스크립트 기반의 게임 기획 및 개발을 위한 통합 개발 환경

Game Planning and Development IDE based on Script

최한용*, 이돈양**, 박의준*

한북대학교 컴퓨터공학과*, 경인여자대학 전산정보학과**

Han-Yong Choi(hychoi@hanbuk.ac.kr)*, Don-Yang Lee(dylee62@empal.com)**,
Wee-Joon Park(parkwj@hanbuk.ac.kr)*

요약

게임 개발 환경을 개선하기 위해 고려해야할 중요한 사항중 하나가 게임을 구현하기 위한 언어이다. 현재 게임을 개발하기 위해 설계되어있는 전용 언어는 부족한 상황이다. 따라서 게임개발은 범용의 프로그래밍 언어를 이용하여 직접적으로 엔진을 모델링하여 구현하고 있다. 따라서 개발자는 쉽게 게임을 개발하기 어려우며 엔진레벨에서 프로그래밍을 해야한다. 엔진을 잘 이해하지 못한 환경에서 게임을 개발하는 것은 상당히 어려운 일이다. 개발자와 기획자는 쉽게 게임을 개발하기 위해 기획자 측면에서사용할 수 있는 추상화된 상위언어를 사용하고자 한다. 그리고 기획자와 개발자 사이의 의사교환을 위한 도구도 없는 상황이다. 그러므로 본 연구에서는 추상화된 엔진레벨에서 모델링이 가능한 스크립트 언어를 정의하였다. 그리고 추상화된 고수준의 언어를 이용하여 게임을 개발할 수 있는 통합환경을 구축하였다. 스크립트 언어는 빠른 속도로 게임을 개발할 수 있도록 하였다. 따라서 개발자는 엔진레벨이 아닌 기획자와 게임을 개발할 수 있다. 또한 기획자와 개발자는 통합환경을 이용하여 스켈레톤 게임을 만들 수 있기 때문에 의사교환 도구로 이용할 수 있다.

■ 중심어 : | 추상 스크립트 | 게임 엔진 | 개발 프로세스 |

Abstract

It is important matter to develop game that language should consider improving game development environment. It is hardly any language that is designed to develop a game. Most developer used general purpose programming language in these situations, and directly modeling with engine. For this reasons, developers cannot easily develop games and must programming on engine level. It is hard to develop a game in this environment that was not able to understand an engine well. Moreover, developer and planner want to use an abstracted high-level language on planner phase and are going to easily develop a game. It was not have communication tool between developer and planner. Therefore, we defined a script language for modeling based on abstract engine level. In addition, we did build IDE to develop game using abstracted high-level language. It was able to develop a game on high-speed development environment. Therefore, developer does not must develop with engine phase but can develop a game with planners. In addition, game planner and developer can use a communication tool because it is able to develop skeleton game.

■ Keyword : | Abstact Script | Game Engine | Developmemnt Process |

접수번호 : #050404-001
접수일자 : 2005년 04월 04일
심사완료일 : 2005년 04월 25일
교신저자 : 최한용, e-mail : hychoi@hanbuk.ac.kr

I. 서 론

게임 시장이 성장하면서 게임의 개발 난이도와 비용 등이 계속해서 증가하고 게임 개발 생명주기가 짧아지고 있다. 빠른 생명주기에 대응하기 위해 반복적으로 사용되는 모듈의 재사용과 생산성을 높이기 위한 툴 도입을 위한 노력이 게임 개발 업계에서도 끊임없이 이루어지고 있다[1]. 개발하고자 하는 게임이 기획되었다면 게임을 개발하기 위한 플랫폼과 기능들을 알아보고 타당한지에 대한 결정을 내려야 한다. 엔진을 개발할 능력이 된다면 게임에 구동하기 위한 엔진을 개발하여야 한다. 그러나 기존의 엔진을 도입해서 개발가능하다면 자신의 콘텐츠에 맞게 엔진을 수정해서 개발할 수도 있다. 그러나 엔진 개발에 소요 비용 대비 효과가 떨어지기 때문에 전문 엔진 개발 업체의 상용 엔진을 사용하는 것이 일반적인 추세가 되고 있다[2]. 따라서 개발기간을 단축시키고 생산성을 높이기 위해 기존의 엔진을 도입하여 개발하는 방법이다. 그러나 엔진을 도입하는 경우 엔진을 직접적으로 제어할 수 있는 정도의 기술 수준이 되지 않으면 개발에 많은 어려움이 있다. 엔진을 도입하여 게임을 개발하는 경우에도 단지 하부 구조를 구현 하지 않았을 뿐, 엔진 내용을 이해할 정도의 수준이 되어야 하는 문제점을 안고 있다. 그리고 게임 기획자는 엔진의 기술적 정보를 이해하지 못하고 있을 경우 기획의도가 개발환경에서 변형되는 경우가 많고, 기획단계에서부터 기획자와 개발자간 의사소통의 어려움을 겪고 있다. 그러므로 프로그램을 이해하는 기획자 수준에서도 스켈레톤 게임을 개발할 수 있는 통합 환경이 필요하며, 동일 장르의 게임을 빠른 시간에 파생적으로 만들어 낼 수 있는 환경을 제공하여야 한다. 그리고 개발 프로덕트의 최종 품질 및 개발비용 등에 관련하여 사용할 엔진을 교체한다고 하더라도 개발 환경은 동일하게 가져갈 수 있도록 한다[3][4].

