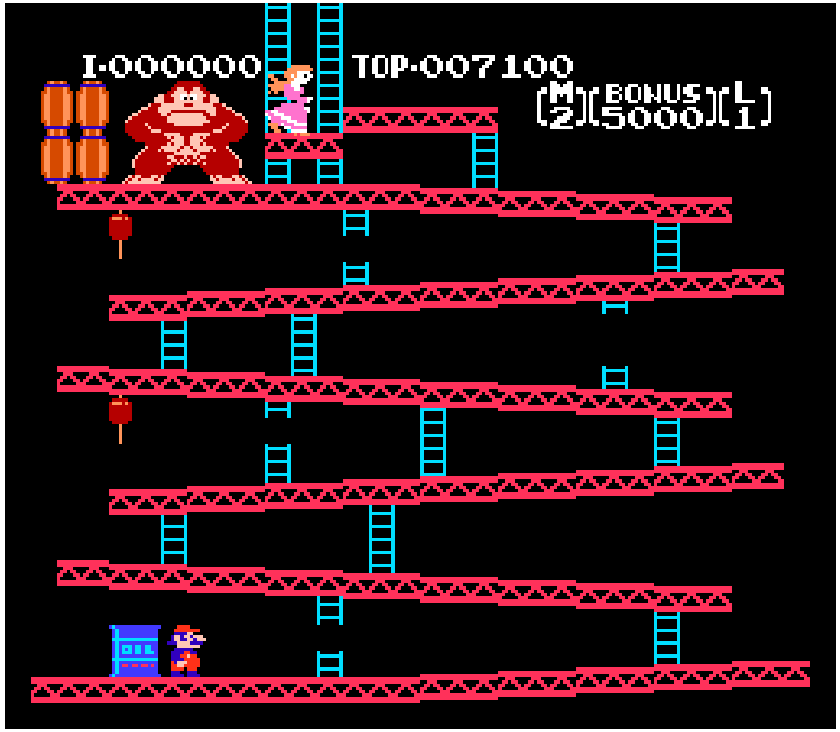
청크의 역할

- 제한된 메모리의 활용
- 게임 전체 흐름을 조정하는 편리한 방식
- 분업에도 유리

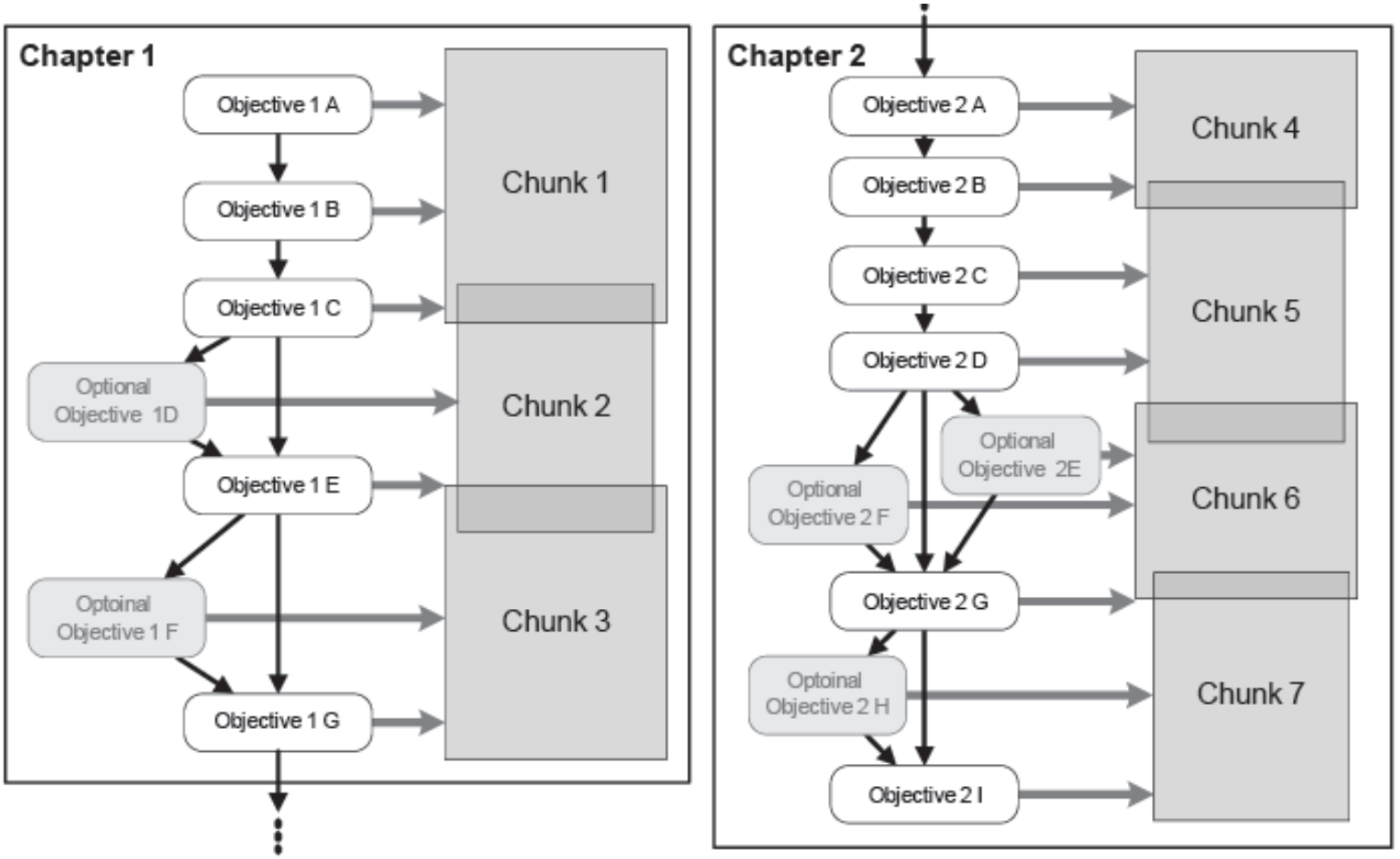
고차원 게임 진행(High-Level Game Flow)

- 플레이어의 목표를 연속적인 배열이나, 트리, 그래프 등의 형태로 정의한는 것
- 목표 - 임무, 스테이지, 레벨, 웨이브 …
- 각 목표의 성공 조건과 실패하는 경우의 벌칙을 지정

- 초기의 게임들은 월드 청크마다 하나의 목표를 지정
  - 새로운 레벨에는 새로운 목표



■ 오늘날의 게임들은 여러 개의 목표와 청크가 상호 연관



동적 요소 구현: 게임 객체

- **게임 객체(Game Object)**
  - 게임 월드의 거의 모든 동적 구소 요소를 지칭
  - Entity, Actor, Agent
  - 게임 객체의 본질: 속성과 행동의 모음
    - 속성 - 게임 객체의 현재 상태
    - 행동 - 시간에 따라, 혹은 이벤트에 반응해서 상태가 변하는 방식
  - 타입(Type)
    - 게임 객체는 타입에 의해 구분
    - 타입이 다른 객체는 다른 속성 스키마와 다른 행동을 가짐
    - 인스턴스 - 한 타입의 인스턴스는 속성 스키마와 행동은 같지만 속성들의 같이 달라짐.
  - 팩맨의 객체 타입
    - 유령, 알약, 파워 쿠키, 팩맨

게임 객체 모델

- **가상의 월드에 존재하는 동적인 존재들을 모델링하고 시뮬레이션할 수 있게 게임 엔진이 지원하는 기능들**

- **어떤 게임을 구성하는 구체적인 존재들을 시뮬레이션할 때 발생하는 문제들을 해결하는 데 쓰이는 특정한 객체지향 프로그래밍 인터페이스**
  - Ogre3d 엔진의 Entity 클래스와 API

