제2장 국내외 게임기술 동향

제1절 미들웨어 기술

1. 미들웨어 기술동향

'미들웨어'란 특정 애플리케이션에 의존적이지 않고 다양한 애플리케이션에서 요구되는 기능을 제공하는 범용성이 높은 소프트웨어로, 소프트웨어의 규모가 커지고 복잡해짐에 따라 전체 개발의 생산성을 높이고 애플리케이션에서 요구하는 특정한 기능의 필요성에 부합하기 위해서 생겨난 제품으로 게임 미들웨어 역시 게임 산업이 커지면서 점점 확대되고 있는 추세이다.

(1) 게임산업의 변화와 게임 엔진

최근의 게임산업계의 큰 변화로는 그래픽 하드웨어의 발전과 PC 이외의 Xbox360, PS3 등과 같은 차세대 콘솔 게임 플랫폼의 등장, 모바일폰, 휴대용 게임기 등과 같은 휴대용게임 및 온라인게임의 성장 등을 꼽을 수 있다.

하루가 다르게 빠르게 변하는 그래픽 하드웨어의 발전으로 이전보다 더욱 향상된 퀄리티의 그래픽이 요구되고 있으며 이에 따라, 그래픽 작업의 프로덕션 파이프라인에 많은 변화를 가져오게 되었다. 이러한 변화로 작업에 필요한 개발 인원이 늘어나고 작업 시간이 증가되었으며 이것은 전체적으로 개발 비용의 증가를 초래시킨 중요한 원인 중의 하나가 되었다.

또한, Xbox360, PS3 등과 같은 차세대(Next-Generation) 콘솔의 등장으로 대응해야 할 플랫폼이 늘어남에 따라 개발이 더욱 복잡하고 어려워지게 되었는데 이것 역시 개발비를 상승시킨 중요한 원인이다. 이러한 이유로 하나의 프로덕트를 다양한 플랫폼으로 출시해서 증가한 개발 비용을 충당하고 수익은 더욱 증대시키는 전략이 점점 일반화되고 있는 추세이다.

게임 개발에 있어 비용을 절감하고 생산성을 높이기 위한 방법으로는 게임 엔진과 같은 미들웨어를 사용하는 방법이 있다. 게임 엔진이란 재사용이 용이한 소프트웨어 라이브러리 혹은 컴포넌트의 집합과 생산성을 높이기 위한 다양한 개발툴 등을 포함하고 있는 것으로 소프트웨어 라이브러리에는 기본적인 렌더링, 애니메이션 등을 위한 라이브러리가 포함되고 개발 툴에는 아티스트를 위한 DCC(Digital Contents

Creation) 툴의 플러그인 등을 포함하는 것이 일반적이다. 게임 엔진을 사용하는 경우 게임 개발과 관련된 다양한 기능들과 범용적인 프레임 워크를 제공하므로 개발 기간을 단축시키고 비용을 절감할 수 있는 장점이 있다

이렇게 미들웨어 솔루션은 개발자들이 게임 타이틀을 이전보다 저렴한 비용에 보다 빨리 개발할 수 있도록 하며, 개발자들은 각 게임에 필요한 기능을 새롭게 개발하는 대신 미들웨어 솔루션으로 해결함으로써 차별화된 게임 개발에 더욱 집중할 수 있도록 한다. 그러나, 개발하고자 하는 게임에 적합하지 않은 미들웨어 솔루션을 사용하는 경우, 오히려 개발을 지연시켜 비용만 증가시킬 수도 있으므로 주의를 요한다.

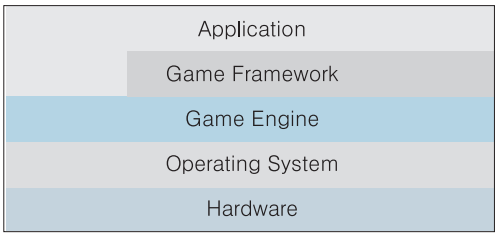
여기서는 게임 소프트웨어의 구조적인 측면에서의 게임 엔진의 모습을 설명하고 이에 따라 필요한 미들웨어를 선택할 때 적합한 미들웨어를 선택할 수 있도록 고려해야 할 사항들을 이야기하고자 한다. 그리고 개발에 주로 사용되는 미들웨어들을 카테고리별로 구분하고 이들 중 특징적인 제품에 대해서는 따로 설명해서 참고할 수 있도록 한다.

앞서 언급한 바와 같이 게임 엔진은 다양한 소프트웨어 라이브러리와 개발 툴을 제공하는 것이 특징인데, 여기에서 미들웨어는 게임 엔진뿐만 아니라, 특정 기능을 제공하는 특화된 제품들까지 모두 포함하는 전체적 의미로 통용된다(게임 엔진은 다양한 솔루션을 제공하는 통합된 개발 환경을 제공하는 미들웨어 솔루션으로 따로 분리해서 설명한다).

(2) 게임 엔진 아키텍처와 미들웨어

최근의 추세는 게임 엔진과 게임 프레임 워크를 분리해서 게임 엔진 레이어에서는 저수준(low-level)의 코어 부분에 해당하는 부분들을 처리하고 게임 개발 시 주로 필요로 하는 부분들은 고수준(high-level)의 층으로 따로 분리하여 개발하는 것이 특징이다.

〈그림 5-2-1-01〉 게임 엔진과 게임 프레임워크 레이어



이렇게 크게 저수준과 고수준으로 구분하는 이유는 첫째, 게임 개발시 저수준과 관련한 부분은 자주 변하지 않으므로 여러 가지 다양한 애플리케이션에 유연하게 사용하는 것이 가능하며 일반적으로 저수준의 API들은 고수준에 비해 진입장벽이 높은데, 개발시 애플리케이션 프로그래머는 대부분 고수준의 API만을 가지고 작업하므로 진입장벽이 높은 저수준의 API들을 직접 알 필요없이 고수준의 API를 따로 제공하면 더욱 빠르게 작업할 수 있어 생산성을 높일 수 있다.

일반적으로 미들웨어의 경우, 위의 그림에서 게임 엔진에 해당하는 부분들이 주종을 이루고 있다. 그 이유는 미들웨어라는 것이 범용적인 특징을 가진 제품이므로 다양한 게임 개발에 대응하기 위해서는 상대적으로 변경 횟수와 주기가 낮은 게임 엔진 부분에 해당하는 쪽이 많기

때문이다. 그러므로 게임 개발시 필요한 기능과 요구 사항을 잘 정리한 다음 이에 따라 미들웨어가 필요한지를 결정하고 필요한 경우에는 비슷한 종류의 미들웨어 중에서 개발하는 게임에 가장 적합한 미들웨어가 무엇인지를 결정해야 한다. 경우에 따라서는 기능별로 하나 이상의 미들웨어를 사용해야 하는 경우도 있을 수 있는데 이 때에는 미들웨어별로 작동하는 데 문제가 없는지도 꼼꼼히 살펴 보아야 한다.

또한, 경우에 따라서는 개발 도중 미들웨어를 게임에 맞게 일부분 커스터마이징해야 할 필요도 있을 수 있다. 이 때 이러한 변경 사항은 가급적이면 개발 초기 단계에 파악해서 우선적으로 개발되도록 하는 것이 좋다. 일반적인 개발 프로세스를 보면 게임 엔진을 먼저 개발한다. 왜냐하면 나중에 이 부분을 변경하는 경우 게임 프레임워크 부분이나 전체 애플리케이션 부분의 변경을 초래하여 큰 문제를 일으킬 수 있기 때문이다. 미들웨어 역시 일반적으로 게임 엔진부에 속하는 제품군이 많기 때문에 수정 및 변경해야 할 기능들이 있는지 먼저 검토해서 우선적으로 작업하는 것이 좋다. 이처럼 전체 개발에 있어 단계별로 우선 순위가 있듯이 미들웨어의 도입 역시 개발 단계의 우선 순위를 고려해서 작업해야 한다.

하드웨어의 발달로 게임의 규모가 커짐에 따라 개발 비용이 상승, 같은 게임을 다양한 플랫폼으로 개발해서 판매하는 것이 점점 일반화되고 있는 추세이다. 멀티 플랫폼의 경우, 개발 초기에 미리 결정하지 않으면 개발이 힘든 부분으로 다양한 플랫폼에서의 개발은 개발 기간이 많이 소요되는데다 기술적으로도 어려움이 있다. 그러므로 이러한 부분은 이미 개발한 플랫폼에서 검증된 미들웨어를 사용하게 되면 쉽게 해결할 수 있다.

2. 미들웨어 제품군

여기서는 게임산업 전반에 걸친 다양한 미들웨어 제품군을 특징적인 카테고리별로 구분하고 이에 대해서 소개한다. 그리고 카테고리별로 분류된 미들웨어 중에서 많이 사용되는 미들웨어나 특기할만한 미들웨어에 대해서는 따로 부연 설명을 추가해서 해당 카테고리의 미들웨어군에 대한 이해 및 선택에 도움이 되도록 했다

(1) 게임 엔진

최근의 게임 엔진의 특징으로는 Next-Gen 게임 개발에 적합한 툴셋(Toolset)의 지원과 함께 다양한 플랫폼을 지원하는 것이 특징이다.

최근 게임 엔진의 새로운 추세 중 하나는 온라인게임 시장의 성장으로 MMOG를 위한 통합 솔루션을 지원하는 게임 엔진이 등장했다는 점이다. 이제 WOW(World Of Warcraft)의 성공으로 MMOG 시장은 더 이상 한국 시장만의 특징이 아니며, 많은 개발사들이 MMOG 개발에 관심을 가지고 있는데 이러한 추세에 맞추어서 클라이언트-서버 통합 솔루션을 제공하는 제품군들이 속속 등장하고 있다.

〈표 5-2-1-01〉 게임 엔진

제 품	웹사이트	특 징
Unreal Engine	http://www.epicgames.com/	Epic Games
GameBryo	http://www.emergent.net/	Emergent Game Technologies
CryEngine	http://www.crytek.com/	Crytek
Offset Engine	http://www.projectoffset.com/	Offset Software
Torque Engine	http://www.garagegames.com/	Garage Games
C4 Engine	http://www.terathon.com/c4engine/	$400에 라이선스 가능
Nebula Device	http://www.radonlabs.de/ http://www.touchdownentertainment.com/	RadonLabs, 오픈 소스 Touchdown Entertainment
Source Engine	http://www.valvesoftware.com/	Valve Software HalfLife2에 사용
Trinigy	http://www.trinigy.de/	-
Virtools	http://www.virtools.com/	Authoring Tool
Worldweaver	http://www.worldweaver.com/	-
Unigine	http://www.unigine.com/	-
Power Reneder	http://www.powerrender.com/	-
Vicious Cycle Software	http://www.viciouscycleinc.com/	-
Agency9	http://www.agency9.com/	-

〈표 5-2-1-02〉 MMOG 엔진

제 품	웹사이트	특 징
BigWorldTech	http://www.bigworldtech.com/	-
Hero Engine	http://www.play.net/playdotnet/platform/	-
Kaneva	http://www.kaneva.com/	-
Multiverse	http://www.multiverse.net/	-
MMO Enhancement Kit for TGE	http://www.mydreamrpg.com/	-

〈표 5-2-1-03〉 렌더링(Rendering)

상품명	웹사이트	특 징
Geomerics	http://www.geomerics.com	lighting
RAD Game Tools - pixomatic	http://www.radgametools.com/	-
Wizaid	http://www.wizaid.com/	visibility
Turtle	http://www.illuminatelabs.com/	Renderer, Light Baking

가. 그래픽 엔진

그 외 다양한 하드웨어 및 플랫폼이 출현하면서 렌더링 모듈의 개발이 복잡해짐에 따라 렌더링만을 위한 그래픽 미들웨어 제품군들도 있다.

나. 애니메이션 엔진

렌더링 엔진과 함께 art-asset을 위한 DCC 툴과 관련한 미들웨어도 다수 존재한다. 이러한 미들웨어들의 특징은 그래픽 디자이너들이 사용하는 DCC 툴에서 생산성을 높일 수 있는 다양한 플러그인을 제공한다는 것이다.

셰이더가 렌더링 기술의 핵심으로 자리 잡으면서 아트 프로덕션 파이프라인에서 그래픽 아티스트들이 셰이더를 쉽게 사용할 수 있는 툴들이 등장하고 있는데 그 중 하나로 범용적인 DCC 툴인 3D Max용 툴의 플러그인으로 설치해서 사용할 수 있는 ShaderFX(http://www.lumonix.net/shaderfx.html)가 있다.

이 외에도 DCC 툴의 모델과 애니메이션을 손쉽게 게임 내에 임포트해서 사용할 수 있는 미들웨어로 Granny3D(http://www.radgametools.com/#Granny)가 있다.

(2) 물리(Physics) 엔진

게임 제작과 관련한 주요 기술 중에서 3D 기술이나 네트워크 기술 못지 않게 그 중요성이 부각되는 기술로 게임 물리 기술을 꼽을 수 있다.

NOVODEX는 AGEIA의 물리 엔진으로 에픽 게임즈가 언리얼 엔진에 탑재하면서 주목받고 있는데 최근에는 PC 플랫폼에 대해서는 무료로 사용할 수 있도록 라이선스 정책을 변경하였다. 또한 GameBryo 엔진에서도 2.2 버전부터 AGEIA의 물리 엔진을 탑재하고 있다.

또 다른 게임 물리 엔진인 ODE는 오픈소스 물리 엔진으로 이미 다수의 상용 프로젝트에 사용되어 상용 물리 엔진의 대안으로 검증되어 앞으로도 많은 프로젝트에서 사용될 것으로 예상된다.

〈표 5-2-1-04〉 애니메이션(Animation)

제 품	웹사이트	특 징
Annosoft	http://www.annosoft.com/	lipsync
EMotion FX	http://www.emotionfx.com/	캐릭터 애니메이션 미들웨어, DCC 툴의 plug-in 및 게임 엔진을 위한 SDK 제공
The Game Creators	http://www.thegamecreators.com/	-
Havok - Animation	http://www.havok.com/	-
Lifemode Interactive	http://www.lifemi.com/	Facial Animation
NaturalMotion	http://www.naturalmotion.com/	-
OC3 Entertainment	http://www.oc3ent.com/	Facial Animation
RAD Game Tools - Granny	http://www.radgametools.com/	DCC 툴의 plug-in 및 게임 엔진을 위한 SDK 제공

〈표 5-2-1-05〉 물리 엔진

제 품	웹사이트	특 징
Ageia	http://www.ageia.com/	PC 플랫폼은 무료
Havok	http://www.havok.com/	-
ODE	http://www.ode.org/	오픈 소스
Bullet	http://www.continuousphysics.com/Bullet/	상업적 사용도 무료
DTECTA	http://www.dtecta.com/	-
Lightsprint-Collider	http://www.lightsprint.com/	-
Pixelux	http://www.pixeluxentertainment.com/	-
Phyar Lab	http://www.spehome.com/	-

〈표 5-2-1-06〉 네트워크 엔진

제 품	웹사이트	특 징
Cybernet Systems	http://www.openskies.net/	대용량 서버 기술
DemonWare	http://www.demonware.net/	다양한 콘솔 게임의 멀티플레이에 사용
Lyra Network	http://www.lyrastudios.com/	대규모의 다중 사용자 네트워킹 지원
Quazal Technologies	http://www.quazal.com/	게임로비, In-Game 네트워킹 등으로 구분된 제품군을 제공. UE3 엔진에 포함
Rakkarsoft	http://www.rakkarsoft.com/	UDP 게임 네트워킹 엔진
ReplicaNet	http://www.replicanet.com/	-
Tincat	http://www.tincat.de/	-
ZeroC	http://www.zeroc.com/	GPL 라이선스
OpenTNL	http://www.opentnl.org/	UDP 게임 네트워킹 엔진 Torque 엔진에서 네트워크 엔진만 따로 분리한 엔진

〈표 5-2-1-07〉 인공 지능 엔진

제 품	웹사이트	특 징
A.I.implant	http://www.ai-implant.com/	DCC 툴의 플러그인으로 모델링 단계에서 적용 가능
DirectA	http://www.directia.com/	-
RWAI	http://www.renderware.com/	-
SimBionic	http://www.simbionic.com/	-
Havok-Behavior	http://www.havok.com/	-
PathEngine	http://www.pathengine.com/	DCC 툴 플러그인 및 SDK 제공
SpirOps	http://www.spirops.com/	Sprinter Cell-Double Agent 에 사용
Trusoft-Artificial Contender	http://www.trusoft.com/	-

(3) 네트워크(Network) 엔진

네트워크 엔진 관련 미들웨어는 기존의 PC게임을 위한 네트워킹 솔루션뿐만 아니라 콘솔 게임으로 확대되고 있다. 따라서 PC뿐만 아니라 콘솔 게임의 네트워킹 솔루션을 함께 제공하는 것이 특징이다. 국내에서는 온라인게임이 대부분을 차지하는 만큼 주의해서 살펴볼 만한 제품군이다.