본 논문에서는 개발에 착수하기 전에 기획자와 개발자간의 의사소통이 가능하며, 엔진단계보다 한 단계 추상화된 수준의 개발환경을 제시하고 있다. 스크립트를 이용하여 게임 내에서 사용되는 오브젝트별로 코드를 삽입할 수 있으며, 게임 전체 관리 부분, 모델의 AI 부분, 카메라 조작 부분으로 구성하였다. 그러나 본 논문의 목표는 직접적인 엔진 제어를 줄여 추상화작업이 가능한 통합 환경 연구를 목표로 하기 때문에 이를 추상화한 상부계층의 스크립트 언어를 이용하여 VM(Virtual Machine)의 기능인 애니메이션, 컬링시스템, 라이트맵, 빌보드, 파티클 시스템의 기능을 제어 할 수 있도록 하였다[5]. 통합 개발 환경은 엔진의 이해도가 낮아도 간단한 스크립트의 문법정도를 익힘으로써 기획자가 쉽게 게임을 개발할 수 있는 환경을 제공 하도록 하였다.

따라서 게임엔진을 직접적으로 모델링하지 않고 개발하기 위한 추상화된 상위 단계의 스크립트 언어를 이용하여 스켈레톤 게임을 기획할 수 있다. 그리고 직접적으로 하부구조의 엔진 구현 및 엔진의 내용을 잘 이해할 정도의 수준인 고급 개발자가 아니더라도 스크립트언어를 이용하여 간단하게 게임을 개발할 수 있 수 있는 통합 환경을 제시하였다. 본 논문은 서론에 이어 2장은 스크립트언어의 구성방법에 대하여 설명하였으며, 3장은 개발프로세스 및 엔진의 구성요소를 중심으로 통합 개발 환경에 대하여 논의하였다. 마지막으로 4장에서는 본 논문에 대한 평가 및 결론에 대하여 논하였다.

II. 스크립트언어의 구성

1. 구성요소

스크립트는 게임 환경에서 프로그래머뿐만 아니라 기획자에 의해서 게임 환경을 컨트롤할 수 있게 도와줄 수 있다. 게임로직을 게임 내부에 넣었을 경우에 게임로직의 작은 변화에도 개발자가 게임로직을 수정해야하는 반면 복잡한 게임로직을 스크립트의 수정으로 빠르게 게임을 변경할 수 있다. 그러나 스크립트를 사용하는 경우 일반적으로 속도의 저하가 있으므로, 전체 게임로직에서 스크립트를 사용하는 정도는 많은 테스트와 게임의 용도를 통해서 결정해야 한다. 게임 내에서 자체적으로 게임 스크립트를 개발하기 위해서 스크립트 컴파일러 및 스크립트가 실행할 수 있는 런타임 VM이 개발되어야 한다.

따라서 본 논문의 스크립트 언어는 [그림 1]과 같이

게임 전체 관리 부분, 모델의 AI 부분, 카메라 조작 부분의 3가지 스크립트 구성요소를 중심으로 설계하였다.