- **게임 엔진을 만든 프로그래밍 언어를 확장하는 개념으로도 볼 수 있음.**
  - 파이썬, 루아 등의 스크립트 언어로 제어할 수 있는 게임 객체 인터페이스

툴 측면의 디자인과 런타임 구현 측면의 디자인의 차이

- **툴 측면의 객체 모델**
  - 디자이너가 월드 에디터에서 보는 게임 객체 타입들의 집합

- **런타임 구현상의 객체 모델**
  - 툴 측면의 객체 모델을 런타임에 구현하는데 사용한 프로그래밍 언어의 기능과 소프트웨어 시스템의 정의됨.

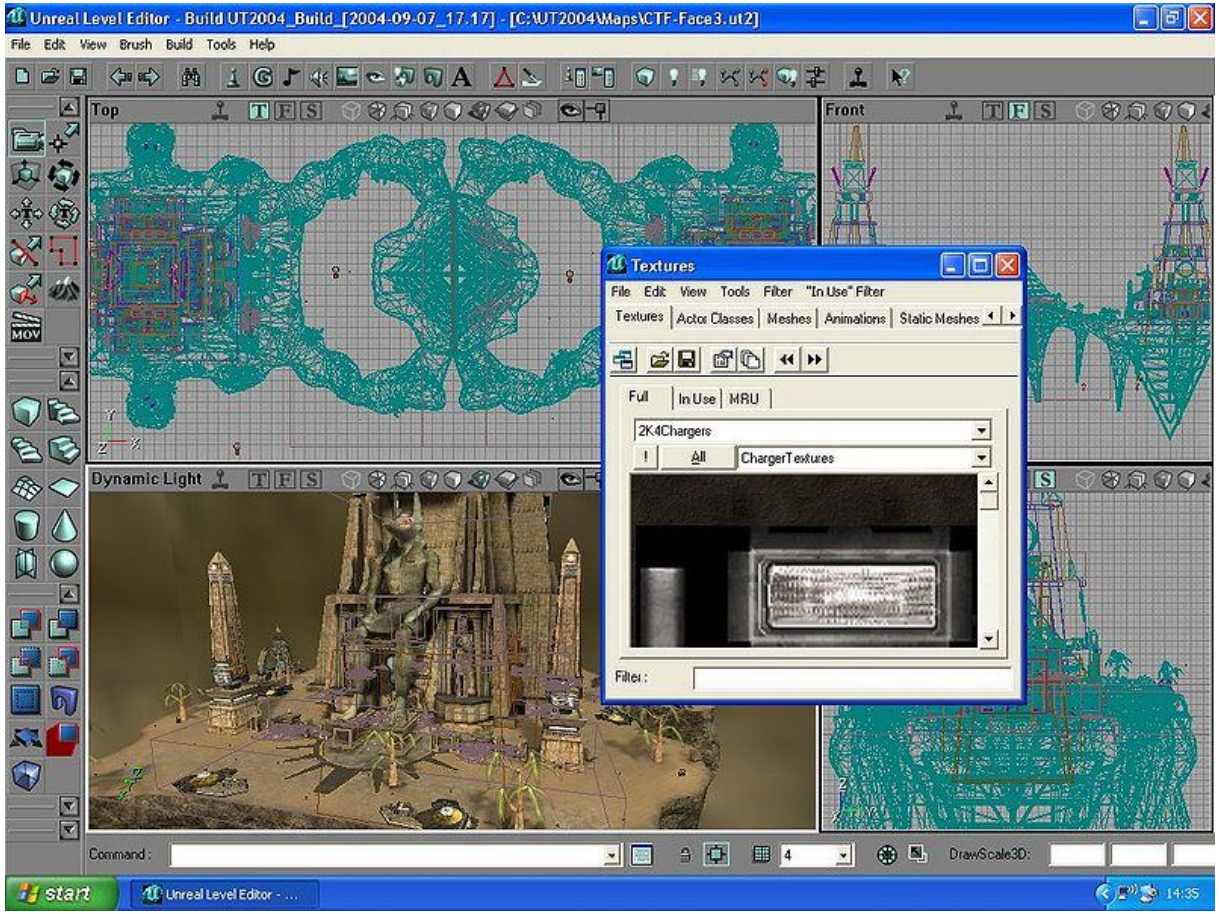
- **월드 에디터를 통한 객체 모델과 런타임 구현에 쓰이는 객체 모델은 일반적으로 일치하지만, 반드시 일치할 필요는 없음.**
  - 툴 상에서의 게임 객체 하나가 게임 실행 관점에서는 여러 클래스의 모음으로 구현될 수 있음.