(4) 인공지능(A.I.) 엔진

게임의 인공지능은 매력적인 게임을 제작하는 데 필수 불가결한 요소일뿐만 아니라 플레이어로 하여금 게임을 다시 플레이하게끔 만들고 게임 내의 균형과 플레이어를 제어하고 피드백을 주는 일 그리고 이것들을 통해서 최종적으로는 플레이어에게 재미를 주는 가장 중요한 요소이다. 때문에 게임이 발전하면서 게임 내의 인공지능에 대한 요구 사항 역시 더욱 정교해지고 최적화된 방법들이 요구되고 있으며, 이러한 요구 사항을 충족시키는 다양한 미들웨어들이 등장하고 있다

(5) 사운드(Sound) 엔진

최근의 연구 결과를 보면 영화나 게임과 같은 콘텐츠에서 사용자의 몰입감을 극대화시키는 가장 큰 요소가 사운드라는 보고가 있다.

사운드는 단순한 게임 배경음악 및 효과음의 출력이 아니다. 배경음악은 게임의 스토리 진행이나 이벤트에 맞게 현재 진행되는 게임 플레이의 느낌에 가장 적합한 음악이 플레이되어야 한다.

효과음은 게임 내의 환경과 상호 연동을 통해서 게임의 현실감을 높여서 사용자가 더욱 몰입할 수 있도록 해야 한다. 예를 들어 물위를 걸을 때의 발자국 소리와 땅 위를 걸을 때의 발자국 소리는 다를 것이다. 또한 칼이 돌벽에 부딪힌 경우와 나무에 부딪힌 경우의 소리도 다르다.

이렇게 배경음은 게임 내의 플레이에 따라

〈표 5-2-1-08〉 사운드 엔진

회 사	웹사이트	특 징
AM3D	http://www.am3d.com/	-
Audiokinetic	http://www.audiokinetic.com/	-
CRI Middleware	http://www.cri-mw.com/	-
Dynatmos	http://www.dynatmos.com/	-
Firelight Technologies	http://www.fmod.org/	FMOD
InterAmus Music Systems	http://www.interamus.com/	-
OpenAL	http://www.openal.org/	Free
Princeton Digital	http://www.princetondigital.com/	-
RAD Game Tools - Miles	http://www.radgametools.com/	-
Sensaura	http://www.sensaura.co.uk/	-
Un4seen Developments	http://www.un4seen.com/	-

환경적인 차이를 고려해서 출력되어야 사용자의 몰입감이 더욱 커질 것이다. 그런데 이러한 게임 환경은 다른 게임 내 요소와도 밀접한 관계가 있다. 마른 땅과 얕은 물 속은 게임 내 플레이어가 반응하는 물리적 저항이 다를 것이며 이런 차이는 물리 엔진과도 관계가 있는 부분이다. 그러므로 물리적 특징이 중요한 게임이라면 사운드 미들웨어 역시 물리 엔진과 같이 다른 미들웨어와 쉽게 연동해서 개발할 수 있는지의 여부를 파악하는 것도 사운드 미들웨어 선택 시 고려되어야 할 중요한 요소이다(※돌비 사운드 등은 일반적으로 별도의 라이선스를 필요로 한다).

사운드 모듈과 관련된 미들웨어로는 Firelight Technologies의 FMOD와 RAD Game Tools의 Miles 미들웨어가 널리 사용되고 있다.

(6) 보안(Security) 솔루션

온라인게임 산업이 성장하면서 게임 아이템 시장이 급성장함에 따라 이제 게임 보안 분야가 하나의 비즈니스로 자리잡게 되었다.

게임 해킹은 게임의 밸런스를 파괴하고 게임 서버의 부하를 늘려 유지 비용을 증가시키며, 온라인게임의 오픈 후 상용화 시기를 지연시키는 문제를 발생시키므로 온라인게임이라면 반드시 미리 대응책을 강구해야 하는 부분이다.

서비스 중 게임의 패킷이 노출되는 경우, 프리서버 등이 등장해서 매출에 악영향을 미치는 경우 등의 여러 가지 해킹이 있으며, 이러한 해킹이 심한 경우에는 제대로 서비스를 하지 못하고 서비스를 종료하는 경우도 발생하게 된다.

해킹을 방지하기 위해서는 적절한 보안 조치가 필요한데 문제는 보안과 관련한 작업은 게임 개발과 매우 상이해서 개발사에서 게임 개발과 보안이라는 두 가지를 모두 해결하기는 어렵다는 것이다. 다행히 보안과 관련해서는 기존의 바이러스 백신 회사 등에서 개발한 다양한 해킹 방지 솔루션들이 있다.

온라인게임 산업이 성장함에 따라 해킹 방법도 더욱 다양하고 교묘해지고 있다. 상용화 단계에 들어 보안을 고려하는 경우, 이미 오픈 베타 때 노출이 되어 그 대응이 이전보다 더욱 어려운 경우가 많다. 때문에 게임 개발 초기부터 보안을 고려해서 개발하는 것이 중요하다.

〈표 5-2-1-09〉 보안 솔루션

제 품	웹사이트	특 징
HackShield	http://www.ahnlab.com/	PC
nProtect	http://www.nprotect.com/	PC
Xecure	http://www.softforum.co.kr/	PC
ECD Systems	http://www.ecdsystems.com/	PC
SecurePlay	http://www.secureplay.com/	ps2 linux
StarForce Technologies	http://www.star-force.com/	PC

(7) 모바일

모바일 시장이 급증한 데다 모바일용 그래픽 칩의 발전으로 모바일게임 개발과 관련한 미들웨어에 대한 요구도 증가 추세에 있다.

(8) 유저 인터페이스

온라인게임의 특징으로 계속되는 콘텐츠의 추가와 유저들의 커뮤니티가 발달하게 되는데 이러한 특징을 게임 내에 구현하다 보면 클라이언트 단에서는 사용자 인터페이스 개발에 많은 시간이 소요된다. 이러한 이유로 원하는 사용자 인터페이스를 빠르게 디자인해서 게임 내에 구현할 수 있는 미들웨어를 사용하면 생산성을 높이고 MMOG의 반복되는 업데이트 개발 시간을 단축시키는 등의 효과를 기대할 수 있다.

(9) 기타

MMP(Massively Multi-Player) 게임은 일반적으로 장기간에 걸쳐 서비스되므로 게임의 밸런싱 작업이 다른 플랫폼의 게임에 비해 훨씬 복잡하며 까다롭다.

또한, MMP 게임의 경우 대부분 사용자간 커

〈표 5-2-1-10〉 모바일게임 관련 미들웨어

제 품	웹사이트	특 징
Exit Games	http://www.exitgames.com/	j2me brew
Ex Machina	http://www.exmachina.nl/	j2me
HeroCraft HiTech	http://www.hitech.herocraft.com/	ppc symb palm
Ideaworks3D	http://www3.ideaworks3d.com/	ppc linux brew symb
In-Fusio	http://www.in-fusio.com/	j2me
J2X Technologies	http://www.j2x.ca/	j2me
Pixelgene	http://www.pixelgene.com/	java brew symb
Qualcomm	http://www.qualcomm.com/	brew
Richmotion	http://www.richmotion.com/	java
Sony Ericsson	http://www.sonyericsson.com/	j2me
Terraplay System	http://www.terraplay.com/	pc ppc xbox ps2 gc j2me symb
Tira Wireless	http://www.tirawireless.com/	j2me brew

〈표 5-2-1-11〉 유저 인터페이스

제 품	웹사이트	특 징
Scaleform	http://www.scaleform.com/	다양한 플랫폼 지원, 다양한 상업 게임에서 사용 Crisys 등
Wintsch Labs	http://www.wintschlabs.com/	.NET 기반
Omega Game	http://www.omegame.com/	Menus Master

〈표 5-2-1-12〉 밸런싱(Balancing)

제 품	웹사이트	특 징
+7 Systems	http://www.plus7systems.com	밸런싱 솔루션

〈표 5-2-1-13〉 비디오(Video)

제 품	웹사이트	특 징
RAD Game Tools - Bink	http://www.radgametools.com/	비디오 코덱

뮤니티가 형성되기 때문에 밸런스에 허점이 있는 경우 커뮤니티를 통해서 급속히 퍼지게 된다는 문제점이 있다. 이러한 특징으로 밸런스 문제에 대해서 신속히 대응하지 못할 경우, 자칫 잘못하면 게임 서비스에 영향을 미칠 수 있을 정도로 심각한 문제로 커질 수 있다. 이와 같이, MMP 게임에서의 밸런스는 게임의 재미를 좌우하는 요소일 뿐만 아니라, 게임 서비스 자체와도 밀접한 관계가 있는 요소이므로 개발팀에서 밸런싱과 관련한 솔루션을 개발할 여력이 없는 경우 밸린싱과 관련한 미들웨어의 사용도 고려해 보아야 한다.

게임 시작 혹은 게임 내에서 미리 렌더링한 동영상을 플레이하는 경우가 많은데 이 경우 동영상 플레이에 많이 사용되는 게임을 위한 비디오 코덱을 위한 솔루션으로 RAD Game Tools의 BINK가 있다.

3. 결론

현재 하드웨어의 발전, 플랫폼의 증가, 사용자 요구 사항의 증가 등으로 게임 개발 프로세스가 세분화되고 정교해지고 있는 추세이다. 때문에 하나의 회사 혹은 개발팀이 개발하고자 하는 게임의 모든 요구사항에 대응하기 힘든 경우가 많다. 따라서 아웃 소싱을 통한 분업화가 이전보다 더욱 절실한 때이다. 그러므로 미들웨어 시장은 이전보다 더욱 커질 것으로 예상된다.

성공적인 게임 개발을 위해서는 개발 초기의 프리 프로덕션 단계에서 개발할 게임의 특징과 예상되는 개발 리스크를 파악한 후 게임의 특징은 최대화하고 리스크는 최소화할 수 있는 방법 중 하나인 미들웨어의 사용에 대해서 충분히 검토해 볼 필요성이 커질 것으로 기대된다.

제 2 절 레벨디자인 기술

1. 멀티플레이용 레벨 디자인 시에 고려해야 할 요소

과거에 비해 게임을 구동하기 위한 하드웨어 자원은 상당히 비약적인 향상을 이루었다. 과거에는 상상도 할 수 없었던 비디오 화면이나 입체 사운드, 그리고 사실적인 물리 연출까지 최근에 나오는 게임들을 보면 10여년 전에는 상상도 할 수 없었던 것들이 현실화되고 있다는 것을 실감할 수 있다.

그렇지만 사실 그렇게 크게 나아지지 않은 기술 분야도 있다. 특히, 네트워크 기술의 경우 그 발전의 폭이 그다지 크지 않다고 할 수 있다. 네트워크 기술은 엄연한 물리적인 제약(실제 신호의 전송거리에 의한 지연 등)을 가지고 있으며, 한번에 전송할 수 있는 데이터의 양을 의미하는 대역폭도 아직은 대부분의 게임에서 충분하지 않다. 그리고 트래픽이나 기타 회선 장애로 인한 서비스의 품질 역시 항상 고려해야 할 요소로 자리잡고 있다. 멀티플레이 온라인게임에서 이런 제한은 게임 디자인 자체에 상당한 영향을 주는 요소이며, 특히 멀티플레이용 레벨을 디자인할 시에 상당한 제약으로 작용한다.

네트워크뿐만 아니라 렌더링이나 인공 지능 등도 멀티플레이용 레벨의 경우 싱글플레이용 레벨을 만들 때와는 다른 형태의 제약이 따른다. 먼저 멀티플레이 게임 개발 시 추가적으로 발생하는 기술적 고려 요소들을 살펴보고, 추가적인 기술적 제약 사항을 파악하여 이런 제약을 효과적으로 해결하는 멀티플레이용 레벨 디자인에 대해서 논의를 전개하고자 한다.

(1) 멀티플레이 게임 개발 시 우선적으로 고려해야 할 기술적 요소는 네트워크

현대의 게임에서 싱글플레이와 멀티플레이를 구분짓는 가장 큰 요소는 원격지간의 머신을 이용해서 플레이를 하느냐, 하지 않느냐 일 것이다. 따라서 싱글플레이 게임에서는 전혀 고려하지 않아도 될 기술적 요소이지만, 멀티플레이 게임에서는 가장 큰 기술적 고려 요소가 네트워크이다.

물리적인 위치에 따른 네트워크 전송지연은 물리적 한계에 의해서 일정 이상 줄일 수 없다고 해도, 과거에 비해 확실히 네트워크의 대역폭은 상당히 향상되었다. 이 말은 데이터가 전달되는 속도는 어쩔 수 없지만, 한번에 많은 양의 데이터를 전달할 수는 있다는 뜻이다. 그러나, 게임에서는 실제로 많은 양의 데이터를 전송할 수 있다고는 해도 항상 실시간으로 처리해야 하는 게임 애플리케이션의 특성상 그 데이터량을 최소화하는 것이 중요하다. 그리고, 현재의 네트워크 환경은 충분한 양의 대역폭이 안정적으로 확보되지는 않는다. 현대의 멀티플레이 게임에서는 상호간에 동기화하여 전송해야 할 데이터가 계속 늘고 있다.

액션이나 슈팅이 주류인 멀티플레이 게임에서 네트워크 전송량의 가장 큰 비율을 차지하는 것은 플레이어 정보 데이터일 것이다. 특히, 플레이어의 위치는 항상 갱신이 이루어지는 아주 중요한 데이터이다. 위치뿐만이 아니라 대부분

의 플레이어가 취하는 액션들도 매우 중요하다. 특히, 현대의 멀티플레이 게임에서는 새로운 게임 요소의 추가를 위해 플레이어가 취하는 액션이 점점 늘고 있으며, 앞으로의 게임들 역시 액션이 계속 늘어날 것이라고 예상할 수 있다. 액션이 추가될수록 대역폭 소모도 그에 비례해 증가하게 될 것이다.

특수 효과용 데이터 전송에 의해 상당한 네트워크 대역폭이 소요되는 요소다. 기본적으로 플레이어의 행동에 의해서 발생하는 특수 효과로는 일인칭 슈팅 게임의 경우로 총알이나 총알 착탄 지점의 파티클 효과들이 대표적이다. 이들 역시 멀티플레이 게임에서는 빠르게 전송해 주어야 하는 정보들이다. 전송 지연에 아주 민감한 정보라는 의미이다. 특수 효과 발생의 플레이어들 간의 동기화는 아주 중요한 문제이다. 발생 시간이 맞지 않을 경우 속임수로 의심 받을 수 있기 때문이다. 게다가 최근의 게임들은 이런 기본적인 특수 효과 외에도 연출이나 물리 엔진 등에 의해서 더욱 많은 양의 특수 효과를 사용한다는 점이다. 현대의 멀티플레이 게임에서 특수 효과의 전체 데이터 전송량은 크게 증가하고 있으며, 앞으로 더욱 늘어나게 될 것이라고 전망한다.

결론적으로, 네트워크 기술은 대역폭의 향상이 이루어지고 있으며, 향상 속도에 맞춰서 멀티플레이용 게임 디자인이 요구하는 데이터 전송량도 추가적으로 늘어가고 있다.