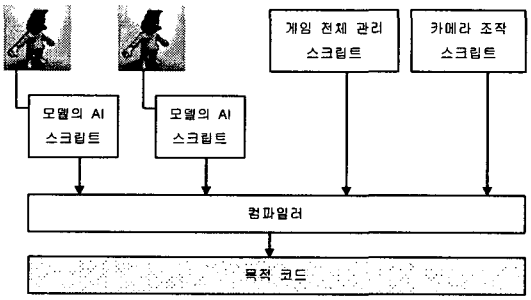


그림 1. 스크립트 언어의 3가지 요소

첫 번째, 게임 전체 관리 부분은 전체적인 게임의 흐름(flow)을 결정하고 게임 제작 전반에 관련한 변수들을 관리하는 부분이다. 게임의 내용을 적재(load)하고 저장(save)하는 기능을 갖는다. 개별 엔진을 유기적으로 통합하고, 각종 데이터를 효율적으로 관리하며, 기획을 충실히 반영하여 컨텐츠를 개발하는 데에 필요한 모듈을 담당한다. 랜더링, 애니메이션 등 엔진의 통합과 게임 객체의 관리를 담당한다.

두 번째, 모델의 AI 부분은 게임에 등장하는 모든 캐릭터의 AI 부분을 조작하는 스크립트 부분이다. 캐릭터의 AI부분은 각 캐릭터마다 따로 프로그래밍을 할 수 있도록 UI가 제공되는데, 게임 전체 관리 부분과 연동되어 게임 시나리오에 따라 캐릭터를 여러 가지 상황으로 만들어 낼 수 있다.

마지막으로, 카메라 조작 부분은 게임 전체와 모델들의 상황을 독자적으로 판단하여 적절한 카메라 뷰(view)를 만들어 내는 스크립트 부분이다

2. 스크립트 언어의 작성

본 연구에서는 스크립트의 문법 학습 시간을 단축하기 위한 C언어의 구조를 간소화하여 스크립트의 구문을 정의하였으며, 의미 구조는 다음과 같이 BNF문법으로 정의하였다. 스크립트는 [그림 2]와 같이 두 가지 방법으로 작성할 수 있다. 스크립트는 각 개체(object)의 동작 스크립트를 편집기(editor)로 편집하여 작성되기도 하며, 간단한 마법사(wizard) 형식으로 자동 생성할 수 있다.

표 1. 스크립트의 BNF

```
1. program → declaration-list
2. declaration-list → declaration-list declaration | declaration
3. declaration → var-declaration | fun-declaration
4. var-declaration → type-specific VAR; | type-specific VAR[NUM];
5. type-specifier → int | string | void
6. fun-declaration → type-specific VAR(params) compound-stmt
7. params → param-list | void
8. param-list → param-list, param | param
9. param → type-specifier VAR | type-specifier VAR [ ]
10. compound-stmt → { local-declarations statement-list }
11. local-declaration → local-declarations var-declaration | empty
12. statement-list → statement-list statement | empty
13. statement → expression-stmt | compound-stmt |
                selection-stmt | iteration-stmt | return-stmt
14. expression-stmt → expression ; | ;
15. selection-stmt → if ( expression ) statement |
                     if ( expression ) statement else statement
16. iteration-stmt → while ( expression ) statement |
    for (expression ; simple-expression ; simple-expression) statement
17. return-stmt → return; | return expression ;
18. expression → var = expression | simple-expression
19. var → VAR | VAR [ expression ]
20. simple-expression → additive-expression relop additive expression
                      | additive-expression
21. relop → <= | < | > | >= | == | !=
22. additive-expression → additive-expression addop term | term
23. addop → + | -
24. term → term mulop factor | factor
25. mulop → * | /
26. factor → ( expression ) | var | call | NUM
27. call → VAR(args)
28. args → arg-list | empty
29. arg-list → arg-list, expression | expression
```

그리고 스크립트 편집기는 좌측에 각 컴포넌트 리스트를 관리하고 우측에서 각 컴포넌트의 스크립팅 과정에서 Syntax highlighting 기능을 제공하여, 문법적 오류를 수정할 수 있도록 하였다.