데이터 주도 게임 엔진

- 초창기의 게임 개발은 대부분 하드코딩.
- 콘텐츠(리소스)의 양이 빠르게 증가함에 따라, 개발자들의 역량이 제작의 병목이 됨.
- 이제, 콘텐츠를 제작하는 사람들이 콘텐츠를 생성하고 컨트롤 할 수 있는 능력을 갖는 것이 중요해짐.
- Data-Driven Game Engine
  - 게임의 행동이 프로그래머가 제작된 SW에 의해서만 조정되지 않고, 아티스트와 디자이너가 데이터를 이용해서 조정할 수 있는 게임 엔진
  - 월드 에디터와 같은 툴이 필요
  - 다양한 입력 범위에 대한 안정적인 처리가 필요

게임 월드 에디터

- 게임 월드 덩어리를 정의하고, 그 안에 동적, 정적 구성 요소들을 채워 넣는 일
- 게임 객체들의 초기 상태값 지정
- 동적 구성 요소들의 행동을 조정
  - 스크립트 언어의 사용 또는 각종 파라미터의 조정



게임 에디터의 기능

- **월드 덩어리 생성과 관리**
  - 생성,결합, 분리,파괴
  - AI 네비게이션 지도, 플레이어가 짚을 수 있는 손잡이 정보, 엄폐 지점 정의
- **게임 월드 시각화**
  - 완성된 게임 월드 콘텐츠를 효과적으로 뷰잉
- **탐색**
  - 월드 안의 자유로운 탐색
- **선택**
  - 구성 요소들의 선택 - 레이캐스팅 등을 이용
- **계층**
  - 구성요소들의 계층화, 계층단위 편집, 시각화, ..
- **속성그리드**
  - 객체들의 속성의 조정
- **물체위치시키기와 정렬**
- **특수 객체 타입**
  - 조명, 파티클, 영역, 스플라인
- **월드 덩어리의 저장 및 로딩**