(2) 싱글플레이에 비해 높게 발생하는 비디오 렌더링 부하

싱글플레이 게임의 비디오 렌더링 시에 걸리는 작업량은 게임 디자이너가 직접적으로 제어가 가능하다. 예를 들자면, 한 화면에 등장하는 인공지능 플레이어의 개수를 직접 조절한다든가, 한 화면에서 연출되는 특수 효과의 발생 횟수를 직접 입력하는 것 등이 있다. 그러나 멀티플레이 게임에서는 게임 디자이너의 직접적인 개입이 힘들다. 따라서, 예기치 못한 부하에 의한 비디오 렌더링 시의 심각한 렉이 발생할 수 있다. 특히 이런 렌더링 렉 현상은 싱글플레이보다 멀티플레이 시에 더욱 심각한 게임성 저하를 가져올 수 있다. 싱글플레이 시는 전체 게임 로직을 렌더링 렉이 풀릴 때까지 지연시킬 수 있지만, 멀티플레이 시는 전체 게임을 멈추는 것이 거의 불가능하므로 플레이어가 렉 상황에서 어이없는 게임 결과를 겪을 수 있기 때문이다.

비디오 렌더링 시 가장 많은 부하를 일으키는 객체는 플레이어다. 플레이어라는 객체는 게임 전체에서 가장 무거운 객체이다. 특히 렌더링 시스템에도 상당한 부하를 준다. 대부분의 게임에서 플레이어는 스키닝되는 스켈레탈 메시이기 때문이다. 스키닝되는 스켈레탈 메시는 특히 요즈음 렌더링 시스템의 추세인 퍼픽셀 라이트닝을 처리하는 부분에서도 상당한 부하를 일으키며, 전통적으로 가장 부하가 많이 걸리는 렌더링 개체이다. 수십만 개의 다각형으로 이루어진 레벨이나 배경 그래픽보다 스키닝되는 캐릭터 하나가 더 큰 부하를 일으킬 수 있다. 멀티플레이 게임에서는 이런 캐릭터가 기본적으로 게임에 참가하는 플레이어의 수 이상으로 늘어나게 된다. 특히 하나의 화면에 다수의 플레이어가 등장하는 경우 상당한 부하를 일으키게 된

다. 싱글플레이 게임에서는 직접적으로 한 화면에 등장하는 인공지능 플레이어의 개수를 조절할 수 있지만 멀티플레이용 게임에서는 한 화면에 등장하는 플레이어 수 자체를 직접 조절하는 것은 불가능하다. 일반적으로 싱글플레이 게임에 비해서 멀티플레이용 게임에 등장하는 캐릭터는 상대적으로 적은 양의 리소스를 사용하도록 그래픽디자인한다. 하지만 이 방법은 지나친 그래픽 품질의 저하를 가져올 수 있다.

또한, 파티클이나 그 외 여러 가지 특수 효과 렌더링도 상당한 부하를 일으킨다. 파티클 효과 같은 특수 효과는 스켈레탈 메시만큼이나 비디오 렌더링 시 상당한 부하를 주는 요소다. 한 화면에서 다수의 특수 효과가 발생할 경우, 비디오 렌더링 부하에 의한 렉을 초래할 수 있다. 싱글플레이 게임에서는 이런 효과 발생 이벤트를 직접 제어할 수 있지만, 멀티플레이용 게임에서는 직접적인 제약이 힘들다. 현대의 게임에서는 자연스러운 효과를 연출하기 위해 사용되는 파티클이나 기타 효과들이 점점 늘어나는 추세이다.

렌더링 기술 분야는 다른 어떤 게임 기술 분야에 비해서 비약적인 향상을 이루었지만, 멀티플레이용 게임 개발 시에는 직접적인 비디오 렌더링 작업량의 조절이 어려우므로 잦은 비디오 렌더링 렉을 유발할 수 있다.

(3) 싱글플레이용에 비해서 간소해져야 할 인공지능

싱글플레이에 등장하는 인공지능 플레이어의 경우에는 게임 내에서의 중요도가 매우 높다. 따라서, 인공지능 시스템에 상당한 양의 프로세서 파워를 할당해야 한다. 그러나, 멀티플레이 게임에서는 추가적으로 네트워크 처리를 위한 프로세서 파워가 필요하므로 상대적으로 인공지능에 할당할 머신 파워가 부족해진다.

싱글플레이 게임에서는 인공지능이 실수하는 것, 길 찾기에 실패해서 인공지능 플레이어가 한자리에 머무르는 현상 등에 대해서 어느 정도는 허용할 수 있다. 그러나, 멀티플레이용 게임의 인공지능 플레이어는 다수의 플레이어가 인공 지능 플레이어를 죽이거나 또는 아군으로서 행동할 때 인공 지능에 의한 실수가 발생하면, 해당 플레이어가 느끼는 좌절감은 싱글플레이보다 훨씬 심하다. 멀티플레이 게임에서 인공 지능 플레이어를 참가시킬 경우 게임 플레이에 미치는 영향은 싱글플레이의 경우보다 훨씬 크다.

인공지능 시스템에 할당한 프로세서 파워는 상대적으로 적은데, 인공 지능 시스템의 결과는 더 좋아야 한다. 사실 두 마리의 토끼를 잡으라는 말과 같으며, 현실적으로 불가능하다. 따라서, 전체적으로 인공지능 시스템에 입력되는 값은 간단해야 하며, 결과를 찾기 쉽도록 게임이 디자인되어야 할 필요가 있다.

(4) 멀티플레이 게임에서의 물리 시스템 사용의 어려움

물리 시스템의 경우 멀티플레이 게임에서의 적극적인 사용은 아직 고려해야 할 사항이다. 기본적으로 물리 시스템이 세세한 수준의 작업이 불가능하고 상당량의 프로세서 파워를 차지하기 때문에 한번에 다수의 이벤트가 발생할 가능성이 높은 멀티플레이 게임에서는 사용하기

가 상당히 까다롭다. 또한 물리 시스템의 결과값이 클라이언트마다 다를 경우 오차를 보정하기 위해서는 상당량의 네트워크 대역폭을 필요로 하기 때문에 더욱 힘들다. 거기다가 물리 시스템의 결과에 따라서 파티클이 생성된다든가 하는 특수 효과가 필요해짐으로써 추가적인 네트워크 사용이 생긴다. 따라서, 현재는 부득이하게 사용하는 경우라고 해도 실제적인 오차 보정을 할 필요가 없는 상황에서만 선택적으로 사용하고 있다.

2. 레벨 디자인을 통한 기술적 제약 사항의 완화

앞서 네트워크, 비디오 렌더링, 인공지능, 물리 시스템에서 멀티플레이 게임 개발 시에 발생하는 제약 사항들에 대해서 간략하게 살펴보았다. 싱글플레이 게임에 비해서 멀티플레이 게임 개발 시 추가적인 제약이 생기는 가장 큰 이유는 플레이어라는 객체의 처리 비용 때문이다. 모든 객체 중에 제일 큰 플레이어가 멀티플레이 게임에서는 다수 존재하게 된다는 점이 문제이다.

따라서, 플레이어 객체 다수를 레벨상의 특정 포인트에 모이지 않도록 하면, 다시 말해서 레벨 상에 균등하게 플레이어가 위치하도록 한다면, 상대적으로 이런 제약을 완화할 수 있게 된다. 그러나 멀티플레이 게임에서는 강제적으로 게임 디자이너가 플레이어의 위치를 제어할 수 없다. 반면, 간접적으로 레벨 디자인 시 플레이어가 레벨 전반에 걸쳐 위치하도록 유도해 주는 것은 가능하다.

그밖에 기술적 고려사항을 충분히 반영해서 멀티플레이용 레벨 디자인을 한다면 어느 정도 추가적인 기술적 제약 사항을 완화시킬 수 있다.

(1) 플레이어의 생성위치를 다양하게 그리고 레벨 전체에 위치시킴

기본적으로 플레이어의 시작 지점이 레벨상에 넓게 분포한다면 플레이어가 한 곳에 몰릴 확률은 줄어들게 된다. 그러나, 이 방법은 레벨 디자인의 형태를 상당히 제약하게 되며, 다양한 매치 방식에 사용하기는 어렵다. 이 방식은 추후에 설명할 순환형 맵에 적합하며, 매치 방식도 팀간 대전 방식보다는 개인간 대결 방식에 적용 가능하다.

(2) 동선이 무한히 창출될 수 있도록 레벨 디자인

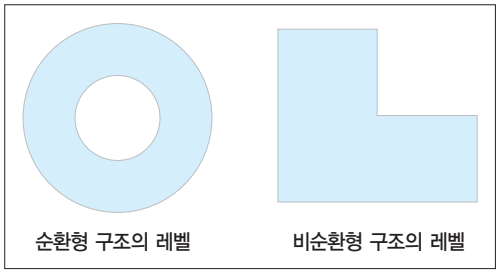
레벨상의 특정 위치로 이동하는 방법이 하나의 루트로 결정되기 보다는 다양한 루트를 통해서 해당 공간에 도달할 수 있도록 레벨을 디자인하는 것이다. 반복적이고 대칭적인 형태로 각각의 공간을 구성하면 다양한 형태로 동선이 창출될 수 있다. 동선이 많아지면 플레이어가 같은 루트를 선택할 확률이 적어지며, 따라서 레벨상의 다양한 위치로 이동할 수 있다.

(3) 레벨 내 아이템을 다양한 곳에 위치시킴

레벨상의 다양한 위치에 아이템이 존재한다면, 플레이어는 해당 아이템을 먹기 위해서 레벨의 다양한 곳으로 이동할 것이다. 이왕이면 레벨의 주 루트 상에 위치시키기 보다는 외곽에 위치시키면 레벨의 다양한 곳에 플레이어들이 위치하도록 유도할 수 있다.

(4) 비순환형보다는 순환형 레벨이 조금 더 공간 활용이 넓음

〈그림 5-2-2-01〉 순환형 구조와 비순환형 구조의 레벨



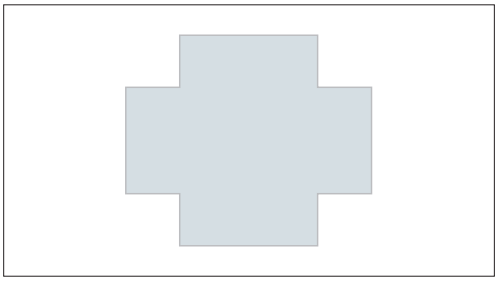
레벨의 동선이 반복적으로 연결된다면 레벨이 순환형 구조를 가지고 있다고 이야기할 수 있으며, 그렇지 않다면 비순환형 구조라고 생각할 수 있다. 게임 방식이나 매치 형식에 의해서 적절한 구조가 선택되는 것이 일반적이지만, 단순히 최적화만 생각한다면, 순환형 구조가 조금 더 전체적인 맵의 활용도가 높으며, 상대적으로 플레이어가 한 곳에 모일 확률이 적다고 할 수 있다. 비순환형 구조는 일반적으로 방향성이 존재하므로 특정 전투 유발 지역으로 플레이어들을 모으게 된다.

3. 네트워크 사용량을 조절하기 위한 레벨 디자인시 유의 사항

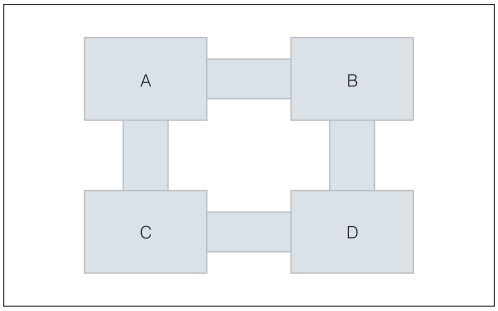
레벨에서 플레이어들을 따로 떨어뜨려 놓는 것만으로는 완전한 단절이 이루어지지 않는다. 이 말은 서로간에 연결된 공간상에서 단순히 거리를 두어 플레이어를 위치시키는 것만으로는 서로간의 데이터 교환이 불필요하다고 할 수가 없다. 멀티플레이 게임에서 데이터 교환이 필요하다는 것은 네트워크의 대역폭을 소모하며, 네트워크 지연에 의해서 시간차가 생긴다는 뜻이다. 따라서, 멀티플레이용 레벨을 만들 때는 최대한 공간이 연결되지 않은 곳에 플레이어들이 위치하도록 유도해 줄 필요가 있다. 간단히 말하면, 서로가 서로를 보지 못한다면, 서로간에 데이터를 전송할 필요는 없다. 즉, 그만큼의 네트워크 자원을 절약할 수 있게 된다.

공간과 공간간의 연결 유무나 해당 공간의 속성 등이 공간 정보라고 할 수 있으며, 각각의 공간은 특정 알고리즘 등을 통해서 하위 공간으로 분할되며, 특정한 구조의 데이터 구조를 가지게 된다. 과거에는 이런 공간 정보를 바탕으로 렌더링 순서나 컬링 등을 빠르게 결정하기 위해서 사용하였으나, 현대의 게임에서는 정보 갱신을

〈그림 5-2-2-02〉 공간 분할은 되지만 명확히 단절된 공간이 없는 레벨 구조



〈그림 5-2-2-03〉 명확히 구분되는 공간이 있는 레벨 구조



5

최소화하기 위해서 사용하는 추세이다. 즉, 서로간의 단절된 공간을 빠르게 찾고 특정 위치에 플레이어가 있다면, 그 위치상의 공간과 단절된 공간에서 발생하는 이벤트 등은 갱신받지 않는 것이다. 이것은 전체적인 처리량을 감소시켜 줄 뿐 아니라 추가적으로 멀티플레이 시 네트워크 자원을 적게 소모하게 해준다는 추가적인 이점이 있다.

즉, 공간 정보가 과거에 비해서 렌더링 시스템 등에서의 활용도는 떨어졌지만, 전체적인 처리량을 줄여준다는 측면에서 특히 네트워크 자원 소모를 줄여줄 수 있다는 점에서 더욱 중요해졌다.

따라서, 레벨 디자인 시 각각의 공간 연결을 최대한 줄여주는 것은 아주 중요하다. 즉, 하나의 넓은 방을 만들기 보다는 작은 방들이 연결된 형태로 레벨을 디자인하는 것이 상대적으로 더 쾌적한 플레이를 보장해 줄 수 있다는 것이다.

4. 렌더링 작업량을 줄이기 위한 동선 설계와 공간 설계

렌더링 시스템의 부하를 줄이기 위한 레벨 디자인도 네트워크를 위한 레벨 디자인과 크게 다르지 않다. 즉 레벨 상에서 단절된 공간이 많이 생긴다면 그만큼 렌더링 부하도 줄어들게 된다.

그러나 렌더링 시스템에서는 완전한 단절보다는 보이지 않는 것이 더욱 중요하다. 즉, 한 공간에 플레이어나 객체들이 있다고 해도 서로가 서로를 가려서 안 보이는 상황이 된다면, 현대적인 렌더링 시스템에서는 그리지 않는다. 과거의 그래픽 하드웨어는 가시성 판단이 직접 지원되지 않았지만 현대의 하드웨어는 하드웨어에서 직접적인 가시성 판단을 지원함으로써 과거와 같은 완전한 공간상의 단절이 되는 경우뿐만 아니라 같은 공간상에서의 서로간의 은폐 또는 차폐를 잘 생각해서 구성한다면 상당한 렌더링 효율을 가질 수 있다.

간단히 일인칭 슈팅 게임을 생각한다면, 레벨 디자인 시 은폐물이나 장애물들을 적절히 공간상에 배치함으로써 상당히 렌더링 부하를 줄일 수 있다.

따라서, 멀티플레이용 레벨 구성 시에는 이런 은폐 요소들을 싱글플레이용 레벨에 비해 더 추가시켜 줄 필요가 있다.

또 생각해야 하는 것이 구조화된 레벨 디자인(Modular Level Design)이다. 이 방식은 기본적인 골격에 레고 블록형태로 각각의 장식물들을 붙이는 방식이라고 생각하면 간단하다. 이 방식은 특히 렌더링 시에 같은 객체가 반복적으로 그려지게 되므로 최소한의 렌더 스테이트 변경을 통해서 그릴 수 있다. 또한, 추가적인 전송이 없으므로 그래픽 하드웨어에서 아주 빠르게 그릴 수 있게 된다. 그리고 전체적으로 리소스 사용량이 상당히 줄어들게 된다. 싱글플레이 게임에서는 이벤트에 따라 순차적으로 레벨 리소스를 로딩할 수 있지만, 멀티플레이 게임에서는 한번에 전체 맵을 메모리에 올리는 것이 훨씬 쾌적한 플레이를 즐길 수 있으므로, 구조화된 레벨 디자인 방식은 상당한 이점을 제공한다.