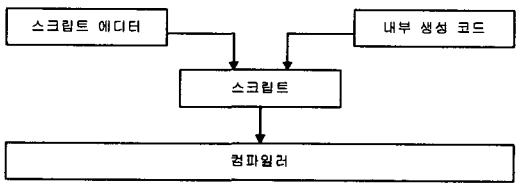


그림 2. 스크립트 편집방법

3. 스크립트의 추상화

통합 개발 환경을 이용하여 작성되는 스크립트는 컴

파일 과정을 거쳐 목적 코드인 게임 전용 코드로 만들어진다. 목적코드에 해당하는 게임 전용 코드는 직접적으로 엔진을 조작할 수 있도록 하였다. 따라서 [그림 3]과 같이 스크립트 언어는 사용자의 개발편의성을 고려하여 이 목적코드를 추상화한 것이다. 그러므로 게임엔진을 잘 이해하고 있는 고급 개발자는 게임엔진 단계에서 직접적으로 제어할 수 있으며, 초급 개발자를 비롯한 기획자는 엔진을 추상화한 개발환경에서 스크립트 언어를 이용하여 게임을 개발할 수 있다.

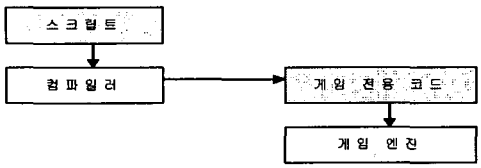


그림 3. 2차 구조의 스크립트 추상화

게임 전용 코드는 게임 엔진을 구동하여 게임에 관련된 실제 작업을 엔진레벨에서 수행하게 된다. 게임 스크립트 컴파일러는 크게 front-end와 back-end의 두 부분으로 구성하였다. 앞단에서는 파싱 작업으로 캐릭터 문자열을 읽어서 AST(abstract syntax tree)로 변환하는 기능을 담당하고 나머지 단계에서 분석이나 변환이 쉽도록 하였다. 이때 구문분석기가 문자열을 스크립트에서 지원하는 키워드와 사용자가 정의하는 ID, 상수값 등의 토큰으로 분리한다. 파싱의 결과로 AST가 생성되고 나머지 단계들은 AST를 기반으로 수행된다. 만들어진 AST는 문법적인 사항만이 트리로 바뀌어져 있을 뿐 그 의미가 올바른지는 알 수 없으므로 파싱 후의 작업들은 나중의 코드 생성할 때 올바른 정보로 코드 생성을 위해서 그 이전에 오류를 검출한다. 오류 점검 요소로는 함수의 return 문이 존재하는지, 함수 call에서 인자가 올바른 타입인지, 선언된 변수의 이름이 규칙에 맞는지 등이다. 이 단계에서 오류가 존재하면 사용자에게 오류 메시지를 보내고 컴파일을 중단한다. 이 단계에서 암시적 형 변환(implicit casting)의 명시적 형 변환(explicit casting)으로의 변환 등을 추가하고 필요없는 AST 노드를 정리함으로써 중간코드형태를 만들 수 있다.

III. 통합 개발 환경

1. 통합 개발 환경

통합 개발 환경은 [그림 4]와 같이 게임 엔진을 이용하여 게임을 개발하는 기본적인 프로세스를 따른다. 기획에서 나온 스토리보드 내용을 게임로직에 적용할 수 있도록 통합적인 개발 시스템을 제공하며 복잡한 게임 개발 프로젝트에서 각 컴포넌트의 제작 관리를 쉽게 하기 위해 개발 도구 내에서 스크립트 에디터와 리소스 관리 및 파티클을 비롯한 게임 정보를 구성할 수 있도록 하였다. 그리고 스크립트를 이용하여 스켈레톤 게임을 구성할 수 있으므로, 게임개발 작업에 착수하기 전에 기획단계에서 구성된 시나리오에 의해 게임이 어떻게 보여질 수 있을지 데모게임을 작성하여 시나리오를 검토할 수 있으며 기획자와 개발자간의 의사교환 방법으로 이용할 수 있다.

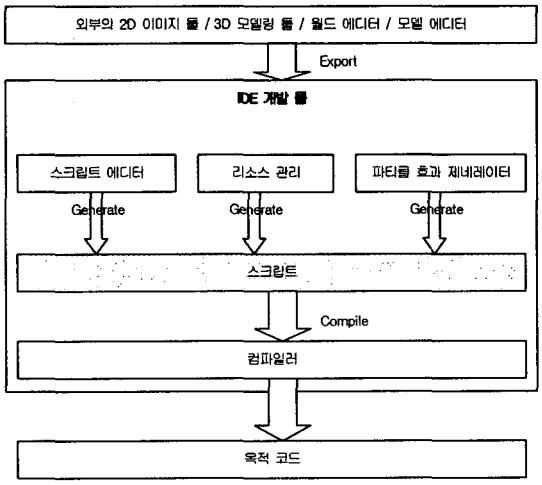


그림 4. 개발 프로세스

스크립트를 구동하기 위한 버추얼 머신은 Microsoft DirectX 기반의 비주얼 C++ 제작된 3D 게임 엔진이며, [그림 5]와 같이 3D 그래픽 라이브러리를 중심으로 설계하였다.