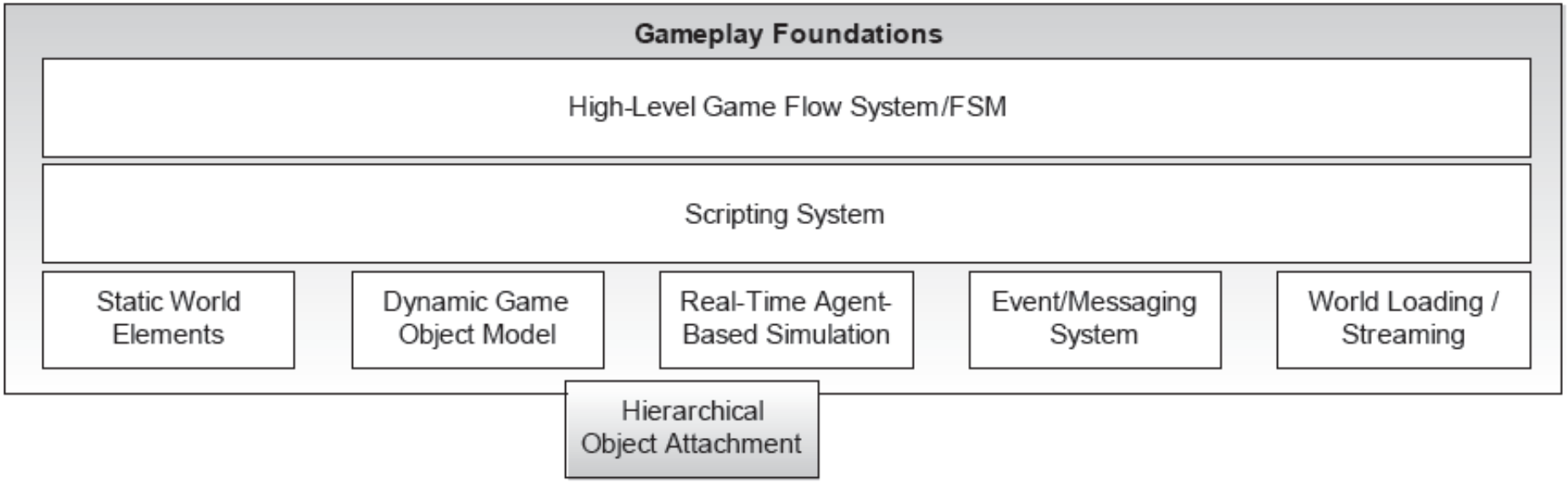
통합된 자원 관리툴

- **UnrealEd(언리얼 에디터)**
  - 게임 엔진과 직접 통한 - 에디터 변경 내용이 실행 중인 게임의 동적 구성 요소에 직접 적용
  - 빠른 반복 개발이 가능
  - 게임 자원 데이타베이스 관리 기능 통합
  - 전체 자원에 대한 실시간 WYSIWYG 보기 지원

- **데이터 처리 비용(Data Processing Costs)**
  - 대부분의 게임 리소스는 원본 소스를 제작한 후, 엔진에 최적화된 형식으로 변경하는 과정을 거침 (Export)
  - 월드 에디터에서 사용하는 자원이 원본일 수도 있고, export 된 형식일 수도 있음.
  - UnrealEd: 에디터로 원본 리소스를 로딩할 때, 포맷 변경이 됨. 빠른 반복 생산은 가능하나, 월드 에디터 상에서 원본 자원을 수정하기는 어려움.
  - 소스 엔진, 쿼이크 엔진: 최종적으로 게임을 빌드할 때, 리소스최적화가 수행.

게임 플레이 기반 시스템

- 게임 메카닉을 만드는데 기반으로 쓰이는 여러 런타임 소프트웨어 컴포넌트
- 게임엔진의 최상위 부분



static background geometry, like buildings, roads, terrain (often a special case), etc.;
dynamic rigid bodies, such as rocks, soda cans, chairs, etc.;
player characters (PC);
non-player characters (NPC);
weapons;
projectiles;
vehicles;
lights (which may be present in the dynamic scene at run time, or only used for static lighting offline);
cameras;

– Python
http://www.python.org

– Lua
http://www.lua.org

– GameMonkey
http://www.somedude.net/gamemonkey

– AngelScript
http://www.angelcode.com/angelscript

- **런타임 객체 모델**
  - 월드에디터를 통해서 인지하는 가상의 객체 모델을 실제로 구현하는 부분
- **레벨관리 및 스트리밍**
  - 가상 월드 콘텐츠를 불러오고 내리는 시스템
  - 빠른 메모리 스트리밍으로 심리스 월드를 구현
- **실시간 객체 모델 업데이트**
  - 모든 객체들의 주기적 업데이트
  - 이 부분에서 게임 엔진의 온갖 시스템들이 합쳐져 유기적인 전체 시스템을 이룸.
- **메시지와 이벤트 처리**
  - 객체들 간의 통신 - 대개 추상적 메시지 시스템을 통함.
  - 이벤트 시스템이라고도 함 - 게임 월드 상태의 변화에 따른 이벤트를 메시지로 전달
- **스크립트 시스템**
- **목표 및 게임 흐름 관리**
  - 챕터, 레벨, 월드 사이의 이동
  - 플레이어의 목표 성취의 기록