5. 그 외 고려해야 할 요소

지금까지 주로 레벨 디자인시 기술적인 제약 사항을 완화하기 위한 방법에 대해서 이야기 했지만, 사실 레벨 디자인에 있어서 가장 중요한 요소는 게임의 재미 요소를 얼마나 잘 표현하는가 일 것이다.

게임의 재미 요소를 유도하는 방법과 기술적인 제약 사항을 완화시키는 레벨 디자인은 서로 융화 가능한 요소도 있고, 서로 상반되는 요소도 있다. 예를 들자면, 무수히 많은 동선의 경우는 재미를 배가시킬 확률이 높지만, 상대적으로 인공지능 시스템에는 부담이 된다. 이런 식으로 무수히 많은 고려 사항이 생기게 된다.

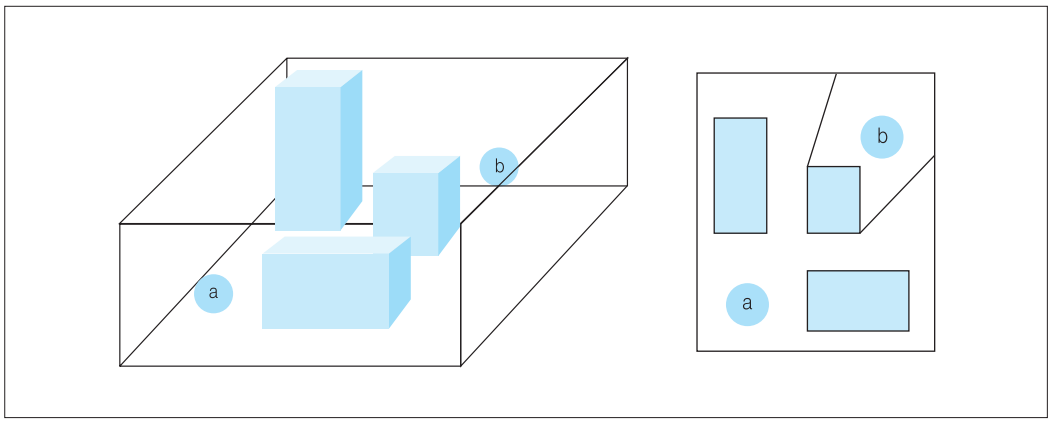
또한, 게임 방식 역시 레벨 디자인 시 상당히 고려해야 할 요소이며, 또한 플레이어의 밀집과도 큰 관련이 있다. 단순히 예를 들자면 데스 매치 방식과 팀 데스 매치 방식에 따라서 레벨 디자인은 크게 달라질 수 있으며, 또한 아무래도 팀 데스 매치 방식이 플레이어가 특정 공간상에 다른 팀 플레이어와 같이 존재할 가능성이 높아진다.

또한, 싱글플레이용 레벨과 멀티플레이용 레벨을 만드는 것은 추가적인 차이가 있다. 예를 들자면 싱글플레이용 맵의 경우는 플레이되는 횟수가 멀티플레이용 레벨에 비해서 훨씬 적다. 멀티플레이용 맵은 끊임없이 무수히 플레이될 수가 있다.

게임 디자인에서 레벨 디자인은 전체 게임 디자인의 절반 이상을 차지하는 아주 중요한 부분이며, 고려해야 할 요소가 아주 많다. 이 글에서는 기술적인 관점에서 멀티플레이 레벨 작성 시 고려해야 할 요소를 간단히 정리해 보았다. 그러나 더욱 중요한 부분은 재미 요소를 배가시키기 위한 요소들이며, 실제적인 레벨 디자인에서는 기술적인 고려사항은 최소한의 제약사항으로 남겨두고, 재미있는 레벨을 만들기 위해 노력해야 할 것이다.

5

〈그림 5-2-2-04〉 구조화된 레벨 디자인의 사례



※a의 위치에서 장애물에 의해서 b는 가려진다. 일인칭 슈팅 게임형태라면 이 경우 b를 안 그릴 수 있다.

제3절 엔진 기술

1. 멀티 코어 CPU를 위한 멀티 쓰레드 게임 엔진 아키텍처

(1) 멀티 쓰레드 게임 엔진 제작의 필요성

최근 CPU업계의 동향에 따르면, 더 이상 코어 클럭의 향상을 통해서 성능을 끌어올리는 방식은 이미 물리적인 한계에 도달했으며, 앞으로는 내부 코어를 늘려가는 방식으로 성능을 끌어올리는 방식으로의 변화가 불가피하다고 한다. 이 말을 조금 다르게 해석하면, 18개월마다 CPU의 내부 집적도가 2배씩 상승한다는 과거의 법칙이 이제는 18개월마다 내부 코어의 수가 2배로 증가하는 것으로 변화될 것이라고 해석할 수 있다. 즉, 앞으로 새로 나올 CPU들은 내부 코어가 복수개인 멀티코어 형태가 될 것임을 의미하며, 이미 현재 출시된 대부분의 신형 CPU들은 2개 이상의 코어를 내장하고 있다.

이런 현상은 PC플랫폼보다 비디오게임 콘솔 플랫폼에서 더욱 두드러진다. PC플랫폼은 현재 2개의 내부 코어를 가지는 CPU가 일반적이지만, Microsoft사의 Xbox360은 내부적으로 3개의 CPU가 각각 2개의 하드웨어 쓰레드를 가져 총 6개의 하드웨어 쓰레드가 있는 형태로 작동하며, Sony Computer Entertaiment의 PlayStation3는 하나의 코어에 8개의 SPE가 있는 형태로 구성 되어 있다. 이런 하드웨어들에서 싱글 쓰레드 형태의 애플리케이션을 실행시킬 경우 실제적으로는 하나의 코어 또는 하드웨어 쓰레드를 사용하게 됨으로써 실제적인 하드웨어 성능을 끌어내지 못하게 된다. 이런 현상은 코어의 개수나 CPU 타입의 차이 등에 의해서 PC 플랫폼보다는 콘솔 플랫폼에서 현재 더욱 두드러지게 나타나고 있다.

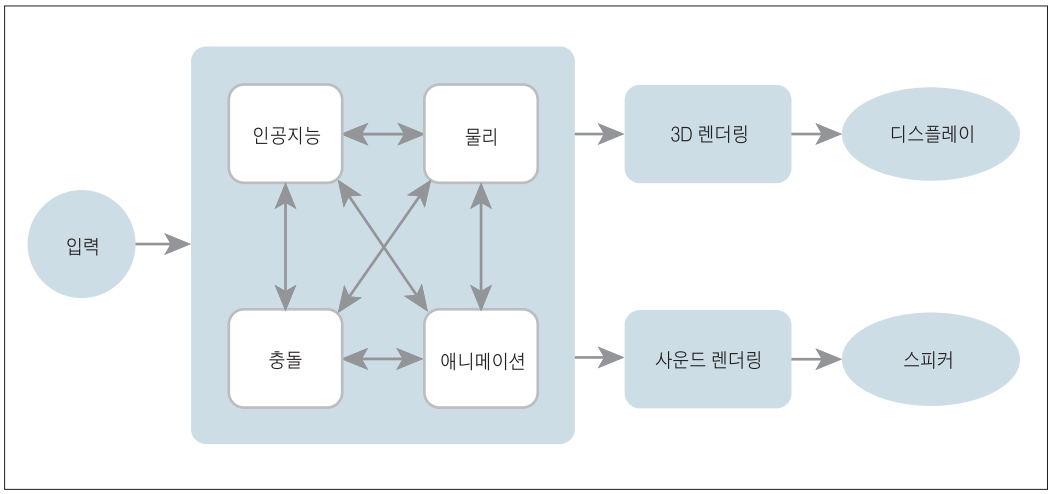
결국 앞으로는 CPU의 성능을 최대한 끌어내기 위해서는 멀티 쓰레드를 사용하는 구조로 게임엔진이 구성되어야 한다.

(2) 게임 엔진을 멀티 쓰레드 형태로 구성하는 것에 있어서의 문제점

게임 엔진을 멀티 쓰레드 구조로 만드는 것은 앞으로의 CPU에서 충분한 게임 엔진의 성능을 이끌어 내기 위해서는 필요한 일이다. 그러나 게임 엔진을 멀티 쓰레드 구조로 만드는 것은 그것 자체로도 상당히 난해한 문제이다. 기본적으로 게임 엔진 자체가 다른 애플리케이션들에 비해서 그 구조가 상당히 복잡하다. 게임 엔진은 많은 수의 서브 시스템(인공지능, 물리 시뮬레이션, 네트워킹, 캐릭터 애니메이션, 렌더링, 사운드 등)으로 구성되어 있으며, 이들의 유기적인 결합을 통해 하나의 게임 엔진으로 구성된다. 이러한 구조 자체는 상당한 복잡도를 가진다. 그리고 이러한 구조는 서브 시스템간의 의존성이 기본적으로 상당히 높아서 병렬 구조로 바꾸는 것이 어려울 수밖에 없다.

그리고 게임 엔진은 이벤트 중심의 애플리케이션이다. 하지만 그 이벤트가 미리 정의된 형태로 발생하지 않는다. 사용자의 입력에 의해 시시각각 변화를 하며, 또한 이런 처리는 실시간에 처리되어야 한다. 많은 수의 서브시스템의 유기적 결합에서 오는 복잡성과 일반적인 실시

〈그림 5-2-3-01〉 게임 엔진의 멀티 쓰레드 구조



간 애플리케이션이 가지는 복잡도가 결합되어 더욱 높은 복잡도를 창출한다.

현재의 프레임 결과물이 이전 프레임의 결과물에 의존적이기 때문으로, 게임 엔진은 이로 인해 프레임을 여러 개로 나누어서 병렬로 처리할 수가 없다. 즉 10분의 1초 전의 프레임, 현재 프레임, 10분 1초 후의 프레임을 동시에 병렬 처리할 수 없고, 항상 현재 프레임만을 이전 프레임에 의존해서 처리해야 한다. 따라서, 병렬 처리 구조를 한번에 여러 개의 프레임을 처리하는 것으로 구성하기 어렵다. 한 프레임에서 일어나는 각 시스템의 처리를 병렬화시켜야 하는 것이다.

각 서브 시스템 전체를 병렬 처리하는 것에도 문제가 있다. 기본적으로 게임 로직이나 인공지능, 애니메이션, 충돌 처리 등은 프레임 정보를 결정하는 부분이며 서로의 정보가 동기화되어야 하는 부분이다. 이 결과를 실제적으로 렌더링하여 사운드나 그래픽 정보로 출력하게 된다. 즉 인공지능이나 물리가 현재 프레임에서 끝나기 전에는 현재 프레임의 렌더링은 수행되지 않는다. 또한, 렌더링 부분은 병렬화하지 않는 것이 좋다. 직렬화된 데이터 스트림으로 그래픽 프로세서나 사운드 프로세서로 전송해 주면 해당 프로세서 내부에서는 병렬로 처리하는 구조로 현재 하드웨어가 구성되어 있기 때문이다. 즉 CPU상에서 병렬화를 한다고 해도 오히려 성능의 저하를 가져오게 될 것이다.

즉, 결론적으로 게임 로직이나 인공 지능, 물리 시스템, 충돌 시스템, 애니메이션 시스템 등이 병렬화를 통해 성능 향상을 얻을 수 있을 것으로 생각한다.

(3) 성능 향상 정도

성능 향상은 실제적으로 다음에 설명한 쓰레드 모델 중에 어느 것을 선택하느냐 또는 현재 엔진이 돌아갈 플랫폼이 어느 것이냐에 따라 큰 영향을 받는다. 그러나, 현재의 PC 플랫폼을 기

준으로 2개의 코어를 가진 CPU를 가진 경우, 병렬 연산 효율 법칙에 따라 2개의 프로세서로 작업하는 경우 이론적으로는 1.3배 정도의 향상이 있게 된다.

암달의 법칙(Amdahl's Law)

$$T_p = \left(\%S + \frac{1-\%S}{N}\right) * T_s$$

$$Speedup = \frac{T_s}{T_p}$$

- Tp : 병렬 프로세스 수행시간
- Ts : 직렬 프로세스 수행시간
- %S : 직렬 프로세스의 수행시간의 전체 수행시간에 대한 비율
- N : 프로세서의 개수

$$T_p = \left(0.3 + \frac{1-\%S}{2}\right) * 1$$

$$T_p = 0.65$$

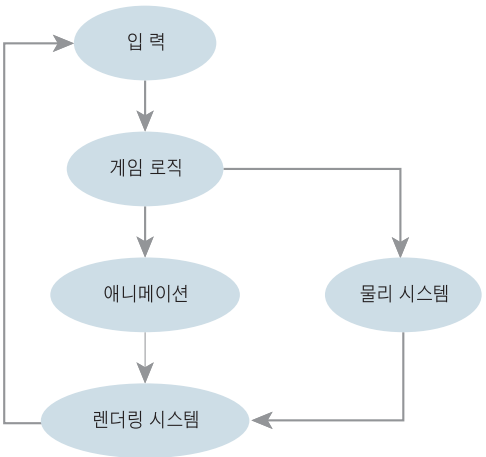
수행 부분	처리 비율	직렬 연산 또는 병렬 연산 구분
인공지능	0.1	병렬
애니메이션	0.2	병렬
파티클 시스템	0.1	병렬
물리 시스템	0.2	병렬
충돌 시스템	0.1	병렬
사운드	0.1	직렬
비디오	0.2	직렬

이것은 병렬 처리시의 작업 분배 시간이나 결과물을 합치는 시간을 제외한 것으로 실제적으로 이것보다 적은 양의 성능 향상이 있게 된다. 또 실제적으로 병렬 처리하는 부분이 전체 프로세스에서 점유하는 비율이 늘어나는 경우(즉 물리나 인공 지능 등이 상당히 무거워졌을 경우) 실제적으로 더 많은 성능향상이 일어난다. 현재의 PC 플랫폼에서 돌아가는 대부분의 온라인게임들은 물리 연산이나 인공지능 연산이 싱글 미션 위주의 패키지 게임에 비해 상대적으로 가벼운 형태이므로, 실제적인 성능 향상은 현재의 PC 온라인 플랫폼에서는 미미할 것으로 생각된다.

2. 게임 엔진에서 적합한 멀티 쓰레드 모델

현재 게임 엔진을 위한 멀티 쓰레드 구조의 경우 동기화냐 비동기화냐, 프로세스의 관점이냐 데이터의 관점이냐에 따라 크게 3가지 모델로 구분해서 생각할 수 있다.

(1) 동기화 함수 병렬 모델 (Synchronous function parallel model)



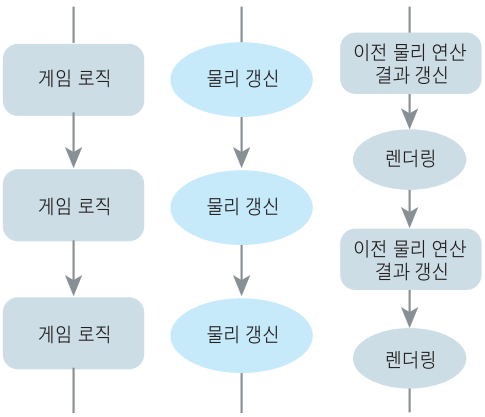
이 모델은 게임 루프 구조에서 서로 상호 작용이 적은 서브 시스템만을 병렬 구조화하는 것이다. 이 경우는 병렬 처리가 되는 동안 시스템 간에는 어떠한 의존성도 없다는 것을 가정한다. 물리 연산이 일어나면서 애니메이션을 처리하는 경우가 한 예가 될 수 있다.