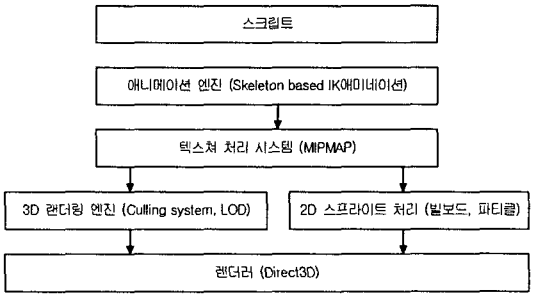


그림 5. 게임 엔진 그래픽 부분

애니메이션 엔진은 게임 내에서 [그림 6]과 같이 캐릭터의 애니메이션 부분을 구현하는 것으로 모션 캡쳐링 된 게임 애니메이션 데이터를 재생할 수 있는 스켈레톤 기반의 애니메이션을 지원한다. 3D MAX와 같은 외부 3D 모델링 프로그램으로부터 메쉬(mesh), 텍스쳐(texture), 애니메이션(animation) 등을 자체 포맷으로 외포(export)하여 엔진에서 사용할 수 있다. 개발 전 과정 중 등장인물의 모든 동작 제어를 위한 정보를 구성한다.

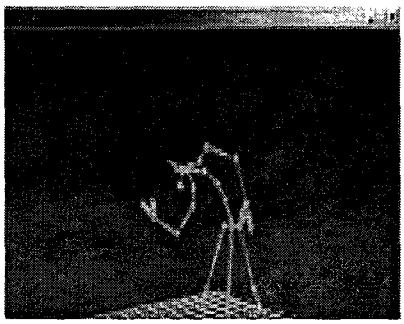


그림 6. 캐릭터 애니메이션

컬링(culling) 시스템은 지오메트리(geometry)가 카메라의 시야범위를 정의하는 박스내부에 있는지, 외부인지, 부분적으로 걸쳐있는지를 판단해서 그려질 것인가 말 것인가를 판단하는 처리 과정이다. 컬링은 [그림 7]과 같이 주어진 노드의 부모 노드에서부터 프러스텀(frustum) 내에 있는지, 외부인지, 부분적인지를 재귀적으로 탐색한다. 모든 노드가 프러스텀(frustum) 내부에 있다면 노드 안에 있는 모든 것을 그리고, 노드가 완전히 외부에 있다면 전혀 그리지 않는다. 부분적으로 걸쳐있다면 현재 노드의 최종노드까지 하위노드로 탐색해 들어가 렌더링될 것인지를 판단한다.

게임 그래픽 데이터를 현재 상황에 맞게 보이는 면을 선별하여 화면 랜더링을 효과적으로 하기 위해 Solid Leaf BSP(Binary Space Partioning tree)트리 방식을 적용하였다. BSP로 전체 공간을 나누고, 카메라 위치에서 보이는 공간을 재계산하기 때문에 항상 보이는 면만 랜더링하므로, 전체 스테이지의 모델 크기와 랜더링 속도는 상관없다.

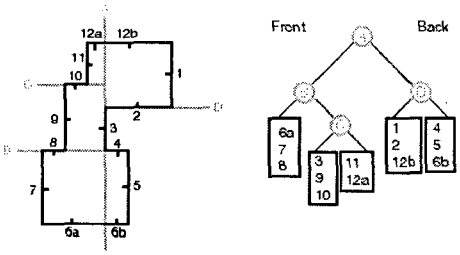
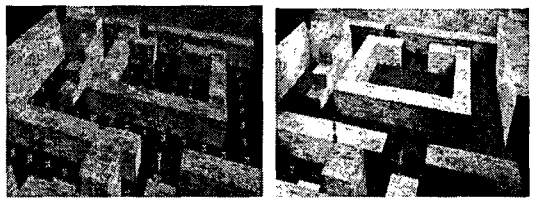


그림 7. 컬링 시스템

이때 맵을 구성하는 삼각 폴리곤(Triangle Polygon)을 n개의 꼭지점을 가지는 다각형으로 변환한 후에 Solid Leaf BSP를 생성한다. 그리고 생성한 BSP 트리를 이용하여 각각의 Leaf를 분할하는 Potal Polygon을 생성하고, Potal Polygon과 Leaf의 연결성을 조사하여 PVS를 생성하도록 하였다.