런타임 객체 모델

- **동적으로 게임 객체를 생성하고 파괴하기**
  - 게임 플레이 중 생성되고 소멸되는 동적 요소의 구현
    - 체력 아이템, 폭발, 새로운 적의 출현

- **로우레벨 엔진 시스템과 연동**
  - 게임 객체의 렌더링, 애니메이션, 물리시뮬레이션을 위해 하위 엔진 시스템과 연계가 필요

- **객체 행동 실시간 시뮬레이션**
  - 모든 게임 객체들의 상태를 동적으로 업데이트해야 함.
  - 경우에 따라 정해진 순서에 따라 업데이트 필요
    - 객체간의 의존성, 객체의 하위 엔진 시스템에 대한 의존성, 하위 시스템 간 상호 의존성

- **새로운 객체 타입을 정의할 수 있는 기능**
  - 월드 에디터에서 새로운 객체 타입을 쉽게 정의하고 추가하는 기능
  - 대부분의 엔진은 프로그래머의 손을 거쳐야 함.

- **고유 객체 식별자 ID**
  - 숫자, 또는 문자

- **게임 객체 질의**
  - 게임 월드 내의 객체들을 찾을 수 있는 방법(식별자들 통해)
  - 특정 조건을 만족하는 객체의 탐색(ex. 플레이어 캐릭터 주위 20미터 안의 모든 적을 탐색)

- **게임 객체 참조(Reference)**
  - 인스턴스의 포인터, 핸들, 스마트 포인터 등

- **유한 상태 기계 지원**
  - 게임 객체를 FSM으로 모델링

- **네트워크 복제(Replication)**
  - 네트워크 게임에서 PC 한 대만 게임 객체를 소유하고 관리
  - 다른 PC에서는 복제된 객체가 존재

- **게임 Save 와 Load**
  - 게임 상태의 저장, 게임 객체들의 serialization

런타임 객체 모델 구조

- **런타임 객체 모델은 월드 에디터에서 보이는 객체 타입과 속성, 행동을 충실하게 재현해야 함.**

- **객체 중심적(Object-centric) 구조**
  - 툴 상의 게임 객체 ←→ 클래스 인스턴스 하나, 또는 여러 개의 인스턴스 집합
  - 객체 = 속성 + 행동
  - 게임 월드는 다양한 게임 객체들의 집합

- **속성 중심적(Property-centric) 구조**
  - 게임 객체들은 고유의 ID로만 표현
  - 속성은 테이블에 담겨있고, ID를 통해서 속성에 액세스
  - 속성들은 하드코딩된 클래스의 인스턴스루 구현
  - 게임 객체 행동 - 객체를 이루는 속성들의 집합에 의해 간접적으로 정의
    - 'health' 속성 - 객체가 손상을입거나 체력을 잃을 수 있고, 죽을 수도 있음.
    - 'MeshInstance' 속성 - 삼각형 메쉬로 3D 렌더링할수 있는 객체

객체 중심 구조 사례

- C로 구현한 단순한 객체 기반 모델: 하이드로 썬더
  - type
    - 보트 : 플레이어 또는 AI 가 컨트롤, 떠있는 푸른/붉은 가속 아이콘, 애니메이션되는 배경 객체
    - 물 표면, 폭포, 파티클 효과
    - 레이스 트랙 섹터 - 2차원 다각형 영역
    - 정적 기하 형상, 2D HUD 요소

```
struct WorldOb_s
{
  Orient_t   m_transform;     /* position/rotation */

  Mesh3d*    m_pMesh;         /* 3D mesh */
  /* ... */
  void*      m_pUserData;     /* custom state */

  void       (*m_pUpdate)();  /* polymorphic update */
  void       (*m_pDraw)();    /* polymorphic draw */
};
typedef struct WorldOb_s WorldOb_t;
```