이 모델은 게임 루프나 애플리케이션 프레임워크를 구성할 때, 병렬 처리를 하는 부분에서 각각 서브 시스템의 태스크를 나누거나 구성하는 데 있어 의존성이 없도록 구성해야 한다는 제약이 있다. 그리고 이 방식은 사용할 수 있는 하드웨어 병렬 프로세서의 개수에 크게 제약을 받는다는 단점이 있다. 병렬화시킬 시스템의 개수와 프로세서의 개수가 일치할 때 가장 효율적이다. 그리고 하나의 프로세서에 하나의 서브 시스템 프로세스를 할당하는 만큼 각각의 프로세스의 크기가 너무 작으면 실제적인 효율이 상당히 떨어진다.

이 모델에서 기대할 수 있는 성능 향상은 전체 게임 루프의 최대 실행 길이가 짧아진다는 데에 있다. 따라서, 얼마만큼 병렬화시킬 부분이 많으냐에 따라 성능 향상 폭이 결정된다. 앞으로 설명할 모델들에 비해서 실제적으로 병렬화되는 부분이 가장 적으므로 성능 향상도 가장 적은 모델이다.

아무래도 이 모델은 실제 병렬화되는 시스템이 거의 의존적이지 않은 것을 가정하므로 게임 엔진을 설계할 때 각 시스템이 서로 최대한 의존적이지 않도록 구성해야 할 것이다.

(2) 비동기화 함수 병렬 모델 (Asynchronous parallel model)



이 모델은 게임 루프를 가지고 있지 않다. 각각의 서브 시스템들은 고유의 수행을 하고 가장 최근에 갱신된 결과를 서로 공유하는 형태로 구성된다. 그림에서 보듯이 렌더링을 수행할 때 물리 시스템의 결과를 기다리는 것이 아니라 가장 최근에 갱신된 결과를 가지고 렌더링을 수행한다.

동기화 함수 모델과 마찬가지로 이 모델도 얼마만큼 병렬화시킬 부분을 만들어 내느냐가 전체적인 성능과 효율을 결정한다. 이 모델은 서브 시스템 간의 결과를 기다리지 않으므로 더 쉽게 병렬화시킬 수 있다. 동기화 모델에서는 렌더링과 물리를 따로 분리하는 것이 힘들었지만, 이 모델에서는 렌더링과 물리를 분리시켜서 연동할 수 있다. 렌더링은 현재 프레임을 그릴 때 이미 갱신된 물리 시스템의 결과를 사용하고, 렌더링이 이루어지는 동안 물리 시스템은 다음 프레임의 물리 연산을 수행하면 된다.

따라서, 동기화 모델보다 상대적으로 더 많이 병렬화시킬 수 있으며, 더 많은 프로세서를 활

용할 수 있다. 또한 동기화 모델과는 달리 프로세서의 개수가 적어도 쓰레드 스케줄링에 의해서 안정적인 성능 향상을 꾀할 수 있다.

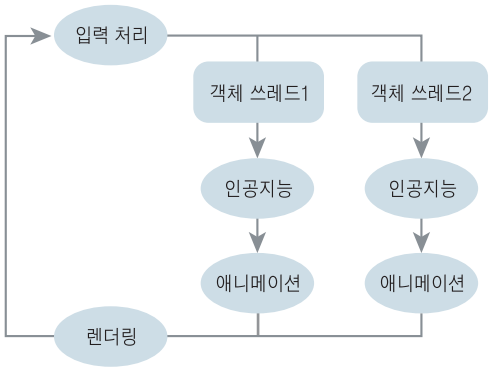
그러나, 이 모델에서는 각각의 서브 시스템 작업간에 타이밍에 의한 문제가 발생할 수 있다. 입력과 물리 그리고 렌더링 시스템이 서로 다른 쓰레드로 동시에 수행된다고 가정했을 때 입력이 가장 먼저 수행되고 그 다음이 물리 그리고 제일 마지막에 렌더링이 수행되어야 가장 효율적이다. 서로간의 수행 타이밍이 맞지 않을 경우 쓰레드간에 대기 시간이 발생하게 되므로써 수행 효율이 떨어지게 된다.

비동기 함수 병렬 모델은 각각의 동시 작업간에 동기화가 상당히 적음을 가정하고 있으며, 전체 프로그램에서 직렬화된 부분이 얼마나 많으냐에 성능이 영향을 받지 않는다. 따라서 성능에 가장 영향을 주는 부분은 얼마나 동시에 많은 작업을 수행시킬 수 있는가에 달려 있다. 또한, 타이밍을 서로간에 맞추기 위해서 각각의 작업의 크기는 균등해야 할 것이다. 하나의 작업은 매우 크나 다른 작업들은 작을 경우 가장 큰 작업에 의해서 병목현상이 발생한다.

이 모델을 사용할 때는 동시에 이루어지는 작업들간에 정보 교환이 필요할 때 점유 문제가 발생할 수 있다. 구현할 때 점유 문제가 발생하지 않도록 신경을 기울일 필요가 있다.

(3) 데이터 병렬 모델(Data Parallel Model)

앞의 두 모델과는 다르게 이 모델은 데이터의 관점에서 병렬화한다. 수행 중인 시스템 내에서 병렬화되어 있는 데이터들을 발견할 수 있으며, 이것이 게임 엔진에서는 게임 객체이다. 이 게임 객체들을 그룹화해 각각의 쓰레드에 할당하는 방식으로 병렬화를 꾀할 수 있다.



이 모델은 각각의 객체들을 어떻게 분류하여 그룹화하느냐가 제일 중요하다. 각각의 객체에 걸리는 프로세스 부하는 균등해야 하며, 서로간의 연관성이 적도록 묶어야 한다. 서로 의존성이 있는 객체의 경우 객체간 통신이 필요해지며, 이 경우 동기화보다는 비동기적인 방법으로 서로간의 통신을 시키는 것이 그나마 성능 저하를 줄일 수 있다. 간단히 생각하면 서로간의 근거리에 있는 객체끼리 하나의 그룹으로 묶는 것을 생각해 볼 수 있다.

각 객체의 쓰레드는 프로세서의 개수에 따라 상당히 유연하게 만들 수 있으며, 병렬화가 힘든 부분은 입력이나 렌더링 등 객체가 직접 관여하지 않는 부분에 한정된다. 따라서 상당히 많은 부분을 병렬화할 수 있다.

이 모델의 성능 향상 역시 얼마나 많은 부분이 병렬화되느냐에 달려 있으며, 실제적으로 이 모델에 의한 병렬화는 엔진의 거의 모든 부분을 병렬화할 수 있으며, 따라서 3가지 모델 중에 성능 향상 폭이 가장 크다.

이 모델의 단점은 병렬 처리할 때 서로 의존

적인 부분이 발생하면 그 처리가 상당히 난해해진다는 점이다. 간단히 생각해서 물리 시스템 같은 경우 하나의 객체가 물리 연산이 되어서 다른 객체에 영향을 줄 수가 있다. 이 경우에 이 결과를 다른 객체에 적용시키기 어렵다. 따라서, 물리나 충돌 시스템의 경우는 별도로 처리하는 것이 용이할 것이다.

3. 실제 구현 시 고려사항 및 결론

앞에 살펴본 3가지 모델 중 현재 가장 많이 사용되는 모델은 두번째 모델인 비동기 함수 병렬 모델이다. 첫번째 방법은 구현이 비교적 간단하지만, 병렬화되는 부분이 적고 그 성능 향상이 코어 개수에 크게 영향을 받지 않아 그다지 각광받지 못하고 있다. 세번째 모델의 경우 코어의 개수가 많을수록 유리하지만 현재의 하드웨어의 코어 프로세서 개수가 그렇게 많지 않아 현재로서는 충분한 성능을 기대하기 어렵다. 앞으로 코어의 개수가 늘어나게 되면 부각되지 않을까 생각된다.

비동기 함수 병렬 모델은 현재의 하드웨어에 가장 적합한 병렬 구조라고 생각된다. 현재까지 나온 많은 상용 엔진들이 이 모델에 기반해서 엔진을 설계하고 있다.

이 모델을 구현할 때의 팁은 점유 문제 발생 시에 록(Lock)을 걸지 않도록 하는 것이다. 록 프리(Lock-Free)로 멀티쓰레드 애플리케이션을 구현하는 방법은 많은 자료가 있으므로 참조하기 바란다. 실제로 점유 문제를 해결하기 위해 크리티컬 섹션 등 API 기능을 활용하는 경우 상당한 성능 저하가 발생하므로 게임 엔진에서는 상당히 심각하게 생각할 문제이다.

현재까지 발표된 여러 자료들을 보면, 대부분 비디오 콘솔 플랫폼에서 각각의 코어에 어떤 일을 하는 쓰레드를 할당했는지에 대한 많은 예를 볼 수 있지만 실제적인 성능 향상 폭에 대한 내용은 찾기가 힘들다. 아마도 멀티 코어의 사용 예만 나오는 것은, 비디오 콘솔 플랫폼들은 멀티 코어의 성능을 끌어내지 않으면 실제로 그 성능을 거의 발휘하기 힘들기 때문에 필수적으로 사용해야 한다는 의미가 아닐까 생각된다.

현재 PC의 경우는 CPU 타입의 차이 등과 현재 듀얼 코어 CPU가 대세인 점을 생각할 때 당장 멀티 쓰레드 구조를 통해 얻게 되는 성능상의 이득은 상당히 미미하지 않을까 생각한다. 현재 실제적인 듀얼 코어에서의 성능 향상에 대한 자료를 찾기가 어려우나, 자체 샘플 테스트와 주변 지인들을 통해 결과를 들어 종합해 본 결과는 현재의 PC 플랫폼에서는 멀티 쓰레드 구조의 게임 엔진이 얻는 성능 향상은 거의 없는 것으로 결론지을 수 있었다.

그러나, 앞으로 4개, 8개의 코어를 가진 CPU가 계속 해서 나올 것으로 라인업이 되어 있으므로 게임 엔진을 멀티 쓰레드 구조로 작성하는 것은 미래를 대비하기 위한 필수 사항이다.

여기서는 일반적인 형태의 멀티 쓰레드 모델을 거론했지만 최근의 내용들은 물리 엔진 자체 또는 렌더링 엔진 자체를 내부적으로 멀티 쓰레드화하는 것들도 많이 거론되므로 이 부분에 대한 지속적인 연구가 필요할 것으로 본다.

제 4 절 자동화도구 기술

1. 온라인게임 레벨 밸런싱 작업의 문제점과 자동화 도구 개발의 필요성

(1) 레벨 밸런싱 작업

레벨이라는 단어는 게임의 장르나 혹은 상황에 따라 여러 가지 의미로 사용이 된다. 예를 들어 RPG 게임에서는 플레이어의 등급을 의미하는 반면에 FPS 게임에서는 플레이어가 플레이하는 맵(레벨 맵이라고도 한다)을 의미하기도 하는 등 경우에 따라 각기 다른 의미로 사용되고 있다. 여기에서는 레벨을 초급, 중급과 같은 플레이어의 각 단계 혹은 수준을 이야기하는 의미로 플레이어가 게임을 플레이할 때 적절한 난이도를 느낄 수 있도록 함으로써 게임의 몰입도와 재미에 직접적으로 관계되는 단어로 사용한다. 그러므로 레벨 밸런싱 작업이란 각 단계의 플레이어 수준에 맞게 게임의 난이도를 조정하는 일을 일컫는다.

그런데 재미와 직결되는 매우 중요한 문제인 레벨 밸런싱 작업은 다양한 게임에 공통적으로 적용할 수 있는 표준화된 방법으로 접근할 수 없다는 어려움이 있다.

(2) 레벨 밸런싱 작업의 어려움

밸런싱에는 '한 사람의 기획자보다 1,000명의 유저가 낫다.' 라는 이야기가 있다. 이 말을 뒷받침하듯이 대부분의 온라인게임은 개발 후 클로즈 베타 테스트 및 오픈 베타 테스트의 기간을 거쳐서 최종적인 레벨 밸런싱 작업을 거치는 것이 일반적이다.

왜냐하면 대개의 경우 개발 기간이 촉박한 데다 레벨 밸런싱 작업에 많은 인력과 시간이 소요되기 때문이다. 그런데 유저를 대상으로 오픈해서 테스트를 시작하게 되면 지속적으로 서비스를 해나가면서 문제점을 수정해야 하므로 개발 단계에서의 수정과 비교할 때 많은 어려움이 따른다. 게다가 밸런싱이 의도한 대로 균형을 이루지 못하는 경우 부득이하게 서비스를 지연할 수밖에 없으므로 이로 인한 경제적 손실을 포함한 여러 가지 문제점들이 발생하게 된다.

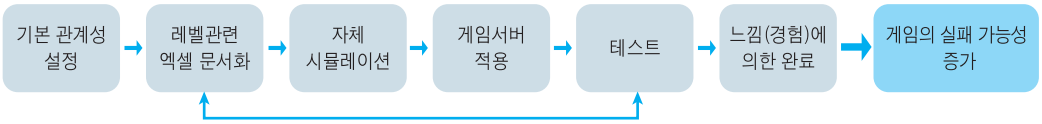
(3) 반복적인 레벨 밸런싱 작업

온라인게임은 개발 기간뿐만 아니라 서비스 기간에도 지속적인 개발과 업데이트가 요구된다. 개발된 콘텐츠를 신속하게 업데이트하고 개발 속도를 빠르게 하기 위해서는 생산성을 높이는 다양한 방법들과 함께 업데이트해야 할 콘텐츠가 이미 서비스되고 있는 전체 게임의 밸런스를 해치지 않도록 조정하는 일도 매우 중요하다. 아무리 좋은 콘텐츠를 신속하게 개발하여 업데이트한다고 하더라도 서비스되고 있는 게임의 밸런스에 심각한 영향을 미친다고 하면 정상적인 게임 진행이 불가능하거나 기존의 게임성에 영향을 미쳐 이용자의 이탈을 가져 오는 등의 치명적인 결과를 초래할 수도 있기 때문이다.

(4) 레벨 밸런싱 검증에 대한 자동화와 시뮬레이션 도구의 필요성

앞서 언급한 바와 같이 레벨 밸런싱 작업은 첫째로 많은 인력과 시간이 소요되는 작업이며,

〈그림 5-2-4-01〉 일반적인 레벨 밸런싱 작업의 프로세스



둘째로 일회성 작업이 아니라 개발기간 및 서비스 기간에 걸쳐 계속해서 반복되는 작업이다. 그러므로 초기 서비스 개시를 예상된 시일에 진행하고 또 서비스 기간에 콘텐츠를 안정적으로 업데이트하기 위해서는 개발 단계부터 밸런싱을 함께 고려해야 하며, 이후에도 지속적인 밸런싱 작업을 위해서 자동화된 레벨 밸런싱 도구를 개발해야 할 필요가 있다.

자동화된 레벨 밸런싱 도구라는 것은 레벨 밸런싱을 검증하기 위해서 주어진 입력값과 수식을 통해서 자동으로 시뮬레이션을 수행하여 원하는 결과물을 사전에 예측, 정리할 수 있도록 도와주는 도구를 말한다. 이러한 도구를 사용하게 될 경우, 유저를 통해서 결과를 얻기 전에 사전에 미리 결과를 어느 정도 예측할 수 있어 오픈 후의 잦은 수정을 피할 수 있다.

그러나 게임 개발의 그래픽, 물리 역학, 네트워킹 등과 관련해서 많은 연구와 발전이 있었지만 레벨 밸런싱과 관련해서는 아직까지 국내외에 표준화된 연구나 자동화 도구가 미비한 상태이며, 대부분 개별 개발사의 자체적인 툴이나 프로세스로 업무를 진행하고 있거나 이마저도 없는 실정이다.

2. 레벨 밸런싱 작업 프로세스

일반적인 레벨 밸런싱 작업의 프로세스는 대부분 다음과 같이 진행된다.

① 레벨 밸런스 검증 요청
② 테스트 일정 수립 및 업무 분배(테스팅 그룹)
③ 테스트 실행 및 결과 정리(테스팅 그룹)
④ 테스트 결과 보고
⑤ 테스트 결과 확인 및 밸런스 수정(게임 디자이너)
⑥ 만족하는 결과가 나올 때 까지 ①부터 반복

위의 프로세스는 개발 후 테스트를 반복해서 검증하는 과정을 거치므로 많은 시간이 소요된다.