(a) 라이트 맵 적용 전 (b) 라이트 맵 적용 후

그림 8. 라이트 맵(light map)

라이트 맵 부분은 광원이 움직이지 않는다는 가정하에서 [그림 8]과 같이 각 라이트 정보를 기본으로 각 폴리곤에 대한 라이트 맵을 미리 생성 하여 여러 개의 광원에 대한 라이트 효과를 미리 계산해 놓았다.

표 2. 라이트 맵 생성 알고리즘

1. 라이트맵을 적용하고자 하는 폴리곤을 평면에 사영시킨 후 폴리곤에 라이트맵에 사용 할 UV좌표를 할당한다.

2. UV 좌표 할당 후, 폴리곤에서 (u=0,v=0), (u=1,v=0), (u=0,v=1) 에 해당하는 좌표를 월드공간으로 변환한다.

3. 해당 벡터가 차지하는 영역에 대해서 루프를 돌며 광원이 미치는 밝기를 계산한다.

게임 내에서는 미리 계산된 라이트 맵을 이용하여 광원이 주는 빛의 효과를 멀티 텍스처로 구현할 수 있으며, [표 2]에 제시된 알고리즘 과정을 거쳐서 라이트 맵을 생성하였다.

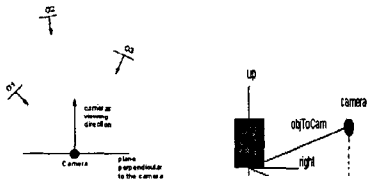


그림 9. 객체의 lookAt 정보

빌보드 파트 부분은 파티클 시스템을 구현하기 위한 기본 파트로서 [그림 9]와 같이 사용자에게 항상 정면을 보여주는 스프라이트 루틴을 개발하였으며 Cylindrical Billboards 스타일을 적용하였다.

표 3. 빌보드 파트 생성 알고리즘

```
1. normalize(objToCamProj)
2. angleCosine = innerProduct (lookAt,objToCa
   mProj)
3. upAux = crossProduct(lookAt,
   objToCamProj)
4. glRotatef(acos(aux)*180/3.14,upAux[0],
   upAux[1], upAux[2]);
```

이때 빌보드 파트를 위한 계산과정은 [표 3]에 제시 알고리즘을 거쳐 생성할 수 있다. 파티클은 [그림 10]과 같은 연기, 불, 폭발 등을 표현하기 위해 수많은 작은 파티클을 생성하는 시스템이며 생성과정은 [그림 11]과같이 파티클 스프라이트 라이브러리를 스프라이트 루틴으로 랜더링 하도록 하였다.

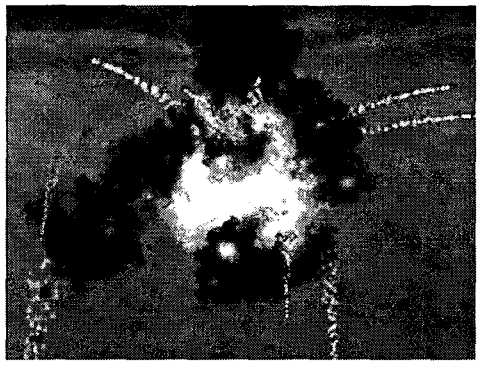


그림 10. 폭발하는 파티클 예

파티클 시스템은 여러 개의 스프라이트를 이용하여 구현하였으며, 파티클에 사용되는 스프라이트는 카메라에 항상 정면이 되도록 하는 빌보드형 스프라이트 내부 루틴을 사용하였다.

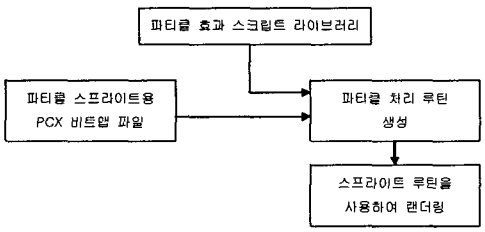


그림 11. 파티클 프로세스

각 파티클의 움직임 처리 루틴에 따라 다양한 효과를 만들어낼 수 있으며, 파티클 움직임 처리는 IDE환경 내의 파티클 제네레이터를 통해 쉽게 만들 수 있도록 하였다. 개발 IDE에서는 파티클 효과용 라이브러리를 통해 미리 만들어진 템플릿 소스를 제공한다. 템플릿 소스 수정을 통해 사용자가 파티클의 움직임을 제어할 수 있도록 하였다.