공통속성

고유 속성

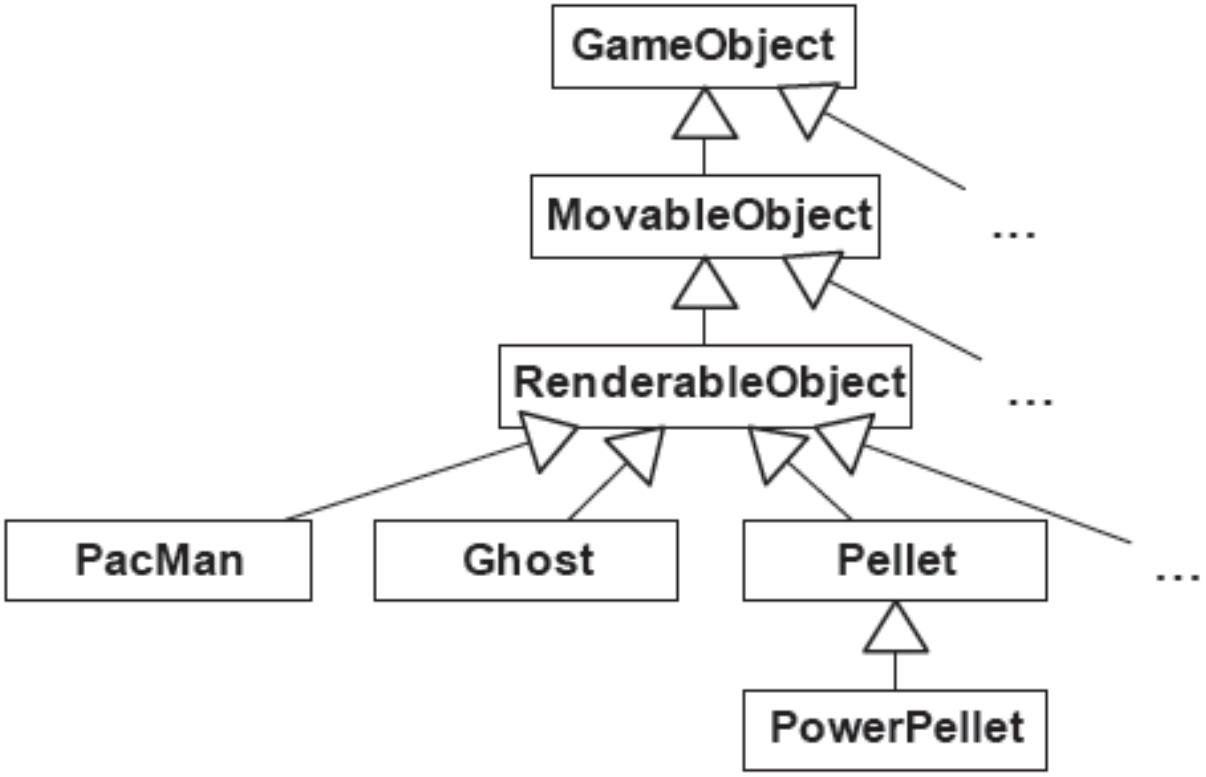
```
struct WorldOb_s
{
  Orient_t  m_transform;     /* position/rotation */

  Mesh3d*   m_pMesh;         /* 3D mesh */
  /* ... */
  void*     m_pUserData;     /* custom state */

  void      (*m_pUpdate)();  /* polymorphic update */
  void      (*m_pDraw)();    /* polymorphic draw */
};
typedef struct WorldOb_s WorldOb_t;
```