그러나 데이터 시트(Data Sheet) 작성부터 테스트, 결과 보고에 이르는 과정까지 레벨 밸런싱과 관련한 작업을 자동화하고 시뮬레이션할 수 있는 도구를 사용하여 개발과 레벨 밸런싱 작업을 함께 진행한다면 개발을 더욱 빠르게 진행하여 게임의 완성도를 높이고 개발 비용을 줄이는 효과가 있다.

그런데 이렇게 개발과 함께 밸런싱 작업을 진행하기 위해서는 시뮬레이션된 결과를 바로 게임에 적용할 수 있는 실시간 적용 프로세스가 요구된다.

온라인게임의 경우 개발 후 서비스하면서 콘텐츠가 지속적으로 업데이트되어 추가되면 서버가 다시 시동하는 데 소요되는 시간도 같이 길어지게 된다. 그래서 처음에는 수정된 수치를

〈그림 5-2-4-02〉 실시간 적용 프로세스

적용하기 위해서 서버를 재시동하는 것이 큰 문제가 되지 않을 수 있지만 나중에는 수치를 조정할 때마다 서버를 재시동하게 되면 밸런싱 작업보다 재시동에 소요되는 시간이 더 걸릴 수 있으므로 가능한 대부분의 데이터 변경이 실시간으로 적용되어 피드백될 수 있도록 하는 것이 중요하다.

3. 레벨 밸런스 자동화 도구의 개발

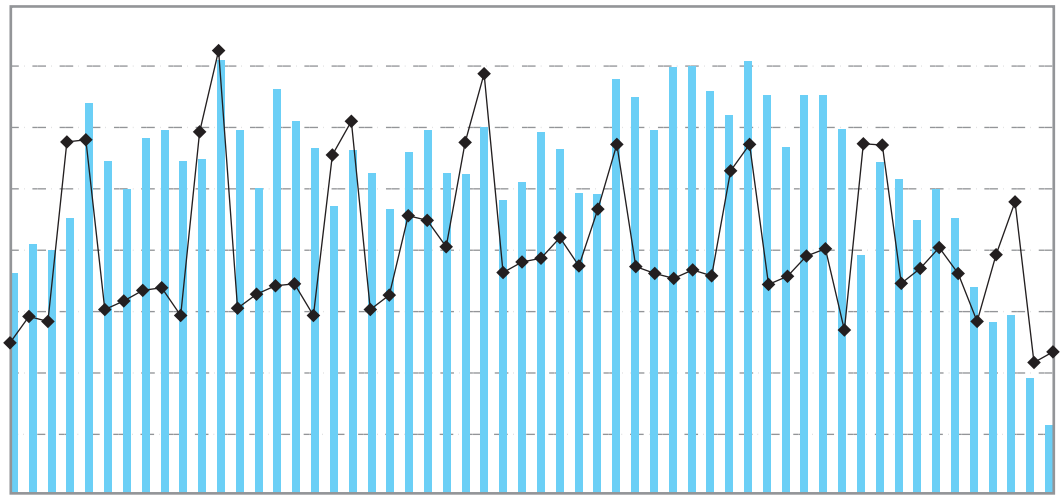
레벨 밸런싱 시스템이란 게임의 밸런스에 영향을 미치는 요소들에 대해 기록, 분석, 적용하여 적은 리소스로 안정된 밸런스를 이룰 수 있도록 게임의 각 요소들을 실시간으로 변경하고 확인해 볼 수 있는 시스템으로 관련된 도구를 제공하여 시뮬레이션 및 결과를 자동으로 행하고 결과를 확인할 수 있도록 하는 것이 특징이다.

게임에서는 밸런스 엔진 및 툴과 연동하여 사용자의 행동 패턴을 이용하여 기록된 수치를 자동화하여 보여주고, 이를 통해 구체적으로 변경 기준을 제시하여 확인해 볼 수 있도록 한다.

일반적으로 레벨 밸런싱 시스템을 사용한 밸런싱 작업은 다음과 같이 진행된다.

① 사용자(기획자)는 밸런스 툴에 시스템 기획에 해당하는 여러 기준과 내용을 입력한다.
② 게임을 테스트한다.
③ 테스트하는 과정에서 밸런스 엔진은 계속

〈그림 5-2-4-03〉 그래프로 변환된 결과값들



적으로 로그를 실시간으로 기록한다.

④ 밸런스 툴은 이렇게 기록된 로그를 자동으로 분석하고, 화면에 출력한다.

⑤ 이를 통해 기획자는 수치 및 밸런스에 대한 점검을 가능토록 한다.

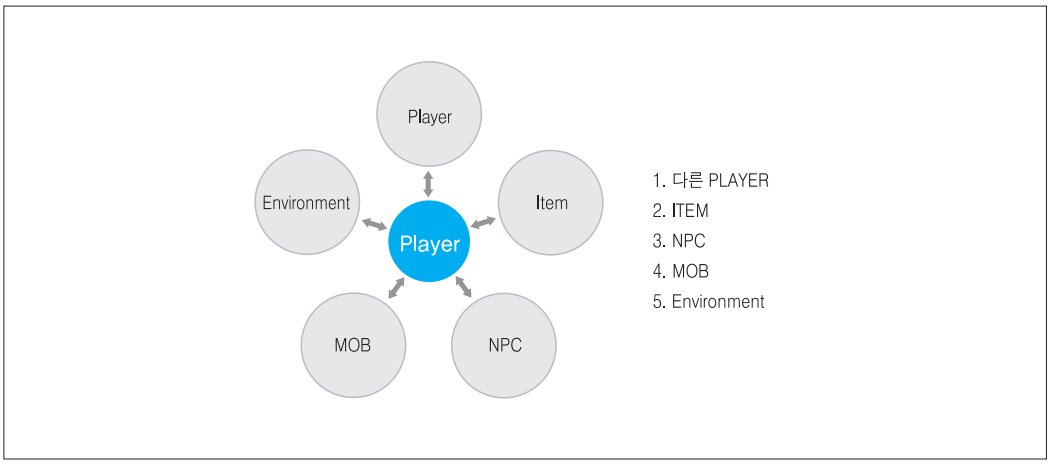
⑥ 기획의도에 맞는 형태를 선택하고 다시 반영할 수 있도록 한다.

이 때 분석된 데이터는 아래 그림과 같이 그래프 등을 이용해서 개발자에게 피드백되는 것이 좋으므로 결과값을 자동으로 그래프화해 주는 툴이 필요하다.

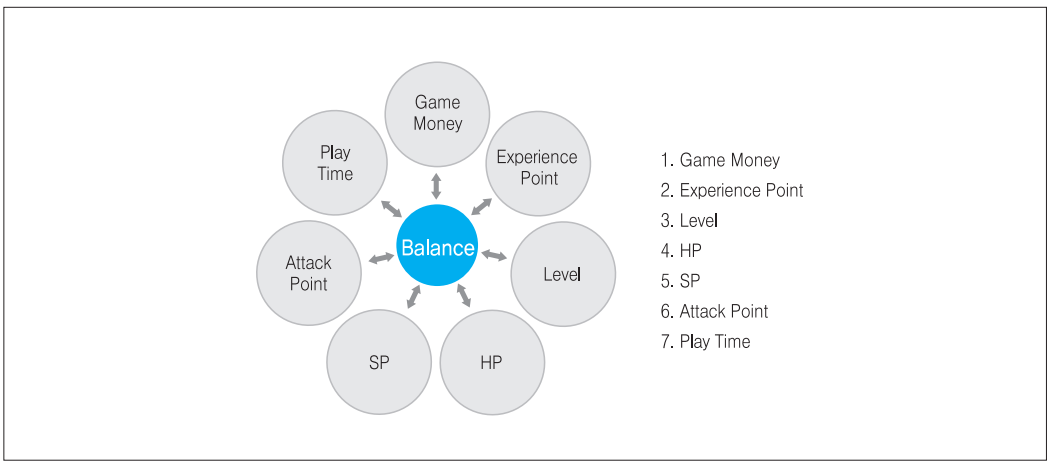
온라인게임의 밸런싱 작업을 위해서는 먼저 게임 내에서 플레이어에게 직접적인 영향을 끼치는 요소와 밸런스를 좌우하는 요소들을 구분해야 하는데 다음은 플레이어에게 직접적인 영향을 끼치는 대표적인 5가지 요소와 밸런스를 좌우하는 핵심 7가지 요소에 대한 그림이다.

온라인게임에서는 플레이어에게 직접 관계하는 5가지의 요소가 밸런스를 좌우하는 7가지 요소의 형태로 플레이어에게 영향을 미치게 된다. 이러한 관점에서 보면 게임의 밸런스라는 것은 플레이어에게 미친 영향이 의도한 바 대로 그 결과가 나타났는지를 검증하는 것이라고 할 수 있다.

〈그림 5-2-4-04〉 온라인게임에서 플레이어에게 직접적인 영향을 끼치는 5가지 요소



〈그림 5-2-4-05〉 온라인게임에서 밸런스를 좌우하는 7가지 요소



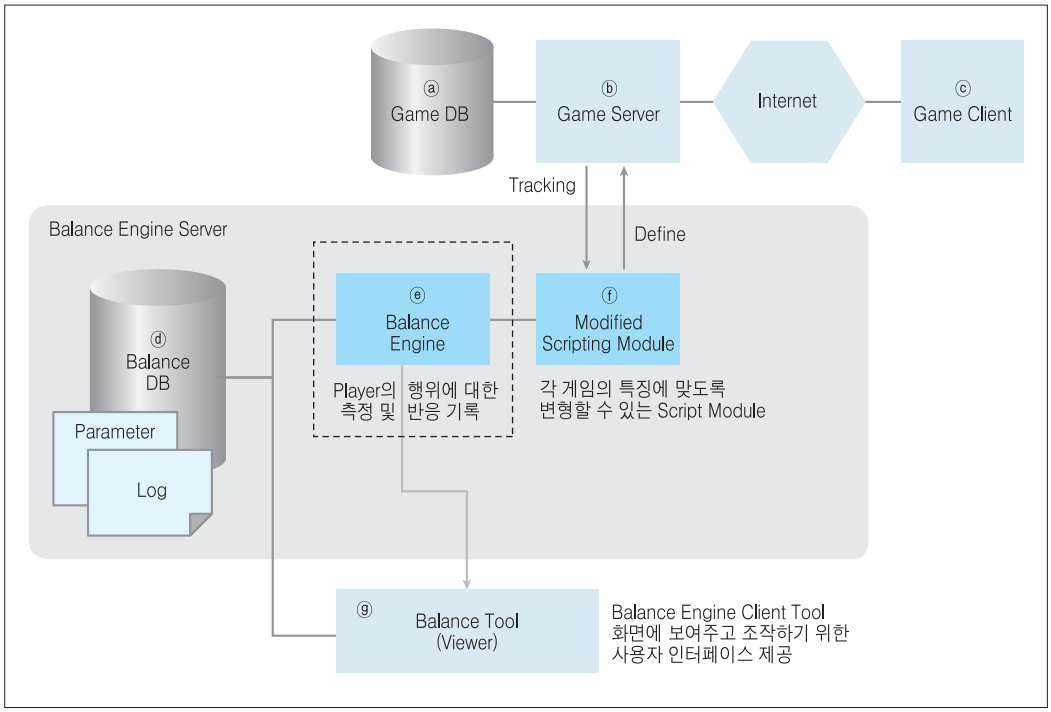
(1) 레벨 밸런싱 주요 단계

게임 내에서 밸런스에 영향을 미치는 구성 요소들로부터 필요한 자료를 만들기 위해서는 다음의 4가지 단계를 거치게 된다.

a. 필터링(Filtering) 단계
b. 청킹(Chunking) 단계
c. 분석(Analysis) 단계
d. 보고(Reporting) 단계

필터링 단계는 분석하고자 하는 목적에 맞도록 로그 데이터에서 필요한 데이터만 추출하는 단계로 로그 데이터는 게임 플레이를 통해서 얻어지게 된다. 청킹 단계는 목적에 적합한 기초 데이터를 산출해 내기 위해 개별 로그 기록들을 유의미한 행위 단위로 그룹핑(grouping)하여 요약된 정보들로 변환하는 단계이다. 즉, 개별 자료를 정보로 바꾸는 단계이다. 분석 단계에서

〈그림 5-2-4-06〉 밸런싱 자동화 도구 시스템 구성도



는 기획자의 의도에 맞게 게임이 진행되는지를 확인하기 위해 다양한 분석을 수행하게 된다. 마지막으로 보고 단계에서는 분석 결과를 레포팅 기능을 이용하여 보고서를 제공하며, 각 요소들 사이의 관계성을 가장 적합한 그래프를 통해서 개발자에게 피드백하게 된다. 레벨 밸런싱 시스템을 구축할 때에는 주어진 입력값들이 위의 단계들을 거치면서 자동으로 분석 및 시뮬레이션되어 원하는 형태로 볼 수 있도록 구성되어야 한다.

(2) 밸런싱 자동화 도구 시스템

밸런싱 자동화 도구 시스템은 개발하는 게임에 따라 여러가지 방법으로 구축할 수 있지만 일반적으로 〈그림 5-2-4-06〉과 같은 구성을 생각할 수 있다.

밸런스 시스템은 크게 밸런스 엔진, 밸런스 DB, 스크립팅 모듈의 서버 사이드 모듈과 밸런싱 툴의 클라이언트 사이드 모듈로 구성이 된다.

가. 서버 사이드(Server-Side)

밸런싱 시스템의 서버 사이드는 스크립트 모듈, 밸런스 엔진, 데이터 베이스로 구성이 되며 스크립트 모듈을 통해 게임 서버와 통신하고 시뮬레이션한 결과를 DB에 기록하게 된다.

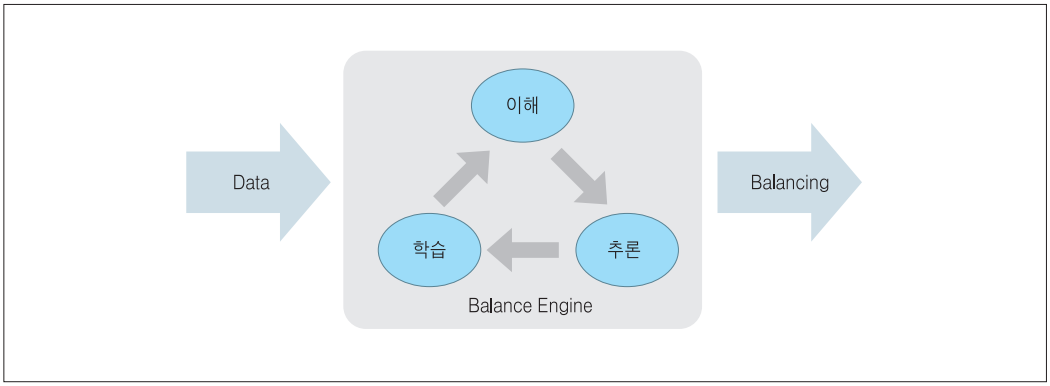
밸런스 엔진은 레벨 밸런싱을 위한 자동화 도구로 스크립팅 모듈을 통해서 게임 서버와 통신하며 적절한 시뮬레이션을 통해서 수치들을 변경시킨다.