2. 리소스 관리 및 게임 구성

IDE 툴의 좌측 창은 리소스 관리 창으로서 게임용 리소스 편집을 위한 외부 에디터와 연결되어 있다.

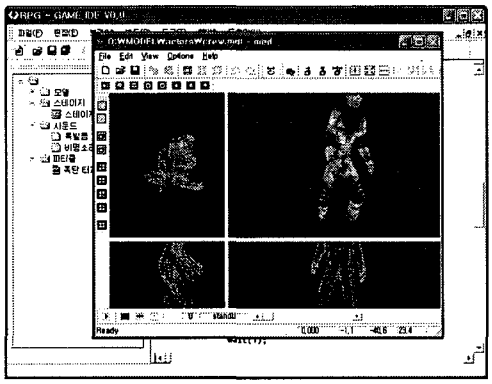


그림 12. 모델 리소스 에디터

따라서 리소스와 IDE의 개발과정은 동적링크에 의해 리소스를 관리할 수 있다. [그림 12]는 리소스에서 제공하고 있는 정보를 스크립트로 조합하여 모델을 생성한 예를 보이고 있다.

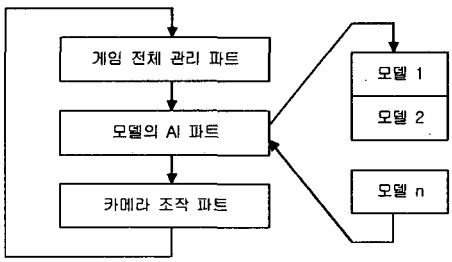


그림 13. 게임 구성

IDE 툴은 각 모델에 대해 고유의 스크립트를 독립적으로 제공한다. 각 모델에 대해 게임 루프를 가질 수 있으며, 모델은 독자적으로 편집이 가능하다. 스크립트에 의해 관리되는 모델은 컴파일되는 목적 코드 상에서 게임 루프를 [그림 13]과 같이 구성하고 있으며, 이 때 모델의 AI 파트는 모델의 숫자가 늘어나는 만큼 리스트로 추가되어 관리된다. 이때 AI는 [그림 14]와 같이 모델에 사용할 AI를 스크립트로 제어할 수 있으므로 FSM (Finite State Machine), FuSM(Fuzzy State Machine), Decision Tree 로 구성되어 있는 엔진레벨의 AI 루틴을 간단하게 제어할 수 있다. 따라서 모델의 AI 파트 정보는 카메라 조작파트의 속성을 이어받도록 하였다.