공통속성

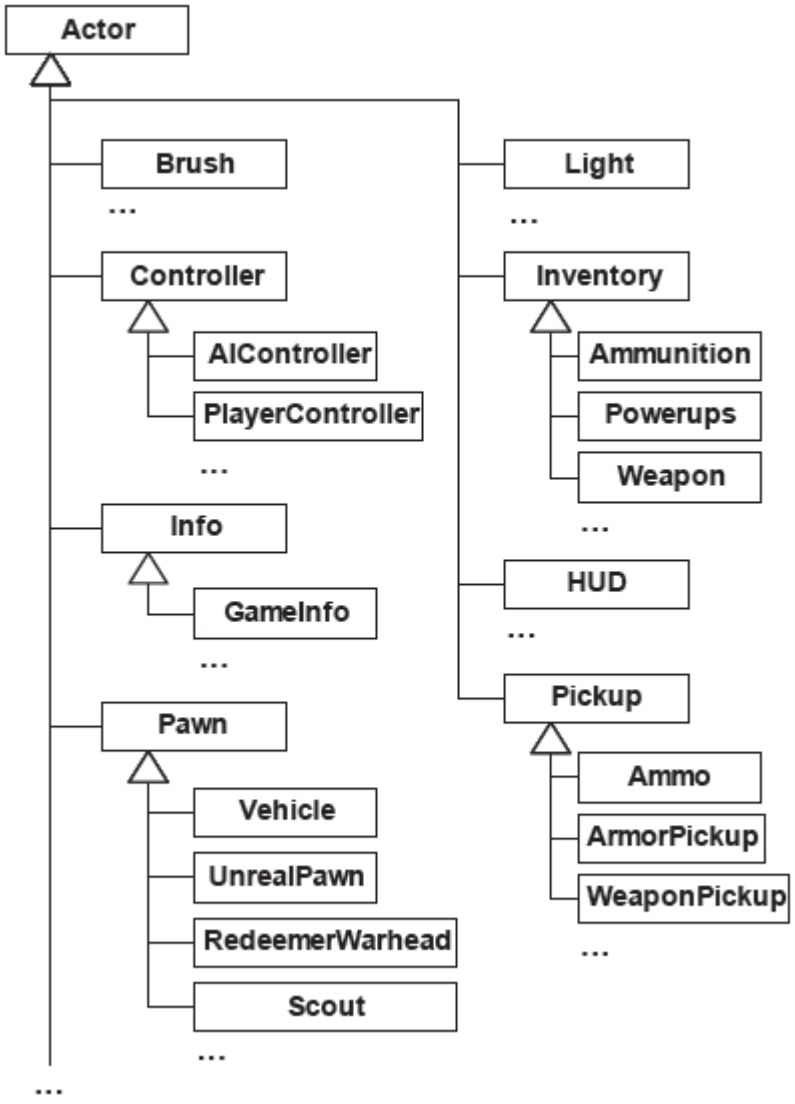
고유 속성

거대 단일 클래스 계층: 팩맨

- 게임 객체 타입을 분류하는데 생물 분류학과 비슷한 분류법을 사용함.
- 클래스 계층이 커지면서 구조는 점점 깊어지는 동시에 넓어짐.
- 하나의 공통 베이스 클래스를 거의 모든 게임 객체들이 상속.



Unreal Tournament 2004 게임 객체 클래스 구조



깊고 넓은 계층의 문제점

- **클래스를 이해하기 힘들고 유지 및 수정이 어려움**
  - 계층이 너무 깊다. 자식 클래스를 이해하려면 부모 클래스를 줄줄이 다 이해해야 함.
  - 자식 클래스의 가상 함수 하나를 수정하면, 자칫 부모 클래스의 가정 규칙을 어길 수 있음.

- **여러 계열 분류 구조를 구현할 수 없음.**
  - 생물학적 분류법(계,문,강,목,과, 속, 종 분류)과 유사한 방법.
  - 문제점: 객체들을 나눌 때 트리의 한 레벨에서는 오직 한 가지 기준에 따라서만 분류가 됨.
  - 일단 기준이 결정되면, 다른 기준에 의해 분류하기가 거의 불가능.
  - 아래 Vehicle 클래스 구조에서 수륙 양용차를 새로이 만드려면?