이때 시뮬레이션을 위해서 인공지능이 내장될

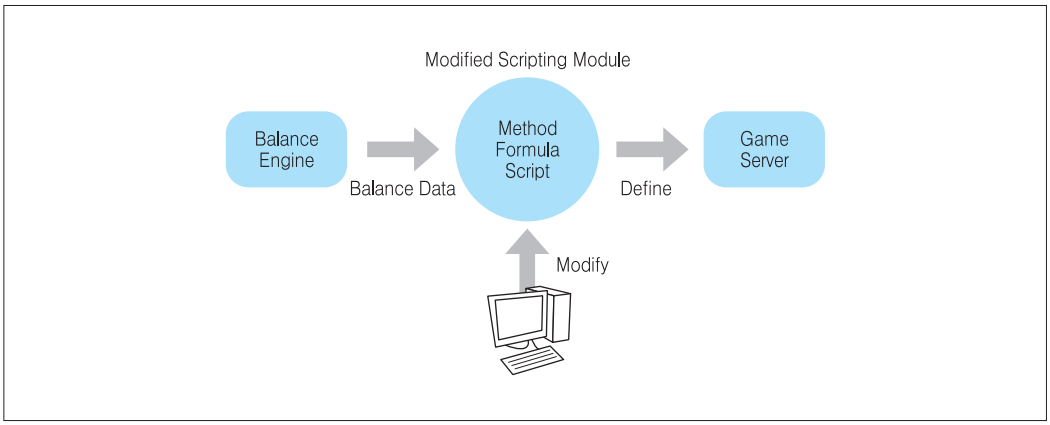
수 있으며 게임 디자이너가 입력한 값들을 주어진 시뮬레이션을 통해서 자동으로 결과를 출력하게 된다. 이러한 시뮬레이션을 위해서 밸런스 엔진에 학습, 이해, 추론 방법을 사용할 수도 있다. 학습은 게임의 특성에 따라 설정된 데이터 및 수치를 가지고 시뮬레이션하는 것을 의미하며, 이해는 학습 결과로 얻어진 로그 및 수치 데이터에 의한 패턴을 분석해서 얻어지게 된다.

별도의 AI를 통해서 학습 및 추론의 과정을 통한 밸런싱 작업을 하는 경우 밸런싱 작업에

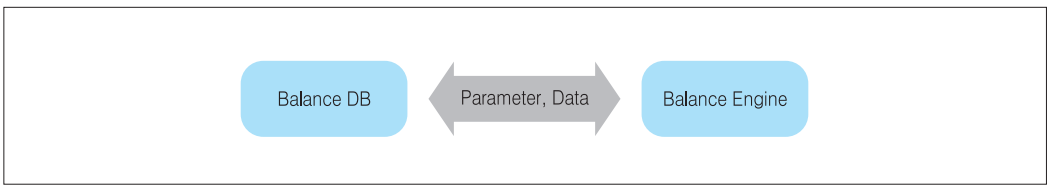
〈그림 5-2-4-07〉 인공지능을 이용한 밸런스 엔진 체계화 사고 과정



〈그림 5-2-4-08〉 스크립트 모듈 역할



〈그림 5-2-4-09〉 Balance DB와 Balance Engine



필요한 수고를 훨씬 덜 수 있다는 장점이 있다.

시뮬레이션에 필요한 수식 혹은 방법들은 게임에 따라 다르며, 같은 게임 내에서도 밸런싱 작업을 진행하면서 계속해서 변경해야 하므로 스크립트를 사용해서 필요할 때마다 쉽게 수정할 수 있도록 하는 것이 좋다. 그러므로 이때에는 게임 서버 역시 스크립팅 인터페이스를 가질 수 있도록 작성되어야 한다.

Balance DB에는 밸런스에 대한 여러 가지 설정자료, 파라미터, 행동패턴에 의한 로그 등이 저장된다. 이러한 수치 데이터는 밸런싱 엔진의 패턴 분석 데이터에 의해 자동으로 변경되어 저장된다.

밸런스 DB를 게임 DB와 분리시켜야 하는데 이렇게 하는 중요한 이유 중의 하나는 개발 단계에서 뿐만 아니라 서비스 단계에서도 밸런스를 위한 파라미터들을 수정하고 히스토리를 저장, 관리해야 하기 때문이다.

나. 클라이언트 사이드(Client-Side)

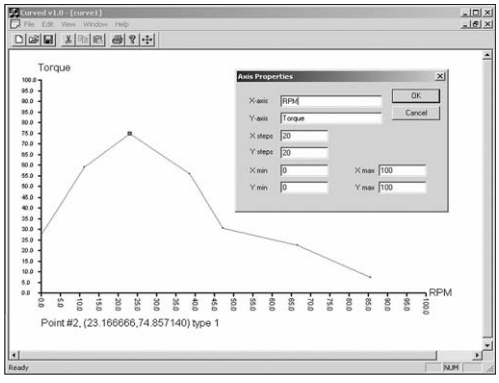
밸런싱 시스템의 클라이언트 사이드는 개발자가 입력이나 피드백된 상태를 확인할 수 있는 툴 형태로 구성이 된다. 이들 툴을 통해서 밸런스 엔진 및 밸런스 DB에 연결하여 밸런스와 관계된 파라미터를 조정하고 시뮬레이션하게 된다.

현재 개발이 초기 단계이거나 개발이 한참 진행 중이거나 서비스 중이더라도 여력이 있는 경우에는 앞에서 언급한 밸런싱 엔진과 같은 밸런싱을 위한 시스템을 구축해서 밸런싱 전략을 수립한다면 레벨 밸런싱 작업을 더욱 빠르고 효율적으로 진행할 수 있을 것으로 기대된다. 만약 이러한 시스템을 자체적으로 구축할 수 없는 경우라면 +7 Balance Engine(http://www.plus7systems.com)과 같은 상용 밸런싱 시스템을 도입하는 것도 고려해 봐야 한다.

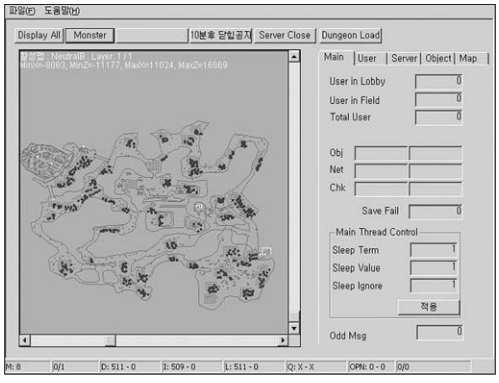
4. 결론

레벨 밸런싱 작업은 개발 초기 단계부터 콘텐츠 개발과 함께 진행하는 것이 좋다. 또한 레벨 밸런싱을 위한 자동화 도구와 시스템을 구축하는 경우 체계적인 밸런싱 작업을 진행할 수 있으며, 오픈 이후에도 큰 어려움 없이 안정된 업데이트할 수 있다는 장점이 있다.

〈그림 5-2-4-10〉 레이싱 게임에서의 밸런스 도식화의 예



〈그림 5-2-4-11〉 몬스터 분포도 - RF Online



제 5 절 온라인게임 DB 기술

1. 온라인게임의 데이터 관리방법

(1) 온라인게임에서 데이터 관리의 문제점

우리나라의 온라인게임은 1992년 한국과학기술원 캠퍼스에서 몇몇 학생에 의해 개발된 최초의 국산 온라인게임 '단군의 땅'을 시작으로 '쥬라기 공원', '바람의 나라' 그리고 '리니지'를 거쳐 발전해왔다. 현재는 소프트웨어의 불법 복제가 만연함에 따라 게임업체의 수익을 보장하지 못하는 패키지 게임은 급격히 감소하고, 온라인게임이 주류를 차지하고 있는 실정이다.

온라인게임은 네트워크를 이용하여 여러 사람이 함께 게임을 진행해 나가는 방식이기 때문에 게임 진행에 필요한 데이터를 관리하는 측면에서 기존의 패키지 게임과는 다른 여러 가지 특징을 가진다.

첫째, 게임 진행을 위한 데이터와 사용자 인증 및 과금을 위한 데이터 관리 방법의 차이로 인한 데이터 처리 속도의 불균형 현상이 발생한다. 온라인게임에서는 패키지 게임과 달리 게임 진행을 위한 데이터 이외에도 사용자 인증 및 과금을 위한 데이터가 필요하다(여기서는 사용자 인증 및 과금 데이터를 '사용자 데이터', 게임의 진행을 위하여 필요한 데이터를 '게임 데이터'라고 부르기로 한다.). 왜냐하면 온라인게임은 사용자가 네트워크를 통해 게임 서버에 접속하여 진행하기 때문에 서버로의 로그인 과정이 필요하고, 게임 진행 시간 및 게임 내에서의 구매 행위 등을 통하여 발생하는 과금 정보를 처리해야 하기 때문이다. 그러므로 패키지 게임과는 달리 사용자 인증 및 과금을 위한 사용자 데이터가 필요하다. 특히, 사용자 인증과 과금 정보는 사용자뿐만 아니라 게임업체의 입장에서도 매우 중요한 정보이므로 데이터가 손상되지 않도록 철저한 보안 및 안전장치가 필요하다. 따라서 대부분의 경우 디스크 데이터베이스 관리 시스템을 이용하여 저장, 관리한다. 한편, 게임 데이터는 게임의 진행 속도에 따라 실시간으로 액세스되는 특성을 가지므로 빠른 처리를 위하여 메모리에서 관리되는 경우가 대부분이다.

만일 게임 내에서 아이템 등의 구매 행위로 인한 과금 요인이 발생하면 그 결과는 과금 데이터가 관리되는 데이터베이스에 반영된다. 일반적으로 과금 정보는 안전한 디스크에서 관리되고, 데이터의 안정성을 위하여 데이터베이스 관리 시스템의 동시성 제어 및 손실 복구 정책에 따라 디스크에 반영되기 때문에 과금 정보에 대한 갱신은 게임 데이터의 액세스에 비하여 매우 속도가 느린 작업이다. 즉, 과금 정보의 갱신으로 인하여 게임의 진행 속도가 느려지는 현상이 발생한다. 그러므로 사용자 데이터 관리 방법과 게임 데이터 관리 방법은 매우 긴밀하게 연동하면서 두 시스템의 속도 차이를 최소화하는 것이 필요하다.

둘째, 동시에 필요한 게임 데이터의 양이 매우 많아진다. 패키지 게임과 온라인게임은 사용자 개인의 관점에서는 관리해야 하는 데이터의 양에 별 차이가 없다. 그러나 온라인게임은 여러 사용자가 동시에 동일한 환경에서 게임을 진

행하므로 게임 서버의 입장에서는 동시 사용자의 수만큼 관리해야 하는 데이터의 양이 증가하게 된다. 특히, 온라인게임 중에서도 가장 대중화되어 있는 MMORPG 게임의 경우, 캐릭터마다 각종 스킬과 보유 아이템에 대한 데이터가 필요하므로 많게는 백 여가지가 넘는 데이터를 관리해야만 한다. 인기있는 게임의 경우, 동시 사용자 수가 수십만명이 넘으므로 관리하여야 하는 게임 데이터의 양도 그만큼 증가한다. 단지 캐릭터에 대한 데이터뿐만 아니라, 모든 던전 및 몬스터 등에 대한 데이터가 사용 중인 상태에 있게 되므로 관리해야 하는 게임 데이터의 양은 더욱 많아지게 된다.

관리해야 하는 데이터의 양의 증가는 여러 가지 문제를 발생시킨다. 데이터 저장 공간이 더욱 많이 필요하게 되고 데이터의 검색, 삽입, 삭제, 갱신, 정렬을 위해 소요되는 시간이 더욱 길어지게 된다. 이는 게임의 속도와 직접적으로 연관되어 결과적으로 게임의 품질을 저하시키는 요인으로 작용한다.

셋째, 게임 데이터가 여러 사용자에 의해 공유되므로 데이터 오류가 발생할 가능성이 높다. 온라인게임은 1인용 게임과는 달리 여러 사용자가 동시에 동일한 환경을 공유하고, 서로 상호 작용을 하면서 진행된다. 이는 게임 데이터가 각 사용자에 의해 독립적으로 액세스되는 것이 아니라 여러 사용자에 의해 동시에 액세스되는 것을 의미한다. 데이터가 단일 사용자에 의해 액세스될 때는 문제가 없더라도, 여러 사용자에 의해 동시에 액세스되는 경우에는 여러 가지 문제가 발생한다. 데이터에 대한 관리 방법을 주요 연구 분야로 삼는 데이터베이스 분야의 연구 결과에 따르면 데이터가 여러 사용자에 의해 동시에 액세스되는 경우 발생하는 이상 현상을 크게 다음의 4가지로 분류하고 있다.

- 갱신 손실(lost update) : 갱신한 값이 또 다른 갱신에 의해 덮어쓰기 되어 무효가 되는 현상
- 오손 읽기(dirty read) : 작업이 완료되지 않은 중간 값을 읽게 되는 현상
- 반복할 수 없는 읽기(unrepeatable read) : 동일한 데이터를 두 번 이상 읽을 때 서로 다른 값을 읽게 되는 현상
- 유령 현상(phantom problem) : 반복할 수 없는 읽기의 특별한 경우로서 어떤 조건을 만족하는 데이터 집합을 두 번 이상 읽을 때 처음에는 없던 새로운 데이터를 읽게 되는 현상

이러한 현상들은 다수 사용자가 동시에 데이터를 액세스하는 경우에 발생할 수 있는 전형적인 문제들로서 데이터베이스 관리 시스템을 사용하면 무난히 해결할 수 있다. 문제는 대부분의 게임에서 게임 데이터를 관리하기 위하여 데이터베이스 관리 시스템을 사용할 수 없다는 데 있다. 데이터베이스 관리 시스템은 편리하기는 하지만 여러 가지 기능을 내장하고 있는 관계로 매우 무겁기 때문에 게임의 진행 속도에 맞게 데이터의 검색, 삽입, 삭제, 갱신을 처리하기 어렵다.

현재 대부분의 온라인게임에서는 게임의 속도를 위하여 게임 데이터를 메모리에서 저장, 관리하는 방법을 사용하고 있기 때문에 위에서 언급한 데이터의 이상 현상에 노출되어 있는 상태이다. 이러한 문제에 대해서는 뒷부분에서 다

시 언급하기로 한다.

넷째, 온라인게임에서는 네트워크를 통하여 진행되는 게임 환경에서 다수 사용자가 동시에 사용하는 데이터를 실시간적으로 처리하여야 한다. 온라인게임은 속도가 비교적 불안정한 네트워크 환경에서 수행되고 실시간으로 진행되기 때문에 게임 데이터에 대한 처리가 더욱 빠르게 수행되어야 한다. 또한 네트워크 속도의 차이로 인하여 발생하는 문제를 해결할 수 있어야 한다.

(2) 기존의 게임 데이터 저장 방법

지금까지 설명한 바와 같이 온라인게임은 다수 사용자가 동시에 게임 데이터를 액세스하는 특성으로 인하여 많은 문제점이 발생한다. 그러나, 많은 온라인게임에서 안정성이 특히 중시되는 사용자 데이터는 데이터베이스 관리 시스템을 이용하여 저장, 관리하고 있으나 게임 데이터는 액세스 속도의 문제 때문에 데이터베이스 관리 시스템을 사용하지 않는다. 대신 게임 데이터를 메모리에서 직접 관리하거나 파일 시스템을 이용하여 관리하고 필요한 경우에만 데이터베이스에 저장하는 방법을 주로 사용한다.

온라인게임에서 게임 데이터를 데이터베이스에 저장하는 방식은 크게 주기적으로 저장하는 방법과 중요한 이벤트가 발생할 때 저장하는 방법으로 분류된다. 주기적으로 저장하는 방법은 미리 정해둔 일정 시간이 경과할 때마다 게임 데이터를 저장하는 방식이다. 이 방법은 동시 사용자 수와 사용률이 증가하면 저장 주기를 길게 조정하고, 반대의 경우에는 저장 주기를 짧게 조정하는 방식으로 게임의 진행 속도에 큰 영향을 주지 않도록 조절이 가능하다는 장점이 있다. 그러나, 경우에 따라서는 저장할 필요가 없는 갱신되지 않은 데이터까지 저장하게 되고, 시스템에 문제가 발생하는 경우 저장된 시점 이후에 갱신된 게임 데이터는 모두 손실되는 문제가 발생한다. 이러한 현상은 게임의 품질에 대한 사용자의 불신을 조장하게 되어 결과적으로 게임의 상용화에 악영향을 끼치게 된다.

중요한 이벤트가 발생할 때 저장하는 방법은 게임 데이터의 중요도에 따라 중요한 게임 데이터가 갱신되는 경우에만 바로 데이터베이스에 저장하는 방식이다. 예를 들어, 새로운 아이템의 획득, 교환, 구매가 발생하거나 중요한 퀘스트를 완료한 경우가 이에 해당한다. 이 방법은 데이터베이스에 대한 액세스 횟수를 약간 줄일 수 있으나, 게임 데이터간의 불일치성이 발생할 수 있고, 온라인게임에서는 중요한 이벤트가 발생하는 빈도가 비교적 높은 편이기 때문에 실효성이 적다는 단점이 있다.