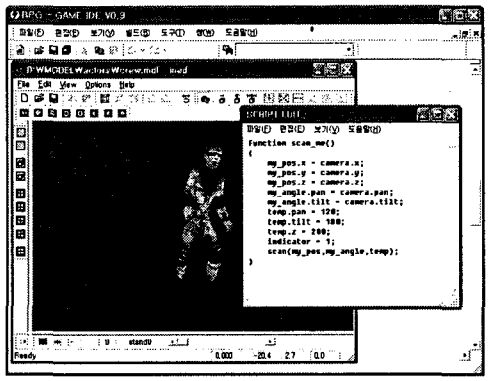


그림 14. 모델의 AI 루틴

IV. 평가 및 결론

일반적으로 고가의 게임 엔진(수천만원~수억원대)을 도입한다면, 생산성과 개발 편의성은 크게 향상시킬 수 있으나, 엔진 자체의 라이센스 비용이 수억원에 달하는 등 개발비용에 큰 부담을 준다는 단점이 있다. 그리고 저가의 게임 엔진의 경우(수십만원대~수백만원대)에는 통합적으로 개발할 수 있는 IDE 환경이 없으며, 자체 스크립트 편집도구 조차 없는 경우가 대부분이어서 일반 개발 환경에 비해 편의성이 열악해진다는 단점이 있다. 따라서 기존의 개발방법 중 엔진을 도입한 개발방법에서 나타나는 문제점은 개발자의 숙련도 그리고 기획자와 개발자간의 의사교환이 있었다. 본 논문에서는 게임개발에서 소요되는 노력 중 엔진개발 이후 엔진을 활용한 게임개발 교육에 소요되는 노력을 줄이고, 엔진도입 시점에서 유사게임을 빠르게 개발하려는데 목적이 있으며 또한 개발자와 기획자간의 의사교환을 위해 개발이전에 엔진을 이용하여 게임을 정의해 볼 수 있었다. 스크립트를 이용하여 엔진을 제어할 경우 [표 4]와 같이 생산성을 향상시키기 위한 요소로써, 본 연구에서 개발한 스크립트를 활용한 IDE는 엔진을 정확히 이해하지

못한 초급 개발자들도 쉽게 적응할 수 있는 수준의 쉬운 스크립트를 제공하여 스켈레톤 게임을 개발할 수 있다. 본 논문에서 제시한 IDE를 이용할 경우 첫 번째, 엔진용 코드 개발은 스크립트를 이용함으로써 엔진의 이해도가 낮아도 쉽게 개발할 수 있다. 두 번째, 개발자의 수준이 고급 프로그래머가 아니더라도 간단하게 개발할 수 있으므로 개발인력 공급이 원활해진다. 세 번째, 개발기간의 단축이다. 동일 장르의 프레임 워크를 빠르게 조립할 수 있으므로 설계기간이 단축된다. 마지막으로 개발환경의 통합이다. 개발환경이 통합되면서 기획자의 의도를 반영하여 개발이 가능하다. 기획자도 스크립트를 활용하여 간단하게 스켈레톤 게임을 개발할 수 있으며, 이것은 기획자가 의도한 게임 구성을 스크립트 조합으로 개발단계의 스켈레톤 게임을 구성하여 개발자와 검토하기 위한 의사교환 방법으로 이용할 수 있다. 따라서 스크립트를 이용한 게임 개발 방법은 게임 개발 라이프사이클을 빠르게 할 수 있으며 생산성을 향상시킬 수 있다.

향후 연구과제로는 엔진의 교체에 따른 스크립트의 확장이 가능해야 하며, 엔진의 구성요소를 부품화하여 개발하고자하는 게임 플렛폼 위에서 조립이 가능해야 할 것이다.

참 고 문 헌

[1] 한국전자통신연구원, "가상현실연구부, 온라인 3D 게임엔진 표준화 연구," 최종연구보고서, 2001.

[2] M. Lewis, and J. Jacobson, "Game Engines in Scientific Research," Communications of the ACM, Vol.45, 2002.

[3] S. Rabin, "AI Game Programming Wisdom," Charles Rivers Media, 2002.

[4] J. E.Laird, "Using a Computer Game to Develop Advanced AI," IEEE Computer, pp.70-75, 2001.

[5] S. Woodcock, "Game AI: the state of the Industry," Game Developer, pp.24-28. 2000.

저 자 소 개

최 한 용(Han-Yong Choi) 정회원

- 1994년 : 경희대학교 전자계산공학과(공학사)
- 1998년 : 경희대학교 전자계산공학과(공학석사)
- 2002년 : 경희대학교 전자계산공학과(공학박사)
- 2004년~현재 : 한북대학교 컴퓨터공학과 교수

<관심분야> : 컴포넌트 설계, 게임공학, 디자인패턴

이 돈 양(Don-Yang Lee) 정회원

- 1987년 : 대구대학교 통계학과(이학사)
- 1993년 : 경희대학교 전자계산공학과(공학석사)
- 2004년 : 경희대학교 전자계산공학과(공학박사)
- 1995년~2002년 : 대한상공회의소
- 2003년~현재 : 경인여대 전산정보 겸임교수

<관심분야> : EJB, 디자인패턴, XML, OOD, AOP

박 의 준(Wee-Joon Park) 정회원

- 1983년 : 경북대학교 전자공학과(공학사)
- 1987년 : 영남대학교 정보처리교육학과(교육학석사)
- 1996년 : 대구효성가톨릭대학교 전산통계학과(이학박사)
- 1992년~2003년 : 김천대학 전산정보처리과 교수
- 2004년~현재 : 한북대학교 컴퓨터공학과 교수

<관심분야> : 프로그래밍언어, 게임콘텐츠, e-비지니스, 교육콘텐츠, 응용s/w