이러한 방법들은 시스템에 이상이 발생하지 않고 게임이 정상적으로 동작하는 동안에는 게임 데이터가 올바른 값을 가진다는 가정에 기반하고 있다. 그러나, 위에서 언급한 바와 같이 동일한 데이터를 다수의 사용자가 동시에 액세스하는 경우에는 게임 데이터의 값 자체에 오류가 발생할 수 있다. 많은 온라인게임에서 게임 데이터의 동시 액세스로 발생하는 문제를 해결하기 위하여 운영체제의 파일 록(lock)을 사용하거나 자체적으로 로킹(locking) 기능을 구현하여 사용한다.

로킹이란 여러 사용자가 동시에 동일한 데이터 항목을 액세스할 때에 미리 정해진 규약에

따라 먼저 록을 획득하도록 하여 데이터에 대한 액세스를 직렬화함으로써 데이터의 오류를 방지하는 장치이다. 록을 사용할 때에는 록의 모드, 록의 단위, 그리고 록 획득 및 반환 규약이 매우 중요하다. 왜냐하면 이러한 요소가 록으로 인한 시스템의 성능 저하 정도와 데이터 오류의 방지 정도를 조절하는 요소가 되기 때문이다. 즉, 록의 단위, 모드, 로킹 규약을 조절함으로써 록으로 인한 시스템의 성능이 많이 저하되더라도 데이터 오류의 발생 가능성을 최소화하도록 설정할 수도 있고, 반대로 데이터 오류의 발생 가능성은 약간 증가하더라도 시스템의 성능을 높이도록 설정할 수도 있다. 그러나, 이러한 기술은 데이터베이스 관리 시스템의 동시성 제어 분야에서 주로 연구되는 것으로 매우 난이도가 높고 복잡한 기술 영역에 속하기 때문에 많은 온라인게임에서 최적의 설정을 찾지 못하고 고가의 하드웨어를 추가로 사용하여 서버를 늘리는 방식으로 해결하고 있는 실정이다. 이로 인하여 동시 접속자수가 늘어날수록 게임 서비스 운영 비용이 급속히 증가하는 문제점이 있다.

(3) 게임 데이터를 위한 저장 시스템

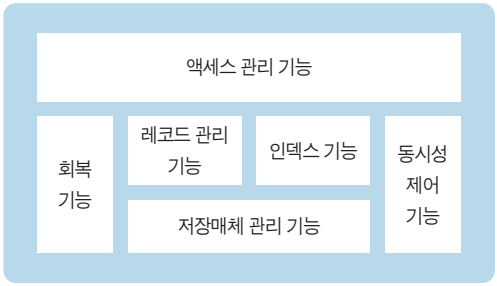
온라인게임에서 여러 사용자가 동시에 액세스하는 데이터를 관리하기 위한 가장 좋은 방법은 빠르고 안정적인 데이터베이스 관리 시스템을 사용하는 것이다. 데이터베이스 관리 시스템은 데이터 보안, 권한부여, 무결성 제약 조건 준수 원칙, 트랜젝션 스케줄링 원칙, 질의 처리 기능, 질의 최적화 기능, 호환성 및 확장성을 위한 각종 유틸리티 등 매우 복잡한 고급 기능을 탑재하고 있는 고가의 소프트웨어이다. 그러나, 이러한 고도의 기능들이 온라인게임에서는 그다지 필요하지 않기 때문에 오히려 시스템의 성능을 저하시키는 요인이 된다. 즉, 현재의 온라인게임에서는 고가의 비용을 소요하는 고급 소프트웨어가 제공하는 불필요한 기능으로 인하여 성능이 저하되는 현상이 발생한다. 이러한 문제점을 해결하는 방법으로 메모리 상주 데이터베이스 관리 시스템을 사용하는 방법이 있다. 이 방법은 디스크 액세스 오버헤드로 인한 성능 저하를 해결할 수 있다. 그러나, 이 시스템도 역시 범용으로 개발되었기 때문에 온라인게임에서는 불필요한 많은 고급 기능을 탑재하고 있으므로 시스템의 성능을 저하시키는 요인이 여전히 존재한다.

〈그림 5-2-5-01〉은 일반적인 데이터베이스 관리 시스템의 구조를 개략적으로 나타낸 것이다. 데이터베이스 관리 시스템은 크게 저장 시스템, 질의 처리기, 각종 API, 각종 유틸리티 등으로 구성된다. 이 중 저장 시스템은 데이터베이스 관리 시스템의 하부 엔진에 해당하는 부분으로서 데이터의 저장, 관리를 위한 핵심적인 기능을 제공한다.

일반적인 경우 데이터베이스 관리 시스템은 몇 개의 서버로 구성되어 응용 프로그램과는 IPC 또는 네트워크를 이용하여 연동된다. 반면에 저장 시스템은 라이브러리 형태로 구성되어 있으므로 저장 시스템을 이용하여 응용 프로그램에 최적화된 데이터 서버를 구축할 수 있는 형태로 되어 있다. 또한, 여러 기능을 선택적으로 조절할 수 있도록 되어 있으므로 데이터베이스 관리 시스템에 비하여 매우 가볍고 속도가 빠른 특징을 가진다. 특히, 데이터베이스 관리

〈그림 5-2-5-01〉 데이터베이스 관리 시스템의 구조

〈그림 5-2-5-02〉 저장 시스템의 구조



시스템이 사용자 편의를 높이기 위하여 제공하는 SQL 등의 질의를 처리하는 질의 처리기와 질의 처리 속도를 높이기 위하여 장착된 질의 최적화기기가 제거되어 사용자 편의성은 떨어지지만 응용 프로그램의 특성에 적합하도록 세심하게 프로그래밍하면 오히려 훨씬 높은 성능을 제공할 수 있다. 더구나 모든 데이터를 메모리에 상주시켜 관리하는 메모리 데이터베이스 관리 시스템의 하부 엔진인 메모리 저장 시스템을 사용하면 메모리에서 관리하는 게임 데이터에 대하여 빠른 액세스와 함께 높은 데이터 안정성을 제공할 수 있다.

널리 알려진 저장 시스템으로는 미국 위스콘신 대학에서 개발한 WiSS, EXODUS, SHORE 등이 있으며, 우리나라에서 개발된 저장 시스템으로는 한국과학기술원에서 개발한 COSMOS 등이 있다.

2. 저장 시스템의 구조 및 기능

본 절에서는 저장 시스템의 구조와 기능을 소개함으로서 온라인게임의 데이터 관리를 위한 새로운 방향을 제시한다. 〈그림 5-2-5-02〉는 저장 시스템의 개략적인 구조를 나타낸 것으로서 물리적인 구조를 도식화한 것이 아니라 내부 기능 위주의 모듈을 도식화한 것이다.

저장 시스템에서 제공하는 일반적인 기능은 다음과 같다.

(1) 저장매체 관리 기능

저장 시스템은 저장 매체를 직접 관리하여 데이터를 저장할 물리적인 영역을 할당, 반환, 관리하는 기능을 제공하고, 새로운 저장매체를 유기적으로 확장, 교체할 수 있는 기능도 제공한다. 이러한 기능은 운영체제에서도 제공하는 기본적인 기능에 해당하지만 저장 시스템에서는 성능을 더욱 높이기 위하여 별도의 저장 매체 관리 기능을 제공하는 경우가 많다. 또한, 저장매체에 대한 접근 횟수를 줄여 성능을 높이기 위하여 독자적으로 버퍼 또는 캐시 기능을 제공한다.

(2) 레코드 관리 기능

저장 시스템은 효율적으로 데이터 레코드를 관리하는 기능을 제공한다. 기본적인 레코드 관리 기능으로는 동일한 형태의 레코드들의 모임인 테이블의 생성, 삭제, 관리 기능과 레코드의 판독, 삽입, 삭제 및 갱신 기능이 있다. 저장 시스템에서는 지원하는 레코드는 파일 시스템에

서와 같은 단순한 이진코드가 아니라 데이터 형(type)을 가지는 여러 속성의 연속이다. 즉, 데이타베이스 관리 시스템에서의 레코드 또는 객체와 동일한 개념이다. 그러므로 파일시스템에서와 같이 사용자가 레코드의 구조를 관리하여 해석할 필요가 없고, 속성 단위로 판독 또는 갱신할 수 있다.

레코드 관리 기능에서는 특정 속성이 비슷한 값을 가진 레코드들을 물리적으로도 인접한 영역에 저장하는 클러스터링 기능을 제공하기도 한다. 클러스터링 기능을 사용하면 버퍼링 효과를 높여 시스템의 성능을 향상시킬 수 있다.

(3) 인덱스 기능

인덱스 기능이란 특정 속성의 값에 대하여 주어진 조건을 만족하는 레코드를 쉽게 검색할 수 있도록 인덱스를 유지, 관리하고 인덱스를 통한 액세스를 제공하는 기능이다. 인덱스 구조로는 주로 트리 형태의 자료 구조가 사용되는데, 디스크 데이터베이스 관리 시스템에서는 B+트리가 주로 사용되고, 메모리 데이터베이스 관리 시스템에서는 T-트리 등의 인덱스 구조가 사용된다. 또한, 인덱스가 구성된 키 값의 크기 순서에 따라 레코드를 순차적으로 액세스하는 인덱스 스캔을 위하여 현재 액세스되는 레코드의 위치를 나타내는 커서 기능도 제공한다.

최근에는 지리 검색 시스템의 발달과 함께 다차원 인덱스 기능을 제공하기도 한다. 다차원 인덱스란 여러 개의 속성을 동일한 수준의 키로 사용하는 인덱스로서 공간상의 영역을 표현하는 데 주로 사용된다. 특히, 온라인게임에서 일정 영역에 속하는 모든 개체에 데미지 또는 마법 효과를 부여하는 경우, 중첩된 영역에 대하여 시도된 두 가지 마법 효과가 상호 작용을 일으키는 것을 표현하는 경우, 미로와 같은 곳에서 목적지를 찾아가는 경로를 제공하는 경우 등에 매우 효과적으로 사용될 수 있다.

(4) 액세스 관리 기능

일반 사용자가 저장 시스템을 효과적으로 이용하기 위해서는 저장 시스템의 여러 가지 기능을 충분히 활용할 수 있도록 사용하기에 편리한 인터페이스와 카탈로그 관리 기능을 제공해야 한다. 이는 사용자가 테이블의 생성 및 삭제, 필요한 인덱스의 구성, 레코드에 대한 액세스 방법의 선택 등을 인터페이스만을 사용함으로써, 저장 시스템의 내부 구조나 움직임을 자세히 알지 못하더라도 그 기능을 충분히 이용할 수 있도록 하기 위한 것이다. 액세스 관리 기능은 이와 같이 사용자가 저장 시스템을 보다 쉽게 사용할 수 있도록 많이 사용되는 기능들을 모아 프로그래밍 언어의 함수 단위 인터페이스를 제공하는 기능이다.

(5) 동시성 제어 기능

동시성 제어 기능은 여러 사용자가 동시에 동일한 데이터를 액세스할 때 발생할 수 있는 데이터 오류를 방지하면서 여러 사용자의 동시 수행 정도를 최대화하는 기능이다. 동시성 제어 기능은 크게 논리적인 동시성 제어 기능과 물리적인 동시성 제어 기능으로 구분된다.

논리적인 동시성 제어 기능은 데이터베이스에 저장된 데이터에 대하여 오류가 발생하지 않도록 동시에 수행되는 여러 트랜잭션의 작업을

조절하는 기능이다. 물리적인 동시성 제어 기능은 데이터베이스에 저장된 데이터뿐만 아니라 시스템이 동작하기 위하여 필요한 모든 데이터에 대하여 데이터 오류가 발생하지 않도록 조절하는 기능이다. 일반적으로 논리적인 동시성 제어를 위해서는 로킹(locking) 등의 방법을 사용하고, 물리적인 동시성 제어를 위해서는 래치(latch) 등의 방법을 사용한다.

특히, 래치는 메모리에서 관리되는 데이터 중에서 여러 사용자에 의해 공유되는 데이터에 대하여 오류가 발생하지 않도록 제어하기 위한 자료구조이다. 래치의 모드에는 읽기 모드와 쓰기 모드가 있어서 읽기 모드의 래치인 경우에는 여러 사용자가 동시에 데이터를 읽는 것이 가능하고, 쓰기 모드의 경우에는 데이터에 대한 독점적인 액세스 권한을 보장한다. 또한, 래치의 단위는 사용자에 의해 조절될 수 있으므로 동시성을 높이기 위하여 래치의 단위를 작게 설정하는 것도 가능하다.

래치는 록에 비하여 매우 단순하고 빠른 연산이므로 온라인게임의 게임 데이터와 같이 메모리에서 관리되는 데이터에 대한 동시성을 제어하는 데 매우 적합하고 유용한 구조라고 할 수 있다.

게임 데이터를 관리하기 위하여 저장 시스템 전체를 사용하지 않더라도 래치를 이용한 물리적인 동시성 제어 방법을 도입하는 것만으로도 시스템의 성능과 데이터의 안정성을 크게 향상시킬 수 있다. 그러나 잘못 사용하는 경우 교착상태가 발생할 가능성이 있으며, 록과는 달리 교착상태를 해결하는 방법이 내장되어 있지 않으므로 데이터의 액세스 특성을 면밀히 분석하여 래치 사용 규약을 설정하는 등 래치 사용에 세심한 주의가 필요하다.

(6) 회복 기능

회복 기능은 데이터에 손상이 발생하는 경우 이를 복구하는 기능으로서 크게 백업 기능과 복구 기능으로 구성된다. 백업 기능은 데이터에 손상이 발생할 것을 대비하여 주기적으로 또는 별도의 명령에 의해 메모리 상의 데이터를 안전한 저장 매체에 복사하는 과정이다. 복구 기능은 백업된 데이터, 데이터베이스의 상태, 그리고 사용자가 수행한 작업에 대한 로그 정보를 바탕으로 최신의 데이터를 구성하는 과정이다. 저장 시스템이 제공하는 회복 기능은 트랜잭션 단위로 최신의 상태를 복구하므로 아이템의 획득, 교환 등과 같은 게임 진행의 중요한 이벤트가 발생하는 경우 해당 작업을 트랜잭션으로 설정하면 시스템에 파손이 발생하더라도 데이터를 안전하게 복구할 수 있다.

3. 결론

온라인게임은 특성상 매우 많은 사용자 데이터와 게임 데이터를 관리해야 하고, 동시에 빠른 데이터 액세스가 필수적이다. 대부분의 온라인게임에서는 사용자 인증과 과금 등의 매우 중요한 정보에 대해서는 데이터베이스 관리 시스템을 사용하고 있으나, 게임 데이터에 대해서는 액세스 속도의 문제 때문에 데이터베이스 관리 시스템을 사용하지 않고, 파일 시스템을 사용하거나 메모리에 모든 게임 데이터를 상주시켜 관리하는 방법을 사용하고 있다. 그러나 이러한 방법은 게임 데이터에 대한 오류가 발생할 가능성이 매우 크고 위험하다. 이를 방지하기 위하여 로킹 등의 방법을 사용하더라도, 데이터 액세스 속도와 데이터의 안정성이라는 두 가지 목적을 모두 달성하기에는 어려움이 많다.

본 고에서는 데이터베이스 관리 시스템의 핵심 엔진에 해당하는 저장 시스템을 소개하고, 게임 데이터를 관리하는 방법으로 저장 시스템을 이용하는 방법을 제시하였다. 저장 시스템은 데이터베이스 관리 시스템의 주요 데이터 관리 기능을 대부분 제공하면서도 여러 가지 사용자의 편의를 위한 부가 기능이 배제되어 데이터베이스 관리 시스템에 비하여 가볍고 빠른 성능을 제공한다. 이러한 저장 시스템의 특성은 빠른 수행 속도와 데이터의 안정성을 동시에 추구해야 하는 온라인게임의 요구 사항에 매우 잘 부합된다. 그러므로 온라인게임에서 데이터의 관리를 위하여 저장 시스템을 이용하는 것은 고려해 볼 만한 가치가 있다.